HW4

January3, 2022

question 1:

a.

1. we do in-order traversal for each tree into two sorted arrays.

2. we traverse the sorted arrays and compare pair of items and return False they are not equal (or the array's length are different) or True if we reached the end of both the arrays.

the total runtime of the algorithm is linear: $O(n_1 + n_2)$

b.

For each node we add two new properties, max and min, which will hold the minimum and maximum keys of the node's subtree respectively. min will be equal to the key of the node in case it doesn't have a left child, the same goes of max property if the node doesn't have a right child.

in order to do that we need to change some of the AVL functions:

1. INSERT function - after the insertion we update the max and min the same way we do for the heights.

Starting from the parent of the inserted node, we loop up until we reach the root, updating the max and min properties of each node along the route, if the inserted node was small or bigger, respectively. if we need to do rotations as well, we also update the max and min after each rotation - the same way we would for the heights.

2. DELETE function - after we delete the node needed to be deleted whether it is the node itself or its successor. we operate the same way as we did in INSERT, updating the max and min of the nodes along the route until we reach the root and in every rotation along the way.

question 2:

a.

H1 - linear hashing:

0	**65**
1	**11**
2	**13**
3	**37**
4	
5	
6	
7	
8	
9	
10	**10**
11	**49**
12	**25**

Number of probes:

65	**10**	**25**	**49**	**11**	**13**	**37**
0	0	0	1	3	2	5

H2 - quadratic hashing:

0	**65**
1	**13**
2	**11**
3	
4	
5	
6	
7	**37**
8	
9	
10	**10**
11	**49**
12	**25**

Number of probes:

65	**10**	**25**	**49**	**11**	**13**	**37**
0	0	0	1	2	1	3

H3 - double hashing:

0	**65**
1	**37**
2	
3	**11**
4	
5	
6	
7	**13**
8	
9	
10	**10**
11	**49**
12	**25**

Number of probes:

65	**10**	**25**	**49**	**11**	**13**	**37**
0	0	0	1	1	1	1

b.

For all the tables the load factor will be the same $\alpha = \frac{7}{13}$

Question 3:

We'll use the Queue data structure (doubly linked list) to create a Queue of orders, each order will hold the vegetable type and delivery address. Additionally, we'll create a hush table which will hold items that composed from the vegetable type and number of orders for this type.

- Insert - we create a new order item and insert the order into the back of the Queue. in each new order we search for the vegetable type in the hush table and if find it we increment the number of orders for that vegetable type by 1 else we add the vegetable type with number of orders equals to 1. runtime - O(1)
- Ship - we retrieve the first item in the head of the Queue, then we search the vegetable in the hash table and decrease the number of orders for that vegetable type by 1. If the number of orders is now 0, we remove the vegetable item from the hash table. runtime - O(1)
- query(vegetable) - we do a search for the vegetable type and retrieve and number of orders if we find it, else we'll return 0. - runtime - O(1)

both of these data structure uses O(m) memory, so the total space complexity is O(m).

Question 4:

a.

True,

The sum of all the degrees is equal to twice the number of edges. Since the sum of the degrees is even and the sum of the degrees of vertices with even degree is even, the sum of the degrees of vertices with odd degree must be even. If the sum of the degrees of vertices with odd degree is even, there must be an even number of those vertices.

b.

True,

As learned in class a connected graph has Euler cycle iff all of its nodes have even degrees.

so if we have a graph with n vertices, if n is even there could be at most n vertices with odd degree, if n is odd there could be at most n-1 vertices with odd degree so the total sum of degree will be even. thus, we can divide them to maximum of $\frac{n}{2}$ pairs or $\lfloor \frac{n}{2} \rfloor$ if n is odd, and add an edge to each one of them. now all the vertices will have even degree and the graph will have a Euler cycle. so the number of added edged cannot exceed $\lfloor \frac{n}{2} \rfloor$

c.

False,

If we had a graph with 3 vertices, all of them connected to each other.

- In BFS starting in vertex 1 results in adding paths (1,2),(1,3) to the tree and the two vertices to the queue. now the algorithm dequeue vertex 2, since all of its adjacents are marked it will move and dequeue vertex 3 which in return will not add any path to the tree. the end result will be a tree where 1 as the root and 2,3 are its children
- In DFS 1 will continue with 2. Then, it will call DFS recursively on 2, and since its adjacents are 1,3 and 1 was already marked, it will take 3. All adjacents to 3 are marked so when the recursive process ends the DFS will result in a tree that goes from vertex 1 to vertex 2 to vertex 3.

d.

True,

Let $G = (V_1, V_2, E)$ be a bipartite graph.

We can sort all the sinks into V_1 and all the sources to V_2. we know there will be no edges between any of the vertices in V_2 because all of them are sources. Similarly there will be no edges between any of the vertices in V_1as all of them are sinks.

e.

False,

Let $G = (V, E)$ directed graph such $\mid V \mid = 3$ and for $\forall u,v \in V$ $(v, u) and (u, v) \in E$. in such a graph for every vertex $d_{in}(v) + d_{out}(v) = 4 > 3 = \frac{6}{2}$ but G does not have a Hamiltonian path.

Question 5:

1. The SSC are $W_1 = (a, b, e), W_2 = (f, g), W_3 = (c, d, h)$.
2. The SCC graph $= (V = (W_1, W_2, W_3), E = ((W_1, W_3), (W_1, W_2), (W_3, W_2)))$

a.

let's prove by contradiction,

- let's claim that there is no path from u to u'.
- We know that there is a edge from W to W', which means that there is a node in W that has an edge to a node in W'.
- because W and W' are SSC every node in W is reachable from any other node in W, the same goes for W'.
- if $u \in W$and $u' \in W'$, u can reach any the node in W, including the one with an edge to W' and u' can reach any the node in W, including the one with an edge to W'.
- Thus, there must be a path between u and u'

b.

If there is path from W toW' this means that there is same path $W_1, W_2..W$ an edge from W to W' so as proven in proof a there is a path from any nodes $u \in W$ and $u' \in W'$.

c.

Let's prove by contradiction,

let's claim that there is a cycle in G^{SCC}.

Let W, W´ be two different SCC nodes on this cycle.

This means there is a path in G^{SCC} from W to W´ and also a path from W´ to W.

Let $u \in W$ and $u' \in W'$. According to proof b: there is a path in G from u to $u\prime$, and also a path from

$u\prime$ to u. This means that u and u´ belong to the same strongly connected component. In contradiction to the assumption that W and W´ are two different strongly connected components.

Question 6:

Algorithm: .

1. Construct $G^* = (V^*, E^*)$, where:
   (a) $V^* = V^1 \bigcup V^2 \bigcup V^3$ consists of three copies of the original vertex set V.
   (b) E^* : for any $(u, v) \in E \setminus E\prime$add the edges $(u^1, v^1), (u^2, v^2), (u^3, v^3)$ to E^*. For any $(u, v) \in E\prime$ add the edges $(u^1, v^2), (u^2, v^3)$ *to* E^*

2. Run BFS(s^1) to find the shortest path in G^* from the first copy of s to any other vertex.

3. Return for each vertex $v \in V : mind(s^1, v^1), d(s^1, v^2), d(s^1, v^3)$

Correctness:

Observation:

1. A path $v_1^1, v_2^1, ..., v_k^1$ *in* G^* is considered as a path with no red edges in $G : v_1, v_2, ..., v_k$ of the same length.

2. A path $v_1^1, ..., v_j^1, v_{j+1}^2, ..., v_k^2$ *in* G^* is considered as a path in $G : v_1, ..., v_k$ of the same length, where (v_j, v_{j+1}) is a single red edge on the path.

3. A path $v_1^1, ..., v_j^1, v_{j+1}^2, ..., v_r^2, v_{r+1}^3, ..., v_k^3$ *in* $G*$ corresponds to a path in $G : v_1, ..., v_k$ of the same length, where the only two red edges $(v_j, v_{j+1}), (v_r, v_{r+1})$.

Following this observation, we get that the shortest path from s to v with at most two red edges corresponds to the shortest path from s^1 to either v^1 , v^2 or v^3 in G^* .

BFS finds the shortest path to any vertex v using the labels.