National University of Singapore

School of Computing

CS2040C - Data Structures and Algorithms

Final Assessment

(Semester 1 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 TEN (10) 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 10.
   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. Write your Student Number in the box below:

A	0							

This portion is for examiner's use only

Section	Maximum Marks	Your Marks	Remarks
A	20		
B	15		
C	65		
Total	100		

A Basics (20 marks)

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

Match the various data structure operations or algorithms on the second column with the *tightest*[1] *worst case* time complexity of those operations/algorithms (the typical time complexities are given in the last column). Every correct matching worth 1 mark.

The operations/algorithms referred below are the **unmodified version**, as per discussion in class, e.g. as currently discussed in VisuAlgo or as currently implemented in C++ STL. For graph-related operations, let n be the number of vertices and m be the number of edges. Otherwise, n denotes the size of data as usual. AM/AL/EL are the abbreviations for Adjacency Matrix/Adjacency List/Edge List, respectively. Unless specifically mentioned, all graph-related operations are performed on simple graphs stored in an AL data structure.

Note that *it is possible* that there are some operations/algorithms that have the same time complexity and thus there *may be* unused time complexity option(s).

No	Operations		Time Complexities
1	Finding the *next larger* item in a Hash Table		$O(1)$
2	Trying to remove a *non-existing* item from a Hash Table		$O(\log n)$
3	Finding the *previous smaller* item in a *possibly unbalanced* BST		$O(n)$
4	Updating a *previous value* into a *new value* in an AVL Tree		$O(n \log n)$
5	Sorting m edges in an *EL* by increasing weights		$O(n^2)$
6	Converting a graph stored in an *AM* into an *AL*		$O(n^3)$
7	Counting the number of *components* of an undirected graph		$O(n + m)$
8	Finding shortest path(s) from $a \rightarrow b$ in an *unweighted graph*		$O(m \log n)$
9	Finding shortest path from $a \rightarrow b$ in a *weighted tree*		$O((n + m) \log n)$
10	Finding shortest paths between *any* pair a, b in a *weighted graph*		$O(n^2 + nm)$

[1]What we meant by tightest worst case time complexity is as follows: If an operation of the stipulated data structure/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 $O(n^3)$, you will get wrong answer for this question.

A.2 Fill in the Blanks (10 marks)

Each answer worth 1 mark.

1. It is _____ to use C-style I/O routines like scanf/printf inside a C++ program.
2. There are two potential algorithms that solve the same problem. If algorithm A runs in $O(n^2)$ and algorithm B runs in $O(n \log n)$ for *any* possible inputs for that problem, we say that algorithm A is _____ than algorithm B.
3. You are given a random stack of N ($1 \leq N \leq 800$) student final assessment scripts identifiable only by 9-characters NUS student numbers. If you want to order the scripts based on increasing student numbers, you will use this sorting algorithm: _____.
4. We can implement two major Stack operations: Push(v) and v = Pop() that are used for Stack ADT operations of the same name using *alternative* data structure while retaining their $O(1)$ performance. That alternative data structure is: _____.
5. We can implement two major Binary Heap operations: Insert(v) and v = ExtractMax() that are used for Priority Queue ADT operations: enqueue(v) and v = dequeue() using *alternative* data structure while retaining their $O(\log n)$ performance. That alternative data structure is: _____.
6. The best data structure to implement Table ADT if you just need the basic three operations: Insert(key, satellite_data), satellite_data = Search(key), Remove(key) and key is of type integer is: _____.
7. To print out the largest integer stored in a C++ STL `set<int> S`, we can use this command: `cout <<` _____ `<< endl;`
8. The most suitable graph data structure if we need to *frequently enumerate neighbors of a vertex* is _____.
9. To check if two vertices in a given graph are connected directly via an edge or indirectly via a path, we can use _____ algorithm.
10. In an unweighted _____ graph of n vertices ($10 \leq n \leq 500$), the shortest path between *any* pair of vertices is always just 1 unit.

Section A Marks = _____ + _____ = _____

B Analysis (15 marks)

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

1. The performance of Hash Table with Open Addressing collision resolution technique *degrades over time* if there are frequent deletions/removals of existing items of Hash Table and insertions of new items into the Hash Table.

2. The largest element in any AVL Tree (of more than 3 vertices) is always a leaf vertex.

3. If we need to store a *near complete weighted* graph (with $V \approx 1K$ vertices), then using either Adjacency Matrix or Adjacency List graph data structure is equally good.

4. Depth First Search algorithm can also be used to find the shortest path in an unweighted graph.

5. The shortest path between any pair of vertices in a tree is always unique.

Section B Marks = _____

C Applications (65 marks)

C.1 Isomorphic BSTs (15 marks)

Inserting 3 distinct integers: {2, 1, 3} into an initially empty Binary Search Tree (BST) will produce Figure 1, Left. Inserting the same 3 distinct integers *but in different sequence*: {2, 3, 1} into an initially empty BST will also produce the same result. We call this phenomenon: *(BS)Tree Isomorphism*. Informally, two BSTs (two trees) that contains the same number of vertices (in this problem, the same set of N vertices from {1, 2, .., N}) connected in the same way are said to be isomorphic.

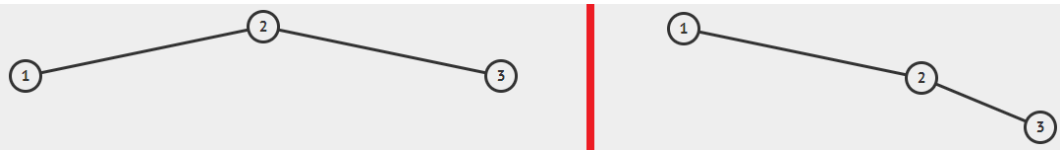

Figure 1: BST Insertion; Left: Inserting {2, 1, 3} or {2, 3, 1}; Right: Inserting {1, 2, 3}

Given an integer N and two permutations A, B of the first N integers {1, 2, ..., N}, your task is to write a function `bool IsIsomorphic(int N, vector<int>& A, vector<int>& B)` that returns TRUE if inserting N integers following *either* permutation A or permutation B into an initially empty BST will produce the same (isomorphic) BST. You have to return FALSE otherwise.

Examples: Let `vector<int> X = {2, 1, 3}, Y = {2, 3, 1}, Z = {1, 2, 3}`,
`IsIsomorphic(3, X, Y)` should return TRUE but `IsIsomorphic(3, X, Z)` should return FALSE.

For up to full 15 marks, your solution must work for $1 \leq N \leq 20$.
For up to partial 8 marks, your solution must work for $1 \leq N \leq 3$ (think carefully).

The time complexity of your algorithm is: O(_____).

C.2 Rain on a 2D Grid, Easier (18 marks)

You are given a 2D grid of size $R \times C$ ($1 \leq R, C \leq 1\,000$). **For this version, all cells have distinct heights ranging from 1 to $R \times C$.** You can assume that the height of cells *outside* this 2D grid to be very high, higher than $R \times C$ (e.g. $R \times C + 1$). A heavy rain falls on this 2D grid. As you know, water flows from higher cell to lower (*or equal-height*) cell that is its neighbor (to its North/East/South/West). **However, as all cells have distinct heights in this version, you will not encounter the equal-height case yet**. A cell will be called *flooded* after this heavy rain if it cannot send the rain water to any other (lower *or equal-height*) cells that are connected to it. Now design an algorithm to count the number of cells that are flooded.

Example 1: If you are given the following 1×9 grid, the answer is 3 as cells with height 1, 5, and 6 (highlighted with square brackets) will be flooded:

```
[1] 2  3  4  7 [6] 8  9 [5]
```

Example 2: If you are given the following 3×5 grid, the answer is 4 as cells with height 1, 3, 7, and 10 (highlighted with square brackets) will be flooded:

```
[1] 2  4 [3] 5
 6 12  8  9 15
11 [7]13 14[10]
```

The time complexity of your algorithm is: O(_____).

C.3 Rain on a 2D Grid, Harder (7 marks)

Disclaimer: The following question is modified from a Kattis problem (CC-by-SA).

You are given the same problem as earlier, but this time the heights of the cells are **not necessarily distinct** (so maybe not all heights between $[1..R \times C]$ are all used). Notice that in this version, there can be two neighboring cells **with the same height**. How are you going to deal with this version?

Example 1: If you are given the following 1×9 grid, the answer is 7 (all the 1s) will be flooded:

```
[1] 2 [1  1  1  1  1  1] 2
```

Example 2: If you are given the following 3×5 grid, the answer is 7 (5 '1s' and 2 '3s') will be flooded:

```
[1] 4 [3  3] 5
[1  1] 8  5  5
11 [1  1]14  5
```

The time complexity of your algorithm is: O(_____).

C.4 Shrinking Tunnel, Easier (18 marks)

You are at the entrance of a dungeon with n ($2 \leq n \leq 10\,000$) junctions and m ($1 \leq m \leq 20\,000$) shrinking tunnels. Every time you go through a tunnel i that connects junction u to junction v (one-way), you will get shorter by a shrinking factor of f_i ($0.0 < f_i < 1.0$) – that's it, you will never disappear and you will never retain your original height.

Now given the map of the dungeon and the shrinking factors of all its tunnels, decide which path that you have to take from the entrance (the junction 0) to reach the exit (the junction $n-1$) in order to *minimize your height loss*. It is guaranteed that at least one such path exists. For this problem, just output your final height relative to your original height in the best path. Design the best algorithm!

For this easier version, **let's the shrinking factor f for all tunnels to be equal**.

Example: If there are $n = 3$ junctions with $m = 3$ tunnels that connect $0 \to 1$, $1 \to 2$, $0 \to 2$, and all tunnels have $f = 0.9$, you should go through $0 \to 2$ to make yourself $0.9\times$ shorter than original rather than taking the detour path $0 \to 1 \to 2$ which makes yourself $0.9 \times 0.9 = 0.81\times$ shorter than original (worse than the 0.9 via direct path $0 \to 2$ earlier). So, you output 0.9.

The time complexity of your algorithm is: O(_____).

C.5 Shrinking Tunnel, Harder (7 marks)

Disclaimer: The following question is modified from a Kattis problem (CC-by-3.0).

You are given the same problem as earlier, but in this harder version, **the shrinking factor f for various tunnels can vary**.

Example: If there are $n = 3$ junctions with $m = 3$ tunnels that connect $0 \rightarrow 1$ with $f = 0.9$, $1 \rightarrow 2$ with $f = 0.9$, and $0 \rightarrow 2$ with $f = 0.8$ (notice the difference), you should go through the detour path $0 \rightarrow 1 \rightarrow 2$ to make yourself $0.9 \times 0.9 = 0.81\times$ shorter than original rather than taking the direct path $0 \rightarrow 2$ which makes yourself $0.8\times$ shorter than original (slightly worse than the 0.81 via detour path $0 \rightarrow 1 \rightarrow 2$ earlier). So, you output 0.81.

The time complexity of your algorithm is: O(_____).

Section C Marks = ____+(____+____)+(____ +____) = _____

– End of this Paper, All the Best –