Artificial Intelligence: Foundations and Applications (AIFA)

AI60005

ASSIGNMENT 1

Term Project On

Multi-Agent Path Finding problem for a warehouse

GROUP DETAILS

Roll Number	Name
17ME10001	Abhinav Gupta
17EE10051	Swatantra Kumar
19ME10056	Sameer Kosurkar

1: Algorithm for optimal solution

For the purpose of finding the optimal solution to this problem, we explored the applicability of the two most popular uninformed search algorithms, a) Depth-first Search (DFS), and b) Breadth-first Search (BFS). While DFS works on a stack data structure and explores as far as possible to the deepest node before backtracking, BFS utilizes a queue data structure and explores as far as possible on every level before backtracking and being forced to expand other nodes.

However, since DFS goes to the deepest node of a branch before backtracking, there is always a risk of doing unnecessary large computations searching solutions deep down when the goal nodes are at a small depth. Additionally, BFS ensures that the solution obtained is an optimal solution. Therefore, we decided to implement this algorithm.

BFS CODE Explanation :

We have taken three structures - box, robot and robots -
box - contains information about each task - x and y coordinates of current and final locations, if the object picked by any robot.
robot - contains info. about each robot - x and y coordinates of current and final locations, is the robot holding any object?, if yes!, then which object.
robots - contains info. about every object and task.

FUNCTIONS :
pickup - for picking up object when the robot's and object's location are same
drop - robot drops object at the same location
down - robot moves down by one step
up - robot moves up by one step
right - robot moves right by one step
left - robot moves left by one step
Stay - robot stays at the same location
Goal - takes in robots(configuration) and returns if the goal configuration is reached or not
Combinations - uses all the above functions and finds all possible children of a given node as an input(robots)
In the main function, we have used a queue of robots(struct type) as the data structure for bfs. In each iteration in the while loop , a node is removed from the queue and explored using the combinations function. The res variable stores the count for no. of times the loop is running.

Heuristic algorithm

A* heuristic algorithm was being implemented. More details on the implementation and the type of heuristic considered can be found below.

Output :

When the program ends running, we get an output saying GOAL FOUND.
Further we get the total time steps taken to complete the tasks and reach the goal. Next line tells us the time taken by the program to run. Last lines outputs the number of configurations explored.

While the program is running, we get continuous outputs showing that the program is running. Also, when a node at different heights is explored compared to the initial node, we get the output showing the height of the tree at that instant.

A* CODE Explanation :

HEURISTIC ESTIMATE : The maximum time step of all robots' time steps required for moving from initial location to final location.
$h = \max(|x - x_{final}| + |y - y_{final}|)$.

We have taken three structures - box, robot and robots -
box - contains information about each task - x and y coordinates of current and final locations, if the object picked by any robot. Also here the heuristics values and goal costs are stored.
robot - contains info. about each robot - x and y coordinates of current and final locations, is the robot holding any object?, if yes!, then which object.
robots - contains info. about every object and task. Also, contains overall value of the heuristics(h), goal costs(g) and heuristics plus goal costs(f).

FUNCTIONS :
pickup - for picking up object when the robot's and object's location are same
drop - robot drops object at the same location
down - robot moves down by one step
up - robot moves up by one step
right - robot moves right by one step
left - robot moves left by one step
Stay - robot stays at the same location
Goal - takes in robots(configuration) and returns if the goal configuration is reached or not
Compare_structs - compares if two configurations are same or not(takes in 2 robots structs)

Combinations - uses all the above functions and finds all possible children of a given node as an input(robots)

In the main function, we have used a vector as a data structure for A*. Because we have to arrange the elements in a sorted manner with respect to the f values. Mostly all the functions are the same as in bfs. While inserting an element in the vector, it inserted bases on the f values.

Input :

INDEX VARIES FROM 0 TO N-1 AND 0 TO M-1 IN GRID
First line contains two spaced integers - n and m - n for no. of rows and m for no. of rows for the grid.

Second line contains single integers - r - no. of robots

Next r lines contain 4 spaced integers

i th line contains x, y, final_x, final_y - x and y coordinates for initial and final locations of the i th robot respectively

Next line contains single integer - t - no. of tasks

Next t lines contain 4 spaced integers

i th line contains x, y, dest_x, dest_y - x and y coordinates for initial and final locations of the i th task respectively

Next line contains single integer - te - no. of temporary locations

Next te lines contain 2 spaced integers

i th line contains x, y - x and y coordinate for the i th temporary location

Next line contains single integer - obs - no. of obstacles

Next obs lines contain 2 spaced integers

i th line contains x, y - x and y coordinate for the i th obstacle location

Example input / outputs :

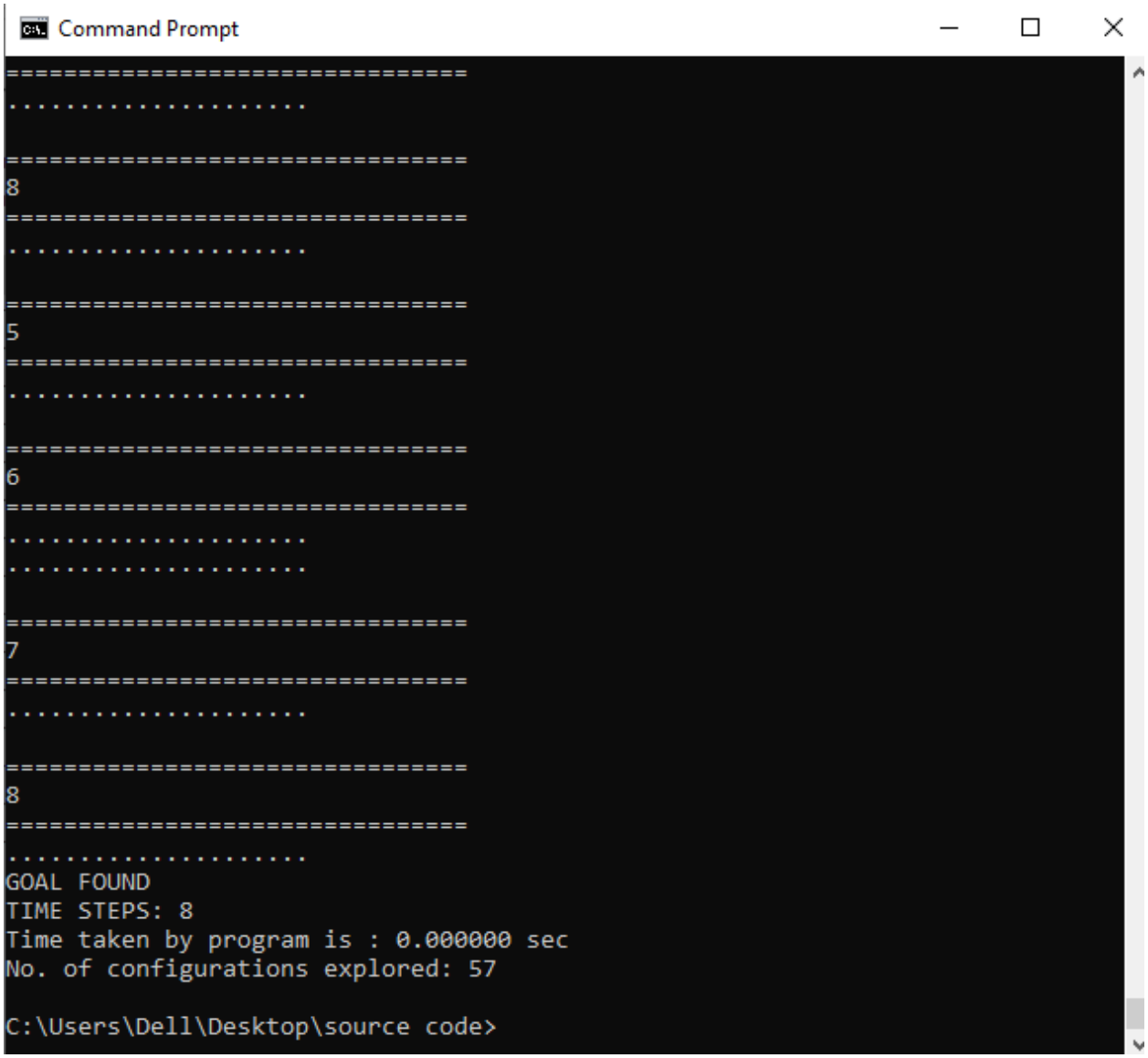
1)

INPUT :

3 3
1
1 1 0 2
1
0 0 2 0
0
2
0 1 2 1

OUTPUT : (OPTIMAL SOLUTION)

OUTPUT : (HEURISTIC APPROACH)

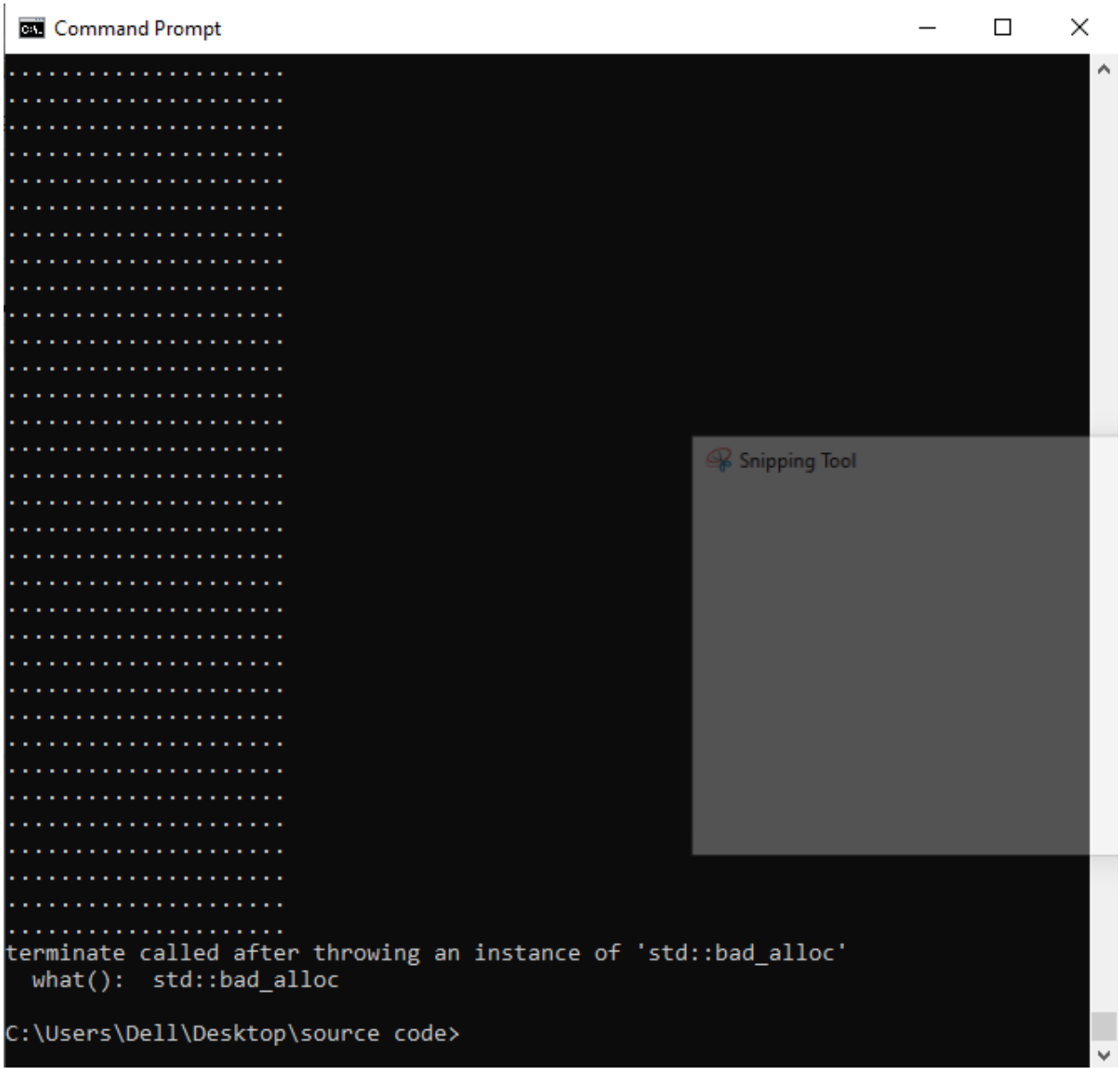


2)

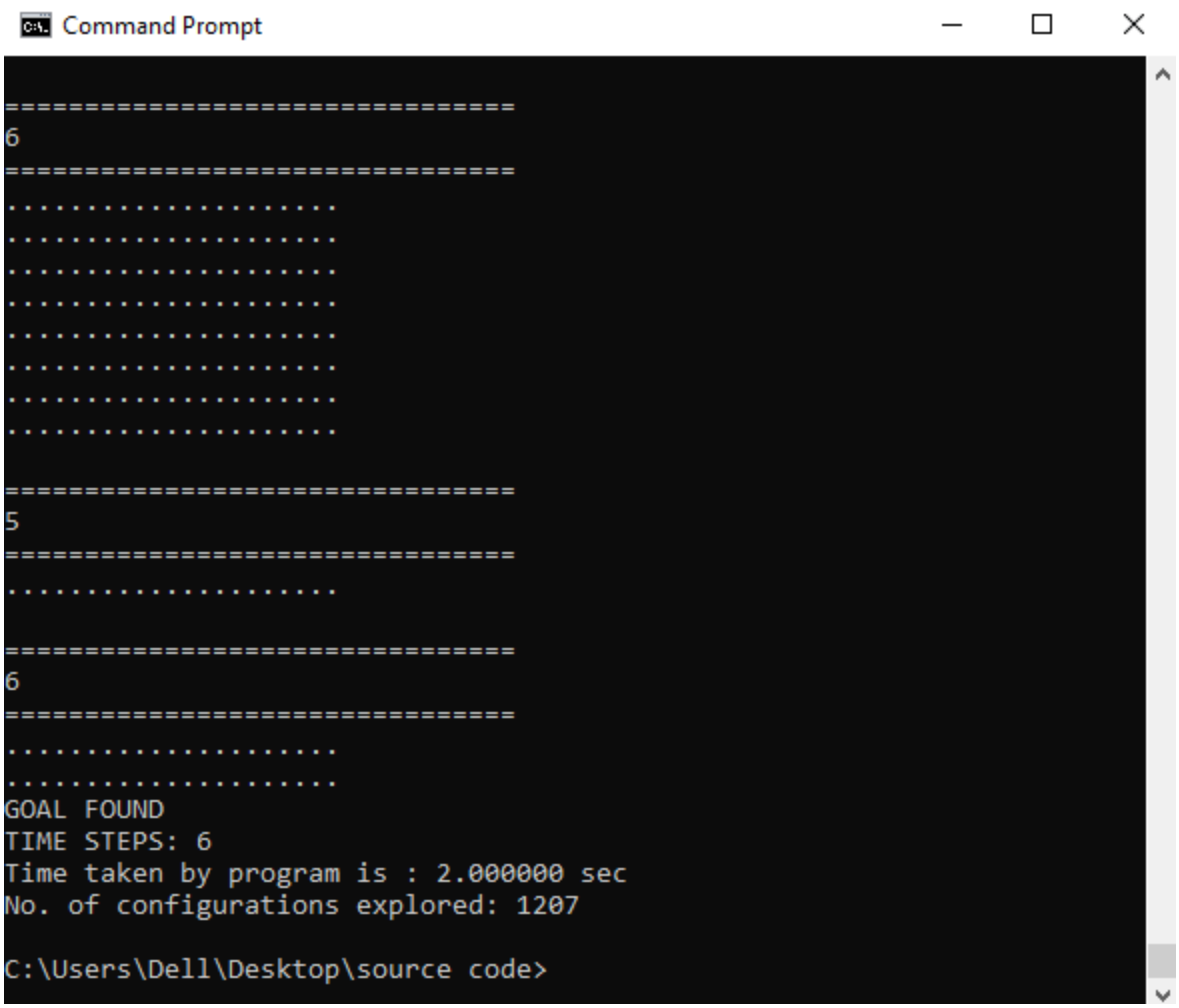
INPUT :

2 3
2
0 0 0 0
1 2 1 2
2
0 2 1 0
1 1 0 1

0
0
OUTPUT : (OPTIMAL SOLUTION)



OUTPUT : (HEURISTIC APPROACH)

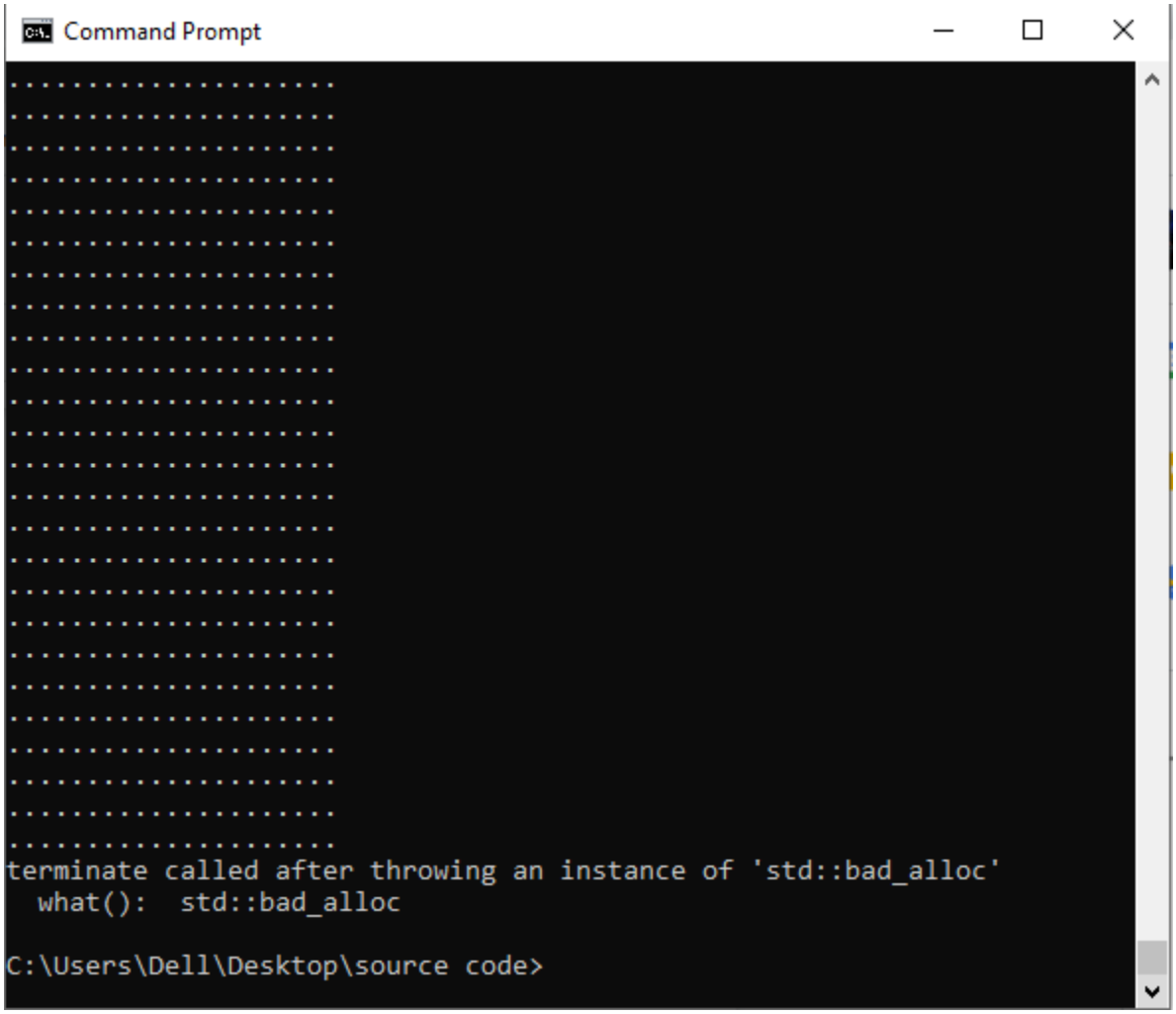


3)

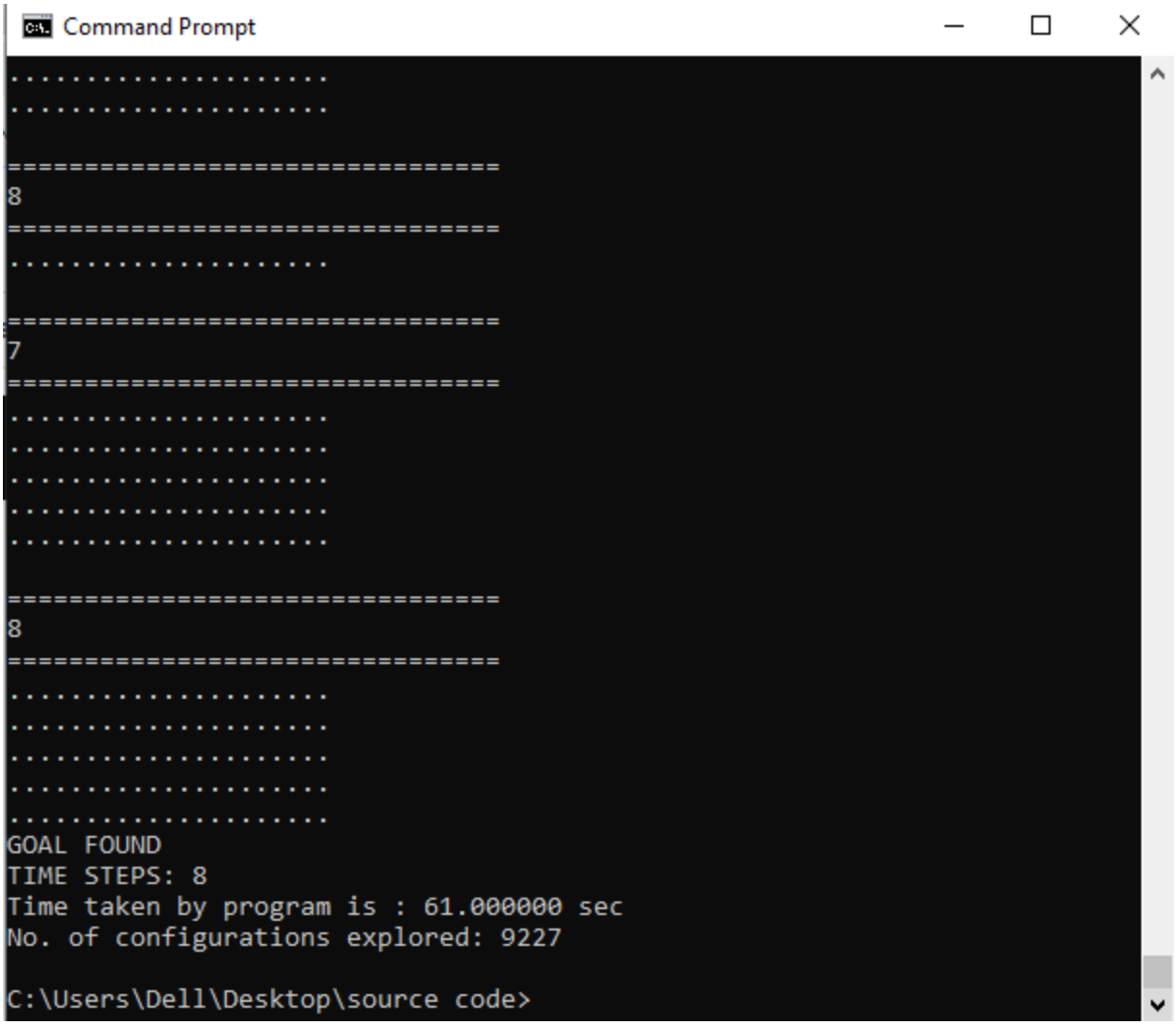
INPUT :

3 4
2
1 1 0 0
0 2 2 0
2
0 3 2 3
1 3 2 2
0
0

OUTPUT : (OPTIMAL SOLUTION)



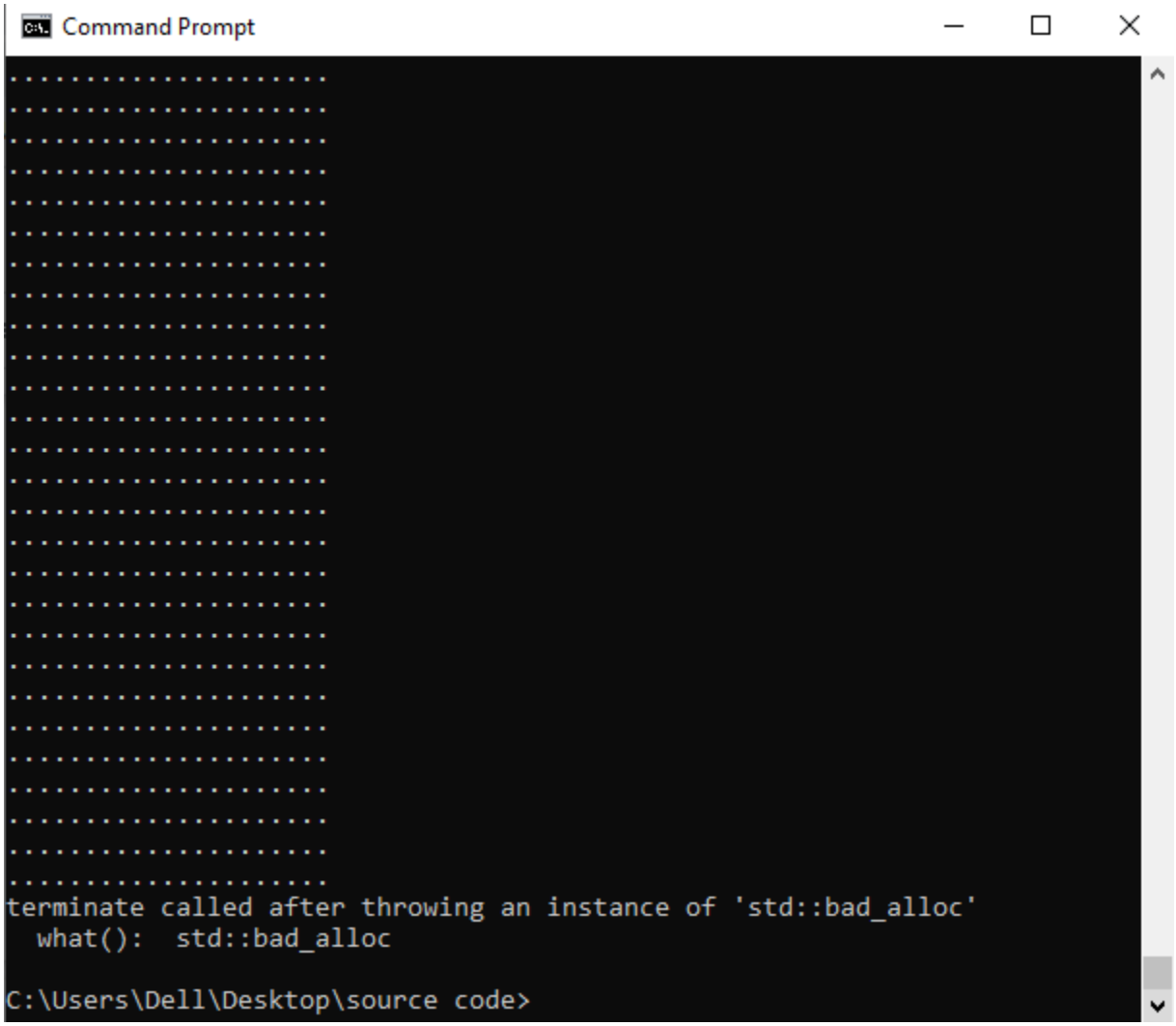
OUTPUT : (HEURISTIC APPROACH)



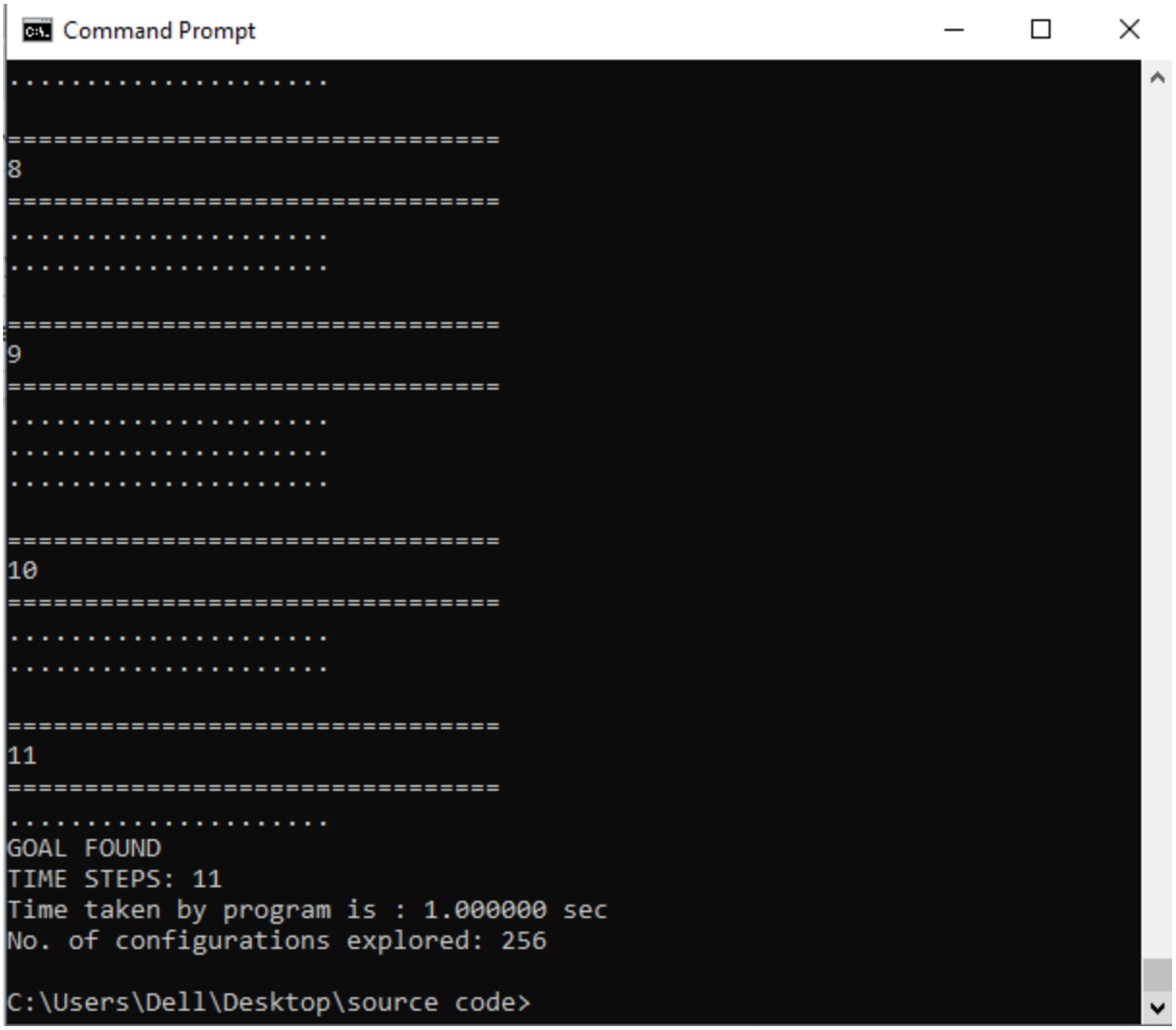
4)

INPUT :
4 5
1
3 0 0 2
1
3 2 3 4
1
2 2
2
1 1
3 3

OUTPUT : (OPTIMAL SOLUTION)



OUTPUT : (HEURISTIC APPROACH)



CONCLUSION :

From the examples, we observe that bfs explores so many nodes as compared to A*. In some cases, bfs gives runtime errors. And in the case of A*, because only a few nodes are explored, time taken is less.