Lecture 22

Graphs and graph search

TDDD86: DALP

Print version of the lecture *Data structures, algorithms and programming paradigms*
26 November 2016

Jalil Boudjadar, Tommy Färnqvist. IDA, Linköping University

22.1

Content

Innehåll

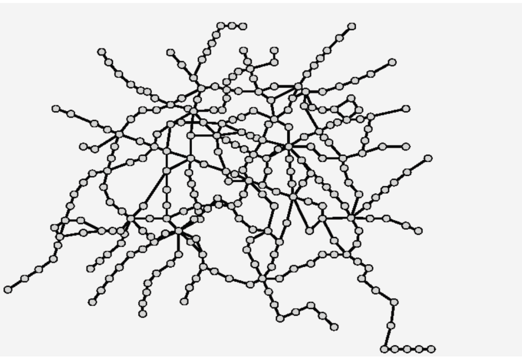
22.2

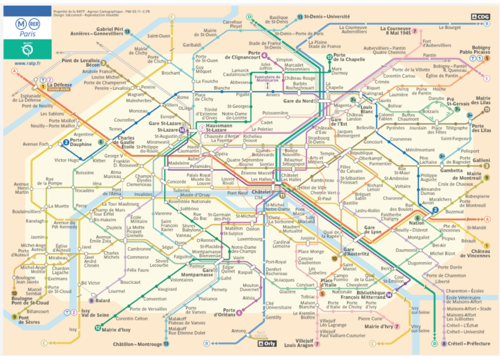
1 Graphs

1.1 Introduction

Definition

- A graph is a pair (V,E), where
  - V is a set of nodes (or vertices)
  - E is a set of pairs of nodes called arcs (or edges)
  - Nodes and arcs are positions and can store elements



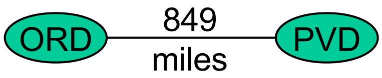


22.3

Arc types

- Directed edge
  - ordered pair of nodes (u,v)
  - u is the start node, v is the destination node
- Undirected edge
  - unordered pair of nodes $\{u,v\}$

- In a directed graph, all arcs are directed
- In an undirected graph, all arcs are undirected



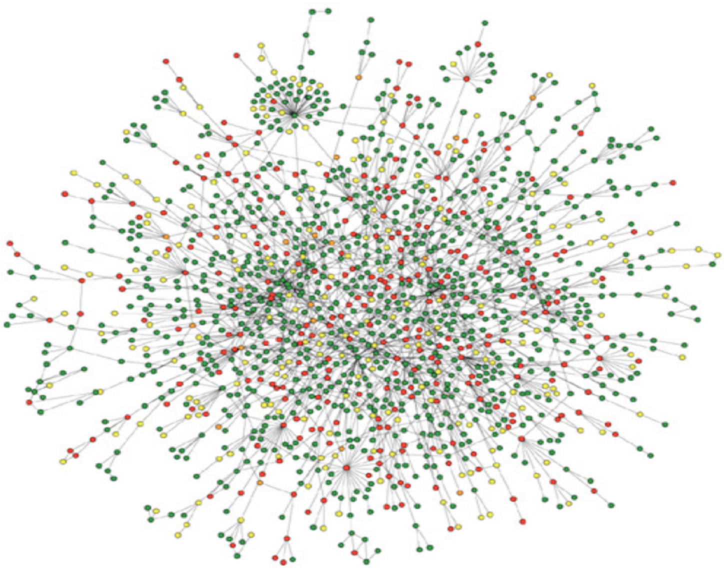
22.4

Why we need to study graph algorithms?

- Thousands of practical applications
- Hundreds known graphing algorithms
- Interestingly abstraction with great applicability
- Branch of computer science and discrete mathematics with many challenges

22.5

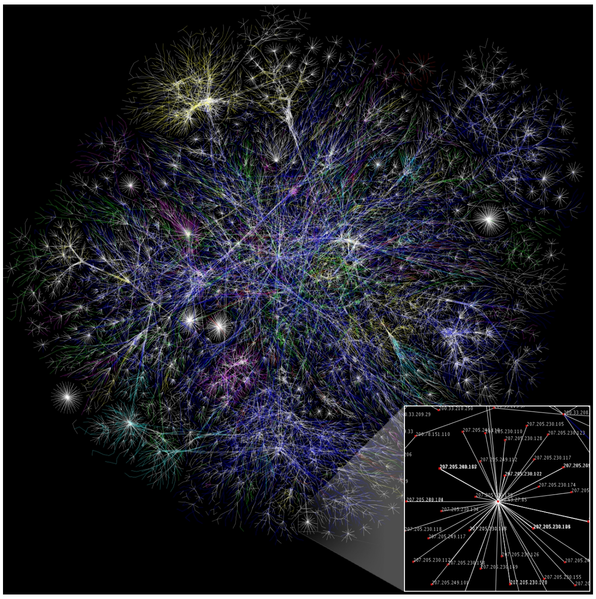
Protein-protein interaction network



Jeong et al, Nature Review | Genetics

22.6

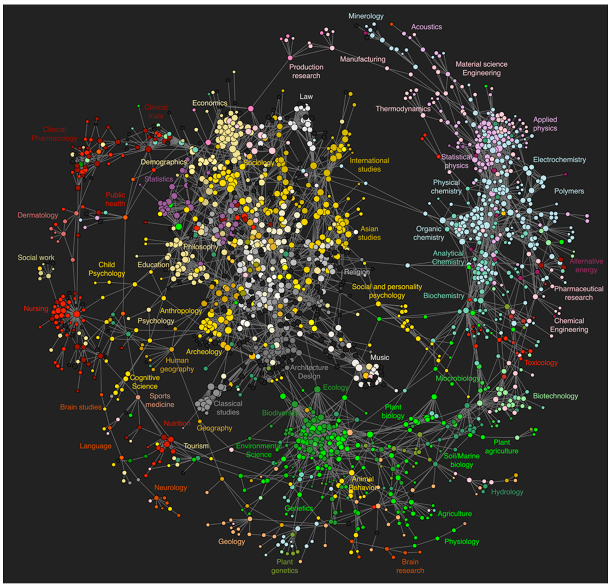
Internet charted by the Opte project



Opte Project

22.7

Scientific stream



http://www.plosone.org/article/info:doi/10.1371/journal.pone.0004803

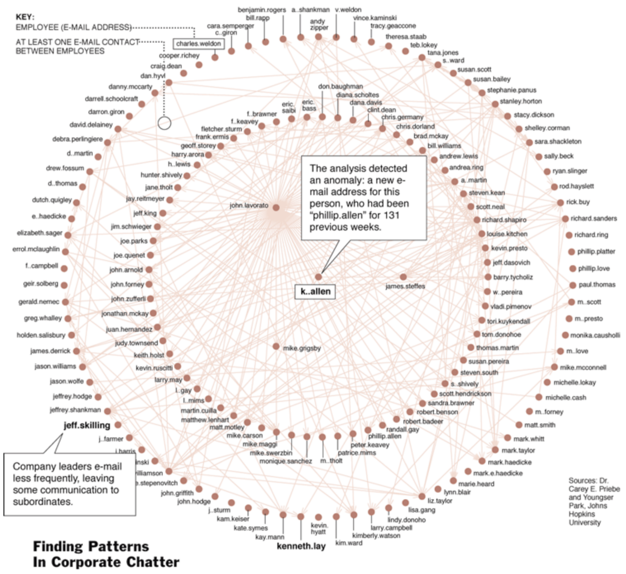
22.8

10 milions Facebook-friends

“Visualizing Friendship” av Paul Butler

22.9

A week’s email within Enron



22.10

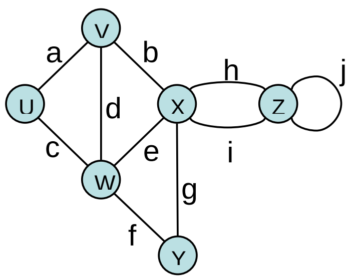
Applications

graph	node	edge
communication	phones, computers	fiber optic cable
circuit	gates, registers, processor	clutch
Financial	Stock, currency	transaction
transport	street intersection, airport	road, air route
Internet	networks	connection
social networking	persons, actors	friendship, relationship
neural network	neuron	synapse
chemical composition	molecular	binding

22.11

Terminology

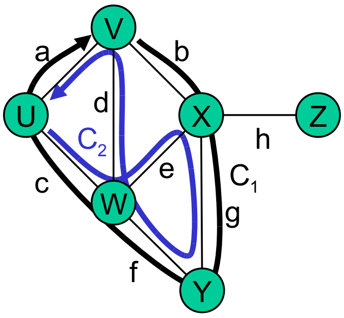
- An edge has endpoints (a has ends U and V)
- Edges ending in a node n are said to be incident (a, d and b are incidents to V)
- Nodes can be adjacent (U and V are adjacent)
- Each node has a degree (X has degree 5)
- Parallel edges (h and i are parallel edges)
- Loops (j is a loop)



Sv. *ändpunkter, incidenta, grannar, grad, parallella, öglor* 22.12

More terminology

- En cycle is a circular sequence of alternating nodes and edges. Each edge is preceded and followed by its endpoints.
- En simple cycle is a cycle such that all of its nodes and arcs are distinct.
- $C_1 = (V, b, X, g, Y, f, W, c, U, a, V)$ is a simple cycle.
- $C_2 = (U, c, W, e, X, g, Y, f, W, d, V, a, U)$ is *not* a simple cycle.



22.13

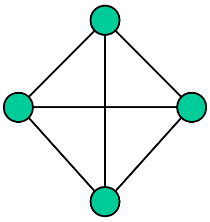
Characteristics

property 1
$\sum_v \deg(v) = 2m$ Proof: Each arc counted twice

property 2
In an undirected graph without loops and parallel arcs, $m \leq n(n-1)/2$ Proof: each node has max degree $(n-1)$

Notation

- n the number of nodes
- m the number of arcs
- $\deg(v)$ is the degree of node v



Exempel
$n = 4$
$m = 6$
$\deg(v) = 3$

22.14

Some algorithmic graph problems

- path. is there a path between s and t?
- shortest route. what is the shortest path between s and t?

- Cycle. is there a cycle in the graph?
- Eulertour. Is there a cycle that uses each arc exactly one time?
- Hamiltoncykel. Is there a cycle that uses each node exactly one time?

- Connectivity. Is there a connection between all nodes?
- MST. What is the best way to bind all nodes together? (Minimum Spanning Tree)
- Bi-connectivity. Is there a node that makes the graph not linked if it is removed?

- Planarity. Is it possible to draw the graph without any arcs intersect?
- Graph-isomorphism. Are two graphs identical apart from the names of the nodes?

Challenge. Which of the above problems is simple? Difficult? Impossible to solve effectively? 22.15

1.2 ADT graph

Main methods of undirected graphs

- Nodes and arcs
  - are positions
  - store elements
- access methods
  - endVertices(e): an array with the 2 endpoints of e
  - opposite(v, e): the opposite node v along e
  - areAdjacent(v, w): **true** iff v and w are adjacent
  - replace(v, x): replaces the element in node v with x
  - replace(e, x): replaces the element in edge e with x

22.16

Main methods of undirected graphs

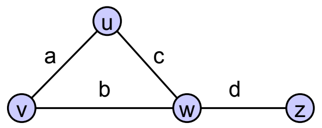
- Update methods
  - insertVertex(o): inserts a node that stores the element o
  - insertEdge(v, w, o): insert an edge (v, w) that storers the element o
  - removeVertex(v): removes node nod v (and its incident edges)
  - removeEdge(e): removes edge e
- Iterator methods
  - incidentEdges(v): the edges incident to v
  - vertices$()$: all nodes in the graph
  - edges$()$: all edges in the graph

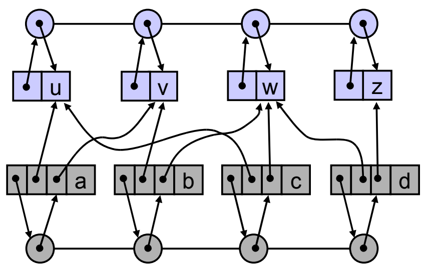
22.17

1.3 Data structures

Edge lists

- A sequence of nodes is a sequence of positions for the node objects
- A sequence of edges is a sequence of positions for the edge objects
- Node objects store elements and references to positions in the sequence of nodes
- Edge objects store elements, object for startnode, object for endnode and reference to position in the sequence of edges

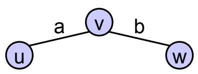


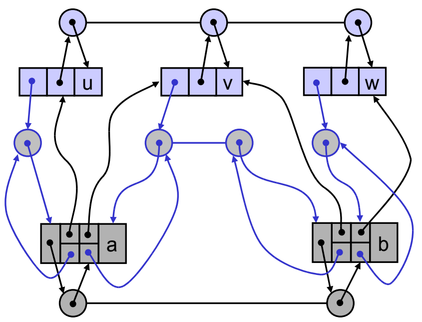


22.18

Adjacency list

- Add extra structure to edge list
- Each node has a sequence of its incident arcs with reference to the arc objects of incident edges
- Arc object extended with references to the associated positions in the incidence sequence of its end-points

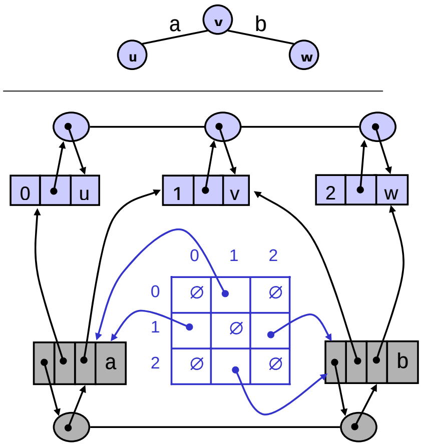




22.19

Adjacency matrix

- Add extra structure to the edge list
- Node objects are extended with integer keys (index) associated with the nodes
- 2-dimensions adjacency array
  - Reference to edge objects for nodes that are adjacent
  - **null** for nodes that are not adjacent



22.20

Asymptotic performance

n noder, ***m*** bågar inga parallella kanter inga öglor	Båglista	Grannlista	Grann-matris
minne	$O(\boldsymbol{n} + \boldsymbol{m})$	$O(\boldsymbol{n} + \boldsymbol{m})$	$O(\boldsymbol{n}^2)$
incidentEdges($\boldsymbol{v}$)	$O(\boldsymbol{m})$	$O(\deg(\boldsymbol{v}))$	$O(\boldsymbol{n})$
areAdjacent ($\boldsymbol{v}, \boldsymbol{w}$)	$O(\boldsymbol{m})$	$O(\min(\deg(\boldsymbol{v}),\deg(\boldsymbol{w})))$	$O(1)$
insertVertex($\boldsymbol{o}$)	$O(1)$	$O(1)$	$O(\boldsymbol{n}^2)$
insertEdge($\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{o}$)	$O(1)$	$O(1)$	$O(1)$
removeVertex($\boldsymbol{v}$)	$O(\boldsymbol{m})$	$O(\deg(\boldsymbol{v}))$	$O(\boldsymbol{n}^2)$
removeEdge($\boldsymbol{e}$)	$O(1)$	$O(1)$	$O(1)$

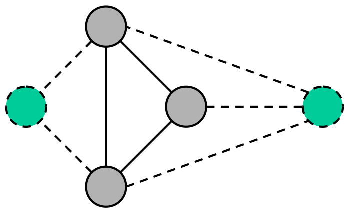
22.21

2 Search in undirected graphs

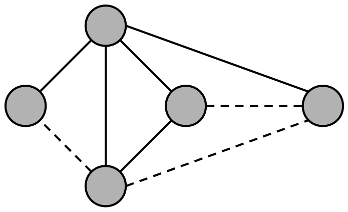
2.1 DFS

Subgraphs

- A subgraph S of a graf G is a graph such that
  - Nodes in S are a subset of nodes in G
  - Edges in S are a subset of edges in G
- A spanning subgraf of G is a subgraph that contains all nodes of G



Delgraf



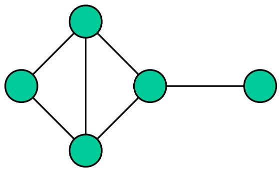
Spännande delgraf

Sv. *delgraf, spännande delgraf*

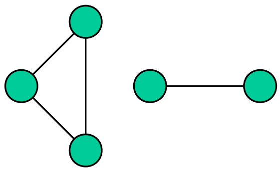
22.22

Connectivity

- A graph is connected if there is a path between each pair of nodes
- A connected component in a graph G is a maximal connected subgraph of G



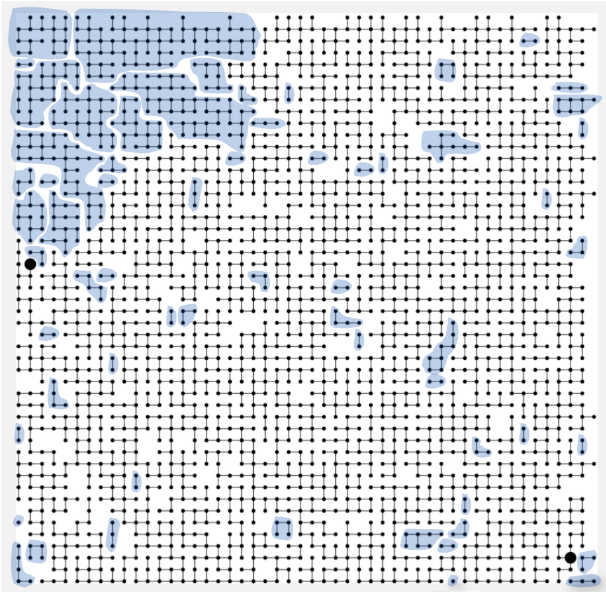
Sammanhängande graf



Ej sammanhängande graf med två sammanhängande komponenter

22.23

Connected components



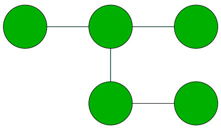
63 sammanhängande komponenter

22.24

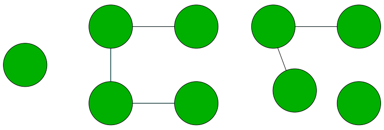
Trees and forests

- A (free) tree is an undirected graph T such that
  - T is connected
  - T does not contain cycles
  - This definition of the tree is different from the rooted tree
- A forest is an undirected graph without cycles
- The connected components in a forest are trees

22.25



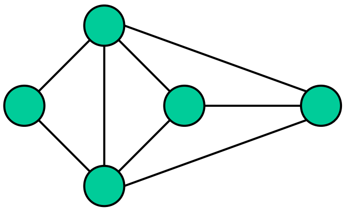
Träd



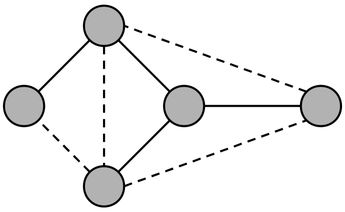
Skog

Spanning trees and forests

- A spanning tree of a connected graph is an spanning subgraph which is a tree
- A spanning tree is not unique if the original graph is a tree
- Spanning trees have applications in the design of communication networks
- A spanning forest of a graph is a spanning subgraph which is a forest



Graf



Spännande träd

22.26

Depth first search

- Depth first search (DFS) is a general technique for traversing a graph. DFS visits the child vertices before visiting the sibling vertices; that is, it traverses the depth of any particular path before exploring its breadth
- DFS in a graph G
  - visits all nodes and arcs G
  - Determines if G is connected
  - Calculates the number of connected components in G
  - Calculates a spanning forest to G
- DFS on a graph with n nodes and m edges takes $O(n+m)$ time
- DFS can be extended to solve other graph problems
  - Find and describe a path between two given nodes in a graph
  - Find a cycle in a graph

22.27

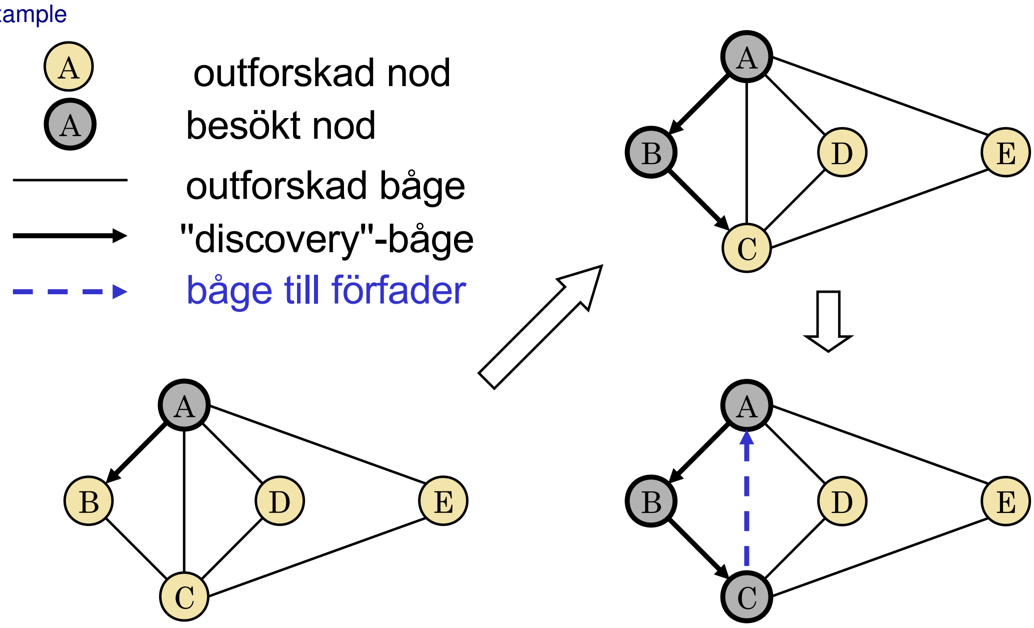
Algorithms for DFS

procedure DFS(G)
  for all $u \in G$.VERTICES() **do**
    SETLABEL(u, $UNEXPLORED$)
  for all $e \in G$.EDGES() **do**
    SETLABEL(e, $UNEXPLORED$)
  for all $v \in G$.VERTICES() **do**
    if GETLABEL(v) $=$ $UNEXPLORED$ **then**
      DFS(G, v)

procedure DFS(G, v)
  SETLABEL(v, $VISITED$)
  for all $e \in G$.INCIDENTEDGES(v) **do**
    if GETLABEL(e) $=$ $UNEXPLORED$ **then**
      $w \leftarrow$ OPPOSITE(v, e)
      if GETLABEL(w) $=$ $UNEXPLORED$ **then**
        SETLABEL(e, $DISCOVERY$)
        DFS(G, w)
      else
        SETLABEL(e, $BACK$)

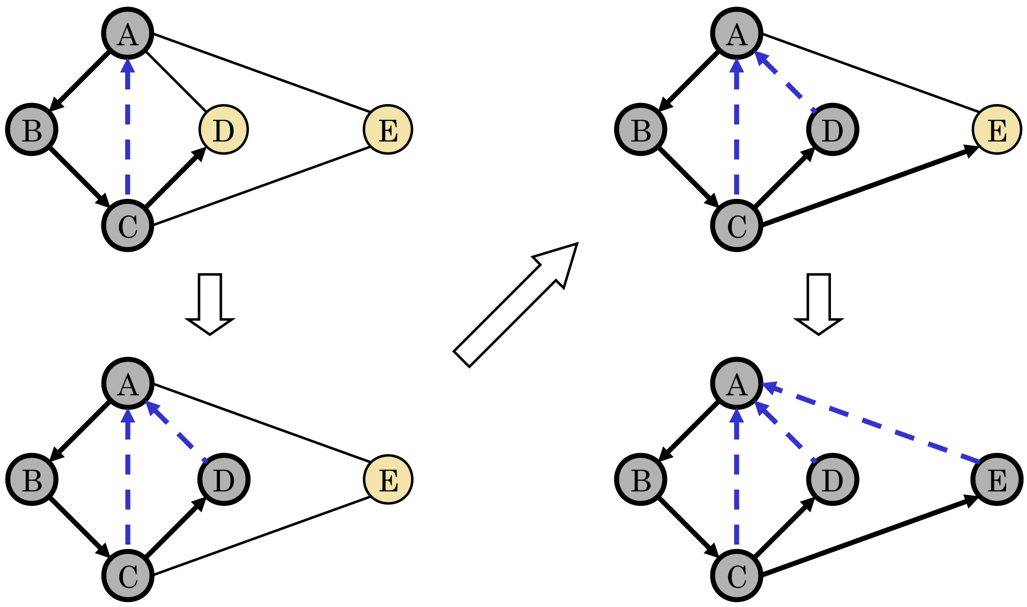
22.28

Example



22.29

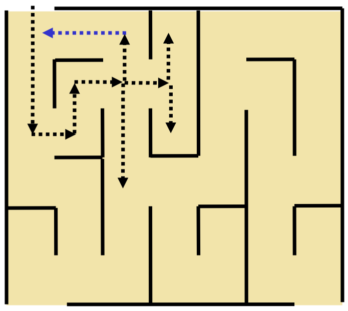
Example



22.30

DFS and labyrinth exploration

- The algorithm for DFS resembles a classic strategy for exploring labyrinths
  - We mark every intersection, corners and dead end (node) we visit
  - We mark every corridor (edge) we go through
  - We keep track of the way back to the entrance (start node) using a recursion stack

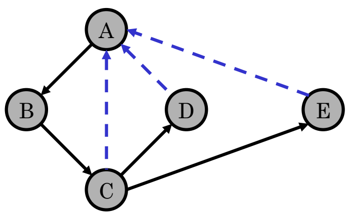


22.31

Characteristics

Property 1
DFS(G, v) visits all nodes and edges in the connected portion of G which v is included in

Property 2
”discovery”-edges DFS(G, v) constitutes a spanning tree to the connected component of G which v is included in



22.32

Analysis of DFS

- Mark/retrieve the marking of a node/edge takes $O(1)$ time
- Each node is marked twice
  - one time as *UNEXPLORED*
  - one time as *VISITED*
- Each edge is marked twice

– one time as *UNEXPLORED*

– one time as *DISCOVERY* or *BACK*

- Method incidentEdges is called once for each node
- DFS runs in time $O(n+m)$ given that the graph is represented by a adjacency list
  - Remember $\sum_v deg(v) = 2m$

22.33

Find paths

- We can specialize the DFS-algorithm to find a path between 2 given nodes v and z
- We call $\text{DFS}(G, v)$ with v as the start node
- We use a stack S to keep track of the way from the start node to the current node
- As soon as we encounter the target node z, we will return the contents of the stack as the target path

```
procedure PATHDFS(G, v, z)
    SETLABEL(v, VISITED)
    S.PUSH(v)
    if v = z then
        print the element in S
        return
    for all e ∈ G.INCIDENTEDGES(v) do
        if GETLABEL(e) = UNEXPLORED then
            w ← OPPOSITE(v, e)
            if GETLABEL(w) = UNEXPLORED then
                SETLABEL(e, DISCOVERY)
                S.PUSH(e)
                PATHDFS(G, w, z)
                S.POP() // e
            else
                SETLABEL(e, BACK)
    S.POP() // v
```

22.34

Find cycles

- We can specialize the DFS-algorithm to find a cycle
- We use a stack S to keep track of the way from the start node to the actual node
- As soon as we encounter an edge (v, w) that leads to an ancestor we return the cycle contained in the stack from the top to the node w

```
procedure CYCLEDFS(G, v, z)
    SETLABEL(v, VISITED)
    S.PUSH(v)
    for all e ∈ G.INCIDENTEDGES(v) do
        if GETLABEL(e) = UNEXPLORED then
            w ← OPPOSITE(v, e)
            S.PUSH(e)
            if GETLABEL(w) = UNEXPLORED then
                SETLABEL(e, DISCOVERY)
                CYCLEDFS(G, w)
                S.POP() // e
            else // find cycle
                repeat
                    o ← S.POP()
                    print o
                until o = w
                return
    S.POP() // v
```

22.35

2.2 BFS

Breadth First Search

- Breadth First Search (BFS) is a general technique to traverse a graph. BFS visits the neighbor vertices before visiting the child vertices.
- BFS on a graph G

- visits all nodes and edges in G
- determines if G is connected
- calculates the number of connected components in G
- calculates a spanning forest of G

- BFS on a graph with n nodes and m edges takes $O(n+m)$ time
- BFS can be extended to solve other graph problems
  - Find and describe the shortest path between two given nodes in a graph
  - Find a simple cycle in a graph, if there is one

22.36

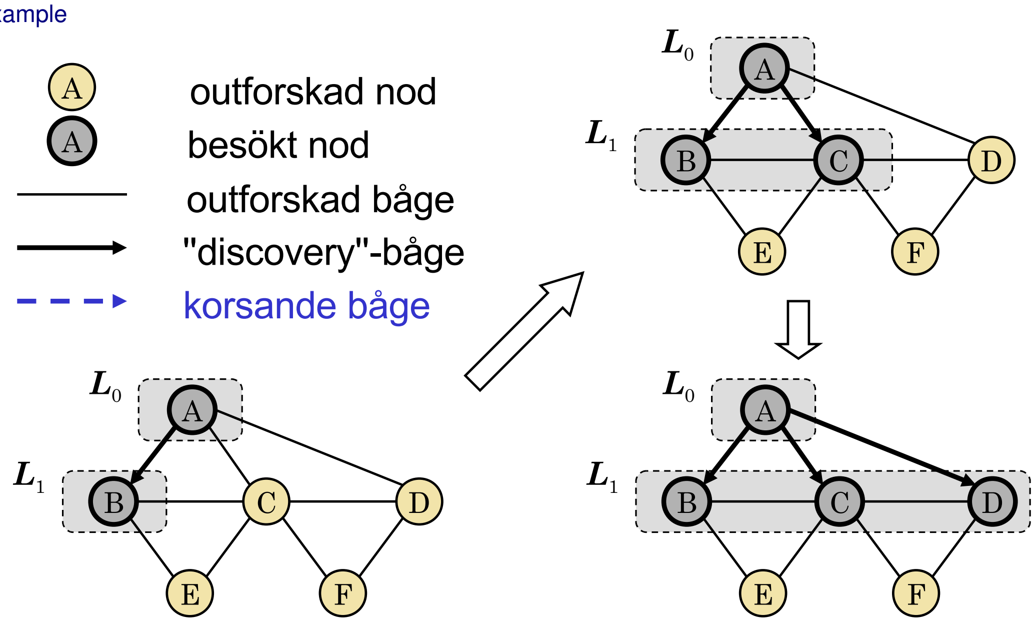
Algorithm for BFS

procedure BFS(G)
    mark all nodes/edges with *UNEXPLORED* as in DFS
    for all $v \in G$.VERTICES() **do**
        if GETLABEL(v) $=$ *UNEXPLORED* **then** BFS(G, v)

procedure BFS(G, s)
    $L_0 \leftarrow$ ny tom sekvens; L_0.INSERTLAST(s); SETLABEL(s, *VISITED*); $i \leftarrow 0$
    while $\neg L_i$.ISEMPTY() **do**
        $L_{i+1} \leftarrow$ ny tom sekvens
        for all $v \in L_i$.ELEMENTS() **do**
            for all $e \in G$.INCIDENTEDGES(v) **do**
                if GETLABEL(e) $=$ *UNEXPLORED* **then**
                    $w \leftarrow$OPPOSITE(v, e)
                    if GETLABEL(w) $=$ *UNEXPLORED* **then**
                        SETLABEL(e, *DISCOVERY*)
                        SETLABEL(w, *VISITED*)
                        L_{i+1}.INSERTLAST(w)
                    else
                        SETLABEL(e, *CROSS*)
        $i \leftarrow i+1$

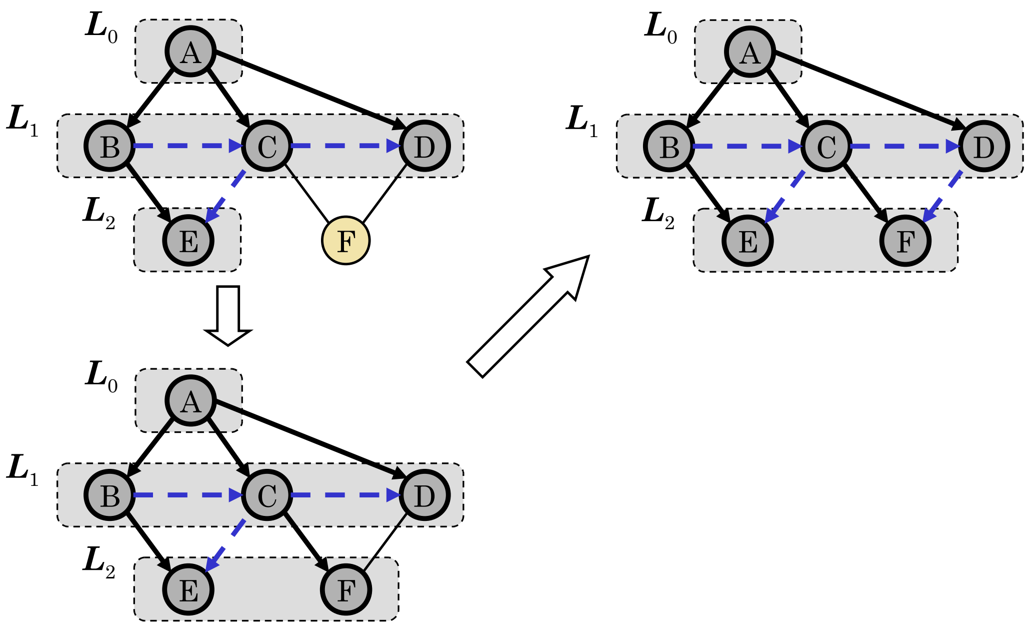
22.37

Example



22.38

Example



22.39

Characteristics

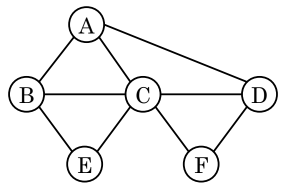
Let G_s denote the connected portion of G as s is included in

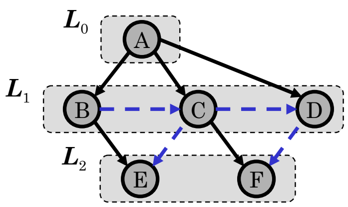
Property 1
BFS(G,s) visits all nodes and edges in G_s

Property 2
"discovery"-edges BFS(G,s) mark up represents a spanning tree T_s of G_s

Property 3
For each node v in L_i

- A path in T_s from s to v has i edges
- Each path from s to v in G_s has at least i edges





22.40

Analysis of BFS

- Mark/retrieve the marking of a node/edge takes $O(1)$ time
- Each node will be marked twice
  - one time as *UNEXPLORED*
  - one time as *VISITED*
- Each edge will be marked twice
  - one time as *UNEXPLORED*
  - one time as *DISCOVERY* or *CROSS*
- Each node is inserted once in a sequence L_i
- Method incidentEdges is called one time for each node
- BFS runs in time $O(n+m)$ given that the graph is represented with an adjacency list
  - Rememeber $\sum_v deg(v) = 2m$

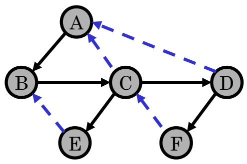
22.41

2.3 DFS vs BFS

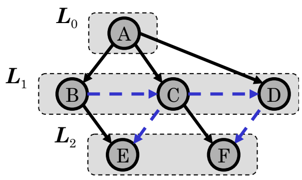
Applications

Tillämpningar	DFS	BFS
Spännande träd, sammanhängande komponenter, stigar, cykler	√	√
Kortaste stigar		√
2-sammanhängande komponenter	√	

DFS



BFS

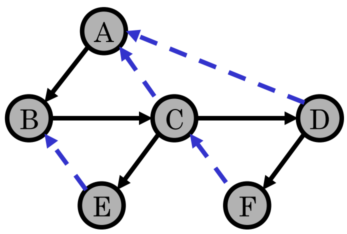


22.42

Edges leading to already visited nodes

edge to the ancestor

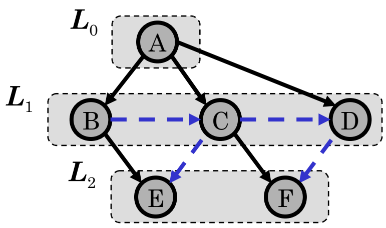
- w is an ancestor of v in the tree of ”discovery”-edges



DFS

shortest paths

- w is at the same level as v or in the next level in the tree of ”discovery”-edges



BFS

22.43