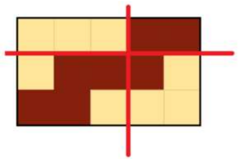
ABC 159E Dividing Chocolate

Given a $n \times m$ chocolate bar, in which each square is black or white. You can cut the bar horizontally or vertically, but you must cut all the way through. What is the minimum number of cuts so that every block has k or less white squares?

$1 \leq n \leq 10 \quad 1 \leq m \leq 1000 \quad 1 \leq k \leq n \times m$

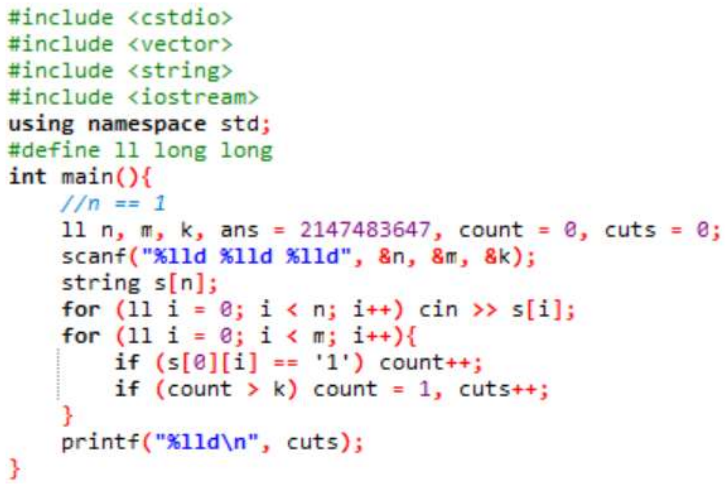
For example, the input below should be cut with 2 cuts:

```
3 5 4
11100
10001
00111
```



You can see that each block has $\leq k = 4$ white squares.

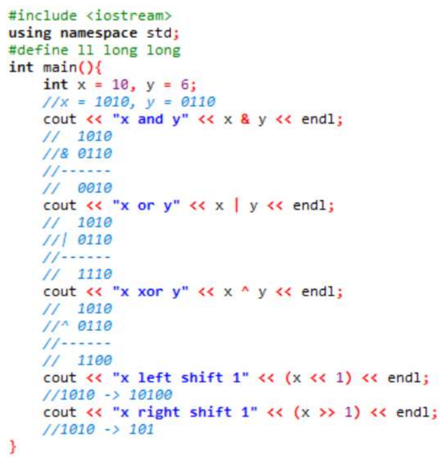
Firstly, let's consider the case where $n = 1$. We can use the greedy approach to solve this problem. We should always choose to cut where the length of the current block is maximized without having more than k white squares. If you choose to cut before that point, the number of white squares in the current block will always stay same or decrease. That means you left 0 or more white squares to future blocks when you could have dealt with them immediately. Therefore, greedy works in this case. This line of thinking can also be applied to other problems, where you think if doing stuff before is always suboptimal. If that's the case, you can always pick the latest time to do stuff, and that will be optimal.



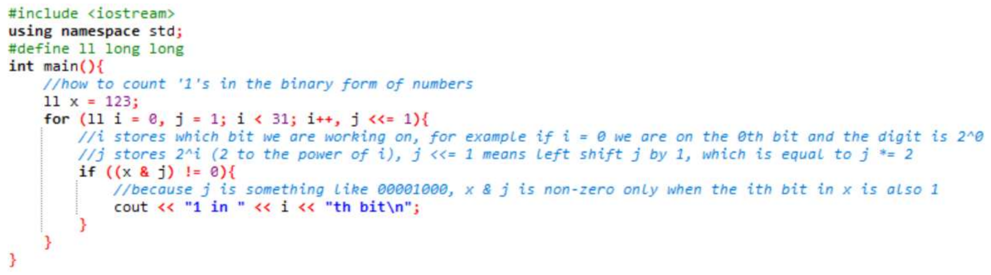
```
#include <cstdio>
#include <vector>
#include <string>
#include <iostream>
using namespace std;
#define ll long long
int main(){
    //n == 1
    ll n, m, k, ans = 2147483647, count = 0, cuts = 0;
    scanf("%lld %lld %lld", &n, &m, &k);
    string s[n];
    for (ll i = 0; i < n; i++) cin >> s[i];
    for (ll i = 0; i < m; i++){
        if (s[0][i] == '1') count++;
        if (count > k) count = 1, cuts++;
    }
    printf("%lld\n", cuts);
}
```

Now we have to consider the horizontal cuts. We observe that $n \leq 10$, so a brute force approach may pass. How do we brute force? We can see that there are $n - 1$ cutting points (as there are n rows), and for each cutting point, we can choose to cut or not cut. Therefore, there are 2^{n-1} ways to cut horizontally. Moreover, we can apply the same trick above, so we can obtain the minimum number of vertical cuts for each way to cut horizontally in $O(m)$. The total time is $O(2^{n-1}m)$, which should pass the time limit.

However, we now face a big problem: how do we implement this? Sure, we can use a recursive function to decide to cut or not to cut for each cutting point, but that would be annoying to code. Instead of that, we can use bitmask! Bitmask is a technique that involves manipulating bits and iterating though 2^n stuff. Let’s learn some bitwise operations:

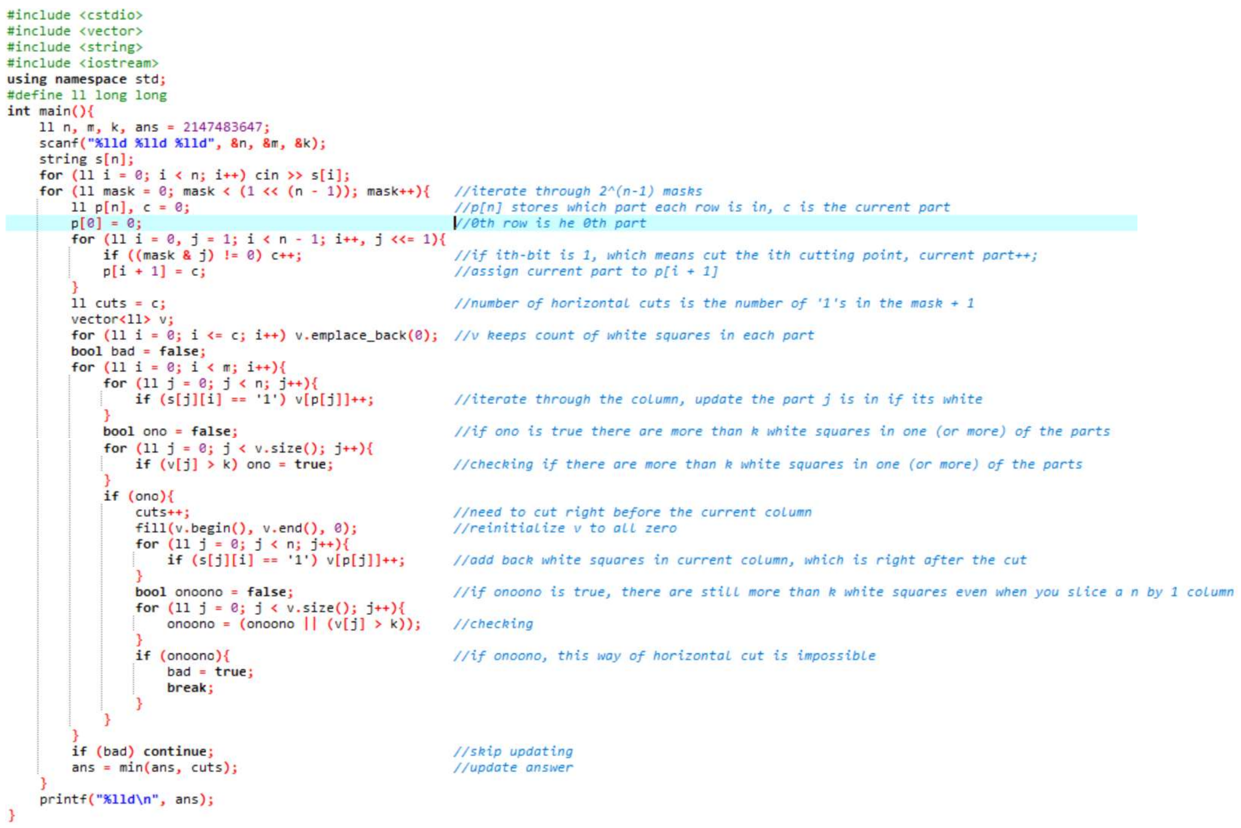


```cpp
#include <iostream>
using namespace std;
#define ll long long
int main(){
    int x = 10, y = 6;
    //x = 1010, y = 0110
    cout << "x and y" << x & y << endl;
    //  1010
    //& 0110
    //------
    //  0010
    cout << "x or y" << x | y << endl;
    //  1010
    //| 0110
    //------
    //  1110
    cout << "x xor y" << x ^ y << endl;
    //  1010
    //^ 0110
    //------
    //  1100
    cout << "x left shift 1" << (x << 1) << endl;
    //1010 -> 10100
    cout << "x right shift 1" << (x >> 1) << endl;
    //1010 -> 101
}
```



```cpp
#include <iostream>
using namespace std;
#define ll long long
int main(){
    //how to count '1's in the binary form of numbers
    ll x = 123;
    for (ll i = 0, j = 1; i < 31; i++, j <<= 1){
        //i stores which bit we are working on, for example if i = 0 we are on the 0th bit and the digit is 2^0
        //j stores 2^i (2 to the power of i), j <<= 1 means left shift j by 1, which is equal to j *= 2
        if ((x & j) != 0){
            //because j is something like 00001000, x & j is non-zero only when the ith bit in x is also 1
            cout << "1 in " << i << "th bit\n";
        }
    }
}
```

Implementation:



```cpp
#include <cstdio>
#include <vector>
#include <string>
#include <iostream>
using namespace std;
#define ll long long
int main(){
    ll n, m, k, ans = 2147483647;
    scanf("%lld %lld %lld", &n, &m, &k);
    string s[n];
    for (ll i = 0; i < n; i++) cin >> s[i];
    for (ll mask = 0; mask < (1 << (n - 1)); mask++){   //iterate through 2^(n-1) masks
        ll p[n], c = 0;                                 //p[n] stores which part each row is in, c is the current part
        p[0] = 0;                                       //0th row is he 0th part
        for (ll i = 0, j = 1; i < n - 1; i++, j <<= 1){
            if ((mask & j) != 0) c++;                   //if ith-bit is 1, which means cut the ith cutting point, current part++;
            p[i + 1] = c;                               //assign current part to p[i + 1]
        }
        ll cuts = c;                                    //number of horizontal cuts is the number of '1's in the mask + 1
        vector<ll> v;
        for (ll i = 0; i <= c; i++) v.emplace_back(0);  //v keeps count of white squares in each part
        bool bad = false;
        for (ll i = 0; i < m; i++){
            for (ll j = 0; j < n; j++){
                if (s[j][i] == '1') v[p[j]]++;          //iterate through the column, update the part j is in if its white
            }
            bool ono = false;                           //if ono is true there are more than k white squares in one (or more) of the parts
            for (ll j = 0; j < v.size(); j++){
                if (v[j] > k) ono = true;               //checking if there are more than k white squares in one (or more) of the parts
            }
            if (ono){
                cuts++;                                 //need to cut right before the current column
                fill(v.begin(), v.end(), 0);            //reinitialize v to all zero
                for (ll j = 0; j < n; j++){
                    if (s[j][i] == '1') v[p[j]]++;      //add back white squares in current column, which is right after the cut
                }
                bool onoono = false;                    //if onoono is true, there are still more than k white squares even when you slice a n by 1 column
                for (ll j = 0; j < v.size(); j++){
                    onoono = (onoono || (v[j] > k));    //checking
                }
                if (onoono){                            //if onoono, this way of horizontal cut is impossible
                    bad = true;
                    break;
                }
            }
        }
        if (bad) continue;                              //skip updating
        ans = min(ans, cuts);                           //update answer
    }
    printf("%lld\n", ans);
}
```