# MAXIMUM GOLD PROBLEM C2_ASSIGNMENT_5

GROUP MEMBERS :

PRITIK SRIVASTAVA (IIT2019192)

CHETAN PATIDAR (IIT2019193)

RAHUL ROY (IIT2019194)

## Abstract

Given a gold mine of n∗ m dimensions. Each field in this mine contains a positive integer which is the amount of gold in tons. Initially the miner is at first column but can be at any row. He can move only (right, right up, right down) that is from a given cell, the miner can move to the cell diagonally up towards the right or right or diagonally down towards the right. Find out maximum amount of gold he can collect.

**The project aims at :Solving the gold mine problem and practically implement the approach by maximum time and space utilisation.**

## Contents

## INTRODUCTION

The miner starts at the first column, (any row). And after every step he moves one step right into the next column (either towards right or diagonally right), until he reaches the final column. Our job is to make the miner move through the path which gives him the maximum gold and return the maximum amount of gold collected.

We will solve the given problem using dynamic programming approach.

This presentation further contains -

## ALGORITHMIC DESIGN

The Algorithm we designed is based on top down approach of dynamic programming to find the most suitable path that provides maximum sum of elements of matrix.

The recursive approach calculates the maximum amount of gold that can be obtained from every path in the matrix and stores it in a dp table to reduce time in calculating overlapping sub problems.

1. The recursive function gets input in the form of mine matrix and its dimensions(n,m), dp matrix, current row(i) and current column(j).

2.The function calculates the maximum amount of gold that can be obtained if miner starts from the current position(i,j) and stores it in dp table.

3. In every move we find the maximum of right, right_up, and right_down and then add it with that mine[i][j] and also store and return the result in dp[i][j].

## ALGORITHMIC DESIGN

For example:
Consider input mine matrix as follows:-

```
 mine= { { 10, 20, 1} ,
         { 30, 10, 100} ,
         { 10, 10, 10} } .
```

We run the recursive function at all the cells in first column and compute our dp table.

The cells in last column of mine and dp table are equal. In the second column the dp table is computed as sum of current cell and max of 1 dp[i+1][j+1],dp[i][[j+1] & dp[i-1][j+1] (if exists) according to the recursive function.
The resultant DP table will be:

```
dp = { { 130, 120, 1} ,
       { 150, 110, 100} ,
       { 130, 110, 10} }
```

In the above table, the maximum value in the found in the first column is our required answer ie. 150

# Program

- **PSEUDO CODE:**
- Algorithm findMaxGold
- **Passed** vector<vector<int>mine

```
function max_gold(mine[][],n,m,i,j,dp[][]) {
if(i<0| | i>n-1| | j<0| | j>m-1)
                                        return 0;
end if
        if(dp[i][j]!=-1)
                          return dp[i][j]
 end if
         temp1← max_gold(mine,n,m,i-1,j+1,dp)
         temp2← max_gold(mine,n,m,i,j+1,dp)
         temp3← max_gold(mine,n,m,i+1,j+1,dp)
```

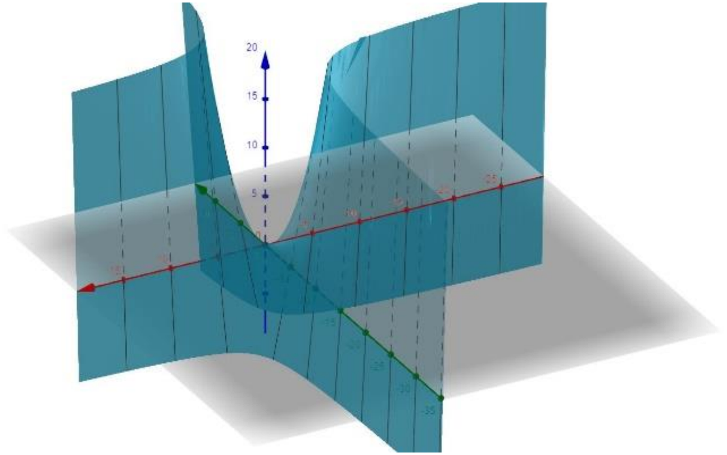
```
dp[i][j] ← max(temp1,temp2,temp3)

 return dp[i][j]
}

function findmaxgold(){

          n← mine.size()
          m← mine[0].size()

     for(i=0;i<n;i++)
            for(j=0;j<m;j++)
                         dp[i][j] ← -1
ans← 0

          for(i=0;i<n;i++)
                  ans← max(ans, max_gold(mine,n,m,i,0,dp))

 return ans
```

# ALGORITHM ANALYSIS

**A. Time complexity**

The Algorithm uses recursion with memorization to produce a dynamic programming solution. The algorithm on worst case traverses through the whole matrix of n∗ m dimension, hence the worst case time complexity of this algorithm is
**O(n∗ m).**

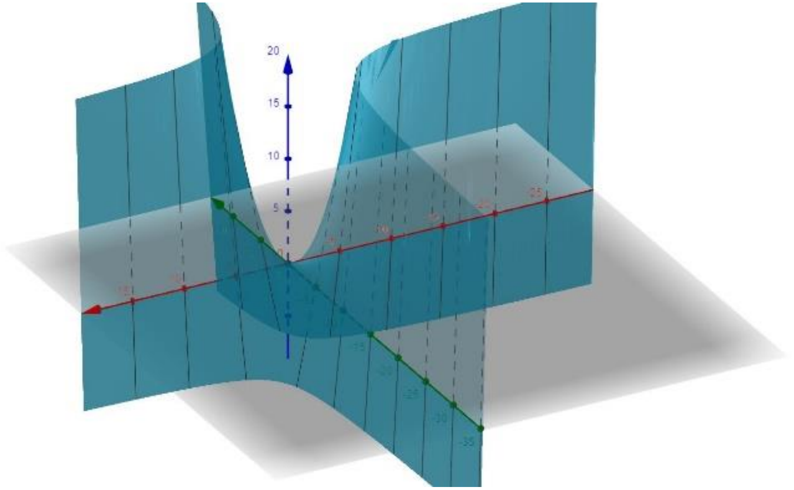
From the graph plotted between the
**n(no. of rows :x-axis),**
**m(no. of columns :y-axis)** and **time(zaxis)**
we can conclude the **O(n∗ m)** behavior
of the algorithm.



## ALGORITHM ANALYSIS

- B. Space complexity :
- The space complexity of the program is **O(n∗ m)** as we store the intermediate results in a dp table which has dimensions of n∗ m.

- Graph Represents the space complexity O(n∗ m).



## CONCLUSION

The experimental study showed that the time complexity of the algorithm using dynamic programming is $O(n * m)$ which depends on the size of the input matrix which is better than the naive approach to use backtracking to produce all the paths that a miner can take from the first column to the last column and for each path calculate the number of gold coins collected on each path which would result in recalculation of recurring subproblems.

## Reference

[1] https://www.geeksforgeeks.org

[2] Introduction to Algorithms by Thomas H.cormen