SAFEST PATH REPORT

18CSC204J -Design and Analysis of Algorithms Laboratory

Submitted by

Rushaan Gandhi [RA2011028010105]

Aditya Singh [RA2011028010089]

Under the guidance of

Dr. M. Shobana

Assistant Professor, Department of Networking and Communication

In Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING
with specialization in Cloud Computing

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

July 2022

BONAFIDE CERTIFICATE

Certified that this mini project report titled “Safest Path” is the Bonafede work done by Rushaan Gandhi [RA2011028010105] and Aditya Singh [RA20110280089] who carried out the mini project work and Laboratory exercises under my supervision for **18CSC204J -Design and Analysis of Algorithms Laboratory**. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE

Dr. M. Shobhana

DAA – Course Faculty

Assistant Professor

Department of Network and Communication

Signature of the Internal Examiner-I

Signature of the Internal Examiner

ABSTRACT

Imagine a case where there is a given path in the form of a rectangular matrix with explosives planted. We are given the coordinates of the explosives, and are described in the path as 0s. If we are at one end of the path and we have to reach the other end safely, we have to avoid the 0s, and traverse through the matrix. This can be done in the following ways: (i) by traversing through the right direction (ii) by traversing through the vertically downward direction (iii) by traversing diagonally downward right direction.

Contribution Table

Contribution	Member
Problem explanation with design techniques used in algorithm and conclusion.	**Aditya Singh [RA2011028010089]**
Problem definition with diagram and explanation of algorithm and conclusion.	**Rushaan Gandhi [RA2011028010105]**

TABLE OF CONTENTS

LIST OF FIGURES

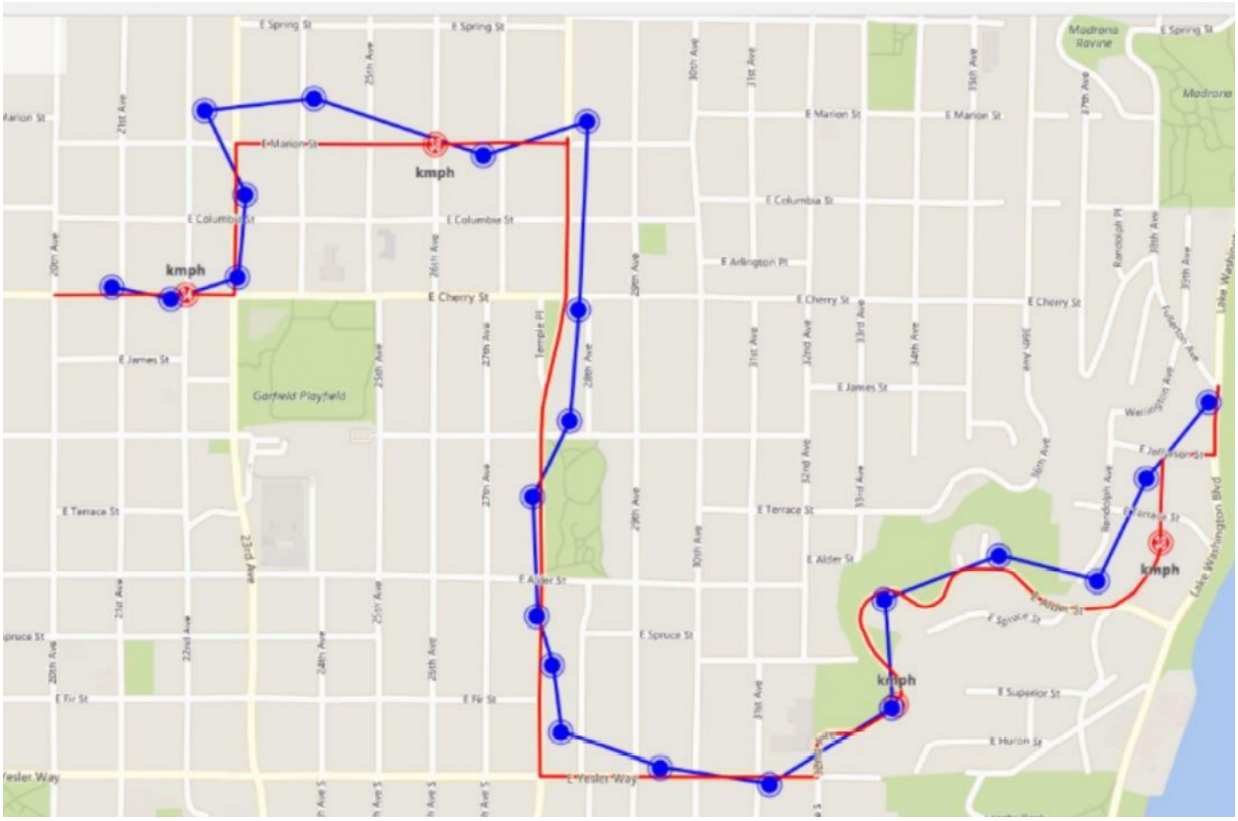
List of Symbols and Abbreviations

Symbols	Full Forms
NLU	Natural Language Understanding
DFS	Depth First Search
Seq	Sequence
BFS	Breadth First Search
Algo	Algorithm

CHAPTER-1

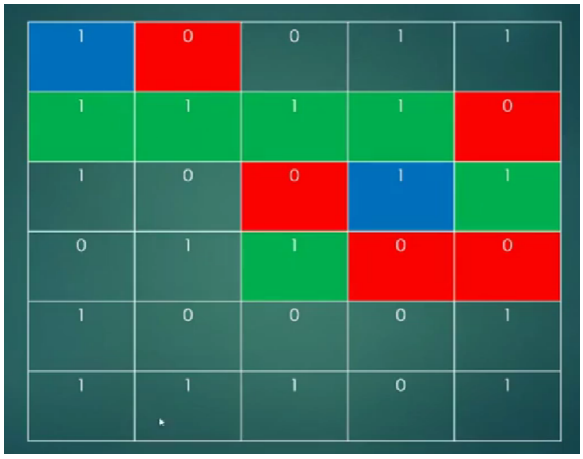
PROBLEM DEFINITION

Given a path in the form of a rectangular matrix having few explosives arbitrarily placed and are marked as 0, calculate all safe routes possible from any cell in the first column to any cell in the last column of the matrix. We have to avoid the explosives. We are allowed to move to adjacent cells and diagonal cells which are not explosives i.e., the route can contain diagonal moves.



CHAPTER-2

PROBLEM EXPLANATION



Description:

1. For simplicity we will fix the (1,1) element of the matrix to 1. So that the probability to start the algo is always 1.

2. As we can see in the above sample matrix, first we will traverse through the first row.

2.a. As soon as we reach up to the (1,2) element, we see it is zero, hence we will back track to (1,1), and find if any other path is possible or not.

3. We see that there is a path possible from (1,1) → (2,1) → (2,2) → (2,3) → (2,4) → (2,5) X.

3.a. Hence, we will back track to the previous solution element i.e. (2,4) and see whether there is another path possible from it.

4. We see from (2,4) there is a path leading to (3,4) →(3,5) which is the element in the last column of the matrix.

5. Hence we have got one solution!

6. Now again, it will backtrack to the previous solution element i.e. (3,4) and find if there is any other path possible.

6.a. We see that the next path is through the element (4,3), but after that there are no paths possible.

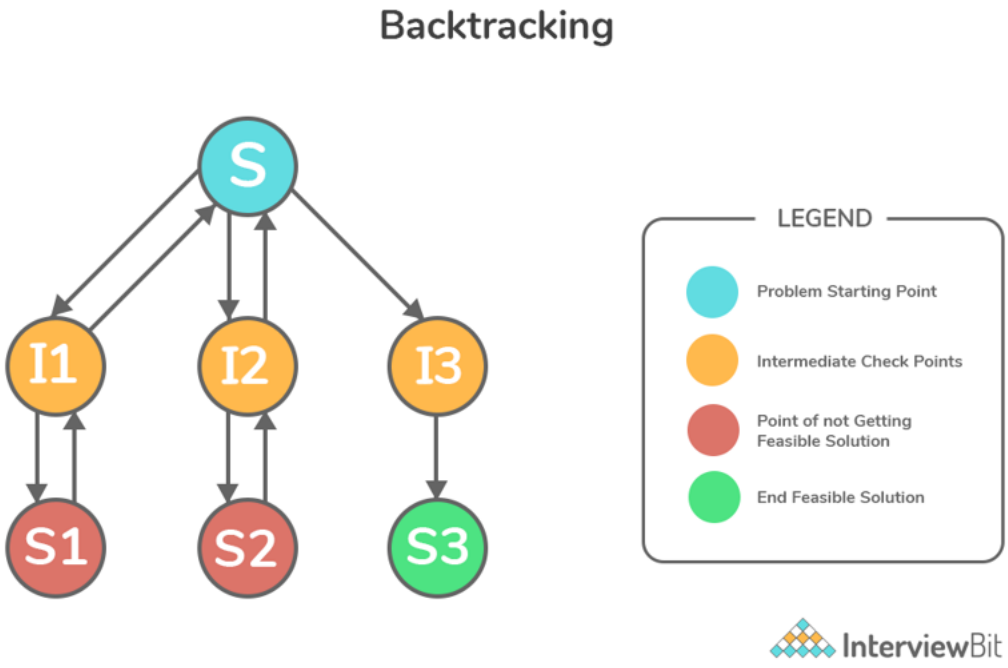
7. Hence our final solution for the given sample problem is:

(1,1) → (2,1) → (2,2) → (2,3) → (2,4) → (3,4) → (3,5)

CHAPTER-3

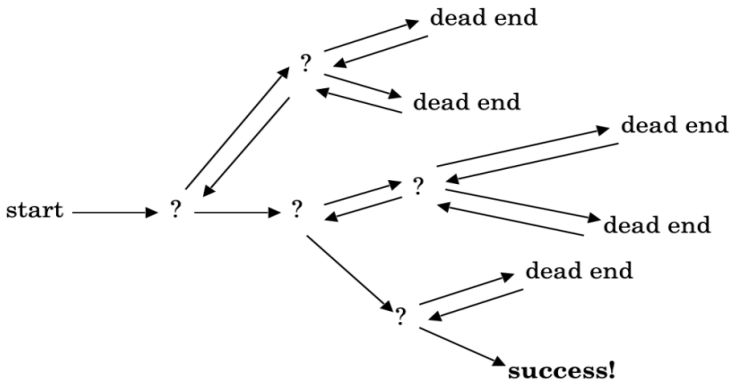
DESIGN TECHNIQUES

Backtracking Algorithm:



- Backtracking is used to solve problems in which a sequence of objects is chosen from a specified set so that the sequence satisfies some criterion.
- Backtracking is a modified depth-first search of a tree.
- Backtracking is the procedure whereby, after determining that a node can lead to nothing but dead nodes, we go back ("backtrack") to the node's parent and proceed with the search on the next child.

Backtracking Animation



1. We will first mark all the adjacent cells of the landmines as unsafe.
2. Then for each safe cell of the first column of the matrix we move forward in all allowed directions and recursively check if they lead to the destination or not.
3. At each intersection, we have to decide between three or fewer choices:

   Go straight

   Go left

   Go right
4. Each choice leads to another set of choices.
5. One or more sequences of choices may (or may not) lead to a
6. solution
7. If the destination is found, we update the value of the shortest path.
8. Else if none of the above solutions work then we return false from our function.

CHAPTER-4

ALGORITHM FOR THE PROBLEM

1. First, we will create a random matrix of 0s and 1s using the timeit and random libraries in python
2. Append the values to the sample path, as we have to print the sum of paths that are formed in the matrix.
3. Now we take input from the users i.e., the number of rows and number of columns.
4. Generate the matrix with rows and columns.
5. Set the first element of the matrix mat[0][0] = 1, else the algo will not start.
6. Use the array "visited" to store the value of the last visited cell.
7. Array q for backtracking algo.
8. We initialize the path value to zero and then start the timer and run the loop.
9. Pop out the first element i.e., (0,0) and store it in a variable (x,y).

```
pt = q.pop(0)
x,y = pt
```

10. If neighboring cells have value 1, we will append that cell to the q which is used for backtracking and also the visited cell.
11. when we reach the last column of the matrix then we increment the path variable by 1.

```
elif(y == columns-1):
    if (mat[x][y] == 1):
        paths+=1
```

12. for the leftmost column, excluding the first and the last row, again if we find the neighboring cell as 1, we append that cell to q and also the last visited value.
13. For the bottom left corner (`x==rows-1&&y==0`) , we check the above cell, cell to the right and the diagonally above right cell. If they contain 1, we append it into q and append it to the visited array.
14. for the entire first row elements (`x,x+2`) we check the vertically down cell, cell to the right and the diagonally below right cell. If they contain 1, we append it into q and append it to the visited array.
15. for the entire last row (`x==rows-1`) we check the vertically down cell, cell to the right and the diagonally below right cell. If they contain 1, we append it into q and append it to the visited array.
16. for the inside elements, we check the vertically down cell, cell to the right and the diagonally below right cell. If they contain 1, we append it into q and append it to the visited array.
17. Stop the timer.
18. add the elements to the sample path
19. print the sample path and the cost.
20. print the total number of paths.
21. print the total time taken.

test case1:

```
Enter number of rows: 5
Enter number of columns: 5

1 1 0 1 0
0 0 1 0 0
1 0 1 1 0
1 1 0 1 1
0 0 0 1 1

Some of the possible paths are:


[0, 0] -> [0, 1] -> [1, 2] -> [2, 3] -> [3, 4]
Cost of the path is: 4

[0, 0] -> [0, 1] -> [1, 2] -> [2, 2] -> [3, 3] -> [4, 4]
Cost of the path is: 5

Total number of paths: 14

Time taken: 0.00013359999999984495 seconds
>>>
```

test case2:

```
Enter number of rows: 10
Enter number of columns: 8

1 0 1 0 1 1 0 1
1 0 0 1 1 0 1 0
1 1 1 0 1 0 0 0
1 0 0 1 1 0 1 1
0 0 0 1 1 0 0 1
1 1 1 0 1 0 0 1
1 1 0 1 0 0 1 1
0 1 0 0 1 0 1 1
0 1 1 0 0 1 1 1
1 0 0 1 1 1 1 0

Some of the possible paths are:


[0, 0] -> [1, 0] -> [2, 1] -> [2, 2] -> [1, 3] -> [0, 4] -> [0, 5] -> [1, 6] -> [0, 7]
Cost of the path is: 8

[0, 0] -> [1, 0] -> [2, 1] -> [2, 2] -> [3, 3] -> [4, 3] -> [5, 2] -> [6, 3] -> [7, 4] -> [8, 5] -> [7, 6] -> [6, 6] -> [5, 7]
Cost of the path is: 12

[0, 0] -> [1, 0] -> [2, 1] -> [2, 2] -> [3, 3] -> [4, 3] -> [5, 2] -> [6, 3] -> [7, 4] -> [8, 5] -> [7, 6] -> [6, 7]
Cost of the path is: 11

[0, 0] -> [1, 0] -> [2, 1] -> [2, 2] -> [3, 3] -> [4, 3] -> [5, 2] -> [6, 3] -> [7, 4] -> [8, 5] -> [7, 6] -> [7, 7]
Cost of the path is: 11

[0, 0] -> [1, 0] -> [2, 1] -> [2, 2] -> [3, 3] -> [4, 3] -> [5, 2] -> [6, 3] -> [7, 4] -> [8, 5] -> [7, 6] -> [8, 7]
Cost of the path is: 11

Total number of paths: 1014

Time taken: 0.022149499999999378 seconds
>>>
```

test case3:

```
Enter number of rows: 3
Enter number of columns: 10

1 1 0 0 0 0 0 1 1 1
1 0 0 1 1 1 1 1 0 1
1 1 0 0 0 0 1 1 0 0


Total number of paths: 0

Time taken: 4.130000000035494e-05 seconds
>>>
```

CHAPTER-5

EXPLANATION OF ALGORITHM

Approach 1

1. A brute force approach could be to generate all the possible paths from the first column of the field to the last column and choose the one with the shortest length.
2. The idea is to use the concept of backtracking.
3. Firstly we mark all the unsafe cells as 0 which are 1. This is done so as to avoid generating paths passing through an unsafe cell.
4. Then for each safe cell in the first column, we generate the path starting from that cell by recursively moving in all the possible directions that are left, right, above, and below. Also, keeping in mind not to include the cells which have been marked as zero.
5. While generating the path we keep track of which cells have already been visited so that we do not move to the same cell. We also keep track of the length of the path generated so far.
6. If a path through the current cell leads to a safe cell in the last column of the field, we update the shortest path length which we have found so far.
7. Otherwise, we mark the current cell as unvisited and backtrack to the previous cell.
8. In case there is no such path that leads to a safe cell in the last column of the field, we return -1.

Alternative Approach

1. This approach is similar to the previous approach.
2. The idea is to traverse the field using BFS instead of using recursion.
3. Firstly we mark all the unsafe cells as 0 which are 1. This is done so as to avoid generating paths passing through an unsafe cell.
4. Now, push all the safe cells, present in the first column, into the queue and apply BFS.
5. During the traversal, we keep track of which cells have already been visited so that we do not move to the same cell.
6. We also keep track of the length of the path required to reach the current cell, by storing this length in a matrix.
7. we reach a safe cell in the last column, then we have found the shortest path.
8. In case there is no such path that leads to a safe cell in the last column of the field, we return -1.

CHAPTER-6

COMPLEXITY ANALYSIS

Time Complexity

O(3^(M*N)), where ‘M’ represents the number of rows in the field and ‘N’ represents the number of columns in the field. Marking the unsafe nodes as 0 requires O(M*N) time. Also, in the worst case, we make three recursive calls for every cell in the field. As there are M*N cells, hence it requires O(3^(M*N)). So, the overall time complexity is O(M*N + 3^(M*N)) = O(3^(M*N))

Space Complexity

O(M*N), where ‘M’ represents the number of rows in the field and ‘N’ represents the number of columns in the field. Extra space is required for the recursion stack. In the worst case, the depth of the recursion stack can be of order O(M*N). Hence, the overall space complexity is O(M*N).

CHAPTER-7

CONCLUSION

Through this project we have implemented the backtracking algorithm to find the shortest and the safest path in a path with landmines. We choose Backtracking Algorithm over Breadth First Search because using backtracking we can almost solve any problems, due to its brute-force nature. It can be used to find all the existing solutions if they exist for any problem. It is a step by-step representation of a solution to a given problem, which is very easy to understand.

REFERENCES

[1] Saksham Saraswat, Siddhartha Mishra, Vikas Mani, Shristi Priya. "GALGOBOT – The College Companion Chatbot". 2020, Fifth International Conference on Intelligent Computing and Control Systems (ICICCS 2021)

[2] Chun Ho Chan, Ho Lam Lee, Wing Kwan Lo, Andrew Kwok-Fai Lui. "Developing a chatbot for college student programme advisement." 2018, International Symposium on Educational Technology

[3] Jitendra Purohit, Aditya Bagwe, Rishabh Mehta, Ojaswini Mangaonkar, Elizabeth George. "Natural Language Processing based Jaro-The Interviewing Chatbot." 2019, 3rd International Conference on Computing Methodologies and Communication (ICCMC)

[5] Hrushikesh Koundinya K, Vaishnavi Putnala, Ajay Krishna Palakurthi, Dr. Ashok Kumar K "SMART COLLEGE CHATBOT USING ML AND PYTHON" 2020, International Conference on System, Computation, Automation and Networking (ICSCAN)

[4] Bhaumik Kohli , Tanupriya Choudhury , Shilpi Sharma , Praveen Kumar. "A Platform for Human-Chatbot Interaction Using Python." 2018, Second International Conference on Green Computing and Internet of Things (ICGCIoT)

[6] Prof.K.Bala, Mukesh Kumar, Sayali Hulawale, Sahil Pandita. "Chat-Bot For College Management System Using A.I." 2017, International Research Journal of Engineering and Technology (IRJET)

APPENDIX

CODE

```
import random
import timeit

# prints the matrix
def d(mat):
	print()
	for item in mat:
		print(*item)
	print()

# appending values to sample path
def getPath(i, j):

	if i==j==0:
		return

	x, y = visited[i][j][0]
	sample_paths.append([i, j])
	getPath(x, y)


rows = int(input("\nEnter number of rows: "))
columns = int(input("Enter number of columns: "))

# generate random matrix of 1 and 0
mat = [[random.randint(0, 1) for _ in range(columns)] for _ in range(rows)]

# the a[0][0] element will be 1
mat[0][0] = 1

# print the matrix
d(mat)

# array to store the last cell visited
visited = [[[] for _ in range(columns)] for _ in range(rows)]

# queue for backtracking
q = [[0, 0]]

# initial there are 0 paths
paths = 0


# start the timer
start = timeit.default_timer()

while(len(q)>0):
	pt = q.pop(0)
	x, y = pt
```

```
# the top left corner
	if (pt == [0, 0]):
		if (mat[x][y+1] != 0):
			q.append([x, y+1])
			visited[x][y+1].append([0, 0])

		if (mat[x+1][y+1] != 0):
			q.append([x+1, y+1])
			visited[x+1][y+1].append([0, 0])

		if (mat[x+1][y] != 0):
			q.append([x+1, y])
			visited[x+1][y].append([0, 0])

	# last column
	elif (y == columns-1):
		if (mat[x][y] == 1):
			paths+=1

	# leftmost column excluding 1st and last row
	elif (x>0 and x<rows-1 and y == 0):
		for i in range(x-1, x+2):
			for j in range(y, y+2):
				if ((i!=x or j!=y) and mat[i][j]!=0 and ([i, j] not in  visited[x][y])):
					q.append([i, j])
					visited[i][j].append([x, y])

	# bottom left corner
	elif (x == rows-1 and y == 0):
		if (mat[x][y+1] != 0 and ([x, y+1] not in visited[x][y])):
			q.append([x, y+1])
			visited[x][y+1].append([x, y])

		if (mat[x-1][y+1] != 0 and ([x-1, y+1] not in visited[x][y])):
			q.append([x-1, y+1])
			visited[x-1][y+1].append([x, y])

		if (mat[x-1][y] != 0 and ([x-1, y] not in visited[x][y])):
			q.append([x-1, y])
			visited[x-1][y].append([x, y])

	# entire first row
	elif (x == 0):
		for i in range(x, x+2):
			for j in range(y-1, y+2):
				if ((i!=x or j!=y) and mat[i][j]!=0 and ([i, j] not in visited[x][y])):
					q.append([i, j])
					visited[i][j].append([x, y])



# entire last row
	elif (x == rows-1):
		for i in range(x-1, x+1):
			for j in range(y-1, y+2):
				if ((i!=x or j!=y) and mat[i][j]!=0 and ([i, j] not in visited[x][y])):
					q.append([i, j])
					visited[i][j].append([x, y])
```

```
        # inside the outermost square
        # excluding 1st and last row as well as 1st and last column
        elif (x>0 and x<rows-1 and y>0 and y<columns-1):
                for i in range(x-1, x+2):
                        for j in range(y-1, y+2):
                                if ((i!=x or j!=y) and mat[i][j]!=0 and ([i, j] not in visited[x][y])):
                                        q.append([i, j])
                                        visited[i][j].append([x, y])

        else:
                print(pt, "Error")
                break

# stop the timer
stop = timeit.default_timer()

all_paths = []

# adding the elements to sample path
for i in range(rows):
        if len(visited[i][columns-1])>0:
                sample_paths = []
                getPath(i, columns-1)
                sample_paths.append([0, 0])
                sample_paths.reverse()
                all_paths.append(sample_paths)

# if paths exist then print some of them
if paths>0:
        print("Some of the possible paths are: \n")

        for item in all_paths:
                print()
                print(*item, sep=' -> ')
                print("Cost of the path is:", len(item)-1)

# print the total number of paths
print("\nTotal number of paths:", paths)

# print the time taken for algorithm
print("\nTime taken:", stop - start, "seconds")
```