Triangle

Description

Given a triangle array, return the minimum path sum from top to bottom.

For each step, you may move to an adjacent number of the row below. More formally, if you are on index i on the current row, you may move to either index i or index i +1on the next row.

给定一个三角形数组，返回从上到下的最小路径和。

对于每一步，您可以移动到下面一行的相邻数字。更正式地说，如果你在当前行的索引i上，你可以移动到下一行的索引i或索引i +1上。

Input

Line1:

The height of the triangle, and 1 &lt;= triangle.height &lt;= 2001<=*triangle*.*height*<=200.

Line2:

All the elements in the triangle, and split by some spaces(for each element,-10^4<= triangle[i][j] <=10^4). We are sure that the number of the elements satisfy:

Output

Print the minimum path sum from top to bottom.

输入

Line1:

三角形的高度，和1 <=三角形。高度& lt; = 2001 < = triangle.height < = 200。

么:

三角形中的所有元素，并用一些空格分割(对于每个元素，-10^4<= triangle[i][j] <=10^4)。我们确信元素的数量满足:

输出

从上到下打印最小路径和。

Sample Input 1

4

2 3 4 6 5 7 4 1 8 3

Sample Output 1

11

Sample Input 2

1

-10

Sample Output 2

-10

Hint

Input:

4

2 3 4 6 5 7 4 1 8 3

Output:

11

Explanation: The triangle looks like:

**2**

**3**4

6**5**7

4**1**83

The minimum path sum from top to bottom is 2 + 3 + 5 + 1 = 11 (bolded above).

**leetcode】120. 三角形最小路径和【数组】【动态规划】**

class Solution {

public:

int minimumTotal(vector<vector<int>>& triangle) {

int n=triangle.size();

vector<int> minx(n+1);

for(int i=n-1;i>=0;i--)

{

for(int j=0;j<=i;j++)

{

minx[j]=min(minx[j],minx[j+1])+triangle[i][j];

}

}

return minx[0];

}

};

作者：wei-mo-5k

链接：https://leetcode-cn.com/problems/triangle/solution/dpkong-jian-you-hua-by-wei-mo-5k-bq94/

来源：力扣（LeetCode）

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

#include<iostream>

#include<algorithm>

using namespace std;

int main(){

    int before[100][100];

    int middle[100][100];

    int N;

    cin>>N;//三角形行数

    for(int i=0;i<N;i++)

    {

        for(int j=0;j<=i;j++)

        {

            cin>>before[i][j];

        }

    }

    for(int i=N-1;i>=0;i--)

    {

        for(int j=0;j<=i;j++)

        {

            if(i==N-1)

            {

                middle[i][j]=before[i][j];

            }

            else

            {

                middle[i][j]=min(middle[i+1][j]+before[i][j],middle[i+1][j+1]+before[i][j]);

            }

        }

    }

    cout<<middle[0][0]<<endl;

    return 0;

}

#include <iostream>

#include <string.h>

#define MAXN 105

using namespace std;

int MaxSum[MAXN],d[MAXN][MAXN];

int dpMaxSum(int i, int j, int n){

    for(int i = 1; i <= n; i++){

        MaxSum[i] = d[n][i];

    }

    for(int i = n - 1; i>=1; i--){

        for(int j = 1; j <= i; j++){

            MaxSum[j] = max(MaxSum[j],MaxSum[j + 1]) + d[i][j];

        }

    }

    return MaxSum[1];

}

int main(){

    int n;

    cin>>n;

    for(int i = 1; i <= n; i++){

        for(int j = 1; j <= i; j++){

            cin>>d[i][j];

        }

    }

    int ans = dpMaxSum(1,1,n);

    cout<<ans<<endl;

    return 0;

}

#include <iostream>

#include <string.h>

#define MAXN 105

using namespace std;

int MaxSum[MAXN],d[MAXN][MAXN];

int dpMaxSum(int i, int j, int n){

    for(int i = 1; i <= n; i++){

        MaxSum[i] = d[n][i];

    }

    for(int i = n - 1; i>=1; i--){

        for(int j = 1; j <= i; j++){

            MaxSum[j] = max(MaxSum[j],MaxSum[j + 1]) + d[i][j];

        }

    }

    return MaxSum[1];

}

int main(){

    int n;

    cin>>n;

    //总元素个数num

    //int num=0;

    //for(int m=n;m>=1;m--)

    //num=num+m;

    //int dd[MAXN];

    //for(int i=1;i<=num;i++)

    //cin>>dd[];

    for(int i = 1; i <= n; i++){

        for(int j = 1; j <= i; j++){

            cin>>d[i][j];

        }

    }

    int ans = dpMaxSum(1,1,n);

    cout<<ans<<endl;

    return 0;

}

#include <iostream>

#include <string.h>

#define MAXN 105

using namespace std;

int MaxSum[MAXN],d[MAXN][MAXN];

int dpMaxSum(int i, int j, int n){

    for(int i = 1; i <= n; i++){

        MaxSum[i] = d[n][i];

    }

    for(int i = n - 1; i>=1; i--){

        for(int j = 1; j <= i; j++){

            MaxSum[j] = max(MaxSum[j],MaxSum[j + 1]) + d[i][j];

        }

    }

    return MaxSum[1];

}

//1+2+。。。。+i+j//(1+i)\*i/2+j

int main(){

    int n;

    cin>>n;

    //总元素个数num

    //int num=0;

    //for(int m=n;m>=1;m--)

    //num=num+m;

    //int dd[MAXN];

    //for(int i=1;i<=num;i++)

    //cin>>dd[];

    for(int i = 1; i <= n; i++){

        for(int j = 1; j <= i; j++){

            cin>>d[i][j];

        }

    }

    int ans = dpMaxSum(1,1,n);

    cout<<ans<<endl;

    return 0;

}

#include <iostream>

#include <string.h>

#define MAXN 105

using namespace std;

int MaxSum[MAXN],d[MAXN][MAXN];

int dpMaxSum(int i, int j, int n){

    for(int i = 1; i <= n; i++){

        MaxSum[i] = d[n][i];

    }

    for(int i = n - 1; i>=1; i--){

        for(int j = 1; j <= i; j++){

            MaxSum[j] = max(MaxSum[j],MaxSum[j + 1]) + d[i][j];

        }

    }

    return MaxSum[1];

}

// int dpMaxSum(int i, int j, int n){

//     for(int i = 1; i <= n; i++){

//         MaxSum[i] = dd[(1+n)\*n/2+i];//n][i

//     }

//     for(int i = n - 1; i>=1; i--){

//             MaxSum[i] = max(MaxSum[j],MaxSum[j + 1]) + d[i][j];

//     }

//     return MaxSum[1];

// }

//1+2+。。。。+i+j//(1+i)\*i/2+j

int main(){

    int n;

    cin>>n;

    //总元素个数num

    //int num=n\*(n+1)/2;

    //int dd[MAXN];

    //for(int i=1;i<=num;i++)

    //cin>>dd[];

    for(int i = 1; i <= n; i++){

        for(int j = 1; j <= i; j++){

            cin>>d[i][j];

        }

    }

    int ans = dpMaxSum(1,1,n);

    cout<<ans<<endl;

    return 0;

}

最终：

#include<iostream>

#include<algorithm>

using namespace std;

int main(){

int before[200][200];

int middle[200][200];

int N;

cin>>N;//三角形行数

for(int i=0;i<N;i++)

{

for(int j=0;j<=i;j++)

{

cin>>before[i][j];

}

}

for(int i=N-1;i>=0;i--)

{

for(int j=0;j<=i;j++)

{

if(i==N-1)

{

middle[i][j]=before[i][j];

}

else

{

middle[i][j]=min(middle[i+1][j]+before[i][j],middle[i+1][j+1]+before[i][j]);

}

}

}

cout<<middle[0][0]<<endl;

return 0;

}

错误的原因，数组开小了，应该是200

Maximum Alternating Subsequence Sum

Description

The alternating sum of a 0-indexed array is defined as the sum of the elements at even indices minus the sum of the elements at odd indices.

For example, the alternating sum of [4,2,5,3] is (4 + 5) - (2 + 3) = 4.

Given an array nums, return the maximum alternating sum of any subsequence of nums (after reindexing the elements of the subsequence).

A subsequence of an array is a new array generated from the original array by deleting some elements (possibly none) without changing the remaining elements' relative order.For example, [2,7,4] is a subsequence of [4,2,3,7,2,1,4] (the underlined elements), while [2,4,2] is not.

Input

Anarray.

1 &lt;= nums.length &lt;= 10^51<=*nums*.*length*<=105

1 &lt;= nums[i] &lt;= 10^51<=*nums*[*i*]<=105

Output

Maximum alternating sum.

Sample Input 1

5 6 7 8

Sample Output 1

8

Explanation: It is optimal to choose the subsequence [8] with alternating sum 8.

Sample Input 2

6 2 1 2 4 5

Sample Output 2

10

Explanation: It is optimal to choose the subsequence [6,1,5] with alternating sum (6 + 5) - 1 = 10.

Sample Input 3

4 2 5 3

Sample Output 3

7

Explanation: It is optimal to choose the subsequence [4,2,5] with alternating sum (4 + 5) - 2 = 7.

单个数字的要求是10^5，但是最后的和可能会超过int的范围，要用long

#include<vector>

#include<iostream>

#include<algorithm>

#include<math.h>

using namespace std;

class Solution {

public:

    long long maxAlternatingSum(vector<int>& nums) {

        int n = nums.size();

        long long odd = 0, even = nums[0];

        for (int i = 1; i < n; ++i) {

            tie(odd, even) = tuple{max(even - nums[i], odd), max(odd + nums[i], even)};

        }

        return even;

    }

};

int main()

{

    Solution solute;

    cout<<solute.superPow(a,b)<<endl;

    return 0;

}

#include<vector>

#include<iostream>

#include<algorithm>

#include<math.h>

using namespace std;

class Solution {

public:

    long long maxAlternatingSum(vector<int>& nums) {

        int n = nums.size();

        vector<long long> dp0(n, 0),  dp1(n,0);

        // dp0[i] 表示到 i 处，最佳子序列长度为偶数时的最大交替和

        // dp1[i] 表示到 i 处，最佳子序列长度为奇数时的最大交替和

        dp1[0] = nums[0]; // 初始化

        for(int i = 1; i < n; ++i)

        {

            dp0[i] = max(dp0[i-1], dp1[i-1]-nums[i]);

            //          当前数不考虑， 前面最佳的奇数长度的状态 - nums[i]

            dp1[i] = max(dp1[i-1], dp0[i-1]+nums[i]);

            //          当前数不考虑， 前面最佳的偶数长度的状态 + nums[i]

        }

        return max(dp0[n-1], dp1[n-1]);

    }

};

int main()

{

    vector<int> nums;

    int n=0;

    while (cin >> n)

    {

        nums.push\_back(n);

        if (cin.get() == '\n') break;

    }

    Solution solute;

    cout<<solute.maxAlternatingSum(nums)<<endl;

    return 0;

}

要换成long类型的nums

#include<vector>

#include<iostream>

#include<algorithm>

#include<math.h>

using namespace std;

class Solution {

public:

    long long maxAlternatingSum(vector<long>& nums) {

        int n = nums.size();

        vector<long long> dp0(n, 0),  dp1(n,0);

        dp1[0] = nums[0];

        for(int i = 1; i < n; ++i)

        {

            dp0[i] = max(dp0[i-1], dp1[i-1]-nums[i]);

            dp1[i] = max(dp1[i-1], dp0[i-1]+nums[i]);

        }

        return max(dp0[n-1], dp1[n-1]);

    }

};

int main()

{

    vector<long> nums;

    int n=0;

    while (cin >> n)

    {

        nums.push\_back(n);

        if (cin.get() == '\n') break;

    }

    Solution solute;

    cout<<solute.maxAlternatingSum(nums)<<endl;

    return 0;

}