Mehboob Ali(17L-4316)

Hamza Arshad(17L-4001)

Artificial Intelligence

Section E

Introduction

Our main goal in this assignment is to compare the completeness and optimality of two different search strategies, i.e. Iterating Deepening (Uninformed Search Strategy) and A* Search (Informed Search Strategy) by solving a scrambled Rubik's cube.

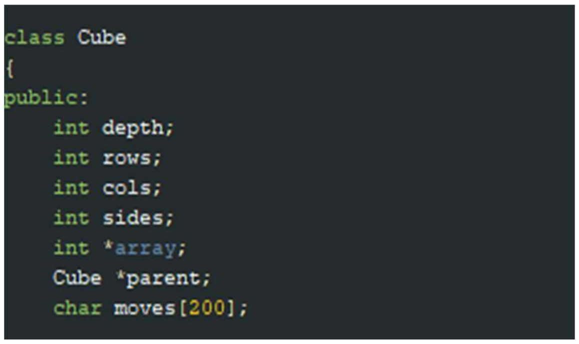
Completeness means that the search strategy is able to find the solution if it EXISTS. Whereas, optimality refers to the solution being low-cost or in terms of the Rubik's cube, minimum number of moves required to solve it.

Methodology

Moves: Front Clockwise, Front Anti-Clockwise, Up Clockwise, Up Anti-Clockwise, Down Clockwise, Down Anti-Clockwise, Left Clockwise, Left Anti-Clockwise, Right Clockwise, Right Anti-Clockwise, Back Clockwise, Back Anti-Clockwise. (12)

Iterative Deepening

Cube State:



```
class Cube
{
public:
    int depth;
    int rows;
    int cols;
    int sides;
    int *array;
    Cube *parent;
    char moves[200];
```

- Array is used to store current configuration of the cube.
- Parent pointer is required to backtrack from the goal to the root node.

Iterative Deepening has been implemented iteratively. There is one wrapper function through which the main DLDFS (Depth Limited Depth First Search) function is called. The depth is incremented each time the function is called and if the goal is not found.

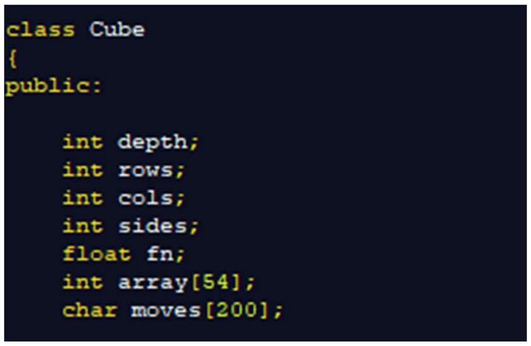
In DLDFS function, OPEN (frontier) stack is used to store the nodes (states) which are yet to be explored while the CLOSED stack is used to keep track of the nodes which have already been visited (extra bookkeeping).

It works in the following way:

- Pop a node from open stack, push it into the closed stack.
- Compare with goal node (overloaded operator), if true; break.
- Else, apply moves (12) on the node and push the successors into the open stack.

A* Search

Cube State:



```
class Cube
{
public:

    int depth;
    int rows;
    int cols;
    int sides;
    float fn;
    int array[54];
    char moves[200];
```

- fn variable is used to store the sum of the gn (step-cost) and hn (heuristic value) of the cube.
- All the remaining variables are the same.

The heart of the A* Search is the heuristic. It works efficiently and optimally as long as the heuristic is admissible and consistent. For the A* Search, we have used two heuristics.

1) Number of misplaced nodes (cubies) which is also known as the hamming distance.
2) 3D Manhattan Distance. It calculates the steps required to move a cubie to its goal state. For it to be admissible, it has to be divided by 8 as only the edges and corners of a particular face are moving at any given time.
3) Additionally, these two have been used in combination to check their effectiveness and optimality.

The main algorithm for A* is similar to that of IDFS. However, priority queue (STL) has been as the frontier instead of stack. Nodes have been prioritized based on fn value (ascending order). The smaller the fn value, the closer to the goal we get.

If a node has already been expanded but gets discovered again with a lower fn value, it is removed from the closed stack and placed in the frontier. Similarly, if a node in the frontier which is yet to be expanded is found again but with a smaller fn value, gets replaced with the previous one with a higher fn value.

Results

Legend: h1 – 3D Manhattan Distance, h2 – Hamming Distance, IDFS (Iterative Deepening First Search)

Note: Same scrambled cube was used for each experiment.

Depth of goal (d)	Search cost (nodes generated)				Time consumed (seconds)			
	IDFS	A* (h1)	A* (h2)	A* (h1+ h2)	IDFS	A* (h1)	A* (h2)	A*(h1+h2)
1	11	1	1	1	<1 (0.003)	<1(0.001)	<1(0.001)	<1(0.001)
2	71	2	2	2	<1 (0.006)	<1(0.01)	<1(0.0034)	<1(0.003)
3	1297	12	39	157	<1 (0.06)	<1(0.111)	<1(0.147)	2.067
4	7752	92	813	-	2.412	<1(0.287)	52.162	-
5	100005	561	-	-	700.693	61.923	-	-
6	127244	1464	-	-	1089.607	168.919	-	-

For IDFS, nodes generated and time consumed increases drastically as the depth increases. However, it is complete (finds a solution) and optimal (finds shortest path to goal with backtracking).

For A* search, results vary based on the admissibility and consistency of the heuristic. After going through the experiments, it could be seen that h1 dominates h2 and (h1+h2) on any node. This domination accentuates the fact it is a better heuristic in terms of completeness, efficiency and optimality.

However, there are other heuristics like the pattern database which is the most efficient amongst all the heuristics for a Rubik's cube.

Conclusion

Comparing the A* search on the basis of different heuristics with iterative deepening on different depths (nodes expanded and time to reach goal state), it could be concluded that A* search algorithm consumes much less time and memory than Iterative Deepening First Search. However, the completeness, optimality and efficiency of the A* Search is dependent on the admissibility of the heuristic used.