CSE 100
Dijkstra’s Shortest Path
Q1: Which data structure is used for implementing BFS?

A. Queue
B. Priority Queue
C. Heap
D. Stack
Q2: Which data structure is used for a fast implementation of Dijkstra’s algorithm?

A. Queue  
B. Priority Queue  
C. Linked list  
D. Stack
Q3: In class we derived the run time for traversing a graph using BFS to be $O(|V|+|E|)$. Which of the following assumptions were made about the underlying representation of the graph in our analysis?

A. The graph is represented by a Adjacency List
B. The graph is represented by a Adjacency Matrix
C. No assumptions were made.
Q4: A minimum spanning tree exists in an undirected graph if and only if which of the following conditions are satisfied by the graph?

A. The graph is connected
B. The graph is sparse
C. The graph is dense
D. The graph is weakly connected
Dijkstra’s Algorithm

$X$: Set of all vertices for which shortest path has been computed

$L(v)$: Shortest path from $S$ to $v$

Initially $X = \{v_0\}$, $L(v_0) = 0$

Goal: Compute the shortest path for a vertex in $V - X$.

How?

For each vertex with edges that cross the frontier, calculate a tentative distance, also known as Dijkstra Score $J$:

$J(v, w) = L(v) + l_{vw}$

In each iteration, choose an edge $(v^*, w^*)$ with the min score and add its vertex $w^*$ to the set $X$. Update its shortest path to be the same as the score $L(w^*) = L(v^*) + l^{v^*w^*}$.

1. $X = \{v_0, v_3\}$, $L(v_0) = 0$, $L(v_3) = 1$

2. $X = \{v_0, v_3, v_2\}$, $L(v_2) = 2$

3. $X = \{v_0, v_3, v_2, v_4\}$, $L(v_4) = 3$

4. $X = \{v_0, v_3, v_2, v_4, v_1\}$, $L(v_1) = 4$

5. $J(v_3, v_1) = 4$

6. $J(v_0, v_1) = 5$
Dijkstra’s Algorithm

For the given graph, assume:
• the source is v0
• shortest path to v0 and v1 has already been computed. As a result X={v0, v1}

Which vertex will be added to the set X next?

A. v2
B. v3
C. Neither
D. Both

What is the revised score for all vertices?
Dijkstra’s Algorithm

At each iteration of Dijkstra’s algo, the vertex that is added to the set X is one that can be reached via a vertex (in X) that in turn has the minimum shortest path.

A. True
B. False It’s the vertex with the minimum score
Running Time of Naïve Implementation of Dijkstra’s Algorithm

1. Initialize $X=\{s\}$
   \[ L(s) = 0, \quad L(v) = 0 \quad \forall v \in V-X \]

2. Iterate until $|X| = |V|$
   (1v1-1) iterations

Main loop: Among all crossing edges $(v, w)$, with $v$ in $X$ and $w$ in $V-X$, pick $(v^*, w^*)$ such that

\[(v^*, w^*) = \arg\min_{v,w} J(v, w)\]

\[J(v, w) = L(v) + l_{vw}\]

\[X = X \cup \{w^*\}\]

\[L(w^*) = J(v^*, w^*)\]

Run Time
\[O(1v1) + O(1v1E1)\]
Towards a fast implementation of Dijkstra’s Algorithm

1. Initialize $X=\{s\}$
2. Iterate until $|X| = |V|$
   
   Main loop: Among all crossing edges $(v, w)$, with $v$ in $X$ and $w$ in $V-X$, pick $(v^*, w^*)$ such that:
   
   $$(v^*, w^*) = \arg \min_{v,w} J(v,w)$$
   
   $$J(v, w) = L(v) + l_{vw}$$
   
   $X = X \cup \{w^*\}$
   
   $L(w^*) = J(v^*, w^*)$

How can we improve the above implementation?
Towards a fast implementation of Dijkstra’s Algorithm

Two key observations
1. When the shortest path to a new vertex is discovered, we only need to update the Dijkstra scores of all vertices that are connected to the new vertex- if we remember previous scores
2. Use a heap to speed up min computations

Check out the example graph on slide 6 also shown here & think about which new edges we need to compute the score for, when the shortest path to v3 was discovered? 
* we only needed to compute the score for edges emanating out of v3
Dijkstra’s Algorithm

- Initialize the graph: Give all vertices a dist of INFINITY, set all “done” flags to false
- Start at s; give s dist = 0 and set prev field to -1
- Enqueue (s, 0) into a priority queue. This queue contain pairs (v, cost) where cost is the best cost path found so far from s to v. It will be ordered by cost, with smallest cost at the head.
- While the priority queue is not empty or until all shortest paths are discovered
  - Dequeue the pair (v, c) from the head of the queue.
  - If v’s “done” is true, continue
  - Else set v’s “done” to true. We have found the shortest path to v. (It’s prev and dist field are already correct).
  - For each of v’s adjacent nodes, w (whose done flag is not true):
    - Calculate the best path cost (also referred to as score), c, to w via v by adding the edge cost for (v, w) to v’s “dist”.
    - If c is less than w’s “dist”, replace w’s “dist” with c, replace its prev by v and enqueue (w, c)
Dijkstra’s Algorithm: Data Structures

• Maintain a sequence (e.g. an array) of vertex objects, indexed by vertex number
  – Vertex objects contain these 3 fields (and others):
    • “dist”: the cost of the best (least-cost) path discovered so far from the start vertex to this vertex
    • “prev”: the vertex number (index) of the previous node on that best path
    • “done”: a boolean indicating whether the “dist” and “prev” fields contain the final best values for this vertex, or not

• Maintain a priority queue
  – The priority queue will contain (pointer-to-vertex, path cost) pairs
  – Path cost is priority, in the sense that low cost means high priority
  – Note: multiple pairs with the same “pointer-to-vertex” part can exist in the priority queue at the same time. These will usually differ in the “path cost” part
Your Turn

The array of vertices, which include dist, prev, and done fields (initialize dist to ‘INFINITY’ and done to ‘false’):

- **V0**: dist = 0, prev = -1, done = t, adj: (V1, 1), (V2, 6), (V3, 3)
- **V1**: dist = 1, prev = V0, done = f, adj: (V2, 4)
- **V2**: dist = 2, prev = V3, done = f, adj: 
- **V3**: dist = 3, prev = V0, done = f, adj: (V2, 1)

The priority queue (set start vertex dist=0, prev=-1, and insert it with priority 0 to start)
Dijkstra’s Algorithm: RunTime

- Initialize the graph: Give all vertices a dist of INFINITY, set all “done” flags to false
- Start at s; give s dist = 0 and set prev field to -1
- Enqueue (s, 0) into a priority queue. This queue contain pairs (v, cost) where cost is the best cost path found so far from s to v. It will be ordered by cost, with smallest cost at the head.
- While the priority queue is not empty or until all shortest paths are discovered
  - Dequeue the pair (v, c) from the priority queue.
  - If v’s “done” is true, continue
  - Else set v’s “done” to true. We have found the shortest path to v. (It’s prev and dist field are already correct).
  - For each of v’s adjacent nodes, w (whose done flag is not true):
    - Calculate the best path cost (also referred to as score), c, to w via v by adding the edge cost for (v, w) to v’s “dist”.
    - If c is less than w’s “dist”, replace w’s “dist” with c, replace its prev by v and enqueue (w, c)

What is the running time of this algorithm in terms of |V| and |E| in a connected graph? (More than one might be correct—which is tighter?)

A. O(|V|²)
B. O(|E| + |V|)
C. O(|E| * log|E|)
D. O(|E| * log|V|)
E. O(|E|*|V|)
Dijkstra’s Algorithm: Running time

- Each element of the adjacency list can be inserted and deleted from the priority queue; there are $|E|$ such elements.
- An insertion or delete-min in a binary heap implementation of a priority queue is $O(\log N)$; here $N = |E|$ worst-case.
- So, the algorithm has total worst-case time cost $O(|E| \log |E|)$.
- Since $|E| \leq |V|^2$, this is $O(|V|^2 \log |V|)$ and also $O(|E| \log |V|)$.

\[ O(|E| \log |E|) \]

\[ = O(|V|^2 \log |V|) \]

\[ = O(|E| \log |V|) \]