CSE 421
Algorithms

Richard Anderson

Lecture 10

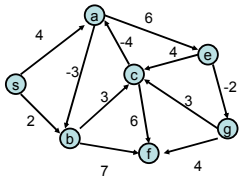
Minimum Spanning Trees

Announcements

- Homework 3 is due now
- Homework 4, due 10/26, available now
- Reading
  - Chapter 5
  - (Sections 4.8, 4.9 will not be covered in class)
- Guest lecturers (10/28 – 11/4)
  - Anna Karlin
  - Venkat Guruswami

Shortest Paths

- Negative Cost Edges
  - Dijkstra's algorithm assumes positive cost edges
  - For some applications, negative cost edges make sense
  - Shortest path not well defined if a graph has a negative cost cycle

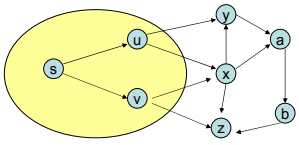


Negative Cost Edge Preview

- Topological Sort can be used for solving the shortest path problem in directed acyclic graphs
- Bellman-Ford algorithm finds shortest paths in a graph with negative cost edges (or reports the existence of a negative cost cycle).

Dijkstra's Algorithm Implementation and Runtime

```
S = {};   d[s] = 0;    d[v] = inf for v != s
While S != V
        Choose v in V-S with minimum d[v]
        Add v to S
        For each  w in the neighborhood of v
                d[w] = min(d[w], d[v] + c(v, w))
```



HEAP OPERATIONS

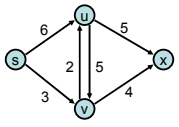
n Extract Min

m Heap Update

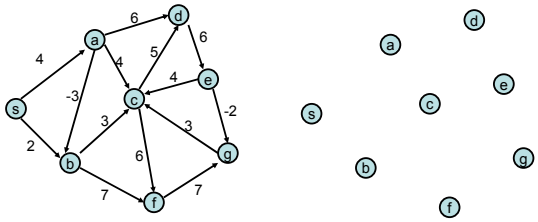
Edge costs are assumed to be non-negative

Bottleneck Shortest Path

- Define the bottleneck distance for a path to be the maximum cost edge along the path



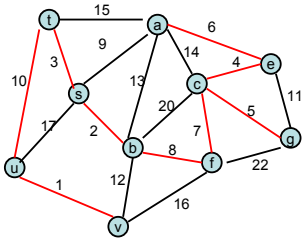
Compute the bottleneck shortest paths



How do you adapt Dijkstra's algorithm to handle bottleneck distances

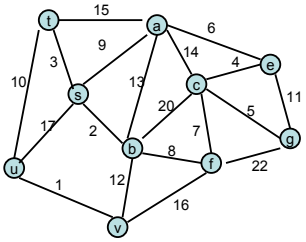
- Does the correctness proof still apply?

Minimum Spanning Tree



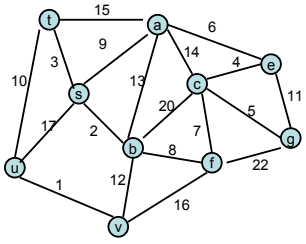
Greedy Algorithm 1 Prim's Algorithm

- Extend a tree by including the cheapest out going edge



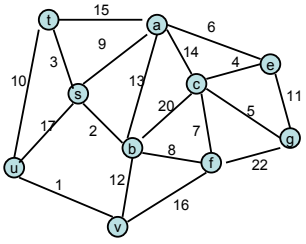
Greedy Algorithm 2 Kruskal's Algorithm

- Add the cheapest edge that joins disjoint components



Greedy Algorithm 3 Reverse-Delete

- Delete the most expensive edge that does not disconnect the graph



Why do the greedy algorithms work?

- For simplicity, assume all edge costs are distinct
- Let S be a subset of V, and suppose e = (u, v) be the minimum cost edge of E, with u in S and v in V-S
- e is in every minimum spanning tree

Proof

- Suppose T is a spanning tree that does not contain e
- Add e to T, this creates a cycle
- The cycle must have some edge $e_1 = (u_1, v_1)$ with u_1 in S and v_1 in V-S
- $T_1 = T - \{e_1\} + \{e\}$ is a spanning tree with lower cost
- Hence, T is not a minimum spanning tree

Optimality Proofs

- Prim's Algorithm computes a MST
- Kruskal's Algorithm computes a MST

Reverse-Delete Algorithm

- Lemma: The most expensive edge on a cycle is never in a minimum spanning tree

Dealing with the assumption

- Force the edge weights to be distinct
  - Add small quantities to the weights
  - Give a tie breaking rule for equal weight edges