National University of Singapore

School of Computing

EXAMINATION FOR

Semester 1 AY2012/2013

# CS2010 - Data Structures and Algorithms II

Nov 2012, 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) questions with sub-questions.
   It comprises SIXTEEN (16) 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. When this final exam starts, **please immediately write your Matriculation Number here: _____ (do not forget the last letter and do not write your name).**
7. All the best :).

---

This portion is for examiner's use only

| Question | Maximum Marks | Student's Marks |
|---|---|---|
| 1 | 20 | |
| 2 | 15 | |
| 3 | 35 | |
| 4 | 30 | |
| Total | 100 | |

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

# 1 Basic Understanding of CS2010 Materials (20 marks)

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

1. List down **four** applications of Depth-First Search (DFS) algorithm **(4 marks)**:
   1). _____,
   2). _____,
   3). _____,
   4). _____.

2. Complete the *comparison* table below. The first entry is given as an example **(10 marks)**.

| | Similarities | Differences |
|---|---|---|
| Adjacency List vs Edge List | 1). Both are graph data structures<br>2). _____ | 1). $O(V+E)$ space for Adj List<br>$O(E)$ space for Edge List<br>2). _____ |
| Depth-First Search (DFS) vs Breadth-First Search (BFS) | 1). _____<br>2). _____ | 1). _____<br>2). _____ |
| Floyd Warshall's vs Bellman Ford's | 1). _____<br>2). _____ | 1). _____<br>2). _____ |

3. The three questions below are based on the DAG shown in Figure 1 (**3 × 2 = 6 marks**). The source vertex $s$ is vertex 0 and the target vertex $t$ is vertex 7 .
   1). The length of the shortest path between $s$ and $t$ is _____.
   2). The length of the longest path between $s$ and $t$ is _____.
   3). The number of paths between $s$ and $t$ is _____.

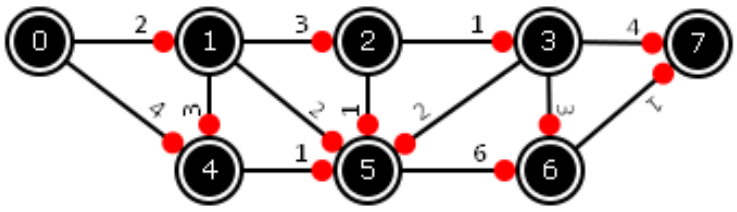



Figure 1: DAG for Section 1 Question 3

## 2 Analysis (15 marks)

Prove (the statement is correct) or disprove (the statement is wrong) 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 number of *odd-degree vertices* in a simple undirected graph must be *even*.

2. Running a bipartite graph checking algorithm as discussed in class (either BFS or DFS version) on a *tree* will always yield true. That is, a tree is always a bipartite graph.

3. There is no case where Prim's algorithm fails to find the Minimum Spanning Tree of a certain input graph and Kruskal's algorithm runs correctly (find the MST) on that same input graph.

4. Finding the *longest* simple path from a source vertex to a target vertex in a *weighted tree* is an easy problem that has solution with polynomial time complexity.

5. The runtime of Floyd Warshall's algorithm is *slower* if the input graph is a *complete graph* (where $E = O(V^2)$) than if the input graph is a *sparse graph* (where $E$ is much less than $V^2$).

# 3 Learn on the Spot (35 marks)

## 3.1 Eulerian Path (18 marks)

### 3.1.1 Definition

In graph theory, an *Eulerian path* is a potentially non-simple path in a graph which visits every edge *exactly once*. This Eulerian path starts and ends at *different* vertex. An undirected graph has an Eulerian path **if and only if** exactly **two vertices have odd degree, and if all of its vertices with nonzero degree belong to a single connected component**. For example, the graph in Figure 2 has an Eulerian path, e.g. $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 4 \rightarrow 5$. Note: Eulerian path is not unique. Path: $0 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow 5$ is also a valid Eulerian path.

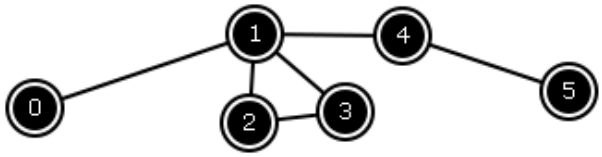
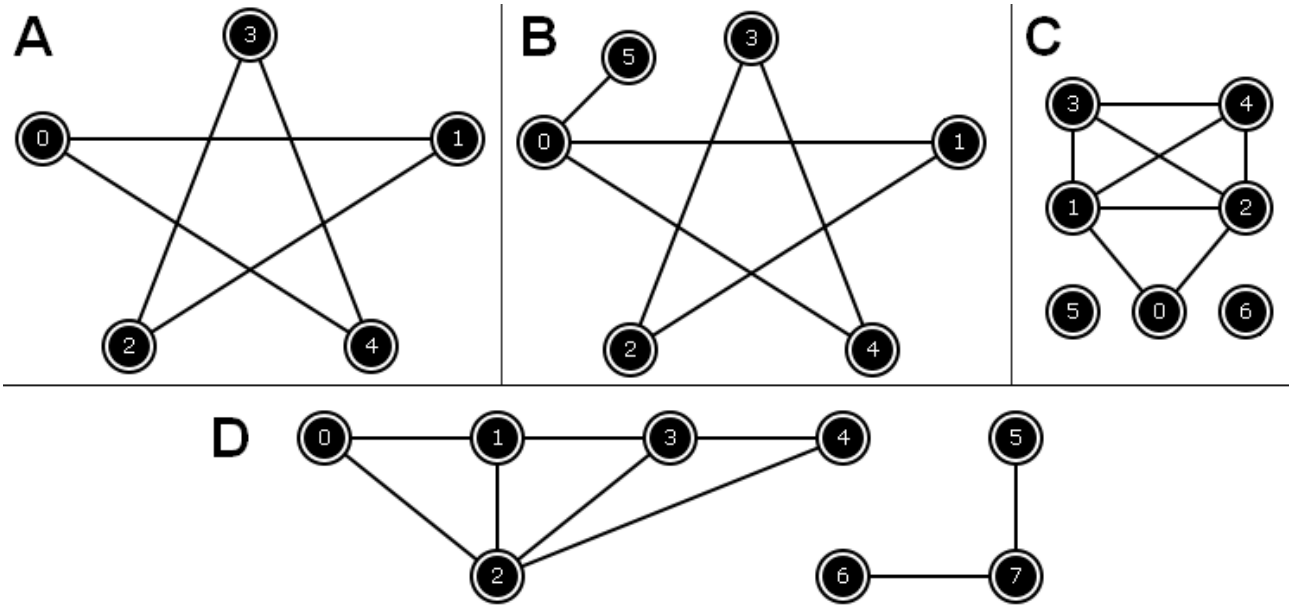



Figure 2: This Graph has an Eulerian Path from Vertex 0 to Vertex 5

### 3.1.2 Manual Checks (6 marks)

For each of the four undirected graphs below, please determine if the graph has an Eulerian Path. If a graph has an Eulerian path, show one such Eulerian path by listing the vertices in the path.




Answer for Graph A: _____.
Answer for Graph B: _____.
Answer for Graph C: _____.
Answer for Graph D: _____.

### 3.1.3 HasEulerianPath (12 marks)

Please implement the following function `HasEulerianPath` that takes in an undirected graph stored in an Adjacency List and returns true if the undirected graph has an Eulerian path, or return false otherwise. Finally, analyze what is the time complexity of your implementation.

The format of the Adjacency List is as discussed in class and replicated here for clarity: An Adjacency List is a Vector of $V$ lists, one for each vertex. List $i$ contains another Vector of IntegerPairs where we store list of IntegerPair information: (the vertex ID, the corresponding edge weight) of neighbors of $i$. Note that bidirectional edge $(i, j)$ of an undirected graph is stored *twice* in this format, i.e. edge $(i \rightarrow j)$ is stored in list $i$ whereas the other edge $(j \rightarrow i)$ is stored in list $j$.

```
Boolean HasEulerianPath(Vector < Vector < IntegerPair > > AdjList) {
```

```
} // The time complexity of this solution is O(__________________)
```

## 3.2 Strongly Connected Directed Graph (17 marks)

### 3.2.1 Definition

A directed graph is called *strongly connected* **if there is a path from each vertex in the graph to every other vertex**. For example, directed graph A in Figure 3, left is **strongly connected**, whereas directed graph B in Figure 3, right is **not strongly connected** because vertex 1 cannot visit vertex 0 and vertex 2 cannot visit the two other vertices. Note that the arrow tip of a directed edge is a circle in Figure 3.

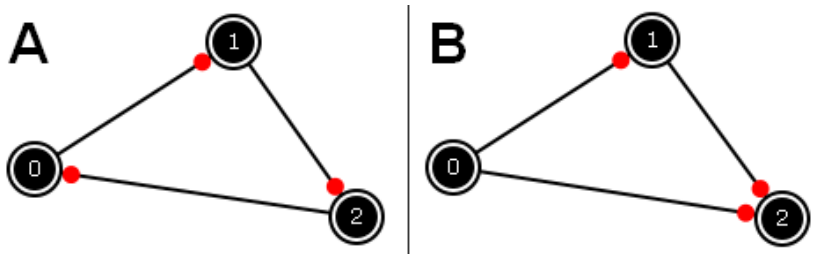
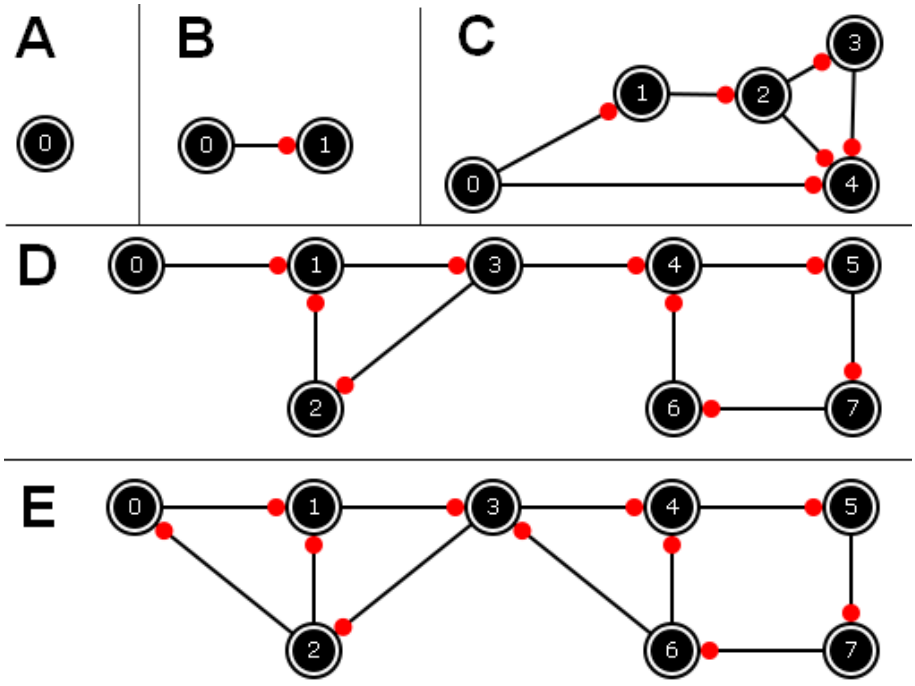



Figure 3: Graph A (left) is Strongly Connected; Graph B (right) is not Strongly Connected

### 3.2.2 Manual Checks (5 marks)

For each of the five directed graphs below, please determine if the graph is strongly connected. If the directed graph is *not* strongly connected, explain the reason briefly.




Answer for Graph A: _____.
Answer for Graph B: _____.
Answer for Graph C: _____.
Answer for Graph D: _____.
Answer for Graph E: _____.

### 3.2.3 IsStronglyConnected (12 marks)

Please implement the following function `IsStronglyConnected` that takes in an unweighted directed graph stored in an Adjacency Matrix and returns true if the directed graph is strongly connected, or return false otherwise. Finally, analyze what is the time complexity of your implementation.

The format of the Adjacency Matrix is as discussed in class and replicated here for clarity: An Adjacency Matrix $M$ is a 2D array of integers. If there is an edge between vertex $i$ and vertex $j$ in the directed graph, cell $M[i][j] = 1$; Otherwise, cell $M[i][j] = 0$.

Note: The concept of Strongly Connected Component and its efficient $O(V + E)$ solution is not taught in CS2010 syllabus. However, there are other algorithms that are already taught in class that can be used to detect if a given directed graph is strongly connected or not. You will get up to 7 marks for an $O(V^3)$ solution or up to 12 marks for an $O(V^2 + VE)$ solution. There is no bonus marks if you can give the more sophisticated $O(V + E)$ solution.

```
Boolean IsStronglyConnected(int[][] AdjMatrix) {




} // The time complexity of this solution is O(__________________)
```

## 4 Two Snakes (30 marks)

'Two Snakes' is a new *two players collaborative game* on an $R \times C$ matrix $M$. Each cell in the matrix $M$ contains a *non-negative* integer less than 1000.

Both players choose *different* columns on the first row (row 0) as their starting point. On the first row, they are allowed to move leftwards or rightwards before they proceed to the second row (row 1).

Then, both players make their way down to the last row (row $R-1$) by moving to adjacent cells (move left, right, or down) step by step. The paths of these two players *cannot overlap*. They also cannot revisit a cell that has been visited before. Remember that since there is no 'move up' step, that means once they move down to the next row, they will *not* be able to move back up to an unvisited cell on the previous row.

Finally, once they reach the last row (row $R-1$), they are allowed to move leftwards or rightwards before they end the game.

The *collaborative score* for both players is simply *the sum of the integers of the cells they have visited in their two non-overlapping paths*. Your task is to write a program to compute the *minimum collaborative score* that two players can achieve by cooperating together.

Example 1: On a $5 \times 5$ matrix $M1$ below (see Figure 4, left), the best path for player 1 is to start from column 2 of row 0 and then take this path: $(0,2) \rightarrow (1,2) \rightarrow (1,1) \rightarrow (1,0) \rightarrow (2,0) \rightarrow (3,0) \rightarrow (4,0)$ and the best path for player 2 is to start from column 4 of row 0 and then take this path: $(0,4) \rightarrow (1,4) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (2,2) \rightarrow (3,2) \rightarrow (4,2)$. These two paths are *non-overlapping* and altogether, both players visited thirteen cells that contain integer 1 and one cell that contains integer 4. Therefore their total collaborative score is $13 \times 1 + 4 = 17$ and this is the minimum possible collaborative score on this matrix. We can interchange the paths of the two players and the result is still the same. However, it may be simpler to say that player 1 is the 'left side' player and player 2 is the 'right side' player.

| M1 | column 0 | column 1 | column 2 | column 3 | column 4 |
|---|---|---|---|---|---|
| row 0 | 5 | 5 | 1 | 5 | 1 |
| row 1 | 1 | 1 | 1 | 1 | 1 |
| row 2 | 1 | 5 | 1 | 1 | 5 |
| row 3 | 1 | 5 | 1 | 5 | 1 |
| row 4 | 1 | 5 | 4 | 5 | 5 |

| M2 | column 0 | column 1 | column 2 |
|---|---|---|---|
| row 0 | 1 | 7 | 1 |
| row 1 | 1 | 2 | 1 |
| row 2 | 9 | 1 | 1 |
| row 3 | 8 | 2 | 4 |

Figure 4: Two Snakes: Example 1: $M1$ (Left) and Example 2: $M2$ (Right)

Example 2: On a $4 \times 3$ matrix $M2$ above (see Figure 4, right), the best path for player 1 is to start from column 0 of row 0 and then take this path: $(0,0) \rightarrow (1,0) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (3,1)$ and the best path for player 2 is to start from column 2 of row 0 and then take this path: $(0,2) \rightarrow (1,2) \rightarrow (2,2) \rightarrow (3,2)$. These two paths are *non-overlapping* and altogether, both player obtained a collaborative score of $7 + 7 = 14$, and this is the minimum possible collaborative score on this matrix.

### 4.1 Subtask 1 (12 marks)

Forget that there are two players in the original problem description. Suppose that there is only **one player** in this game. Determine the minimum *(individual)* score that this *single* player can achieve. All other rules are the same. For the same two examples earlier, the best individual paths are shown below. The minimum individual score for Example 1: $M1$ is 7. The minimum individual score for Example 2: $M2$ is 6. Constraints: $1 \leq R, C \leq 1000$. Analyze the time complexity of your solution!

| M1 | column 0 | column 1 | column 2 | column 3 | column 4 |
|---|---|---|---|---|---|
| row 0 | 5 | 5 | 1 | 5 | 1 |
| row 1 | 1 | 1 | 1 | 1 | 1 |
| row 2 | 1 | 5 | 1 | 1 | 5 |
| row 3 | 1 | 5 | 1 | 5 | 1 |
| row 4 | 1 | 5 | 4 | 5 | 5 |

| M2 | column 0 | column 1 | column 2 |
|---|---|---|---|
| row 0 | 1 | 7 | 1 |
| row 1 | 1 | 2 | 1 |
| row 2 | 9 | 1 | 1 |
| row 3 | 8 | 2 | 4 |

Figure 5: One Snake: Example 1: $M1$ (Left) and Example 2: $M2$ (Right)

The space in this Page 10 is given for you to *sketch* your solution ideas. You can leave this page blank. In case your solution in Page 11 is incorrect, we will look at this Page 10 to give partial marks.

```
int OneSnake(int R, int C, int[][] M) {

} // The time complexity of this solution is O(__________________)
```

### 4.2 Prelude to Subtask 2-3-4, Manual Checks (3 marks)

To verify whether you have understood the problem, please solve the 'Two Snakes' problem on these five test cases shown in Figure 6. For each test case, you need to indicate the two non-overlapping paths taken by both players directly in Figure 6 (see examples on drawing such non-overlapping paths in Figure 4) and state the corresponding minimum collaborative score.

| M3 | column 0 | column 1 | column 2 |
|---|---|---|---|
| row 0 | 1 | 7 | 2 |

score = ___

| M4 | column 0 | column 1 | column 2 |
|---|---|---|---|
| row 0 | 1 | 7 | 2 |
| row 1 | 1 | 7 | 2 |

score = ___

| M5 | column 0 | column 1 | column 2 |
|---|---|---|---|
| row 0 | 1 | 9 | 1 |
| row 1 | 1 | 1 | 1 |
| row 2 | 1 | 8 | 9 |
| row 3 | 1 | 8 | 9 |

score = ___

| M6 | column 0 | column 1 | column 2 | column 3 | column 4 |
|---|---|---|---|---|---|
| row 0 | 0 | 0 | 9 | 0 | 0 |
| row 1 | 9 | 0 | 9 | 0 | 9 |
| row 2 | 0 | 0 | 9 | 0 | 0 |
| row 3 | 0 | 9 | 9 | 9 | 0 |
| row 4 | 0 | 0 | 0 | 0 | 0 |

score = ___

| M7 | column 0 | column 1 | column 2 | column 3 | column 4 |
|---|---|---|---|---|---|
| row 0 | 0 | 9 | 9 | 9 | 9 |
| row 1 | 0 | 9 | 9 | 9 | 9 |
| row 2 | 0 | 0 | 0 | 0 | 0 |
| row 3 | 9 | 9 | 9 | 9 | 0 |
| row 4 | 9 | 9 | 9 | 9 | 0 |
| row 5 | 9 | 9 | 9 | 9 | 0 |

score = ___

Figure 6: Solve These Five Test Cases, score = max(0, 3 - number of mistakes)

### 4.3 Subtask 2 (1 mark)

What is the solution for 'Two Snakes' problem if $R = 1; 2 \leq C \leq 30$?
That is, $R$ is always 1, e.g. $M3$ in Figure 6.

### 4.4 Subtask 3 (2 marks)

What is the solution for 'Two Snakes' problem if $1 \leq R \leq 2; 2 \leq C \leq 30$?
That is, $R$ can be up to 2, e.g. $M4$ in Figure 6.

### 4.5 Subtask 4 (12 marks)

Solve the full 'Two Snakes' problem and analyze the time complexity of your solution. There are different possible solutions for this classic but rarely used problem. You can get full 12 marks by only using algorithms taught in class. There are better but more advanced algorithms but we will not give bonus marks for such solution. Your solution will be awarded marks based on the criteria below:

| Max Marks | Requirements |
|---|---|
| 1 | Incomplete solution |
| 4 | An $O(R \times C^5)$ solution; OK for $1 \leq R \leq 100; 2 \leq C \leq 15$ |
| 7 | An $O(R \times C^4)$ solution; OK for $1 \leq R \leq 100; 2 \leq C \leq 30$ |
| 12 | An $O(k \times R \times C^2)$ solution where $k$ is a constant factor; OK for $2 \leq R, C \leq 200$ |
| 12 | An $O$(RC log RC) solution ~~where $k_1$ is a constant factor~~; OK for $2 \leq R, C \leq 1000$ |
| 12 | An $O(k_2 \times R \times C)$ solution ~~where $k_2 < k_1$~~; OK for $2 \leq R, C \leq 5000$ |

The space in this Page 13 is given for you to *sketch* your solution ideas. You can leave this page blank. In case your solution in Page 14-15 is incorrect, we will look at this Page 13 to give partial marks.

Major hint for **4 marks solution** (there are better solutions; you can ignore this hint if you have found a better one): We can model this problem as a DAG where the vertex is a triple $(r, c1, c2)$, $0 \leq r < R$, and $0 \leq c1 < c2 < C$. This vertex/triple represents a state where player 1 (left side) is currently in $(r, c1)$ and player 2 (right side) is currently in $(r, c2)$ and both players will go down to the next row $r + 1$ from column $c1$ and $c2$, respectively. That is, we add *one more parameter* to a standard vertex in a matrix, i.e. pair $(r, c1)$, by one more parameter: $c2$, to model *two players* of this problem. Now, state $(r, c1, c2)$ has a weighted edge to state $(r + 1, c1', c2')$ iff $c1' \in [0 \ldots c2\text{-}1]$ and $c2' \in [max(c1, c1') + 1 \ldots C\text{-}1]$. This way, the path of player 1 and player 2 will never overlap and acyclic. The computation of the edge weight and all other details are left to the students.

```
int TwoSnakes(int R, int C, int[][] M) {

} // The time complexity of this solution is O(__________________)
```

```
// You can use this page to continue your solution.
```

Credits for 'Two Snakes': Koh Zi Chun and Shen Chuanqi.
The harder version of this problem first appears in Singapore International Olympiad in Informatics (IOI) Selection Contest 2012.

– End of this Paper –