1937. Maximum Number of Points with Cost

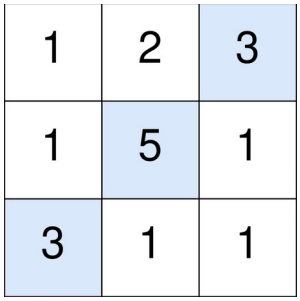
you are given an `m x n` integer matrix `points` (**0-indexed**). Starting with `0` points, you want to **maximize** the number of points you can get from the matrix.

To gain points, you must pick one cell in **each row**. Picking the cell at coordinates `(r, c)` will **add** `points[r][c]` to your score.
However, you will lose points if you pick a cell too far from the cell that you picked in the previous row. For every two adjacent rows `r` and `r + 1` (where `0 <= r < m - 1`), picking cells at coordinates `(r, c1)` and `(r + 1, c2)` will **subtract** `abs(c1 - c2)` from your score.
Return *the **maximum** number of points you can achieve*.
`abs(x)` is defined as:

- `x` for `x >= 0`.
- `-x` for `x < 0`.

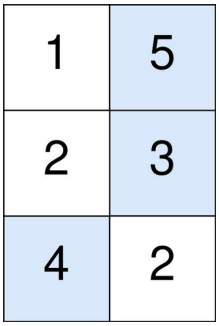
Example 1:

```
Input: points = [[1,2,3],[1,5,1],[3,1,1]]
Output: 9
Explanation:
The blue cells denote the optimal cells
to pick, which have coordinates (0, 2),
(1, 1), and (2, 0).
You add 3 + 5 + 3 = 11 to your score.
However, you must subtract abs(2 - 1) +
abs(1 - 0) = 2 from your score.
Your final score is 11 - 2 = 9.
```



Example 2:

```
Input: points = [[1,5],[2,3],[4,2]]
Output: 11
Explanation:
The blue cells denote the optimal cells
to pick, which have coordinates (0, 1),
(1, 1), and (2, 0).
You add 5 + 3 + 4 = 12 to your score.
However, you must subtract abs(1 - 1) +
abs(1 - 0) = 1 from your score.
Your final score is 12 - 1 = 11.
```



Constraints:

- `m == points.length`
- `n == points[r].length`
- `1 <= m, n <= 10^5`
- `1 <= m * n <= 10^5`
- `0 <= points[r][c] <= 10^5`

1937. Maximum Number of Points with Cost

```
/*
   Dynamic programming: Memoization
   Time complexity: o(n.mn)=O(mn^2) (TLE)
   Space complexity: O(2mn)=O(mn)
*/
typedef vector<long long> vi;
typedef vector<vi> vvi;

class Solution {
public:
   long long maxPoints(std::vector<std::vector<int>>& points) {
      int m=points.size();
      int n=points[0].size();

      vvi memo(m,vi(n,-1));
      auto solve=[&](int i, int j, auto& self)->long long{
         if(i>=m) return 0;
         if(memo[i][j]!=-1) return memo[i][j];
         long long ans=0;
         for(int k=0;k<n;++k) {
            long long val=points[i][k]-abs(k-j)+self(i+1,k,self);
            ans=std::max(ans,val);

         }
         return memo[i][j]=ans;
      };

      long long ans=0;
      for(int i=0;i<n;++i) {
         long long val=(long long)points[0][i]+solve(1,i,solve);
         ans=std::max(ans,val);
      }

      return ans;
   }
};
```

1937. Maximum Number of Points with Cost

```
/*
   Dynamic programming: Tabulation
   Time complexity: O(n+m(n+n^2)+n)=O(2n+mn+mn^2)=O(mn^2) (TLE)
   Space complexity: O(n)
*/
typedef vector<long long> vi;
typedef vector<vi> vvi;
class Solution {
public:
   long long maxPoints(std::vector<std::vector<int>>& points) {
      int m=points.size();
      int n=points[0].size();

      vi dp(points[0].begin(),points[0].end());

      for(int i=1;i<m;++i){
         vi tmp=dp;
         for(int j=0;j<n;++j){
            long long mx=-1;
            for(int k=0;k<n;++k){
               mx=std::max(mx,tmp[k]+points[i][j]-(abs(k-j)));
            }
            dp[j]=mx;
         }
      }

      long long ans=*std::max_element(dp.begin(),dp.end());

      return ans;
   }
};
```

1937. Maximum Number of Points with Cost

```
/*
   Dynamic programming: Tabulation+prefix,suffix max
   Time complexity: O(n+m.6n+n)=O(2n+6mn)=O(m.n) (AC)
   Space complexity: O(3n)=O(n)
*/
typedef vector<long long> vi;
typedef vector<vi> vvi;

class Solution {
public:
   long long maxPoints(std::vector<std::vector<int>>& points) {
      int m=points.size();
      int n=points[0].size();

      vi dp(points[0].begin(),points[0].end());

      for(int i=1;i<m;++i){
         vi prefix_max(n,0),suffix_max(n,0);
         prefix_max[0]=dp[0];
         for(int j=1;j<n;++j){
            prefix_max[j]=std::max(dp[j],prefix_max[j-1]-1);
         }

         suffix_max[n-1]=dp[n-1];
         for(int j=n-2;j>=0;--j){
            suffix_max[j]=std::max(dp[j],suffix_max[j+1]-1);
         }

         // dp=points[i]
         std::transform(points[i].begin(),points[i].end(),dp.begin(),[](int x) {return (long long)x;});

         for(int j=0;j<n;++j){
            dp[j]+=std::max(prefix_max[j],suffix_max[j]);
         }
      }

      long long ans=*std::max_element(dp.begin(),dp.end());

      return ans;
   }
};
```