0616066_HW1 Report

Usage:
g++ 0616066_hw1.cpp
./a.out “algorithm type” “input puzzle file” “output solution file”
Algorithm type: bfs / dfs / ids / astar / idastar

Algorithm implementation:

BFS:
Use FIFO queue to store the board state,
Use FIFO queue to store “path” to each board state,
The data structure of path is vector<board>,
Use map<int(depth), int(count)> to store how many node is created at each depth.
Use vector<board> discovered to store board that is already discovered,
If a board state is already discovered, it’s “next move” will not be considered and will not be push into the queue.
Loop till the “board state” is empty or found solution.

DFS:
Use FILO stack to store the board state,
Use FILO stack to store “path” to each board state,
The data structure of path is vector<board>,
Use map<int(depth), int(count)> to store how many node is created at each depth.
Use vector<board> discovered to store board that is already discovered,
If a board state is already discovered, it’s “next move” will not be considered and will not be push into the queue.
Loop till the “board state” is empty or found solution.

IDS:
Use limit_depth to limit the tree depth, if solution didn’t sound in this depth limit, limit_depth++.
Use FILO stack to store the board state,
Use FILO stack to store “path” to each board state,
The data structure of path is vector<board>,
Use map<int(depth), int(count)> to store how many node is created at each depth.
Use vector<board> discovered to store board that is already discovered,
If a board state is already discovered, it’s “next move” will not be considered and will not be push into the queue.
Loop till the “board state” is empty or found solution.

A*:
Use priority queue to store the board state,
Use priority queue to store “path” to each board state,
Use “the number of cars directly blocking the way of the red car to the exit. ” as Heuristics function.
The data structure of path is vector<board>,

Use map<int(depth), int(count)> to store how many node is created at each depth.
Use vector<board> discovered to store board that is already discovered,
If a board state is already discovered, it's "next move" will not be considered and will not be push into the queue.
Loop till the "board state" is empty or found solution.

IDA*:
Use limit_depth to limit the tree depth, if solution didn't sound in this depth limit, limit_depth++.
Use priority queue to store the board state,
Use priority queue to store "path" to each board state,
Use "the number of cars directly blocking the way of the red car to the exit. " as Heuristics function.
The data structure of path is vector<board>,
Use map<int(depth), int(count)> to store how many node is created at each depth.
Use vector<board> discovered to store board that is already discovered,
If a board state is already discovered, it's "next move" will not be considered and will not be push into the queue.
Loop till the "board state" is empty or found solution.

Experiment result:
BFS:

```
(base) ryan@Ryande-MBP18 HW1 % ./a.out bfs ./prog1_puzzle/L01.txt L01_bfs.txt

STARTING BOARD STATE:

 O1 O1 _  _  _  O7
 O4 _  _  O6 _  O7
 O4 X  X  O6 _  O7
 O4 _  _  O6 _  _
 O5 _  _  _  O2 O2
 O5 _  O3 O3 O3 _

The solution file L01_bfs.txt is created.
TOTAL CYCLES:0
MOVE # OF THE SOLUTION:16
STEP WHEN FOUND FIRST SOLUTION:7329
TOTAL NODES GENERATED:1072
MAXIMUM # OF NODES KEPT IN MEMORY:7329
TOTAL bfs TIME:0.826296 seconds
```

DFS:

```
(base) ryan@Ryande-MBP18 HW1 % ./a.out dfs ./prog1_puzzle/L01.txt L01_dfs.txt

STARTING BOARD STATE:

 O1 O1 _  _  _  O7
 O4 _  _  O6 _  O7
 O4 X  X  O6 _  O7
 O4 _  _  O6 _  _
 O5 _  _  _  O2 O2
 O5 _  O3 O3 O3 _

The solution sequence file L01_dfs.txt is created.
TOTAL CYCLES:0
MOVE # OF THE SOLUTION:845
STEP WHEN FOUND FIRST SOLUTION:1849
TOTAL NODES GENERATED:900
MAXIMUM # OF NODES KEPT IN MEMORY:1849
TOTAL dfs TIME:3.08591 seconds
```

IDS:

```
(base) ryan@Ryande-MBP18 HW1 % ./a.out ids ./prog1_puzzle/L01.txt L01_ids.txt

STARTING BOARD STATE:

 01 01 _  _  _  07
 04 _  _  06 _  07
 04 X  X  06 _  07
 04 _  _  06 _  _
 05 _  _  _  02 02
 05 _  03 03 03 _

The solution sequence file L01_ids.txt is created.
TOTAL CYCLES:0
MOVE # OF THE SOLUTION:48
STEP WHEN FOUND FIRST SOLUTION:3459
TOTAL NODES GENERATED:960
MAXIMUM # OF NODES KEPT IN MEMORY:79082
TOTAL ids TIME:7.31353 seconds
```

A*:

```
(base) ryan@Ryande-MBP18 HW1 % ./a.out astar ./prog1_puzzle/L01.txt L01_astar.txt

STARTING BOARD STATE:

 01 01 _  _  _  07
 04 _  _  06 _  07
 04 X  X  06 _  07
 04 _  _  06 _  _
 05 _  _  _  02 02
 05 _  03 03 03 _

The solution sequence file L01_astar.txt is created.
TOTAL CYCLES:0
MOVE # OF THE SOLUTION:18
STEP WHEN FOUND FIRST SOLUTION:133
TOTAL NODES GENERATED:75
MAXIMUM # OF NODES KEPT IN MEMORY:133
TOTAL astar TIME:0.027216 seconds
```

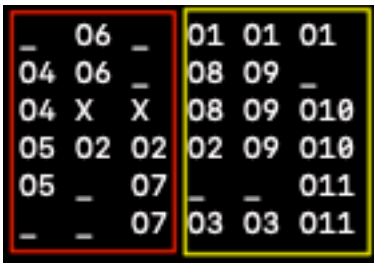
IDA*:

```
(base) ryan@Ryande-MBP18 HW1 % ./a.out idastar ./prog1_puzzle/L01.txt L01_idastar.txt

STARTING BOARD STATE:

 01 01 _  _  _  07
 04 _  _  06 _  07
 04 X  X  06 _  07
 04 _  _  06 _  _
 05 _  _  _  02 02
 05 _  03 03 03 _

The solution sequence file L01_idastar.txt is created.
TOTAL CYCLES:0
MOVE # OF THE SHORTEST SOLUTION:18
STEP WHEN FOUND FIRST SOLUTION:133
TOTAL NODES GENERATED:75
MAXIMUM # OF NODES KEPT IN MEMORY:32410
TOTAL idastar TIME:5.27065 seconds
```

Algorithm compare:
We can see that A* algorithm is the fastest way to find solution, and DFS has the largest “moves # of solution”, both of them seem reasonable.

However IDS and IDA* take long time to execute is because I initial limit_depth=1, and every time complete the tree but have't found solution, I increase limit_depth and clear the tree, simply rebuild tree from beginning. So there are many tree were being build again and again, the better way is to build tree from previous stage.

I have design another Heuristics function. The main idea of the function is let "left side of the target vehicle's board tighter " and "right side of target vehicle's board sparser".
For example (X is the target vehicle):



Wishing red box tighter and yellow box tighter.
Heuristics function = (right_count / right_area) / (left_count/left_area)
In this case: (12/18) / (15 / 18)
However, the performance of this Heuristics function is not good as previous one, but is faster than bfs.
Original Heuristics function: 4.09 sec

```
(base) ryan@Ryande-MBP18 HW1 % ./a.out astar ./prog1_puzzle/L26.txt L26_astar.txt

STARTING BOARD STATE:

 _  06 _  01 01 01
 04 06 _  08 09 _
 04 X  X  08 09 010
 05 02 02 02 09 010
 05 _  07 _  _  011
 _  _  07 03 03 011

The solution sequence file L26_astar.txt is created.
TOTAL CYCLES:0
MOVE # OF THE SOLUTION:98
STEP WHEN FOUND FIRST SOLUTION:9272
TOTAL NODES GENERATED:1929
MAXIMUM # OF NODES KEPT IN MEMORY:9272
TOTAL astar TIME:4.09384 seconds
```

My Heuristics function: 7.51 sec

```
(base) ryan@Ryande-MBP18 HW1 % ./a.out astar ./prog1_puzzle/L26.txt L26_astarh2.txt

STARTING BOARD STATE:

 _  06 _  01 01 01
 04 06 _  08 09 _
 04 X  X  08 09 010
 05 02 02 02 09 010
 05 _  07 _  _  011
 _  _  07 03 03 011

The solution sequence file L26_astarh2.txt is created.
TOTAL CYCLES:0
MOVE # OF THE SOLUTION:181
STEP WHEN FOUND FIRST SOLUTION:6125
TOTAL NODES GENERATED:2328
MAXIMUM # OF NODES KEPT IN MEMORY:6125
TOTAL astar TIME:7.52195 seconds
```

Code: https://github.com/ryanlu2240/NCTU_AI_2021_spring/blob/master/HW1/0616066_HW1.cpp