Algorithm Design & Analysis (CSE222)

Lecture-16

Outline

- Strong Component
- Minimum Spanning Tree

Minimum Spanning Tree

Let G = (V, E, w) be a weighted graph, w: E → R, i.e., w(e) is a real value.

Minimum spanning tree of G is defined by a tree T such that it minimizes the tree weight defined as follows.

$$w(T) := \sum_{e \in T} w(e)$$

Outline

- Minimum Spanning Tree
- Jarniks / Prims
- Kruskals

Unique Minimum Spanning Tree

Claim: If all the edge weights in a connected graph G are distinct then G has a unique minimum spanning tree.

Proof:

- Let G has two minimum spanning trees T and T'.
- Let e be the min weight edge in T \ T' and similarly e' in T' \ T. Let w(e) < w(e')
- The subgraph T' U {e} has a cycle that passes through e.
- Let e'' be some edge of the cycle that is not in T. It exists as T is a tree.
- Since, e ∈ T, and e'' ≠ e so e'' ∈ T' \ T. Hence, w(e) < w(e') ≤ w(e'').
- Consider T'' = T' + e \ e''. Then w(T'') = w(T') + w(e) - w(e'') < w(T')!
- But T and T' were the minimum spanning trees. So w(e'') = w(e).
- Even if T'' = T, we will have w(e') = w(e) so T and T' are the same tree.

Minimum Spanning Tree

Given a graph G, maintain an acyclic subgraph F, such it is also a subgraph of the MST of G.

At each step F encounters two types of edges,

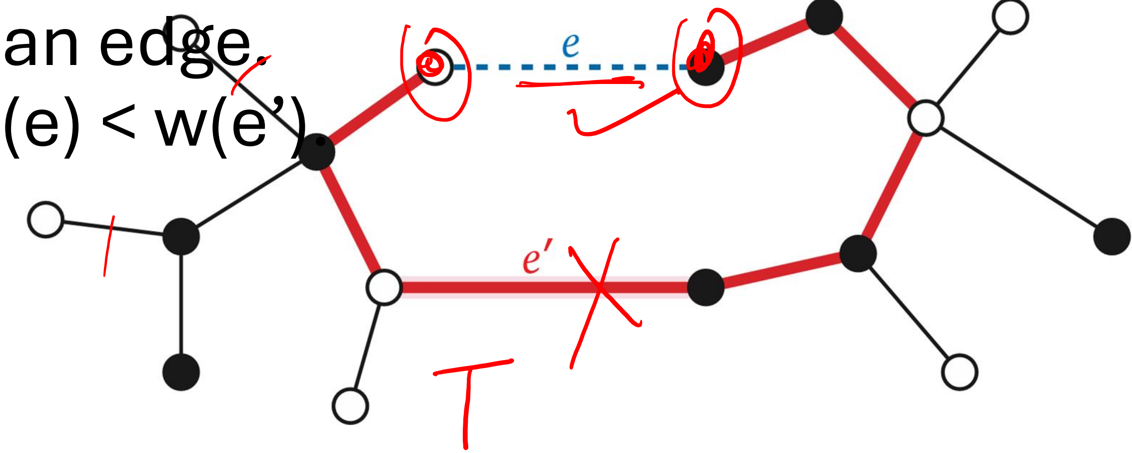
- Useless: if the edge not in F but both end points in the same component of F.
- Safe: minimum weighted edge with exactly an endpoint in a component of F.

Assume we have unique edge weights, ties are broken arbitrarily.

Minimum Spanning Tree

<u>Claim</u>: Minimum spanning tree of a graph G contains every safe edge.

<u>Proof</u>:
- For any subset S ⊆ V, the MST contains the min weight edge with an end in S
- Let e be the lightest edge with exactly one end in S.
- Let ~~T~~ be the minimum spanning tree that does ~~not~~ contain e.
- Remove an edge e’ from T such that the two endpoints of e are in two different components. Let e’ be such an edge.
- Consider T’ = T + e \ e’. By definition w(e) < w(e’).
- So, w(T’) < w(T)!

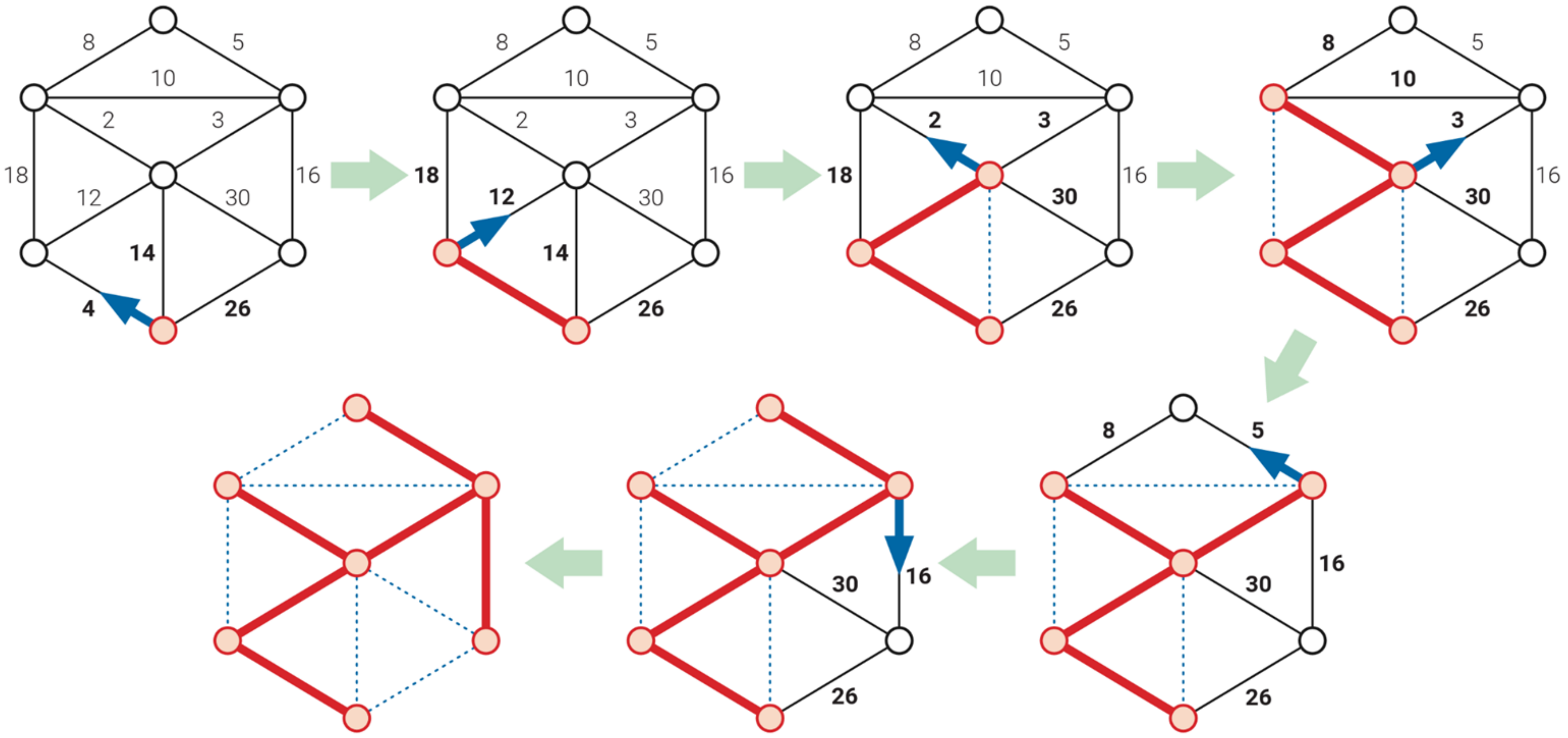


Outline

- Minimum Spanning Tree
- Jarniks / Prims
- Kruskals

Jarnik’s / Prim’s Algorithm

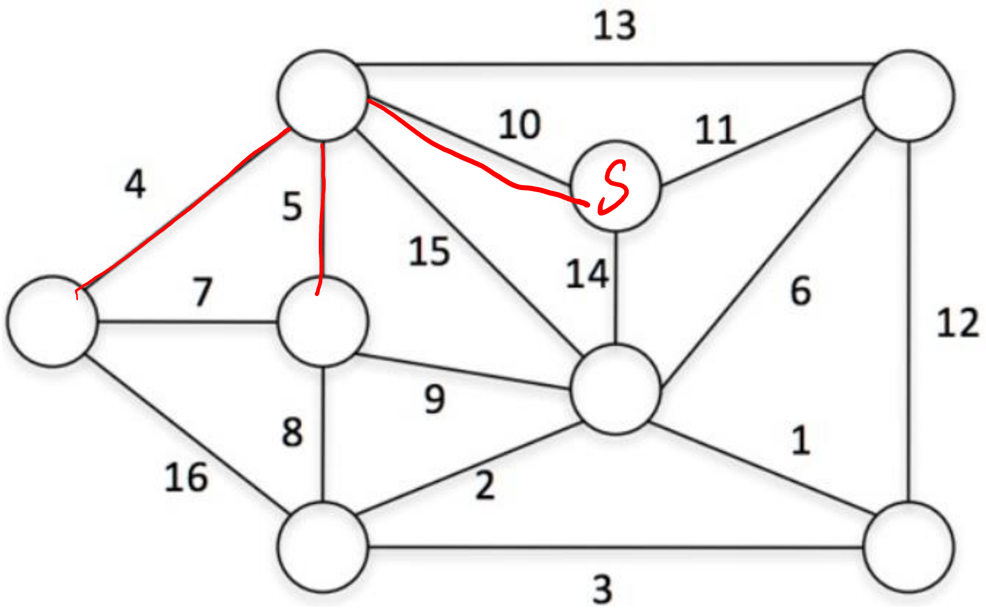
Start from a vertex and repeatedly add safe edge to T.



Jarnik's / Prim's Algorithm

- Maintain priority queue of edges adjacent to T.
- Dequeue minimum weight edge from the queue and check if both end points are in T.
- If not, then add the end point of the edge to T and update the priority queue.

Running time?

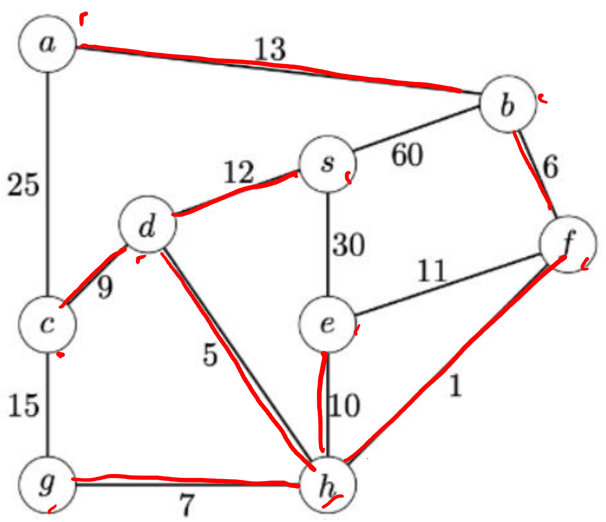


Outline

- Minimum Spanning Tree
- Jarniks / Prims
- Kruskals

Kruskal's

Add end point of edges to T in an increasing weight order, if the edge is safe.



Kruskal's Algorithm

Consider a data structure that supports following operations.

MakeSet(v): Create a set containing only the vertex v.
Find(v): Return unique identifier of the set containing v.
Union(u, v): Replace the sets containing u and v with their union.

```
KRUSKAL(V, E):
    sort E by increasing weight
    F ← (V, ∅)
    for each vertex v ∈ V
        MAKESET(v)
    for i ← 1 to |E|
        uv ← ith lightest edge in E
        if FIND(u) ≠ FIND(v)
            UNION(u, v)
            add uv to F
    return F
```

Kruskal's Algorithm

Consider a data structure that supports following operations.

2|E| Find operations
|V| - 1 Union operations
Amortized running time of Union is O(log |V|)

So, O(E + Vlog V)

```
KRUSKAL(V, E):
  sort E by increasing weight
  F ← (V, ∅)
  for each vertex v ∈ V
      MAKESET(v)
  for i ← 1 to |E|
      uv ← ith lightest edge in E
      if FIND(u) ≠ FIND(v)
          UNION(u, v)
          add uv to F
  return F
```

Reference

Slides

Jeff Erickson Chp-7

Algorithms Design by Kleinberg & Tardos - Chp 4.5