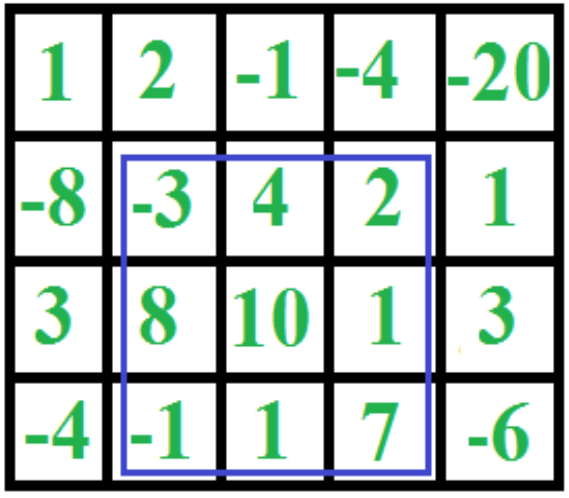
Dynamic Programming | Set 27 (Maximum sum rectangle in a 2D matrix)

Given a 2D array, find the maximum sum subarray in it. For example, in the following 2D array, the maximum sum subarray is highlighted with blue rectangle and sum of this subarray is 29.



This problem is mainly an extension of Largest Sum Contiguous Subarray for 1D array.

The **naive solution** for this problem is to check every possible rectangle in given 2D array. This solution requires 4 nested loops and time complexity of this solution would be O(n^4).

Kadane's algorithm for 1D array can be used to reduce the time complexity to O(n^3). The idea is to fix the left and right columns one by one and find the maximum sum contiguous rows for every left and right column pair. We basically find top and bottom row numbers (which have

Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion

maximum sum) for every fixed left and right column pair. To find the top and bottom row numbers, calculate sun of elements in every row from left to right and store these sums in an array say temp[]. So temp[i] indicates sum of elements from left to right in row i. If we apply Kadane's 1D algorithm on temp[], and get the maximum sum subarray of temp, this maximum sum would be the maximum possible sum with left and right as boundary columns. To get the overall maximum sum, we compare this sum with the maximum sum so far.

```
// Program to find maximum sum subarray in a given 2D array
#include <stdio.h>
#include <string.h>
#include <limits.h>
#define ROW 4
#define COL 5

// Implementation of Kadane's algorithm for 1D array. The function ret
// maximum sum and stores starting and ending indexes of the maximum s
// at addresses pointed by start and finish pointers respectively.
int kadane(int* arr, int* start, int* finish, int n)
{
    // initialize sum, maxSum and
    int sum = 0, maxSum = INT_MIN, i;

    // Just some initial value to check for all negative values case
    *finish = -1;

    // local variable
    int local_start = 0;

    for (i = 0; i < n; ++i)
    {
        sum += arr[i];
        if (sum < 0)
        {
            sum = 0;
            local_start = i+1;
        }
        else if (sum > maxSum)
        {
            maxSum = sum;
            *start = local_start;
            *finish = i;
        }
    }

     // There is at-least one non-negative number
```

Geometric Algorithms

Popular Posts

```
    if (*finish != -1)
        return maxSum;

    // Special Case: When all numbers in arr[] are negative
    maxSum = arr[0];
    *start = *finish = 0;

    // Find the maximum element in array
    for (i = 1; i < n; i++)
    {
        if (arr[i] > maxSum)
        {
            maxSum = arr[i];
            *start = *finish = i;
        }
    }
    return maxSum;
}

// The main function that finds maximum sum rectangle in M[][]
void findMaxSum(int M[][COL])
{
    // Variables to store the final output
    int maxSum = INT_MIN, finalLeft, finalRight, finalTop, finalBottom

    int left, right, i;
    int temp[ROW], sum, start, finish;

    // Set the left column
    for (left = 0; left < COL; ++left)
    {
        // Initialize all elements of temp as 0
        memset(temp, 0, sizeof(temp));

        // Set the right column for the left column set by outer loop
        for (right = left; right < COL; ++right)
        {
            // Calculate sum between current left and right for every
            for (i = 0; i < ROW; ++i)
                temp[i] += M[i][right];

            // Find the maximum sum subarray in temp[]. The kadane() f
            // also sets values of start and finish.  So 'sum' is sum
            // rectangle between (start, left) and (finish, right) whi
            //  maximum sum with boundary columns strictly as left and
            sum = kadane(temp, &start, &finish, ROW);
```

```
            // Compare sum with maximum sum so far. If sum is more, th
            // maxSum and other output values
            if (sum > maxSum)
            {
                maxSum = sum;
                finalLeft = left;
                finalRight = right;
                finalTop = start;
                finalBottom = finish;
            }
        }
    }

    // Print final values
    printf("(Top, Left) (%d, %d)\n", finalTop, finalLeft);
    printf("(Bottom, Right) (%d, %d)\n", finalBottom, finalRight);
    printf("Max sum is: %d\n", maxSum);
}

// Driver program to test above functions
int main()
{
    int M[ROW][COL] = {{1, 2, -1, -4, -20},
                       {-8, -3, 4, 2, 1},
                       {3, 8, 10, 1, 3},
                       {-4, -1, 1, 7, -6}
                      };

    findMaxSum(M);

    return 0;
}
```

Output:

```
(Top, Left) (1, 1)
(Bottom, Right) (3, 3)
Max sum is: 29
```

Time Complexity: O(n^3)

This article is compiled by Aashish Barnwal. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recent Comments

▶ C++ Code
AdChoices
▶ SUM Function
▶ The SUM of All
▶ Convert SUM
AdChoices
▶ Convert SUM
▶ SUM Time
▶ SUM To

Related Tpoics:

- Remove minimum elements from either side such that 2*min becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3

Whatever • a month ago

Time Complexity is precisely O(Col^2 * Row) !!

 • Reply • Share ›

Guest • a month ago

Time Complexity is precisely O(Col^2 * Row) :)

 • Reply • Share ›

Guest • 2 months ago

Hi, there is an O(n^2) algorithm for this using O(n^2) extra space.

Input :- given matrix a[][] of dimension m*n.

Algo:-

1) create a new matrix n[][] of dimension m*n.

2) n[i][j] = for k = 0 to i (sum of a[k][j]). That is the new matrix is the column wi

3) Now apply the well known standard "Max rectangle in histogram" algorithm
http://www.geeksforgeeks.org/l...

a) Compare the curr_max with max.

return max.

Time complexity - O(n^2).

 • Reply • Share ›

jafar ➔ Guest • 2 months ago

I don't see how that will give us the desired result.since applying the m
slot' e.g: using 3 out of a slot with value 4.or using different sets of row

rectangle.

Reply • Share ›

Aditya • 3 months ago

This will not work for matrix
{ {-1,-1,-1},
{9, 9, -1},
{-1,-1,-1}}.
It will give result as 17 but the answer should be 18.

Reply • Share ›

meh Aditya • 3 months ago

It works, I tried the code and it returns 18.

Reply • Share ›

Mythreya J L • 3 months ago

Can be reduced to O(n^2).

1. Find integral "image" representation of the matrix: O(n^2)
2. Find the position of the maximum element, (x1,y1) in the integral image: O(

3. Find the position of the minimum element, (x0,y0) that appears strictly befor
means it has to be at a lower row AND lower column: O(n^2).

Maximum subarray is (x0:x1, y0:y1), time complexity: O(n^2)+O(n^2)+O(n^2) =

Reply • Share ›

jerrym Mythreya J L • 3 months ago

Unfortunately, there is no guarantee that the maximum element in the i
of the maximum subarray.

Reply • Share ›

Sumit Khanna • 5 months ago

Hey!...time complexity for BRUTE FORCE or NAIVE is O(n^6) and not O(n^4). bottomRights for every topleft,,and N^2 for calculating sum of every considered rectangles,,,thus total time is O(n^6) ...

2 ▲ | ▼ • Reply • Share ›

AlienOnEarth ➦ Sumit Khanna • 16 days ago

Yes Sumit Khanna. Even I was thinking the same. I think you are right. confirmed on other websites.

@GeeksforGeeks:
Please consider this once and update if looks correct to you.

▲ | ▼ • Reply • Share ›

intdydx • 7 months ago

I implemented an FFT-based solution here:
https://github.com/thearn/maxi...

▲ | ▼ • Reply • Share ›

Sumant Kumar Dev • 9 months ago

Neha Modi WA for {{1,-9,-5,2,8},

{11, 4, 3, 7, 4},

{1, -5, -7, 8, -3},

{2, 3, -4, 5, -2}.

};.

2 ▲ | ▼ • Reply • Share ›

Neha Modi • 9 months ago

of order n^2.

#include<iostream.h>

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
int max(int a, int b, int c).
{ if(a>b && a>c).
return a;.
if(b>a && b>c).
return b;.
return c;.
}
void main()
{ int arr[4][5]={{1,2,-1,-4,-20},{-8,-3,4,2,1},{3,8,10,1,3},{-4,-1,1,7,-6}}, s[4][5], i,j;.
int max1=0;.
for(i=0;i<4;i++).
s[i][0]=arr[i][0];.
for(j=0;j<5;j++).
s[0][j]=arr[0][j];.
```

see more

• Reply • Share ›

Yash Pareek • 10 months ago

@pankaj....it is like if u have array from left to right as:
1 2
3 4
5 6

then temp={(1+2),(3+4),(5+6)};

1 • Reply • Share ›

Pankaj Goyal • 10 months ago

can anybody tell me how are the values filled in the temp[] array? horizontally

• Reply • Share ›

fenglvming • 10 months ago

```
 /* Paste your code here (You may delete these lines if not writing c
for (left = 0; left < COL; ++left)
    {
        // Initialize all elements of temp as 0
        memset(temp, 0, sizeof(temp));

        // Set the right column for the left column set by outer loop
        for (right = left; right < COL; ++right)
        {
            // Calculate sum between current left and right for every
            for (i = 0; i < ROW; ++i)
                temp[i] += M[i][right];

            // Find the maximum sum subarray in temp[]. The kadane() f
            // also sets values of start and finish.  So 'sum' is sum
            // rectangle between (start, left) and (finish, right) wh
            //  maximum sum with boundary columns strictly as left an
            sum = kadane(temp, &start, &finish, ROW);
```

see more

▲ | ▼ • Reply • Share ›

Born Actor • 10 months ago

```
 #include <iostream>
#include<string>
#include<sstream>
#include<iomanip>
# include <stdio.h>
# include <math.h>
#include <vector>
```

```
#include <stdlib.h>
using namespace std;
int n_r;
int n_c;
int n;
int a[50];
pair < int , pair < int , int > > function(int i, int j);
 pair < int , pair < pair < int,  int > , pair < int ,int > > > answe
pair < int , pair < int , int > > function2(int i, int m, int j);
pair < int , pair < int , int > > max(pair < int , pair < int , int >
int b[50][50];
```

see more

• Reply • Share ›

GeeksforGeeks • 11 months ago

Ashish Anand: Thanks for suggesting the fix. We have updated the post.

• Reply • Share ›

logic_bomber • 11 months ago

Why is it under 'Dynamic Programming'? Do help me understand what proper here.

• Reply • Share ›

GeeksforGeeks • 11 months ago

Thanks for pointing this out. We will look into this and update the post.

• Reply • Share ›

Chaitanya T V Krishna ➦ GeeksforGeeks • 6 months ago

We do not escape n2 complexity in column. But we use Kandane's alg sum contiguous rows to convert n2 to n.

• Reply • Share ›

Vikas Singla • 11 months ago

GeeksforGeeks.....above code gives wrong sol for my following comments....
correct if I&#039m not wrong.

• Reply • Share ›

Vikas Singla • 11 months ago

#define ROW 6.
#define COL 1.
int main()
{

int M[ROW][COL] = {-1,-2,4,-3,-2, 2.

};.

findMaxSum(M);.

return 0;.
}

solution with this is.
(Top, Left) (5, 0).
(Bottom, Right) (2, 0).
Max sum is: 4.

which is wrong I think so....

• Reply • Share ›

Paparao Veeragandham • 11 months ago

> Given a matrix of both +ve **and** -ve numbers, find out the maximum sum

```
int s[ROW][COL];
```

```
void computeSumMatrix(int a[][COL], int r, int c) {
 for (int i = 0; i < r; i++)
  if (i == 0)
   s[i][0] = a[i][0];
  else
   s[i][0] = s[i - 1][0] + a[i][0];

 for (int j = 0; j < c; j++)
  if (j == 0)
   s[0][j] = a[0][j];
  else
   s[0][j] = s[0][j - 1] + a[0][j];

 for (int i = 1; i < r; i++) {
  for (int j = 1; j < c; j++) {
```

see more

• Reply • Share ›

rkl • a year ago

The lines

```
// Calculate sum between current left and right for every row 'i'
      for (i = 0; i < ROW; ++i)
          temp[i] += M[i][right];
```

is wrong.

We should calculate temp[i] = M[i][left] + ... + M[i][right]
which should be done in a loop.

```
for (i = 0; i < ROW; ++i)
```

```
for (i = 0; i < row; ++i)
    for(int j = left; j <=right; j++)
        temp[i] += M[i][j];
```

Reply • Share ›

rkl ↗ rkl • a year ago

I think I got the point. Please ignore my previous comment.
In the first iteration when right=left, there is only 1 column in temp per i.
temp[i] = 0 (temp is initialized to 0) + M[i][right or left, its same in first it

In 2nd iteration, when right=left+1,
temp[i] = old_temp[i] (which is M[i][left]) + M[i][right]

Reply • Share ›

abhishek08aug • a year ago

Intelligent :D

```
/* Paste your code here (You may delete these lines if not writing c
```

Reply • Share ›

chitra • a year ago

```
/* Paste your code here (You may delete these lines if not writing c
```

Reply • Share ›

sk • a year ago

Your kadance method wont work to find start and finish index when given 2d m

So given solution will be wrong.

Reply • Share ›

Ashish Anand • a year ago

Alankrita is right. The function kadane is wrongly written.

It should rather be like below:

```
int kadane(int* arr, int* start, int* finish, int n)
{

// initialize sum, maxSum and start

int sum = 0, maxSum = INT_MIN, i;

// local variable

int local_start = 0;

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

{

sum += arr[i];
```

see more

• Reply • Share ›

Kashish Babbar • a year ago

How is the naive solution O(n^4) ?
Shouldn't it be O(n^6) if we are checking all the possible rectangles ?
O(n^2) for choosing the start point and O(n^2) for choosing the end point i.e. 4 another two for loops ?

• Reply • Share ›

Prateek Caire ➦ Kashish Babbar • 9 months ago

Even i think same. GFG needs to correct..

```
/* Paste your code here (You may delete these lines if not wri
```

Reply • Share ›

curieuxtoujours • a year ago

there is a O(n^2) solution with O(n^2) space complexity.

1 Reply • Share ›

curieuxtoujours curieuxtoujours • a year ago

Algo :

1. Create another matrix of the same size mat[R][C].
2. Fill the mat[R][C] such that mat[i][j] contains the sum of the matrix M
something similar to kadane's algo.
3. Now run two loops for i and j .. like the above post , but now you don
you can find it from previously constructed mat in constant time. How

Reply • Share ›

cdegree curieuxtoujours • a year ago

for start position, you need n^2;
and for end position, you also need n^2.
so is that n^4?

Reply • Share ›

cdegree cdegree • a year ago

I mean O(n^2) time
O(n^2) time
and O(n^4) time

Reply • Share ›

Arvind B R • a year ago

this part is wrong ,start will be updated even though if the start is not a part of

```
for (i = 0; i < n; ++i)
    {
        sum += arr[i];
        if (sum < 0)
        {
            sum = 0;
            *start = i+1;
        }
        else if (sum > maxSum)
        {
            maxSum = sum;
            *finish = i;
        }
    }
```

1 ^ | v • Reply • Share ›

cdegree ➚ Arvind B R • a year ago

can't agree more

^ | v • Reply • Share ›

Sah39 ➚ Arvind B R • a year ago

hai da BR....

```
/* Paste your code here (You may delete these lines if not wri
```

^ | v • Reply • Share ›

Sandeep Jain • a year ago

The above code is for 2D array. For 1D array, you may use http://www.geeksf

 • Reply • Share ›

Alankrita Pathak • a year ago

I just copied and executed code of implementation of Kadane&#039s algorithm
int kadane(int* arr, int* start, int* finish, int n).
for the given input {-1,-2,4,-3,-2, 2} I am getting start = 5 , and finish = 2..I gues
outcome..help me in this issue..

 • Reply • Share ›

jobin • a year ago

you should rename the variable start to localstart
else if (sum > maxSum)
{ maxSum = sum;
*finish = i;
*start = localstart;
}

otherwise your start will be modified everytime the sum upto current point from
not the starting point for max contiguous sum.

 • Reply • Share ›

rahul • a year ago

Should the complexity not be O(n4) as Kadane's itself is O(n).

```
/* Paste your code here (You may delete these lines if not writing c
```

 • Reply • Share ›

Zack ➦ rahul • a year ago

No,because your could do the sum of row in a growing manner.

say you want to calculate any combination of columns start from 1,
1--> col[] = column 1 's value
2--> add column 1's value to col[] and you get column 1-2's sum
and so on.
So the for i=0..n-1, for j=i..n takes O(n^2)
and each iteration takes O(n) to calculate the best range sum.

```
/* Paste your code here (You may delete these lines if not wri
```

• Reply • Share ›

Warrior101 ➚ Zack • 2 months ago

No. The complexity is O(n^4), even if you do progressive adds. are 3 for-loops and with Kadanes, that becomes O(n^4).

• Reply • Share ›

San Holo ➚ Zack • a year ago

But look - say you have 2D array 3x3.
The structure is:
for loop1(for loop2(for loop3, for loop4))
First, the most outer one will be done 3 times. The second - for
The third AND the fourth will be done the same amount of times
So it's like this:
3 -> 3+3 - 6 times during each loop, gives us 18.
2 -> 3+3 - 18 + 12 = 30
1 -> 3+3 - 30 + 6 = 36.
n^3 = 27; 36 > 27.
I'll try to figure out if there's a way to connect both loops... Of cc
hope it is), if there's an error in how I think, please respond.

• Reply • Share ›

San Holo ➚ San Holo • a year ago

Well, anyway, I corrected my code and it's definitely n^3

• Reply • Share ›

Subscribe Add Disqus to your site