Paths in Graphs
**Single-source shortest paths (SSSP)**

**Input**: Graph $G$, "source" vertex $s \in V$

**Output**: $\forall u \in V$, $d(s,u) =$ length of shortest path from $s$ to $u$

Unweighted: all edges length $1$

- **Breadth-first search**
- **Positive lengths**: $l : E \rightarrow \{1, 2, 3, \ldots\}$
- **Dijkstra**
- **Arbitrary length edges**
- **Bellman-Ford**
Unweighted graphs

DPS:

Neighbors of neighbors (have not yet seen)

Dist 2

Neighbors dist 1
Breadth-first search 

bfs \((G, s)\)

dist \([s] = 0\) 

\(\forall u \neq s,\ dist\ [u] = \infty\)

\[Q = \{s\}\ (queue\ containing\ s)\]

while \(Q\) is not empty

\(u =\) dequeue \((Q)\)

for all \(v\) s.t. \((u, v) \in E\)

if \(dist\ [v] = \infty\)

enqueue \((Q, v)\)

\(dist\ [v] = dist\ [u] + 1\)

Runtime: \(O(n + m)\) time 
linear, same as \(DFS\)

DFS is just \(BFS\) w/ stack
Positive lengths $g. ¥ +71$

Dijkstra’s algorithm

1. Compute $v_1 =$ closest vertex to $s$ and $d(s,v_1)$

2. Compute $v_2 =$ 2nd closest to $s$ and $d(s,v_2)$

3. ” $v_3 =$ 3rd ”

$K =$ the set of “known” vertices at some step

$K = \{v_1, ..., v_{c3}\}$

$U =$ “unknown” vertices $U = V \setminus K$
Q: Given \( K = \{ v_1, \ldots, v_3 \} \). How to compute \( V_{i+1} = \text{closest vertex not in } K \) in \( U \)?

\( V_{i+1} = \text{endpoint of the shortest path of this form} \)

\[
\begin{align*}
\text{dist}[v_{i+1}] &= \text{dist}[v_j] + \ell(v_j, v_{i+1}) \\
& \quad \text{u minimizes } \text{dist}[v_j] + \ell(v_j, v_{i+1})
\end{align*}
\]
Dijkstra has dist[v] for each v ∈ V

\[ \text{dist}[v] = \begin{cases} d(s, v) & \text{if } u \in K \\ \min_{u \in K} \text{dist}[u] + l(u, v) & \end{cases} \]

After adding \( v_{\text{ini}} \) to \( K \)

If \( (v_{\text{ini}}, w) \in E \)

\[ \text{dist}[w] = \min \{ \text{dist}[w], \text{dist}[v_{\text{ini}}] + l(v_{\text{ini}}, w) \} \]

[update \( (v_{\text{ini}}, w) \)]
Dijkstra \((G, l, s)\)

\[
\begin{align*}
dist[s] &= 0 \\
\forall u \neq s, \ dist[u] &= \infty \\
U &= \emptyset \quad \text{(insert \((u, dist[u])\) \# \(u\)}
\end{align*}
\]

while \(U\) is not empty
  
  Choose \(u \in U\) with minimum
  
  Remove \(u\) from \(U\), \(dist[u] \leftarrow \text{Delete Min}(u)\)

for each \(v\) s.t. \((u, v) \in E\)
  
  \[
  dist[v] = \min(\text{dist}[v], \ dist[u] + l(u, v))
  \]

Decrease Key \((v, \text{dist}[v])\)

Priority queue

Contains a set of (element, key) pairs

- Insert \((elem, key)\)
- Decrease Key \((elem, key)\) \((\text{Replaces elem's old key w/ new key})\)
- Delete Min \((c)\)
Dijkstra's runtime

- Insert n times
- Delete Min n times
- Decrease Key m times

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Insert</th>
<th>DelMin</th>
<th>Decrease</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n^2 + m) = O(n^2)$</td>
</tr>
<tr>
<td>binary heap</td>
<td>$\log(n)$</td>
<td>$\log(n)$</td>
<td>$\log(n)$</td>
<td>$O((n+m) \log(n))$</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>$O(1)$</td>
<td>$O(\log(n))$</td>
<td>$O(1)$</td>
<td>$O(n \log(n) + m)$</td>
</tr>
</tbody>
</table>

Mikkel Thorup 2004: $O(n \log \log n + m)$
Negative weights

Does it make sense?

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

1. Update is "safe": \( \text{dist}[v] \geq d(s,v) \)
2. Suppose shortest \( s \rightarrow v \) path is \( s \rightarrow \cdots \rightarrow g \rightarrow v \) and \( d[v] = d(s,u) \). Then after update, \( \text{dist}[v] = d(s,v) \)

**update** \((u,v)\): \[ u \rightarrow v \]

well-defined if all cycles have positive lengths
Bellman-Ford \((G, \ell, s)\):

\[\text{For } i = 1 \ldots N-1 \]

\[\text{update all the edges } \]

\[\text{update}(u,v) \]

Runtime: \(O(nm)\)

(best known for arbitrary weights)