CPSC 322 Assignment 1

Jose Abraham Torres Juarez - 79507828
Gustavo Martin - 62580121

1 Question 1 (27points): Comparing Search Algorithms

1.1 Depth-first search

(a) What nodes are expanded by the algorithm?

$$\{a, b, c, d, e, f, g, h, i, z\}$$

(b) What path is returned by the algorithm?

$$a- > b- > c- > d- > e- > f- > g- > h- > i- > z$$

(c) What is the cost of this path? 64

1.2 Breadth-first search

(a) What nodes are expanded by the algorithm?

$$\{a, b, e, c, f, g, d, f, g, h, i, z\}$$

(b) What path is returned by the algorithm?

$$a- > e- > g- > z$$

(c) What is the cost of this path? 16

1.3 A*

(a) What nodes are expanded by the algorithm?

$$\{a, e, b, g, c, f, d, z\}$$

(b) What path is returned by the algorithm?

$$a- > e- > g- > z$$

(c) What is the cost of this path? 16

1.4 Branch-and-bound

(a) What nodes are expanded by the algorithm?

$$\{a, b, c, d, e, f, g, h, i, z, i, z, z, g, h, i, z, f, g, e, f, g, h, z, g, h, i, z\}$$

(b) What path is returned by the algorithm?

$$a- > e- > g- > z$$

(c) What is the cost of this path? 16

1.5

(a) Did BFS and B&B find the optimal solution for this graph?
Yes, both algorithm found the best path $a- > e- > g- > z$

(b) Are BFS and B&B optimal in general? Explain your answer.
No, BFS is not optimal if the costs of the arcs are different. And B&B only if the heuristics are admissible.

(c) Did B&B expand fewer nodes than A*? Explain if your answer is true in general for these two algorithms and why.
No, in general B&B will expand more nodes than A*, because even if it finds a solution, it will keep searching in the graph for shorter solutions.

2 Question 2 (36points): Uninformed Search: Peg Solitaire

2.1 Represent peg solitaire as a search problem

(a) How would you represent a node/state?
As a matrix representing the the positions of all the pegs in the board. A peg would be represented as a 1, a free hole would be represented as a 0 and a place where the peg can not move would be represented by a -1.

The following matrix is the representation of the initial state of the game.

$$\begin{bmatrix} -1 & -1 & 1 & 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 & 1 & -1 & -1 \end{bmatrix}$$

(b) In your representation, what is the goal node?
The goal node would have the following representation.

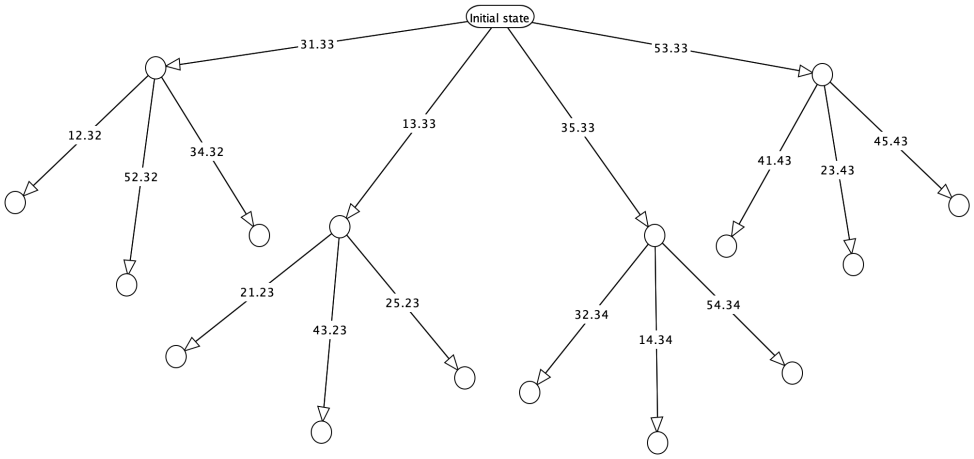
$$\begin{bmatrix} -1 & -1 & 0 & 0 & 0 & -1 & -1 \\ -1 & -1 & 0 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & -1 & -1 \\ -1 & -1 & 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$

(c) How would you represent the arcs?
We would use the following notation $ij.i'j'$, being i the row, j the column of the peg that is moving to the column i' and row j', e.g., an arc with the label 31.33 will represent that a peg in row 3 column 1 will be moved to row 3 column 3.

(d) How many possible board states are there? Note: this is not the same as the number of "valid" or "reachable" game states, which is a much more challenging problem.
There are 33 holes in the board, and the domain of this holes is binary 1,0, either there is a peg on the hole or not. So the number of possible states would be 2^{33}.

2.2 The search tree:

(a) Write out the first three levels (counting the root as level 1) of the search tree. (Only label the arcs; labeling the nodes would be too much work).
The following diagram is the representation of how the search tree would look.



(b) What can you say about the length of the solution(s)?
The solution must be at level 32, because at every step from node to node, we lose one peg, we start with 32 at the goal is to get to 1 peg in all the board.

2.3 The search algorithm:

(a) What kind of search algorithm would you use for this problem? Justify your answer
An uninformed algorithm, because we neither have difference in the costs of the paths nor heuristics. We think that DFS will be a good algorithm to find the solution of this problem.

(b) Would you use cycle-checking? Justify your answer.
No, you can't undo a movement so there is no possibility for cycles.

(c) Would you use multiple-path-pruning? Justify your answer. No, because it is not possible to arrive at the same state with less movements. We can prove this because at every level the number of pegs is reduced by 1, at level 1 we have 32 pegs, at level 2 31 pegs, at 3 we have 30 pegs and so on and so forth. So it is guaranteed that they will not be a shorter path to get to any state.

3 Question 3 (24 Points) Free Cell

3.1 Represent this as a search problem

(a) How would you represent a node/state?
As an array of 16 stacks, each stack represent a zone in the board, 4 free cells, 4 foundations and 8 columns. The free cells stack will have a maximum size of 1.

(b) In your representation, what is the goal node?
The goal node will be when all free cells and columns stacks are empty, and each of the 4 foundation stacks are filled with ascending order cards of the same suit per stack.

(c) How would you represent the arcs?
The arcs will be represented with the information of the movement, from which stack we move the top card to which stack, i.e., column1_freecell1 or column1_heartsFoundation.

3.2

Give an admissible heuristic for this problem; explain why your heuristic is admissible. More points will be given for tighter lower bounds; for example, h=0 is a trivial (and useless) heuristic, and thus it is not acceptable.
An admissible heuristic for this problem will be the number of cards that are not in the foundation stacks, because we know that progress is made when a card is added to the foundation stack. This is pretty optimistic because it is considering that on each move you can send one card to the foundation, this is admissible because it will never be an overestimate.

4 Question 4 (15points) Modified Heuristics

4.1 Reduce h(n), trying in three different ways:

(a) Can A* still find the optimal path?
Yes, in all the tests when reducing $h(n)$ the algorithm successfully returned the optimal path.

(b) Is the efficiency of A* increased or reduced?
When reducing the $h(n)$s of the nodes that are not in the path the efficiency is reduced, on the contrary if the $h(n)$s of the nodes that are in the path the efficiency is increased. If we reduced the $h(n)$s on all the nodes in the graph A* acts in the same way when the $h(n)$s were not reduced.

(c) Try to draw a more general conclusion regarding the changes in efficiency and optimality.
If the heuristics of just the optimal path are reduced the algorithm will be more efficient.

4.2 Set h(n) as the exact distance from n to a goal

To do this, you should add the costs of arcs on the optimal path from every node to the goal node, and set the sum as the heuristic function of the node.

(a) Can A* still find the optimal path?
Yes A* still finds the optimal path.

(b) Is the efficiency of A* increased or reduced?
The efficiency of A* is increased. This would be the best theoretical heuristic.

(c) Try to draw a more general conclusion regarding the changes in efficiency and optimality.
By making the $h(n)$s of each nodes as the exact distance from n to a goal, we are setting the perfect heuristic for the graph. It is in our best interest to come up with a heuristic with values that resemble the actual value to have an efficient algorithm.

4.3 Increase h(n), also trying in three different ways

(a) When can A* still find the optimal path?
Yes, even if the heuristics are not admissible

(b) When is the efficiency of A* improved or reduced?
Efficiency of A* is improved when we choose high heuristics for the non-optimal nodes. Although for this specific case efficiency was improved when increasing the heuristics for the optimal path, this will not always be the case, usually it will reduce efficiency

(c) Try to draw a more general conclusion regarding the changes in efficiency and optimality.
We should use a heuristic that sets high values for non-optimal paths. This is the opposite effect of the **4.1** exercise.