What is bitmask?

Bitmask is a technique that replaces Boolean array with integers. By doing so, it will be much more convenient to code, and it will also be faster (a little bit).

It is used when you want to iterate through 2^n things, such as states of n on / off switches.

In addition to convenience, it is also very important as you need to learn bitmask before learning bitmask DP. (actually just DP but with a bitmask in the state, kind of like tetrisudoku)

We all know that integers are represented in binary form in computers. For example, the integer $123 = 1111011_2$. You can see that this is very similar to a Boolean array. Therefore, in theory, if your Boolean array size is ≤ 32, you can use an integer to replace the array.

Bitwise operations

~ (bitwise not)

A unary operator, it will flip the bits. For example, 10111000 → 01000111

rarely used, also the last bit in an int is used to store the sign (+/-), so if you ~ a positive integer, it will become negative.

& (bitwise and)

resulting bit is 1 only if both bit is 1. 10110110 & 00101101 → 00100100

| (bitwise or)

resulting bit is 0 only if both bit is 0. 10110110 & 00101101 → 10111111

^ (bitwise xor)

resulting bit is 1 if exactly one of them is 1.

10110110 ^ 00101101 → 10011011

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

<< (shift left)

shift the whole thing left by adding a zero at the front

5 << 2 = 20 (5 = …000101, 20 = …010100)

>> (shift right)

shift the whole thing right by deleting the first bit, add zero at the back

5 >> 1 = 2 (5 = …0101, 2 = …0010)

4 >> 1 = 2 (4 = …0100, 2 = …0010)

you can also do &=, |= and ^=

Tricks

If utilized cleverly, the bitwise operators can do cool tricks! (also maybe a little bit faster because all bitwise operations are $O(1)$)

- a << b and a >> b is the same with * pow(2, b) and / pow(2, b) respectively. (not hard to see why) Next time you want to use 2^n in your code, just write (1 << n)!
  - to avoid overflow, write (1ll << n)
- can use bitwise and to check even / odd
  - if ((a & 1) == 1) cout << "odd";
- set ith bit to 1
  - a |= (1 << i)
- toggle ith bit
  - a ^= (1 << i)
- check ith bit
  - b = a & (1 << i)

We can also output the binary form as follows:

```
for (int i = 31; i >= 0; i--){
    if ((a & (1 << i)) != 0){
        cout << "1";
    }else{
        cout << "0";
    }
}
```

https://www.geeksforgeeks.org/bitwise-hacks-for-competitive-programming/

https://codeforwin.org/2018/05/10-cool-bitwise-operator-hacks-and-tricks.html

https://www.geeksforgeeks.org/bit-tricks-competitive-programming/

https://www.quora.com/How-do-I-understand-bitwise-tricks-in-C++-for-competitive-programming

Problem 1: bitwise operations

Initially, we have a sequence {1}. For each step, we will copy the sequence and append it to itself, and insert the minimum positive integer not in the sequence into the middle. After repeating this n times, what is the kth integer in the sequence? (n <= 50)

For example, if n = 4 and k = 8

{1} → {1, 2, 1} → {1, 2, 1, 3, 1, 2, 1} → {1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1}

k = 8th number is 4

This feels very similar to the binary numbers. Binary numbers can also be "generated" by making a copy, and add 1 to the second half. Also, the length of the sequence after i steps is $2^{i+1} - 1$. Let's try to compare them side by side. Hmm. What if we add a zero?

0	00	000	1	000	0	000
1	01	001	2	001	1	001
	10	010	1	010	2	010
	11	011	3	011	1	011
		100	1	100	3	100
		101	2	101	1	101
		110	1	110	2	110
		111	4	111	1	111
"generating" binary numbers			side by side 1		side by side 2	

Wow! We can see that in rightmost list, the numbers in the sequence corresponds to the position of the first '1' in the binary form of k!

To find the first '1', we can just loop through the bits of k starting from the 0th bit.

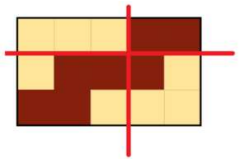
Problem 2: bitmask + greedy

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.

```cpp
#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.

Problem 3: bitmask + graph

Given an undirected graph with n vertices and m edges, in an operation, you can select a vertex, and that will toggle the neighbours of that vertex and itself. (from 0 to 1, from 1 to 0) Initially all vertices is 0. Find the minimum number of operations to make all vertices 1.

(n <= 25)

Observation 1: changing the order doesn't change the result. If you do an operation on a, then on b, the result is the same as doing an operation on b first, then doing on a.

Observation 2: each vertex only needs to be toggled at most once. If you do it twice, the result will be the same as doing nothing.

From the above observations, we can see that we can brute force all 2^n possible ways to do operations on the vertices.

pseudocode 1:

```
for (mask = 0 ~ 2^n - 1){
    initialize as all 0
    for (i = 0 ~ n - 1){
        if (ith bit in mask is 1){
            for (j = 0 ~ adj[i].size() - 1){
                toggle adj[i][j]
            }
            toggle i
        }
    }
    check if all 1, update answer if it is
}
//vector<int> adj[n] is the adjacency list of the graph
//adj[i] contains the neighbours of i
```

However, this is $O(n^2 2^n)$ and may TLE! Can we do better?

Since n is so small, we can store the adjacency matrix using an array of bitmasks! This way, we can toggle all the neighbours and itself in $O(1)$!

pseudocode 2:

```
for (mask = 0 ~ 2^n - 1){
    state = 0;
    for (i = 0 ~ n - 1){
        if (ith bit in mask is 1){
            state ^= adj[i]; //toggle all neighbours
            state ^= 1 << i; //toggle itself
        }
    }
    check if all 1, update answer if it is
}
//int adj[n] is the adjacency matrix of the graph
//jth bit in adj[i] is 1 if theres an edge between i and j
```

Wow! Now it is $O(n2^n)$.

Problem 1: https://codeforces.com/contest/743/problem/B

Code: https://pastebin.com/PE0GiJS2

Problem 2: https://atcoder.jp/contests/abc159/tasks/abc159_e

Code: https://pastebin.com/SR2Bz605

Problem 3: https://www.luogu.com.cn/problem/P2962

(the version I talked about is an easier version of the problem)

Code: https://pastebin.com/sn56Jb0v (for the easier version)

You can ask me if you wanna know how to solve the harder version 😉

Some HKOJ tasks you can do:

M1031 Camping

M0433 HKOI Judge