

TDDI16 – Föreläsning 6

Grafer

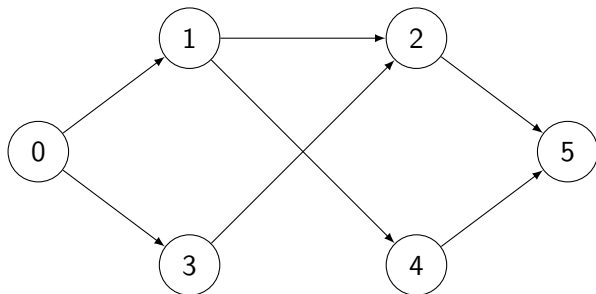
Filip Strömbäck

Planering

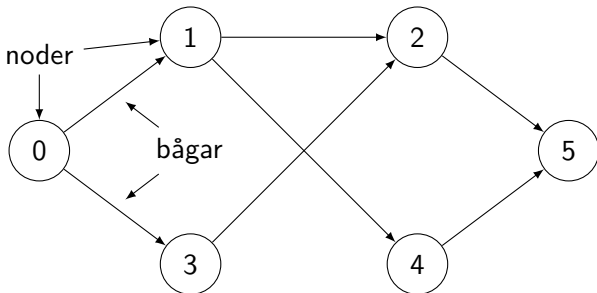
Vecka	Fö	Lab
35	Komplexitet, Linjära strukturer	----
36	Träd, AVL-träd	1---
37	Hashning, meet-in-the-middle	12--
38	Grafer och kortaste vägen	-23-
39	Fler grafalgoritmer	--3-
40	Sortering	--34
41	Repetition	---4

- 1 Grafer
- 2 Graftsökning i oviktade grafer
- 3 Graftsökning i viktade grafer
- 4 Graftsökning applicerat på andra problem
- 5 Sammanfattning

Vad är en graf?

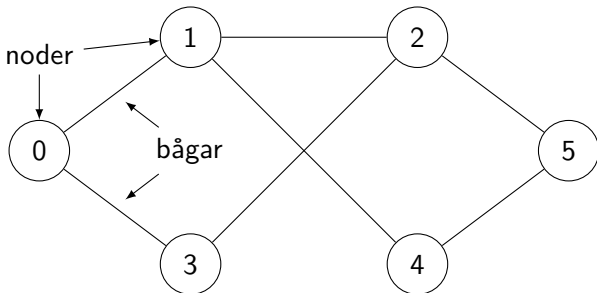


Vad är en graf?



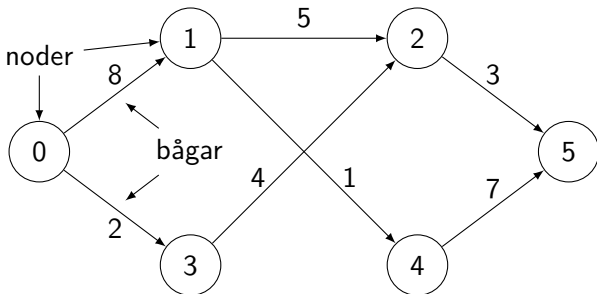
Riktad graf

Vad är en graf?



Oriktad graf

Vad är en graf?



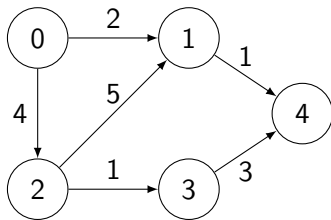
Viktade bågar

Hur representeras en graf?

- Formell definition:
 $G = (V, E)$
 V : en mängd av alla noder (*vertices*)
 E : en mängd av par motsvarande alla bågar (*edges*)
- Grannmatris
- Grannlistor
- Generera grafen "on the fly"

Grannmatrix (adjacency matrix)

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



Minnesanvändning

$\mathcal{O}(|V|^2)$

Finns det en båge från x till y ?

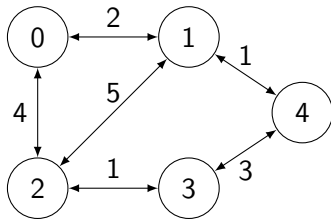
$\mathcal{O}(1)$

Vilka bågar finns från x ?

$\mathcal{O}(|V|)$

Grannmatris (adjacency matrix)

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



Minnesanvändning

$\mathcal{O}(|V|^2)$

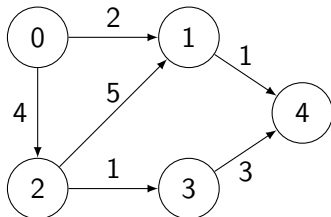
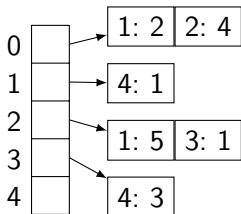
Finns det en båge från x till y ?

$\mathcal{O}(1)$

Vilka bågar finns från x ?

$\mathcal{O}(|V|)$

Grannlista (adjacency list)



Minnesanvändning

$$\mathcal{O}(|V| + |E|)$$

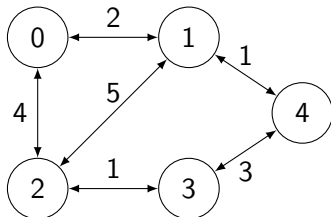
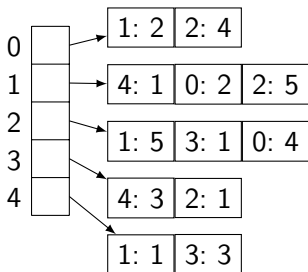
Finns det en båge från x till y ?

$$\mathcal{O}(|E|)$$

Vilka bågar finns från x ?

$$\mathcal{O}(|E|)$$

Grannlista (adjacency list)



Minnesanvändning

$$\mathcal{O}(|V| + |E|)$$

Finns det en båge från x till y ?

$$\mathcal{O}(|E|)$$

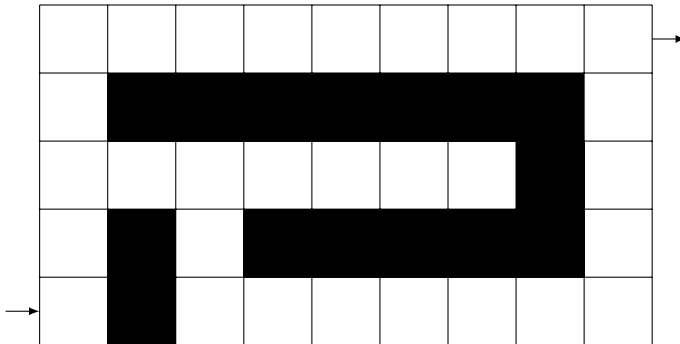
Vilka bågar finns från x ?

$$\mathcal{O}(|E|)$$

- 1 Grafer
- 2 Graftsökning i oviktade grafer
- 3 Graftsökning i viktade grafer
- 4 Graftsökning applicerat på andra problem
- 5 Sammanfattning

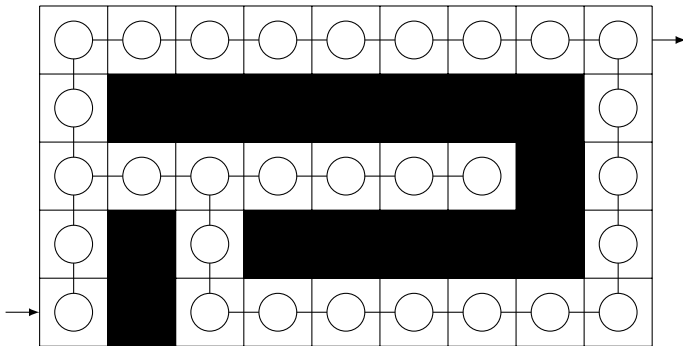
Problem

Hitta en väg genom labyrinten:



Problem

Hitta en väg genom labyrinten:



Grafdefinition

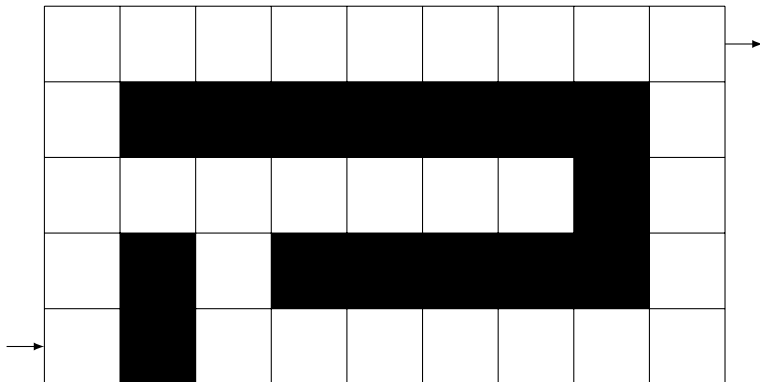
```
struct Node {  
    vector<int> edges;  
    bool visited = false;  
    int previous;  
};  
  
vector<Node> graph;
```


Djupet först (DFS)

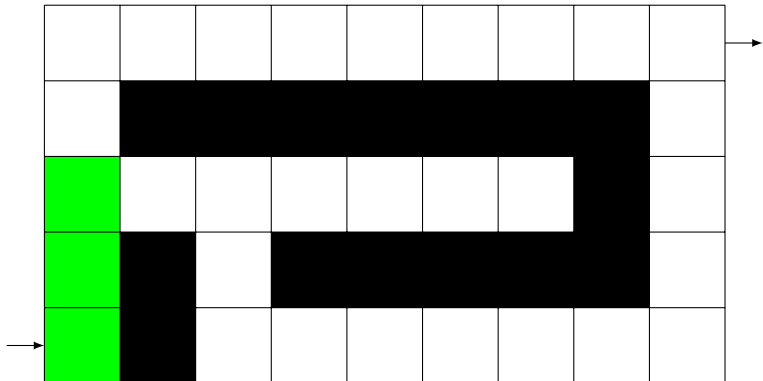
```
bool dfs(int from, int to) {
    if (from == to)
        return true;

    for (int x : graph[from].edges) {
        if (!graph[x].visited) {
            graph[x].visited = true;
            if (dfs(x, to))
                return true;
        }
    }
    return false;
}
```

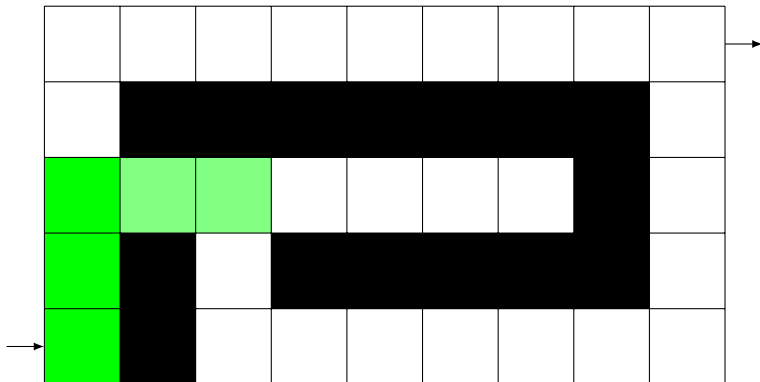
Djupet först (DFS)



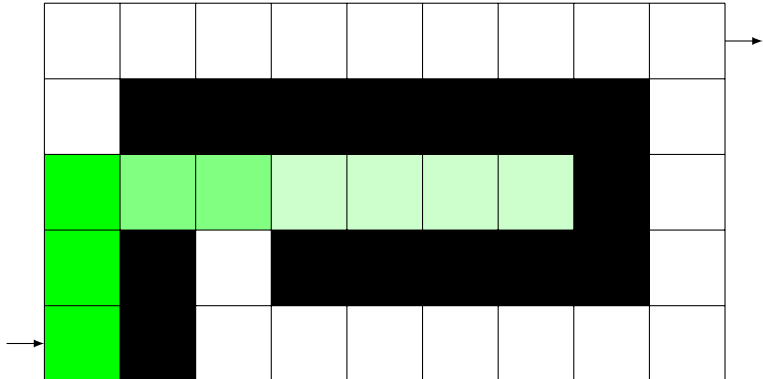
Djupet först (DFS)



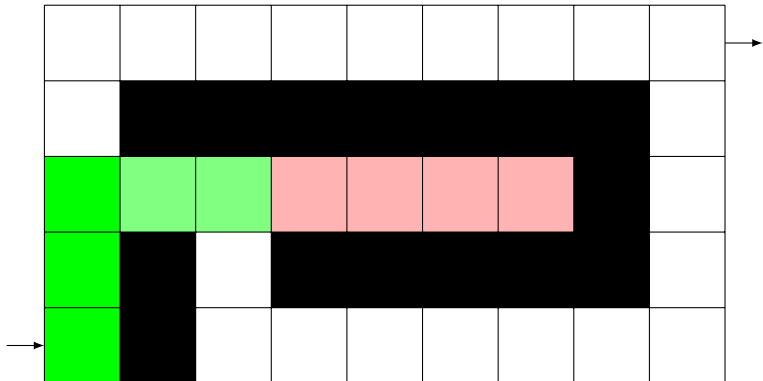
Djupet först (DFS)



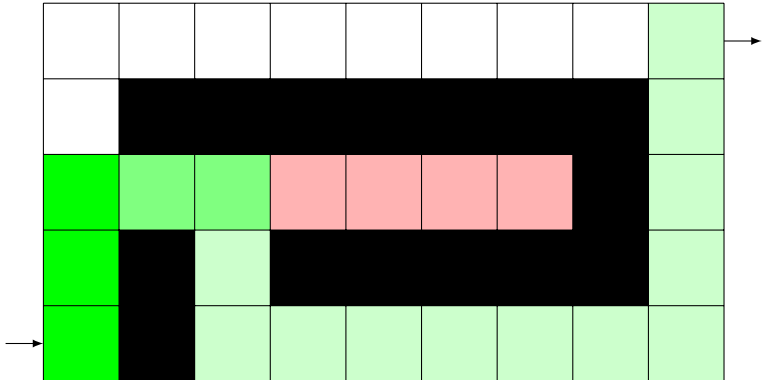
Djupet först (DFS)



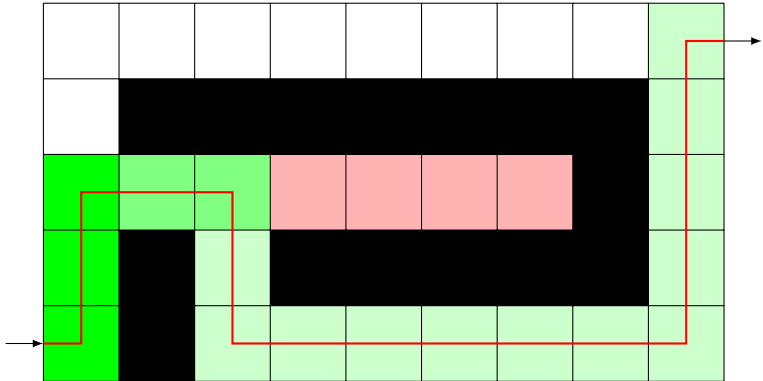
Djupet först (DFS)



Djupet först (DFS)



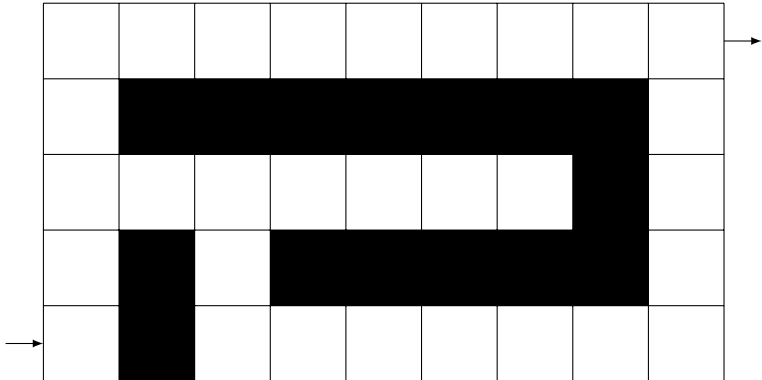
Djupet först (DFS)



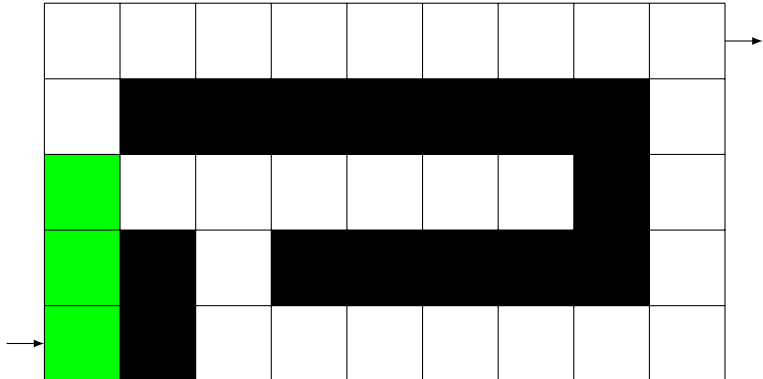
Bredden först (BFS)

```
bool bfs(int from, int to) {
    queue<int> q; q.push(from);
    graph[from].visited = true;
    while (!q.empty()) {
        int current = q.top(); q.pop();
        for (int x : graph[current].edges) {
            if (x == to) return true;
            // om x inte redan är besökt, markera
            // den som besökt och lägg på kö
        }
    }
    return false;
}
```

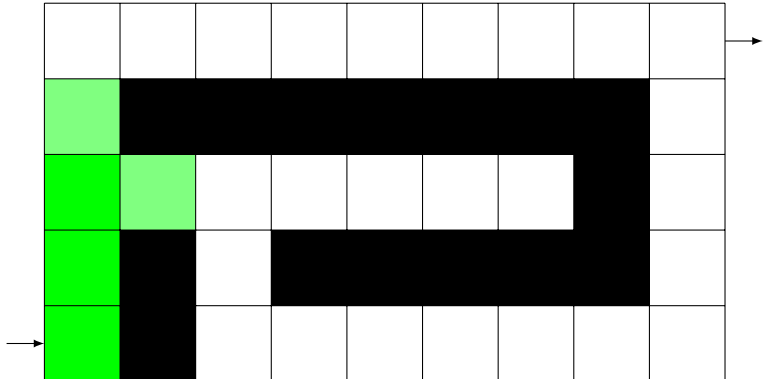
Bredden först (BFS)



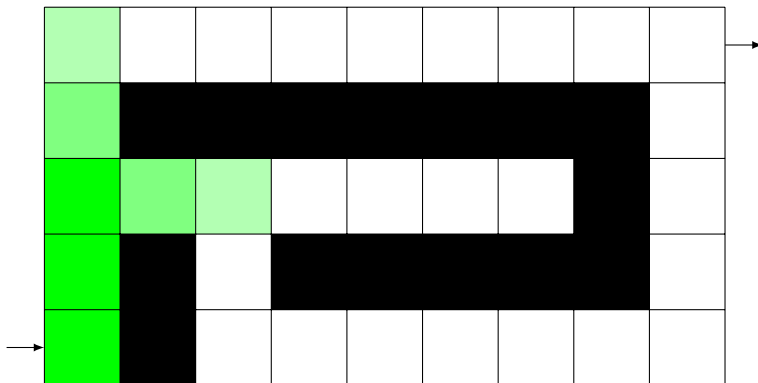
Bredden först (BFS)



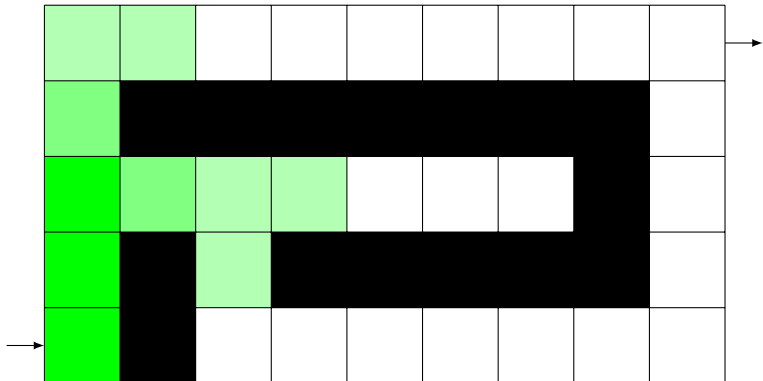
Bredden först (BFS)



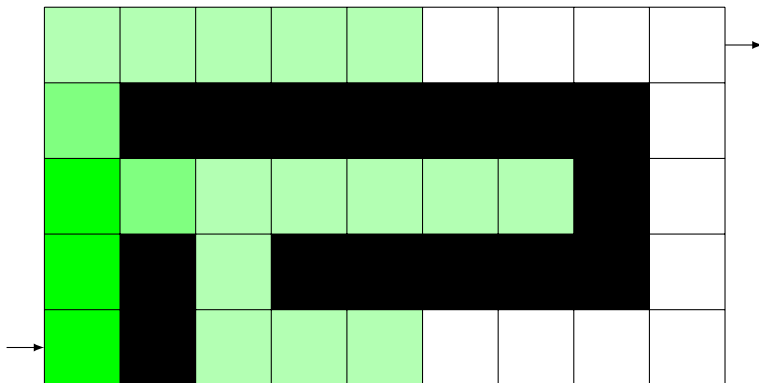
Bredden först (BFS)



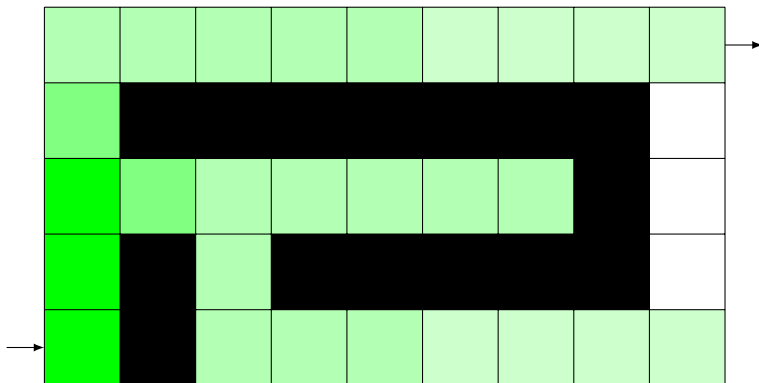
Bredden först (BFS)



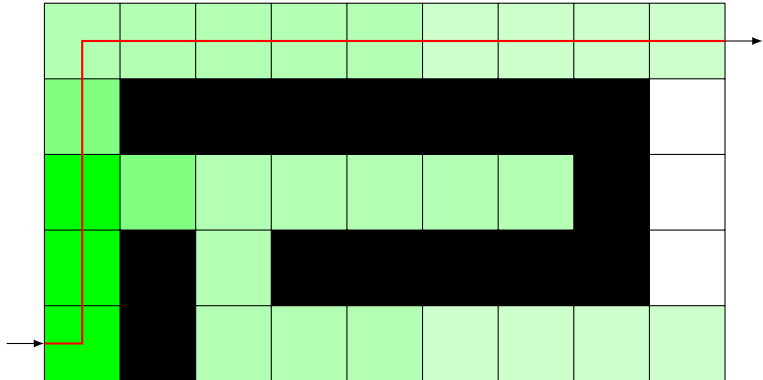
Bredden först (BFS)



Bredden först (BFS)



Bredden först (BFS)



Implicit graf

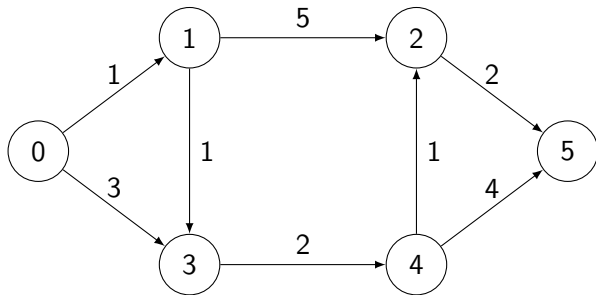
```
class Point {  
public:  
    int x, y;  
    // ...  
};  
  
bool map[width][height];
```

Implicit BFS

```
bool bfs(Point from, Point to) {
    queue<Point> q; q.push(from);
    bool visited[width][height] = false;
    visited[from.x][from.y] = true;
    while (!q.empty()) {
        Point current = q.top(); q.pop();
        for (Point x : current.neighbors()) {
            if (c == current) return true;
            // Om 'x' inte är besökt, lägg den på kön.
        }
    }
    return false;
}
```

- 1 Grafer
- 2 Grafsökning i oviktade grafer
- 3 **Grafsökning i viktade grafer**
- 4 Grafsökning applicerat på andra problem
- 5 Sammanfattning

BFS i en viktad graf



Vilken väg väljs?

Inte så bra...

- Eftersom vi använder en kö antar BFS att alla bågar har samma vikt...
- Alltså ger den lösningen som traverserar minst antal noder

Idé:

- Vad händer om vi i stället hela tiden väljer den nod i kön som representerar kortast sträcka?
- Dijkstras algoritm

Dijkstra

```
bool dijkstra(int from, int to) {
    priority_queue<...> q; q.push(from);
    while (!q.empty()) {
        int current = q.top(); q.pop();
        if (graph[current].visited) continue;
        graph[current].visited = true;
        if (current == to) return true;
        for (int x : graph[current].edges) {
            // Uppdatera x och lägg på kön om bättre
        }
    }
    return false;
}
```

Dijkstra – tidskomplexitet

- $\mathcal{O}((|E| + |V|) \log(|V|))$
- Observation:
För sammanhängande grafer gäller: $|V| \leq |E| \leq |V|^2$
- Alltså: $\mathcal{O}(|V|^2 \log(|V|))$, $\mathcal{O}(|E| \log(|V|))$ är också OK

- Notera: $\mathcal{O}(|E| + |V| \log |V|)$ är om bättre heap används

Dijkstra – egenskaper

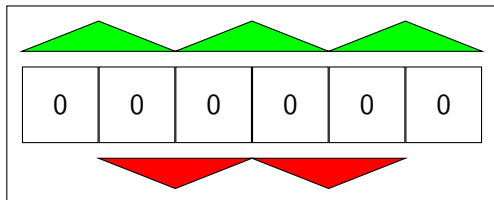
Hittar kortaste vägen i en *viktad graf*

- Tid: $\mathcal{O}((|E| + |V|) \log(|V|))$ (med heap)
- BFS är ofta bättre på oviktade grafer
- Kan optimeras med hjälp av heuristik: A^*
- Förutsätter att det inte finns några **negativa cykler**
Bellman-Ford klarar negativa cykler, men tar $\mathcal{O}(|E||V|)$
- Att hitta kortaste vägen mellan alla par av noder görs snabbare med *Floyd-Warshall*, som tar $\mathcal{O}(|V|^3)$ jämfört med $\mathcal{O}(|V|^3 \log(|V|))$

- 1 Grafer
- 2 Graftsökning i oviktade grafer
- 3 Graftsökning i viktade grafer
- 4 **Graftsökning applicerat på andra problem**
- 5 Sammanfattning

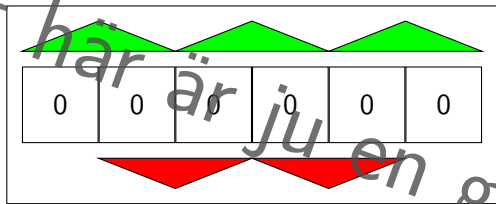
Problem

Du har hittat ett gammalt kodlås som du vill låsa upp. Du vet att koden är 568919, och just nu visar siffrorna på låset 000000. Låset har knappar för att öka och minska de olika siffrorna, dock fungerar de inte riktigt som de ska. Hur skriver du in koden med så få knapptryck som möjligt? Vilka koder går att skriva?



Problem

Du har hittat ett gammalt kodlås som du vill låsa upp. Du vet att koden är 568919, och just nu visar siffrorna på låset 000000. Låset har knappar för att öka och minska de olika siffrorna, dock fungerar de inte riktigt som de ska. Hur skriver du in koden med så få knapptryck som möjligt? Vilka koder går att skriva?



Lösningssidé

Representera problemet som en riktad graf:

- En nod för varje kombination
- Varje båge motsvarar ett knapptryck

Lösningssidé

Representera problemet som en riktad graf:

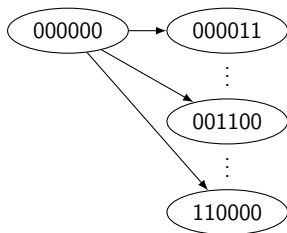
- En nod för varje kombination
- Varje båge motsvarar ett knapptryck

- Oviktad graf \Rightarrow BFS
- Varje väg motsvarar en lösning!

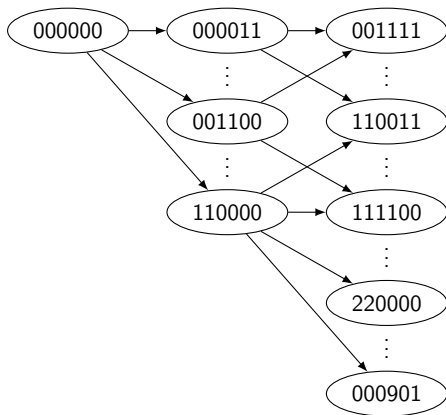
Lösningssidé

000000

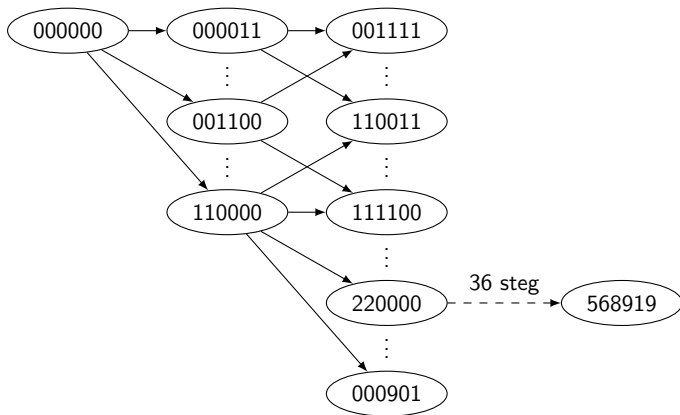
Lösningssidé



Lösningssidé



Lösningssidé



- 1 Grafer
- 2 Graftsökning i oviktade grafer
- 3 Graftsökning i viktade grafer
- 4 Graftsökning applicerat på andra problem
- 5 **Sammanfattning**

I kursen framöver

- Denna veckan
 - Se till att börja på lab 3 under veckan
- Nästa föreläsning
 - Fler grafalgoritmer, uppspännande träd etc.
- Uppgifter i Kattis
 - `hidingplaces` (enkel)
Var är det bäst att gömma sig från en springare på ett schackbräde?
 - `coast` (svårare)
Hur lång är kustlinjen på en given karta?

Filip Strömbäck

www.liu.se