CS 331 - Assignment 1

Wenbo Hou, Zhi Jiang

April 21, 2017

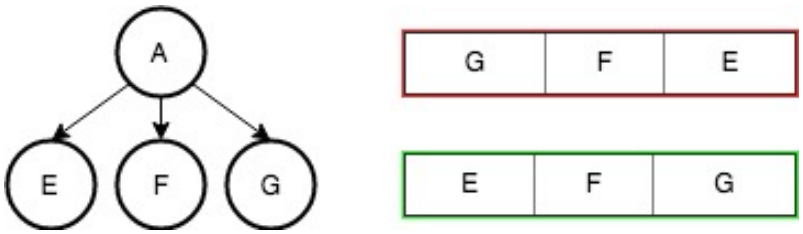
1 Methodology

1.1 BFS

In graph search pseudo-code, to check the node is goal state or not after frontier pops a node. But in BFS pseudo-code, after expanding a node, to check each child of the node is goal state or not. In our implement, we followed the steps of BFS pseudo-code to check node.

1.2 DFS

As we know, There is a LIFO queue to store nodes we will use in the DFS. But there is a trap if we don't understand the principle of DFS completely. In the following tree. G is the last node created by expanding function as a child of A, so in the following first queue (red frame), it will be stored into the first position of the queue and then the queue will pop it next time. It is incorrect obviously that the order of nodes in the queue because in DFS the queue should pop E firstly. Thus, the correct queue should be like the second one (Green frame).



So in our codes, we use a variable i as an index to ensure each child node is at the correct position. According to following codes, you can see when the first child node is created, it will be stored into the first position of the queue and i is still changed. So next child node will be stored correctly into the second position of the queue.

```
i = 0
for child in child_nodes:
    ....
    if is_not_in(child, frontier) and (not explored.has_key(key)):
        frontier.insert(i, child)
        i += 1
```

1.3 IDDFS

In IDDFS, the depth limit we set in from 0 to infinite, because in fact, we wouldn't know a good depth cutoff in advance. If we just set a particular value, probably it will be a too shallow limit and it could not find a solution. On the other hand, we still implemented a hash table to store repeated states.

1.4 A-Star

In A-Star algorithm, $g(n)$ is the path cost from the start node to n so the path cost is 1 between two nodes. In term of heuristic function, The admissible heuristic is different between the number

of human on the left bank of goal and current state. For example, in case 3, the current state is that there are 30 missionaries and 15 cannibals on the left bank. As we know there are 100 missionaries and 90 cannibals, so the admissible heuristic is $(100 - 30) + (90 - 15) = 145$. The number of humans on current state always is less than or equal to goal, so an admissible heuristic will never overestimate the cost to reach the goal. So when the search approaches to the goal, the admissible heuristic will decrease.

2 Results

	BFS	**DFS**	**IDDFS**	**A-Star**
Case 1	Solution: 12 Expanded: 13	Solution: 12 Expanded: 12	Solution: 12 Expanded: 85	Solution: 12 Expanded: 13
Case 2	Solution: 34 Expanded: 69	Solution: 34 Expanded: 34	Solution: 34 Expanded: 1197	Solution: 34 Expanded: 68
Case 3	Solution: 378 Expanded: 2181	Solution: 378 Expanded: 378	Solution: 378 Expanded: 410594	Solution: 378 Expanded: 2180

3 Discussion

In term of results, they are almost what we expected. BFS, IDDFS, and A-Star should have the same solution because all of them can gain the optimal solution. We thought the DFS should have a large number of nodes on the solution because it cannot ensure to find the optimal solution. But actually it still found the optimal solution. There is another interesting thing in results of the DFS, which is we found the number of nodes on the solution is equal to the number of expanded nodes on DFS for each case. We tried to change the order on the action, and then we got different results about the number of nodes on the solution and number of expanded nodes. The right path is probably changed from the most left path to the most right path after changing the order on the action because different order on the action will make a different structure of the tree. In our opinion, this algorithm found the right path fortunately at the beginning thus it did not get a deeper solution and did not expand more nodes.

On the other hand, when we focus on the number of expanded nodes, the IDDFS has the most number because it repeated the early parts of the search as it deepens. The number of expanded nodes in each case of BFS and A-Star are very close. We think the cause is the A-Star and the BFS both are complete. As for the run time, all of the algorithms can quickly output solution except IDDFS. IDDFS spent more time on case 3 than other algorithms. The reason is IDDFS has to start with root as the depth is changed.

4 Conclusion

According to these results, we learned many useful techniques in these algorithms. First of all, as we know, the disadvantage of DFS is that there exists a possibility that it may go down a path forever. So the principle of IDDFS can effectively resolve this issue because it can impose a cutoff depth to limit this search. If the depth of a search excess the limit, it will be cutoff so it would never get stuck in a path forever. Secondly, we understand the importance of heuristics. In our program, the heuristic function should be good because it still found the optimal solution for each case. Finally, the explored node list plays an important role in each algorithm. When we expand a node, we have to check whether it have been visited by this list, so it can prevent repeated nodes and redundant paths. Repeated nodes and redundant paths will cause invalid search, even make the search get stuck in a path forever.

We know BFS, IDDFS and A-Star all could find the optimal solution, but as for spaces complexity in this game, it is clear that the number of expanded nodes of A-Star is less than BFS and IDDFS. Although DFS has the least number of expanded nodes, it is uncertain to find the optimal solution by using this search strategy. On the other hand, these number of missionaries and cannibals in each case are not large enough, so we cannot find an obvious difference on time among them except IDDFS. In summary, We think the A-Star should be the best, but pre-condition is the A-Star must have a good heuristic because good heuristic can reduce the time significantly.