CS 341: Algorithms
Module 7: Graph Algorithms

Armin Jamshidpey, Eugene Zima

Based on lecture notes by many previous CS 341 instructors

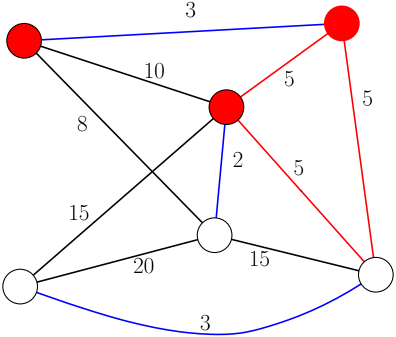
David R. Cheriton School of Computer Science, University of Waterloo

Spring 2019

Minimum spanning tree

- Problem: In an undirected graph with non-negative weights on edges, find a spanning tree of minimum total weight
- Motivation: cheapest interconnection (electrical circuit, computer network, highway system)
- Important subroutine for network optimization problems

Example of MST



Blue edges with any two of red edges give an MST

MST approaches

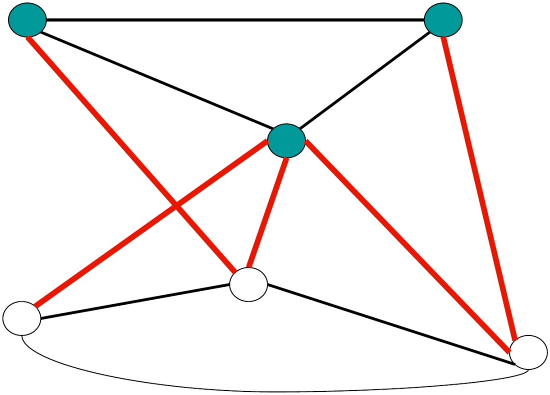
- Solution is not necessarily unique
- There are many algorithms
- General idea of greedy algorithms:

$A \leftarrow \emptyset$

```
while A is not a spanning tree
  find edge e of least weight with
  "certain properties"
  add e to A
```

Finding an edge to add

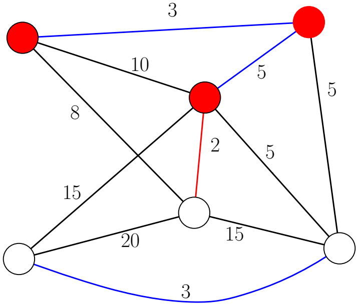
- Cut: partition of $V = (S, V - S)$
- Crossing edge: has endpoint in each set



Correctness of MST algorithm

Theorem 1

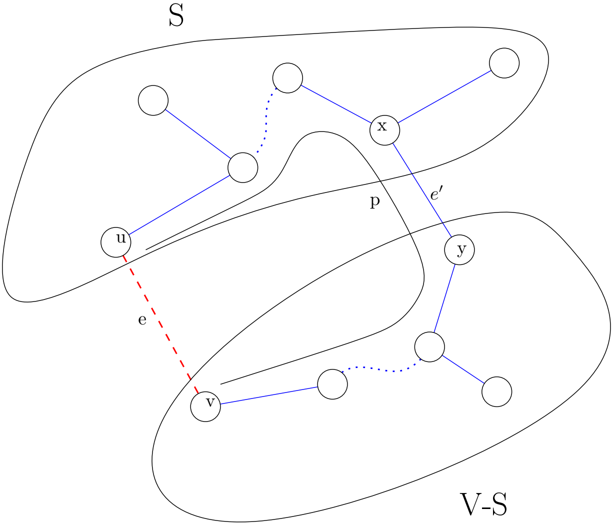
If A can be extended to a MST, and no edge in A crosses $(S, V - S)$, then the edge of minimum weight crossing this cut can be added to A.



Proof of Theorem 1

- Suppose we can find A, S, and edge e contradicting this. Then $A \bigcup \{e\}$ cannot be extended to a MST.
- Let T be a MST extending A.
- Adding e to T creates a unique cycle.

Proof of Theorem 1

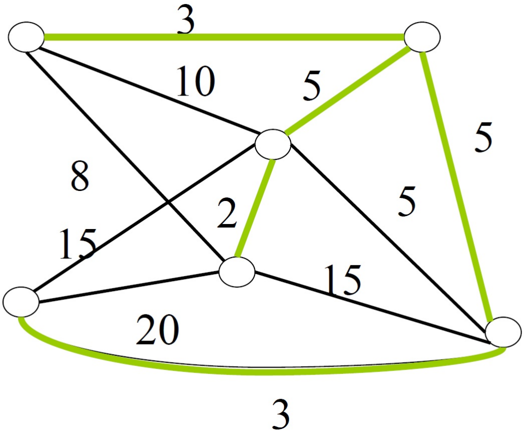


Proof of Theorem 1

- Adding e to T creates unique cycle
- Some other edge e' in this cycle also crosses the cut defined by S
- e' is not in A since it crosses the cut
- Weight of e' is at least weight of e
- $T \bigcup \{e\} - \{e'\}$ is also a spanning tree and must have weight no greater than T
- This is a MST extending $A \bigcup \{e\}$.

Kruskal's algorithm

- "certain property" $=$ can be added to A without forming a cycle



Correctness of Kruskal's algorithm

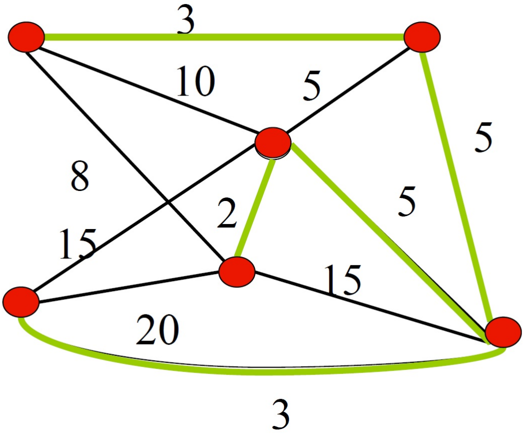
- During the algorithm, A is a forest of trees (stops when it is a single tree)
- What is the cut we can use in Theorem 1?
- $e = (u, v)$ is lightest edge that can be added without forming a cycle
- Let S be the vertices in the tree in A containing u
- v must be in $V - S$ (or e would form cycle)
- e must be lightest edge crossing this cut

Running time of Kruskal's algorithm

- Maintain components of A
- Presort edges, run through them in order
- Given $e = (u, v)$, it can be added to A if and only if u, v are in different components
- Adding e to A merges these components
- Need union-find data structure
- Loop executed $n - 1$ times
- Sequence of $n - 1$ unions and $2m$ finds can be done in $O(m \log n)$ time
- Algorithm takes $\Theta(m \log n)$ time

Prim's algorithm

- "certain property" = one endpoint shared with edge in A, one is not (i.e. leaves A)



Correctness of Prim's algorithm

- A is always a single tree (stops when it is a spanning tree)
- What is the cut we can use in Thm 1?
- $S =$ endpoints of edges in A (or starting vertex s if A is empty)
- Implementation is not so obvious: how do we find lightest edge crossing cut $(S, V - S)$?

Implementation of Prim's algorithm

- For each vertex v in $V - S$, maintain $near[v] = a \in S$ such that edge (v, a) is lightest edge from v to S
- Initially $near[v] = s$
- Add to S the vertex w minimizing weight of $(near[w], w)$ [takes $\Theta(n)$ time]
- When w added to S, update $near[v]$ if (w, v) is lighter than $(near[v], v)$ [takes $\Theta(n)$ time]
- Total running time $\Theta(n^2)$

Better implementation

- Keep vertices not in S in heap, ordered by near values
- Removing min or updating single near value takes takes $\Theta(\log n)$ time
- $n - 1$ removals, m updates
- Running time is $\Theta(m \log n)$
- Even better improvement uses Fibonacci heaps to get time of $\Theta(m + n \log n)$.

Single-source shortest path

- Think of weights as lengths of edges
- Given a weighted graph and source s, $\delta(s, v)$ = length of shortest $s - v$ path
- We wish to compute all $\delta(s, v)$ [and the corresponding paths]
- If edge weights are nonnegative, a greedy algorithm will work (Dijkstra's algorithm)
- General proof in book simplified here

Dijkstra's algorithm

- Looks similar to Prim's MST algorithm
- Start with source s in set S
- For vertices v not in S, maintain quantities
  - $\pi[v]$, a vertex in S
  - $d[v]$ which is $\delta(s, \pi[v]) + w(\pi[v], v)$
- Intuition: $d[v]$ is the length of the shortest path to v using vertices in S only (call this an S-internal path), and $\pi[v]$ is the last vertex in S on this path

Dijkstra's algorithm

- Initially $S \leftarrow \{s\}$, $d[s] \leftarrow 0$ and for all v not in S, if $(s, v) \in E$ then $\pi[v] \leftarrow s$ and $d[v] \leftarrow w(s, v)$ otherwise $\pi[v] \leftarrow nil$ and $d[v] \leftarrow \infty$
- To choose a vertex u to add to S, pick one with smallest d-value
- Update other d-values with

$$d[v] \leftarrow min\{d[v], d[u] + w(u, v)\}$$

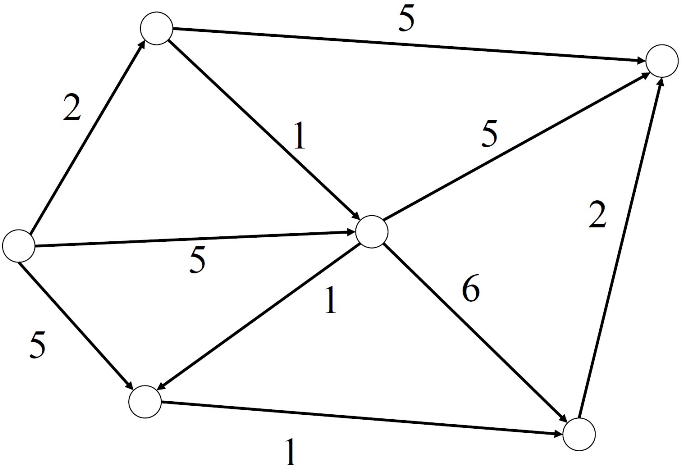
- We prove this works by induction on the size of S

Pseudocode for Dijkstra's algorithm

```
Initialize S, d, π
while S ≠ V
    u ← v ∉ S minimizing d[v]
    add u to S
    for v ∉ S
        d[v] ← min{d[v], d[u] + w(u, v)}
        (if d[v] changes, π[v] ← u)
```

- Running time of algorithm is $\Theta(n^2)$

Example of Dijkstra's alg'm



Proof of Dijkstra's algorithm

- Prove by induction on $|S|$ that
  1. For all $v \in S$, $d[v] = \delta(s, v)$
  2. For all $w \notin S$, $d[w]$= length of minimum S- internal $s - w$ path (so $d[w] \geq \delta(s, w)$) and $\pi[w] =$ last vertex on such a path
  3. For any $v \in S$, $w \notin S$, $d[v] \leq d[w]$
- Base case: $|S| = 1$
  - Since $d[s] = 0$ and for all v not in S, $\pi[v] = s$ and $d[v] = w(s, v)$, these are trivially true

Proving statement 1

- Assume statements true for $|S| = k - 1$
- When k^{th} vertex u chosen to be added to S, $d[u]$= length of a shortest S-internal path to u (by inductive hypothesis 2)
- Suppose $d[u] > \delta(s, u)$
- Choose any shortest $s - u$ path P
- It leaves S for the first time by some edge (x, y) and by ind.hyp. 1, $d[x] = \delta(s, x)$
- The segment of P from s to y has length $d[y]$ (by ind.hyp 2) so $d[y] \leq \delta(s, u) < d[u]$, contradicting the choice of u; so statement 1 is true

Proving statement 2

- Thus when k^{th} vertex u added to S, $d[u] = \delta(s, u)$, as required
- After u added, what do shortest S-internal path to $v \notin S$ look like?
- If one does not use u, then it must be the shortest $(S - \{u\})$-internal path to v, and this path has length $d[v] \leq d[u] + w(u, v)$, so the algorithm does not change anything
- If one uses u and (u, v) is the last edge, the path has length $\delta(s, u) + w(u, v)$, so the algorithm updates correctly

Proving statements 2 and 3

- If one uses u but some (y, v) is the last edge
  - $d[y] \leq d[u]$ (ind. hyp. 3)
  - u was just added, so the shortest $s - y$ path doesn't use u
  - Adding (y, v) gives a shortest S-internal path to v avoiding u
  - The algorithm does not change anything
- Thus statement 2 is proved
- Statement 3 follows because of the choice of u minimizing $d[u]$
- Where did we use non-negativity?