CS101 Algorithms and Data Structures
Fall 2023
Homework 10

Due date: December 24, 2023, at 23:59

1. Please write your solutions in English.
2. Submit your solutions to gradescope.com.
3. Set your FULL name to your Chinese name and your STUDENT ID correctly in Account Settings.
4. If you want to submit a handwritten version, scan it clearly. `CamScanner` is recommended.
5. When submitting, match your solutions to the problems correctly.
6. No late submission will be accepted.
7. Violations to any of the above may result in zero points.

1. (6 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get 1 point if you select a non-empty subset of the correct answers.

Write your answers in the following table.

(a)	(b)	(c)

(a) (2') Which of the following statements about Dijkstra's algorithm is/are true?

A. Once a vertex is marked as visited, its distance will never be updated.

B. The time complexity of Dijkstra's algorithm using complete binary heap is $\Theta(|E| \log |V|)$.

C. If we use Dijkstra's algorithm to find the distance from vertex s to vertex t, then when we first push t into the heap, we find the shortest path from s to t and stop the algorithm.

D. If vertex u is marked visited before v, then `dist[u]` $\leq$ `dist[v]`.

(b) (2') Which of the following statements about A* search algorithm is/are true?

A. If we use heuristic function $h(u) = c$ for any $u \in V$ where c is a positive constant, then the A* search algorithm will be the same with Dijkstra's algorithm.

B. An admissible heuristic function ensures optimality of both A* tree search algorithm and A* graph search algorithm.

C. A consistent heuristic function ensures optimality of both A* tree search algorithm and A* graph search algorithm.

D. Suppose we want to search for the shortest path from a city to another on a map. If we use the heuristic function $h(u) = dis(u, t)$, the Euclidean distance between u and the destination t, then this is a consistent heuristic function.

(c) (2') Which of the following statements about Bellman-Ford algorithm is/are true?

A. In a DAG with probably negative edge weights, Bellman-Ford algorithm is guaranteed to find the shortest path from source s to any vertex if it can be reached from s.

B. Suppose the unique shortest path from source s to a vertex t has l edges. It is impossible that we find this shortest path from s to t in less than l iterations.

C. If during the i-th iteration, there is no update on `dist` array, then we can stop the algorithm but still get correct results.

D. After Bellman-Ford algorithm, the `prev` array defines a tree rooted at the source vertex, and the tree is also an MST of the original graph.

2. (7 points) Choose an Algorithm

You are given a list of questions where you need to find an algorithm to solve each of them. If multiple algorithms apply, choose the most efficient one.

The algorithms that you can choose from are:
A. BFS B. Topological Sort C. Dijkstra's Algorithm
D. Bellman-Ford Algorithm E. A* Search Algorithm

Write the index (A,B,C,...) in the blank.

(a) (1') Given a graph with probably negative edge weights, we want to know whether there is a negative cycle, i.e. a cycle such that its edge weights sum up to negative. **D**

(b) (1') This is a modified version of the Travelling salesman problem. There are n points on 2D a plane. The trajectory of length k is an ordered sequence $(a_1, a_2, \ldots, a_k)$ where a_i refers to a point. The length of a trajectory is $\sum_{i=1}^{k-1} dis(a_i, a_{i+1})$. We want to find the shortest trajectory of length k. **C**

(c) (1') Given a graph and a source vertex s, we want to find the vertices that can be reached from s through no more than k edges. **A**

(d) (1') Given a DAG with probably negative edge weight and a source vertex s, we want to find the shortest path from s to every vertex. **D**

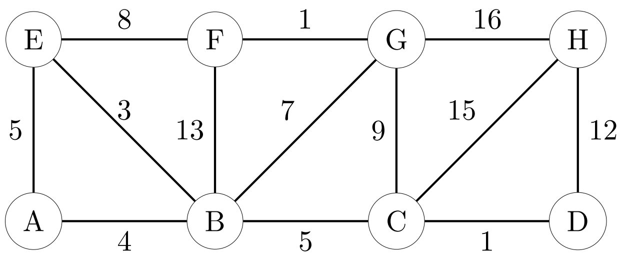
(e) (1') Given a directed graph with non-negative edge weights and a source vertex s, we want to find the shortest path from s to every vertex. **C**

(f) (1') There is a signal source in a 2D grid. You have a sensor that tells the distance between you and the signal source but the number is noisy, i.e. it may not be exactly correct but is close to the real distance. You start from a grid and can go to an adjacent grid each time. We want to find the signal source in shortest time. **E**

(g) (1') Given a undirected graph, we want to check whether it is a bipartite graph. **A**

3. (4 points) Dijkstra's Algorithm

Given the following weighted graph, run Dijkstra's algorithm by considering A as the source vertex. Write down the vertex you select and update current distance `dis[i]` of all vertices in each iteration.



	vertex	`dis[A]`	`dis[B]`	`dis[C]`	`dis[D]`	`dis[E]`	`dis[F]`	`dis[G]`	`dis[H]`
initial	/	0	∞	∞	∞	∞	∞	∞	∞
iteration 1	A	0	4	∞	∞	5	∞	∞	∞
iteration 2	B	0	4	9	∞	5	17	11	∞
iteration 3	E	0	4	9	∞	5	13	11	∞
iteration 4	C	0	4	9	10	5	13	11	24
iteration 5	D	0	4	9	10	5	13	11	22
iteration 6	G	0	4	9	10	5	12	11	22
iteration 7	F	0	4	9	10	5	12	11	22
iteration 8	H	0	4	9	10	5	12	11	22

4. (6 points) Noise

The road of SC101 country is represented by an undirected graph $G = (V, E)$ and each vertex represents a city or an airport. The set of airports is represented by $T \subset V$.

A vertex v is said to *be effected by noise* from an airport t if the length of the shortest path from t to v is not larger than R. We want to find all the vertices that have noise.

Please design an algorithm to solve this question. You may call any algorithm learned in lecture as a subroutine but be sure to indicate its input. You **don't** need to prove the correctness of your algorithm. Then, analyze your **time complexity**.

To get full credits, the time complexity of your algorithm should not exceed $O((|V| + |E|) \log |V|)$.

Input:

- $G = (V, E)$, where $V = \{1, 2, \dots, n\}$, $E = \{(u_i, v_i, w_i) : i = 1, 2, \dots, m\}$, $w_i > 0$
- $T = \{t_1, \dots, t_l\}$, where $|T| = l$, $1 \le t_i \le n$
- A positive number R

Output: All vertices v that are effected by noise from **at least one** airport $t \in T$.

Solution:

To find all vertices affected by noise from at least one airport within a given distance R, we can use an efficient algorithm that involves a single execution of Dijkstra's algorithm from a "abstract" node. The pseudocode for the algorithm is as follows:

```
function FindNoisedVerticesEfficiently($G, T, R$)
    $G' \leftarrow$ AddAbstractSource($G, T$)
    $distance \leftarrow$ Dijkstra($G', s$)
    $noisedVertices \leftarrow$ Set()
    for all $v \in V$ do
        if $distance[v] \le R$ then
            $noisedVertices$.add($v$)
        end if
    end for
    return $noisedVertices$
end function

function AddAbstractSource($G, T$)
    $G' \leftarrow$ copy of $G$
    $s \leftarrow$ new vertex not in $V$
    $V'$.add($s$)
    for all $t \in T$ do
        $E'$.add($(s, t, 0)$)
    end for
    return $G'$
end function
```

Time Complexity Analysis:

Adding the abstract and zero-weight edges takes $O(l)$ time, where $l = |T|$. Running Dijkstra's algorithm once on the new graph G' has a time complexity of $O((|V'|+|E'|)\log|V'|)$. Since we only add l vertices and l edges, $|V'| = |V|+1$ and $|E'| = |E|+l$, which does not significantly change the complexity. The overall time complexity remains $O((|V|+|E|)\log|V|)$, as the addition of the abstract and its edges does not asymptotically increase the number of vertices or edges by more than a constant factor.

This algorithm efficiently finds all vertices affected by noise from at least one airport within the specified range R and runs within the desired time complexity.

5. (8 points) Travel

There is a country with n cities and m directed edges. The edge from u to v has a length $w_{u,v}$. Traveler Bob is starting at city 1 and going to city n.

Bob can either take *car* or *train* as his vehicle. Suppose Bob is now at city v_0, and there is a path $v_0 \to v_1 \to \cdots \to v_k$. Then he can go to v_k through this path, either by car with a cost $\sum_{i=1}^{k} w_{v_{i-1},v_i}$ (the sum of the weights of the edges in this path), or by train with a cost $t_{v_0} + c \times k$, where t_{v_0} is the price of train ticket at city v_0, c is a constant, and k is the number of edges in this path.

Bob can freely decide which vehicle to take at any city. He can switch his vehicle multiple times. His goal is to minimize the total cost.

Please design an algorithm to solve this question. You may call any algorithm learned in lecture as a subroutine but be sure to indicate its input. You **don't** need to prove the correctness of your algorithm. Then, write your **time complexity**.

To get full credits, the time complexity of your algorithm should not exceed $O((n+m)\log n)$.

Input. n, m, $E = \{(u_i, v_i, w_{u_i,v_i} > 0) : i = 1, 2, \ldots, m\}$, $\{t_{v_i} : v_i = 1, 2, \ldots, n\}$, $c > 0$.

Output. The minimal total cost.

Solution:

To solve this problem, we can use a modified Dijkstra's algorithm. Here's the pseudo-code for the modified Dijkstra's algorithm:

```
function modifiedDijkstra($n$, $m$, $E$, $t$, $c$)
Initialize arrays $min\_cost\_car[1 \ldots n]$ and $min\_cost\_train[1 \ldots n]$ with $\infty$
$min\_cost\_car[1] \leftarrow 0$
$min\_cost\_train[1] \leftarrow t[1]$
Initialize priority queue $Q$
$Q$.push((0, 1, 'car'))
$Q$.push(($t[1]$, 1, 'train'))
while $Q$ is not empty do
    ($current\_cost$, $current\_city$, $current\_mode$) $\leftarrow$ $Q$.extract-min()
    if ($current\_mode$ is 'car' AND $current\_cost > min\_cost\_car[current\_city]$) OR ($current\_mode$ is 'train' AND $current\_cost > min\_cost\_train[current\_city]$) then
        Continue
    end if
    if $current\_mode$ is 'car' then
        for each ($current\_city$, $next\_city$, $w$) in $E$ do
            $new\_cost \leftarrow current\_cost + w$
            if $new\_cost < min\_cost\_car[next\_city]$ then
                $min\_cost\_car[next\_city] \leftarrow new\_cost$
                $Q$.push(($new\_cost$, $next\_city$, 'car'))
            end if
        end for
        $new\_cost \leftarrow current\_cost + t[current\_city] + c$
        if $new\_cost < min\_cost\_train[current\_city]$ then
```

```
            $min\_cost\_train[current\_city] \leftarrow new\_cost$
            $Q$.push(($new\_cost, current\_city$, 'train'))
        end if
    else if $current\_mode$ is 'train' then
        for each ($current\_city, next\_city, w$) in $E$ do
            $new\_cost \leftarrow current\_cost + c$
            if $new\_cost < min\_cost\_train[next\_city]$ then
                $min\_cost\_train[next\_city] \leftarrow new\_cost$
                $Q$.push(($new\_cost, next\_city$, 'train'))
            end if
        end for
    end if
end while
return min($min\_cost\_car[n], min\_cost\_train[n]$)
```

Time Complexity Analysis:

1. The priority queue operations (insertion and extraction) take O(log n) time.
2. Each edge is relaxed at most once for each mode (car and train), resulting in O(m) relaxations.
3. Therefore, the overall time complexity of the algorithm is O((n + m) log n).