TDDE73 - Programmering, datastrukturer och algoritmer

Grafer

Christoffer Holm

Institutionen för datavetenskap

LINKÖPING UNIVERSITY

1. Representation av grafer
2. Grafsökning
3. Heap

LINKÖPING UNIVERSITY

1 Representation av grafer
2 Grafsökning
3 Heap

LINKÖPING UNIVERSITY

Representation av grafer

Mål med representationen

- lagra noder

Representation av grafer

Mål med representationen

- lagra noder
- lagra bågar

Representation av grafer

Mål med representationen

- lagra noder
- lagra bågar
- lagra eventuella vikter

Representation av grafer

Mål med representationen

- lagra noder
- lagra bågar
- lagra eventuella vikter
- insättning (noder och bågar)

Representation av grafer

Mål med representationen

- lagra noder
- lagra bågar
- lagra eventuella vikter
- insättning (noder och bågar)
- borttagning (noder och bågar)

Representation av grafer

Mål med representationen

- lagra noder
- lagra bågar
- lagra eventuella vikter
- insättning (noder och bågar)
- borttagning (noder och bågar)
- sökning (?)

Representation av grafer

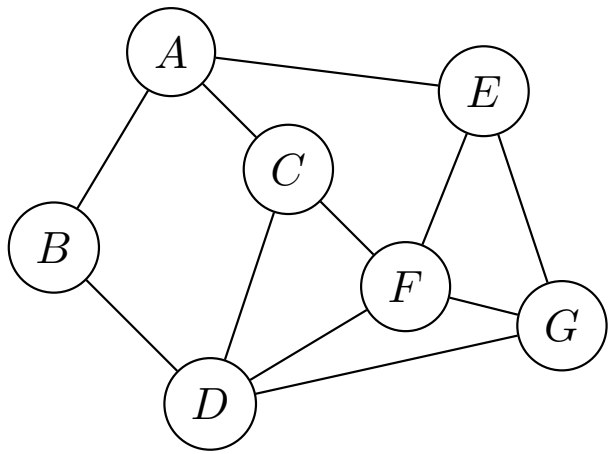
Idé #1

Ide #1

Lagra alla noder i en array och alla bågar som par i en annan array.

Representation av grafer

Linjär representation



noder:

A	B	C	D	E	F	G

bågar:

A, B	A, C	A, E	B, D	C, D	C, F	D, F	D, G	E, F	E, G	F, G

Representation av grafer

Analys av linjär representation

- Insättning av nod: $\mathcal{O}(|V|)$

Representation av grafer

Analys av linjär representation

- Insättning av nod: $\mathcal{O}(|V|)$
- Insättning av båge: $\mathcal{O}(|E|)$

Representation av grafer

Analys av linjär representation

- Insättning av nod: $\mathcal{O}(|V|)$
- Insättning av båge: $\mathcal{O}(|E|)$
- Borttagning av nod: $\mathcal{O}(|V|)$

Representation av grafer

Analys av linjär representation

- Insättning av nod: $\mathcal{O}(|V|)$
- Insättning av båge: $\mathcal{O}(|E|)$
- Borttagning av nod: $\mathcal{O}(|V|)$
- Borttagning av båge: $\mathcal{O}(|E|)$

Representation av grafer

Analys av linjär representation

- Insättning av nod: $\mathcal{O}(|V|)$
- Insättning av båge: $\mathcal{O}(|E|)$
- Borttagning av nod: $\mathcal{O}(|V|)$
- Borttagning av båge: $\mathcal{O}(|E|)$
- Hitta om nod existerar: $\mathcal{O}(|V|)$

Representation av grafer

Analys av linjär representation

- Insättning av nod: $\mathcal{O}(|V|)$
- Insättning av båge: $\mathcal{O}(|E|)$
- Borttagning av nod: $\mathcal{O}(|V|)$
- Borttagning av båge: $\mathcal{O}(|E|)$
- Hitta om nod existerar: $\mathcal{O}(|V|)$
- Hitta om båge existerar: $\mathcal{O}(|E|)$

Representation av grafer

Analys av linjär representation

- Insättning av nod: $\mathcal{O}(|V|)$
- Insättning av båge: $\mathcal{O}(|E|)$
- Borttagning av nod: $\mathcal{O}(|V|)$
- Borttagning av båge: $\mathcal{O}(|E|)$
- Hitta om nod existerar: $\mathcal{O}(|V|)$
- Hitta om båge existerar: $\mathcal{O}(|E|)$
- Om allting hålls sorterat blir det istället logaritmiska tidskomplexiteter...

Representation av grafer

Analys av linjär representation

- Insättning av nod: $\mathcal{O}(|V|)$
- Insättning av båge: $\mathcal{O}(|E|)$
- Borttagning av nod: $\mathcal{O}(|V|)$
- Borttagning av båge: $\mathcal{O}(|E|)$
- Hitta om nod existerar: $\mathcal{O}(|V|)$
- Hitta om båge existerar: $\mathcal{O}(|E|)$
- Om allting hålls sorterat blir det istället logaritmiska tidskomplexiteter...
- Vi kan dock göra ännu bättre!

Representation av grafer

Observation

Givet $|V| = n$:

- Om oriktad graf $\Rightarrow |E| \leq \frac{n(n-1)}{2}$
- Om riktad graf $\Rightarrow |E| \leq n(n-1)$

Representation av grafer

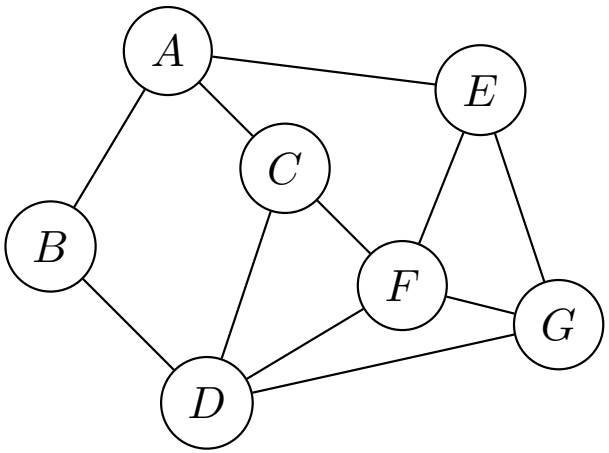
Idé #2

Idé #2

Lagra bågarna som en matris!

Representation av grafer

Grannmatris



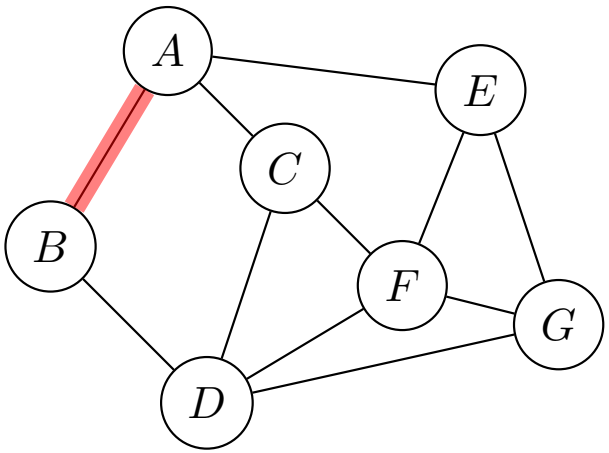
noder:

0	1	2	3	4	5	6
A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0							
1							
2							
3							
4							
5							
6							

Representation av grafer

Grannmatris

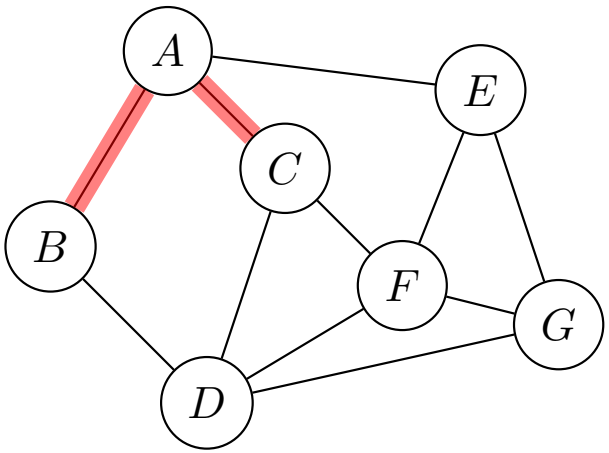


noder:	0	1	2	3	4	5	6
	A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0		1					
1	1						
2							
3							
4							
5							
6							

Representation av grafer

Grannmatris



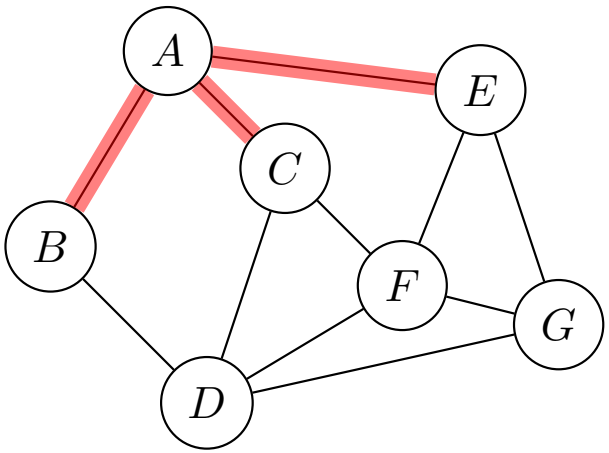
noder:

0	1	2	3	4	5	6
A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0		1	1				
1	1						
2	1						
3							
4							
5							
6							

Representation av grafer

Grannmatris

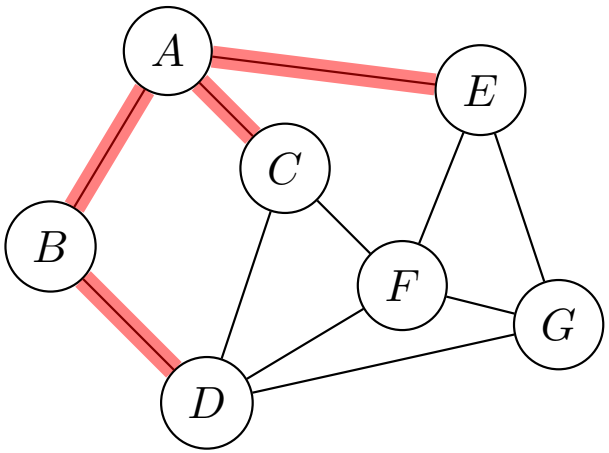


noder:	0	1	2	3	4	5	6
	A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0		1	1		1		
1	1						
2	1						
3							
4	1						
5							
6							

Representation av grafer

Grannmatris

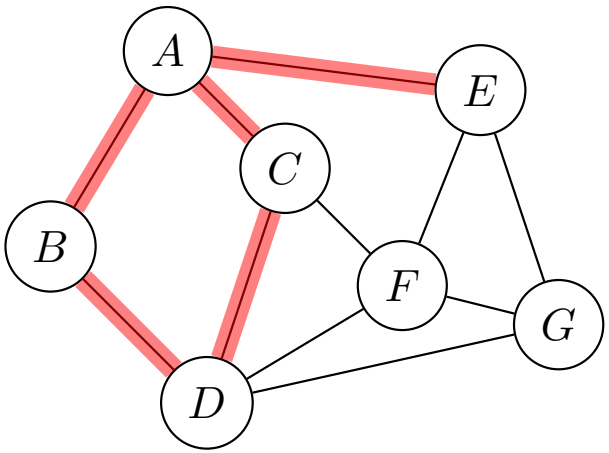


noder:	0	1	2	3	4	5	6
	A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0		1	1		1		
1	1			1			
2	1						
3		1					
4	1						
5							
6							

Representation av grafer

Grannmatris



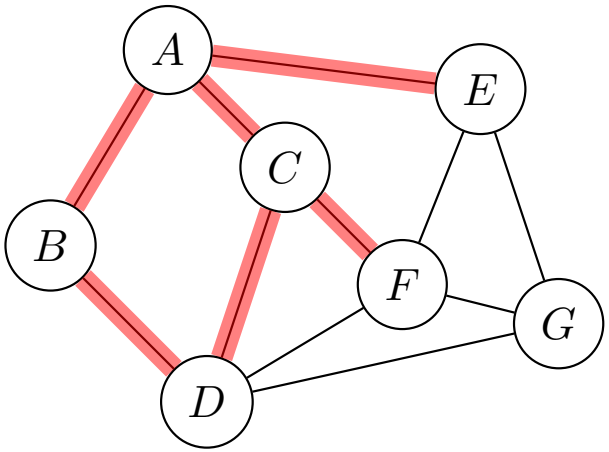
noder:

0	1	2	3	4	5	6
A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0		1	1		1		
1	1			1			
2	1			1			
3		1	1				
4	1						
5							
6							

Representation av grafer

Grannmatris

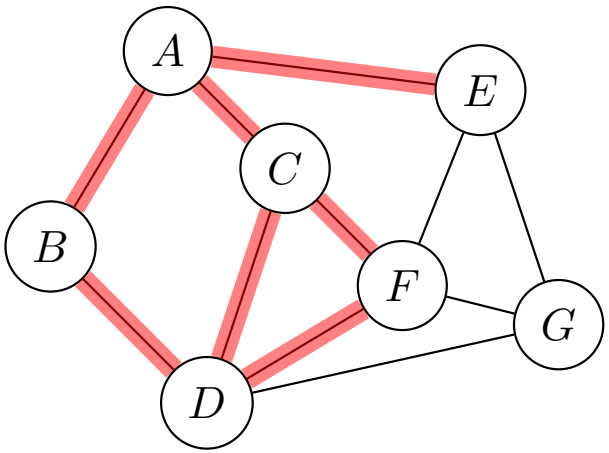


noder:	0	1	2	3	4	5	6
	A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0		1	1		1		
1	1			1			
2	1			1		1	
3		1	1				
4	1						
5			1				
6							

Representation av grafer

Grannmatris



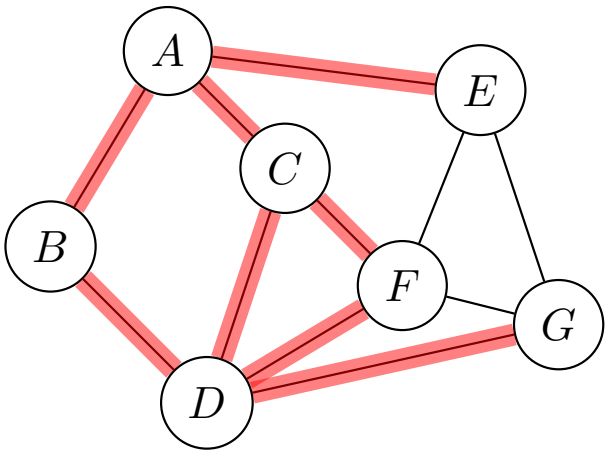
noder:

0	1	2	3	4	5	6
A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0		1	1		1		
1	1			1			
2	1			1		1	
3		1	1			1	
4	1						
5			1	1			
6							

Representation av grafer

Grannmatris



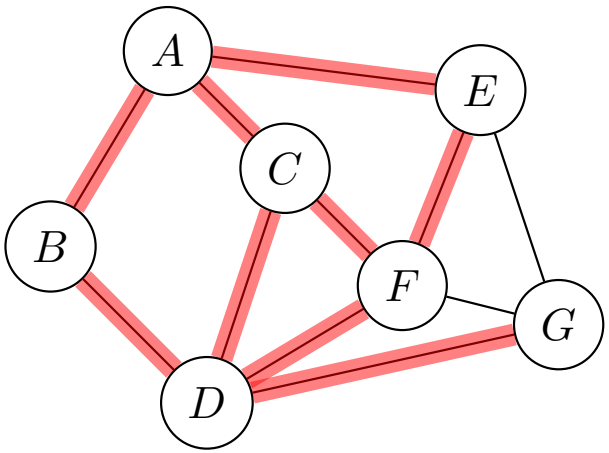
noder:

0	1	2	3	4	5	6
A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0		1	1		1		
1	1			1			
2	1			1		1	
3		1	1			1	1
4	1						
5			1	1			
6				1			

Representation av grafer

Grannmatris



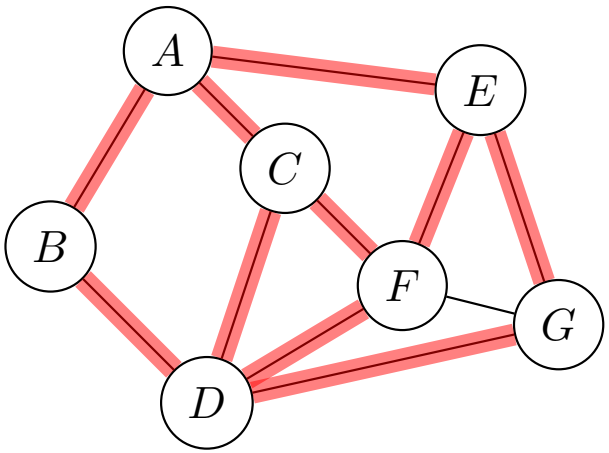
noder:

0	1	2	3	4	5	6
A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0		1	1		1		
1	1			1			
2	1			1		1	
3		1	1			1	1
4	1					1	
5			1	1	1		
6				1			

Representation av grafer

Grannmatris

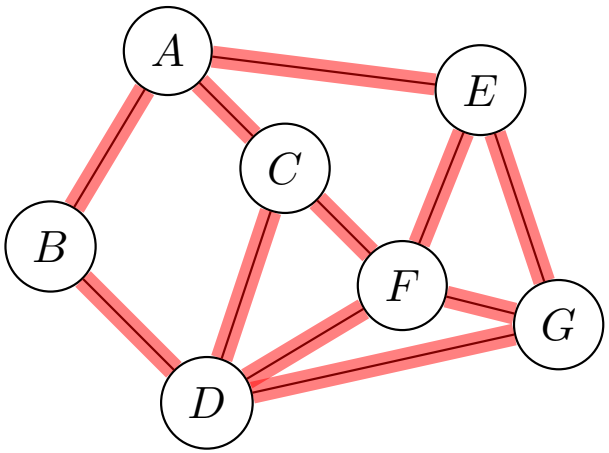


noder:	0	1	2	3	4	5	6
	A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0		1	1		1		
1	1			1			
2	1			1		1	
3		1	1			1	1
4	1					1	1
5			1	1	1		
6				1	1		

Representation av grafer

Grannmatris



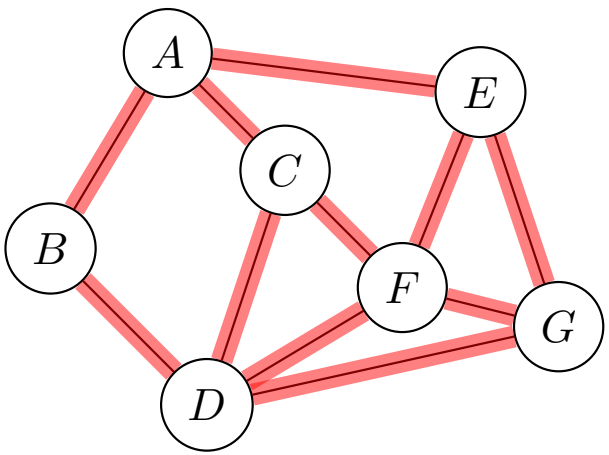
noder:

0	1	2	3	4	5	6
A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0		1	1		1		
1	1			1			
2	1			1		1	
3		1	1			1	1
4	1					1	1
5			1	1	1		1
6				1	1	1	

Representation av grafer

Grannmatris

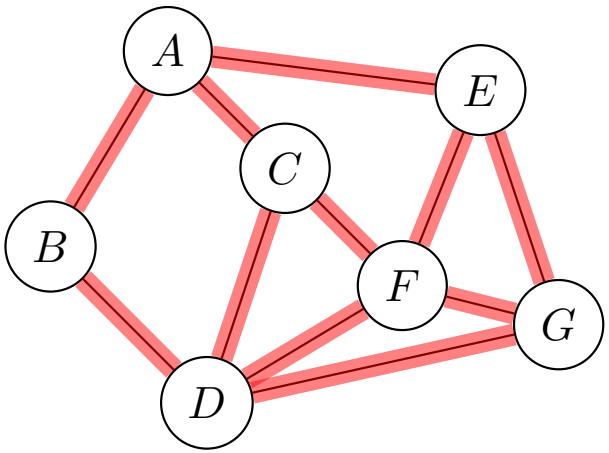


noder:	0	1	2	3	4	5	6
	A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0	0	1	1	0	1	0	0
1	1	0	0	1	0	0	0
2	1	0	0	1	0	1	0
3	0	1	1	0	0	1	1
4	1	0	0	0	0	1	1
5	0	0	1	1	1	0	1
6	0	0	0	1	1	1	0

Representation av grafer

Grannmatris



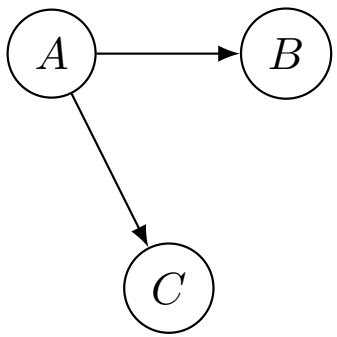
noder:	0	1	2	3	4	5	6
	A	B	C	D	E	F	G

	0	1	2	3	4	5	6
0	0	1	1	0	1	0	0
1	1	0	0		0	0	0
2	1	0	0	1		1	0
3	0	1	1	0	0		1
4	1	0	0	0	0	1	1
5	0	0	1	1	1	0	1
6	0	0	0	1	1	1	0

räcker för oriktade grafer

Representation av grafer

Varför $\mathcal{O}\left(|V|^2\right)$?



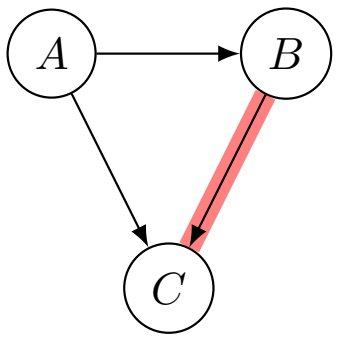
noder:

0	1	2
A	B	C

	0	1	2
0	0	1	1
1	0	0	0
2	0	0	0

Representation av grafer

Varför $\mathcal{O}\left(|V|^2\right)$?



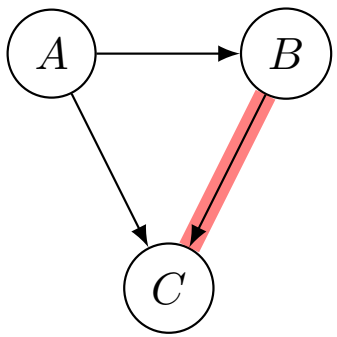
noder:

0	1	2
A	B	C

	0	1	2
0	0	1	1
1	0	0	0
2	0	0	0

Representation av grafer

Varför $\mathcal{O}\left(|V|^2\right)$?



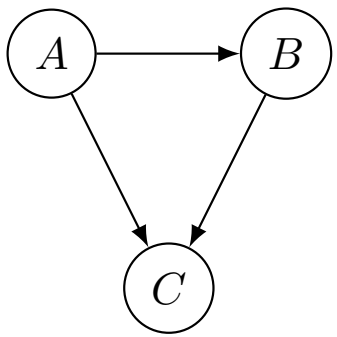
noder:

0	1	2
A	B	C

	0	1	2
0	0	1	1
1	0	0	1
2	0	0	0

Representation av grafer

Varför $\mathcal{O}\left(|V|^2\right)$?

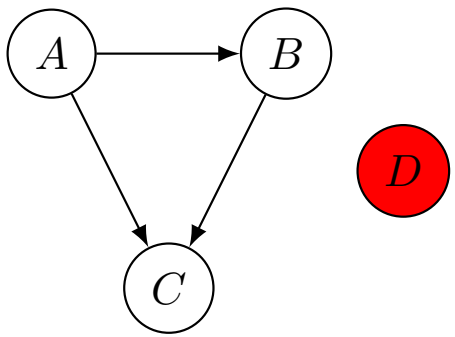


noder:	0	1	2
	A	B	C

	0	1	2
0	0	1	1
1	0	0	1
2	0	0	0

Representation av grafer

Varför $\mathcal{O}\left(|V|^2\right)$?

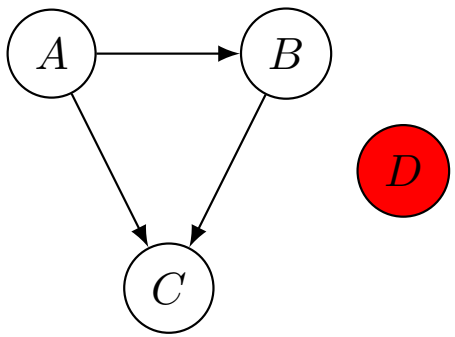


	0	1	2
noder:	A	B	C

	0	1	2
0	0	1	1
1	0	0	1
2	0	0	0

Representation av grafer

Varför $\mathcal{O}\left(|V|^2\right)$?



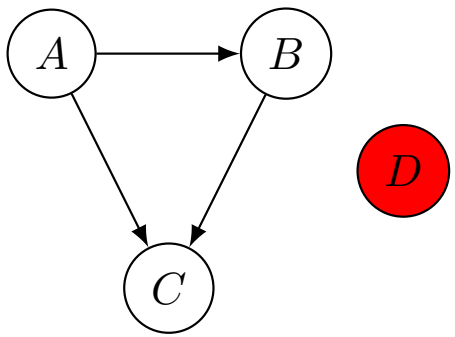
noder:

0	1	2	3
A	B	C	D

	0	1	2
0	0	1	1
1	0	0	1
2	0	0	0

Representation av grafer

Varför $\mathcal{O}\left(|V|^2\right)$?



noder:

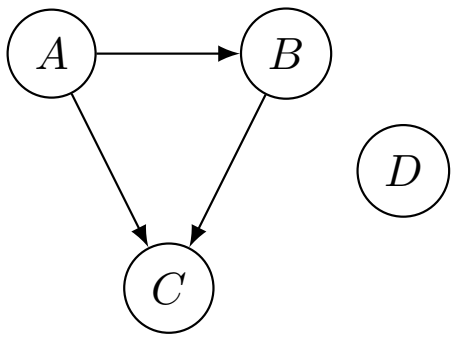
0	1	2	3
A	B	C	D

	0	1	2
0	0	1	1
1	0	0	1
2	0	0	0

onödigt!

Representation av grafer

Varför $\mathcal{O}\left(|V|^2\right)$?

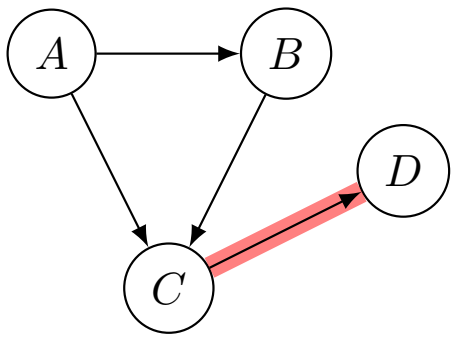


noder:	0	1	2	3
	A	B	C	D

	0	1	2	3
0	0	1	1	0
1	0	0	1	0
2	0	0	0	0
3	0	0	0	0

Representation av grafer

Varför $\mathcal{O}\left(|V|^2\right)$?



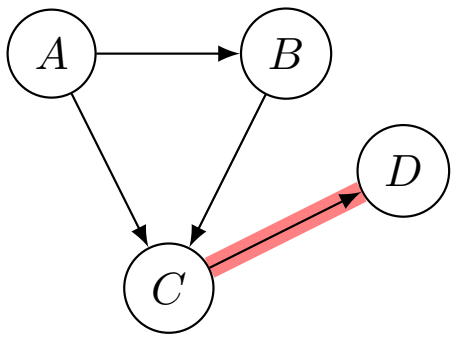
noder:

0	1	2	3
A	B	C	D

	0	1	2	3
0	0	1	1	0
1	0	0	1	0
2	0	0	0	0
3	0	0	0	0

Representation av grafer

Varför $\mathcal{O}\left(|V|^2\right)$?

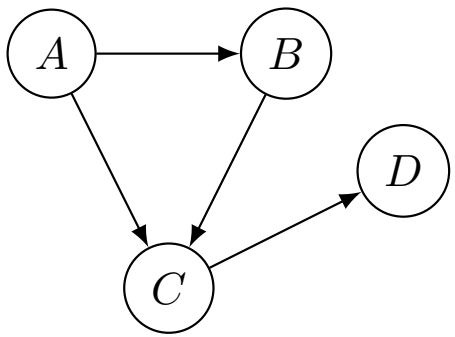


noder:	0	1	2	3
	A	B	C	D

	0	1	2	3
0	0	1	1	0
1	0	0	1	0
2	0	0	0	1
3	0	0	0	0

Representation av grafer

Varför $\mathcal{O}\left(|V|^2\right)$?



noder:

0	1	2	3
A	B	C	D

	0	1	2	3
0	0	1	1	0
1	0	0	1	0
2	0	0	0	1
3	0	0	0	0

Representation av grafer

Slutsatser om grannmatris

- Använder en del minne

Representation av grafer

Slutsatser om grannmatris

- Använder en del minne
- Väldigt snabb om bågarna ändras ofta

Representation av grafer

Slutsatser om grannmatris

- Använder en del minne
- Väldigt snabb om bågarna ändras ofta
- Vi vill helst inte ändra noderna för det är dyrt

Representation av grafer

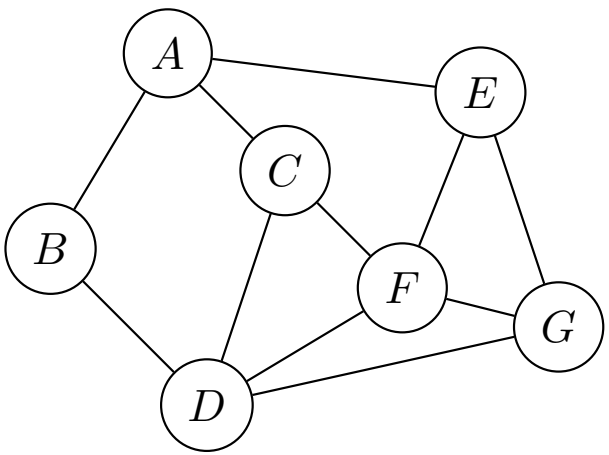
Idé #3

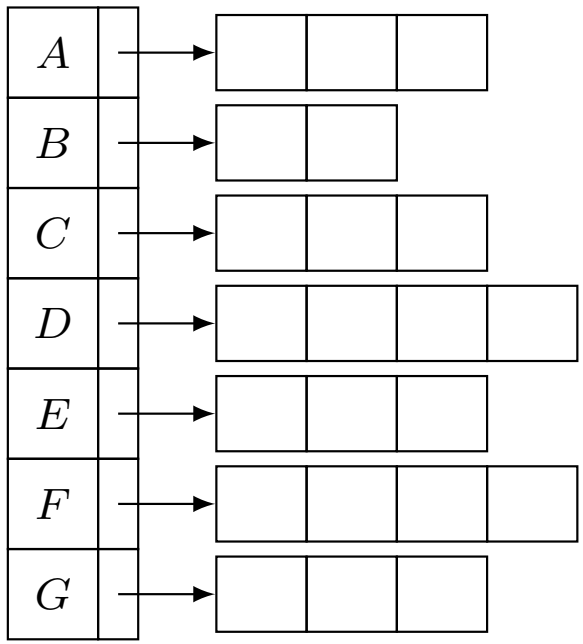
Idé #3

För varje nod, lagra vilka noder den har en båge till

Representation av grafer

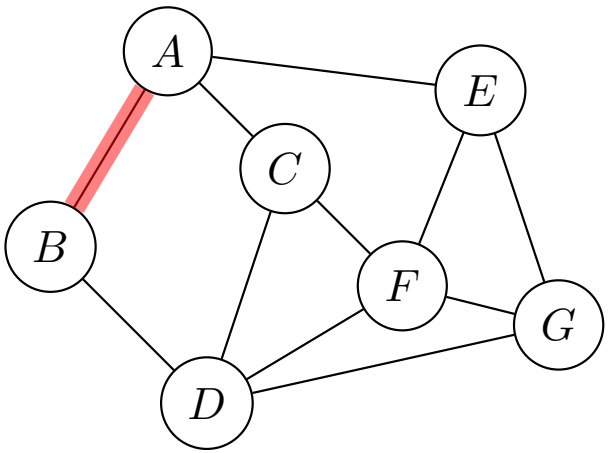
Grannlista

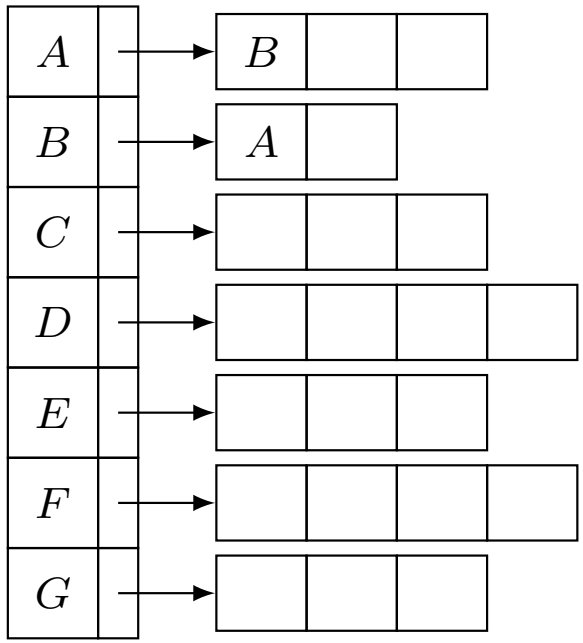




Representation av grafer

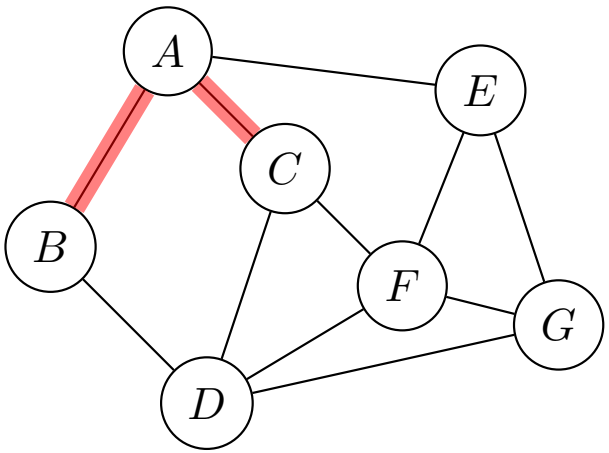
Grannlista

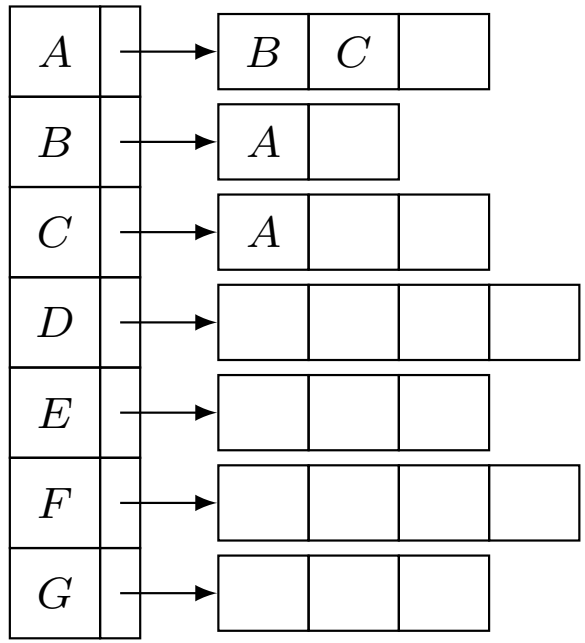




Representation av grafer

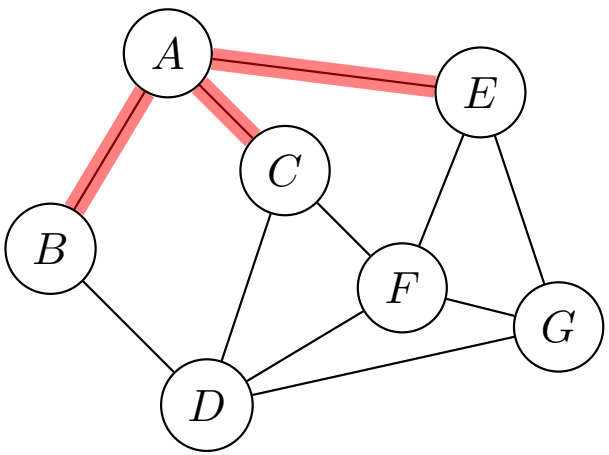
Grannlista

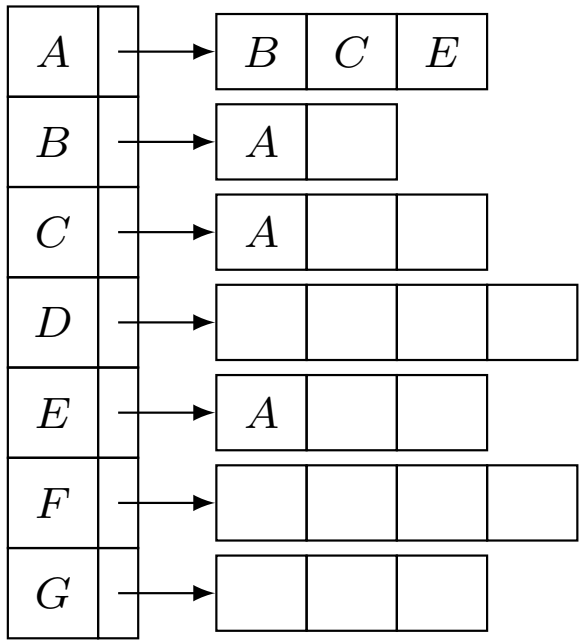




Representation av grafer

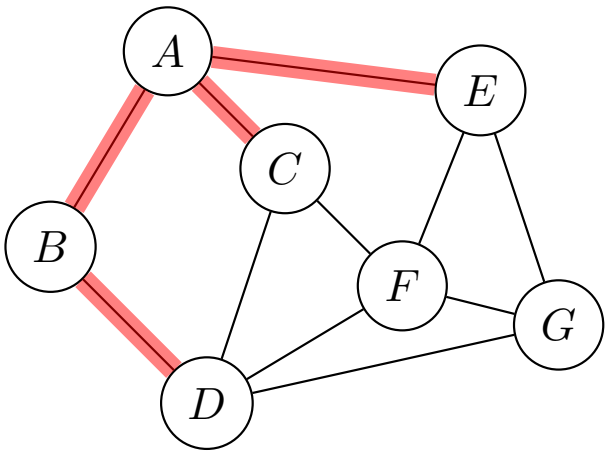
Grannlista

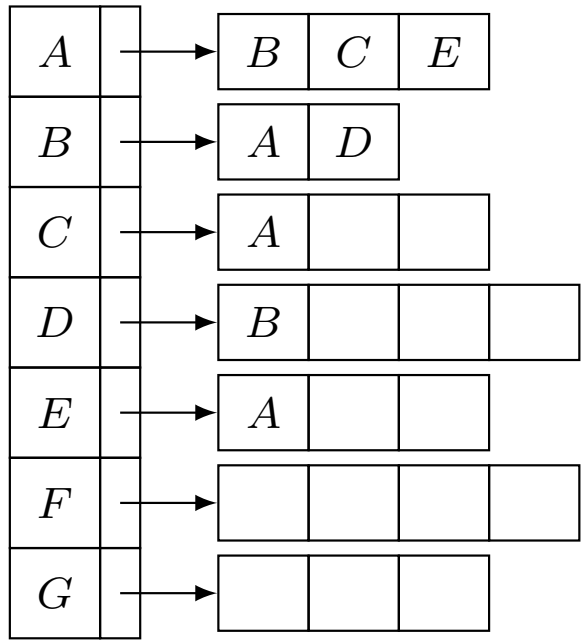




Representation av grafer

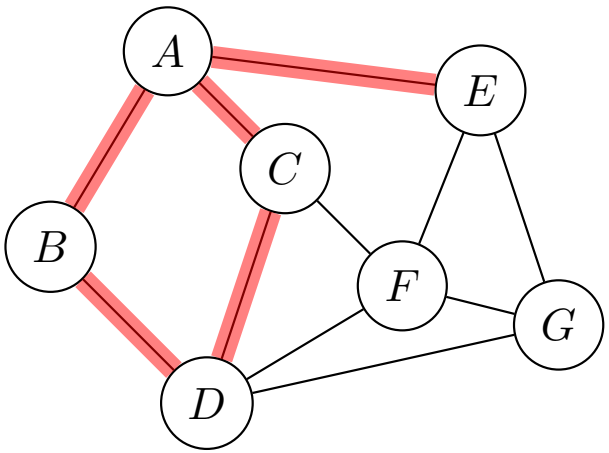
Grannlista

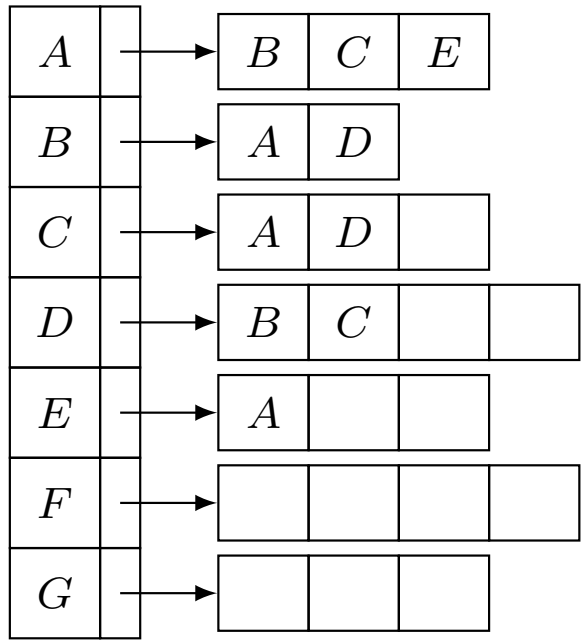




Representation av grafer

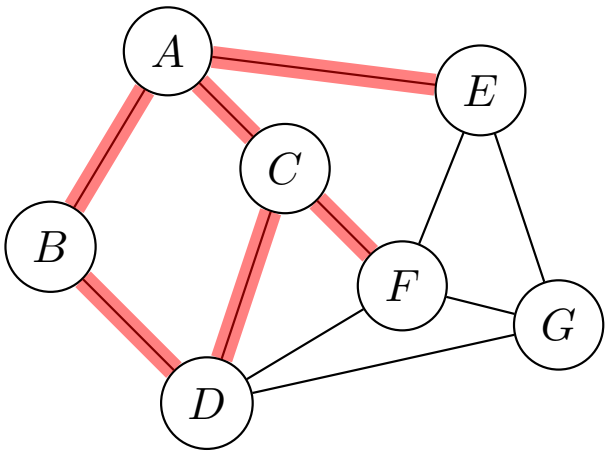
Grannlista

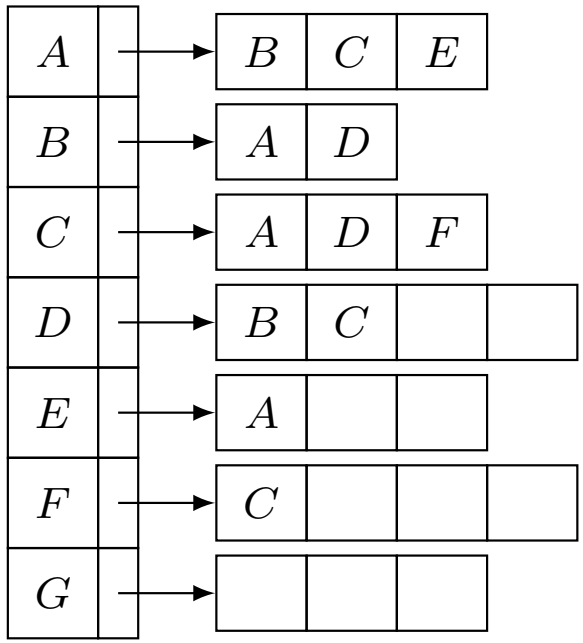




Representation av grafer

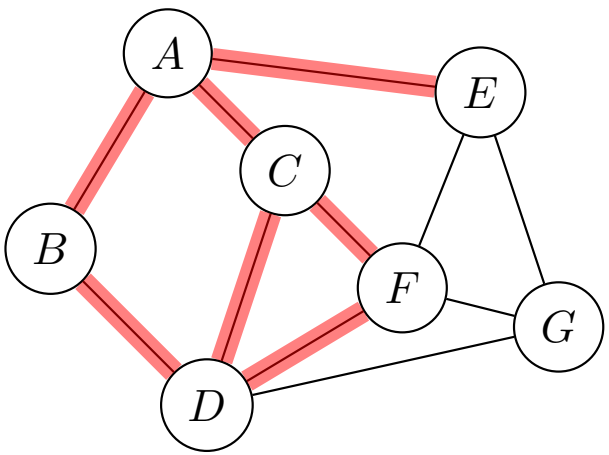
Grannlista

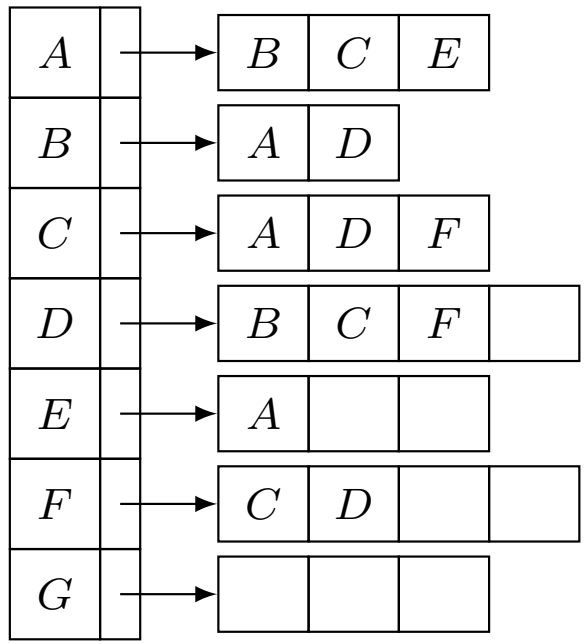




Representation av grafer

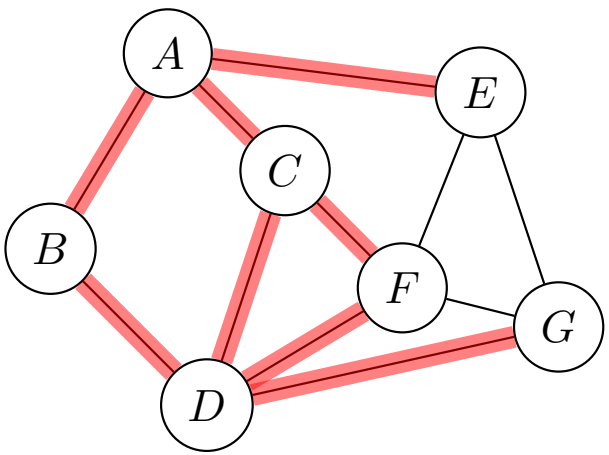
Grannlista

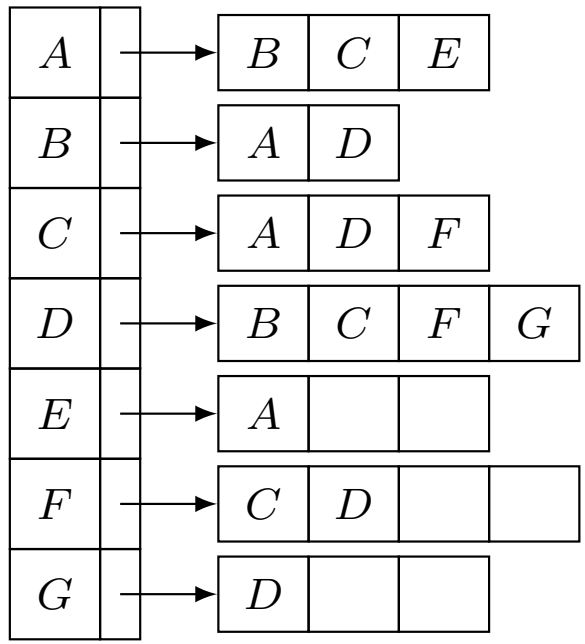




Representation av grafer

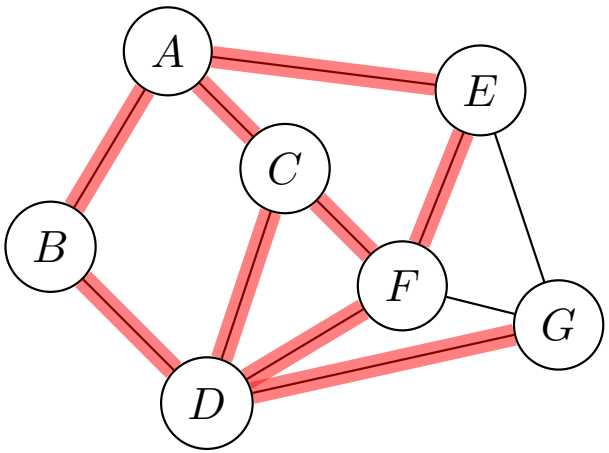
Grannlista

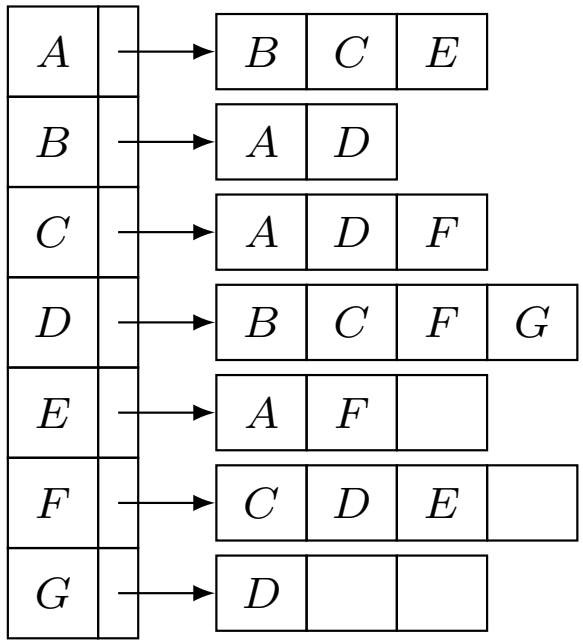




Representation av grafer

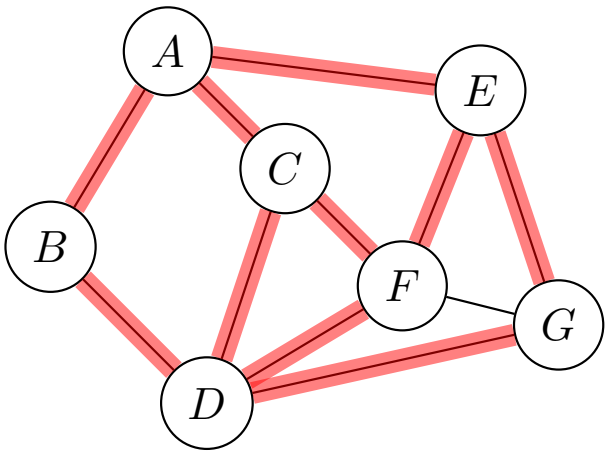
Grannlista

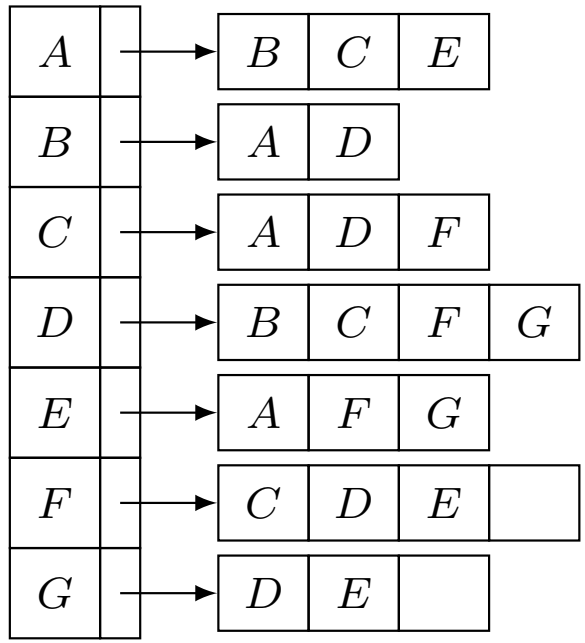




Representation av grafer

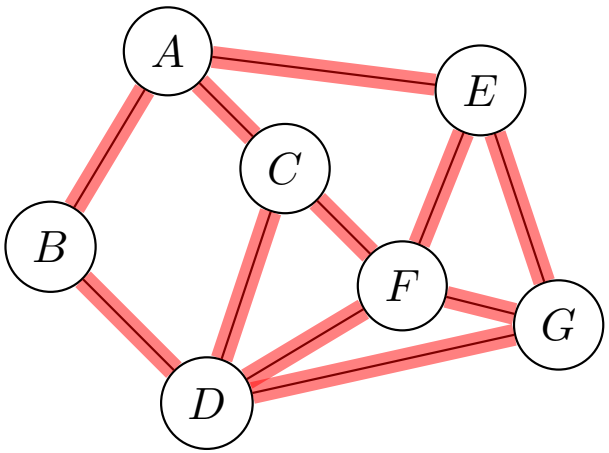
Grannlista

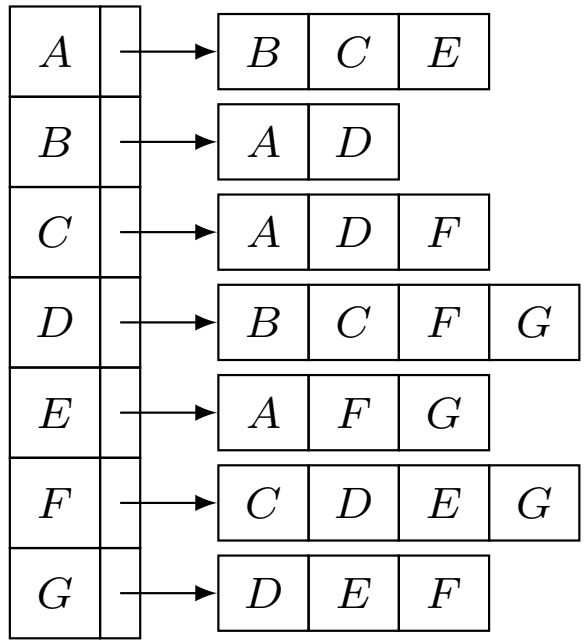




Representation av grafer

Grannlista





Representation av grafer

Jämförelse grannmatris och grannlista

Grannmatris

- Insättning nod: $\mathcal{O}\left(|V|^2\right)$
- Insättning båge: $\mathcal{O}\left(1\right)$
- Ta bort nod: $\mathcal{O}\left(|V|^2\right)$
- Ta bort båge: $\mathcal{O}\left(1\right)$
- Hitta nod: $\mathcal{O}\left(\log|V|\right)$
- Hitta båge: $\mathcal{O}\left(1\right)$
- Minne: $\mathcal{O}\left(|V|+|V|^2\right)$

Grannlista

- Insättning nod: $\mathcal{O}\left(1\right)$
- Insättning båge: $\mathcal{O}\left(1\right)$
- Ta bort nod: $\mathcal{O}\left(|V|\right)$
- Ta bort båge: $\mathcal{O}\left(\max_{v\in V}\left\{\deg(v)\right\}\right)$
- Hitta nod: $\mathcal{O}\left(1\right)$ amorterat
- Hitta båge: $\mathcal{O}\left(|V|\right)$
- Minne: $\mathcal{O}\left(|V|+|E|\right)$

Representation av grafer

Jämförelse grannmatris och grannlista

Grannmatris

- Insättning nod: $\mathcal{O}\left(|V|^2\right)$
- Insättning båge: $\mathcal{O}\left(1\right)$
- Ta bort nod: $\mathcal{O}\left(|V|^2\right)$
- Ta bort båge: $\mathcal{O}\left(1\right)$
- Hitta nod: $\mathcal{O}\left(\log|V|\right)$
- Hitta båge: $\mathcal{O}\left(1\right)$
- Minne: $\mathcal{O}\left(|V|+|V|^2\right)$

Grannlista

- Insättning nod: $\mathcal{O}\left(1\right)$
- Insättning båge: $\mathcal{O}\left(1\right)$
- Ta bort nod: $\mathcal{O}\left(|V|\right)$
- Ta bort båge: $\mathcal{O}\left(\max_{v\in V}\left\{\deg(v)\right\}\right)$
- Hitta nod: $\mathcal{O}\left(1\right)$ amorterat
- Hitta båge: $\mathcal{O}\left(|V|\right)$
- Minne: $\mathcal{O}\left(|V|+|E|\right)$

Ingen klockren vinnare...

Grafsökning

Länkad representation

```
1  class Node
2  {
3  public:
4    int data;
5    vector<Node*> edges;
6  };
```

Representation av grafer

Implicita grafer

```
1 vector<int> get_neighbours(int node)
2 {
3   vector<int> neighbours { };
4   // beräkna alla grannar till `node`
5   return neighbours;
6 }
```

Representation av grafer

Hitta alla noder

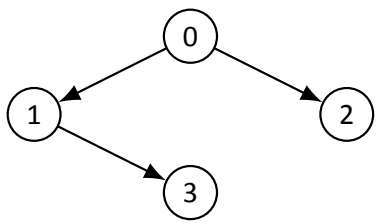
Hur kan vi hitta alla noder för dessa representationer?

1 Representation av grafer
2 Grafsökning
3 Heap

LINKÖPING UNIVERSITY

Grafsökning

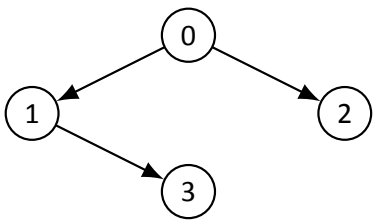
Rekursiv sökning – länkad struktur




```
1  vector<Node*> find_nodes(Node* node)
2  {
3    if (node == nullptr) return { };
4
5    vector<Node*> neighbours { };
6    for (Node* next : node->edges)
7    {
8      vector<Node*> nodes {
9        find_nodes(next);
10     };
11     for (Node* curr : nodes)
12       neighbours.push_back(curr);
13   }
14
15   return neighbours;
16 }
```

Grafsökning

Rekursiv sökning – länkad struktur

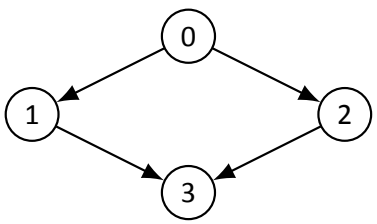



```
1  vector<Node*> find_nodes(Node* node)
2  {
3    if (node == nullptr) return { };
4
5    vector<Node*> neighbours { };
6    for (Node* next : node->edges)
7    {
8      vector<Node*> nodes {
9        find_nodes(next);
10     };
11     for (Node* curr : nodes)
12       neighbours.push_back(curr);
13   }
14
15   return neighbours;
16 }
```

Funkar!

Grafsökning

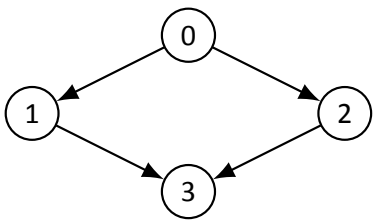
Ett problem




```
1  vector<Node*> find_nodes(Node* node)
2  {
3    if (node == nullptr) return { };
4
5    vector<Node*> nodes { node };
6    for (Node* next : node->edges)
7    {
8      vector<Node*> neighbours {
9        find_nodes(next);
10     };
11     for (Node* curr : neighbours)
12       nodes.push_back(curr);
13   }
14
15   return nodes;
16 }
```

Grafsökning

Ett problem

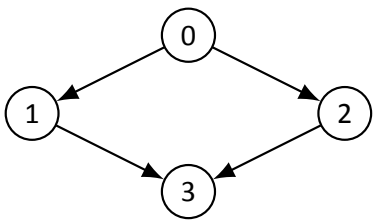



```
1  vector<Node*> find_nodes(Node* node)
2  {
3    if (node == nullptr) return { };
4
5    vector<Node*> nodes { node };
6    for (Node* next : node->edges)
7    {
8      vector<Node*> neighbours {
9        find_nodes(next);
10     };
11     for (Node* curr : neighbours)
12       nodes.push_back(curr);
13   }
14
15   return nodes;
16 }
```

dubletter...

Grafsökning

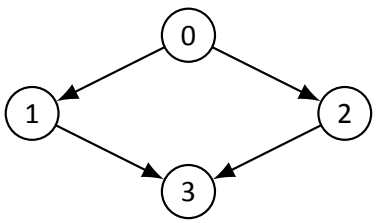
Lösning!




```
1  set<Node*> find_nodes(Node* node)
2  {
3    if (node == nullptr) return { };
4
5    set<Node*> nodes { node };
6    for (Node* next : node->edges)
7    {
8      set<Node*> neighbours {
9        find_nodes(next);
10     };
11     for (Node* curr : neighbours)
12       nodes.push_back(curr);
13   }
14
15   return nodes;
16 }
```

Grafsökning

Lösning!

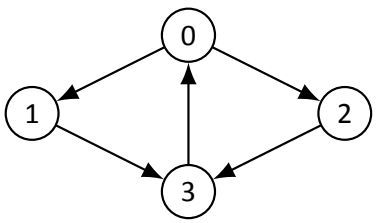



```
1  set<Node*> find_nodes(Node* node)
2  {
3    if (node == nullptr) return { };
4
5    set<Node*> nodes { node };
6    for (Node* next : node->edges)
7    {
8      set<Node*> neighbours {
9        find_nodes(next);
10     };
11     for (Node* curr : neighbours)
12       nodes.push_back(curr);
13   }
14
15   return nodes;
16 }
```

Funkar!

Grafsökning

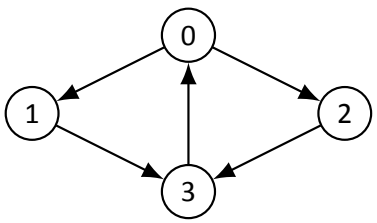
Ett nytt problem...




```
1  set<Node*> find_nodes(Node* node)
2  {
3    if (node == nullptr) return { };
4
5    set<Node*> nodes { node };
6    for (Node* next : node->edges)
7    {
8      set<Node*> neighbours {
9        find_nodes(next);
10     };
11     for (Node* curr : neighbours)
12       nodes.push_back(curr);
13   }
14
15   return nodes;
16 }
```

Grafsökning

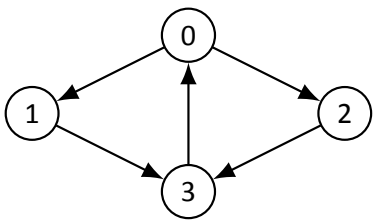
Ett nytt problem...




```
1  set<Node*> find_nodes(Node* node)
2  {
3    if (node == nullptr) return { };
4
5    set<Node*> nodes { node };
6    for (Node* next : node->edges)
7    {
8      set<Node*> neighbours {
9        find_nodes(next);
10     };
11     for (Node* curr : neighbours)
12       nodes.push_back(curr);
13   }
14
15   return nodes;
16 }
```

Grafsökning

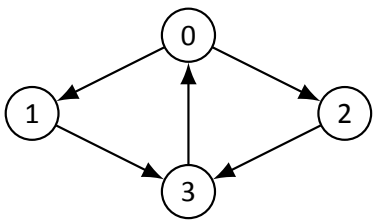
Djupet-först-sökning (DFS)




```
1  void find_nodes_help(Node* node, set<Node*>& found)
2  {
3    if (node == nullptr) return { };
4    if (found.count(node) > 0) return { };
5
6    found.insert(node);
7    for (Node* next : node->edges)
8      find_nodes_help(next, found);
9  }
10
11 set<Node*> find_nodes(Node* node)
12 {
13   set<Node*> nodes { };
14   find_nodes_help(node, nodes);
15   return nodes;
16 }
```

Grafsökning

Djupet-först-sökning (DFS)




```
1  void find_nodes_help(Node* node, set<Node*>& found)
2  {
3    if (node == nullptr) return { };
4    if (found.count(node) > 0) return { };
5
6    found.insert(node);
7    for (Node* next : node->edges)
8      find_nodes_help(next, found);
9  }
10
11 set<Node*> find_nodes(Node* node)
12 {
13   set<Node*> nodes { };
14   find_nodes_help(node, nodes);
15   return nodes;
16 }
```

Funkar!

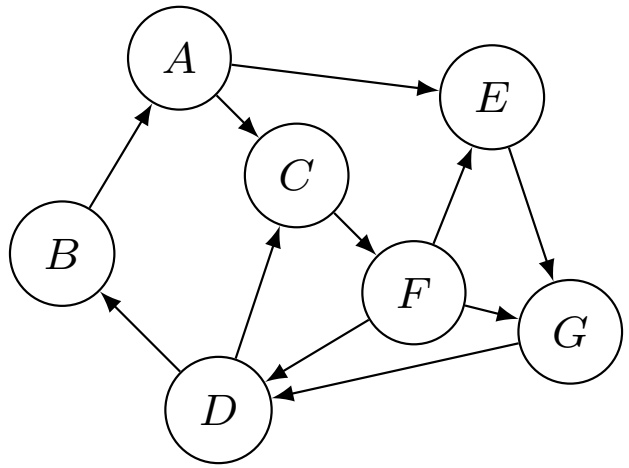
Grafsökning

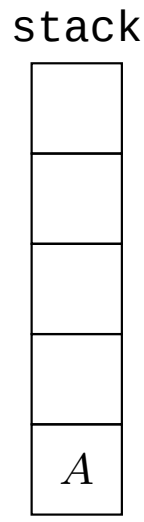
Iterativ variant

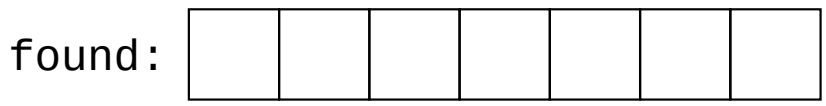
```
1  set<Node*> DFS(Node* start)
2  {
3    set<Node*> found;
4    stack<Node*> stack;
5    stack.push(start);
6    while (!stack.empty())
7    {
8      Node* node { stack.top() };
9      stack.pop();
10     if (node == nullptr) continue;
11     if (found.count(node) > 0) continue;
12     found.insert(node);
13     for (Node* next : node->edges)
14       stack.push(next);
15   }
16   return found;
17 }
```

Grafsökning

djupet-först-sökning (DFS)

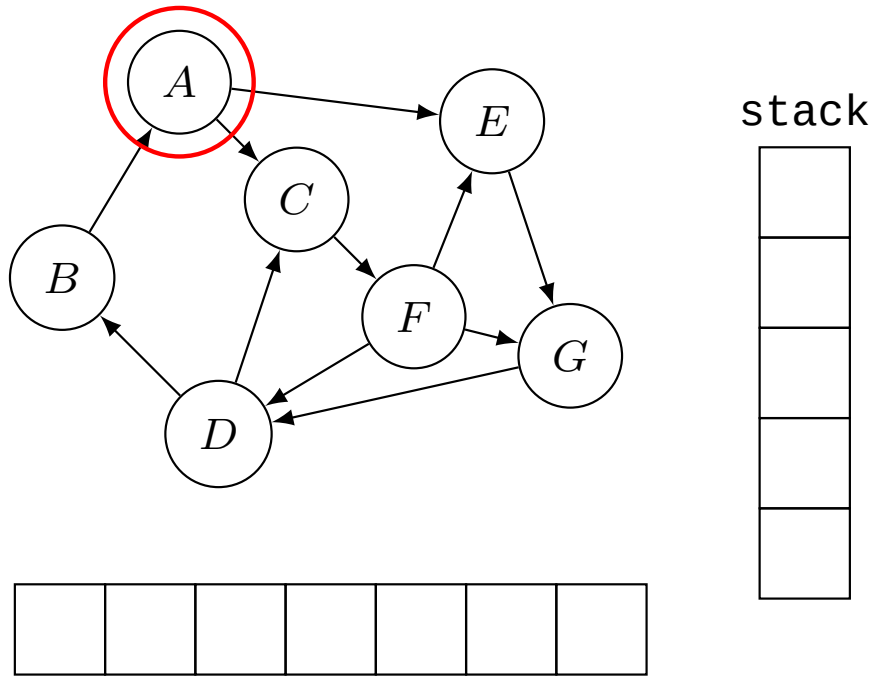






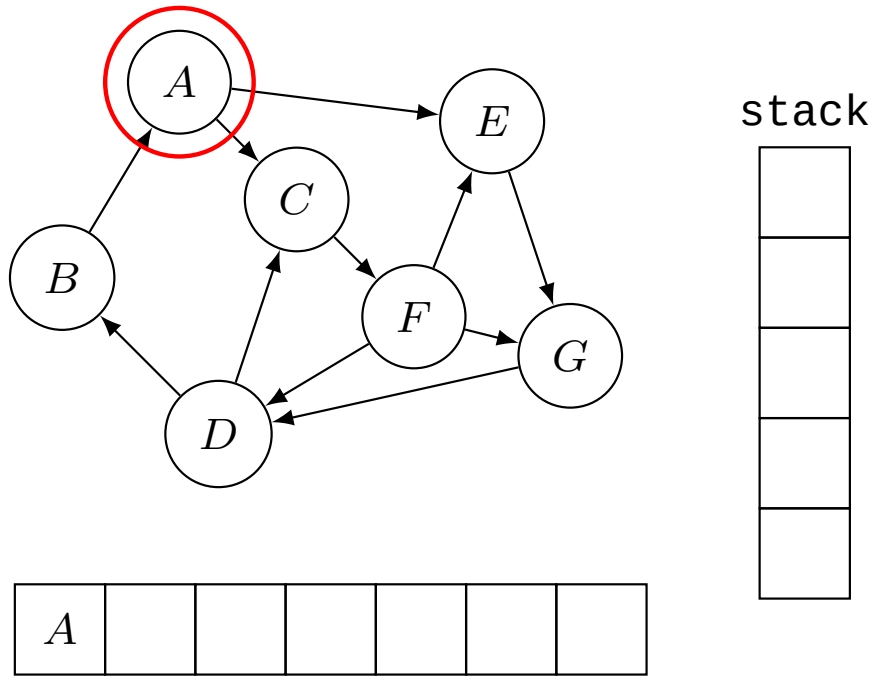
Grafsökning

djupet-först-sökning (DFS)



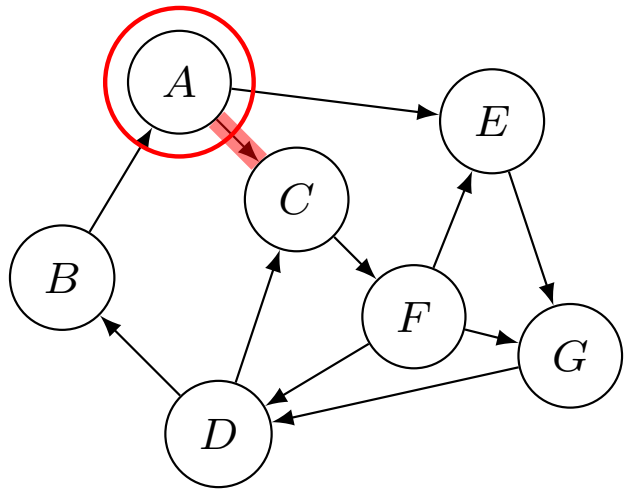
Grafsökning

djupet-först-sökning (DFS)

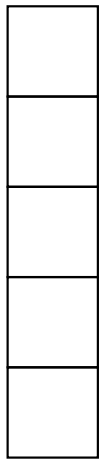


Grafsökning

djupet-först-sökning (DFS)



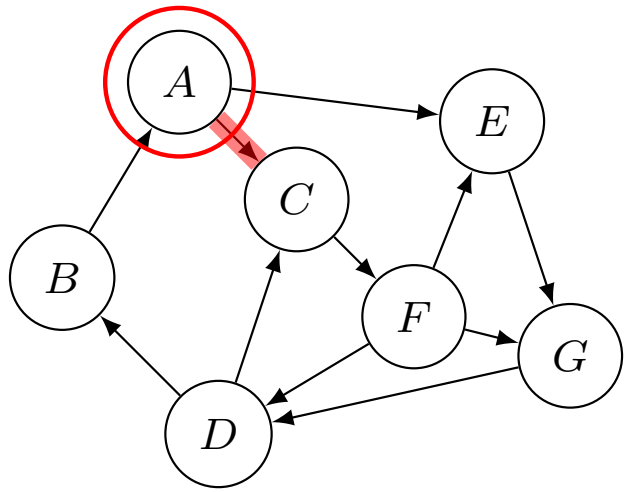
stack



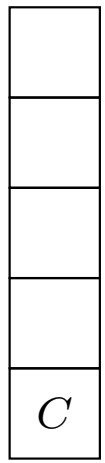
found:

Grafsökning

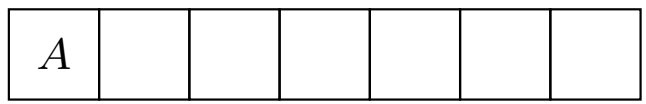
djupet-först-sökning (DFS)



stack

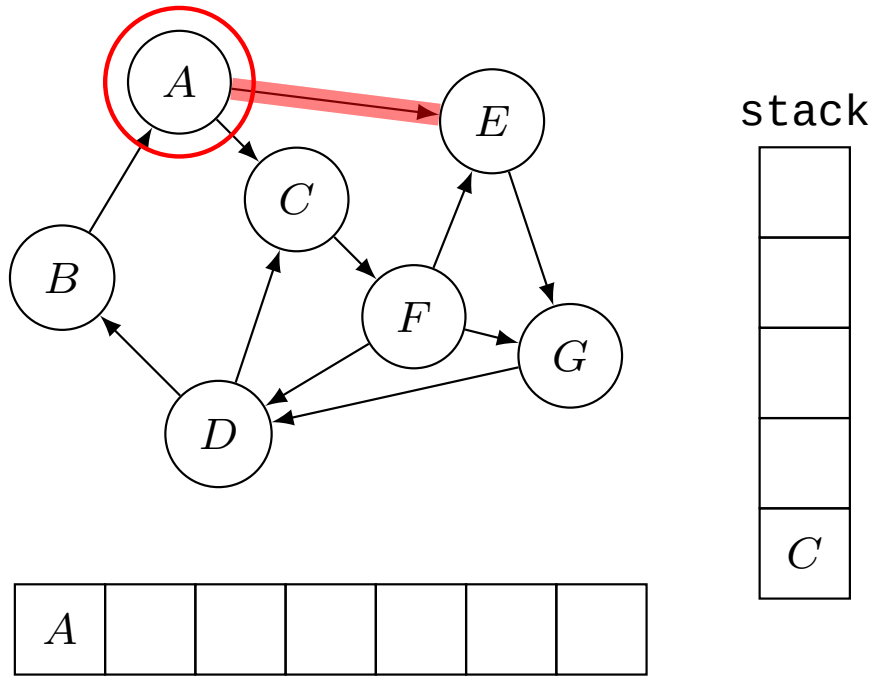


found:



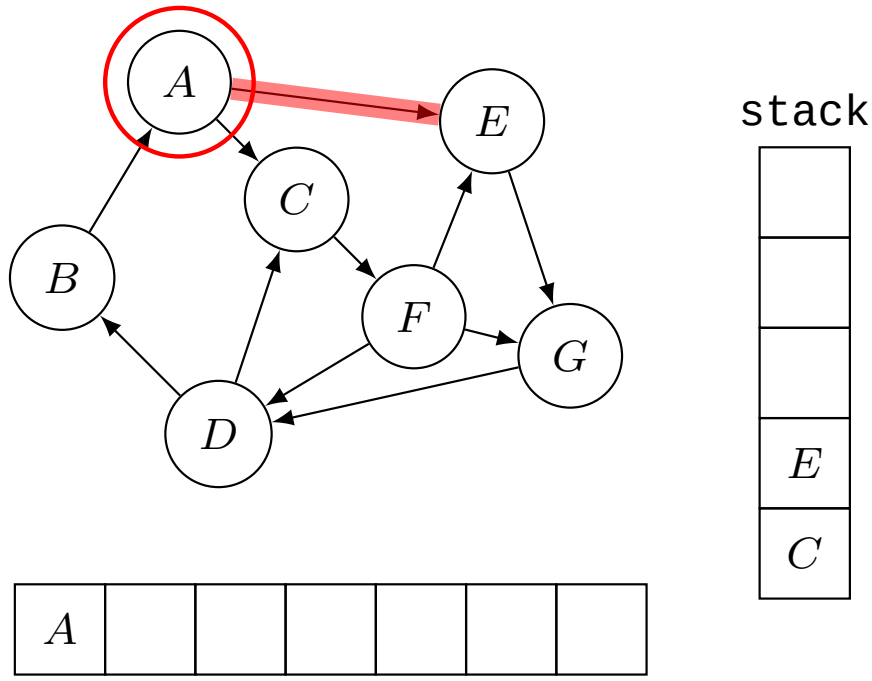
Grafsökning

djupet-först-sökning (DFS)



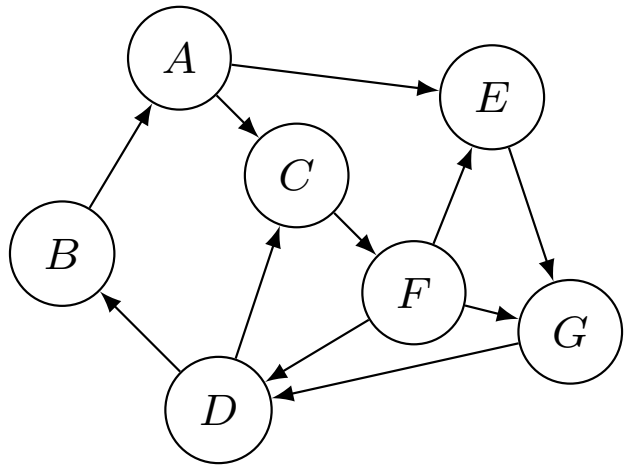
Grafsökning

djupet-först-sökning (DFS)

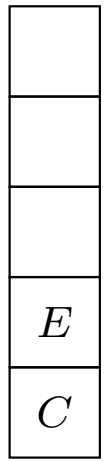


Grafsökning

djupet-först-sökning (DFS)



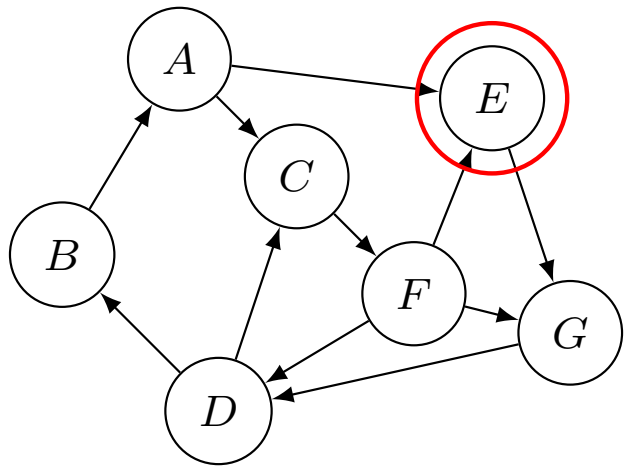
stack



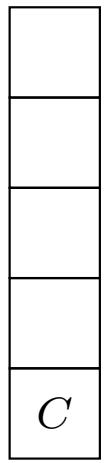
found:

Grafsökning

djupet-först-sökning (DFS)



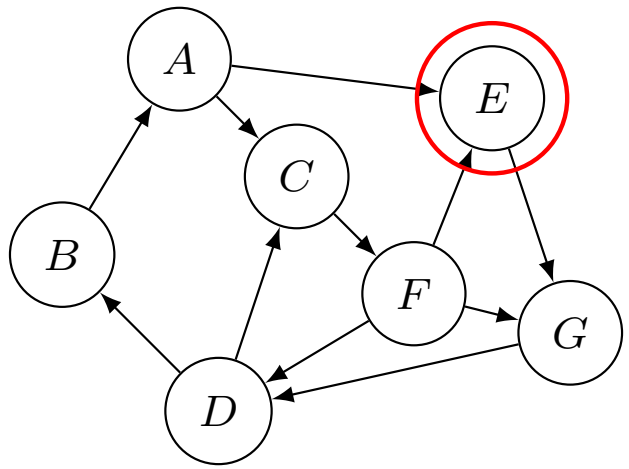
stack



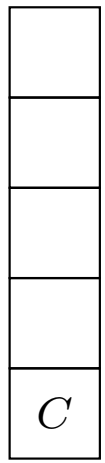
found:

Grafsökning

djupet-först-sökning (DFS)



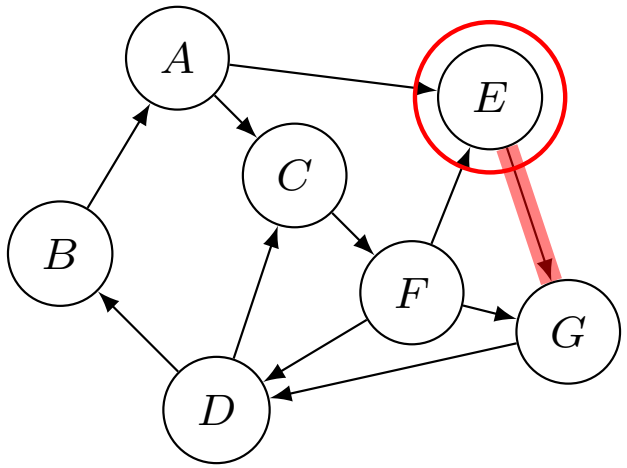
stack



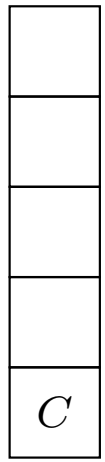
found:

Grafsökning

djupet-först-sökning (DFS)



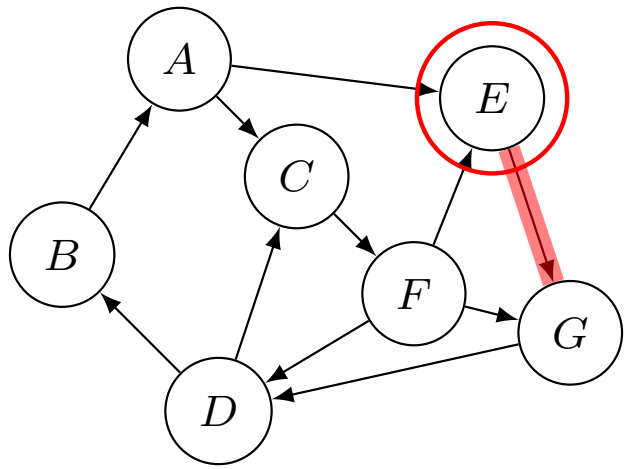
stack



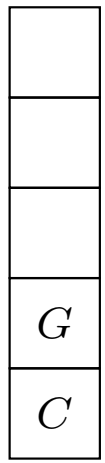
found:

Grafsökning

djupet-först-sökning (DFS)



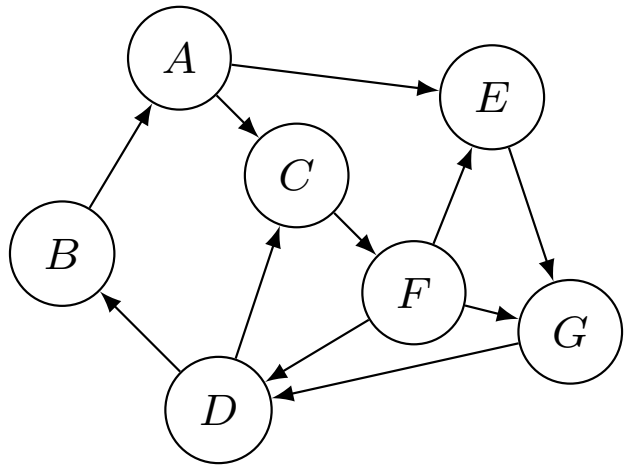
stack



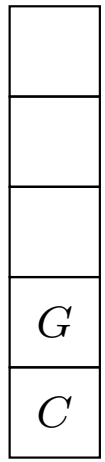
found:

Grafsökning

djupet-först-sökning (DFS)



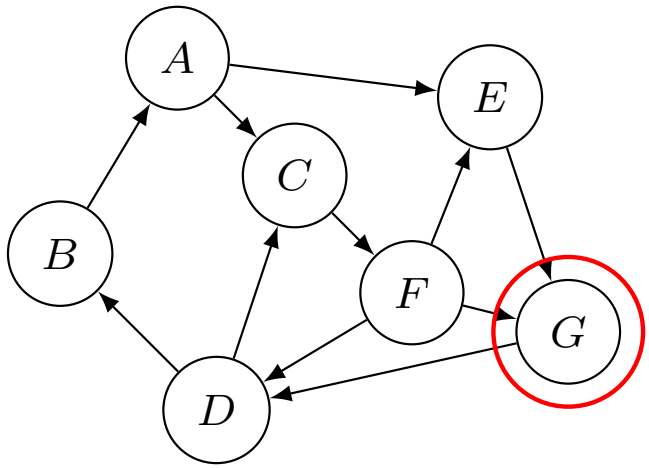
stack

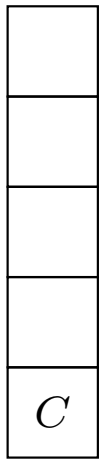


found:

Grafsökning

djupet-först-sökning (DFS)

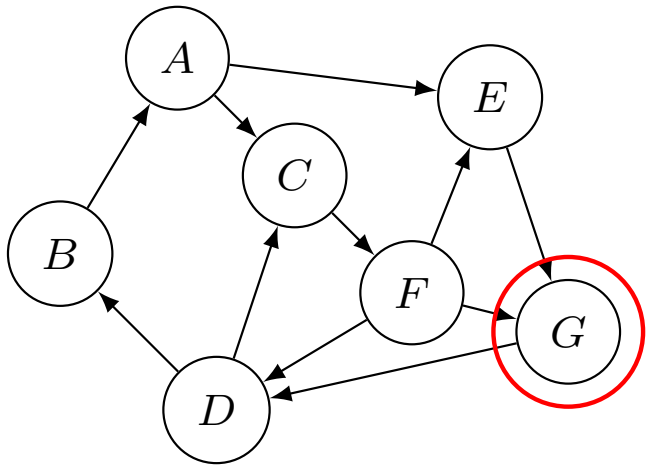




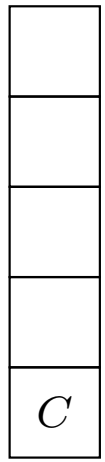
found: A E

Grafsökning

djupet-först-sökning (DFS)



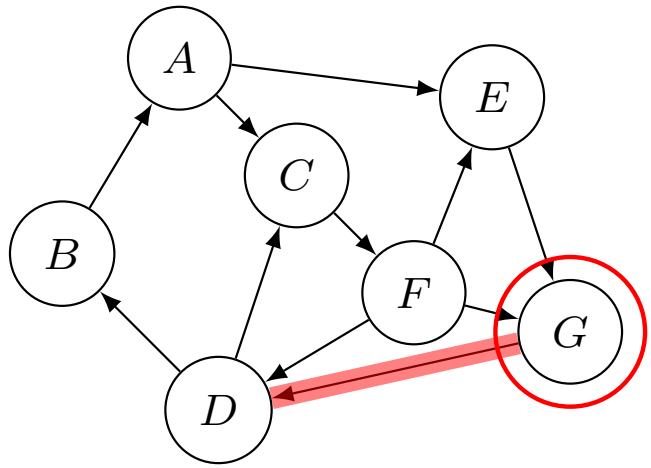
stack



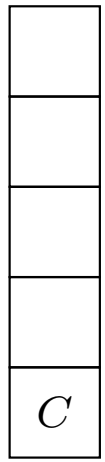
found:

Grafsökning

djupet-först-sökning (DFS)



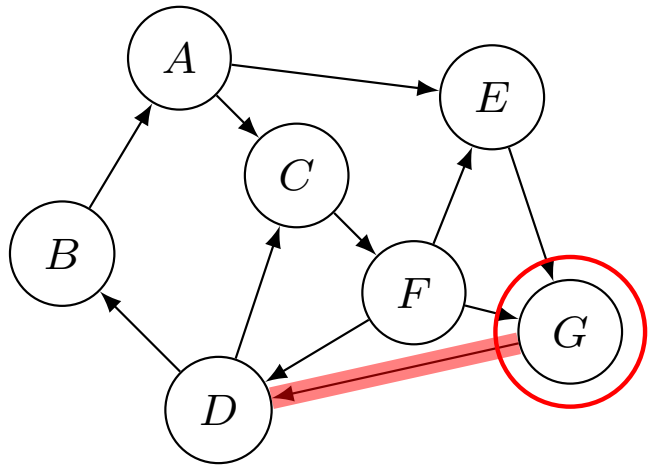
stack



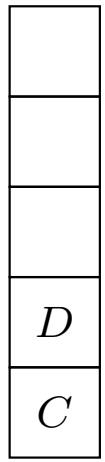
found:

Grafsökning

djupet-först-sökning (DFS)



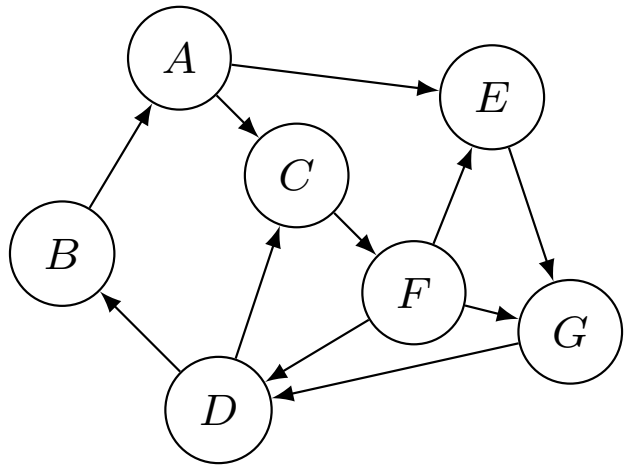
stack



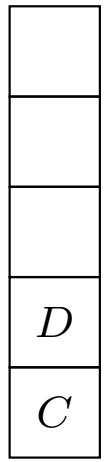
found:

Grafsökning

djupet-först-sökning (DFS)



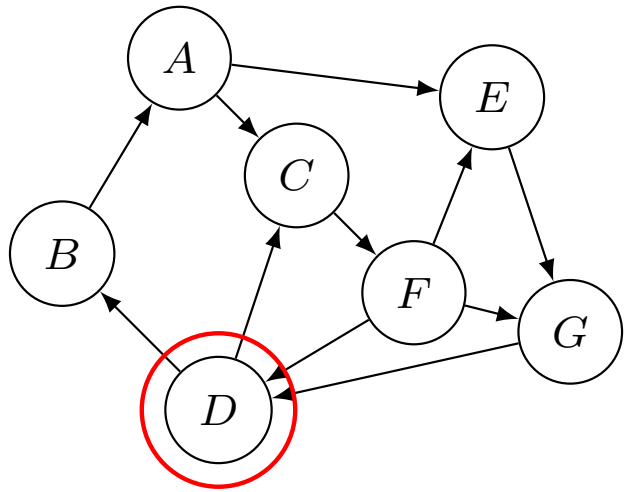
stack

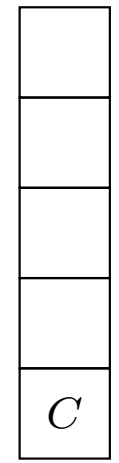


found:

Grafsökning

djupet-först-sökning (DFS)

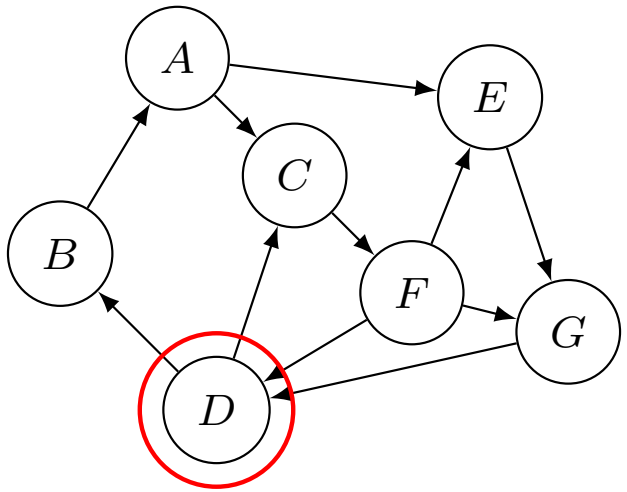




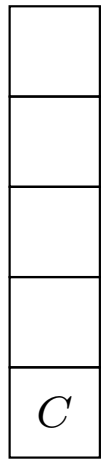
found: A E G

Grafsökning

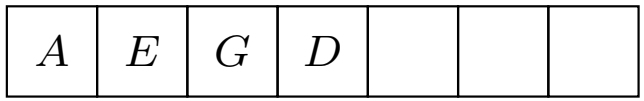
djupet-först-sökning (DFS)



stack

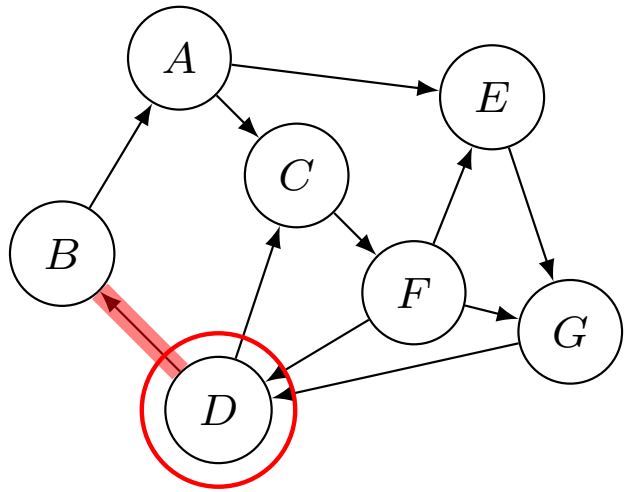


found:

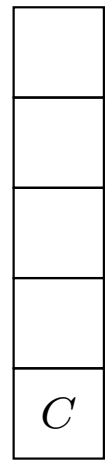


Grafsökning

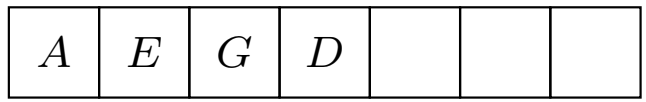
djupet-först-sökning (DFS)



stack

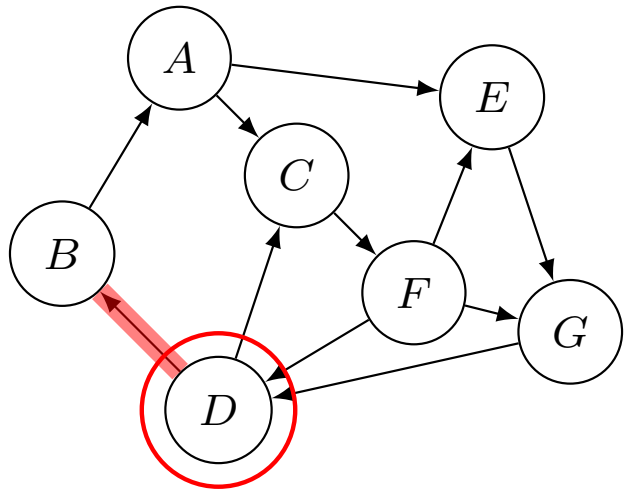


found:

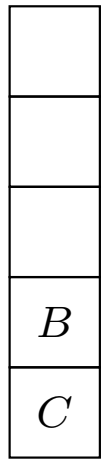


Grafsökning

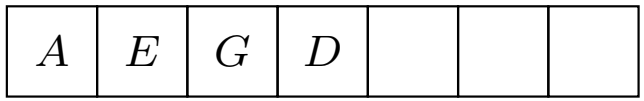
djupet-först-sökning (DFS)



stack

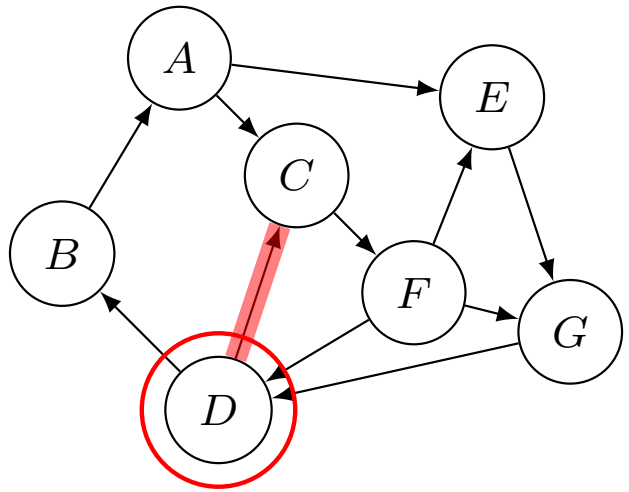


found:

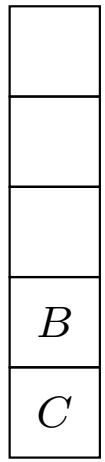


Grafsökning

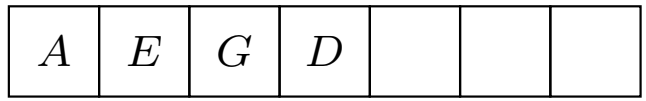
djupet-först-sökning (DFS)



stack

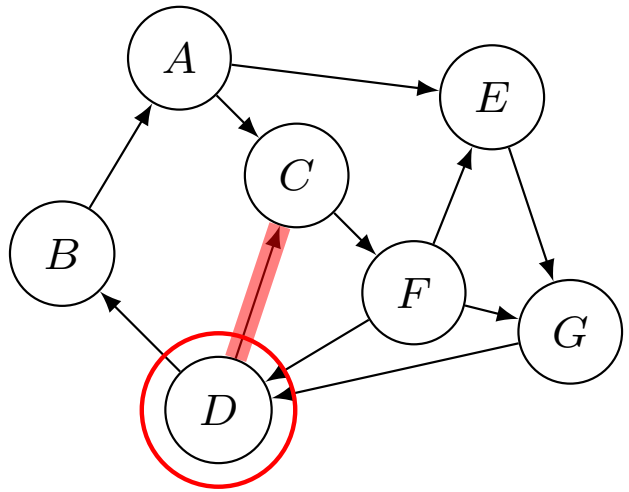


found:

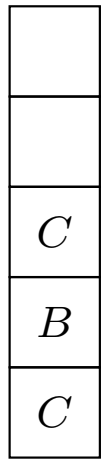


Grafsökning

djupet-först-sökning (DFS)



stack

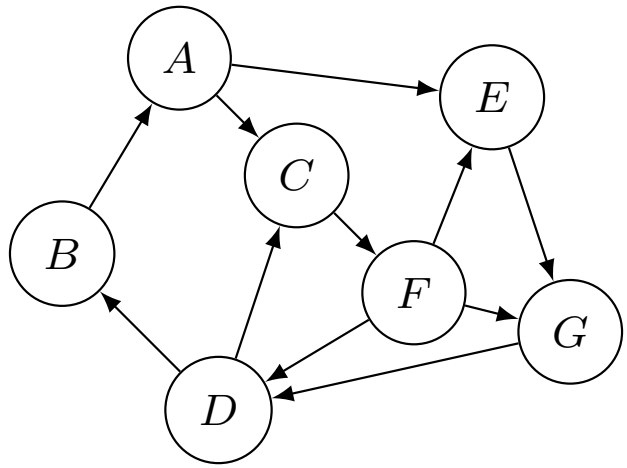


found:

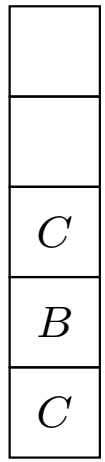
A	E	G	D			

Grafsökning

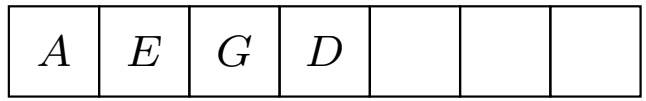
djupet-först-sökning (DFS)



stack

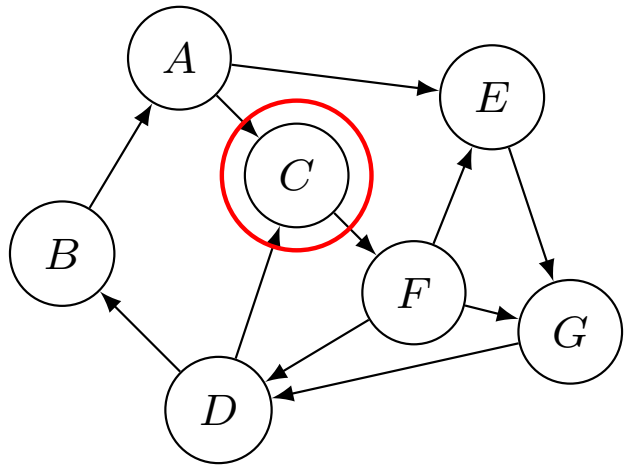


found:

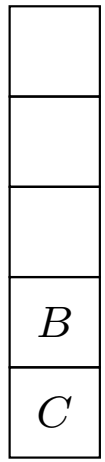


Grafsökning

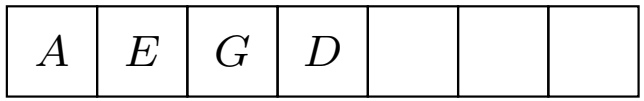
djupet-först-sökning (DFS)



stack

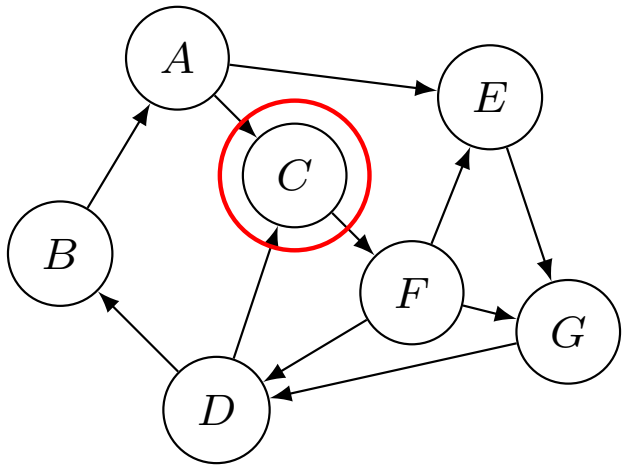


found:

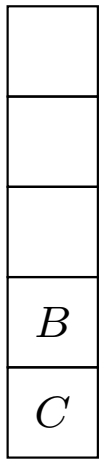


Grafsökning

djupet-först-sökning (DFS)



stack

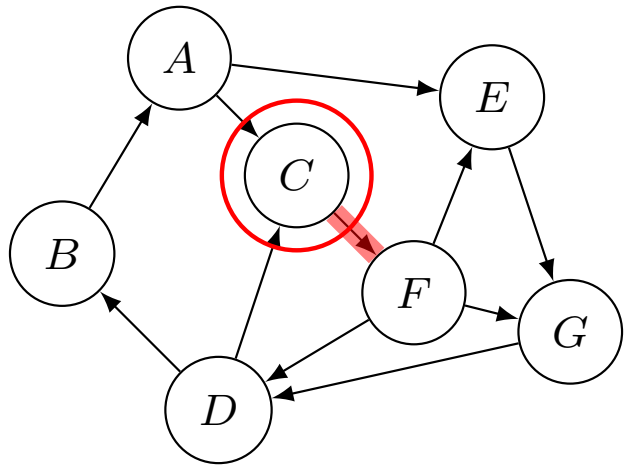


found:

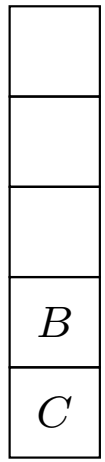
A	E	G	D	C		

Grafsökning

djupet-först-sökning (DFS)



stack

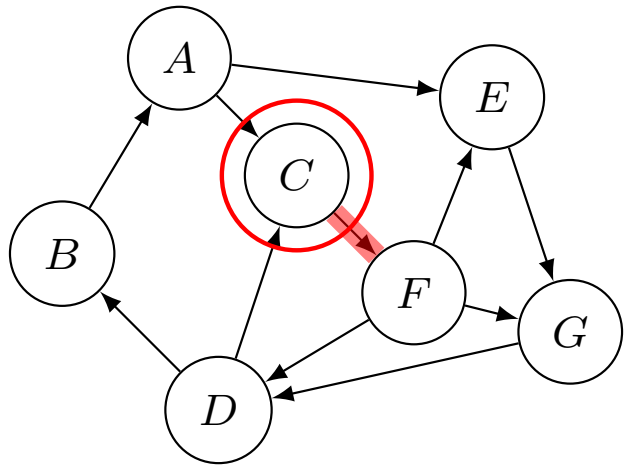


found:

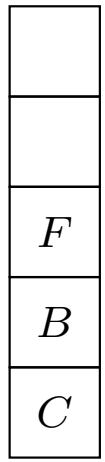
A	E	G	D	C		

Grafsökning

djupet-först-sökning (DFS)



stack

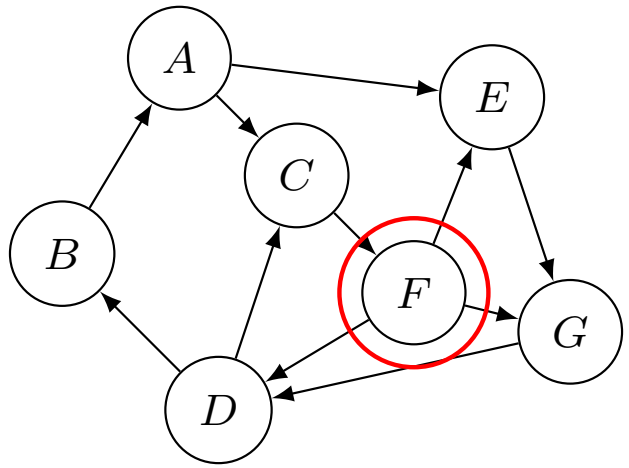


found:

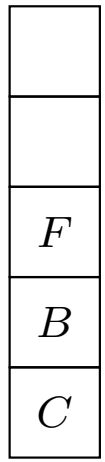
A	E	G	D	C		

Grafsökning

djupet-först-sökning (DFS)



stack

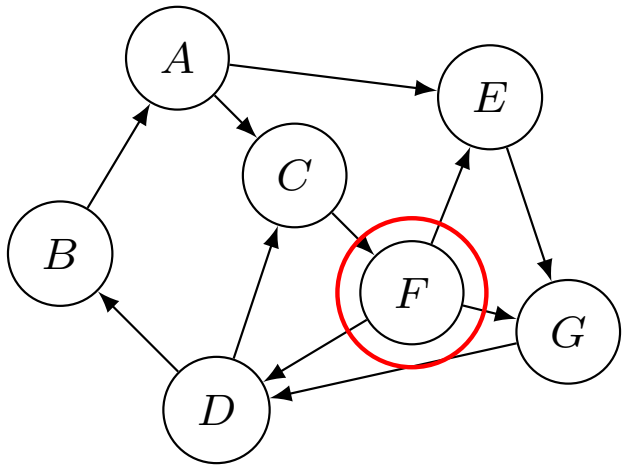


found:

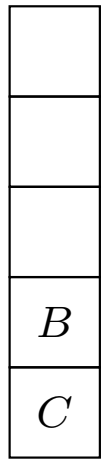
A	E	G	D	C		

Grafsökning

djupet-först-sökning (DFS)



stack

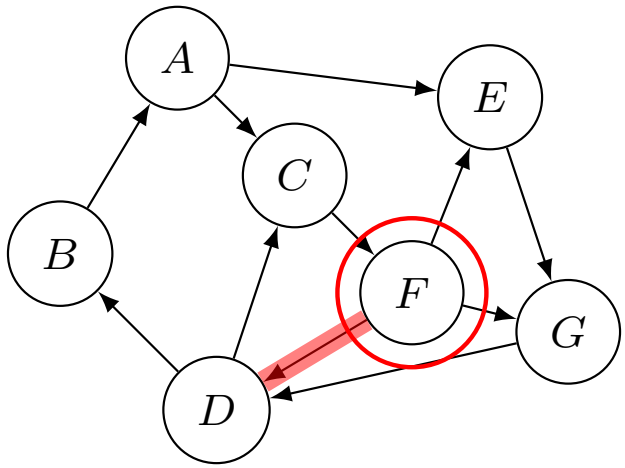


found:

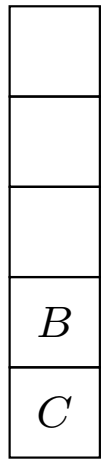
A	E	G	D	C	F	

Grafsökning

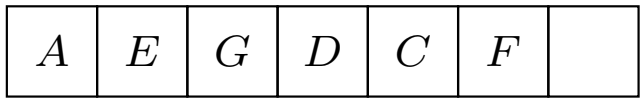
djupet-först-sökning (DFS)



stack

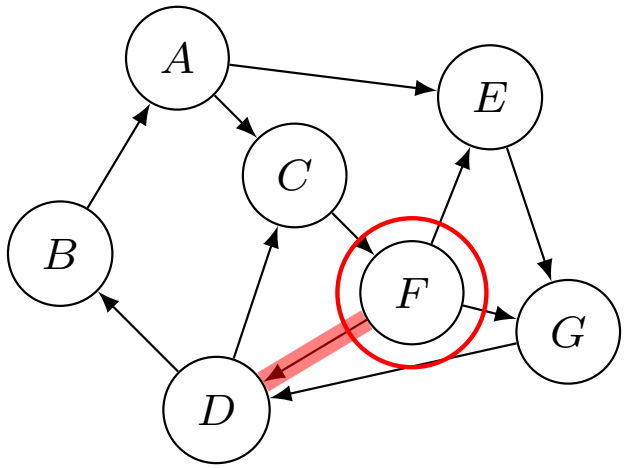


found:

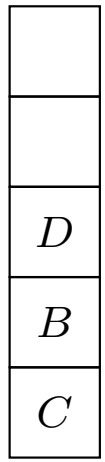


Grafsökning

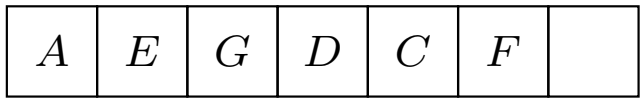
djupet-först-sökning (DFS)



stack

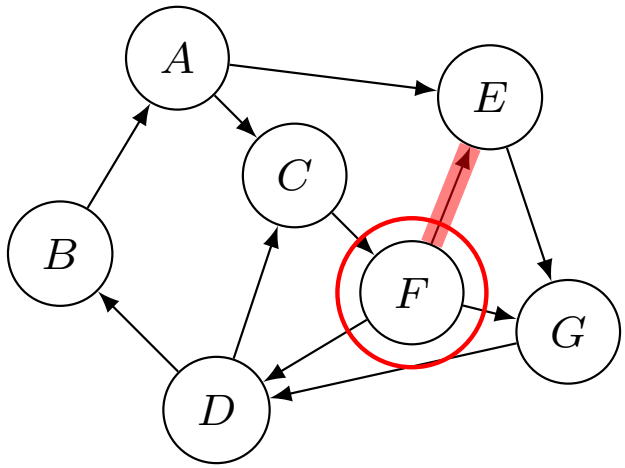


found:

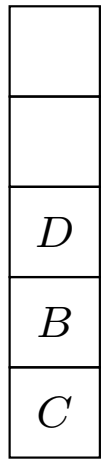


Grafsökning

djupet-först-sökning (DFS)



stack

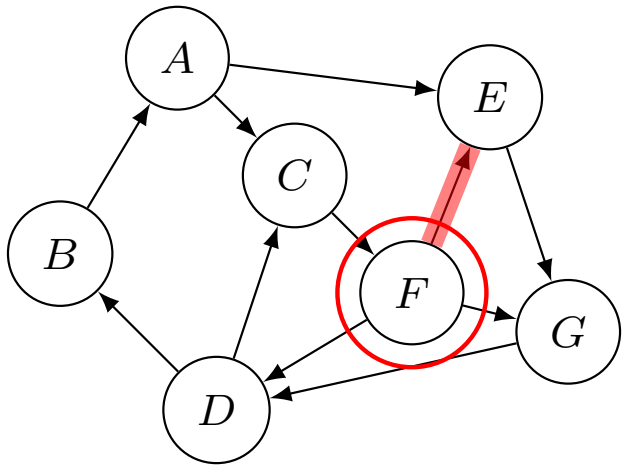


found:

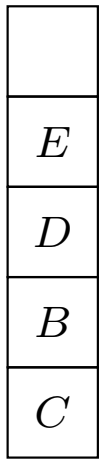
A	E	G	D	C	F	

Grafsökning

djupet-först-sökning (DFS)



stack

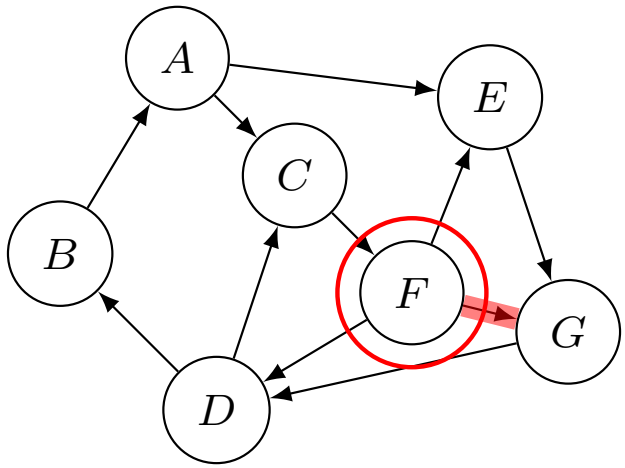


found:

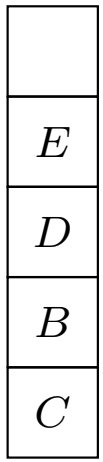
A	E	G	D	C	F	

Grafsökning

djupet-först-sökning (DFS)



stack

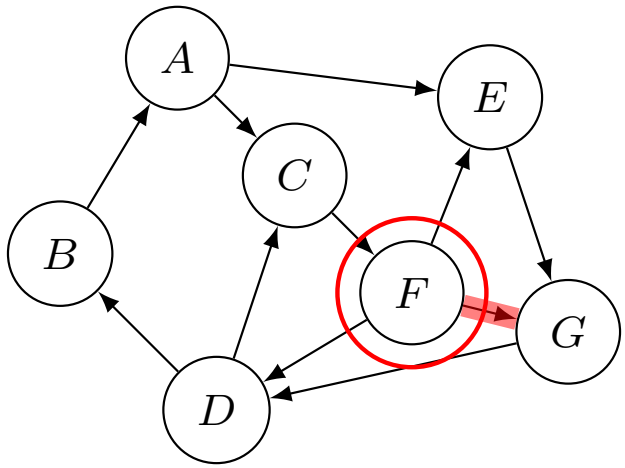


found:

A	E	G	D	C	F	

Grafsökning

djupet-först-sökning (DFS)



stack

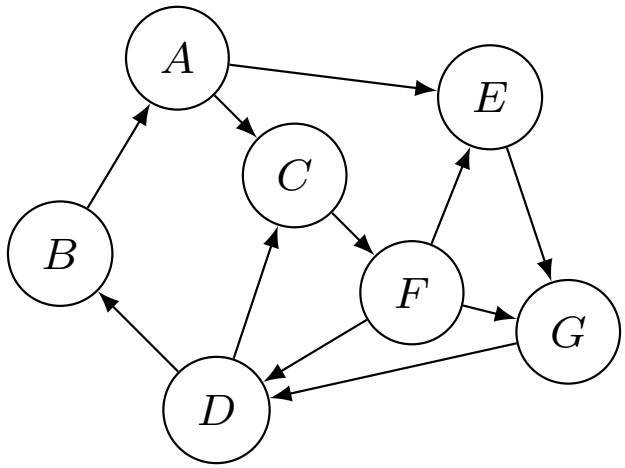
G
E
D
B
C

found:

A	E	G	D	C	F	

Grafsökning

djupet-först-sökning (DFS)



stack

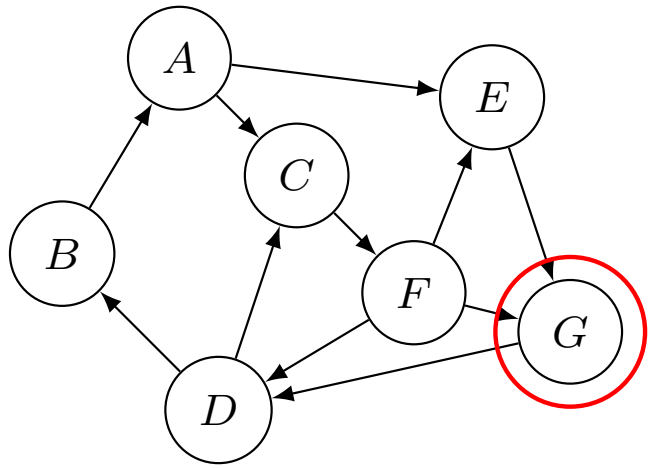
G
E
D
B
C

found:

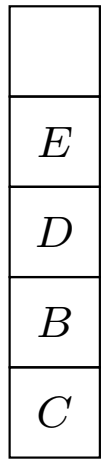
A	E	G	D	C	F	

Grafsökning

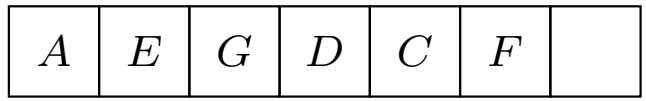
djupet-först-sökning (DFS)



stack

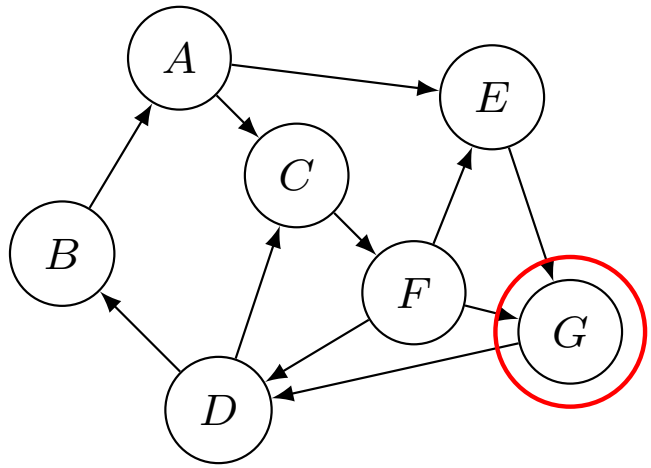


found:

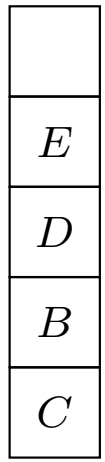


Grafsökning

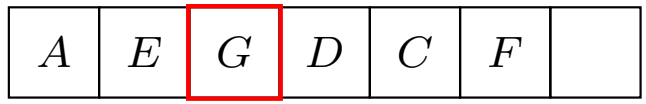
djupet-först-sökning (DFS)



stack

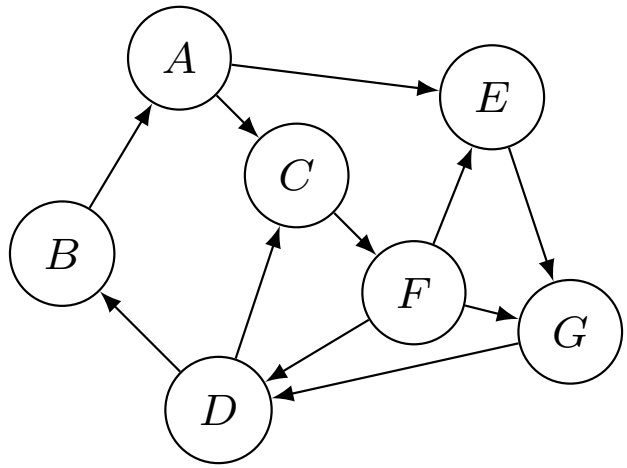


found:

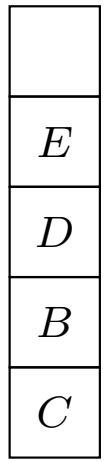


Grafsökning

djupet-först-sökning (DFS)



stack

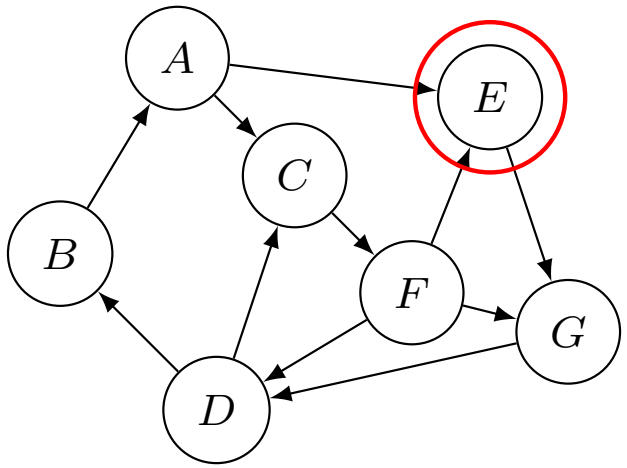


found:

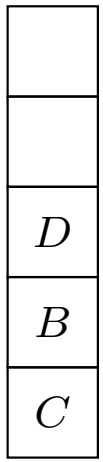
A	E	G	D	C	F	

Grafsökning

djupet-först-sökning (DFS)



stack

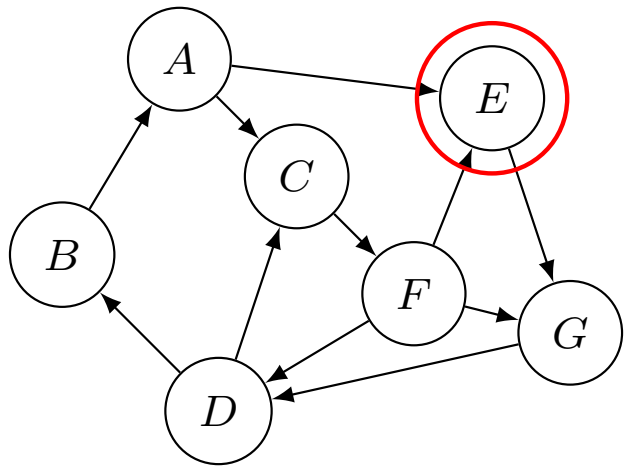


found:

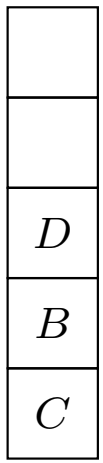
A	E	G	D	C	F	

Grafsökning

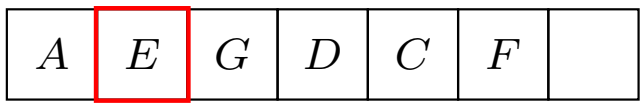
djupet-först-sökning (DFS)



stack

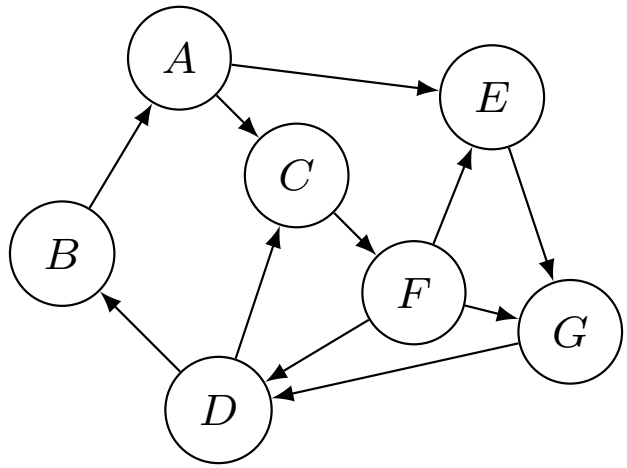


found:

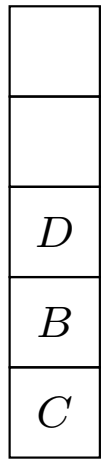


Grafsökning

djupet-först-sökning (DFS)



stack

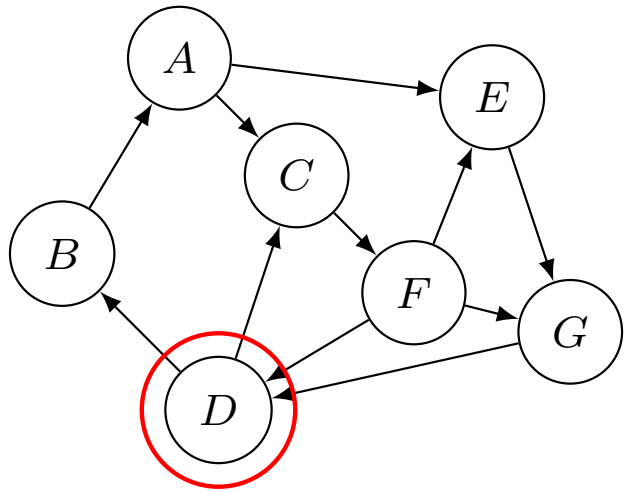


found:

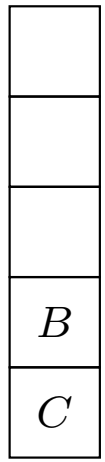
A	E	G	D	C	F	

Grafsökning

djupet-först-sökning (DFS)



stack

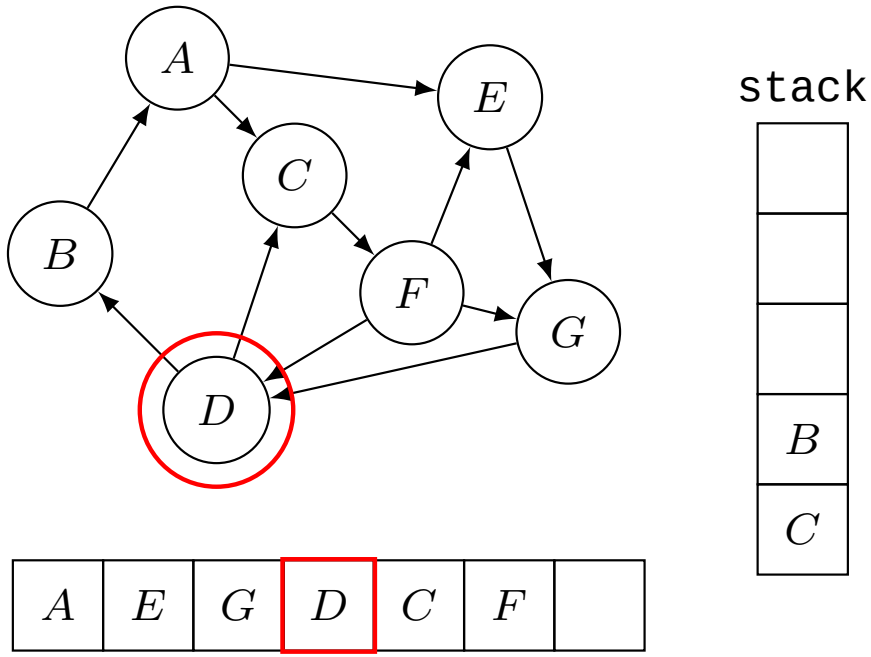


found:

A	E	G	D	C	F	

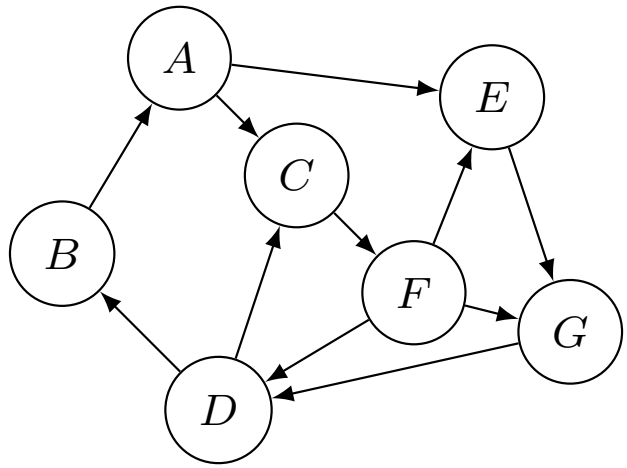
Grafsökning

djupet-först-sökning (DFS)

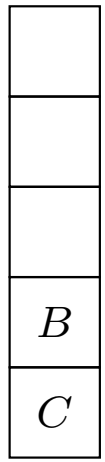


Grafsökning

djupet-först-sökning (DFS)



stack

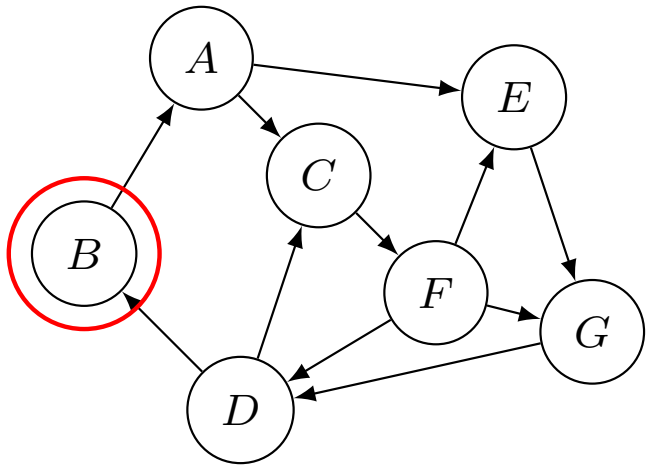


found:

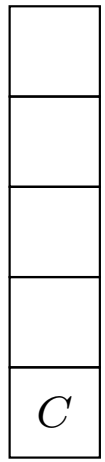
A	E	G	D	C	F	

Grafsökning

djupet-först-sökning (DFS)



stack

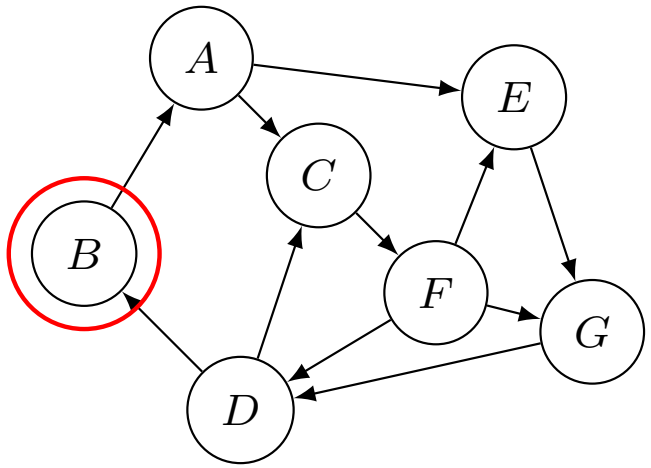


found:

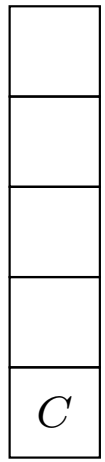
A	E	G	D	C	F	

Grafsökning

djupet-först-sökning (DFS)



stack

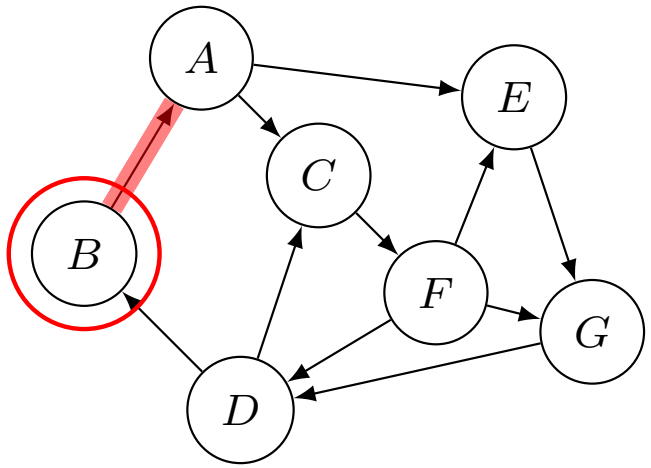


found:

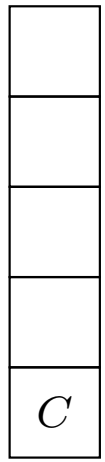
A	E	G	D	C	F	B

Grafsökning

djupet-först-sökning (DFS)



stack

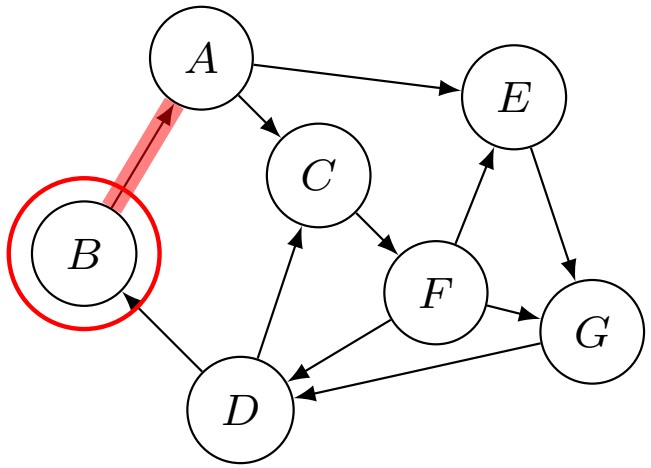


found:

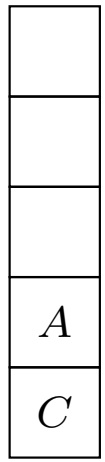
A	E	G	D	C	F	B

Grafsökning

djupet-först-sökning (DFS)



stack

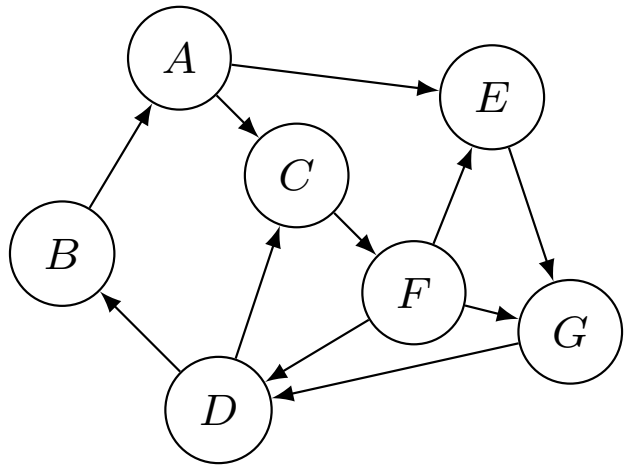


found:

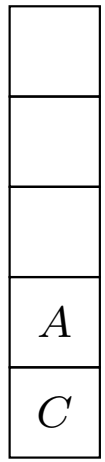
A	E	G	D	C	F	B

Grafsökning

djupet-först-sökning (DFS)



stack

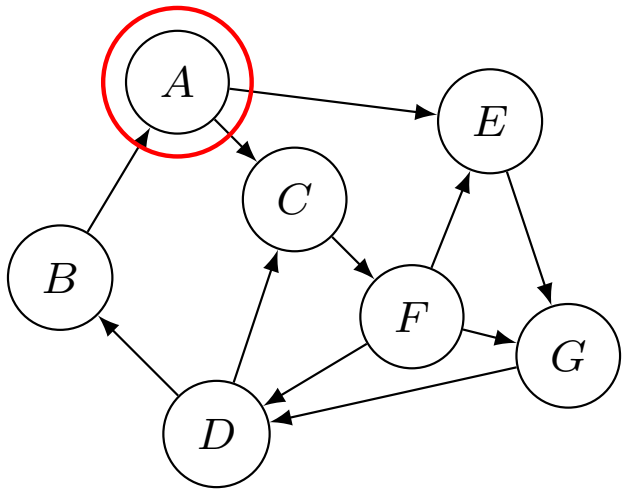


found:

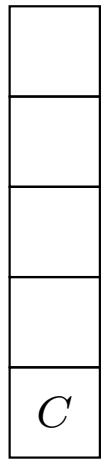
A	E	G	D	C	F	B

Grafsökning

djupet-först-sökning (DFS)



stack

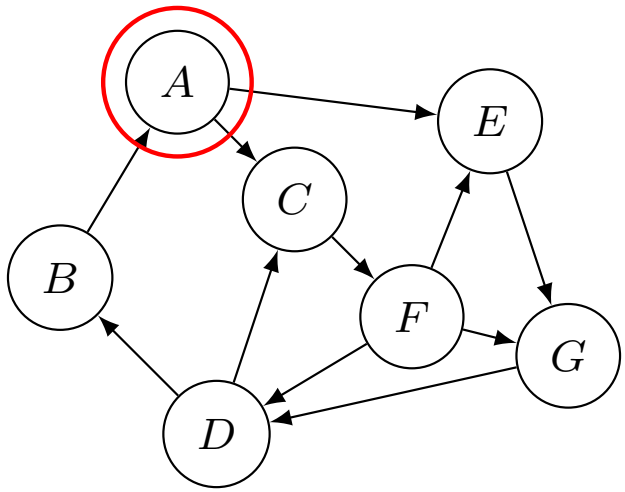


found:

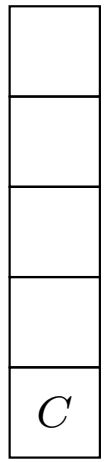
A	E	G	D	C	F	B

Grafsökning

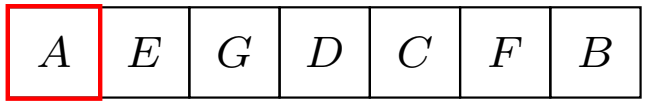
djupet-först-sökning (DFS)



stack

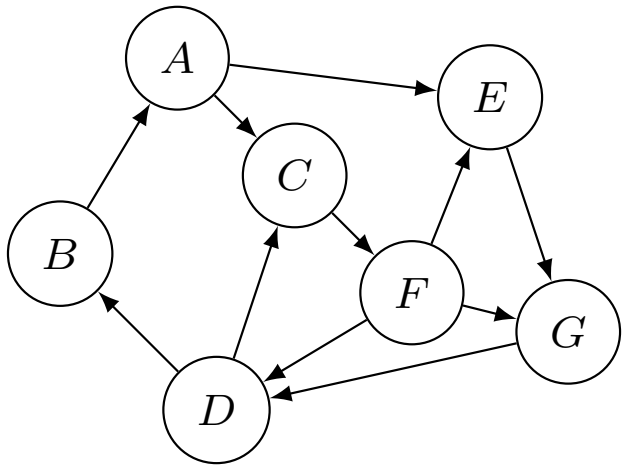


found:

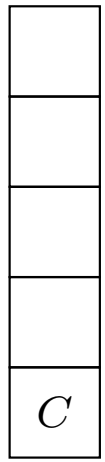


Grafsökning

djupet-först-sökning (DFS)



stack

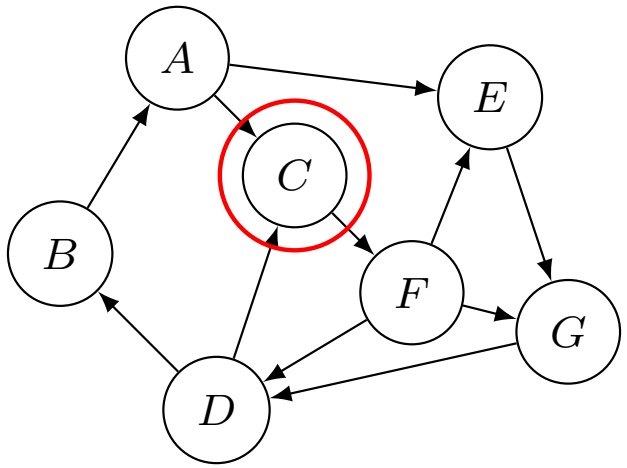


found:

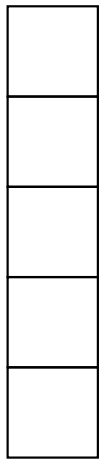
A	E	G	D	C	F	B

Grafsökning

djupet-först-sökning (DFS)



stack

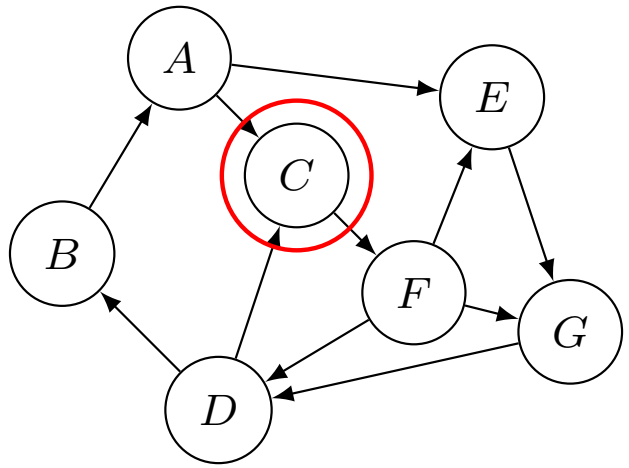


found:

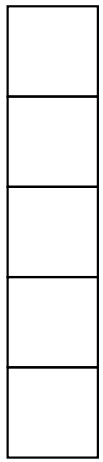
A	E	G	D	C	F	B

Grafsökning

djupet-först-sökning (DFS)



stack

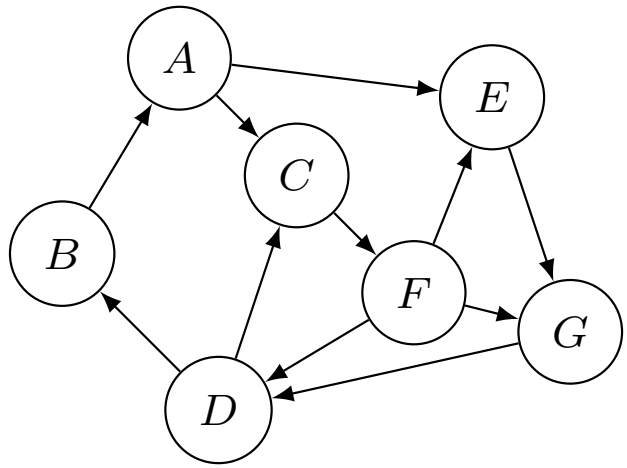


found:

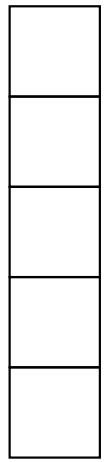
A	E	G	D	C	F	B

Grafsökning

djupet-först-sökning (DFS)



stack



found:

A	E	G	D	C	F	B

Grafsökning

Kan vi göra i någon annan ordning?

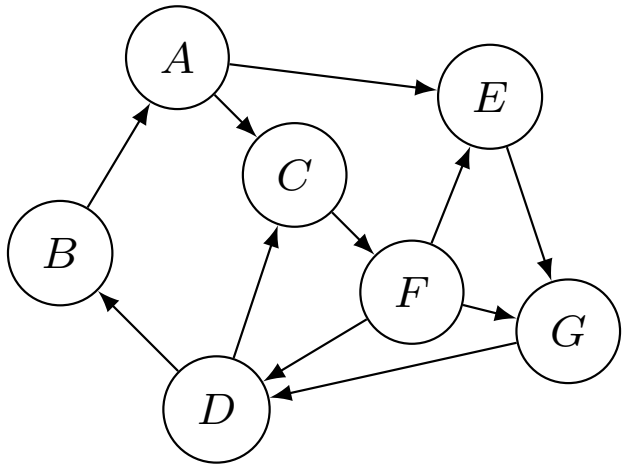
Grafsökning

Byt ut stacken mot en kö!

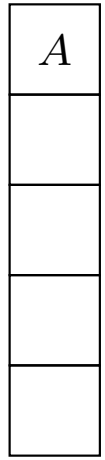
```
1  set<Node*> DFS(Node* start)
2  {
3    set<Node*> found;
4    queue<Node*> queue;
5    queue.push(start);
6    while (!queue.empty())
7    {
8      Node* node { queue.front() };
9      queue.pop();
10     if (node == nullptr) continue;
11     if (found.count(node) > 0) continue;
12     found.insert(node);
13     for (Node* next : neighbours)
14       queue.push(next);
15   }
16   return found;
17 }
```

Grafsökning

bredden-först-sökning (BFS)



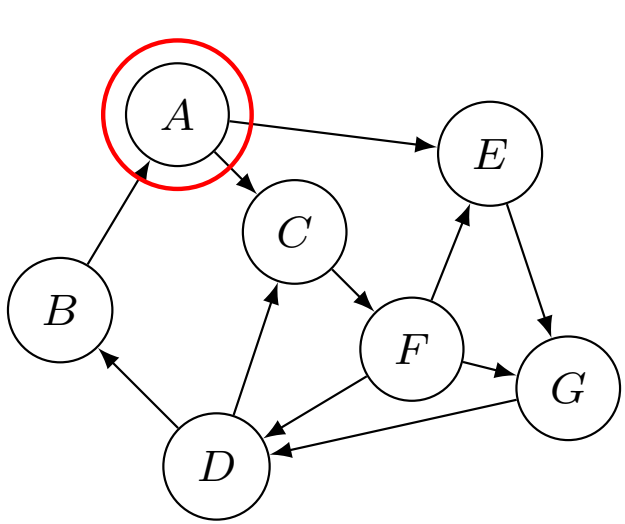
queue



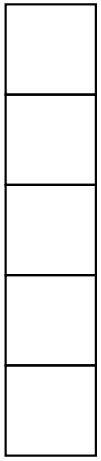
found:

Grafsökning

bredden-först-sökning (BFS)



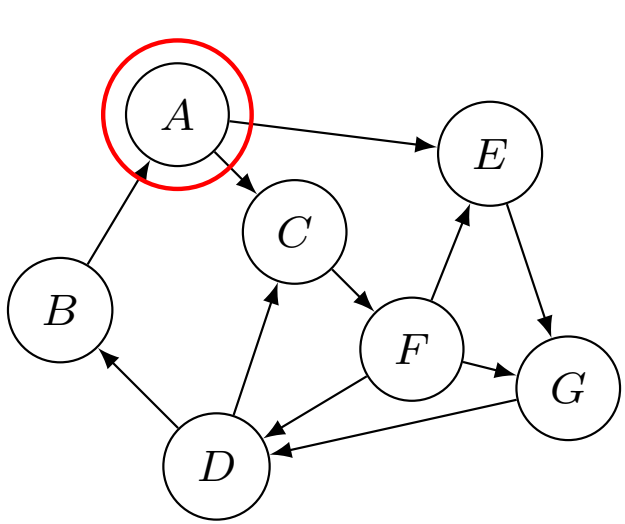
queue



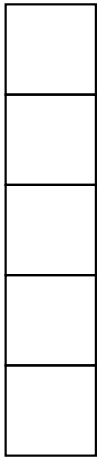
found:

Grafsökning

bredden-först-sökning (BFS)



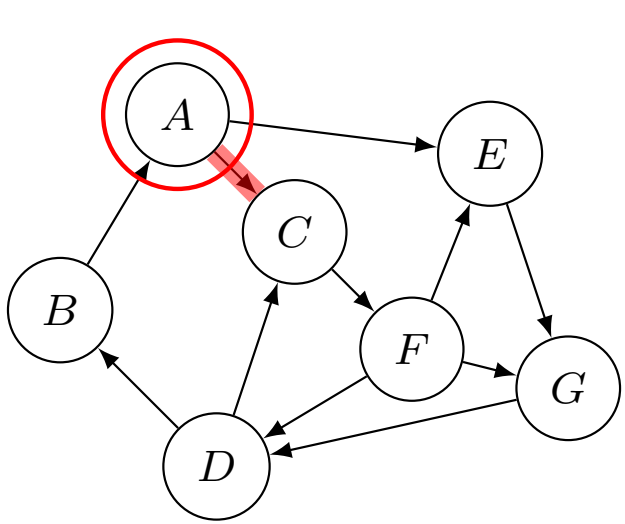
queue



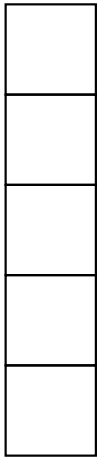
found: A

Grafsökning

bredden-först-sökning (BFS)



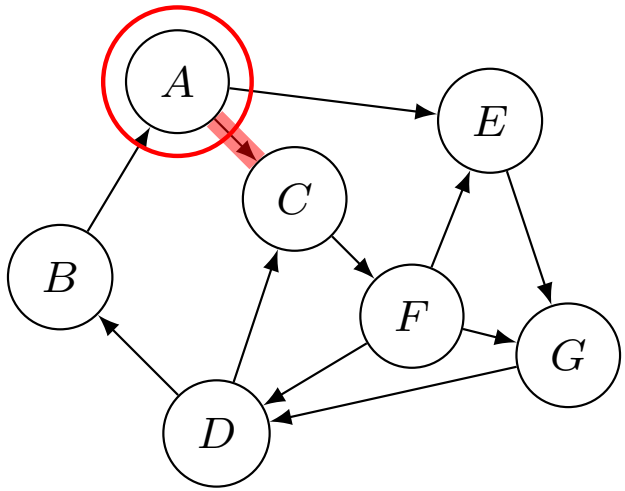
queue



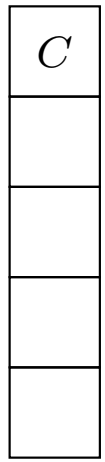
found:

Grafsökning

bredden-först-sökning (BFS)



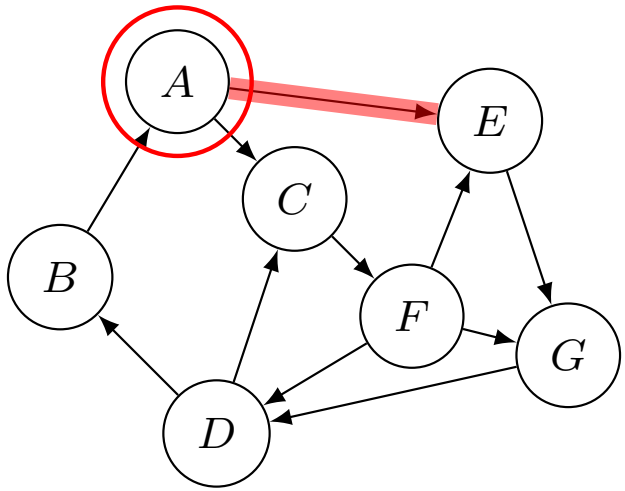
queue



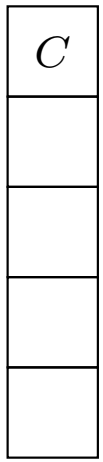
found:

Grafsökning

bredden-först-sökning (BFS)



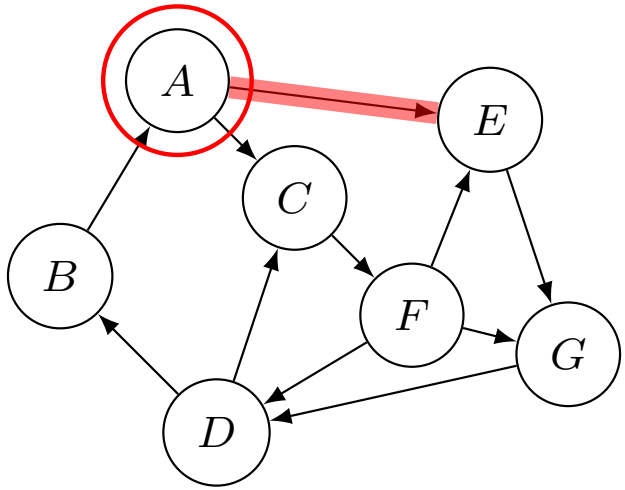
queue



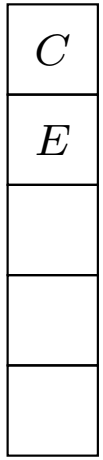
found:

Grafsökning

bredden-först-sökning (BFS)



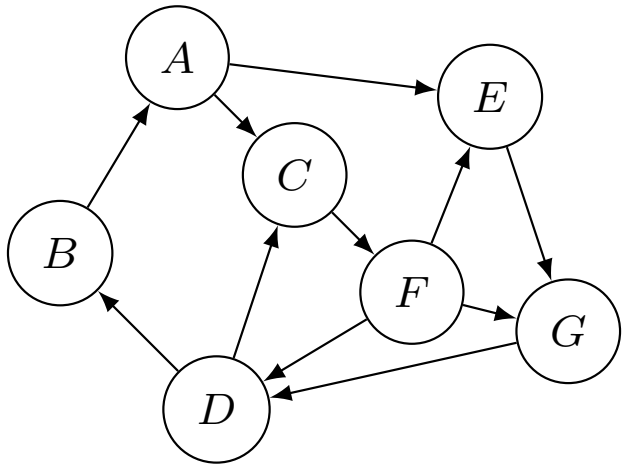
queue



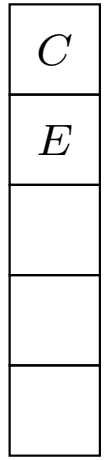
found:

Grafsökning

bredden-först-sökning (BFS)



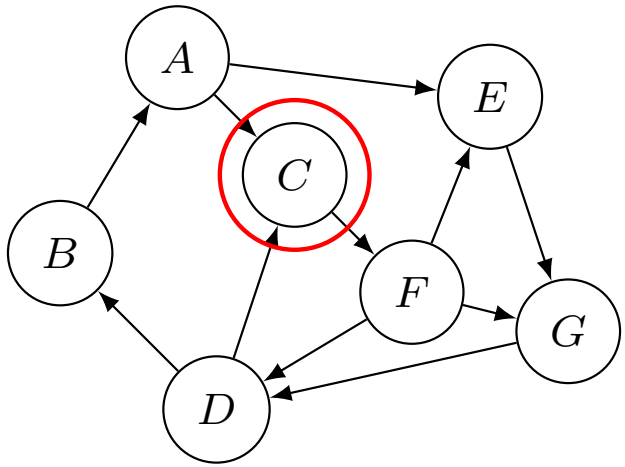
queue



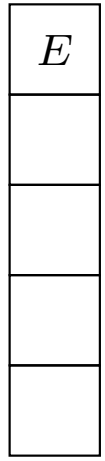
found:

Grafsökning

bredden-först-sökning (BFS)



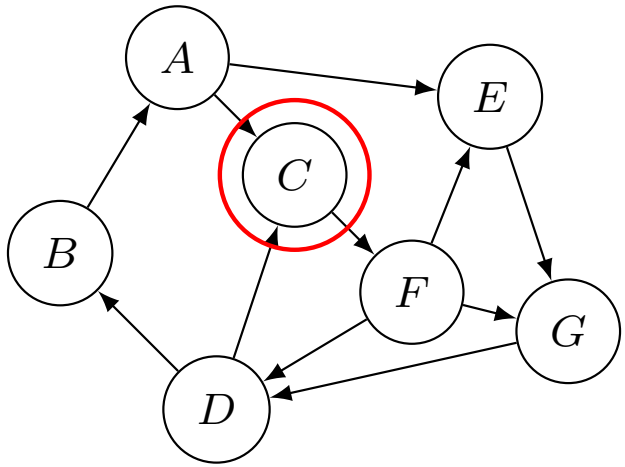
queue



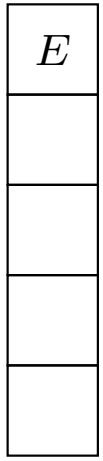
found:

Grafsökning

bredden-först-sökning (BFS)



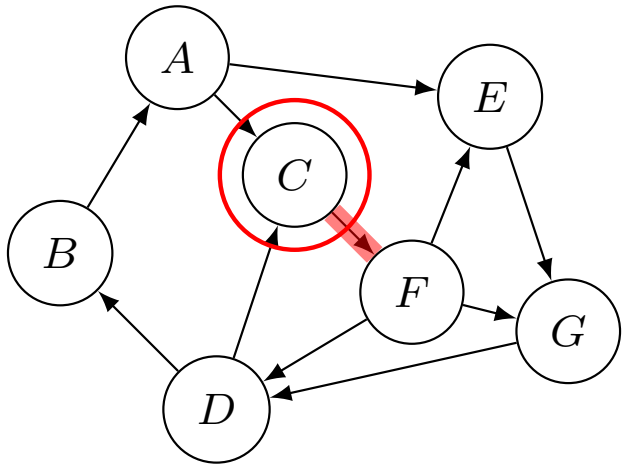
queue



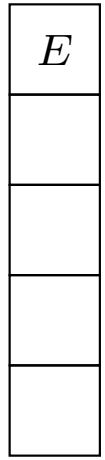
found:

Grafsökning

bredden-först-sökning (BFS)



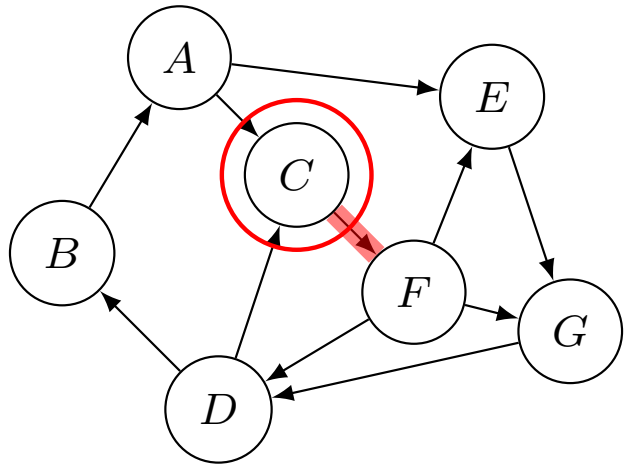
queue



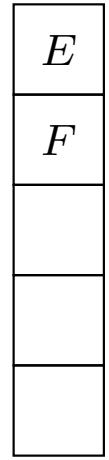
found:

Grafsökning

bredden-först-sökning (BFS)



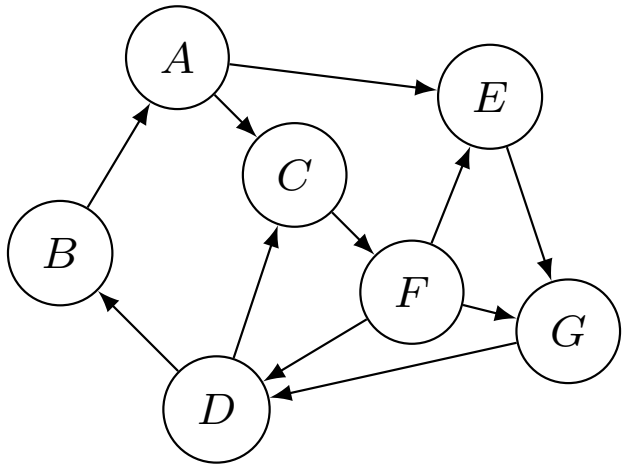
queue



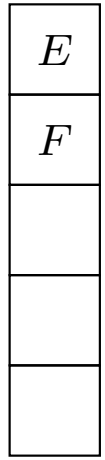
found:

Grafsökning

bredden-först-sökning (BFS)



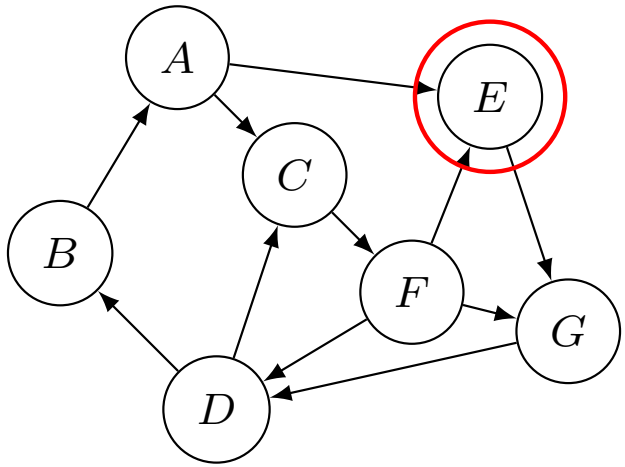
queue



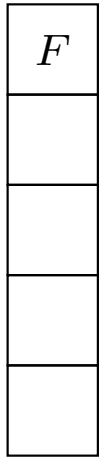
found:

Grafsökning

bredden-först-sökning (BFS)



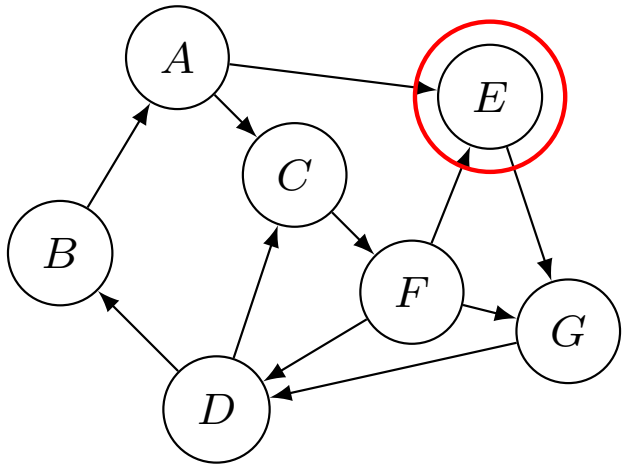
queue



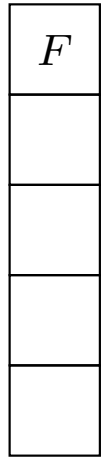
found:

Grafsökning

bredden-först-sökning (BFS)



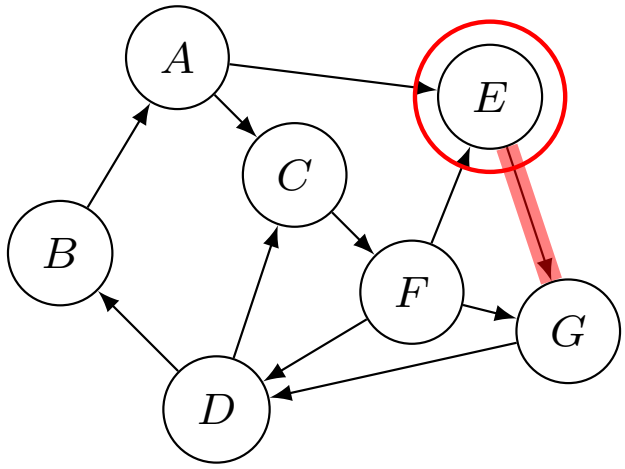
queue



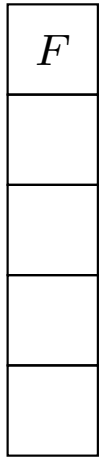
found:

Grafsökning

bredden-först-sökning (BFS)



queue

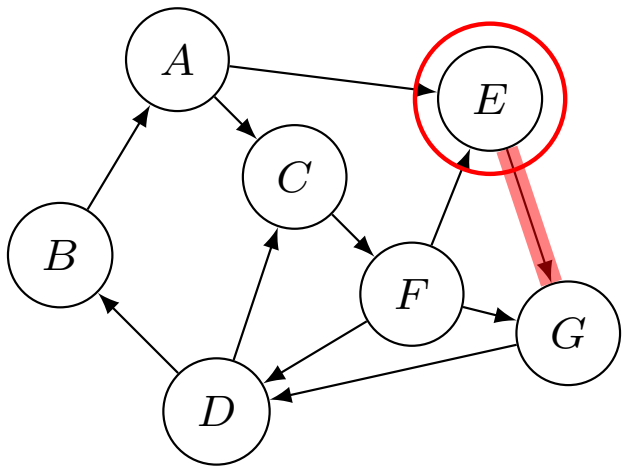


found:

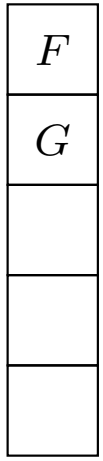
A	C	E				

Grafsökning

bredden-först-sökning (BFS)



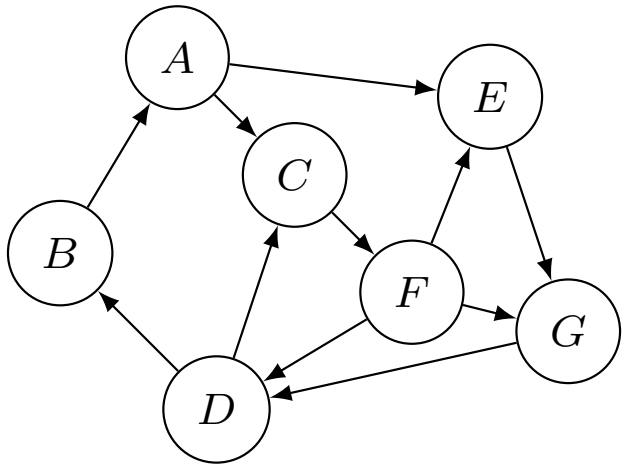
queue



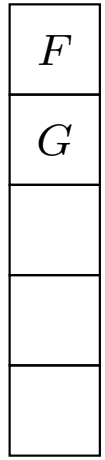
found:

Grafsökning

bredden-först-sökning (BFS)



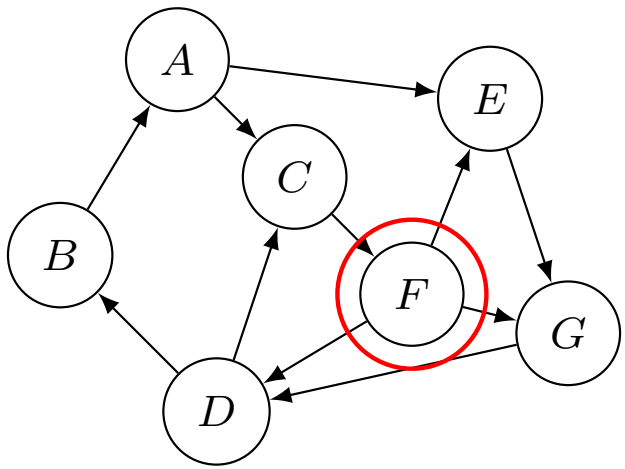
queue



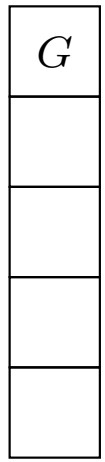
found:

Grafsökning

bredden-först-sökning (BFS)



queue

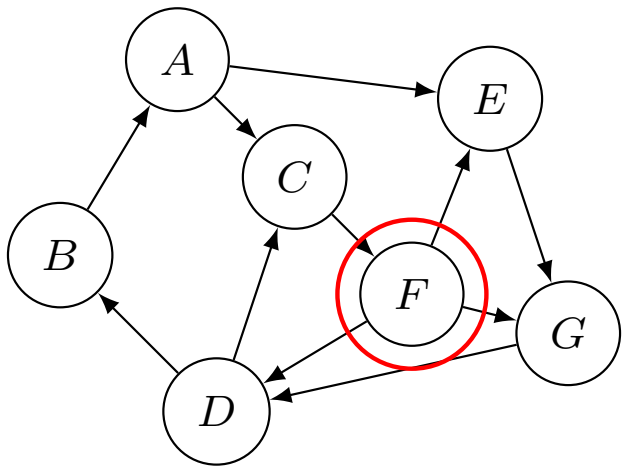


found:

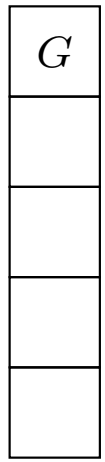
A	C	E				

Grafsökning

bredden-först-sökning (BFS)



queue

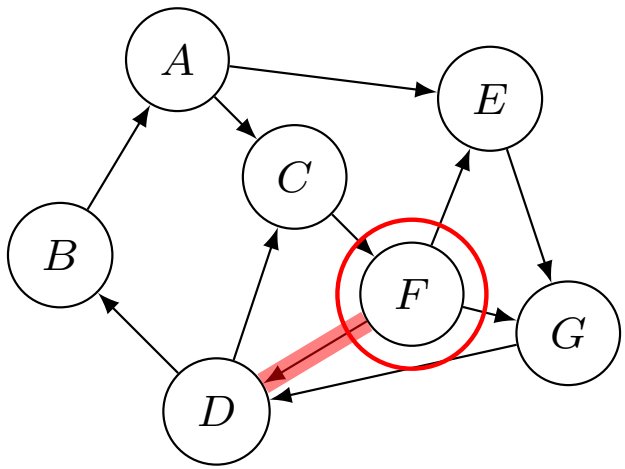


found:

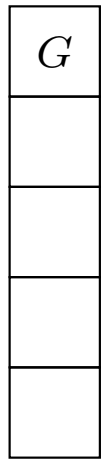
A	C	E	F			

Grafsökning

bredden-först-sökning (BFS)



queue

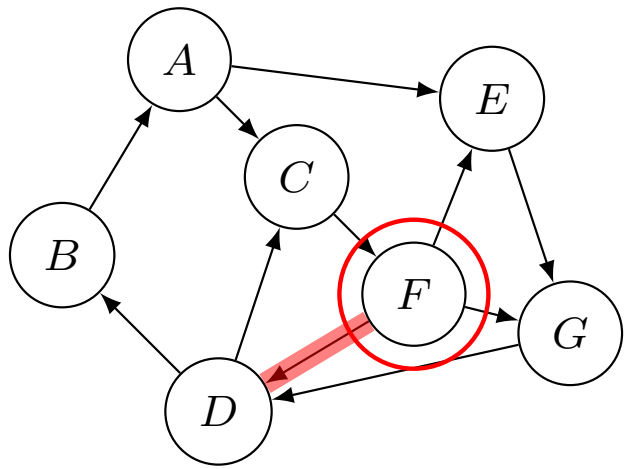


found:

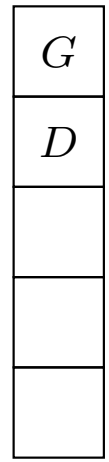
A	C	E	F			

Grafsökning

bredden-först-sökning (BFS)



queue

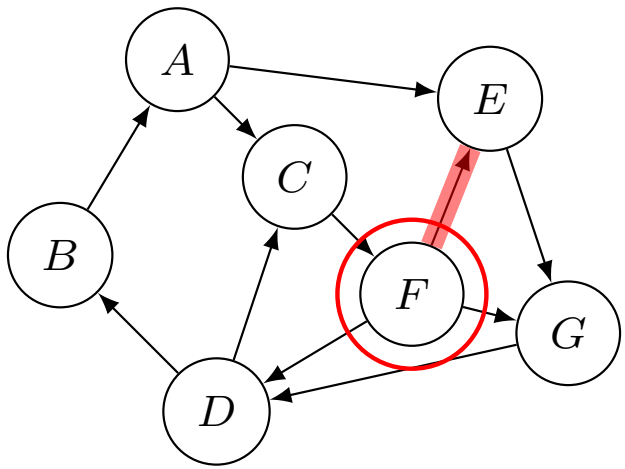


found:

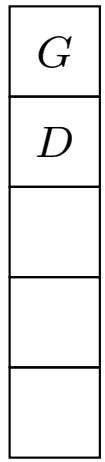
A	C	E	F			

Grafsökning

bredden-först-sökning (BFS)



queue

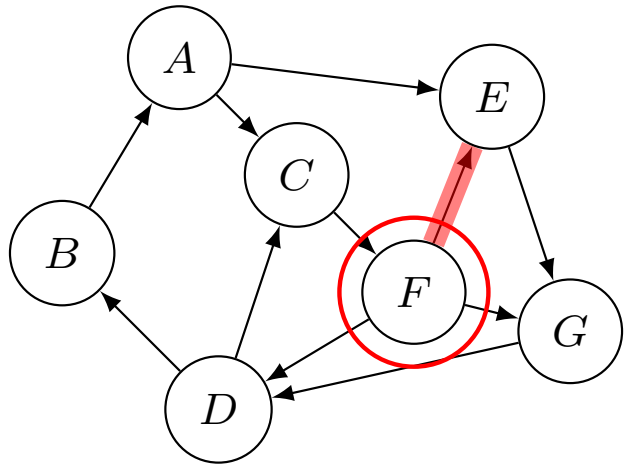


found:

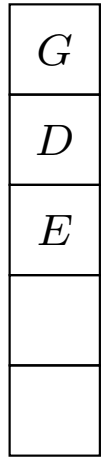
A	C	E	F			

Grafsökning

bredden-först-sökning (BFS)



queue

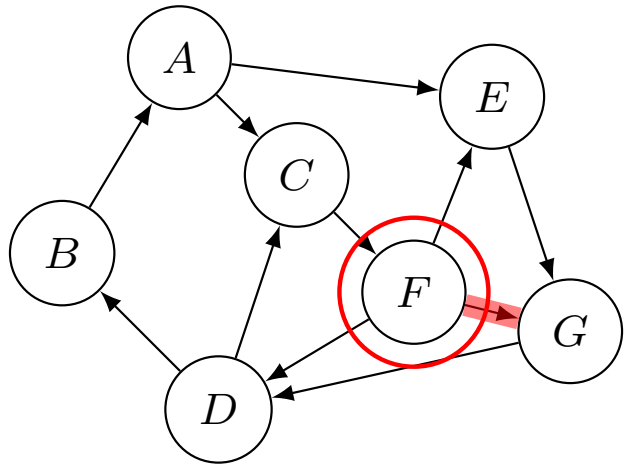


found:

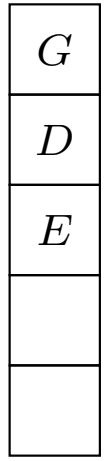
A	C	E	F			

Grafsökning

bredden-först-sökning (BFS)



queue

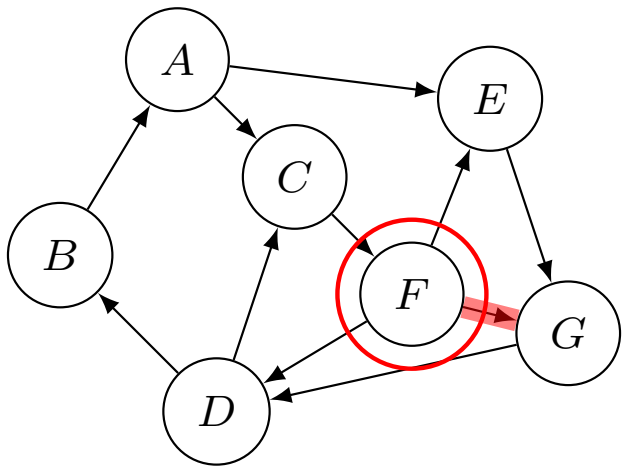


found:

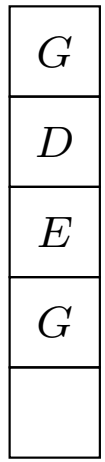
A	C	E	F			

Grafsökning

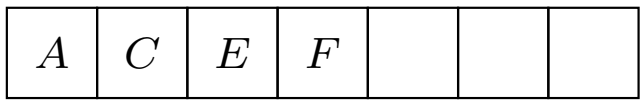
bredden-först-sökning (BFS)



queue

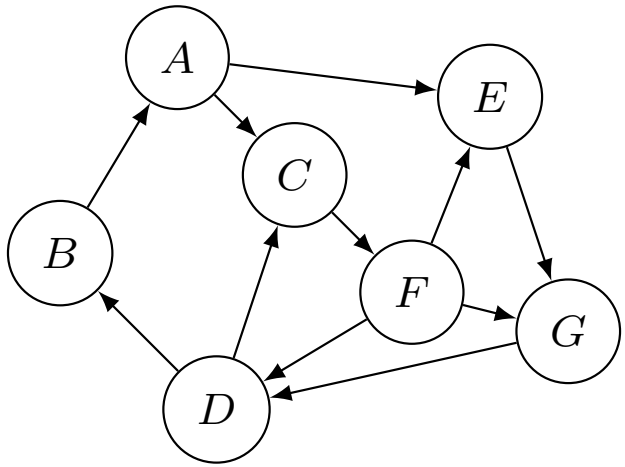


found:

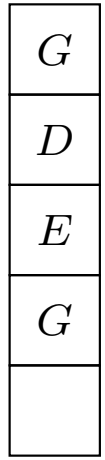


Grafsökning

bredden-först-sökning (BFS)



queue

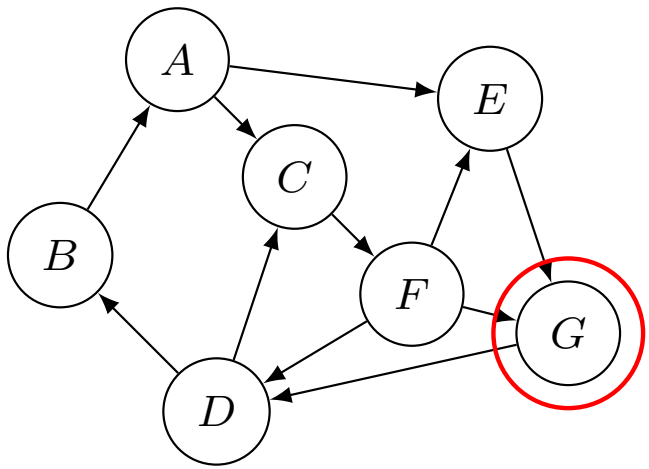


found:

A	C	E	F			

Grafsökning

bredden-först-sökning (BFS)



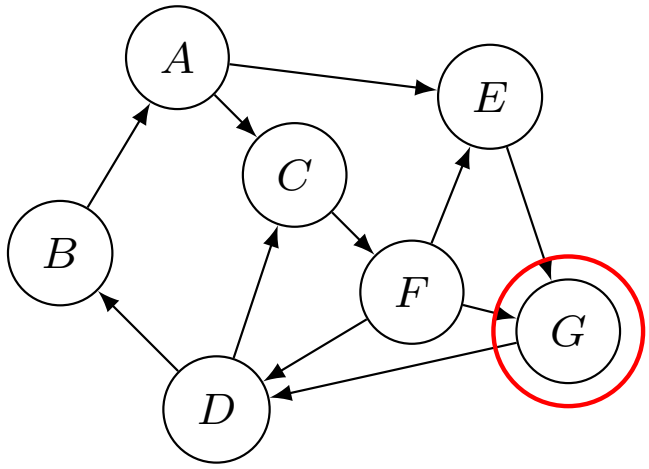
queue

found:

A	C	E	F			

Grafsökning

bredden-först-sökning (BFS)



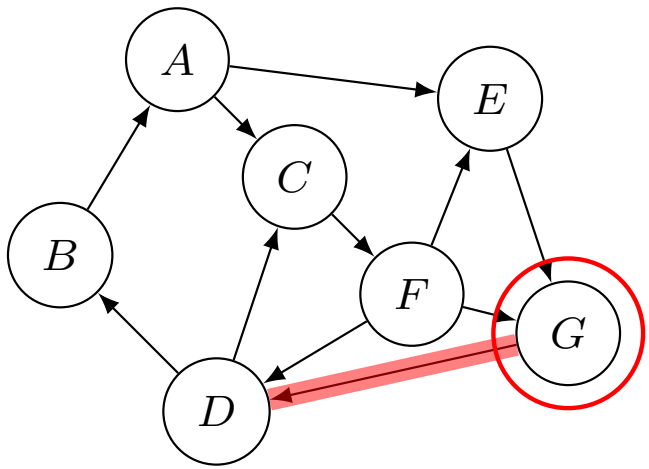
queue

found:

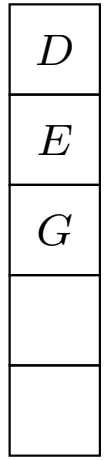
A	C	E	F	G		

Grafsökning

bredden-först-sökning (BFS)



queue

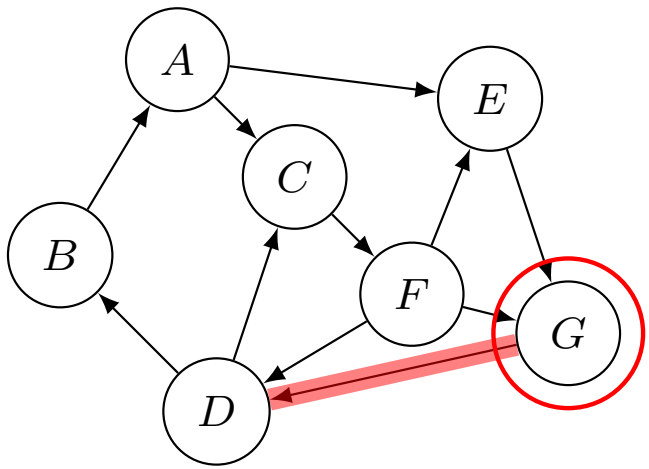


found:

A	C	E	F	G		

Grafsökning

bredden-först-sökning (BFS)



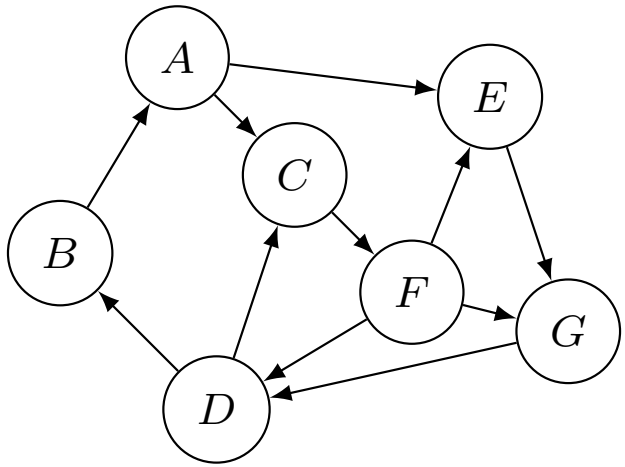
queue

found:

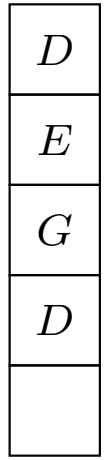
A	C	E	F	G		

Grafsökning

bredden-först-sökning (BFS)



queue

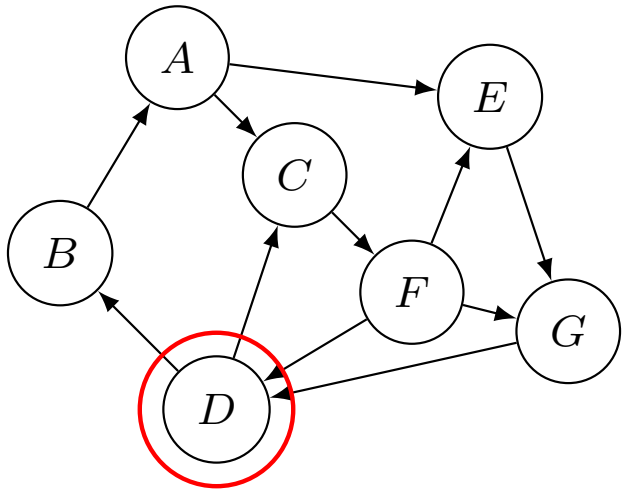


found:

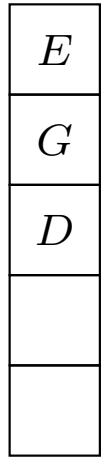
A	C	E	F	G		

Grafsökning

bredden-först-sökning (BFS)



queue

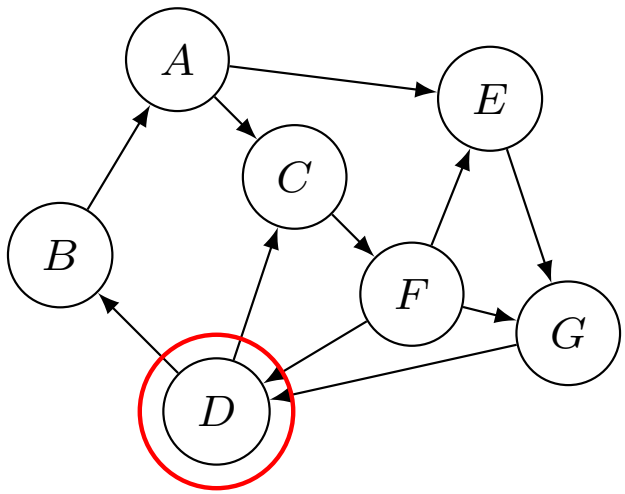


found:

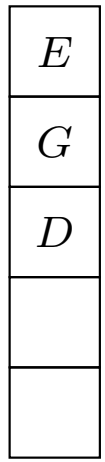
A	C	E	F	G		

Grafsökning

bredden-först-sökning (BFS)



queue

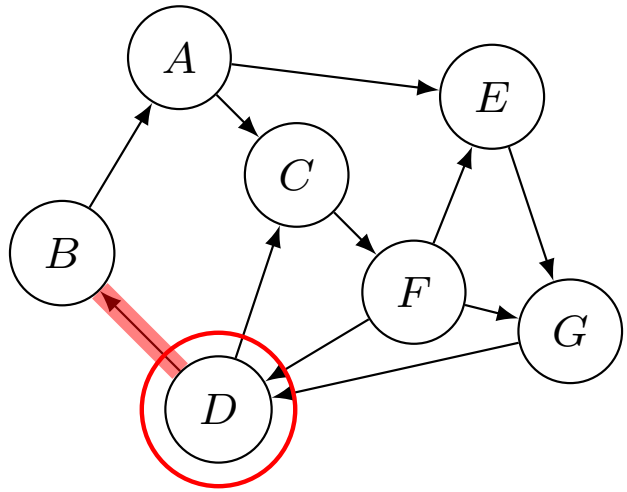


found:

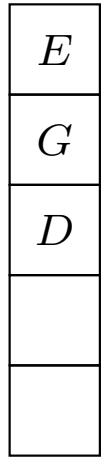
A	C	E	F	G	D	

Grafsökning

bredden-först-sökning (BFS)



queue

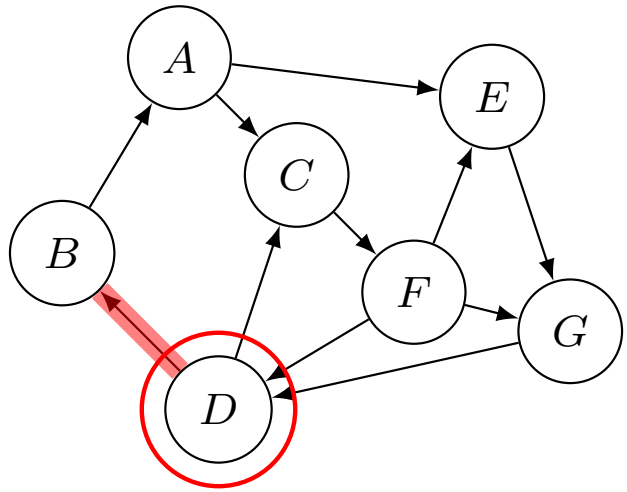


found:

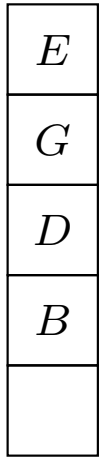
A	C	E	F	G	D	

Grafsökning

bredden-först-sökning (BFS)



queue

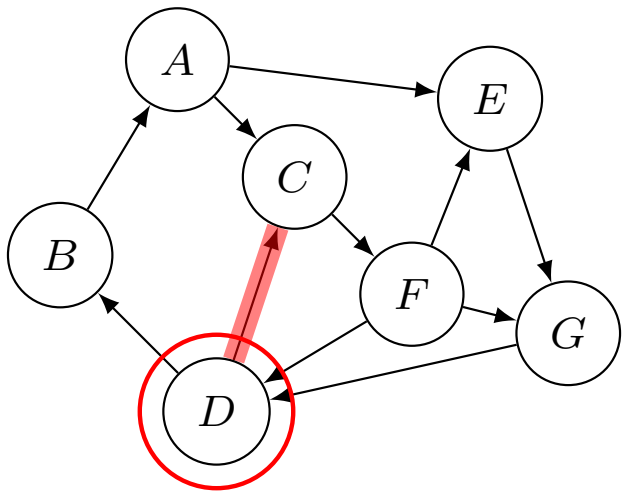


found:

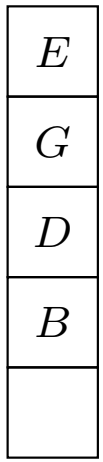
A	C	E	F	G	D	

Grafsökning

bredden-först-sökning (BFS)



queue

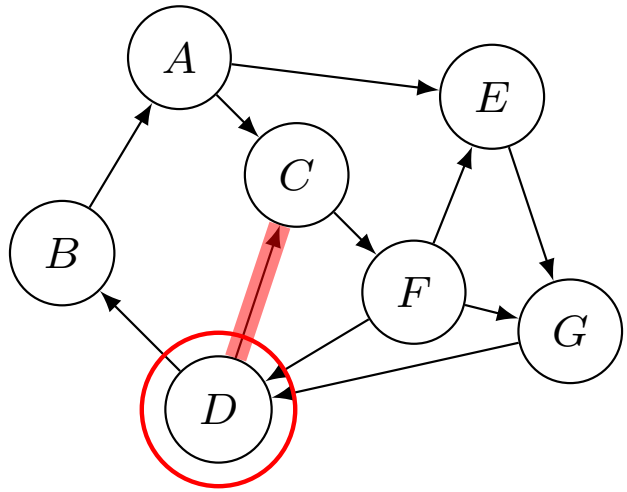


found:

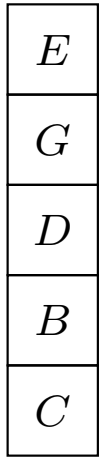
A	C	E	F	G	D	

Grafsökning

bredden-först-sökning (BFS)



queue

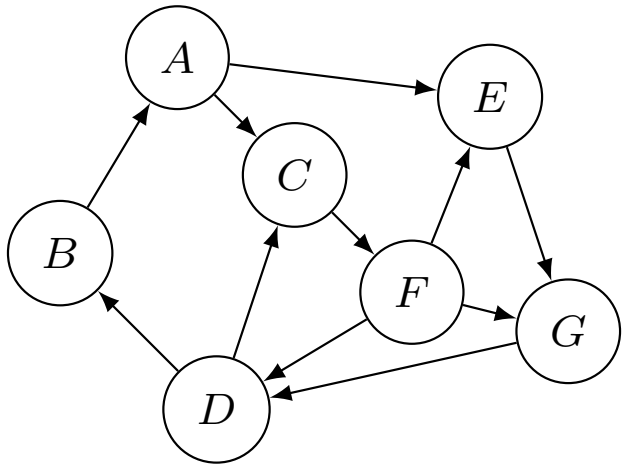


found:

A	C	E	F	G	D	

Grafsökning

bredden-först-sökning (BFS)



queue

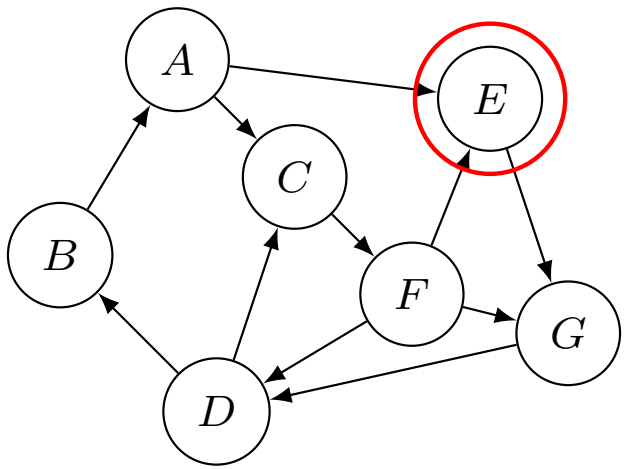
E
G
D
B
C

found:

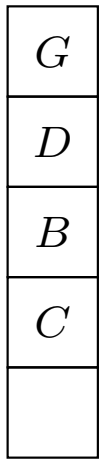
A	C	E	F	G	D	

Grafsökning

bredden-först-sökning (BFS)



queue

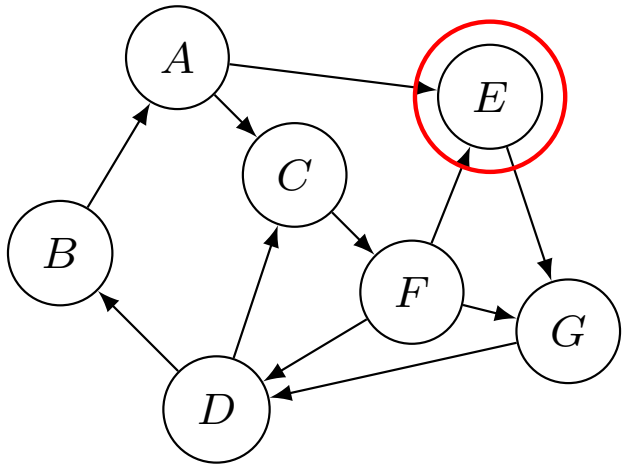


found:

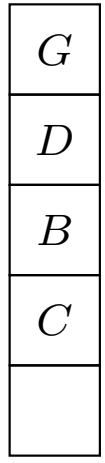
A	C	E	F	G	D	

Grafsökning

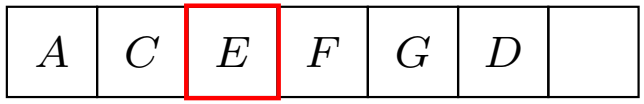
bredden-först-sökning (BFS)



queue

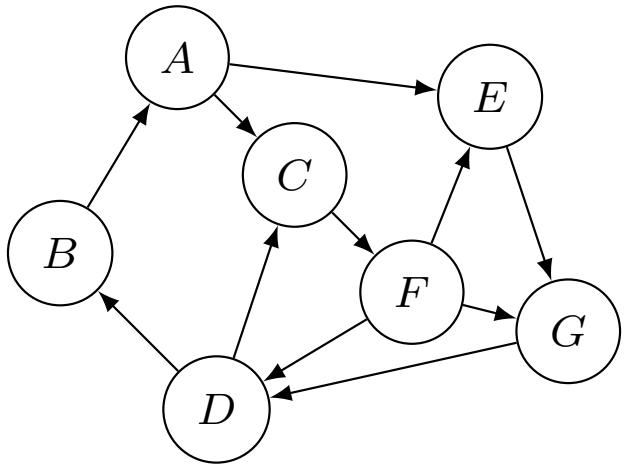


found:

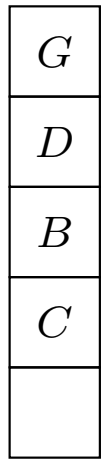


Grafsökning

bredden-först-sökning (BFS)



queue

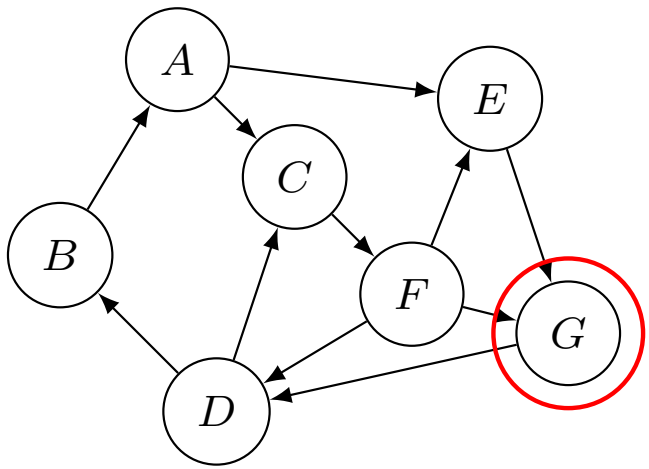


found:

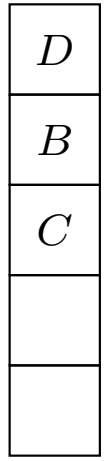
A	C	E	F	G	D	

Grafsökning

bredden-först-sökning (BFS)



queue

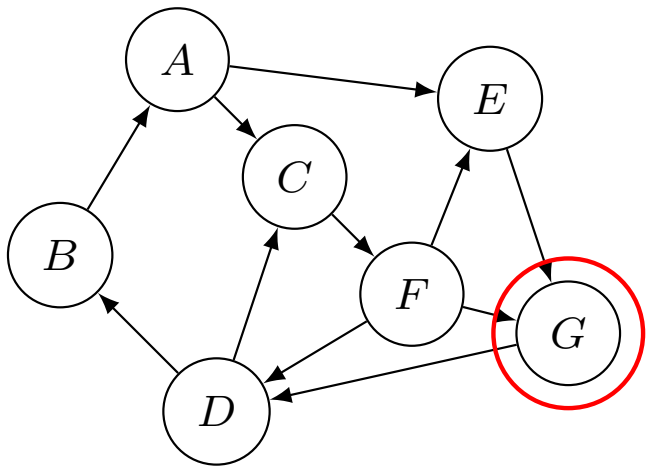


found:

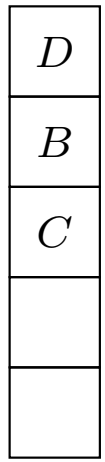
A	C	E	F	G	D	

Grafsökning

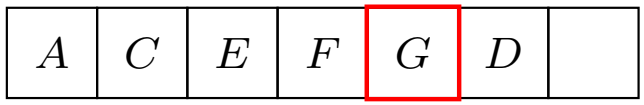
bredden-först-sökning (BFS)



queue

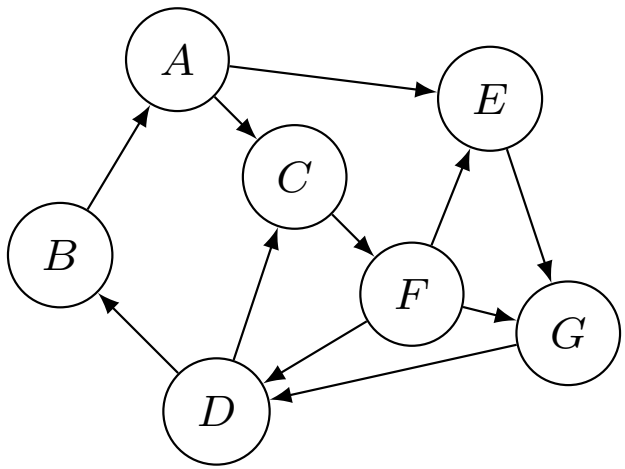


found:

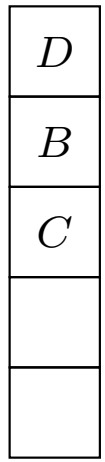


Grafsökning

bredden-först-sökning (BFS)



queue

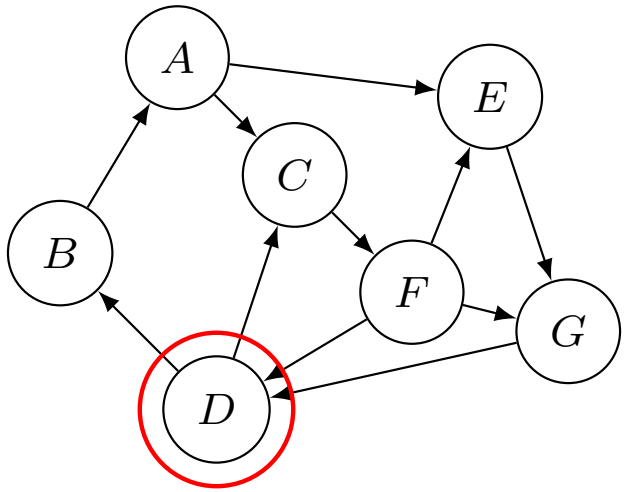


found:

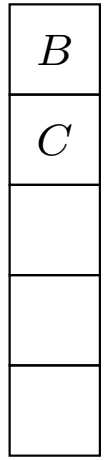
A	C	E	F	G	D	

Grafsökning

bredden-först-sökning (BFS)



queue

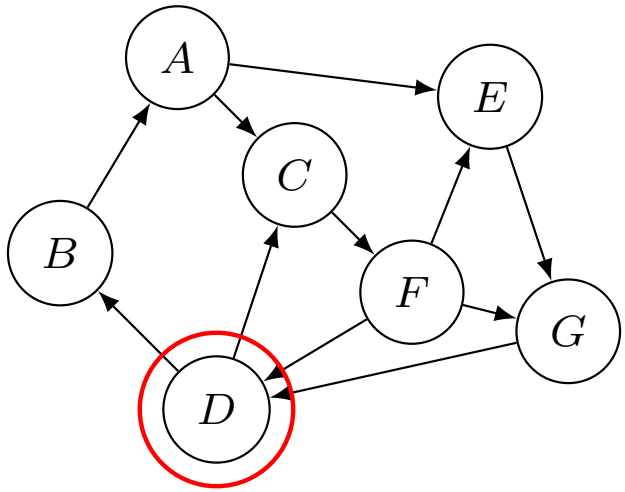


found:

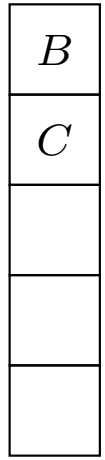
A	C	E	F	G	D	

Grafsökning

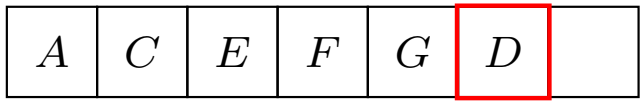
bredden-först-sökning (BFS)



queue

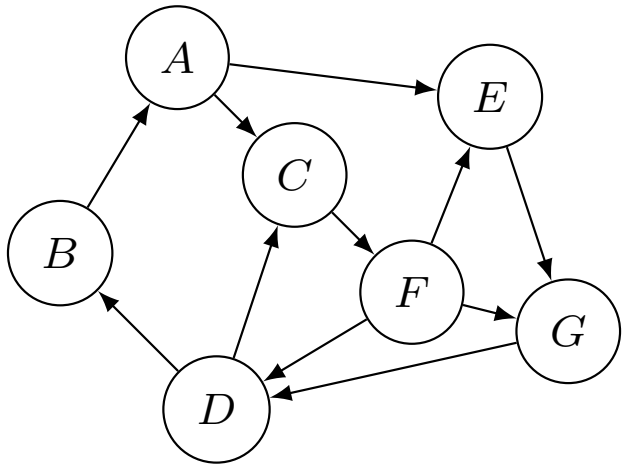


found:

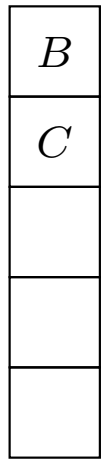


Grafsökning

bredden-först-sökning (BFS)



queue

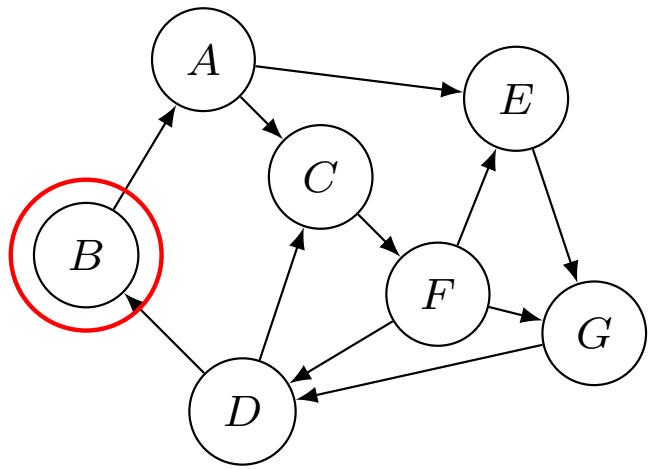


found:

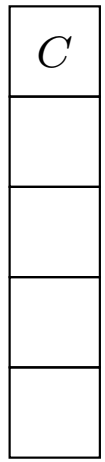
A	C	E	F	G	D	

Grafsökning

bredden-först-sökning (BFS)



queue

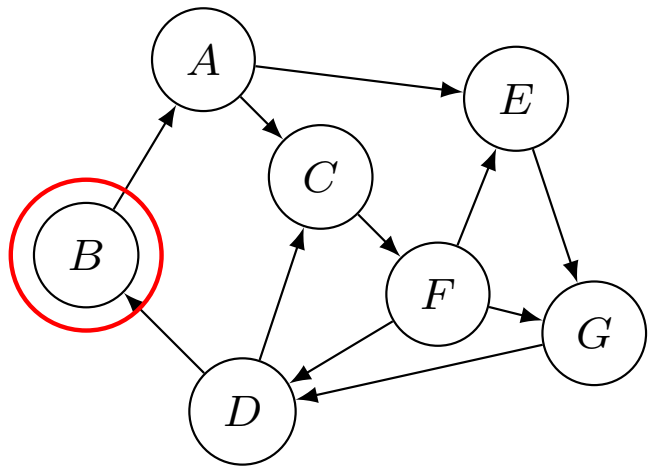


found:

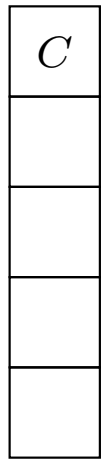
A	C	E	F	G	D	

Grafsökning

bredden-först-sökning (BFS)



queue

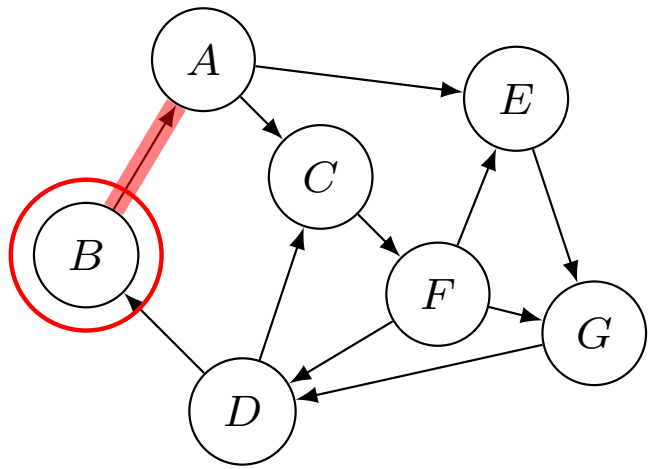


found:

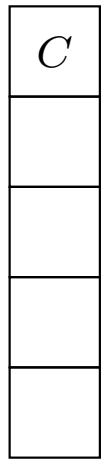
A	C	E	F	G	D	B

Grafsökning

bredden-först-sökning (BFS)



queue

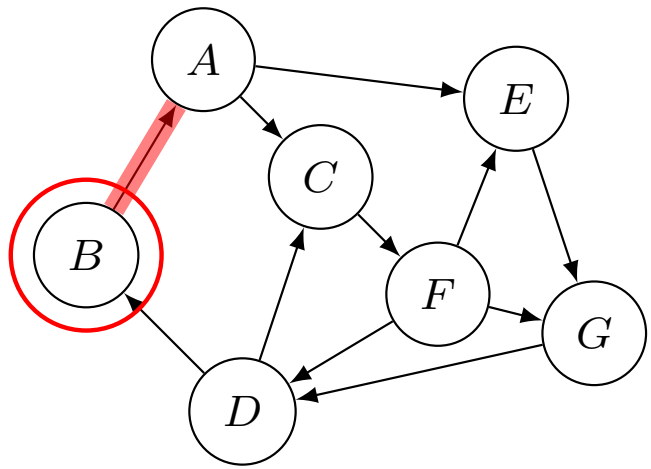


found:

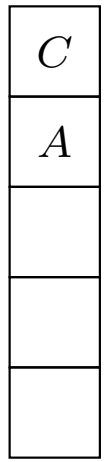
A	C	E	F	G	D	B

Grafsökning

bredden-först-sökning (BFS)



queue

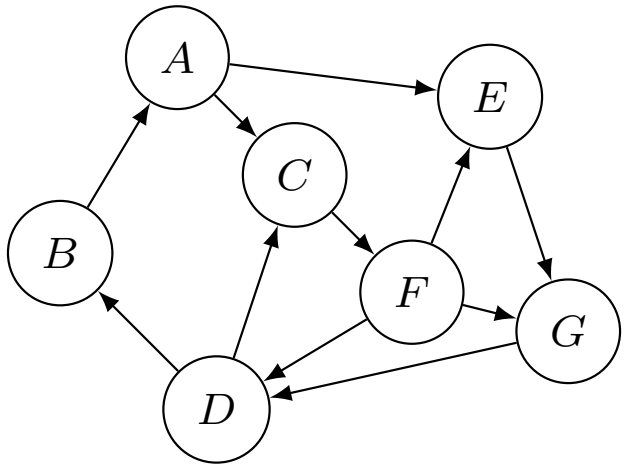


found:

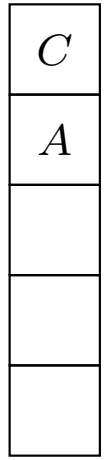
A	C	E	F	G	D	B

Grafsökning

bredden-först-sökning (BFS)



queue

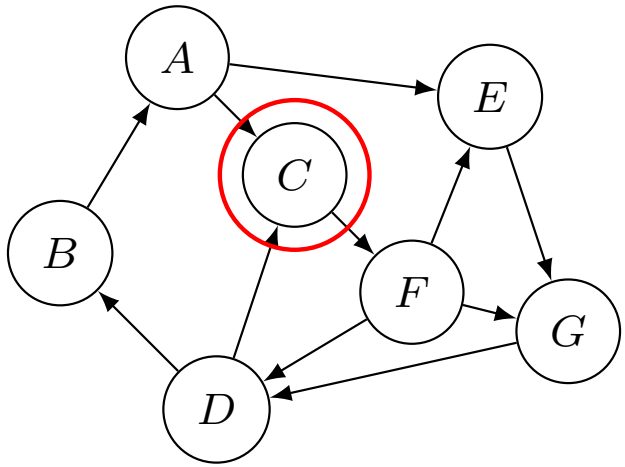


found:

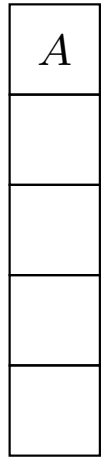
A	C	E	F	G	D	B

Grafsökning

bredden-först-sökning (BFS)



queue

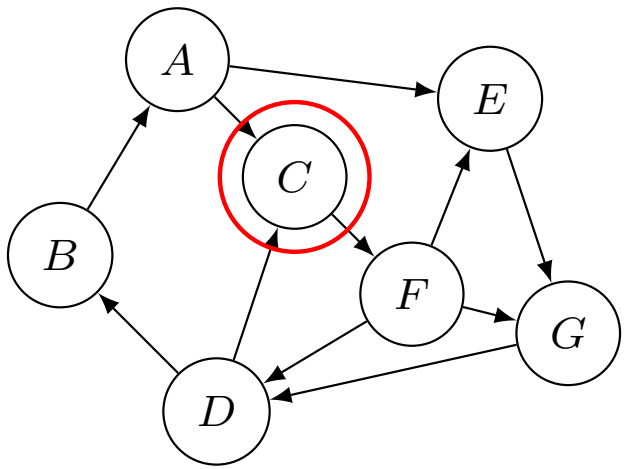


found:

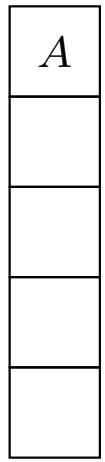
A	C	E	F	G	D	B

Grafsökning

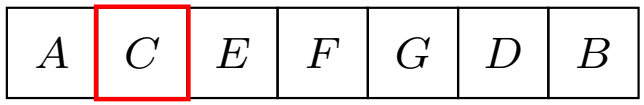
bredden-först-sökning (BFS)



queue

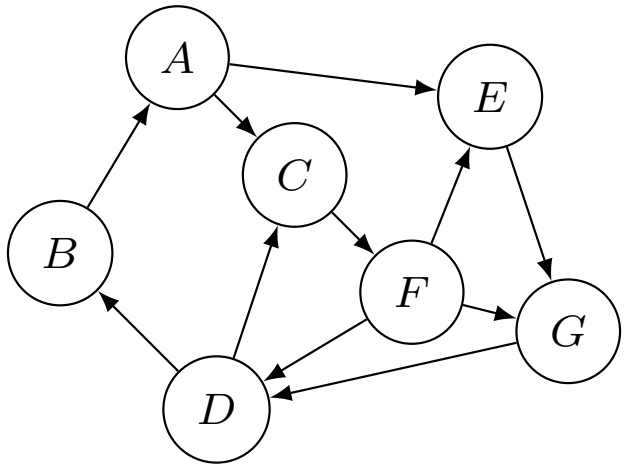


found:

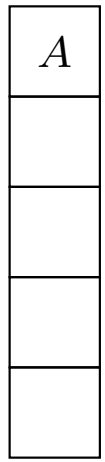


Grafsökning

bredden-först-sökning (BFS)



queue

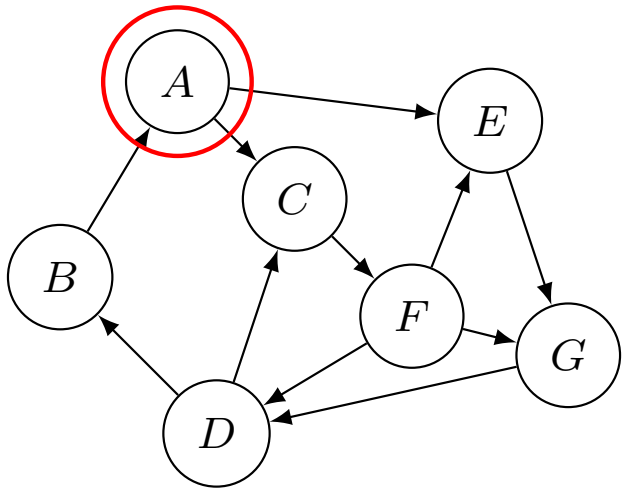


found:

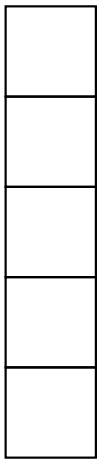
A	C	E	F	G	D	B

Grafsökning

bredden-först-sökning (BFS)



queue

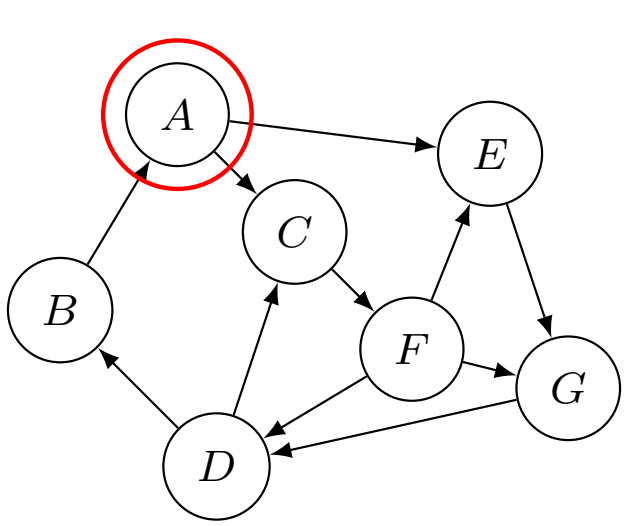


found:

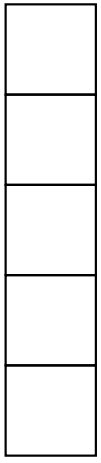
A	C	E	F	G	D	B

Grafsökning

bredden-först-sökning (BFS)



queue

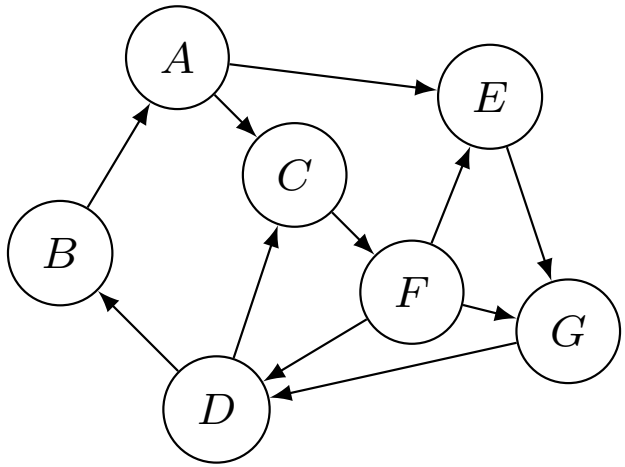


found:

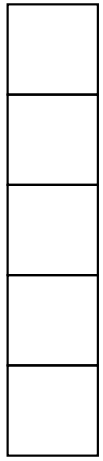
A	C	E	F	G	D	B

Grafsökning

bredden-först-sökning (BFS)



queue

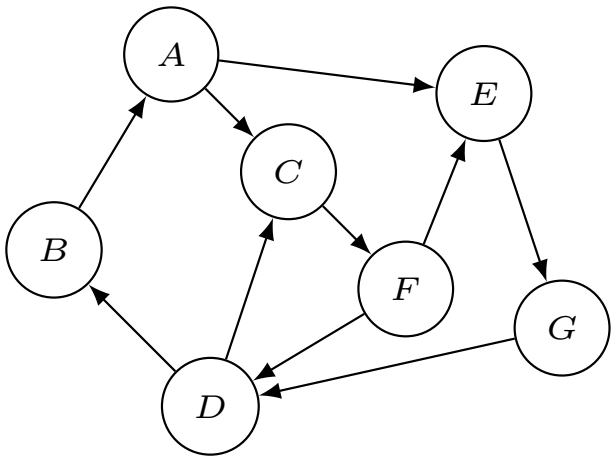


found:

A	C	E	F	G	D	B

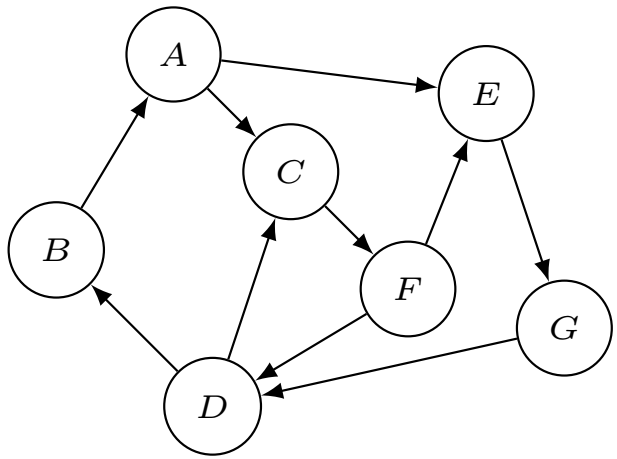
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

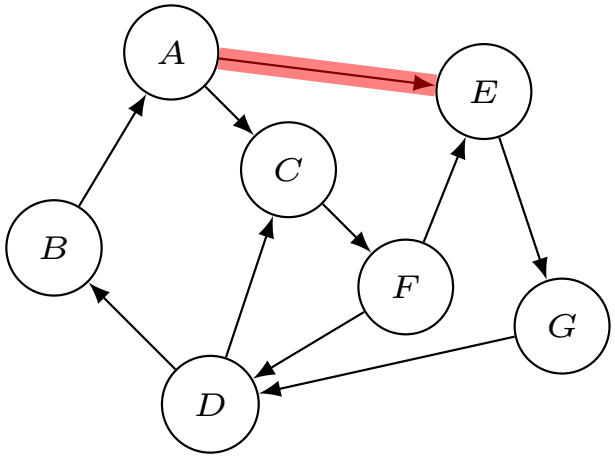


BFS ordning

A	C	E	F	G	D	B

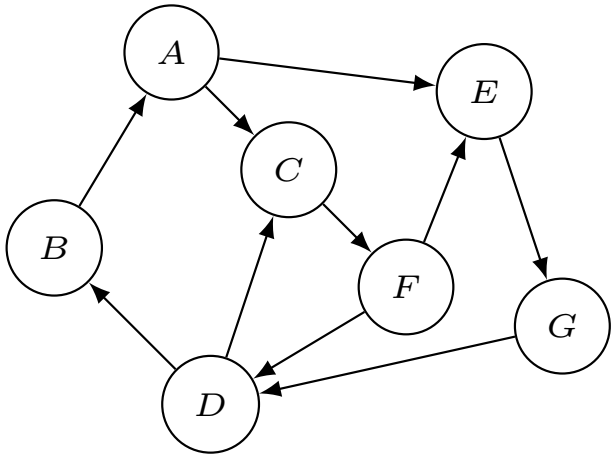
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

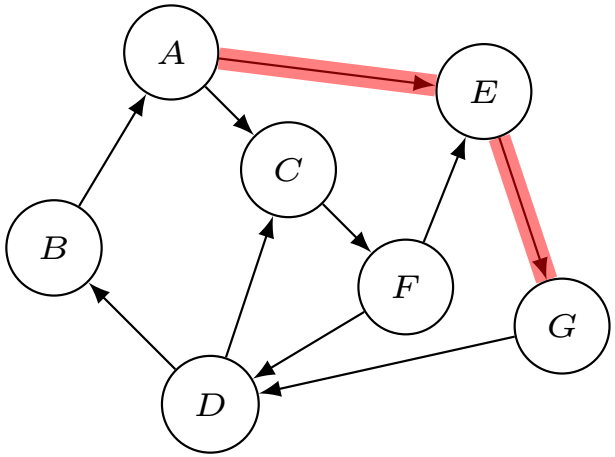


BFS ordning

A	C	E	F	G	D	B

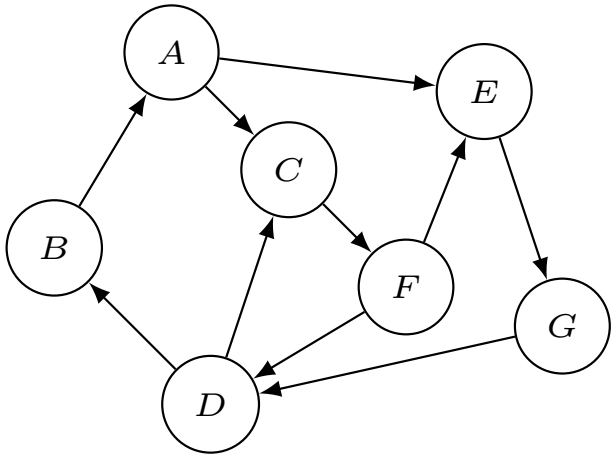
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

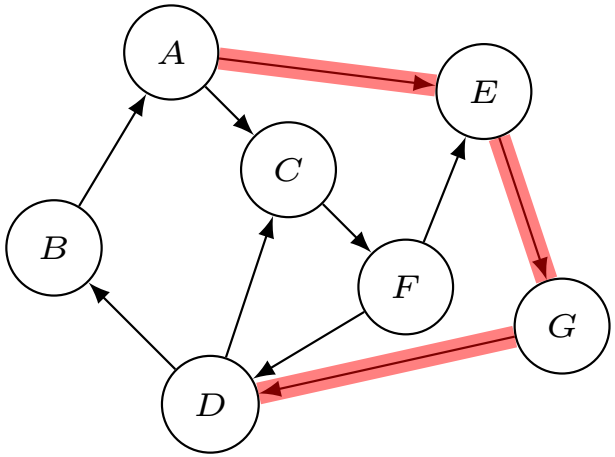


BFS ordning

A	C	E	F	G	D	B

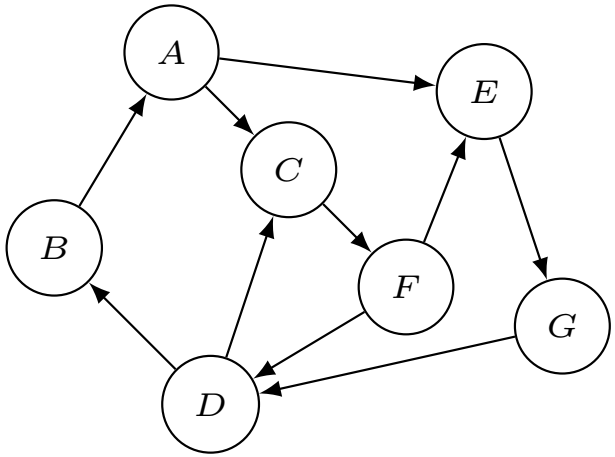
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

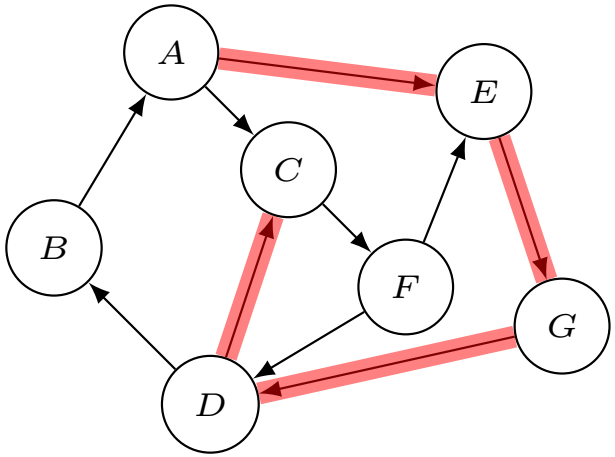


BFS ordning

A	C	E	F	G	D	B

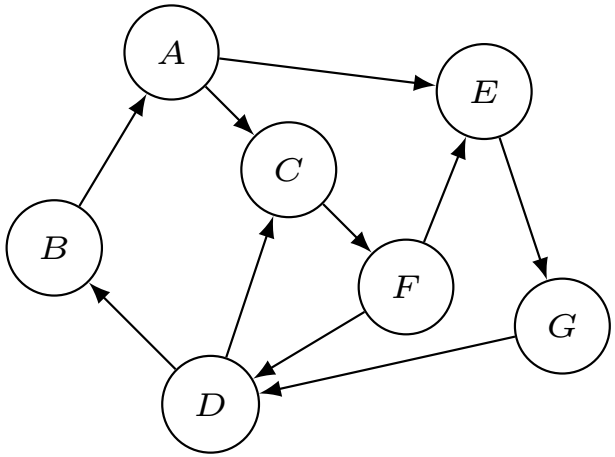
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

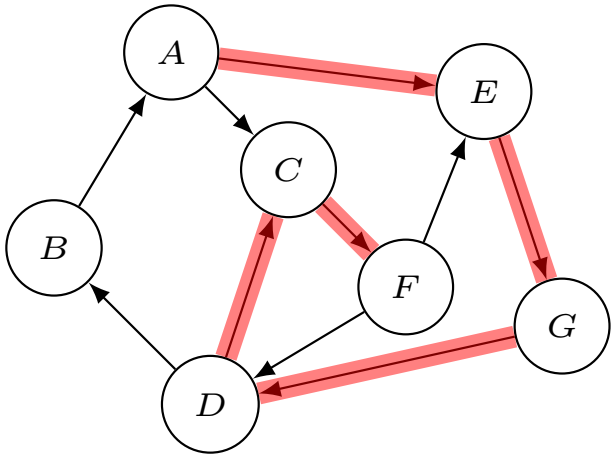


BFS ordning

A	C	E	F	G	D	B

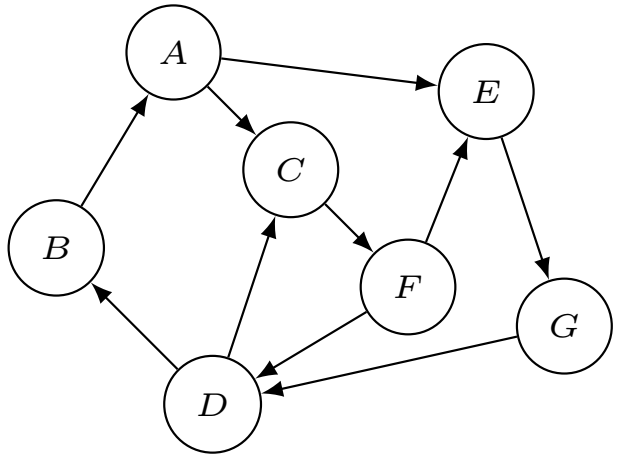
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

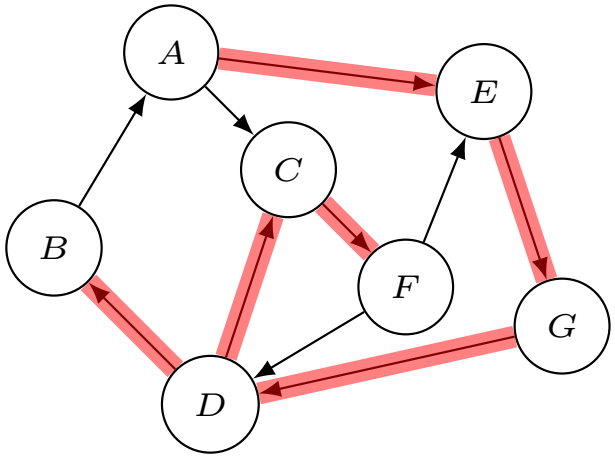


BFS ordning

A	C	E	F	G	D	B

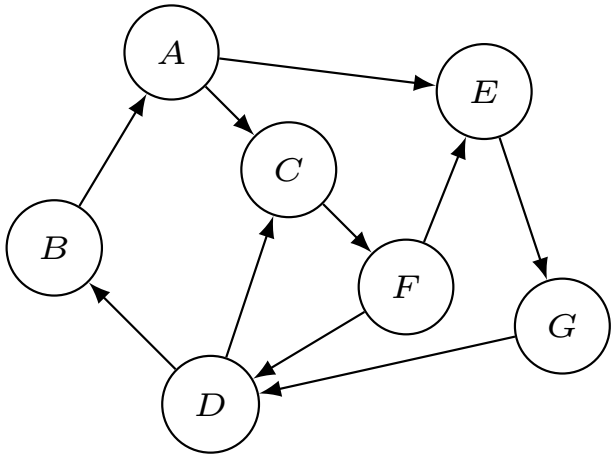
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

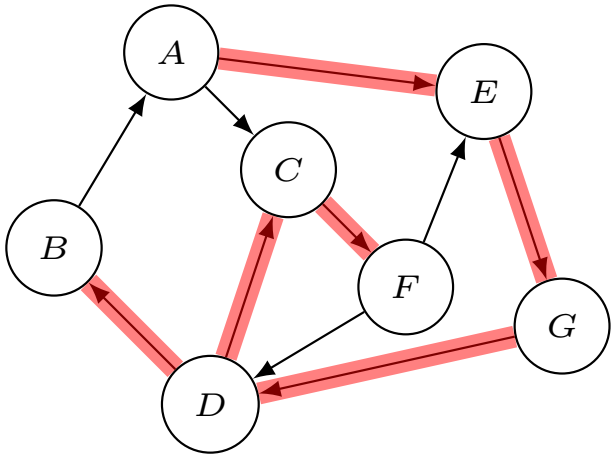


BFS ordning

A	C	E	F	G	D	B

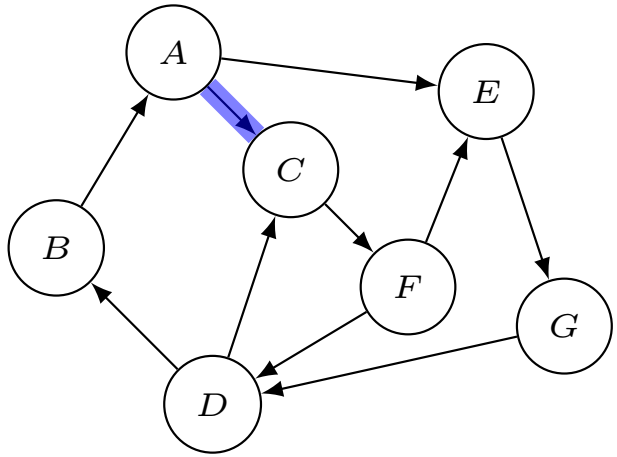
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

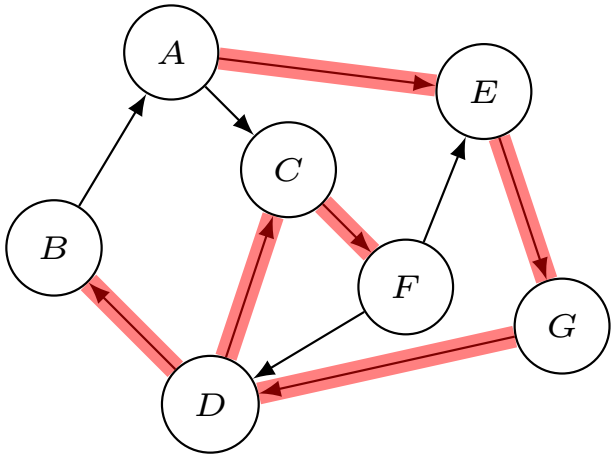


BFS ordning

A	C	E	F	G	D	B

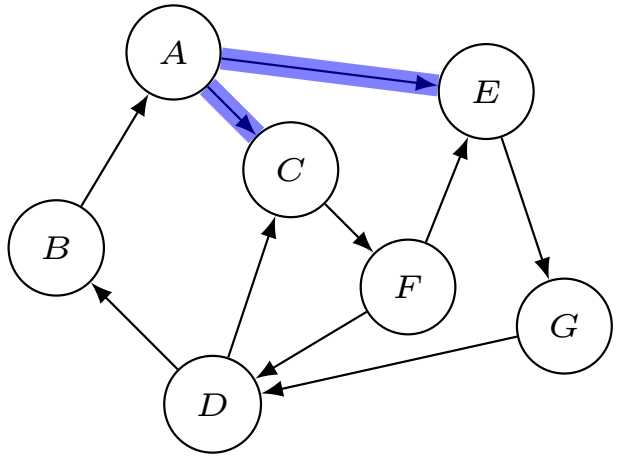
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

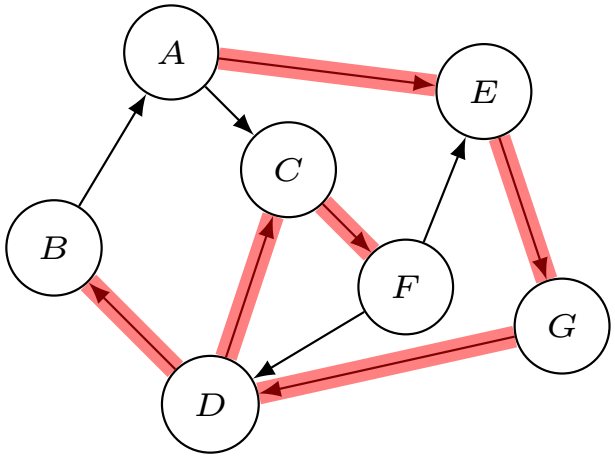


BFS ordning

A	C	E	F	G	D	B

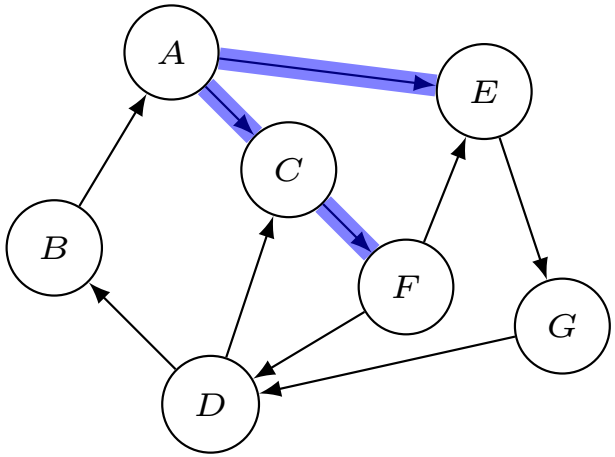
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

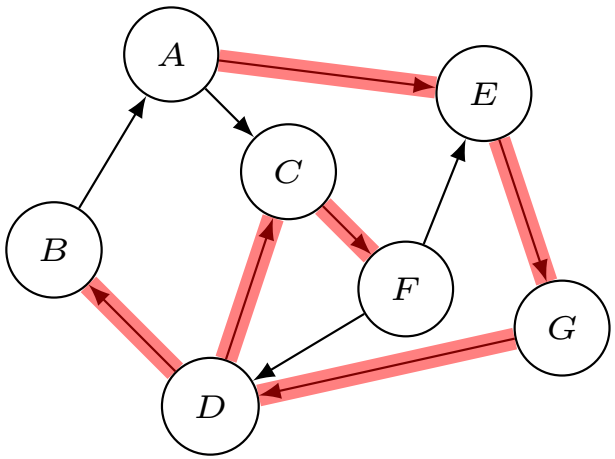


BFS ordning

A	C	E	F	G	D	B

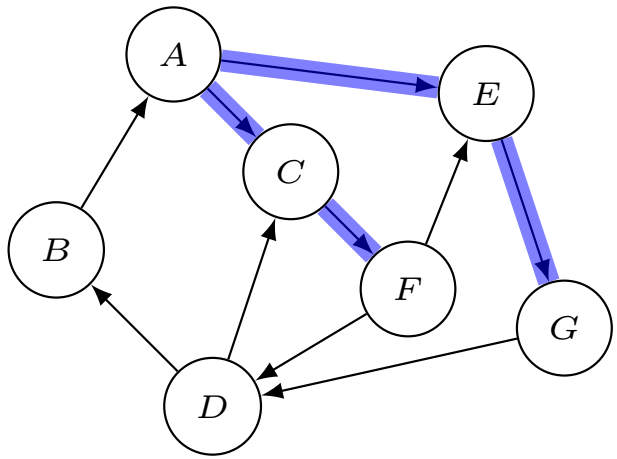
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

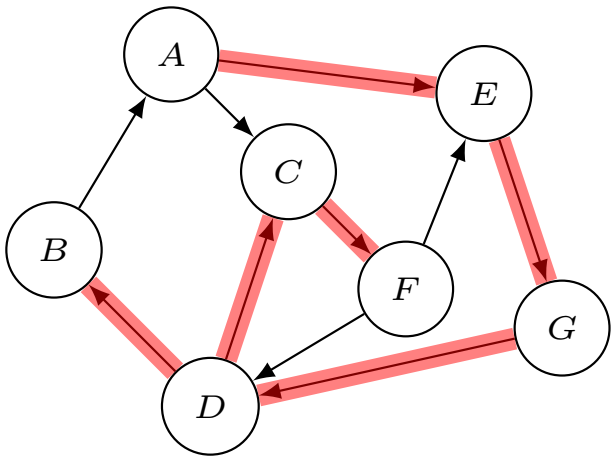


BFS ordning

A	C	E	F	G	D	B

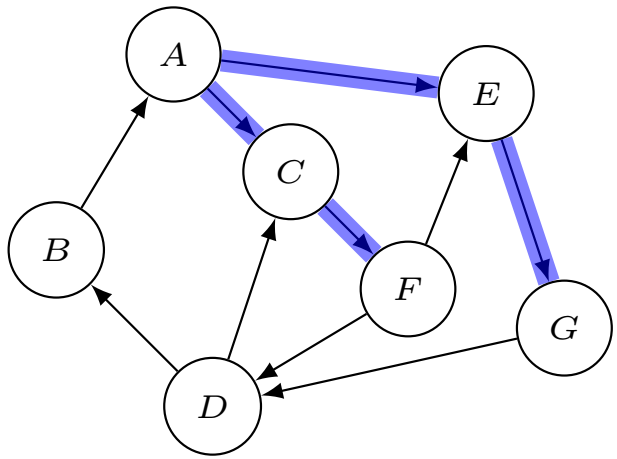
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

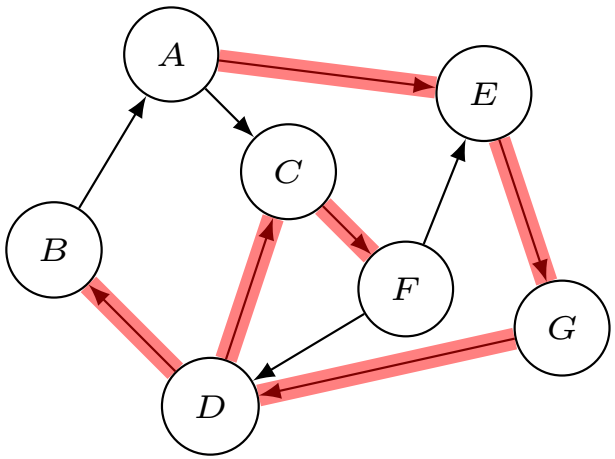


BFS ordning

A	C	E	F	G	D	B

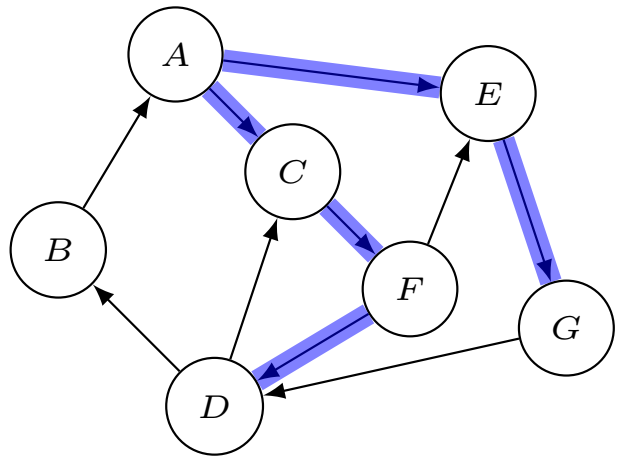
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

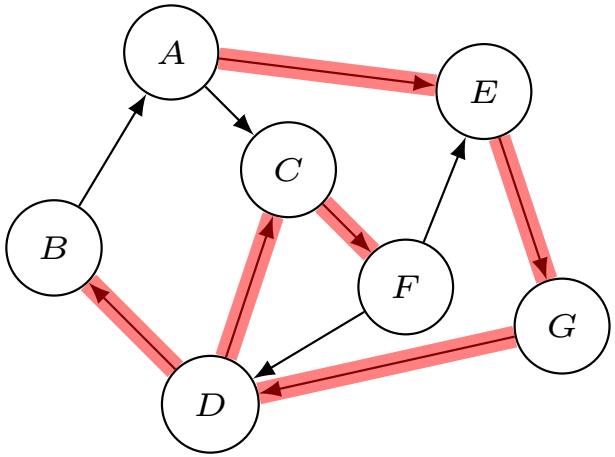


BFS ordning

A	C	E	F	G	D	B

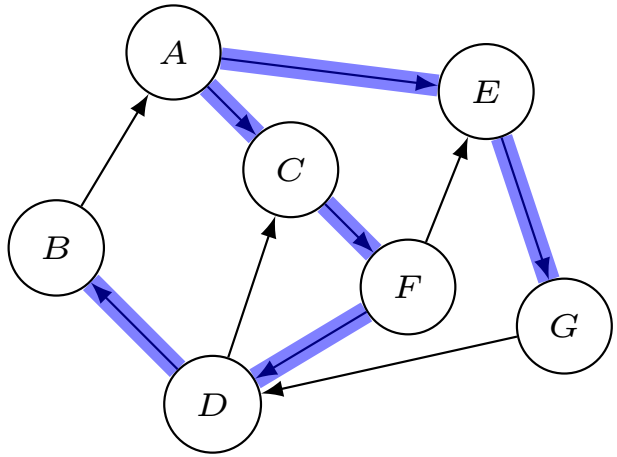
Grafsökning

Jämförelse DFS och BFS



DFS ordning

A	E	G	D	C	F	B

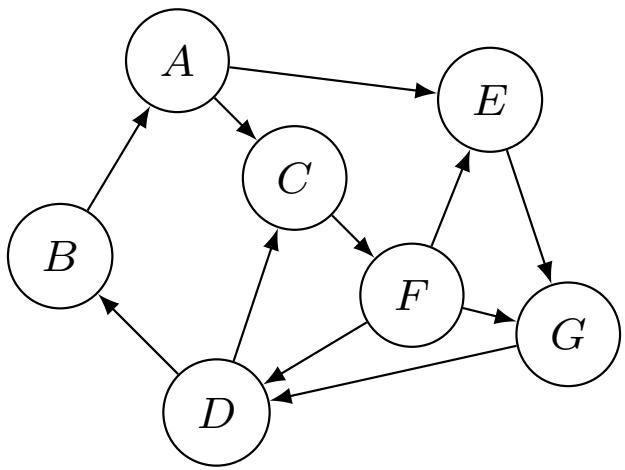


BFS ordning

A	C	E	F	G	D	B

Grafsökning

Hitta en kortaste väg mellan A och F



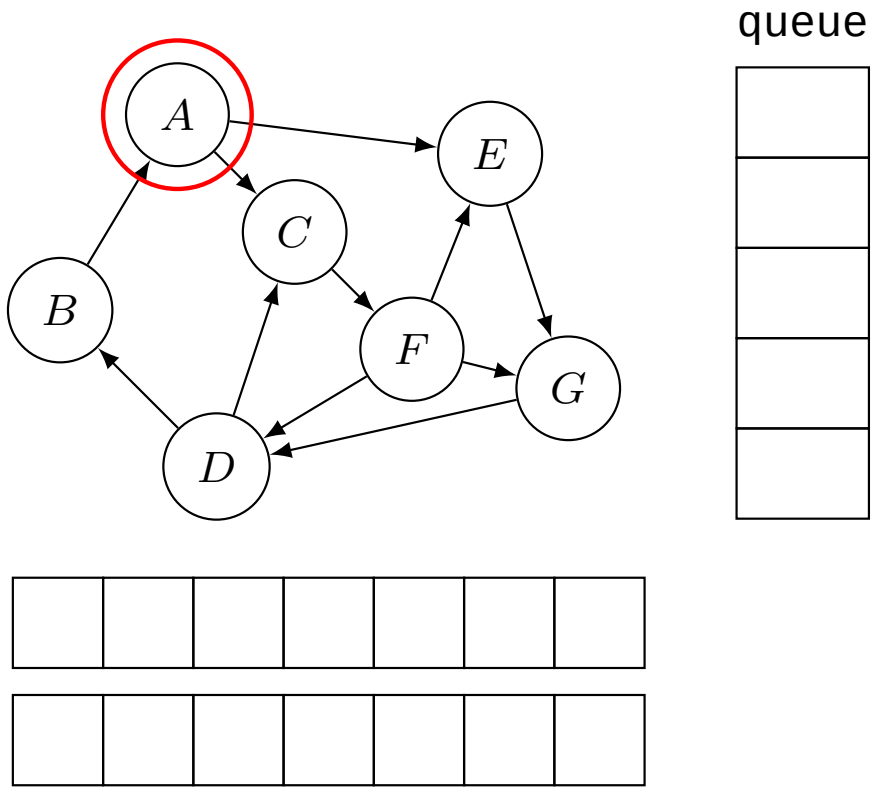
queue

$A, -$

found:							
from:							

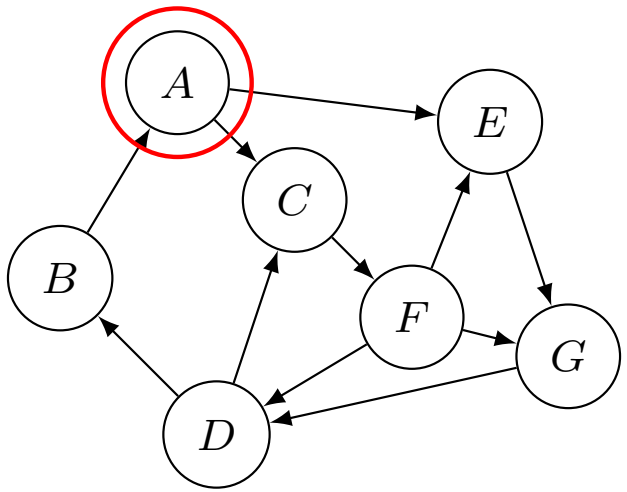
Grafsökning

Hitta en kortaste väg mellan A och F



Grafsökning

Hitta en kortaste väg mellan A och F

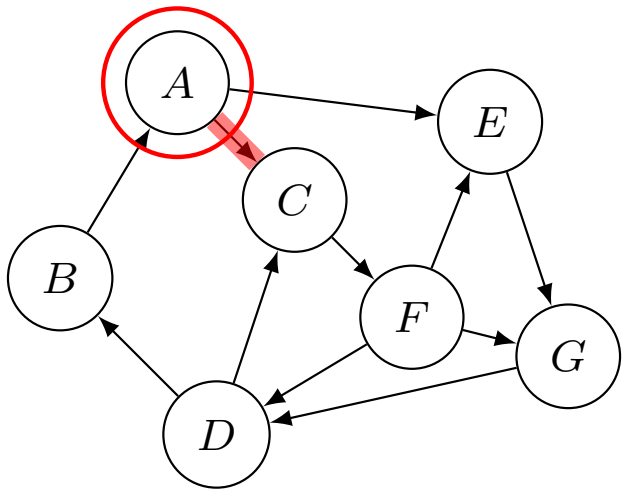


queue

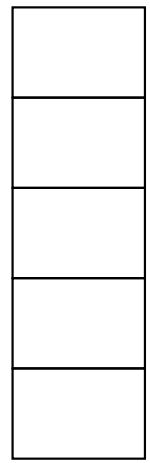
found:	A						
from:	$-$						

Grafsökning

Hitta en kortaste väg mellan A och F



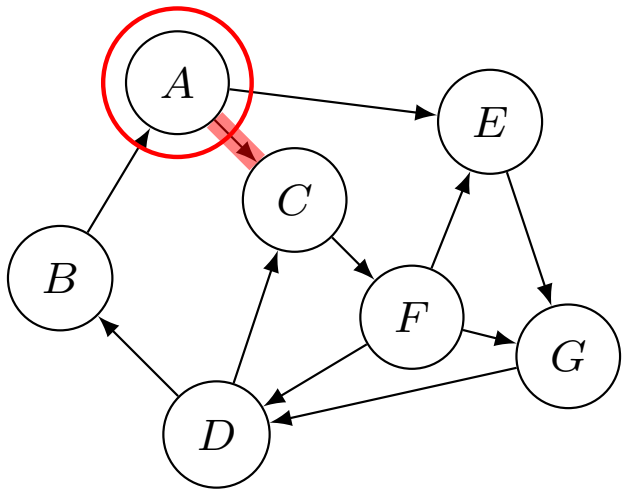
queue



found:	A						
from:	$-$						

Grafsökning

Hitta en kortaste väg mellan A och F



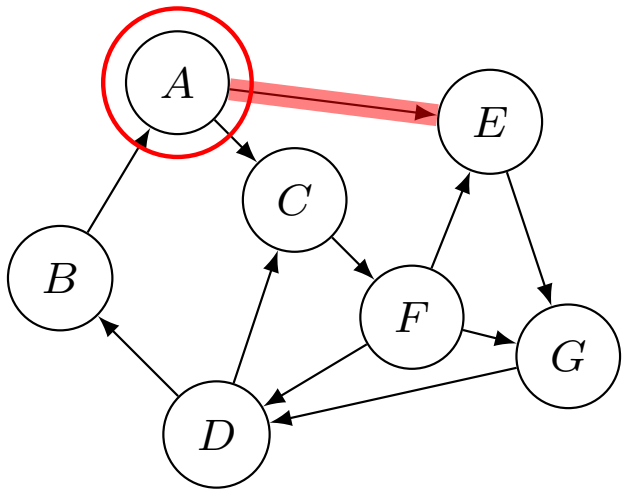
queue

C, A

found:	A						
from:	$-$						

Grafsökning

Hitta en kortaste väg mellan A och F



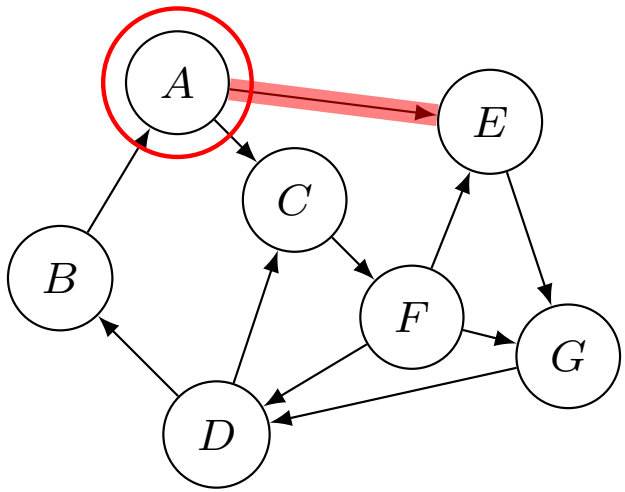
queue

C, A

found:	A						
from:	$-$						

Grafsökning

Hitta en kortaste väg mellan A och F



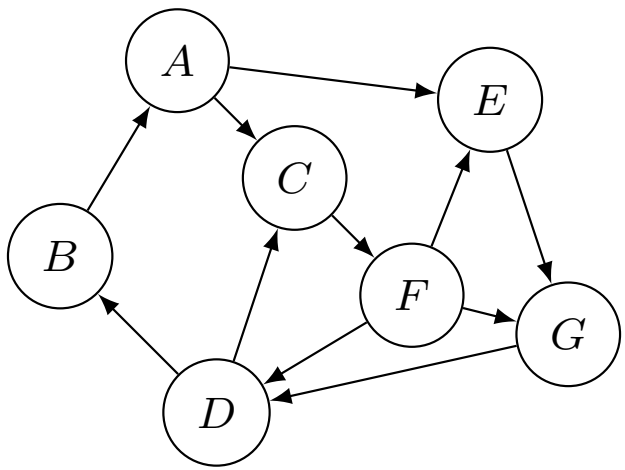
queue

C, A
E, A

found:	A						
from:	$-$						

Grafsökning

Hitta en kortaste väg mellan A och F



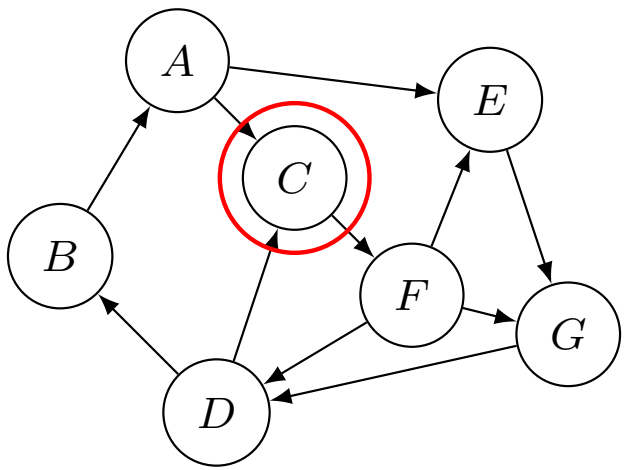
queue

C, A
E, A

found:	A						
from:	$-$						

Grafsökning

Hitta en kortaste väg mellan A och F



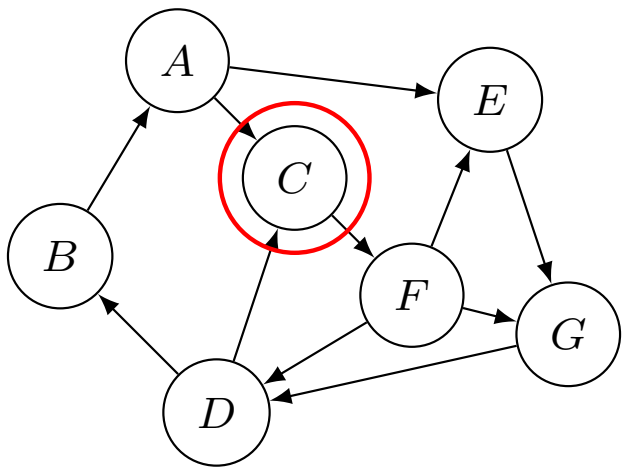
queue

E, A

found:	A						
from:	$-$						

Grafsökning

Hitta en kortaste väg mellan A och F



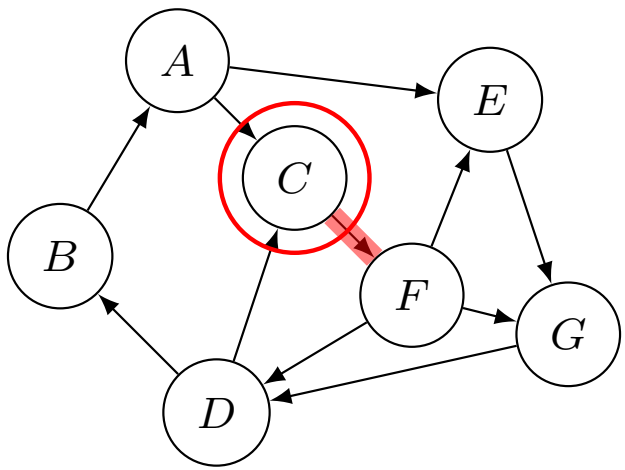
queue

E, A

found:	A	C					
from:	$-$	A					

Grafsökning

Hitta en kortaste väg mellan A och F



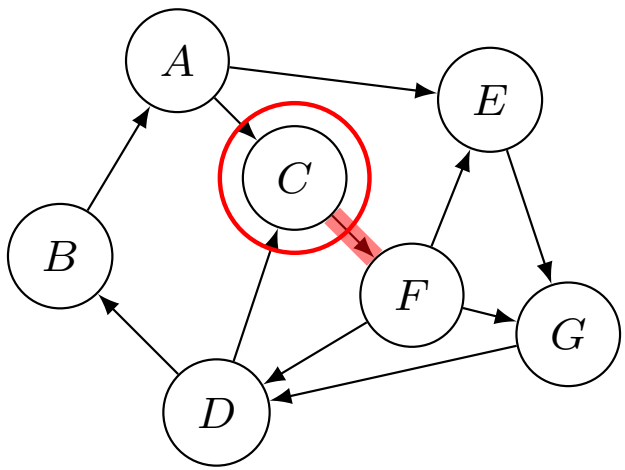
queue

E, A

found:	A	C					
from:	$-$	A					

Grafsökning

Hitta en kortaste väg mellan A och F



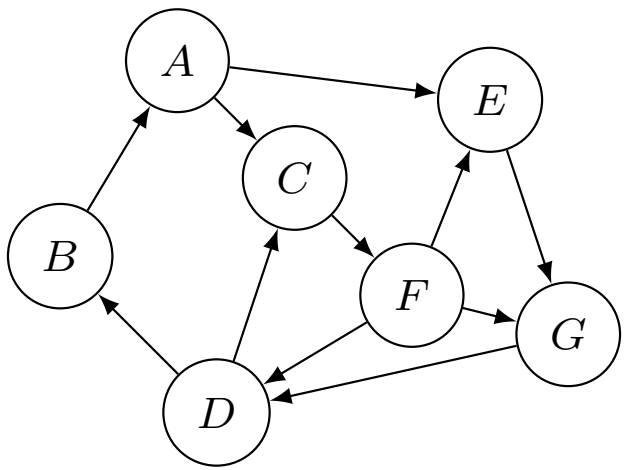
queue

E, A
F, C

found:	A	C					
from:	$-$	A					

Grafsökning

Hitta en kortaste väg mellan A och F



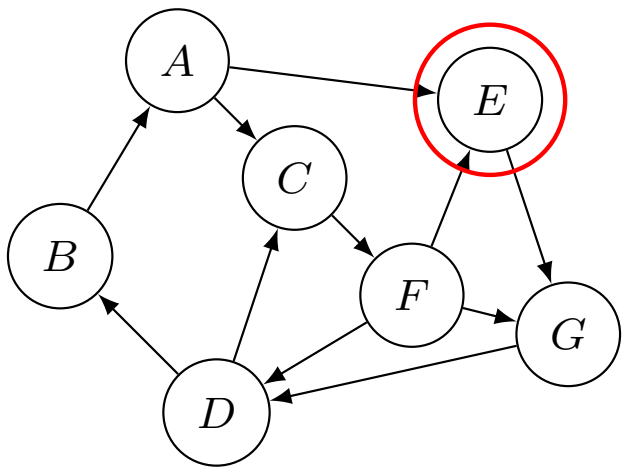
queue

E, A
F, C

found:	A	C					
from:	$-$	A					

Grafsökning

Hitta en kortaste väg mellan A och F



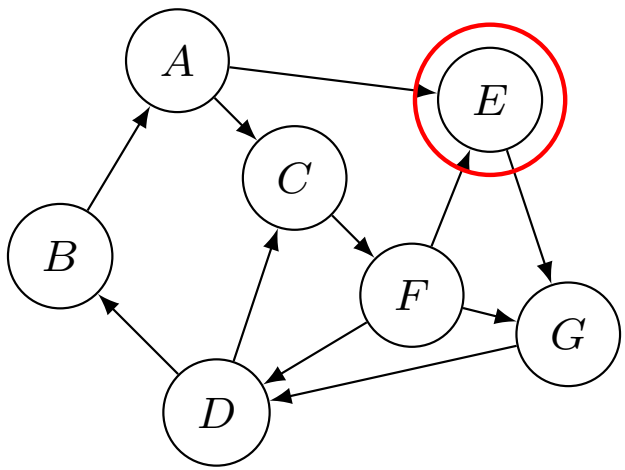
queue

F, C

found:	A	C					
from:	$-$	A					

Grafsökning

Hitta en kortaste väg mellan A och F



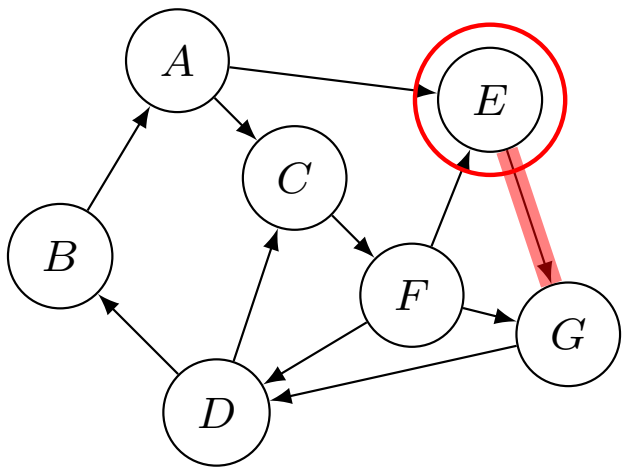
queue

F, C

found:	A	C	E				
from:	$-$	A	A				

Grafsökning

Hitta en kortaste väg mellan A och F



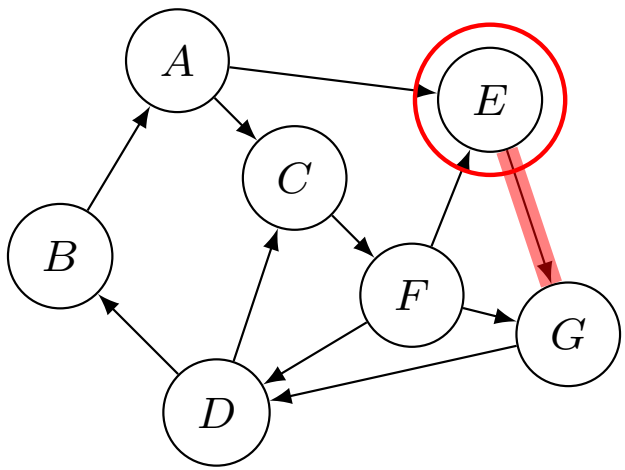
queue

F, C

found:	A	C	E				
from:	$-$	A	A				

Grafsökning

Hitta en kortaste väg mellan A och F



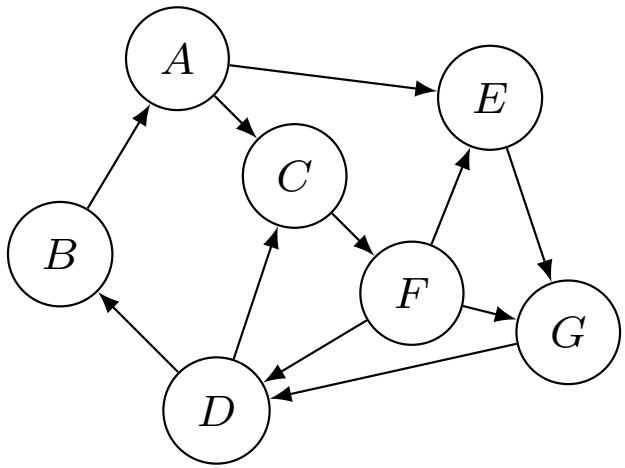
queue

F, C
G, E

found:	A	C	E				
from:	$-$	A	A				

Grafsökning

Hitta en kortaste väg mellan A och F



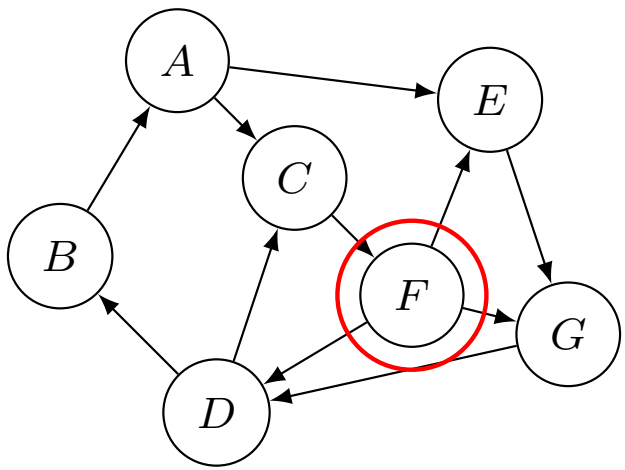
queue

F, C
G, E

found:	A	C	E				
from:	$-$	A	A				

Grafsökning

Hitta en kortaste väg mellan A och F



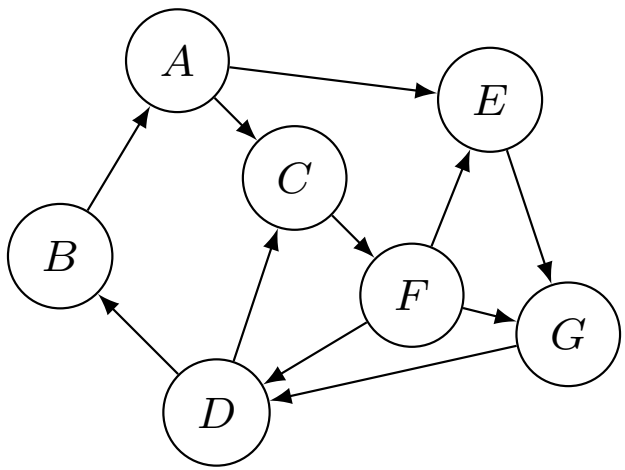
queue

G, E

found:	A	C	E				
from:	$-$	A	A				

Grafsökning

Hitta en kortaste väg mellan A och F



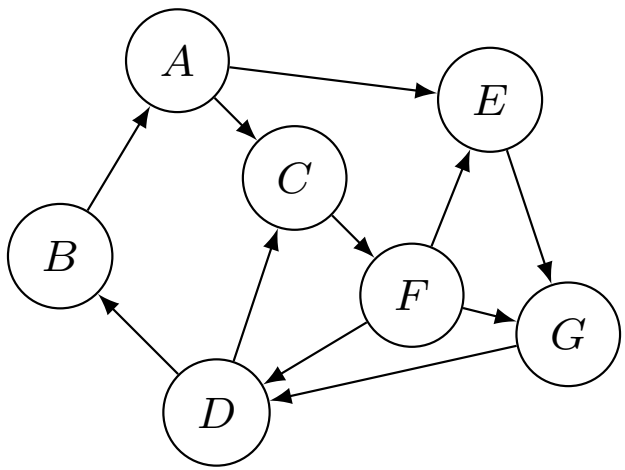
queue

G, E

found:	A	C	E	F			
from:	$-$	A	A	C			

Grafsökning

Hitta en kortaste väg mellan A och F



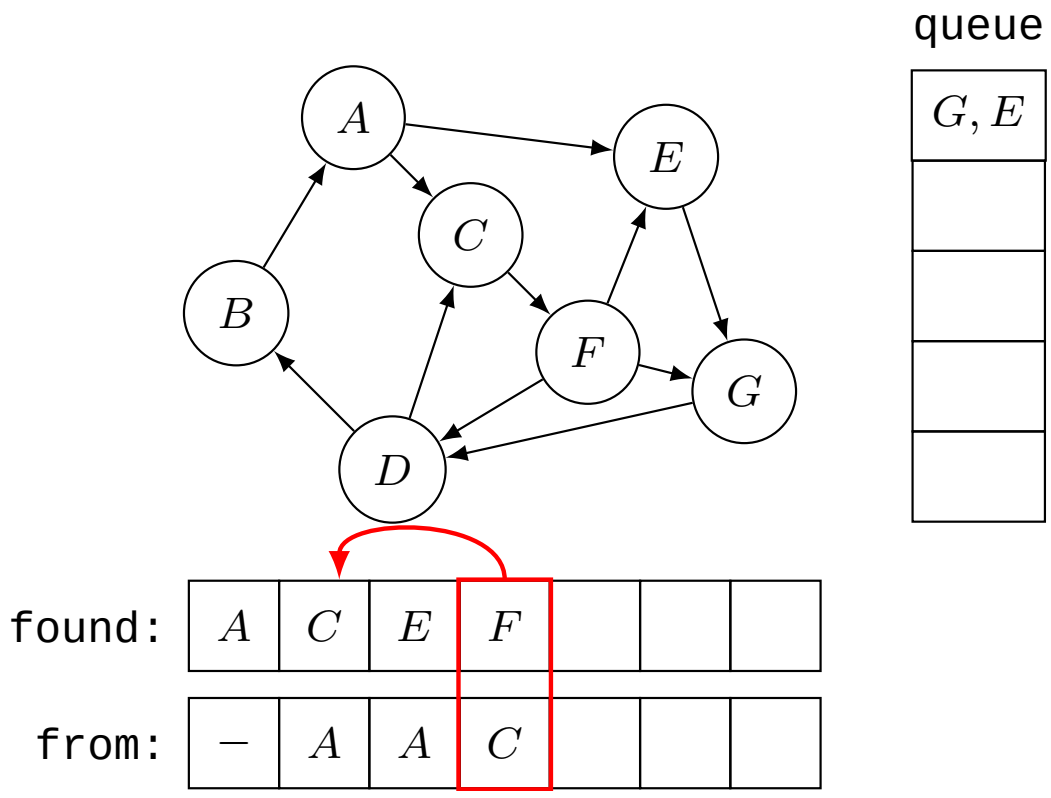
queue

G, E

found:	A	C	E	F			
from:	$-$	A	A	C			

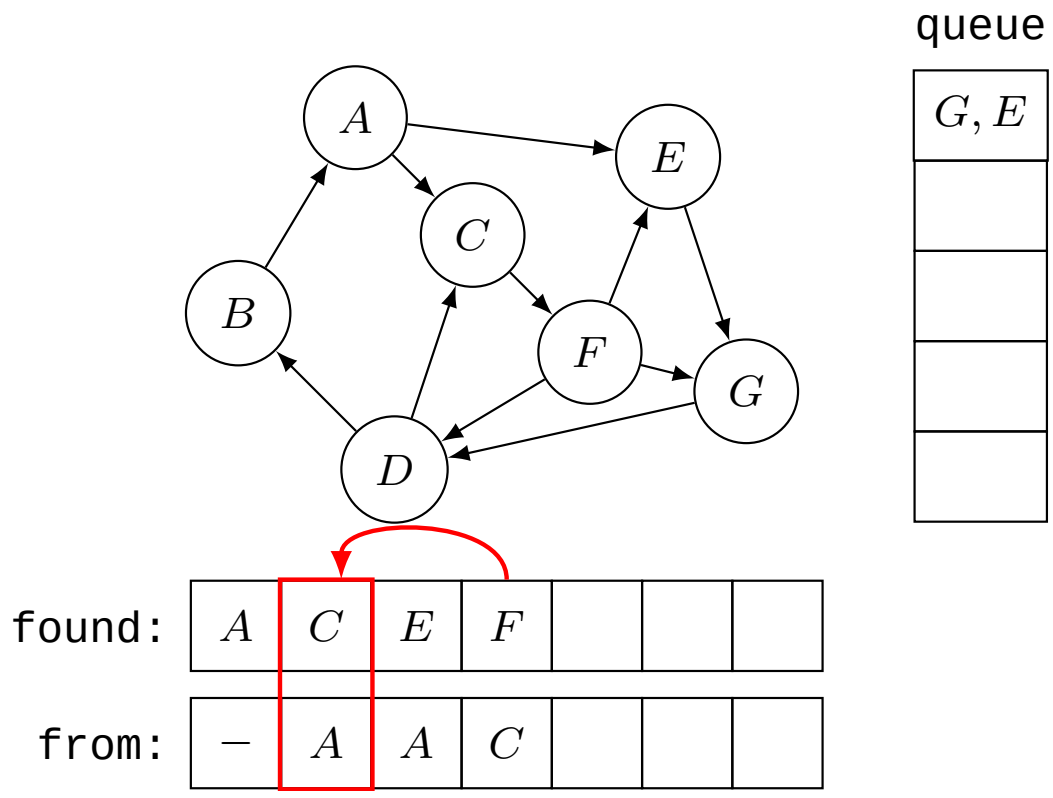
Grafsökning

Hitta en kortaste väg mellan A och F



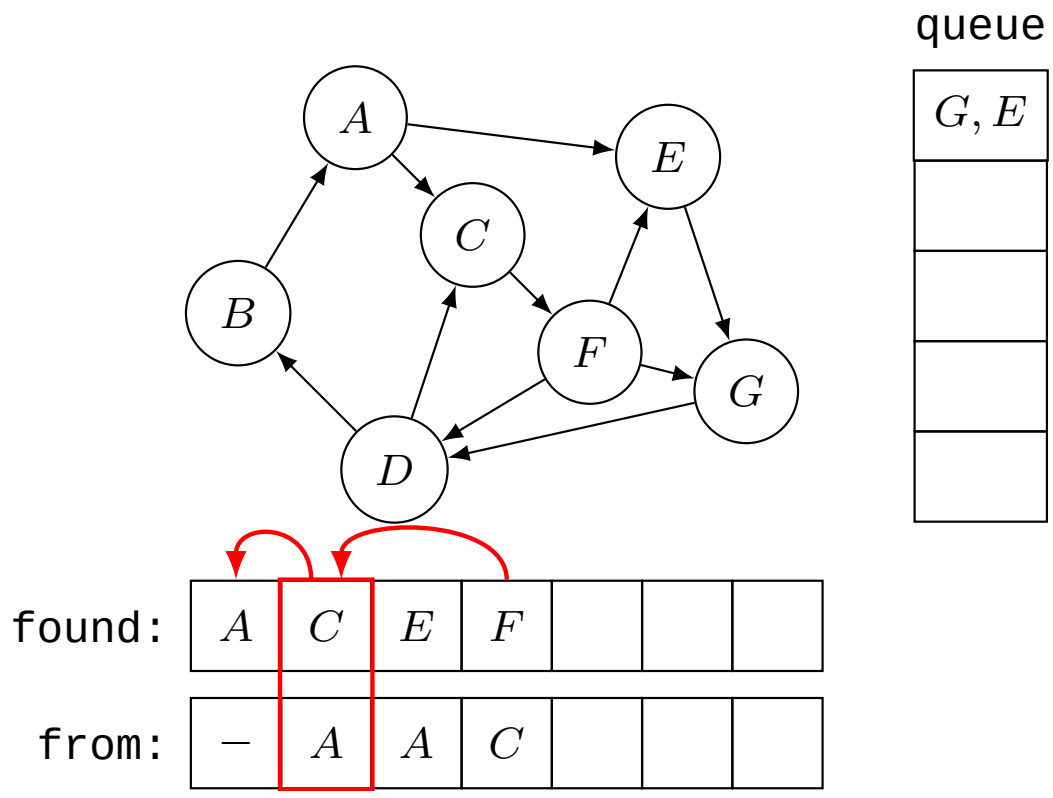
Grafsökning

Hitta en kortaste väg mellan A och F



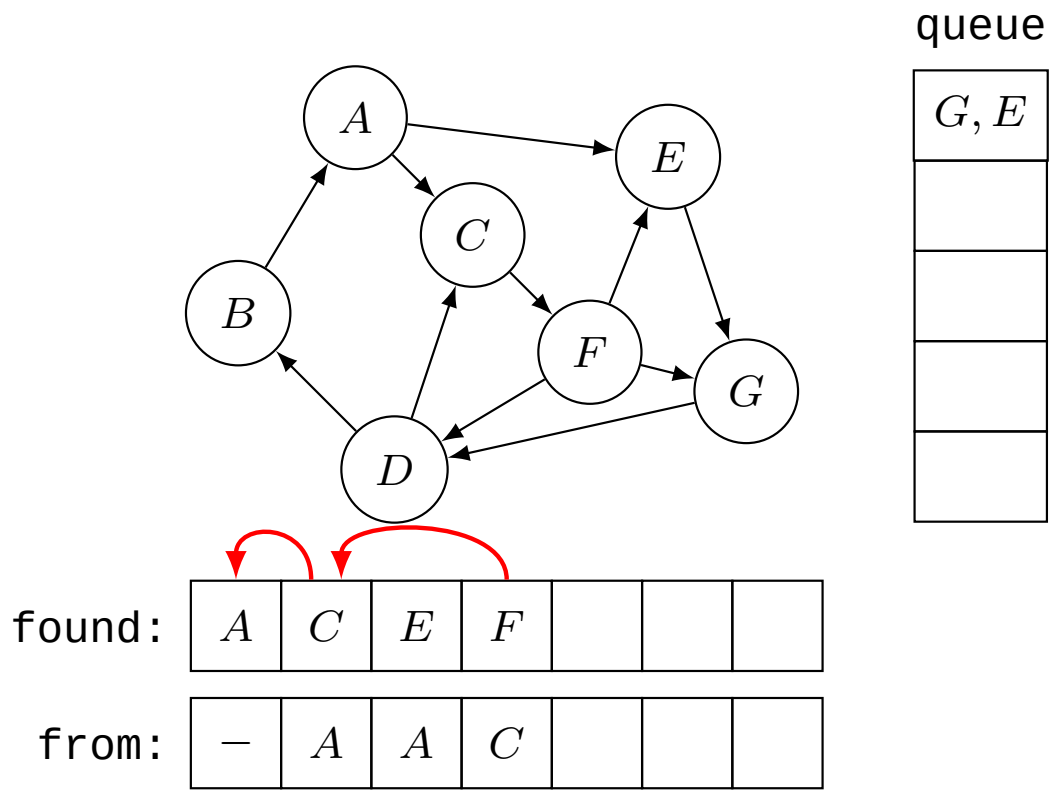
Grafsökning

Hitta en kortaste väg mellan A och F



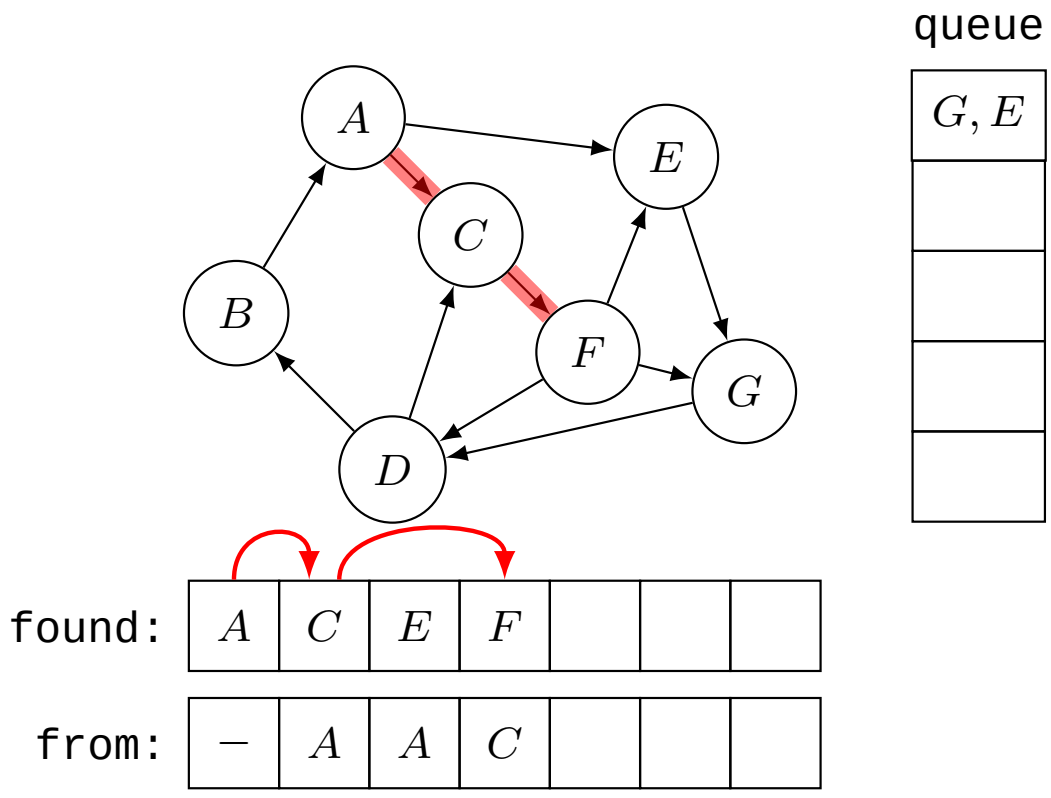
Grafsökning

Hitta en kortaste väg mellan A och F



Grafsökning

Hitta en kortaste väg mellan A och F



Grafsökning

Hitta kortaste väg mellan två noder

```
1  vector<Node*> shortest_path(Node* start, int end_id)
2  {
3    unordered_map<Node*, Node*> from;
4    queue<pair<Node*, Node*>> queue;
5    queue.push({ start, nullptr });
6    while (!queue.empty())
7    {
8      auto [curr, prev] = queue.front();
9      queue.pop();
10     if (from.count(curr) > 0) continue;
11     if (curr->id == end_id)
12       return produce_path(from, curr);
13
14     parents[curr] = prev;
15     for (Node* next : curr->edges)
16       queue.push({ next, curr });
17   }
18   return { }; // ingen väg hittade
19 }
```

Grafsökning

Hitta kortaste väg mellan två noder

```
1  vector<Node*> produce_path(unordered_map<Node*, Node*> const& from,
2                             Node* end)
3  {
4    vector<Node*> path;
5    Node* curr { end };
6    while (curr != nullptr)
7    {
8      path.push_back(curr);
9      curr = from.at(curr);
10   }
11
12   reverse(path.begin(), path.end());
13   return path;
14 }
```

Grafsökning

Billigaste väg

Definition: Billigaste väg

I en viktad graf $G_w = (V, E)$ ges *kostnaden* för en väg $P = \langle v_1, v_2, \ldots, v_n \rangle$ av

$$c(P) = \sum_{i=1}^{n-1} w(v_i, v_{i+1})$$

En billigaste väg är en väg P sådan att $c(P) \leq c(Q)$ för alla vägar Q. Dessa vägar hittas m.h.a. *Dijkstras algoritm*.

1 Representation av grafer
2 Grafsökning
3 Heap

LINKÖPING UNIVERSITY

Heap

Mål med prioritetskö

- Slå upp högst prioritet: $\mathcal{O}(1)$
- Ta bort högst prioritet: Helst bättre än $\mathcal{O}(n)$
- Insättningen: Helst bättre än $\mathcal{O}(n)$

Heap

Mål med prioritetskö

- Slå upp högst prioritet: $\mathcal{O}(1)$
- Ta bort högst prioritet: Helst bättre än $\mathcal{O}(n)$
- Insättningen: Helst bättre än $\mathcal{O}(n)$
- Antag att högsta prioritet är det *minsta* värdet

Heap

Idé

Idé #1

Använd ett balanserat binärt sökträd, då är allt $\mathcal{O}(\log n)$!

Heap

Observation #1

Observation #1

Att hitta rotnoden i ett träd görs i $\mathcal{O}(1)$.

Heap

Idé #2

Idé #2

Behåll det balanserade binära trädet men skippa sökstrukturen. Lagra istället det minsta värdet i roten.

Heap

Observation #2

Observation #2

Om vi strukturerar trädet s.a. det näst-minsta värdet också kan hittas i konstant tid så blir borttagningen mycket enklare.

Heap

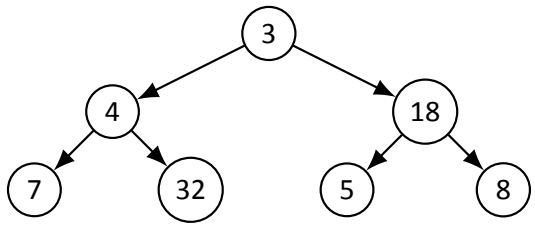
Idé #3

Idé #3

Se till att det näst-minsta värdet alltid ligger som ett barn till roten.

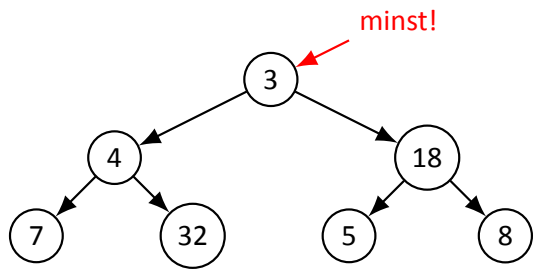
Heap

Vad vi har hittills



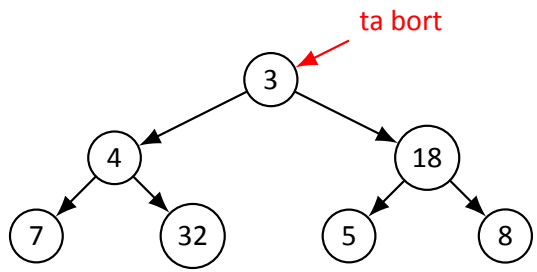
Heap

Vad vi har hittills



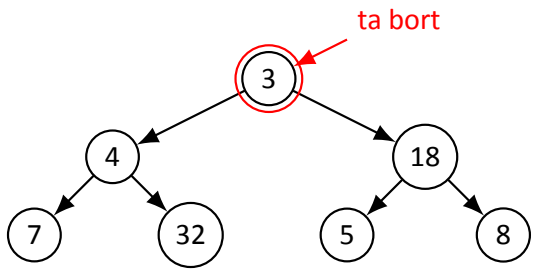
Heap

Vad vi har hittills



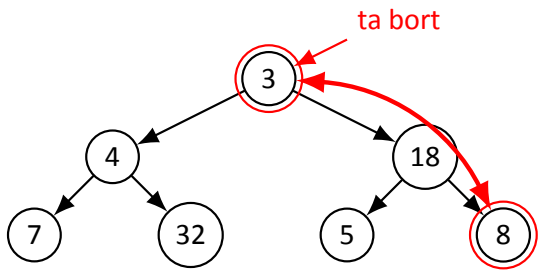
Heap

Vad vi har hittills



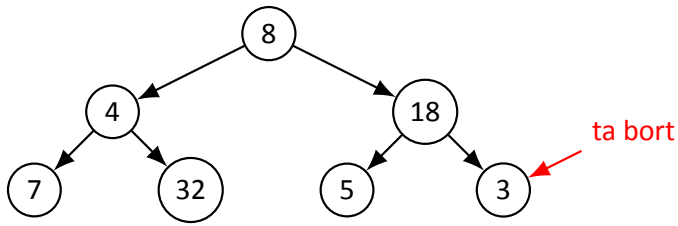
Heap

Vad vi har hittills



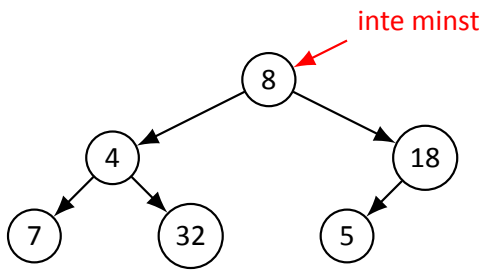
Heap

Vad vi har hittills



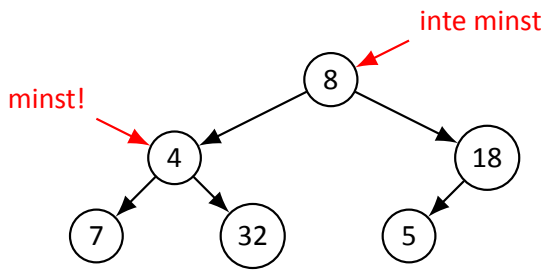
Heap

Vad vi har hittills



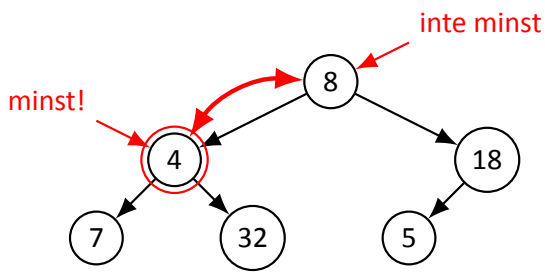
Heap

Vad vi har hittills



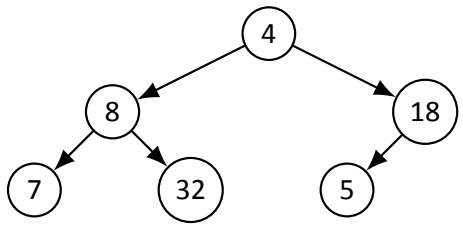
Heap

Vad vi har hittills



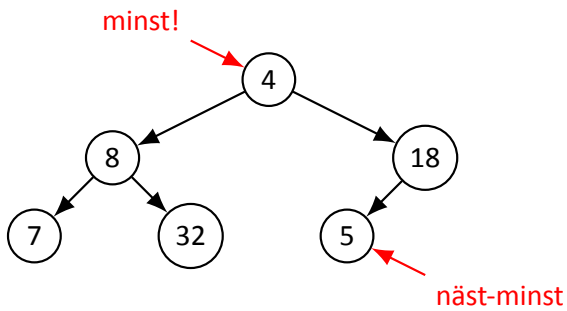
Heap

Vad vi har hittills



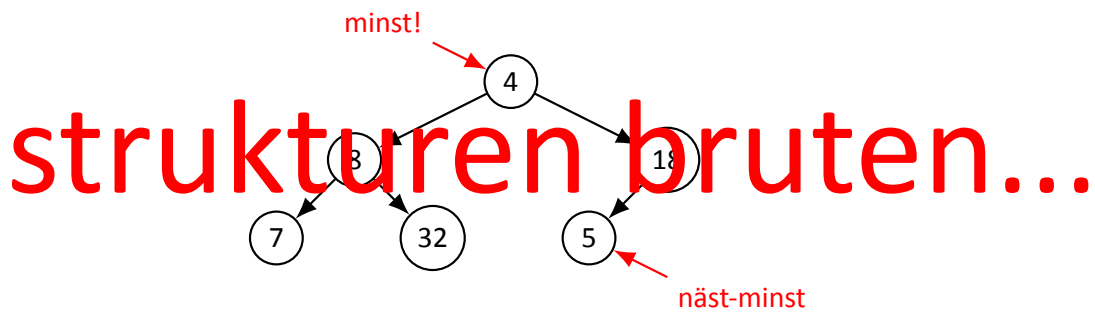
Heap

Vad vi har hittills



Heap

Vad vi har hittills



Heap

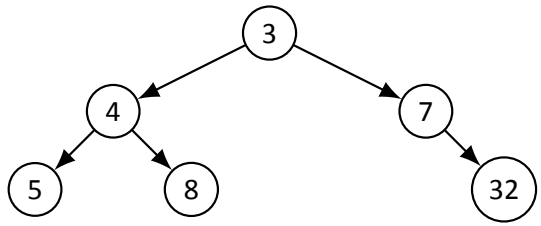
Idé #4

Idé #4

Vad händer om vi säger att alla barn ska vara *större* än sina föräldrar?

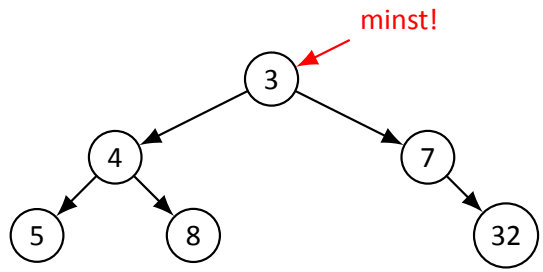
Heap

Mer struktur



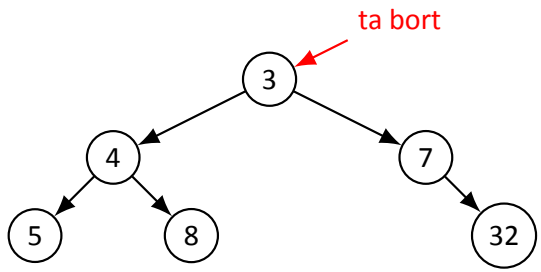
Heap

Mer struktur



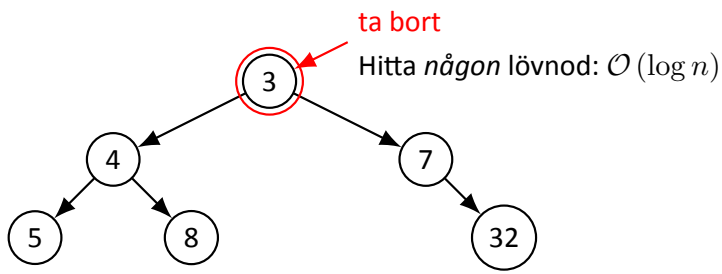
Heap

Mer struktur



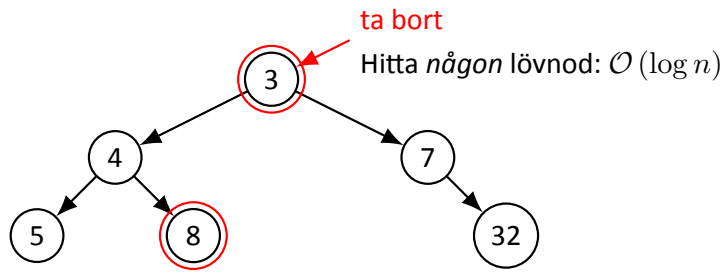
Heap

Mer struktur



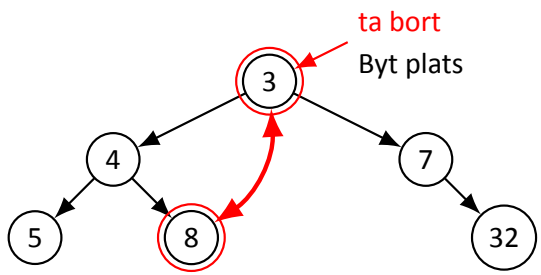
Heap

Mer struktur



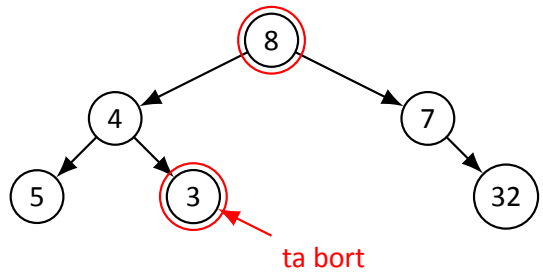
Heap

Mer struktur



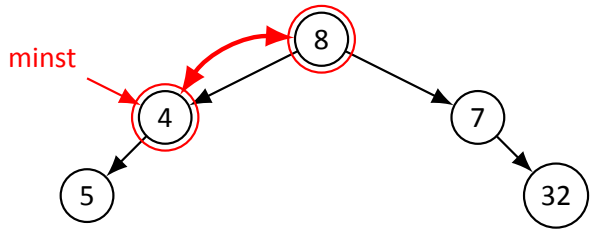
Heap

Mer struktur



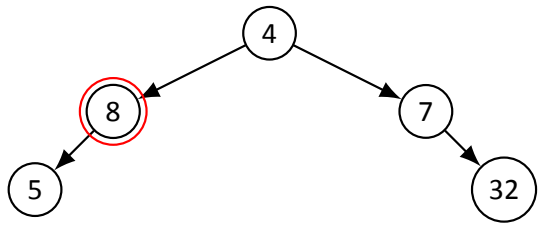
Heap

Mer struktur



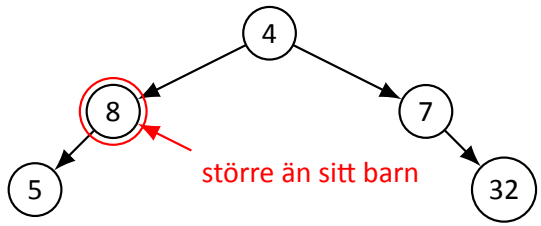
Heap

Mer struktur



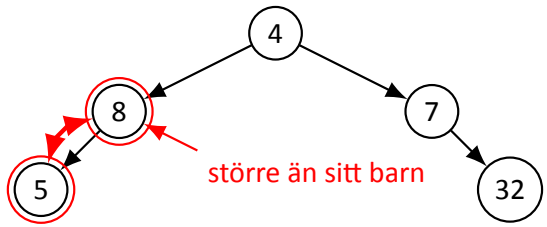
Heap

Mer struktur



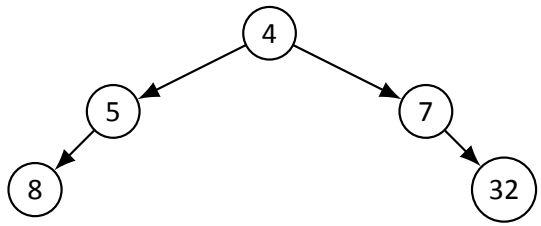
Heap

Mer struktur



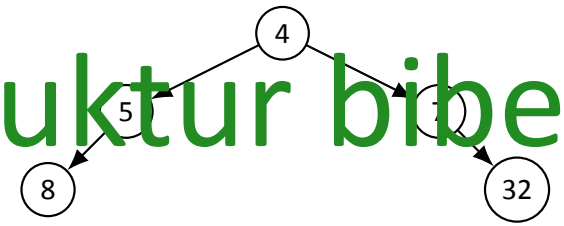
Heap

Mer struktur



Heap

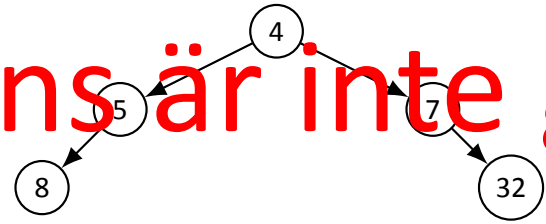
Mer struktur



Struktur bibehålls!

Heap

Mer struktur



Men balans är inte garanterat...

Heap

Nästan komplett träd

Definition: Komplett– och nästan komplett träd

- Ett träd $\mathcal{T}$ är *komplett* om det har exakt $2^k - 1$ noder fördelade i exakt k nivåer.

Heap

Nästan komplett träd

Definition: Komplett– och nästan komplett träd

- Ett träd $\mathcal{T}$ är *komplett* om det har exakt $2^k - 1$ noder fördelade i exakt k nivåer.
- Ett träd $\mathcal{T}$ är *nästan komplett* om alla nivår förutom sista är fulla, och den sista nivån är fylld från vänster till höger.

Heap

Nästan komplett träd

Definition: Komplett– och nästan komplett träd

- Ett träd $\mathcal{T}$ är *komplett* om det har exakt $2^k - 1$ noder fördelade i exakt k nivåer.
- Ett träd $\mathcal{T}$ är *nästan komplett* om alla nivår förutom sista är fulla, och den sista nivån är fylld från vänster till höger.
- **Följd:** Ett nästan komplett träd är balanserat

Heap

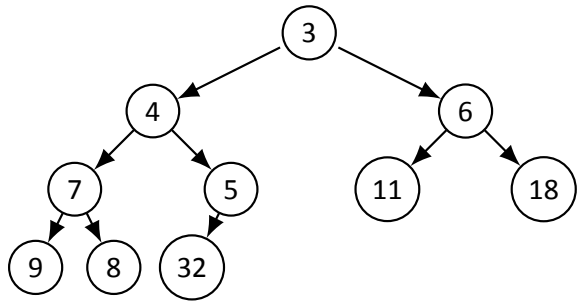
Idé #5

Idé #5

- Lagra vår prioritetskö i ett nästan komplett träd, d.v.s. fyll varje nivå från vänster till höger vid insättning.
- Vid borttagning, välj alltid den lövnod som är mest till höger på sista nivån

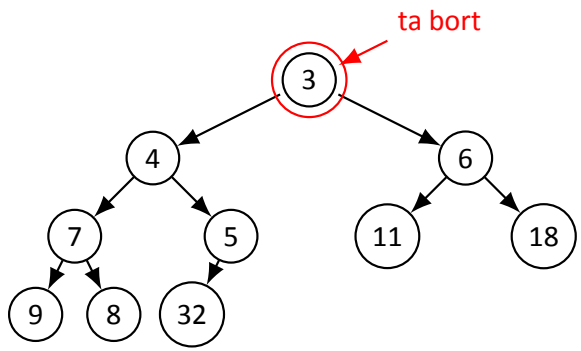
Heap

Nästan komplett träd



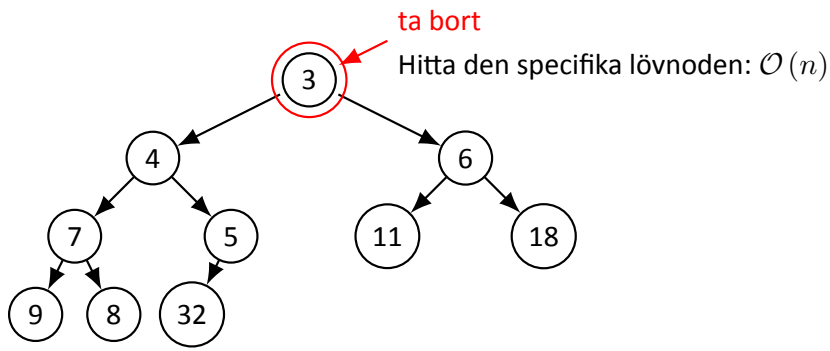
Heap

Nästan komplett träd



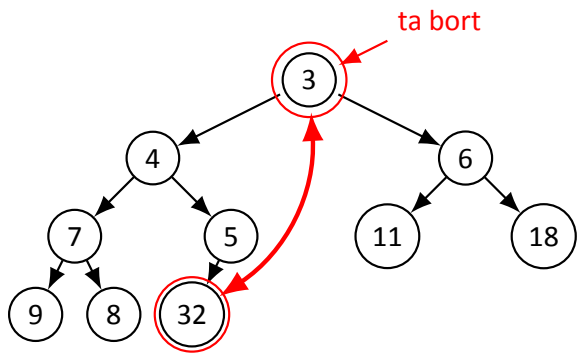
Heap

Nästan komplett träd



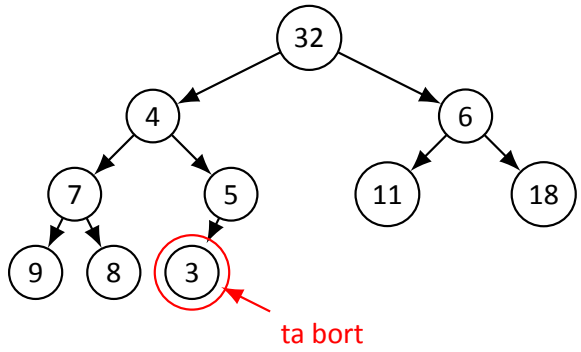
Heap

Nästan komplett träd



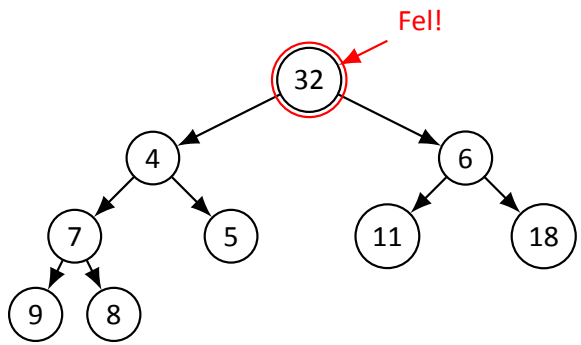
Heap

Nästan komplett träd



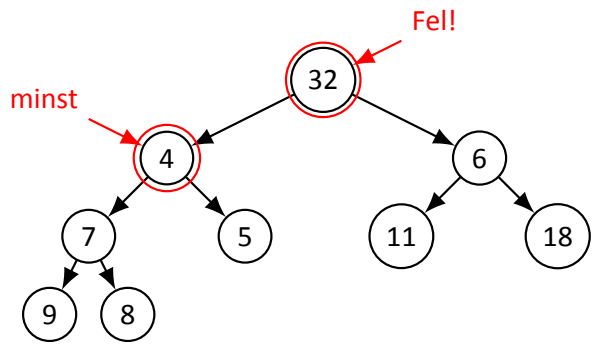
Heap

Nästan komplett träd



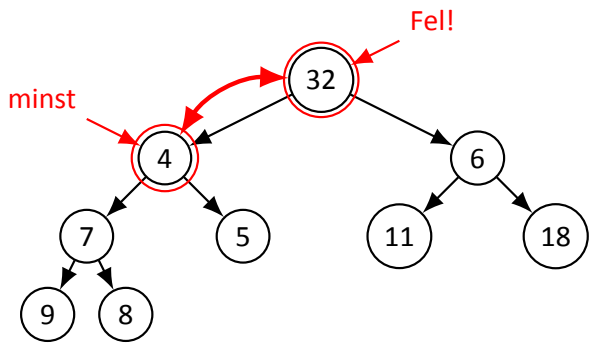
Heap

Nästan komplett träd



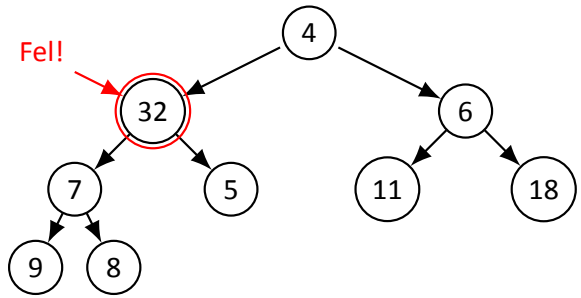
Heap

Nästan komplett träd



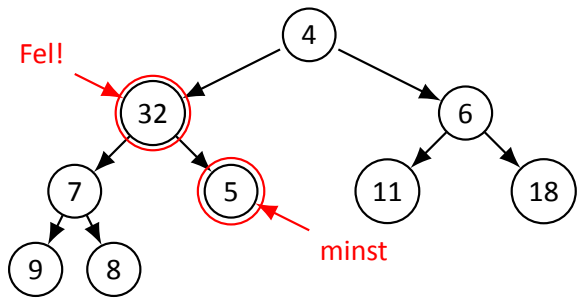
Heap

Nästan komplett träd



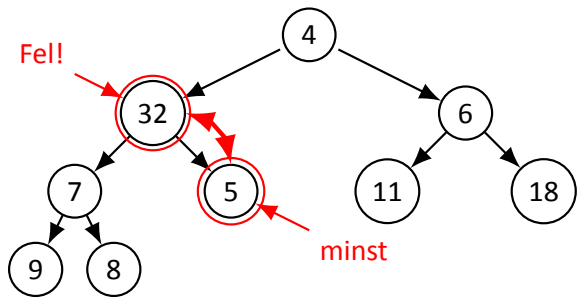
Heap

Nästan komplett träd



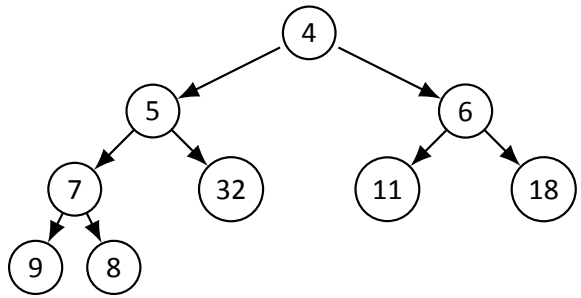
Heap

Nästan komplett träd



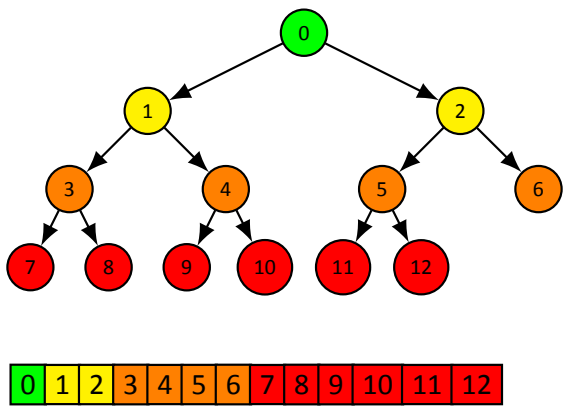
Heap

Nästan komplett träd



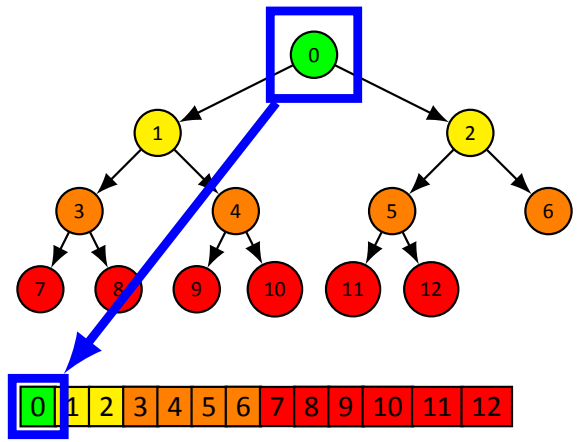
Heap

Lagra ett nästan komplett träd i en dynamisk array



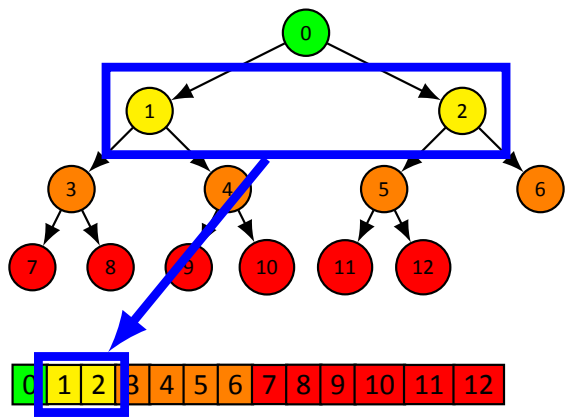
Heap

Lagra ett nästan komplett träd i en dynamisk array



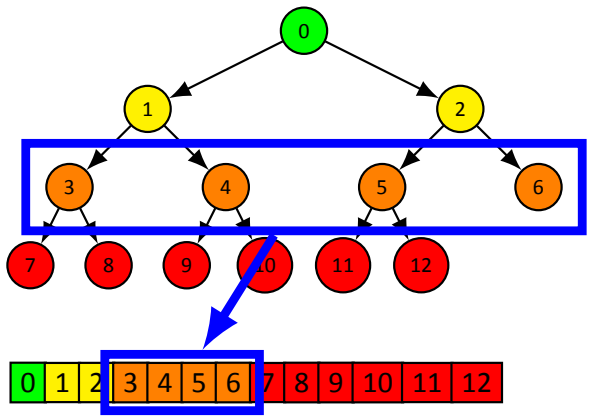
Heap

Lagra ett nästan komplett träd i en dynamisk array



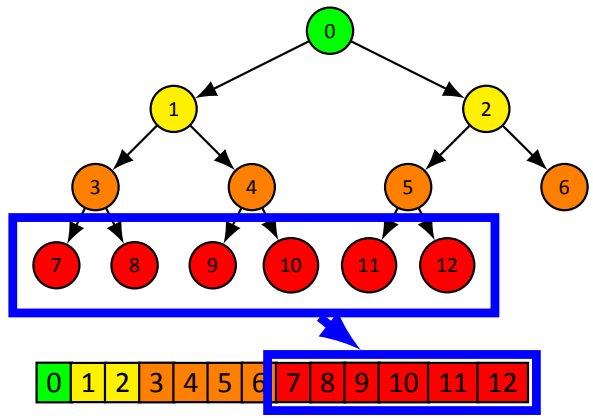
Heap

Lagra ett nästan komplett träd i en dynamisk array



Heap

Lagra ett nästan komplett träd i en dynamisk array



Heap

Lagra ett nästan komplett träd i en dynamisk array

Avbildning från träd till array

Vi avbildar ett nästan komplett träd till en array såhär:

Heap

Lagra ett nästan komplett träd i en dynamisk array

Avbildning från träd till array

Vi avbildar ett nästan komplett träd till en array såhär:

- Rotnoden lagras på index 0

Heap

Lagra ett nästan komplett träd i en dynamisk array

Avbildning från träd till array

Vi avbildar ett nästan komplett träd till en array såhär:

- Rotnoden lagras på index 0
- Nod på index i lagrar sitt vänstra barn på index $2i + 1$

Heap

Lagra ett nästan komplett träd i en dynamisk array

Avbildning från träd till array

Vi avbildar ett nästan komplett träd till en array såhär:

- Rotnoden lagras på index 0
- Nod på index i lagrar sitt vänstra barn på index $2i + 1$
- Nod på index i lagrar sitt högra barn på index $2i + 2$

Heap

Lagra ett nästan komplett träd i en dynamisk array

Avbildning från träd till array

Vi avbildar ett nästan komplett träd till en array såhär:

- Rotnoden lagras på index 0
- Nod på index i lagrar sitt vänstra barn på index $2i + 1$
- Nod på index i lagrar sitt högra barn på index $2i + 2$
- Nod på index i har nod på index $\left\lfloor \frac{i-1}{2} \right\rfloor$ som förälder

Heap

Heaps

Definition: Heap

- En *min-heap* är ett binärt träd $\mathcal{H}$ där:
  (i) $\mathcal{H}$ är ett *nästan komplett* träd
  (ii) Varje nod i $\mathcal{H}$ är *mindre* än båda sina barn

Heap

Heaps

Definition: Heap

- En *min-heap* är ett binärt träd $\mathcal{H}$ där:
  - (i) $\mathcal{H}$ är ett *nästan komplett* träd
  - (ii) Varje nod i $\mathcal{H}$ är *mindre* än båda sina barn
- En *max-heap* är ett binärt träd $\mathcal{H}$ där:
  - (i) $\mathcal{H}$ är ett *nästan komplett* träd
  - (ii) Varje nod i $\mathcal{H}$ är *större* än båda sina barn

Heap

Heaps, egenskaper

- Hitta minsta/största: $\mathcal{O}(1)$

Heap

Heaps, egenskaper

- Hitta minsta/största: $\mathcal{O}(1)$
- Insättning: $\mathcal{O}(\log n)$

Heap

Heaps, egenskaper

- Hitta minsta/största: $\mathcal{O}(1)$
- Insättning: $\mathcal{O}(\log n)$
- Borttagning: $\mathcal{O}(\log n)$

Heap

Implementation

```
1  class Min_Heap
2  {
3  public:
4
5    Min_Heap() = default;
6
7    void push(int value);
8
9    int pop_min();
10   int peek_min() const;
11
12 private:
13
14   void sift_down(unsigned index);
15   void sift_up(unsigned index);
16
17   vector<int> values;
18 };
```

Heap

Implementation: Hjälpfunktioner

```
1  unsigned left(unsigned index)
2  {
3    return 2*index + 1;
4  }
5
6  unsigned right(unsigned index)
7  {
8    return 2*index + 2;
9  }
10
11 unsigned parent(unsigned index)
12 {
13   return (index - 1) / 2;
14 }
```

Heap

Implementation: sift-down

```
1  void Min_Heap::sift_down(unsigned index)
2  {
3    if (index >= values.size()) return;
4    unsigned l { left (index) }; // hitta vänsterbarn
5    unsigned r { right(index) }; // hitta högerbarn
6
7    // om vänster inte finns så finns inga barn
8    if (l >= values.size()) return;
9
10   unsigned next { l }; // hitta det minsta barnet
11   if (r < values.size() && values[r] < values[l])
12     next = r;
13
14   // om vi redan är mindre än vårt minsta barn är vi klara
15   if (values[index] < values[next]) return;
16
17   // annars byter vi plats med minsta barnet och rekurserar
18   swap(values[index], values[next]);
19   sift_down(next);
20 }
```

Heap

Implementation: sift-up

```
1  void Min_Heap::sift_up(unsigned index)
2  {
3    if (index == 0 || index >= values.size()) return;
4
5    // hitta föräldern
6    unsigned p { parent(index) };
7
8    // om heap egenskapen redan är uppfylld är vi klara
9    if (values[p] < values[index]) return;
10
11   // annars byter vi värdena och rekurserar
12   swap(values[p], values[index]);
13   sift_up(p);
14 }
```

Heap

Implementation: Insättning, borttagning och hitta

```
1  void Min_Heap::push(int value)
2  {
3    values.push_back(value); // lägg till som sista lövnod
4    sift_up(values.size() - 1); // flytta uppåt tills det är en heap igen
5  }
6
7  int Min_Heap::pop_min()
8  {
9    int min { values.front() }; // spara det minsta värdet
10   swap(values.front(), values.back()); // byt plats med sista lövnod
11   values.pop_back(); // ta bort sista lövnoden
12   sift_down(0); // flytta nedåt tills det är en heap igen
13
14 }
15
16 int Min_Heap::peek_min()
17 {
18   return values.front();
19 }
```

Heap

Avslut

- Kan man göra bättre?

Heap

Avslut

- Kan man göra bättre?
- Ja, det finns t.ex. något som heter en *Fibonacci heap* där insättning blir $\mathcal{O}(1)$ amorterat.

Heap

Avslut

- Kan man göra bättre?
- Ja, det finns t.ex. något som heter en *Fibonacci heap* där insättning blir $\mathcal{O}(1)$ amorterat.
- En Fibonacci heap är dock mer komplicerad att implementera och analysera.

Heap

Avslut

- Kan man göra bättre?
- Ja, det finns t.ex. något som heter en *Fibonacci heap* där insättning blir $\mathcal{O}(1)$ amorterat.
- En Fibonacci heap är dock mer komplicerad att implementera och analysera.
- Fibonacci heaps bygger på den något simplare *Binomial heap*

Heap

Avslut

- Kan man göra bättre?
- Ja, det finns t.ex. något som heter en *Fibonacci heap* där insättning blir $\mathcal{O}(1)$ amorterat.
- En Fibonacci heap är dock mer komplicerad att implementera och analysera.
- Fibonacci heaps bygger på den något simplare *Binomial heap*
- `https://www.cs.usfca.edu/~galles/visualization/FibonacciHeap.html`

www.liu.se

LINKÖPING UNIVERSITY