Pizza: solutions

The Bitmasks

Motivation

Suppose that you want to select a subset of items from another set.

A way to do that is to make a list of booleans, that indicates if that item is chosen or not.

Another way, is to use the bitmaks or masks.

what's a bitmask ?

Every information is represented in binary. This binary representation could act as a sequence of boolean values when is compared with another binary representation using the biwise operators (&, |, ^, <<, >>, ~).

For example, from a binary representation we can know if an (some) object(s) is(are) chosen or not. The least significant bit (LSB) indicates if the 1^{st} object is chosen or not, the second bit indicates if the 2^{nd} object is chosen or not, etc.

From a set of 5 objects, if the 1^{st}, the 3^{rd} and the 4^{th} objects are chosen, the bitmask will be: 01101

Some Manipulations of bitmasks

Example #1 : know the letter's case

Binary representation of uppercase letters:

letter	binary
'A'	0100 0001
'B'	0100 0010
'Z'	0101 1010

Binary representation of lowercase letters:

letter	binary
'a'	0110 0001
'b'	0110 0010
'z'	0111 1010

The 5th bit of the binary representation of uppercase letters is alaways 0, and is equal to 1 in binary representation of lowercase letters.

Making a bitwise and (&) with the mask 100000 (32 in decimal), gives:

- 0 if uppercase letter.
- ≠ 0 if lowercase letter.

	‘Z’	01011010			‘z’	01111010
&	bitmask	00100000		&	bitmask	00100000
	0	0000000			Non-0	00100000

Code C++ :

```
#include <bits/stdc++.h>

using namespace std;

int main() {
  char c;
  cin >> c;

  //if ( (c & 32) == 0)
  //if ( (c & 0x20) == 0)
  if ( (c & 0b00100000) == 0)
    cout << "Uppercase.\n";
  else
    cout << "Lowercase.\n";
  return 0;
}
```

Example #2 : convert the letter case

Using the bitwise *(^)* and the bitmask 00100000 on the letter :

	‘Z’	01011010			‘z’	01111010
^	bitmask	00100000		^	bitmask	00100000
	‘z’	01111010			‘Z’	01011010

Example #3 : multiply an integer times 2

Apply an left shifting (<<) of 1.

13 = 00001101
13 << 1 = 00001101 << 1 = 00011010 = 26.

Example #4 : Divide an integer over 2 (the quotient)

Apply an right shifting (>>) of 1.
17 / 2 = 17 >> 1 = 00010001 >> 1 = 00001000 = 8

Example #5 : the number of subset that could be obtained from a set of n items

the number of subset that could be obtained from a set of n items is 2^n = **1<< n**.

Example #6 : Add the j^{th} item to a subset *A*

Change the j^{th} bit to 1.
Principle :
1- Apply a left shifting of j bits of the integer 1.
2- The new content of A is the old content of A bitwise-or the result obtained in 1.
Formula: A |= (1 << j)

Example : adding the item 6 to a subset A = {1,4}
A = 00010010
1 << 6 = 00000001 << 6 = 01000000
A | 01000000 = 00010010 | 01000000 = 0**1**010010
The subset *A* become: {1,4,6}

Example 7 : Delete the j^{th} item from a subset *A*

Change the j^{th} bit à 0
Principle :
1- 1 << j
2- Apply a bitwise not on the result obtained in 1.
3- The new content of A is the old content of A bitwise-and the result obtained in 2.
Formula: A &= ~(1 << j)

Example : delete the item 6 from A = {1,4,6}
A = 01010010
1 << 6 = 00000001 << 6 = 01000000
~01000000 = 10111111

A & 10111111 = 01010010 & 10111111 = 0**0**010010
The subset *A* become: {1,4}

Example 8 : Check if a j^{th} item is in a subset *A*

Check if the j^{th} bit is 1.
Principle :
1- 1 << j
2- make a bitwise-and between A and the result in 1.
Formula: check = A & (1 << j)
if check == 0, j doesn't exist in A.
if check != 0, j exists in A.

Example :
let a set of N integers, we want to make the sum of all formed subsets items.
Example: For the set {2,6,11}
the program display :
{2}: 2
{6}: 6
{2, 6}: 8
{11}: 11
{2, 11}: 13
{6, 11}: 17
{2, 6, 11}: 19

we have 2^3 subsets.

arr:	2	6	11
j:	0	1	0

Subsets i	
0	{}
1	{2}
2	{6}
3	{2, 6}
4	{11}
5	{2, 11}
6	{6, 11}
7	{2, 6, 11}

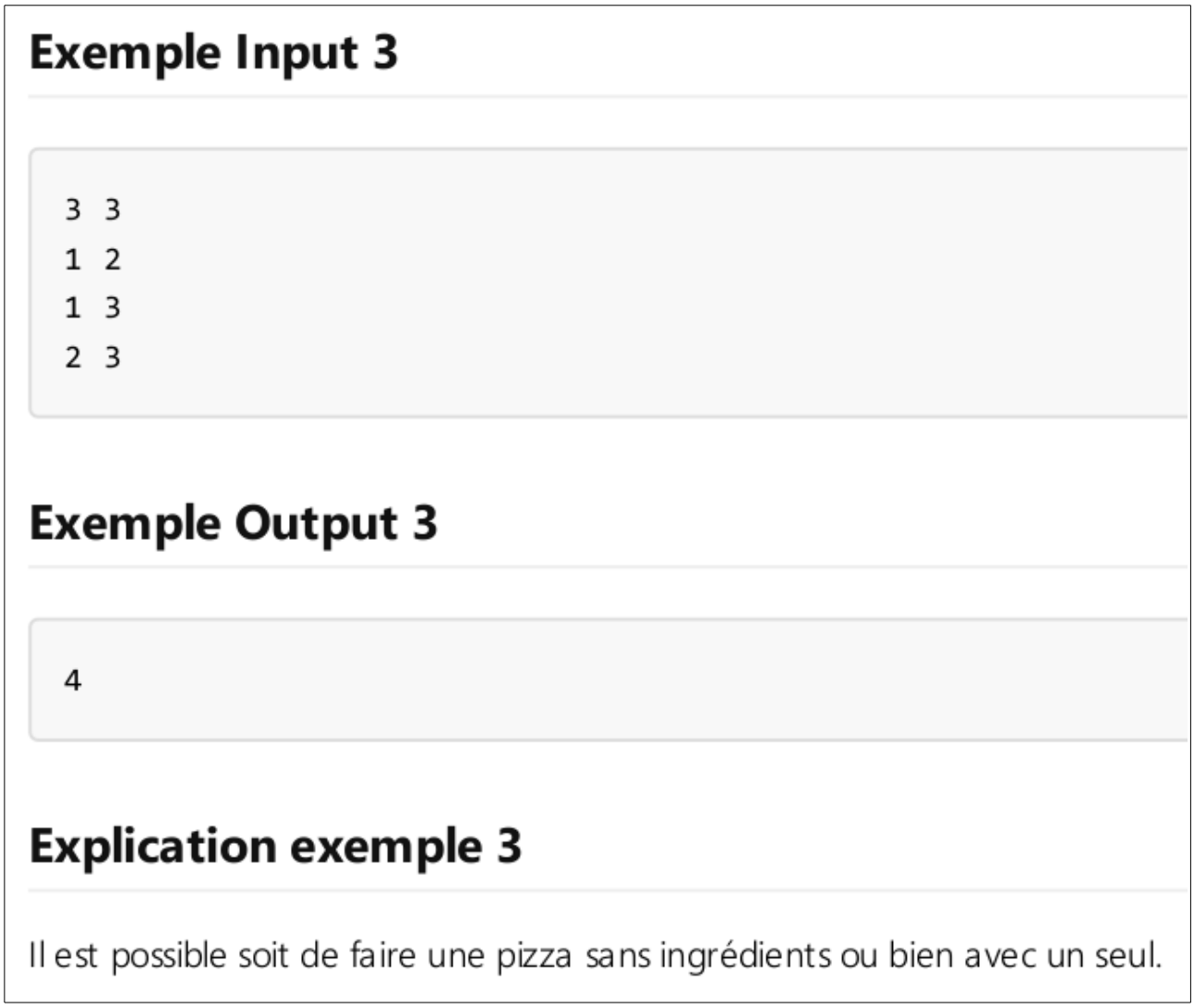
To do that, for each subset i, and for each position j, check if j exists in i. (Check table below)

i	*j*	*i* **& (1 <<** *j***)**		*sum*	*result*
0	0	0 & 1 = 0	2 ∉{}	0	
	1	00 & 10 = 0	6 ∉{}	0	
	2	000 & 100 = 0	11 ∉{}	0	
					{}: 0
1	0	1 & 1 != 0	**2 ∈ {2}**	2	
	1	01 & 10 = 0	6 ∉ {2}	0	
	2	001 & 100 = 0	11 ∉ {2}	0	
					{2} : 2
2	0	10 & 01 = 0	2 ∉ {6}	0	
	1	10 & 10 != 0	**6 ∈ {6}**	6	
	2	010 & 100 = 0	11 ∉ {6}	0	
					{6} : 6
3	0	11 & 01 != 0	2 ∈ {2, 6}	2	
	1	11 & 10 != 0	**6 ∈ {2,6}**	8	
	2	011 & 100 = 0	11 ∉ {2, 6}	8	
					{2,6} : 8
4	0	100 & 001 = 0	2 ∉ {11}	0	
	1	100 & 010 = 0	6 ∉ {11}	0	
	2	100 & 100 != 0	**11 ∈ {11}**	11	
					{11} : 11
5	0	101 & 001 != 0	2 ∈ {2, 11}	2	
	1	101 & 010 = 0	6 ∉ {2, 11}	2	
	2	101 & 100 != 0	11 ∈ {2, 11}	13	
					{2,11} : 13
6	0	110 & 001 = 0	2 ∉ {6, 11}	0	
	1	110 & 010 != 0	6 ∈ {6, 11}	6	
	2	110 & 100 != 0	**11 ∈ {6,11}**	17	
					{6,11} : 17
7	0	111 & 001 != 0	2 ∈ {2, 6, 11}	2	
	1	111 & 010 != 0	6∈ {2, 6, 11}	8	
	2	111 & 100 != 0	**11 ∈ {2, 6, 11}**	19	
					{2,6,11} : 19

Code C++ : O($2^n * n$)

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

void display(vector <int > v, int n) {
  for (auto vi: v)
    cout << vi << ' ';
  cout << '\n';
}

void fillArr (vector <int> & v, int n) {
  for (int i = 0 ; i < n ; ++i) {
    int vi;
    cin >> vi;
    v.push_back(vi);
  }
}

void allSubsets(vector <int> v, int n) {
  int numberSubsets = 1 << n;
  for (int i = 0 ; i < numberSubsets ; ++i) {
    int s = 0;
    cout << '{';
    for (int j = 0 ; j < n ; ++j) {
      if ( (i & (1 << j)) != 0 ) {
        s += v[j];
        cout << v[j] <<  ", ";
      }
    }
    cout <<'}';
    cout << ": " << s;
    cout << '\n';
  }
}
int main() {
  int n;
  cin >> n;

  vector <int> v;
  fillArr(v, n);
  cout << '\n';
  display(v, n);
  cout << '\n';
  allSubsets(v, n);

  return 0;
}
```

let’s take the sample #3



Exemple Input 3

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

Exemple Output 3

```
4
```

Explication exemple 3

Il est possible soit de faire une pizza sans ingrédients ou bien avec un seul.

Let’s call S the subset of ingredients : S = {1, 2, 3}

We can make a pizza :

- Without any ingredients from S : {}
- With one ingredient :
  - ✗ {1}
  - ✗ {2}
  - ✗ {3}

But not with:

- {1, 2}
- {2, 3}
- {1, 2, 3}

We can make 4 pizzas.

To compute the number of pizzas that we can make with the different combinations of ingredients that could be mixable together, with the respect of the constraints. We have to check if the ingredients a and b (those not to mix together) exist or not in the different sets of ingredients. If the pair (a, b) not exist, then we increment one to the number of pizzas.

Example :

S = {1, 2, 3}

Do not mix : {1, 2}, {1, 3} et {2, 3}

sets	**Pizzas**
	0
{}	1
{1}	2
{2}	3
{1, 2}	3
{3}	4
{1, 3}	4
{2, 3}	4
{1, 2, 3}	4

A solution is to store the not to be mixed ingredients in a vector of pairs:

$notMixed$:	{0, 1}	{0, 2}	{1, 2}
j:	*0*	*1*	*2*

For each set i, check if the a^{th} and the b^{th} bits are HIGH (equal to 1) in i.

i	Binary of i	Set i	j	notMixed[j].first and notMixed[j].second are both in the set I or not ?	Pizzas that we can make
					0
0	0000	{}	0	no	
			1	no	
			2	no	
					1
1	0001	{0}	0	no	
			1	no	
			2	no	
					2
2	0010	{1}	0	no	
			1	no	
			2	no	
					3
3	0011	{0, 1}	0	yes	3, add nothing and quit.
4	0100	{2}	0	no	
			1	no	
			2	no	
					4
5	0101	{0, 2}	0	no	
			1	yes	4, add nothing and quit.
					4
6	0110	{1, 2}	0	no	
			1	no	
			2	yes	4, add nothing and quit.
					4
7	0111	{0, 1, 2}	0	yes	4, nothing to add an quit
					4

Lang: C++
knowledge requirement: bitmaks
running time: O(2^N * M)

```
#include <bits/stdc++.h>

using namespace std;

vector<pair<int, int>> notMixed;

int main() {
  ios::sync_with_stdio(false);
  int N, M;
  cin >> N >> M;

  for (int i = 0 ; i < M ; ++i) {
    int a, b;
    cin >> a >> b;
    notMixed.push_back({--a, --b});
  }

  int numberSubsets = 1 << N;
  int ans = 0;
  for (int i = 0 ; i < numberSubsets ; ++i) {
    int toAdd = 1;
    for (int j = 0 ; j < M ; ++j) {
      if ( (i & (1 << notMixed[j].first)) && (i & (1 << notMixed[j].second)) ) {
        toAdd = 0;
        break;
      }
    }

    ans += toAdd;
  }

  cout << ans << '\n';

  return 0;
}
```