Parma University, Math. Department

Algorithms and Data Structures
Mimimum Spanning Trees

Marco Pellegrini

Summary

- This lecture:
  - Minimum spanning trees in weighted graphs
  - Prim's algorithm
  - Kruskal's algorithm
  - Proof of correctness
- Material: CLR 24.1-2

Minimum spanning trees

- We are given a weighted undirected graph G=(V,E) with weight function w:E→R. G is connected.
- We want to find the smallest sub-graph G' of G which has as vertices V, is connected and has minimum total weight.
- Example 1: V represent switch points, E represents telephone lines and w(e) represents the cost of setting up the telephone line. Finding the MST is equivalent to finding the cheapest way to have any node to communicate with any node.
- Example 2: V= Pins on a board to be electrically equivalent, E= pairs of possible connections, w(e) length of the wire e.

MST

- We assume all the weights to be positive numbers.
- The graph G'=(V,E') which is a connected subgraph of G=(V,E) of minimum weight must be a Tree.
- Why? Suppose you have G' which is not a tree, then it has a cycle. If you delete an edge of the cycle the new graph is still connected and its weight is smaller.
- We repeat until we get a tree T.
- Lemma (CLR 5.2): If a graph on n nodes is connected and has n-1 edges then it is a tree.
- Note that there can possibly be many different MST for the same graph G.

Algorithms: Kruskal and Prim.

- We shall see two algorithms for computing a MST of G.
- Kruskal algorithm is similar to the algorithm for finding connected components using disjoint sets operations.
- Prim's algorithm is similar to Dijkstra's algorithm.
- Both are based on the idea that the obvious move based on the current state is the right thing to do. That is: to make a tree of smallest weight choose the next edge that does not create cycles and has minimum weight.
- Note that there are many other problems where this does not work.
- We will give one proof of correctness that is good for both algorithms.

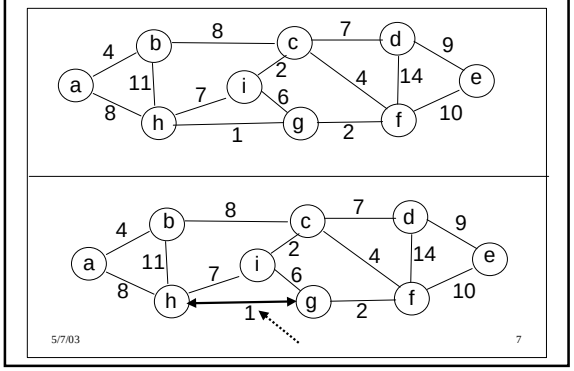
Kruskal's algorithm

Represent the tree T as a set A of edges and a disjoint-set data structure D representing the connected components of T.

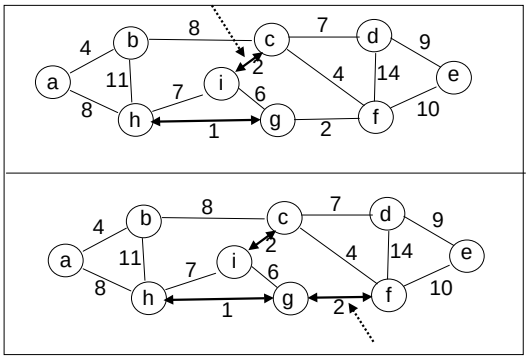
Initially A is empty and for each node V we have one singleton set in D.

```
MST-Kruskal(V,E)
    A := Emptyset,
    FOR each v IN V DO
      make-set(D,v)
    END FOR
    sort E by nondecreasing weight order
    FOR each (u,v) in E in the order DO
      IF findset(D,u) ≠ findset(D,v)
        THEN A = A ∪ {(u,v)}
             union(u,v)
      END IF
    END FOR
    Return A
```

Example

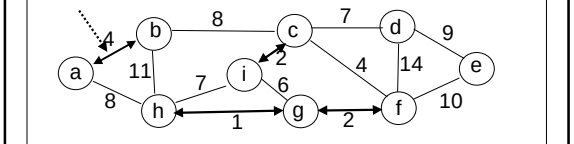


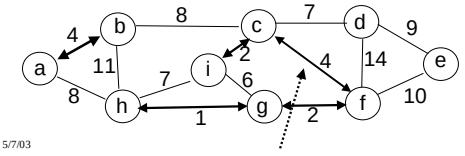
5/7/03 7



5/7/03 8

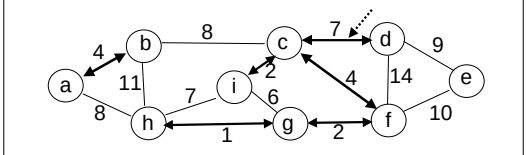
Example (3)





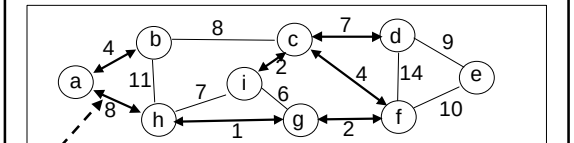
5/7/03 9

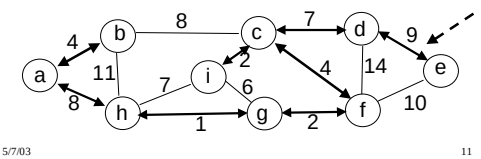
Example (4)



5/7/03 10

Example (5)

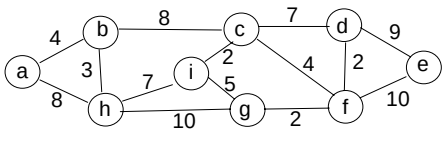




5/7/03 11

Exercise:

- Simulate the execution of Kruskal's algorithm on the graph below.
- If you have edges with equal weight just pick one, the MST is not unique when there are equal weights.
- What is the total weight of your MST?



5/7/03 12

Running time of Kruskal's algorithm

- The first FOR loop takes time O(|V|).
- Sorting the edges takes time O(|E| log |E|).
- The we have a loop executed |E| times, in the loop we call twice find-set and once union. So we have a sequence of 3|E| disjoint set operations over |V| elements.
- The implementation of union-by-size, would make these operations in time O(|E| log |V|).
  Using path-compression and union-by rank we can achieve O(|E| log* |V|).
- The total time is thus O(|E| log |E|).

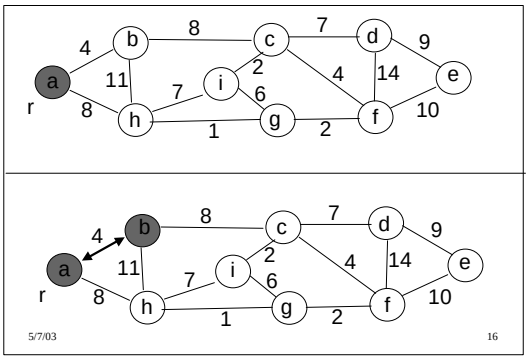
Prim's algorithm

- Prim's algorithm keeps at any moment only one connected component of T and makes it grow until it contains all nodes.
- We need a priority-queue Q supporting operations extract-min and decrease-key.
- We need a key k[v] associated to every node which is the weight of the edge of smaller weight that joins v in V/T to a node in T.
- We need an array p[1..n] that stores for every v the adjacent node u in T that we use when we make (u,v) an edge of T.
- We start from a node r.

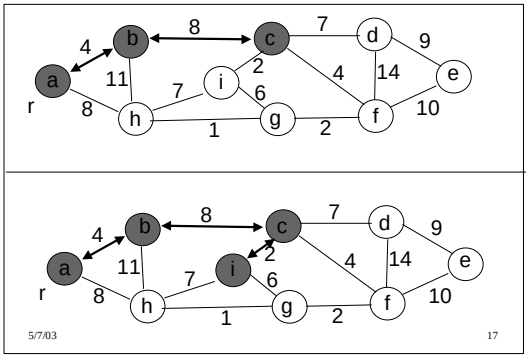
Prim's algorithm

```
MST-Prim(G,r)
    FOR each v in V DO
       k[v] := infinity
    END FOR
    k[r] := 0
    p[r] := NIL
    Q := marke-priority-queue(V,k)
    WHILE NOT empty(Q) DO
       u := extract-min(Q)
       FOR each v adjacent to u DO
          IF (v is in Q) AND w(u,v) < k[v]
            THEN  k[v] := w(u,v)
                  p[v] := u
                  decreasekey(Q,v,k[v])
          END IF
       END FOR
    END WHILE
```

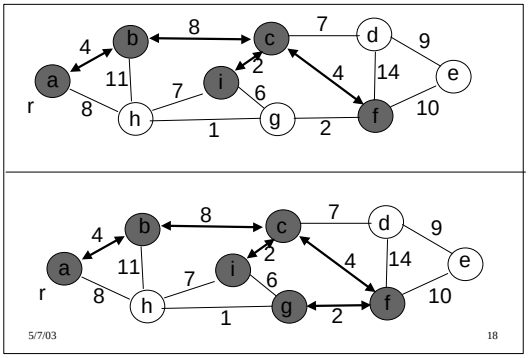

Example (1)



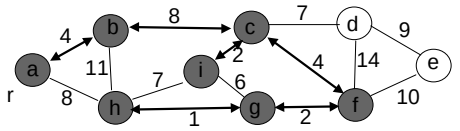
Example (2)



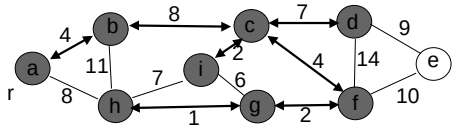
Example (3)

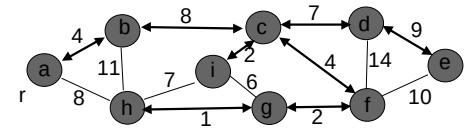


Example (4)



Example (5)





Exercise

- Show a MST produced by Prim's algorithm on the graph of the example when we start from node e.

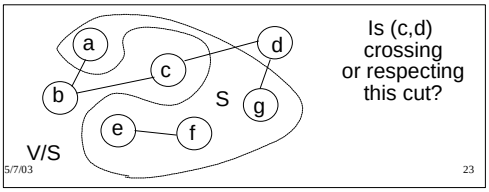
Analysis of the running time of Prim's algorithm

- The structure is very similar to Dijkstra SSSP algorithm. The analysis is also very similar.
- Using results in HW 4.7 you can implement decrease-key and extract-min and heapify in time O(log |V|).
- The total time is O((|E| +|V|) log |V|).
- It is possible to use more sophisticated heaps (Fibonacci heaps) and achieve time O(|E| + |V| log |V|).

Proof of correctness of MST algorithms

(Preliminary definitions)

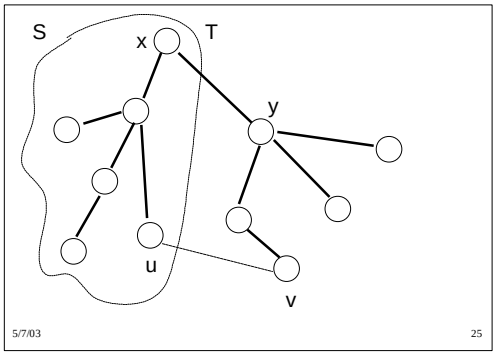
- A cut is a partition of the vertices of G into two sets (S, V/S).
- An edge e=(u,v) respects the cut if both u and v are in S or both u and v are in V/S.
- An edge with one adjacent vertex in S and one in V/S crosses the cut.



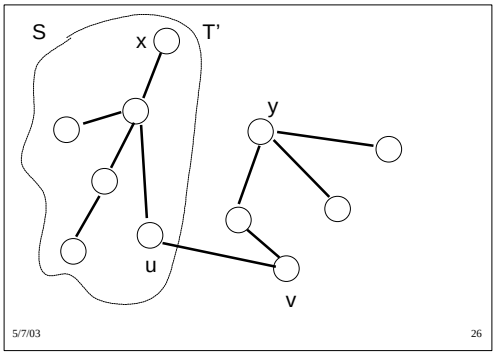
Main Theorem

- Let G=(V,E) be an undirected connected graph with real positive edge weight function w.
  Let (S, S/V) be a cut, and A a set of edges that
  (1) respect the cut
  (2) are a subset of the edges of an MST T.
  Let (u,v) be the crossing edge with smallest weight.
- We want to prove that there exists an MST T' that contains A and (u,v).
- Case 1: (u,v) is in T, so T'=T and we are done.
- Case 2: (u,v) is not in T, so T+{(u,v)} has a cycle. Such cycle must cross the cut at least twice in (u,v) and in say (x,y). (x,y) is not in A because it is a crossing edge. If we remove (x,y) and add (u,v) we obtain a new tree T'. But w(T') ≤ w(T), so it must be w(T) = w(T'). We have found T' containing A and (u,v).

A drawing is worth 1000 words.



A drawing is worth 1000 words.



Correctness of Prim's algorithm

- In Prim's algorithm initially the cut is ({r}, V/{r}) and A is empty. Then selecting the edge from r with smallest weight we have an edge that the main theorem guarantees belonging to an MST.
- During the algorithm the cut is always between the black nodes extracted from the heap and all the other nodes. Since we always choose the crossing edge of minimum weight we are guaranteed by the main theorem that the set of edges we are building is in an MST.
- Finally when we have extracted all the nodes from the priority queue we have an MST tree.

Correctness of Kruskal's algorithm

- Initially A is empty. Let (u,v) be the edge with smallest weight. Our cut is ({u}, V/{u}). So the main theorem guarantees that we there will be an MST containing (u,v).
- During the algorithm the disjoint sets data structure represents a set A of edges organized in different connected components.
- If (a,b) is the next not visited edge in the sorted order of the edges and a is in component C1 and b in component C2 the our cut is (C1, V/C1). Note that A respects this cut and (a,b) is the crossing edge of smallest weight for the cut. So the main theorem says we are safe in adding (a,b) to the cut.
- If (a,b) belong to the same component C1, then there is no cut for which (a,b) is crossing and respecting A. So (a,b) is not selected.

Conclusions and HW

We have seen two methods for computing the tree of smallest weight spanning a weighted undirected graph: Prim's Algorithm and Kruskal's algorithms.

HW4.8 (CLR 24.2-4) Suppose the weigths are integers in the range [1,..,|V|], can we make Kruskal and Prim algorithms faster?

HW4.9 (CLR 24.2-5) Suppose the weigths are integers in the range [1,..,W], for a constant W,
can we make Kruskal and Prim algorithms faster?