Recall: Minimum Spanning Tree (MST) Problem
last day: Kruskal’s algorithm
today: a different greedy algorithm

Prin’s Algorithm
Grow one connected component in a greedy fashion (i.e., by adding a vertex \( v \in V - C \) that is one end of a minimum weight edge leaving \( C \)).

Choose vertex \( v \in V - C \) connected to a minimum weight edge \( e = (u, v) \) between \( C \) and \( V - C \).

\( C = \) set of vertices reached by \( T \) so far

\[
C := \{s\} \\
T := \emptyset \\
\textbf{while} \ C \neq V \ \textbf{do} \\
\quad \text{find vertex } u \in V - C \text{ such that there exists a } u \in C \\
\quad \quad \text{with } e = (u, v) \text{ a minimum weight edge leaving } C. \\
\quad C := C \cup \{v\} \\
\quad T := T \cup \{e\} \\
\textbf{od}

Correctness: The exact same exchange argument works.
And in fact, we could prove one lemma that gives correctness of both algorithms (see text).
Prim’s Algorithm: Implementation

We need to find a vertex in $V - C$ connected to a minimum weight edge leaving $C$, the connected component of $T$.

For $v \in V - C$, define

$$weight(v) = \begin{cases} \infty & \text{if no edge } (u, v) \text{ with } u \in C \\ \min \{w(e) \mid e = (u, v) \in E \text{ and } u \in C} & \text{otherwise} \end{cases}$$

Priority Queue using the heap data structure

Maintain set $V - C$ as an array in heap order, according to weight as defined above.

- ExtractMin() : remove and return vertex with minimal weight
- Insert($v$, weight($v$)) : insert vertex $v$ with weight($v$)
- Delete($v$) : delete vertex $v$

Can be implemented at $O(\log k)$ time per operations, $k = |V - C|$.

Note: For Delete, need to keep track of array of vertices $\bar{C}[1 \ldots n]$ with

$$\bar{C}[v] = \begin{cases} -1 & \text{if } v \notin V - C \\ \text{location of vertex } v \text{ in heap} & \text{otherwise} \end{cases}$$
Prim’s Algorithm: Analysis

\begin{align*}
C &:= \{s\} \\
T &:= \emptyset \\
\textbf{while } & C \neq V \textbf{ do} \\
& \quad \text{find vertex } v \in V - C \text{ such that there exists a } u \in C \\
& \quad \quad \text{with } e = (u, v) \text{ a minimum weight edge leaving } C. \\
& \quad C := C \cup \{v\} \\
& \quad T := T \cup \{e\} \\
\textbf{od}
\end{align*}

- One ExtractMin to find \(v\).
- Scan through \(v\)'s adjacency list to find
  - \(e = (u, v)\) with \(w(e) = \text{weight}(v)\).
  - all edges \(e' = (v, v')\) with \(v' \in V - C\) reduce weight of \(v'\) in heap if necessary.

Size of heap is \(O(n)\):
- \(n - 1\) ExtractMin operations
- \(O(m)\) “reduce weight” operations can be implemented using Delete + Insert

Total cost: \(O(m \log n)\)
Shortest Paths in Edge Weighted Graphs

Recall that BFS from $v$ finds shortest paths from $v$ in unweighted undirected graphs.

General input: directed or undirected graph with weights on edges

- Shortest path $A$ to $D$ is $ABD$, weight 5.
- Shortest path $A$ to $E$ is $ABE$, 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: graph or 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: We have tree of shortest paths to all vertices in set \( B \). Initially \( B = \{s\} \).

Choose edge \((x, y)\), \( x \in B \), \( y \notin B \) to minimize \( d(s, x) + w(x, y) \), where \( d(s, x) \) is the (known) minimum distance from \( s \) to \( x \).

Call this minimum \( d \).

\[
d(s, y) := d \\
\text{add (}x, y\text{) to tree (}\text{Parent}(y) := x)\
\]

This approach is greedy in the sense that we always add the vertex with the next minimum distance from \( s \).
Claim $d$ is the minimum distance from $s$ to $y$.
(this justifies the output of the problem being a tree)

Proof Any path $\pi$ from $s$ to $y$ consists of

- $\pi_1$ — initial part of path in $B$
- $e = (u, v)$ — first edge leaving $B$
- $\pi_2$ — rest of path

$$w(\pi) \geq w(\pi_1) + w(u, v) \geq d(s, u) + w(u, v) \geq d$$
using that $w(\pi_2) \geq 0$

Note: the proof breaks down for negative weight edges.

Therefore, by induction on $|B|$, the algorithm correctly finds $d(s, v)$ for all $v$. 
Dijkstra’s Algorithm: Implementation

Keep “tentative distance” \( d(v) \) \( \forall v \notin B \).

\( d(v) = \) minimum weight path from \( s \) to \( v \) with all but last edge in \( B \)

\[
\begin{align*}
    &d(v) := \infty \quad \forall v \neq s \\
    &d(s) := 0 \\
    &B := \emptyset \\
    \textbf{while } |B| < n \textbf{ do} \\
    &\quad y := \text{vertex of } V - B \text{ with minimum } d \text{ value} \quad \text{— from heap} \\
    &\quad B := B \cup \{y\} \\
    &\quad \textbf{for} \text{ each edge } (y, z) \textbf{ do} \\
    &\quad \quad \textbf{if } d(y) + w(y, z) < d(z) \textbf{ then} \\
    &\quad \quad \quad d(z) := d(y) + w(y, z) \quad \text{— and update heap} \\
    &\quad \quad \quad \text{Parent}(z) := y \\
    &\quad \textbf{fi} \\
    &\quad \textbf{od} \\
    \textbf{od}
\end{align*}
\]

Store the \( d \) values in a heap of size \( \leq n \).

Modifying a \( d \) value takes \( O(\log n) \) to adjust heap.

Total time, assuming \( G \) connected:

\[
O\left(n \log n + \underbrace{m \log n}_{\text{find min}} + \underbrace{m \log n}_{\text{adjust heap}}\right) = O(m \log n)
\]
Actually, there is a fancier “Fibonacci heap” that gives $O(n \log n + m)$ (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.