2-D Range Maximum Sum

A problem that is simple to solve in one dimension is often much more difficult to solve in more than one dimension. Consider satisfying a boolean expression in conjunctive normal form in which each conjunct consists of exactly 3 disjuncts. This problem (3-SAT) is NP-complete. The problem 2-SAT is solved quite efficiently, however. In contrast, some problems belong to the same complexity class regardless of the dimensionality of the problem. Given a 2-dimensional array of positive and negative integers, find the sub-rectangle with the largest sum. The sum of a rectangle is the sum of all the elements in that rectangle. In this problem the subrectangle with the largest sum is referred to as the maximal sub-rectangle. A sub-rectangle is any contiguous sub-array of size 1 × 1 or greater located within the whole array. As an example, the maximal sub-rectangle of the array: 0 −2 −7 0 9 2 −6 2 −4 1 −4 1 −1 8 0 −2 is in the lower-left-hand corner: 9 2 −4 1 −1 8 and has the sum of 15.

Input

The input consists of an N × N array of integers. The input begins with a single positive integer N on a line by itself indicating the size of the square two dimensional array. This is followed by N2 integers separated by white-space (newlines and spaces). These N2 integers make up the array in row-major order (i.e., all numbers on the first row, left-to-right, then all numbers on the second row, left-to-right, etc.). N may be as large as 100. The numbers in the array will be in the range [−127, 127].

Output

The output is the sum of the maximal sub-rectangle.

Sample Input

4 0 -2 -7 0 9 2 -6 2 -4 1 -4 1 -1 8 0 -2

Sample Output

15

#include<bits/stdc++.h>

#define pb push\_back

#define pii pair<int,int>

#define int long long int

#define vec vector<int> // 2-D RANGE MAXIMUM SUM

#define inf 1e18 // SOLUTION COMPLEXITY -> O(N^4) OPTIMISED BY CUMMULATIVE SUM FROM O(N^6) SOLn.

using namespace std;

int32\_t main()

{

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL);

cout.tie(NULL);

int tt=1;

//cin>>tt;

while(tt--)

{

int n;

cin>>n;

int i,j;

// Taking input as well as Doing Cumulative 2 D sum

int A[n+1][n+1];

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

{

for(j=0;j<n;j++)

{

cin>>A[i][j];

if(i>0)

A[i][j]+=A[i-1][j];

if(j>0)

A[i][j]+=A[i][j-1];

if(i>0 && j>0)

A[i][j]-=A[i-1][j-1]; // By Inclusion Exclusion Principle avoiding double sum

}

}

int maxSubRect= -127\*100\*100,subRect=0,k,l;

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

{

for(j=0;j<n;j++)

{

for(k=i;k<n;k++)

{

for(l=j;l<n;l++)

{

subRect=A[k][l];

if(i>0)

subRect-=A[i-1][l];

if(j>0)

subRect-=A[k][j-1];

if(i>0 && j>0)

subRect+=A[i-1][j-1];

maxSubRect=max(maxSubRect,subRect); // This is our answer

}

}

}

}

cout<<maxSubRect<<"\n";

}

}