Driving directions

From Austin to Athens – many possible routes:
Problem: Single-Pair Shortest Path

Input: A directed graph \( G = (V, E) \) where each edge 
\((v_i, v_j)\) has a weight \( w(i, j) \).

Output: A “shortest” path from \( u \) to \( v \).

Weight of path: Given a path \( p = < v_1, ..., v_k > \), its weight is:

\[
w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})
\]  

(1)

“shortest” path = path of minimum weight. We use \( \sigma(u, v) \) to denote this minimum weight.
Different variants of shortest path problems

- **Single-pair shortest path (SPSP):**
  Find a shortest path from $u$ to $v$.

- **Single-source shortest paths (SSSP):**
  Find a shortest path from source $s$ to all vertices $v \in V$.

- **All-pairs shortest paths (APSP):**
  Find a shortest path from $u$ to $v$ for all $u, v \in V$.

- No algorithm is known for computing a single-pair shortest path better than solving the SSSP problem in the worst case. So we will only focus on SSSP.
Lemma 24.1: **Subpaths** of shortest paths are shortest paths.

**Proof:** Cut and paste:

If some subpath were not a shortest path, we could substitute the shorter subpath and create an even shorter total path.
Properties of shortest paths (2): Triangle Inequality

\[ \sigma[s, v] \leq \sigma[s, u] + \sigma[u, v] \] (2)
Is shortest-path well-defined?

Negative weight cycle ⇒ no shortest path.

Argument: path can be shortened by traversing a negative cycle.
Dijkstra’s algorithm: Idea

- Maintain a set \( S \) of vertices whose final shortest-path weights from the source \( s \) have already been determined (\( S \) is just like the set \( A \) in Prim’s algorithm).

- The set \( S \) initially contains only the source \( s \).

- The algorithm repeatedly selects the vertex \( u \in V - S \) with the minimum shortest-path estimate (like the key in Prim’s algorithm), adds \( u \) to \( S \), and relaxes all edges leaving \( u \).

- **Input requirement**: \( w(u, v) \geq 0 \), for all \((u, v) \in E\).
**Dijkstra’s algorithm: Data Structures**

**Data Structures:**

$S : \text{ Vertices whose shortest paths have already been determined.}$

$V - S : \text{ Remainder.}$

$d : \text{ } d[v] \text{ tells the current best estimate of shortest path to the source.}$

$\pi : \text{ } \pi[v] \text{ tells the predecessor for vertex } v \text{ in the current shortest path.}$
Dijkstra’s Algorithm: Auxiliary Functions

`InitializeSingleSource(G, s) {`

for each vertex \( v \in V \) do

\[
d[v] = \infty \\
\pi[v] = \text{nil} \\
d[s] = 0
\]

`}

`Relax(u, v, w) {`

if \( d[v] > d[u] + w(u, v) \) then

\[
d[v] = d[u] + w(u, v) \\
\pi[v] = u
\]

`}
Relaxation

\[ d[s] = 0 \]

\[ d[v] \]

\[ d[u] \]

if \( d[v] > d[u] + w(u, v) \) then 
\[ d[v] = d[u] + w(u, v) \]

else do not change \( d[v] \)

Before relaxation:

\[ \begin{array}{c}
u \quad 2 \\
d[u]=5 \\
\end{array} \quad \begin{array}{c}
v \\
d[v]=9 \\
\end{array} \]

After relaxation:

\[ \begin{array}{c}
\begin{array}{c}
2 \\
d[u]=5 \\
\end{array} \quad \begin{array}{c}
v \\
d[v]=7 \\
\end{array} \end{array} \]
Dijkstra’s algorithm: Example

\[ S = \langle \rangle \quad Q = \langle z, u, v, x, y \rangle \]

\[ S = \langle z \rangle \quad Q = \langle u, x, v, y \rangle \]

\[ S = \langle z, u \rangle \quad Q = \langle x, y, v \rangle \]
Example

\( S = \langle z, u, x \rangle \quad Q = \langle v, y \rangle \)

\( S = \langle z, u, x, v \rangle \quad Q = \langle y \rangle \)

\( S = \langle z, u, x, v, y \rangle \quad Q = \langle \rangle \)
Dijkstra’s algorithm

Dijkstra\((G(V, E), w, s)\)  /* s is the source */

1. InitializeSingleSource\((G, s)\);

2. \(S := \emptyset\); /* Make \(S\) empty */

3. \(Q := V\); /* put all vertices into a Priority Queue */

4. while \(Q\) is not empty

5. \(u := \text{Extract-Min}(Q)\); /* get the vertex which is closest to the source \(s\), and remove it from the queue */

6. \(S := S \cup u\); /* Add \(u\) to \(S\) */

7. for each \(v \in \text{Adj}[u]\) /* update the \(ds\) to \(s\) */

8. Relax \((u, v, w, Q)\);
Similarity with Prim’s algorithm

\textbf{MST-PRIM} \((G(V, E), w, r) \)  
/* \( r \) is the arbitrarily selected starting point */

\begin{align*}
1 & \text{ for each } u \in V \\
2 & \quad \text{\( key[u] := \infty; \)} \\
3 & \quad \text{\( key[r] := 0; \)} \quad /* \text{the first to be picked into } V_A */ \\
4 & Q := V; \quad /* \text{put all vertices into a PQ */} \\
5 & \text{while } Q \text{ is not empty} \\
6 & \quad u := \text{Extract-Min}(Q); \quad /* \text{extract the vertex which is closest to the tree } A */ \\
7 & \quad \text{for each } v \in \text{Adj}[u] \quad /* \text{update the dist. to } A */ \\
8 & \quad \quad \text{if } v \in Q \text{ and } w(u, v) < \text{key}[v] \\
9 & \quad \quad \quad \text{key}[v] := w(u, v)
\end{align*}
Use a **Binary Heap** to implement the min-priority queue.

**DIJKSTRA** \((G(V, E), w, r)\)

1. InitializeSingleSource(G, s); \(\Theta(|V|)\)
2. \(S := \emptyset;\) \(\Theta(1)\)
3. \(Q := V;\) \(\text{Build-Min-Heap: } O(|V|)\)
4. \textbf{while} \(Q\) is not empty \(\text{|V| times}\)
5. \(u := \text{Extract-Min}(Q);\) \(\text{Extract-Min: } O(lg|V|)\)
6. \(S := S \cup u;\) \(\Theta(1)\)
7. \textbf{for each} \(v \in \text{Adj}[u]\) \(\text{O(|E|)}\)
8. \(\text{relax}(u, v, w, Q);\) \(\text{/Decrease-Key */ } O(lg(|V|))\)
Correctness of Dijkstra’s algorithm

- We need to show that when the algorithm finishes, 
  \( d[u] = \sigma[s,u] \) for every \( u \) in \( V \).

- We’ll show that when \( u \) is inserted to \( S \), \( d[u] = \sigma[s,u] \).
Assume: $d[u] > \sigma[s,u]$ — Proof by contradiction

Let $u$ be the first vertex such added to $S$ s.t. $d[u] > \sigma[s,u]$.

When $x$ was added to $S$, $d[x] = \delta(s,x)$ and edge $(x,y)$ was relaxed $\implies d[y] \leq \delta(s,x) + w(x,y) \leq \delta(s,u)$

Thus, $d[y] \leq \delta(s,u) \leq d[u]$. But $d[u] \leq d[y]$ because $u$ was chosen to be added before $y \implies d[u] = \delta(s,u) \implies$ contradiction!
Dijkstra’s algorithm

- Where are we using the assumption that the weights are $\geq 0$?

A historic note:

- Prim’s algorithm was invented in 1957.

- Dijkstra’s algorithm was invented in 1959, without the use of a priority queue.