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

Minimum spanning trees (MST) - Kruskal
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

Lemma: Algo is correct.

By contradiction, suppose we have OUR solution and an OPT solution and OUR ≠ OPT Let e be the 1st edge according to our order that OUR-OPT.

Let f be an edge on OPT-OUR on a cycle in OPT+e. Then, w(f) ≥ w(e). If w(f) > w(e), then OPT-f+e is more optimal (lower weight) than OPT♭. If w(f) = w(e), then let OPT♭ = OPT-f+e

and continue the argument with OPT♭ that agrees with us on more edges. Keep doing this until OUR = OPT♭.
Minimum spanning trees (MST) - Kruskal

Implementation?

\[ O(m \log n) \] see next slides

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 \( T \)
**Minimum spanning trees (MST) - Kruskal**

**Implementation?**

- **Union-Find** datastructure

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

**Union (u,v)**
1. if size[boss[u]]>size[boss[v]] then
2.  set[boss[u]]=set[boss[u]] union set[boss[v]]
3.  size[boss[u]]+=size[boss[v]]
4.  for every z in set[boss[v]] do
5.  boss[z]=boss[u]
6. else do steps 2.-5. with u,v switched
KruskalUnionFind ( G=(V,E,w) )
1. Let T=∅
2. Sort the edges in increasing order of weight
3. Init(V)
4. For edge e=(u,v) do
   5. If Find(u)! = Find(v) then
      6. Add e to T
      7. Union(u,v)
7. 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 \( \text{size}[	ext{boss}[u]] > \text{size}[	ext{boss}[v]] \) then
2. \( \text{set}[	ext{boss}[u]] = \text{set}[	ext{boss}[u]] \cup \text{set}[	ext{boss}[v]] \)
3. \( \text{size}[	ext{boss}[u]] += \text{size}[	ext{boss}[v]] \)
4. for every z in \( \text{set}[	ext{boss}[v]] \) do
5. \( \text{boss}[z] = \text{boss}[u] \)
6. else do steps 2.-5. with u,v switched
Minimum spanning trees (MST) - Kruskal

Analysis of Union-Find

Lemma:

$n$ Unions take $O(n \log n)$ time

Union $(u,v)$

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. $\text{size}[\text{boss}[u]] += \text{size}[\text{boss}[v]]$
4. for every $z$ in set[\text{boss}[v]] do
5. $\text{boss}[z] = \text{boss}[u]$
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

Prim: grow your own spanning tree

Will keep only the lowest weight edge from the tree to a neighboring vertex

Growing spanning tree edges out of it → take the smallest edge
Prim ( G=(V,E,w) )
1. Let T=∅, H=∅
2. For every vertex v do
3. \( \text{cost}[v]=\infty, \text{parent}[v]=\text{null} \)
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,\text{parent}[u]) to T
   • Update(u)
   • Return T

Update (u)
1. For every neighbor v of u
2. If \( \text{cost}[v]>w(u,v) \) then
3. \( \text{cost}[v]=w(u,v), \text{parent}[v]=u \)
4. If v not in H then
5. Add v to H
Minimum spanning trees (MST) - Prim

Lemma: Prim is correct.

Similar to Kruskal

Running time:

$O(m \log n)$ if $H$ is a heap, or $O(n^2)$ if $H$ is an array (or the costs sit inside the vertex objects)
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 \(H=\emptyset\)
2. For every vertex \(v\) do
3. \(\text{dist}[v]=\infty\)
4. \(\text{dist}[s]=0\)
5. Update \((s)\)
6. For \(i=1\) to \(n-1\) do
7. \(u=\text{extract vertex from } H\) of smallest cost
8. Update \((u)\)
   - Return \(\text{dist}[]\)

**Update \((u)\)**

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

**Run time:**

- \(O(m \log n)\) if using a heap
- \(O(n^2)\) if not a heap
Single source shortest paths - Dijkstra

Lemma: Dijkstra is correct.

Running time:

If negative weight edges:
Lemma: Dijkstra is correct.

by induction on the # of finalized vertices,
we show that the current tree connecting s to the
finalized vertices is a shortest path tree
Base case: distance to s is 0, it is OK
and its neighbor
dist initialization property
and the yellow vertices
hold dist that is the
shortest possible
using edges of the tree
plus, at the end, one
extra edge
yellow: vertices reachable from Tcurrent by a single edge
find the min of yellow vertices
a is the to-be-finalized vertex
we know, by \(H\): \(\text{dist}(u) \leq \text{dist}(x)\)
then length of pink \(\geq \text{dist}(x) + \) the rest of pink \(\geq \text{dist}(u)\)
All pairs shortest paths - Floyd-Warshall

Input: \( G=(V,E,w) \), \( w \) non-negative
Output: shortest paths between all pairs of vertices

Idea 1: \( n \times \text{Dijkstra} \)
   running time: \( O(n \cdot m \cdot \log n) \)
   or \( O(n^3) \)

Idea 2: dyn. prog.

\[ S[i,j,k] = \text{the length of the shortest path from } i \text{ to } j, \]
\[ \text{when the path can use intermediate vertices with labels } 1, 2, \ldots, k \]
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:

\[ S[i,j,k] = \begin{cases} 
  w(i,j) & \text{if } k = 0 \text{ and } (i,j) \in E \\
  \infty & \text{if } k = 0 \text{ and } (i,j) \notin E \\
  \min\{S[i,j,k-1], S[i,k,k-1] + S[k,j,k-1]\} & \text{if } k > 0 
\end{cases} \]

- **Possibility 1:** the shortest path from \(i\) to \(j\) does not use vertex \(k\).
- **Possibility 2:** using \(k\).
All pairs shortest paths - Floyd-Warshall

Heart of the algorithm:

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

How to compute \( S[i,j,k] \)?

\[ S[i,j,k] = \]
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 { S[i,j,k-1], S[i,k,k-1] + S[k,j,k-1] }
8.   S[i,j,k] = min { S[i,j,k-1], S[i,k,k-1] + S[k,j,k-1] } if k > 0
10. Return S[i,j,n] ∀i,j {1,...,n}

S[i,j,k] =

\[ S[i,j,k] = \begin{cases} 
  w(i,j) & \text{if } k = 0 \\
  \min \{ S[i,j,k-1], S[i,k,k-1] + S[k,j,k-1] \} & \text{if } k > 0
\end{cases} \]

Running time: O(n^3)

works for directed graphs
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.

Note:
it is not OK to increase the weights by $|\min\text{weights}|$ and then run Dijkstra.
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:

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[\] \)
Single source shortest paths - Bellman-Ford

Lemma: Bellman-Ford is correct.

Running time: $O(nm)$

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 $d[]$