Shortest Paths

Input: Graph \( G = (V, \mathcal{E}) \), edge weights \( l_{vw} \) for edge \((v, w)\), \( s \in V \)
\( n = |V|, \ m = |E| \)

Output: \( \forall \ v \in V, \ L(v) = \text{shortest path from } s \text{ to } v \)
\( L(v) = \infty \) if \( s, v \) not connected

Graph given as adjacency list \( A_G \) (array of lists)
\[ A_G[v] = \{ \{ u, l_{uv} \}, \{ w, l_{vw} \}, \{ x, l_{vx} \} \} \]

BFS (use when all \( l_{uv} \) have same value)

\[ L[v] = \infty \quad \forall v \in V \quad \text{// will store shortest paths} \]

\[ X[v] = \text{false} \quad \forall v \in V \quad \text{// mark True when "explored", separate Array} \]

\[ A = \{ s \}; \ A. \text{add}(s); \quad \text{// initialize QUEUE} \]

\[ L[s] = 0 \]

\[ X[s] = \text{true} \]

While \( A \) is not empty:
\( v = A. \text{pop} \)
For each edge \((v, w)\):
If \( \text{Ex}[w] = \text{false} \):
\( A. \text{add}(w); \ \text{Ex}[v] = \text{true}; \ L[w] = L[v] + 1 \)
Explain: Why is runtime $O(n+m_s)$

- Looks at edges (in for loop)
- Each edge can only be examined when its adjoining vertex is popped. Each vertex can only show up in QUEUE one time. $\Rightarrow$ Each edge is examined once.

- Looking at each edge takes constant time b/c use adjacency list.

$O(n)$ to initialize $\Rightarrow$ $O(n+m_s)$

$O(m_s)$ to do while loop

$\#$ edges connected to $s$
Dijkstra's Algorithm

\( X = \{s\} \)
\( A[s] = 0 \)
\( B[s] = \emptyset \)

While \( X \neq V \)
- Among edges \((v, w) \in E\)
  with \( v \in X\), \( w \in V - X\),
  pick edge that minimizes
  \[ A[v] + l_{vw} \]

\[ \text{Dijkstra's greedy criterion} \]

Let \((v^*, w^*)\) be minimizing edge
- \( X = X + w^* \)
- \( A[w^*] = A[v^*] + l_{v^*w^*} \)
- \( B[w^*] = B[v^*] + (v^*, w^*) \)

Q: What is run time using adjacency list graph input?

```plaintext
for v \in V:
  if X[v] == 1:
    for w \in A_G[v]:
      if X[w] == 0:
        Calculate DGC
```
While Loop: \( n \) times

- \( \square \) routine: \( O(m) \) times, since checks at most \( m \) edges

Total = \( O(nm) \)
Use data structure to get speed up!

Q: Which data structure is best for this algorithm

A) Stack   B) Queue   C) Heap/Priority  D) Binary Search Tree

Repeatedly finding minimum, like heap sort

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time for heap with n items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract element with minimum key</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Insert element</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Learn minimum key</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Delete an element (if known position in heap)</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>
Q: What should be store in heap, and what should keys be? What is run time?

A: Store edges from $X$ to $V-X$ in heap, with key given by $A[u] + d_{u,v}$ for edge $(u, v)$

- $n$ times outer loop
- can have to do $O(n)$ insertions and deletions each round (for each edge connected from $X$ to new vertex and each edge from new vertex to $V-X$)

$$n \log m = O(n^2 \log m)$$

Better than $O(nm)$, but we can do better!
Instead

- Store vertices $v \in V - X$ in heap
- Key of $v$ is
  \[ v.\text{key} = \min_{w \in X} \{ A[w] + l_{w,v} \} \]
- For each element $v$,
  \[ v.\text{p} = \arg\min_{w \in X} \{ A[w] + l_{w,v} \} \]
- If $v$ is not directly connected to $X$, $k(v) = \infty$
  \[ v.\text{p} = \emptyset \]

Why better?

Only need to check vertices in $A_G[w^*]$
Dijkstra Heap

\[ X[v] = 0; A[v] = \infty; B[v] = \emptyset; \forall v \in V \]
\[ v.key = \infty \text{ for all } v \in V \]
\[ v.p = \emptyset \text{ for all } v \in V \]
\[ s.key = 0. \]

Heapify all \( v \in V \)

while (Heap is not empty)
  - Let \( w = \) vertex with min \( \text{key} \)
  - Remove \( w \); \( X[w] = 1; A[w] = w.\text{key} \)
  - \( B[w] = B[w.p] + (w.p, w) \)

  - for \( u \in A_g[w] \) & \( u \) not explored
    - Check if need to update \( u.\text{key} \)
    - If yes, remove and reinsert
Example:

\[ \begin{array}{c|c|c|c|c}
 & S & U & V & X \\
\hline
S & 0 & \infty & \infty & \infty \\
U & \infty & 3 & \infty & \infty \\
V & \infty & \infty & 5 & \infty \\
X & \infty & \infty & \infty & 8 \\
\end{array} \]

Round 1: [0 | φ] \rightarrow [3 | 3] \rightarrow [4 | 4] \rightarrow [6 | 6]

Round 2: [3 | S] \rightarrow [5 | 5] \rightarrow [4 | 4] 

Round 3: [4 | U] \rightarrow [4 | U] \rightarrow [6 | 6] 

H