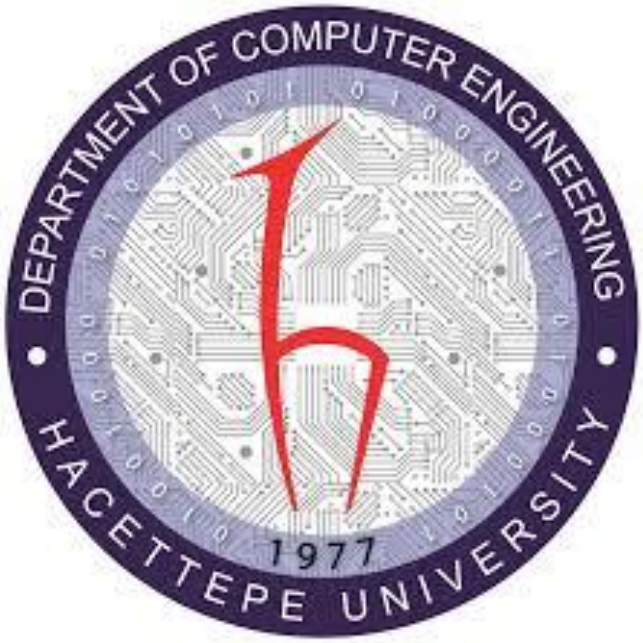


# HACETTEPE UNIVERSITY

# DEPARTMENT OF COMPUTER ENGINEERING

# BBM405

# HOMEWORK #1

**Subject: Applying Search Algorithms on Search Problems**
**Deadline: 10.5.2020**
**Programming Language: Python**
**Student ID: 21727432**
**Student Name: Ali Kayadibi**

# INTRODUCTION / AIM

In this homework, we are expected to implement four search algorithms such as:

- Depth First search
- Breadth First Search
- Uniform Cost Search
- A* search.

Depth First Search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

Uniform cost search is a tree search algorithm related to breadth-first search.Uniform cost search determines a path to the goal state that has the lowest weight.

A* is a graph traversal and path search algorithm, which is often used in computer science due to its completeness, optimality, and optimal efficiency.One major practical drawback is its O(b^d) space complexity, as it stores all generated nodes in memory.

After implementing our algorithms, we are expected to find creative search problems suitable to apply our search algorithm.After applying our algorithms, we are expected to compare our results we obtained from these four algorithms based on four criteria.They are:

- Optimality
- Completeness
- Time
- Space Complexity.

In this homework we are expected to find two different search problems one small one larger scale.I will call them Problem 1 and Problem 2.

# Implementation of Search Algorithms

I have uptaded search algorithms I took from https://github.com/chitholian/AI-Search-Algorithms. for my problems and added a print path part to output better.For second problem https://github.com/speix/8-puzzle-solver is used.

## Problem 1

For Problem 1 i have selected a path finding problem. My scenario is, i have maze a mice and a cheese is randomly dropped on random coordinates on randomly generated map where i represent 1's with can't go through(walls, objects etc..) and 0's with can go through.Mice's aim is to find the cheese using these four algorithms.

```
11011001101101010010101101010011
10001000010100110010011000010l
01000011000011011011100110010O
00110000000110001100101011010l
00000111101100000110000001000O
10011000100101000000101111111O
11101111110100100011110001000O
01101001011110100111101111001O
00110101011001001000011000010O
01010101011000110101100111000l
11101011101101110010001111100l
11111111100111100010101100101l
11001011010110100000000000110O
11001100011000011001110010101l
10001111101101101011111001110l
10111110011111110101000001110O
10001101110000000111010011100l
11100001101010011101111100011O
10010001000110101001000110000l
10100010110100011010011110000l
01100101000000100011101110001l
10000101111010110111100010010l
11110010111010111001110100011O
10000111111111000110000000001O
00011100001011110001101111111O
10100010111110000010000010010l
10001010100110100111101110110
00100110100001010011101010100O
01101101011011011111010011011l
01000100000011111010011010111l
```

This is my map. It is randomly generated , every coordinate is represented as nodes, we have one starting node(randomly generated) this represent where the mouse is dropped and a finish node(randomly generated) this represent where the cheese is.Mouse's aim is to find and eat the cheese.

For creating a map , I create a square map with given size, for this experiment i will be using 100, 200, 300 and 400 to compare my algorithms.

# Creating Square Map for Problem 1

I have a create map function which creates a square map with given size.It puts random ones and zeros inside this map.Ones represent can't go through and zeros represent can go through.We will be using this in search algorithms and this will change algorithm's performance.

**In a* search algorithm, heuristic values are bird flight distance values. In uniform cost search, the distance between the nodes are costs to reach from that node to that node.**

# EXPERIMENTS FOR PROBLEM 1

For experimenting Problem 1, i have created four different square maps with(100x100, 200x200, 300x300, 400x400) sizes.

For 100x100 i got these results:

Start coordinate :98,94
End coordinate :2,76
astar : 100x100-> runtime = 0.531376838684082
bfs : 100x100-> runtime = 0.07905411720275879
dfs : 100x100-> runtime = 0.11107897758483887
ucs : 100x100-> runtime = 1.0297324657440186

For 200x200 i got these results:

Start coordinate :172,40
End coordinate :158,102
astar : 200x200-> runtime = 0.16811895370483398
bfs : 200x200-> runtime = 0.12609100341796875
dfs : 200x200-> runtime = 0.09907031059265137
ucs : 200x200-> runtime = 1.5641107559204102

For 300x300 i got these results:

Start coordinate :39,69
End coordinate :219,59
astar : 300x300-> runtime = 5.941223382949829
bfs : 300x300-> runtime = 1.0207266807556152
Unexpected error for DFS: <class 'RecursionError'>
ucs : 300x300-> runtime = 12.834121704101562

For 400x400 i got these results:

Start coordinate :352,180
End coordinate :248,13
astar : 400x400-> runtime = 5.243725776672363
bfs : 400x400-> runtime = 2.0754759311676025
Unexpected error for DFS: <class 'RecursionError'>
ucs : 400x400-> runtime = 40.0834858417511

# ANALYSIS FROM EXPERIMENT 1

How do we evaluate and compare search algorithms?

- **Completeness** : Is it guaranteed to find a solution (if one exists)?
- **Optimality** : Does it find the “best” solution (if there are more than one)?
- **Time Complexity** : How long does it take to find a solution?
- **Space Complexity** : How much memory does it require?

### *Completeness*

From these experiment results, we can see that all three algorithms except Depth First Search algorithm always guarantees a solution(optimal or not).But in same cases, depth first search algorithm gives Recursion Error to us.This means all three except Depth First Search algorithm are complete.

### *Optimality*

There can be multiple solutions but which algorithm/s can choose the best solution?. To find this i have tested my algorithms and i will compare the paths they found.

**Path for A* Search:** 32,35 =>31,35 =>30,35 =>29,34 =>29,33 =>28,32 =>27,32 =>26,31 =>25,32 =>24,33 =>23,34 =>22,34 =>21,35 =>20,36 =>19,36 =>18,36 =>17,37 =>16,38 =>16,39 =>15,40 =>14,40 =>13,41 =>12,42 =>12,43 =>13,44
**Cost**:29.740000000000002

**Path for BFS:** 32,35 =>31,35 =>30,35 =>29,34 =>29,33 =>28,32 =>27,31 =>26,31 =>25,32 =>24,33 =>23,34 =>22,34 =>21,34 =>20,35 =>19,36 =>18,36 =>17,37 =>16,38 =>16,39 =>15,40 =>14,40 =>13,41 =>14,42 =>14,43 =>13,44

**Path for DFS:** 32,35 =>31,35 =>30,35 =>29,34 =>29,33 =>30,32 =>31,31 =>32,31 =>33,32 =>34,33 =>34,34 =>35,35 =>34,36 =>33,37 =>32,37 =>31,38 =>30,38 =>29,39 =>28,38 =>27,39 =>26,40 =>25,39 =>24,40 =>23,41 =>22,40 =>21,41 =>20,40 =>20,39 =>19,38 =>19,37 =>18,36 =>17,36 =>16,37 =>15,38 =>14,38 =>13,39 =>13,40 =>13,41 =>12,42 =>11,41 =>10,41 =>9,42 =>9,43 =>9,44 =>10,45 =>10,46 =>11,47 =>12,48 =>13,47 =>13,46 =>13,45 =>13,44

**Path for UCS:** 32,35 =>31,35 =>30,35 =>29,34 =>29,33 =>28,32 =>27,32 =>26,31 =>25,32 =>24,33 =>23,33 =>22,34 =>21,34 =>20,35 =>19,36 =>18,36 =>17,37 =>16,38 =>16,39 =>15,40 =>14,40 =>13,41 =>12,42 =>12,43 =>13,44
**Cost**:29.740000000000002

**As we see here, A* search and UCS finds the optimal solution with same cost but BFS and DPS finds solution but these solutions are not optimal.I have tried the same map and what i have found is, if the cost between nodes are all equal(sometimes becomes 1 instead of sqrt(2)) the BFS gives the optimal solution.**

**Path for A* Search:** 7,14 =>8,15 =>9,14 =>10,14 =>11,15 =>12,16 =>13,16 =>14,17 =>15,17 =>16,17 =>17,17 =>18,18 =>19,19 =>20,20 =>21,20 =>22,20 =>23,21 =>24,22 =>25,23 =>26,24 =>27,25 =>28,26 =>29,27 =>30,28 =>31,28 =>32,27 =>33,26 =>34,26 =>35,25 =>36,24 =>37,23 =>38,22 =>39,21 =>40,20 =>40,19 =>39,18 =>39,17 =>40,16 =>41,15 =>42,14 =>41,13
**Cost**:51.88999999999996

**Path for BFS:** 7,14 =>8,15 =>9,14 =>10,14 =>11,15 =>12,16 =>13,16 =>14,17 =>15,17 =>16,17 =>17,17 =>18,18 =>19,19 =>20,20 =>21,20 =>22,21 =>23,21 =>24,22 =>25,23 =>26,24 =>27,25 =>28,26 =>29,27 =>30,28 =>31,28 =>32,27 =>33,26 =>34,26 =>35,25 =>36,24 =>37,23 =>38,22 =>39,21 =>40,20 =>40,19 =>39,18 =>40,17 =>40,16 =>41,15 =>42,14 =>41,13

**Path for DFS:** 7,14 =>7,15 =>6,16 =>5,16 =>4,16 =>3,17 =>2,18 =>3,19 =>2,20 =>1,20 =>0,21 =>1,22 =>0,23 =>1,24 =>2,24 =>3,24 =>4,24 =>5,25 =>5,26 =>4,27 =>3,27 =>2,27 =>1,28 =>1,29 =>2,30 =>3,31 =>4,31 =>5,31 =>6,31 =>7,30 =>8,29 =>7,28 =>8,27 =>7,26 =>8,25 =>9,25 =>10,24 =>9,23 =>8,23 =>7,22 =>8,21 =>7,20 =>8,19 =>9,19 =>10,19 =>11,20 =>12,21 =>12,22 =>13,23 =>14,22 =>14,21 =>15,20 =>16,21 =>17,21 =>18,20 =>19,19 =>20,20 =>21,20 =>22,20 =>23,21 =>24,22 =>25,23 =>26,24 =>27,25 =>28,26 =>27,27 =>27,28 =>28,29 =>27,30 =>26,31 =>26,32 =>25,33 =>25,34 =>25,35 =>26,36 =>25,37 =>24,38 =>23,39 =>22,40 =>21,39 =>20,40 =>19,39 =>18,39 =>17,38 =>16,37 =>16,36 =>15,35 =>16,34 =>16,33 =>17,32 =>18,31 =>17,30 =>16,30 =>15,31 =>14,30 =>13,29 =>12,30 =>11,31 =>11,32 =>10,33 =>9,33 =>8,34 =>9,35 =>9,36 =>10,37 =>11,38 =>12,38 =>13,38 =>14,38 =>15,39 =>16,40 =>17,41 =>18,42 =>17,43 =>17,44 =>16,45 =>15,44 =>14,45 =>13,44 =>12,43 =>11,44 =>10,44 =>9,45 =>10,46 =>11,47 =>12,48 =>13,49 =>14,48 =>15,48 =>16,48 =>17,49 =>18,48 =>19,48 =>20,49 =>21,48 =>21,47 =>21,46 =>20,45 =>19,44 =>20,43 =>21,43 =>22,44 =>23,44 =>24,44 =>25,43 =>26,44 =>27,43 =>28,43 =>29,43 =>30,42 =>30,41 =>31,40 =>31,39 =>30,38 =>30,37 =>31,36 =>32,37 =>33,36 =>33,35 =>32,34 =>32,33 =>33,32 =>34,32 =>35,31 =>36,32 =>36,33 =>37,34 =>38,35 =>38,36 =>38,37 =>39,38 =>39,39 =>39,40 =>40,41 =>41,41 =>42,40 =>43,41 =>44,40 =>44,39 =>45,38 =>45,37 =>44,36 =>43,37 =>42,37 =>41,36 =>42,35 =>43,34 =>43,33 =>44,32 =>43,31 =>44,30 =>43,29 =>42,29 =>41,28 =>42,27 =>41,26 =>40,25 =>39,24 =>39,23 =>38,22 =>37,23 =>37,24 =>36,25 =>35,26 =>35,27 =>35,28 =>34,29 =>33,29 =>32,29 =>31,28 =>32,27 =>33,26 =>33,25 =>32,24 =>33,23 =>34,22 =>34,21 =>33,20 =>34,19 =>33,18 =>34,17 =>35,16 =>34,15 =>33,14 =>32,13 =>33,12 =>33,11 =>33,10 =>33,9 =>34,8 =>35,8 =>36,9 =>35,10 =>36,11 =>37,12 =>38,12 =>39,13 =>40,12 =>41,13

**Path for UCS:** 7,14 =>8,15 =>9,14 =>10,14 =>11,15 =>12,16 =>13,16 =>14,17 =>15,17 =>16,17 =>17,17 =>18,18 =>19,19 =>20,20 =>21,20 =>22,20 =>23,21 =>24,22 =>25,23 =>26,24 =>27,25 =>28,26 =>29,27 =>30,28 =>31,28 =>32,27 =>33,26 =>34,26 =>35,25 =>36,24 =>37,23 =>38,22 =>39,21 =>40,20 =>40,19 =>39,18 =>39,17 =>40,16 =>41,15 =>42,14 =>41,13
**Cost**:51.88999999999996

**From this we can say, A* Search and UCS are optimal, BFS can sometimes be optimal, DFS is not optimal.**

***Time Complexity***

- **Branching Factor(b)**: In computing, tree data structures, and game theory, the branching factor is the number of children at each node, the outdegree. If this value is not uniform, an average branching factor can be calculated.Maximum braching factor of the search tree.
- **Depth(d)**: The depth of a node is the number of edges from the node to the tree's root node.Depth of the optimal solution.
- **C***: Cost of the optimal solution
- **m**: Maximum depth of search space

- **BFS:** O($b^d$)
- **UCS:** Number of nodes with g(n) <= C*
- **DFS:** O($b^m$)
- **A*:** Number of nodes with g(n) <= C*

In my experiment, Uniform Cost Search took the most time almost everytime, A* is faster than UNS and BFS was constantly the fastest.DFS was giving error on large scale problems but when it doesn't it was between UCS and A*.

***Space Complexity***

- **Branching Factor(b)**: In computing, tree data structures, and game theory, the branching factor is the number of children at each node, the outdegree. If this value is not uniform, an average branching factor can be calculated.Maximum braching factor of the search tree.
- **Depth(d)**: The depth of a node is the number of edges from the node to the tree's root node.Depth of the optimal solution.
- **C***: Cost of the optimal solution
- **m**: Maximum depth of search space

- **BFS:** O($b^d$)
- **UCS:** Number of nodes with g(n) <= C*
- **DFS:** O(b.m)
- **A*:** Number of nodes with g(n) <= C*

| | | |
|---|---|---|
| map1.PNG | 8 gün önce | 10.8 kB |
| Test 0 | 9 dakika önce | 120 B |
| Test 0-astar | 8 gün önce | 1.63 MB |
| Test 0-bfs | 8 gün önce | 1.2 MB |
| Test 0-dfs | 9 dakika önce | 104 B |
| Test 0-ucs | 8 gün önce | 5.33 MB |
| Test 1 | 8 gün önce | 40.4 kB |
| Test 1-astar | 8 gün önce | 27.1 MB |
| Test 1-bfs | 8 gün önce | 14.5 MB |
| Test 1-dfs | 8 gün önce | 114 MB |
| Test 1-ucs | 8 gün önce | 59.3 MB |
| Test 2 | 8 gün önce | 90.6 kB |
| Test 2-astar | 8 gün önce | 61.9 MB |
| Test 2-bfs | 8 gün önce | 43.8 MB |
| Test 2-dfs | 8 gün önce | 17.8 MB |
| Test 2-ucs | 8 gün önce | 171 MB |
| Test 3 | 8 gün önce | 251 kB |
| Test 3-astar | 8 gün önce | 175 MB |
| Test 3-bfs | 8 gün önce | 90.2 MB |
| Test 3-dfs | 8 gün önce | 4.05 GB |
| Test 3-ucs | 8 gün önce | 344 MB |

## Another Creative Example For Problem 1

In this problem we have a map and rows represents cities and columns are neighbors of that city(diagonals are the cities with themselves) I have shown neighbors with 0 and not neighbors with 1 but we have a problem. In this map some of the cities are effected by coronavirus, so if we want to go from point A to point B we can't use the cities effected by coronavirus.

In this example

My map is the same map from above. It is randomly generated , every coordinate is represented as nodes, we have one starting node(randomly generated) this represent starting city and a finish node(randomly generated) this represent where the target city.Driver's aim is to go to target without visiting cities effected by coronavirus.And for this we have randomly generated list of cities effected by coronavirus.

For creating a map , I create a square map with given size, for this experiment i will be using 300, 400, 500 and 600 to compare my algorithms.

I have tried both problems with lesser sizes but there was no difference between them so i set my starting maps high.

# EXPERIMENTS FOR ANOTHER EXAMPLE ON PROBLEM 1

For experimenting Problem 1, i have created four different square maps with(100x100, 200x200, 300x300, 400x400) sizes.

**Corona List** = [0, 185, 136, 275, 159, 143, 192, 188, 39, 174, 187, 182, 171, 202, 168, 94, 157, 166, 138, 229, 42, 160, 9, 105, 10, 196, 234, 107, 280, 10]
Start coordinate :275,44
End coordinate :267,214

astar : 300x300-> runtime = 0.9166512489318848
bfs : 300x300-> runtime = 0.08203816413879395
dfs : 300x300-> runtime = 6.033273696899414
ucs : 300x300-> runtime = 0.8325905799865723

**Corona List** = [188, 217, 198, 378, 163, 251, 126, 83, 194, 144, 66, 346, 322, 177, 42, 201, 324, 231, 156, 334, 28, 59, 229, 386, 332, 326, 56, 315, 272, 77, 105, 244, 369, 330, 77, 222, 97, 273, 216, 263]
Start coordinate :11,89
End coordinate :103,347

astar : 400x400-> runtime = 0.129018402099609375
bfs : 400x400-> runtime = 0.10400233268737793
dfs : 400x400-> runtime = 26.26977562904358
ucs : 400x400-> runtime = 1.4102897644042969

**Corona List** = [209, 324, 63, 480, 473, 302, 390, 63, 107, 161, 146, 20, 278, 417, 48, 100, 263, 168, 465, 245, 431, 241, 42, 461, 76, 361, 29, 385, 12, 205, 359, 217, 76, 215, 12, 176, 365, 250, 436, 249, 304, 178, 481, 361, 114, 118, 396, 213, 97, 467]
Start coordinate :10,390
End coordinate :70,392

astar : 500x500-> runtime = 6.824112640742357
bfs : 500x500-> runtime = 3.1357892345621684
dfs : 500x500-> runtime = 74.80299186706543
ucs : 500x500-> runtime = 15.689256267815321

**Corona List** = [19, 28, 434, 117, 36, 515, 197, 254, 542, 504, 341, 180, 594, 479, 231, 86, 575, 482, 581, 556, 124, 423, 421, 346, 439, 121, 545, 559, 566, 591, 64, 501, 345, 178, 387, 319, 12, 505, 399, 5, 149, 110, 599, 196, 450, 224, 462, 321, 446, 286, 326, 497, 588, 351, 283, 327, 585, 263, 332, 351]
Start coordinate :205,372
End coordinate :571,50

astar : 600x600-> runtime = 12.12438509812367
bfs : 600x600-> runtime = 3.9543874321490
dfs : 600x600-> runtime = 54.11534833908081
ucs : 600x600-> runtime = 17.4451236015148

Our analysis is same as above, just wanted to test another creative example for problem 1.But our paths are too long to paste here.

| | | |
|---|---|---|
| map1.PNG | 8 gün önce | 10.8 kB |
| Test 0 | 12 dakika önce | 90.6 kB |
| Test 0-astar | 12 dakika önce | 7.85 MB |
| Test 0-bfs | 12 dakika önce | 1.6 MB |
| Test 0-dfs | 12 dakika önce | 262 MB |
| Test 0-ucs | 12 dakika önce | 6.41 MB |
| Test 1 | 12 dakika önce | 161 kB |
| Test 1-astar | 12 dakika önce | 225 kB |
| Test 1-bfs | 12 dakika önce | 48.4 kB |
| Test 1-dfs | 12 dakika önce | 905 MB |
| Test 1-ucs | 12 dakika önce | 150 kB |
| Test 2 | 12 dakika önce | 251 kB |
| Test 2-astar | 11 dakika önce | 124 B |
| Test 2-bfs | 11 dakika önce | 111 B |
| Test 2-dfs | 10 dakika önce | 1.97 GB |
| Test 2-ucs | 10 dakika önce | 118 B |
| Test 3 | 10 dakika önce | 361 kB |
| Test 3-astar | 10 dakika önce | 18.6 MB |
| Test 3-bfs | 10 dakika önce | 3.14 MB |
| Test 3-dfs | 9 dakika önce | 1.58 GB |
| Test 3-ucs | 8 dakika önce | 11.3 MB |

## Problem 2

For problem 2, i choose 8 puzzle problem.My scenario is, our computer got hacked by Hacettepe CS student and he/she encrypted our photos with 8 puzzle problem and she/he also encrypted our sister's favorite photo if she sees encrypted photo, she will be upset and hearthbroken to see her favorite photo in that mess, and she is coming to our room you can hear her footsteps.From footsteps we assume we have 5 seconds to solve this mess.Can you select the correct search algorithm for this? or will you see your sister sad?

Basically 8 puzzle proglem is puzzle isgiven a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match final configuration using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space.

| Goal: | Easy: | Medium: | Hard: | Worst: |
|---|---|---|---|---|
| 1 2 3 | 1 3 4 | 2 8 1 | 2 8 1 | 8 7 |
| 4 5 6 | 8 6 2 | 4 3 | 4 6 3 | 6 5 4 |
| 7 8 | 7 5 | 7 6 5 | 7 5 | 3 2 1 |

In our experiment, we will try worst case to see differences of algorithms.

This is how I run andgive input map to program: python AI.py dfs 0,8,7,6,5,4,3,2,1

In A* Search, heuristic was to order the moves based on the number of tiles out of place.I also took into consideration how far away the out of place tiles was from their correct position (called Manhattan distances). This entire process determines the heuristic value (H). I can then take H at any given time and successively find the path to a solution. But this will not provide us with the shortest solution.

# EXPERIMENTS FOR PROBLEM 2

**python AI.py dfs 0,8,7,6,5,4,3,2,1**

# AST:

path_to_goal: ['Right', 'Down', 'Down', 'Right', 'Up', 'Up', 'Left', 'Down', 'Down', 'Left', 'Up', 'Right', 'Down', 'Right', 'Up', 'Up', 'Left', 'Left', 'Down', 'Right', 'Up', 'Right', 'Down', 'Left', 'Down', 'Right', 'Up', 'Up', 'Left', 'Left']

cost_of_path: 30

nodes_expanded: 12893

fringe_size: 6327

max_fringe_size: 6328

search_depth: 30

max_search_depth: 30

running_time: 3.70171429

# **BFS**:

path_to_goal: ['Down', 'Right', 'Down', 'Right', 'Up', 'Up', 'Left', 'Down', 'Down', 'Right', 'Up', 'Up', 'Left', 'Down', 'Down', 'Left', 'Up', 'Up', 'Right', 'Down', 'Down', 'Left', 'Up', 'Right', 'Down', 'Right', 'Up', 'Left', 'Up', 'Left']

cost_of_path: 30

nodes_expanded: 181423

fringe_size: 16

max_fringe_size: 24048

search_depth: 30

max_search_depth: 31

running_time: 8.10659433

# **DFS**:

path_to_goal:[“ It takes 79 word page to print all paths so I cant print paths.”]

cost_of_path: 41910

nodes_expanded: 144633

fringe_size: 30358

max_fringe_size: 42826

search_depth: 41910

max_search_depth: 65982

running_time: 6.66473980

# UCS:

path_to_goal: ['Right', 'Down', 'Down', 'Right', 'Up', 'Up', 'Left', 'Down', 'Down', 'Left', 'Up', 'Right', 'Down', 'Right', 'Up', 'Up', 'Left', 'Left', 'Down', 'Right', 'Up', 'Right', 'Down', 'Left', 'Down', 'Right', 'Up', 'Up', 'Left', 'Left']

cost_of_path: 30

nodes_expanded: 123040

fringe_size: 18

max_fringe_size: 28

search_depth: 30

max_search_depth: 32

running_time: 11.18519917

# ANALYSIS FROM PROBLEM 2

How do we evaluate and compare search algorithms?

- **Completeness** : Is it guaranteed to find a solution (if one exists)?
- **Optimality** : Does it find the “best” solution (if there are more than one)?
- **Time Complexity** : How long does it take to find a solution?
- **Space Complexity** : How much memory does it require?

***Completeness***

From these experiment results, we can see that all four algorithms was able to produce solution.But from BBM405 lesson experience, I know that Depth First Search is not complete.Maybe my case(3x3) is not complex enough.In the future, I may try more complex case to see if DFS will produce solution or not.

***Optimality***

There can be multiple solutions but which algorithm/s can choose the best solution?. To find this i have tested my algorithms and i will compare the paths they found.

AST: ['Right', 'Down', 'Down', 'Right', 'Up', 'Up', 'Left', 'Down', 'Down', 'Left', 'Up', 'Right', 'Down', 'Right', 'Up', 'Up', 'Left', 'Left', 'Down', 'Right', 'Up', 'Right', 'Down', 'Left', 'Down', 'Right', 'Up', 'Up', 'Left', 'Left']

cost_of_path: 30

BFS: ['Down', 'Right', 'Down', 'Right', 'Up', 'Up', 'Left', 'Down', 'Down', 'Right', 'Up', 'Up', 'Left', 'Down', 'Down', 'Left', 'Up', 'Up', 'Right', 'Down', 'Down', 'Left', 'Up', 'Right', 'Down', 'Right', 'Up', 'Left', 'Up', 'Left']

cost_of_path: 30

DFS:[" It takes 79 word page to print all paths so I cant print paths."]

cost_of_path: 41910

UCS: ['Right', 'Down', 'Down', 'Right', 'Up', 'Up', 'Left', 'Down', 'Down', 'Left', 'Up', 'Right', 'Down', 'Right', 'Up', 'Up', 'Left', 'Left', 'Down', 'Right', 'Up', 'Right', 'Down', 'Left', 'Down', 'Right', 'Up', 'Up', 'Left', 'Left']

cost_of_path: 30

**As we see here, A* search and UCS finds the optimal solution with same cost, same buth but BFS and DPS finds solution but these solutions are not optimal.BFS finds a path with same cost but different path.If our problem was more complex I believe the cost would be different as well.DFS finds a path but it costs 41910 and it takes 79 Word page to print the path.**

***Time Complexity***

- **Branching Factor(b)**: In computing, tree data structures, and game theory, the branching factor is the number of children at each node, the outdegree. If this value is not uniform, an average branching factor can be calculated.Maximum braching factor of the search tree.
- **Depth(d)**: The depth of a node is the number of edges from the node to the tree's root node.Depth of the optimal solution.
- **C***: Cost of the optimal solution
- **m**: Maximum depth of search space

- **BFS:** $O(b^d)$

- **UCS:** Number of nodes with g(n) <= C*
- **DFS:** O($b^m$)
- **A*:** Number of nodes with g(n) <= C*

In my experiment, Uniform Cost Search took the most time almost everytime, A* was fastest following DFS and BFS.Their running time was:

- AST: running_time: 3.70171429
- BFS: running_time: 8.10659433
- DFS: running_time: 6.66473980
- UCS: running_time: 11.18519917

More complex puzzle and input is needed to see better difference in Space and Time Complexity.

***Space Complexity***

- **Branching Factor(b)**: In computing, tree data structures, and game theory, the branching factor is the number of children at each node, the outdegree. If this value is not uniform, an average branching factor can be calculated.Maximum braching factor of the search tree.
- **Depth(d)**: The depth of a node is the number of edges from the node to the tree's root node.Depth of the optimal solution.
- **C***: Cost of the optimal solution
- **m**: Maximum depth of search space

- **BFS:** O($b^d$)
- **UCS:** Number of nodes with g(n) <= C*
- **DFS:** O(b.m)
- **A*:** Number of nodes with g(n) <= C*

This is the memory usage i got from running these algorithms.

- Dfs : used=6192308224,
- Ast: used=6168481792,
- Ida: used=6161678336,
- Bfs: used=6201212928,

As I said before, More complex puzzle and input is needed to see better difference in Space and Time Complexity.

Even ith 4x4 15 Puzzle, it was taking too much time and too much RAM( Python was using +4 GB RAMs and my computer is frozen multiple times so I couldn't scale this problem.

# REFERENCES


https://github.com/chitholian/AI-Search-Algorithms

https://sites.cs.ucsb.edu/~yuxiangw/classes/CS165A-2019winter/Lectures/Lecture-11.pdf

https://en.wikipedia.org/wiki/A*_search_algorithm

https://math.wikia.org/wiki/Uniform_cost_search

https://en.wikipedia.org/wiki/Breadth-first_search

https://en.wikipedia.org/wiki/Depth-first_search

https://images.slideplayer.com/31/9777163/slides/slide_7.jpg

https://slideplayer.com/slide/1512813/

https://slideplayer.com/slide/1512813/5/images/59/Comparison+of+search+strategies.jpg

https://www.geeksforgeeks.org/8-puzzle-problem-using-branch-and-bound/

https://www.d.umn.edu/~jrichar4/8puz.html

https://github.com/speix/8-puzzle-solver

https://www.geeksforgeeks.org/search-algorithms-in-ai/