CS2040S 2022/2023 Semester 1 Final

MCQ

This section has 10 questions and is worth 30 marks. 3 marks per question.

Do all questions in this section.

1. We have 10 distinct integers in an AVL balanced binary search tree. How many of those integers could be the root of the tree?

a. 2
b. 4
c. 6
d. 8

Ans:

Consider the smallest element that can be the root. To make the root smallest, we would want the left subtree to be the AVL tree with the fewest nodes with height h and the right subtree to be the AVL tree with most nodes (namely a complete binary tree) with height h + 1. Since we have 10 nodes in the tree, we will need h = 2, in which case the left subtree contains 2 nodes and the right subtree contains 7 nodes. So the smallest element that can be the root is the 3-rd smallest integer. With the same argument, the largest element that can be the root is the 3-rd largest integer. All integers in between can also be the root. In total, we have 6 integers that can possibly be the root.

2. Consider a min heap of size n implemented using a d-ary tree ($d \geq 2$), where each node can have at most d children. The original property that each node is smaller than its children still holds. Suppose that the tree is a complete d-ary tree, and the min heap is implemented using a 1-based compact array where the navigation operation to the parent and children of a node takes O(1) time. What would be the time needed to extract the minimum element?

a. O(log n)
b. $O(d \log_d n)$
c. $O(\log_d n)$
d. O(n)

Ans:

The height of the tree would be $\log_d n$. On each level of the tree, you will need to compare the node with all d children to determine which node to swap. In total, we will need $O(d \log_d n)$ time.

3. Which of the following description about trees is incorrect?

a. Inserting an element in a BST with *n* nodes takes worst case O(*n*) time
b. Assuming a graph where the edge weights are all unique, within the cut set of any cut of the graph, there is one and only one edge which will be in the MST of the graph
c. When we insert an element into an AVL tree, the time needed for re-balancing is O(log n) in the worst case
d. To find the MST of a dense graph (there are $E = O(V^2)$ edges), Prim's or its variant is preferred over Kruskal's algorithm in general.

Ans:

option a - If the BST is extremely left-skewed, inserting an element could require us to go through all nodes in the BST which cost O(n) time.
option b - We only guarantee that in every cut in a graph, at least one edge will be in the minimum spanning tree. As a counter-example, a complete graph with 3 nodes will have a cut where both the 2 edges are in MST.
option c - As shown in the lectures, insertion only requires 1 rebalancing operation which can be done in O(1) time and since we only need to go from the root to some deepest leaf for the insertion, time taken is the height of the AVL which is O(log n) time.
option d - Prim's algorithm (and its variant specifically for dense graphs) is generally faster than Kruskal's algorithm for dense graphs, as you don't have to sort all edges in one go.

4. In the adjacency list representation of a graph, Adj[v] stores an arraylist of all neighbors of v. We can probably improve it by replacing the arraylist Adj[v] with an unordered map (hash table) or ordered map (AVL tree), with keys being the neighboring nodes.

   Which of the following is correct?

a. If we use hash tables, inserting an edge will run in worst case O(1) time
b. If we use hash tables, BFS will run faster than the original arraylist representation
c. If we use AVL trees, BFS will run faster than the original arraylist representation
d. If we use AVL trees, searching for an edge will run in worst case O(log *d*) time, where *d* is the maximum number of neighbors

Ans:

This is true by the property of AVL trees. Using hash tables, inserting an edge can take O(d) time, depending on the collision resolution. BFS will not run faster with maps, since in BFS, all we need is iterating through all neighbours of a node. Maps don't provide a faster iteration through all neighbours than arraylist.

5. Below is the result of a traversal of a complete binary search tree.

   11, 29, 20, 50, 91, 65, 41

   Which of the following may be the possible way of the traversal?

a. in-order traversal
b. pre-order traversal
c. post-order traversal
d. pre-order traversal but switch the order of left and right subtree traversal to right and left subtree traversal

Ans:

It is not in sorted order, so it is not in-order traversal; The first 3 elements is not decreasing, so it is not pre-order traversal; the first 3 elements is also no increasing therefore it is also not option d.

6. Which of the following transformation of the graph is correct. That is, after the transformation, the algorithm returns the same results as the correct answer before the transformation, supposing the correct answer is unique?

a. To solve the SSSP problem for a given source s in a weighted directed graph with possibly negative edge weights, we can add a positive constant to all edge weights to eliminate the negative edge weights. We can then run original Dijkstra's algorithm on s and get the correct shortest path in terms of the vertices and edges involved.
b. To find the maximum spanning tree, we can multiply all edge weights by −1 and apply Prim's or Kruskal's algorithm, then for the resultant tree simply multiply all edges by -1 again to get the maximum spanning tree.
c. To solve the APSP problem on an undirected weighted graph, we can first compute the MST of the graph and then run Floyd-Warshall on the MST instead.
d. To find the path with the minimum product of edge weights between two vertices in a general directed weighted graph, we can always replace edge weights with the negative of their logarithms, then we can simply use original Dijkstra's algorithm to find the required path (in terms of vertices and edges involved)

Ans:

option a - Simply adding a constant to all edge weights will increase the total cost of each path differently. The weight of paths with more edges will be increased more. As a result, the shortest path might not be the same after the transformation.
option b - This is correct. The maximum spanning tree is just the minimum spanning tree when all edge weights are reversed.
option c - The shortest path between two vertices might not be in the spanning forest of the graph.
option d - We cannot apply logarithm if there are edges of negative weight.

7. We have a directed graph with V ($V > 0$) vertices, E ($E > 0$) directed edges (no bi-directed edges) and if we view the directed edges as undirected edges we have C ($C > 1$) connected components. Which of the following observations cannot lead to a conclusion that the graph (with directed edges) does not contain a cycle?

a. $E = V - C$
b. Kosaraju's algorithm returns V strongly connected components
c. DFS topological sort algorithm fails to return a valid topological ordering of the graph.
d. Assign all edges a weight of −1, and running Bellman-Ford algorithm from source vertex 0 doesn't report a negative cycle

Ans:
Option a - Suppose we view all the directed edges as undirected edges, having E = V - C edges and C connected components indicate that each of the connected components must be a tree. This implies that there also cannot exist a cycle in the directed graph.
Option b - As discussed in the tutorials, if Kosaraju's algorithm returns V strongly connected components, the size of each SCC must be one, which implies that there cannot be cycles (otherwise there must exist some SCC with size >= 2).
Option c - DFS topological sort also works for disconnected graphs, and a failure to return a valid topological sort indicate that there are cycles in the graph.
Option d - Even if there is a negative cycle, if none of the vertices in the cycle is reachable from vertex 0, then Bellman-Ford will not report it (since all reachable vertices ultimately will converge to there SP costs).

Since for option c, DFS topological sort failing to return a valid topological ordering only lead to the conclusion that there is some cycle in the graph and not that there is no cycle, so option c is also correct. Thus all students who choose option c will get the marks.

8. Apart from BFS/DFS, we can also use union-find disjoint sets (UFDS) to count the number of components of an undirected graph. We can start with V disjoint sets, each containing a single vertex. Then for each edge (u, v) in the graph, we call unionSet(u, v).

   Which of the following description of the resulting UFDS is incorrect?

a. The number of connected components is equal to the number of vertices whose parent is itself.
b. The maximum height of all disjoint sets in the UFDS is O($\alpha(V)$)
c. The number of connected components is equal to the number of unique returned values of find(v) for each vertex v in the graph.
d. The running time is O($E\ \alpha(V)$)

Ans:

As shown in tutorial 6, the maximum height of a disjoint set in a UFDS with V items can be O(logV) and not O($\alpha(V)$).

9. In a binary **min** heap with *n* nodes, you would like to find the **largest** element inside the heap. What is the maximum number of nodes you need to check to find the largest element?

a. n
b. ceil(n/2)
c. ceil(log n)
d. 1

Ans:

We only need to look for the largest element in all of the leaf nodes. The maximum number of leaves in a binary tree with n nodes is reached when the tree is complete, and the number is ceil(n/2)

10. You are given a undirected, connected and positively weighted graph of V vertices and E edges which is stored in an Adjacency list. You are told that all shortest paths in the graph have a special property: the weights of the edges along the shortest path from source to destination is always strictly increasing in value.

    Given a source s and a destination s', which of the following statements will find the shortest path from s to s' in the stated time complexity.

    I) perform modified Dijkstra from s to find Shortest Path (SP) cost from s to s' in O(ElogV) time

    II) perform bellman ford from s to find SP cost from s to s' in O(VE) time

    III) perform one-pass bellman ford from s to find SP cost from s to s' in O(V+E) time

    IV) convert Adjacency list to edgelist in O(V+E) time and sort the edgelist in ascending order. Go through the edglist from smallest to largest weighted and relax the edges once. This will find the SP cost from s to s' in O(ElogE) time.

a. I and II are true only
b. II is true only
c. I, II and IV are true only
d. All the statements are true

Ans:

I is true since it is a connected graph, V-1 <= E < $O(V^2)$, thus O(ElogE) = O(ElogV)

II is obviously true

III is not true if the graph is not a tree.

IV is actually true, since the SP has edges with increasing edge weights, a "good order" to relax all the edges in one-pass is to do so in increasing order of the edge weights (this ensures all vertices along the SP will have their SP found 1st and the SP cost propagated to the successor vertex). Time complexity is dominated by the sorting of the edgelist thus this will take O(ElogE) time.

Analysis

This section has 4 questions and is worth 16 marks. 4 marks per question.

Please select True or False and then type in your reasons for your answer.

Correct answer (true/false) is worth 2 marks.

Correct explanation is worth 2 marks. Partially correct explanation worth 1 marks.

Do all questions in this section.

11. For a directed weighted graph where number of edges E = $O(V^2)$, the fastest algorithm to find APSP in the graph is only Floyd-Warshall among those discussed in CS2040S.

Ans: False

You can use the "dense graph" variant for Dijkstra which runs in O(V^2) time per source vertex, so it will take O(V^3) too to solve APSP.

Grading Scheme

2 marks - Suggesting that if we perform the variant of Dijkstra's for dense graph for every node.
2 marks - Mentioning that **if the graph is a DAG**, then running one-pass Bellman-Ford from every node can be done in O(V^3) time.
2 marks - Mentioning that **if the edges in the graph has the same weight**, then running BFS from every node can be done in O(V^3) time.
0 mark - Claiming that no other algorithm can run in O(V^3) time.
0 mark - Considering the case where the graph is a tree, which is impossible if E=O(V^2).

12. An alternate way to perform fast heap create in O(N) time is to copy the N keys into the 1-based compact array then start from index 0 and go to index *heapsize* performing **ShiftUp** on every index along the way.

Ans: False.
This does create a correct heap of the N keys, but does it in O(NlogN) time. This is equivalent to simply inserting each key into the heap and calling shiftup <- slow heap create.
The first key inserted is at index 1, then 2nd key inserted is at index 2 and so on .....

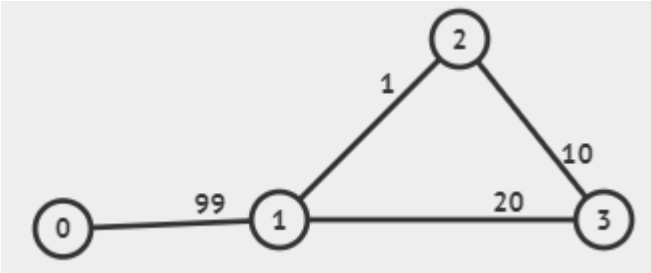
Grading Scheme

2 marks - Claiming that the alternative way costs O(N log N) time instead.
2 marks - Since we are starting at index 0 of the 1-based compact array, shifting up the element at index 1 to index 0 may cause undefined behavior. *Not intended* as question should have been "start from index 1".
1 mark - Simply saying that we cannot start from index 0. This is not totally correct since as taught in lecture, we can make up of a 0-based array but ignore index 0 and start from index 1 to make it 1-base.
0 mark - Claiming that the resulting heap is incorrect (violating the requirement of heap).
0 mark - Raising counterexamples where we cannot do a shift up a single node in a min heap because it is already the smallest, claiming that we need to check the value before doing "shift up". This is not valid reason because this check is already in the ShiftUp function

13. Any valid minimax path between a pair of vertices u and v in an undirected weighted graph G must also be a valid path between u and v in some MST of G.

Ans: False

An example is given below.



In this graph, 0-1-3 is a valid minimax path from 0 to 3 but it is not a valid path from 0 to 3 in any MST since 1-3 is the largest edge in the cycle 1-2-3-1 and thus should be removed in all MSTs.

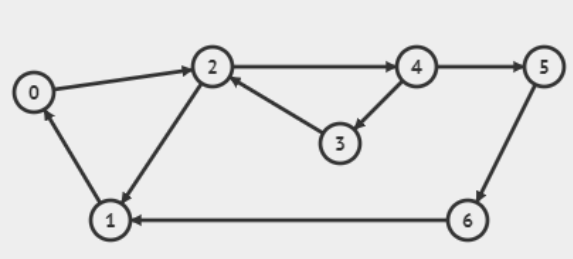
Grading Scheme

2 marks - Giving counterexamples or describing a case where the minimax path is **definitely not** in the MST.
1 mark - Correctly identify the relationship between the two: paths in MST must be minimax, but minimax might not necessarily be in MST, but fail to give a clear condition/counterexample where minimax path is definitely not in the MST.
0 mark - Simply saying that the definition or the purpose for MST and minimax is different, and claim that the results must be different, disregarding the fact that MST gives the minimax paths.
0 mark - Giving an incorrect counterexample, typically messing up the concepts of minimax path, MST and shortest path.
0 mark - Giving a counterexample where the MST is not unique, and claim that a minimax path is not in the MST, whereas it is actually in another valid MST.

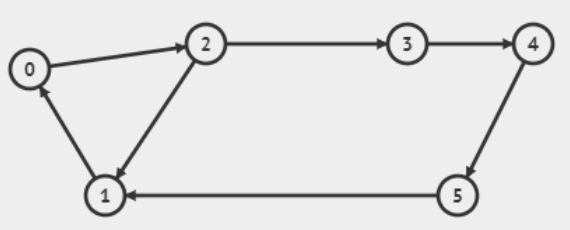
14. Removing any vertex in an SCC of a simple directed graph will result in the remaining vertices (of the SCC) no longer forming one SCC anymore.

Ans: False

An example is the graph below which is made up of 1 SCC



If we remove vertex 3, we will still get 1 SCC as follows



Grading Scheme

2 marks - Giving counterexamples or clearly describing a case where after deleting a vertex in an SCC, the remaining still forms an SCC.
1 mark - Misread the question and giving an example showing that the SCC still holds after **deleting an edge**. These examples typically will also work as a counterexample for deleting a vertex.
0 mark - Giving an incorrect counterexample, e.g. deleting a vertex that is not in the same SCC as the other nodes, messing up the concepts of SCC and connected components, etc.

Structured Questions

This section has 5 questions and is worth 54 marks.

Write in **pseudo-code**.

Any algorithm/data structure/data structure operation not taught in CS2040 must be described, there must be no black boxes.

Partial marks will be awarded for correct answers not meeting the time complexity required.

Note:

1. If you are using a Hashtable/Hashset you may assume that the insert, search and delete operations take worst case O(1) time.

2. If you want to solve the question as a graph problem, and the graph modeling is not given, give the graph modeling first (if the required graph has not been described in the question). That is, state what are the vertices, what are the edges and how are the vertices linked by the edges.

15. **UFDS Manipulation**

You are given a UFDS (with access to the p and rank array) which consist of 1 disjoint set of N items (items are numbered from 0 to N-1). Give a O(NlogN) or better algorithm to find and return the size of all subtrees in the UFDS (i.e the size of the tree rooted at each item in the disjoint set) in an array. State the time complexity of your algorithm.

Ans:

1. Create an array A of size N initialized to 0.

2. Call findSet(v) for all v from 0 to N-1. In findSet, do not perform path compression since this will destroy the tree structure and for each item i encountered along the path from v to the root, increment A[i] by 1 (since v must be in the subtree rooted at i).

3. return A.

Since path compression is not used, maximum height will be O(logN) thus findSet(v) will take O(logN) per call and making N calls will take O(NlogN) time.

an alternative solution

1. Create an array A of size N initialized to 1

2. Create an array A' of size N storing pairs. Now go through rank array and for each item i, add (i,rank[i]) to back of A'.

3. Sort A' by rank field. Thus we get the items sorted by their rank.

3. Go through A', and for (i,j) , A[p[i]] += A[i]. // get the size of subtree rooted at item i and add it to the
// size of the subtree rooted at its parent.

4. return A.

This solution works, even though rank is not the actual height. The observation is that the rank of all items in the subtree rooted at some item i must be smaller than the rank of i (during union, the shorter tree will be placed under the taller tree based on rank, or if the rank is the same, one is put under the other and the rank of the new representative item is incremented by 1).

Thus if we go through all the items from smallest rank to largest rank, its equivalent to starting from the leaves (which have size = 1) which will propagate their sizes to their parents, and so on. Thus when we reach any item we are sure to have found the size of the subtree rooted at the item and its size will then be propagated to its parents. This is equivalent to thinking of the disjoint set as a graph with directed edges going from the each item vertex to its parent vertex. This essentially forms a DAG and sorting by rank is a valid topological ordering of the DAG then we just need to find the sizes of the subtrees of the items from left to right of such a topological ordering.

Grading Scheme

Correct solutions
O(NlogN) or below -> 10 marks
O(N^2) -> 6 marks

wrong solutions
1. If findSet or equivalent is used but otherwise totally wrong -> 2 marks
2. if for each item i only find p[i] and increment the size of subtree at p[i] instead of trace all the back to representative item and increment size of each item along the way. This only counts the number of direct children in the subtree rooted at each item -> 4 marks

other wrong/vague/incomplete solutions
0 to 1 mark depending on how wrong/vague/incomplete.

mistakes made
1. forget to disable path-compression in findSet if using the 1st solution -> -2 mark
2. other mistakes -> -1 mark each

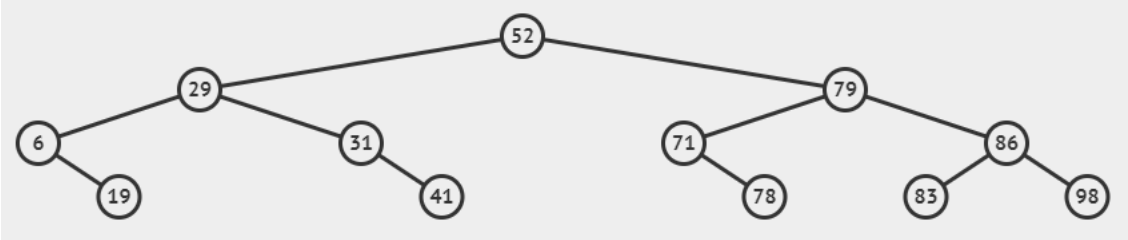
common mistake

1. students only through the p array and for each p[i] increment A[p[i]] by 1. This only find the number of children that an item has in the subtree rooted at the item and not the size of the entire subtree.

2. Forget to disable path-compression when using findSet.

AVL manipulations

a) Given an AVL of N distinct keys and k which is some key in the AVL, give a worst case O(logN) algorithm to find the key with a rank of r in the tree rooted at k. You may assume r is always a valid rank. An example is given below:



In the AVL given, if k = 79 and r = 4, then 83 should be returned.

b) Given an AVL of distinct keys and also the number of keys N in the AVL (N is an odd number), if the median key in the AVL is not the root, give a worst case O(NlogN) or better algorithm that makes use of only the AVL operations — **search(key)**, **insert(key)**, **delete(key)**, **min**, **max**, **rank(key)** and **select(value)** to make the median key the root of the AVL. Only O(1) extra space is allowed.

Ans:
a)
1. Let root' = search(k) // root' is a reference to the node containing k
2. return select(r,root') // i.e start from root' instead of the root of the AVL

time taken is O(logN) since both search and select runs in time O(logN)

b)
1. Let r be the key at the root. rrank = rank(r).
2. mrank = ceil(N/2)
3. if (rrank < mrank) // key with median rank is in right subtree and right subtree is heavier
   for i = 1 to | r' - m |
    delete(r) // note that since N > 3 the root of the AVL must have both left and right child thus
              // delete will replace the root with the successor
    insert(r)
    r = new key at root
4. else // key with median rank is in left subtree and left subtree is heavier
    do the same as step 3 but instead of replacing the root with successor for the delete operations,
    replace with predecessor.

// note that if the median is not the root, it must be in the larger of the subtree rooted at the left
// or right child of the root, thus a delete operation will not result in a cascade of re-balancing
// rotations that will rotate away the new root.

Will take time = O(|r'-m'|*log N)

Grading Scheme

a)
Correct Solutions
O(logN) solution -> 5 marks
O((logN)^2) solution -> 4 marks
O(N) solution -> 3 marks
O(NlogN) solution -> 2 marks

mistakes made
-3 marks if assume that the node that contains the key k is given and start select from that node as root node.

-2 marks, if starting from the root of the AVL, but the rank of the key to be found in the entire AVL is not calculated correctly.

-2 marks for each error made in updating the rank of the key to be found if solution implements the rank function.

-1 mark for other mistakes

cap at -4 marks for mistakes.

vague/wrong/incomplete solutions
0 to 1 mark depending on how vague/wrong/incomplete

==========

b)
I wanted to make it easier for you by asking you to think in terms of only the given AVL operations in the question. However, I forgot that the delete operation had to be modified to make the solution work (solutions involving the operations without modification is hard to proof correctness or time complexity) .... *facepalm*. Thus to be fair, I have voided this part and have given everyone 10 marks.

16. **Road Network**

The local government is rebuilding the roads of a certain district in order to improve traffic congestion. There are N locations in the district numbered from 0 to N-1 and M pairs of the locations are proposed to be linked by 2-way roads. There is always a way to get from 1 location to another via the proposed roads.

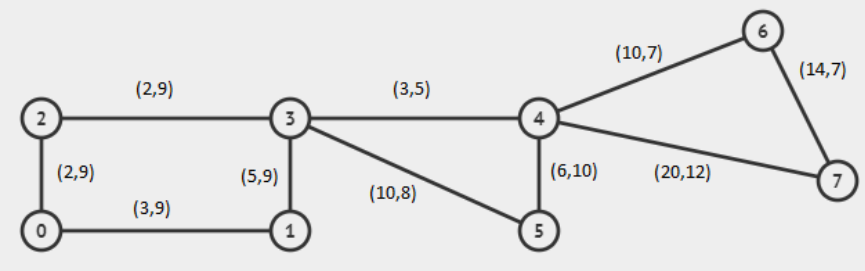
Each proposed road is given a cost (integer value > 0) to build and also a flow index (integer value >= 0) which describes how well traffic will flow through that road if it is built. **The lower the flow index the higher the congestion** and vice versa of the proposed road.

You are in-charge of the project, and you will have to build the roads that satisfy the following 2 conditions, where condition 1 is more important than condition 2:

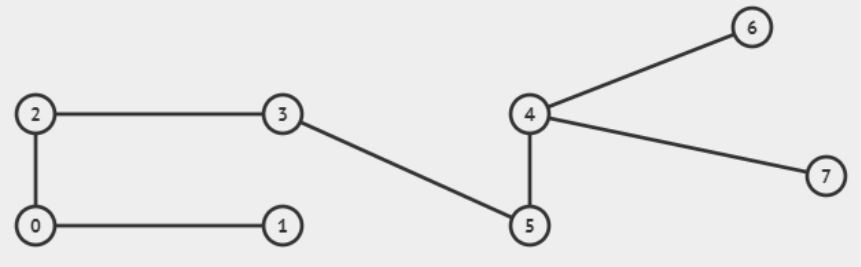
1. Roads are to be built such that the congestion of any route used to get between any 2 locations is minimized. The congestion of a route is defined as the highest congestion among all roads along the route.

2. Use the minimum cost to build the roads so that there is a way to get from any location to any other location such that condition 1 is not violated.

An example of proposed roads linking 8 locations (numbered from 0 to 7) is given below. Note that the weight on each edge is a pair (c,f) where c is the cost and f is the flow index



For the given example, the following are a set of roads (without showing the edge weights) that should be built which will satisfy condition 1 and 2 in that order respectively.



You are given an array A of the M roads expressed as tuples of the form (x,y,c,f) where x and y are locations to be linked by the road, c is the cost of the building the road and f is the flow index of the road.

Describe the most efficient algorithm you can think of to solve the problem, i.e come up with the set of roads to be built (any set which satisfies the 2 conditions will do) and state its time complexity.

Ans:

To satisfy the 1st condition this is actually to find the maximin path for all routes based on the flow index (the higher the flow index, the lower the congestion).

To satisfy the 2nd condition this is standard MST based on the cost.

Since 1st condition is more important than the 2nd, we simply sort array A (which acts as an edge list) according to descending order of flow index, and for roads/edges with the same flow index, sort them in ascending order of cost.

After that simply perform standard Kruskal's algorithm.

Grading Scheme

Correct answers:

- O(M log M) algorithm: Use Prim's algorithm to find the maximum spanning tree, where the edge comparing function is defined to sort flows in descending orders and break ties according to cost. (10 marks)
- O(M log M) algorithm: Alternatively, we sort all edges with this comparison function first, then perform Kruskal's algorithm (10 marks)

Incorrect answers:

- O(M^3) algorithm: Use Floyd-Warshall algorithm to find the maximin spanning tree, and correctly define what each D[i][j] stores (should be storing predecessor, flow index and cost) as well as the updating function, taking the tiebreaker into account. Another problem with this is that the combined paths might not form a tree. (5 marks)
- O((M+N) log M) algorithm: Using Dijkstra's or Bellman-Ford algorithm to find the "longest" path spanning tree starting from some random node. The algorithm is likely not working since we cannot find the longest path with Dijsktra's and even if we reverse the edge weight, we might have negative cycles. Resulting shortest path spanning tree is likely not the same as the MST. (2 marks)

Common mistakes and deductions:

General scheme for all solutions (the marks listed here is the max possible, for specific solution look at their grading scheme):

- Didn't take cost into account. (max 5 marks)
- Cannot even output a valid tree. (max 2 marks)
- Give at least 1 mark to answers that at least make a bit of sense.

For algorithms using Prim's or Kruskal's to find MST (max 10 marks):

- Small mistake regarding the sorting order of flow index. (-0 marks)
- Sort the cost first and break ties with flow index (wrong priority) (-1 mark)
- Unclear or incorrect (-1~3 marks) or no specification of how to sort the edges, or consider only the flow. (-5 marks)
- Try to find all the valid maximum spanning trees for flows, then find the one with minimum cost. This will give correct result but is slow if the number of MSTs is large. (-3~4 marks)

- Try to find separate MSTs, one for maximum flow and the other for minimum cost, then try to find consensus between the two. This is likely not going to work and resulting in a sub-optimal answer in terms of cost. (-5~7 marks)

For algorithms using Floyd-Warshall algorithm to find MAXimin paths (max 5 marks):

- Not specifying the content of the matrix and the updating function, not considering the cost, resulting in sub-optimal results. (-1~3 marks)

For incorrect algorithms finding shortest path spanning tree (max 2 marks):

- Not adjusting the relax function and the priority queues correctly. (-1~2 marks)

17. **Graphs, graphs, graphs**

You are given a connected graph G of V vertices (numbered from 0 to V-1) and E edges where the vertices are given weights of W1,W2 or W3 (where W1,W2 and W3 are distinct positive integer values). Each pair of vertices x and y in G is linked by an undirected edge as follows:

i.) if weight(x) = weight(y) then x and y is linked by an undirected edge of weight = weight(x) (e.g if weight(x) = weight(y) = W1 then the edge that links them is weight W1)

ii.) if weight(x) != weight(y) then x and y is linked by an undirected edge of weight = max(W1,W2,W3)+1.

The graph thus obtained is a complete graph. You are guaranteed that there will be > 1 vertex with weight W1, W2 and W3 respectively in the graph.

Given such a graph in the form of an unsorted edgelist and also an array A of size V where A[i] = weight(i) for each vertex i, give the most efficient algorithm to find the cost of the MST of the graph (here the cost of a ST is still only summing up the weights of edges in the ST only and not the weights of the vertices). State the time complexity of your algorithm.

Ans:

1. O(V) time algo. go through the array A and count number of unique vertices of each weight. Let this count be V1,V2 and V3 respectively. The cost of the MST of the graph is then ((V1-1)*W1)+((V2-1)*W2)+((V3-1)*W3)+2*(max(W1,W2,W3)+1).

Rationale for the above solution. Look at it from the point of view of how Kruskal's will "solve" MST for this kind of graph. First it will sort all the edges by weight. Without loss of generality, assume that W1 < W2 < W3. So in the sorted edgelist, W1 edges will be processed first and a spanning tree involving all vertices of weight W1 will be formed. Then W2 edges will be processed and a spanning tree involving all vertices of weight W2 will be formed. Same thing for the weight W3 vertices. At this point we have 3 trees and the weight of their edges are exactly (number of W1 vertices – 1)* W1+(number of W2 vertices – 1)* W2+(number of W3 vertices – 1)* W3. Finally to form the MST these 3 trees only need 2 edges to link them. Since the remaining edges are those which link vertices of different weight and have weight MAX(W1,W2,W3)+1, the total weight of the MST is simply the weights of the 3 trees + 2*(MAX(W1,W2,W3)+1).

2. O(V+E) time algo that does BFS on each "component" made of vertices of W1,W2 and W3 respectively.

3. O(ElogV) time algo for Prim's or Kruskal's.

Grading Scheme

Correct answers:

- O(V) algorithm: Count the number of vertices of each weight, then calculate the cost of MST directly (10 marks)
- O(V) algorithm: Sort vertices of each weight in separate arrays, construct the MST by linking adjacent vertices in the array together, plus two edges connecting the 3 connected components. (10 marks)
- O(V^2) algorithm: Use variant of Prim's algorithm for dense graph to calculate the MST. (7 marks)
- O(V^2 alpha(V^2)) algorithm: Sort edges of each weight in separate 4 arrays, perform Kruskal's on each of the array (in the order from smallest weight to largest weight). (7 marks)
- O(V^2 log V) algorithm: Use original Prim's or Kruskal's algorithm. (5 marks)

Incorrect algorithms:

- Use Bellman-Ford, Dijkstra's or Floyd-Warshall algorithm to find shortest path spanning tree or try to find MST. (0 mark)

Common mistakes:

- Not taking the edges of weight max(W1, W2, W3)+1 into account (-1 mark)
- Wrong number of edges when calculating the total cost (usually forgetting that we need V-1 edges instead of V edges in the tree) (-1 mark)
- Unclear, redundant or illegal operation on graphs. (-1~3 marks)

19. **Alternative route**

District X is the central business district of city Y and people from other districts in Y will go to X for work in the morning and go back home in the evening. These people are so familiar with how to get from their home to X that they will always use the fastest route to get there. **In fact all fastest route to get from any district to X is used by someone**.

One day in the middle of the night, there was an earthquake and many of the roads in city Y have been damaged.

In the morning, you hear from the news that the roads used by people to get from their district to X are the most badly affected since those roads are already under stress from been over usage. Since you still have to get to work in district X, you are worried that some roads along your usual route may not be passable. It is also unlikely to find any route from your district to X that is totally composed of roads not used by anyone thus the best alternative route you would like to use is

the fastest route made up of only 1 road which is not used by anyone to get from their district to X.

The road network of city Y is represented as a graph of V vertices and E edges where each district is a vertex and 2-way roads linking 2 districts are undirected edges linking the respective vertices. The weight of each edge is a positive integer value representing the time to travel the edge/road. It is guaranteed that you can always get from any district to any other district in city Y via the road network.

Come up with the most efficient algorithm to find and return the cost of the best alternative route as given above. If no such route exist return -1. State the time complexity of your algorithm.

You may assume the graph is stored in an adjacency list.

Ans:

Here the requirement is that only 1 road need be a road not used by any fastest route from any district to X.
To find the fastest route from any district to X is equivalent to finding the fastest route from X to any of the other districts. However we don't just 1 fastest route we want to find the edges belonging to all fastest route from X to any of the other districts, since all fastest routes are used by someone as stated in the problem description. To do this change the predecessor array so that each index stored a linked list of predecessors rather than just 1 predecessor (so that we track the edges that belongs to all possible fastest route from X to another of the other vertices)

1. Run modified Dijkstra from X to get the SPST. → O(ElogV)
   i. modify the relaxation condition so that when D[v] == D[u] + w(u,v) we will add u into the head of the linked list at p[v] and when D[v] < D[u] + w(u,v) we will clear clear the linked list by pointing head to null then add u into the now empty linked list. → O(1) time

2. Get all the edges from the adjacency list and put into hash table → O(V+E) time

3. For each edge in the predecessor array obtained from step 1 check if they are in the hashtable. If they are not add them to a new edges list EL'. → O(V+E)

4. run modified Dijkstra 2 times, once using your home location as source (let D_h be the distance array obtained), once using x as the source (let D_x be the distance array obtained) → O(ElogV)

5. Let bestcost = infinity

6. for each edge (y,z) in EL' → O(E)
    If (D_h[y]+weight(y,z)+D_x[z] < bestcost)
        bestcost = D_h[y]+weight(y,z)+D_x[z]
    if (D_h[z]+weight(y,z)+D_x[y] < bestcost) // need to consider the reverse (z,y) since undirected
        bestcost = D_h[z]+weight(y,z)+D_x[y]

7. If bestcost == infinity return -1 else return bestcost

Total time = O(ElogV)

Grading Scheme

Grading of solution that are reasonably close to correct solution

1. Find the SP from X to the other districts and modify predecessor array accordingly to contain the edges along all SP to a vertex -> 3 marks
- -1 mark if did not modify predecessor array to get all edges along all SPs from X to the other districts.
- -1 mark if using each other district as source vertex to find SP to X (slower -> this will be O(VElogV)/O(V^3)

2. Correctly identify and store all roads not used in any SP from X to any of the other districts (call this set of roads R)-> 2 marks
- -1 mark for each mistake here

3. Correctly form a graph from the predecessor array in 2. and run Dijkstra 2 times once from X and once from K to get the 2 distance arrays -> 1 mark
4. Go through all the roads in R and correctly form candidate alternate SP as required and get the minimum cost correctly -> 3 marks
- -2 marks for major mistake here
- -1 mark for minor mistake here

wrong/vague/incomplete solution

1. A complete but wrong solution -> 0 to 1 mark depending on how wrong
2. vague solution that states using Dijkstra/bellman-ford to find SP but no details on how to use it to solve the problem -> 1 mark
3. If use Dijkstra to find SP from X to other districts but no more elaboration after that (this is part of the solution) -> 2 mark
4. Other incomplete and wrong solution or incomplete and unable to find a working part in the solution -> 0 mark

Common mistake

1. Forget to modify the predecessor array to store all edges to each vertex v that lie along some SP to the v, since we need to identify all edges/roads that will lie along all possible SPs from each district to X, and also get the complement of the such edges, i.e the edges/roads that do not lie along any SP from any district to X.

2. Forget to form a graph from the edges in the modified predecessor array and run SSSP on that graph(since only 1 edge is a road not used in any SPs to X and the rest must therefore be roads that lie along some SP to X), rather than on the original graph.

Misunderstood version

There are some students who misunderstood the question to mean that the alternative route required is the fastest route made up of only 1 road that is not used by any fastest route from any district to X. **At most 5 marks is given** for correct solutions to such an interpretation of the question.

Reason is because the context given is that it is hard to find a fastest route made up totally of roads not used in any fastest route from any district to X, that's why the alternate fastest route is to be computed. If this alternate fastest route is made up of only 1 road and that road is one not used any fastest route then it will exactly fit the hard to find kind of route and so would not make sense, and so students should have clarified the question during the exam.

Grading of solution that are reasonably close to correct solution

1. Find the SP from X to the other districts and modify predecessor array accordingly to contain the edges along all SP to a vertex -> 3 marks
    - -1 mark if did not modify predecessor array to get all edges along all SPs from X to the other districts.
    - -1 mark if using each other district as source vertex to find SP to X/or use Floyd Warshall (slower -> this will be O(VElogV)

2. Correctly identify and store all roads not used in any SP from X to any of the other districts (call this set of roads R) and find if there is a a road in R from X to K or K to X if there is return the weight of such an edge, otherwise return -1-> 2 marks
    - -1 mark for each mistake here.

wrong/vague/incomplete solution

same as the grading scheme for the correctly understood version