# COMS 4701 Artificial Intelligence Homework 2

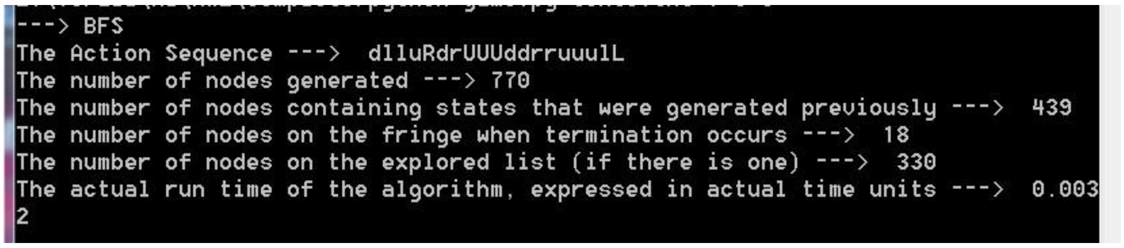
**Kaili Zhang kz2203**

<u>In test, All the puzzles provided could be solved!!! But some of them need really long time about 10 minites. Python is always slow.</u>

## To Run the Program:

python game.py txt_file_name.txt Algorithm_Flag Heuristic_Flag Deadlock_Flag
Algorithm_Flag: 1 – BFS; 2 – DFS; 3 – Normalized Cost; 4 – Greedy Search; 5 – A* Search
Heuristic_Flag: 1 – Function1; 2 – Function2; 3 – Function3
Deadlock_Flag: 0 – No Deadlock Check; 1 – Use Deadlock Check

Example: **python game.py text3.txt 1 1 0**
Solve problem in text3 using BFS without deadlock check.
Output Form:



```
---> BFS
The Action Sequence --->  dlluRdrUUUddrruuulL
The number of nodes generated ---> 770
The number of nodes containing states that were generated previously --->  439
The number of nodes on the fringe when termination occurs --->  18
The number of nodes on the explored list (if there is one) --->  330
The actual run time of the algorithm, expressed in actual time units --->  0.003
2
```

## What is New :

1. Heuristic Function
   HF_1 : 1) find the Manhattan distance between a box and the nearest goal.
   2) sum the previous distance together.
   This function only considered the distance between box and goal.
   HF_2: sum the Manhattan distances between person and all goals.
   This function only considered person and goal.
   HF_3: average of previous two heuristic value.
   This function considered both distances from person and boxes to goals.
   It seems that if we consider more information of state, then, better action the computer will do. However, result can be seen in Result Analysis Part.
2. Deadlock Check
   There are three basic situation which may cause deadlock:
   1) If a box is at corner of walls;
   2) If a box is at corner consisted by walls and box;
   3) If a box is at corner consisted by boxes;
   4) If a box is along a wall or a wall containing boxes, and no goals is at the vertical direction to box's moving direction.

I implemented all of these four deadlock situation.
As a result, about 1/3 nodes are avoid to be generated.

3. A hashing Idea used to record state
   Since python always slower than C or other programming language, I tried hard to make it run faster.
   Instead of map the puzzle graph into a two dimension array, I used a string to represent the puzzle which is also the state situation. This string links every line of original graph together.
   When comparing the difference between states, we may only need to compare two strings.
   Python's comparison method between string uses C language, so it is faster.
4. How to make it run much more faster
   I also used priority queue instead of sorting method to gain the smallest cost state.
   There's a tool of python which could make our script run as C language. But need to download.

## What else I have done:

➔ Another more intelligent method.

In common ideas, states change if person moved. However, actually, we can define that state changes if boxes moved. We firstly get the action sequence of box, and then, we calculate person's action sequence between adjacent box actions.
Best advantage is that, a lot of time and space are saved.
Disadvantage is that algorithm becomes quite difficult to implement.
I wrote hundreds lines of codes, but still can not complete. I also add them into assignment folder.
Firstly, we divide the graph into different space based on connectivity and sign some special positions. This is done by inverted DFS.
Whether a box could be moved based on if person could arrive at a correct position to move the box. So, I used the pace and special positions to finish that judgment.
Expansion criteria is also based on those search algorithms.

## Result Analysis:

1. For simple problems, BFS seems to be the fastest and DFS always give out very long action sequence.
2. We found that if we open the deadlock checking operation, for some simple problem, all of the 5 algorithms will give out the same action sequence even the statistical information differ. That means, for simple problem, deadlock checking may help search algorithm find the optimal solution, especially for in-optimal algorithms (greedy and DFS).
3. About the three heuristic functions, we can only test it on Greedy because A* always gives the optimal solution.
   I found that HF_1 which only considered boxes may provide a shorter action sequence but cost a little more time. HF_2 and HF_3 that related with person provide a longer action sequence but cost less time. And, for heuristic algorithm, most time are spend on calculating heuristic values.