

CSE 326 Data Structures

Dave Bacon

Dijkstra's Algorithm and Minimal Spanning
Trees

12

Logisitics

- Project 3 writeup due on Thursday!
- Homework 7 will be due Wed, March 7
- Read Chapter 9 of Weiss

Dijkstra's Alg: Implementation Optimized

Initialize the cost of each node to ∞

Initialize the cost of the source to 0

While there are unknown nodes left in the graph

 Select the unknown node b with the lowest cost

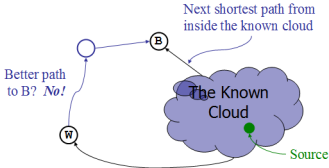
 Mark b as known

 For each node a adjacent to b

a 's cost = $\min(a$'s old cost, b 's cost + cost of (b, a))

Running time?

Correctness: The Cloud Proof



How does Dijkstra's decide which vertex to add to the Known set next???

- If path to **B** is shortest, path to **W** must be *at least as long* (or else we would have picked **W** as the next vertex)
- So any path *through W* to **B** cannot be any shorter!

Correctness: Inside the Cloud

Prove by induction on # of nodes in the cloud:

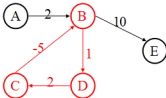
Initial cloud is just the source with shortest path 0

Assume: Everything inside the cloud has the correct shortest path

Inductive step: Only when we prove the shortest path to some node v (which is not in the cloud) is correct, we add it to the cloud

When does Dijkstra's algorithm not work?

The Trouble with Negative Weight Cycles



What's the shortest path from A to E?

Problem?

Dijkstra's vs BFS

At each step:

- 1) Pick closest unknown vertex
- 2) Add it to finished vertices
- 3) Update distances

Dijkstra's Algorithm

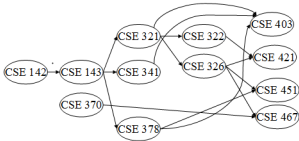
At each step:

- 1) Pick vertex from queue
- 2) Add it to visited vertices
- 3) Update queue with neighbors

Breadth-first Search

Some Similarities:

Acyclic Graphs?



Minimum Spanning Trees

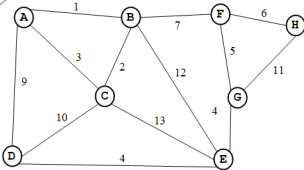
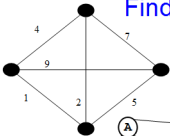
Given an undirected graph $G=(V,E)$, find a graph $G'=(V, E')$ such that:

- E' is a subset of E
- $|E'| = |V| - 1$
- G' is connected
- $\sum_{(u,v) \in E'} c_{uv}$ is minimal

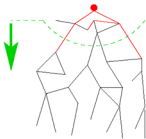
G' is a **minimum spanning tree**.

Applications: wiring a house, power grids, Internet connections

Find the MST



Two Different Approaches



Prim's Algorithm

Almost identical to Dijkstra's

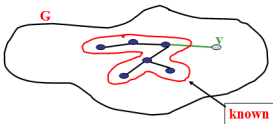


Kruskals's Algorithm

Completely different!

Prim's algorithm

Idea: Grow a tree by adding an edge from the “known” vertices to the “unknown” vertices. Pick the edge with the smallest weight.



Prim's Algorithm for MST

A *node-based greedy algorithm*

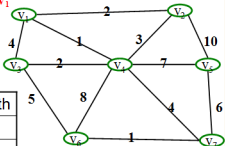
Builds MST by greedily adding nodes

1. Select a node to be the "root"
 - mark it as *known*
 - Update cost of all its neighbors
2. While there are *unknown* nodes left in the graph
 - a. Select an *unknown* node b with the smallest cost from some *known* node a
 - b. Mark b as *known*
 - c. Add (a, b) to MST
 - d. Update cost of all nodes adjacent to b

Note: cost from some a ,
not from root

Find MST using Prim's

V	Kwn	Distance	path
v1			
v2			
v3			
v4			
v5			
v6			
v7			



Order Declared Known:

V_1

Prim's Algorithm Analysis

Running time:

Same as Dijkstra's: $O(|E| \log |V|)$

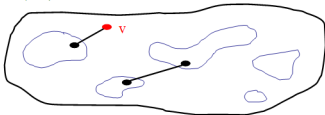
Correctness:

Proof is similar to Dijkstra's

Kruskal's MST Algorithm

Idea: Grow a **forest** out of edges that do not create a cycle. Pick an **edge with the smallest weight**.

$G=(V,E)$



Kruskal's Algorithm for MST

An edge-based greedy algorithm

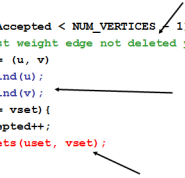
Builds MST by greedily adding edges

1. Initialize with
 - empty MST
 - all vertices marked unconnected
 - all edges **unmarked**
2. While there are still **unmarked** edges
 - a. Pick the lowest cost edge (u, v) and mark it
 - b. If u and v are not already connected, add (u, v) to the MST and mark u and v as connected to each other

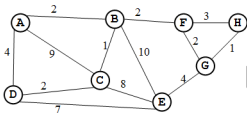
Doesn't it sound familiar?

Kruskal code

```
void Graph::kruskal() {  
    int edgesAccepted = 0;  
    DisjSet s(NUM_VERTICES);  
  
    while (edgesAccepted < NUM_VERTICES - 1) {  
        e = smallest weight edge not deleted yet;  
        // edge e = (u, v)  
        uset = s.find(u);  
        vset = s.find(v);  
        if (uset != vset) {  
            edgesAccepted++;  
            s.unionSets(uset, vset);  
        }  
    }  
}
```



Find MST using Kruskal's



Total Cost:

- Now find the MST using Prim's method.
- Under what conditions will these methods give the same result?

Kruskal's Algorithm: Correctness

It clearly generates a spanning tree. Call it T_K .

Suppose T_K is *not* minimum:

Pick another spanning tree T_{min} with *lower* cost than T_K

Pick the smallest edge $e_1=(u,v)$ in T_K that is not in T_{min}

T_{min} already has a path p in T_{min} from u to v

⇒ Adding e_1 to T_{min} will create a cycle in T_{min}

Pick an edge e_2 in p that Kruskal's algorithm considered *after*

adding e_1 (must exist: u and v unconnected when e_1 considered)

⇒ $\text{cost}(e_2) \geq \text{cost}(e_1)$

⇒ can replace e_2 with e_1 in T_{min} without increasing cost!

Keep doing this until T_{min} is identical to T_K

⇒ T_K must also be minimal – contradiction!

Single-Source Shortest Path

- Given a graph $G = (V, E)$ and a single distinguished vertex s , find the shortest weighted path from s to every other vertex in G .

All-Pairs Shortest Path:

- Find the shortest paths between all pairs of vertices in the graph.
- How?

Analysis

- Total running time for Dijkstra's:
 $O(|V|^2 + |E|)$ (linear scan)
 $O(|V| \log |V| + |E| \log |V|)$ (heaps)

What if we want to find the shortest path
from each point to ALL other points?

Dynamic Programming

Algorithmic technique that systematically records the answers to sub-problems in a table and re-uses those recorded results (rather than re-computing them).

Simple Example: Calculating the Nth Fibonacci number.

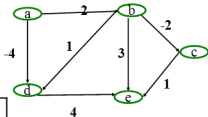
$$\text{Fib}(N) = \text{Fib}(N-1) + \text{Fib}(N-2)$$

Floyd-Warshall

```
For (int k = 0; k < n; k++)  
  For (int i = 0; i < n; i++)  
    For (int j = 0; j < n; j++)  
      M[i][j] =  
        min(M[i][j], M[i][k] + M[k][j])
```

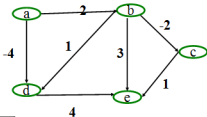

Initial state of the matrix:

	a	b	c	d	e
a	0	2	∞	-4	∞
b	∞	0	-2	1	3
c	∞	∞	0	∞	1
d	∞	∞	∞	0	4
e	∞	∞	∞	∞	0



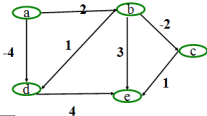
$$M[i][j] = \min(M[i][j], M[i][k] + M[k][j])$$

Floyd-Warshall



	a	b	c	d	e
a	0				
b		0			
c			0		
d				0	
e					0

Floyd-Warshall -
for All-pairs
shortest path



	a	b	c	d	e
a	0	2	0	-4	0
b	-	0	-2	1	-1
c	-	-	0	-	1
d	-	-	-	0	4
e	-	-	-	-	0

Final Matrix
Contents

Transitive Closure

The transitive closure of a graph $G=(V,E)$
Is the graph $G^* = (V, E^*)$ where

$$E^* = \{ (i,j) : \text{there is a path from vertex } i \text{ to} \\ \text{vertex } j \text{ in } G \}$$

“All-pairs reachability”