Joel Alexis Bustamante A00226215

Nancy Espinosa Castillo A01206782

Lab Report: Implementing (Un)informed Search Algorithms

Which heuristics did you use for the A* algorithm?

The used heuristics where the following ones:

- Consistent heuristic: The number of blocks that are incorrectly placed in each stack.
- Non-consistent heuristic: It was used a fixed value of 1, this number underestimates the cost if at least is needed one movement (the minimum possible cost with a movement is 2).

Test your program with a couple of different problems. Increase the size of the problem to test the limits of your program. Make a table comparing **how many nodes are searched** to find the answer for each problem. For this table, you should compare a number of different problems (at least 3) to avoid a statistical bias. **Which of the four algorithms searches the least nodes and which one take the most?**

Test 1:

```
3
(A); (B); (C); ()
(); (A); (B); (C)
```

Test 2:

```
3
(A); (B); (C); (D)

(X); (X); (X); (A, B, C)
```

Test 3:

```
4

(A, B, C, D); (); (); (E)

(X); (X); (E, A); (B, C, D)
```

	Visited nodes		
Algorithm	**Test 1**	**Test 2**	**Test 3**
A* (consistent)	35	309	2928
A* (Non-consistent)	38	510	3885
BFS	79	1647	8147

DFS	74	1632	Without solution

If the complexity of the problem increases, it will increase the number of nodes to visit. As you can see in the table the difference between the algorithms is every time higher. The consistent A* searches the smallest number of nodes while DFS and BFS the highest.

Why does this happen?

1. A* always tries to find the path with the lowest cost, every step it is closer to the solution, this particular behavior produces that it doesn't need to expand unneeded nodes (if it has an admissible heuristic).
2. DFS and BFS will try to expand all the nodes in the worst case scenario, they do not know anything about the future cost in taking a specific path. Basically, both of them only follows a sequence of nodes until find the solution, if the goal state is closer to the first exploration paths, they will make a good searching but in the contrary case the result will be too expensive.
3. DFS will get stuck in an infinite cycle if it doesn't remember the visited states.

Which algorithms are optimal? Why?

1. A* is an optimal algorithm because maintains as priority queue of options that are ordered. Also this algorithm keeps searching until it finds the best option of a set alternatives and how good can be an alternative be is based on the heuristic and on actual costs found in the search so far, even with an inconsistent heuristic.
2. Although BFS is an algorithm that eventually will find a solution, it is not a optimal algorithm because it does not guarantee that the solution is the best or the shortest path. In some specific situations where every movement cost the same and every possible solution is at the same depth it could be considered optimal.
3. DFS can be an optimal algorithm if the tree is finite, all the pats have the same length and the cost for all the movements is the same. However, in general DFS is not an optimal algorithm because there is nothing that guarantees that it will always find the best path.

In your opinion, what are the benefits of simpler algorithms versus more complex ones?

Simple algorithms can have more advantages to find a solution in a small set of data, because the time to build it and test it is less than a complex algorithm. However, if you want an optimal solution you can achieve it with algorithms that have more complexity.