Contents

Chapter 1

Search in a row wise and column wise sorted matrix

Given an n x n matrix, where every row and column is sorted in increasing order. Given a number x, how to decide whether this x is in the matrix. The designed algorithm should have linear time complexity.

Thanks to devendraiiitfor suggesting below approach.

1) Start with top right element
2) Loop: compare this element e with x
....i) if they are equal then return its position
...ii) e x then move it to left (if out of bound of matrix then break return false)
3) repeat the i), ii) and iii) till you find element or returned false

Implementation:

```
 #include<stdio.h>

/* Searches the element x in mat[][]. If the element is found,
    then prints its position and returns true, otherwise prints
    "not found" and returns false */
int search(int mat[4][4], int n, int x)
{
   int i = 0, j = n-1;  //set indexes for top right element
   while ( i < n && j >= 0 )
   {
      if ( mat[i][j] == x )
      {
         printf("\n Found at %d, %d", i, j);
         return 1;
      }
      if ( mat[i][j] > x )
        j--;
      else //  if mat[i][j] < x
        i++;
   }

   printf("\n Element not found");
   return 0;  // if ( i==n || j== -1 )
}
```

```
// driver program to test above function
int main()
{
  int mat[4][4] = { {10, 20, 30, 40},
                    {15, 25, 35, 45},
                    {27, 29, 37, 48},
                    {32, 33, 39, 50},
                  };
  search(mat, 4, 29);
  getchar();
  return 0;
}
```

Time Complexity: O(n)

The above approach will also work for m x n matrix (not only for n x n). Complexity would be O(m + n).

Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.

Source

http://www.geeksforgeeks.org/search-in-row-wise-and-column-wise-sorted-matrix/

Category: Arrays

Chapter 2

Print a given matrix in spiral form

Given a 2D array, print it in spiral form. See the following examples.

```
Input:
        1    2   3   4
        5    6   7   8
        9   10  11  12
        13  14  15  16
Output:
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```

```
Input:
        1   2   3   4  5   6
        7   8   9  10  11  12
        13  14  15 16  17  18
Output:
1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11
```

Solution:

```
/* This code is adopted from the solution given
   @ http://effprog.blogspot.com/2011/01/spiral-printing-of-two-dimensional.html */

#include <stdio.h>
#define R 3
#define C 6

void spiralPrint(int m, int n, int a[R][C])
{
    int i, k = 0, l = 0;

    /*  k - starting row index
        m - ending row index
        l - starting column index
```

```
        n - ending column index
        i - iterator
    */

    while (k < m && l < n)
    {
        /* Print the first row from the remaining rows */
        for (i = l; i < n; ++i)
        {
            printf("%d ", a[k][i]);
        }
        k++;

        /* Print the last column from the remaining columns */
        for (i = k; i < m; ++i)
        {
            printf("%d ", a[i][n-1]);
        }
        n--;

        /* Print the last row from the remaining rows */
        if ( k < m)
        {
            for (i = n-1; i >= l; --i)
            {
                printf("%d ", a[m-1][i]);
            }
            m--;
        }

        /* Print the first column from the remaining columns */
        if (l < n)
        {
            for (i = m-1; i >= k; --i)
            {
                printf("%d ", a[i][l]);
            }
            l++;
        }
    }
}

/* Driver program to test above functions */
int main()
{
    int a[R][C] = { {1,  2,  3,  4,  5,  6},
        {7,  8,  9,  10, 11, 12},
        {13, 14, 15, 16, 17, 18}
    };

    spiralPrint(R, C, a);
    return 0;
}
```

```
/* OUTPUT:
  1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11
*/
```

Time Complexity: Time complexity of the above solution is O(mn).

Please write comments if you find the above code incorrect, or find other ways to solve the same problem.

Source

http://www.geeksforgeeks.org/print-a-given-matrix-in-spiral-form/

Category: Arrays

Chapter 3

About wgpshashank

Shashank is Passionate About Computer Science, Problem Solving & Technology,he graduated from Birla Institute of Technology Mesra. Design and Analysis of Algorithms ,Application of Data Structures are his area of Interested & he Wants to Contribute to Computer Science. You can find him more active on his personal blog "Cracking The Code" http://shashank7s.blogspot.com Cheers !!!

Post navigation

← Find the repeating and the missing | Added 3 new methods C++ Internals | Default Constructors | Set 1 →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 4

A Boolean Matrix Question

Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as 1.

```
Example 1
The matrix
1 0
0 0
should be changed to following
1 1
1 0

Example 2
The matrix
0 0 0
0 0 1
should be changed to following
0 0 1
1 1 1

Example 3
The matrix
1 0 0 1
0 0 1 0
0 0 0 0
should be changed to following
1 1 1 1
1 1 1 1
1 0 1 1
```

Method 1 (Use two temporary arrays)
1) Create two temporary arrays row[M] and col[N]. Initialize all values of row[] and col[] as 0.
2) Traverse the input matrix mat[M][N]. If you see an entry mat[i][j] as true, then mark row[i] and col[j] as true.
3) Traverse the input matrix mat[M][N] again. For each entry mat[i][j], check the values of row[i] and col[j]. If any of the two values (row[i] or col[j]) is true, then mark mat[i][j] as true.

Thanks to Dixit Sethi for suggesting this method.

```
#include <stdio.h>
#define R 3
#define C 4

void modifyMatrix(bool mat[R][C])
{
    bool row[R];
    bool col[C];

    int i, j;


    /* Initialize all values of row[] as 0 */
    for (i = 0; i < R; i++)
    {
       row[i] = 0;
    }


    /* Initialize all values of col[] as 0 */
    for (i = 0; i < C; i++)
    {
       col[i] = 0;
    }


    /* Store the rows and columns to be marked as 1 in row[] and col[]
       arrays respectively */
    for (i = 0; i < R; i++)
    {
        for (j = 0; j < C; j++)
        {
            if (mat[i][j] == 1)
            {
                row[i] = 1;
                col[j] = 1;
            }
        }
    }

    /* Modify the input matrix mat[] using the above constructed row[] and
       col[] arrays */
    for (i = 0; i < R; i++)
    {
        for (j = 0; j < C; j++)
        {
            if ( row[i] == 1 || col[j] == 1 )
            {
                mat[i][j] = 1;
            }
        }
```

```
    }
}

/* A utility function to print a 2D matrix */
void printMatrix(bool mat[R][C])
{
    int i, j;
    for (i = 0; i < R; i++)
    {
        for (j = 0; j < C; j++)
        {
            printf("%d ", mat[i][j]);
        }
        printf("\n");
    }
}

/* Driver program to test above functions */
int main()
{
    bool mat[R][C] = { {1, 0, 0, 1},
        {0, 0, 1, 0},
        {0, 0, 0, 0},
    };

    printf("Input Matrix \n");
    printMatrix(mat);

    modifyMatrix(mat);

    printf("Matrix after modification \n");
    printMatrix(mat);

    return 0;
}
```

Output:

```
Input Matrix
1 0 0 1
0 0 1 0
0 0 0 0
Matrix after modification
1 1 1 1
1 1 1 1
1 0 1 1
```

Time Complexity: O(M*N)
Auxiliary Space: O(M + N)

Method 2 (A Space Optimized Version of Method 1)
This method is a space optimized version of above method 1. This method uses the first row and first column

of the input matrix in place of the auxiliary arrays row[] and col[] of method 1. So what we do is: first take care of first row and column and store the info about these two in two flag variables rowFlag and colFlag. Once we have this info, we can use first row and first column as auxiliary arrays and apply method 1 for submatrix (matrix excluding first row and first column) of size (M-1)*(N-1).

1) Scan the first row and set a variable rowFlag to indicate whether we need to set all 1s in first row or not.
2) Scan the first column and set a variable colFlag to indicate whether we need to set all 1s in first column or not.
3) Use first row and first column as the auxiliary arrays row[] and col[] respectively, consider the matrix as submatrix starting from second row and second column and apply method 1.
4) Finally, using rowFlag and colFlag, update first row and first column if needed.

Time Complexity: O(M*N)
Auxiliary Space: O(1)

Thanks to Sidhfor suggesting this method.

Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.

Source

http://www.geeksforgeeks.org/a-boolean-matrix-question/

Category: Arrays

Chapter 5

Print unique rows in a given boolean matrix

Given a binary matrix, print all unique rows of the given matrix.

```
Input:
    {0, 1, 0, 0, 1}
        {1, 0, 1, 1, 0}
        {0, 1, 0, 0, 1}
        {1, 1, 1, 0, 0}
Output:
    0 1 0 0 1
    1 0 1 1 0
    1 1 1 0 0
```

Method 1 (Simple)
A simple approach is to check each row with all processed rows. Print the first row. Now, starting from the second row, for each row, compare the row with already processed rows. If the row matches with any of the processed rows, don't print it. If the current row doesn't match with any row, print it.

Time complexity: O(ROW^2 x COL)
Auxiliary Space: O(1)

Method 2 (Use Binary Search Tree)
Find the decimal equivalent of each row and insert it into BST. Each node of the BST will contain two fields, one field for the decimal value, other for row number. Do not insert a node if it is duplicated. Finally, traverse the BST and print the corresponding rows.

Time complexity: O(ROW x COL + ROW x log(ROW))
Auxiliary Space: O(ROW)

This method will lead to Integer Overflow if number of columns is large.

Method 3 (Use Trie data structure)
Since the matrix is boolean, a variant of Trie data structure can be used where each node will be having two children one for 0 and other for 1. Insert each row in the Trie. If the row is already there, don't print the row. If row is not there in Trie, insert it in Trie and print it.

Below is C implementation of method 3.

```
//Given a binary matrix of M X N of integers, you need to return only unique rows of binary array
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define ROW 4
#define COL 5

// A Trie node
typedef struct Node
{
    bool isEndOfCol;
    struct Node *child[2]; // Only two children needed for 0 and 1
} Node;


// A utility function to allocate memory for a new Trie node
Node* newNode()
{
    Node* temp = (Node *)malloc( sizeof( Node ) );
    temp->isEndOfCol = 0;
    temp->child[0] = temp->child[1] = NULL;
    return temp;
}

// Inserts a new matrix row to Trie.  If row is already
// present, then returns 0, otherwise insets the row and
// return 1
bool insert( Node** root, int (*M)[COL], int row, int col )
{
    // base case
    if ( *root == NULL )
        *root = newNode();

    // Recur if there are more entries in this row
    if ( col < COL )
        return insert ( &( (*root)->child[ M[row][col] ] ), M, row, col+1 );

    else // If all entries of this row are processed
    {
        // unique row found, return 1
        if ( !( (*root)->isEndOfCol ) )
            return (*root)->isEndOfCol = 1;

        // duplicate row found, return 0
        return 0;
    }
}

// A utility function to print a row
void printRow( int (*M)[COL], int row )
{
    int i;
```

```
    for( i = 0; i < COL; ++i )
        printf( "%d ", M[row][i] );
    printf("\n");
}

// The main function that prints all unique rows in a
// given matrix.
void findUniqueRows( int (*M)[COL] )
{
    Node* root = NULL; // create an empty Trie
    int i;

    // Iterate through all rows
    for ( i = 0; i < ROW; ++i )
        // insert row to TRIE
        if ( insert(&root, M, i, 0) )
            // unique row found, print it
            printRow( M, i );
}

// Driver program to test above functions
int main()
{
    int M[ROW][COL] = {{0, 1, 0, 0, 1},
        {1, 0, 1, 1, 0},
        {0, 1, 0, 0, 1},
        {1, 0, 1, 0, 0}
    };

    findUniqueRows( M );

    return 0;
}
```

Time complexity: O(ROW x COL)
Auxiliary Space: O(ROW x COL)

This method has better time complexity. Also, relative order of rows is maintained while printing.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

http://www.geeksforgeeks.org/print-unique-rows/

Category: Arrays Tags: Advance Data Structures, Advanced Data Structures

Post navigation

← Median of two sorted arrays of different sizes Microsoft Interview | Set 8 →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 6

Maximum size square sub-matrix with all 1s

Given a binary matrix, find out the maximum size square sub-matrix with all 1s.

For example, consider the below binary matrix.

```
0  1  1  0  1
1  1  0  1  0
0  1  1  1  0
1  1  1  1  0
1  1  1  1  1
0  0  0  0  0
```

The maximum square sub-matrix with all set bits is

```
1  1  1
1  1  1
1  1  1
```

Algorithm:
Let the given binary matrix be M[R][C]. The idea of the algorithm is to construct an auxiliary size matrix S[][] in which each entry S[i][j] represents size of the square sub-matrix with all 1s including M[i][j] where M[i][j] is the rightmost and bottommost entry in sub-matrix.

```
1) Construct a sum matrix S[R][C] for the given M[R][C].
     a) Copy first row and first columns as it is from M[][] to S[][]
     b) For other entries, use following expressions to construct S[][]
         If M[i][j] is 1 then
            S[i][j] = min(S[i][j-1], S[i-1][j], S[i-1][j-1]) + 1
         Else /*If M[i][j] is 0*/
            S[i][j] = 0
2) Find the maximum entry in S[R][C]
3) Using the value and coordinates of maximum entry in S[i], print
   sub-matrix of M[][]
```

For the given M[R][C] in above example, constructed S[R][C] would be:

```
0  1  1  0  1
1  1  0  1  0
0  1  1  1  0
1  1  2  2  0
1  2  2  3  1
0  0  0  0  0
```

The value of maximum entry in above matrix is 3 and coordinates of the entry are (4, 3). Using the maximum value and its coordinates, we can find out the required sub-matrix.

```
#include<stdio.h>
#define bool int
#define R 6
#define C 5

void printMaxSubSquare(bool M[R][C])
{
  int i,j;
  int S[R][C];
  int max_of_s, max_i, max_j;

  /* Set first column of S[][]*/
  for(i = 0; i < R; i++)
     S[i][0] = M[i][0];

  /* Set first row of S[][]*/
  for(j = 0; j < C; j++)
     S[0][j] = M[0][j];

  /* Construct other entries of S[][]*/
  for(i = 1; i < R; i++)
  {
    for(j = 1; j < C; j++)
    {
      if(M[i][j] == 1)
        S[i][j] = min(S[i][j-1], S[i-1][j], S[i-1][j-1]) + 1;
      else
        S[i][j] = 0;
    }
  }

  /* Find the maximum entry, and indexes of maximum entry
     in S[][] */
  max_of_s = S[0][0]; max_i = 0; max_j = 0;
  for(i = 0; i < R; i++)
  {
    for(j = 0; j < C; j++)
    {
      if(max_of_s < S[i][j])
```

```
      {
         max_of_s = S[i][j];
         max_i = i;
         max_j = j;
      }
    }
  }

  printf("\n Maximum size sub-matrix is: \n");
  for(i = max_i; i > max_i - max_of_s; i--)
  {
    for(j = max_j; j > max_j - max_of_s; j--)
    {
      printf("%d ", M[i][j]);
    }
    printf("\n");
  }
}

/* UTILITY FUNCTIONS */
/* Function to get minimum of three values */
int min(int a, int b, int c)
{
  int m = a;
  if (m > b)
    m = b;
  if (m > c)
    m = c;
  return m;
}

/* Driver function to test above functions */
int main()
{
  bool M[R][C] =  {{0, 1, 1, 0, 1},
                   {1, 1, 0, 1, 0},
                   {0, 1, 1, 1, 0},
                   {1, 1, 1, 1, 0},
                   {1, 1, 1, 1, 1},
                   {0, 0, 0, 0, 0}};

  printMaxSubSquare(M);
  getchar();
}
```

Time Complexity: O(m*n) where m is number of rows and n is number of columns in the given matrix.
Auxiliary Space: O(m*n) where m is number of rows and n is number of columns in the given matrix.
Algorithmic Paradigm: Dynamic Programming

Please write comments if you find any bug in above code/algorithm, or find other ways to solve the same problem

Source

http://www.geeksforgeeks.org/maximum-size-sub-matrix-with-all-1s-in-a-binary-matrix/

Category: Arrays Tags: Dynamic Programming

Post navigation

← Print list items containing all characters of a given word Inorder Tree Traversal without Recursion →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 7

Inplace M x N size matrix transpose | Updated

About four months of gap (missing GFG), a new post. Given an M x N matrix, transpose the matrix without auxiliary memory.It is easy to transpose matrix using an auxiliary array. If the matrix is symmetric in size, we can transpose the matrix inplace by mirroring the 2D array across it's diagonal (try yourself). How to transpose an arbitrary size matrix inplace? See the following matrix,

```
 a b c       a d g j
d e f  ==>  b e h k
g h i       c f i l
j k l
```

As per 2D numbering in C/C++, corresponding location mapping looks like,

```
Org element New
0     a     0
1     b     4
2     c     8
3     d     1
4     e     5
5     f     9
6     g     2
7     h     6
8     i    10
9     j     3
10    k     7
11    l    11
```

Note that the first and last elements stay in their original location. We can easily see the transformation forms few permutation cycles. 1->4->5->9->3->1 – Total 5 elements form the cycle 2->8->10->7->6->2 – Another 5 elements form the cycle 0 – Self cycle 11 – Self cycle From the above example, we can easily devise an algorithm to move the elements along these cycles. *How can we generate permutation cycles?* Number of elements in both the matrices are constant, given by N = R * C, where R is row count and C is column count. An element at location *ol* (old location in R x C matrix), moved to *nl* (new location in C x R matrix). We need to establish relation between *ol, nl, R* and *C*. Assume *ol = A[or][oc]*. In C/C++ we can calculate the element address as,

```
 ol = or x C + oc (ignore base reference for simplicity)
```

It is to be moved to new location *nl* in the transposed matrix, say *nl = A[nr][nc]*, or in C/C++ terms

```
 nl = nr x R + nc (R - column count, C is row count as the matrix is transposed)
```

Observe, *nr = oc* and *nc = or*, so replacing these for *nl*,

```
 nl = oc x R + or -----> [eq 1]
```

after solving for relation between *ol* and *nl*, we get

```
 ol     = or x C     + oc
ol x R = or x C x R + oc x R
       = or x N     + oc x R    (from the fact R * C = N)
       = or x N     + (nl - or) --- from [eq 1]
       = or x (N-1) + nl
```

OR,

```
 nl = ol x R - or x (N-1)
```

Note that the values of *nl* and *ol* never go beyond *N-1*, so considering modulo division on both the sides by (*N-1*), we get the following based on properties of congruence,

```
 nl mod (N-1) = (ol x R - or x (N-1)) mod (N-1)
              = (ol x R) mod (N-1) - or x (N-1) mod(N-1)
              = ol x R mod (N-1), since second term evaluates to zero
nl = (ol x R) mod (N-1), since nl is always less than N-1
```

A curious reader might have observed the significance of above relation. Every location is scaled by a factor of R (row size). It is obvious from the matrix that every location is displaced by scaled factor of R. The actual multiplier depends on congruence class of (N-1), i.e. the multiplier can be both -ve and +ve value of the congruent class. Hence every location transformation is simple modulo division. These modulo divisions form cyclic permutations. We need some book keeping information to keep track of already moved elements. Here is code for inplace matrix transformation,

```
// Program for in-place matrix transpose
#include <stdio.h>
#include <iostream>
#include <bitset>
#define HASH_SIZE 128

using namespace std;

// A utility function to print a 2D array of size nr x nc and base address A
void Print2DArray(int *A, int nr, int nc)
{
    for(int r = 0; r < nr; r++)
    {
```

```
        for(int c = 0; c < nc; c++)
            printf("%4d", *(A + r*nc + c));

        printf("\n");
    }

    printf("\n\n");
}

// Non-square matrix transpose of matrix of size r x c and base address A
void MatrixInplaceTranspose(int *A, int r, int c)
{
    int size = r*c - 1;
    int t; // holds element to be replaced, eventually becomes next element to move
    int next; // location of 't' to be moved
    int cycleBegin; // holds start of cycle
    int i; // iterator
    bitset<HASH_SIZE> b; // hash to mark moved elements

    b.reset();
    b[0] = b[size] = 1;
    i = 1; // Note that A[0] and A[size-1] won't move
    while (i < size)
    {
        cycleBegin = i;
        t = A[i];
        do
        {
            // Input matrix [r x c]
            // Output matrix 1
            // i_new = (i*r)%(N-1)
            next = (i*r)%size;
            swap(A[next], t);
            b[i] = 1;
            i = next;
        }
        while (i != cycleBegin);

        // Get Next Move (what about querying random location?)
        for (i = 1; i < size && b[i]; i++)
            ;
        cout << endl;
    }
}

// Driver program to test above function
int main(void)
{
    int r = 5, c = 6;
    int size = r*c;
    int *A = new int[size];

    for(int i = 0; i < size; i++)
        A[i] = i+1;
```

```
    Print2DArray(A, r, c);
    MatrixInplaceTranspose(A, r, c);
    Print2DArray(A, c, r);

    delete[] A;

    return 0;
}
```

Output:

```
    1   2   3   4   5   6
   7   8   9  10  11  12
  13  14  15  16  17  18
  19  20  21  22  23  24
  25  26  27  28  29  30

   1   7  13  19  25
   2   8  14  20  26
   3   9  15  21  27
   4  10  16  22  28
   5  11  17  23  29
   6  12  18  24  30
```

Extension: 17 – March – 2013 Some readers identified similarity between the matrix transpose and string transformation. Without much theory I am presenting the problem and solution. In given array of elements like [a1b2c3d4e5f6g7h8i9j1k2l3m4]. Convert it to [abcdefghijklm1234567891234]. The program should run inplace. What we need is an inplace transpose. Given below is code.

```
#include <stdio.h>
#include <iostream>
#include <bitset>
#define HASH_SIZE 128

using namespace std;

typedef char data_t;

void Print2DArray(char A[], int nr, int nc) {
   int size = nr*nc;
   for(int i = 0; i < size; i++)
      printf("%4c", *(A + i));

   printf("\n");
}

void MatrixTransposeInplaceArrangement(data_t A[], int r, int c) {
   int size = r*c - 1;
   data_t t; // holds element to be replaced, eventually becomes next element to move
   int next; // location of 't' to be moved
```

```
   int cycleBegin; // holds start of cycle
   int i; // iterator
   bitset<HASH_SIZE> b; // hash to mark moved elements

   b.reset();
   b[0] = b[size] = 1;
   i = 1; // Note that A[0] and A[size-1] won't move
   while( i < size ) {
      cycleBegin = i;
      t = A[i];
      do {
         // Input matrix [r x c]
         // Output matrix 1
         // i_new = (i*r)%size
         next = (i*r)%size;
         swap(A[next], t);
         b[i] = 1;
         i = next;
      } while( i != cycleBegin );

      // Get Next Move (what about querying random location?)
      for(i = 1; i < size && b[i]; i++)
         ;
      cout << endl;
   }
}

void Fill(data_t buf[], int size) {
   // Fill abcd ...
   for(int i = 0; i < size; i++)
   buf[i] = 'a'+i;

   // Fill 0123 ...
   buf += size;
   for(int i = 0; i < size; i++)
      buf[i] = '0'+i;
}

void TestCase_01(void) {
   int r = 2, c = 10;
   int size = r*c;
   data_t *A = new data_t[size];

   Fill(A, c);

   Print2DArray(A, r, c), cout << endl;
   MatrixTransposeInplaceArrangement(A, r, c);
   Print2DArray(A, c, r), cout << endl;

   delete[] A;
}

int main() {
   TestCase_01();
```

```
    return 0;
}
```

The post is incomplete without mentioning two links.

1. Aashish covered good theory behind cycle leader algorithm. See his post on string transformation.

2. As usual, Sambasiva demonstrated his exceptional skills in recursion to the problem. Ensure to understand his solution.

— Venki. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

http://www.geeksforgeeks.org/inplace-m-x-n-size-matrix-transpose/

Category: Arrays

Post navigation

← Space and time efficient Binomial Coefficient Longest Palindromic Substring | Set 2 →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 8

Print Matrix Diagonally

Given a 2D matrix, print all elements of the given matrix in diagonal order. For example, consider the following 5 X 4 input matrix.

```
 1   2   3   4
 5   6   7   8
 9  10  11  12
13  14  15  16
17  18  19  20
```

Diagonal printing of the above matrix is

```
 1
 5   2
 9   6   3
13  10   7   4
17  14  11   8
18  15  12
19  16
20
```

Following is C++ code for diagonal printing.

The diagonal printing of a given matrix 'matrix[ROW][COL]' always has 'ROW + COL – 1 lines in output

```
#include <stdio.h>
#include <stdlib.h>

#define ROW 5
#define COL 4

// A utility function to find min of two integers
int min(int a, int b)
{ return (a < b)? a: b; }
```

```
// A utility function to find min of three integers
int min(int a, int b, int c)
{ return min(min(a, b), c);}

// A utility function to find max of two integers
int max(int a, int b)
{ return (a > b)? a: b; }

// The main function that prints given matrix in diagonal order
void diagonalOrder(int matrix[][COL])
{
    // There will be ROW+COL-1 lines in the output
    for (int line=1; line<=(ROW + COL -1); line++)
    {
        /* Get column index of the first element in this line of output.
           The index is 0 for first ROW lines and line - ROW for remaining
           lines  */
        int start_col =  max(0, line-ROW);

        /* Get count of elements in this line. The count of elements is
           equal to minimum of line number, COL-start_col and ROW */
         int count = min(line, (COL-start_col), ROW);

        /* Print elements of this line */
        for (int j=0; j<count; j++)
            printf("%5d ", matrix[min(ROW, line)-j-1][start_col+j]);

        /* Ptint elements of next diagonal on next line */
        printf("\n");
    }
}

// Utility function to print a matrix
void printMatrix(int matrix[ROW][COL])
{
    for (int i=0; i< ROW; i++)
    {
        for (int j=0; j<COL; j++)
            printf("%5d ", matrix[i][j]);
        printf("\n");
    }
}

// Driver program to test above functions
int main()
{
    int M[ROW][COL] = {{1, 2, 3, 4},
                       {5, 6, 7, 8},
                       {9, 10, 11, 12},
                       {13, 14, 15, 16},
                       {17, 18, 19, 20},
                      };
    printf ("Given matrix is \n");
    printMatrix(M);
```

```
    printf ("\nDiagonal printing of matrix is \n");
    diagonalOrder(M);
    return 0;
}
```

Output:

```
 Given matrix is
    1     2     3     4
    5     6     7     8
    9    10    11    12
   13    14    15    16
   17    18    19    20

Diagonal printing of matrix is
    1
    5     2
    9     6     3
   13    10     7     4
   17    14    11     8
   18    15    12
   19    16
   20
```

Below is an **Alternate Method** to solve the above problem.

```
Matrix =>       1     2     3     4
                5     6     7     8
                9     10    11   12
                13    14    15   16
                17    18    19   20

Observe the sequence
          1 /  2 /  3 /  4
           / 5  /  6 /  7 /  8
               /  9 / 10 / 11 / 12
                   / 13 / 14 / 15 / 16
                       / 17 / 18 / 19 / 20
```

```
#include<bits/stdc++.h>
#define R 5
#define C 4
using namespace std;

bool isvalid(int i, int j)
{
    if (i < 0 || i >= R || j >= C || j < 0) return false;
    return true;
```

```
}

void diagonalOrder(int arr[][C])
{
    /* through this for loop we choose each element of first column
    as starting point and print diagonal starting at it.
    arr[0][0], arr[1][0]....arr[R-1][0] are all starting points */
    for (int k = 0; k < R; k++)
    {
        cout << arr[k][0] << " ";
        int i = k-1;    // set row index for next point in diagonal
        int j = 1;      //  set column index for next point in diagonal

        /* Print Diagonally upward */
        while (isvalid(i,j))
        {
            cout << arr[i][j] << " ";
            i--;
            j++;    // move in upright direction
        }
        cout << endl;
    }

    /* through this for loop we choose each element of last row
       as starting point (except the [0][c-1] it has already been
       processed in previous for loop) and print diagonal starting at it.
       arr[R-1][0], arr[R-1][1]....arr[R-1][c-1] are all starting points */

    //Note : we start from k = 1 to C-1;
    for (int k = 1; k < C; k++)
    {
        cout << arr[R-1][k] << " ";
        int i = R-2; // set row index for next point in diagonal
        int j = k+1; // set column index for next point in diagonal

        /* Print Diagonally upward */
        while (isvalid(i,j))
        {
            cout << arr[i][j] << " ";
            i--;
            j++; // move in upright direction
        }
        cout << endl;
    }
}

// Driver program to test above
int main()
{

    int arr[][C] = {{1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {13, 14, 15, 16},
```

```
        {17, 18, 19, 20},
    };
    diagonalOrder(arr);
    return 0;
}
```

Output:

```
 1
5 2
9 6 3
13 10 7 4
17 14 11 8
18 15 12
19 16
20
```

Thanks to Gaurav Ahirwar for suggesting this method.

This article is compiled by Ashish Anand and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

http://www.geeksforgeeks.org/print-matrix-diagonally/

Category: Arrays

Post navigation

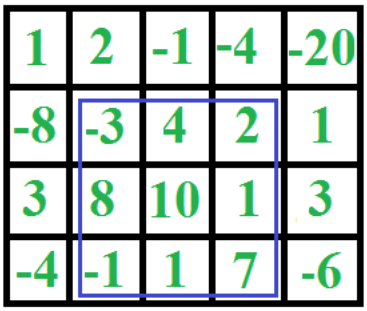
← Tug of War Divide and Conquer | Set 3 (Maximum Subarray Sum) →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 9

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 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 returns the
// maximum sum and stores starting and ending indexes of the maximum sum subarray
// 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
    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 row 'i'
            for (i = 0; i < ROW; ++i)
                temp[i] += M[i][right];

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

            // Compare sum with maximum sum so far. If sum is more, then update
            // 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 byAashish Barnwal. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

http://www.geeksforgeeks.org/dynamic-programming-set-27-max-sum-rectangle-in-a-2d-matrix/

Category: Arrays Tags: Dynamic Programming

Post navigation

← D E Shaw Interview | Set 1 [TopTalent.in] In Conversation With Nithin On What It Takes To Get Into Goldman Sachs →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 10

Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)

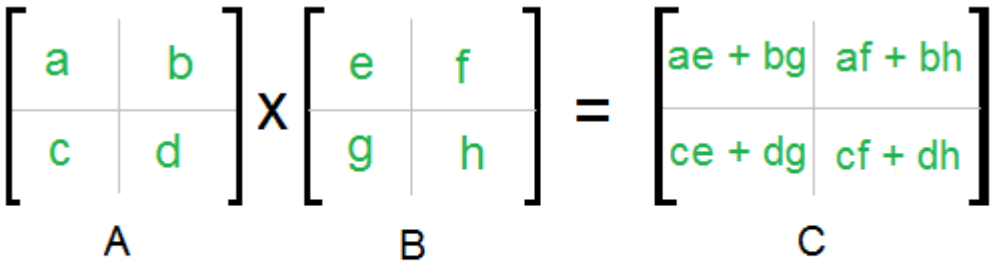
Given two square matrices A and B of size n x n each, find their multiplication matrix.

Naive Method
Following is a simple way to multiply two matrices.

```
void multiply(int A[][N], int B[][N], int C[][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            C[i][j] = 0;
            for (int k = 0; k < N; k++)
            {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}
```

Time Complexity of above method is $O(N^3)$.

Divide and Conquer
Following is simple Divide and Conquer method to multiply two square matrices.
1) Divide matrices A and B in 4 sub-matrices of size N/2 x N/2 as shown in the below diagram.
2) Calculate following values recursively. ae + bg, af + bh, ce + dg and cf + dh.



A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2

In the above method, we do 8 multiplications for matrices of size N/2 x N/2 and 4 additions. Addition of two matrices takes $O(N^2)$ time. So the time complexity can be written as

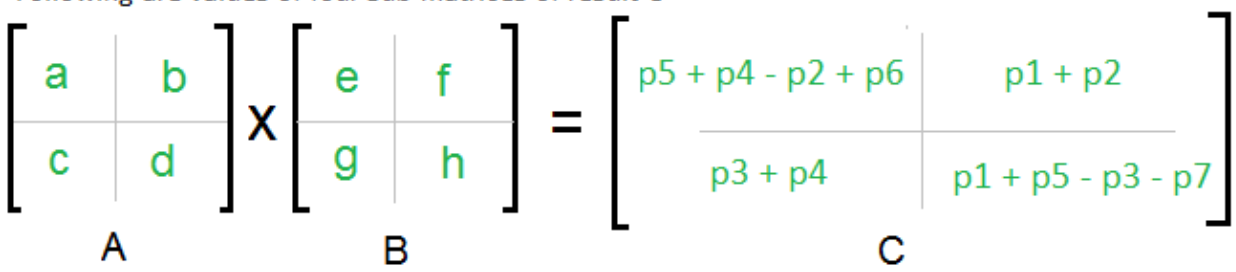
```
T(N) = 8T(N/2) + O(N2)

From Master's Theorem, time complexity of above method is O(N3)
which is unfortunately same as the above naive method.
```

Simple Divide and Conquer also leads to $O(N^3)$, can there be a better way?

In the above divide and conquer method, the main component for high time complexity is 8 recursive calls. The idea of **Strassen's method** is to reduce the number of recursive calls to 7. Strassen's method is similar to above simple divide and conquer method in the sense that this method also divide matrices to sub-matrices of size N/2 x N/2 as shown in the above diagram, but in Strassen's method, the four sub-matrices of result are calculated using following formulae.

p1 = a(f - h)
p2 = (a + b)h
p3 = (c + d)e
p4 = d(g - e)
p5 = (a + d)(e + h)
p6 = (b - d)(g + h)
p7 = (a - c)(e + f)

The A x B can be calculated using above seven multiplications.
Following are values of four sub-matrices of result C



A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2
p1, p2, p3, p4, p5, p6 and p7 are submatrices of size N/2 x N/2

Time Complexity of Strassen's Method
Addition and Subtraction of two matrices takes $O(N^2)$ time. So time complexity can be written as

```
T(N) = 7T(N/2) +  O(N2)

From Master's Theorem, time complexity of above method is
O(NLog7) which is approximately O(N2.8074)
```

Generally Strassen's Method is not preferred for practical applications for following reasons.
1) The constants used in Strassen's method are high and for a typical application Naive method works better.
2) For Sparse matrices, there are better methods especially designed for them.
3) The submatrices in recursion take extra space.
4) Because of the limited precision of computer arithmetic on noninteger values, larger errors accumulate in Strassen's algorithm than in Naive Method (Source: CLRS Book)

References:

Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest
https://www.youtube.com/watch?v=LOLebQ8nKHA
https://www.youtube.com/watch?v=QXY4RskLQcI

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

http://www.geeksforgeeks.org/strassens-matrix-multiplication/

Chapter 11

Create a matrix with alternating rectangles of O and X

Write a code which inputs two numbers m and n and creates a matrix of size m x n (m rows and n columns) in which every elements is either X or 0. The Xs and 0s must be filled alternatively, the matrix should have outermost rectangle of Xs, then a rectangle of 0s, then a rectangle of Xs, and so on.

Examples:

```
Input: m = 3, n = 3
Output: Following matrix
X X X
X 0 X
X X X

Input: m = 4, n = 5
Output: Following matrix
X X X X X
X 0 0 0 X
X 0 0 0 X
X X X X X

Input:  m = 5, n = 5
Output: Following matrix
X X X X X
X 0 0 0 X
X 0 X 0 X
X 0 0 0 X
X X X X X

Input:  m = 6, n = 7
Output: Following matrix
X X X X X X X
X 0 0 0 0 0 X
X 0 X X X 0 X
X 0 X X X 0 X
X 0 0 0 0 0 X
X X X X X X X
```

We strongly recommend to minimize the browser and try this yourself first.

This question was asked in campus recruitment of Shreepartners Gurgaon. I followed the following approach.

1) Use the code for Printing Matrix in Spiral form.
2) Instead of printing the array, inserted the element 'X' or '0' alternatively in the array.

Following is C implementation of the above approach.

```
#include <stdio.h>

// Function to print alternating rectangles of 0 and X
void fill0X(int m, int n)
{
    /*  k - starting row index
        m - ending row index
        l - starting column index
        n - ending column index
        i - iterator    */
    int i, k = 0, l = 0;

    // Store given number of rows and columns for later use
    int r = m, c = n;

    // A 2D array to store the output to be printed
    char a[m][n];
    char x = 'X'; // Iniitialize the character to be stoed in a[][]

    // Fill characters in a[][] in spiral form. Every iteration fills
    // one rectangle of either Xs or Os
    while (k < m && l < n)
    {
        /* Fill the first row from the remaining rows */
        for (i = l; i < n; ++i)
            a[k][i] = x;
        k++;

        /* Fill the last column from the remaining columns */
        for (i = k; i < m; ++i)
            a[i][n-1] = x;
        n--;

        /* Fill the last row from the remaining rows */
        if (k < m)
        {
            for (i = n-1; i >= l; --i)
                a[m-1][i] = x;
            m--;
        }

        /* Print the first column from the remaining columns */
        if (l < n)
        {
            for (i = m-1; i >= k; --i)
```

```
                a[i][l] = x;
            l++;
        }

        // Flip character for next iteration
        x = (x == '0')? 'X': '0';
    }

    // Print the filled matrix
    for (i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
            printf("%c ", a[i][j]);
        printf("\n");
    }
}

/* Driver program to test above functions */
int main()
{
    puts("Output for m = 5, n = 6");
    fill0X(5, 6);

    puts("\nOutput for m = 4, n = 4");
    fill0X(4, 4);

    puts("\nOutput for m = 3, n = 4");
    fill0X(3, 4);

    return 0;
}
```

Output:

```
 Output for m = 5, n = 6
X X X X X X
X 0 0 0 0 X
X 0 X X 0 X
X 0 0 0 0 X
X X X X X X

Output for m = 4, n = 4
X X X X
X 0 0 X
X 0 0 X
X X X X

Output for m = 3, n = 4
X X X X
X 0 0 X
X X X X
```

Time Complexity: O(mn)
Auxiliary Space: O(mn)

Please suggest if someone has a better solution which is more efficient in terms of space and time.

This article is contributed by **Deepak Bisht**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

http://www.geeksforgeeks.org/create-a-matrix-with-alternating-rectangles-of-0-and-x/

Category: Arrays

Chapter 12

Find the row with maximum number of 1s

Given a boolean 2D array, where each row is sorted. Find the row with the maximum number of 1s.

```
Example
Input matrix
0 1 1 1
0 0 1 1
1 1 1 1  // this row has maximum 1s
0 0 0 0

Output: 2
```

A simple method is to do a row wise traversal of the matrix, count the number of 1s in each row and compare the count with max. Finally, return the index of row with maximum 1s. The time complexity of this method is O(m*n) where m is number of rows and n is number of columns in matrix.

We can do better. Since each row is sorted, we can **use Binary Search** to count of 1s in each row. We find the index of first instance of 1 in each row. The count of 1s will be equal to total number of columns minus the index of first 1.

See the following code for implementation of the above approach.

```
#include <stdio.h>
#define R 4
#define C 4

/* A function to find the index of first index of 1 in a boolean array arr[] */
int first(bool arr[], int low, int high)
{
  if(high >= low)
  {
    // get the middle index
    int mid = low + (high - low)/2;
```

```
    // check if the element at middle index is first 1
    if ( ( mid == 0 || arr[mid-1] == 0) && arr[mid] == 1)
      return mid;

    // if the element is 0, recur for right side
    else if (arr[mid] == 0)
      return first(arr, (mid + 1), high);

    else // If element is not first 1, recur for left side
      return first(arr, low, (mid -1));
  }
  return -1;
}

// The main function that returns index of row with maximum number of 1s.
int rowWithMax1s(bool mat[R][C])
{
    int max_row_index = 0, max = -1; // Initialize max values

    // Traverse for each row and count number of 1s by finding the index
    // of first 1
    int i, index;
    for (i = 0; i < R; i++)
    {
       index = first (mat[i], 0, C-1);
       if (index != -1 && C-index > max)
       {
           max = C - index;
           max_row_index = i;
       }
    }

    return max_row_index;
}

/* Driver program to test above functions */
int main()
{
    bool mat[R][C] = { {0, 0, 0, 1},
        {0, 1, 1, 1},
        {1, 1, 1, 1},
        {0, 0, 0, 0}
    };

    printf("Index of row with maximum 1s is %d \n", rowWithMax1s(mat));

    return 0;
}
```

Output:

```
 Index of row with maximum 1s is 2
```

Time Complexity: O(mLogn) where m is number of rows and n is number of columns in matrix.

The above solution **can be optimized further**. Instead of doing binary search in every row, we first check whether the row has more 1s than max so far. If the row has more 1s, then only count 1s in the row. Also, to count 1s in a row, we don't do binary search in complete row, we do search in before the index of last max.

Following is an optimized version of the above solution.

```
// The main function that returns index of row with maximum number of 1s.
int rowWithMax1s(bool mat[R][C])
{
    int i, index;

    // Initialize max using values from first row.
    int max_row_index = 0;
    int max = C - first(mat[0], 0, C-1);

    // Traverse for each row and count number of 1s by finding the index
    // of first 1
    for (i = 1; i < R; i++)
    {
        // Count 1s in this row only if this row has more 1s than
        // max so far
        if (mat[i][C-max-1] == 1)
        {
            // Note the optimization here also
            index = first (mat[i], 0, C-max);

            if (index != -1 && C-index > max)
            {
                max = C - index;
                max_row_index = i;
            }
        }
    }
    return max_row_index;
}
```

The worst case time complexity of the above optimized version is also O(mLogn), the will solution work better on average. Thanks to Naveen Kumar Singh for suggesting the above solution.

Sources: this and this

The worst case of the above solution occurs for a matrix like following.
0 0 0 … 0 1
0 0 0 ..0 1 1
0 … 0 1 1 1
….0 1 1 1 1

Following method works in O(m+n) time complexity in worst case.

Step1: Get the index of first (or leftmost) 1 in the first row.

Step2: Do following for every row after the first row
…IF the element on left of previous leftmost 1 is 0, ignore this row.

…ELSE Move left until a 0 is found. Update the leftmost index to this index and max_row_index to be the current row.

The time complexity is O(m+n) because we can possibly go as far left as we came ahead in the first step.

Following is C++ implementation of this method.

```
// The main function that returns index of row with maximum number of 1s.
int rowWithMax1s(bool mat[R][C])
{
    // Initialize first row as row with max 1s
    int max_row_index = 0;

    // The function first() returns index of first 1 in row 0.
    // Use this index to initialize the index of leftmost 1 seen so far
    int j = first(mat[0], 0, C-1) - 1;
    if (j == -1) // if 1 is not present in first row
      j = C - 1;

    for (int i = 1; i < R; i++)
    {
        // Move left until a 0 is found
        while (j >= 0 && mat[i][j] == 1)
        {
           j = j-1;  // Update the index of leftmost 1 seen so far
           max_row_index = i;  // Update max_row_index
        }
    }
    return max_row_index;
}
```

Thanks to Tylor, Ankan and Palash for their inputs.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

http://www.geeksforgeeks.org/find-the-row-with-maximum-number-1s/

Chapter 13

Print all elements in sorted order from row and column wise sorted matrix

Given an n x n matrix, where every row and column is sorted in non-decreasing order. Print all elements of matrix in sorted order.

Example:

```
Input: mat[][]  =  { {10, 20, 30, 40},
                     {15, 25, 35, 45},
                     {27, 29, 37, 48},
                     {32, 33, 39, 50},
                   };

Output:
Elements of matrix in sorted order
10 15 20 25 27 29 30 32 33 35 37 39 40 45 48 50
```

We strongly recommend to minimize the browser and try this yourself first.

We can use **Young Tableau** to solve the above problem. The idea is to consider given 2D array as Young Tableau and call extract minimum O(N)

```
// A C++ program to Print all elements in sorted order from row and
// column wise sorted matrix
#include<iostream>
#include<climits>
using namespace std;

#define INF INT_MAX
#define N 4

// A utility function to youngify a Young Tableau.  This is different
```

```
// from standard youngify.  It assumes that the value at mat[0][0] is
// infinite.
void youngify(int mat[][N], int i, int j)
{
    // Find the values at down and right sides of mat[i][j]
    int downVal  = (i+1 < N)? mat[i+1][j]: INF;
    int rightVal = (j+1 < N)? mat[i][j+1]: INF;

    // If mat[i][j] is the down right corner element, return
    if (downVal==INF && rightVal==INF)
        return;

    // Move the smaller of two values (downVal and rightVal) to
    // mat[i][j] and recur for smaller value
    if (downVal < rightVal)
    {
        mat[i][j] = downVal;
        mat[i+1][j] = INF;
        youngify(mat, i+1, j);
    }
    else
    {
        mat[i][j] = rightVal;
        mat[i][j+1] = INF;
        youngify(mat, i, j+1);
    }
}

// A utility function to extract minimum element from Young tableau
int extractMin(int mat[][N])
{
    int ret = mat[0][0];
    mat[0][0] = INF;
    youngify(mat, 0, 0);
    return ret;
}

// This function uses extractMin() to print elements in sorted order
void printSorted(int mat[][N])
{
   cout << "Elements of matrix in sorted order \n";
   for (int i=0; i<N*N; i++)
     cout << extractMin(mat) << " ";
}

// driver program to test above function
int main()
{
  int mat[N][N] = { {10, 20, 30, 40},
                    {15, 25, 35, 45},
                    {27, 29, 37, 48},
                    {32, 33, 39, 50},
                  };
  printSorted(mat);
```

```
  return 0;
}
```

Output:

```
Elements of matrix in sorted order
10 15 20 25 27 29 30 32 33 35 37 39 40 45 48 50
```

Time complexity of extract minimum is O(N) and it is called $O(N^2)$ times. Therefore the overall time complexity is $O(N^3)$.

A **better solution** is to use the approach used for merging k sorted arrays. The idea is to use a Min Heap of size N which stores elements of first column. The do extract minimum. In extract minimum, replace the minimum element with the next element of the row from which the element is extracted. Time complexity of this solution is $O(N^2LogN)$.

```
// C++ program to merge k sorted arrays of size n each.
#include<iostream>
#include<climits>
using namespace std;

#define N 4

// A min heap node
struct MinHeapNode
{
    int element; // The element to be stored
    int i; // index of the row from which the element is taken
    int j; // index of the next element to be picked from row
};

// Prototype of a utility function to swap two min heap nodes
void swap(MinHeapNode *x, MinHeapNode *y);

// A class for Min Heap
class MinHeap
{
    MinHeapNode *harr; // pointer to array of elements in heap
    int heap_size; // size of min heap
public:
    // Constructor: creates a min heap of given size
    MinHeap(MinHeapNode a[], int size);

    // to heapify a subtree with root at given index
    void MinHeapify(int );

    // to get index of left child of node at index i
    int left(int i) { return (2*i + 1); }

    // to get index of right child of node at index i
    int right(int i) { return (2*i + 2); }
```

```
    // to get the root
    MinHeapNode getMin() { return harr[0]; }

    // to replace root with new node x and heapify() new root
    void replaceMin(MinHeapNode x) { harr[0] = x;  MinHeapify(0); }
};

// This function prints elements of a given matrix in non-decreasing
//  order. It assumes that ma[][] is sorted row wise sorted.
void printSorted(int mat[][N])
{
    // Create a min heap with k heap nodes.  Every heap node
    // has first element of an array
    MinHeapNode *harr = new MinHeapNode[N];
    for (int i = 0; i < N; i++)
    {
        harr[i].element = mat[i][0]; // Store the first element
        harr[i].i = i;  // index of row
        harr[i].j = 1;  // Index of next element to be stored from row
    }
    MinHeap hp(harr, N); // Create the min heap

    // Now one by one get the minimum element from min
    // heap and replace it with next element of its array
    for (int count = 0; count < N*N; count++)
    {
        // Get the minimum element and store it in output
        MinHeapNode root = hp.getMin();

        cout << root.element << " ";

        // Find the next elelement that will replace current
        // root of heap. The next element belongs to same
        // array as the current root.
        if (root.j < N)
        {
            root.element = mat[root.i][root.j];
            root.j += 1;
        }
        // If root was the last element of its array
        else root.element =  INT_MAX; //INT_MAX is for infinite

        // Replace root with next element of array
        hp.replaceMin(root);
    }
}

// FOLLOWING ARE IMPLEMENTATIONS OF STANDARD MIN HEAP METHODS
// FROM CORMEN BOOK
// Constructor: Builds a heap from a given array a[] of given size
MinHeap::MinHeap(MinHeapNode a[], int size)
{
    heap_size = size;
```

```
    harr = a;  // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
}

// A recursive method to heapify a subtree with root at given index
// This method assumes that the subtrees are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size && harr[l].element < harr[i].element)
        smallest = l;
    if (r < heap_size && harr[r].element < harr[smallest].element)
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
    }
}

// A utility function to swap two elements
void swap(MinHeapNode *x, MinHeapNode *y)
{
    MinHeapNode temp = *x;  *x = *y;  *y = temp;
}

// driver program to test above function
int main()
{
  int mat[N][N] = { {10, 20, 30, 40},
                    {15, 25, 35, 45},
                    {27, 29, 37, 48},
                    {32, 33, 39, 50},
                  };
  printSorted(mat);
  return 0;
}
```

Output:

```
 10 15 20 25 27 29 30 32 33 35 37 39 40 45 48 50
```

Exercise:
Above solutions work for a square matrix. Extend the above solutions to work for an M*N rectangular matrix.

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

Source

http://www.geeksforgeeks.org/print-elements-sorted-order-row-column-wise-sorted-matrix/

Category: Arrays Tags: Matrix

Post navigation

← Amazon interview Experience | Set 138 (For SDE 1) Serialize and Deserialize a Binary Tree →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 14

Given an n x n square matrix, find sum of all sub-squares of size k x k

Given an n x n square matrix, find sum of all sub-squares of size k x k where k is smaller than or equal to n.

Examples

```
Input:
n = 5, k = 3
arr[][] = { {1, 1, 1, 1, 1},
            {2, 2, 2, 2, 2},
            {3, 3, 3, 3, 3},
            {4, 4, 4, 4, 4},
            {5, 5, 5, 5, 5},
          };
Output:
       18  18  18
       27  27  27
       36  36  36
```

```
Input:
n = 3, k = 2
arr[][] = { {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9},
          };
Output:
       12  16
       24  28
```

A **Simple Solution** is to one by one pick starting point (leftmost-topmost corner) of all possible sub-squares. Once the starting point is picked, calculate sum of sub-square starting with the picked starting point.

Following is C++ implementation of this idea.

```
// A simple C++ program to find sum of all subsquares of size k x k
#include <iostream>
using namespace std;

// Size of given matrix
#define n 5

// A simple function to find sum of all sub-squares of size k x k
// in a given square matrix of size n x n
void printSumSimple(int mat[][n], int k)
{
   // k must be smaller than or equal to n
   if (k > n) return;

   // row number of first cell in current sub-square of size k x k
   for (int i=0; i<n-k+1; i++)
   {
      // column of first cell in current sub-square of size k x k
      for (int j=0; j<n-k+1; j++)
      {
          // Calculate and print sum of current sub-square
          int sum = 0;
          for (int p=i; p<k+i; p++)
             for (int q=j; q<k+j; q++)
                 sum += mat[p][q];
           cout << sum << "  ";
      }

      // Line separator for sub-squares starting with next row
      cout << endl;
   }
}

// Driver program to test above function
int main()
{
    int mat[n][n] = {{1, 1, 1, 1, 1},
                     {2, 2, 2, 2, 2},
                     {3, 3, 3, 3, 3},
                     {4, 4, 4, 4, 4},
                     {5, 5, 5, 5, 5},
                    };
    int k = 3;
    printSumSimple(mat, k);
    return 0;
}
```

Output:

```
  18  18  18
  27  27  27
  36  36  36
```

Time complexity of above solution is $O(k^2n^2)$. We can solve this problem in $O(n^2)$ time using a **Tricky Solution**. The idea is to preprocess the given square matrix. In the preprocessing step, calculate sum of all vertical strips of size k x 1 in a temporary square matrix stripSum[][]. Once we have sum of all vertical strips, we can calculate sum of first sub-square in a row as sum of first k strips in that row, and for remaining sub-squares, we can calculate sum in O(1) time by removing the leftmost strip of previous subsquare and adding the rightmost strip of new square.

Following is C++ implementation of this idea.

```
// An efficient C++ program to find sum of all subsquares of size k x k
#include <iostream>
using namespace std;

// Size of given matrix
#define n 5

// A O(n^2) function to find sum of all sub-squares of size k x k
// in a given square matrix of size n x n
void printSumTricky(int mat[][n], int k)
{
   // k must be smaller than or equal to n
   if (k > n) return;

   // 1: PREPROCESSING
   // To store sums of all strips of size k x 1
   int stripSum[n][n];

   // Go column by column
   for (int j=0; j<n; j++)
   {
       // Calculate sum of first k x 1 rectangle in this column
       int sum = 0;
       for (int i=0; i<k; i++)
          sum += mat[i][j];
       stripSum[0][j] = sum;

       // Calculate sum of remaining rectangles
       for (int i=1; i<n-k+1; i++)
       {
            sum += (mat[i+k-1][j] - mat[i-1][j]);
            stripSum[i][j] = sum;
       }
   }

   // 2: CALCULATE SUM of Sub-Squares using stripSum[][]
   for (int i=0; i<n-k+1; i++)
   {
      // Calculate and print sum of first subsquare in this row
      int sum = 0;
      for (int j = 0; j<k; j++)
           sum += stripSum[i][j];
      cout << sum << "  ";

      // Calculate sum of remaining squares in current row by
```

```
      // removing the leftmost strip of previous sub-square and
      // adding a new strip
      for (int j=1; j<n-k+1; j++)
      {
         sum += (stripSum[i][j+k-1] - stripSum[i][j-1]);
         cout << sum << "  ";
      }

      cout << endl;
   }
}

// Driver program to test above function
int main()
{
    int mat[n][n] = {{1, 1, 1, 1, 1},
                     {2, 2, 2, 2, 2},
                     {3, 3, 3, 3, 3},
                     {4, 4, 4, 4, 4},
                     {5, 5, 5, 5, 5},
                    };
    int k = 3;
    printSumTricky(mat, k);
    return 0;
}
```

Output:

```
  18  18  18
  27  27  27
  36  36  36
```

This article is contributed by **Rahul Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

http://www.geeksforgeeks.org/given-n-x-n-square-matrix-find-sum-sub-squares-size-k-x-k/

Category: Arrays Tags: Matrix

Post navigation

← SAP Labs Interview Experience for Developer Associate Analysis of Algorithm | Set 5 (Amortized Analysis Introduction) →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 15

Count number of islands where every island is row-wise and column-wise separated

Given a rectangular matrix which has only two possible values 'X' and 'O'. The values 'X' always appear in form of rectangular islands and these islands are always row-wise and column-wise separated by at least one line of 'O's. Note that islands can only be diagonally adjacent. Count the number of islands in the given matrix.

Examples:

```
mat[M][N] =  {{'O', 'O', 'O'},
              {'X', 'X', 'O'},
              {'X', 'X', 'O'},
              {'O', 'O', 'X'},
              {'O', 'O', 'X'},
              {'X', 'X', 'O'}
             };
Output: Number of islands is 3

mat[M][N] =  {{'X', 'O', 'O', 'O', 'O', 'O'},
              {'X', 'O', 'X', 'X', 'X', 'X'},
              {'O', 'O', 'O', 'O', 'O', 'O'},
              {'X', 'X', 'X', 'O', 'X', 'X'},
              {'X', 'X', 'X', 'O', 'X', 'X'},
              {'O', 'O', 'O', 'O', 'X', 'X'},
             };
Output: Number of islands is 4
```

We strongly recommend to minimize your browser and try this yourself first.

The idea is to count all top-leftmost corners of given matrix. We can check if a 'X' is top left or not by checking following conditions.
1) A 'X' is top of rectangle if the cell just above it is a 'O'
2) A 'X' is leftmost of rectangle if the cell just left of it is a 'O'

Note that we must check for both conditions as there may be more than one top cells and more than one leftmost cells in a rectangular island. Below is C++ implementation of above idea.

```
// A C++ program to count the number of rectangular
// islands where every island is separated by a line
#include<iostream>
using namespace std;

// Size of given matrix is M X N
#define M 6
#define N 3

// This function takes a matrix of 'X' and 'O'
// and returns the number of rectangular islands
// of 'X' where no two islands are row-wise or
// column-wise adjacent, the islands may be diagonaly
// adjacent
int countIslands(int mat[][N])
{
    int count = 0; // Initialize result

    // Traverse the input matrix
    for (int i=0; i<M; i++)
    {
        for (int j=0; j<N; j++)
        {
            // If current cell is 'X', then check
            // whether this is top-leftmost of a
            // rectangle. If yes, then increment count
            if (mat[i][j] == 'X')
            {
                if ((i == 0 || mat[i-1][j] == 'O') &&
                    (j == 0 || mat[i][j-1] == 'O'))
                    count++;
            }
        }
    }

    return count;
}

// Driver program to test above function
int main()
{
    int mat[M][N] =  {{'O', 'O', 'O'},
                      {'X', 'X', 'O'},
                      {'X', 'X', 'O'},
                      {'O', 'O', 'X'},
                      {'O', 'O', 'X'},
                      {'X', 'X', 'O'}
                     };
    cout << "Number of rectangular islands is "
         << countIslands(mat);
```

```
    return 0;
}
```

Output:

```
Number of rectangular islands is 3
```

Time complexity of this solution is O(MN).

This article is contributed by **Udit Gupta**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Source

http://www.geeksforgeeks.org/count-number-islands-every-island-separated-line/

Category: Arrays

Post navigation

← Amazon Interview Experience | Set 178 (For SDE-1) Bit Fields in C →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 16

Find a common element in all rows of a given row-wise sorted matrix

Given a matrix where every row is sorted in increasing order. Write a function that finds and returns a common element in all rows. If there is no common element, then returns -1.

Example:

```
 Input: mat[4][5] = { {1, 2, 3, 4, 5},
                      {2, 4, 5, 8, 10},
                      {3, 5, 7, 9, 11},
                      {1, 3, 5, 7, 9},
                    };
Output: 5
```

A **O(m*n*n) simple solution** is to take every element of first row and search it in all other rows, till we find a common element. Time complexity of this solution is O(m*n*n) where m is number of rows and n is number of columns in given matrix. This can be improved to O(m*n*Logn) if we use Binary Search instead of linear search.

We can solve this problem **in O(mn) time** using the approach similar to merge of Merge Sort. The idea is to start from the last column of every row. If elements at all last columns are same, then we found the common element. Otherwise we find the minimum of all last columns. Once we find a minimum element, we know that all other elements in last columns cannot be a common element, so we reduce last column index for all rows except for the row which has minimum value. We keep repeating these steps till either all elements at current last column don't become same, or a last column index reaches 0.

Below is C implementation of above idea.

```
// A C program to find a common element in all rows of a
// row wise sorted array
#include<stdio.h>

// Specify number of rows and columns
#define M 4
#define N 5

// Returns common element in all rows of mat[M][N]. If there is no
```

```
// common element, then -1 is returned
int findCommon(int mat[M][N])
{
    // An array to store indexes of current last column
    int column[M];
    int min_row;  // To store index of row whose current
                  // last element is minimum

    // Initialize current last element of all rows
    int i;
    for (i=0; i<M; i++)
        column[i] = N-1;

    min_row = 0; // Initialize min_row as first row

    // Keep finding min_row in current last column, till either
    // all elements of last column become same or we hit first column.
    while (column[min_row] >= 0)
    {
       // Find minimum in current last column
       for (i=0; i<M; i++)
       {
          if (mat[i][column[i]] < mat[min_row][column[min_row]] )
             min_row = i;
       }

       // eq_count is count of elements equal to minimum in current last
       // column.
       int eq_count = 0;

       // Travers current last column elements again to update it
       for (i=0; i<M; i++)
       {
          //  Decrease last column index of a row whose value is more
          // than minimum.
          if (mat[i][column[i]] > mat[min_row][column[min_row]])
          {
             if (column[i] == 0)
                return -1;

             column[i] -= 1; // Reduce last column index by 1
          }
          else
             eq_count++;
       }

       // If equal count becomes M, return the value
       if (eq_count == M)
          return mat[min_row][column[min_row]];
    }
    return -1;
}

// driver program to test above function
```

```
int main()
{
  int mat[M][N] = { {1, 2, 3, 4, 5},
                    {2, 4, 5, 8, 10},
                    {3, 5, 7, 9, 11},
                    {1, 3, 5, 7, 9},
                  };
  int result = findCommon(mat);
  if (result == -1)
    printf("No common element");
  else
    printf("Common element is %d", result);
  return 0;
}
```

Output:

```
Common element is 5
```

Explanation for working of above code
Let us understand working of above code for following example.

Initially entries in last column array are N-1, i.e., {4, 4, 4, 4}
{1, 2, 3, 4, **5**},
{2, 4, 5, 8, **10**},
{3, 5, 7, 9, **11**},
{1, 3, 5, 7, **9**},

The value of min_row is 0, so values of last column index for rows with value greater than 5 is reduced by one. So column[] becomes {4, 3, 3, 3}.
{1, 2, 3, 4, **5**},
{2, 4, 5, **8**, 10},
{3, 5, 7, **9**, 11},
{1, 3, 5, **7**, 9},

The value of min_row remains 0 and and value of last column index for rows with value greater than 5 is reduced by one. So column[] becomes {4, 2, 2, 2}.
{1, 2, 3, 4, **5**},
{2, 4, **5**, 8, 10},
{3, 5, **7**, 9, 11},
{1, 3, **5**, 7, 9},

The value of min_row remains 0 and value of last column index for rows with value greater than 5 is reduced by one. So colomun[] becomes {4, 2, 1, 2}.
{1, 2, 3, 4, **5**},
{2, 4, **5**, 8, 10},
{3, **5**, 7, 9, 11},
{1, 3, **5**, 7, 9},

Now all values in current last columns of all rows is same, so 5 is returned.

A Hashing Based Solution
We can also use hashing. This solution works even if the rows are not sorted. It can be used to print all common elements.

```
Step1:  Create a Hash Table with all key as distinct elements
        of row1. Value for all these will be 0.

Step2:
For i = 1 to M-1
 For j = 0 to N-1
  If (mat[i][j] is already present in Hash Table)
   If (And this is not a repetition in current row.
      This can be checked by comparing HashTable value with
      row number)
         Update the value of this key in HashTable with current
         row number

Step3: Iterate over HashTable and print all those keys for
       which value = M
```

Time complexity of the above hashing based solution is O(MN) under the assumption that search and insert in HashTable take O(1) time. Thanks to Nishant for suggesting this solution in a comment below.

Exercise: Given n sorted arrays of size m each, find all common elements in all arrays in O(mn) time.

This article is contributed by **Anand Agrawal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

http://www.geeksforgeeks.org/find-common-element-rows-row-wise-sorted-matrix/

Chapter 17

Given a matrix of ‘O’ and ‘X’, replace ‘O’ with ‘X’ if surrounded by ‘X’

Given a matrix where every element is either ‘O’ or ‘X’, replace ‘O’ with ‘X’ if surrounded by ‘X’. A ‘O’ (or a set of ‘O’) is considered to be by surrounded by ‘X’ if there are ‘X’ at locations just below, just above, just left and just right of it.

Examples:

```
 Input: mat[M][N] =  {{'X', 'O', 'X', 'X', 'X', 'X'},
                     {'X', 'O', 'X', 'X', 'O', 'X'},
                     {'X', 'X', 'X', 'O', 'O', 'X'},
                     {'O', 'X', 'X', 'X', 'X', 'X'},
                     {'X', 'X', 'X', 'O', 'X', 'O'},
                     {'O', 'O', 'X', 'O', 'O', 'O'},
                    };
Output: mat[M][N] =  {{'X', 'O', 'X', 'X', 'X', 'X'},
                      {'X', 'O', 'X', 'X', 'X', 'X'},
                      {'X', 'X', 'X', 'X', 'X', 'X'},
                      {'O', 'X', 'X', 'X', 'X', 'X'},
                      {'X', 'X', 'X', 'O', 'X', 'O'},
                      {'O', 'O', 'X', 'O', 'O', 'O'},
                    };



 Input: mat[M][N] =  {{'X', 'X', 'X', 'X'}
                     {'X', 'O', 'X', 'X'}
                     {'X', 'O', 'O', 'X'}
                     {'X', 'O', 'X', 'X'}
                     {'X', 'X', 'O', 'O'}
                    };

 Input: mat[M][N] =  {{'X', 'X', 'X', 'X'}
                     {'X', 'X', 'X', 'X'}
                     {'X', 'X', 'X', 'X'}
                     {'X', 'X', 'X', 'X'}
                     {'X', 'X', 'O', 'O'}
                    };
```

We strongly recommend to minimize your browser and try this yourself first.

This is mainly an application of Flood-Fill algorithm. The main difference here is that a 'O' is not replaced by 'X' if it lies in region that ends on a boundary. Following are simple steps to do this special flood fill.

1) Traverse the given matrix and replace all 'O' with a special character '-'.

2) Traverse four edges of given matrix and call floodFill('-', 'O') for every '-' on edges. The remaining '-' are the characters that indicate 'O's (in the original matrix) to be replaced by 'X'.

3) Traverse the matrix and replace all '-'s with 'X's.

Let us see steps of above algorithm with an example. Let following be the input matrix.

```
mat[M][N] =  {{'X', 'O', 'X', 'X', 'X', 'X'},
              {'X', 'O', 'X', 'X', 'O', 'X'},
              {'X', 'X', 'X', 'O', 'O', 'X'},
              {'O', 'X', 'X', 'X', 'X', 'X'},
              {'X', 'X', 'X', 'O', 'X', 'O'},
              {'O', 'O', 'X', 'O', 'O', 'O'},
             };
```

Step 1: Replace all 'O' with '-'.

```
mat[M][N] =  {{'X', '-', 'X', 'X', 'X', 'X'},
              {'X', '-', 'X', 'X', '-', 'X'},
              {'X', 'X', 'X', '-', '-', 'X'},
              {'-', 'X', 'X', 'X', 'X', 'X'},
              {'X', 'X', 'X', '-', 'X', '-'},
              {'-', '-', 'X', '-', '-', '-'},
             };
```

Step 2: Call floodFill('-', 'O') for all edge elements with value equals to '-'

```
mat[M][N] =  {{'X', 'O', 'X', 'X', 'X', 'X'},
              {'X', 'O', 'X', 'X', '-', 'X'},
              {'X', 'X', 'X', '-', '-', 'X'},
              {'O', 'X', 'X', 'X', 'X', 'X'},
              {'X', 'X', 'X', 'O', 'X', 'O'},
              {'O', 'O', 'X', 'O', 'O', 'O'},
             };
```

Step 3: Replace all '-' with 'X'.

```
mat[M][N] =  {{'X', 'O', 'X', 'X', 'X', 'X'},
              {'X', 'O', 'X', 'X', 'X', 'X'},
              {'X', 'X', 'X', 'X', 'X', 'X'},
              {'O', 'X', 'X', 'X', 'X', 'X'},
              {'X', 'X', 'X', 'O', 'X', 'O'},
              {'O', 'O', 'X', 'O', 'O', 'O'},
             };
```

The following is C++ implementation of above algorithm.

```
// A C++ program to replace all 'O's with 'X''s if surrounded by 'X'
#include<iostream>
using namespace std;

// Size of given matrix is M X N
#define M 6
#define N 6


// A recursive function to replace previous value 'prevV' at  '(x, y)'
// and all surrounding values of (x, y) with new value 'newV'.
void floodFillUtil(char mat[][N], int x, int y, char prevV, char newV)
{
    // Base cases
    if (x < 0 || x >= M || y < 0 || y >= N)
        return;
    if (mat[x][y] != prevV)
        return;

    // Replace the color at (x, y)
    mat[x][y] = newV;

    // Recur for north, east, south and west
    floodFillUtil(mat, x+1, y, prevV, newV);
    floodFillUtil(mat, x-1, y, prevV, newV);
    floodFillUtil(mat, x, y+1, prevV, newV);
    floodFillUtil(mat, x, y-1, prevV, newV);
}

// Returns size of maximum size subsquare matrix
// surrounded by 'X'
int replaceSurrounded(char mat[][N])
{
   // Step 1: Replace all 'O'  with '-'
   for (int i=0; i<M; i++)
      for (int j=0; j<N; j++)
          if (mat[i][j] == 'O')
             mat[i][j] = '-';

   // Call floodFill for all '-' lying on edges
   for (int i=0; i<M; i++)   // Left side
      if (mat[i][0] == '-')
        floodFillUtil(mat, i, 0, '-', 'O');
   for (int i=0; i<M; i++)  //  Right side
      if (mat[i][N-1] == '-')
        floodFillUtil(mat, i, N-1, '-', 'O');
   for (int i=0; i<N; i++)   // Top side
      if (mat[0][i] == '-')
        floodFillUtil(mat, 0, i, '-', 'O');
   for (int i=0; i<N; i++)  // Bottom side
      if (mat[M-1][i] == '-')
```

```
        floodFillUtil(mat, M-1, i, '-', 'O');

   // Step 3: Replace all '-' with 'X'
   for (int i=0; i<M; i++)
      for (int j=0; j<N; j++)
         if (mat[i][j] == '-')
             mat[i][j] = 'X';

}

// Driver program to test above function
int main()
{
    char mat[][N] =  {{'X', 'O', 'X', 'O', 'X', 'X'},
                      {'X', 'O', 'X', 'X', 'O', 'X'},
                      {'X', 'X', 'X', 'O', 'X', 'X'},
                      {'O', 'X', 'X', 'X', 'X', 'X'},
                      {'X', 'X', 'X', 'O', 'X', 'O'},
                      {'O', 'O', 'X', 'O', 'O', 'O'},
                     };
    replaceSurrounded(mat);


    for (int i=0; i<M; i++)
    {
      for (int j=0; j<N; j++)
          cout << mat[i][j] << " ";
      cout << endl;
    }
    return 0;
}
```

Output:

```
 X O X O X X
X O X X X X
X X X X X X
O X X X X X
X X X O X O
O O X O O O
```

Time Complexity of the above solution is O(MN). Note that every element of matrix is processed at most three times.

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

Source

http://www.geeksforgeeks.org/given-matrix-o-x-replace-o-x-surrounded-x/

Category: Arrays

Post navigation

← One Convergence Device Interview Experience | Set 1 (On-Campus) Amazon Interview Experience | Set 164 (For SDE I) →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 18

Find the longest path in a matrix with given constraints

Given a n*n matrix where numbers all numbers are distinct and are distributed from range 1 to n^2, find the maximum length path (starting from any cell) such that all cells along the path are increasing order with a difference of 1.

We can move in 4 directions from a given cell (i, j), i.e., we can move to (i+1, j) or (i, j+1) or (i-1, j) or (i, j-1) with the condition that the adjacen

Example:

```
Input:  mat[][] = {{1, 2, 9}
                   {5, 3, 8}
                   {4, 6, 7}}
Output: 4
The longest path is 6-7-8-9.
```

We strongly recommend you to minimize your browser and try this yourself first.
The idea is simple, we calculate longest path beginning with every cell. Once we have computed longest for all cells, we return maximum of all longest paths. One important observation in this approach is many overlapping subproblems. Therefore this problem can be optimally solved using Dynamic Programming.

Below is Dynamic Programming based C implementation that uses a lookup table dp[][] to check if a problem is already solved or not.

```
#include<bits/stdc++.h>
#define n 3
using namespace std;

// Returns length of the longest path beginning with mat[i][j].
// This function mainly uses lookup table dp[n][n]
int findLongestFromACell(int i, int j, int mat[n][n], int dp[n][n])
{
    // Base case
    if (i<0 || i>=n || j<0 || j>=n)
        return 0;
```

```
    // If this subproblem is already solved
    if (dp[i][j] != -1)
        return dp[i][j];

    // Since all numbers are unique and in range from 1 to n*n,
    // there is atmost one possible direction from any cell
    if ((mat[i][j] +1) == mat[i][j+1])
       return dp[i][j] = 1 + findLongestFromACell(i,j+1,mat,dp);

    if (mat[i][j] +1 == mat[i][j-1])
       return dp[i][j] = 1 + findLongestFromACell(i,j-1,mat,dp);

    if (mat[i][j] +1 == mat[i-1][j])
       return dp[i][j] = 1 + findLongestFromACell(i-1,j,mat,dp);

    if (mat[i][j] +1 == mat[i+1][j])
       return dp[i][j] = 1 + findLongestFromACell(i+1,j,mat,dp);

    // If none of the adjacent fours is one greater
    return dp[i][j] = 1;
}

// Returns length of the longest path beginning with any cell
int finLongestOverAll(int mat[n][n])
{
    int result = 1;  // Initialize result

    // Create a lookup table and fill all entries in it as -1
    int dp[n][n];
    memset(dp, -1, sizeof dp);

    // Compute longest path beginning from all cells
    for (int i=0; i<n; i++)
    {
      for (int j=0; j<n; j++)
       {
          if (dp[i][j] == -1)
             findLongestFromACell(i, j, mat, dp);

          //  Update result if needed
          result = max(result, dp[i][j]);
       }
     }

     return result;
}

// Driver program
int main()
{
   int  mat[n][n] = {{1, 2, 9},
                     {5, 3, 8},
                     {4, 6, 7}};
```

```
    cout << "Length of the longest path is "
         << finLongestOverAll(mat);
    return 0;
}
```

Output:

```
Length of the longest path is 4
```

Time complexity of the above solution is $O(n^2)$. It may seem more at first look. If we take a closer look, we can notice that all values of dp[i][j] are computed only once.

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

Source

http://www.geeksforgeeks.org/find-the-longest-path-in-a-matrix-with-given-constraints/

Category: Arrays Tags: Dynamic Programming, Matrix

Post navigation

← Sort a stack using recursion Paytm Interview Experience | Set 5 (Recruitment Drive) →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 19

Given a Boolean Matrix, find k such that all elements in k'th row are 0 and k'th column are 1

Given a square boolean matrix mat[n][n], find k such that all elements in k'th row are 0 and all elements in k'th column are 1. The value of mat[k][k] can be anything (either 0 or 1). If no such k exists, return -1.

Examples:

```
Input: bool mat[n][n] = { {1, 0, 0, 0},
                          {1, 1, 1, 0},
                          {1, 1, 0, 0},
                          {1, 1, 1, 0},
                        };
Output: 0
All elements in 0'th row are 0 and all elements in
0'th column are 1.  mat[0][0] is 1 (can be any value)


Input: bool mat[n][n] = {{0, 1, 1, 0, 1},
                         {0, 0, 0, 0, 0},
                         {1, 1, 1, 0, 0},
                         {1, 1, 1, 1, 0},
                         {1, 1, 1, 1, 1}};
Output: 1
All elements in 1'st row are 0 and all elements in
1'st column are 1.  mat[1][1] is 0 (can be any value)


Input: bool mat[n][n] = {{0, 1, 1, 0, 1},
                         {0, 0, 0, 0, 0},
                         {1, 1, 1, 0, 0},
                         {1, 0, 1, 1, 0},
                         {1, 1, 1, 1, 1}};
Output: -1
There is no k such that k'th row elements are 0 and
```

```
k'th column elements are 1.
```

Expected time complexity is O(n)

We strongly recommend you to minimize your browser and try this yourself first.

A **Simple Solution** is check all rows one by one. If we find a row 'i' such that all elements of this row are 0 except mat[i][i] which may be either 0 or 1, then we check all values in column 'i'. If all values are 1 in the column, then we return i. Time complexity of this solution is $O(n^2)$.

An **Efficient Solution** can solve this problem in O(n) time. The solution is based on below facts.
1) There can be at most one k that can be qualified to be an answer (Why? Note that if k'th row has all 0's probably except mat[k][k], then no column can have all 1)s.
2) If we traverse the given matrix from a corner (preferably from top right and bottom left), we can quickly discard complete row or complete column based on below rules.
….a) If mat[i][j] is 0 and i != j, then column j cannot be the solution.
….b) If mat[i][j] is 1 and i != j, then row i cannot be the solution.

Below is complete algorithm based on above observations.

```
1) Start from top right corner, i.e., i = 0, j = n-1.
   Initialize result as -1.

2) Do following until we find the result or reach outside the matrix.

......a) If mat[i][j] is 0, then check all elements on left of j in current row.
.........If all elements on left of j are also 0, then set result as i. Note
.........that i may not be result, but if there is a result, then it must be i
.........(Why? we reach mat[i][j] after discarding all rows above it and all
.........columns on right of it)

.........If all left side elements of i'th row are not 0, them this row cannot
.........be a solution, increment i.

......b) If mat[i][j] is 1, then check all elements below i in current column.
.........If all elements below i are 1, then set result as j. Note that j may
......... not be result, but if there is a result, then it must be j

.........If all elements of j'th column are not 1, them this column cannot be a
.........solution decrement j.

3) If result is -1, return it.

4) Else check validity of result by checking all row and column
          elements of result
```

Below is C++ implementation based on above idea.

C++

```
// C++ program to find i such that all entries in i'th row are 0
// and all entries in i't column are 1
#include <iostream>
```

```
using namespace std;
#define n 5

int find(bool arr[n][n])
{
    // Start from top-most rightmost corner
    // (We could start from other corners also)
    int i=0, j=n-1;

    // Initialize result
    int res = -1;

    // Find the index (This loop runs at most 2n times, we either
    // increment row number or decrement column number)
    while (i<n && j>=0)
    {
        // If current element is 0, then this row may be a solution
        if (arr[i][j] == 0)
        {
            // Check for all elements in this row
            while (j >= 0 && (arr[i][j] == 0 || i == j))
                j--;

            // If all values are 0, then store this row as result
            if (j == -1)
            {
                res = i;
                break;
            }

            // We reach here if we found a 1 in current row, so this
            //  row cannot be a solution, increment row number
            else i++;
        }
        else // If current element is 1
        {
            // Check for all elements in this column
            while (i<n && (arr[i][j] == 1 || i == j))
                i++;

            // If all elements are 1
            if (i == n)
            {
                res = j;
                break;
            }

            // We reach here if we found a 0 in current column, so this
            // column cannot be a solution, increment column number
            else j--;
        }
    }

    // If we could not find result in above loop, then result doesn't exist
```

```
    if (res == -1)
       return res;

    // Check if above computed res is valid
    for (int i=0; i<n; i++)
       if (res != i && arr[i][res] != 1)
          return -1;
    for (int j=0; j<n; j++)
       if (res != j && arr[res][j] != 0)
          return -1;

    return res;
}


/* Driver program to test above functions */
int main()
{
    bool mat[n][n] = {{0, 0, 1, 1, 0},
                      {0, 0, 0, 1, 0},
                      {1, 1, 1, 1, 0},
                      {0, 0, 0, 0, 0},
                      {1, 1, 1, 1, 1}};
    cout << find(mat);

    return 0;
}
```

Python

```
''' Python program to find k such that all elements in k'th row
    are 0 and k'th column are 1'''

def find(arr):

    # store length of the array
    n = len(arr)

    # start from top right-most corner
    i = 0
    j = n - 1

    # initialise result
    res = -1

    # find the index (This loop runs at most 2n times, we
    # either increment row number or decrement column number)
    while i < n and j >= 0:

        # if the current element is 0, then this row may be a solution
        if arr[i][j] == 0:
```

```
            # check for all the elements in this row
            while j >= 0 and (arr[i][j] == 0 or i == j):
                j -= 1

            # if all values are 0, update result as row number
            if j == -1:
                res = i
                break

            # if found a 1 in current row, the row can't be a
            # solution, increment row number
            else: i += 1

        # if the current element is 1
        else:

            #check for all the elements in this column
            while i < n and (arr[i][j] == 1 or i == j):
                i +=1

            # if all elements are 1, update result as col number
            if i == n:
                res = j
                break

            # if found a 0 in current column, the column can't be a
            # solution, decrement column number
            else: j -= 1

    # if we couldn't find result in above loop, result doesn't exist
    if res == -1:
        return res

    # check if the above computed res value is valid
    for i in range(0, n):
        if res != i and arr[i][res] != 1:
            return -1
    for j in range(0, j):
        if res != j and arr[res][j] != 0:
            return -1;

    return res;

# test find(arr) function
arr = [ [0,0,1,1,0],
        [0,0,0,1,0],
        [1,1,1,1,0],
        [0,0,0,0,0],
        [1,1,1,1,1] ]

print find(arr)
```

Output:

```
3
```

Time complexity of this solution is O(n). Note that we traverse at most 2n elements in the main while loop.

This article is contributed by **Ashish Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

http://www.geeksforgeeks.org/find-k-such-that-all-elements-in-kth-row-are-0-and-kth-column-are-1-in-a-boolean-matrix/

Category: Arrays Tags: Matrix

Post navigation

← Adobe Interview Experience | Set 30 (Off-Campus For Member Technical Staff) How to use GIT in Ubuntu ? (Part -2) →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 20

Find the largest rectangle of 1's with swapping of columns allowed

Given a matrix with 0 and 1's, find the largest rectangle of all 1's in the matrix. The rectangle can be formed by swapping any pair of columns of given matrix.

Example:

```
Input: bool mat[][] = { {0, 1, 0, 1, 0},
                        {0, 1, 0, 1, 1},
                        {1, 1, 0, 1, 0}
                      };
Output: 6
The largest rectangle's area is 6. The rectangle
can be formed by swapping column 2 with 3
The matrix after swapping will be
     0 0 1 1 0
     0 0 1 1 1
     1 0 1 1 0
```

```
Input: bool mat[R][C] = { {0, 1, 0, 1, 0},
                          {0, 1, 1, 1, 1},
                          {1, 1, 1, 0, 1},
                          {1, 1, 1, 1, 1}
                        };
Output: 9
```

We strongly recommend you to minimize your browser and try this yourself first.
The idea is to use an auxiliary matrix to store count of consecutive 1's in every column. Once we have these counts, we sort all rows of auxiliary matrix in non-increasing order of counts. Finally traverse the sorted rows to find the maximum area.

Below are detailed steps for first example mentioned above.

Step 1: First of all, calculate no. of consecutive 1's in every column. An auxiliary array hist[][] is used to store the counts of consecutive 1's. So for the above first example, contents of hist[R][C] would be

```
0 1 0 1 0
0 2 0 2 1
1 3 0 3 0
```

Time complexity of this step is O(R*C)

Step 2: Sort the rows in non-increasing fashion. After sorting step the matrix hist[][] would be

```
1 1 0 0 0
2 2 1 0 0
3 3 1 0 0
```

This step can be done in O(R * (R + C)). Since we know that the values are in range from 0 to R, we can use counting sort for every row.

Step 3: Traverse each row of hist[][] and check for the max area. Since every row is sorted by count of 1's, current area can be calculated by multiplying column number with value in hist[i][j]. This step also takes O(R * C) time.

Below is C++ implementation based of above idea.

```
// C++ program to find the largest rectangle of 1's with swapping
// of columns allowed.
#include<bits/stdc++.h>
#define R 3
#define C 5
using namespace std;

// Returns area of the largest rectangle of 1's
int maxArea(bool mat[R][C])
{
    // An auxiliary array to store count of consecutive 1's
    // in every column.
    int hist[R+1][C+1];

    // Step 1: Fill the auxiliary array hist[][]
    for (int i=0; i<C; i++)
    {
        // First row in hist[][] is copy of first row in mat[][]
        hist[0][i] = mat[0][i];

        // Fill remaining rows of hist[][]
        for (int j=1; j<R; j++)
            hist[j][i] = (mat[j][i]==0)? 0: hist[j-1][i]+1;
    }


    // Step 2: Sort rows of hist[][] in non-increasing order
    for (int i=0; i<R; i++)
    {
        int count[R+1] = {0};
```

```
        // counting occurrence
        for (int j=0; j<C; j++)
            count[hist[i][j]]++;

        //  Traverse the count array from right side
        int col_no = 0;
        for (int j=R; j>=0; j--)
        {
            if (count[j] > 0)
            {
                for (int k=0; k<count[j]; k++)
                {
                    hist[i][col_no] = j;
                    col_no++;
                }
            }
        }
    }

    // Step 3: Traverse the sorted hist[][] to find maximum area
    int curr_area, max_area = 0;
    for (int i=0; i<R; i++)
    {
        for (int j=0; j<C; j++)
        {
            // Since values are in decreasing order,
            // The area ending with cell (i, j) can
            // be obtained by multiplying column number
            // with value of hist[i][j]
            curr_area = (j+1)*hist[i][j];
            if (curr_area > max_area)
                max_area = curr_area;
        }
    }
    return max_area;
}

// Driver program
int main()
{
    bool mat[R][C] = { {0, 1, 0, 1, 0},
                       {0, 1, 0, 1, 1},
                       {1, 1, 0, 1, 0}
                     };
    cout << "Area of the largest rectangle is " << maxArea(mat);
    return 0;
}
```

Output:

```
 Area of the largest rectangle is 6
```

Time complexity of above solution is O(R * (R + C)) where R is number of rows and C is number of

columns in input matrix.

Extra space: O(R * C)

This article is contributed by **Shivprasad Choudhary**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

http://www.geeksforgeeks.org/find-the-largest-rectangle-of-1s-with-swapping-of-columns-allowed/

Category: Arrays Tags: Matrix

Post navigation

← Windows 10 –Feel the Difference enStage Bangalore Interview Experience for Software Engineer position →

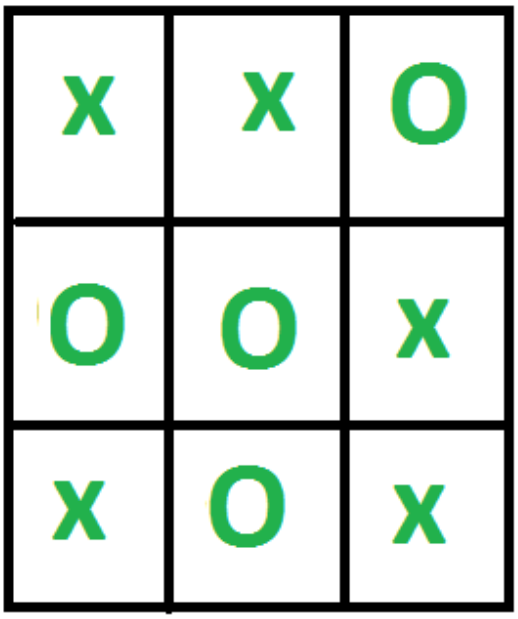
Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 21

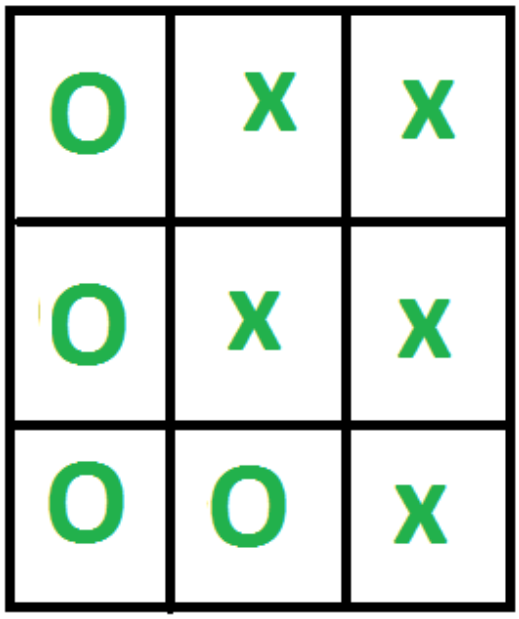
Validity of a given Tic-Tac-Toe board configuration

A Tic-Tac-Toe board is given after some moves are played. Find out if the given board is valid, i.e., is it possible to reach this board position after some moves or not.

Note that every arbitrary filled grid of 9 spaces isn't valid e.g. a grid filled with 3 X and 6 O isn't valid situation because each player needs to take alternate turns.



Valid Board



Invalid Board
(Both X and O cannot win)

Input is given as a 1D array of size 9.

```
Input: board[] =  {'X', 'X', 'O',
```

```
                  'O', 'O', 'X',
                  'X', 'O', 'X'};
Output: Valid

Input: board[] =  {'O', 'X', 'X',
                  'O', 'X', 'X',
                  'O', 'O', 'X'};
Output: Invalid
(Both X and O cannot win)

Input: board[] =  {'O', 'X', ' ',
                  ' ', ' ', ' ',
                  ' ', ' ', ' '};
Output: Valid
(Valid board with only two moves played)
```

We strongly recommend you to minimize your browser and try this yourself first.

Basically, to find the validity of an input grid, we can think of the conditions when an input grid is invalid. Let no. of "X"s be countX and no. of "O"s be countO. Since we know that the game starts with X, a given grid of Tic-Tac-Toe game would be definitely invalid if following two conditions meet
a) countX != countO AND
b) countX != countO + 1

Since "X" is always the first move, second condition is also required.

Now does it mean that all the remaining board positions are valid one? The answer is NO. Think of the cases when input grid is such that both X and O are making straight lines. This is also not valid position because the game ends when one player wins. So we need to check the following condition as well
c) If input grid shows that both the players are in winning situation, it's an invalid position.
d) If input grid shows that the player with O has put a straight-line (i.e. is in win condition) and countX != countO, it's an invalid position. The reason is that O plays his move only after X plays his move. Since X has started the game, O would win when both X and O has played equal no. of moves.
e) If input grid shows that X is in winning condition than xCount must be one greater that oCount.

Armed with above conditions i.e. a), b), c) and d), we can now easily formulate an algorithm/program to check the validity of a given Tic-Tac-Toe board position.

```
1)  countX == countO or countX == countO + 1
2)  If O is in win condition then check
     a)     If X also wins, not valid
     b)     If xbox != obox , not valid
3)  If X is in win condition then check if xCount is
     one more than oCount or not
```

Another way to find the validity of a given board is using 'inverse method' i.e. rule out all the possibilities when a given board is invalid.

```
// C++ program to check whether a given tic tac toe
// board is valid or not
#include <iostream>
using namespace std;
```

```
// This matrix is used to find indexes to check all
// possible wining triplets in board[0..8]
int win[8][3] = {{0, 1, 2}, // Check first row.
                 {3, 4, 5}, // Check second Row
                 {6, 7, 8}, // Check third Row
                 {0, 3, 6}, // Check first column
                 {1, 4, 7}, // Check second Column
                 {2, 5, 8}, // Check third Column
                 {0, 4, 8}, // Check first Diagonal
                 {2, 4, 6}}; // Check second Diagonal

// Returns true if character 'c' wins. c can be either
// 'X' or 'O'
bool isCWin(char *board, char c)
{
    // Check all possible winning combinations
    for (int i=0; i<8; i++)
        if (board[win[i][0]] == c &&
            board[win[i][1]] == c &&
            board[win[i][2]] == c )
            return true;
    return false;
}

// Returns true if given board is valid, else returns false
bool isValid(char board[9])
{
    // Count number of 'X' and 'O' in the given board
    int xCount=0, oCount=0;
    for (int i=0; i<9; i++)
    {
       if (board[i]=='X') xCount++;
       if (board[i]=='O') oCount++;
    }

    // Board can be valid only if either xCount and oCount
    // is same or xount is one more than oCount
    if (xCount==oCount || xCount==oCount+1)
    {
        // Check if 'O' is winner
        if (isCWin(board, 'O'))
        {
            // Check if 'X' is also winner, then
            // return false
            if (isCWin(board, 'X'))
                 return false;

            // Else return true xCount and yCount are same
            return (xCount == oCount);
        }

        // If 'X' wins, then count of X must be greater
        if (isCWin(board, 'X') && xCount != oCount + 1)
```

```
        return false;

        // If 'O' is not winner, then return true
        return true;
    }
    return false;
}

// Driver program
int main()
{
   char board[] = {'X', 'X', 'O',
                   'O', 'O', 'X',
                   'X', 'O', 'X'};
   (isValid(board))? cout << "Given board is valid":
                     cout << "Given board is not valid";
  return 0;
}
```

Output:

```
Given board is valid
```

Thanks to Utkarshfor suggesting this solution here.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

http://www.geeksforgeeks.org/validity-of-a-given-tic-tac-toe-board-configuration/

Category: Misc Tags: Matrix

Post navigation

← How to make a website using WordPress (Part – 1) SAP Labs Interview Experience | Set 10 (For Developer Specialist, 4-6 yrs) →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 22

Minimum Initial Points to Reach Destination

Given a grid with each cell consisting of positive, negative or no points i.e, zero points. We can move across a cell only if we have positive points (> 0). Whenever we pass through a cell, points in that cell are added to our overall points. We need to find minimum initial points to reach cell (m-1, n-1) from (0, 0).

Constraints :

- From a cell (i, j) we can move to (i+1, j) or (i, j+1).
- We cannot move from (i, j) if your overall points at (i, j) is
- We have to reach at (n-1, m-1) with minimum positive points i.e., > 0.

```
Input: points[m][n] = { {-2, -3,   3},
                        {-5, -10,  1},
                        {10,  30, -5}
                      };
Output: 7
Explanation:
7 is the minimum value to reach destination with
positive throughout the path. Below is the path.

(0,0) -> (0,1) -> (0,2) -> (1, 2) -> (2, 2)

We start from (0, 0) with 7, we reach(0, 1)
with 5, (0, 2) with 2, (1, 2) with 5, (2, 2)
with and finally we have 1 point (we needed
greater than 0 points at the end).
```

We strongly recommend you to minimize your browser and try this yourself first.

At the first look, this problem looks similar Max/Min Cost Path, but maximum overall points gained will not guarantee the minimum initial points. Also, it is compulsory in the current problem that the points never drops to zero or below. For instance, Suppose following two paths exists from source to destination cell.

We can solve this problem through bottom-up table filling dynamic programing technique.

- To begin with, we should maintain a 2D array dp of the same size as the grid, where dp[i][j] represents the minimum points that guarantees the continuation of the journey to destination before entering the cell (i, j). It's but obvious that dp[0][0] is our final solution. Hence, for this problem, we need to fill the table from the bottom right corner to left top.
- Now, let us decide minimum points needed to leave cell (i, j) (remember we are moving from bottom to up). There are only two paths to choose: (i+1, j) and (i, j+1). Of course we will choose the cell that the player can finish the rest of his journey with a smaller initial points. Therefore we have: **min_Points_on_exit = min(dp[i+1][j], dp[i][j+1])**

Now we know how to compute min_Points_on_exit, but we need to fill the table dp[][] to get the solution in dp[0][0].

How to compute dp[i][j]?
The value of dp[i][j] can be written as below.

dp[i][j] = max(min_Points_on_exit – points[i][j], 1)

Let us see how above expression covers all cases.

- If points[i][j] == 0, then nothing is gained in this cell; the player can leave the cell with the same points as he enters the room with, i.e. dp[i][j] = min_Points_on_exit.
- If dp[i][j] If dp[i][j] > 0, then the player could enter (i, j) with points as little as min_Points_on_exit – points[i][j]. since he could gain "points[i][j]" points in this cell. However, the value of min_Points_on_exit – points[i][j] might drop to 0 or below in this situation. When this happens, we must clip the value to 1 in order to make sure dp[i][j] stays positive:
  dp[i][j] = max(min_Points_on_exit – points[i][j], 1).

Finally return dp[0][0] which is our answer.

Below is C++ implementation of above algorithm.

```
// C++ program to find minimum initial points to reach destination
#include<bits/stdc++.h>
#define R 3
#define C 3
using namespace std;

int minInitialPoints(int points[][C])
{
    // dp[i][j] represents the minimum initial points player
    // should have so that when starts with cell(i, j) successfully
    // reaches the destination cell(m-1, n-1)
    int dp[R][C];
    int m = R, n = C;

    // Base case
    dp[m-1][n-1] = points[m-1][n-1] > 0? 1:
                   abs(points[m-1][n-1]) + 1;

    // Fill last row and last column as base to fill
    // entire table
    for (int i = m-2; i >= 0; i--)
         dp[i][n-1] = max(dp[i+1][n-1] - points[i][n-1], 1);
```

```
    for (int j = n-2; j >= 0; j--)
         dp[m-1][j] = max(dp[m-1][j+1] - points[m-1][j], 1);

    // fill the table in bottom-up fashion
    for (int i=m-2; i>=0; i--)
    {
        for (int j=n-2; j>=0; j--)
        {
            int min_points_on_exit = min(dp[i+1][j], dp[i][j+1]);
            dp[i][j] = max(min_points_on_exit - points[i][j], 1);
        }
     }

     return dp[0][0];
}

// Driver Program
int main()
{

    int points[R][C] = { {-2,-3,3},
                         {-5,-10,1},
                         {10,30,-5}
                       };
    cout << "Minimum Initial Points Required: "
         << minInitialPoints(points);
    return 0;
}
```

Output:

```
 Minimum Initial Points Required: 7
```

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

Source

http://www.geeksforgeeks.org/minimum-positive-points-to-reach-destination/

Category: Arrays Tags: Dynamic Programming, Matrix

Post navigation

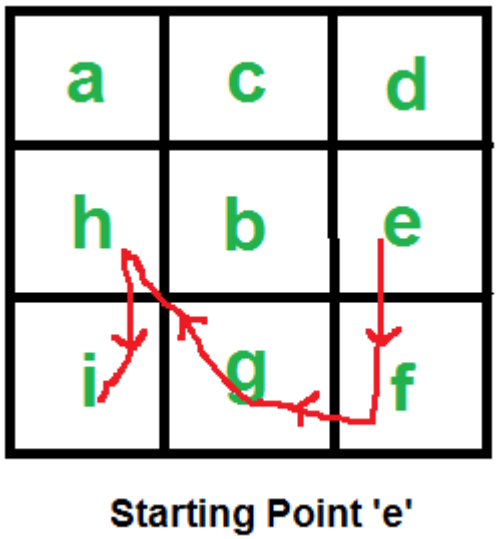
← Amazon Interview Experience | Set 227 (On-Campus for Internship and Full-Time) Count of n digit numbers whose sum of digits equals to given sum →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 23

Find length of the longest consecutive path from a given starting character

Given a matrix of characters. Find length of the longest path from a given character, such that all characters in the path are consecutive to each other, i.e., every character in path is next to previous in alphabetical order. It is allowed to move in all 8 directions from a cell.



Example

```
Input: mat[][] = { {a, c, d},
                   {h, b, e},
                   {i, g, f}}
      Starting Point = 'e'

Output: 5
If starting point is 'e', then longest path with consecutive
characters is "e f g h i".
```

```
Input: mat[R][C] = { {b, e, f},
                     {h, d, a},
                     {i, c, a}};
      Starting Point = 'b'

Output: 1
'c' is not present in all adjacent cells of 'b'
```

We strongly recommend you to minimize your browser and try this yourself first.

The idea is to first search given starting character in the given matrix. Do Depth First Search (DFS) from all occurrences to find all consecutive paths. While doing DFS, we may encounter many subproblems again and again. So we use dynamic programming to store results of subproblems.

Below is C++ implementation of above idea.

```
// C++ program to find the longest consecutive path
#include<bits/stdc++.h>
#define R 3
#define C 3
using namespace std;

// tool matrices to recur for adjacent cells.
int x[] = {0, 1, 1, -1, 1, 0, -1, -1};
int y[] = {1, 0, 1, 1, -1, -1, 0, -1};

// dp[i][j] Stores length of longest consecutive path
// starting at arr[i][j].
int dp[R][C];

// check whether mat[i][j] is a valid cell or not.
bool isvalid(int i, int j)
{
    if (i < 0 || j < 0 || i >= R || j >= C)
      return false;
    return true;
}

// Check whether current character is adjacent to previous
// character (character processed in parent call) or not.
bool isadjacent(char prev, char curr)
{
    return ((curr - prev) == 1);
}

// i, j are the indices of the current cell and prev is the
// character processed in the parent call.. also mat[i][j]
// is our current character.
int getLenUtil(char mat[R][C], int i, int j, char prev)
{
     // If this cell is not valid or current character is not
     // adjacent to previous one (e.g. d is not adjacent to b )
     // or if this cell is already included in the path than return 0.
```

```
    if (!isvalid(i, j) || !isadjacent(prev, mat[i][j]))
         return 0;

    // If this subproblem is already solved , return the answer
    if (dp[i][j] != -1)
        return dp[i][j];

    int ans = 0;  // Initialize answer

    // recur for paths with differnt adjacent cells and store
    // the length of longest path.
    for (int k=0; k<8; k++)
      ans = max(ans, 1 + getLenUtil(mat, i + x[k],
                                    j + y[k], mat[i][j]));

    // save the answer and return
    return dp[i][j] = ans;
}

// Returns length of the longest path with all characters consecutive
// to each other.  This function first initializes dp array that
// is used to store results of subproblems, then it calls
// recursive DFS based function getLenUtil() to find max length path
int getLen(char mat[R][C], char s)
{
    memset(dp, -1, sizeof dp);
    int ans = 0;

    for (int i=0; i<R; i++)
    {
        for (int j=0; j<C; j++)
        {
            // check for each possible starting point
            if (mat[i][j] == s) {

                // recur for all eight adjacent cells
                for (int k=0; k<8; k++)
                  ans = max(ans, 1 + getLenUtil(mat,
                                    i + x[k], j + y[k], s));
            }
        }
    }
    return ans;
}

// Driver program
int main() {

    char mat[R][C] = { {'a','c','d'},
                       { 'h','b','a'},
                       { 'i','g','f'}};

    cout << getLen(mat, 'a') << endl;
    cout << getLen(mat, 'e') << endl;
```

```
    cout << getLen(mat, 'b') << endl;
    cout << getLen(mat, 'f') << endl;
    return 0;
}
```

Output:

```
 4
0
3
4
```

Thanks to Gaurav Ahirwar for above solution.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

http://www.geeksforgeeks.org/find-length-of-the-longest-consecutive-path-in-a-character-matrix/

Category: Arrays Tags: Dynamic Programming, Matrix

Post navigation

← Fiberlink (maas360) Interview Experience | Set 4 (Off-Campus) Goldman Sachs Interview Experience | Set 10 (On-Campus) →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 24

Collect maximum points in a grid using two traversals

Given a matrix where every cell represents points. How to collect maximum points using two traversals under following conditions?

Let the dimensions of given grid be R x C.

1) The first traversal starts from top left corner, i.e., (0, 0) and should reach left bottom corner, i.e., (R-1, 0). The second traversal starts from top right corner, i.e., (0, C-1) and should reach bottom right corner, i.e., (R-1, C-1)/

2) From a point (i, j), we can move to (i+1, j+1) or (i+1, j-1) or (i+1, j)

3) A traversal gets all points of a particular cell through which it passes. If one traversal has already collected points of a cell, then the other traversal gets no points if goes through that cell again.

```
Input :
    int arr[R][C] = {{3, 6, 8, 2},
                     {5, 2, 4, 3},
                     {1, 1, 20, 10},
                     {1, 1, 20, 10},
                     {1, 1, 20, 10},
                    };

     Output: 73

Explanation :

First traversal collects total points of value 3 + 2 + 20 + 1 + 1 = 27

Second traversal collects total points of value 2 + 4 + 10 + 20 + 10 = 46.
Total Points collected = 27 + 46 = 73.
```

Source: http://qa.geeksforgeeks.org/1485/running-through-the-grid-to-get-maximum-nutritional-value

We strongly recommend you to minimize your browser and try this yourself first.
The idea is to do both traversals concurrently. We start first from (0, 0) and second traversal from (0, C-1) simultaneously. The important thing to note is, at any particular step both traversals will be in same row

as in all possible three moves, row number is increased. Let (x1, y1) and (x2, y2) denote current positions of first and second traversals respectively. Thus at any time x1 will be equal to x2 as both of them move forward but variation is possible along y. Since variation in y could occur in 3 ways no change (y), go left (y – 1), go right (y + 1). So in total 9 combinations among y1, y2 are possible. The 9 cases as mentioned below after base cases.

```
Both traversals always move forward along x
Base Cases:
// If destinations reached
if (x == R-1 && y1 == 0 && y2 == C-1)
maxPoints(arr, x, y1, y2) = arr[x][y1] + arr[x][y2];

// If any of the two locations is invalid (going out of grid)
if input is not valid
maxPoints(arr, x, y1, y2) = -INF  (minus infinite)

// If both traversals are at same cell, then we count the value of cell
// only once.
If y1 and y2 are same
    result = arr[x][y1]
Else
    result = arr[x][y1] + arr[x][y2]

result  +=  max { // Max of 9 cases
                  maxPoints(arr, x+1, y1+1, y2),
                  maxPoints(arr, x+1, y1+1, y2+1),
                  maxPoints(arr, x+1, y1+1, y2-1),
                  maxPoints(arr, x+1, y1-1, y2),
                  maxPoints(arr, x+1, y1-1, y2+1),
                  maxPoints(arr, x+1, y1-1, y2-1),
                  maxPoints(arr, x+1, y1, y2),
                  maxPoints(arr, x+1, y1, y2+1),
                  maxPoints(arr, x+1, y1, y2-1)
                }
```

The above recursive solution has many subproblems that are solved again and again. Therefore, we can use Dynamic Programming to solve the above problem more efficiently. Below is memoization (Memoization is alternative to table based iterative solution in Dynamic Programming) based implementation. In below implementation, we use a memoization table 'mem' to keep track of already solved problems.

```
// A Memoization based program to find maximum collection
// using two traversals of a grid
#include<bits/stdc++.h>
using namespace std;
#define R 5
#define C 4

// checks whether a given input is valid or not
bool isValid(int x, int y1, int y2)
{
```

```
    return (x >= 0 && x < R && y1 >=0 &&
            y1 < C && y2 >=0 && y2 < C);
}

// Driver function to collect max value
int getMaxUtil(int arr[R][C], int mem[R][C][C], int x, int y1, int y2)
{
    /*---------- BASE CASES -----------*/
    // if P1 or P2 is at an invalid cell
    if (!isValid(x, y1, y2)) return INT_MIN;

    // if both traversals reach their destinations
    if (x == R-1 && y1 == 0 && y2 == C-1)
           return arr[x][y1] + arr[x][y2];

    // If both traversals are at last row but not at their destination
    if (x == R-1) return INT_MIN;

    // If subproblem is already solved
    if (mem[x][y1][y2] != -1) return mem[x][y1][y2];

    // Initialize answer for this subproblem
    int ans = INT_MIN;

    // this variable is used to store gain of current cell(s)
    int temp = (y1 == y2)? arr[x][y1]: arr[x][y1] + arr[x][y2];

    /* Recur for all possible cases, then store and return the
       one with max value */
    ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1, y2-1));
    ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1, y2+1));
    ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1, y2));

    ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1-1, y2));
    ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1-1, y2-1));
    ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1-1, y2+1));

    ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1+1, y2));
    ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1+1, y2-1));
    ans = max(ans, temp + getMaxUtil(arr, mem, x+1, y1+1, y2+1));

    return (mem[x][y1][y2] = ans);
}

// This is mainly a wrapper over recursive function getMaxUtil().
// This function creates a table for memoization and calls
// getMaxUtil()
int geMaxCollection(int arr[R][C])
{
    // Create a memoization table and initialize all entries as -1
    int mem[R][C][C];
    memset(mem, -1, sizeof(mem));

    // Calculation maximum value using memoization based function
```

```
    // getMaxUtil()
    return getMaxUtil(arr, mem, 0, 0, C-1);
}

// Driver program to test above functions
int main()
{
    int arr[R][C] = {{3, 6, 8, 2},
                     {5, 2, 4, 3},
                     {1, 1, 20, 10},
                     {1, 1, 20, 10},
                     {1, 1, 20, 10},
                    };
    cout << "Maximum collection is " << geMaxCollection(arr);
    return 0;
}
```

Output:

```
Maximum collection is 73
```

Thanks to Gaurav Ahirwar for suggesting above problem and solution here.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

http://www.geeksforgeeks.org/collect-maximum-points-in-a-grid-using-two-traversals/

Category: Arrays Tags: Dynamic Programming, Matrix

Post navigation

← Amazon Interview Experience | Set 213 (Off-Campus for SDE1) Amazon Interview Experience | 214 (On-Campus) →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Chapter 25

Rotate Matrix Elements

Given a matrix, clockwise rotate elements in it.

Examples:

```
Input
1    2    3
4    5    6
7    8    9

Output:
4    1    2
7    5    3
8    9    6

For 4*4 matrix
Input:
1    2    3    4
5    6    7    8
9    10   11   12
13   14   15   16

Output:
5    1    2    3
9    10   6    4
13   11   7    8
14   15   16   12
```

We strongly recommend you to minimize your browser and try this yourself first.

The idea is to use loops similar to the program for printing a matrix in spiral form. One by one rotate all rings of elements, starting from the outermost. To rotate a ring, we need to do following.
1) Move elements of top row.
2) Move elements of last column.
3) Move elements of bottom row.
4) Move elements of first column.
Repeat above steps for inner ring while there is an inner ring.

Below is the implementation of above idea. Thanks to Gaurav Ahirwar for suggesting below solution here.

C/C++

```
// C++ program to rotate a matrix

#include <bits/stdc++.h>
#define R 4
#define C 4
using namespace std;

// A function to rotate a matrix mat[][] of size R x C.
// Initially, m = R and n = C
void rotatematrix(int m, int n, int mat[R][C])
{
    int row = 0, col = 0;
    int prev, curr;

    /*
       row - Staring row index
       m - ending row index
       col - starting column index
       n - ending column index
       i - iterator
    */
    while (row < m && col < n)
    {

        if (row + 1 == m || col + 1 == n)
            break;

        // Store the first element of next row, this
        // element will replace first element of current
        // row
        prev = mat[row + 1][col];

         /* Move elements of first row from the remaining rows */
        for (int i = col; i < n; i++)
        {
            curr = mat[row][i];
            mat[row][i] = prev;
            prev = curr;
        }
        row++;

        /* Move elements of last column from the remaining columns */
        for (int i = row; i < m; i++)
        {
            curr = mat[i][n-1];
            mat[i][n-1] = prev;
            prev = curr;
        }
        n--;

         /* Move elements of last row from the remaining rows */
```

```
        if (row < m)
        {
            for (int i = n-1; i >= col; i--)
            {
                curr = mat[m-1][i];
                mat[m-1][i] = prev;
                prev = curr;
            }
        }
        m--;

        /* Move elements of first column from the remaining rows */
        if (col < n)
        {
            for (int i = m-1; i >= row; i--)
            {
                curr = mat[i][col];
                mat[i][col] = prev;
                prev = curr;
            }
        }
        col++;
    }

    // Print rotated matrix
    for (int i=0; i<R; i++)
    {
        for (int j=0; j<C; j++)
          cout << mat[i][j] << " ";
        cout << endl;
    }
}

/* Driver program to test above functions */
int main()
{
    // Test Case 1
    int a[R][C] = { {1,  2,  3,  4},
        {5,  6,  7,  8},
        {9,  10, 11, 12},
        {13, 14, 15, 16}  };

    // Tese Case 2
    /* int a[R][C] = {{1, 2, 3},
                      {4, 5, 6},
                      {7, 8, 9}
                     };
     */  rotatematrix(R, C, a);
    return 0;
}
```

Python

```
# Python program to rotate a matrix

# Function to rotate a matrix
def rotateMatrix(mat):

    if not len(mat):
        return

    """
        top : starting row index
        bottom : ending row index
        left : starting column index
        right : ending column index
    """

    top = 0
    bottom = len(mat)-1

    left = 0
    right = len(mat[0])-1

    while left < right and top < bottom:

        # Store the first element of next row,
        # this element will replace first element of
        # current row
        prev = mat[top+1][left]

        # Move elements of top row one step right
        for i in range(left, right+1):
            curr = mat[top][i]
            mat[top][i] = prev
            prev = curr

        top += 1

        # Move elements of rightmost column one step downwards
        for i in range(top, bottom+1):
            curr = mat[i][right]
            mat[i][right] = prev
            prev = curr

        right -= 1

        # Move elements of bottom row one step left
        for i in range(right, left-1, -1):
            curr = mat[bottom][i]
            mat[bottom][i] = prev
            prev = curr

        bottom -= 1

        # Move elements of leftmost column one step upwards
```

```
        for i in range(bottom, top-1, -1):
            curr = mat[i][left]
            mat[i][left] = prev
            prev = curr

        left += 1

    return mat

# Utility Function
def printMatrix(mat):
    for row in mat:
        print row


# Test case 1
matrix =[
            [1,  2,  3,  4 ],
            [5,  6,  7,  8 ],
            [9,  10, 11, 12 ],
            [13, 14, 15, 16 ]
        ]
# Test case 2
"""
matrix =[
            [1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]
        ]
"""

matrix = rotateMatrix(matrix)
# Print modified matrix
printMatrix(matrix)
```

Output:

```
 5 1 2 3
9 10 6 4
13 11 7 8
14 15 16 12
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

http://www.geeksforgeeks.org/rotate-matrix-elements/

Chapter 26

Find sum of all elements in a matrix except the elements in row and/or column of given cell?

Given a 2D matrix and a set of cell indexes e.g., an array of (i, j) where i indicates row and j column. For every given cell index (i, j), find sums of all matrix elements except the elements present in i'th row and/or j'th column.

```
Example:
mat[][]  = { {1, 1, 2}
             {3, 4, 6}
             {5, 3, 2} }
Array of Cell Indexes: {(0, 0), (1, 1), (0, 1)}
Output:  15, 10, 16
```

We strongly recommend you to minimize your browser and try this yourself first.

Source: http://qa.geeksforgeeks.org/622/select-column-matrix-then-find-remaining-elements-matrices?show=625#a625

A **Naive Solution** is to one by once consider all given cell indexes. For every cell index (i, j), find the sum of matrix elements that are not present either at i'th row or at j'th column. Below is C++ implementation of the Naive approach.

```
#include<bits/stdc++.h>
#define R 3
#define C 3
using namespace std;

// A structure to represent a cell index
struct Cell
{
    int r; // r is row, varies from 0 to R-1
    int c; // c is column, varies from 0 to C-1
```

```
};

// A simple solution to find sums for a given array of cell indexes
void printSums(int mat[][C], struct Cell arr[], int n)
{
    // Iterate through all cell indexes
    for (int i=0; i<n; i++)
    {
        int sum = 0, r = arr[i].r, c = arr[i].c;

        // Compute sum for current cell index
        for (int j=0; j<R; j++)
            for (int k=0; k<C; k++)
                if (j != r && k != c)
                    sum += mat[j][k];
        cout << sum << endl;
    }
}

// Driver program to test above
int main()
{
    int mat[][C] = {{1, 1, 2}, {3, 4, 6}, {5, 3, 2}};
    struct Cell arr[] = {{0, 0}, {1, 1}, {0, 1}};
    int n = sizeof(arr)/sizeof(arr[0]);
    printSums(mat, arr, n);
    return 0;
}
```

Output:

```
 15
10
16
```

Time complexity of the above solution is O(n * R * C) where n is number of given cell indexes and R x C is matrix size.

An **Efficient Solution** can compute all sums in O(R x C + n) time. The idea is to precompute total sum, row and column sums before processing the given array of indexes. Below are details
1. Calculate sum of matrix, call it sum.
2. Calculate sum of individual rows and columns. (row[] and col[])
3. For a cell index (i, j), the desired sum will be "sum- row[i] – col[j] + arr[i][j]"

Below is C++ implementation of above idea.

```
// An efficient C++ program to compute sum for given array of cell indexes
#include<bits/stdc++.h>
#define R 3
#define C 3
using namespace std;

// A structure to represent a cell index
```

```
struct Cell
{
    int r; // r is row, varies from 0 to R-1
    int c; // c is column, varies from 0 to C-1
};

void printSums(int mat[][C], struct Cell arr[], int n)
{
    int sum = 0;
    int row[R] = {};
    int col[C] = {};

    // Compute sum of all elements, sum of every row and sum every column
    for (int i=0; i<R; i++)
    {
      for (int j=0; j<C; j++)
       {
             sum += mat[i][j];
             col[j] += mat[i][j];
             row[i] += mat[i][j];
       }
    }

    // Compute the desired sum for all given cell indexes
    for (int i=0; i<n; i++)
    {
        int ro = arr[i].r, co = arr[i].c;
        cout << sum - row[ro] - col[co] + mat[ro][co] << endl;
    }
}

// Driver program to test above function
int main()
{
    int mat[][C] = {{1, 1, 2}, {3, 4, 6}, {5, 3, 2}};
    struct Cell arr[] = {{0, 0}, {1, 1}, {0, 1}};
    int n = sizeof(arr)/sizeof(arr[0]);
    printSums(mat, arr, n);
    return 0;
}
```

Output:

```
 15
10
16
```

Time Complexity: O(R x C + n)
Auxiliary Space: O(R + C)

Thanks to Gaurav Ahirwar for suggesting this efficient solution here.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

http://www.geeksforgeeks.org/find-sum-of-all-elements-in-a-matrix-except-the-elements-in-given-row-andor-column-2/

Category: Arrays Tags: Matrix

Post navigation

← Amazon Interview Experience | 198 (For SDE1) Xome interview experience for software developer →

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.