**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](http://www.geeksforgeeks.org/archives/23485) for suggesting the above solution.

Sources: [this](http://www.geeksforgeeks.org/forum/topic/amazons-question-please-suggest-best-approach) and [this](http://www.geeksforgeeks.org/archives/23485)

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;

}