Task-2: Assignment Report

COP290- Design Practices

Indian Institute of Technology Delhi

Name: Deepanshu **Entry Number: 2019CS50427**
Name: Prashant Mishra **Entry Number: 2019CS50506**

1 Introduction

A droid has to reach from source to destination while collecting all the stones on its route.

2 Problem

We can simulate the given problem as a computer science problem using graph data structure. Each node will represent a cell (including barriers) in the maze and edges connecting neighbouring cells.
A neighbouring cell pair is such that we can reach from one cell to another in one move.

3 Basic solution

One solution is to traverse whole of the graph and collect the stones. But that would be a very long route where a lot of times goes into the paths that are redundant. The time complexity of this solution would be O(N) where N is the number of vertices and space complexity will be O(1).
This solution is helpful when the number of stones is close to the total number of nodes. We would not waste time and space avoiding small number of blank nodes.

4 One solution without TSP

One of the solutions is for the case when number of stones ¡¡ total number of nodes.
For optimised time, we will be using BFS to look for stones. We will then find the path between stones, and take the cummulative summation to find the optimised path.

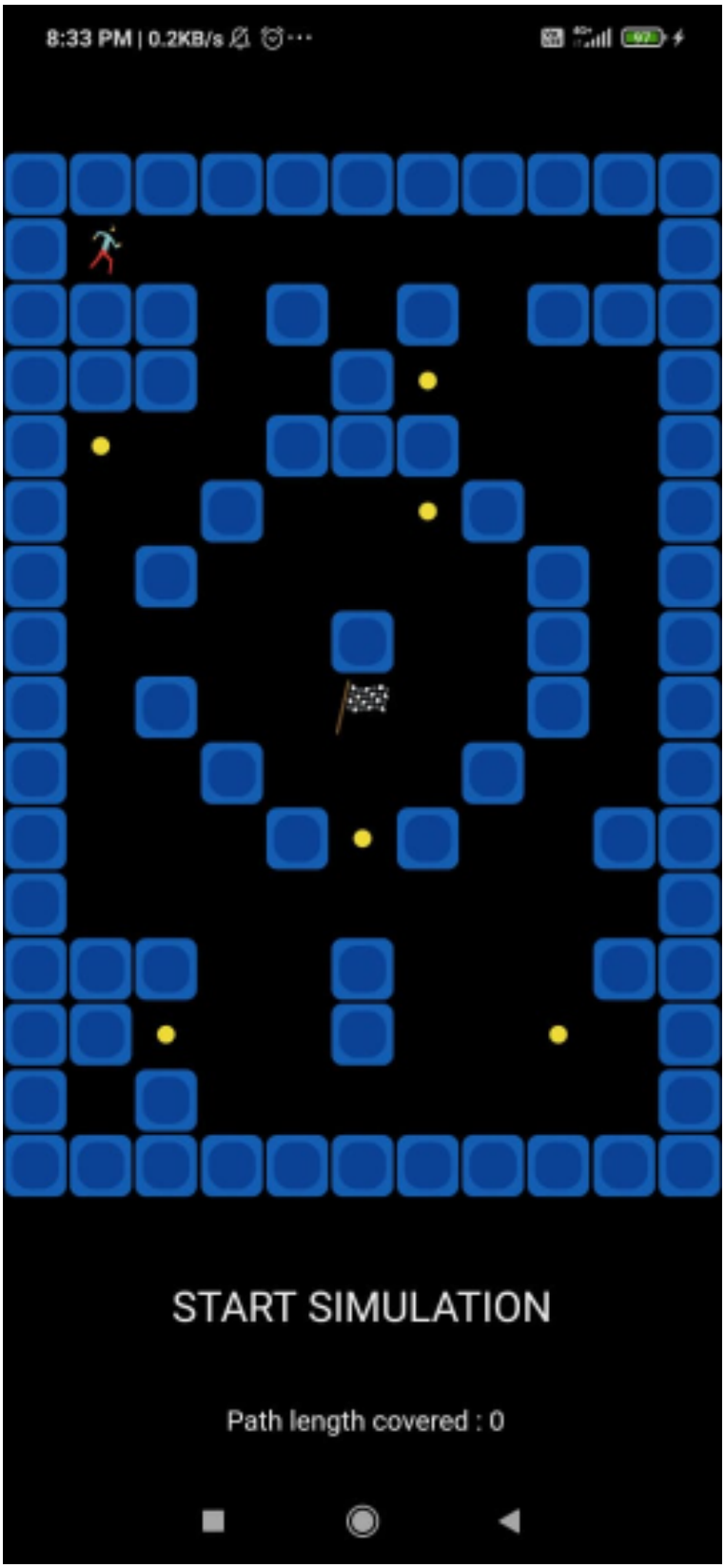

Figure 1: Three Stage Model of Memory

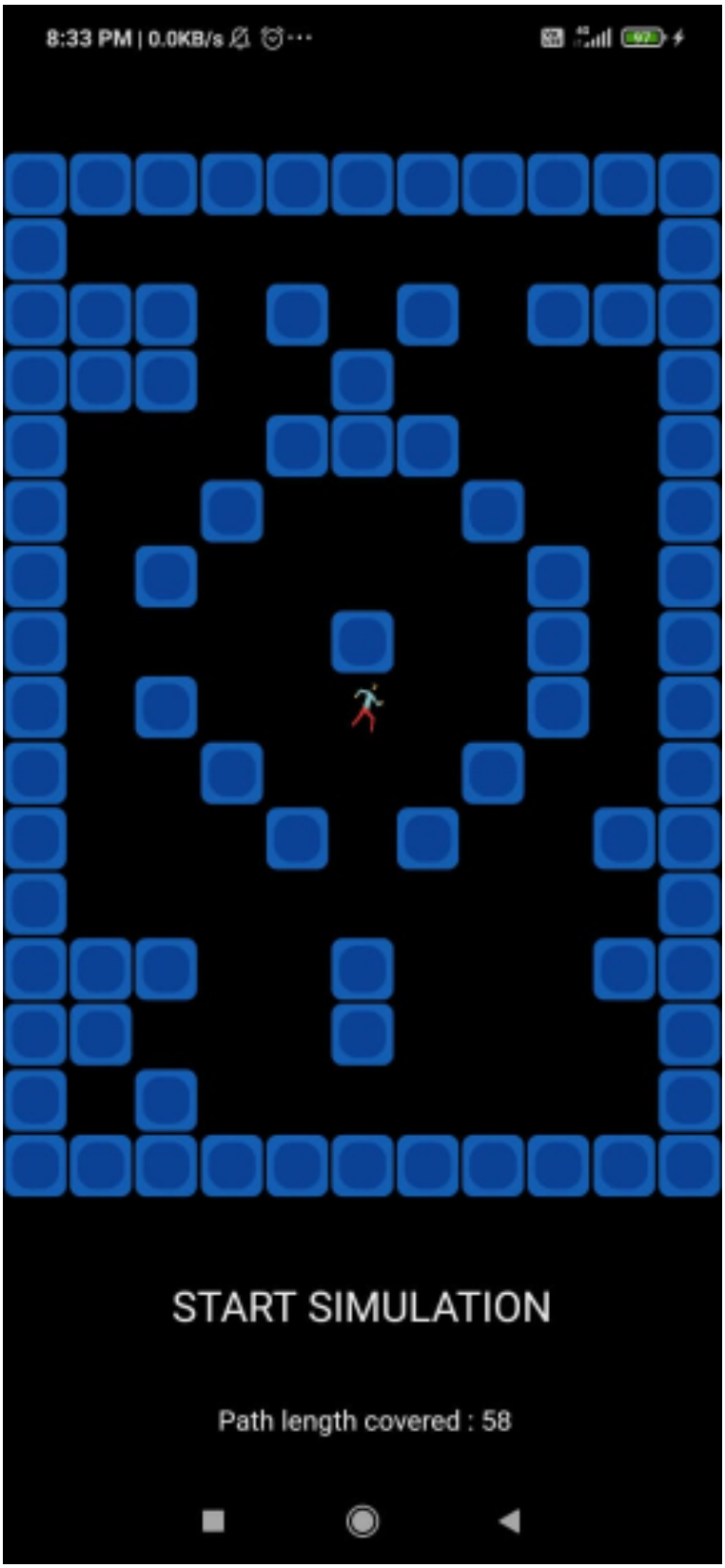

Figure 2: Three Stage Model of Memory

5 Solution Analysis

For the analysis, we will use N for number of rows, M for number of columns, K for number of stones.
Following is the detailed analysis of the algorithm

5.1 Function wise analysis

The description and time complexity of various functions is as follows:

1. **Search**
   Returns the index which contains the shortest path for the given pair.
   Time complexity: O(1) & $O(K^2)$ if K is large

2. **list_append**
   Appends one path in the list of paths which would be further used to take the minimum path.
   Time complexity: O(M*N)

3. **possible_permutation**
   Does the calculation of total distance for each permutation.
   Time complexity: O(1) & $O(K^3)$ if K is large

4. **heapPermutation**
   Generates permutation using Heap Algorithm.
   Time complexity: O(1)

5. **Find_Smallest_Path_from_src_dest**
   Finds the smallest path for a given source and destination.
   Time complexity: O(1) & O(K!) if K is large

6. **isValid**
   Returns true if the cell is within the boundary ranges and is not visited, else returns false.
   Time complexity: O(1)

7. **BFS**
   Carries out Bredth First Search for a given source and destination.
   Time complexity: O(MN)

8. **main**
   Initialises the variables and calls the functions from here. Deciding function is BFS as it has highest complexity.
   Time complexity: O(MN)

The total space used is O(MN) as we have to store the paths which can be of size MN in the worst case.
The time complexity of this algorithm will be same as that main function. The time taken by the main function is of the same order as of General BFS i.e size of the graph , hence in this case, the size of maze i.e O(MN).

5.2 Correctness of the algorithm

Following is the correctness logic for BFS:

1. Let us denote source as s, destination as d and array of stones as s1,s2,..,s(k) where k is number of stones. The general path will look like :-¿
   General path -¿ s—-s(i)—s(j)—d
   i.e we have a path with starting vertex as "s" and ending vertex as "d" and in between we have some permutation of stones vertices say s(i) where $1 \leq i \leq k$.

2. So, we can decompose our general path as follows :
   General-Path: -¿ s–(s1) + (s1)—(s2)+....+ s(k)–d

3. Thus, we have divided our original path into (k+1) different paths.

4. Now to make our general path to shortest path we have to minimize the length/size of each (k+1) paths, so that size of overall path can be minimized, so that we can obtain shortest path.

5. Now at the first step we calculated the shortest path between each pair from the set of vertices s, s1,..,s(i)...,s(k),..,d by applying **BFS** on the grid by taking one element of pair as the source and other as the destination. This is done by using the function **BFS**.

6. Then, we apply brute force method for calculating the minimum path length formed by path s..s(i)...d or we can say.
   s+**permutation_of**s1,s2,..,s(i)..,s(k-1),s(k)+d.

7. So by obtaining the permutation of stones vertices i.e s1,..s(k) and taking the sum of decomposed path-length which we already have from step-(5) (i.e., the path-length of each pair).

8. We can calculate the path length of the general path using it. Hence we,get the minimum path length possible. This process takes (k)! iteration to perform and is done in program using function :

   - heapPermutation
   - possible_permutation
   - Find_Smallest_Path_from_src_dest

9. Now after getting minimum path length from source (s) to destination(d) which includes visiting all cells where stones are present in the maze. We can combine all the decompose path lengths of each pair to obtain the final Shortest Path as per the condition.

   This is performed in the main function using the list_append function which is used to join each piece wise path to get final Shortest Path.

Hence this method gives us the shortest path as per Problem Statement.

Note: We have assumed that number of stones is not significant as compared to total grid size. Otherwise if number of stones will becomes large and the order of algorithm would change.

6 Simulation

We made the simulation in the app itself. (GIF attached and APK provided in the repository). Following were the major design decisions that we took for the simulation part:

1. We stored the output from the algorithm in the list.
2. Now we generate the grid screen as done in part-1.
3. We will place the player at source, stones and barriers at their position and flag at the destination.
4. We will now increment the player position according to the output path provided.
5. Additionally, we will wait for a while at a infinity stone and play some sound to indicate that the stone has been picked.
6. After complete traversal of the path, the player will be at destination and the simulation will terminate showing relevant statistics.

7 Solution with TSP

TSP is another algorithm for carrying out a similar problem.
It is for the case when source and destination are at same location. Most of the implementation is similar to previous algorithm. The other funtion that is used are:

We have assumed that number of stones is constant , otherwise if number of stones will becomes variable then order of Algorithm would not be O(MN).
Note Note , in this program as we have assumed number of stones to be constant , Hence I neglected The Exponential Time Complexity of TSP Algorithms.
AS for stones number vary from 1-6 ,it is to small, hence neglected.
We have assumed that number of stones is constant , otherwise if number of stones will becomes variable then order of Algorithm would not be O(MN).

8 Correctness of TSP

1. We have to find the shortest path such that one can move from source to destination by collecting all the stones present in the maze.

2. Let denote source as s, destination as d and an array of stones as s1,s2,..,s(k) where k is the number of stones.Our general path will then look like :-¿ General path -¿ s—-s(i)—s(j)—d
   i.e we have a path with starting vertex as "s" and ending vertex as "d" and in between we have some permutation of stones vertices say s(i) where 1≤i≤k.

3. So, we can decompose our general path as follows : General-Path: -¿ s–(s1) + (s1)—(s2)+....+ s(k)–d (this is just an example not necessary actual shortest path)

4. So, we have divided our original path into (k+1) different paths.

5. Now to make our general path to shortest path we have to minimize the length/size of each (k+1) paths , so that size of overall path can be minimized so that we can obtain shortest path.

6. Now at first step we calculated the shortest path between "each pair" from the set of vertices s, s1,..,s(i)...,s(k),..,d by applying **BFS** on grid by taking one element of pair as source and other as destination. This is done by using function **BFS**.

7. We will try to convert this problem into Traveling Salesman Problem :

   - Assuming Complet connected Graph with vertices as cells of source and stones.
   - Weight of an Edge connecting two vertices represent the shortest path length between that two vertices which we already have calculated in step 5.
   - We will define our Cost-Matrix as follows , where a(i,j) := Shortest path length from source "i" to destination "j" for i not equal to (!=) j and a(i,j)= infinity for i = j.
   - We have mapped each cell of source and stones such that it represent cost matrix , and it is done as follows i=0 represent source.
   - i=1 represent cell of stone s1 , similarly i=k represent s(k). So, finally size of Cost-Matrix is (k+1)*(k+1).
   - Now we have to start from source and have to visit all other vertices (i.e., vertices representing stones cells) before returning to starting vertices.

   Hence , we have to minimize the cost of travels (i.e., sum of edge weight during traversal). This problem is same as Original Traveling Salesman Problem.

8. We now apply Traveling Salesman Algorithms . We have used following functions:

   - solve

- rowReduction
- columnReduction
- calculateCost

)

9. By assuming Correctness of TSP Algorithm, we will obtain paths which will minimize the cost of travels on complete graph. This is done by using function **printPath**.

10. Now, as TSP algorithms, gives the order of traversal of vertices (which are none other than source and stones vertices !!) so , by using this order , we will construct our original "Shortest Path". This is done in "main" function by taking help of **Search** funtion.

11. Hence finally we obtain required Shortest Path.

9 References

1. https://www.lancaster.ac.uk/staff/letchfoa/articles/2013-Steiner-TSP.pdf

2. Compact Formulations of the Steiner Traveling Salesman Problem and Related Problems

3. The Steiner Traveling Salesman Problem and Its Extensions

4. https://www.techiedelight.com/travelling-salesman-problem-using-branch-and-bound/

5. https://www.geeksforgeeks.org/heaps-algorithm-for-generating-permutations/