From Last Time

① Can prove that path compression alone with $\underline{n}$ Make-Set ops, $\underline{f}$ FIND-SET ops, and <u>arbitrary # of unions</u> (up to $n-1$) cost $\Theta(n+f(1+\log_{2+\frac{f}{n}} n))$

② $\Theta(m\alpha(n))$, where m = total # ops, n = # of elements, running time using forest of trees with union-by-rank and path compression proven in §21.4
— YOU ARE NOT RESPONSIBLE FOR THIS

Greedy Algorithms

Greedy algorithm: Overall problem solved in series of steps. Choice made at each step looks best at the moment without explicit reference to overall problem. "locally optimal"

Example:

We need to make 99¢ in change with minimum # of coins. We do this with a greedy algorithm automatically.

$$99¢ = (25¢) \times 3 + (10¢) \times 2 + (5¢) \times 0 + (1¢) \times 4$$

$$\begin{array}{r} 99 \\ -75 \\ \hline 24 \\ -20 \\ \hline 4 \\ 0 \\ \hline 0 \end{array}$$

3 quarters + 2 dimes + 4 pennies

This greedy algorithm gives correct solution.
Starting with largest coin, take as many as possible without going over.

BUT ...
1. start with pennies: $99¢ = (1¢) \times 99 \Rightarrow 99$ coins
2. If the dime was replaced by an 11¢ piece, make 15¢:
   greedy: $15¢ = (11¢) \times 1 + (5¢) \times 0 + (1¢) \times 4 \rightarrow 5$ coins
   correct answer: $15¢ = (11¢) \times 0 + (5¢) \times 3 \rightarrow 3$ coins

So greedy algorithms
- sometimes yield correct solution (globally optimal)
- sometimes do not
  - change made is correct, but minimum number of coins not achieved

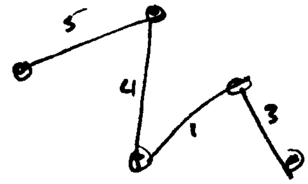
Depends on
- structure of algorithm (forward/reverse)
- structure of problem (coin values)

We will see that

① in some cases can <u>prove</u> greedy algorithm leads to globally optimal solution

② in other cases greedy solutions lead to good solutions (avg case behavior) that are rapid to find and worthwhile, but not guaranteed optimal.

Graphs (CLRS Appendix B.4)
- Graph is a set of vertices (points) connected by edges (lines that join two points) - directed/undirected
- "Weight" is additional information such as distance between vertices (associated with edge)
- "Connected": a path exists between any pair of vertices by traversing (multiple) edges



Minimum Spanning Tree (MST) Problem

Input: A connected, undirected graph $G=(V,E)$ with weight function $w: E \to \mathbb{R}$

Output: A spanning tree, T (connecting all vertices) of minimum weight

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

sum over edges of tree

Why is this problem interesting?

① Shortest Path Connectivity

ⓐ For electric circuitry, often need to wire together set of contacts. Desirable to use minimal amount of wire. → MST problem

ⓑ On a larger scale, to connect multiple sites by telecommunications network, want minimal cost scheme. Weights could be length of wire or cost to install, or other

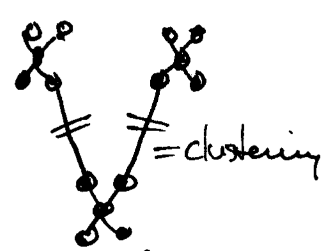
② One form of clustering achieved by cutting longest edges in MST

Examples

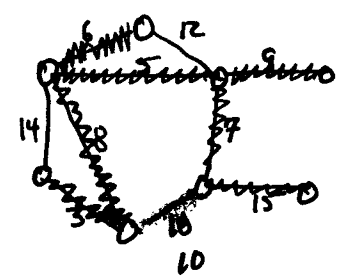
not minimal → minimal

no cycles in minimum if all edges are positive

"all edges possible"



minimal



Some properties: # of edges $= |E| = O(V^2)$ → bounded from above

If graph G connected, then $|E| \geq |V| - 1 \Rightarrow \lg|E| = \Theta(\lg|V|)$

↳ bounded from below

Would you think a greedy algorithm would work here?

Can growing a tree one step at a time (in a greedy manner) lead to a globally optimum smallest weight solution?

$\boxed{\text{THE MST PROPERTY}}$ → Somewhat different from book

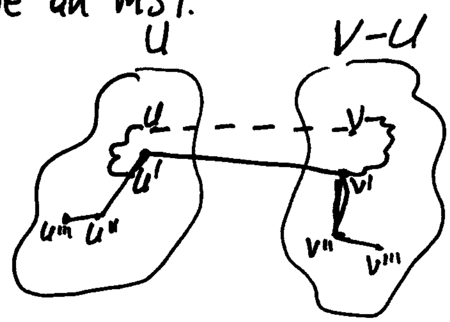
Theorem: Let $G = (V, E)$ be connected graph with cost function defined on edges. Let U be some proper subset of V. If (u, v) is an edge of lowest cost such that $u \in U$ and $v \in V - U$, then there is an MST containing (u, v).

"light edge"

Proof: (Every MST satisfies above) (By "cut-and-paste")

Assume the theorem false: no MST that includes (u, v).

Let T be an MST.



Can't have cycles, even if weight is negative, because then not a tree

- adding (u, v) introduces a cycle, because T is an MST so already has path from u to v
- there must be another edge from U to $V - U$, (u', v') WLOG and there may be more than one

- deleting edge (u', v') breaks cycle, giving tree T'
- T' has weight $\leq T$ because (u, v) was lowest cost edge
- Thus, our assumption is wrong and theorem is true: (u, v) in MST

Now do you think greedy algorithm might work? We can use local information to exclude (and include) edges from growing MSTs.

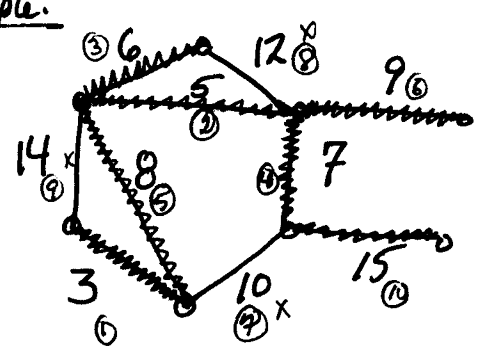
Size of search space – All graphs – Each edge can be in or out: $2^{|E|}$

→ some will not be trees
→ some will not be connected
→ some will not be minimum cost
→ could enumerate each, evaluate whether connected (disjoint set operations) and compute weight

$\boxed{\text{Kruskal's Algorithm}}$

- Initially $T = (V, \emptyset)$ (vertices but no edges)
- Examine edges of E in "increasing" weight order (non-decreasing)
  - If edge connects two ~~components~~ unconnected components, then add edge to T
  - Else discard edge and continue (forms cycle)
  - Can terminate when all edges in single connected component

Example:



Correctness of ~~Proof~~ Alg.: Loop invariant

Prior to each iteration, T is a subset of a MST

Initialization: T has no edges, so trivially satisfied

Maintenance: Edges are only accepted in loop if part of MST

Termination: All edges are examined and added to T if in MST, so T must be MST

Book does somewhat stronger proof for non-unique MST.

Pseudo Code for Implementation

MST-Kruskal (G, w)

initialize edge list to $\emptyset$ — $A \leftarrow \emptyset$ → $O(1)$

make a forest of trees (root only) for the vertices — for each vertex $v \in V[G]$
    do Make-Set(v) → account for below

DISJOINT SET OPS OF LAST TIME

sort the edges of E into non-decreasing order by w → $O(E \lg E)$

for each edge $(u, v) \in E$ in non-decreasing order

    if edge connects disconnected components → do if Find-Set$(u) \neq$ Find-Set(v)

    add edge to MST → then $A \leftarrow A \cup \{(u, v)\}$

    join disconnected components → UNION(u, v)

return A

$O(E)$ Find-Set & Union operations plus $O(V)$ Make-Sets

$\Rightarrow O((V+E)\alpha(V))$

connected: $|E| \geq |V|+1$

$\Rightarrow O(E\alpha(V))$

$\alpha(V) = O(\lg V) = O(\lg E)$

$\Rightarrow O(E \lg E) \overset{(\lg E = O(\lg V))}{\Rightarrow} O(E \lg V)$

—//—