Algorithms Exercise #3

a. Environment

I. OS: Windows 11

II. Compiler: GNU GCC Compiler

III. IDE: Code::Blocks 20.03

b. Results

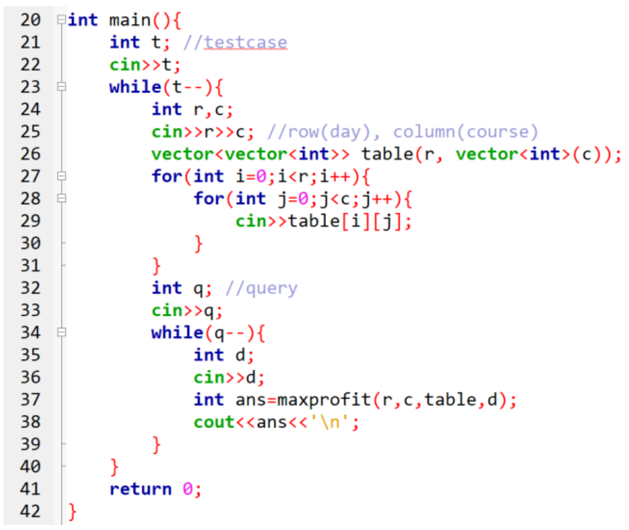
I. Method:

i. Main function:

(1) The main function handles input and output.

(2) Using vector to store the elements of profit table.

(3) Using while loop to process each testcase and query.



```
int main(){
    int t; //testcase
    cin>>t;
    while(t--){
        int r,c;
        cin>>r>>c; //row(day), column(course)
        vector<vector<int>> table(r, vector<int>(c));
        for(int i=0;i<r;i++){
            for(int j=0;j<c;j++){
                cin>>table[i][j];
            }
        }
        int q; //query
        cin>>q;
        while(q--){
            int d;
            cin>>d;
            int ans=maxprofit(r,c,table,d);
            cout<<ans<<'\n';
        }
    }
    return 0;
}
```

ii. Dynamic programming:

(1) The *'maxprofit'* function employs a bottom-up dynamic programming method to solve resource allocation problem.

(2) It uses 2D vector *'dp'* to store intermediate results, the value of dp[i][j] represents the maximum profit of assigning days to i courses.

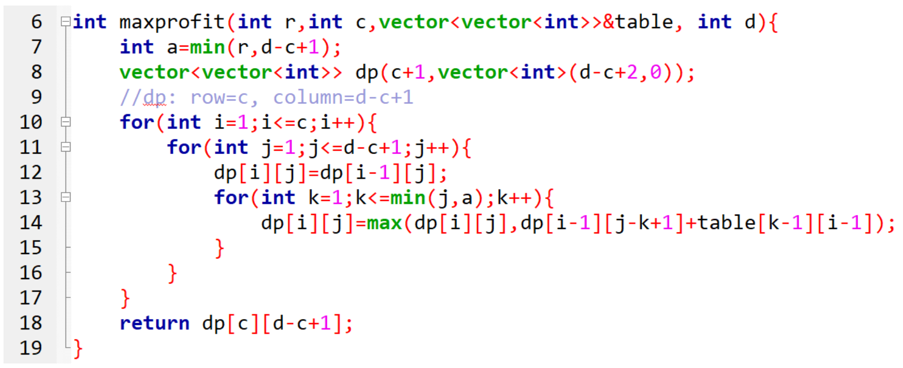
(3) Because each course should study at least one day, the size of dp column is d-c+1.

(4) Recursively define the value of an optimal solution by iterative approach. Initialize with the value of the previous row *dp[i][j]=dp[i-1][j]*

(5) Consider all possible k and update the value using the optimal substructure: *dp[i][j] = max(dp[i][j], dp[i-1][j-k+1] + table[k-1][i-1])*.

(6) Termination condition: The result is stored in *dp[c][d-c+1]*

(7) The reason of using *‘int a=min(r, d-c+1);’* and *‘for(int k=1;k<=min(j,a);k++)’* is to deal with the condition that if r<d-c+1 or d>r*c, ensuring the proper allocation of the dynamic programming table.



```cpp
int maxprofit(int r,int c,vector<vector<int>>&table, int d){
    int a=min(r,d-c+1);
    vector<vector<int>> dp(c+1,vector<int>(d-c+2,0));
    //dp: row=c, column=d-c+1
    for(int i=1;i<=c;i++){
        for(int j=1;j<=d-c+1;j++){
            dp[i][j]=dp[i-1][j];
            for(int k=1;k<=min(j,a);k++){
                dp[i][j]=max(dp[i][j],dp[i-1][j-k+1]+table[k-1][i-1]);
            }
        }
    }
    return dp[c][d-c+1];
}
```