National University of Singapore

School of Computing

CS2040 - Data Structures and Algorithms Final Assessment

(Semester 4 AY2017/18)

Time Allowed: 2 hours

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper contains THREE (3) sections.
   It comprises TWELVE (12) printed pages, including this page.
3. This is an **Open Book Assessment**.
4. Answer **ALL** questions within the **boxed space** in this booklet.
   Only if you need more space, then you can use the empty page 12.
   You can use either pen or pencil. Just make sure that you write **legibly**!
5. Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
   Read all the questions first! Some questions might be easier than they appear.
6. You can use **pseudo-code** in your answer but beware of penalty marks for **ambiguous answer**.
   You can use **standard, non-modified** algorithm discussed in class by just mentioning its name.
7. Please write your Student Number below. Do **NOT** write your name.

A	0	1						

This portion is for examiner's use only

Section	Maximum Marks	Your Marks	Remarks
A	20		
B	30		
C	50		
Total	100		

A Basics (20 marks)

A.1 Worst Case Time Complexity Analysis (10 marks)

Write down the *tightest*[1] *worst case* time complexity of the various data structure operations or algorithms below. Each correct answer is worth 1 mark.

The operations (algorithms) referred below are the **unmodified version**, as per discussion in class, e.g. as currently explained in VisuAlgo or as currently implemented in Java API (version 8). Unless otherwise mentioned, there are currently n elements in the data structure. For graph-related operations, let n be the number of vertices and m the number of edges. You can assume that $10\,000 \le n \le 100\,000$. AM/AL/DAG/SSSP are the abbreviations for Adjacency Matrix/Adjacency List/Directed Acyclic Graph/Single-Source Shortest Paths, respectively. Unless specifically mentioned, all graph-related operations are performed on simple graphs stored in an AL data structure.

`PQ` is a Java PriorityQueue currently with n elements, `HM` is a Java `HashMap<Integer, Integer>` currently with n elements, `TM` is a Java `TreeMap<Integer, Integer>` currently with n elements. `v` is a 32-bit Integer.

No	Operations	Time Complexities
1	`while (n-- > 0) System.out.println(PQ.poll());`	$O($_______$)$
2	`for (int i = 0; i < n; i += 3) HM.add(i);`	$O($_______$)$
3	`HM.remove(v); // but v does not exist`	$O($_______$)$
4	`System.out.println(TM.lowerKey(TM.lastKey()));`	$O($_______$)$
5	`System.out.println(TM.containsValue(v));`	$O($_______$)$
6	Transpose a DAG stored in an AM	$O($_______$)$
7	Store complement of an unweighted graph in AL_1 into AL_2	$O($_______$)$
8	Running `DFS(u)` when out-degree of vertex $u = 0$	$O($_______$)$
9	Running `BFS(u)` where u is the root of a tree	$O($_______$)$
10	Compute SSSP from source s in a weighted Complete Graph	$O($_______$)$

[1] What we meant by tightest worst case time complexity is as follows: If an operation of the stipulated data structure/an algorithm needs at best $O(n^3)$ if given the worst possible input but you answer higher time complexities than that, e.g. $O(n^4)$ – which technically also upperbounds $O(n^3)$, your answer will be graded as wrong.

Section A.1 Marks = _____

A.2 Fill in the Blanks (10 marks)

Each correct answer is worth 1 mark.

1. We can use *Radix Sort* to sort n Strings that have at most d lowercase alphabet characters in $O($__________$)$ time.
2. n $(n > 1)$ Characters of a String Z are pushed/enqueued one by one into a Stack `S`/Queue `Q`, respectively. After all n Characters are pushed/enqueued into `S`/`Q`, they are popped/dequeued from `S`/`Q`, respectively. Amazingly, the sequence of popped/dequeued Characters are identical. Thus Z is a ___________________ String.
3. The time complexity of partial Heap Sort algorithm if we are only interested to find k smallest Integers in a Binary Min Heap containing n Integers is $O($___________________$)$.
4. When the range of the Integer keys is small, e.g. $[0..M\text{-}1]$, we can use Direct Addressing Table (DAT). A few examples of small range of Integer (or easily convertible into Integer) keys are: 'Singapore bus numbers [2..991] (at the moment)', ASCII values [0..255], etc... Mention **one more** (Integer) keys where DAT is applicable: ____.
5. Java `Hashtable` *always* rehash its contents into a larger Hashtable with twice table size whenever n (number of keys) exceeds α (load factor) $\times$ m (table size). Currently, α is set to default 0.75. If we know that we will not have more than 100 000 keys to be inserted into our Hash Table, we shall set the smallest initial m to be: ____________________________ to avoid rehash.
6. Java `TreeSet` cannot have duplicates as it implements Interface `Set` that specifies this no-duplicate requirement. Therefore, if need to allow duplicates in our application, we shall use ______________________________________ instead.
7. The best data structure to store a complete binary tree is ___________________.
8. The best data structure to store a *complete unweighted* graph of V vertices ___________________.
9. The maximum number of Connected Components (CCs) in a general graph G with V vertices and E edges (for any number of edge(s)) is __________.
10. The shortest path from the source vertex s to another vertex v in a general weighted graph G is ___________________ if vertex v is actually unreachable from s.

Section A Marks = _____ + _____ = _____

B Intermediate (30 marks)

B.1 Analysis (12 marks)

Prove (show that the statement is correct) or disprove (give a counter example) the statements below.

1. We can prove that during AVL Tree insertion, if there is a need to do tree rotation, then rotating just the lowest vertex with balance factor +2 (or -2) is sufficient.

2. To insert n (e.g. $10\,000 \leq n \leq 100\,000$) *sorted* Integers into an AVL Tree, we need to call $O(\log n)$ AVL-Tree insertion n times, hence $O(n \log n)$ overall. We cannot do better than this, unlike the case with the faster $O(n)$ version create heap vs the standard $O(n \log n)$ create heap.

3. Running `DFS(u)` and `BFS(v)` where $u \neq v$ in a graph with V (V much larger than 2) vertices and E edges will always yield *different set* of visited vertices.

4. If there is only one possible path from vertex u to vertex v in a general weighted graph, then the shortest path from source vertex s to v must be that path $s \rightarrow ... \rightarrow u \rightarrow ... \rightarrow v$.

Section B.1 Marks = _____

B.2 Implications (6 marks)

Imagine that we have n ($10\,000 \leq n \leq 100\,000$) Strings of length not more than $k = 20$ characters. We want to insert these n Strings into a Hash Table of size $m = 255$ that uses Separate Chaining as collision resolution technique. The hash function used is $h(v)$ = ASCII value of the first character of the string. You can assume that the n Strings are collection of n city names in the world.

In class, we learn that Separate Chaining usually uses auxiliary data structure: (Doubly) Linked List to store keys that collide into the same hash value. In this question, we replace Linked List with Balanced BST as the auxiliary data structure. What are the implications (which can be either negative, *neutral*, or even *positive*) of such modification? Each correct answer is worth 1 mark.

1.

2.

3.

4.

5.

6.

Section B.2 Marks = _____

B.3 Create Test Cases (12 marks)

Create Test Cases for each scenario below. Each valid test case worth 3 marks.

1. Suppose that we have an initially empty Hash Table with table size $m = 5$. The keys are Integer keys. The hash function is $h(v) = v\%m$. If there is a collision, we use Quadratic Probing with probing sequence $(h(v) + k \times k)\%m$ as collision resolution technique for each probing step k. Insert 3, 4, or up to 5 Integers so that version of Quadratic Probing fails to find an empty slot albeit there is at least one empty cell.

2. Insert $V = 12$ distinct integers **one by one** into an initially empty AVL Tree so that the resulting AVL Tree has height (number of edges from root to the deepest leaf) = 4. To simplify grading, the Integers must be positive and not more than 13.

3. Draw the *highest possible* rooted (root at the top and leaves at the bottom) undirected unweighted Tree with $V = 15$ vertices consisting of 8 leaf vertices and 7 internal vertices.

4. Draw an undirected Graph with $V = 9$ vertices and $E = 12$ edges. Identify two vertices: Source vertex s and destination/sink vertex t so that there are 16 *distinct* shortest paths from s to t. Two paths are distinct if there is at least one edge difference between the two paths.

Section B Marks = _____ + _____ + _____ = _____

C Applications (50 marks)

C.1 SSSP in a Special "SLL" (20 marks)

Disclaimer: This question is a modified version of a Kattis problem (CC-by-SA).

You are given a special (Single/Singly) Linked List (SLL) with n ($2 \leq n \leq 10^5$) vertices (labeled vertex 0 (head) $\rightarrow 1 \rightarrow 2 \rightarrow \ldots \rightarrow n$-1 (tail); yes it is a rather long SLL). Going through a normal edge costs 1 unit. You will be given the value of n in the first line of input.

Not just that, you are also given a special Integer S ($2 \leq S \leq 10^5$) as the second line of input. You are told that for every pair of distinct vertices i and j ($i, j > 0$) in the linked list, there is also a special edge/reference from vertex $i \rightarrow j$ if $j > i$, $i\%S = 0$ and $j\%S = 0$. Going through a special edge costs 2 unit. Of course, because of these special edges, the SLL now looks like a graph.

Your job is to find the shortest path from vertex 0 to vertex n-1 and output this shortest path value (we do not need the actual shortest path).

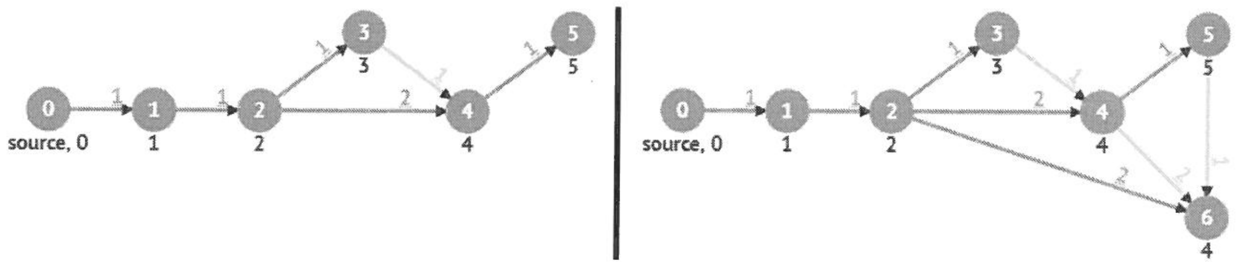

Figure 1: Left picture: $n = 6$, $S = 2$, shortest path value = 5 (via $0 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5$ or $0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5$); Right picture: $n = 7$, $S = 2$, shortest path value = 4 (only via $0 \rightarrow 1 \rightarrow 2 \rightarrow 6$).

Your solution: Design an efficient algorithm using the technique that we have learned in class (or beyond) to solve the problem above (remember that the last **3 marks** are for time complexity analysis). A skeleton Java code has been provided for you:

```
import java.io.*;

class C1 {
  public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    PrintWriter pw = new PrintWriter(System.out);
    // show how to read the inputs here (2 marks)
    // show how to compute the answer here (14 marks)
    // you can use Java API (if necessary) to implement any ADT that you need























    // show how to print the output here (1 mark)






    pw.close();
  }
} // the time complexity of my solution above is O(            ) (3 marks)
```

Section C.1 Marks = _____

C.2 Post World Cup 2018 Football Matches (Easier, 20 marks)

After FIFA World Cup 2018 ended recently, you are so into football. You now join a group of (virtual) friends (there are N players numbered from 1 to N and $2 \leq N \leq 100\,000$ - yes it is a large online gaming world) playing approximately M ($1 \leq M \leq 500\,000$) online football video game matches in the last few weeks alone. The first line of input contains two integers N and M in this format "`N M`".

The results of these M matches are recorded as M lines in this format: "`u > v`" that means player u beats player v (the actual final score is irrelevant). If you are wondering why there is no draw, it is because the players turn on the penalty shootout option so that all football matches that end in draw will continue to penalty shootout that will uniquely determine the winner eventually.

You want to test your hypothesis whether there is a transitivity relation between any three players a, b, c, i.e. if player a beats player b and player b beats player c, then player a should also beat player c (if they happen to go head to head in at least one of the M matches). If this hypothesis remains true for all M matches, output "`Correct hypothesis`". However if there is at least one case where this is not true, output "`The ball is round`".

Your solution: Design an efficient algorithm using the technique that we have learned in class (or beyond) to solve the problem above (remember that the last **3 marks** are for time complexity analysis). A skeleton Java code has been provided for you:

```
import java.io.*;

class C2 {
  // use this space for additional method(s) or global variable(s), if needed
  public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    PrintWriter pw = new PrintWriter(System.out);
    // show how to read the inputs here (5 marks)









    // show how to compute the answer here (10 marks)
    // you can use Java API (if necessary) to implement any ADT that you need



















    // show how to print the output here (2 marks)




    pw.close();
  }
} // the time complexity of my solution above is O(            ) (3 marks)
```

Section C.2 Marks = ____

C.3 Post World Cup 2018 Football Matches (The Real One, 10 marks)

Disclaimer: This question is a modified version of a Kattis problem (CC-by-SA).

Please refer to the previous question for the full problem statement. For this question, the only change is this. The format of the M matches results is either: "`u > v`", "`u = v`", or "`u < v`" that means player u beats/draws-with/loses-against player v. That's it, this time the matches can end in draws. Everything else is the same.

So if player a and b draw, player b beats player c, player c and d draw, then your hypothesis is broken if player a draws or beaten by player d (if they happen to go head to head in at least one of the M matches).

Your solution: As your solution *may likely* use the solution that you have written in Section C.2, just mention in *pseudocode* on what are the *changes* that you will make to your code in Section C.2 above in order to make your solution work for this variant that can have draws (remember that the last **3 marks** are for time complexity analysis).

```
// the time complexity of my solution above is O(            ) (3 marks)
```

Section C Marks = ____ + ____ + ____ = ____

Extra blank paper:

– End of this Paper, All the Best –