CS 101A Fall 2025 - Quiz 6
December 30, 2025 - 20 Minutes **Name:** **Student ID:**

---

1. (30 points) True or False

(a) (3') On a connected, directed graph with only positive edge weights, Bellman-Ford runs asymptotically as fast as Dijkstra. ◯ True √ False

(b) (3') Suppose we want to find the shortest path between all pairs of vertices on graph $G = (V, E)$, where $|E| = 3|V|$ and all edge weights are positive. In this case, Dijkstra's may runs faster than Floyd-Warshall algorithm.
√ True ◯ False

(c) (3') Given a directed graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$, and G has no negative cycle . In Bellman-Ford's algorithm, after k out-most iterations, the shortest path from v_1 to v_n that consists of at most k edges is computed.
√ True ◯ False

(d) (3') After applying Bellman-Ford's algorithm on node v, if there are no negative cycles, we have the minimum distance between any two different nodes v_i and v_j. ◯ True √ False

(e) (3') On a graph with n vertices and m edges, if all edges have positive weights, Bellman-Ford's algorithm uses $O(mn)$ iterations to find the shortest distance path of a single source.
√ True ◯ False

(f) (3') In the Floyd–Warshall algorithm, if after the jth iteration ($j = |V| - 2$) of the outer loop there exists an index i such that $d[i][i] < 0$, then the graph must contain a negative-weight cycle.
√ True ◯ False

(g) (3') In any connected graph without a negative cycle, A* tree-search algorithm with consistent heuristics can always find the shortest path between two nodes.
√ True ◯ False

(h) (3') The time complexity of Dijkstra using binary heap is $O(|E|+|V| \ln |V|)$. ◯ True √ False

(i) (3') A* Graph Search algorithm returns the optimal shortest path if the heuristic function is admissible. ◯ True √ False

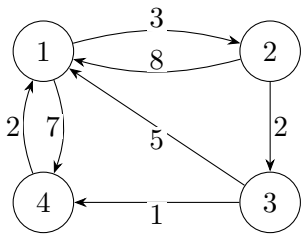
Solution: The heuristic function should be consistent.

(j) (3') In A* graph search algorithm with a consistent heuristic function, if vertex u is marked visited before v, then $d(u) + h(u) \leq d(v) + h(v)$, where $d(u)$ is the distance from the start vertex to u.
√ True ◯ False

2. (10 points) Find parent node

Consider the directed weighted graph $G = (V, E)$ shown below, where $V = \{1, 2, 3, 4\}$ and the vertex order is fixed as $1, 2, 3, 4$. We run the **Floyd–Warshall algorithm** to compute all-pairs shortest paths, and we maintain the matrix $P^{(k)}$, defined as follows: $p_{i,j} = \begin{cases} j, & \text{If there is an edge from } i \text{ to } j, \\ \varnothing, & \text{Otherwise.} \end{cases}$

(a) (10') Write down the matrices $P^{(1)}$ and $P^{(2)}$.



Solution:

$P^{(1)} =$

	1	2	3	4
1	∅	2	∅	4
2	1	∅	3	1
3	1	1	∅	4
4	1	1	∅	∅

$P^{(2)} =$

	1	2	3	4
1	∅	2	2	4
2	1	∅	3	1
3	1	1	∅	4
4	1	1	1	∅

3. (12 points) Lets code!

We want to find a single source min distance with Dijkstra's Algorithm and A* **graph search** algorithm. **Suppose all edges have positive weight and the heuristic function is consistent**. The graph mentioned in this problem is a simple directed graph.

Note: 'w' is a weight map, where you can get any edge (u, v)'s weight by using 'w(u, v)'. 'h' is a consistent heuristic function, you can get the heuristic value of a node u by using 'h(u)'.

dist[i] represents the shortest distance from s to i, pre[i] represents the previous node on the shortest path from s to i.

Q is a min-heap storing a tuple: (key, value), and sorted by value. And you have the following operations for Q:

1. **Q.push({u, val})**: put a tuple (u, val) into the heap.
2. **{u, val} = Q.pop()**: get the tuple with minimum value in the heap, and then pop the tuple out of the heap.
3. **Q.update({u, val})**: find the tuple in the heap whose key is 'u', and update its value into 'val'.

What you need to do is to write some **pseudo code** to fill in to implement the algorithms with the given operations.

(a) (6') Implement Dijkstra's algorithm.

Solution:

```
1    {u, \_} = Q.pop()
2    for each neighbor v of u:
3        if  dist[v] > dist[u] + w(u, v):
```

Algorithm 1 Single Source Shortest Path Algorithm

Input: Weight map w, min-heap Q, Source node s, heuristic function h .
Output: The shortest distance from s to all other nodes, and their previous node in the shortest path.
for $i \leftarrow 0$ **to** V **do**
    dist[i] $\leftarrow$ Inf
    pre[i] $\leftarrow$ NULL
    Q.push({i, dist[i]})
end for
dist[s] $\leftarrow$ 0
Q.update({s, 0})
while Q is not empty **do**
    Fill this part with your pseudo code
    ...
end while
return dist, pre

```
4            dist[v] = dist[u] + w(u, v);
5            pre[v] = u;
6            Q.update({v, dist[v]})
```

(b) (6') Implement A* **graph search** algorithm. You can use $h(v)$ to get the heuristic value for any node v.

Hint: The algorithm should not differ too much from Dijkstra's algorithm.

Solution:

```
1    {u, \_} = Q.pop()
2    for each neighbor v of u:
3        if  dist[v] > dist[u] + w(u, v):
4            dist[v] = dist[u] + w(u, v);
5            pre[v] = u;
6            Q.update({v, dist[v]+h(v)})
```