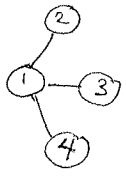
Graph review

Sections 22.1 & 23 of CLRS

A graph G is a set of vertices V (or nodes) and a set of edges $E \subseteq V \times V$

$G = (V, E)$

Example: $V = \{1, 2, 3, 4\}$ $E = \{(1,2) (1,3) (1,4)\}$



Edges may be directed $(1,2) \neq (2,1)$

or undirected $(1,2) = (2,1)$ ← sometimes undirected edges are written as $\{1,2\}$ to emphasize this.

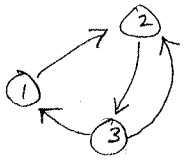
Used to model

telephone, computer, circuit boards, www, references in scientific papers, social networks, dependecies (building projects)

Representation

Let $n = |V|$ and $m = |E|$

Adjacency matrix: $A[u,v] = 1$ if $(u,v) \in E$, 0 otherwise



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array}$$

- \+ $O(1)$ time to check if $(u,v) \in E$
- − $\Theta(n^2)$ space
- − $\Theta(n)$ to find neighbors of v

if the graph is undirected this matrix is symmetric

Adjacency List

$N[v]$ = list of the neighbors of v

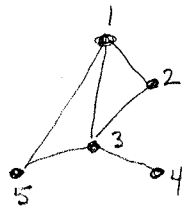
- \+ $O(m+n)$ space
- \+ $O(\deg(v))$ to find neighbors of v
- − $\Theta(\deg(u))$ to check if $(u,v) \in E$

degree of v ≡ the number of neighbors of v

$N[1] = [2]$
$N[2] = [3]$
$N[3] = [1, 2]$

if the graph is undirected, each edge (u,v) causes 2 entries: one in $N[u]$ and one in $N[v]$

Connectivity



A path from u to v is a sequence of vertices $[v_0\ v_1\ v_2 \dots v_k]$ where $v_0 = u$ and $v_k = v$ and $(v_{i-1}, v_i) \in E$ for all i.

path length = #edges

path = [1,2,1,5,3,4]

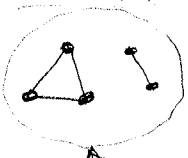
cycle = [1,2,3,4,3,1]

not a cycle [3,4,3]

A cycle is a path from u to u of length ≥ 1 (directed), length ≥ 3 (undirected) with no repeated vertices (except $v_0 = v_k$).

A simple path (cycle) has no repeated vertices (except $v_0 = v_k$).

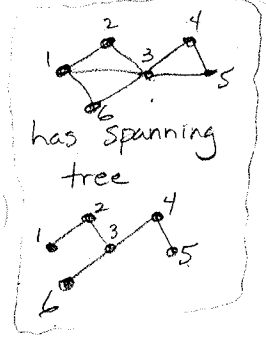
An undirected graph is connected if there is a path between every pair of (distinct) vertices



not connected

A tree is a connected, acyclic, undirected graph.

↑ no cycles



has spanning tree

A spanning tree of a graph $G = (V, E)$ is a tree $T = (V, E')$ where $E' \subseteq E$.

Minimum Spanning Trees

Given a connected, undirected graph $G = (V, E)$ with edge weights $w: E \to \mathbb{R}$

Find a spanning tree of G with smallest total edge weight.

This is an optimization problem. We want the best (smallest) feasible solution (spanning tree).

Other possible optimization problems:

- Shortest weighted path from u to v.
- Smallest weighted tour (path containing all vertices)

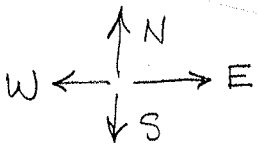
Greedy Algorithm

Make a sequence of "locally" optimal choices and hope that they lead to a the optimal solution.

For example:

Goal: Camp as close to the north rim of the grand canyon as possible.

Grand Canyon



Locally: at every intersection choose the southernmost branch.

Note: This greedy algorithm doesn't work!

Three Greedy Algorithms for Minimum Spanning Tree

(1) Kruskal's Algorithm:

Start with empty tree $T = (V, \emptyset)$

Consider edges of G in order of increasing weight

If T plus edge doesn't create a cycle, add it.

(2) Prim's Algorithm

Start with a single vertex in T

Add to T the vertex from G that is not in T but has the least weight edge to a vertex in T (and add this edge)

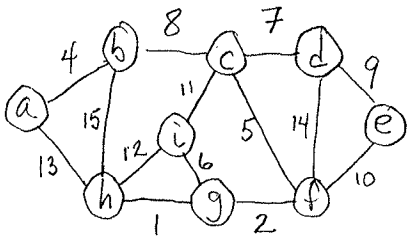
repeat until all vertices in G are in T

(3) Backwards Kruskal:

Start with $T = G$

Consider edges of G in order of decreasing weight

If T minus edge remains connected, remove it.



Why do these algorithms work?

we'll handle this later.

Assume edge weights are distinct.

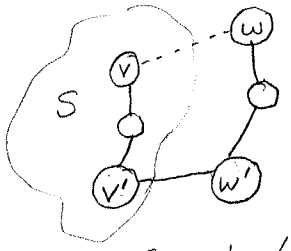
Theorem Let S be any subset of nodes ($S \neq \emptyset$, $S \neq V$)
Let $e = (v, w)$ be the minimum weight edge with $v \in S$ and $w \in V - S$.

How many different MSTs are there?

Then (every) minimum spanning tree contains e.

proof Suppose T is a mst spanning tree without e.

Since T is a spanning tree it contains a path from v to w.



Let v' be the last vertex on this path in S, and w' be the next vertex.

$(v' w') \in T$ and

(since $v' \in S$ and $w' \in V-S$) $w(v'w') > w(v,w)$

Edge Swap

→ So remove (v', w') from T and add (v, w) to get T'

T' is still connected (any path that used (v', w') can use the path from v' to w' that has (v, w) in it)

T' is acyclic (why?) T' is connected and has $n-1$ edges.

T' is a spanning tree with total weight < weight of T.

Contradiction

Kruskal's Algorithm

```
K-MST(G, w)
    T = (V, ∅)
    Sort edges of G by (increasing) weight        O(m lg m)
    For each edge (u,v) in order
        if (u,v) doesn't create a cycle in T
        then add it to T
    return T
```

How do we determine if (u,v) creates a cycle in T?

(1) do Breadth First or Depth First search from u in T to see if we can reach v

$\Rightarrow$ K-MST takes $O(m \lg m + m \cdot n)$

$= O(mn)$ (remember $m \leq n^2$) ← slow

(2) Keep track of the connected components of T

(A connected component in a graph is a maximal set of vertices that are all reachable from each other. (i.e. there is a path between all of them))

← 3 connected components

If u and v are in the same connected component then adding (u,v) creates a cycle.
If they are in different components then no cycle.

What we need:

Find(u) returns the set (component) containing u

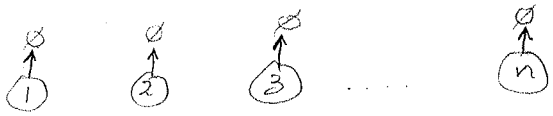
Union(u,v) put all the elements (vertices) in u's and v's sets into the same set

Note: every element (vertex) belongs to exactly one set.
Initially every element is in its own set

Disjoint Sets Data Structure. Chapter 21

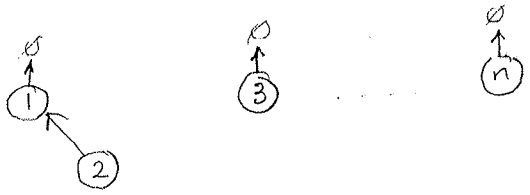
Implementation: Use a forest of trees

Initially:



n one-node trees (each has a null parent pointer)

Union(1,2)

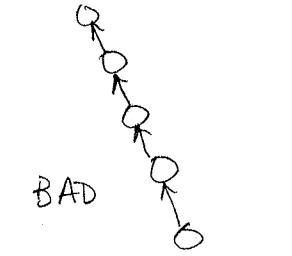


Union(u,v)

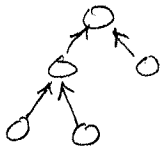
x = Find root of u's tree
y = Find root of v's tree
make y point to x

Find(u) ≡ follow parent pointers from u to the root of u's tree. Output that root.

Running time of Union(u,v) and Find(u) depend on the length of the path from u to the root of u's tree.



tall, skinny tree



GOOD

short, bushy tree

Union by Rank

For each tree node v keep a value called rank(v).
Initially rank[v] = 0 for all v

```
Union-by-Rank (u,v)
        x = Find (u)
        y = Find (v)
        if rank[x] > rank[y]
                make y point to x
        else
                make x point to y
                if rank[x]=rank[y] then rank[y]++
```

Lemma (A) Each node with rank r has at least 2^r descendants. (including itself)

proof by induction on r

For $r=0$ true the node itself is $2^0=1$ descendant.

Suppose the claim is true for all nodes with rank $< r$.

A node gets rank $r>0$ when it and another node of rank $r-1$ are Union'ed together. By I.H. each has $\geq 2^{r-1}$ descendants so the rank r node becomes the root of a tree with $\geq 2^{r-1}+2^{r-1} = 2^r$ descendants.

Corollary No node has rank $> \lg n$

Otherwise it would have more than n descendants.

What is rank? rank(v) $\geq$ length of any path from a leaf in v's tree to v.

(In fact, so far, rank = height.)

So... no Find(v) operation takes more than $O(\lg n)$ time.

Kruskal's Alg running time using disjoint sets with Union-by-rank takes:

$$O(m \lg m + m \lg n) = O(m \lg n) \text{ time}$$

(remember $m < n^2$)

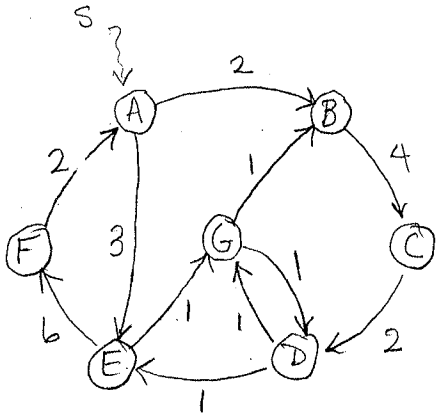
Shortest Path (from a single source vertex)

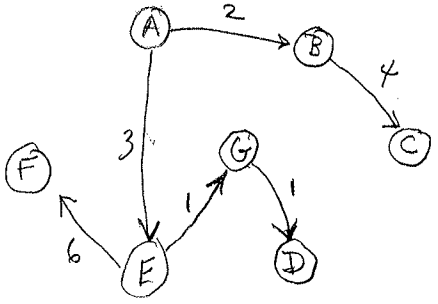
Given an edge-weighted directed graph G and a source vertex s
Find the shortest paths from s to every other vertex in G.

(if undirected then treat each edge as two directed edges)

Facts about shortest paths:

1. No shortest path contains the same vertex twice (provided edge weights are positive)
2. The prefix of a shortest path is a shortest path.





Shortest path tree (rooted at s=A)

	Shortest path length from s
S=A	0
B	2
E	3
G	4
D	5
C	6
F	9

Idea Build shortest path tree T in order of path length from s.

Greedy idea Start with $T = (\ ,\emptyset)$
Add vertex to T that is **closest to s**
using edges of T plus one additional edge.

Repeat

add this edge to T

Dijkstra's Algorithm (like Prim's)

$d[u]$ = the length of a shortest path from s to u using edges from T plus (perhaps) one additional edge

$p[u]$ = vertex that precedes u on this path

Dijkstra ($G=(V,E)$, s)

$d[u] = \infty$ $\quad p[u] = \emptyset$ for all $u \in V$

$d[s] = 0$

Q.insert $(u, d[u])$ for all $u \in V$

$T = (\emptyset, \emptyset)$

for $k = 1$ to n

  $u = Q$.deletemin ( )

  Add u and edge $(u\ p[u])$ to T

  for each $(u,v) \in E$ do

    if $v \notin T$ and $d[v] > d[u] + w(u,v)$

    then $p[v] = u$

      $d[v] = d[u] + w(u,v)$

      Q.updatepriority $(v, d[v])$

Example

$s = d$