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)**
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.
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

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

Lemma: Algo is correct.

Let OPT be a minimum spanning tree
Let BLUE be the output of Kruskal.

If e be a blue edge in OPT
then OPT ∪ e contains a cycle,
not all edges are blue (all but e are green)

Let f be an edge on the cycle
that is green but not blue
by contradiction, assume OPT ≠ BLUE
choices: - w(f) > w(e)
        - w(f) < w(e)
        - w(f) = w(e)

cannot be sec. OPT - f, e
improves optimum!

if -w(f) < w(e)
cannot be, Kruskal would add f instead of e

then OPT ∪ e another optimum.

Kruseral (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
6. Return T

Minimum spanning trees (MST) - Kruskal

Implementation?

idea 1: checking for cycles: e=(u,v)
    use BFS or DFS to check if u and v are already
    connected
    if yes, e would form a cycle
    if not, add e to T

then, steps 4-5: O(n) b/c |T|≤n-1

#iterations of loop on line 3: m
therefore: O(m·n)
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

Implementation?

- Every vertex knows its hec (label)
- At first, hec label = vertex label
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}  \[\text{Find}(u)\]
   \[\text{return boss}[u]\]

**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. \[\text{boss}[z]=\text{boss}[u]\]
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

where: \( k = n-1 \)

when changing \( z \)'s boss, the size of the set \( z \) is at least doubled

\[ \Rightarrow \] steps 4-5 take \( O(n \log n) \) across all unions (b.c. \( O(\log n) \) steps)

Union \((u,v)\)
1. if \( \text{size}\[\text{boss}\[u]\] > \text{size}\[\text{boss}\[v]\] \) then
2. set[\text{boss}\[u\]] = set[\text{boss}\[u\]] 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. \( \text{boss}[z] = \text{boss}[u] \)
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: $O(|E| \log |E|) + O(|V| \log |V|)$

$= O(m \log n)$
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

Array representation for H:
runtime: O(n^2)

Heap representation of H:
update (need to keep a pointer to the location in the heap for every vertex)
runtime: O(m log n)
Lemma: Prim is correct.

Suppose that blue sp. tree is the output of Prim
let green sp. tree be an OPT MST

By contradiction, assume that OPT ≠ Prim ST
let e be the edge in Prim but not OPT (blue but not green) that was added to Prim ST. First
then, OPT + e contains a cycle. Let f be the edge on the cycle ≠ adj to the Prim tree set. Adding e
obervation: \( w(f) < w(e) \) (otherwise Prim would take f instead of e)

What if \( w(f) > w(e) \): then OPT + f + e is a sp. tree with weight \(<\) weight(OPT)
What if \( w(f) = w(e) \): then OPT + f + e is another opt. sp. tree.

Running time:
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. \( \text{dist}[v]=\infty \)
4. \( \text{dist}[s]=0 \)
5. Update (s)
6. For i=1 to n-1 do
7. \( u=\text{extract vertex from } H \text{ of smallest cost} \)
8. Update(u)
   • Return 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
Lemma: Dijkstra is correct.

We will show that after each update:
- the vertices reached by a Dijkstra tree edge have their shortest distance from $s$ computed correctly,
- the other vertices store distance = shortest distance using Dijkstra tree + one more edge

we will show this by induction on the #vertices in the Dijkstra tree (i.e. on $i$)

**BASE CASE:** \[ i = 0 \]
update(s): \[ \checkmark \] (IH)

**IND. CASE:** assuming the above holds for $i$, we will show it for $i+1$

let $v$ be the vertex finalized in the $(i+1)$st iteration

Running time:

like Prim, i.e. $O(m \log n)$ if using heaps, $O(n^2)$ if using an array

by contradiction, suppose $\exists$ a shorter path from $s$ to $v$

let $x$ be the first vertex on the path after leaving the Dijkstra tree

then $\text{dist}[x] \geq \text{dist}[v]$ bec. $v$ is min in $H$

then white path of distance $\geq \text{dist}[x]$ bec edges nonneg.

$\Rightarrow$ white is not shorter than $\text{dist}[x]$
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

running time: $O(mn \log n)$ or $O(n^3)$
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:

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

Number the vertices 1, ..., n (in circles)

If \( k \) is on the path, it is there at most once, bec. weights are nonnegative (i.e.: no cycles)

\[ S[i,j,0] = \begin{cases} w(i,j) & \text{if edge } (i,j) \text{ exists} \\ \infty & \text{otherwise} \end{cases} \]

\[ S[i,j,k] = \begin{cases} S[i,j,k-1] & \text{if not using} \\ \text{vertex } k \\ S[k,i,k-1] + S[k,j,k-1] & \text{if using} \\ \text{vertex } k \end{cases} \]
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. \( S[i,j,k] = \min \{ \)
8. \( S[i,j,k-1], \)
9. \( S[i,k,k-1]+S[k,j,k-1] \) \}
10. Return \( S[i,j,n] \) for all \( i,j \)

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