CSE 417: Algorithms and Computational Complexity

Winter 2001

Lecture 11

Instructor: Paul Beame

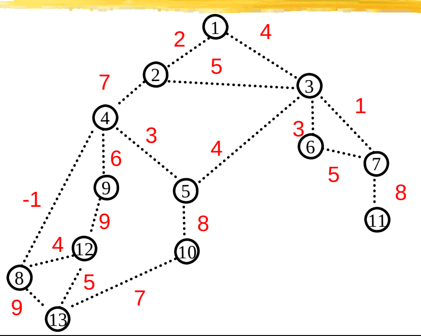
1

Minimum Spanning Trees (Forests)

- Given an undirected graph G=(V,E) with each edge e having a weight w(e)
- Find a subgraph T of G of minimum total weight s.t. every pair of vertices connected in G are also connected in T
  - if G is connected then T is a tree otherwise it is a forest

2

Weighted Undirected Graph



3

First Greedy Algorithm

- Prim's Algorithm:
  - start at a vertex v
  - add the cheapest edge adjacent to v
  - repeatedly add the cheapest edge that joins the vertices explored so far to the rest of the graph.

4

Second Greedy Algorithm

- Kruskal's Algorithm
  - Start with the vertices and no edges
  - Repeatedly add the cheapest edge that joins two different components. i.e. that doesn't create a cycle

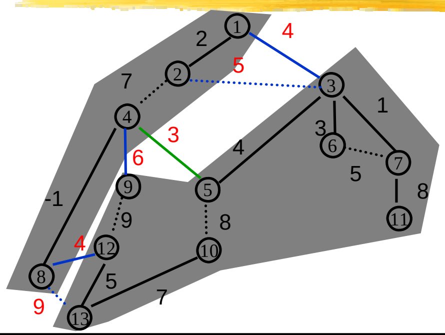
5

Why greed is good

- **Definition:** Given a graph G=(V,E), a **cut** of G is a partition of V into two non-empty pieces, S and V-S
- **Lemma:** For every cut (S,V-S) of G, there is a minimum spanning tree (or forest) containing any **cheapest edge crossing the cut**, i.e. connecting some node in S with some node in V-S.
  - call such an edge **safe**

6

Cuts and Spanning Trees



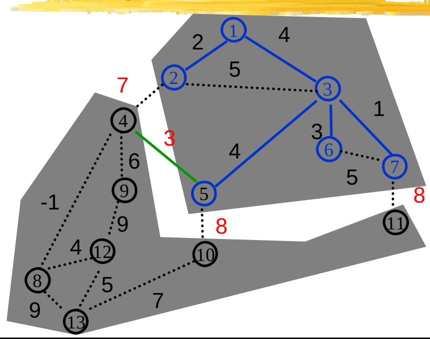
7

The greedy algorithms always choose safe edges

- Prim's Algorithm
  - Always chooses cheapest edge from current tree to rest of the graph
  - This is cheapest edge across a cut which has the vertices of that tree on one side.

8

Prim's Algorithm



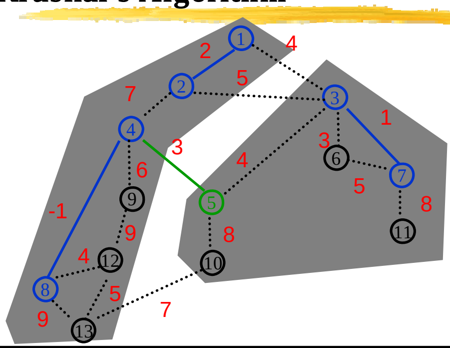
9

The greedy algorithms always choose safe edges

- Kruskal's Algorithm
  - Always chooses cheapest edge connected two pieces of the graph that aren't yet connected
  - This is the cheapest edge across any cut which has those two pieces on different sides and doesn't split any current pieces.

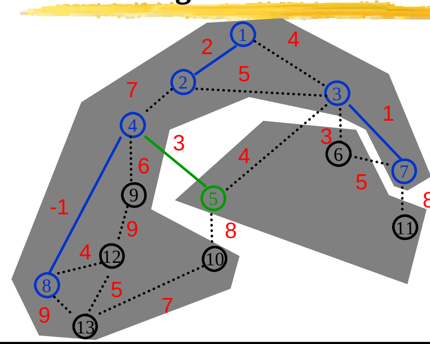
10

Kruskal's Algorithm



11

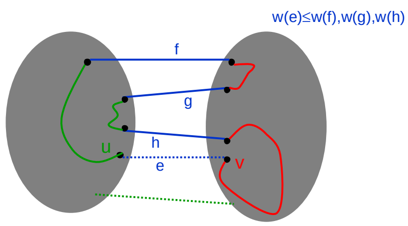
Kruskal's Algorithm



12

Proof of Lemma

Suppose you have an MST not using cheapest edge e

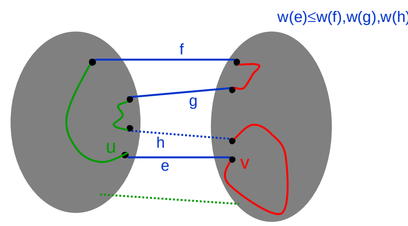


Endpoints of e, u and v must be connected in T

13

Proof of Lemma

Replacing h by e does not increase weight of T



All the same points are connected by the new tree

14

Naive Prim's Algorithm Implementation & Analysis

- Computing the minimum weight edge at each stage. O(m) per step new vertex
- n vertices in total
- O(nm) overall

15

Kruskal's Algorithm Implementation & Analysis

- First sort the edges by weight O(m log m)
- Go through edges from smallest to largest
  - if endpoints of edge e are currently in different components
    - then add to the graph
    - else skip
- Union-find data structure handles last part
- Total cost of last part: O(m α(n)) where α(n)<< log m
- Overall O(m log n)

16

Union-find disjoint sets data structure

- Maintaining components
  - start with n different components
    - one per vertex
  - find components of the two endpoints of e
    - 2m finds
  - union two components when edge connecting them is added
    - n-1 unions

17

Prim's Algorithm with Priority Queues

- For each vertex u not in tree maintain current cheapest edge from tree to u
  - Store u in priority queue with key = weight of this edge
- Operations:
  - n-1 insertions (each vertex added once)
  - n-1 delete-mins (each vertex deleted once)
    - pick the vertex of smallest key, remove it from the p.q. and add its edge to the graph
  - <m decrease-keys (each edge updates one vertex)

18

Prim's Algorithm with Priority Queues

- Priority queue implementations
  - Array
    - insert $O(1)$, delete-min $O(n)$, decrease-key $O(1)$
    - total $O(n+n^2+m)=O(n^2)$
  - Heap
    - insert, delete-min, decrease-key all $O(\log n)$
    - total $O(m \log n)$
  - d-Heap ($d=m/n$)
    - insert, delete-min, decrease-key all $O(\log_{m/n} n)$
    - total $O(m \log_{m/n} n)$

19