Yujia Li A98064697
yul200@ucsd.edu

CSE150 Assignment 1
Problem 6

Problem 2

For the problem 2 that is give us a heads up for using BFS to solve random input matrix puzzles with random conditions. So, Breath-First Search (graph data strctures) that we used to solve the question. The Breath-First Search algorithm starts at a vertex I and visits, first the neighbors of i, then the neighbors of the neghbors of i, so on…it uses a queue(FIFO) that initially contains only i. It then repeatedly extracts an element from q and adds its neighbors to q and check these neighbors hve never been in q before. The important rule of BFS is the algorithm for graphs has to ensure that it doesn't add the same vertex to q more than once. So, for adding each vertex(point) from q takes constant time per vertex for a total of O(n) time. Since each vertex is processed at most once by the inner loop, each adjacency list is processed at most once, so each edge is processed at most once. So, the total of BFS time running in O(n+m) time.

Description of the problem and the algorithms used

The problem is to implement a algorithm to solve random input puzzles, see if it can be solved by the 8-puzzle rules, no matter how many steps it takes. The algorithm used is BFS, which is one of the graph-search algorithms.

Data structure used

The data structure used to implement BFS graph search is Queue(FIFO). In this way, the positions at level n would be inserted before any positions at level n+1, and de-queue before any positions at level n+1 as well, which is exactly by the rules of BFS.

Problem 3

Depth-First Search is similar to eh algorithm for traversing binary trees. And it first fully explores one sub tree before returning to the current node and then

exploring the other sub tree. But different between using DFS and BFS is, DFS need using stack instead of a queue. Basically, the DFS is as a recursive algorithm, it stats by visiting r, when visiting a vertex i, we first mark i as grey. Next we scan i's adjacency list and recursively visit any untouched vertex we find in the list. Then it is done with processing i, and color untouched to touched and return. The algorithm for each run on

Description of the problem and the algorithms used

The problem is to implement a algorithm to solve random input puzzles, see if it can be solved by the 8-puzzle rules, and it can take no more than 5 steps. The algorithm used is DFS.

Data structure used

The data structure used to implement DFS graph search is Stack(FILO). In this way, the positions at level n would be inserted before any positions at level n+1, and de-queue after any positions at level n+1, which is exactly by the rules of DFS, search down to a path till dead end, come back and start another path.

Problem 4

IDDFS is a state space search strategy in which a depth-limited search is run repeatedly, increasing the depth limit with each iteration until it reaches d, the depth of the shallowest goal state. IDDFS equivalent to BFS but uses much less memory, on each iteration. It visits the nodes in the search tree in the same order as depth-first search, but the cumulative order in which nodes are first visited is effectively breath-first.

Description of the problem and the algorithms used

The problem is to implement a algorithm to solve random input puzzles, see if it can be solved by the 8-puzzle rules, and it can take no more than 12 steps. The algorithm used is Deeping depth-first algorithm.

Data structure used

The data structure used to implement Deeping depth-first search is Stack(FILO). In this way, the positions at level n would be inserted before any positions at level n+1, and de-queue after any positions at level n+1, which is exactly by the rules of DFS, search down to a path till dead end, come back and start another path. By limit the

depths it can take each iteration and call it 12 times. If the results is found before 12 times, return the result.

Problem 5

A* is one efficient algorithm to find the two closed points, it is better than Dijkstra that only knowing searching around randomly, but A* is very optimal or must be leass than or equal to the real cost. In A*, there must has two node, one we call OPEN and other one we call CLOSED, so take smallest node from open and N node is the target node, looking for the X node along the way if X node has value(weights) less than N then, put X in OPEN, and Put N node into CLOSE. Follow the value(weights) to make order of node in OPEN. So, A* is very efficient way to calculate point to point, much better than BFS or DFS.

Description of the problem and the algorithms used

The problem is to implement a algorithm to solve random input puzzles, see if it can be solved by the 8-puzzle rules. The algorithm used is A* algorithm, with Manhattan distance as the heuristic.

Data structure used

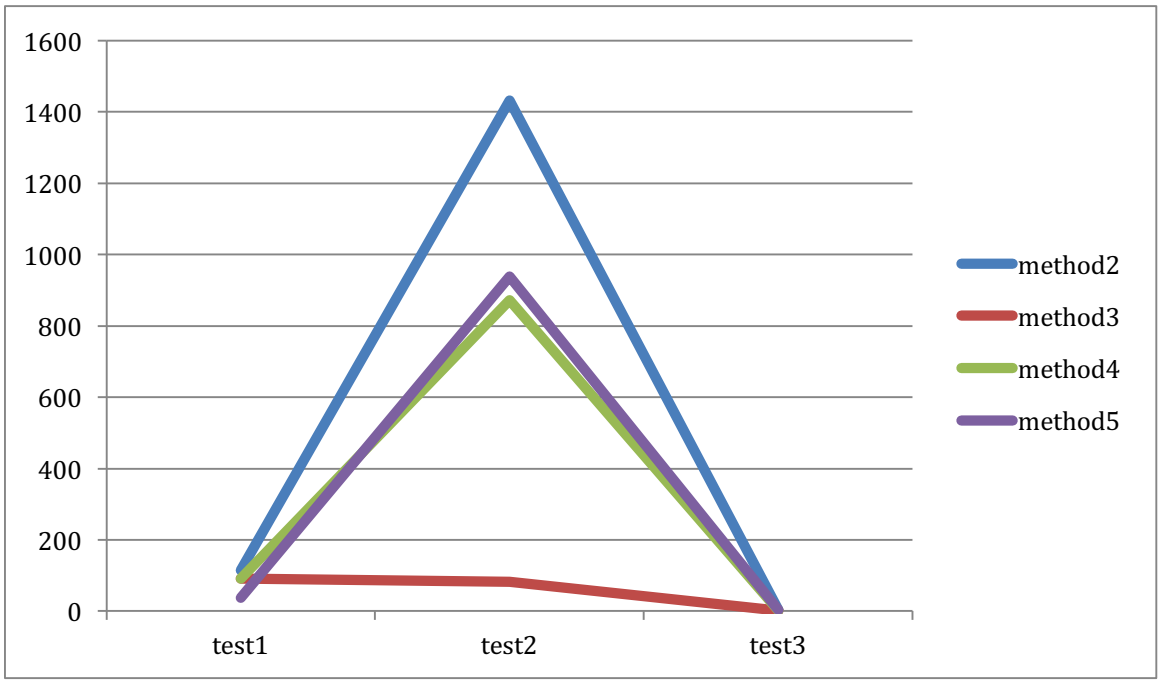
The data structure used to implement A* graph search is Priority Queue. In this way, each position would be sorted every time there is a new position inserted into the Priority Queue based on their priority. Their priority , in this situation, is the addition of steps it visited and the steps to the destination.

The below table shows how many different stages/different tables each method need to go over to find the correct solution:

	Method 2	Method 3	Method 4	Method 5	Actual steps
6,2,0,3,4 1,5,7,8,9 10,11,12,13,14 15,16,17,18,19 20,21,22,23,24	114 0m0.053s	91 0m0.038s	91 0m0.052s	37 0m0.038s	LDRUL
5,4,2,3	1433	0m0.032s	873	938	DLLUULDRUL

1,9,6,0 8,10,11,7	0m0.210s	unsolvable	0m0.265s	0m0.133s	
1,4,2 3,0,5	4 0m0.024s	2 0m0.027s	2 0m0.026s	3 0m0.029s	UL
Time Complexity	b^d	b^m	b^d	b^d	
Space Complexity	b^d	bm	bd	b^d	

Result of the analysis with graph



As above graph showing, the Breath-first search will take more time complexity and space complexity to the others, due to BfS searching from no direction and randomly searching, so it will take more time and space then IDDFS and A*. And, DFS actually is off of this test, because for the testing, it isn't passed. Due to the algorithm of DFS, and also base on the condition rules, so DFS is kill at very beginning search, so the data will be incomplete. So, we call DFS for this kind question unstable search, we can see from the table, on the test by given 3 by 2 matrix, DFS works almost same and can compete BFS

and IDDFS. So, DFS is unstable and only for some particular questions. IDDFS is a moderate search base on DFS that go with levels, it just level search for each time, when can't find target or closed point go back the parent node see there has any other sub tree or leaf can go to find target point, if not return to the top. A* is good for taking care of two closest points, but it also depend on the distance of the target, if the destination is far away, A* will need search more points, so the time and space will reduce by it.