Recall Min. Spanning Tree (MST) problem.

Last day: Kruskal’s algorithm
today: a different greedy algorithm

**Prim’s Algorithm.**

Grow one connected component in a greedy fashion (i.e. by adding min. weight edge leaving the component).

\[ C \]

Choose min. weight edge leaving \( C \)

\[ C = \text{set of vertices reached by } T \text{ so far} \]

initialize \( C \leftarrow \emptyset , T \leftarrow \emptyset \)

while \( C \neq V \)

find min. weight edge \( e = (u, v) \) from \( u \in C \to v \in V \setminus C \)

\[ T \leftarrow T \cup e \]

\[ C \leftarrow C \cup \{v\} \]

end

**Correctness.** The exact same exchange argument works. And in fact, we could prove some lemma that gives correctness of both algs. (see text).
Prim - implementation.
In general, we need to find min. weight edge leaving C, the connected component of T.

Priority Queue data structure
Maintain set of weighted elements (in our case, edges leaving C)
Operations
  - Find and delete min weight element = min. key
  - Insert
  - Delete

Can be implemented as a heap (see CS 240 textbook) at $O(\log k)$ time per operation, $k = \#\text{elements}$

In our case
\[
\delta(C) = \text{edges leaving } C
\]
\[
k = \#\text{elements} \leq m
\]

Changes to $\delta(C)$ when $v$ is added to $C$:
  - edges from $C$ to $v$ leave $\delta(C)$
  - other edges adjacent to $v$ enter $\delta(C)$

We can find these edges by going through $v$'s adjacency list.
Each edge enters $\delta(C)$ once and leaves it once
Priority Queue operations

\[ n-1 \quad \text{find min} \]
\[ m \quad \text{insert} \]
\[ m \quad \text{delete} \]

Total cost

\[ O(n \log m + m \log m) = O(m \log n) \]

It is slightly more efficient to keep a priority queue of vertices \( V \setminus C \) with weight(v) = min weight edge from C to v

We will see this approach for next algorithm.

Then

size of PQ = n

update is key-change \( O(\log n) \)

still gives \( O(m \log n) \) - total.
Shortest Paths in Edge Weighted Graphs

Recall that BFS from \( \text{v} \) finds shortest paths from \( \text{v} \) in unweighted undirected graphs.

General input: directed or undirected graph with weights on edges.

\[ \begin{array}{c}
\text{E} & \text{A} \\
\text{B} & \text{1} & \text{3} & \text{5} \\
\text{C} & \text{2} & \text{1} & \text{2} & \text{D} \\
\end{array} \]

Shortest path A to D is \( \text{ABD} \), weight 5.
A to E: \( \text{ABE} \), weight 4.

Note: Does a MST always contain the shortest paths?
No, e.g. above, shortest path E to D is edge (E, D), weight 2.

We will study several shortest path algorithms.
Today: Dijkstra’s algorithm.
Dijkstra's Algorithm 1959

Input: digraph $G = (V, E)$, $w : E \rightarrow \mathbb{R}_{\geq 0}$, $s \in V$

Output: shortest path from $s$ to every other vertex $v$.

Idea: Grow tree of shortest paths starting from $s$.

General step: have tree of shortest paths to all vertices in set $B$

Initially $B = \{s\}$

Choose edge $(x, y) \in E \setminus B$, $y \notin B$

to minimize $d(s, x) + w(x, y)$

distance from $s$ to $x$ - known

Call this min. $d$

$d(s, y) \leftarrow d$

add $(x, y)$ to tree (Parent(y) $\leftarrow x$)

Note similarity & differences to Prim's MST alg.
This is greedy in the sense that we always add the vertex with next min distance from s

claim \( d \) is the min distance from \( s \) to \( y \). [This justifies the output being a tree]

Proof. Any path \( \Pi \) from \( s \) to \( y \) consists of

\( \Pi_1 \) - initial part of path in \( B \)
\( \Pi_2 \) - rest of path.

\( w(\Pi) \geq w(\Pi_1) + w((u,v)) \geq d(s,u) + w((u,v)) \geq d \)

- using that \( w(\Pi_2) \geq 0 \)
- the proof breaks down for neg. weight cycles

Therefore, by induction on \( |B| \), the alg. correctly finds \( d(s,v) \) for all \( v \).

Implementations

- want to choose edge leaving \( B \) to minimize some value
- could make a heap of edges \((x,y) \in B, y \notin B\)
  where \( \text{value}(x,y) = d(s,x) + w(x,y) \)
  This heap has size \( O(m) \)
- More efficient: a heap of vertices
Keep "tentative distance" \( d(v) \) \( \forall v \in B \)
\( d(v) = \min \) weight path from \( S \) to \( v \) with all but last edge in \( B \)

Initialize
\[
\begin{align*}
    d(v) &\leftarrow \infty \quad \forall v \neq s \\
    d(s) &\leftarrow 0 \\
    B &\leftarrow \emptyset
\end{align*}
\]

While \( |B| < n \)

\[ y \leftarrow \text{vertex of } V \setminus B \text{ with min. } d \text{ value from heap} \]

\[ B \leftarrow B \cup \{y\} \quad \text{ # note that } d \text{ is true distance for each edge } (y,z) \]

\[ \text{for each edge } (y,z) \text{ do} \]

\[ \text{if } d(y) + w(y,z) < d(z) \text{ then} \]

\[ d(z) \leftarrow d(y) + w(y,z) \quad \text{and update heap} \]

\[ \text{Parent}(z) \leftarrow y \]

end

end

Store \( d \) values in a heap, size is \( \leq n \)

Modifying a \( d \) value takes \( O(\log n) \) to adjust heap (can delete + add or use Decrease-key operation)

Total time \( O(n \log n + m \log n) = O(m \log n) \) assuming \( G \) connected

Actually, there is a "fancier" Fibonacci heap, that gives \( O(m \log n) \) see CLRS
Dijkstra was known for many contributions to computer science, e.g. structured programming, concurrent programming. He designed the above algorithm to demonstrate the capabilities of a new computer (to find railway journeys in the Netherlands).

At that time (the 50's) the result was not considered important. He wrote:

At the time, algorithms were hardly considered a scientific topic. I wouldn’t have known where to publish it . . . . The mathematical culture of the day was very much identified with the continuum and infinity. Could a finite discrete problem be of any interest? The number of paths from here to there on a finite graph is finite; each path is a finite length; you must search for the minimum of a finite set. Any finite set has a minimum – next problem, please. It was not considered mathematically respectable . . .