**1. 기초 강의**

동영상 강의 컨텐츠 확인 > 1. 비트연산  
Link : <https://swexpertacademy.com/main/learn/course/subjectDetail.do?courseId=CONTENTS_REVIEW&subjectId=AYVXaAXqQRcDFARs>  
**※ 출석은 강의 수강 내역으로 확인합니다**.

**2. 비트 연산**

**2.1 비트(bit)**

컴퓨터에서 자료를 표현하기 위해 비트를 사용합니다.

1bit = 0 또는 1

8bits = 1byte

**2.2 비트 연산자**

다음은 6가지 비트 연산자입니다.

|  |  |  |
| --- | --- | --- |
| 비트 연산자 | | a = 0b1010, b = 0b0100 |
| & | AND | a & b = 0b0000 |
| | | OR | a | b = 0b1110 |
| ^ | XOR | a ^ b = 0b1110 |
| ~ | NOT | ~a = 0b0101 |
| << | 왼쪽 Shift | a << n = a \* 2n |
| >> | 오른쪽 Shift | a >> n = a \* 2-n |

비트 연산의 우선순위에 주의가 필요합니다. 일반적으로 사용하는 사칙연산 +, -, \*, /은 비교, 논리 연산자(==, >, && 등)보다 우선순위가 높습니다. 하지만 비트 연산은 논리 연산보다 우선순위가 높으나 **비교 연산보단** **낮습니다**.

따라서 아래 코드는 주의가 필요합니다.

if (x & y == 0) // if (x & (y == 0)) 과 같음!

**2.3 비트 연산 응용**

**& AND, | OR**

◆ 비트 집합 두 개를 AND하면 교집합, OR하면 합집합을 구할 수 있습니다. (**2.4** 참고)

**^ XOR**

◆ true/false를 번갈아 바꾸는 스위치를 구현할 수 있습니다

◆ 어떤 수에서 몇 개의 bit를 바꿔서 대응되는 수를 구할 수 있습니다. 대표적인 예로 ASCII 코드가 있습니다. ASCII 코드에서 짝이 맞는 문자끼리 다른 bit를 XOR시키는 기법을 이용한 대소문자 변환 함수입니다.

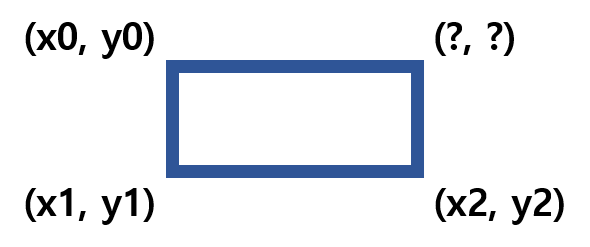
char case\_convert(char alphabet) {

return alphabet ^ 32;

}

◆ 같은 값끼리 XOR하면 0이 되는 특징은 많은 곳에 적용할 수 있습니다.

각 변이 x축 또는 y축에 평행한 직사각형이 있습니다. 이 직사각형의 세 꼭지점의 좌표가 주어졌을 때 남은 한 꼭지점의 좌표는 어떻게 구할 수 있을까요?



여러 방법이 있겠지만 XOR을 사용하면 쉽습니다. 남은 한 점의 좌표는 (x0 ^ x1 ^ x2, y0 ^ y1 ^ y2)입니다.

◆ SWAP을 구현할 수 있습니다.

// WARNING 권장되지 않는 코드

void xor\_swap(int\* x, int\* y) {

\*x ^= \*y;

\*y ^= \*x;

\*x ^= \*y;

}

XOR SWAP은 임시 변수를 만들어 바꾸는 SWAP보다 효과적이지 않고, 두 포인터가 같은 곳을 가리킬 때 제대로 동작하지 않으므로 권장되지 않습니다.

**~ NOT**

◆ 비트 집합에 사용하면 가지고 있지 않은 원소들을 구할 수 있습니다. (**2.4** 참고)

◆ 음의 인덱스로 사용할 수 있습니다.

// 앞에서부터 i번째 원소와 뒤에서부터 i번째 원소를 출력하는 코드

std::vector<int> vec {0, 1, 2, 3, 4};

for (size\_t i = 0; i < vec.size(); ++i) {

printf("%d %d\n", vec.begin()[i], vec.end()[~i]);

}

**<<, >> shift**

◆ 2의 거듭제곱 곱셈/나눗셈

정수 자료형을 왼쪽으로 i칸 밀거나 오른쪽으로 i칸 미는 연산은 각각 2^i를 곱하거나 2^i로 나누는 연산과 같습니다. 특히 / 연산은 느리므로 나누는 수가 2의 거듭제곱일 경우 >>로 바꾸면 성능 향상을 얻을 수 있습니다.

마찬가지로 %(나머지) 연산도 나누는 수가 2의 제곱수일 경우 &로 바꿀 수 있습니다.

// WARNING 음수일 때 제대로 동작하지 않음

void div(int num, int x) {

printf("%d / 2^%d = %d ... %d\n", num, x, num >> x, num & ((1 << x) - 1));

}

**2.4 비트마스킹**  
각 Bit를 하나의 Flag로 활용한다면 자료 저장과 집합 표현을 쉽게 할 수 있습니다.

사람에 0~31 사이의 번호가 매겨져 있고, 사람 A의 친구 목록이 {0, 3, 6, 7, 10, 13, 28}이고, B의 친구 목록이 {0, 1, 4, 5, 6, 17, 21, 28} 이라고 합시다.

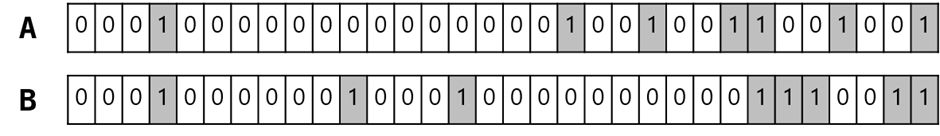
이 때 다음과 같은 문제를 풀어봅시다.

1. A, B 모두와 친구인 사람은?
2. A 또는 B와 친구인 사람은?

반복문을 이용해서 문제를 풀 수 도 있지만, 비트를 사용하면 이러한 "집합" 연산이 간단해집니다.

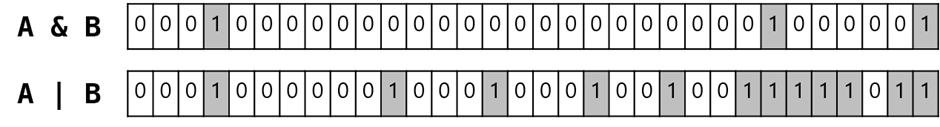
친구 목록을 사람 번호로 저장하지 않고 x번째 사람이 내 친구라면, x번째 비트를 1로 표시하는 방식으로 바꿔보겠습니다.

그러면 A와 B의 친구 목록을 아래처럼 비트로 나타낼 수 있습니다.



이렇게 비트로 친구 목록을 저장하면 앞의 두 문제를 빠르게 해결할 수 있습니다.

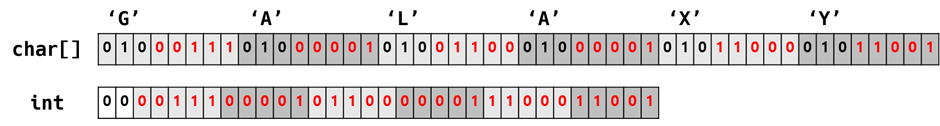
1)번 문제는 두 친구목록을 & 연산해 구할 수 있고 2)번 문제는 | 연산해 구할 수 있습니다.

****

**2.5 데이터 압축**

문자열 두 개를 비교하는 데에는 O(문자열의 길이)의 시간이 듭니다. 만약 사용하는 문자의 가짓수가 적다면 필요한 bit만 골라내서 정수형 자료형에 압축할 수 있습니다.

예를 들어 문자열이 알파벳 대문자로만 이루어졌다면 알파벳끼리를 구분하는 데에 1이상 26이하의 값만 필요합니다. 이는 5bits 만으로도 표현할 수 있죠.



이렇게 압축된 정수의 대소 비교 결과는 원래 문자열의 사전순 비교 결과와 같습니다.

실제로 Pro 시험에는 이런 작은 길이의 문자열을 다루는 문제가 많고, 압축 기법은 굉장히 많이 사용됩니다.

// 12자 이내의 알파벳 대문자 문자열을 하나의 long long 변수에 압축

// WARNING 문자열의 끝(‘\0’) 이후에도 전부 ‘\0’으로 채워져 있어야 함

long long compress(char str[13]) {

long long res = 0;

for (size\_t i = 0; i < 12; ++i) {

res = (res << 5) | (str[i] ^ 64);

}

return res;

}

**3. 기본 문제**

    · 새로운 불면증 치료법  
    · 이진수 표현  
    · 동아리실 관리하기