Problem Set 7

Due dates: Electronic submission of the pdf file of this homework is due on **3/24/2023 before 11:59pm** on canvas. The homework must be typeset with LaTeX to receive any credit. All answers must be formulated in your own words.

Watch out for additional material that will appear on Thursday! Deadline is on Friday, as usual.
Name: Ayushri Jain

Resources.

Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms, 3rd edition, The MIT Press, 2009 (or 4th edition)

On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment. Furthermore, I have disclosed all resources (people, books, web sites, etc.) that have been used to prepare this homework.

Signature: Ajain

Read the chapters on “Elementary Graph Algorithms” and “Single-Source Shortest Paths” in our textbook before attempting to answer these questions.

Problem 1 (20 points)**.** Give an algorithm that determines whether or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(V)$ time, independent of the number $|E|$ of edges.

Solution. To check whether an undirected graph $G = (V, E)$ contains a cycle, we can perform a customized depth-first search (DFS) traversal on the graph. We will run DFS on each unvisited vertex. When we visit a vertex, we will mark it as visited in visit array (size V) and in addition, we will maintain a parent array (size V) to keep track of the parent of each vertex in the DFS tree. And then we will call DFS on each adjacent vertex of u. If we encounter a vertex v that is already visited, and v is not the parent of the current vertex u, then we can declare that cycle exists in the graph. If this declaration never happens, then it means that cycle is not present in the graph G. The time complexity for this algorithm is $O(V)$ because each vertex is visited once and each vertex in its adjacency list is also visited at most once and if any vertex is visited again, cycle is present and we stop, hence independent of number of edges.

Problem 2 (20 points)**.** Given a weighted, directed graph $G = (V, E)$ with no negative-weight cycles, let m be the maximum over all vertices $v \in V$ of the minimum number of edges in a shortest path from the source s to v. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in $m + 1$ passes, even if m is not known in advance.

Solution. Since m is the maximum number of edges in a shortest path from source s to v, then each vertex v must have got their shortest path estimate $d[v]$ in m iterations according to the path relaxation property. Thus, after m iterations, $d[v]$ will not change according to upper bound property. So, we can maintain a flag which can tell us if any edge v got relaxed in the current iteration or not. Let us initialize a variable $flag = 1$ just before the outer for loop. Now let's insert an if condition inside the outer for loop just above the inner for loop which breaks the outer for loop if $flag = 0$. If the condition is not satisfied, we need to enter the inner for loop. So let us make $flag = 0$ each time when the condition is not met just before we enter the inner loop, i.e. in the inner for loop, flag is initially 0. The inner for loop goes over the edges and tries to relax the edges and if any edge is relaxed, we update the flag value $flag = 1$. If no edges are relaxed in the current iteration (means i^{th} iteration), then the flag value will remain 0 and we will get to know this in the next iteration ($(i + 1)^{th}$ iteration). Then based on our if condition, we can terminate the outer for loop. And as mentioned previously, this particular iteration i after which no edges are relaxed will be m. Hence, we will be able to terminate the algorithm in $m + 1$ iterations even if m is unknown.

Problem 3 (20 points)**.** Suppose that we change line 4 of Dijkstra's algorithm in our textbook to the following.

4 **while** $|Q| > 1$

This change causes the while loop to execute $|V| - 1$ times instead of $|V|$ times. Is this proposed algorithm correct? [Use the version of Dijkstra's algorithm from the textbook]

Solution. Currently, the line 4 of Dijkstra's algorithm contains condition which checks if priority queue is empty or not : while $|Q| \neq \emptyset$. If we change it to while $|Q| > 1$, it means that we do not want to extract the last vertex in priority queue. Let that leftover vertex be denoted by u and if u is reachable from source s then there will exist a shortest path from s to x (or set of vertices) to u. In this case, the edge $x - u$ will get relaxed, i.e. $d[u]$ gets updated when vertex x is extracted in the Dijkstra's algorithm. In the other case if vertex u can't be reached from source s, the $d[u]$ value will remain at ∞ which is correct. Thus, the algorithm is still correct even if we do not pick out the last vertex from the priority queue.

Problem 4 (40 points)**.** Help Professor Charlie Eppes find the most likely escape routes of thieves that robbed a bookstore on Texas Avenue in College Station. The map will be published on Thursday evening. In preparation, you might want to implement Dijkstra's single-source shortest path algorithm, so that you can join the manhunt on Thursday evening. [Edge weight 1 means very desirable street, weight 2 means less desirable street]

Solution. Considering that the graph is bidirectional, these are the edges and their corresponding weights -

1. 1-2 or 2-1 : weight 1
2. 1-11 or 11-1: weight 1
3. 2-3 or 3-2: weight 1
4. 2-21 or 21-2: weight 1
5. 3-4 or 4-3: weight 1
6. 3-8 or 8-3 : weight 2
7. 4-5 or 5-4: weight 1
8. 5-6 or 6-5: weight 2
9. 5-7 or 7-5: weight 1
10. 5-22 or 22-5: weight 1
11. 6-7 or 7-6: weight 1
12. 7-8 or 8-7: weight 1
13. 8-9 or 9-8 : weight 1

14. 9-10 or 10-9: weight 1
15. 9-19 or 19-9: weight 1
16. 10-11 or 11-10 : weight 1
17. 10-18 or 18-10: weight 2
18. 11-12 or 12-11: weight 2
19. 11-17 or 17-11 : weight 1
20. 12-13 or 13-12 : weight 2
21. 13-14 or 14-13 : weight 2
22. 13-21 or 21-13: weight 1
23. 14-15 or 15-14 : weight 1
24. 14-16 or 16-14: weight 1
25. 14-20 or 20-14: weight 1
26. 16-17 or 17-16: weight 2
27. 17-18 or 18-17 : weight 2
28. 20-21 or 21-20 : weight 2
29. 20-22 or 22-20: weight 1
30. 21-22 or 22-21: weight 2

We know that in Dijkstra's algorithm, the nodes are inserted in a min priority queue (min heap).
Assumption 1: If weights are same, then the node with less numeric value will be in front of min priority queue.
Assumption 2: We only update the $d[i]$ value if it is less than current value.
We need to start from source 1.
As per the Dijkstra's algorithm, initially all the entries in the d array is ∞ where $d[i]$ represents the weight of the shortest path from source found so far and since 1 is the source node, $d[1] = 0$. This is our iteration 0. All the nodes are in the priority queue with 1 at the front.
Priority queue: 1, 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22
Iteration 1 : 1 is at front, its neighbors can be updated, i.e. nodes 2, 11.

There is an edge between 1 and 2, so $d[2] = 1$.

There is an edge between 1 and 11, so $d[11] = 1$.
1 is removed from priority queue.
Priority queue: 2,11, 3,4,5,6,7,8,9,10,12,13,14,15,16,17,18,19,20,21,22
Iteration 2 : 2 is at front and its neighbors are 3 and 21 for which d values can be updated.

There is an edge between 2 and 3, so $d[3] = d[2] + 1 = 1 + 1 = 2$.

There is an edge between 2 and 21, so $d[21] = d[2] + 1 = 1 + 1 = 2$.
2 is removed from priority queue.
Priority queue: 11, 3,21, 4,5,6,7,8,9,10,12,13,14,15,16,17,18,19,20,22
Iteration 3 : 11 is at front and its neighbors are 1,10,12,17 for which d values can be updated.

$d[1]$ is not updated since it will be greater than current value.

There is an edge between 11 and 10, so $d[10] = 1 + 1 = 2$.

There is an edge between 11 and 12, so $d[12] = 1 + 2 = 3$.

There is an edge between 11 and 17, so $d[17] = 1 + 1 = 2$.
11 is removed from priority queue.
Priority queue: 3,10,17,21, 12, 4,5,6,7,8,9,13,14,15,16,18,19,20,22
Iteration 4 : 3 is at front and its neighbors are 2,4,8 for which d values can be updated.

$d[2]$ is not updated since it will be greater than current value.

There is an edge between 3 and 4, so $d[4] = 2 + 1 = 3$.

There is an edge between 3 and 8, so $d[8] = 2 + 2 = 4$.
3 is removed from priority queue.
Priority queue: 10,17,21, 4,12, 8, 5,6,7,9,13,14,15,16,18,19,20,22
Iteration 5 : 10 is at front and its neighbors are 9,11,18 for which d values can be updated.

There is an edge between 10 and 9, so $d[9] = 2 + 1 = 3$.

$d[11]$ is not updated since it will be greater than current value.

There is an edge between 10 and 18, so $d[18] = 2 + 2 = 4$.
10 is removed from priority queue.
Priority queue: 17,21, 4,9,12, 8,18, 5,6,7,13,14,15,16,19,20,22
Iteration 6 : 17 is at front and its neighbors are 11,16,18 for which d values can be updated.

$d[11]$ is not updated since it will be greater than current value.

There is an edge between 17 and 16, so $d[16] = 2 + 2 = 4$.

$d[18]$ is not updated based on assumption 2.
17 is removed from priority queue.
Priority queue: 21, 4,9,12, 8,16,18, 5,6,7,13,14,15,19,20,22
Iteration 7 : 21 is at front and its neighbors are 2,13,20,22 for which d values can be updated.

$d[2]$ is not updated since it will be greater than current value.

There is an edge between 21 and 13, so $d[13] = 2 + 1 = 3$.

There is an edge between 21 and 20, so $d[20] = 2 + 2 = 4$.

There is an edge between 21 and 22, so $d[22] = 2 + 2 = 4$.
21 is removed from priority queue.
Priority queue: 4,9,12,13, 8,16,18,20,22, 5,6,7,14,15,19
Iteration 8 : 4 is at front and its neighbors are 3,5 for which d values can be updated.

$d[3]$ is not updated since it will be greater than current value.

There is an edge between 4 and 5, so $d[5] = 3 + 1 = 4$.
4 is removed from priority queue.
Priority queue: 9,12,13, 5,8,16,18,20,22, 6,7,14,15,19
Iteration 9 : 9 is at front and its neighbors are 8,10,19 for which d values can be updated.

$d[8]$ is not updated since it will be greater than current value.

$d[10]$ is not updated since it will be greater than current value.

There is an edge between 9 and 19, so $d[19] = 3 + 1 = 4$.
9 is removed from priority queue.
Priority queue: 12,13, 5,8,16,18,19,20,22, 6,7,14,15
Iteration 10 : 12 is at front and its neighbors are 11,13 for which d values can be updated.

$d[11]$ is not updated since it will be greater than current value.

$d[13]$ is not updated since it will be greater than current value.
12 is removed from priority queue.
Priority queue: 13, 5,8,16,18,19,20,22, 6,7,14,15
Iteration 11 : 13 is at front and its neighbors are 12,14,21 for which d values can be updated.

$d[12]$ is not updated since it will be greater than current value.

There is an edge between 13 and 14, so $d[14] = 3 + 2 = 5$.

$d[21]$ is not updated since it will be greater than current value.
13 is removed from priority queue.
Priority queue: 5,8,16,18,19,20,22, 14, 6,7,15
Iteration 12 : 5 is at front and its neighbors are 4,6,7,22 for which d values can be updated.

$d[4]$ is not updated since it will be greater than current value.

There is an edge between 5 and 6, so $d[6] = 4 + 2 = 6$.

There is an edge between 5 and 7, so $d[7] = 4 + 1 = 5$.

$d[22]$ is not updated since it will be greater than current value.
5 is removed from priority queue.
Priority queue: 8,16,18,19,20,22, 7,14, 6, 15
Iteration 13 : 8 is at front and its neighbors are 3,7,9 for which d values can be updated.

$d[3]$ is not updated since it will be greater than current value.

$d[7]$ is not updated based on assumption 2.

$d[9]$ is not updated since it will be greater than current value.
8 is removed from priority queue.
Priority queue: 16,18,19,20,22, 7,14, 6, 15
Iteration 14 : 16 is at front and its neighbors are 14,17 for which d values can be updated.

$d[14]$ is not updated since it will be greater than current value.

$d[17]$ is not updated since it will be greater than current value.
16 is removed from priority queue.
Priority queue: 18,19,20,22, 7,14, 6, 15
Iteration 15 : 18 is at front and its neighbors are 10,17 for which d values can be updated.

$d[10]$ is not updated since it will be greater than current value.

$d[17]$ is not updated since it will be greater than current value.
18 is removed from priority queue.
Priority queue: 19,20,22, 7,14, 6, 15
Iteration 16 : 19 is at front and its neighbor is 9 for which d values can be updated.

$d[9]$ is not updated since it will be greater than current value.
19 is removed from priority queue.
Priority queue: 20,22, 7,14, 6, 15
Iteration 17 : 20 is at front and its neighbors are 14,21,22 for which d values can be updated.

$d[14]$ is not updated since it will be greater than current value.

$d[21]$ is not updated since it will be greater than current value.

$d[22]$ is not updated since it will be greater than current value.
20 is removed from priority queue.
Priority queue: 22, 7,14, 6, 15
Iteration 18 : 22 is at front and its neighbors are 5,20,21 for which d values can be updated.

$d[5]$ is not updated since it will be greater than current value.

$d[20]$ is not updated since it will be greater than current value.

$d[21]$ is not updated since it will be greater than current value.
22 is removed from priority queue.
Priority queue: 7,14, 6, 15
Iteration 19 : 7 is at front and its neighbors are 5,6,8 for which d values can be updated.

$d[5]$ is not updated since it will be greater than current value.

$d[6]$ is not updated since it will be greater than current value.

$d[8]$ is not updated since it will be greater than current value.
7 is removed from priority queue.
Priority queue: 14, 6, 15

Iteration 20 : 14 is at front and its neighbors are 13,15,16,20 for which d values can be updated.

$d[13]$ is not updated since it will be greater than current value.

There is an edge between 14 and 15, so $d[15] = 5 + 1 = 6$.

$d[16]$ is not updated since it will be greater than current value.

$d[20]$ is not updated since it will be greater than current value.
14 is removed from priority queue.
Priority queue: 6,15
Iteration 21 : 6 is at front and its neighbors are 5,7 for which d values can be updated.

$d[5]$ is not updated since it will be greater than current value.

$d[7]$ is not updated since it will be greater than current value.
6 is removed from priority queue.
Priority queue: 15
Iteration 22 : 15 is at front and its neighbor is 14 for which d values can be updated.

$d[14]$ is not updated since it will be greater than current value.
15 is removed from priority queue.
Priority queue is empty now, so we stop.
The array d is evolved in the following manner after each iteration (Iteration(column), Node(row)):

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	∞	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	∞	∞	2	2	2	2	2	2	2	2	2	2	2	2	2	2
4	∞	∞	∞	∞	3	3	3	3	3	3	3	3	3	3	3	3
5	∞	∞	∞	∞	∞	∞	∞	∞	4	4	4	4	4	4	4	4
6	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	6	6	6
7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	5	5	5
8	∞	∞	∞	∞	4	4	4	4	4	4	4	4	4	4	4	4
9	∞	∞	∞	∞	∞	3	3	3	3	3	3	3	3	3	3	3
10	∞	∞	∞	2	2	2	2	2	2	2	2	2	2	2	2	2
11	∞	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12	∞	∞	∞	3	3	3	3	3	3	3	3	3	3	3	3	3
13	∞	∞	∞	∞	∞	∞	∞	3	3	3	3	3	3	3	3	3
14	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	5	5	5	5	5
15	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
16	∞	∞	∞	∞	∞	∞	4	4	4	4	4	4	4	4	4	4
17	∞	∞	∞	2	2	2	2	2	2	2	2	2	2	2	2	2
18	∞	∞	∞	∞	∞	4	4	4	4	4	4	4	4	4	4	4
19	∞	∞	∞	∞	∞	∞	∞	∞	∞	4	4	4	4	4	4	4
20	∞	∞	∞	∞	∞	∞	∞	4	4	4	4	4	4	4	4	4
21	∞	∞	2	2	2	2	2	2	2	2	2	2	2	2	2	2
22	∞	∞	∞	∞	∞	∞	∞	4	4	4	4	4	4	4	4	4

	16	17	18	19	20	21	22
1	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1
3	2	2	2	2	2	2	2
4	3	3	3	3	3	3	3
5	4	4	4	4	4	4	4
6	6	6	6	6	6	6	6
7	5	5	5	5	5	5	5
8	4	4	4	4	4	4	4
9	3	3	3	3	3	3	3
10	2	2	2	2	2	2	2
11	1	1	1	1	1	1	1
12	3	3	3	3	3	3	3
13	3	3	3	3	3	3	3
14	5	5	5	5	5	5	5
15	∞	∞	∞	∞	6	6	6
16	4	4	4	4	4	4	4
17	2	2	2	2	2	2	2
18	4	4	4	4	4	4	4
19	4	4	4	4	4	4	4
20	4	4	4	4	4	4	4
21	2	2	2	2	2	2	2
22	4	4	4	4	4	4	4

According to question, we require distance of 6,8,9,15,16,22 from node 1 which are given below -

$d[6] = 6$

$d[8] = 4$

$d[9] = 3$

$d[15] = 6$

$d[16] = 4$

$d[22] = 4$

In case it is required to answer the shortest path to the most likely of the six destinations, then the node 9 has the shortest path = 3 (according to weight) among the given 6 destinations and the explanation is given above.