National University of Singapore

School of Computing

CS2010 - Data Structures and Algorithms II

(Semester 1: AY2014/15)

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 FOUR (4) sections with sub-questions.
   It comprises TWELVE (12) printed pages, including this page.
3. This is an **Open Book Assessment**. 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 assessment starts, **please immediately write your Student Number here: A0**[__][__][__][__][__][__][__] **(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	20	
2	15	
3	20	
4	25	
Total	80	

1 Basic Understanding of CS2010 (20 marks); Marks = _____

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

1. **(5 marks)** List down **five** *categorically distinct* applications of BFS algorithm:
   1). _____
   _____,
   2). _____
   _____,
   3). _____
   _____,
   4). _____
   _____,
   5). _____
   _____.

2. **(7 marks)** Future feature[1] of VisuAlgo online quiz: "Draw a graph as your answer".

   (a) **(3 marks)** Draw $E = 10$ undirected and weighted edges to create a simple connected weighted undirected graph with $V = 5$ vertices below (the positions of the vertices are fixed) so that either the optimized Prim's (that stops after taking V vertices or $E = V$-1 edges) or optimized Kruskal's algorithm (that stops after taking $E = V$-1 edges, i.e. when there is only 1 disjoint set left) **only** need to examine $E = 5\text{-}1 = 4$ edges and immediately declare those 4 edges as the MST. The weights of the edges have to be distinct.

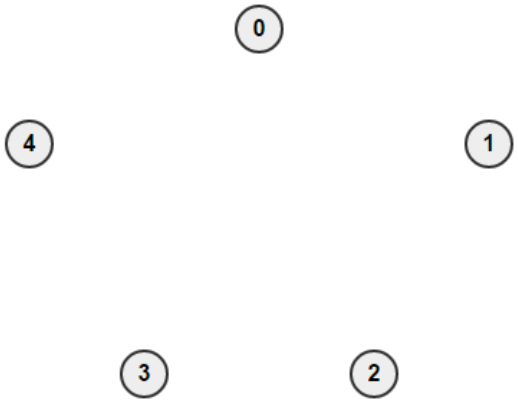


Figure 1: Easy Test Case for the Optimized Prim's/Kruskal's Algorithm

Please give a brief explanation on why your graph is the required answer:
_____
_____.

[1] We hope that this feature is available at `http://training.visualgo.net` by year 2015.

(b) **(4 marks)** Draw $E = 10$ undirected and weighted edges to create a simple connected weighted undirected graph with $V = 7$ vertices below (the positions of the vertices are fixed) so that the optimized Kruskal's algorithm have to examine **all** $E = 10$ **edges** before stopping with the MST. The weights of the edges have to be distinct.

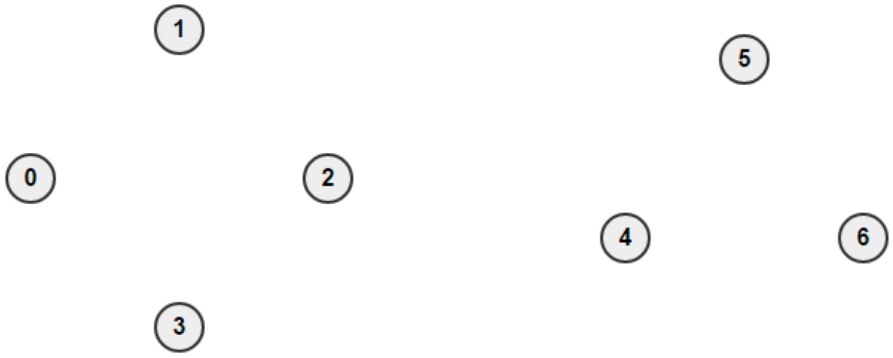


Figure 2: Hard Test Case for the Optimized Kruskal's Algorithm

Please give a brief explanation on why your graph is the required answer:

---.

3. **(8 marks)** The questions below are based on the following 4×5 grid of **hexadecimal**:

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

Note: A/B/C/D/E/F are hexadecimal symbols of decimal 10/11/12/13/14/15, respectively.
A cell with content '#' has value 1000.

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 with value X to its North/East/South/West cell with value Y if $X > Y$.

The questions (each question worth **2 marks**):
1). There are _____ cells that are reachable from the source vertex (inclusive).
2). The **length of** the shortest **unweighted** path between s and t is _____ edges.
3). The **length of** the longest **unweighted** path between s and t is _____ edges.
4). 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. The edge with the largest weight inside a connected weighted graph G is always **not part of any Minimum Spanning Tree (MST)** of G.

2. All *bridges* inside a connected weighted graph G are always part of the MST of G.
   A bridge is defined as an edge of a graph G whose removal disconnects G.

3. There is only one unique possible MST (structure) in a connected weighted graph G where all E edges have distinct weight.

4. Let the edge (a, b) be the edge with the minimum weight over all other edges in the graph G. The shortest path from source vertex a to destination vertex b **must definitely pass through this edge** (a, b). Note: There may be other possible paths from vertex a to vertex b in G.

5. Suppose that a Directed Acyclic Graph (DAG) G has X different topological orders. If we delete a single edge of G, the resulting graph G' will **definitely has** $> X$ different topological orders.

3 Learn on the Spot (20 marks)

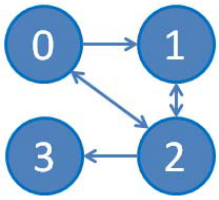
3.1 Compact and Fast Adjacency Matrix for Small Unweighted Graphs

3.1.1 Definition

The Adjacency Matrix of an unweighted graph is typically implemented using a 2D Boolean array `Boolean[][] AdjMat = new Boolean[V][V]` where V is the number of vertices in the graph. The value of `AdjMat[i][j]` is 1 if there is an unweighted edge `(i, j)` in the graph, or 0 otherwise.

When the graph is small ($0 \leq V \leq 32$), we can replace the 2D integer array with a 1D integer array `int[] CAM = new int[V]`. Now we compress each row of Booleans into one bit string, reverse the bit string (left $\longleftrightarrow$ right), and store the integer (decimal) value of this bit string.

For example, see a small unweighted (directed) graph as shown in Figure 3—left which has the typical 2D Boolean Adjacency Matrix as shown in Figure 3—middle. Now if we compress each row into one bit string, reverse it, and store the corresponding integer value of that bit string, we have a more compact graph data structure as shown in Figure 3—right, i.e. an integer array with 4 values: {`6,4,11,0`} is all we need to represent the graph shown in Figure 3—left.



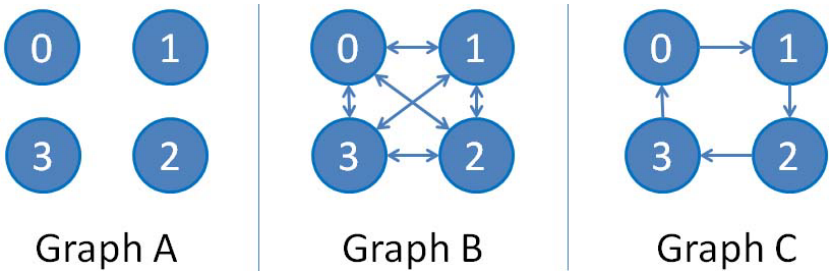
	0	1	2	3
0	0	1	1	0
1	0	0	1	0
2	1	1	0	1
3	0	0	0	0

	Bit String	Reversed	Integer
0	0110	0110	6
1	0010	0100	4
2	1101	1011	11
3	0000	0000	0

Figure 3: The Compact and Fast Adjacency Matrix for This Graph is {`6,4,11,0`}

3.1.2 Manual Checks (6 marks); Marks = _____

For each of the three small unweighted graphs below, please determine the content of array `CompactAdjMat`. You have to mention the integer values, not just the bit string. Please see the sample in Figure 3.
You will be given 1 mark if there is a minor mistake or 2 marks if correct.



Answer for Graph A: _____.

Answer for Graph B: _____.

Answer for Graph C: _____.

3.1.3 Compact Adjacency Matrix Operations (14 marks); Marks = _____ + _____ = _____

In this section, please implement **five** basic functions to manipulate this graph data structure.
The skeleton file for class `CompactAdjacencyMatrix` has been given below.
One sample function `EdgeExists` has been written for you.
In case you forget the required bit manipulation functions, you can refer to Table 1.

Task	**Bit operation**	**Example(s)**	
		Before	**After**
Set/turn on the j-th bit of integer S	$S \mid= (1 << j)$	$S = 6, j = 0$	$S = 7$
Check if the j-th bit of integer S is on	$T = S \,\&\, (1 << j)$	$S = 6, j = 1$	$T = 2$
		$S = 6, j = 0$	$T = 0$
Clear/turn off the j-th bit of integer S	$S \,\&= \sim (1 << j)$	$S = 6, j = 1$	$S = 4$
Toggle the j-th bit of integer S	$S \,\hat{}= (1 << j)$	$S = 6, j = 1$	$S = 4$
		$S = 6, j = 0$	$S = 7$
Get the value of the least significant bit of S	$T = S \,\&\, (-S)$	$S = 6$	$T = 2$
		$S = 4$	$T = 4$
Turn on all n bits of S	$S = (1 << n) - 1$	$S = 0, n = 3$	$S = 7$

Table 1: Summary of $O(1)$ Bit Operations

```
class CompactAdjacencyMatrix { // to represent unweighted graph G
  private int[] CAM; // this is the internal array name for this data structure
  private int V; // V = CAM.length;
  public CompactAdjacencyMatrix(int _V) { CAM = new int[_V]; V = _V; } // constructor

  public Boolean EdgeExists(int i, int j) { // sample function
    // returns true if edge (i, j) exists or false otherwise, 0 <= i, j < V
    return (CAM[i] & (1 << j)) != 0;
  } // this is an O(1) function

  public void AddDirectedEdge(int i, int j) { // 2 marks
    // add unweighted edge i->j into G, it is guaranteed that 0 <= i, j < V
    // if edge(i, j) is already exists before, this function changes nothing




  } // to get the marks, your correct function should have time complexity of O(1)

  public void RemoveDirectedEdge(int i, int j) { // 2 marks
    // remove unweighted edge i->j from G, it is guaranteed that 0 <= i, j < V
    // if edge(i, j) does not exists before, this function changes nothing




  } // to get the marks, your correct function should have time complexity of O(1)
```

```
  public void RemoveAllOutgoingEdges(int i) { // 2 marks
    // remove all outgoing edges of vertex i, it is guaranteed that 0 <= i < V
    // if vertex i has no outgoing edge before, this function changes nothing




  } // to get the marks, your correct function should have time complexity of O(1)

  public int CountK(int i) { // 4 marks
    // count k, the number of neighbors of vertex i, it is guaranteed that 0 <= i < V
    int ans = 0;









    return ans;
  } // to get 3 marks, your correct function should have time complexity of O(V)
    // to get 4 marks, your correct function should have time complexity of O(k)
    // where k is the number of neighbors of vertex i
    // O(1) solution exists but this is very hard and excluded from the marking scheme

  public void Complement() { // 4 marks
    // update data structure to represents the complement G' of this graph G,
    // i.e. if edge(i, j) does not exists in G, it will exist in G', and vice versa.
    // The complement of Figure 3 is {8,9,0,7}
    // The complement of graph A in Section 3.1.2 is graph B in Section 3.1.2









  } // to get 3 marks, your correct function should have time complexity of O(V^2)
    // to get 4 marks, your correct function should have time complexity of O(V)

} // other lines that are not relevant for this question are hidden.
```

4 Applications (25 marks); Marks = _____ + _____ + _____ = _____

4.1 Robot Turtles

Warning: This is the last question in this final exam and also the hardest question.
You are encouraged to only attempt this question when you have completed all other questions.

4.1.1 Definition

Robot Turtles (http://www.robotturtles.com/) is a new board game for 'sneakily' teaching programming to young children. For this question, we deal with the **simplified form** of this board game.

Given an 8x8 board (the puzzle) that contains character:

- **'R'** – that means a robot turtle.
  There will only be one character **'R'** in the board and initially the robot turtle faces East.
- **'J'** – that means a jewel.
  There will only be one **'J'** in the board. The objective of the game is to make the robot turtle moves to the cell that contains a jewel. The robot turtle can be in **any orientation** when it reaches the cell that contains a jewel. The game is immediately **declared as complete** when this jewel cell is reached.
- **'.'** – that means a free cell (the robot turtle can freely move to this cell).
- **'#'** – that means an obstacle cell (the robot turtle cannot move to this cell, see below).

The player is given a set of movement cards:

- 'Move Forward' cards (there are **F** such cards at the beginning).
  This movement card instructs the robot turtle to move one cell forward according to it's orientation. That is, if the robot turtle is currently facing East/South/West/North, it will move rightwards/downwards/leftwards/upwards, respectively. However, if such 'Move Forward' movement will cause the robot turtle to exit the 8x8 board or hit an obstacle cell, the robot turtle will simply stays on its cell, a.k.a. a wasted move.
- 'Turn Left' Cards (there are **L** such cards at the beginning).
  This movement card rotates the robot turtle to the left, that is, if the robot turtle is currently facing East/South/West/North, it will now face North/East/South/West, respectively.
- 'Turn Right' Cards (there are **R** such cards at the beginning).
  This movement card rotates the robot turtle to the right, that is, if the robot turtle is currently facing East/South/West/North, it will now face South/West/North/East, respectively.

4.1.2 Application 1: Minimum Number of 'Move Forward' Card(s) Used

For the first application, your task is to determine the **minimum number of 'Move Forward' card(s)** that should be used by the player to move the robot turtle (remember that initially the robot turtle faces East) to the jewel cell. You can assume for this application 1 that the player can reuse 'Move Forward', 'Turn Left', and 'Turn Right' cards as required, i.e. the cards are unlimited.

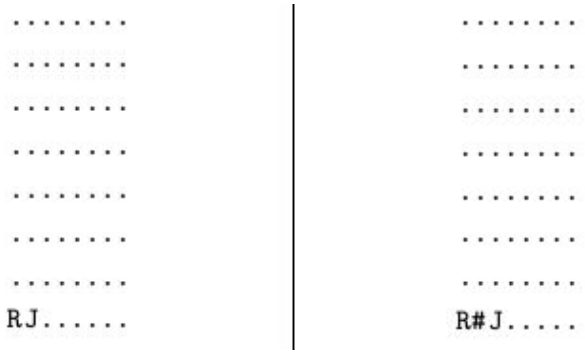


Figure 4: Left: Example 1, Right: Example 2

Example 1: If the 8x8 board is as shown in the left side of Figure 4, then the answer is **one** 'Move Forward' card only.

Example 2: If the 8x8 board is as shown in the right side of Figure 4, then the answer is **four** 'Move Forward' cards as the player can issue the following sequence of cards: 'Turn Left', **'Move Forward'**, 'Turn Right', **'Move Forward'**, **'Move Forward'**, 'Turn Right', and finally **'Move Forward'** to move the robot turtle to the jewel cell.

4.1.3 *Graph Modeling (7 marks); Marks = _____

1. What do the vertices and the edges of your Graph represent? (2 marks)

2. What is the upper bound of the number of vertices and edges in your Graph? (1 mark)
   $|V| =$
   $|E| =$

3. What is the Graph problem that you want to solve? (2 marks)

4. What is the best Graph algorithm to solve this problem and it's time complexity? (2 marks)

4.1.4 Application 2: Minimum Number of Card(s) Used

For the second application, your task is to determine the **minimum number of card(s)** that should be used by the player to move the robot turtle to the jewel cell. You can assume for this application 2 that the player can reuse 'Move Forward', 'Turn Left', and 'Turn Right' cards as required, i.e. the cards are unlimited.

Example 1: If the 8x8 board is as shown in the left side of Figure 4, then the answer is **one** card only: 'Move Forward'.

Example 2: If the 8x8 board is as shown in the right side of Figure 4, then the answer is **seven** cards as the player can issue the following sequence of cards: 'Turn Left', 'Move Forward', 'Turn Right', 'Move Forward', 'Move Forward', 'Turn Right', and finally 'Move Forward' to move the robot turtle to the jewel cell.

4.1.5 **Graph Modeling (8 marks); Marks = _____

1. What do the vertices and the edges of your Graph represent? (3 marks)

2. What is the upper bound of the number of vertices and edges in your Graph? (1 mark)

   $|V| =$
   $|E| =$

3. What is the Graph problem that you want to solve? (2 marks)

4. What is the best Graph algorithm to solve this problem and it's time complexity? (2 marks)

4.1.6 Application 3: How Many Ways to Complete the Puzzle?

For the third application, your task is to determine the **the number of ways the player can complete the puzzle?** This time, the initial number of 'Move Forward', 'Turn Left', and 'Turn Right' cards are limited to **F**, **L**, and **R**, respectively.

Example 1: If the 8x8 board is as shown in the left side of Figure 4 and the player has 1 'Move Forward' card, 1 'Turn Left' card, and 1 'Turn Right' card, then there are 3 ways:

1. Just issue a 'Move Forward' card.
2. 'Turn Left', 'Turn Right' (unnecessary moves, but valid), and finally 'Move Forward'.
3. 'Turn Right', 'Turn Left' (unnecessary moves, but valid), and finally 'Move Forward'.

Example 2: If the 8x8 board is as shown in the right side of Figure 4 and the player has 4 (FOUR) 'Move Forward' cards, 1 'Turn Left' card, and 2 'Turn Right' cards, then there are only 1 way:

1. 'Turn Left', 'Move Forward', 'Turn Right', 'Move Forward', 'Move Forward', 'Turn Right', and finally 'Move Forward'.

Example 3: If the 8x8 board is as shown in the right side of Figure 4 and the player has 3 (THREE) 'Move Forward' cards, 1 'Turn Left' card, and 2 'Turn Right' cards, then there is no (or 0) way to complete the puzzle as the player lacks 1 'Move Forward' card.

Example 4: If the 8x8 board is as shown in the right side of Figure 4 and the player has 5 (FIVE) 'Move Forward' cards, 1 'Turn Left' card, and 2 'Turn Right' cards, then there are 2 ways:

1. 'Turn Left', 'Move Forward', 'Turn Right', 'Move Forward', 'Move Forward', 'Turn Right', and finally 'Move Forward' (same as in Example 2).
2. 'Move Forward' (a wasted move), 'Turn Left', 'Move Forward', 'Turn Right', 'Move Forward', 'Move Forward', 'Turn Right', and finally 'Move Forward'.

4.1.7 ***Your Solution (10 marks); Marks = _____

For this part, you are not given any guiding sub-questions. You are free to use the space in this page and in the next page to write down your best solution and analyze it's time complexity.

You can continue your answer for Section 4.1.7 here.

– End of this Paper –