Contents

ALGORITHMS

TANMAY AGARWAL

1 NQueens Problem

In this problem we first create a driver code and then in that we define a board which is initialised to all zeros of the desired size. This board is basically our chess board and we implement our methods on this. We then call a utility function and pass this board in it as well as the zero column

n the utility function we define the base case, which is that if the col number is ==n-1 we will return True which just means that all the queens have been placed After this we iterate for all the rows : if issafe the given board with row and col, then : we set $board[row][col] = 1$ and then recursively call the utility function again providing the board and the $col + 1$. If that is true we return true, else we immediately backtrack by setting $board = 0$ and return false. The reason we backtrack is the fact that then either of the recursive utility functions returns false, it means that if we place a queen in that row and col we can be attacked by other queen

Now the issafe function checks for three aspects : right(row=same, col iterated), lower diagonal(using zip), upper diagonal(using zip). Remember we check the left upper diagonals and not the right , because we check whether a previously placed queen can cause troubles and that is done using the left diagonals and the left rows

NOTE : In the gold table problem we check from the right in the gold table and in this we use the left diagonals and that both the problems follow a different approach

2 MazeSolve

In this Problem again we create a driver function in which we initialise a solution arrest with all 0's and also our driver function takes a maze as input. Now this driver code initialises another utility function which takes in maze,0,0 and sol. 0,0 are the starting row and column

Now the base case of the utility function is that if the co-ordinates are equal to the last row and last column, then we set that row's and col's solution to be 1 and return True Now we say if $issafe(maze, x, y)$: we check the base case for this loop which is that whether or not the solution array is already 1 for this particular coordinate. If yes, return False cause this is not the solution and it failed some previous loops. When the base is false, we initialise the $sol[co-ordintes] = 1$ and recursively call the utility function for $x+1, x-1, y+1, y-1$ which return True if the cases are true If any of it is false we backtrack setting $sol[co-ordintes] = 0$ and return false

The issue function checks that the co-ordinates are within the boundary ($0 < x < n$) and that the original maze at that co-ordinates contains a 1. If all of it is satisfied it returns True or else False

2.1 Difference between NQueens and MazeSolver

The first and major difference is the issafe functions. The next is that the base case of the utility functions is different and that in Nqueens, the utility function has a loop, which loops over all rows whereas the mazesolver has no loops as it is not needed Now in the issafe in utility, nqueens doesn't have a base case whereas mazesolver does

3 Gold Mine Problem

This problem is very simple and does not have any backtracking in it. In this problem we create only one function which takes in the original gold table and the MxN dimensions.

it initialises a goldtable with zeros. Now it makes 3 moves: right, rightUp and rightDown on the goldtable(not the of gold table). It then adds the max of these 3 moves with the value in the og gold table and stores it in the goldtable. This process is iterated over all rows and cols but col starts from last and ends at first

The res will be in the first col

4 Linked Lists

4.1 Remove Duplicates Algo

In this first before starting any loops we initialise a temporary head named head to be equal to (self.head). We do this so that we need to want to disturb the original head pointer as we need the og head pointer to stay in its place.

Now we start a loop saying if head.next != None : we initialise a pointer named pre on head and another pointer, temp on the next of head. Now we start another loop saying while temp.next != None : We check if the temp data == head data. If yes: we say that pre.next=temp.next. Then we increment temp and set pre=temp.

If not we just say that pre=temp and temp=temp.next and finally in the ending of this loop we also increment our head pointer

4.2 Odd Even Algo

In this first we need to find the length of the list because our final action depends upon whether the list is even in length or odd in length. Now we store the length of the list in a variable n. then we initialise a temporary head to be self.head(head=self.head) and we assign a temp variable to be head.next.

Then we immediately assign a prev variable to the temp(prev=temp). This prev pointer will point to the second node(which is the fist even node) of the list and hence we need to come back to it for connecting the remaining nodes after completing the odd connections.(Note : the remaining connections will be all even connections now). Now we start a loop saying while temp.next != None: we connect head.next to temp.next and then increment head by doing a head=temp and then increment temp(temp=temp.next). After this loop our temp will point to the last node and head to the last but one node

if the list is even : head will be pointing to last odd node in the end and temp to last even. Hence to complete the list we need to connect the final odd node to our starting even node which is the prev pointer. Hence we say if list is even : head.next=pre and temp.next=None(as temp will be pointing to the last even node)

else if the list is odd, temp is the last odd node. hence temp.next= pre and head.next = None

5 Quick Foot Notes

In the special reverse function problem we assign an initialise j outside of all the loops because for swapping we do not need the index to start from the first position always.

But for problems like finding duplicates and triplets and rest, we initialise the j index inside the i loop because we need the j to start from 0 every time the i loop is incremented

Also we use while loops at many places because we would like to check a condition and then increment. For loops only check the condition for termination. While loops are used to increment after checking a few conditions. Inside a while loop, many if and else blocks can be placed to check multiple conditions and increment accordingly. In short use for loops when there is no condition required to check before incrementing(generally avoid for loops for array based problems)