Homework Sheet 11

Author	Matriculation Number	Tutor
Abdullah Oğuz Topçuoğlu	7063561	Maryna Dernovaia
Ahmed Waleed Ahmed Badawy Shora	7069708	Jan-Hendrik Gindorf
Yousef Mostafa Farouk Farag	7073030	Thorben Johr

Exercise 1

a) Bellman–Ford Algorithm

Initialization

$$d^{(0)} = [0, \infty, \infty, \infty, \infty, \infty, \infty].$$

Since $|V| = 7$, Bellman–Ford performs 6 iterations.

Iteration 1

$$d^{(1)} = [0, 3, -2, -1, 7, 2, 2].$$

Iteration 2

$$d^{(2)} = [0, 3, -2, -1, 7, 2, 1].$$

Iteration 3

$$d^{(3)} = [0, 3, -2, -1, 7, 2, 1].$$

Iteration 4

$$d^{(4)} = [0, 3, -2, -1, 7, 2, 1].$$

Iteration 5

$$d^{(5)} = [0, 3, -2, -1, 7, 2, 1].$$

Iteration 6

$$d^{(6)} = [0, 3, -2, -1, 7, 2, 1].$$

Final Bellman–Ford Distances

$$\boxed{d = [0, 3, -2, -1, 7, 2, 1]}.$$

b) Dijkstra's Algorithm

Although the graph contains negative-weight edges, we nevertheless execute Dijkstra's algorithm from source node 1.

Initialization

$$d = [0, \infty, \infty, \infty, \infty, \infty, \infty].$$

Iteration 1 (settle node 1)

$$d = [0, 3, -1, 0, \infty, \infty, \infty].$$

Iteration 2 (settle node 3)

$$d = [0, 3, -1, 0, \infty, \infty, 2].$$

Iteration 3 (settle node 4)

$$d = [0, 3, -1, 0, \infty, 3, 2].$$

Iteration 4 (settle node 7)

$$d = [0, 3, -1, 0, \infty, 3, 2].$$

Iteration 5 (settle node 2) Relaxing $(2, 4)$ would improve the distance to node 4, but node 4 has already been settled. Thus no update occurs.

Iteration 6 (settle node 6)

$$d = [0, 3, -1, 0, 8, 3, 2].$$

Iteration 7 (settle node 5) No improvement occurs.

Final Dijkstra Distances

$$\boxed{d = [0, 3, -1, 0, 8, 3, 2]}.$$

Comparison

Node	Bellman–Ford	Dijkstra
1	0	0
2	3	3
3	-2	-1
4	-1	0
5	7	8
6	2	3
7	1	2

The distances computed by Dijkstra's algorithm differ from the correct shortest-path distances found by Bellman–Ford. This confirms that Dijkstra's algorithm fails on graphs with negative-weight edges, even when no negative-weight cycles are present.

Exercise 2

```
MinimizeFurthestDelivery(G, s1, s2, s3):
//we return the distance array from running Dijkstra and save it to d1,d2, d3 respective
  d1 = Dijkstra(G, s1)
  d2 = Dijkstra(G, s2)
  d3 = Dijkstra(G, s3)

  best = infinity
  t = 0

  for each v in V:
      temp = max(d1[v], d2[v], d3[v])
      if temp < best:
          best = temp
          t = v

  return t
```

Correctness Proof We show that the algorithm returns a city t that minimizes the delivery time to the furthest client.

Since all edge weights in G are nonnegative, Dijkstra's algorithm correctly computes shortest-path distances. Therefore, for each $i \in \{1, 2, 3\}$ and for every vertex $v \in V$, the value $d_i[v]$ computed by the algorithm equals the minimum travel time from s_i to v.

For any vertex $v \in V$, the delivery time to the furthest client located at s_1, s_2, s_3 is

$$M(v) = \max\{d(s_1, v), d(s_2, v), d(s_3, v)\}.$$

The algorithm explicitly computes $M(v)$ for each vertex v and maintains a variable `best` that stores the smallest value of $M(v)$ seen so far, together with the corresponding vertex t. After the loop terminates, `best` equals

$$\min_{v \in V} M(v),$$

and the stored vertex t satisfies

$$t = \arg\min_{v \in V} \max\{d(s_1, v), d(s_2, v), d(s_3, v)\}.$$

Hence, the algorithm returns a city that minimizes the delivery time to the furthest client and is therefore correct.

Runtime Analysis Each call to Dijkstra's algorithm runs in $O((m+n)\log n)$ time. The algorithm runs Dijkstra three times, which results in a total time of

$$O((m + n)\log n).$$

The final loop iterates once over all vertices and performs constant-time operations per vertex, which takes $O(n)$ time. This does not affect the asymptotic running time.

Exercise 3

We will traverse the board using BFS and thats it. The board is a directed graph. The ladders and snakes create edges between the nodes and we have edges between consecutive nodes if there is no snake head at that node. Assuming the board data is given as two lists (snakes and latters) and a number n for grid size, we will first create a graph and then run BFS on it.
Pseudocode:

```
int SmallestNumberOfSteps(int n, list of (int, int) snakes, list of (int, int) ladders)
  // Create graph
  graph G
  for i from 1 to n do
    G.addNode(i)

  for i from 1 to n-1 do
      G.addEdge(i, i+1) // if there is a snake head, we will remove the edge below
  for each (i, j) in snakes do
    if G.hasEdge(i, i+1) then
      G.removeEdge(i, i+1)
    G.addEdge(i, j)
  for each (i, j) in ladders do
    G.addEdge(i, j)  // assuming ladders can't be on the same cells as snakes

  // BFS
  queue Q
  array visited // initially all false
  array distance // initially all infinity
  Q.enqueue(1)
  visited.add(1)
  distance[1] = 0

  while not Q.isEmpty() do
    current = Q.dequeue()
    for each neighbor in G.getNeighbors(current) do
      if neighbor not in visited then
        visited.add(neighbor)
        distance[neighbor] = distance[current] + 1
        Q.enqueue(neighbor)

  return distance[n]
```

Correctness:
In the graph representation the "the smallest number of steps to move from 1 to n" is equivalent to finding the shortest path from node 1 to node n. BFS is guaranteed to find the shortest path in an unweighted graph. Since each move from one node to another is considered as one step, BFS will correctly compute the smallest number of steps required to reach node n from node 1.

Running Time Analysis:
Creating the graph takes O(n + $|S|$ + $|L|$) time. Running BFS takes O(n + m) time, where m is the number of edges in the graph. In our case, m is at most O(n + $|S|$ + $|L|$) since each node can have edges to its next node, and additional edges from snakes and ladders. Thus, the overall time complexity of the algorithm is O(n + $|S|$ + $|L|$).

Exercise 4

a)

```
W'(u,v):
  return (n+1) * w(u,v) +1
Relax (u,v):
  if d[u] + w'(u,v) < d[v]
     d[v] = d[u] + w'(u,v)
     partent[v] = u
Algo (node s):
   Dijkstra s
```

b)

```
W'(u,v):
  return (n+1) * w(u,v) - 1
Relax (u,v):
  if d[u] + w'(u,v) < d[v]
     d[v] = d[u] + w'(u,v)
     partent[v] = u
Algo (node s):
   Dijkstra s
```

Correctness Proof Let $G = (V, E, w)$ be a graph with positive integer edge weights and let $n = |V|$. Let $d(s, t)$ denote the shortest-path distance from s to t with respect to w.

Part (a): Minimum number of edges Define modified edge weights

$$w'(u, v) = (n + 1) \cdot w(u, v) + 1.$$

For any path P,

$$\sum_{e \in P} w'(e) = (n+1) \sum_{e \in P} w(e) + |P|.$$

Since all edge weights are positive integers, if two paths P_1 and P_2 satisfy

$$\sum w(P_1) > \sum w(P_2),$$

then

$$\sum w(P_1) \geq \sum w(P_2) + 1.$$

Any simple path contains at most $n-1$ edges, hence

$$|P_1| - |P_2| \geq -(n-1).$$

Therefore,

$$\sum w'(P_1) - \sum w'(P_2) \geq (n+1) \cdot 1 - (n-1) > 0.$$

Thus, minimizing total weight with respect to w' implies minimizing total weight with respect to w. Among all shortest paths, the first term is constant, so minimizing w' is equivalent to minimizing $|P|$. Since $w'(e) > 0$ for all edges, Dijkstra's algorithm correctly finds such a path.

Part (b): Maximum number of edges Define modified edge weights

$$w'(u, v) = (n+1) \cdot w(u, v) - 1.$$

For any path P,

$$\sum_{e \in P} w'(e) = (n+1) \sum_{e \in P} w(e) - |P|.$$

By the same argument as in part (a), minimizing w' enforces

$$\sum w(P) = d(s, t).$$

Among all shortest paths, minimizing $\sum w'(P)$ is equivalent to maximizing $|P|$. Moreover,

$$w'(u, v) \geq (n+1) \cdot 1 - 1 = n \geq 1,$$

so all modified weights are positive and Dijkstra's algorithm applies.

Runtime Analysis Computing modified weights takes $O(m)$ time. Running Dijkstra's algorithm with a binary tree takes

$$O((m+n) \log n).$$