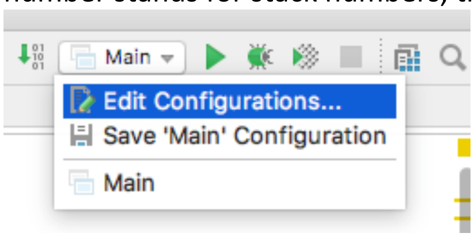
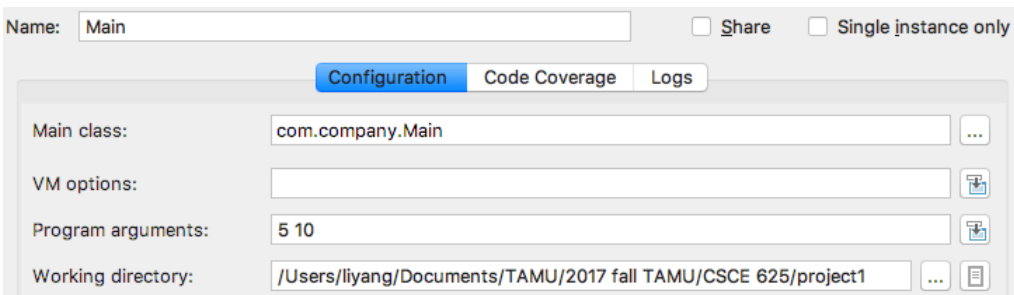
 Yang Li UIN: 126004157

CSCE 625 Project #1 Report

Basic information:
I finished this project by JAVA and tested the program by using IntelliJ IDEA.

Compile:
First choice: Eclipse and IntelliJ can be used to run this program. In order to change the number of blocks and stacks we can adjust this parameter by using edit configuration function. The first number stands for stack numbers, the other means blocks number.





Second choice: we can also run this program in terminal (Mac) or command line (Windows).
Compile:
(there are two folders, named src1 and src2, represent two version of heuristic function)

1. Enter the folder of this project: cd path_of_the_folder
2. Enter src2: cd src2 (if you want to run the first heuristic function, please enter src1)
3. javac com/company/*.java

Run:

1. java -cp . com.company.Main no_stack no_blocks

Output Demo:

1. **Situation for 3 stacks and 5 blocks**

CSCE 625 Project #1

Yang Li UIN: 126004157

```
mail-alt:src liyang$ javac com/company/*.java
mail-alt:src liyang$ java -cp . com.company.Main 3 5
initial state :
1 | [B, C, E]
2 | []
3 | [A, D]
iter = 0 , queue = 0 ,  f = g + h = 6 , depth = 0
iter = 1 , queue = 3 ,  f = g + h = 6 , depth = 1
iter = 2 , queue = 4 ,  f = g + h = 6 , depth = 1
iter = 3 , queue = 7 ,  f = g + h = 7 , depth = 2
iter = 4 , queue = 8 ,  f = g + h = 7 , depth = 2
iter = 5 , queue = 7 ,  f = g + h = 7 , depth = 2
iter = 6 , queue = 8 ,  f = g + h = 7 , depth = 1
iter = 7 , queue = 11 ,  f = g + h = 7 , depth = 2
```

```
iter = 55 , queue = 84 ,  f = g + h = 9 , depth = 5
iter = 56 , queue = 85 ,  f = g + h = 9 , depth = 3
iter = 57 , queue = 86 ,  f = g + h = 9 , depth = 6
iter = 58 , queue = 87 ,  f = g + h = 9 , depth = 4
iter = 59 , queue = 87 ,  f = g + h = 9 , depth = 7
iter = 60 , queue = 88 ,  f = g + h = 9 , depth = 3
iter = 61 , queue = 89 ,  f = g + h = 9 , depth = 8
iter = 62 , queue = 89 f = g + h = 9 , depth = 9
Success! depth = 9 , total_goal_tests = 62 , max_queue_size = 90
solution path :
from start to end we need go through follow state
the 0 iteration
1 | [B, C, E]
2 | []
3 | [A, D]

the 1 iteration
1 | [B, C]
2 | [E]
3 | [A, D]

the 2 iteration
1 | [B, C]
2 | [E, D]
3 | [A]

the 3 iteration
1 | [B]
2 | [E, D, C]
3 | [A]

the 4 iteration
1 | []
2 | [E, D, C, B]
3 | [A]

the 5 iteration
1 | [A]
2 | [E, D, C, B]
3 | []

the 6 iteration
1 | [A, B]
2 | [E, D, C]
3 | []

the 7 iteration
1 | [A, B, C]
2 | [E, D]
3 | []

the 8 iteration
1 | [A, B, C, D]
2 | [E]
3 | []

the 9 iteration
1 | [A, B, C, D, E]
2 | []
3 | []
```

CSCE 625 Project #1

Yang Li UIN: 126004157

2. Situation for 20 stacks and 26 blocks （second heuristic function ）

```
1 | []
2 | []
3 | [K, T]
4 | [J, O, P]
5 | [Q]
6 | [N]
7 | [F, H]
8 | []
9 | [R, U]
10 | [M]
11 | [Z]
12 | [Y]
13 | [G]
14 | [B, I]
15 | [E, V, X]
16 | [L]
17 | []
18 | [D, W]
19 | [C]
20 | [A, S]
```

```
iter = 159 , queue = 44304 ,  f = g + h = 41 , depth = 36
iter = 160 , queue = 44398 ,  f = g + h = 41 , depth = 37
iter = 161 , queue = 44473 ,  f = g + h = 41 , depth = 38
iter = 162 , queue = 44529 ,  f = g + h = 41 , depth = 39
iter = 163 , queue = 44566 ,  f = g + h = 41 , depth = 40
iter = 164 , queue = 44566 f = g + h = 41 , depth = 41
Success! depth = 41 , total_goal_tests = 164 , max_queue_size = 44567
solution path :
from start to end we need go through follow state
the 0 iteration
1 | []
2 | []
3 | [K, T]
4 | [J, O, P]
5 | [Q]
6 | [N]
7 | [F, H]
8 | []
9 | [R, U]
10 | [M]
11 | [Z]
12 | [Y]
13 | [G]
14 | [B, I]
15 | [E, V, X]
16 | [L]
17 | []
18 | [D, W]
19 | [C]
20 | [A, S]
```

……

```
the 41 iteration
1 | [A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z]
2 | []
3 | []
4 | []
5 | []
6 | []
7 | []
8 | []
9 | []
10 | []
11 | []
12 | []
13 | []
14 | []
15 | []
16 | []
17 | []
18 | []
19 | []
20 | []
```

Situation for 5 stacks and 10 blocks

```
[mail-alt:src liyang$ java -cp . com.company.Main 5 10
initial state :
1 | [A, J]
2 | [H]
3 | []
4 | [B, E, G, I]
5 | [C, D, F]
iter = 0 , queue = 0 ,  f = g + h = 14 , depth = 0
iter = 1 , queue = 15 ,  f = g + h = 14 , depth = 1
iter = 2 , queue = 26 ,  f = g + h = 14 , depth = 1
iter = 3 , queue = 41 ,  f = g + h = 14 , depth = 2
iter = 4 , queue = 51 ,  f = g + h = 14 , depth = 1
iter = 5 , queue = 62 ,  f = g + h = 14 , depth = 2
iter = 6 , queue = 77 ,  f = g + h = 14 , depth = 2
iter = 7 , queue = 88 ,  f = g + h = 14 , depth = 2
iter = 8 , queue = 102 ,  f = g + h = 14 , depth = 2
iter = 9 , queue = 117 ,  f = g + h = 14 , depth = 3
```

```
iter = 4211 , queue = 35665 ,  f = g + h = 15 , depth = 11
iter = 4212 , queue = 35672 ,  f = g + h = 15 , depth = 3
iter = 4213 , queue = 35679 ,  f = g + h = 15 , depth = 12
iter = 4214 , queue = 35686 ,  f = g + h = 15 , depth = 3
iter = 4215 , queue = 35695 ,  f = g + h = 15 , depth = 13
iter = 4216 , queue = 35698 ,  f = g + h = 15 , depth = 5
iter = 4217 , queue = 35705 ,  f = g + h = 15 , depth = 14
iter = 4218 , queue = 35705 f = g + h = 15 , depth = 15
Success! depth = 15 , total_goal_tests = 4218 , max_queue_size = 35706
solution path :
from start to end we need go through follow state
the 0 iteration
1 | [A, J]
2 | [H]
3 | []
4 | [B, E, G, I]
5 | [C, D, F]

the 1 iteration
1 | [A]
2 | [H]
3 | [J]
4 | [B, E, G, I]
5 | [C, D, F]

the 2 iteration
1 | [A]
2 | [H]
3 | [J, I]
4 | [B, E, G]
5 | [C, D, F]

the 3 iteration
1 | [A]
2 | [H, F]
3 | [J, I]
4 | [B, E, G]
5 | [C, D]

the 4 iteration
1 | [A]
2 | [H, F]
3 | [J, I, G]
4 | [B, E]
5 | [C, D]

the 5 iteration
1 | [A]
2 | [H, F]
3 | [J, I, G, E]
4 | [B]
5 | [C, D]

the 6 iteration
1 | [A, B]
2 | [H, F]
3 | [J, I, G, E]
4 | []
5 | [C, D]
```

```
the 7 iteration
1 | [A, B]
2 | [H, F]
3 | [J, I, G, E]
4 | [D]
5 | [C]

the 8 iteration
1 | [A, B, C]
2 | [H, F]
3 | [J, I, G, E]
4 | [D]
5 | []

the 9 iteration
1 | [A, B, C, D]
2 | [H, F]
3 | [J, I, G, E]
4 | []
5 | []

the 10 iteration
1 | [A, B, C, D, E]
2 | [H, F]
3 | [J, I, G]
4 | []
5 | []

the 11 iteration
1 | [A, B, C, D, E, F]
2 | [H]
3 | [J, I, G]
4 | []
5 | []

the 12 iteration
1 | [A, B, C, D, E, F, G]
2 | [H]
3 | [J, I]
4 | []
5 | []

the 13 iteration
1 | [A, B, C, D, E, F, G, H]
2 | []
3 | [J, I]
4 | []
5 | []

the 14 iteration
1 | [A, B, C, D, E, F, G, H, I]
2 | []
3 | [J]
4 | []
5 | []

the 15 iteration
1 | [A, B, C, D, E, F, G, H, I, J]
2 | []
3 | []
4 | []
5 | []
```

CSCE 625 Project #1

Yang Li UIN: 126004157

Heuristic Description:

This project is meanly focused on using A* search to solve "Blocksworld" problem. As we know A* search is based on f(n), which equals g(n) + h(n). g(n) is the distance that node n away from start position and h(n) is the estimate of distance between node n and end positon. g(n) is a well-defined value, therefore, the performance is heavily relay on heuristic function.
I actually implemented two **Heuristic function.**

The program I turned in was implemented by the second Heuristic function.

First version of Heuristic function.
In this project, I calculate heuristic value as follow:
Our target is always defined to have all the blocks in alphabetical order on stack one.
At beginning, our heuristic is 0.
First: Check the first stack of current state "Blocksworld".
If the position of the target block in current stack one is different with the first block of target state. Then that means if we want to change current state to target state we at least need to remove all blocks in the first stack of current state. That means :
heuristic = heuristic + Number_of_blocks_in_current_state_first_stack - targetStateIndex

If the first position is same as target state. Then we need to check where we get wrong blocks. Let's assume the wrong position is pWrong. So, we get
(Number_of_blocks_in_current_state_first_stack) minus (pWrong wrong blocks in stack one).
That is the number of blocks we need to remove from the first stack.

Second: for stacks other than first one, all blocks are in wrong position, therefore we need to move all of them to stack one. That means:
heuristic = heuristic + all_Number_of_blocks_in_stacks.

After this, we need to pay attention to **one situation**: if current blocks are in the same order as target position. Eg. In stack 3, our blocks are B C. Then that means when we want to put B, C in target position, we have to move away blocks C at first. As for that, if we want to move B, C to right position, we need 2 + 1 = 3 steps at least.
For the reason stated above, we need to find all that blocks, and add one to heuristic every time that situation occurs.

This heuristic is admissible. For the reason that we only consider at least what we should do. We calculate how many moves we need to do if we want to move the blocks to the first stack. That is at least what we should do. In order to change all blocks to first blocks we at least need to remove all wrong blocks at stack one and move all blocks located in other stacks to stack one. A clever point here is we took the situation that I described above in to account.

CSCE 625 Project #1

Yang Li UIN: 126004157

Second version of Heuristic function.

Our target is always defined to have all the blocks in alphabetical order on stack one.
At beginning, our heuristic is 0.

For all nodes in target, E.g. (from A – F)
Ever node is a target block and we check how many steps we need to take if we want to move the target block in current state to the right position.

First:

<u>First case:</u> if the stacks other than the first stack of current state "Blocksworld" contains target block.
If the position of the target block in current state is different with the block position in target state. Then that means if we want to put target block in right position then we should remove all nodes whose position in the first stack of current state are higher than the right position of first node.
That means :(targetStateIndex is the position the block should be)
heuristic = heuristic + Number_of_blocks_in_current_state_first_stack - targetStateIndex.
After that, in order to change to target state, we still need to put some blocks above the block that we just moved. That number would be:
hValue = hValue + blockSize - targetStateIndex;

<u>second case:</u> if the first stack contains target block.
hValue = hValue + modifyStates.get(0).size() - targetStateIndex;
hValue = hValue + targetStackOne.size() - targetStateIndex;
means the total number of blocks we moved away from stack one plus the total number of steps to add blocks to stack one.

Second:

After above, we need to pay attention to one situation: if current blocks are in the same order as target position. Eg. In stack 3, our blocks are B C. Then that means when we want to put B, C in target position, we have to move away blocks C at first. As for that, if we want to move B, C to right position, we need 2 + 1 = 3 steps at least.
For the reason stated above, we need to find all that blocks, and add one to heuristic every time that situation occurs.
For the first stack, we need to check all nodes under the target blocks.

This heuristic is not admissible. We calculate all steps we need to do for every node, however, when we actually move blocks, we do not need do all steps for every node. It over estimates the minimum cost.

CSCE 625 Project #1

Yang Li UIN: 126004157

Performance:

First Heuristic function.

I will show details for 3 stack situations and give a general statistic result for other situations.

stack : 3 block: 5	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Iteration Number	13	25	69	14	78	29	62	154	34	34
Maximum Queue size	21	36	130	21	111	47	93	194	68	63
path length	5	7	9	5	9	7	9	10	8	7

stack : 3 block: 6	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Iteration Number	398	175	104	34	171	1114	197	101	376	138
Maximum Queue size	545	268	193	70	304	1306	291	158	524	222
path length	12	11	10	8	11	13	11	10	12	11

stack : 3 block: 7	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Iteration Number	5	2968	1613	3586	1195	1510	2729	1103	3927	161
Maximum Queue size	8	4354	2101	4839	1636	2024	3695	1734	4808	267
path length	3	15	15	15	14	14	15	14	16	11

stack : 3 block: 10	**1**	**2**	**3**	**4**	**5**	**7**	**8**	**9**	**10**
Iteration Number	332077	11298	14883	41161	434040	62087	2958	54071	649362
Maximum Queue size	442202	17709	22429	59598	528269	105172	6334	75924	841408
path length	23	19	19	21	23	21	18	20	24

CSCE 625 Project #1

Yang Li UIN: 126004157

stack : 5 **block: 10**	**1**	**2**	**3**	**4**	**5**
Iteration Number	18163	10705	17548	16492	14546
Maximum Queue size	133840	67613	135104	164028	71328
path length	18	16	17	17	15

6	**7**	**8**	**9**	**10**
22298	17942	8143	46284	62211
131802	135567	61627	266755	262330
17	17	18	17	17

Statistic result of 12 situations:

Number of stack	**Number of block**	**Mean Iteration Number**	**Mean Maximum Queue size**	**Mean path length**	**Number of success**
3	**5**	48	78	8	10/10
3	**6**	280	388	11	10/10
3	**7**	7580	11000	13	10/10
3	**8**	17831	21602	16	10/10
3	**9**	95246	67342	19	10/10
3	**10**	161305	211595	23	10/10
4	**10**	32142	93859	18	10/10
5	**10**	23433	142999	17	10/10
6	**10**	34202	325055	16	10/10
7	**10**	12352	184123	15	10/10
7	**11**	73523	601324	17	10/10
7	**12**	34490	584063	18	10/10

CSCE 625 Project #1

Yang Li UIN: 126004157

Performance:

second Heuristic function.

Number of stack	Number of block	Mean Iteration Number	Mean Maximum Queue size	Mean path length	Number of success
3	**5**	22.2	49.9	9	10/10
3	**7**	330.5	697	15	10/10
3	**10**	10960	21231	29	10/10
5	**10**	1513	5784	24	10/10
6	**10**	2188	28999	21	10/10
7	**10**	42.5	1140.5	16	10/10
7	**15**	16242	379419	32	10/10
10	**15**	294	18354	25	10/10
10	**20**	29365	83549	25	3/6
15	**26**	12342	553667	47.5	2/5
20	**26**	300	67830	46	4/5

CSCE 625 Project #1

Yang Li UIN: 126004157

Discussion:

As we can see in the form. The first **Heuristic function** can get the optimal path. However, program needs do lots of iteration to achieve that result.

For this **Heuristic function,** the largest problems I tried to solve is 7 stacks and 12 blocks. For more complicated situation, this **Heuristic function** might can solve. However, it will cost too much time. The queue size grows exponentially as the iteration number.

As the number of stacks grows up, we can see that the max heap size grows according to it. problem becomes easier. That is because when we have more stacks, our successor state will become more than before, and we have more choice to move our blocks, therefore it is easier to transform to the target state.

For the second **Heuristic function.** It obviously decreases the iteration number. However, the path is longer than optimal. Nevertheless, during the test I found in some situations this heuristic could find a path in a way that do not have even one way away from the right path. At the same time, I also find another problem. When I did the test for some specific situations, the outcome might have a huge different. For example, when I test 5 stacks and 10 blocks for 10 times, the iteration number that distributed around 200 ~ 300 happens about 8 in 10 and for the rest two times, the iteration numbers are around 3000.

When we keep the number of stacks unchanged and increase the number of blocks. The problem will become harder and need more iteration to solve.

For the future improvement, I think we could do some adjust for second Heuristic function according to the performance.