National University of Singapore

School of Computing

Semester 1 (2013/2014)

CS2010 - Data Structures and Algorithms II

Written Quiz 2 (15%)

Wednesday, November 06, 2013, 10.00am-11.17am (77 minutes)

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this question paper until you are told to do so.
2. Quiz 2 is conducted at COM1-2-206/SR1.
3. This question paper contains THREE (3) sections with sub-questions.
   It comprises TEN (10) printed pages, including this page.
4. Write all your answers in this question paper, **but only in the space provided**.
   You can use either pen or pencil. Just make sure that you write **legibly**!
   Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
5. This is an **Open Book Examination**. You can check the lecture notes, tutorial files, problem set files, Steven's 'Competitive Programming' book (any editions), or *any other printed material* that you think will be useful. But remember that the more time that you spend flipping through your files implies that you have less time to actually answering the questions.
6. **Please write your Matriculation Number here:** ____________________
   and your Tutorial Group Number (or Tutor Name): ____________________
   But *do not* write your name in order to facilitate unbiased grading.
7. You are encouraged to use Java syntax to answer questions in this paper as it is much clearer. However, as promised, you can still use pseudo-code but beware of penalty marks for **ambiguous answer**. You can also use **standard, non-modified** algorithms discussed in class by just mentioning their names. However, if you need to modify **any part** of the standard algorithm, you have to write the **full algorithm**.
8. All the best :).

–This page is intentionally left blank. You can use it as ‘rough paper’–

1 Short Questions: (15 marks); Marks = _____

Most of the answers for this set of questions can be found in the lecture notes (or mentioned during the lecture), tutorial files, PS files, Competitive Programming book, etc.
Can you find the answers *as fast as you can*? It is $O(1)$ if you already have the answer in your memory. Some questions still require a bit of thinking process though.

1. (2 marks) There are two important heuristics used in the Union-Find Disjoint Sets (UFDS) data structure in order to make its operations very efficient. They are:
   1. ______________________________.
   2. ______________________________.
2. (3 marks) Group these five graph algorithms together if they have the same standard time complexity: BFS, Prim's, DFS, Kruskal's, Bellman Ford's.
   You can assume that the algorithms are as shown in class and without any fancy optimizations.
   Group 1: ______________________________.
   Group 2 (if any): ______________________________.
   Group 3 (if any): ______________________________.
   Group 4 (if any): ______________________________.
   Group 5 (if any): ______________________________.
3. (6 marks) For these six sub-questions, you are **not allowed** to use any additional data structure. What is the **tightest** time complexity of DFS if it is run from a source vertex v of a graph G, if:
   1. This graph G is a general graph and is stored in an Adjacency List: __________.
   2. This graph G is a tree and is stored in an Adjacency List: __________.
   3. This graph G is a general graph and is stored in an Adjacency Matrix: __________.
   4. This graph G is a tree and is stored in an Adjacency Matrix: __________.
   5. This graph G is a general graph and is stored in an Edge List: __________.
   6. This graph G is a tree and is stored in an Edge List: __________.
4. (4 marks) What is the **best** algorithm (write down the name and it's default time complexity) to find the shortest paths from source s to the other vertices in the graph if the graph is:
   1. undirected, all edges have positive weights: ______________.
   2. directed, unweighted: ______________.
   3. acyclic, directed, weighted: ______________.
   4. undirected, weighted, and may have negative weight cycle: ______________.

2 Analysis (15 marks); Marks = _____

Prove (the statement is true) or disprove (the statement is false) the following statements below.
If you want to prove it, provide the proof (preferred) or at least a convincing argument.
If you want to disprove it, provide at least one counter example.
Three marks per each statement below (1 mark for saying correct/wrong, 2 marks for explanation):
Note: You are only given a small amount of space below (i.e. do **not** write too long-winded answer)!

1. The original Dijkstra's algorithm (verbatim, as outlined in Lecture 09) can be used to detect the presence of negative weight cycle.

   No, it cant because each vertex is only visited once and having a cycle implies that a vertex is visited again

   can detect negative edge -> when we relax an edge of a vertex that is not in pq(already been visited once) but cannot detect a negative cycle each vertex is only visited once, so vertices will not be added into pq

2. The worst case time complexity of Bellman Ford's algorithm when executed on a graph that is stored in an **Adjacency Matrix** is definitely $O(V^3)$ and there is no way to make it faster in general case. If necessary, you are **allowed** to use additional data structure.

   FALSE we can convert the adjacency matrix to edge list by running bfs/dfs on the adjacency matrix to find all the edge weights in the graph
   then using the edge list we can do normal bellman ford algo in O(VE) time
   totaal time complexity is at most O(V^2 E) + O(VE) = O(V^2 E)

3. There is **no input graph** that can make Bellman Ford's algorithm runs slower than $O(VE)$.
   PS: The input graph has to be a simple graph without self-loop or multiple edges.

   simple graph: a graph with no loop(no edge starting and going back to same vertex) or
   multiple edges(only one edge between any two vertices)

   false. a simple connect graph will take O(V^2)

   true

4. From the same source vertex s, the Depth First Search spanning tree starting from s will **always be different** than the Shortest Path spanning tree starting from s on any input graph.
   PS: The input graph has to be a simple graph without self-loop or multiple edges.

   false. in a simple graph with 2 vertices the dfs spanning tree is the same as the shortest path spanning tree
   which is the edge between the two vertices

   even for tree. shortest path spanning tree is same as dfs spanning tree

5. From the same source vertex s, both Depth First Search and Breadth First Search algorithms starting from s will **always** visit the same set of vertices on any input graph.
   PS: The input graph has to be a simple graph without self-loop or multiple edges.

   True. both dfs and bfs will produce a spanning tree of the graph which is a set of all the vertices in the input graph
   the only difference is that the order will be different as a set is unordered

   both will visit all vertices reacheable from S(those in the same component)

3 Applications (20 marks), Student's Marks: ___ + ___ + ___ = ___

3.1 Social Network (4 marks), Student's Marks: ___

Steven wants to build another social network website to rival Facebook. He wants his social network to support at least 1/3 of the current world population (the world population is estimated to be around 7.2 billion as of 9 October 2013) so that it can beat Facebook (which has around 1.1 billion users according to a news on 1 May 2013). One important feature that will be frequently used in Steven's social network is the feature to list down someone's friend list quickly.

You are Steven's technical advisor. Please recommend one of the three graph data structures taught in class (Adjacency Matrix, Adjacency List, or Edge List) which **has a higher chance** of potentially usable for this ambitious project.

Note: Marks only awarded for the (short) explanation, not for the choice.

Adjacency list as accessing a user takes worst case O(V) time and listing down all of the users friends takes worst case O(2E) time so the total time taken is O(VE) on average as it is difficult for all 1.1 billion users to each be connected to each other and form a complete graph as compared to O(V^2) of adjacency list regardless of user and edge list only tells us all the different connections in the graph but not easy to pick out only user's friends

3.2 Improvement of Prim's? (4 marks), Student's Marks: ___

Steven feels that his `PrimDemo.java` can be improved **a bit**. Please look at the main part of Steven's Java code that implements Prim's algorithm below (all comments are removed so that the three new important comments stand out):

```
private static void process(int vtx) {
  taken.set(vtx, true);
  for (int j = 0; j < AdjList.get(vtx).size(); j++) {
    IntegerPair v = AdjList.get(vtx).get(j);
    if (!taken.get(v.first())) {
      pq.offer(new IntegerPair(v.second(), v.first()));
} }

// inside the Main method
  taken = new Vector < Boolean >(); taken.addAll(Collections.nCopies(V, false));
  pq = new PriorityQueue < IntegerPair > ();
  process(0);
  int taken_counter = 1; // NEW ADDITION, PREVIOUSLY IT DOES NOT EXIST
  int mst_cost = 0;
  // PREVIOUSLY: while (!pq.isEmpty()) {
  while (taken_counter < V) { // NOTICE THE CHANGE OF LOOP CONDITION
    IntegerPair front = pq.poll();
    if (!taken.get(front.second())) {
      mst_cost += front.first();
      taken_counter++; // NEW ADDITION, PREVIOUSLY IT DOES NOT EXIST
      process(front.second());
  } }

  System.out.printf("Final MST cost %d\n", mst_cost);
```

Now, explain if this modification results in a faster and still valid Prim's algorithm implementation? If yes, on what cases it will be faster? If no, explain why!

it will not work as prims is a a greedy algorithm which chooses the smallest neighbour of each vertex which is why the pq was used. this pseudo code doesnt do any ordering or check for finding the locally optimum choice

3.3 Lego Mindstorms EV3 (12 + 5 bonus = 17 marks)

Steven has a new toy, LEGO Mindstorms EV3 (which he will pass to Jane in a few years time). He wants to command his robot to move from one cell to another cell in an $M \times N$ grid containing mostly '.' (passable cells) and some '#' (blocked cells). The diagram below shows a sample 4×7 grid with 8 obstacle cells (the '#'s):

```
.......
.#####.
.#...#.
...#...
```

Initially, Steven's robot is at coordinate (0, 0)—the top-left corner of the grid—and faces east.
At each cell, Steven's robot can only do one of the two actions below:

1. Move forward by one cell according to it's current direction.
   For example, if the robot currently at coordinate (0, 0) and faces east, it will be in coordinate (0, 1) and still faces east after this action.
   Such action consumes 3 seconds.
2. Rotate the robot by 90 degrees clockwise (turn right); the robot stays in the current cell.
   For example, if the robot currently at coordinate (0, 0) and faces east, it will still be in coordinate (0, 0) but now faces south after this action.
   Such action consumes 2 seconds.

Write a program that Steven can upload to his robot so that his robot can move from coordinate (0, 0) to coordinate (M-1, N-1) using the **minimum amount of time**! Of course, Steven's robot **cannot** go outside the $M \times N$ grid and cannot move to a blocked cell throughout the execution of the program. When the robot stops at coordinate (M-1, N-1), it can face any direction.

On the sample grid above, the best robot path is as indicated below with a total time of 29 seconds, i.e. move forward 6 times (18 seconds), turn right (2 seconds), and then finally move forward 3 times (9 seconds):

```
    111
0369258
.#####3<-(actually 23)
.#...#6<-(actually 26)
...#..9<-(actually 29)
```

On another sample grid below, the best robot path is as indicated below with a total time of 20 seconds, i.e. turn right (2 seconds), move forward 2 times (6 seconds), turn right 3 times so that the robot faces south → west → north → east (6 seconds), and finally move forward 2 times (6 seconds):

```
0##
5##
860<-(actually 20)
 ^---(actually 16)
```

3.3.1 Graph Modeling (9 marks), Student's Marks: ___

Convert the problem above into a graph problem!
What do the vertices and the edges of your graph represent? (3 marks)

What is the upper bound of the number of vertices and edges in your graph? (2 marks)

What is the graph problem that you want to solve? (2 marks)

What is the most appropriate graph algorithm to solve this problem? (2 marks)

Note for Section 3.3.2. There are different possible solutions for this original but not new problem. Some of the possible solutions just require CS2010 knowledge taught so far. After this Quiz 2, you can attempt the same problem (plus one additional constraint) as PS8. For Quiz 2, your written solution will be awarded one of the four possible marks based on the criteria below:

Marks	Requirements
1	Incomplete or very buggy solution (NOTE: Partial Marks are in Section 3.3.1)
3	Your solution only work if turn right action consumes 0 second, i.e. 'instantaneous'.
3 + 2 = 5	Your solution is judged to be better than the 3 marks solution but not 100% correct.
3 + 5 = 8	Your solution is 100% correct.
	Steven knows that writing the complete solution will take a lot of time.
	Therefore, if you can complete this in 77 minutes, you will get bonus 5 marks.

3.3.2 Your Solution (3 + bonus 5 = 8 marks), Student's Marks: ___ Total Q3.3: ___

Write your solution below. Pseudo code is OK, but Steven prefers the more clear Java based solution. Assume that the $M \times N$ grid has been read from the input file and stored in a 2D array `grid`.

```
int Query(char[][] grid, int M, int N) { // grid, M, N are as described above









}
```

– End of this Paper –

Candidates, please do not touch this table!

Question	Maximum Marks	Student's Marks
1	15	
2	15	
3	20 + 5	
Total	min(50, 50 + 5)	

Note that the marks for this Written Quiz 2 is capped to 50 marks.