**First Name : Ashwin Kumar**
**Last Name : Gopi Valliammal**
**Email : gopivall@usc.edu**
**Course : CSCI 561 Summer 09**
**Date : July-17-2009**

# Project 1 : Search Algorithms

## Introduction :

The focus of project 1 is mainly on the implementation of the search algorithms like Breath First Search(BFS), Depth First Search(DFS), Uniform Cost Search (UCS) and A* search. We are to implement these search algorithms in the pacman maze. The pacman maze is represented in a tree structure with the root node representing the initial position of pacman in the maze and the goal node representing the food position. The pacman uses this implemented search algorithms to find its way through the maze to the food. After successfully reaching the food the pacman exits printing the total path cost from the initial position and the total number of search nodes expanded.

## Files Submitted :

1.ReadMe

2.search.py

3.util.py

4.codes.txt

5.Project1Documentation.pdf

## Files Edited :

1.search.py - Search algorithms are implemented

2.util.py - The pop function in priority queue is altered to return the item and its priority

## Implementation :

### Question 1 : Depth-first search (DFS) algorithm

The Depth first search is implemented using two lists one is a open list which is a stack and the other is closed list which is a queue. The open list contains the list of nodes that are expanded and the closed list contains the list of nodes that are visited already. When a goal node is reached the closed list contains all the nodes that were visited during the search along with the goal node. A separate dictionary is maintained to keep track of the parent child relationship. Once we reach the goal node we use the dictionary to back track till the root node and this sequence is the list of moves that is returned from this function. The pacman moves in the puzzle according the moves list that is returned from this function

**Question 2 : Breath-first search (BFS) algorithm .**

The implementation of the BFS is very similar to DFS the only difference is that instead of a stack to represent a open list we make use of a queue.

**Question 3 : Uniform cost search (UCS) algorithm**

The implementation of the UCS is very similar to the BFS but instead of the queue to represent a open list we make use of a priority queue. This priority queue is always sorted. The expanded nodes are inserted with a priority that is equal to the total cost of the path of the expanded node from the root node.

**Question 4: A* Search**

A * search is a combination of uniform cost search with greedy approach. This is very similar to the UCS. The only difference is that the priority is a summation of the distance of the expanded node from root node plus a heuristic function. Mostly the heuristic function is a Straight Line Distance(SLD) function.

## Testing and Sample Output :

**Note: I have changed util.py and search.py both of these files that I submit are needed for testing my code. Please make sure that you are using the util.py that I submit along with this document**

**Tiny Maze**

python pacman.py -l tinyMaze -p SearchAgent

Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 23
Pacman emerges victorious! Score: 500

python pacman.py -l tinyMaze -p SearchAgent -a fn=bfs

Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 31
Pacman emerges victorious! Score: 502

python pacman.py -l tinyMaze -p SearchAgent -a fn=ucs

Path found with total cost of 8 in 0.1 seconds
Search nodes expanded: 28
Pacman emerges victorious! Score: 502

python pacman.py -l tinyMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

Path found with total cost of 8 in 0.1 seconds
Search nodes expanded: 14
Pacman emerges victorious! Score: 502

**Medium Maze**

python pacman.py -l mediumMaze -p SearchAgent

Path found with total cost of 130 in 0.4 seconds
Search nodes expanded: 204
Pacman emerges victorious! Score: 380

python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs

Path found with total cost of 68 in 0.8 seconds
Search nodes expanded: 547
Pacman emerges victorious! Score: 442

python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

Path found with total cost of 68 in 1.1 seconds
Search nodes expanded: 542
Pacman emerges victorious! Score: 442

python pacman.py -l mediumMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

Path found with total cost of 68 in 0.8 seconds
Search nodes expanded: 409
Pacman emerges victorious! Score: 442

**Big Maze**

python pacman.py -l bigMaze -z .5 -p SearchAgent

Path found with total cost of 210 in 1.1 seconds
Search nodes expanded: 660
Pacman emerges victorious! Score: 300

python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=bfs

Path found with total cost of 210 in 2.4 seconds
Search nodes expanded: 1236
Pacman emerges victorious! Score: 300

python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=ucs

Path found with total cost of 210 in 4.1 seconds
Search nodes expanded: 1233
Pacman emerges victorious! Score: 300

python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

Path found with total cost of 210 in 3.4 seconds
Search nodes expanded: 1062
Pacman emerges victorious! Score: 300

**Others**

python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

Warning: this does not look like a regular search maze
Path found with total cost of 1 in 0.7 seconds
Search nodes expanded: 376
Pacman emerges victorious! Score: 646

python pacman.py -l mediumScaryMaze -p StayWestSearchAgent

Path found with total cost of 68719479864 in 0.4 seconds
Search nodes expanded: 176
Pacman emerges victorious! Score: 418

## Results and Inference :

| Search Fn / Maze | DFS | BFS | UCS | A* |
|---|---|---|---|---|
| **TINY MAZE** | PC=10<br>NE=23 | PC=8<br>NE=31 | PC=8<br>NE=28 | PC=8<br>NE=14 |
| **MEDIUM MAZE** | PC=130<br>NE=204 | PC=68<br>NE=547 | PC=68<br>NE=542 | PC=68<br>NE=409 |
| **BIG MAZE** | PC=210<br>NE=660 | PC=210<br>NE=1236 | PC=210<br>NE=1233 | PC=210<br>NE=1062 |

**Legends : PC=Path cost , NE=Nodes expanded**

The above tables gives the results of the search algorithm on different sizes of the mazes. As it can be seen from the above table A* expands less number of nodes when compared to BFS and UCS in all the 3 sizes of mazes which is correct since A* is optimal.