

## User

explain the algorithm of the code in detail :// Created by Nguyễn Mạnh Quân #include<bits/stdc++.h> using namespace std; #define mp make\_pair #define fi first #define se second #define pb push\_back #define sz size() #define ll long long #define FOR(i, a, b) for(int i = a; i <= b; ++i) #define FORD(i, a, b) for(int i = a; i >= b; --i) #define F(i, a, b) for(int i = a; i < b; ++i) #define FD(i, a, b) for(int i = a; i > b; --i) #define faster() ios\_base::sync\_with\_stdio(0); cin.tie(NULL); cout.tie(NULL); #define vi vector #define vll vector #define vb vector #define endl '\n' int main() { faster(); int n; cin >> n; int a[n][n], ans = 0; F(i, 0, n) F(j, 0, n) cin >> a[i][j]; F(i, 0, n) { F(j, 0, n) { int x = 0, y = 0, k = 0; while(k <= i && k <= j && i + k < n && j + k < n) { x += a[i - k][j - k] + a[i + k][j + k]; y += a[i - k][j + k] + a[i + k][j - k]; ans = max(ans, x - y); ++k; } x = 0; y = 0; k = 0; while(k <= i && k <= j && i + k + 1 < n && j + k + 1 < n) { x += a[i - k][j - k] + a[i + k + 1][j + k + 1]; y += a[i - k][j + k + 1] + a[i + k + 1][j - k]; ans = max(ans, x - y); ++k; } } } cout << ans; return 0; }

## Bing

The code is written in C++ and it is a program that finds the maximum difference between the sums of two diagonals of a square matrix. The algorithm of the code is as follows:

- The code defines some macros, such as mp for make\_pair, fi for first, se for second, etc., to make the code more concise and readable.
- The code also defines a function faster() that speeds up the input and output operations by synchronizing the standard C and C++ streams and disabling the tie between them.
- The code then reads an integer n from the input, which is the size of the square matrix, and declares a two-dimensional array a[n][n] to store the matrix elements. It also declares an integer variable ans to store the final answer, and initializes it to zero.
- The code then uses two nested loops to read the matrix elements from the input and store them in the array a[n][n].
- The code then uses another two nested loops to iterate over all possible positions of the matrix, denoted by (i, j). For each position, it performs the following steps:
  - It declares three integer variables x, y, and k, and initializes them to zero. These variables will be used to store the sums of the diagonals and the offset from the current position, respectively.
  - It then uses a while loop to calculate the sums of the two diagonals that pass through (i, j) and have the same length. The loop condition checks that k does not exceed i or j, and that (i + k) and (j + k) are within the matrix bounds. Inside the loop, it does the following:
    - It adds the values of a[i - k][j - k] and a[i + k][j + k] to x, which represents the sum of the main diagonal.
    - It adds the values of a[i - k][j + k] and a[i + k][j - k] to y, which represents the sum of the anti-diagonal.
    - It updates ans to be the maximum of itself and x - y, which is the difference between the sums of the two diagonals.
    - It increments k by one, to move to the next diagonal element.
  - It then resets x, y, and k to zero, and uses another while loop to calculate the sums of the two diagonals that pass through (i + 1, j + 1) and (i, j) and have different lengths. The loop condition checks that k does not exceed i or j, and that (i + k + 1) and (j + k + 1) are within the matrix bounds. Inside the loop, it does the following:
    - It adds the values of a[i - k][j - k] and a[i + k + 1][j + k + 1] to x, which represents the sum of one diagonal.
    - It adds the values of a[i - k][j + k + 1] and a[i + k + 1][j - k] to y, which represents the sum of another diagonal.
    - It updates ans to be the maximum of itself and x - y, which is again the difference between the sums of the two diagonals.
    - It increments k by one, to move to the next diagonal element.
- Finally, after iterating over all possible positions, the code prints out the value of ans, which is the maximum difference between any two diagonals of the matrix.

## User

why they use the second loop?

## Bing

The second loop is used to handle the case when the two diagonals have different lengths. For example, if the matrix is:

4 5 6 7 8 9

Then the first loop will calculate the sums of the diagonals that pass through (1, 1) and have the same length, which are:

- Main diagonal:  $1 + 5 + 9 = 15$
- Anti-diagonal:  $3 + 5 + 7 = 15$

The difference between these sums is zero.

The second loop will calculate the sums of the diagonals that pass through (2, 2) and (1, 1) and have different lengths, which are:

- One diagonal:  $4 + 8 = 12$
- Another diagonal:  $2 + 6 = 8$

The difference between these sums is four.

The second loop will also calculate the sums of the diagonals that pass through (0, 0) and (1, 1) and have different lengths, which are:

- One diagonal:  $5 = 5$
- Another diagonal:  $1 + 4 = 5$

The difference between these sums is zero.

Therefore, the second loop is necessary to find the maximum difference between any two diagonals of the matrix, which in this case is four.