EC 504
Final Exam, Spring 2021

May, 2021

Instructions

- Put your name on the first page of your answer sheet.
- This exam is open book/notes, but no consultations with anyone other than the instructors are permitted during the exam. Specifically, don't use your keyboard unless you are using the chat window. During the exam, you must be logged into Zoom with your camera on. Try to log in 10 minutes before the exam.
- You have one hour and 55 minutes to complete this exam. There are 8 questions. All work must be shown to be counted.
- At the end of the exam, you will have 15 minutes to scan and upload your work to Gradescope. Please do not delay as Gradescope will close at 11:10 Boston Time.

1. (20 pts) Answer True or False to each of the questions below, **AND YOU MUST PROVIDE SOME FORM OF EXPLANATION (very brief is OK)**. Each question is worth 2 points. Answer true only if it is true as stated, with no additional assumptions.

   (a) In a forward star representation of a directed graph, the edges must be sorted by start node.
   Solution: True; that is the definition of a forward star.

   (b) Suppose T_1, T_2 are minimum spanning trees for the same graph G. Then the largest edge weight of T_1 must be the same weight as the largest edge weight of T_2.
   Solution: TRUE. Otherwise, can find a contradiction by taking the largest of the two largest weights out of that tree and replacing it with an edge from the other tree.

   (c) In Bellman-Ford's shortest path algorithm, the temporary distance labels D assigned to vertices which have not yet been scanned (i.e. left the work queue) can be interpreted as the minimum distance among all paths from the start vertex to the given vertices with intermediate vertices restricted to the vertices that have already been scanned (left the work queue).
   Solution: False. That is Dijkstra's algorithm.

   (d) Consider a minimum spanning tree T in a weighted, undirected graph G. Suppose we decrease the weight of a single edge that is in the minimum spanning tree T. Then, it is possible that T is no longer a minimum spanning tree in the modified graph.
   Solution: False. Kruskal's algorithm would try to insert this edge into the minimum spanning tree earlier, but it would find the same edges.

   (e) Consider a minimum spanning tree problem in a connected, weighted graph with positive and negative weights which can include negative weight cycles. Then, both Prim's and Kruskal's algorithms can be used to find the minimum weight spanning tree in this graph.
   Solution: TRUE. The algorithms do not care as to whether the weights are positive or negative.

   (f) Consider an NP-complete decision problem. Let n denote the size of a problem instance. Then, any candidate solution for an instance can be evaluated to determine whether the answer is true or not in time $O(n^k)$ for some nonnegative integer k. **Solution:** True. This is what non-deterministic machines do: evaluate candidates.

   (g) A problem is said to be NP-complete if it can be reduced to an instance of every other problem which is in class NP. **Solution:** False. Every other problem in NP can be reduced to an instance of this problem, and the reduction must be polynomial.

   (h) Given a weighted, directed, dense graph with non-negative weights, consider the problem of finding whether the longest simple path in the graph that starts with a given origin node o has length greater than or equal to a threshold T. This problem is in class NP. **Solution:** True. Verifying the length of a candidate path can be done in polynomial time.

(i) The class NP consists of all decision problems which cannot be solved in time $O(n^k)$ for some positive integer k, where n is the size of the problem instance description in bits.

Solution: False. Plenty of decision problems in NP can be solved in linear time, since it includes P.

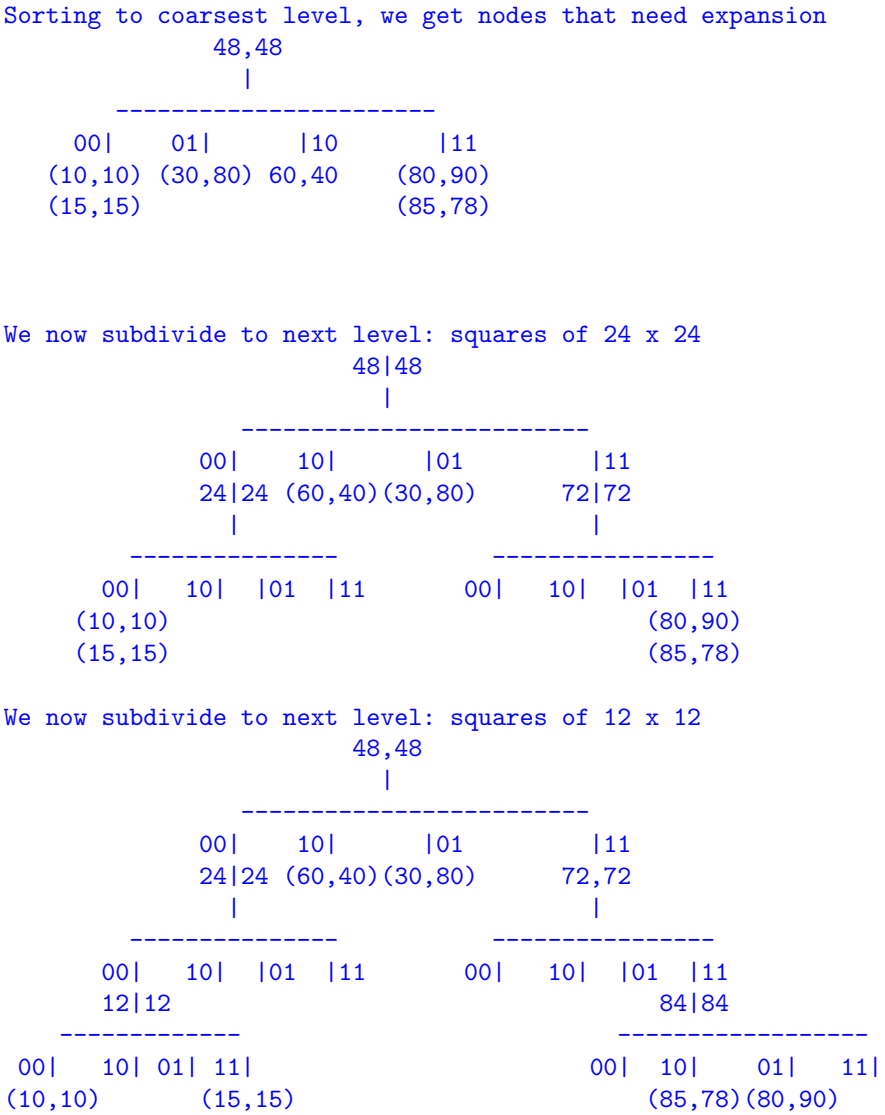
(j) Every problem in class NP is also NP-Hard. **Solution:** False; only NP-complete problems in class NP are NP-Hard.

2. (15 pts) Consider the following 2-dimensional set of points:

$$(80, 90), (85, 78), (60, 40), (10, 10), (15, 15), (30, 80)$$

(a) Arrange the elements in a PR quadtree with value range 0-96 for both the first and second coordinates. Display the answer as a tree, with children arranged left-to-right in the following order: SW, SE, NW, NE.

Solution:

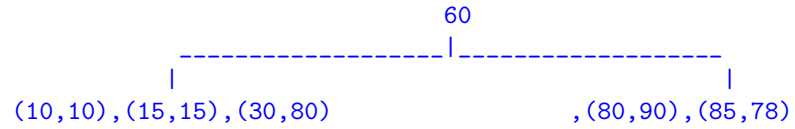


(b) Form a balanced 2-d tree using the median of the elements to split, with all the keys at the leaves of the tree. When there are an even set of points to split at a median value, select the larger of the two possible median values as the median navigation key. Assume also that when a key is equal to the navigation key, it is added to the right subtree (that is, greater than or equal).

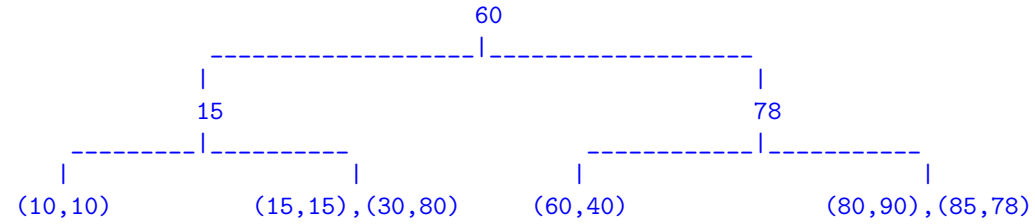
Solution: First, find the median of the first coordinates. Sorting by the first coordinate yields:

$$(10, 10), (15, 15), (30, 80), (60, 40), (80, 90), (85, 78)$$

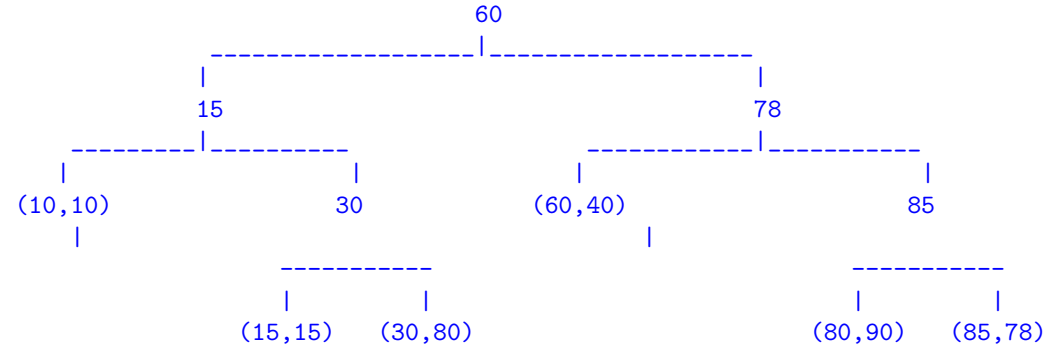
The median value is either 30 or 60, so we select 60 as indicated in the problem, resulting in



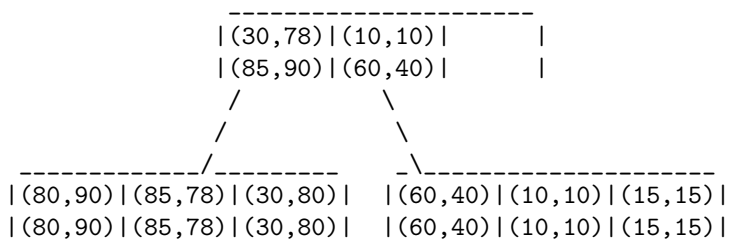
The median of the second coordinate on the right is 78, so we branch on 78. For the left list, the median is 15, so we branch on 15. We get:



Repeat the process for each leaf with two entries, branching the median of the x-values.

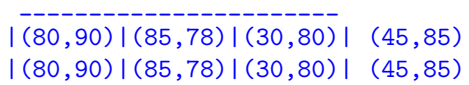


(c) Below is an R-tree of order $M = 3, m = 1$, where the minimum number of children (and keys) is 1, formed using linear splitting, with the following variation: when splitting an area defined by points, use as seeds the two points that are farthest in distance from each other. The R-tree already contains the six points: $(80, 90), (85, 78), (60, 40), (10, 10), (15, 15), (30, 80)$. Rectangles are displayed as pairs of points corresponding to the SW and NE corners. Assume we insert into areas with smallest perimeter increase.

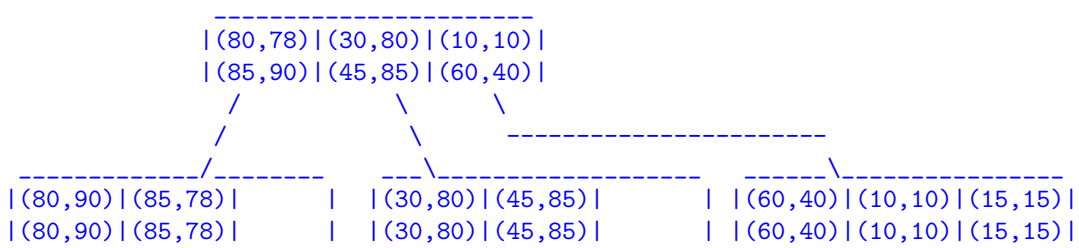


Insert the following key in the R-trees: (45,85), and display the resulting R-tree.

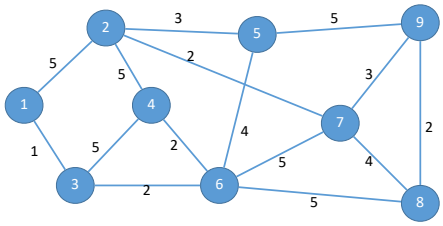
Solution: Inserting (45,85) will go into the left rectangle, and saturate the left leaf. We must split that leaf, resulting in



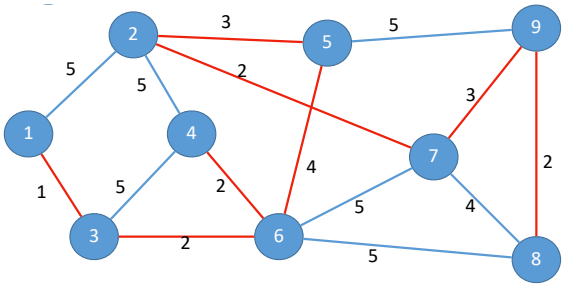
The farthest points are (80,90) and (30,80). Using the linear split rule, (45,85) has least area increase when grouped with (30,80), and (85,78) has least area increase when grouped with (80,90). The resulting tree is:



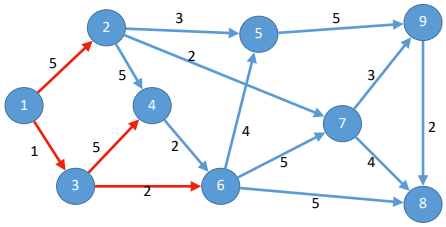
3. (10 points) Consider the undirected graph on the right, where the numbers on edges consist of weights. Find a minimum spanning tree for the graph on the right. Indicate what algorithm you are using, and show some of the work associated with that algorithm. Display the minimum spanning tree graphically, and show the total weight of the minimum spanning tree.



Solution: The MST is shown, with total weight 19.



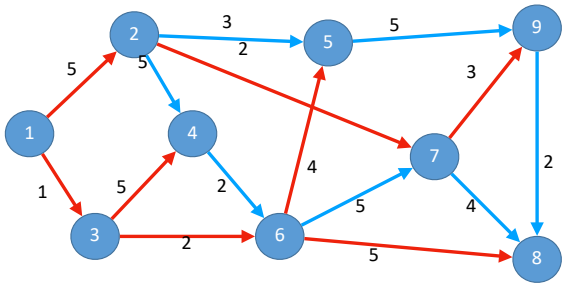
4. (10 points) Consider the directed graph on the right, where the numbers on edges consist of distances. We are going to find a shortest path tree from node 1 to every other node using Dijkstra's algorithm. The figure shows the partial tree grown after five iterations, scanning vertex 1, then vertex 3, then vertex 6, then vertex 2, and then vertex 4.



(a) What are the next two vertices to be scanned (i.e. popped from the priority queue), and what are the distances from 1 to those vertices?

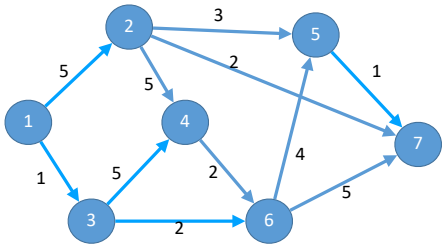
Solution: The next two vertices will be 5 and 7, both at distance 7 from vertex 1.

(b) Draw the shortest path tree from vertex 1 to all other vertices.

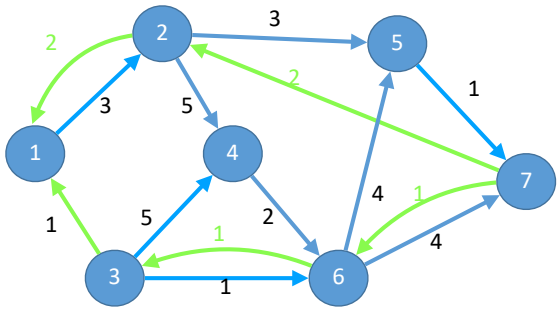


Solution:

5. (15 points) Consider the directed graph on the right, where the numbers on edges consist of edge capacities. Our goal is to find the max flow from vertex 1 to vertex 7. As such, we will do it using both the Ford-Fulkerson algorithm, and the Preflow-push algorithm, as guided by the questions below.



(a) Assume that we have made an augmentation of one unit of flow on path 1 - 3 - 6 - 7, and another augmentation of 2 units of flow on path 1 - 2 - 7. Construct the residual network, with residual capacities on each forward and reverse edge.



Solution:

(b) On that residual network, find the shortest (least number of hops) augmenting path from vertex 1 to vertex 7, and determine its capacity.

Solution: The path is 1 - 2 - 5 - 7, capacity 1.

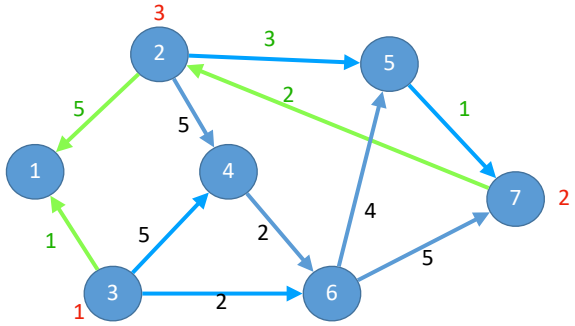
(c) What is the max flow from vertex 1 to vertex 7 in this network?

Solution: It is 6. There will be one more augmenting path, capacity 2.

(d) Assume we are starting over with no flow in the network, and we are going to use the Preflow-Push algorithm to find the max flow. Compute the initial vertex excesses and the initial vertex distances to vertex 7, including the initial distance label for vertex 1.

Solution: The initial distance labels are $d(1) = 7, d(3) = d(4) = 2, d(2) = d(5) = d(6) = 1$.
The initial excess is $e(2) = 5$, $e(3) = 1$.

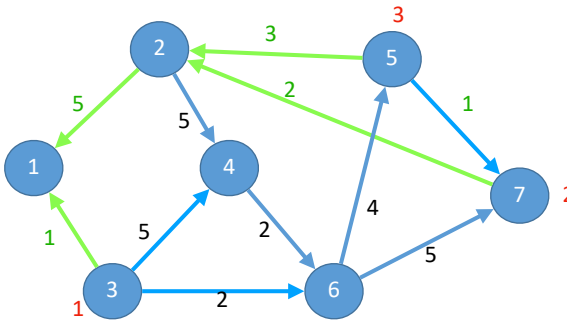
(e) For the first iteration, we start with the largest excess, and push 2 units of flow from vertex 2 to vertex 7, a saturating push. Draw the residual graph that remains after this push, showing the excess at vertices with excess.



Solution:

(f) Assume we want to execute another push from vertex 2, that still has excess 3. Describe the steps required to perform this push, and draw the residual graph that remains after the push is complete, together with excesses at the vertices with excess.

Solution: First, vertex 2 has label 1. There is no admissible edge, so vertex 2 must relabel. It has vertex 5 as a neighbor with vertex 1, so it relabels to 2. Then, it pushes all 3 excess units of flow to vertex 5. The resulting residual network is shown below.



6. (10 pts) Consider the following knapsack problem: We have six objects with different values and V_i and sizes w_i. The object values and sizes are listed in the table below:

Object number	1	2	3	4	5	6
Value	10	11	12	13	14	15
Size	1	1	2	3	4	3

(a) Suppose we are given a knapsack with total size 11, and you are allowed to use fractional assignments. What is the maximum value that fits in the knapsack for the above tasks?

Solution: Sort them by value per unit size in decreasing order, we get the following order: 1, 2, 3, 6, 4, 5.

The optimal fractional schedule will include all of objects 1, 2, 3, 6, 4 and 1/4 of object 5, for a total value of 64.5.

(b) Assume we are not allowed fractional assignments. For the knapsack with total size 11, the dynamic programming requires computing $Opt(j, k)$, the best value considering only objects $1, \ldots, j$, using total size k. The table below has partially computed these numbers. Compute the remaining elements in the table, $Opt(j, k), j = 5, 6; k = 5, 6, 7, 8, 9, 10, 11$.

Capacity-> Max .Task	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	10	10	10	10	10	10	10	10	10	10	10
2	0	11	21	21	21	21	21	21	21	21	21	21
3	0	11	21	23	33	33	33	33	33	33	33	33
4	0	11	21	23	33	34	36	46	46	46	46	46
5	0	11	21	23	33							
6	0	11	21	23	33							

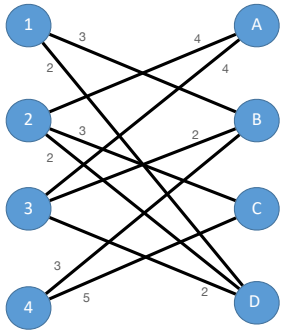
Solution: The table is

Capacity-> Max .Task	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	10	10	10	10	10	10	10	10	10	10	10
2	0	11	21	21	21	21	21	21	21	21	21	21
3	0	11	21	23	33	33	33	33	33	33	33	33
4	0	11	21	23	33	34	36	46	46	46	46	46
5	0	11	21	23	33	34	36	46	47	48	60	60
6	0	11	21	23	33	36	38	48	49	51	61	62

The optimal integer value is 62, including objects 1, 2, 3, 5, 6.

(c) What is the maximum value of the tasks that fit in the knapsack with size 11?

7. (10 pts) Consider the sparse assignment problem shown in the figure on the right, where the costs of each edge are indicated in the figure. We are interested in computing the minimum cost assignment using the successive shortest path algorithm.



(a) To begin the algorithm, what are the initial prices assigned to vertices A, B, C, D?

Solution: Prices are the minimum cost of every incoming edge: Hence, price of A is 4, B is 2, C is 3, D is 2.

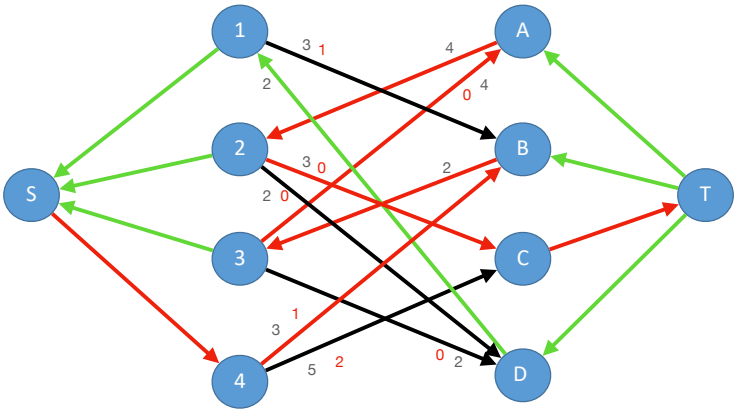
(b) Given the initial prices, what are the reduced costs of the edge $\{1, B\}$ and the edge $\{4, C\}$?

Solution: Reduced cost of $\{1, B\}$ is 3 - 2 = 1. Reduced cost of $\{4, C\}$ is 5 - 3 = 2.

(c) Now, assume we have proceeded with the algorithm, and we have made assignments of vertex 1 to vertex D, vertex 2 to vertex A, and vertex 3 to vertex B, all which have reduced cost 0, and the prices are still the same as the initial prices. Compute and show the residual network with the reduced cost distances, and find the final shortest augmenting path on the network.

(d) Using the original costs, what is the cost of the optimal assignment you found?

Solution: The residual network is shown below, with reduced costs in red, using the prices on the table. The shortest augmenting path is shown in red. The final assignments after augmentation are 1-D, 2-C, 3-A, 4-B. Optimal cost is 12.



v	d(v)	P
1	0	0
2	0	0
3	0	0
4	0	0
A	0	4
B	0	2
C	0	3
D	0	2

8. (10 points) Consider an array of non-negative integers, with positions numbered from 1 to n, where the value at a position represents the size of the maximum jump forward that can be made from that position. For instance, $[1, 2, 4, 0]$ means that starting at position 1, you can take a jump forward of size 1 to position 2; starting at position 2, you can take a jump forward of either 1 or 2 positions; starting at position 3, you can take a jump forward of 1, 2, 3, or 4 positions, and starting at position 4, you can't take any jumps forward.

(a) Given an array of nonnegative integers $[a_1, a_2, \ldots, a_n]$, describe an algorithm for finding the minimum number of jumps required to go from position 1 to position n, where the size of the jump from position i must be less than or equal to a_i.

Solution: There are many ways of doing this. One way is to use dynamic programming. A more elegant way is to reduce it to a shortest path on a directed acyclic graph with all edges having distance 1. Form the graph where each position is a vertex (1 to n), and there is an edge from position i to positions $i + 1, \ldots, i + a_i$, representing the possible transitions, with each edge having length 1. Find the shortest path in this graph from vertex 1 to vertex n using breadth-first search (e.g. Dijkstra), or Bellman-Ford, or any other approach).

(b) Estimate the computational complexity of the above algorithm as a function of n.

Solution: $O(n^2)$ using BFS.

(c) Use the algorithm to compute the minimum number of jumps to go from position 1 to position 11 for the array

$$[2, 3, 1, 3, 4, 2, 1, 2, 2, 1, 0]$$

Solution: Breadth first search distances: $D[1] = 0, D[2] = 1, D[3] = 1, D[4] = 2, D[5] = 2, D[6] = 3, D[7] = 3, D[8] = 3, D[9] = 3, D[10] = 4, D[11] = 4$.

Minimum number is 4 jumps, with path $1 \to 2 \to 5 \to 9 \to 11$.

As a comment, the following algorithm was proposed, and claimed to be $O(n)$: For each point, keep track of the step you can take. Iterate from position 1 to position n, until you can reach the end. From that point, backtrack to get the minimum number of jumps. Claim this is $O(n)$.

Note that this does not work unless you store the minimum number of jumps to reach position i at the same time. Hence, this will be $O(n^2)$, as at each step, you may have to update $O(n)$ minimum number of jumps.

To illustrate, consider the array

$$[1, 4, 5, 5, 1, 1, 1, 3, 1, 1, 1]$$

Here is the sequence of maximum reach we compute: 2,6,8,9,9,9,9,11,11,11.

Hence, we know that the last jump should be from position 8. However, we must remember what the best way to reach position 8 was with minimum number of jumps, which is in 3 jumps.