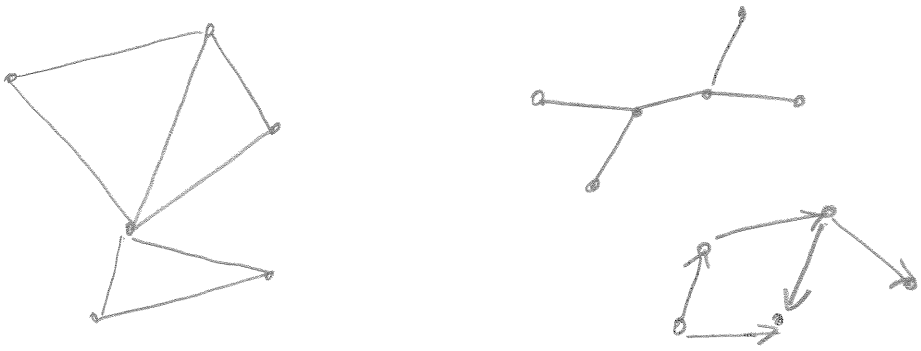
Graphs



consist of

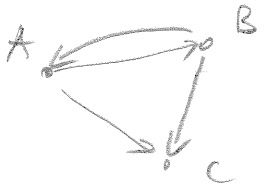
- vertices
- edges (undirected or directed)

Formally, $G = (V, E)$

V: set of vertices

E: set of edges

- directed edges: $E \subseteq \underbrace{V \times V}_{\text{pairs of vertices}}$



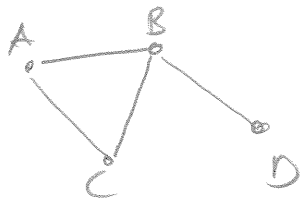
$V = \{A, B, C\}$

$E = \{(A,B), (A,C), (B,A), (B,C)\}$

Digraph!

- undirected edges : $E \subseteq \mathcal{P}_2(V)$

set of all 2-element subsets of V



$V = \{A, B, C, D\}$

$E = \{\{A,B\}, \{A,C\}, \{B,C\}, \{B,D\}\}$

Let $|V| = n$. What is the maximal number of

- directed edges (without loops)
- undirected edges

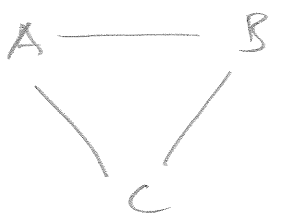
between nodes in V?

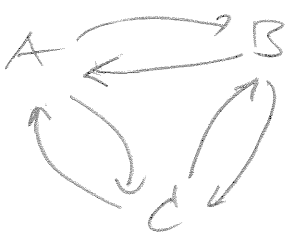
A: $n(n-1)$ $\quad$ $n(n-1)/2$

How many digraphs / graphs?

A: $2^{n(n-1)}$ $\quad$ $2^{n(n-1)/2}$

Model undirected edges by 2 directed edges

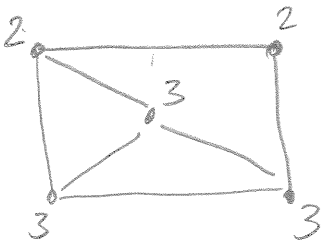




Terminology

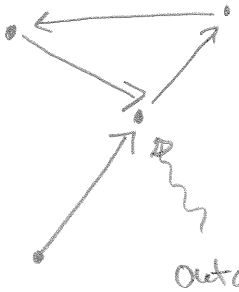
- $u \bullet \longrightarrow \bullet v$ — v is adjacent to u

  $(u, v) \in E$

- 

  $\deg(u)$ "degree of u"

  = # adj. vertices

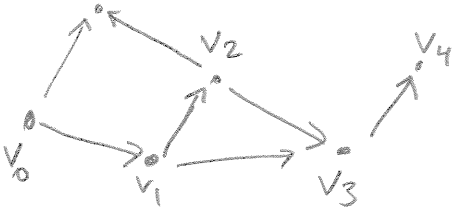
- 

  $\text{outdeg}(u)$ "outdegree"

  = # outgoing edges

  $\text{outdeg}(\cdot) = 1$
  $\text{indeg}(\cdot) = 2$

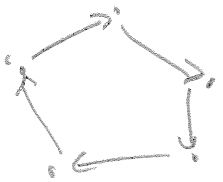
  $\text{indeg}(u)$
  = # incoming edges

- 

  $v_0\ v_1\ v_2\ v_3\ v_4$ is a path (m)

  simple path: $v_i \neq v_j$ for $i \neq j$

  cycle: $v_0 = v_m$, otherwise $v_i \neq v_j$ for $i \neq j$



- G undirected: G is <u>connected</u> if there is a path from u to v f.a. $u, v \in V$ (<u>Strongly connected</u> if digraph)

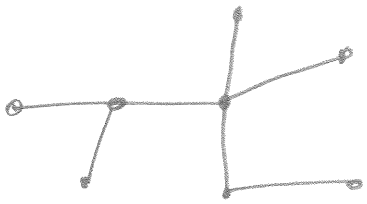
- $G' = (V', E')$ is a <u>subgraph</u> of $G = (V, E)$ if
  - $V' \subseteq V$, $E' \subseteq E$ (and G' is a graph)

- <u>Connected component</u> of G

maximal connected subgraph of G

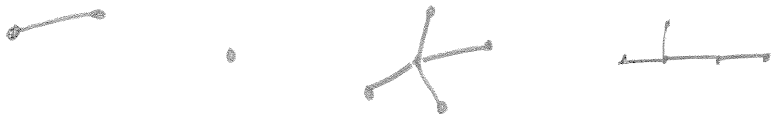
(<u>Strongly connected component</u> in digraph)

- Tree



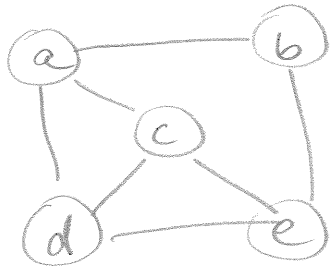
connected graph without cycles

- Forest

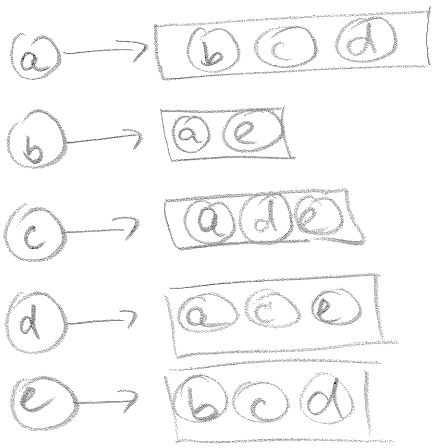


all connected components are trees

Representing Graphs



Adjacency list for each vertex



Space:

$$|V| + \sum_{v \in V} \deg(v)$$

$$= |V| + 2|E|$$

$$= \Theta(V + E)$$

(abuse of notation)

How change for digraphs?

Adjacency matrix

	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	0	1
c	1	0	0	1	1
d	1	0	1	0	1
e	0	1	1	1	0

$V = \{1, \ldots, n\}$,

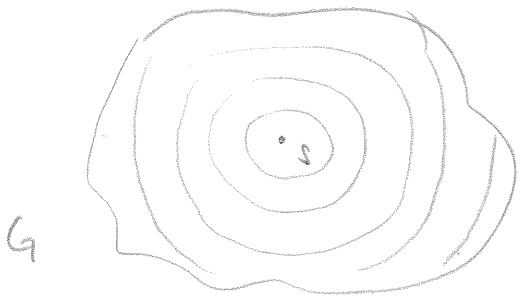
$G = (V, E)$

$$M(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

... and for digraphs?

Breadth-First Search

$G = (V, E)$ undirected, $s \in V$ start node



Idea: 1st round: nodes v with $dist(s,v) = 1$

2nd round: nodes v with $dist(s,v) = 2$

- - -

where

$dist(s,v)$ = length of shortest path from s to v

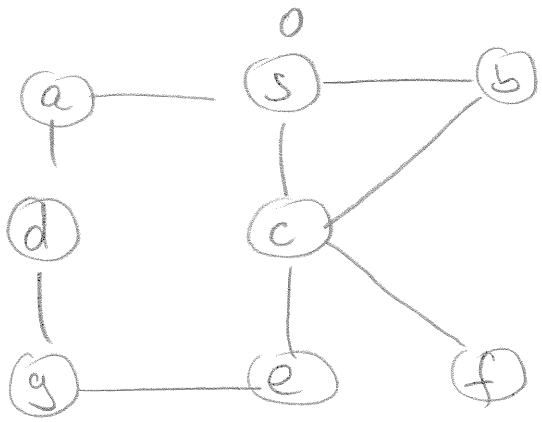
Technique:

- nodes can have colours white, grey, black
- nodes have adj. list
- use FIFO queue

Also

- predecessors ($\rightarrow$ spanning tree)
- $dist(s,v)$

Example:



Q: s

```
BFS(G, s)
    for u ∈ G.V                          } initialize
        u.col  = 'w'
        u.dist = ∞
        u.pred = nil
    s.col = 'g' ;  s.dist = 0
    Q.init()
    while not Q.isEmpty() do
        u := Q.dequeue()
        for v in u.adj do
            if v.col = 'w'
                then v.col := 'g' ;
                     v.dist := u.dist + 1
                     v.pred := u
                     Q.enqueue(v)

        u.col := 'b'
```

Breadth-First Search (BFS) — Summary.

- 3 colors for nodes
  - white: unexplored
  - gray: under exploration
  - black: finished
- undirected graph
- computes
  - distance from start node s: $dist(s,v)$
  - (some) spanning tree
- FIFO queue for nodes to explore

BFS Running Time

• Initialization	$\Theta(V)$
• Enqueue vertex if while	$\Theta(V)$
• Scan adj. list	$\Theta(E)$
total	$\Theta(V+E)$

Correctness of BFS

1) $v.dist = dist(s,v)$ f.a. v

2) pred defines a spanning tree

ad 1) Proof in [CLRS]

ad 2) The graph defined by "pred" is
- connected
- cycle-free (due to dist: $v.pred = u \Rightarrow v.dist = u.dist + 1$)

$\Rightarrow$ tree

Consequence

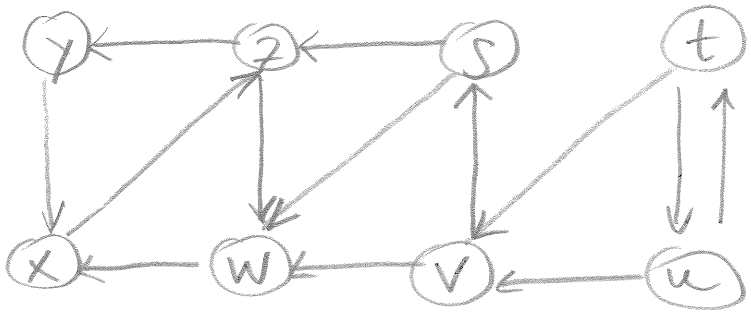
- order of exploration of adjacent nodes does not matter

Depth-First Search

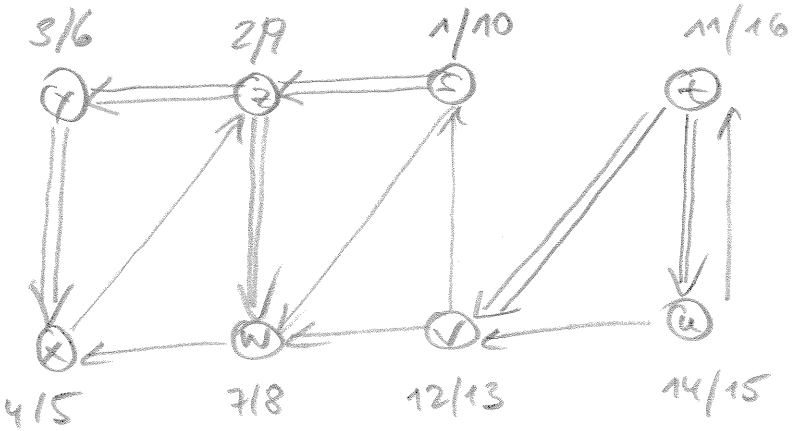
Technique

- recursive alg. ⇒ implicit stack
- also for disconnected graphs, starts from all nodes
- nodes have fields st (starttime), et (end time)
- for digraphs (typically)

DFS: Example Graph



Traversal



Edge types

- Tree edges: 'g' → 'w'
- Back edges: 'g' → 'g' (indicate cycles)
- Forward edges: 'g' → 'b', from ancestor to descendant
- Cross edges: 'g' → 'b', otherwise

Parenthesization

Starttime at node v: (v opening parenthesis

Endtime at node v: v) closing parenthesis

For our traversal

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
(s	(z	(y	(x	x)	y)	(w	w)	z)	s)	(t	(v	v)	(u	u)	t)

Parentheses indicate tree structure

Theorem: For every DFS traversal, start and end times have a parenthesis structure

DFS Code

```
DFS-All(G)
    for u ∈ G.V do                    } Initialization
        u.col := 'w'
        u.pred := nil
    time := 0
    for u ∈ G.V do
        if u.col = 'w' then DFS(u)

DFS(u)
    u.col := 'g'
    time ++
    u.starttime := time
    for v ∈ u.adj do
        if v.col = 'w' then
            v.pred := u
            DFS(v)
    u.col := 'b'
    time ++
    u.endtime := time
```

DFS Running Time

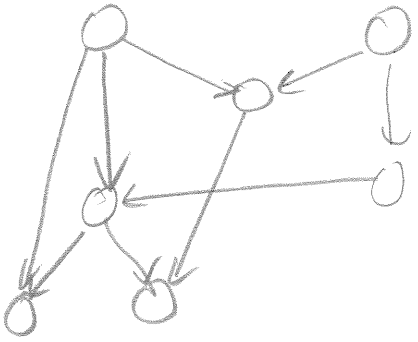
- DFS - All
  - initialization $\Theta(V)$
  - calls to DFS $\Theta(V)$

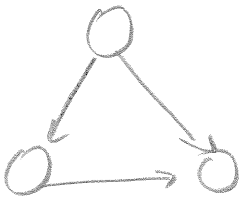
- DFS
  - called once per vertex $\Theta(V)$
  - explores all adjacent edges $\Theta(E)$

  total $\Theta(V+E)$

DAGs and Topological Sorting

DAG = Directed Acyclic Graph





Model precedences: (x) → (y) = "x before y"

DAGs and DFS

Theorem: G digraph. Then

G is a DAG ⇔ DFS yields no back edge ('g' → 'g')

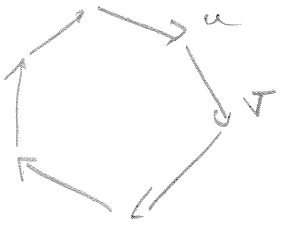
Proof: "⇒" (indirect)

Suppose $u \to v$ is back edge

v gray, u gray
⇒ path $v \dashrightarrow u$ (in DFS tree)
⇒ edge $u \to v$ completes cycle

"$\Leftarrow$" (indirect)

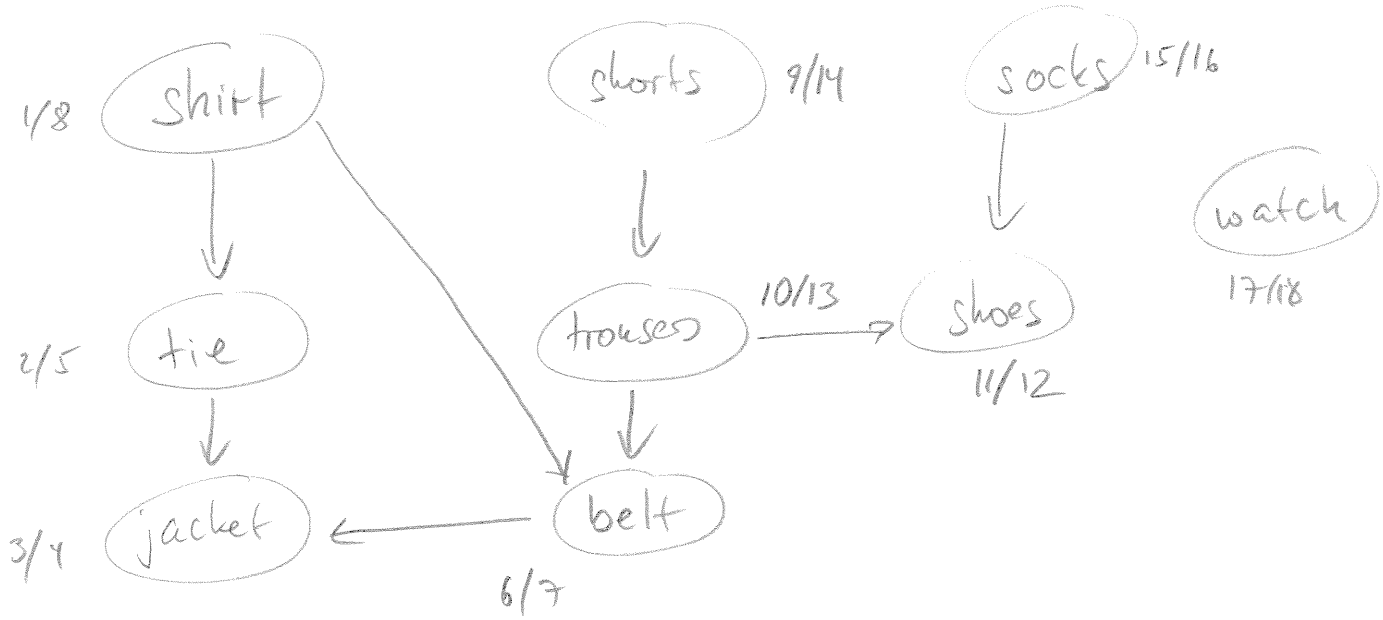
Suppose G has cycle C



- Suppose v is first node in C discovered
- let u be pred of v in C
- When DFS discovers v, all of C is white
- Before return of DFS(v), all nodes reachable on C are visited
  - $\Rightarrow$ u is visited after v
  - $\Rightarrow$ u is descendant of v
  - $\Rightarrow$ (u,v) is back edge

[Dressing Example]

Dressing Example



Topological Sorting

$G = (V, E)$ DAG.

Find linear ordering $v_1, v_2, \ldots, v_n$ of V

- sth. $(v_i, v_j) \in E \Rightarrow i < j$ (i.e., v_i before v_j)

Idea: If $(u,v) \in G$, then v is finished before u

G DAG $\Rightarrow$ (u,v) is no back edge

$\Rightarrow$ 2 cases when (u,v) is explored

- v is black $\Rightarrow$ ✓
- v is white $\Rightarrow$ v descendant of u in DFS-tree $\Rightarrow$ ✓

Topological Sorting: Algorithm

- Run DFS
- sort nodes according to finishing time in descending order