**Faculty of Business and Information Technology**

**INFR 2820U: Algorithms and Data Structures: Winter 2019**

**March 6, 2019 (75 minutes)**

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Student number: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Please read the following instructions carefully:

1. **Write your answers on the question sheet and submit it at the end of the exam. No other submissions will be accepted.**
2. Pay attention to your handwriting. **If I cannot read it, I cannot mark it.**
3. This exam consists of a total of 8 pages (including this page). Please make sure that you have all the pages.
4. You are allowed one 2-side, A4, hand-written cheat sheet.
5. Communication with other people is not allowed during the exam. You may only communicate with the instructor, the teaching assistants, or the invigilators.

Academic Integrity is a central value of this university.  Please read and sign the following statement:

*I am the sole author of all the work and solutions for this exam, as submitted in my name in this paper, and I have followed the instructions stated in this page.*

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Good Luck*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Exercise I** | **Exercise II** | **Exercise III** | **Exercise IV** | **Exercise V** | **Total (25)** |
|  |  |  |  |  |  |

1. Given the following graph G1 in Figure 1:

9

12

2

7

1

10

1

3

2

2

1

3

8

2

1

2

2

1

**Figure 1. G1**

1. **(2 points)** Give the depth-first traversal of the graph, starting from vertex C. ( in case you have the choice between multiple vertices to visit, always select the vertex that has smaller lexicographic order first, i.e. in alphabetical order)

**Answer:**

C->B->D->H->L->A->J->i->K->E->M->F->G

1. **(2 points)** Give the breadth-first traversal of the graph, starting from vertex L. ( in case you have the choice between multiple vertices to visit, always select the vertex that has smaller lexicographic order first, i.e. in alphabetical order)

**Answer:**

L->A->B->F->G->H->J->C->D->E->I->K->M

1. **(5 points)** Find the shortest path between **node L** and all other nodes in the graph, using Dijkstra's algorithm. List all nodes in order on the path and the cost of the path in the following table.

**Answer:**

|  |  |  |
| --- | --- | --- |
| **Destination Node** | **Path** | **Cost** |
| **L->A** | **L,A** | **1** |
| **L->B** | **L,B** | **2** |
| **L->C** | **L,B,E,K,I,C** | **11** |
| **L->D** | **L,H,D** | **2** |
| **L->E** | **L,B,E** | **5** |
| **L->F** | **L,F** | **7** |
| **L->G** | **L,G** | **1** |
| **L->H** | **L,H** | **1** |
| **L->I** | **L,B,E,K,I** | **8** |
| **L->J** | **L,A,J** | **3** |
| **L->K** | **L,B,E,K** | **6** |
| **L->M** | **L,B,E,K,M** | **8** |

1. **(3 points)** Given the following graph G2 in Figure 2:

5

9

12

6

7

15

10

1

3

4

2

14

40

8

20

130

30

60

**Figure 2. G2**

Using the Kruskal’s algorithm, find the minimum spanning tree for graph G2 and its cost. Only show the final result.

**Answer:**

9

12

6

7

15

10

1

3

4

2

14

40

8

20

130

30

60

5

Total Cost = 159

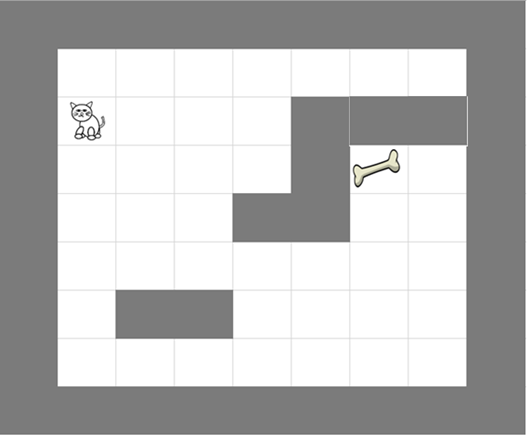
1. **(3 points)** Apply topological sorting (ordering) on the following Directed Acyclic Graph (DAG) G3 of tasks to create a linear ordering of these tasks. Fill your answer in the *TopologicalOrder* table below. At any point of time, if you have the choice between multiple tasks to add to the execution (processing) queue, select first the one that has the lowest lexicographic order (Similar to what we did in the class).

**Figure 3. G3**

**Answer:**

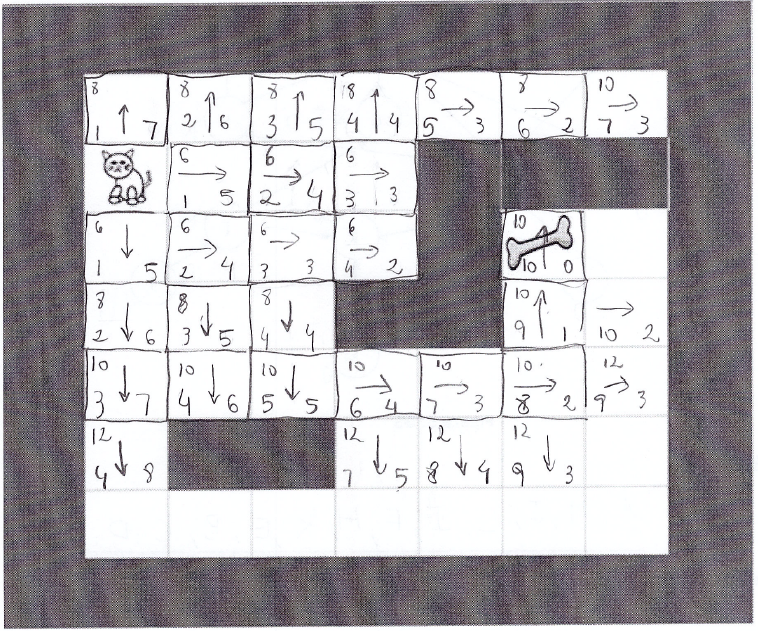
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *TopologicalOrder* | A | M | J | L | I | F | H | K | E | B | C | D | G |

1. **(5 points)** Use the same approach from Assignment 3 and as illustrated in the class to show how the cat Moody can use the A\* algorithm to get to the bone in graph G4. Use also the same movement cost and heuristic computation approach from Assignment 3. (i.e. Moody can move only up, down, left, or right. Moody can use the Manhattan distance to compute its heuristic distance to the bone). In case you have multiple squares on the open list with equal scores (F), use the one that was most recently added to the open list. Show all your work and write your answers on the same graph.



**Figure 4. G4**

Answer



1. **(5 points)** Write the pseudo code for an algorithm *Rearrrange(Q)* that takes as argument a queue Q of integers, and then rearrange the numbers so that the negative numbers are at the front of the Q, followed by all the zeros, followed by all the positive numbers. The order of negative numbers should be the same as in the original queue, while the order of positive number should be the reverse as in the original queue. The only basic operations that you can use on the queue are the following: *addQueue*(), *deleteQueue*(), *Qsize*(), *Rear()* and *Front*(). You can use **only one** additional structure ( stack or queue). If you decided to use a stack, the only operations available to use on the stack are: *push()*, *pop()*, *top()*, and *isEmptyStack()*.

Example:

If Q = [ 3, 7, -1, 5, -2, 10, -12, 0, 4, 0, 4, -1, 0, 15].

After calling *Rearrrange(Q)*, Q will look like

Q = [ -1, -2, -12, -1, 0, 0, 0, 15, 4, 4, 10, 5, 7, 3].

Assuming that each of the following operations has a complexity of O(1) (*addQueue*(), *deleteQueue*(), *Rear(), Front*(), *push()*, *pop()*, and *top()*), what is the complexity of your algorithm as a function of the size of the queue. Justify your answer. An answer without proper justification does not count.

**Solution**

Count0 = 0 // to count the number of zeros in the Q

// Take each element of the Q

For i = 1 to Q.size()

Item = Q.front(); // O(1)

Q.dequeue(); // O(1)

If Item < 0 // put the item back in the Q

Q.enqueue(item); // O(1)

If item = 0 // add 1 to the count0

Count0 = count0 + 1;

If Item > 0 // add it to the stack

S.push (Item); // O(1)

// put enough zeros in the Q now

For I = 1 to count0

Q.enqueue(0);

// take items back from the stack, and put them in reverse order in the Q

For i = 1 to S.size()

Item = S.top(); // O(1)

S.pop(); // O(1)

Q.enqueue(Item); // O(1)

**Complexity**

**Assuming you have n elements in the Q, then the first for loop, repeat n times, and each iteration has a complexity of O( 1+1+1+1). So the total complexity for the for loop is O(4n) = O(n)**

**The second for loop has no stack or Q operation, so we can assume its complexity is O(1)**

**Worst case scenario, all elements end in the stack,so for the third for loop, which repeat n times, and each iteration has a complexity of O( 1+1+1). So the total complexity is O(3n) = O(n)**

**So the total complexity for the the for loops is O(n+1+n) = O(n)**

This Page was left empty intentionally