# Bitmasks

## Agenda

- **Bitwise Operations**
- **Iterative Complete Search**

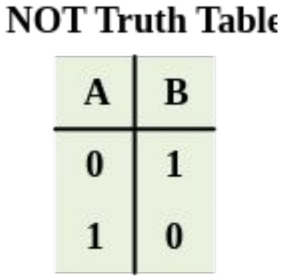
## Bitwise Operations

| | |
|---|---|
| NOT | ~ |
| OR | \| |
| AND | & |
| XOR | ^ |
| SHR | >> |
| SHL | << |

## NOT ~

A   = 12  (0000 0000 0000 1100)

~A = -13 (1111 1111 1111 0011)



NOT Truth Table

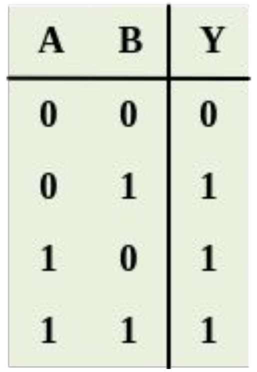
| A | B |
|---|---|
| 0 | 1 |
| 1 | 0 |

## OR |

A = 72 (0100 1000)
B = 184 (1011 1000)

A | B = 248 (1111 1000)

OR Truth Table

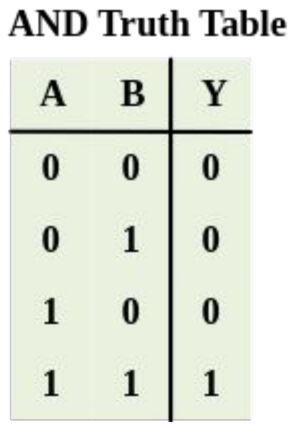


| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

## AND &

A = 72 (0100 1000)
B = 184 (1011 1000)

A & B = 8 (0000 1000)

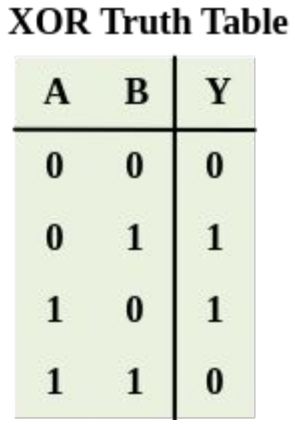

AND Truth Table

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

# XOR ^

A   = 72  (0100 1000)
B  = 184 (1011 1000)

A ^ B = 240 (1111 0000)



XOR Truth Table

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## SHR >>

It's considered as division by power of 2s

| | | |
|---|---|---|
| A | = | 5 (00101) |
| A>>1 | = | 2 (00010) |
| | | |
| B | = | 12(01100) |
| B>>2 | = | 3 (00011) |

## SHL <<

It's considered as multiplying by power of 2s

| | | |
|---|---|---|
| A | = | 5 (00101) |
| A<<1 | = | 10 (01010) |
| | | |
| B | = | 12(001100) |
| B<<2 | = | 48 (110000) |

## Bit Masking

- **32 int**: bits take indices from 0 to 31
- **64 int**: bits take indices from 0 to 63
- Check a bit

```
(mask >> bitIndx) & 1 == 1;
```

- Set a bit to one

```
mask = mask | (1 << bitIndx);
```

- Set a bit to zero

```
mask = mask & ~(1 << bitIndx);
```

## Bit Masking

- Flip a bit

```
mask = mask ^ (1 << bitIndx);
```

- Check if a number if odd `num & 1 == 1;`
- Xor tricks

```
x ^ y ^ x = y

x ^ y false when (x == y)
```

- Number of ones

```
__builtin_popcount(num);
__builtin_popcountll(num);
```

## Example

- Given a positive integer, find if it is a power of two or not

```
int num, cnt=0;
cin>>num;
for(int i=0; (1LL<<i) <= num; i++){
    if((num>>i)&1)
        cnt++;
}
cout<<((cnt==1) ? "YES" : "NO")<<'\n';
```

## Iterative Complete Search

- Complete search (aka brute force) is a method for solving a problem by traversing the entire search space in search of a solution.
- A simple example for that is printing all numbers  between 1 and 100 which is divisible by 5. The brute force solution is to try each number between 1 and 100 and check if it’s divisible by 5 or not.
- Sometimes, the search space is not that easy to be implemented in one single for loop.

## Problem

Let’s say you have some **unique** numbers and a target sum, and you want to get the number of subsets that sum up to the target.

**Numbers**: 1 , 5 , 2 , 7 , 3 , 9

**Target**: 6

**Number of valid subsets**: 2

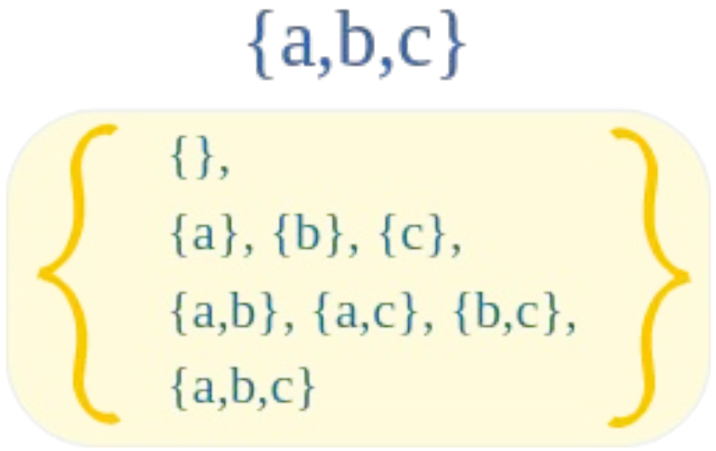
{1,5}, {1,2,3}

But how to generate all different subsets ?

## Solution

- The solution is to implement a **powerset** of the numbers

```
for(int mask=0; mask < (1<<n); mask++){
    int sum=0;
    for(int i=0; i<n; i++)
        if((mask>>i)&1)
            sum+=nums[i];
    if(sum==target)
        counter++;
    }
}
```



## To Solve

- Raising Bacteria
- Sum It Up