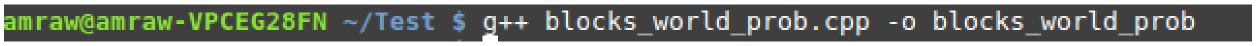
1. Compile and Running code

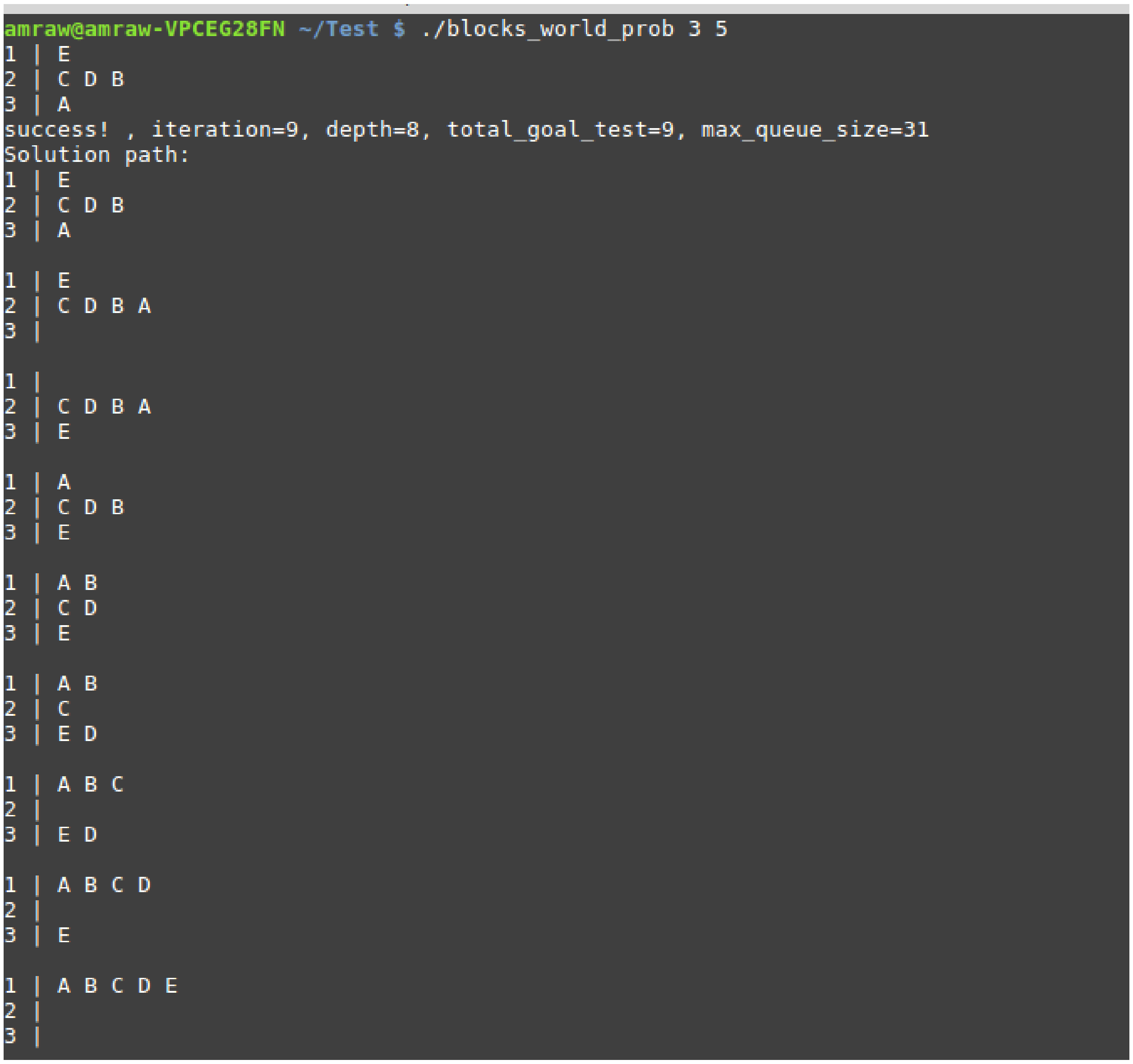
a) Go to the directory containing the source code file "blocks_world_prob.cpp" using cd command in Linux terminal.

b) Then execute command "**g++ blocks_world_prob.cpp -o blocks_world_prob**" on the terminal. You need a g++ package installed on Linux system for compiling the source code.



```
amraw@amraw-VPCEG28FN ~/Test $ g++ blocks_world_prob.cpp -o blocks_world_prob
```

c) Then run the executable file generated above **" ./blocks_world_prob 3 5". Where 3 is stack size and 5 is the number of blocks. So, the first parameter passed is stack size and the second parameter is a number of blocks.**



```
amraw@amraw-VPCEG28FN ~/Test $ ./blocks_world_prob 3 5
1 | E
2 | C D B
3 | A
success! , iteration=9, depth=8, total_goal_test=9, max_queue_size=31
Solution path:
1 | E
2 | C D B
3 | A

1 | E
2 | C D B A
3 |

1 |
2 | C D B A
3 | E

1 | A
2 | C D B
3 | E

1 | A B
2 | C D
3 | E

1 | A B
2 | C
3 | E D

1 | A B C
2 |
3 | E D

1 | A B C D
2 |
3 | E

1 | A B C D E
2 |
3 |
```

2. Limitations

a) The value of stack Size should be greater than or equal to 3 because my random state generator requires at least 3 stack size to generate a valid random state.

b) The program will stop working if the number of iterations exceeds 17,000.

3. Cost Value

The cost is calculated as number of blocks moved to reach that state.

4. Heurist

The function "calculateHeuristicValue" calls the subroutines which update heuristic variable "hNvalue" which is return by the function. The functionality of each subroutine is explained below: -

a) heuristic1: - This subroutine calculates for each block out of position the number of blocks places above it which are higher than it in goal state. For example: -
1 |A
2 |
3 | B D C E
In the above example, the first block out of position is 'B' so it will start searching 'B' in each stack starting from stack 1 to higher stack until it finds it. When 'B' is found in the above example which is at stack 3 then it will iterate over it and increment hNvalue for all the blocks which are placed above it in the goal state. For example, for block B the blocks D, C, and E are placed above it in the final goal state so hNvalue is incremented three times. Similarly, for block C there is only one block which is above it in goal state i.e E so hNvalue is incremented by one. Similarly, for block D only block E is considered so hNvalue is incremented by one. For block D block C is not considered because it is below it in the goal state.

The main idea is to count the number of blocks to be shifted in order to get each block which is out of position. In above example for block D, the block C is not counted because block C will be already placed in the first stack over B when block D is to be placed in that stack. So, we don't need to shift C to stack 2 in order to get D because C will go to stack 1 over B.

b) heuristic2: -
It counts the number of blocks which are out of position in the first row to be shifted to another row to create space for the block which is next in order as per the goal state.
For example: -
1| D B A
2| C
3| E
In the above example we need to move 3 blocks to make this row empty for block A. Suppose from above state two successors generated given below: -

1| D B   1| D B A C
2| C A   and   2|

```
3| E                                          3| E
```

The heuristic value of the second state is greater than the heuristic value of the first state. The main idea is to increase the heuristic value of child at same level with greater number of out of place block in the first row.

c) heuristic3: -
This heuristic supports the movement of the block over a block which is placed above it in goal state. For example: -

```
1|
2|F
3|B C
4|A D E
```

The successor state generated from above are

```
1|                                         1|
2|F                    or                  2|F E
3|B C E                                    3|B C
4|A D                                      4|A D
```

Then the second state will have lesser heuristic value because E is moved over a block F which is placed above it in goal state.

d) heuristic4: -
It reduces the value of heuristic by the number of the blocks which are aligned in the correct order in the first row. For example: -

```
1|
2|B C
3|D E A
```

From the above state two successor states generated: -

```
1 |A                                 1|
2 |B C               or              2|B C A
3 |D E                               3|D E
```

The first state will have lesser heuristic value than the second one.

e) heuristic5: -
This subroutine calculates how far the current state is from the goal state. For example. For example:

```
1| E B C D A
2|
3|
```

Here the function return value 5. Although, blocks B, C, and D are in right order since the top first block is not A hence it consider all the block out of order.

These heuristics complement each other and help in finding the goal state faster and in a minimum number of iteration.

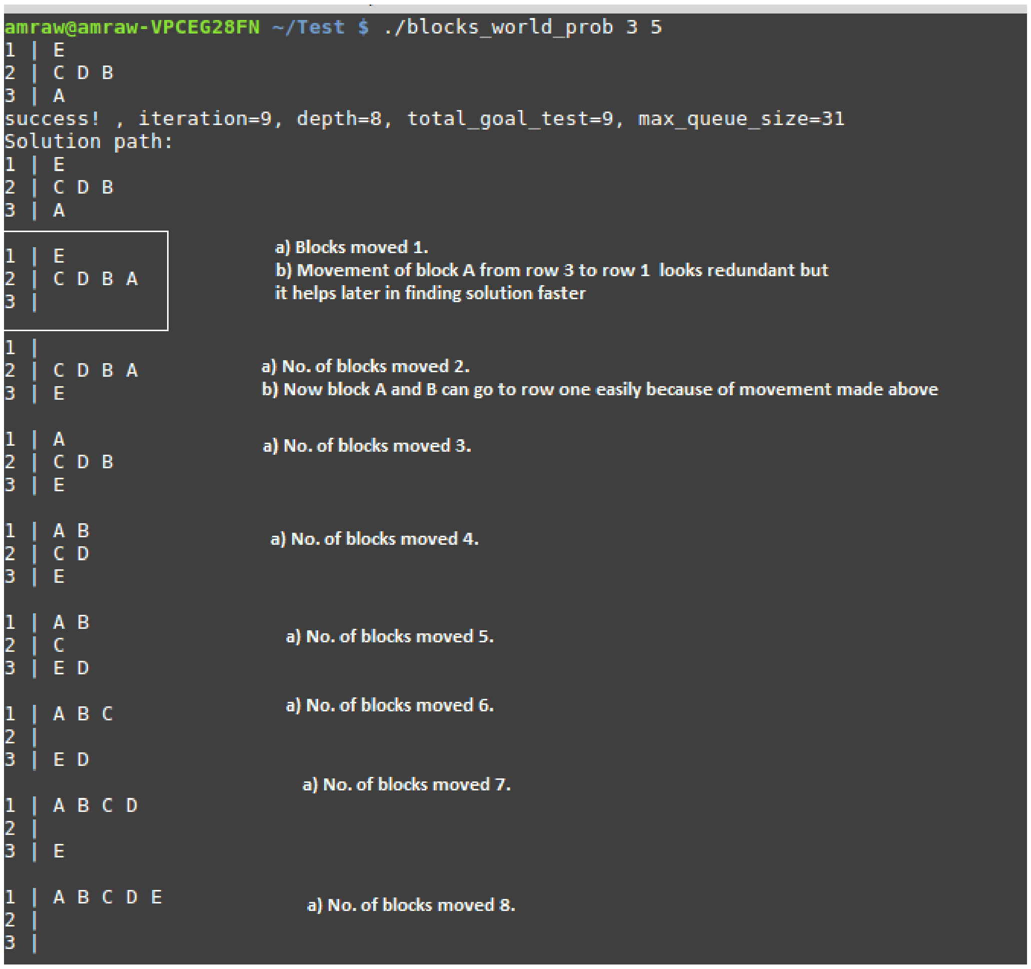
5. Performance Analysis

Test cases table is given below: -

Sr. No.	No. of Block	Stack Size	No. of goal test	No. of Iteration	Maximum Queue Size	Mean path length	Time(Sec)	No. of failures
1	5	3	10	10	46	9	0.04	
2	6	3	38	38	138	12	0.184	
3	7	3	13	13	47	11	0.06	
4	8	3	255	258	1185	19	0.3	
5	9	3	585	594	2778	23	1.2	
6	10	3	1194	1230	5362	26	2.02	1 out of 10 test cases is failing
7	12	3	9801	9890	47038	35	37.24	4 out of 10 test cases is failing
8	10	4	4681	4688	50360	28	32.22	
9	10	5	56	56	897	18	0.160	
10	10	6	23	23	481	17	0.136	
11	10	7	68	68	2295	16	0.15	
12	10	8	15	15	405	15	0.1048	
13	8	10	16	16	623	10	0.09153	

Conclusions from above table: -

a) The heuristic is performing as per my expectation. In most of the cases, I had run it gave good results.
b) The largest problem I am able to solve is 13 blocks on 3 stacks. For some inputs it gave output but for some cases, it was not able to generate the output.
c) As you can see from test cases 9 to 14 as the stack size increases it's become quite easy to solve the problem because heuristic3 has more option to move the block to an empty stack or above another block which is placed above it in goal state.
d) My program generally finds optimal solution because of following reason: -



```
amraw@amraw-VPCEG28FN ~/Test $ ./blocks_world_prob 3 5
1 | E
2 | C D B
3 | A
success! , iteration=9, depth=8, total_goal_test=9, max_queue_size=31
Solution path:
1 | E
2 | C D B
3 | A

1 | E
2 | C D B A
3 |

1 |
2 | C D B A
3 | E

1 | A
2 | C D B
3 | E

1 | A B
2 | C D
3 | E

1 | A B
2 | C
3 | E D

1 | A B C
2 |
3 | E D

1 | A B C D
2 |
3 | E

1 | A B C D E
2 |
3 |
```

a) Blocks moved 1.
b) Movement of block A from row 3 to row 1 looks redundant but it helps later in finding solution faster

a) No. of blocks moved 2.
b) Now block A and B can go to row one easily because of movement made above

a) No. of blocks moved 3.

a) No. of blocks moved 4.

a) No. of blocks moved 5.

a) No. of blocks moved 6.

a) No. of blocks moved 7.

a) No. of blocks moved 8.

i) Heuristic3 and heuristic 1 return lesser for second state because movement of block A over B is best possible move then moving E over A or B or moving B over A.
ii) Heuristic2 and heuristic 1 returns lesser value for third state because row one is empty and block A can be move there.
iii) Heursitic5 and heuristic6 return lesser value for the fourth state because A has reached it correct position.
iv) Heursitic4 and heuristic5 return lesser value for the fifth state because A and B has reached it correct position.
v) Heuristic3 and heuristic 1 return lesser value for sixth state because movement of block D over E is best possible move then moving D over B.
vi) Heursitic4 and heuristic5 return lesser value for the seventh state because A, B and C has reached it correct position.
vii) Heursitic4 and heuristic5 return lesser value for the eighth state because A, B, C and D has reached its correct position.
viii) Heursitic4 and heuristic5 return lesser value for the ninth state because it is goal state.

As you can see that heuristic4, heuristic2 and heuristic5 are helping in converging to the goal state faster. Whereas heuristic3 and heuristic 1 is helping in finding arrangements of blocks where it is easy to find block from A to E on top by reducing heuristic value of states where a block move over a block which is placed above it in goal state. In the solution path each child state has a cost (no. of blocks moved) one more than its parent state.

e) From above data my Queue size is increasing approximately 4 or 5 times the number of iteration.

6. Implementation related points: -

a) I used **map in c++** to store the visited nodes which really improve the performance of my program because it uses a **hash table** to store data and searching in hash table is quite easy.

b) My random state generator uses **18 combinations of 7 operations** to generate a random state which is most of the time is unique and distributed.

c) Function is added to free the dynamic memory allocated.