CSE 571 Fall 2022 HW2

Submitted By: Aman Peshin (1225476655)

Exercise 1.1

a. Yes, breadth-first search is a special case of uniform cost search where each edge cost from a parent to a child node is of equal cost. Both algorithms search for the goal state or node in the same fashion. Let's say for a parent node both search techniques expand the child nodes of A, to select which node needs to be expanded next. BFS uses a FIFO queue which will expand the left-most node and then expand the other child nodes of A. Whereas UCS will expand the node based on the edge cost while using a priority Queue. Here UCS will encounter the same cost for all the child nodes and will follow BFS search.

b. Yes, depth-first search is a special case of best-first search. If we want best first to mimic DFS we need to add a heuristic function that best first will work on, we can define the function as
   h (n) = - depth
   Here for level 1, we get the heuristic value as -1
   for level 2, we get a heuristic value as -2
   for level k, we get a heuristic value as –n
   We can see that the heuristics keep decreasing as we reach goal nodes.
   Best first search will have the same heuristic value for any node at the same level, and hence if we take two successive levels, 2 and 3. BFS will act like a depth-first search because at level 3 the cost would be -3 and -3 and at level 2 it will be -2. Hence it will expand the node at level 3 due to a lower heuristic value.

c. Yes, uniform cost search is a special case of the A* search algorithm. The very fundamental difference is that UCS evaluates the path based on g(n) the cost encountered till node n whereas A* evaluates the path based on the sum of g(n) and h(n) the heuristic value from node n to the goal state. If we make h(n) = 0. Then it's UCS.

Exercise 1.4

a. The size of the state space graph is 16.
   Let us consider the number of states for a single bank. We can have the following states:
   1. No one at the bank
   2. Only Cabbage
   3. Only Fox

4. Only Goat
5. Cabbage and Fox
6.Cabbage and Goat
7. Goat and Fox
8. All the three Fox, Goat and Cabbage.
Now that we consider the other bank as well, we can multiply the states by 2. The farmer will be present in either bank in these states.

Problem formulation:
Initial state: Farmer, cabbage, goat, and wolf on the same bank.
Successor function: S: item (may or may not be present), crossing left | crossing right → S'
Goal test: Farmer, cabbage, goat, and wolf have all crossed the river
Path cost: 1 per crossing the river

An admissible heuristic for this problem is given by this function
Where any of the 4 I.e. fox, farmer, goat, and cabbage if they are present on the left bank they are given the value 0 and if they are on the right bank they are given 1
If all of them are on the right bank then the heuristic value is zero indicating the goal state. The weight of value two is included because it will take the farmer two trips to take the animal and cross the river. So for example, if he crosses with the fox first, the value of fox goes to 1, so it will take him another two trips each to carry the cabbage and the chicken next.

The 1 – farmer term is to factor in if the farmer has already dropped any item on the right bank.

$$h(n) = 2\{ (1-\text{cabbage}) + (1-\text{fox}) + (1-\text{goat}) \} + (1\text{-farmer})$$

Exercise 1.5

Cost Iterative Deepening Search technique

Properties of Algorithm:

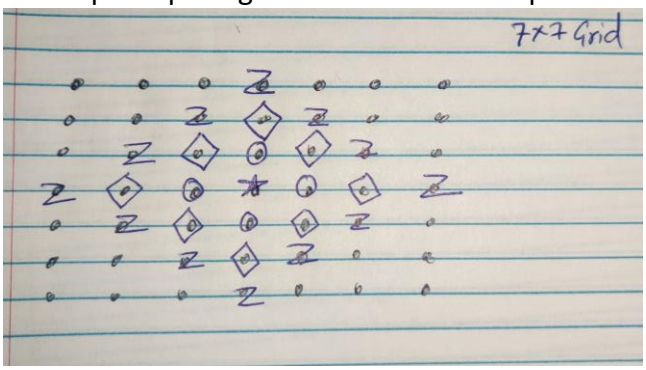
a. Optimality: The algorithm is optimal
b. Space Complexity: O(bc), where b is the branching factor and c is the cutoff cost
c. Time complexity:

```
function ITERATIVE-LENGTHENING-SEARCH(problem) returns a solution, or failure
        let cutoff_cost be 0
        source_node ← root_node of problem whose PATH-COST = 0
        frontier ←a priority queue ordered by PATH-COST, with node as the only element
        explored ←an empty set
        while cutoff_cost >= 0 do

            /* We are increasing the cost contour here and then running Depth limited search
till that cost contour*/
            if len( frontier ) == 0 then return failure
            output = RECURSIVE-DLS(source_node, problem, cutoff_cost, frontier)
            if output.solution? then returns node
            node←POP-MAX-COST-NODE( frontier )
            if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
            add node.STATE to explored

            for each action in problem.ACTIONS(node.STATE) do
                child ←CHILD-NODE(problem, node, action)
                if child.STATE is not in explored or frontier then
                frontier ←frontier.append(child, frontier)
                else if child.STATE is in frontier with higher PATH-COST then
                replace that frontier node with child
            increment ← MAX_COST_NODE( frontier )
            cutoff_cost += increment


function RECURSIVE-CLS(node, problem, limit, frontier) returns a solution, or failure/cutoff
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    else if limit <= 0 then return cutoff
    else
        for each action in problem.ACTIONS(node.STATE) do
            child ←CHILD-NODE(problem, node, action)
            cost_to_child = CHILD-NODE-COST(frontier)
            result ←RECURSIVE-DLS(child, problem, limit − cost_to_child, frontier)
            if result = cutoff then cutoff has occured
            else if result != failure then return result
     if cutoff occurred? then return cutoff else return failure
```

EXERCISE 1.2

a. The branching factor b here is 5, the actions we can take include going north, south, east, west, and staying put at the same node.

b. The total number of distinct states here per nth level is 1+2n(n+1)
At n=0, we get 1 state, at n=1 we get 5, at n=2 we get 13, and so on, here we have at every level n the cumulative distinct states, but for the total number, we see an Arithmetic progression and then reduce the formula to 1+4+8+12… by taking 4 common. In the following diagram assume the no op state to be represented along z axis, it will make the visualization a bit easier and will not contribute to distinct states. Like superimposing the same state on top it.



In the diagram star denotes root node, circle denotes distinct nodes at level 1, diamond denotes distinct nodes at level 2 and Z denotes distinct nodes at level 3

c. 5^d . Here we don't consider distinct nodes, breadth first search will loop through all nodes at every level d, even the no op nodes as well. Hence we get the total nodes BFS loops through to be 5^d . Its equivalent to a search tree with branching factor 5.

d. The maximum number of distinct nodes that BFS expands at level d would be 1+2(x+y)(x+y+1) here d=x+y. Only a new node that is explored would be added to distinct. How x+y = d? let us take the node on the right of the topmost Z node. Intutively the level of that node is 4, now taking the x, y coordinates that we need to traverse to reach that state will be x=3 and y=1.

e. Total number of nodes expanded by breadth first graph would be again same as d, since a state space graph will have only distinct states represented. No op will be the same node as its parent.

f. Yes, this is an admissible heuristic for the given problem. It is admissible since traversing this graph can only be along the x and y axis and since the heuristic is giving us that value exactly therefore, in this case, the heuristic is the exact value of the cost of the path taken. Since it is not greater than the lowest path cost value the path can take, it is admissible.

g. A* will expand the sum of x and y number of nodes, where x and y denote the number of steps along the two axes. If we generalize for the state (u, v) we can modify the number of nodes that A* takes to be the sum of |x-u| and |y-v|.

h. Yes, even if we remove some links the search pattern may take a longer path to find the goal state, hence the heuristic will still give us a lower value and will be admissible.

i. No, if we add paths between non-adjacent nodes there may be a case that a shorter path exists and hence the heuristic will show a greater value than the actual path. But if the heuristics were given by coordinate straight line distance then it would be admissible.

EXERCISE 1.3

a. The total number of states is n^2 P n where n cars can take up any position and not overlap in a space of n^2 points. This can be expanded to n^2 (n^2 -1) (n^2 -2) …. (n^2 –n+1) which is n^{2n} or the order of n^{2n}

b. The branching factor is 5^n .
We know that a car can go up, down, left, right, and stay in place. Let's say that it has a car in its vicinity of 1 step. It can still jump as long as there is no other car blocking it. In the best-case scenario, each car can jump to 5 unique places. Since there are n cars we get the branching factor to be 5*5*5…. N times. This is the best estimate when it has no cars blocking its move.

c. The heuristic for this problem can be given by $h(x_i, y_i) = |n-i+1 - x_i| + |n - y_i|$
This heuristic gives an exact estimate of cost since the cars can only move in 4 directions along x and y axis and since no other cars are present there is no jumping which would decrease the actual cost.

d. Let us limit n for calculating the actual path cost from the current state to the goal.
Here we have 5 cars that occupy the bottom 5 places of the 5*5 grid.
Now adding the number of steps it takes for the goal state we get the value to be 7.
How?
for car = 1 & 5 it takes 8 steps
for car = 2 & 4 it takes 6 steps
for car = 3 it takes 5 steps.

Now for the heuristic given in
we have summation over all heuristics and that gives us h (n) to be 32.
Here we see that the heuristic has a value greater than the actual path cost and is not admissible

For the maximum of all the heuristics, we get max (8, 6, 4, 6, 8) = 8.
This is not an admissible heuristic since it is greater than the actual path cost.

For the minimum of all the heuristics, we get min (8, 6, 4, 6, 8) = 4.

This is an admissible heuristic since it is lesser than the actual path cost.