Dijkstra’s Algorithm

I 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(s) is length of s-to-v path
Initially, dist(s) = ∞, for all v
dijkstra(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], dist[v] + c(v,w) ];
}
}

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

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

Prim’s Algorithm

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

An Example Graph and Its MST

Greedy Algorithms
Similar Code Structures

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

Remembering Your Choices

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

- 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
- Implement using arrays
- Initially, all items have no parent and size 1
- 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)
Union-by-Size Lemma

**Lemma**
A tree with height $h$ contains at least $2^h$ nodes

**Proof**
- 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)

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

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

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?

Yes, It Helps

**Theorem (Tarjan)**
Using weighted union and path compression, a sequence of $n$ union/find operations takes time $O(n \alpha(n))$

**Definition (Ackerman’s function)**
- $A(p, q) = 2^{q^p}$ if $p = 0$
- $A(p, q) = 2 \cdot A(p-1, A(p, q-1))$ if $q > 1$, $p > 0$
- $A(0, q) = 2 + \ldots + 2^q = 2^{q+1} - 1$
- $A(1, q) = 2 \cdot \ldots \cdot 2^q = 2^{q+1} - 1$
- $A(2, q) = 2^{2^q}$ (a height-$q$ stack of $2$’s)

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

Definition for $\alpha(n)$

- $\alpha(n) = \min \{ x : A(x, x) \geq n \}$
- Note that $\alpha(n) \leq 4$ for any integer $n$ that we are ever likely to encounter

**Claim:** $\Omega(n \alpha(n))$ for union/find

**Definition (inverse Ackerman’s function)**
- $\alpha(n) = \min \{ x : A(x, x) \geq n \}$
- Is the $\alpha(n)$ factor really necessary?
- Yes: Tarjan showed a lower bound of $\Omega(n \alpha(n))$ for union/find
- Claim: the inverse Ackerman’s function is not just an artifact of this one problem

Ackerman’s Function

- $A(0, q) = 2 + \ldots + 2 = 2q$
- $A(1, q) = 2 \cdot \ldots \cdot 2 = 2^{q+1} - 1$
- $A(2, q) = 2^{2^q}$ (a height-$q$ stack of $2$’s)

- Thus $A(2, q) \approx 2^{2^q}$
- Each level does the operation from the previous level $q$ times
- $A(2, q) = 2^{2^q}$
- What is $A(3, 4)$?
- $A(4, 4)$ must be extremely large