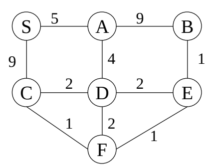
More Graph Algorithms: Minimum Spanning Trees

CS211
Fall 2000

Dijkstra's Algorithm

- Intuition
  - Edges are threads; vertices are beads
  - Pick up at s; mark each node as it leave the table
- Note: Negative edge-costs are *not allowed*



- s is the start vertex
- c(i,j) is the cost from i to j
- Initially, vertices are unmarked
- dist[v] is length of s-to-v path
- Initially, dist[v] = ∞, for all v

```
dijsktra(s):
    dist[s] = 0;
    while (some vertices are unmarked) {
        v = unmarked vertex with
            smallest dist;
        Mark v;            // v leaves "table"
        for (each w adj to v) {
            dist[w] = min
                [ dist[w], dist[v] + c(v,w) ];
        }
    }
```

2

Greedy Algorithms

- Dijkstra's Algorithm is an example of a Greedy Algorithm
- The Greedy Strategy is an algorithm design technique
  - Like Divide & Conquer
- The Greedy Strategy is used to solve optimization problems
  - The goal is to find the *best* solution
- Works when the problem has the *greedy-choice property*
  - A global optimum can be reached by making locally optimum choices
- Problem: Given an amount of money, find the smallest number of coins to make that amount
- Solution: Use a Greedy Algorithm
  - Give as many large coins as you can
- This greedy strategy produces the optimum number of coins for the US coin system
- Different money system ⇒ greedy strategy may fail
  - For example: suppose the US introduces a 4¢ coin

3

Minimum Spanning Trees

Definition

A *spanning tree* of an undirected graph G is a *tree* whose nodes are the vertices of G and whose edges are a subset of the edges of G

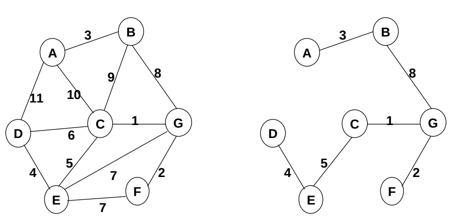
Definition

A *Minimum Spanning Tree (MST)* for a weighted graph G is the spanning tree of least cost (sum of edge-weights)

- Alternately, an MST can be defined as the least-cost set of edges so that all the vertices are connected
  - This has to be a tree… Why?
- A greedy strategy works for this problem
  - Add vertices one at a time
  - Always add the one that is closest to the current tree
  - This is called Prim's Algorithm

4

An Example Graph and Its MST



5

Prim's Algorithm

- s is the start vertex
- c(i,j) is the cost from i to j
- Initially, vertices are unmarked
- dist[v] is length of smallest tree-to-v edge
- Initially, dist[v] = ∞, for all v

```
prim(s):
    dist[s] = 0;
    while (some vertices are unmarked) {
        v = unmarked vertex with
            smallest dist;
        Mark v;
        for (each w adj to v) {
            dist[w] = min[ dist[w], c(v,w) ];
        }
    }
```

- Runtime analysis
  - $O(v^2)$ for adj matrix
    - While-loop is executed v times
    - For-loop takes O(v) time
  - O(e + v log v) for adj list
    - Use a PQ
    - Regular PQ produces time O(v + e log e)
    - Can improve to O(e + v log v) by using fancier heap

6

Similar Code Structures

```
while (some vertices are unmarked) {
    v = best of unmarked vertices;
    Mark v;
    for (each w adj to v)
        Update w;
}
```

- bfsDistance
  - best: next in queue
  - update: dist[w] = dist[v]+1
- dijkstra
  - best: next in PQ
  - update:dist[w] =min [dist[w],dist[v]+cost(v,w)]
- prim
  - best: next in PQ
  - update: dist[w] = min [dist[w],cost(v,w)]

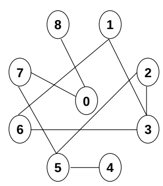
Remembering Your Choices

- How can you remember which choices were made?
  - Whenever dist[w] is updated we can remember the current v by using parent[w] = v;
  - Can use the parent info to construct the *bfs tree*, the *shortest path tree*, or the *minimum spanning tree*

```
while (some vertices are unmarked) {
    v = best of unmarked vertices;
    Mark v;
    for (each w adj to v)
        Update w;
        if (w changed) parent[w] = v;
}
```


New Problem: Connectivity

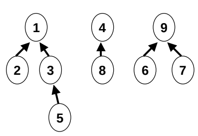
- Given a set of integer pairs (p,q), determine if p' and q' are connected
- Example:
  - Given pairs (1,3) (2,3) (5,4) (6,3) (7,5) (1,6) (7,0) (0,8) (5,2)
  - Are 4 and 6 connected?
- How can a computer resolve this for a large set?



Union and Find

- We break this problem into two operations
  - Union: Combine two sets
  - Find: Given an item, determine the "name" of the set that contains it
- Many applications
  - Checking components of a dynamic graph
  - Computers in a network: Can p communicate with q?
  - Minimum Spanning Trees

Union/Find using Reverse Trees



The root is the "name" of the set

- Find
  - Follow links to root
  - Time O(n) in the worst case
- Union
  - Link root of one tree to the root of the other
  - Time O(1) in the worst case

An Improvement: Union by Size

- Note: Every union takes one tree and moves everything in it one step farther from the root
- Idea: Make the *smaller* tree be the one that moves down
- Can show
  - Time for union is O(1)
  - Time for find is O(log n)
- Implement using arrays
- Initially, all items have no parent and size 1

	parent	size
0		
1		
2		
.		
.		
.		
.		
.		
.		
n		

Union-by-Size Lemma

<u>Lemma</u>

A tree with height h contains at least 2^h nodes

<u>Proof</u>

- The only way in which a node can change its level is when it is within the *smaller* of two trees participating in a union
- Thus, when any node *x* drops a level, the tree that it is within doubles in size (or more)
- If a node is at level h then it is within a tree of size at least 2^h

<u>Corollary</u>

Worst-case time for find is O(log n) where n is the total number of items

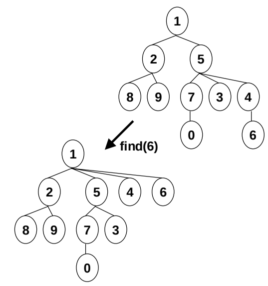
<u>Proof</u>

- The largest possible tree contains n nodes, so the deepest node is at level *log n*

13

Union-by-Size + Path Compression

- Idea: Every time we "find" something, we update every item we touch so that it points at the root
  - This is *almost* free since we have to touch these items anyway
  - Intuition: next time we find one of these items it will be quicker
- Does this help?



14

Yes, It Helps

<u>Theorem</u> (Tarjan)

Using weighted union and path compression, a sequence of n union/find operations takes time O(n α(n))

- The function α(n) is the inverse of *Ackerman's function* and it grows <u>*very*</u> slowly

<u>Definition</u> (Ackerman's function)

$$A(p,q) = \begin{cases} 2q & \text{if } p = 0 \\ 0 & \text{if } q=0, p>0 \\ 2 & \text{if } q=1, p>0 \\ A(p-1, A(p, q-1)) & \text{if } q>1, p>0 \end{cases}$$

This definition is a bit different from the text's version, but both have similar properties

15

Ackerman's Function

- A(0,q) = 2 + … + 2 = 2q
- A(1,q) = 2 $*$ … $*$ 2 = 2^q
- A(2,q) = $2^{2^{\cdot^{\cdot^{2}}}}$ (a height-q stack of 2's)
- Thus A(2,4) = 2^{16} = 65,536
- Each level does the operation from the previous level q times
- What is A(3,4)?
- So A(4,4) must be <u>*extremely*</u> large

16

Definition for α(n)

<u>Definition</u> (*inverse Ackerman's function*)

α(n) =

least x such that A(x,x) $\geq$ n

Note that $\alpha(n) \leq 4$ for any integer n that we are *ever* likely to encounter

- Is the α(n) factor really necessary?
  - Yes: Tarjan showed a *lower* bound of Ω(n α(n)) for union/find
  - Claim: the inverse Ackerman's function is not just an artifact of this one problem

17