02. Bitwise Operation

2019 Summer / 20141574 임지환

비트마스크(Bitmasking)

- 정수의 이진수 표현을 자료구조로 쓰는 기법

- 이진수 형태의 비트연산을 통한 여러 이점 有

비트연산자

- & : AND
- | : OR
- ~ : NOT
- ^ : XOR
- >> : shift right
- << : shift left

비트연산자

- **& : AND**
- | : OR
- ~ : NOT
- ^ : XOR
- <<: shift left
- >>: shift right

- 두 비트가 모두 1일 경우 return 1

```
0&0 = 0&1 = 1&0 = 0
1&1 = 1
x&0 = 0
x&1 = x
```

```
  11001010
& 01101001
----------
  01001000
```

비트연산자

- & : AND
- | : **OR**
- ~ : NOT
- ^ : XOR
- <<: shift left
- >>: shift right

- 두 비트가 모두 0일 경우 return 0

```
0|1 = 1|0 = 1|1 = 1
0|0 = 0
x|0 = x
x|1 = 1
```

```
  11001010
| 01101001
----------
  11101011
```

비트연산자

- & : AND
- | : OR
- ~ : **NOT**
- ^ : XOR
- <<: shift left
- >>: shift right

- 비트를 반전

```
~0 = 1
~1 = 0
~3 =
```

```
 ~ 01101001
----------
  10010110
```

비트연산자

- & : AND
- | : OR
- **~ : NOT**
- ^ : XOR
- <<: shift left
- >>: shift right

- 비트를 반전

```
~0b0 = 1
~0b1 = 0
~3 = -4
```

```
 ~ 01101001
----------
  10010110
```

비트연산자

- & : AND
- | : OR
- ~ : NOT
- ^ : **XOR**
- <<: shift left
- >>: shift right

- 두 비트가 서로 다를 경우 return 1

```
0^1 = 1
0^0 = 1^1 = 0
x^0 = x
x^1 = ~x
```

```
  11001010
^ 01101001
----------
  10100011
```

비트연산자

- & : AND
- | : OR
- ~ : NOT
- ^ : XOR
- **<<: shift left**
- >>: shift right

- 주어진 범위내의 모든 비트를 왼쪽으로 이동

`0b1<<3 = 0b1000=2^3`

`(2^30)<<1 =`

```
   11001010
<<         2
-----------
   00101000
```

비트연산자

- & : AND
- | : OR
- ~ : NOT
- ^ : XOR
- **<<: shift left**
- >>: shift right

- 주어진 범위내의 모든 비트를 왼쪽으로 이동

`0b1<<3 = 0b1000=2^3`

`(2^30)<<1 = -2147483648 = -2^31`

```
   11001010
<<         2
----------
   00101000
```

비트연산자

- & : AND
- | : OR
- ~ : NOT
- ^ : XOR
- <<: shift left
- **>>: shift right**

- 주어진 범위내의 모든 비트를 오른쪽으로 이동

`0b1000<<2 = 0b0010=2`^{1}

```
   11001010
>>        2
-----------
   00101000
```

자주 사용되는 Bit Operations

n번째 비트가 켜져 있는지 확인

- `if (x & (1<<n))`

```
  11001010        11001010
& 01000000      & 00010000
----------      ----------
  01000000        00000000
```

자주 사용되는 Bit Operations

n번째 비트가 켜져 있는지 확인

- `if (x & (1<<n))`

주의!!! ~~`if (x & (1<<n) == 1)`~~

```
  11001010        11001010
& 01000000      & 00010000
----------      ----------
  01000000        00000000
```

자주 사용되는 Bit Operations

n번째 비트**만** 켜져 있는지 확인

- `if (!(x & ~(1<<n)))`

```
  11001010        00010000
& 10111111      & 11101111
----------      ----------
  10001010        00000000
```

자주 사용되는 Bit Operations

n번째 비트 켜기

- **x |= (1<<n)**

```
  11001010          11001010
| 00010000        | 00001000
----------        ----------
  11011010          11001010
```

자주 사용되는 Bit Operations

n번째 비트 끄기

- **x &= ~(1<<n)**

```
   11001010          11001010
 & 11110111        & 11101111
 ----------        ----------
   11000010          11001010
```

자주 사용되는 Bit Operations

n개의 비트 모두 켜기

- **x = (1<<n) - 1**

자주 하는 실수

Overflow

```
1<<62 = ?

(1<<31)<<1 = ?
```

자주 하는 실수

Overflow

`1<<62 = ?`

`(1<<31)<<1 = ?`

`1LL<<62`

자주 하는 실수

연산자 우선순위

See reference

#11723 집합

비어있는 공집합 S가 주어졌을 때, 아래 연산을 수행하는 프로그램을 작성하시오.

- `add x` : S에 x를 추가한다. (1 ≤ x ≤ 20) S에 x가 이미 있는 경우에는 연산을 무시한다.
- `remove x` : S에서 x를 제거한다. (1 ≤ x ≤ 20) S에 x가 없는 경우에는 연산을 무시한다.
- `check x` : S에 x가 있으면 1을, 없으면 0을 출력한다.
- `toggle x` : S에 x가 있으면 x를 제거하고, 없으면 x를 추가한다. (1 ≤ x ≤ 20)
- `all` : S를 {1, 2, ..., 20} 으로 바꾼다.
- `empty` : S를 공집합으로 바꾼다.

입력

첫째 줄에 수행해야 하는 연산의 수 M (1 ≤ M ≤ 3,000,000)이 주어진다.

둘째 줄부터 M개의 줄에 수행해야 하는 연산이 한 줄에 하나씩 주어진다.

#11723 집합

- 사실 비트연산을 안써도 풀립니다.
- 그래도 연습해봅시다.
- 연산을 문자열로 받는 부분에서 시간초과가 날 수 있으니 주의합시다.

#10464 XOR

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1 초	256 MB	237	120	84	73.043%

문제

*S*에서 *F*까지의 모든 정수를 XOR한 값은 무엇일까?

입력

입력의 첫 번째 줄에는 테스트 케이스의 개수 T($1 \le T \le 1000$)가 주어진다.

다음 T 개의 줄에는 두 개의 정수 S와 F가 주어진다. ($1 \le S \le F \le 1\,000\,000\,000$)

#10464 XOR

- 시작지점 ~ 끝지점의 길이가 최대 10^9만큼 나올 수 있다.

#10464 XOR

- 일단 전부 써볼까?

00000000
00000001
00000010
00000011
00000100
00000101
00000110
00000111
00001000
00001001
00001010
00001011
00001100
00001101
00001110
00001111
...

#10464 XOR

- 일단 전부 써볼까?

00000000
00000001
00000010
00000011
00000100
00000101
00000110
00000111
00001000
00001001
00001010
00001011
00001100
00001101
00001110
00001111
...

#10464 XOR

- 일단 전부 써볼까?

00000000
00000001
00000010
00000011
00000100
00000101
00000110
00000111
00001000
00001001
00001010
00001011
00001100
00001101
00001110
00001111
...

#10464 XOR

- 일단 전부 써볼까?

00000000
00000001
00000010
00000011
00000100
00000101
00000110
00000111
00001000
00001001
00001010
00001011
00001100
00001101
00001110
00001111
...

#10464 XOR

함수 하나를 다음과 같이 정의한다.

`func(n)` := 1~n까지의 모든 수를 XOR한 결과

S~F에서의 XOR을 한 결과 : `func(F) ^ func(S-1)`

#10464 XOR

```
int f(int n) {
    int ret = 0;
    for (int i = (n / 4) * 4; i <= n; i++)
        ret ^= i;
    return ret;
}
```

https://raararaara.blog.me/221557560377
Full Source Code : http://boj.kr/96c1b4bbc44847d6a814656f9319204d

#2098 외판원 순회

- Travelling Salesman Problem

https://www.acmicpc.net/problem/2098

#2098 외판원 순회

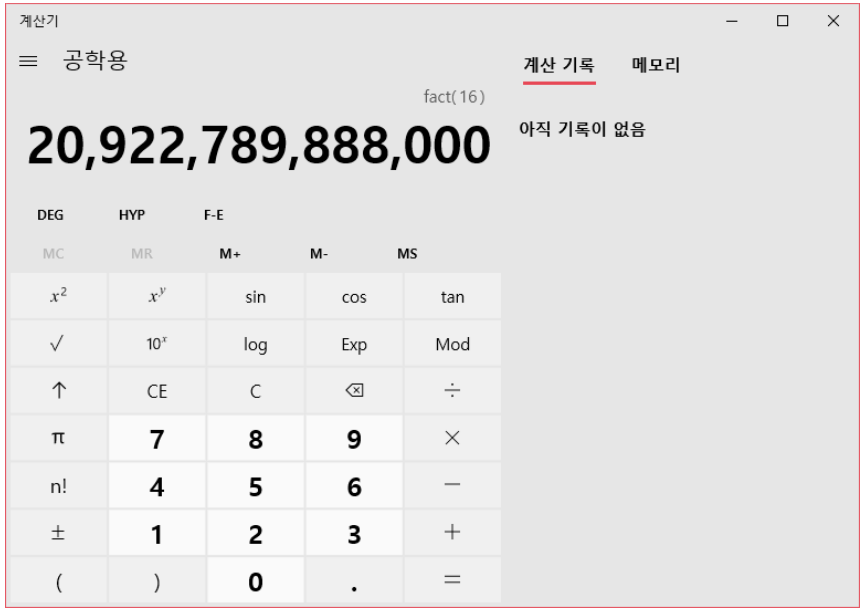
- 모든 경우의 수를 전부 만들어본 후 비교한다면?

#2098 외판원 순회

- 모든 경우의 수를 전부 만들어본 후 비교한다면?

$$O(n!)$$

n = 16, 16! =



#2098 외판원 순회

- 모든 지점 방문 후 시작 지점으로 돌아와야 한다.

  ⇒ **어디에서 시작해도 무방하다. (0으로 고정)**

- 어느 곳에 방문을 했는지를 저장할 필요가 있다.

  ⇒ 지점이 16개 이하이므로 방문여부에 대한 state를 저장할 수 있겠다.

  ⇒ **dist[visit][last]**

  visit : 지금까지 방문한 정점 정보

  last : 마지막으로 방문한 정점

#2098 외판원 순회

`dist[visit][last]`

`visit`의 모든 비트를 체크해서, i번째 비트가 꺼져 있다면

`last`번째 노드에서 i번 노드로 가는 거리를 가산

`visit`의 i번째 비트를 켜고 다음 지점으로 (다음 재귀 함수 실행)

#2098 외판원 순회

- Time Complexity

방문여부를 저장하는 state의 경우의 수는 **1<<n=2^n**

마지막 방문지점의 개수는 **n**개

각 지점에서 다른 지점으로 방문할지를 체크하는 횟수 : **n**개

$$O(2^n n^2)$$

#2098 외판원 순회

```
int solve(int vis, int cur) {
    if (vis == (1 << n) - 1) return w[cur][0];
    int&ret = dp[vis][cur];
    if (ret != -1) return ret;
    ret = INF;
    for (int next = 0; next < n; next++) {
        if (vis & (1 << next)) continue;
        ret = min(ret, solve(vis | (1 << next), next) + w[cur][next]);
    }

    return ret;
}
```

Full Source Code : http://boj.kr/2f030234c482465b86f80589c94cb50f

Problem Set

- 5 11723 집합
- 2 15787 기차가 어둠을 헤치고 은하수를
- 4 10275 Gold Rush
- 2 1194 달이 차오른다, 가자.
- 1 2098 외판원 순회
- 5 1562 계단수
- 1 1102 발전소
- ? 10464 XOR
- 5 11191 XOR Maximization
- 4 16685 XOR 포커
- 5 13505 두 수 XOR
- 3 12844 XOR

Last week's Solution

- 3 11659 구간 합 구하기 4
- 2 11660 구간 합 구하기 5
- 3 11969 Breed Counting
- 1 KickStart Ro.B Building Palindromes
- 1 1806 부분합
- 1 CF 1196D2 RGB Substring
- 3 CF 1132C Painting the Fence
- 1 15589 Lifeguards (Silver)
- 1 CF 1076E Vasya and a Tree
- 4 12745 Traffic (Small)
- 4 12746 Traffic (Large)

Hints for Given Problem

다음 장에는 스포일러를 포함하고 있습니다.

#15787 기차가 어둠을 헤치고 은하수를 

- 전체적인 구성은 #11723 집합과 차이가 없을 것.
- 문제가 생길만한 부분은 2가지가 있는데
  1. **이미 목록에 존재하는 기록이 있는지**

     ⇒ set 등의 자료구조를 활용하여 중복 확인
  2. **한칸씩 이동하는 부분에서 문제가 있을 수도.**

     ⇒ 20번째 앉은 사람이 이동하면 21번째로 가는가, 혹은 없어지는가?

#10275 Gold Rush

- 브론즈 문제가 맞습니다. 다들 가능하시리라 생각합니다.

http://raararaara.blog.me/221522009449

#1194 달이 차오른다, 가자. 

- BFS긴 한데, state가 하나가 더 추가가 되어야 한다. = 열쇠 소지여부.

```
struct P {
    int r, c, keylist;
};
queue<P> q;

int vis[50][50][1 << 7];
```

#1562 계단수

```
lint dp[current][last_num][visited];
```

#1102 발전소

- TSP보다 쉬운 문제입니다. TSP를 잘 이해하고 다시 풀어보세요.

#11191 XOR Maximization

- 선형대수학에서
    1. Vector Space에서의 basis
    2. Homogeneous linear equation

에 대한 이해가 필요합니다.

https://raararaara.blog.me/221511605448

#16685 XOR 포커

- 전체적인 아이디어는 XOR Maximization과 같습니다.
- 앞의 문제를 풀고 왔다면 "짝수 개 선택" 이라는 부분만 문제가 될겁니다.

https://raararaara.blog.me/221513539119

#13505 두 수 XOR

- 트라이(Trie) 에 대한 지식이 필요합니다.
- 이 문제에서는 0과 1만 등장하므로 이진트라이를 사용합니다.

https://raararaara.blog.me/221510731570

- 뭔가 다른 방식으로 풀어보려는 시도를 했었는데, 한번 맞았었는데 재채점 이후 오답처리가 되었습니다. 다른 방법 알아내시면 연락주세요.

#12844 XOR

- 세그먼트 트리(Segment Tree)에 대한 지식이 필요합니다.

  ⇒ 19-1 고급 8주차

- Lazy Propagation에 대한 지식이 필요합니다.

  ⇒19-S 고급 1주차