National University of Singapore

School of Computing

EXAMINATION FOR

Semester 1 AY2013/2014

CS2010 - Data Structures and Algorithms II

Nov 2013, Time Allowed: 2 hours

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this question paper until you are told to do so.
2. This examination paper contains FOUR (4) sections with sub-questions.
   It comprises THIRTEEN (13) printed pages, including this page.
3. This is an **Open Book Examination**. You can check the lecture notes, tutorial files, problem set files, or any other books. But remember that the more time that you spend flipping through your files implies that you have less time to actually answering the questions.
4. Answer **ALL** questions within the space in this booklet.
   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. When this final exam starts, **please immediately write your Matriculation Number here:** ____________________________ **(do not forget the last letter and do not write your name).**

This portion is for examiner's use only

Section	Maximum Marks	Student's Marks
1	21	
2	15	
3	20	
4	44	
Total	100	

1 Basic Understanding of CS2010 (21 marks); Marks = _____

Please fill in your answers on the blank spaces provided. Each question has different marks.

1. **(3 marks)** List down **three** applications of Kruskal's algorithm:
   1). ____
   ____,
   2). ____
   ____,
   3). ____
   ____.

2. **(12 marks)** Give a good and a bad input graph for the graph algorithms in the table below.
   - A good input graph will make the algorithm runs fast (i.e. in reasonable time) and produces the correct answer as intended for that algorithm (1 mark each).
   - A bad input graph will make the algorithm runs slowly (i.e. at its worst case time complexity), be trapped in an infinite loop, or produces the wrong answer (2 marks each).

   To standardize the answers, we assume that $V > 100$ and $E \geq V - 1$.
   Marks are only awarded if the (short) reason for your chosen answer is valid.
   The first entry is given as an example.

Graph algorithm	**Good input graph**	**Bad input graph**
1. Modified Dijsktra's My reason	Graph with non-negative weight The algorithm works correctly and runs in $O((V + E) \log V)$	Graph with negative weight cycle This causes Modified Dijkstra's to be trapped in an infinite loop
2. Toposort with DFS My reason	________________ ________________ ________________ ________________	________________ ________________ ________________ ________________
3. Prim's My reason	________________ ________________ ________________ ________________	________________ ________________ ________________ ________________
4. Original Dijsktra's My reason	________________ ________________ ________________ ________________	________________ ________________ ________________ ________________
5. Floyd Warshall's My reason	________________ ________________ ________________ ________________	________________ ________________ ________________ ________________

3. The questions below are based on the following 3×5 grid of **hexadecimal**:

E	D	C	B	7
8	4	3	5	6
A	9	2	1	0

Note: A, B, C, D, E, and F are hexadecimal symbols of decimal 10, 11, 12, 13, 14, and 15, respectively. We we do not use F (15) in the grid above as there are only $3 \times 5 = 15$ cells.

The source vertex s is the top-left cell and the target vertex t is the bottom-right cell.
We can go from one cell A to its North/East/South/West cell B only if the value of $A > B$.

The questions (**3 × 2 = 6 marks**):
1). The **length of** the shortest **unweighted** path between s and t is ________________ edges.
2). The **length of** the longest **unweighted** path between s and t is ________________ edges.
3). The **number of** paths between s and t is ________________.

2 Analysis (15 marks); Marks = _____

Prove (the statement is correct) or disprove (the statement is wrong) the 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. A Union-Find Disjoint Sets (UFDS) data structure which **uses both** the 'path compression' and 'union by rank' heuristics initially contains $N \geq 128$ disjoint sets. Therefore, **there is a way** to call the `unionSet(i, j)` operations of this UFDS in such a way that we end up with one set containing N items represented by a tree with height (rank) N-1.

2. Finding the **longest** path in a weighted graph when all edges are **non-positive** is an easy problem that has solution with polynomial time complexity.

3. There exists a Directed Acyclic Graph (DAG) with **exactly** two possible topological sorts only.

4. There is **no faster way** to compute the **All-Pairs Shortest Paths (APSP)** information of a **Directed Acyclic Graph (DAG)** other than to use the $O(V^3)$ Floyd Warshall's algorithm.

5. There exists a **polynomial time** algorithm to solve the **Traveling Salesman Problem** by modifying Depth-First Search (DFS) algorithm into a backtracking algorithm.

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

3 Learn on the Spot (20 marks)

3.1 Shortest Path Faster Algorithm (SPFA)

3.1.1 The Algorithm

Shortest Path Faster Algorithm (SPFA) is a Single-Source Shortest Paths (SSSP) algorithm that uses a queue to eliminate redundant operations in Bellman Ford's algorithm. The origin of this algorithm is unclear but some sources claim that this algorithm was published in Chinese by Duan Fanding in 1994. This algorithm is popular among Chinese programmers but it is not yet well known in other parts of the world. The term 'faster' can be misleading as it is not actually faster than a good implementation of Dijkstra's algorithm.

SPFA requires the following data structures:

1. A graph stored in an Adjacency List: `AdjList`.
2. A Vector of Integers `d` to record the distance from source vertex to every vertex.
3. A Queue of Integers `q` to store the vertices that has potential to cause edge relaxation.
4. A Vector of Booleans `in_queue` to denote if a vertex is currently in the queue or not.

The first three data structures are the same as in Dijkstra's or Bellman Ford's algorithms discussed in class. The fourth one is unique to SPFA. Below is one possible Java implementation of SPFA:

```
// assume the graph of n vertices is stored in AdjList and the source is vertex S
  Vector<Integer> d = new Vector<Integer>();
  for (i = 0; i < n; i++) d.add(INF);
  d.set(S, 0);
  Queue<Integer> q = new LinkedList<Integer>();
  q.offer(S);
  Vector<Boolean> in_queue = new Vector<Boolean>();
  for (i = 0; i < n; i++) in_queue.add(false);
  in_queue.set(S, true);

  while (!q.isEmpty()) {
    int u = q.poll();
    in_queue.set(u, false); // SUB-QUESTION 4 INVOLVES THIS LINE
    for (j = 0; j < AdjList.get(u).size(); j++) { // all outgoing edges from u
      int v = AdjList.get(u).get(j).first();
      int weight_u_v = AdjList.get(u).get(j).second();
      if (d.get(u) + weight_u_v < d.get(v)) { // if can relax
        d.set(v, d.get(u) + weight_u_v); // relax
        if (!in_queue.get(v)) { // SUB-QUESTION 5 INVOLVES THIS LINE
          q.offer(v);
          in_queue.set(v, true);
  } } } }
```

3.1.2 Basic Understanding of SPFA Algorithm (20 marks); Marks = _____

Run the SPFA code above on the three sample graphs that have been shown in class.
Assume that the neighbors of each vertex are sorted in ascending order of vertex number.
The execution of SPFA code on graph in Figure 1.A is shown below as an example.

First, we put vertex 2 in the queue q. The queue q now contains {2}.
Process vertex 2, relaxes 2 → 0, 2 → 1, and 2 → 3, sets d[0], d[1], and d[3] to 6, 2, and 7, respectively. The queue q now contains {0, 1, 3}.
Process vertex 0, relaxes 0 → 4 and sets d[4] to 7. The queue q now contains {1, 3, 4}.
Process vertex 1, relaxes 1 → 3 and sets d[3] to 5. But vertex 3 is already in q, we do not add another duplicate. Edge 1 → 4 cannot be relaxed and thus ignored. The queue q now contains {3, 4}.
Process vertex 3. But nothing happen as edge 3 → 4 cannot be relaxed and thus ignored. The queue q now contains {4}.
Process vertex 4. But nothing happen as vertex 4 has no outgoing edge.
The queue q is now empty and SPFA algorithm stops here.

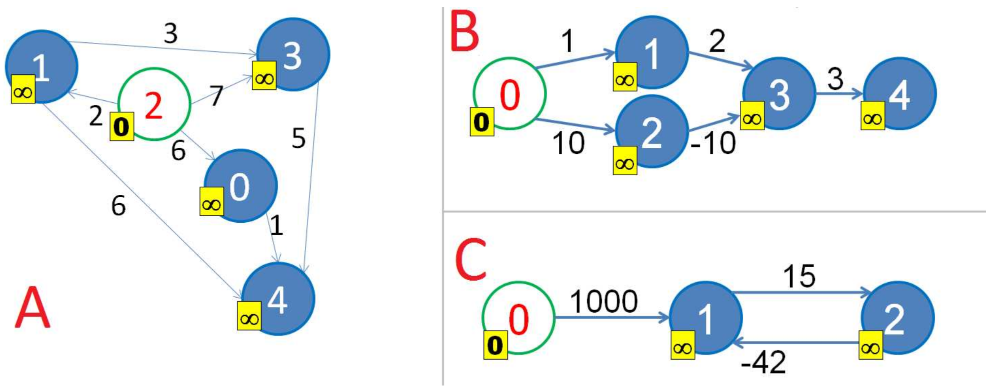

Figure 1: Sample Graphs as Shown in Class

1. (4 marks) Briefly explain the execution of SPFA algorithm on Figure 1.B.

2. (4 marks) Briefly explain the execution of SPFA algorithm on Figure 1.C.

// You can continue your answer for sub-question 2 here.

3. (4 marks) Does this algorithm **always** terminate and give correct SSSP information on weighted graph with negative weight cycle?

4. (4 marks) What happen if the line below is commented from the code in the previous page:
   . `in_queue.set(u, false);`
   Will the SPFA algorithm still runs correctly?

5. (4 marks) What happen if the line below:
   . `if (!in_queue.get(v)) {`
   is replaced with
   . `if (true) {`
   Will the SPFA algorithm still runs correctly?
   If still correct, does it becomes faster or slower?
   If this modification may cause wrong answer, explain why!

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

4 Applications (44 marks); Marks = _____ + _____ = _____

4.1 Facebook Privacy Setting (19 marks)

4.1.1 Definition

In Facebook, we can set our privacy setting so that only our Friends of Friends (and our direct friends) can look at our profile (see Figure 2). Our direct friends are classified as having degree 1 (one hop) to us. Our 'friends of friends' are therefore classified as having degree 2 (two hops away).

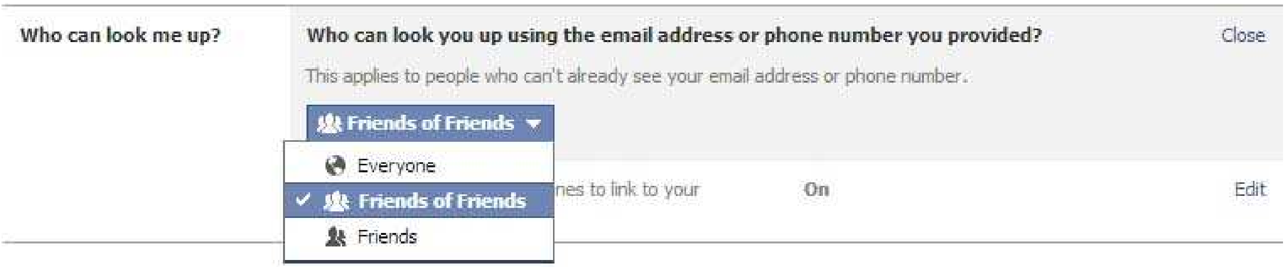

Figure 2: Restricting Access to Friends of Friends (degree 2) Only

Let's **assume** that we have **somehow** managed to store Facebook graph in an **Adjacency List** called `AdjList` (PS: We know that Facebook uses a much better graph data structure than this). Each user profile is given a unique integer index i. The friend list of profile i is stored in `AdjList[i]` and **sorted in ascending order**.

Now, we now want to check whether a user i can access user j's profile if user j sets his/her privacy setting to be degree 2 only as shown in Figure 2. The function `CanAccess(i, j)` should return true if i is classified as degree 2, 1, or 0 of user j, or return false otherwise[1].

4.1.2 (Optional) $O(V + E)$ Naive Solution (7 marks); Marks = _____

Please implement the function `CanAccess(i, j)` using a simple $O(V + E)$ naive solution.
Note that if you can already find the better $O(k)$ solution in Section 4.1.3, you can choose to leave this part blank and still get the full marks. But please write the $O(V + E)$ naive solution in case you have no clue or unsure with your $O(k)$ solution.

```
function Boolean CanAccess(int i, int j) { // write an O(V + E) solution









}
```

[1] This problem is from a real Google interview question, but it has been rewritten using another background story.

```
% you can continue writing the O(V + E) solution here in case you need more space
```

4.1.3 $O(k)$ Efficient Solution (12 marks); Marks = _____ + _____ = _____

The size of V or E in a Facebook graph can be very big. Therefore, an $O(V + E)$ solution may not be the best solution. Please implement the function `CanAccess(i, j)` using an **efficient** $O(k)$ **solution** where $k = k_i + k_j$ and k_i/k_j is the number of friends of user i/j. The memory is also at premium and therefore we **cannot use additional data structure** other than a few extra variables. The grading scheme for this part is very strict, i.e. 0 (blank, incorrect, slower than $O(k)$, or uses additional data structure), or 10 (minor error(s)), 12 (fully correct).

```
function Boolean CanAccess(int i, int j) { // write an O(k) solution

}
```

4.2 Marks Analysis (25 marks)

4.2.1 Definition

Steven prepared 5 sections in his other exam (not this exam). Being a very statistics-minded lecturer, he has estimated the **expected range of scores** for each section for that exam as in Table 1:

Section 1	**Section 2**	**Section 3 (Bonus)**	**Section 4**	**Section 5**
[8-13]	[14-19]	[1-1]	[20-25]	[15-26]

Table 1: Expected Range of Scores for Various Sections of an Exam

Now, Steven has a weird request. He wants to know how many ways a student can achieve a unique score of 77 on that exam?

After thinking 'quite hard', Steven realizes that the answer for his own question is 108 different ways. Of course, he uses a much smarter way than enumerating all the 108 ways:

1. 13+19+1+25+19 = 77
2. 13+19+1+24+20 = 77
3. 13+19+1+23+21 = 77

... 104 others ways ...

108. 11+14+1+25+26 = 77

Now, Steven wants to generalize this problem. Given N sections ($2 \leq N \leq 7$), N pairs of lowest (lo) and highest (hi) scores for each section ($0 \leq lo \leq hi \leq 100$), and a target score T ($0 \leq T \leq 100$), compute how many ways we can sum the (integer) scores from each section (which must be within the given range) so that the sum equals to T.

4.2.2 Graph Modeling (12 marks); Marks = _____

Major hint: This problem **can be modeled** as a Directed Acyclic Graph (DAG).

1. What do the vertices and the edges of your DAG represent? (5 marks)

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

3. What is the (graph) problem that you want to solve? (3 mark)

4. What is the most appropriate (graph) algorithm to solve this problem? (2 mark)

4.2.3 Your Solution (13 marks); Marks = _____

```
// you can assume that: int N, int T, int[] lo, int[] hi
// have been declared and populated with appropriate values, i.e.
// for the sample, N = 5, T = 77, lo = {8,14,1,20,15}, and hi = {13,19,1,25,26}
int Query() { // returns the required answer: for the sample, the answer is 108










}
```

– End of this Paper –