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.
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 \( T \)
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 U e does not contain a cycle then
5. Add e to T
6. Return T

Lemma: Algo is correct.

by contradiction, suppose OPT is a min-span-tree, BLUE is the output of Kruskal and BLUE ≠ OPT

let e be a blue edge that is not in OPT

then OPT+e contains a cycle (the cycle is not all blue because it were, then BLUE would contain a cycle → contradiction w. line 4)
⇒ there is a green edge on the cycle that is not blue ⇒ let it be f

choices:
- w(e)>w(f) cannot happen because Kruskal would add f to the blue sp. tree
- w(e)<w(f) cannot happen because OPT-f+e results in a sp. tree with cost < OPT
- w(e)=w(f) can happen, then create new OPT

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 U e does not contain a cycle then
5. Add e to T
6. Return T
Minimum spanning trees (MST) - Kruskal

Implementation?

at the beg. , every vertex (a tree on its own),
gets an identifier
with every edge that does not create a cycle,
we union the corresponding sets of vertices

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}

**Find (v):**
return boss[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
Minimum spanning trees (MST) - Kruskal

Analysis of Union-Find

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

1. let's look at an element \( u \): if a member of set of size \( l \)
   and its boss is changed in a union with
   set of size \( p \), then the size of the new
   set is \( \geq 2l \) \( \text{sec. } l \leq p \)
2. every vertex changes its boss at most \( \log n \) times, i.e.

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
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 ( 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.

Let blue be the output of Prim.
Let green be an optimum MST.

Partial Prim tree

Let e be the first edge added to the Prim MST and it is not green.
Let f be the green edge adjacent to an edge of the Prim's partial tree.

We know: \( w(f) > w(e) \) (otherwise, Prim would choose f over e).

Possible:
1. \( w(f) > w(e) \), then we can improve the OPT by replacing f with e.
2. \( w(f) = w(e) \), then modify OPT by removing f and adding e.

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?
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 \((u)\)
8. 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 \(H\)
Lemma: Dijkstra is correct.

We will show that after each update, the following holds for the current Dijkstra tree:

1) The distances to vertices on the tree are computed correctly (i.e., they are the shortest possible distances)
2) Other distances equal to the shortest distance using tree edges plus exactly one other edge

By induction on the finalized vertices (i.e., on i from line 6)

**Base Case:** $i=0$ \(\checkmark\)

**Inductive Case:** $i>0$, assuming (*) holds for $i-1$

Suppose $x$ is a vertex and $\text{dist}(x)$ is the shortest distance to $x$.

**Suppose a shorter path exists to $v$** (< $\text{dist}(v)$)

**$\Rightarrow$ Green**

Let $v$ be the first unvisited vertex on the green path, then $\text{dist}(x) \geq \text{dist}(v)$

Then the green path at length $\text{dist}(v)$ is in the graph.

Running time: same as Prim's algorithm, i.e., $O(n^2)$ or $O(m \log n)$ depending on the implementation.
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 $O(n^3)$ or $O(mn + n^2 \log n)$
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:

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

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

\[ S[i,j,k] = \min \{ S[i,j,k-1], S[i,k,k-1] + S[k,j,k-1] \} \]

- if the shortest path does not use vertex k
- if using k
All pairs shortest paths - Floyd-Warshall

Heart of the algorithm:

\[ S[i,j,k] = \]

the length of the shortest path from \( i \) to \( j \) 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. \( S[i,j,k] = \quad \forall i,j \)
9. 10. 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.
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: \(O(mn)\)