Minimum spanning trees (MST)

Def:

A **spanning tree** of a graph $G$ is an acyclic subset of edges of $G$ connecting all vertices in $G$.

A sub-**forest** of $G$ is an acyclic subset of edges of $G$. 
Minimum spanning trees (MST)

Def:
A **spanning tree** of a graph $G$ is an acyclic subset of edges of $G$ connecting all vertices in $G$.

A sub-**forest** of $G$ is an acyclic subset of edges of $G$.

Def:
Given is a weighted (undirected) graph $G=(V,E,w)$ where $w:E \rightarrow \text{Reals}$ defines a weight of every edge in $E$. A **minimum spanning tree** of $G$ is a spanning tree with the minimum total weight of edges.
Minimum spanning trees (MST)

Def:
A spanning tree of a graph $G$ is an acyclic subset of edges of $G$ connecting all vertices in $G$.

A sub-forest of $G$ is an acyclic subset of edges of $G$.

Def:
Given is a weighted (undirected) graph $G=(V,E,w)$ where $w:E\rightarrow \text{Reals}$ defines a weight of every edge in $E$. A minimum spanning tree of $G$ is a spanning tree with the minimum total weight of edges.
Minimum spanning trees (MST) - Kruskal

Kruskal \( G=(V,E,w) \)

1. Let \( T=\emptyset \)
2. Sort the edges in increasing order of weight
3. For edge \( e \) do
4. If \( T \cup e \) does not contain a cycle then
5. Add \( e \) to \( T \)
6. Return \( T \)

Running time:
- sort: \( O(m \cdot \log n) \)
- to check for a cycle in a graph:
  - can run DFS/BFS to find connected components
    - visited set \( V \) contains all vertices in \( G \)
    - check the edge
      - if it's in the component of \( e \)
        - add it
      - otherwise, cycle detected

Together: \( O(m \cdot \log n) \)
Minimum spanning trees (MST) - Kruskal

Lemma: Algo is correct.

1. Let $T = \emptyset$
2. Sort the edges in increasing order of weight
3. For edge $e$ do
   - If $T \cup e$ does not contain a cycle then
     - Add $e$ to $T$
4. Return $T$

Kruskal $G=(V,E,w)$

1. Let $T = \emptyset$
2. Sort the edges in increasing order of weight
3. For edge $e$ do
   - If $T \cup e$ does not contain a cycle then
     - Add $e$ to $T$
4. Return $T$
Kruskal (G=(V,E,w))
1. Let T=∅
2. Sort the edges in increasing order of weight
3. For edge e do
   4. If T ∪ e does not contain a cycle then
      5. Add e to T
5. Return T

Minimum spanning trees (MST) - Kruskal

Implementation?
Can do better than O(m(n+m))?
T is a forest

15
1
8
4
3
7
5
7
1
2
6

Kruskal (G=(V,E,w))
1. Let T=∅
2. Sort the edges in increasing order of weight
3. For edge e do
   4. If T ∪ e does not contain a cycle then
      5. Add e to T
6. Return T
Kruskal (G=(V,E,w))
1. Let T=∅
2. Sort the edges in increasing order of weight
3. For edge e do
   4. If T ∪ e does not contain a cycle then
      5. Add e to T
4. Return T

Minimum spanning trees (MST) - Kruskal

Implementation?

idea: make Kruskal its heart
Minimum spanning trees (MST) - Kruskal

Implementation?

Kruskal ( \( G=(V,E,w) \) )
1. Let \( T=\emptyset \)  init union-find datastructure
2. Sort the edges in increasing order of weight
3. For edge \( e \) do
4. If \( T \cup e \) does not contain a cycle then
5. Add \( e \) to \( T \), \( \text{union}(u,v) \)
6. Return \( T \)
Minimum spanning trees (MST) - Kruskal

Implementation?

**Kruskal (G=(V,E,w))**
1. Let T=∅
2. Sort the edges in increasing order of weight
3. For edge e do
4.   If T ∪ e does not contain a cycle then
5.      Add e to T
6. Return T
Minimum spanning trees (MST) - Kruskal

Implementation?

- **Union-Find** datastructure

Init \((V)\)

1. for every vertex \(v\) do
2. \(\text{boss}[v] = v\)
3. \(\text{size}[v] = 1\)
4. \(\text{set}[v] = \{v\}\)

Union \((u,v)\)

1. if \(\text{size}[\text{boss}[u]] > \text{size}[\text{boss}[v]]\) then
2. \(\text{set}[\text{boss}[u]] = \text{set}[\text{boss}[u]] \cup \text{set}[\text{boss}[v]]\)
3. \(\text{size}[\text{boss}[u]] += \text{size}[\text{boss}[v]]\)
4. for every \(z\) in \(\text{set}[\text{boss}[v]]\) do
5. \(\text{boss}[z] = \text{boss}[u]\)
6. else do steps 2.-5. with \(u,v\) switched
Lemma:
\[ k \text{ Unions take } O(k \log k) \text{ time} \]

1. if \( \text{size}[\text{boss}[u]] > \text{size}[\text{boss}[v]] \) then
2. set[\text{boss}[u]]=set[\text{boss}[u]] \text{ union set}[\text{boss}[v]]
3. size[\text{boss}[u]]+=size[\text{boss}[v]]
4. for every \( z \) in set[\text{boss}[v]] do
5. boss[\( z \)]=boss[\( u \)]
6. else do steps 2.-5. with \( u,v \) switched
Minimum spanning trees (MST) - Kruskal

Analysis of Union-Find

Lemma:
k Unions take $O(k \log k)$ time

Corollary:
The running time of Kruskal is: $O(|E| \log |E|) + O(|V| \log |V|)$
Minimum spanning trees (MST) - Prim
Minimum spanning trees (MST) - Prim

Prim (G=(V,E,w))
1. Let T=∅, H=∅
2. For every vertex v do
3.   cost[v]=∞, parent[v]=null
4. Let u be a vertex
5. Update (u)
6. For i=1 to n-1 do
7.   u=vertex from H of smallest cost (remove)
   • Add (u,parent[u]) to T
   • Update(u)
   • Return T

Update (u)
1. For every neighbor v of u
2. If cost[v]>w(u,v) then
3.   cost[v]=w(u,v), parent[v]=u
4. If v not in H then
5.   Add v to H
Lemma: Prim is correct.

Similar to Kruskal

Running time:
Single source shortest paths - Dijkstra

Input: \( G=(V,E,w) \) and a vertex \( s \) (\( w \) non-negative)

Output: shortest paths from \( s \) to every other vertex

Can use similar idea to Prim?
Single source shortest paths - Dijkstra

Input: $G=(V,E,w)$ and a vertex $s$ (w non-negative)

Output: shortest paths from $s$ to every other vertex

Can use similar idea to Prim?
Dijkstra (G=(V,E,w), s)
1. Let H=∅
2. For every vertex v do
3. dist[v]=∞
4. dist[s]=0
5. Update (s)
6. For i=1 to n-1 do
7. u=extract vertex from H of smallest cost
8. Update(u)
• Return dist[]

Update (u)
1. For every neighbor v of u
2. If dist[v]>dist[u]+w(u,v) then
3. dist[v]=dist[u]+w(u,v)
4. If v not in H then
5. Add v to H

Running time:
if option1:
steps 6-7 : O(n^2)
updater: O(deg) per vtx
⇒ O(m) overall

Option 1: represent H by keeping track of dist for every v by finding min 0(n)
Dijkstra ( G=(V,E,w), s )
1. Let H=∅
2. For every vertex v do
3. dist[v]=∞
4. dist[s]=0
5. Update (s)
6. For i=1 to n-1 do
7. u=extract vertex from H of smallest cost
8. Update(u)
   • Return dist[]

Update (u)
1. For every neighbor v of u
2. If dist[v]>dist[u]+w(u,v) then
3. dist[v]=dist[u]+w(u,v)
4. If v not in H then
5. Add v to H

Running time:
if H is a heap:
O(m log n)
because extract min and update are O(log n)
Single source shortest paths - Dijkstra

Lemma: Dijkstra is correct.

Proof:

Running time:
Single source shortest paths - Dijkstra

Lemma: Dijkstra is correct.

Note: Dijkstra does not work for graphs w. negative-weight edges.

Note 2: Does work for directed graphs (w. non-negative weights)

Running time:

let's try to shift weights up by smallest weight

Hamiltonian cycle - every vtx

does not work
All pairs shortest paths - Floyd-Warshall

Input: $G=(V,E,w)$, $w$ non-negative

Output: shortest paths between all pairs of vertices
All pairs shortest paths - Floyd-Warshall

Input: \( G=(V,E,w) \), \( w \) non-negative

Output: shortest paths between all pairs of vertices

Idea 1:

• Use Dijkstra from every vertex
All pairs shortest paths - Floyd-Warshall

Input: \( G=(V,E,w) \), \( w \) non-negative

Output: shortest paths between all pairs of vertices

Idea 1:

• Use Dijkstra from every vertex

Idea 2:

• How about dynamic programming?
All pairs shortest paths – Floyd-Warshall

Heart of the algorithm:

\[ S[i,j,k] = \text{the length of the shortest path from } i \text{ to } j \text{ using only vertices } \leq k \]
Heart of the algorithm:

\[ S[i,j,k] = \begin{cases} \text{the length of the} \\ \text{shortest path} \\ \text{from } i \text{ to } j \text{ using} \\ \text{only vertices } \leq k \end{cases} \]

How to compute \( S[i,j,k] \)?
Floyd-Warshall (G=(V,E,w))

1. For i=1 to |V| do
2.   For j=1 to |V| do
3.     S[i,j,0] = w(i,j)
4.   For k=1 to |V| do
5.       For i=1 to |V| do
6.         For j=1 to |V| do
7.           S[i,j,k] = min {
8.             S[i,j,k-1], S[i,k,k-1] + S[k,j,k-1] }
9.       Return ?

S[i,j,k] = \begin{cases} 
  w(i,j) & \text{if } k = 0 \\
  \min \{ S[i,j,k-1], S[i,k,k-1] + S[k,j,k-1] \} & \text{if } k > 0
\end{cases}
Single source shortest paths – Bellman-Ford

Input:
directed $G=(V,E,w)$ and a vertex $s$

Output:
• FALSE if exists reachable negative-weight cycle,
• distance to every vertex, otherwise.
Bellman-Ford (G=(V,E,w), s)

1. For every vertex v
2. \(d[v] = \infty\)
3. \(d[s] = 0\)
4. For \(i = 1\) to \(|V| - 1\) do
5. For every edge \((u,v)\) in \(E\) do
6. If \(d[v] > d[u] + w(u,v)\) then
7. \(d[v] = d[u] + w(u,v)\)
8. For every edge \((u,v)\) in \(E\) do
9. If \(d[v] > d[u] + w(u,v)\) then
10. Return NEGATIVE CYCLE
11. Return \(d[]\)
Bellman-Ford (G=(V,E,w), s)

1. For every vertex v
2. \( d[v] = \infty \)
3. \( d[s] = 0 \)
4. For \( i=1 \) to \(|V| - 1\) do
5. For every edge \((u,v)\) in E do
6. If \( d[v] > d[u] + w(u,v) \) then
7. \( d[v] = d[u] + w(u,v) \)
8. For every edge \((u,v)\) in E do
9. If \( d[v] > d[u] + w(u,v) \) then
10. Return NEGATIVE CYCLE
11. Return \( d[] \)

Lemma: Bellman-Ford is correct.

Running time: