# Copyright

The content of this book comes from [geeksforgeeks.org](http://geeksforgeeks.org) and it's licensed under [Creative Commons Attribution-NonCommercial-NoDerivs 2.5 India](http://creativecommons.org/licenses/by-nc-nd/2.5/in/deed.en_US)

Made by [Jing](http://www.jing-zhou.me/). 2015.

Updated on September 12, 2015

Head over to [this github repo](https://github.com/gnijuohz/geeksforgeeks-as-books) to report issues or contribute.

# 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](http://www.geeksforgeeks.org/dynamic-programming-set-1/) (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](http://qa.geeksforgeeks.org/1485/running-through-the-grid-to-get-maximum-nutritional-value).

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](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Dynamic Programming](http://www.geeksforgeeks.org/tag/dynamic-programming/), [Matrix](http://www.geeksforgeeks.org/tag/matrix/)

Post navigation

[← Amazon Interview Experience | Set 213 (Off-Campus for SDE1)](http://www.geeksforgeeks.org/amazon-interview-experience-set-213-off-campus-for-sde1/) [Amazon Interview Experience | 214 (On-Campus) →](http://www.geeksforgeeks.org/amazon-interview-experience-214-on-campus/)

# 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](http://geeksquiz.com/binary-search/) instead of linear search.

We can solve this problem **in O(mn) time** using the approach similar to merge of [Merge Sort](http://geeksquiz.com/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.

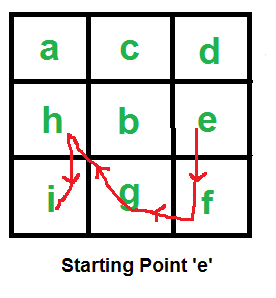
This article is contributed by **Anand Agrawal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

### Source

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

# 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.

[](http://d2dskowxfbo68o.cloudfront.net/wp-content/uploads/matrix.png)

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](http://qa.geeksforgeeks.org/user/Mr.Lazy) 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](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Dynamic Programming](http://www.geeksforgeeks.org/tag/dynamic-programming/), [Matrix](http://www.geeksforgeeks.org/tag/matrix/)

Post navigation

[← Fiberlink (maas360) Interview Experience | Set 4 (Off-Campus)](http://www.geeksforgeeks.org/fiberlink-maas360-interview-experience-set-4-off-campus/) [Goldman Sachs Interview Experience | Set 10 (On-Campus) →](http://www.geeksforgeeks.org/goldman-sachs-interview-experience-set-10-on-campus/)

# 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](https://www.facebook.com/COOL.DUDE.BORN.NUD3?fref=ts&ref=br_tf) for suggesting this efficient solution [here](http://qa.geeksforgeeks.org/622/select-column-matrix-then-find-remaining-elements-matrices?show=625#a625).

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](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Matrix](http://www.geeksforgeeks.org/tag/matrix/)

Post navigation

[← Amazon Interview Experience | 198 (For SDE1)](http://www.geeksforgeeks.org/amazon-interview-experience-198-for-sde1/) [Xome interview experience for software developer →](http://www.geeksforgeeks.org/xome-interview-experience-for-software-developer/)

# 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](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Matrix](http://www.geeksforgeeks.org/tag/matrix/)

Post navigation

[← Windows 10 –Feel the Difference](http://www.geeksforgeeks.org/windows-10-feel-the-difference/)

# 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(k2n2). We can solve this problem in O(n2) 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](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Matrix](http://www.geeksforgeeks.org/tag/matrix/)

Post navigation

[← SAP Labs Interview Experience for Developer Associate](http://www.geeksforgeeks.org/sap-labs-interview-experience-developer-associate/) [Analysis of Algorithm | Set 5 (Amortized Analysis Introduction) →](http://www.geeksforgeeks.org/analysis-algorithm-set-5-amortized-analysis-introduction/)

# 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](http://www.geeksforgeeks.org/dynamic-programming-set-6-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](http://sudo%20aptitude%20install%20build-essential). 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](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Dynamic Programming](http://www.geeksforgeeks.org/tag/dynamic-programming/), [Matrix](http://www.geeksforgeeks.org/tag/matrix/)

Post navigation

[← Amazon Interview Experience | Set 227 (On-Campus for Internship and Full-Time)](http://www.geeksforgeeks.org/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 →](http://www.geeksforgeeks.org/count-of-n-digit-numbers-whose-sum-of-digits-equals-to-given-sum/)

# Mobile Numeric Keypad Problem

[](http://d2dskowxfbo68o.cloudfront.net/wp-content/uploads/mobile1.png)Given the mobile numeric keypad. You can only press buttons that are up, left, right or down to the current button. You are not allowed to press bottom row corner buttons (i.e. \* and # ).  
 Given a number N, find out the number of possible numbers of given length.

Examples:  
 For N=1, number of possible numbers would be 10 (0, 1, 2, 3, …., 9)  
 For N=2, number of possible numbers would be 36  
 Possible numbers: 00,08 11,12,14 22,21,23,25 and so on.  
 If we start with 0, valid numbers will be 00, 08 (count: 2)  
 If we start with 1, valid numbers will be 11, 12, 14 (count: 3)  
 If we start with 2, valid numbers will be 22, 21, 23,25 (count: 4)  
 If we start with 3, valid numbers will be 33, 32, 36 (count: 3)  
 If we start with 4, valid numbers will be 44,41,45,47 (count: 4)  
 If we start with 5, valid numbers will be 55,54,52,56,58 (count: 5)  
 ………………………………  
 ………………………………

We need to print the count of possible numbers.

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

N = 1 is trivial case, number of possible numbers would be 10 (0, 1, 2, 3, …., 9)  
 For N > 1, we need to start from some button, then move to any of the four direction (up, left, right or down) which takes to a valid button (should not go to \*, #). Keep doing this until N length number is obtained (depth first traversal).

**Recursive Solution:**  
 Mobile Keypad is a rectangular grid of 4X3 (4 rows and 3 columns)  
 Lets say Count(i, j, N) represents the count of N length numbers starting from position (i, j)

If N = 1  
 Count(i, j, N) = 10   
Else  
 Count(i, j, N) = Sum of all Count(r, c, N-1) where (r, c) is new   
 position after valid move of length 1 from current   
 position (i, j)

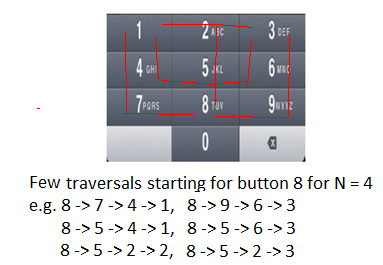
Following is C implementation of above recursive formula.

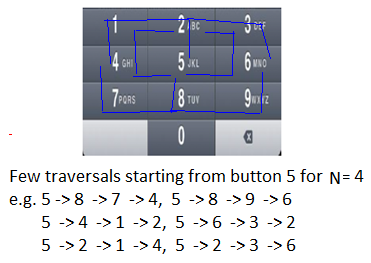
// A Naive Recursive C program to count number of possible numbers  
// of given length  
#include <stdio.h>  
  
// left, up, right, down move from current location  
int row[] = {0, 0, -1, 0, 1};  
int col[] = {0, -1, 0, 1, 0};  
  
// Returns count of numbers of length n starting from key position  
// (i, j) in a numeric keyboard.  
int getCountUtil(char keypad[][3], int i, int j, int n)  
{  
 if (keypad == NULL || n <= 0)  
 return 0;  
  
 // From a given key, only one number is possible of length 1  
 if (n == 1)  
 return 1;  
  
 int k=0, move=0, ro=0, co=0, totalCount = 0;  
  
 // move left, up, right, down from current location and if  
 // new location is valid, then get number count of length  
 // (n-1) from that new position and add in count obtained so far  
 for (move=0; move<5; move++)  
 {  
 ro = i + row[move];  
 co = j + col[move];  
 if (ro >= 0 && ro <= 3 && co >=0 && co <= 2 &&  
 keypad[ro][co] != '\*' && keypad[ro][co] != '#')  
 {  
 totalCount += getCountUtil(keypad, ro, co, n-1);  
 }  
 }  
  
 return totalCount;  
}  
  
// Return count of all possible numbers of length n  
// in a given numeric keyboard  
int getCount(char keypad[][3], int n)  
{  
 // Base cases  
 if (keypad == NULL || n <= 0)  
 return 0;  
 if (n == 1)  
 return 10;  
  
 int i=0, j=0, totalCount = 0;  
 for (i=0; i<4; i++) // Loop on keypad row  
 {  
 for (j=0; j<3; j++) // Loop on keypad column  
 {  
 // Process for 0 to 9 digits  
 if (keypad[i][j] != '\*' && keypad[i][j] != '#')  
 {  
 // Get count when number is starting from key  
 // position (i, j) and add in count obtained so far  
 totalCount += getCountUtil(keypad, i, j, n);  
 }  
 }  
 }  
 return totalCount;  
}  
  
// Driver program to test above function  
int main(int argc, char \*argv[])  
{  
 char keypad[4][3] = {{'1','2','3'},  
 {'4','5','6'},  
 {'7','8','9'},  
 {'\*','0','#'}};  
 printf("Count for numbers of length %d: %d\n", 1, getCount(keypad, 1));  
 printf("Count for numbers of length %d: %d\n", 2, getCount(keypad, 2));  
 printf("Count for numbers of length %d: %d\n", 3, getCount(keypad, 3));  
 printf("Count for numbers of length %d: %d\n", 4, getCount(keypad, 4));  
 printf("Count for numbers of length %d: %d\n", 5, getCount(keypad, 5));  
  
 return 0;  
}

Output:

Count for numbers of length 1: 10  
Count for numbers of length 2: 36  
Count for numbers of length 3: 138  
Count for numbers of length 4: 532  
Count for numbers of length 5: 2062

**Dynamic Programming**  
 There are many repeated traversal on smaller paths (traversal for smaller N) to find all possible longer paths (traversal for bigger N). See following two diagrams for example. In this traversal, for N = 4 from two starting positions (buttons ‘4’ and ‘8’), we can see there are few repeated traversals for N = 2 (e.g. 4 -> 1, 6 -> 3, 8 -> 9, 8 -> 7 etc).

[](http://d2dskowxfbo68o.cloudfront.net/wp-content/uploads/mobile2.png)

[](http://d2dskowxfbo68o.cloudfront.net/wp-content/uploads/mobile3.png)

Since the problem has both properties: [Optimal Substructure](http://www.geeksforgeeks.org/dynamic-programming-set-2-optimal-substructure-property/) and [Overlapping Subproblems](http://www.geeksforgeeks.org/dynamic-programming-set-1/), it can be efficiently solved using dynamic programming.

Following is C program for dynamic programming implementation.

// A Dynamic Programming based C program to count number of  
// possible numbers of given length  
#include <stdio.h>  
  
// Return count of all possible numbers of length n  
// in a given numeric keyboard  
int getCount(char keypad[][3], int n)  
{  
 if(keypad == NULL || n <= 0)  
 return 0;  
 if(n == 1)  
 return 10;  
  
 // left, up, right, down move from current location  
 int row[] = {0, 0, -1, 0, 1};  
 int col[] = {0, -1, 0, 1, 0};  
  
 // taking n+1 for simplicity - count[i][j] will store  
 // number count starting with digit i and length j  
 int count[10][n+1];  
 int i=0, j=0, k=0, move=0, ro=0, co=0, num = 0;  
 int nextNum=0, totalCount = 0;  
  
 // count numbers starting with digit i and of lengths 0 and 1  
 for (i=0; i<=9; i++)  
 {  
 count[i][0] = 0;  
 count[i][1] = 1;  
 }  
  
 // Bottom up - Get number count of length 2, 3, 4, ... , n  
 for (k=2; k<=n; k++)  
 {  
 for (i=0; i<4; i++) // Loop on keypad row  
 {  
 for (j=0; j<3; j++) // Loop on keypad column  
 {  
 // Process for 0 to 9 digits  
 if (keypad[i][j] != '\*' && keypad[i][j] != '#')  
 {  
 // Here we are counting the numbers starting with  
 // digit keypad[i][j] and of length k keypad[i][j]  
 // will become 1st digit, and we need to look for  
 // (k-1) more digits  
 num = keypad[i][j] - '0';  
 count[num][k] = 0;  
  
 // move left, up, right, down from current location  
 // and if new location is valid, then get number  
 // count of length (k-1) from that new digit and  
 // add in count we found so far  
 for (move=0; move<5; move++)  
 {  
 ro = i + row[move];  
 co = j + col[move];  
 if (ro >= 0 && ro <= 3 && co >=0 && co <= 2 &&  
 keypad[ro][co] != '\*' && keypad[ro][co] != '#')  
 {  
 nextNum = keypad[ro][co] - '0';  
 count[num][k] += count[nextNum][k-1];  
 }  
 }  
 }  
 }  
 }  
 }  
  
 // Get count of all possible numbers of length "n" starting  
 // with digit 0, 1, 2, ..., 9  
 totalCount = 0;  
 for (i=0; i<=9; i++)  
 totalCount += count[i][n];  
 return totalCount;  
}  
  
// Driver program to test above function  
int main(int argc, char \*argv[])  
{  
 char keypad[4][3] = {{'1','2','3'},  
 {'4','5','6'},  
 {'7','8','9'},  
 {'\*','0','#'}};  
 printf("Count for numbers of length %d: %d\n", 1, getCount(keypad, 1));  
 printf("Count for numbers of length %d: %d\n", 2, getCount(keypad, 2));  
 printf("Count for numbers of length %d: %d\n", 3, getCount(keypad, 3));  
 printf("Count for numbers of length %d: %d\n", 4, getCount(keypad, 4));  
 printf("Count for numbers of length %d: %d\n", 5, getCount(keypad, 5));  
  
 return 0;  
}

Output:

Count for numbers of length 1: 10  
Count for numbers of length 2: 36  
Count for numbers of length 3: 138  
Count for numbers of length 4: 532  
Count for numbers of length 5: 2062

**A Space Optimized Solution:**  
 The above dynamic programming approach also runs in O(n) time and requires O(n) auxiliary space, as only one for loop runs n times, other for loops runs for constant time. We can see that nth iteration needs data from (n-1)th iteration only, so we need not keep the data from older iterations. We can have a space efficient dynamic programming approach with just two arrays of size 10. Thanks to Nik for suggesting this solution.

// A Space Optimized C program to count number of possible numbers  
// of given length  
#include <stdio.h>  
  
// Return count of all possible numbers of length n  
// in a given numeric keyboard  
int getCount(char keypad[][3], int n)  
{  
 if(keypad == NULL || n <= 0)  
 return 0;  
 if(n == 1)  
 return 10;  
  
 // odd[i], even[i] arrays represent count of numbers starting  
 // with digit i for any length j  
 int odd[10], even[10];  
 int i = 0, j = 0, useOdd = 0, totalCount = 0;  
  
 for (i=0; i<=9; i++)  
 odd[i] = 1; // for j = 1  
  
 for (j=2; j<=n; j++) // Bottom Up calculation from j = 2 to n  
 {  
 useOdd = 1 - useOdd;  
  
 // Here we are explicitly writing lines for each number 0  
 // to 9. But it can always be written as DFS on 4X3 grid  
 // using row, column array valid moves  
 if(useOdd == 1)  
 {  
 even[0] = odd[0] + odd[8];  
 even[1] = odd[1] + odd[2] + odd[4];  
 even[2] = odd[2] + odd[1] + odd[3] + odd[5];  
 even[3] = odd[3] + odd[2] + odd[6];  
 even[4] = odd[4] + odd[1] + odd[5] + odd[7];  
 even[5] = odd[5] + odd[2] + odd[4] + odd[8] + odd[6];  
 even[6] = odd[6] + odd[3] + odd[5] + odd[9];  
 even[7] = odd[7] + odd[4] + odd[8];  
 even[8] = odd[8] + odd[0] + odd[5] + odd[7] + odd[9];  
 even[9] = odd[9] + odd[6] + odd[8];  
 }  
 else  
 {  
 odd[0] = even[0] + even[8];  
 odd[1] = even[1] + even[2] + even[4];  
 odd[2] = even[2] + even[1] + even[3] + even[5];  
 odd[3] = even[3] + even[2] + even[6];  
 odd[4] = even[4] + even[1] + even[5] + even[7];  
 odd[5] = even[5] + even[2] + even[4] + even[8] + even[6];  
 odd[6] = even[6] + even[3] + even[5] + even[9];  
 odd[7] = even[7] + even[4] + even[8];  
 odd[8] = even[8] + even[0] + even[5] + even[7] + even[9];  
 odd[9] = even[9] + even[6] + even[8];  
 }  
 }  
  
 // Get count of all possible numbers of length "n" starting  
 // with digit 0, 1, 2, ..., 9  
 totalCount = 0;  
 if(useOdd == 1)  
 {  
 for (i=0; i<=9; i++)  
 totalCount += even[i];  
 }  
 else  
 {  
 for (i=0; i<=9; i++)  
 totalCount += odd[i];  
 }  
 return totalCount;  
}  
  
// Driver program to test above function  
int main()  
{  
 char keypad[4][3] = {{'1','2','3'},  
 {'4','5','6'},  
 {'7','8','9'},  
 {'\*','0','#'}  
 };  
 printf("Count for numbers of length %d: %d\n", 1, getCount(keypad, 1));  
 printf("Count for numbers of length %d: %d\n", 2, getCount(keypad, 2));  
 printf("Count for numbers of length %d: %d\n", 3, getCount(keypad, 3));  
 printf("Count for numbers of length %d: %d\n", 4, getCount(keypad, 4));  
 printf("Count for numbers of length %d: %d\n", 5, getCount(keypad, 5));  
  
 return 0;  
}

Output:

Count for numbers of length 1: 10  
Count for numbers of length 2: 36  
Count for numbers of length 3: 138  
Count for numbers of length 4: 532  
Count for numbers of length 5: 2062

This article is contributed by **Anurag Singh**. 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/mobile-numeric-keypad-problem/>

Category: [Misc](http://www.geeksforgeeks.org/category/c-programs/) Tags: [Dynamic Programming](http://www.geeksforgeeks.org/tag/dynamic-programming/), [Matrix](http://www.geeksforgeeks.org/tag/matrix/)

Post navigation

[← Comparison of Autoboxed Integer objects in Java](http://www.geeksforgeeks.org/comparison-autoboxed-integer-objects-java/) [Arista Networks Interview | Set 3 →](http://www.geeksforgeeks.org/arista-networks-interview-set-3/)

# 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**](http://en.wikipedia.org/wiki/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(N2) times. Therefore the overall time complexity is O(N3).

A **better solution** is to use the [approach used for merging k sorted arrays](http://www.geeksforgeeks.org/merge-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(N2LogN).

// 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](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Matrix](http://www.geeksforgeeks.org/tag/matrix/)

Post navigation

[← Amazon interview Experience | Set 138 (For SDE 1)](http://www.geeksforgeeks.org/amazon-interview-experience-set-138-sde-1/) [Serialize and Deserialize a Binary Tree →](http://www.geeksforgeeks.org/serialize-deserialize-binary-tree/)

# 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](http://www.geeksforgeeks.org/print-a-given-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 C++ implementation of above idea. Thanks to Gaurav Ahirwar for suggesting below solution [here](http://qa.geeksforgeeks.org/917/rotate-the-matrix-elements/).

// 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;  
}

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

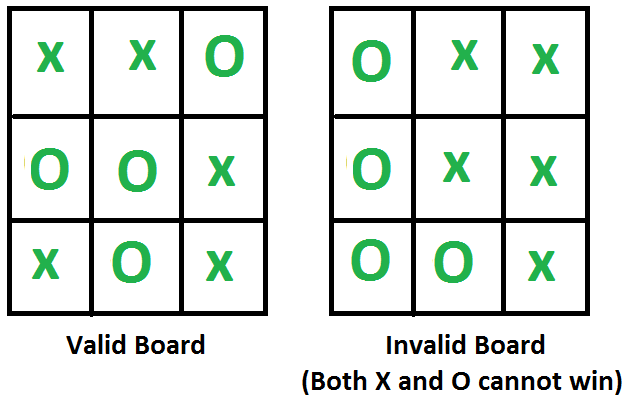
<http://www.geeksforgeeks.org/rotate-matrix-elements/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [Matrix](http://www.geeksforgeeks.org/tag/matrix/)

# 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.

[](http://d2dskowxfbo68o.cloudfront.net/wp-content/uploads/tictactoe.png)

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 [Utkarsh](http://qa.geeksforgeeks.org/user/utkarsh111)for suggesting this solution [here](http://qa.geeksforgeeks.org/1763/adobe-question-validity-of-tic-tac-toe-board-position?show=1763#q1763).

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](http://www.geeksforgeeks.org/category/c-programs/) Tags: [Matrix](http://www.geeksforgeeks.org/tag/matrix/)

Post navigation

[← How to make a website using WordPress (Part – 1)](http://www.geeksforgeeks.org/how-to-make-a-website-using-wordpress-part-1/) [SAP Labs Interview Experience | Set 10 (For Developer Specialist, 4-6 yrs) →](http://www.geeksforgeeks.org/sap-labs-interview-experience-set-10-for-developer-specialist-4-6-yrs/)