<u>DAA MINI PROJECT</u>

PROJECT REPORT

TOPIC:

GOLD MINE PROBLEM USING DYNAMIC APPROACH

Team Members:

Y. SREE SANTHOSH – RA2011030010214

S. PRANAV – RA2011030010212

K.ABHIRAM – RA2011030010218

TEAM MEMBERS CONTRIBUTION:

Member	Registration Number	Contribution
Y. SREE SANTHOSH	**RA2011030010214**	**Provided Problem Definition, problem explanation**
S. PRANAV	**RA2011030010212**	**Algorithm, coding part**
K. ABHIRAM	**RA2011030010218**	**Design technique used, Time complexity**

PROBLEM DEFINITION

You have been given a gold mine represented by a 2-d matrix of size ('N' * 'M') 'N' rows and 'M' columns. Each field/cell in this mine contains a positive integer, the amount of gold in kgs.

Initially, the miner is at the first column but can be at any row.

He can move only right, right up, or right down. That is from a given cell and the miner can move to the cell diagonally up towards the right or right or diagonally down towards the right.

Find out the maximum amount of gold he can collect.

PROBLEM EXPLANATION

1. You are given a number n, representing the number of rows.

2. You are given a number m, representing the number of columns.

3. You are given n*m numbers, representing elements of 2d array a, which represents a gold mine.

4. You are standing in front of left wall and are supposed to dig to the right wall. You can start from

   any row in the left wall.

5. You are allowed to move 1 cell right-up (d1), 1 cell right (d2) or 1 cell right-down(d3).

6. Each cell has a value that is the amount of gold available in the cell.

7. You are required to identify the maximum amount of gold that can be dug out from the mine.

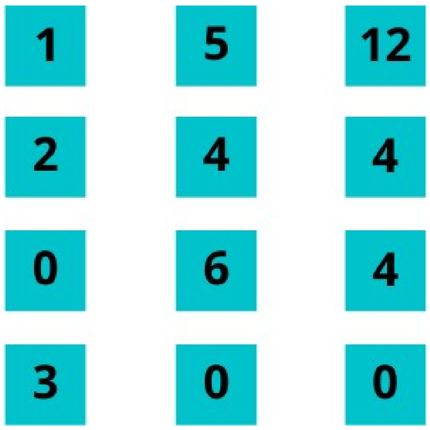
Dynamic Programming:

We are using dynamic programming approach for solving this problem.

This approach is an algorithm design method used when the solution to a problem can be viewed as a result of sequence of subproblems/decisions. It solves problems by combining solutions to subproblems. It is usually applied when subproblems overlap, i.e. when subproblems share subsubproblems.

It is different from the divide and conquer approach as divide and conquer does more work than necessary by repeatedly solving the common subsubproblems whereas in the case of dynamic programming, the subsubproblems are computed only once and their solutions are stored in a table, avoiding the re-computation of the subsubproblems every time.

So, when you think about the problem after looking at the above two approaches, it seems simple enough, right? We need the maximum gold that can be acquired, so applying the greedy approach should do the trick.

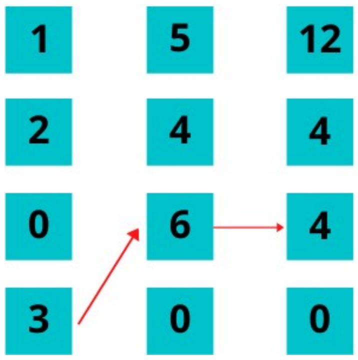


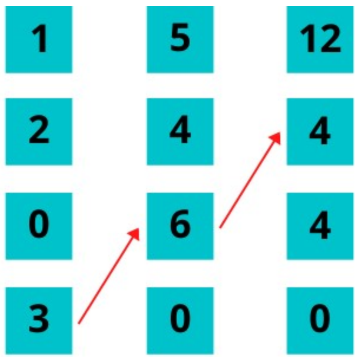
Take a look at this example. According to the rules of the gold mining problem, the miner can be at any row but should be at the first column.

So, now that we are testing the greedy approach, we should select the maximum element in the 1st column, which is 3 located at (2,0).

After we take 3, we can either take 6(diagonally above to the right) or 0(to the right). As 6>0, we take 6.

Now, we can either go to the 4(diagonally above to the right), the other 4(to the right), or the 0(diagonally right). As 4>0, we can take either of the 4's(both are correct).



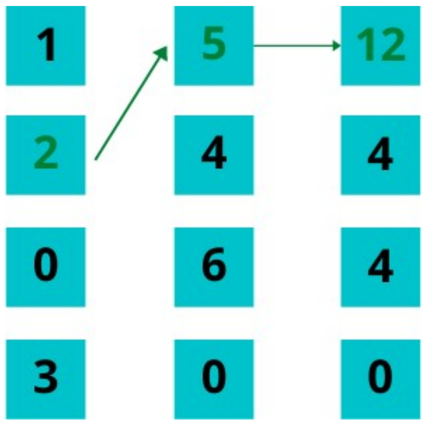


The final solution achieved is: 3+6+4=13

We then check all other possible solutions to confirm our answer using the Brute Force Approach.

The Brute Force Approach or exhaustive search approach, is a very general problem-solving technique that consists of systematically enumerating all possible solutions and checking whether each candidate satisfies the problem's statement.

So, when we check out all the possible solutions, we see that the maximum solution that can be arrived is 2+5+12 = 19



Dynamic Approach solution

To solve this problem, we have to keep computing the value of maximum gold at each cell and update them in a table every time.

If we are given a matrix of dimension x*y , Gold_Mine [x][y],we begin by creating a matrix of same dimension Max_Gold[x][y]

Initialization :

We initialize each cell except cells in the 1st column to 0. The cells in the 1st column already have the maximum gold that can be taken. So, it would be:

1	0	0
2	0	0
0	0	0
3	0	0

Step 2

After initialization, for the remaining cells, we take each cell (column-wise) and compute the maximum gold that the miner can achieve by standing at that particular cell.

Same way, after calculating values for each cell in 2nd column, the updated table would look like:

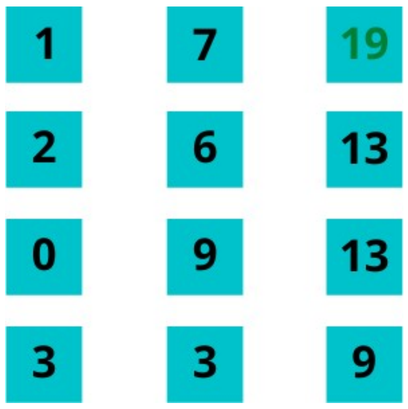
1	7	0
2	6	0
0	9	0
3	3	0

For the 3rd column:

After computing values for 3rd column after applying the formula, the final table would look like:

1	7	19
2	6	13
0	9	13
3	3	9

Now, the last step is to search through the entire last column and pick the maximum value in it. That value will give us the maximum amount of gold that can be mined by the miner.



1	7	19
2	6	13
0	9	13
3	3	9

In this case, as 19 is the maximum cost in the last column, it is the maximum amount of gold that can be mined.

DESIGN TECHNIQUE USED

Dynamic Programming:

We are using dynamic programming approach for solving this problem.

This approach is an algorithm design method used when the solution to a problem can be viewed as a result of sequence of sub problems/decisions. It solves problems by combining solutions to sub problems. It is usually applied when sub problems overlap, i.e. when sub problems share subsubproblems.

It is different from the divide and conquer approach as divide and conquer does more work than necessary by repeatedly solving the common subsubproblems whereas in the case of dynamic programming, the subsubproblems are computed only once and their solutions are stored in a table, avoiding the re-computation of the subsubproblems every time.

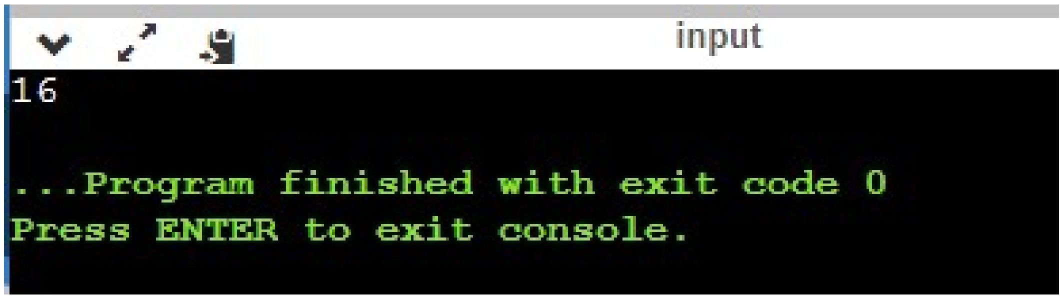
ALGORITHM

```
if ((x < 0) || (x == n) || (y == m)) {
      return 0;
    }

    // Right upper diagonal
    int rightUpperDiagonal
      = collectGold(gold, x - 1, y + 1, n, m);

    // right
    int right = collectGold(gold, x, y + 1, n, m);

    // Lower right diagonal
    int rightLowerDiagonal
      = collectGold(gold, x + 1, y + 1, n, m);

    // Return the maximum and store the value
    return gold[x][y]
      + Math.max(Math.max(rightUpperDiagonal,
                          rightLowerDiagonal),
                 right);
```

CODE

```
#include<bits/stdc++.h>
using namespace std;

int collectGold(vector<vector<int>> gold, int x, int y,
int n, int m) {

    // Base condition.
    if ((x < 0) || (x == n) || (y == m)) {
        return 0;
    }



    // Right upper diagonal
    int rightUpperDiagonal = collectGold(gold, x - 1, y
+ 1, n, m);

     // right
    int right = collectGold(gold, x, y + 1, n, m);

    // Lower right diagonal
    int rightLowerDiagonal = collectGold(gold, x + 1, y
+ 1, n, m);

    // Return the maximum and store the value
    return  gold[x][y] + max(max(rightUpperDiagonal,
rightLowerDiagonal), right);
}

int getMaxGold(vector<vector<int>> gold, int n, int m)
{
    int maxGold = 0;

    for (int i = 0; i < n; i++) {

        // Recursive function call for  ith row.
        int goldCollected = collectGold(gold, i, 0, n,
m);
        maxGold = max(maxGold, goldCollected);
    }
    return maxGold;
}

// Driver Code
int main()
{
    vector<vector<int>> gold { {1, 3, 1, 5},
        {2, 2, 4, 1},
        {5, 0, 2, 3},
        {0, 6, 1, 2}
    };
    int m = 4, n = 4;
    cout << getMaxGold(gold, n, m);
    return 0;
}
```

OUTPUT:



```
16

...Program finished with exit code 0
Press ENTER to exit console.
```

COMPLEXITY ANALYSIS

Time complexity: O(3N*M)

Auxiliary Space: O(N*M)

CONCLUSION

By using Dynamic Programming we were able to mine maximum gold from gold mine with conditions of moving in right , right upwards and right downwards directions.

Thus we have successfully solved the gold mine program using dynamic programming approach.

REFERENCES

https://www.geeksforgeeks.org/gold-mine-problem/

https://www.codingninjas.com/codestudio/problem-details/gold-mine-problem_799363