NATIONAL UNIVERSITY OF SINGAPORE

CS2040 – DATA STRUCTURES AND ALGORITHMS

(Semester 1: AY2019/20)

Time Allowed: 2 Hours

INSTRUCTIONS TO STUDENTS

1. Do **NOT** open the question paper until you are told to do so.
2. This assessment paper consists of **Twelve (12)** printed pages and **Nine (9)** questions with possible subsections.

   Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
3. This is an **Open Book Assessment**. You can check the lecture notes, tutorial files, problem set files, CP3 book, or any other books that you think will be useful.
4. When this Assessment starts, **please immediately write your Student Number** below. Do not write your name.
5. You may write your answers in pencil except student number below which should be written with a pen.
6. All the best!

STUDENT NUMBER:

A								

(Write your Student Number above legibly with a pen.)

Questions	Possible	Marks
Q1-5	**15**	
Q6	**25**	
Q7	**20**	
Q8	**15**	
Q9	**25**	
Total	**100**	

Section A – Analysis (15 Marks)

Prove (the statement is correct) or disprove (the statement is wrong) the following statements below. If you want to prove it, provide the proof or at least a convincing argument. If you want to disprove it, provide at least one counter example. 3 marks per each statement below (1 mark for circling true or false, 2 marks for explanation):

1. After doing FindSet(x) operation in Union-Find Disjoint Sets data structure with the path compression heuristic activated, the real height (not the rank) of the disjoint set that contains x always decreases, when x is not the root of the disjoint set. **[True/False]**

False. If an element is already a direct child of the root(height = 1) doing findSet on it will not decrease its height

2. Given any AVL tree of height 4, deleting any vertex in the tree will not result in more than 1 rebalancing operation. **[True/False]**

true. when any vertex is deleted, the avl is already balanced at each level
hence invariant may be violated at most once which will not result in more than 1 rebalancing operation

False. Already have an example in the lecture notes which is an AVL of height 4 where deleting a vertex results in 2 rebalancing operations.

3. The time cost for changing a binary MAX heap to a binary MIN heap is O(n). **[True/False]**

false. extract max takes O(log n) so add each element in to PQ sorted from min to max -> nlogN
then run create heap on the PQ -> O(N) -> total time complexity: O(n log n)

True. Just treat the array containing the max heap as input and call O(N) fast heap create on it to get the min heap

4. To form the MST of an undirected connected graph, exactly 1 edge is removed from every cycle in the graph, namely the largest edge in the cycle. **[True/False]**

false, in some cycles more than 1 edge is removed. In prim's it chooses the locally optimal choice
the smallest edge recursively while avoiding cycles. kruskal's just chooses all the smallest edges without creating a cycle.

5. It is possible for every vertex in a DAG to have an outgoing edge to some other vertex in the DAG. **[True/False]**

false, if every vertex in a dag has an edge to another edge in the dag, the vth edge will have form a cycle
total num of edges = n where n is the num of vertices and this violated dag property where max num of edges is n-1

False. Argument is similar to the proof that there must be a vertex with no incoming edges in a DAG as given in the lecture notes.
If every vertex in a DAG has an outgoing edge that means, we can pick any vertex X and choose an outgoing edge from X and move to the next vertex from that vertex we can do the same since every vertex has at least one outgoing edge. After we have visited more than N vertices like this where N is the number of vertices in the DAG, then 1 vertex Y must have been repeated. The edges used between the 2 successive visits of Y will form a cycle, and this is a contradiction that the graph is a DAG.

For section B, Partial marks will be awarded for correct answers which do not meet the time complexity required (or if the required time complexity of not given, then any correct answer that is less efficient than the lecturer's answer) . You can write in pseudo-code. Provide enough details to all user defined DSes/algorithms/modifications to taught DSes and algorithms so as to show understanding of the solution.

Section B – Application (85 Marks)

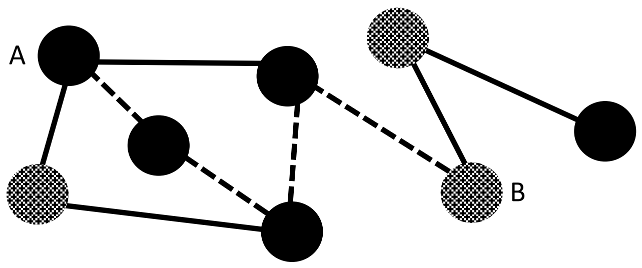
6. Battle Nations [25 marks]

The world of Aquarius is at war! This world is made up of mostly water with N islands which are connected by M bridges where ($N-1 \le M \le \frac{(N*(N-1))}{2}$). You may assume that there is always a way to get from any island to any other island. Each island belongs to either country **X** or country **Y**. These two countries are now at war with each other.

In order to attack an island of the enemy country, the attacking country must choose an island A that it owns to initiate the attack and an island B of the enemy country to attack.

Any **valid path** from A to B the attacker can choose must start from A and end at B such that B is the **first enemy island** encountered along the path (otherwise B cannot be attacked first without attacking the previous enemy island(s) along the path!).

Any bridge along a valid path may potentially be targeted by the enemy country's aerial bombers to be destroyed so as to prevent the attack from happening. Each bridge is given an integer value W ($1 \le W \le 10{,}000$) indicating its difficulty to be destroyed (**larger** W means it is more difficult to be destroyed).



Legend:

● = Island belonging to country X

▨ = Island belonging to country Y

The figure above is a small example of the world Aquarius, where the circles are the islands and the lines are the bridges connecting the islands. Here country X is the attacker selecting island A to attack country Y's island B. The dotted edges show a possible valid path to get from A to B.

**read b) and c) first before answering a)!*

a) If you want to represent the world of Aquarius as a graph $G(V, E)$ to answer part b) and c),

i.) What do the vertices in V represent and what information will you store in a vertex? **[2 marks]**

Vertices represent the islands in Aquarius and the info stored is the country that each island belongs to

vertices are the islands. Use integer values 0 to N-1 to label each island. Have an attribute nation to indicate which nation the vertex belongs to. E.g 1 for country X and 2 for country Y

ii.) What do the edges in E represent and what information will you store in an edge? **[2 marks]**

edges represented the bridges between one island to another and info stored is how weak the bridge is

iii.) Is this a directed or undirected graph? **[1 mark]**

it is undirected graph cause bridges can be crossed both ways

iv.) What is the best graph data structure D to store this graph to answer b) and c)? **[1 mark]**

adjacency list

b) Given the graph data structure D as modelled in a), given island A of an attacker and island B of the enemy to attack, give the best algorithm you can think of to find the best valid path to get from A to B such that the weakest bridge (based on its difficulty value of being destroyed) is **maximized** and output the difficulty value of that weakest bridge.
If there is no valid path from A to B output -1. **[9 marks]**

run Prim's on the adjacency list to find the minimum spanning tree
keep track of the largest weak bridge encountered
if no no valid ou

This is basically a maximin problem with a slight modification.
1.) Have a variable weakest that keep tracks of the weight of the weakest bridge in the best valid path from A to B. Initialize weakest to -1.

2.) Run Kruskal's algorithm on D but instead of sorting in ascending order, sort by descending order (to get a maximum spanning tree)

3.) Next, modify Kruskals's algorithm as follows:

```
For each edge (u,v,w) in sorted edge list

    //two attacker islands are connected
    If (u.nation == A.nation && v.nation == A.nation) // edge connects attacker islands
       If !UFDS.isSameSet(u,v)
          UFDS.unionSet(u,v)
          weakest = w

       else if ((u.nation == A.nation && v == B) ||
       (v.nation == A.nation && u == B)) // edge connects attacker island to B
            If !UFDS.isSameSet(u,v)
              UFDS.unionSet(u,v)
              weakest = w

    If UFDS.isSameSet(A,B) // found best valid path from A to B
      Break out of for loop
```

4.) output weakest if UFDS,isSameSet(A,B) else output -1
Same time complexity as standard Kruskal's : O(MlogN)

c) Assuming C is the set of islands owned by the attacker and given a fixed enemy island B. For each of the islands A' in C with a valid path to B, the best path (path that maximizes the minimum edge) has been found and edges in the best path from A' to B has been put into an edge list EL.
However, the enemy has just destroyed a weakest bridge (u', v') along the best path from an attacker island A to B, so the edge representing that bridge has been removed from EL.
Given D and EL, now provide an algorithm to find the next best valid path from A to B in $O(N + M)$ time. Specify any modifications you need to make to the graph $G(V, E)$ and D, and any extra data structures you use.
Again, if there is no more valid path output -1, otherwise output the weight of the weakest bridge in that next best valid path. **[10 marks]**

7. **Sorting or Not? [20 marks]**

a) Given an **unsorted** array A of N unique **floating point** values of no fixed precision, where each value may only differ from its correct position in the **sorted** array by no more than K positions (K is much smaller than N), give an algorithm that will fully sort A in $O(NlogK)$ time. For example, if K = 3 and after sorting A, the value 10.3 is at index 10, then in the original unsorted array 10.3 can be possibly found at indices 7,8,9,10,11,12,13. **[10 marks]**

run merge sort on the array

b) Given a K by N matrix L, where each row represents a sorted array of N unique **floating point** values of no fixed precision (thus K sorted arrays), give an algorithm to merge them into 1 sorted array of $K * N$ elements in $O(KNlogK)$ time. **[10 marks]**

apply merge sort on first two rows, new total rows = k -1
then apply on first two rows again, new total rows = k - 2
repeat until all rows are merged -> only one row left
this does merge sort k - 1 times hence time complexity is O(KNlogK)

8. **Number Candies [15 marks]** *(May be difficult)*

The current rage among children are number candies. As the name suggest, these candies are given a number from 1 to N where N is at least 10 million and can be up to the **billions**! The ultimate goal is to collect all N of these candies. That is of course quite impossible since the number candies carried by any candy store is random (and with lots of repeats), and no store can carry anywhere near even 10 million candies.

However, the candy manufacturer has made the task easier. If anyone can collect all candies numbered sequentially from 1 to M where $M < N$ and up to 100,000, the manufacturer will give him/her all the N numbered candies.

One particularly rich kid has hired L helpers, where $M < L < N$ and up to 1,000,000. Each helper will buy one candy from each of the L candy store around the country. He hopes that by doing so he will be able to get all the candies numbered from 1 to M.

As his helpers come back one by one with the candy they have bought, the rich kid has a hard time trying to figure out if he has all the candies numbered from 1 to M.

Using what you have learned in CS2040, give an algorithm and required data structure(s) to help the rich kid answer the following query in better than $O(Min(logN, M)$ time:

What is the largest numbered candy he has that is in an unbroken sequence from candy #1? If the largest numbered candy found in such a way is > M just return M. If he does not have candy #1 then return -1.

You have to answer this query each time a candy is being brought back by a helper**.** So there will be L such queries (each to be answered in better than $O(Min(logN, M)$ time). *Note that the candies need not be brought back in ascending order of candy number*.

E.g if a helper brings back candy #15 and the candies the rich kid has is currently 1,2,3,4,5,6,7,20,21,23,25,26,27,30, then after the inclusion of candy #15, the largest numbered candy he has that is in an unbroken sequence from candy #1 is candy #7.

You can perform a preprocessing step to help you answer the query. The preprocessing step must run in time better than O($NlogN$) time.

Please analyze the time complexity of your solution.

9. Time Traveler [25 marks]

Mark is an officer from the Bureau of Time Traveling Law Enforcement (BOTTLE for short). He has a device which allows him to go forward in time in order to apprehend time traveling criminals. Starting from the present day which we will call day 0, Mark can use his device to jump forward in time by a certain number of days.

Each time he uses the device he can choose a value from a set L ($|L| > 0$) containing the number of days to jump. For example, given L = {2, 5, 7, 10, 30}, Mark can choose 7 which means he will jump forward by 7 days into the future. He can make multiple jumps by doing so, however there is a maximum range of the number of days he can go forward to, which is given by the value $T - 1$. Once he goes past $T - 1$, he will simply wrap around to day 0 and continue from there. For example, given L as above, if T = 20, and he chooses 30, after he goes past day 20 in the future he will simply return to day 0 and continue up to day 10 in the future which is where he will find himself.

Now choosing each value from L to make a jump requires a certain amount of chrono-energy and each value in L has an associated amount of chrono-energy which is represented in another set M. For example, again given L = {2, 5, 7, 10, 30}, and M = {4,7,10,20,40}, if Mark chooses 5 then he will need to use 7 chrono-energy to jump forward 5 days. If he chooses 10 then he will need to use 20 chrono-energy to jump forward 10 days. However, the time machine can only store X units of chrono-energy thus Mark can only make a limited number of jumps before he has to stop.

Given the set L, the set M, the value T, the value F which is number of day in the future he wants to arrive at from current day (day 0) and X the amount of chrono-energy the time machine starts with, model the problem as a graph problem and answer the query **JumpPossible(L, M,T,F,X)** which will return true if he can make a series of jump to arrive at day F without running out of chrono-energy (he can hit 0 when he arrive at day F), or false otherwise.

For example, if L = {2, 5, 7, 10, 30}, M = {4,7,10,20,40}, T = 20:

Now if F = 10, X = 8, then there is no way for him to arrive 10 days in the future, since making 1 jump of 10 requires 20 chrono-energy, making 5 jumps of 2 require 5*4 = 20 chrono-energy, making 1 jump of 30 which will wrap around and land him at day 10 will require 40 chrono-energy. Any other combinations will also require too much chrono-energy.

Now if F = 17, X = 31 then one possible way for him to jump forward 17 days is by making the series of jumps: 7,10 which cost only 30 chrono-energy.

a) Model the graph $G(V, E)$ to answer **JumpPossible(L,M,T,F,X)**

i.) What do the vertices V represent and what information will you store in a vertex? **[2 marks]**

vertices represent no of days he can jump forward into the future info stored is num of days

There is a vertex for each number from 0 to T-1

ii.) What do the edges E represent and what information will you store in an edge? **[2 marks]**

edges represent the jump to each vertices and info stored is the chrono energy that will be consumed to make the jump

Edges are directed and for any vertex pair x,y there is an edge from x to y if there is an i such that y = (x+L[i])%T. The weight of the edge is M[i]. Ignore edges that point back to the vertex itself or multiple edges pointing to same vertex y (for these simply use the smallest weighted edge).

~~iii.) Is this a directed or undirected graph? **[1 mark]**~~

iv.) What is the best graph DS to store this graph to answer **JumpPossible(L,M,T,F,X)**? **[1 mark]**

adjacency list

b) Now using the graph in a) give an algorithm for **JumpPossible(L,M,T,F,X)** that runs in time better than $O(|L| * T^2)$. **[9 marks]**

Simply run original or modified dijkstra on the graph modelled in a) using vertex 0 as the source vertex. If D[F] != -1 && D[F] <= X then return true else return false
# of vertices = T
# of edges = O(|L|*T)
Thus time for djikstra algo = O((T+|L|*T)logT) = O(|L|*TlogT) which is better than O(|L|*T^2)

c) Mark has modified his time machine so that in his series of time travel jumps to reach a certain day in the future, he can at **any point** and **only once** pick one day from a set L' of days (L' ∩ L = ∅ and $|L'| \leq |L|$), that he can jump forward to without using any chrono-energy. Given this new modification, detail any changes to the graph in a) and give an algorithm for **JumpPossible(L,L',M,T,F,X)** that will still run in time better than $O(|L| * T^2)$. **[10 marks]**

For example, if L = {2, 5, 7, 10, 30}, L' = {1, 4, 8}, M = {4,7,10,20,40}, T = 20:

If F = 12, X = 8 then one possible way to jump 12 days into the future is to first make a jump of 2 for 4 chrono-energy followed by a free jump of 8, then followed by another jump of 2 for 4 chrono-energy to reach 12 days in the future. In all, this use exactly 8 chrono-energy. However, Mark cannot make a free jump of 4 followed by another free jump of 8 to reach 12 days, since he can only do a free jump once in his series of jumps.

**~~~ END OF PAPER ~~~**