Lecture 24:
• Minimum Spanning Trees (wrap up)
• Bellman-Ford Shortest Path Algorithm

Announcements
• Project Step 3 out (due Thurs, 12/1)
• HW-6 out (due Thur, 12/1)

Minimum Spanning Trees

A "spanning tree" of an undirected graph $G = (V, E)$ is:
• a subset of edges $T \subseteq E$ forming a tree (e.g., no cycles)
• a path exists between each pair of nodes $u, v \in V$ ... vs a “forest”

A minimum spanning tree $T \subseteq E$ has the smallest edge-weight sum:
$$\sum_{e \in T} w(e)$$

Note: minimum of all possible spanning trees (could be ties)

Check In: What is the MST in this graph?
**Prim’s MST Algorithm**

**Basic Idea:**
- Build up the tree $T$ by selecting next smallest edge to add
- Maintains an “exclusion” set $X$
- Repeats until there are no more edges to add ($X$ is $V$)

**Algorithm:** Prim’s Algorithm

**Input:** Undirected, connected graph $G = (V, E)$ and weight function $w$.

**Result:** Tree $T$ containing each $v \in V$.

```
1 begin
2     X ← {s}   // pick an arbitrary vertex $s$
3     T ← ∅
4     while there is an edge $(u, v)$ with $u \in X$ and $v \not\in X$ do
5         $(u', v')$ ← such an edge with minimum cost $w(u, v)$
6         $X ← X \cup \{v'\}$
7         $T ← T \cup \{(u', v')\}$
```

**Notes:**
- Each pass augments $X$ by one new vertex that connects $X$ and $V - X$
- Edge chosen minizies the edge weight $w(u, v)$
- Note that the algorithm works regardless of starting vertex
- Prim’s algorithm works for graphs with negative weights

**Check In:** Trace the algorithm for the example graph

```
0 --- 1
|    |
|    |
4 --- 3
```

0 1
2 3
1
4
5
2 3
### Prim’s MST Algorithm

**Analysis:** $O(VE)$ for similar reasons as Dijkstra’s algorithm

- loop iterates $|V| - 1$ times
- each iteration search $O(E)$ edges

**Check In:** Can we just use Prim’s Algorithm for finding shortest paths?

- no! ... consider this graph:

```
  s   t   v
  \
  10  25  \
  \
  u
  20
```

- Dijkstra’s “greedily” selects edges to minimize path costs from $s$
- Prim’s “greedily” selects edges to minimize edge costs on every path

---

### Kruskal’s MST Algorithm

**Basic Idea:**

- Sort edges by weight (lowest to highest)
- Add edges to the tree in ascending order, avoiding cycles

**Algorithm:** Kruskal’s Algorithm

**Input:** Undirected, connected graph $G = (V, E)$ and weight function $w$.

**Result:** Tree $T$ containing each $v \in V$.

1. **begin**
2. $T \leftarrow \emptyset$
3. sort edges of $E$ by weight \quad \text{// e.g., via MergeSort}
4. for each $e \in E$ via sort order do
5. \quad if $T \cup \{e\}$ is acyclic then
6. \qquad $T \leftarrow T \cup \{e\}$

---

S. Bowers  CPSC 450, Fall 2022
Kruskal’s MST Algorithm

Check In: Trace the algorithm for the following graph

Iterating through the edges (in ascending order of weight)
1. add edge (1, 4) ... weight of 1
2. add edge (0, 2) ... weight of 2
3. add edge (2, 1) ... weight of 3
4. skip edge (0, 1) ... weight of 4
5. add edge (1, 5) ... weight of 5

Speeding up MST Algorithms

Use a Heap in Prim’s Algorithm ... cost becomes $O((V + E) \log V)$
- Algorithm is a bit more involved

Use Union-Find (Disjoint-Set) with Kruskal ... cost is also $O((V + E) \log V)$
- Still relatively straightforward

Note: For HW-6, only do straightforward implementations
- i.e., don’t need to use heaps or disjoint-set data structures
- Kruskal’s doesn’t perform well due to acyclic calls ...
Bellman-Ford Algorithm (and Negative Edge Weights)

**Basic Idea:**

- single-source shortest path algorithm (like Dijkstra's Algorithm)
- handles negative edge weights (unlike Dijkstra's Algorithm)

**Note on Negative Cycles:**

Check In: What is $\delta(s, v)$? ... $s (\rightarrow t \rightarrow u)^n \rightarrow t \rightarrow v$ gives $-\infty$

Bellman-Ford finds shortest paths for negative edge weights ...

Bellman-Ford Algorithm

**Algorithm:** Bellman-Ford Algorithm

**Input:** Weighted directed graph $G = (V, E)$ and source vertex $s \in V$

**Result:** Path weights $\text{dist}(s, v)$ for each $v \in V$ and negative cycle test

```
begin
for $v \in V$ do
    $\text{dist}(s, v) = \infty$
    $\text{dist}(s, s) = 0$
for $i = 1$ to $|V| - 1$ do
    for each $(u, v) \in E$ do
        if $\text{dist}(s, v) > \text{dist}(s, u) + w(u, v)$ then
            $\text{dist}(s, v) = \text{dist}(s, u) + w(u, v)$
for each edge $(u, v) \in E$
    if $\text{dist}(s, v) > \text{dist}(s, u) + w(u, v)$ then
        return false
return true
```

S. Bowers CPSC 450, Fall 2022 10
Bellman-Ford Algorithm (cont)

Analysis:
• $|V| - 1$ iterations, each iteration processes $|E|$ edges
• Thus, overall cost is $O(VE)$

Check In: Trace Bellman-Ford over the following graph with $s = 0$

Bellman-Ford Algorithm (cont)

1. Recall: a shortest path $p$ from $s$ to $v$ has $w(p) = \delta(s, v)$

2. Path relaxation
   • Updating $dist(s, v)$ when $dist(s, v) > dist(s, u) + w(u, v)$

3. The path relaxation property ... can prove via induction on $k$
   • If $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from $s = v_0$ to $v_k$
   • ... and the edges of $p$ are relaxed in order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$
   • ... then $dist(s, v_k) = \delta(s, v_k)$
   • The property holds regardless of other relaxations that are done
4. Consider any vertex $v$ reachable from $s$
   - Let $p = (v_0, v_1, \ldots, v_k)$ for $v_0 = s, v_k = v$ be a shortest path from $s$ to $v$
   - ... note: a shortest path is always a simple path (unique vertices)
   - ... so $p$ contains at most $|V|$ vertices and $|V| - 1$ edges, i.e., $k \leq |V| - 1$
   - Each of the $|V| - 1$ iterations relax all $|E|$ edges
   - So in the $i$-th iteration, for $i = 1, 2, \ldots, k$, edge $(v_{i-1}, v_i)$ is relaxed
   - Which satisfies path relaxation property for $p$, giving $\text{dist}(s, v) = \delta(s, v)$

Can think of the algorithm as “growing” shortest paths of length $i$:
   - when $i = 1$, all shortest paths 1 edge from $s$
   - when $i = 2$, all shortest paths 2 edges from $s$
   - and so on to the max length of a possible shortest path