**Data structures 2 final project**

**Students: IDs:**

Bishoy refaat riad 5804

Islam tarek fahmy 5785

Adham Khaled 5422

Karim elsayad 6023

Table of Contents

[1) Detailed Description of each program. 2](#_Toc41761918)

[A. Part1 : Linked in 2](#_Toc41761919)

[B. Part2 : maze 2](#_Toc41761920)

[C. part3 :shortest path 2](#_Toc41761921)

[2) Description of the Algorithm including data structures used and running time. 3](#_Toc41761922)

[A. Bfs graph traversal : 3](#_Toc41761923)

[B. Dijkstra : 3](#_Toc41761924)

[C. Priority queues : 3](#_Toc41761925)

[3) Sample Runs of your program. 3](#_Toc41761926)

[4) Division of labor among team members. 4](#_Toc41761927)

1. Detailed Description of each program.
2. Part1 : Linked in

We simply take the nodes represent it with linked lists

we bfs From the source saving in the queue both the node number

and the remaning distances then we save it in an array inversely in which

the index represents the remaning distances so that arr[1] stores the nodes

with distance k from source

1. Part2 : maze

-start by taking input from user , the graph representation used is a matrix

-we use bfs (as its guaranteed to give us the shortest path) to solve the maze. We visit the cells in all 4 directions

-visited cells are marked with -1\*(i+1) to trace the route while printing

if we move up we put value -1 , left put -2 ,down put -3 ,right put -4 in the cell

-go to the “isSafe” method to check if they are available

“isSafe” is a method to determine if the next step is possible by checking the bounds and confirm that the cell value is 0

-we then enqueue them , and repeat

-at the start of every loop check if we reached the end of the maze

-when done we trace back the moves we marked with -1\*(i+1) and print the path

1. part3 :shortest path

-we start by creating the graph we use arraylists for representation and class “Route”

class “Route” contains 3 int parameters point, weight (contains total cost), time

Note : While filling the weight parameter it will be filled as M\*(1+time)+cost to account for the price per hour , this will lead to an extra M as there is no waiting for 1 hour in the last stop , this is fixed by subtracting M and using math.max with 0 when printing the final value.

-we then fill the graph with user input and move to a method called “Dijkstra” implementing the djkstra algorithm

-we created 3 arrays

“d[ ]” contains least cost relative to the start

“t[ ]” contain the time for the shortest path for each point relative to the start

“shortest[ ]” we need to trace the path taken so we use this array to conatin the previous point for each shortest path. for example in the case provided with the assigment d[4]=3 , and d[3]=1

-we initialize the values for “d[ ]” with -1 , we use this instead of max.integer to account for edge cases ,and since Dijkstra doesn’t deal with negative values we added a condition to account for the -1 while comparing.

-we then use a priority queue and class “RouteComparator” to have weight as the value compared while filling the priority queue

-contuine with a normal djkstra algorthim , further explained in 2) B

-when done check d[end] as it should contain the shortest distance , and proceed to printing the path using method “printPathRec” which checks the value of shortest[end] and move recursively to print each point in the shortest path.

-if d[end]=-1 that means the values was never updated meaning there is no path between start and end , and the code will print so

1. Description of the Algorithm including data structures used and running time.

1. Linked in :-

Data structures used : adjacency list for graph representation and queue for BFS and class where we stored the distance from original node and the node number   
 Algorithms used : BFS

Run Time : O(n)

2.

1. Sample Runs of your program.
2. Part1
3. Part2
4. Part3
5. Division of labor among team members.

Bishoy : Supervised the whole project and was responsible for part 1 implementation and testing

Islam : implemented and tested part 2 was

Adham : implemented part 3 and generated hard test cases and edge cases for both part 2 and 1

Karim : implemented part 3 and generated hard test cases and edge cases for part 3, 2 and 1