Question 1 [2 marks]

Write a program using the PROLOG language to count how many 7 in the list.

For example: how\_many\_7([6,8,5,3],0).

How\_many\_7([5,7,9,7,7,3],3).

How\_many\_7([5,1,7,8,7],2).

The program:

%how to run the program

%how\_many\_7([6,8,5,3,7,7], Count).

%output Count = 2.

how\_many\_7([], 0). % base case: an empty list contains 0 7s

how\_many\_7([Head|Tail], Count) :- % recursive case

Head =:= 7, % if the head of the list is 7

how\_many\_7(Tail, Subcount), % count the number of 7s in the tail

Count is Subcount + 1. % add 1 to the count of 7s

how\_many\_7([Head|Tail], Count) :- % recursive case

Head =\= 7, % if the head of the list is not 7

how\_many\_7(Tail, Count). % count the number of 7s in the tail, without adding to the count

Include screenshots:

Graphical user interface, text, application

Description automatically generated

Question 2 [3 marks]

Modify the predicate canget in the program of Fig 2.14, that was described in the class, to return the steps that the monkey will do and return them in the list L).

The goal should be:

canget(state( monkey\_not\_middle, onfloor, box\_not\_middle,hasnot),state( Y, W, A,has),[],L).

write the new predicate canget and show the results (screen shot)

% Figure 2.14 A program for the monkey and banana problem.

% move( State1, Move, State2): making Move in State1 results in State2;

% a state is represented by a term:

% state( MonkeyHorizontal, MonkeyVertical, BoxPosition, HasBanana)

move( state( middle, onbox, middle, hasnot), % Before move

grasp, % Grasp banana

state( middle, onbox, middle, has) ). % After move

move( state( P, onfloor, P, H),

climb, % Climb box

state( P, onbox, P, H) ).

move( state( P1, onfloor, P1, H),

push( P1, P2), % Push box from P1 to P2

state( P2, onfloor, P2, H) ).

move( state( P1, onfloor, B, H),

walk( P1, P2), % Walk from P1 to P2

state( P2, onfloor, B, H) ).

The goal should be:

canget(state( monkey\_not\_middle, onfloor, box\_not\_middle,hasnot),state( Y, W, A,has),[],L).

write the new predicate canget and show the results (screen shot)

The code:

Text

Description automatically generated

The output:

Text

Description automatically generated

Question 3 [2 marks]

Find the code (all possible codes if there are any) and show how you got the solution and which rules were applied.



Answer:

As per condition 4: {4,3,8} nothing is correct, i.e, neither numbers and placements are incorrect.

so we should not use 4,3,8 in our key.

and Based on the given conditions, we can determine the code as follows:

From the first condition {7, 8, 2}, we know that one of the numbers is correct and well placed. Therefore, the code must contain either 7, 8 or 2 but we can eliminate 8 so we only have 7 or 2

From the second condition {7, 1, 6}, we know that one of the numbers is correct but in the wrong place. Since we already know that one of the correct numbers is in the first two positions, we can deduce that the code must contain either 1 or 6 in the third position.

From the third condition {2, 0, 7}, we know that two numbers are correct but in the wrong place. Since we already know that one of the correct numbers is in the first two positions, we can deduce that the code must contain 2 in the third position.

From the fourth condition {4, 8, 0}, we know that one of the numbers is correct but in the wrong place. However, we also know that the code cannot contain 4 or 8. Therefore, the only possibility is that the code contains 0 in the first position.

So we conclude:

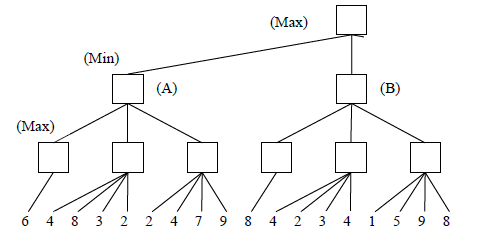
* The code cannot contain 3, 4 or 8.
* The code must contain 8 or 2 in the first two rules but we eliminate 8 on rule 4.
* The code must contain either 1 or 6 in the second rule.
* The code must contain 0 in the first position.

Therefore, we end up with a possible code which is, (0,6,2)

Question 4 [2 marks]

In the game tree below, it is Max's turn to move. At each leaf node is the estimated score of that resulting position as returned by the heuristic static evaluator.

* Perform Mini-Max search and label each branch node with its value.
* Cross out each leaf node that would be pruned by alpha-beta pruning



Text, letter

Description automatically generated

Question 5 [2 marks]

You have a 3-liter bowl, a 7-liter bowl and a 10-liter bowl, the last of which is filled with water. How can you measure exactly 5 liters of water, without spilling any of it outside?

Describe the states, the goal state, and the operators. Draw some of the states in the tree showing a path to the goal state.

Hint: the initial state can be described as:

10 7 3

|  |  |  |
| --- | --- | --- |
| 10 | 0 | 0 |

Answer: the initial state is

10 7 3

|  |  |  |
| --- | --- | --- |
| 10 | 0 | 0 |

The goal state is:

|  |  |  |
| --- | --- | --- |
| 5 | 5 | 0 |

We started with the initial state and kept moving from a state to another till we reached our goal state using the tree of search

Text, letter

Description automatically generated

Question 6 [4 marks]

The sliding-tile puzzle consists of 1 2 3 tiles and two empty spaces as in the configuration shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 |  | 3 |  |

The puzzle has two legal moves with associated costs:

* A tile may move into an adjacent empty location. This has a cost of 1.
* A tile can hop over one or two other tiles into an empty position. This has a cost equal to the number of tiles jumped over multiplied by 2 (do not count spaces when jumping).

The goal is to have the following state:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 3 |  |  | 2 | 1 |

1. Draw part of the search tree and one path to the goal with the total cost.

Diagram, schematic

Description automatically generated

b) Propose a heuristic for solving this problem and draw the search tree after using the heuristic.

Manhattan distance: sum of absolute distances of all numbered tiles form their goal positions

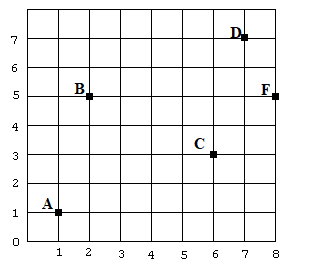
To propose a heuristic for solving the problem, we can use A∗ algorithm to find a solution path.

H = 1+2+3 = 6

Diagram, schematic

Description automatically generated

Question 7 [4 marks]



The following points A(1,1), B(2,5), C(6,3), D(7,7), F(8,5) are located in the 2 dimensional space as shown:

It is required to start at point (1,1) and visit all other points at least once. You can move only on the shown dark lines (horizontal and vertical lines), and it will cost one dollar to move one unit (for example: it costs 5 dollars to move from A to B).

* How many solutions? infinite
* Draw part of the search tree showing at least two possible routes and their total costs.

Text, letter

Description automatically generated

* Propose a heuristic to solve the problem with the minimum cost, show the solution path and the tree after using the heuristic.

One possible heuristic to solve this problem with minimum cost is the Manhattan distance heuristic. The Manhattan distance between two points is the sum of the absolute differences of their coordinates.

Manhattan distance = |x1 - x2| + |y1 - y2|

Text, letter

Description automatically generated

Question 8 [1 mark]

Fill the following table and find a *longest common subsequence* (LCS) of two DNA sequences.

S1 = CAGTCC

S2 = ATCGC

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | C | A | G | T | C | C |
|  | **0** | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | **0** | **1** | **1** | 1 | 1 | 1 |
| T | 0 | 0 | 1 | 1 | **2** | 2 | 2 |
| C | 0 | 1 | 1 | 1 | **2** | 3 | 3 |
| G | 0 | 1 | 1 | 2 | 2 | 3 | 3 |
| C | 0 | 1 | 1 | 2 | 2 | 4 | **4** |

Answer: From the traceback, you get ATCC as an LCS.

Remember:

When you fill in a cell, you consider:

* The cell directly to the left of it
* The cell directly above it
* The cell to the above-left of it

The three values below correspond, respectively, to the values returned by the three recursive subproblems I listed earlier.

* *V1* = the value in the cell to the left
* *V2* = the value in the cell above
* *V3* = the value in the cell to the above-left

You fill in the empty cell with the maximum of these three numbers:

*V1*

*V2*

*V3* + 1 if *C1* equals *C2*, or *V3* if *C1* is not equal to *C2*, where *C1* is the character above the current cell and *C2* is the character to the left of the current cell

Note that, add arrows that point back to which of those three cells that been used to get the value for the current cell.

In the case of equal numbers, where the new number could have come from more than one cell, pick an arbitrary one: **the one to the above-left, say have more priority than the one comes from left than the one comes from above.**