CS362 Artificial Intelligence Lab Report 1 Team Ascent

Adarsh Shukla (202051008)
Arun Kumar Gupta (202051035)
Aman Anand (202051019)
Kalash Singh Jadoun (202051097)

I. Lab Assignment 0

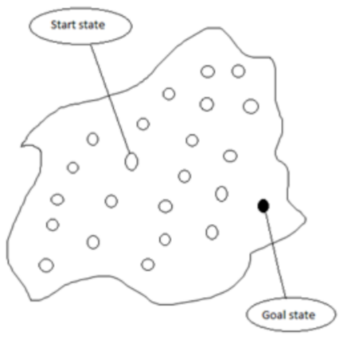
Objective: To be able to model a given problem in terms of state space search problem and solve the same using BFS/DFS

Problem Statement:

1) In the rabbit leap problem, three east-bound rabbits stand in a line blocked by three west-bound rabbits. They are crossing a stream with stones placed in the east west direction in a line. There is one empty stone between them. The rabbits can only move forward one or two steps. They can jump over one rabbit if the need arises, but not more than that. Are they smart enough to cross each other without having to step into the water?
2) The missionaries and cannibals problem is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place. This problem is famous in AI because it was the subject of the first paper that approached problem-formulation from an analytical viewpoint.

Solution:

- We model a problem as state space search such that in state space each element is state. A state is a description of the world in which the problem solver operates.The given state is called START STATE. The desired state is called GOAL STATE.The task is to make a sequence of move such that Agent ends up in a goal state.

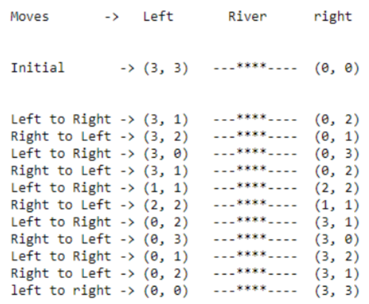


State space search issues are the ones presented. We've provided a starting point, steps that could be done to move from one state to the next, and a goal state that we need to get to. A path is formed by a series of actions, and a solution is a path from the starting point to the finished condition. A state space can be visualised as a directed graph, with the vertices denoting states and the lines between them denoting actions. Above two problems satisfy the property of state space so can model these problem as state space problem.

2. SOLUTION USING BFS

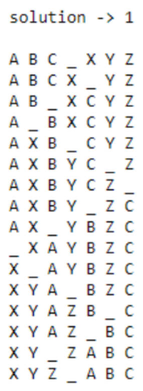
We used a breadth-first search to solve the problem. Inspects the nodes (states) in the order in which they are available. For that, we used the first-in, first-out queue data structure. Before moving on to the next level, BFS examines every node at the previous level. BFS thus finds a solution at the right moment.

Rabbit leap problem: No,the solution we have got from bfs is not optimal one.we have explored a total of 44 state as compared to DFS(23 states).So here DFS solution is optimal one.



```
Moves        ->  Left      River      right


Initial        -> (3, 3)   ---****----  (0, 0)


Left to Right -> (3, 1)   ---****----  (0, 2)
Right to Left -> (3, 2)   ---****----  (0, 1)
Left to Right -> (3, 0)   ---****----  (0, 3)
Right to Left -> (3, 1)   ---****----  (0, 2)
Left to Right -> (1, 1)   ---****----  (2, 2)
Right to Left -> (2, 2)   ---****----  (1, 1)
Left to Right -> (0, 2)   ---****----  (3, 1)
Right to Left -> (0, 3)   ---****----  (3, 0)
Left to Right -> (0, 1)   ---****----  (3, 2)
Right to Left -> (0, 2)   ---****----  (3, 1)
left to right -> (0, 0)   ---****----  (3, 3)
```

Miss and can: No,the solution we have got from BFS is not optimal one.we have explored a total of 22 state as compared to DFS(20 states).So here DFS solution is optimal one.

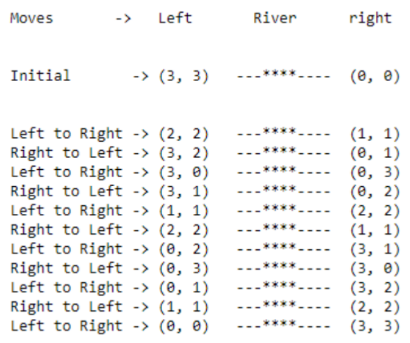


```
solution -> 1

A B C _ X Y Z
A B C X _ Y Z
A B _ X C Y Z
A _ B X C Y Z
A X B _ C Y Z
A X B Y C _ Z
A X B Y C Z _
A X B Y _ Z C
A X _ Y B Z C
_ X A Y B Z C
X _ A Y B Z C
X Y A _ B Z C
X Y A Z B _ C
X Y A Z _ B C
X Y _ Z A B C
X Y Z _ A B C
```

3. SOLUTION USING DFS

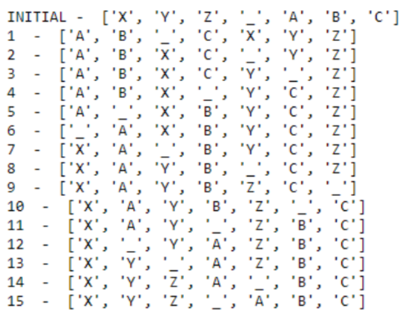
The BFS algorithm and the DFS algorithm are marginally different. DFS algorithm poses a stack of OPEN LIST risks. A fresh candidate is added at the top of the list. Deep in this case refers to choosing a node from the OPEN LIST that is the furthest and deepest away from the START NODE. A programme that prints the solution steps for the issues was developed by our team.

Rabbit leap problem: No,the solution we have got from bfs is not optimal one.we have explored a total of 44 state as compared to dfs(23 states).So here dfs solution is optimal one.



```
Moves       ->  Left      River       right


Initial      -> (3, 3)   ---****----  (0, 0)


Left to Right -> (2, 2)   ---****----  (1, 1)
Right to Left -> (3, 2)   ---****----  (0, 1)
Left to Right -> (3, 0)   ---****----  (0, 3)
Right to Left -> (3, 1)   ---****----  (0, 2)
Left to Right -> (1, 1)   ---****----  (2, 2)
Right to Left -> (2, 2)   ---****----  (1, 1)
Left to Right -> (0, 2)   ---****----  (3, 1)
Right to Left -> (0, 3)   ---****----  (3, 0)
Left to Right -> (0, 1)   ---****----  (3, 2)
Right to Left -> (1, 1)   ---****----  (2, 2)
Left to Right -> (0, 0)   ---****----  (3, 3)
```

Miss and can: No,the solution we have got from bfs is not optimal one.we have explored a total of 22 state as compared to dfs(20 states).So here dfs solution is optimal one.



```
INITIAL -  ['X', 'Y', 'Z', '_', 'A', 'B', 'C']
1  -  ['A', 'B', '_', 'C', 'X', 'Y', 'Z']
2  -  ['A', 'B', 'X', 'C', '_', 'Y', 'Z']
3  -  ['A', 'B', 'X', 'C', 'Y', '_', 'Z']
4  -  ['A', 'B', 'X', '_', 'Y', 'C', 'Z']
5  -  ['A', '_', 'X', 'B', 'Y', 'C', 'Z']
6  -  ['_', 'A', 'X', 'B', 'Y', 'C', 'Z']
7  -  ['X', 'A', '_', 'B', 'Y', 'C', 'Z']
8  -  ['X', 'A', 'Y', 'B', '_', 'C', 'Z']
9  -  ['X', 'A', 'Y', 'B', 'Z', 'C', '_']
10  -  ['X', 'A', 'Y', 'B', 'Z', '_', 'C']
11  -  ['X', 'A', 'Y', '_', 'Z', 'B', 'C']
12  -  ['X', '_', 'Y', 'A', 'Z', 'B', 'C']
13  -  ['X', 'Y', '_', 'A', 'Z', 'B', 'C']
14  -  ['X', 'Y', 'Z', 'A', '_', 'B', 'C']
15  -  ['X', 'Y', 'Z', '_', 'A', 'B', 'C']
```

4. COMPARING SOLUTIONS

We have seen that bfs and dfs give different solution.Major points we have seen are :
1. BFS is better when target is closer to the source and DFS is better when target is far away from the source.
2. Solution depends on which how you are describing next possible state from current state.
3. In our case DFS gives the solution after visiting 20 states while BFS gives solution in 22 states.
4. Solution from both BfS and DFS may be different.

Complexity: We compare time and space complexity of above two problems.Lets assume b be the branching factor and d be the shallowest solution depth m be the maximum depth of the search tree.
A. Miss and can: In this problem we have b(branching factor) = 5
B. Rabbit leap: In this problem we have b(branching factor) = 4

DFS
So TC = $O(b^m)$
and SC = $O(b^m)$
BFS
So TC = $O(b^d)$
and SC = $O(b^d)$
We have seen both bfs and dfs complexity depends on branching factor(b) and depth(d).

REFERENCES

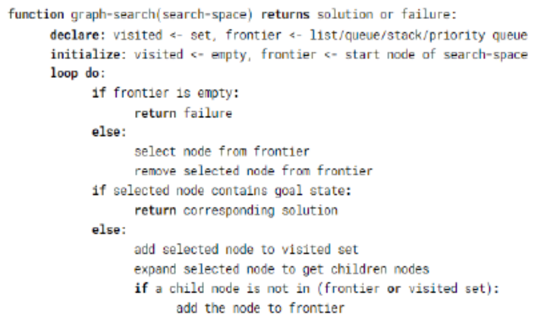
1) Deepak Khemani (2013) A First Course in Artificial Intelligence, Addison-Wesley Professional.
2) Peter Norvig and Stuart J. Russell (1994) Artificial Intelligence: A Modern Approach, 4thd ed.
3) Best case analysis alpha beta pruning [http://www.cs.utsa.edu/ bylander/cs5233/a-b-analysis.pdf]
4) Nim game strategy http://web.mit.edu/sp.268/www/nim.pdf

II. Lab Assignment 1

(A) Write a pseudocode for a graph search agent. Represent the agent in the form of a flow chart. Clearly mention all the implementation details with reasons.

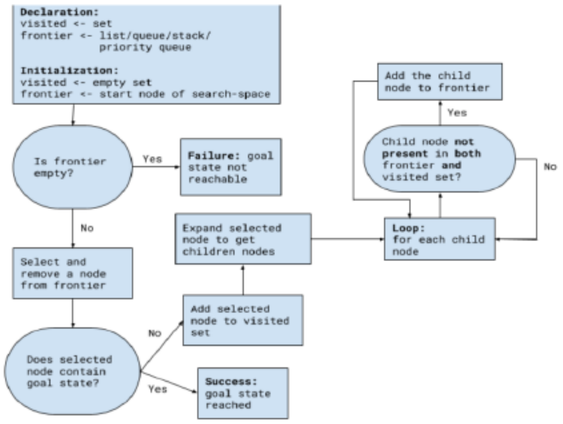
Solution :

(1) Pseudocode of a general graph search algorithm:

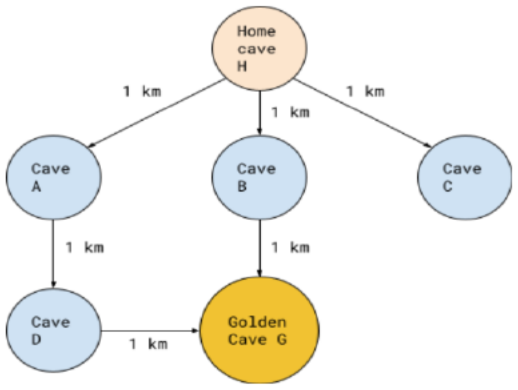


```
function graph-search(search-space) returns solution or failure:
    declare: visited <- set, frontier <- list/queue/stack/priority queue
    initialize: visited <- empty, frontier <- start node of search-space
    loop do:
        if frontier is empty:
            return failure
        else:
            select node from frontier
            remove selected node from frontier
        if selected node contains goal state:
            return corresponding solution
        else:
            add selected node to visited set
            expand selected node to get children nodes
            if a child node is not in (frontier or visited set):
                add the node to frontier
```

(2) Flowchart-


Declaration:
visited <- set
frontier <- list/queue/stack/ priority queue
Initialization:
visited <- empty set
frontier <- start node of search-space
Is frontier empty?
Yes
Failure: goal state not reachable
No
Select and remove a node from frontier
Does selected node contain goal state?
Yes
Success: goal state reached
No
Add selected node to visited set
Expand selected node to get children nodes
Loop: for each child node
Child node not present in both frontier and visited set?
Yes
Add the child node to frontier
No

(3) Implementation details with an example:
To better understand how to implement a graph search algorithm, we will use a practical example. In this example, we will use a queue as the frontier.



As the leader of a forest tribe, you've been informed by another tribe that a golden-colored cave in the forest holds an abundant food supply for your tribe. Your responsibility is to venture into the forest alone and discover the quickest route to the cave. The other tribe has indicated that this particular cave stands out from the rest, as it's not the typical brown-colored cave.

Before embarking on your journey to find the golden cave, it's essential to consider a few factors. Each junction or point along the way will have a cave at its center, including your starting point, which is your home cave, and all the caves are 1 km apart. At each junction, you can only see the names of the adjacent caves and the paths leading to them, and it's possible to reach a dead-end with no further paths. Finally, you can only determine the color of a cave once you arrive at it.

To aid you in your journey, you have a container of water and two lists. One list, called the "visited list," keeps track of the caves you've already explored to prevent revisiting them and getting stuck in an endless loop. The other list, called the "frontier list," tracks the caves you plan to visit next. Additionally, you'll record the path leading to the golden cave to share with your tribe upon your return. Starting from your home cave, which is H, you add it to the frontier list. Upon reaching H, you check if it's the golden cave (which it isn't) and remove it from the frontier list before adding it to the visited list. From H, you can see three paths leading to caves A, B, and C.

For the next three caves, you check if they're already visited or present in the frontier list. If not, you add them to the frontier list. In this case, caves A, B, and C meet both conditions, so you add them to the frontier list. You decide to explore cave A first and upon reaching it, you remove it from the frontier list and add it to the visited list. After checking its color and finding that it's not the golden cave, you look at the paths from cave A and find that you can go to cave D and your home cave H. Since D meets both conditions, you add it to the frontier list, realizing that you can reach it from H via A in a total distance of 2 km. Although H is no longer in the frontier list, it's in the visited list and doesn't require further exploration.

After exploring cave A, you move on to cave B. Upon reaching B, you remove it from the frontier list and add it to the visited list. Checking its color reveals that it's not the golden cave. You then see that you can go to cave G or back to your home cave H from B. After verifying that G meets both conditions, you add it to the frontier list, with a total path distance of 2 km (H-B-G). As H is already visited and not in the frontier list, there's nothing more to do with it. Finally, you explore cave C, remove it from the frontier list, and add it to the visited list. Although C is brown, you take a break and drink some water, hoping to find the golden cave soon. From C, you can only go back to your home cave H, which is already visited and not in the frontier list.

After visiting cave D and adding it to the visited list, you check where you can go next. You notice paths leading to cave G and A from D. Checking the conditions for adding a cave to the frontier list, you realize that G is not in the visited list but is already in the frontier list, so you don't need to add it again. Similarly, since A is already visited and not in the frontier, you won't be revisiting it.

After exploring all the other caves, you are left with only cave G in the frontier list. You visit G, add it to the visited list, and discover that it is the golden cave! You have finally found the cave with unlimited food supplies, and you have also found the optimal path to it. Excitedly, you return home to your tribe and share the directions to the golden cave. You tell them that they need to go from home cave H to cave B, and then take a direct path to cave G, which is the golden cave. The path is 2km long.

(B) Write a collection of functions imitating the environment for Puzzle-8.

Solution:

Code : Link

(C) Describe what is Iterative Deepening Search.

Solution -

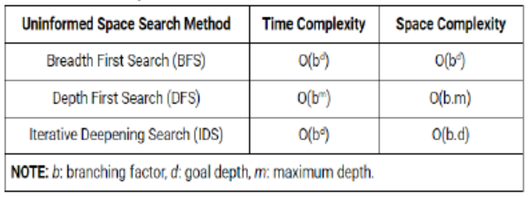
1- Introduction -

DFS has a major drawback in that it may take a long time to reach the goal node if it is located near the root node but not explored by the DFS agent until later. BFS, on the other hand, explores nodes level-by-level and can reach a goal node closer to the root more quickly. However, BFS requires a lot of memory. IDS overcomes these limitations by combining the advantages of DFS (space efficiency) and BFS (faster reach to goal nodes closer to the root) through performing a depth-limited DFS. IDS is also called Iterative Deepening DFS.

(2) How it works

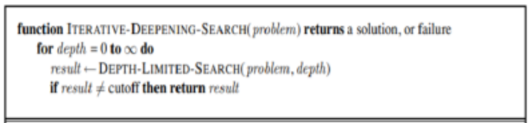
Iterative Deepening Search (IDS) increases the depth limit gradually starting from zero, and then progressing to 1, 2, and so on until a goal node is found at a depth limit of d, which is closest to the root. At each iteration, IDS begins the search routine from the root node. Although the top-level nodes are generated multiple times, it is not costly because most of the nodes in a tree are in the bottom level, and the branching factor is usually more than one. Thus, IDS functions similarly to BFS, but with linear space complexity. Additionally, IDS is complete in nature when the branching factor is finite.

(3) Performance and Comparisons



Uninformed Space Search Method	Time Complexity	Space Complexity
Breadth First Search (BFS)	$O(b^d)$	$O(b^d)$
Depth First Search (DFS)	$O(b^m)$	$O(b.m)$
Iterative Deepening Search (IDS)	$O(b^d)$	$O(b.d)$
NOTE: *b*: branching factor, *d*: goal depth, *m*: maximum depth.		

(4) Algorithm



```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result
```

Figure 3.18 The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits. It terminates when a solution is found or if the depth-limited search returns *failure*, meaning that no solution exists.

(5) Example

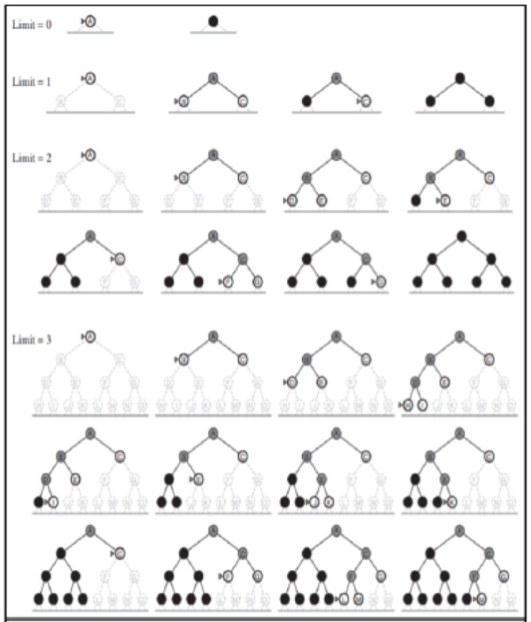


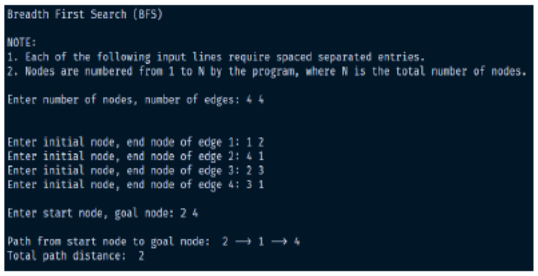
Figure 3.19 Four iterations of iterative deepening search on a binary tree.

(D) Considering the cost associated with every move to be the same (uniform cost), write a function which can backtrack and produce the path taken to reach the goal state from the source/ initial state.

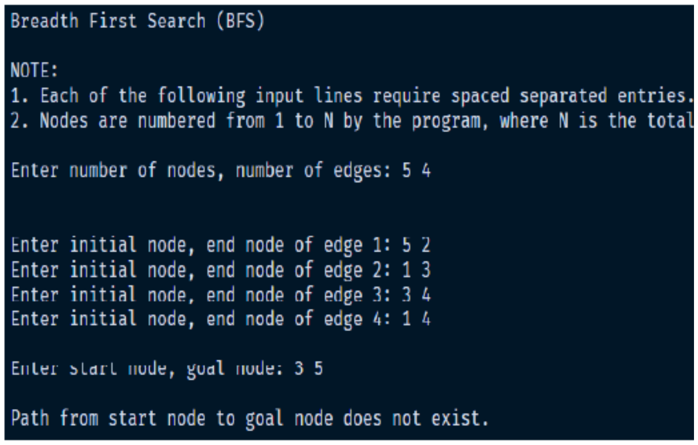
Solution :

(1) Code : Link

(2) Results



```
Breadth First Search (BFS)

NOTE:
1. Each of the following input lines require spaced separated entries.
2. Nodes are numbered from 1 to N by the program, where N is the total number of nodes.

Enter number of nodes, number of edges: 4 4


Enter initial node, end node of edge 1: 1 2
Enter initial node, end node of edge 2: 4 1
Enter initial node, end node of edge 3: 2 3
Enter initial node, end node of edge 4: 3 1

Enter start node, goal node: 2 4

Path from start node to goal node:  2 ⟶ 1 ⟶ 4
Total path distance:  2
```



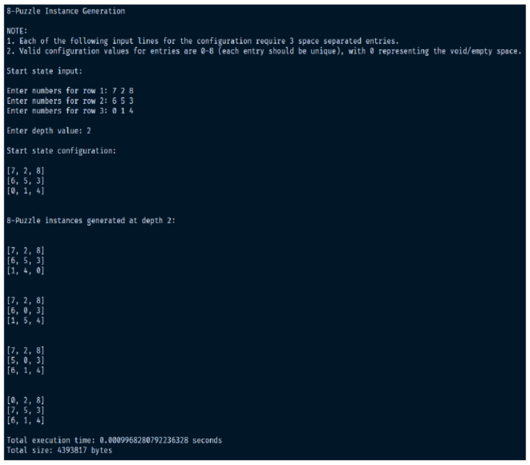
```
Breadth First Search (BFS)

NOTE:
1. Each of the following input lines require spaced separated entries.
2. Nodes are numbered from 1 to N by the program, where N is the total

Enter number of nodes, number of edges: 5 4


Enter initial node, end node of edge 1: 5 2
Enter initial node, end node of edge 2: 1 3
Enter initial node, end node of edge 3: 3 4
Enter initial node, end node of edge 4: 1 4

Enter start node, goal node: 3 5

Path from start node to goal node does not exist.
```

(E) Generate Puzzle-8 instances with the goal state at depth "d".

Solution :

(1) Code : Link

(2) Results

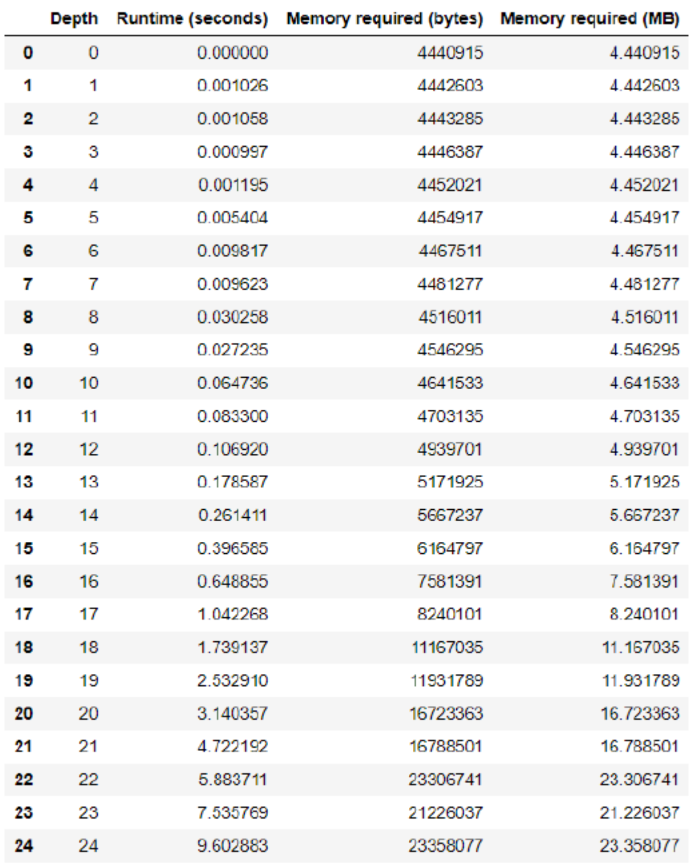


```
8-Puzzle Instance Generation

NOTE:
1. Each of the following input lines for the configuration require 3 space separated entries.
2. Valid configuration values for entries are 0-8 (each entry should be unique), with 0 representing the void/empty space.

Start state input:

Enter numbers for row 1: 7 2 8
Enter numbers for row 2: 6 5 3
Enter numbers for row 3: 0 1 4

Enter depth value: 2

Start state configuration:

[7, 2, 8]
[6, 5, 3]
[0, 1, 4]


8-Puzzle instances generated at depth 2:


[7, 2, 8]
[6, 5, 3]
[1, 4, 0]


[7, 2, 8]
[6, 0, 3]
[1, 5, 4]


[7, 2, 8]
[5, 0, 3]
[6, 1, 4]


[0, 2, 8]
[7, 5, 3]
[6, 1, 4]

Total execution time: 0.0009968280792236328 seconds
Total size: 4393817 bytes
```

(F) Prepare a table indicating the memory and time requirements to solve Puzzle-8 instances (depth "d") using your graph search agent.

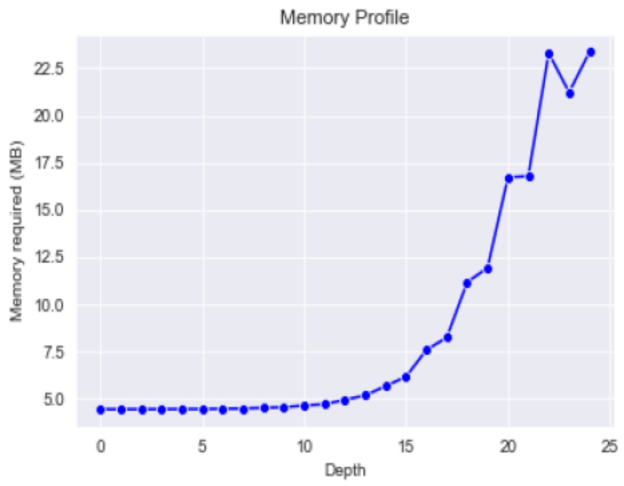
Solution :

(1) Profile table

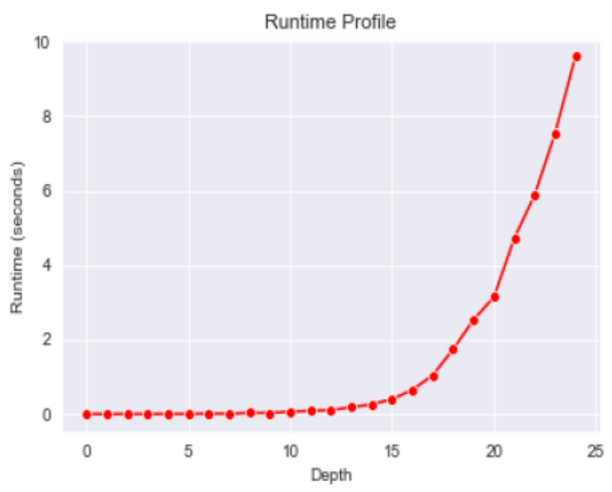


	Depth	Runtime (seconds)	Memory required (bytes)	Memory required (MB)
0	0	0.000000	4440915	4.440915
1	1	0.001026	4442603	4.442603
2	2	0.001058	4443285	4.443285
3	3	0.000997	4446387	4.446387
4	4	0.001195	4452021	4.452021
5	5	0.005404	4454917	4.454917
6	6	0.009817	4467511	4.467511
7	7	0.009623	4481277	4.481277
8	8	0.030258	4516011	4.516011
9	9	0.027235	4546295	4.546295
10	10	0.064736	4641533	4.641533
11	11	0.083300	4703135	4.703135
12	12	0.106920	4939701	4.939701
13	13	0.178587	5171925	5.171925
14	14	0.261411	5667237	5.667237
15	15	0.396585	6164797	6.164797
16	16	0.648855	7581391	7.581391
17	17	1.042268	8240101	8.240101
18	18	1.739137	11167035	11.167035
19	19	2.532910	11931789	11.931789
20	20	3.140357	16723363	16.723363
21	21	4.722192	16788501	16.788501
22	22	5.883711	23306741	23.306741
23	23	7.535769	21226037	21.226037
24	24	9.602883	23358077	23.358077

(2) Memory requirement profile graph



(3) Runtime profile



References for lab assignment 1

- Russell Norvig, 2010, Artificial intelligence: a modern approach textbook
- Iterative deepening notes [http://cseweb.ucsd.edu/elkan/130/itdeep.html]
- Iterative Deepening Search(IDS) or Iterative Deepening Depth First Search(IDDFS)[https://www.geeksforgeeks.org/iterative-deepeningsearchids-iterative-deepening-depth-first-searchiddfs/]

III. Lab Assignment 2

A. Read about the game of marble solitaire. Figure shows the initial board configuration. The goal is to reach the board configuration where only one marble is left at the centre. To solve marble solitaire, (1) Implement priority queue based search considering path cost, (2) suggest two different heuristic functions with justification, (3) Implement best first search algorithm, (4) Implement A*, (5) Compare the results of various search algorithms.

(1) Implement priority queue based search considering path cost.
Code : Link
2) Suggest two different heuristic functions with justification.

There are two distinct heuristic methods:
1. The first heuristic involves calculating the sum of Manhattan distances of all the pegs from the center.
2. The second heuristic calculates the sum of two exponentials of the maximum distance between the peg's horizontal and vertical positions from the center.

(3) Implement best first search algorithm.

1. Best first search Algorithm for first heuristic.
Code : Link

2. Best first search Algorithm for second heuristic.
Code : Link
(4) Implement A*

1. A* for first heuristic
Code : Link

2. A* for second heuristic
Code : Link
(5) Compare the results of various search algorithms

Peg solitaire is np-complete problem.

1. Priority queue-based search involves assigning priority to states based on their depth, which increases with each state explored. Due to this, there can be multiple states with the same depth as the search proceeds. Consequently, the search process can take a significant amount of time, typically around 6 to 8 hours depending on the processing power of the computer.

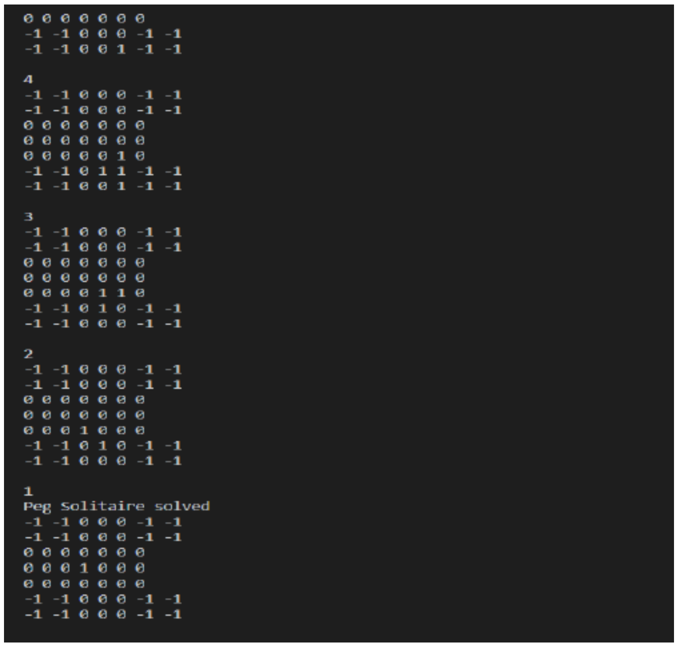
2. The heuristic that calculates the sum of all the peg's Manhattan distances from the center is an improvement over priority queue-based search. However, the heuristic that adds the sum of 2 exponential values based on the maximum horizontal and vertical distance of the peg from the center is a much more effective approach, and we were able to reach the goal state in just 15-20 minutes.

3. Given the heuristic value h(n) and the depth of the peg g(n), A star search algorithm provides the best possible solution.

Solution :

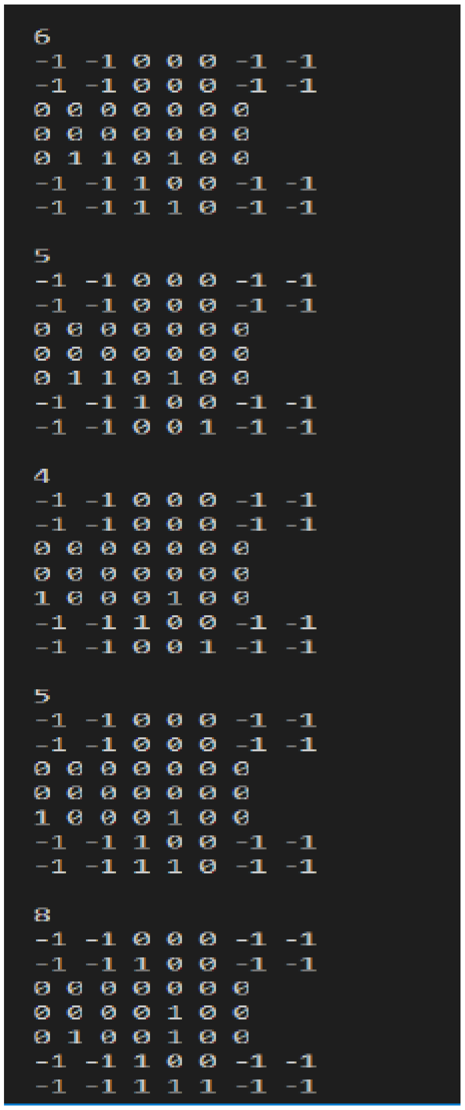
The first image is of the final solution (titled 'Output')

The subsequent images are of the processes 1,2,3,4,5 (process1, process2 ,. . . ..)



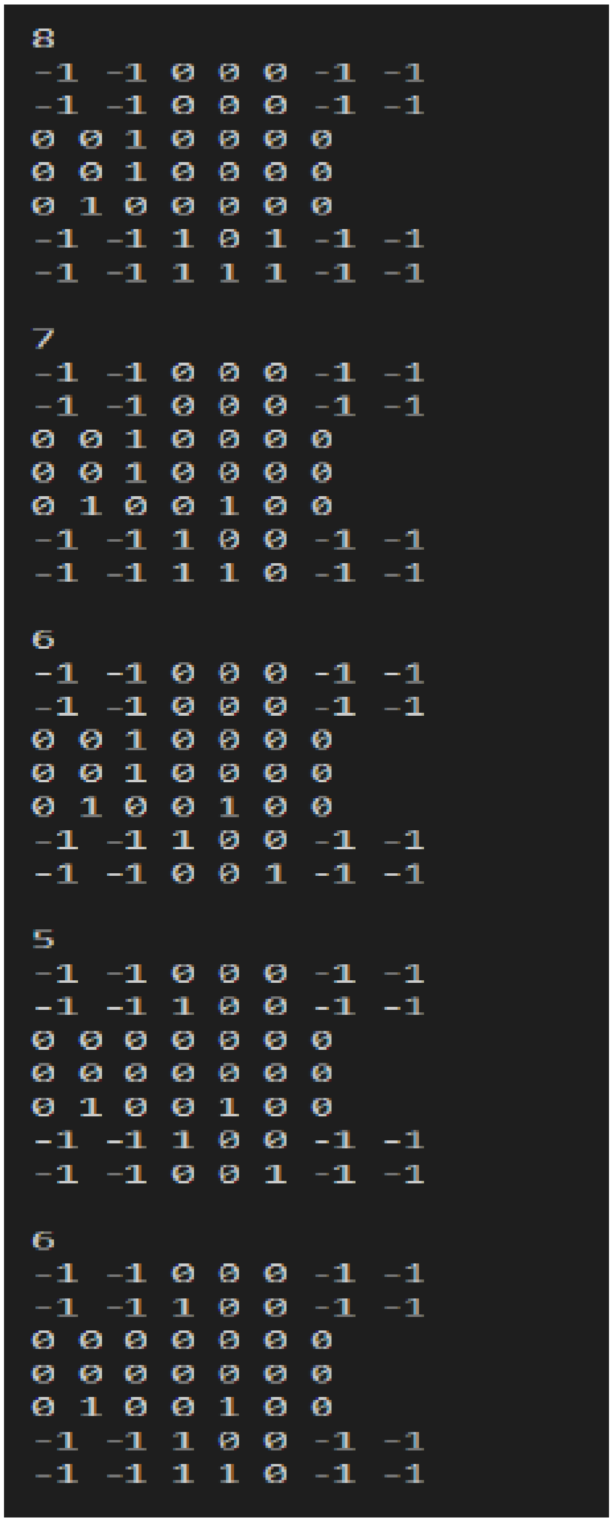
```
0 0 0 0 0 0 0
 -1 -1 0 0 0 -1 -1
 -1 -1 0 0 1 -1 -1

4
 -1 -1 0 0 0 -1 -1
 -1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 1 0
 -1 -1 0 1 1 -1 -1
 -1 -1 0 0 1 -1 -1

3
 -1 -1 0 0 0 -1 -1
 -1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 1 1 0
 -1 -1 0 1 0 -1 -1
 -1 -1 0 0 0 -1 -1

2
 -1 -1 0 0 0 -1 -1
 -1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
 -1 -1 0 1 0 -1 -1
 -1 -1 0 0 0 -1 -1

1
Peg Solitaire solved
 -1 -1 0 0 0 -1 -1
 -1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
 -1 -1 0 0 0 -1 -1
 -1 -1 0 0 0 -1 -1
```

Output



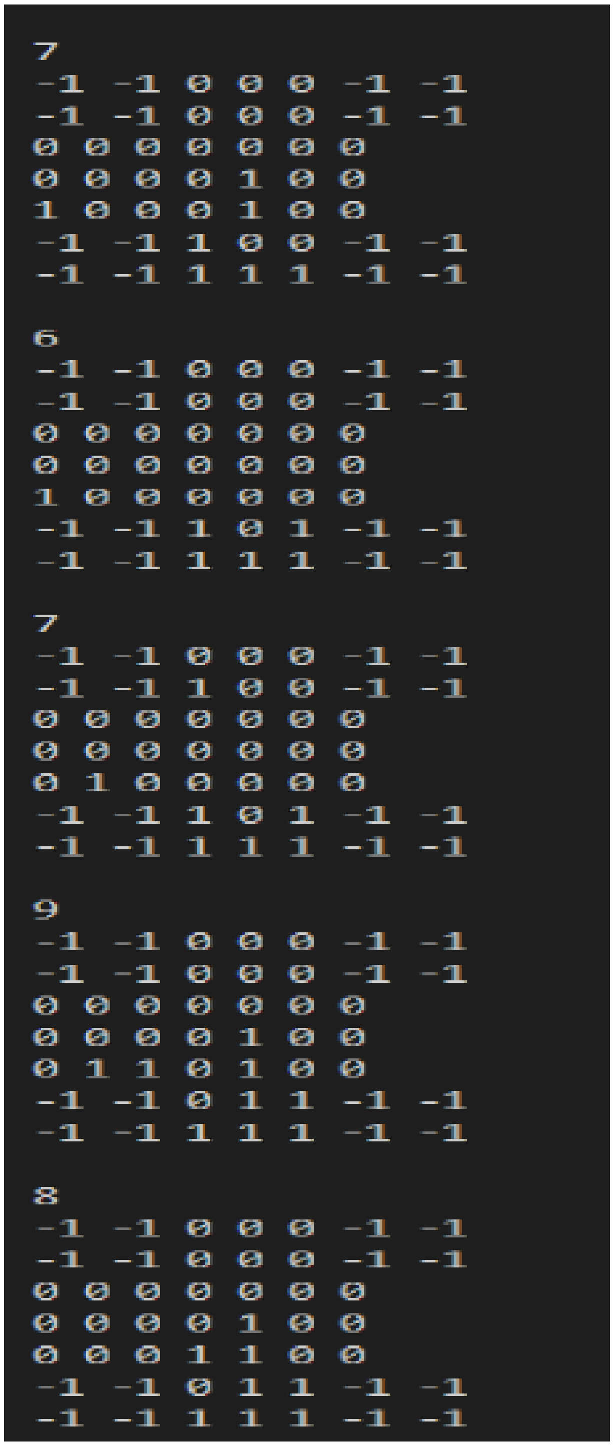
```
6
 -1 -1 0 0 0 -1 -1
 -1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 0 1 0 0
 -1 -1 1 0 0 -1 -1
 -1 -1 1 1 0 -1 -1

5
 -1 -1 0 0 0 -1 -1
 -1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 0 1 0 0
 -1 -1 1 0 0 -1 -1
 -1 -1 0 0 1 -1 -1

4
 -1 -1 0 0 0 -1 -1
 -1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
1 0 0 0 1 0 0
 -1 -1 1 0 0 -1 -1
 -1 -1 0 0 1 -1 -1

5
 -1 -1 0 0 0 -1 -1
 -1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
1 0 0 0 1 0 0
 -1 -1 1 0 0 -1 -1
 -1 -1 1 1 0 -1 -1

8
 -1 -1 0 0 0 -1 -1
 -1 -1 1 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 1 0 0
0 1 0 0 1 0 0
 -1 -1 1 0 0 -1 -1
 -1 -1 1 1 1 -1 -1
```

Process 1



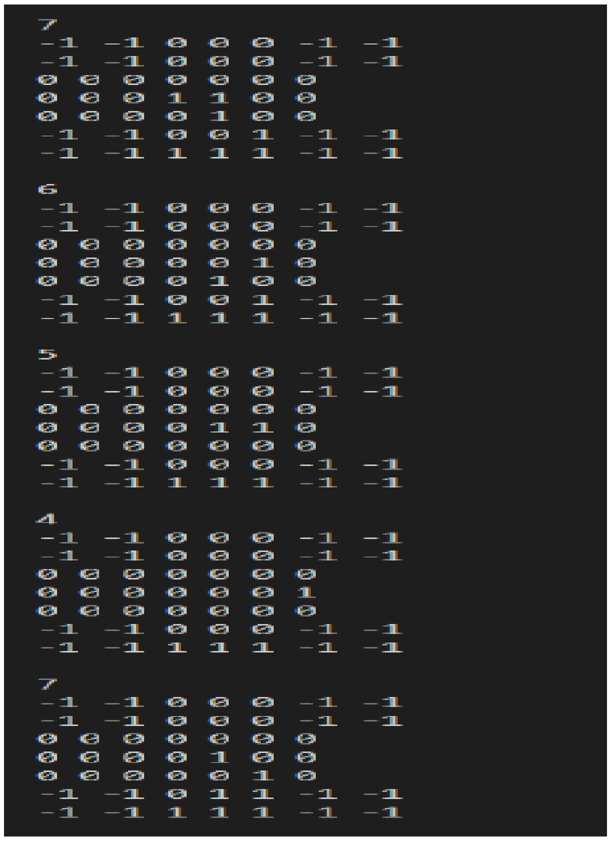
```
8
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 1 0 0 0 0
0 0 1 0 0 0 0
0 1 0 0 0 0 0
-1 -1 1 0 1 -1 -1
-1 -1 1 1 1 -1 -1

7
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 1 0 0 0 0
0 0 1 0 0 0 0
0 1 0 0 1 0 0
-1 -1 1 0 0 -1 -1
-1 -1 1 1 0 -1 -1

6
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 1 0 0 0 0
0 0 1 0 0 0 0
0 1 0 0 1 0 0
-1 -1 1 0 0 -1 -1
-1 -1 0 0 1 -1 -1

5
-1 -1 0 0 0 -1 -1
-1 -1 1 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 0 0 1 0 0
-1 -1 1 0 0 -1 -1
-1 -1 0 0 1 -1 -1

6
-1 -1 0 0 0 -1 -1
-1 -1 1 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 0 0 1 0 0
-1 -1 1 0 0 -1 -1
-1 -1 1 1 0 -1 -1
```

Process 2



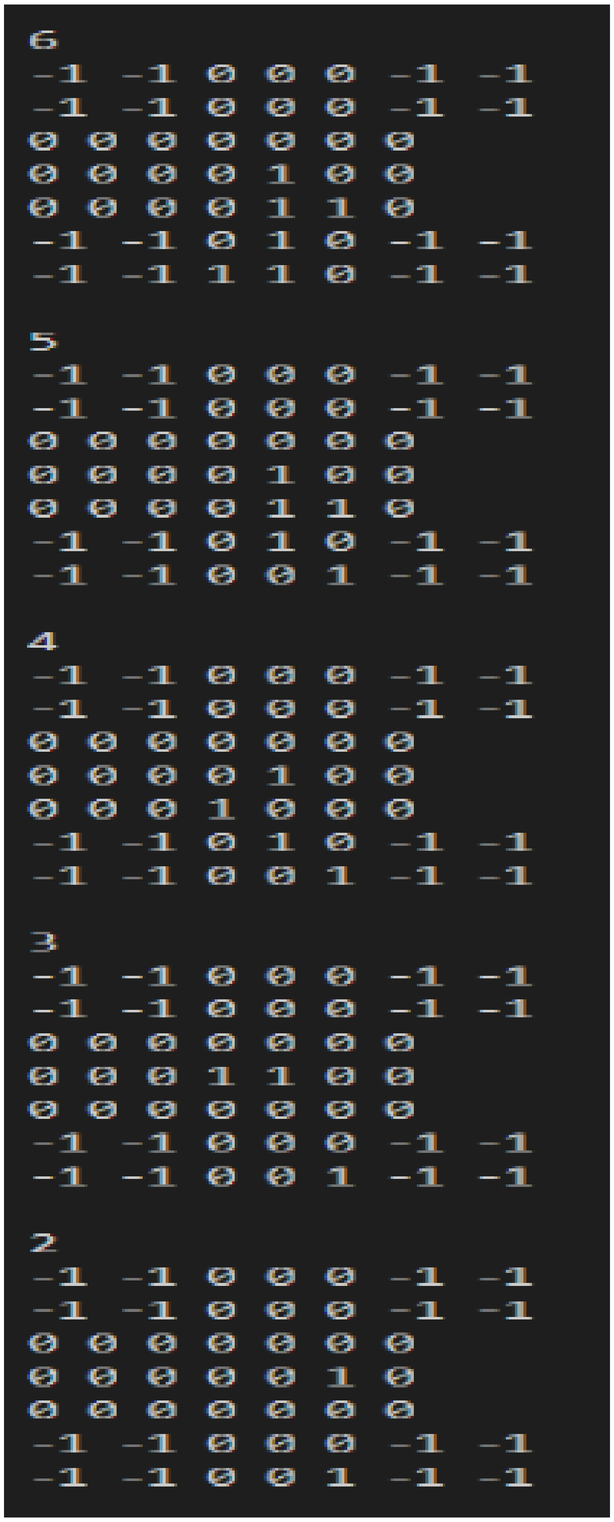
```
7
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 1 0 0
1 0 0 0 1 0 0
-1 -1 1 0 0 -1 -1
-1 -1 1 1 1 -1 -1

6
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
1 0 0 0 0 0 0
-1 -1 1 0 1 -1 -1
-1 -1 1 1 1 -1 -1

7
-1 -1 0 0 0 -1 -1
-1 -1 1 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 0 0 0 0 0
-1 -1 1 0 1 -1 -1
-1 -1 1 1 1 -1 -1

9
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 1 0 0
0 1 1 0 1 0 0
-1 -1 0 1 1 -1 -1
-1 -1 1 1 1 -1 -1

8
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 1 0 0
0 0 0 1 1 0 0
-1 -1 0 1 1 -1 -1
-1 -1 1 1 1 -1 -1
```

Process 3



```
7
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 1 1 0 0
0 0 0 0 1 0 0
-1 -1 0 0 1 -1 -1
-1 -1 1 1 1 -1 -1

6
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 1 0
0 0 0 0 1 0 0
-1 -1 0 0 1 -1 -1
-1 -1 1 1 1 -1 -1

5
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 1 1 0
0 0 0 0 0 0 0
-1 -1 0 0 0 -1 -1
-1 -1 1 1 1 -1 -1

4
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 0 1
0 0 0 0 0 0 0
-1 -1 0 0 0 -1 -1
-1 -1 1 1 1 -1 -1

7
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 1 0 0
0 0 0 0 0 1 0
-1 -1 0 1 1 -1 -1
-1 -1 1 1 1 -1 -1
```

Process 4



```
6
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 1 0 0
0 0 0 0 1 1 0
-1 -1 0 1 0 -1 -1
-1 -1 1 1 0 -1 -1

5
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 1 0 0
0 0 0 0 1 1 0
-1 -1 0 1 0 -1 -1
-1 -1 0 0 1 -1 -1

4
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 1 0 0
0 0 0 1 0 0 0
-1 -1 0 1 0 -1 -1
-1 -1 0 0 1 -1 -1

3
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 1 1 0 0
0 0 0 0 0 0 0
-1 -1 0 0 0 -1 -1
-1 -1 0 0 1 -1 -1

2
-1 -1 0 0 0 -1 -1
-1 -1 0 0 0 -1 -1
0 0 0 0 0 0 0
0 0 0 0 0 1 0
0 0 0 0 0 0 0
-1 -1 0 0 0 -1 -1
-1 -1 0 0 1 -1 -1
```

Process 5

B. Write a program to randomly generate k-SAT problems. The program must accept values for k, m the number of clauses in the formula, and n the number of variables. Each clause of length k must contain distinct variables or their negation. Instances generated by this

algorithm belong to fixed clause length models of SAT and are known as uniform random k-SAT problems.

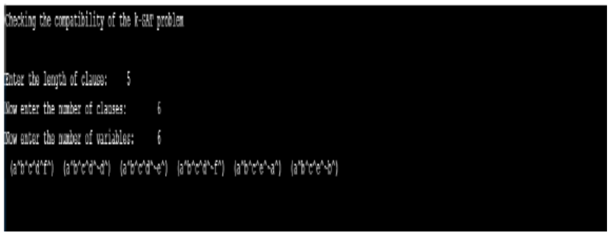
Solution :

1. What is k-SAT?

The Boolean satisfiability problem involves determining if a given Boolean formula can be assigned TRUE or FALSE values for its variables in a consistent way, such that the entire formula evaluates to TRUE. The uniform kSAT model is a popular method for generating random SAT problems, which involves selecting m clauses independently and uniformly.

A k-SAT formula is a set of clauses, each of which is a disjunction of k literals over boolean variables. The k-SAT problem is known to be NP-Complete. The problem is composed of literals, which represent true/false values assigned to variables, and clauses, which are sets of literals combined by disjunction. A formula can have multiple variables and clauses, such as the example formula with five variables (a, b, c, d, e) and six clauses.
Code : Link

3. Results:



```
Checking the compatibility of the k-SAT problem


Enter the length of clause:    5
Now enter the number of clauses:      6
Now enter the number of variables:    6
 (a^b^c^d^f^)  (a^b^c^d^~d^)  (a^b^c^d^~e^)  (a^b^c^d^~f^)  (a^b^c^e^~a^)  (a^b^c^e^~b^)
```

C. Write programs to solve a set of uniform random 3-SAT problems for different combinations of m and n, and compare their performance. Try the Hill-Climbing, Beam-Search with beam widths 3 and 4, Variable-Neighborhood-Descent with 3 neighborhood functions. Use two different heuristic functions and compare them with respect to penetrance.

Solution :

The 3-SAT Problem

The 3-SAT problem, which involves determining if there exists a consistent interpretation of a Boolean formula consisting of 3-literal clauses, is an NP-Complete problem. Literals represent the true-false value assigned to a variable, and clauses are formed by combining literals with disjunction. These clauses are then combined with conjunction to form a CNF problem. If a consistent relation exists in the variable matrix such that at least one case is not covered, then the CNF-problem is considered to be satisfiable. The task is to generate 3-SAT problems with a given number of clauses (m) and variables (n).
Code : Link

2. Approach

The problem has a limited number of literals, which is at most 2 to the power of n. Additionally, the problem assumes that variables are not repeated within a clause, limiting the number of unique literals to n. The number of different clauses that can be generated for this problem is given by the formula (4n/3(n-1)(n-2))Cm, which will provide all the possible clauses. The goal is to find a way to make the output of every clause evaluate to 1.

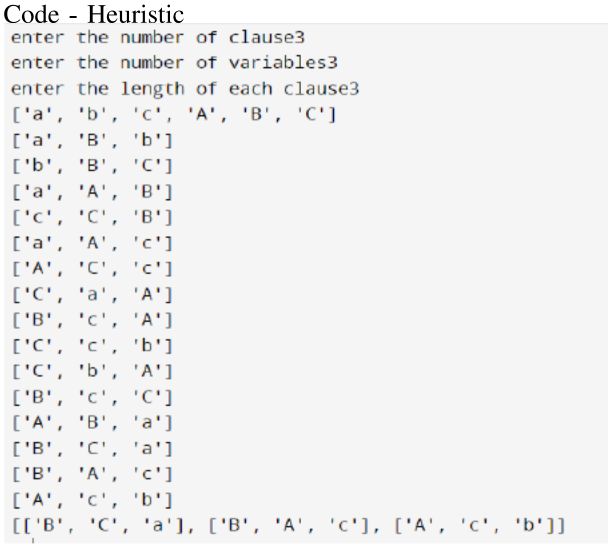
3. Implementation

Heuristic Approach

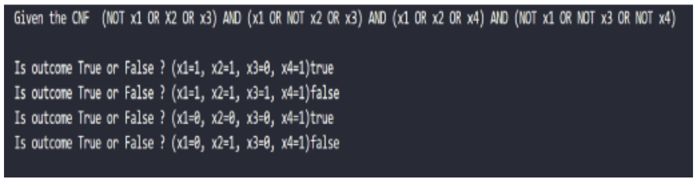
The approach to solve the problem involves creating a tree to represent all possible clauses. Each tree has a root node representing the first literal in the clause. The root node has two possible values: 1 and 0. If it is 1, we find the number of possible nodes below it, and if it is 0, we move on to the next root node and repeat the process. If we reach the node number equal to m, we give it the value 1. This process is repeated for all possible clauses to obtain a 3-SAT. Hill-Climbing search and Beam search are used with priority decided based on Heuristic Function 1.

Code - Hill Climb

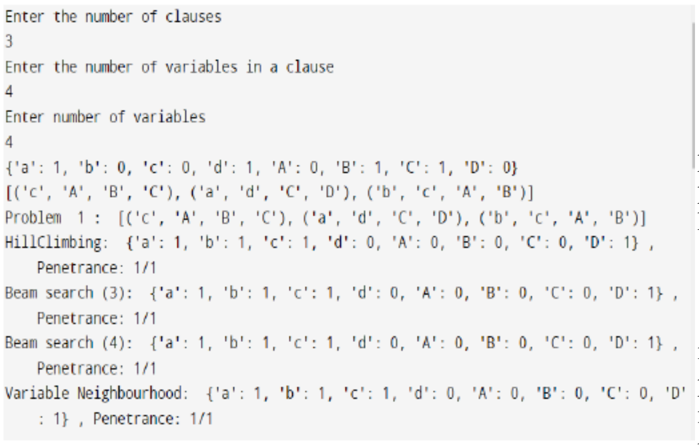
Code - Heuristic



```
enter the number of clause3
enter the number of variables3
enter the length of each clause3
['a', 'b', 'c', 'A', 'B', 'C']
['a', 'B', 'b']
['b', 'B', 'C']
['a', 'A', 'B']
['c', 'C', 'B']
['a', 'A', 'c']
['A', 'C', 'c']
['C', 'a', 'A']
['B', 'c', 'A']
['C', 'c', 'b']
['C', 'b', 'A']
['B', 'c', 'C']
['A', 'B', 'a']
['B', 'C', 'a']
['B', 'A', 'c']
['A', 'c', 'b']
[['B', 'C', 'a'], ['B', 'A', 'c'], ['A', 'c', 'b']]
```



```
Given the CNF  (NOT x1 OR X2 OR x3) AND (x1 OR NOT x2 OR x3) AND (x1 OR x2 OR x4) AND (NOT x1 OR NOT x3 OR NOT x4)

Is outcome True or False ? (x1=1, x2=1, x3=0, x4=1)true
Is outcome True or False ? (x1=1, x2=1, x3=1, x4=1)false
Is outcome True or False ? (x1=0, x2=0, x3=0, x4=1)true
Is outcome True or False ? (x1=0, x2=1, x3=0, x4=1)false
```



```
Enter the number of clauses
3
Enter the number of variables in a clause
4
Enter number of variables
4
{'a': 1, 'b': 0, 'c': 0, 'd': 1, 'A': 0, 'B': 1, 'C': 1, 'D': 0}
[('c', 'A', 'B', 'C'), ('a', 'd', 'C', 'D'), ('b', 'c', 'A', 'B')]
Problem  1 :  [('c', 'A', 'B', 'C'), ('a', 'd', 'C', 'D'), ('b', 'c', 'A', 'B')]
HillClimbing:  {'a': 1, 'b': 1, 'c': 1, 'd': 0, 'A': 0, 'B': 0, 'C': 0, 'D': 1} ,
   Penetrance: 1/1
Beam search (3):  {'a': 1, 'b': 1, 'c': 1, 'd': 0, 'A': 0, 'B': 0, 'C': 0, 'D': 1} ,
   Penetrance: 1/1
Beam search (4):  {'a': 1, 'b': 1, 'c': 1, 'd': 0, 'A': 0, 'B': 0, 'C': 0, 'D': 1} ,
   Penetrance: 1/1
Variable Neighbourhood:  {'a': 1, 'b': 1, 'c': 1, 'd': 0, 'A': 0, 'B': 0, 'C': 0, 'D'
   : 1} , Penetrance: 1/1
```

4. Conclusion:

The time complexity of the problem is 2^n

References for Lab assignment 2

- Peg solitaire [https://en.m.wikipedia.org/wiki/Peg solitaire]
- Russell Norvig, 2010, Artificial intelligence: a modern approach
- Boolean satisfiability problem [https://en.wikipedia.org/wiki/Boolean satisfiability problem/]
- 2-Satisfiability (2-SAT) Problem [https://www.geeksforgeeks.org/2-satisfiability-2-sat-problem/]

IV. Lab Assignment 4

1. What is the size of the game tree for Noughts and Crosses? Sketch the game tree.

Solution:
The no of leave nodes = 9*8*7*6.....*1 = 9! = 362,880

As there are nine possible positions on the board when it is empty where the first move can be made, there are eight positions with respect to these nine moves where the second move can be made, and so on.

For d=1 there will be 9 nodes,for d = 2 there will be 9*8 nodes for d=3 there will be 9*8*7 nodes and so on.

So,total no of nodes = P9!/(9d)! where d = 1 to 9 and this will also be the size of the tree.

Link for Game Tree : Click on this Text

2. Read about the game of Nim (a player left with no move losing the game). For the initial configuration of the game with three piles of objects as shown in Figure, show that regardless of the strategy of player-1, player-2 will always win. Try to explain the reason with the MINIMAX value backup argument on the game tree.

Solution:

If and only if nim sum = 0 (XOR of all three piles) and the player whose turn it is never makes a mistake, that is, never permits the opponent player to make nim sum = 0, then we know that it is a losing pose for that player.

So, for the first move, Player 1's best course of action is to make Nim Sum equal to 0, which is achieved by creating the piles [10,3,9]. After that, player 2 won't be able to make the nim sum zero again, therefore whatever action they take will result in a non-zero nim sum. When player 1's turn comes around, that player can once more perform the optimal move, which is to make the nim sum 0. If player 1 consistently makes a negative sum, regardless of the second player's move, they will win every time.

The nim game's game tree is illustrated below; we were unable to render the entire tree, so we only displayed a portion of it. Player 1 always takes the max value move, which is the move where the nim sum is 0 (since it has max value), while Player 2 is doing its appropriate move (we have shown only some of its move) It is evident from the figure that player 2 cannot win no matter what plays are made if player 1 has chosen the best course of action.

So, the question that is used to demonstrate that player 2 will win regardless of player 1's action is incorrect. Instead, it should be stated that Player 1 will prevail despite any action taken by Player 2.

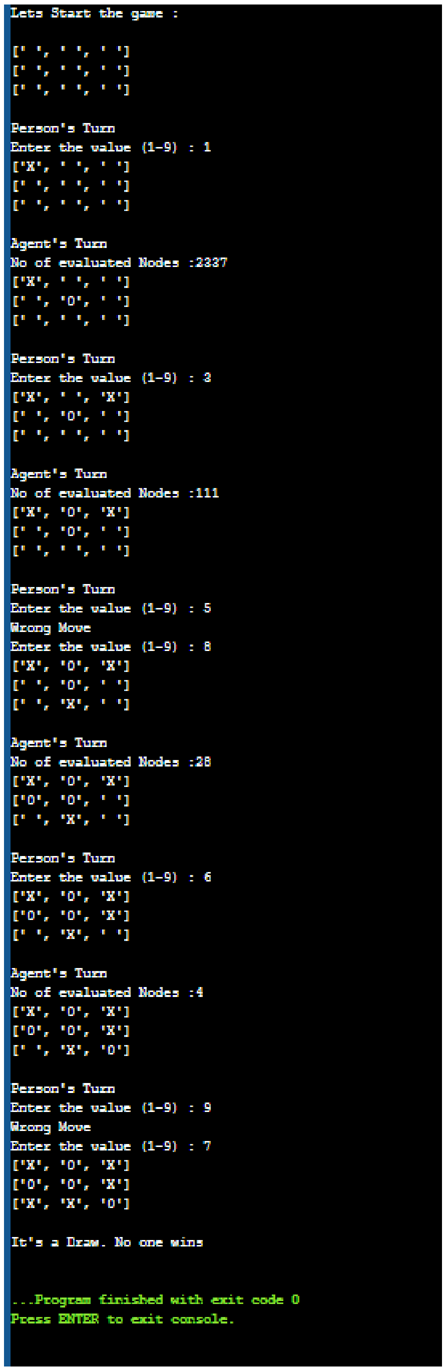
Player 2 will always prevail if player 1 chooses to make any other move, such as [10,1,9], and player 2 chooses to make [8,1,9] in response. Consequently, we get the conclusion that player 2 has no chance of winning if player 1 makes the optimal move first.

Link for the Nim Tree : **Link for Game Tree : Click on this Text**

3. Implement MINIMAX and alpha-beta pruning agents. Report on number of evaluated nodes for Noughts and Crosses game tree.

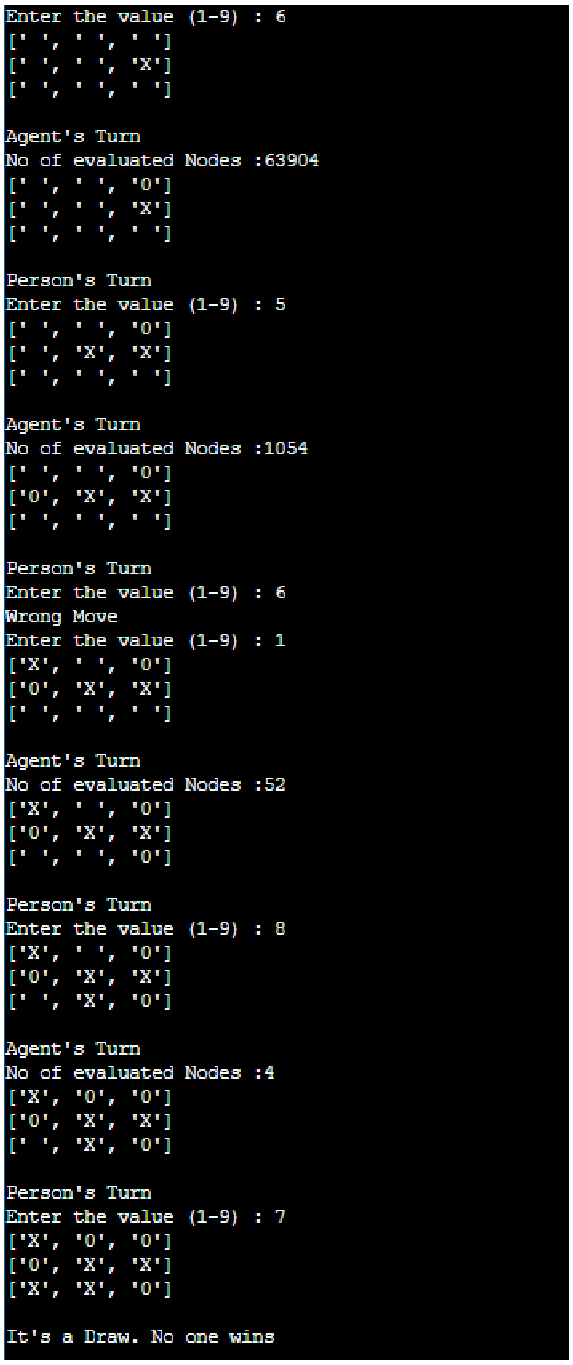
Solution:

1) **Codes**
   - **MinMax :** Link
   - **AlphaBeta :** Link
2) **Results**



```
Lets Start the game :

[' ', ' ', ' ']
[' ', ' ', ' ']
[' ', ' ', ' ']

Person's Turn
Enter the value (1-9) : 1
['X', ' ', ' ']
[' ', ' ', ' ']
[' ', ' ', ' ']

Agent's Turn
No of evaluated Nodes :2337
['X', ' ', ' ']
[' ', '0', ' ']
[' ', ' ', ' ']

Person's Turn
Enter the value (1-9) : 3
['X', ' ', 'X']
[' ', '0', ' ']
[' ', ' ', ' ']

Agent's Turn
No of evaluated Nodes :111
['X', '0', 'X']
[' ', '0', ' ']
[' ', ' ', ' ']

Person's Turn
Enter the value (1-9) : 5
Wrong Move
Enter the value (1-9) : 8
['X', '0', 'X']
[' ', '0', ' ']
[' ', 'X', ' ']

Agent's Turn
No of evaluated Nodes :28
['X', '0', 'X']
['0', '0', ' ']
[' ', 'X', ' ']

Person's Turn
Enter the value (1-9) : 6
['X', '0', 'X']
['0', '0', 'X']
[' ', 'X', ' ']

Agent's Turn
No of evaluated Nodes :4
['X', '0', 'X']
['0', '0', 'X']
[' ', 'X', '0']

Person's Turn
Enter the value (1-9) : 9
Wrong Move
Enter the value (1-9) : 7
['X', '0', 'X']
['0', '0', 'X']
['X', 'X', '0']

It's a Draw. No one wins


...Program finished with exit code 0
Press ENTER to exit console.
```

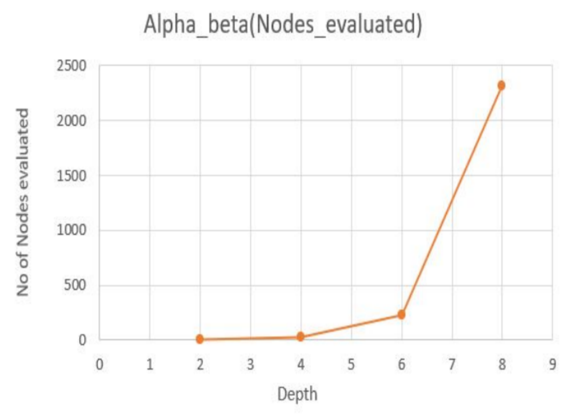
Result by using MinMax

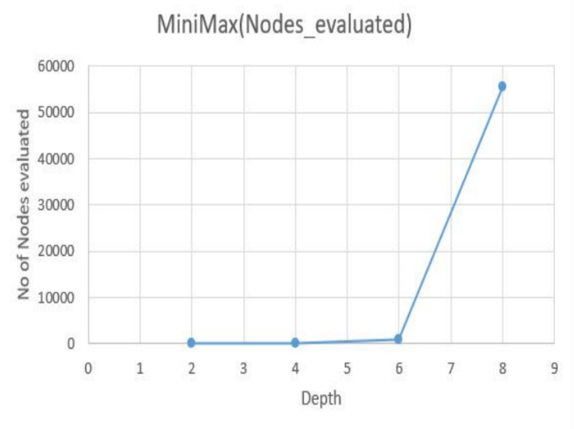


```
Enter the value (1-9) : 6
[' ', ' ', ' ']
[' ', ' ', 'X']
[' ', ' ', ' ']

Agent's Turn
No of evaluated Nodes :63904
[' ', ' ', '0']
[' ', ' ', 'X']
[' ', ' ', ' ']

Person's Turn
Enter the value (1-9) : 5
[' ', ' ', '0']
[' ', 'X', 'X']
[' ', ' ', ' ']

Agent's Turn
No of evaluated Nodes :1054
[' ', ' ', '0']
['0', 'X', 'X']
[' ', ' ', ' ']

Person's Turn
Enter the value (1-9) : 6
Wrong Move
Enter the value (1-9) : 1
['X', ' ', '0']
['0', 'X', 'X']
[' ', ' ', ' ']

Agent's Turn
No of evaluated Nodes :52
['X', ' ', '0']
['0', 'X', 'X']
[' ', ' ', '0']

Person's Turn
Enter the value (1-9) : 8
['X', ' ', '0']
['0', 'X', 'X']
[' ', 'X', '0']

Agent's Turn
No of evaluated Nodes :4
['X', '0', '0']
['0', 'X', 'X']
[' ', 'X', '0']

Person's Turn
Enter the value (1-9) : 7
['X', '0', '0']
['0', 'X', 'X']
['X', 'X', '0']

It's a Draw. No one wins
```

Result by using Alpha Beta pruning

Depth	MiniMax(Nodes_evaluated)	Alpha_beta(Nodes_evaluated)
8	55504	2315
6	932	229
4	50	32
2	4	4





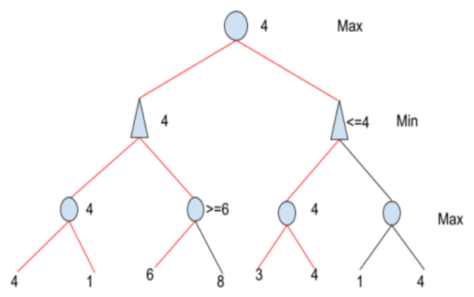
It is evident from the fact that more nodes are evaluated in minimax than in alpha beta pruning.

4. Using recurrence relation show that under perfect ordering of leaf nodes, the alpha-beta pruning time complexity is

$$O(b^{m/2})$$

where b is the effective branching factor and m is the depth of the tree.

Solution:
Let's take an example of best case of alpha beta here. End line denotes the visited nodes. Here branching factor is 2 an depth is 3.



From the figure it is obvious that either we know the precise value of the state or its bound. We require the precise value of one of a state's children as well as bounds on the other children in order to calculate the exact value of the state. We require the precise value of one of a state's children in order to set a bound on its value. Let S(k) be the minimum number of states that must be taken into account when determining the precise value of a state based on these observations. Similar to this, when we need to know a bound on the value of a state, let R(k) be the smallest number of states to be taken into account k ply from the state.

Similarly, let R(k) be the minimum number of states to be considered k ply from a given state when we need to know a bound on the state's value.

Thus, we have:

$$S(k) = S(k-1) + (b-1)R(k-1)$$
The exact value of one child and bound on others, Where
$$R(k-1) = S(k-1)$$
i.e. the exact value of one child, after expanding we get
$$S(k) = S(k-1) + (b-1)R(k-1)$$
$$= (S(k-2) + (b-1)R(k-2)) + (b-1)R(k-1)$$
$$= bS(k-2) + (b-1)R(k-2)$$
$$= bS(k-2) + (b-1)S(k-3)$$
This gives,
$$S(2) = b^2 + b - 1 = 5 for b = 2$$
$$\text{so : } S(k) < (2b-1)S(k-2) < 2bS(k-2)$$

That is, the branching factor every two levels is less than 2b, which means the effective branching factor is less than $\sqrt{2}b$. In effect, alpha-beta pruning can nearly search double the minimax search at a same times Thus
For best case the alpha-beta pruning time complexity is O(bm=2)

Conclusion:

We can draw the conclusion from the lab that the Minimax algorithm and alpha beta pruning are useful for playing games. Nevertheless, because Minimax assesses every node, it becomes highly challenging to solve games with greater depth and branching factors, like chess. The alpha beta pruning technique is helpful in this situation because it is optimal and does not analyze every node, as is also evident from the table that is supplied above. Minimax requires more time than alpha-beta pruning.

We also proved that time complexity for alpha beta is O(bm=2) that is it can search double the minimax search at the same time.

References for Lab Assignment 4

- - Mastering Tic-Tac-Toe with Minimax Algorithm in Python [https://levelup.gitconnected.com/mastering-tic-tac-toe-with-minimax-algorithm-3394d65fa88f]
- - Alpha Beta Pruning with Example [https://www.youtube.com/watch?v=dEs kbvu 0slist= PLxCzCOWd7aiHGhOHV-nwb0HR5US5GFKFIindex=20]
- - Minimax Algorithm in Game Playing [https: //www.youtube.com/watch?v=Ntu8nNBL28olist= PLxCzCOWd7aiHGhOHV-nwb0HR5US5GFKFI index=19]
- - Complexity analysis of alpha beta pruning of a full tree [https://math.stackexchange.com/questions/1471238/ complexity-analysis-of-alpha-beta-pruning-of-a-full-tree]