Lecture #7
Shortest Paths in Graphs

Last time:

1) All edges have same weight $\Rightarrow$ BFS

today:

2) Edges can have different positive weights
$\Rightarrow$ Dijkstra's Algorithm

3) Edges can have negative weights
$\Rightarrow$ Bellman-Ford Algorithm

4) Detecting negative cycles

5) Shortest Paths in DAGs

Notation: \( G = (V,E), l : E \rightarrow \mathbb{N} \) gives length of each edge

\[ d(s,v) = \text{length of shortest path from } s \text{ to } t \]
Example

Idea: at each step: update

\[ K = \text{vertices to which we know shortest path} \]

There are no vertices outside of \( K \) with shorter paths to them than those inside \( K \).
Dijkstra's Algorithm

\[
\text{Dijkstra} (G, s) \quad \text{... } G = (V, E)
\]

\[\text{dist}[s] = 0,\]

\[\forall v \neq s, \text{dist}[v] = \infty\]

\[K = \emptyset \quad \text{... vertices for which shortest paths known}\]

while \( K \neq V \)

\[\text{pick } u \in V \setminus K \text{ with smallest } \text{dist}[u]\]

\[K = K \cup \{ u \}\]

for all \((u, v) \in E\)

\[\text{update}(u, v) (\text{dist}[v] = \min (\text{dist}[v], \text{dist}[u] + l(u, v)))\]
Proof of Correctness for Dijkstra
Notation: \( d(s,v) = \text{length of a shortest path from } s \text{ to } v \)

Claim: At anytime \( \forall v \in K \), dist[v] = d(s,v)

Proof: Induction

Base Case: \( K = \emptyset \) trivial

First Step: \( K = \{ s \} \) \( d(s,s) = 0 = \text{dist}[s] \)

Induction Step: Let \( v \) be vertex with smallest dist[v], claim dist[v] = d(s,v)

Let \( s \rightarrow a \rightarrow b \rightarrow v \in K \) be a shortest path

all \( v \in K \) first one not in \( K \): Fact every prefix of a shortest path is a shortest path

1. If \( b = v \) \( \Rightarrow \) dist[v] = dist[b] since \( b = v \)
   \[ \leq \text{dist}[a] + \ell(q, b) \] inner loop of alg
   \[ = d(s, a) + \ell(q, b) \] since each by induction
   \[ = d(s, b) \] since \( s \rightarrow b \) is a (prefix of a) shortest path
   \[ = d(s, v) \] alg choosing \( v \)

2. If \( b \neq v \) dist[b] \( < \) dist[v] contradicts alg choosing \( v \)
Dijkstra’s Algorithm, Updated

Dijkstra \((G, s)\) \[ G = (V, E) \]

dist \([s] = 0\)

\[ \forall v \neq s \; , \; \text{dist}[v] = \infty \]

\[ K = \emptyset \quad U = V \quad (U = V \setminus K) \]

\[ \text{Initialize Priority Queue} \quad Q \leftarrow V \; , \; \text{keys} = \text{dist} \]

\[ \text{while } K \neq V \; \text{not empty} \]

pick \( u \in U \; \forall \{K \text{ with smallest dist}[u]\} \)

\[ K = K \cup \{u\} \; \text{remove } u \text{ from } U \]

for all \((u, v) \in E\)

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

\[ \Rightarrow \text{need data structure for picking smallest dist}(u), \text{updating Priority Queue: Binary Heap } \\text{Delete Min, DecreaseKey } \text{O}(\log |V|) \]

\[ \text{... Fibonacci Heap} \]
Running time for Dijkstra

Cost = # operation

- Make Queue once: $1V$ inserts, $\Rightarrow$ cost = $O(1V \log 1V)$ or $O(1V)$
- Delete Min: once per vertex: $1V$
- Decrease Key: once per edge: $1E$

Overall time: $O((1V + 1E) \log 1V)$ using binary heap
$O(1V \log(1V + 1E))$ using Fibonacci
useful if $|E| \gg |V| \text{ “Dense graph”}$

More complicated algs nearly $O(1V + 1E)$
Shortest Paths with Positive or Negative Edge Lengths

\[ \text{dist}(A, C) = 2 - 3 - 4, \ldots \]

if \((u, v) \in E\) update \((u, v)\): \(\text{dist}[v] = \min(\text{dist}[v], \text{dist}[u] + l(u, v))\)

1. update "safe" \(\text{dist}[v] \geq d(s, v)\)
   \[ \Rightarrow \text{extra updates OK} \]

2. if shortest path from \(s\) to \(v\) is \(s \rightarrow u \rightarrow v\) and \(\text{dist}[v] = d(s, v)\)
   then after update \(\text{dist}[v] = d(s, v)\)
Bellman–Ford

Shortest path $s \rightarrow u_1 \rightarrow u_2 \rightarrow u_3 \rightarrow \ldots \rightarrow u_t \rightarrow v$

For $i = 1$ to $|V| - 1$

for all $(u, v) \in E$, update $(u, v)$

no negative cycles $\Rightarrow$ all shortest paths have $\leq |V| - 1$ vertices

$\Rightarrow$ all updates appear in desired order $\Rightarrow$ all $\text{dist}[v] = d(s, v)$

Running time $= (|V| - 1) \cdot |E| \cdot \text{time(update)} = O(|V| \cdot |E|) = O(|V|^3)$
Shortest Paths in DAGs

DAG → no cycles → no negative cycles
   → pos + neg edge lengths ok

Idea: Topologically sort, starting with s

s → u_1 → u_2 → u_3...
   all edges go left to right

→ all shortest paths look like

(s, u_i_1) (u_i_1, u_i_2) (u_i_2, u_i_3) ... i_1 < i_2 < i_3...

→ updating all (u_i, v_k) in order of increasing i works

Cost = topological sort = DFS = O(V'E)

+ updating in order → O(V' + 2E)
Detecting Negative Cycles

Bellman Ford assumed no neg cycles $\Rightarrow$ all shortest path have $\leq |V| - 1$ vertices

Thm: No neg cycles $\iff$ Running Bellman Ford for one more iteration (update all edges once more)
doesn't change any dist[v]

$\Rightarrow$ run Bellman |V| times instead of |V| - 1, signal neg cycle if any dist[v] changes

Proof: No neg cycles $\Rightarrow$ all shortest path, have |V| - 1 vertices

$\Rightarrow$ updating once more "safe", nothing changes

RunBF, no dist changes

$\text{dist}[a_i] \leq \text{dist}[a_{t}] + \ell (a_{t}, a_i)$

no neg cycle!
\( i = 0 \quad i = 1 \quad i = 2 \quad \text{victory!} \)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>-4</td>
<td>-4</td>
<td>-4</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td>8</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
</tbody>
</table>