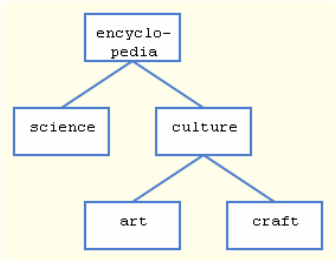
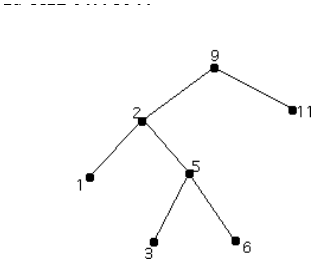
Trees, Glorious Trees

There all kinds of trees:
apple trees, palm trees, and so on...



...and also these binary trees and such:



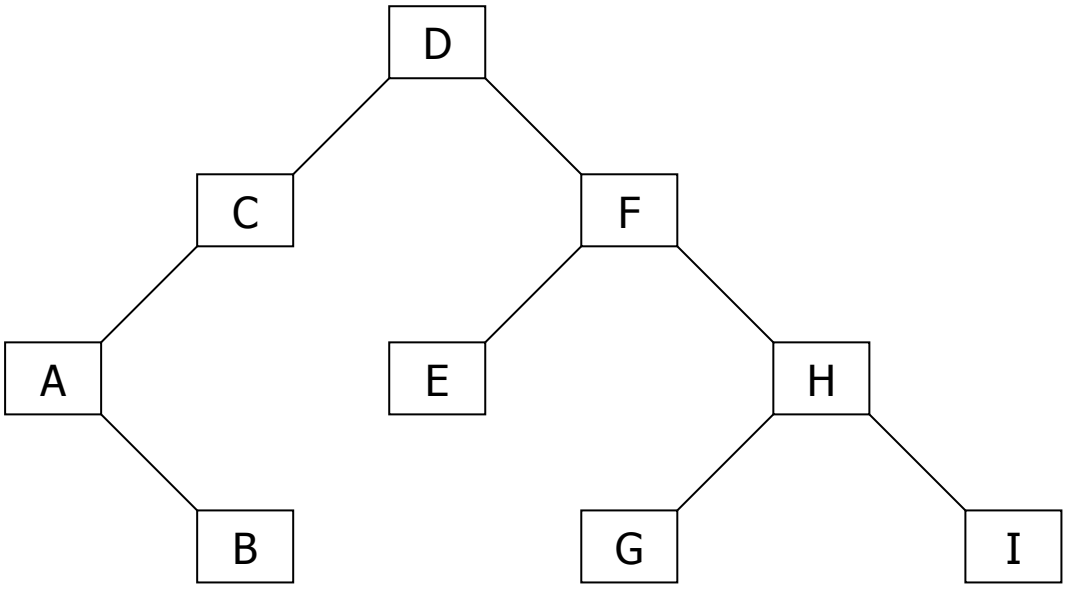


Species
rooted/unrooted trees
binary/ternary/quad trees
splay trees
parse trees
electrical nets
decision trees
game trees

Algorithms
Floyd-Warshal
Dijkstra
Kruskal
DFS/BFS/BestFS/etc
Union-Find (path compression)
Huffman
A*
Prim

Applications
Shortest path in graphs
Minimal cost connecting
Facility placement

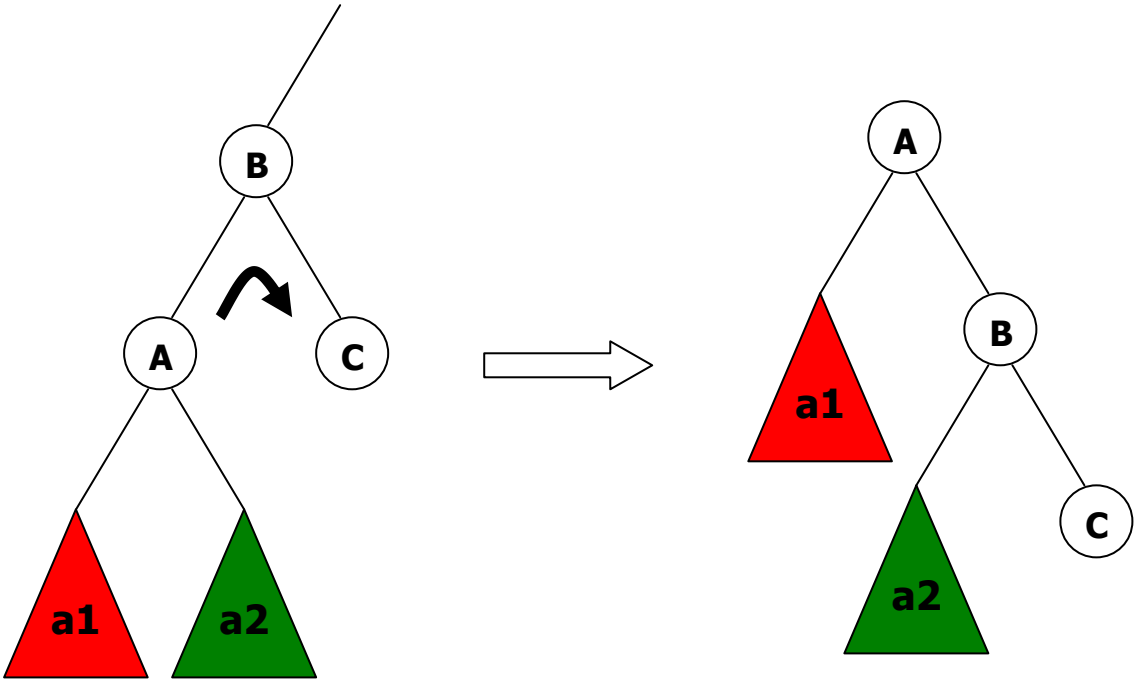
Binary Search Tree



	1	2	3	4	5	6	7	8	9
Label:	B	I	G	C	H	E	F	A	D
Left:	-	-	-	8	3	-	6	-	4
Right:	-	-	-	-	2	-	5	1	7

Root: [9]

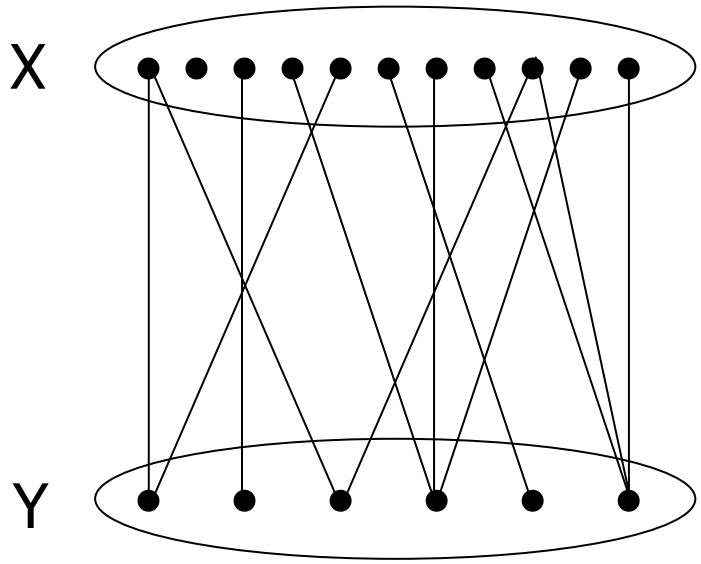
Simple rotation



Bipartite Graphs

G(V,E) is Bipartite: $V = X \cup Y$, $E \subseteq X \times Y$

$x,x' \in X$, but $xx' \notin E$
$y,y' \in Y$, but $yy' \notin E$



Theorem:

Bipartite

$\Longleftrightarrow$ 2 - colorable

$\Longleftrightarrow$ No odd length cycles

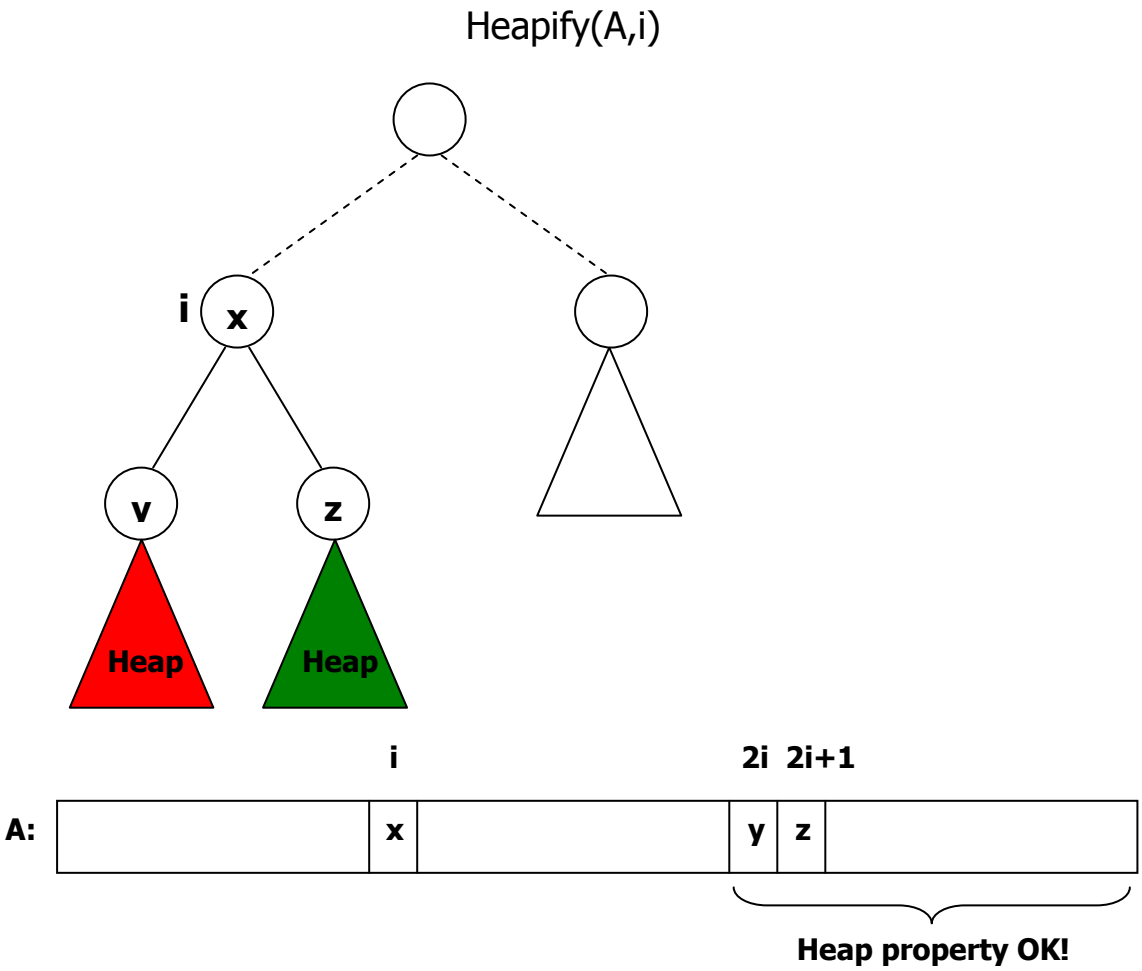
Heap

Complete Binary tree, but usual implementation uses Array A[] where:

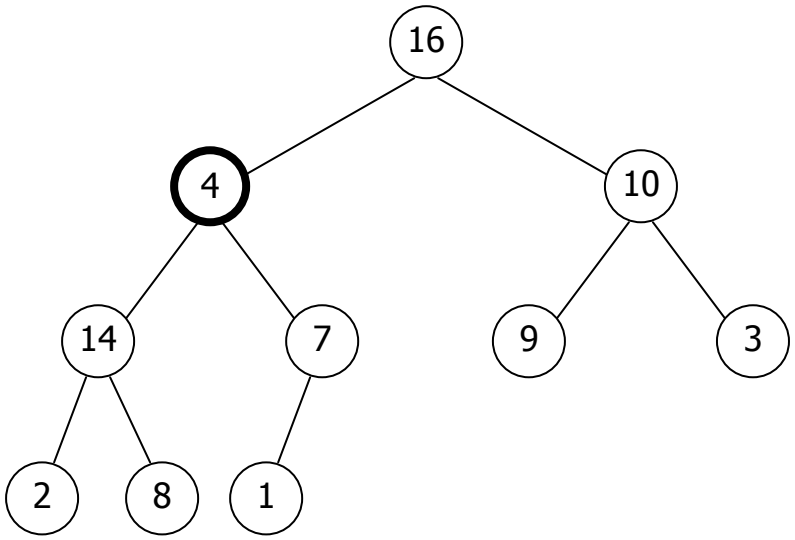
Left(A[i]) = A[2i]
Right(A[i]) = A[2i+1]
Root = A[1]
Parent(A[i]) = A[$\lfloor i/2 \rfloor$]

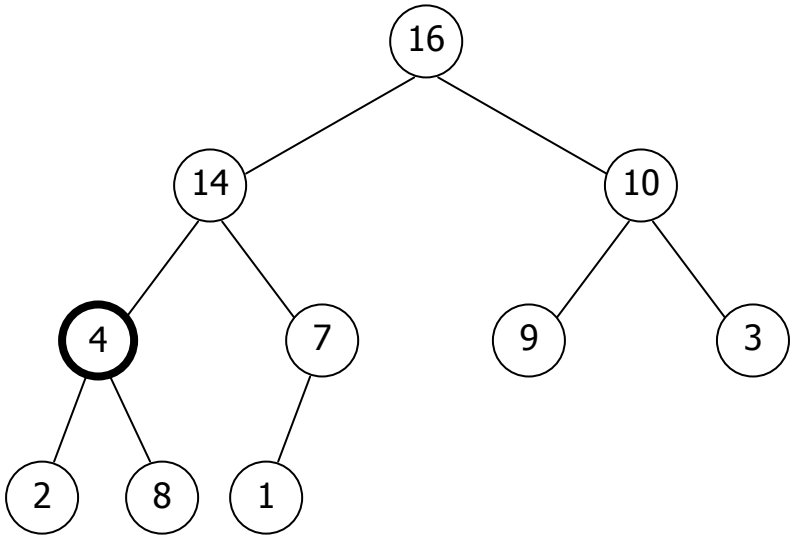
Heap property:
$A[i] \leq$ Parent(A[i])

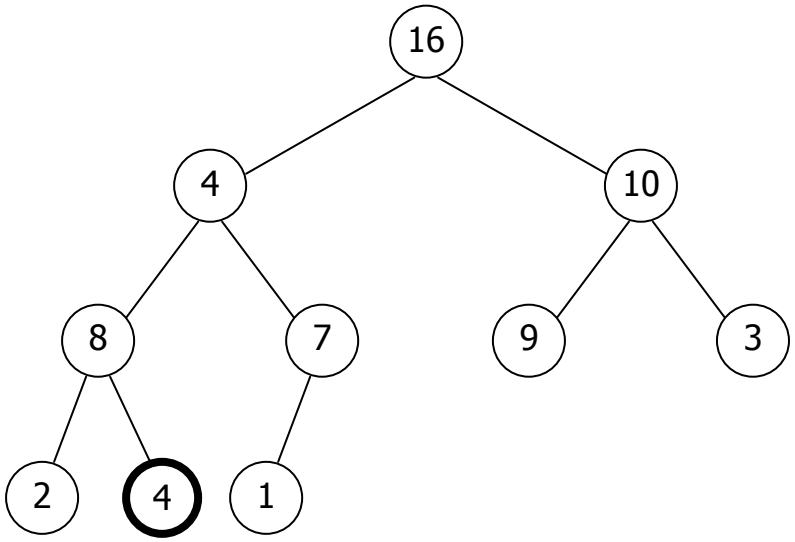
Important Algorithms:
Heapify(A,i)
Build_Heap(A)
HeapSort
Heap_Insert(A, value)



Maintaining the heap property



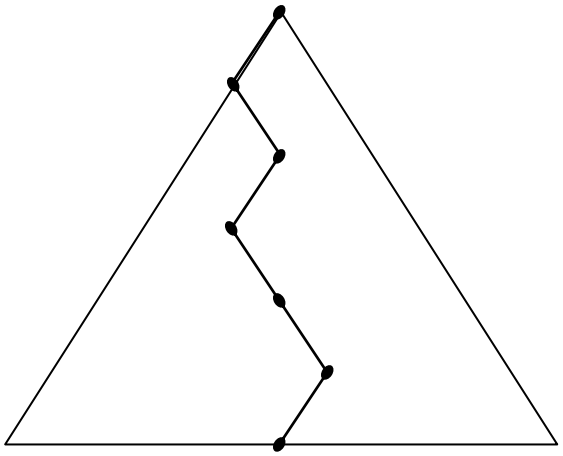




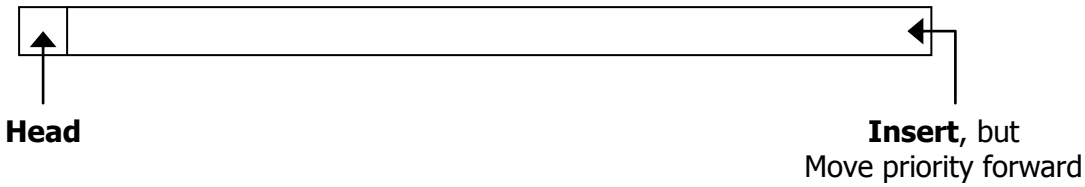
Variation of Heapify

- Lower # of comparisons
- Keep Larger Child Pointers (LCP) (bit vector)
- Unique LCP path from root to one leaf
- "Binary Insert" on this path instead of "float down"
- Update LCP after exchanges

Complexity:
Regular – 2log(n) operations
Variation – log(n)+loglog(n)



Priority Queues



Insert
Max
Extract-Max

Applications

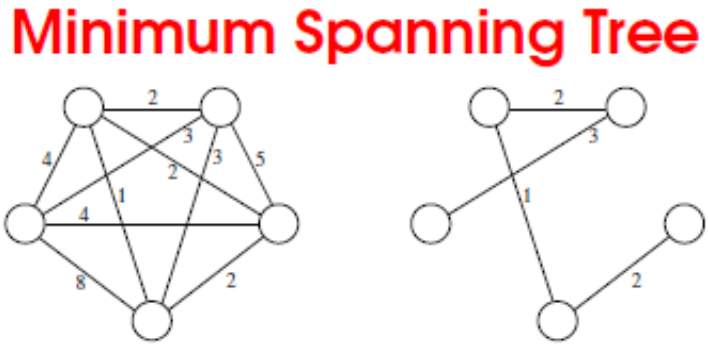
- Scheduling jobs by priority
- BEZEQ protectzia
- Minimum Spanning Tree algorithm (MST)

Minimum Spanning Tree - MST

G= (V,E) graph
w(u,v) weight of (u,v) $\in$ E

Find tree T $\subseteq$ E connecting all vertices (Spanning V) such that,

Min w(T) = $\sum_{(u,v)\in T}$ w(u,v)



Note:
1. G is connected ⇔ G has a spanning tree
2. |T| = |V|-1
3. T acyclic

Motivation for MST:
Connect all "pins", "cities", "nodes" with cheapest/shortest total amount of "wire", "roads", "cost".

"General" MST algorithm – a greedy strategy:

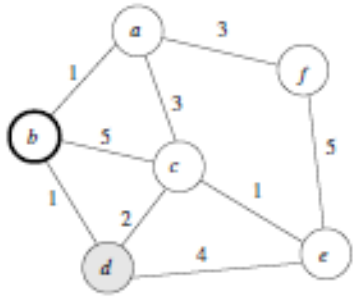
1. T ← ϕ
2. <u>while</u> T not spanning (|T| < |V|-1)
3. <u>do</u> choose safe edge (v,u) to add (T remains acyclic)
4. T ← T $\cup$ {(v,u)} <u>end do</u>
5. return T

Notes: **Prim's Algorithm (Army ground force attack)**

- T is an expanding Tree (starting from a chosen vertex as "root")
- uses priority queue for the vertices not in the tree
- Key(v) is cost of the cheapest "safe" edge which will extend the tree by adding vertex v. The safe edge is (v, Π(v))
- Π(v) is the (eventual) parent of v in T.
-

Minimum Spanning Tree: Prim's Algorithm

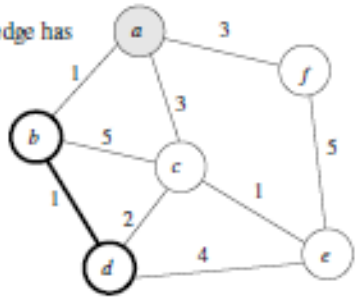
(a) Original graph



	a	b	c	d	e	f
d[ ]	1	0	5	1	∞	∞

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

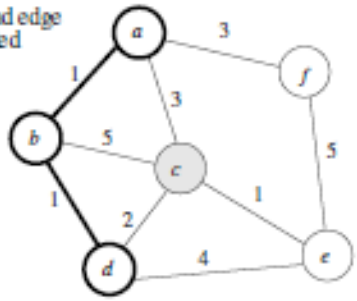
(b) After the first edge has been selected



	a	b	c	d	e	f
d[ ]	1	0	2	1	4	∞

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

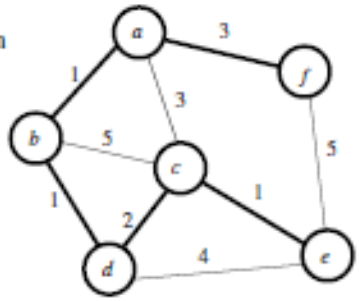
(c) After the second edge has been selected



	a	b	c	d	e	f
d[ ]	1	0	2	1	4	3

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

(d) Final minimum spanning tree



	a	b	c	d	e	f
d[ ]	1	0	2	1	1	3

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

Minimum Spanning Tree: Prim's Algorithm

```
1.   procedure PRIM_MST(V, E, w, r)
2.   begin
3.       V_T := {r};
4.       d[r] := 0;
5.       for all v ∈ (V − V_T) do
6.           if edge (r, v) exists set d[v] := w(r, v);
7.           else set d[v] := ∞;
8.       while V_T ≠ V do
9.       begin
10.          find a vertex u such that d[u] := min{d[v]|v ∈ (V − V_T)};
11.          V_T := V_T ∪ {u};
12.          for all v ∈ (V − V_T) do
13.              d[v] := min{d[v], w(u, v)};
14.      endwhile
15.  end PRIM_MST
```

Notes: **Kruskal's Algorithm (Air force attack)**

- F is an expanding forest
- Can use priority queue (reverse heap)
- Complexity O(E*logE)

```
KRUSKAL(G):
1    A = ∅
2    for each v ∈ V(G) do
3        MAKE-SET(v)
     end-do
4    for each (u, v)
     ordered by increasing weight w(u, v) do
5        if FIND-SET(u) ≠ FIND-SET(v):
6            A = A ∪ {(u, v)}
7            UNION(u, v)
     end-do
8    return A
```

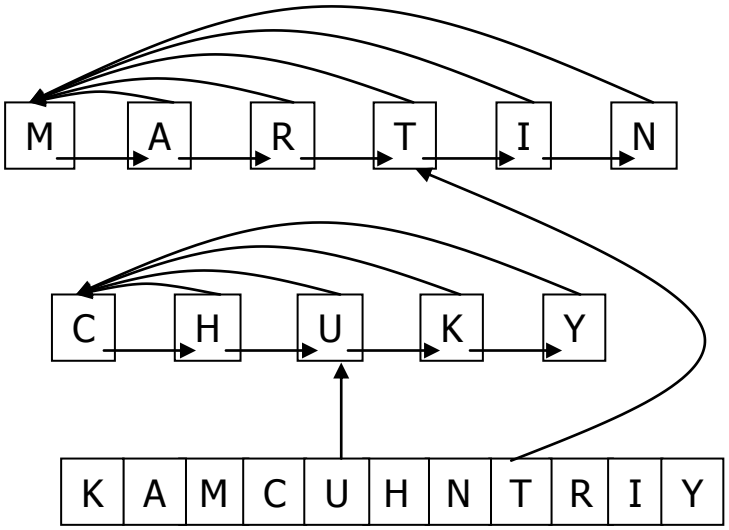
Data structures for <u>disjoint sets</u>

$S = \{ S_1 , S_2 , \ldots , S_k \}$ $\quad S_i \cap S_j = \phi \quad \forall\, i \neq j$

<u>Representative</u> - unique $r_i \in S_i$

<u>Find-Set(x)</u> – returns pointer to the unique representative of the set containing x.

<u>Union(x,y)</u> – combines their two sets, i.e., $S_x \cup S_y$ and assigns its representative.

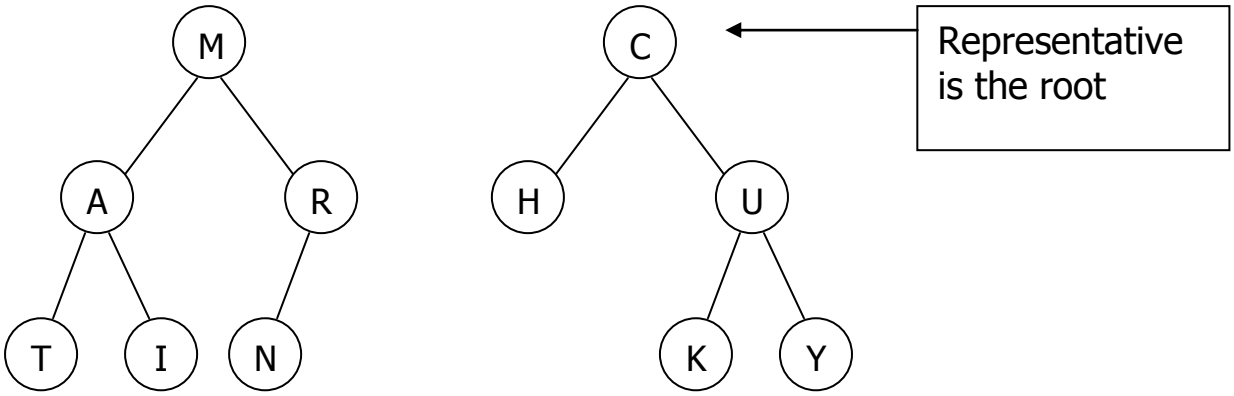


Find-Set O(1)
Union O(length of shorter list)

Complexity: O(m+n*logn)
|S|=n ; m= #of union, finds

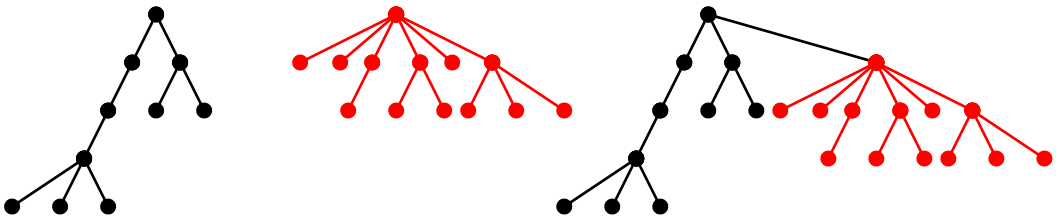
Union by concatenating smaller list to end of longer list

Forest of trees

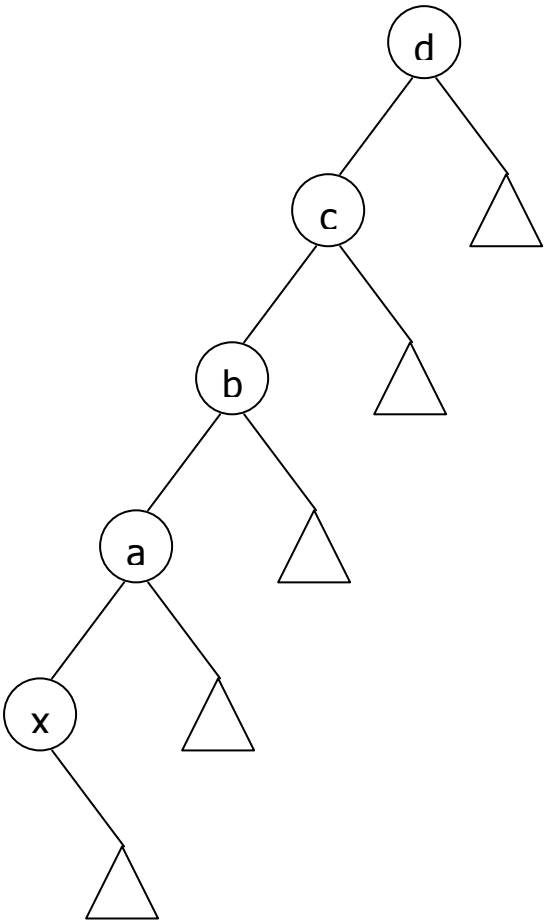


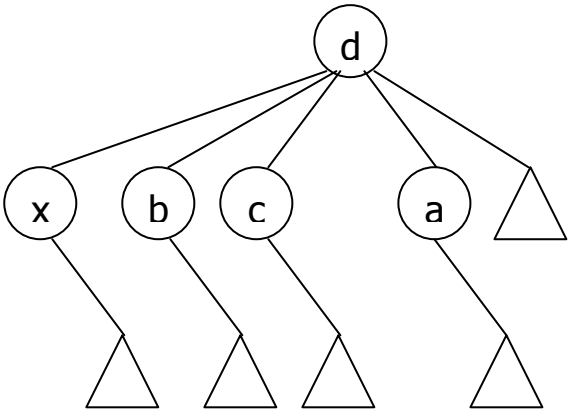
Find-Set O(distance from root)
Union O(1)

Union by making root of shorter tree a child of root of longer tree



Path compression





Complexity (with path compression)

m = #Union + finds
n = |S|

$O(m^* \alpha(m,n))$
$\alpha(m,n) \leq 4$, if $n \leq$ estimated number of atoms in the universe

P[x] parent
rank[x] $>\approx$ height of x (leaf to x)
equality for all roots

Make-Set(x)
P[x] ← x
rank[x] ← 0

Union(x, y)
r ← Find_Set(x)
s ← Find_Set(x)
if ($r \neq s$)
Link(r,s)

Link(r, s)
if (rank (r) > rank (s))
then P[s] ← r
else P[r] ← s
if (rank(r) == rank(s)
then rank(s) ← rank(s)+1

Find_Set(x)
if($x \neq P[x]$)
then P[x] ← Find_Set(P[x])
return P[x]

Single-Source Shortest Paths

For a weighted graph G = (V,E;w),

Find the shortest paths from a chosen vertex v to all other vertices in V .

Dijkstra's algorithm is somewhat similar to Prim's algorithm.

It maintains a set of nodes for which the shortest paths are known.

It grows this set based on the node closest to source using one of the nodes in the current shortest path set.

Single-Source Shortest Paths: Dijkstra's Algorithm

```
1.   procedure DIJKSTRA_SINGLE_SOURCE_SP(V, E, w, s)
2.   begin
3.       V_T := {s};
4.       for all v ∈ (V − V_T) do
5.           if (s, v) exists set l[v] := w(s, v);
6.           else set l[v] := ∞;
7.       while V_T ≠ V do
8.       begin
9.           find a vertex u such that l[u] := min{l[v]|v ∈ (V − V_T)};
10.          V_T := V_T ∪ {u};
11.          for all v ∈ (V − V_T) do
12.              l[v] := min{l[v], l[u] + w(u, v)};
13.      endwhile
14.  end DIJKSTRA_SINGLE_SOURCE_SP
```

EXAMPLE[..\barcelona summer school\04demo-dijkstra-1.ppt](..\barcelona summer school\04demo-dijkstra-1.ppt)

Other algorithms and techniques you should know:

MAXFLOW – maximum flow in a weighted network with constraints

BFS – Breadth First Search

DFS – Depth First Search

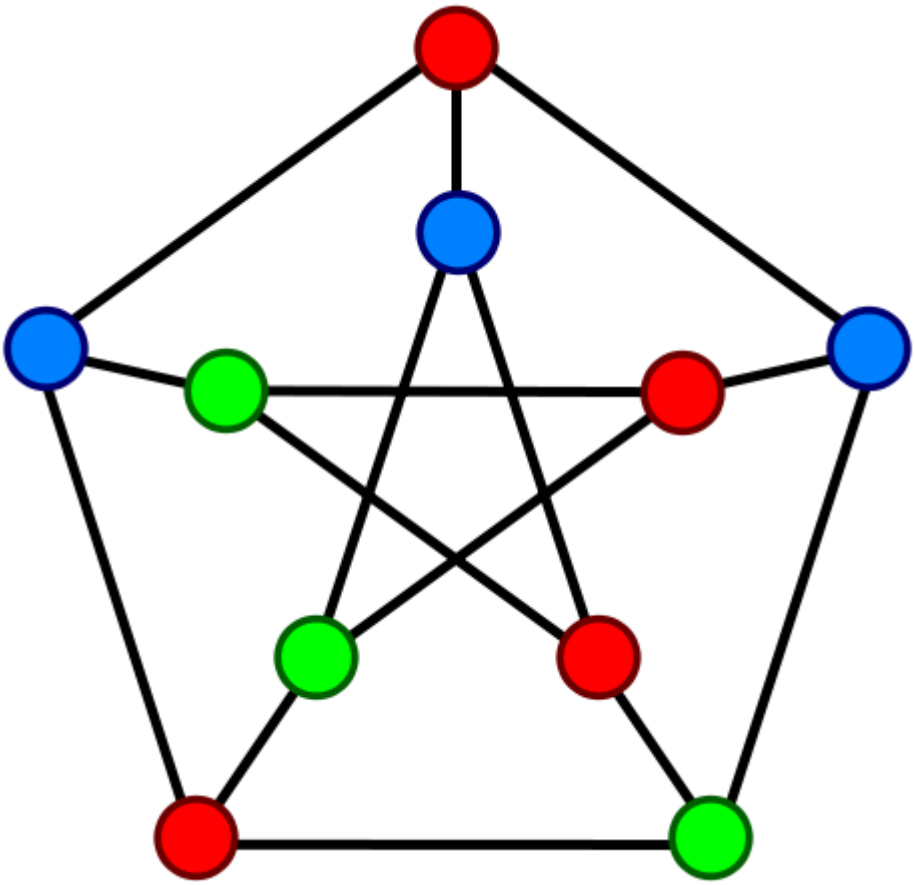
NPC – NP-completeness and problem reductions

How do I search thee, oh graph?

Let me count the ways...

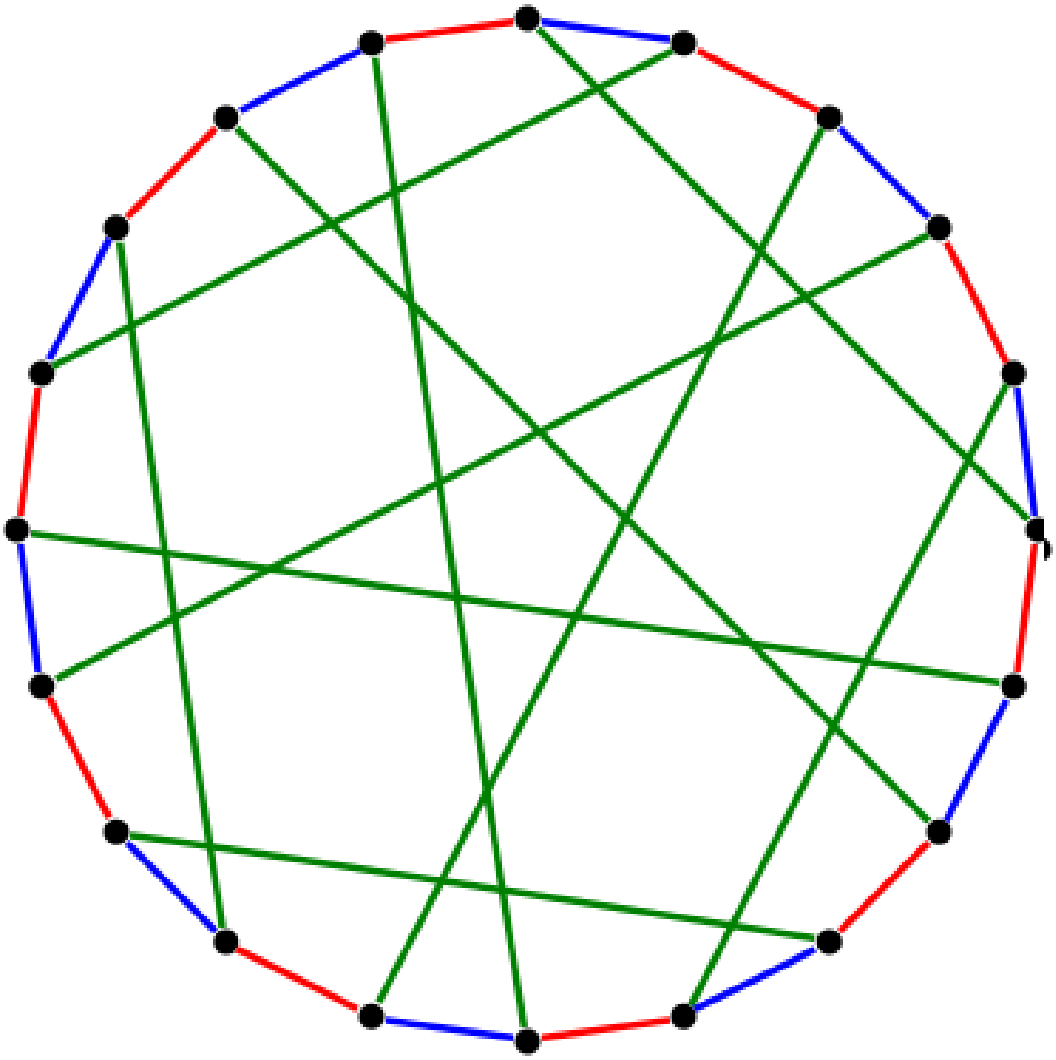
LexBFS, MCS, A*, IDA*, SMA*, LexDFS, …..

Graph Coloring (of vertices)



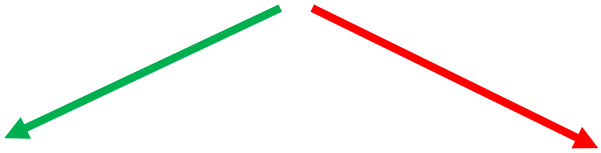
A proper vertex coloring of the Petersen graph with 3 colors, the minimum number possible.

Graph Coloring (of edges)



A 3-edge-coloring of the Desargues graph.

Algorithms for Graph Coloring



Structured Graphs

Often have polynomial-time algorithms

"exploit the structure"

General graphs

NP-Hard -- so heuristics are usually needed

Who is Daniel Brélaz?

Planar Graphs

Those which can be drawn on the Euclidean plane with edges crossing only at the vertices.

Example: