CS 460, HOMEWORK 4

INSTRUCTOR: HOA VU

Each question is worth 25 points.

When you are asked to design an algorithm, do the following: a) describe the algorithm, b) explain (or more rigorously prove) why it is correct, and c) provide the running time.

DO NOT look up solutions online. Any violation will be reported. I will be happy to provide some hints or ideas if you get really stuck during class or office hours.

[Erickson] denotes the book by Jeff Erickson (available for free online at `http://jeffe.cs.illinois.edu/teaching/algorithms/`).

[DPV] denotes the book by Dasgupta, Papadimitriou, and Vazirani (the required textbook).

For graph problems: If you use a standard graph algorithm that we went over in class, you don't have to rewrite the algorithm. However, you must specify the graph that you construct from a non-graph input (i.e., which are the vertices, edge? Weighted or unweighted? Directed or undirected? If directed, you must specify the edges' directions).

- **Question 1:**
  (1) Consider a map that is discretized into n by n squares. Each square has a number that represents the altitude. This map is represented by a 2D array A. Specifically, $A[i, j]$ contains the altitude of the square (i, j).
  Suppose one can only go from a square to an adjacent square with a lower altitude. Design an algorithm and state the running time (in terms of n) that outputs whether it is possible to go from the top left square to the bottom right square. The algorithm should be as efficient as possible.
  (2) Design an algorithm, as efficient as possible, that outputs all the squares that can reach the bottom right square (again, assume that one can only go from a square to an adjacent square with a lower altitude). In particular, the algorithm should output a list of squares that if you start there, it is possible to reach the bottom right square.
  (3) Suppose now that it is always possible to go from a square to its adjacent squares. However, the required energy for such a move is the absolute difference between the altitudes of the two squares. In other words, going from square (i, j) to an adjacent square (i', j') requires $|A[i, j] - A[i', j']|$ unit of energy. Design an algorithm that find the path (if there is one) from the top left

square to the bottom right square that uses the least amount of energy. State the running time.

- **Question 2:** Suppose we are given a graph whose edge weights are integers in the set $\{1, 2, \ldots, 100\}$ and a source vertex s. Show how to find the shortest paths from s to all other vertices in $O(|E| + |V|)$ time.
- **Question 3:** Given an array of numbers $A[1 \ldots n]$, provide an algorithm that rearranges the entries of A so that after the rearrangement, the sum $\sum_{i=1}^{n} iA[i]$ is maximized. Make sure you prove the correctness.
- **Question 4:** Prove that for any cycle C in the graph, if the weight of an edge e of C is larger than any of the individual weights of all other edges of C, then this edge cannot belong to an MST.
- **Question 5 (extra credit)**: Read and summarize the proof of Dijkstra's algorithm from the link on Canvas.

Question 1:

1.Consider a map that is discretized into n by n squares. Each square has a number that represents the altitude. This map is represented by a 2D array A. Specifically, A[i, j] contains the altitude of the square (i, j). Suppose one can only go from a square to an adjacent square with a lower altitude. Design an algorithm and state the running time (in terms of n) that outputs whether it is possible to go from the top left square to the bottom right square. The algorithm should be efficient as possible.

Answer: Let define each square as a vertex. Each square has a number that represents the altitude and it only can go from a square to an adjacent square with a lower altitude. Connect all vertices from the top left square to the bottom right square and convert them to the graph. Run DFS/BFS on this graph to find the path from the top left square to the bottom right square.

2. Design an algorithm, as efficient as possible, that outputs all the squares that can reach the bottom right square (again, assume that only go from a square to an adjacent square with a lower altitude). In particular, the algorithm should output a list of squares that if you start there, it is possible to reach the bottom right square.

Answer: Let define each square as a vertex. Similarly the first question, we would connect all the vertices from the top left square to the bottom right square with only go from a square to an adjacent square with a lower altitude. We convert them to the graph G. Let call G' is the graph we reverse all the edges' direction. Run DFS/BFS in the graph G' to see if there are vertices are reachable from the bottom right square. Then we get the output would be a list of

vertices (squares) that are reachable from the bottom right square.

3. Suppose now that it is always possible to go from a square to its adjacent squares. However, the required energy for such a move is the absolute difference between the altitudes of the two squares. In other words, going from square (i, j) to an adjacent square (i', j') requires |A[i,j] - A[i',j']| unit of energy. Design an algorithm that find the path (if there is one) from the top left square to the bottom right square that uses the least amount of energy. State the running time.

Answer: Since we define each square as a vertex and now each vertex can go to its adjacent square and using the required energy for such a move which is the length between vertices. Hence, to design an algorithm that find the path (if there is one) from the top left square to the bottom right square that uses the least amount of energy which means we need to design an algorithm to find the shortest path from the top left square to the bottom right square. Convert them to the graph G then using Dijkstra's algorithm to find the shortest path in graph G.

Question 2: Suppose we are given a graph whose edge weights are integers in the set {1, 2, 3, …, 100} and a source vertex s. Show how to find the shortest paths from s to all other vertices in O(|V| + |E|) time.

Answer: We will use BFS for this problem. However, one important observation about BFS is that all edges need to have the same weight. So in this case, we can modify the graph and split all edges of weight {2, 3, …, 100} into several edges of weight 1 each such that replacing the edge u-v with a path as follows u-uv1-uv2-..-v (uv1, uv2, … are new nodes) with the weight of each edge in between as 1. Do this for

every edge. Now that all weights of the resultant graph are 1, then we can use BFS to find the shortest path in the modified graph. Since the running time of BFS is O(|V| + |E|) so the total running time for this problem is O(|V| + |E|).

Question 3: Given an array of numbers A[1...n], provide an algorithm that rearranges the entries of A so that after the rearrangement, the sum iA[i] is maximized. Make sure to provide the correctness.
Answer: If we arrange the array in sorted (increase) order, we can see that the minimum index will multiply with minimum number and maximum index will multiply with maximum order.
Steps to follow:
1) Sort the array
2) Create a variable sum to store the final answer.
3) Traverse the array and non-decreasing the value of the sum
sum = sum + arr[i] * i
4) Return the sum.

Running time: O(N*log(N))

Proof: the idea is basically based on the fact that the largest value should be scaled maximum and the smallest value should be scaled minimum. So we multiply the minimum value of i with the minimum value of arr[i]. So sorting the given array in non-decreasing order and compute the sum of i * arr[i].
We can see arr[i] * i + arr[i+1]*(i+1) >= arr[i] * (i+1) + arr[i+1] * i
<=> arr[i] * i + arr[i+1] * i + arr[i+1] >= arr[i] * i + arr[i] + arr[i+1] * i
<=> arr[i+1] >= arr[i]

Question 4: Prove that for any cycle C in the graph, if the weight of an edge e of C is larger than any of the individual weights of all other

edges of C, then this edge cannot belong to an MST.

Answer: Proof by contradiction. We assume that the edge which is in C and larger than any of the individual weights of all other edges of C belongs to an MST - T.
The cycle C includes edge e so there exists an edge e' in C which is not in T because T is a tree and does not contain a cycle.
Create a new tree T' by removing e from T and adding e'.
Since the weight of e is the largest then T' has smaller sum of weights compared to T which is a contradiction since T is a MST.
Therefore, the edge e cannot belong to an MST.

Question 5 (extra credit): Read and summarize the proof of Dijkstra's algorithm from the link on Canvas.

Answer: Suppose we have G is the input graph, s is the source vertex, l(uv) is the length of an edge from u to v and V is the set of vertices. Let d(v) be the label found by the algorithm and h(v) be the shortest path distance from s to v. We want to prove d(v) = h(v) for every vertex v at the end.
Base case: When |R| = 1, the only node in R is s, for which we're obviously found the shortest path.
Induction hypothesis: Let u be the last vertex added to R and R' = R ∪ {u}. Suppose for every x in R', d(x) = h(x). We need to show that d(u) = h(u) to complete the proof.
Let say Q is the shortest path from s to u and l(Q) < d(u).
Q starts in R' and at some leaves R' (to get to u which is not in R').
Let xy be the first edge along Q that leaves R'. Qx will be the subpath from s to x. Then, l(Qx) + l(xy) < l(Q).
But d(x) is the shortest path then d(x) < l(Qx) or d(x) + l(xy) < l(Q)
And y is adjacent to x, d(y) must have been updated by the algorithm,

then $d(y) < d(x) + l(xy)$.
Since u has the smallest distance then $d(u) < d(y)$.
We combine all of these equations, we have:
$d(u) < d(y) < d(x) + l(xy) < l(Q) < d(u)$
Then $d(u) < d(u)$ which is a contradiction.
Therefore, there is no shorter path Q must exist and so $d(u) = h(u)$ and $R = V$.