Lecture 23:
• Dijkstra’s Algorithm
• Minimum Spanning Trees

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

Weighted Graphs

Check In: What are the shortest path weights starting at 0?
• $\delta(0, 0) = 0$ ... all vertices have trivial paths to themselves
• $\delta(0, 1) = 1$ ... path $0 \rightarrow 1$
• $\delta(0, 3) = 3$ ... path $0 \rightarrow 1 \rightarrow 3$
• $\delta(0, 2) = 6$ ... path $0 \rightarrow 1 \rightarrow 3 \rightarrow 2$
Dijkstra's Algorithm: Single-Source Shortest Paths

Basic Idea:

• Similar to BFS (but maintain an “exclusion” set \( X \))
• But at each step, add a frontier edge with minimal edge weight

Algorithm: Dijkstra’s Algorithm

Input: Directed graph \( G = (V, E) \), weight function \( w \), and source \( s \in V \).

Result: Shortest path weights \( \text{dist}(s, v) \) for each \( v \in V \).

\begin{verbatim}
1 begin
2 \( X \leftarrow \{s\} \)
3 for \( v \in V \) do
4 \( \text{dist}(s, v) = \infty \)
5 \( \text{dist}(s, s) = 0 \)
6 while there is an edge \((u, v)\) with \( u \in X \) and \( v \notin X \) do
7 \( (u', v') \leftarrow \) such an edge minimizing \( \text{dist}(s, u) + w(u, v) \)
8 \( X \leftarrow X \cup \{v'\} \)
9 \( \text{dist}(s, v') \leftarrow \text{dist}(s, u') + w(u', v') \)
\end{verbatim}

Notes

• Each pass augments \( X \) by one new vertex that connects \( X \) and \( V - X \)
• Specifically, by choosing a \((u, v)\) that minimizes the “Dijkstra score”:

\[
\text{dist}(s, u) + w(u, v)
\]

Check In: Trace the algorithm for the following graph for \( s = 0 \)
Dijkstra’s Algorithm

A “straightforward” implementation:

Store $X$ as a $|V|$-element boolean array excluded

- e.g., $\text{excluded}[v] = \text{true}$ implies $v \in X$

On each pass, go through each edge $e \in E$: .. can pre-compute $E$

- and check if $e$ goes from $X$ to $V - X$
- keep track of the $e$ that minimizes the Dijkstra score
- optional: remove $e$ from pre-computed list

Cost:

- each pass checks $O(E)$ edges
- since each pass adds one vertex to $X$, there are $O(V)$ passes
- gives a total cost of $O(VE)$

Negative Edge Weights

With non-negative weighted edges

- adding an edge to a path cannot decrease total path cost
- and thus path costs are monotonically increasing

With negative weighted edges

- path costs can decrease as (negative) edges are added
- but Dijkstra’s algorithm relies on monotonically increasing path weights

Example: What are the shortest paths? What does Dijkstra’s algo compute?
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 $u, v \in V$...

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?

```
0 1
2 3

1
4
3
```

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 $v \notin X$ do
5    ($u'$, $v'$) ← such an edge with minimum cost $w(u, v)$
6    X ← X ∪ {$v'$}
7    T ← T ∪ {($u'$, $v'$)}
```

S. Bowers CPSC 450, Fall 2022 7

S. Bowers CPSC 450, Fall 2022 8
Prim’s MST Algorithm

Notes:

• Each pass augments $X$ by one new vertex that connects $X$ and $V - X$
• Edge chosen minimizes 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

![Graph](attachment:image.png)

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:

![Graph](attachment:image.png)

• 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 // e.g., via MergeSort
4. for each \(e \in E\) via sort order do
5. if \(T \cup \{e\}\) is acyclic then
6. \(T \leftarrow T \cup \{e\}\)

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 ...