**CSCE 625 – PROJECT #1**

Geethik Narayana Kamineni

UIN – 825008990

**Instructions for running the Blocks world puzzle solver**

Approach of code:

State.java is used to store the state of the nodes as list of lists. Node.java has the characteristic properties of a node as in the State, depth, scores of f, g and h.

ProblemGenerator.java is used to generate the initial states and the goal states. The initial states are generated both randomly and in a custom way as in the functions randomState and customState in it.

Solver.java is the driver program which calls the A\* search on the given initial nodes along with specified heuristics. In the case of node generation, the successor function is used which generates all the nodes which are not visited and not in frontier with less depth. Nodes are marked visited at the time of generation itself to prevent the queue for expanding rapidly. Traceback function is used for printing back the solution.

Compiling:

javac State.java

javac Node.java

javac ProblemGenerator.java

javac Sovler.java

Running:

java Solver

**Example program traces**

**Trace 1**

Enter number of stacks:

3

Enter number of Blocks:

5

Initial State:

1 | A B

2 | D

3 | C E

Goal State:

1 | A B C D E

2 |

3 |

Enter Heuristic choice:

Choose for heuristics from below:

Press 0 for Blocks out of place

Press 1 for Blocks counted twice in stack one

Press any other key to exit

0

iter\_count: 1

iter\_count: 2

iter\_count: 3

….

iter\_count: 16

iter\_count: 17

iter\_count: 18

Success!!

Solution:

1 | A B

2 | D

3 | C E

1 | A B

2 | D E

3 | C

1 | A B C

2 | D E

3 |

1 | A B C

2 | D

3 | E

1 | A B C D

2 |

3 | E

1 | A B C D E

2 |

3 |

Depth: 5 Goal\_tests: 18 Max\_Queue\_size: 32

Enter Heuristic choice:

Choose for heuristics from below:

Press 0 for Blocks out of place

Press 1 for Blocks counted twice in stack one

Press any other key to exit

1

iter\_count: 1

iter\_count: 2

….

iter\_count: 6

iter\_count: 7

Success!!

Solution:

1 | A B

2 | D

3 | C E

1 | A B

2 | D E

3 | C

1 | A B C

2 | D E

3 |

1 | A B C

2 | D

3 | E

1 | A B C D

2 |

3 | E

1 | A B C D E

2 |

3 |

Depth: 5 Goal\_tests: 7 Max\_Queue\_size: 17

**Trace 2**

Enter number of stacks:

4

Enter number of Blocks:

6

Initial State:

1 | F

2 | D

3 | A B C E

4 |

Goal State:

1 | A B C D E F

2 |

3 |

4 |

Enter Heuristic choice:

Choose for heuristics from below:

Press 0 for Blocks out of place

Press 1 for Blocks counted twice in stack one

Press any other key to exit

0

iter\_count: 1

iter\_count: 2

iter\_count: 3

……

iter\_count: 992

iter\_count: 993

iter\_count: 994

iter\_count: 995

Success!!

Solution:

1 | F

2 | D

3 | A B C E

4 |

1 |

2 | D

3 | A B C E

4 | F

1 |

2 | D

3 | A B C

4 | F E

1 |

2 | D C

3 | A B

4 | F E

1 |

2 | D C B

3 | A

4 | F E

1 | A

2 | D C B

3 |

4 | F E

1 | A B

2 | D C

3 |

4 | F E

1 | A B C

2 | D

3 |

4 | F E

1 | A B C D

2 |

3 |

4 | F E

1 | A B C D E

2 |

3 |

4 | F

1 | A B C D E F

2 |

3 |

4 |

Depth: 10 Goal\_tests: 995 Max\_Queue\_size: 2531

Enter Heuristic choice:

Choose for heuristics from below:

Press 0 for Blocks out of place

Press 1 for Blocks counted twice in stack one

Press any other key to exit

1

iter\_count: 1

iter\_count: 2

….

iter\_count: 16

iter\_count: 17

iter\_count: 18

Success!!

Solution:

1 | F

2 | D

3 | A B C E

4 |

1 |

2 | D

3 | A B C E

4 | F

1 |

2 | D

3 | A B C

4 | F E

1 |

2 | D C

3 | A B

4 | F E

1 |

2 | D C B

3 | A

4 | F E

1 | A

2 | D C B

3 |

4 | F E

1 | A B

2 | D C

3 |

4 | F E

1 | A B C

2 | D

3 |

4 | F E

1 | A B C D

2 |

3 |

4 | F E

1 | A B C D E

2 |

3 |

4 | F

1 | A B C D E F

2 |

3 |

4 |

Depth: 10 Goal\_tests: 18 Max\_Queue\_size: 101

**Description of Heuristic**

The f(n) of A\* search is given by: f(n)=g(n)+h(n)

Where g(n) – path cost

h(n) – heuristic cost

The heuristics involved in the A\* search algorithm are:

*h0* – Number of blocks out of place

*h1­* – Calculated as (# of blocks to be popped out in stack 1+ # of blocks popped out from other stacks to remove the min block + # of blocks incorrectly placed initially \*# of stacks)

The last part of the heuristic of (# of blocks incorrectly placed initially \*# of stacks) is just penalising the nodes by assigning cost to them, so that the algorithm finds chooses the node with least cost, which in turn results in choice closer to the goal state.

This penalising factor could be varied, but on increasing the penalising factor further, no significant progress was observed.

Example: If the initial state is:

1 | B F

2 | A D E

3 | C

4 |

In the example, g(n)=0 and h(n) for *h0­­­­­­* is h(n)=6

For *h1­­­* the value of h(n)= (2-0) +2+6\*4=28

So, f(n)0­=6 and f(n)1­=28

Now since, this state is not the goal state, at depth 1, we get:

1 | B 1 | B 1 | B

2 | A D E F 2 | A D E 2 | A D E

3 | C 3 | C F 3 | C

4 | 4 | 4 | F

h(n)0­=6 h(n)0­=6 h(n)0­=6

h(n)1­=1+3+6\*4 h(n)1­=1+2+1+6\*4 h(n)1­=2+2+24

=28 =28 =28

And g(n)=1 in all the 3 nodes generated since they are at the same depth of 1 from the initial state.

**Admissibility**

We observe that in the above example, the solution was found to be at depth of 10 from Trace 2, but the h(n) scores of the heuristic provides that *h0­­*=6 and *h1­*=28 which reflects that *h0­* is under estimating and *h1­* is over estimating.

So, we can infer that *h0 ­* is admissible whereas *h1­* is not.

**Tabulated results**

**Blocks=5 and Stacks=3**

For heuristic *h0*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Start state # | # of blocks | # of stacks | Depth of Solution | Goal tests | Max Frontier size |
| 1 | 5 | 3 | 8 | 92 | 126 |
| 2 | 5 | 3 | 6 | 36 | 54 |
| 3 | 5 | 3 | 7 | 56 | 78 |
| 4 | 5 | 3 | 10 | 344 | 316 |
| 5 | 5 | 3 | 7 | 57 | 79 |
| 6 | 5 | 3 | 8 | 73 | 96 |
| 7 | 5 | 3 | 8 | 117 | 138 |
| 8 | 5 | 3 | 9 | 104 | 158 |
| 9 | 5 | 3 | 7 | 39 | 59 |
| 10 | 5 | 3 | 10 | 321 | 305 |
|  |  | AVG | 8 | 123.9 | 140.9 |

For heuristic *h1*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Start state # | # of blocks | # of stacks | Depth of Solution | Goal tests | Max Frontier size |
| 1 | 5 | 3 | 8 | 11 | 19 |
| 2 | 5 | 3 | 6 | 8 | 18 |
| 3 | 5 | 3 | 7 | 10 | 22 |
| 4 | 5 | 3 | 10 | 13 | 29 |
| 5 | 5 | 3 | 7 | 10 | 22 |
| 6 | 5 | 3 | 8 | 10 | 18 |
| 7 | 5 | 3 | 8 | 15 | 35 |
| 8 | 5 | 3 | 9 | 13 | 21 |
| 9 | 5 | 3 | 7 | 8 | 16 |
| 10 | 5 | 3 | 10 | 12 | 21 |
|  |  | AVG | 8 | 11 | 22.1 |

From the values from the above two tables we can infer that though the depth of the solution on average remains the same, the number of goal tests being performed and the max queue size drops by around 90% and 70% respectively.

**Blocks=6 and Stacks=4**

For heuristic *h0*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Start state # | # of blocks | # of stacks | Depth of Solution | Goal tests | Max Frontier size |
| 1 | 6 | 4 | 8 | 430 | 1002 |
| 2 | 6 | 4 | 8 | 185 | 557 |
| 3 | 6 | 4 | 11 | 923 | 2271 |
| 4 | 6 | 4 | 7 | 105 | 299 |
| 5 | 6 | 4 | 11 | 984 | 2478 |
| 6 | 6 | 4 | 10 | 1027 | 2627 |
| 7 | 6 | 4 | 10 | 1561 | 3327 |
| 8 | 6 | 4 | 7 | 80 | 308 |
| 9 | 6 | 4 | 11 | 2314 | 4920 |
| 10 | 6 | 4 | 9 | 507 | 1328 |
|  |  | AVG | 9.2 | 811.6 | 1911.8 |

For heuristic *h1*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Start state # | # of blocks | # of stacks | Depth of Solution | Goal tests | Max Frontier size |
| 1 | 6 | 4 | 7 | 8 | 36 |
| 2 | 6 | 4 | 8 | 12 | 56 |
| 3 | 6 | 4 | 11 | 36 | 187 |
| 4 | 6 | 4 | 7 | 11 | 39 |
| 5 | 6 | 4 | 11 | 21 | 108 |
| 6 | 6 | 4 | 10 | 11 | 51 |
| 7 | 6 | 4 | 9 | 13 | 71 |
| 8 | 6 | 4 | 7 | 10 | 57 |
| 9 | 6 | 4 | 12 | 19 | 112 |
| 10 | 6 | 4 | 9 | 12 | 68 |
|  |  | AVG | 9.1 | 15.3 | 78.5 |

Even from these tables of comparison between *h0* and *h1* we can observe the pattern that the number of goal tests and the queue size stays low in the new heuristic though the depth of the solution remains the same. If we see the case 9 of the above tables the depth of the solution in the new heuristic is more than that in the case of *h0­*, this could be since all nodes are marked visited during the time of generation rather than during exploration, so might give us sub-optimal solutions.

**Blocks=7 and Stacks=3**

Since, for *h0­* the runtime is too long, the computations for the tables on increasing the blocks are only done for *h1­*.

For heuristic *h1*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Start state # | # of blocks | # of stacks | Depth of Solution | Goal tests | Max Frontier size |
| 1 | 7 | 3 | 16 | 36 | 90 |
| 2 | 7 | 3 | 11 | 18 | 42 |
| 3 | 7 | 3 | 10 | 12 | 30 |
| 4 | 7 | 3 | 14 | 19 | 35 |
| 5 | 7 | 3 | 18 | 163 | 338 |
| 6 | 7 | 3 | 17 | 33 | 77 |
| 7 | 7 | 3 | 12 | 26 | 40 |
| 8 | 7 | 3 | 16 | 75 | 185 |
| 9 | 7 | 3 | 15 | 31 | 78 |
| 10 | 7 | 3 | 14 | 41 | 108 |

**Blocks=10 and Stacks-3**

For heuristic *h1*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Start state # | # of blocks | # of stacks | Depth of Solution | Goal tests | Max Frontier size |
| 1 | 10 | 3 | 14 | 17 | 43 |
| 2 | 10 | 3 | 26 | 1042 | 1928 |
| 3 | 10 | 3 | 30 | 3972 | 6294 |
| 4 | 10 | 3 | 23 | 94 | 208 |
| 5 | 10 | 3 | 26 | 78 | 178 |
| 6 | 10 | 3 | 24 | 38 | 88 |
| 7 | 10 | 3 | 18 | 24 | 60 |
| 8 | 10 | 3 | 28 | 2595 | 4616 |
| 9 | 10 | 3 | 27 | 85 | 187 |
| 10 | 10 | 3 | 21 | 983 | 2205 |

**Blocks=8 and Stacks=5**

For heuristic *h1*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Start state # | # of blocks | # of stacks | Depth of Solution | Goal tests | Max Frontier size |
| 1 | 8 | 5 | 12 | 13 | 128 |
| 2 | 8 | 5 | 14 | 18 | 198 |
| 3 | 8 | 5 | 13 | 17 | 144 |
| 4 | 8 | 5 | 14 | 19 | 198 |
| 5 | 8 | 5 | 14 | 16 | 137 |
| 6 | 8 | 5 | 10 | 11 | 93 |
| 7 | 8 | 5 | 13 | 14 | 119 |
| 8 | 8 | 5 | 11 | 14 | 160 |
| 9 | 8 | 5 | 12 | 14 | 151 |
| 10 | 8 | 5 | 10 | 11 | 115 |

**Failures**

In the case of *h0, ­*the goal test bound of 10,000 crosses easily even while the new heuristic finds a solution.

**Blocks = 9 and Stacks = 4**

*h0* – Failure

*h1* – Depth:17 Goal Tests:35 Queue size:222

**Blocks = 10 and Stacks = 5**

*h0* – Failure

*h1* – Depth:18 Goal Tests:38 Queue size:401

**Blocks = 25 and Stacks = 10**

*h0* – Failure

*h1* – Depth:43 Goal Tests:81 Queue size:5309

**Blocks = 25 and Stacks = 15**

*h0* – Failure

*h1* – Depth:43 Goal Tests:81 Queue size:5309