SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

APS 기본

SAMSUNG SW ACADEMY FOR YOUTH

목차

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

완전 검색 - 부분 집합

부분 집합

- 집합에 포함된 원소들을 선택하는 것이다.

- 다수의 중요 알고리즘들이 원소들의 그룹에서 최적의 부분 집합을 찾는 것이다.
  - 예> 배낭 짐싸기(knapsack)

부분 집합

부분집합의 수

- 집합의 원소가 n개일 때, 공집합을 포함한 부분집합의 수는 2^n개이다.
- 이는 각 원소를 부분집합에 포함시키거나 포함시키지 않는 2가지 경우를 모든 원소에 적용한 경우의 수와 같다.
- 예)

{1, 2, 3, 4}

2 x 2 x 2 x 2 = 16가지

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

부분 집합 구현 - 반복문

부분 집합 생성하는 방법

- 예) {1, 2, 3} 집합의 모든 부분집합(Power Set) 생성
- 반복문을 통한 부분집합 생성

```
FOR i in 1 → 0
    selected[1] ← i                          // 원소1
    FOR j in 1 → 0
        selected[2] ← j                      // 원소2
        FOR k in 1 → 0
            selected[3] ← k                  // 원소3
            FOR m in 1 → 3                   //생성된 부분집합 출력
                if selected[i] == 1 then
                        print i
```

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

부분 집합 구현 - 재귀

부분 집합 생성하는 방법

- **입력된 숫자로 구성된 집합의 모든 부분집합(Power Set) 생성**
- **재귀적 구현을 통해 생성하는 방법**
  - 각 원소를 부분집합에 포함/비포함의 형태로 재귀적 구현을 함

```
input[] : 숫자 배열
isSelected[] : 부분집합에 포함/비포함 여부를 저장한 배열

generateSubSet(cnt)  // cnt: 현재까지 처리한 원소개수

        if(cnt == N)
                부분집합 완성
        else
                isSelected[cnt] ← true
                generateSubSet(cnt +1)
                isSelected[cnt] ← false
                generateSubSet(cnt +1)

end generateSubSet()
```

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

완전 검색 - 부분 집합의 합

부분 집합의 합 문제

- 유한 개의 정수로 이루어진 집합이 있을 때, 이 집합의 부분집합 중에서
  그 집합의 원소를 모두 더한 값이 0이 되는 경우가 있는지를 알아내는 문제
- 예를 들어, { -7, -3, -2, 5, 8 }라는 집합이 있을 때, { -3, -2, 5 }는 이 집합의 부분집합이면서

  (-3)+(-2)+5=0이므로 이 경우의 답은 참이 된다.
- 완전검색 기법으로 부분집합 합 문제를 풀기 위해서는, 우선 집합의 모든 부분집합을 생성한 후에 각 부분집합의 합을 계산해야 한다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

비트 연산

비트 연산

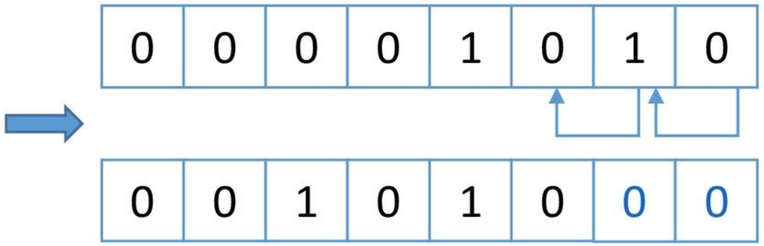
비트 연산자

연산자	연산자의 기능
&	비트단위로 AND 연산을 한다. 예) num1 & num2
|	비트단위로 OR 연산을 한다. 예) num1 | num2
^	비트단위로 XOR 연산을 한다. (같으면 0, 다르면 1) 예) num1 ^ num2
~	단항 연산자로서 피연산자의 모든 비트를 반전시킨다. 예) ~num
<<	피연산자의 비트 열을 왼쪽으로 이동시킨다. 예) num << 2
>>	피연산자의 비트 열을 오른쪽으로 이동시킨다. 예) num >> 2

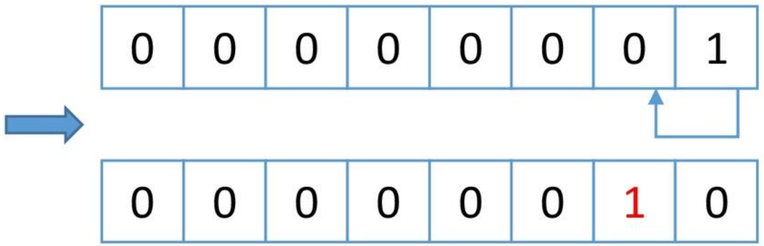
비트 연산자

value << n

- value를 n비트 만큼 왼쪽으로 shift
- 왼쪽으로 밀어내고 남는 오른쪽 자리는 0으로 채움
- 10<<2



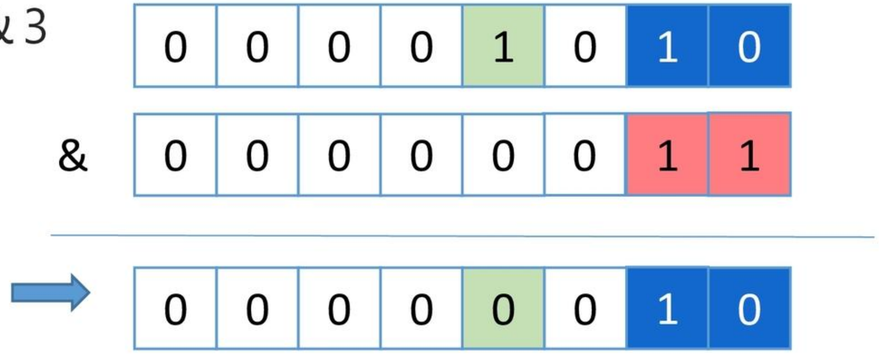
- 1<<1



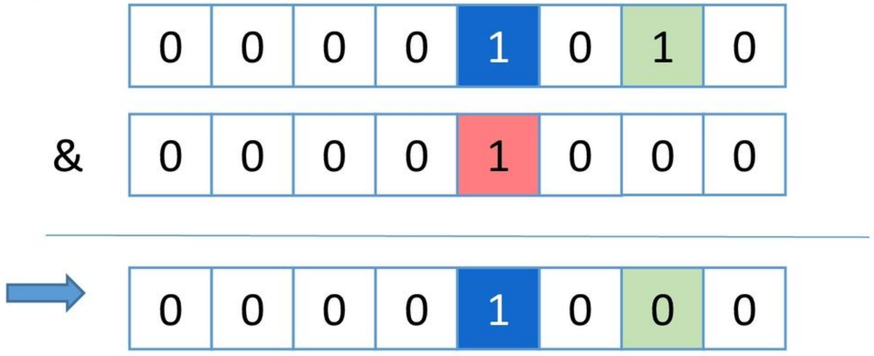
비트 연산자

value1 & value2

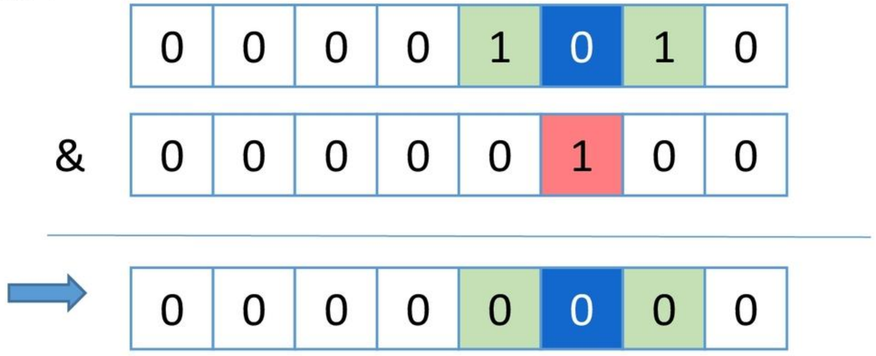
- value1과 value2를 & 연산
- 각 비트열을 비교하여 두 비트 모두 1이면 1, 아니면 0으로 처리
- 10 & 3



- 10 & 1<<3



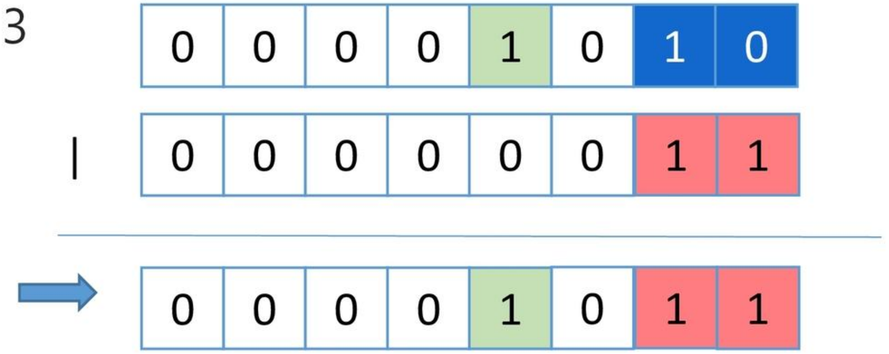
10 & 1<<2



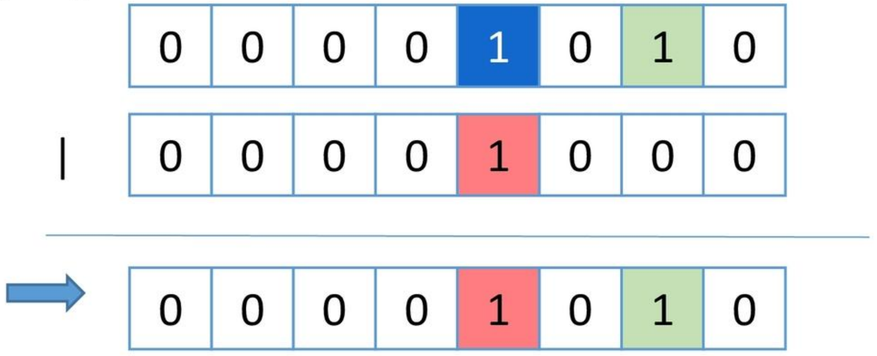
비트 연산자

value1 | value2

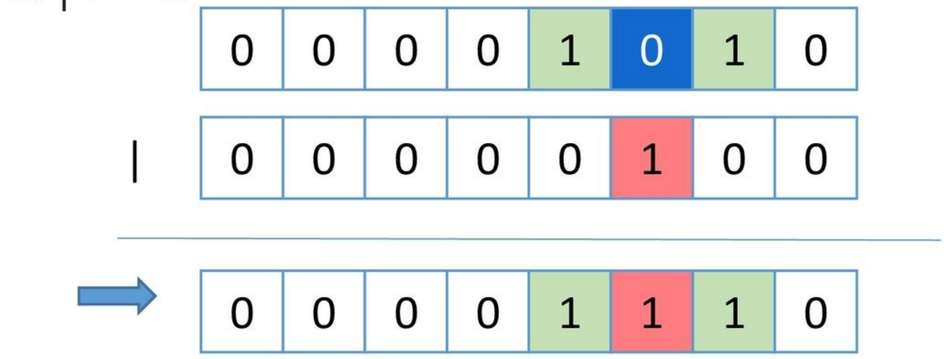
- value1과 value2를 | 연산
- 각 비트열을 비교하여 두 비트 모두 0이면 0, 아니면 1로 처리
- 10 | 3



- 10 | 1<<3



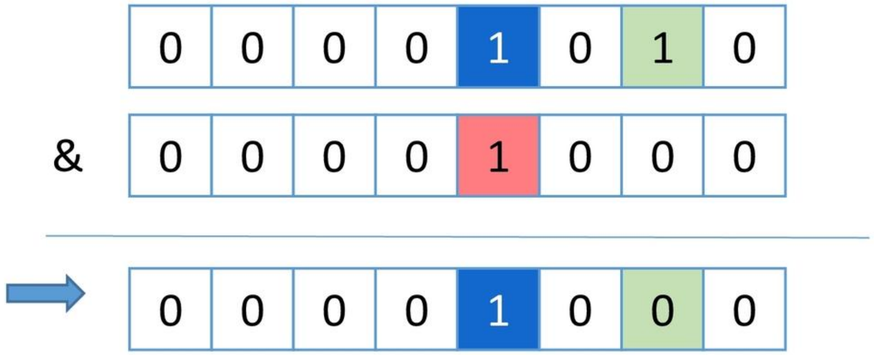
10 | 1<<2



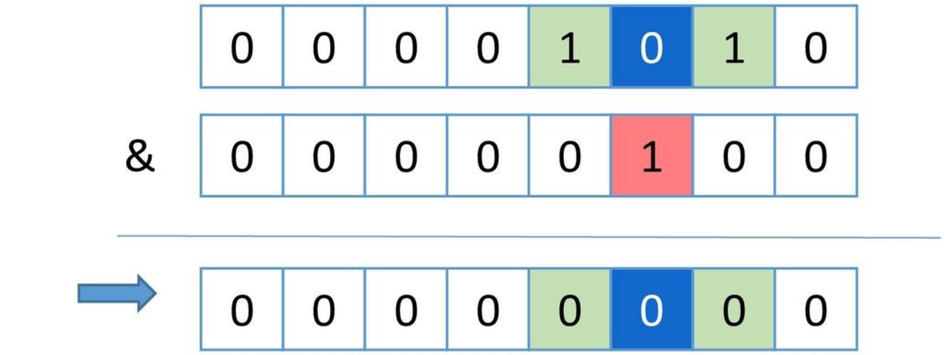
비트 연산자

& 와 | 비교

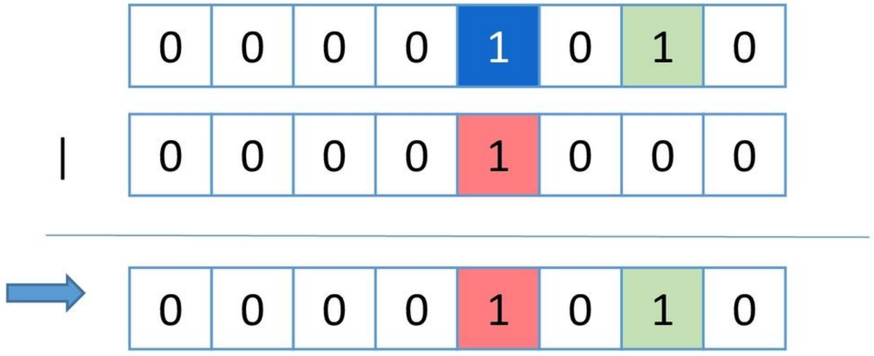
- 10 &1<<3



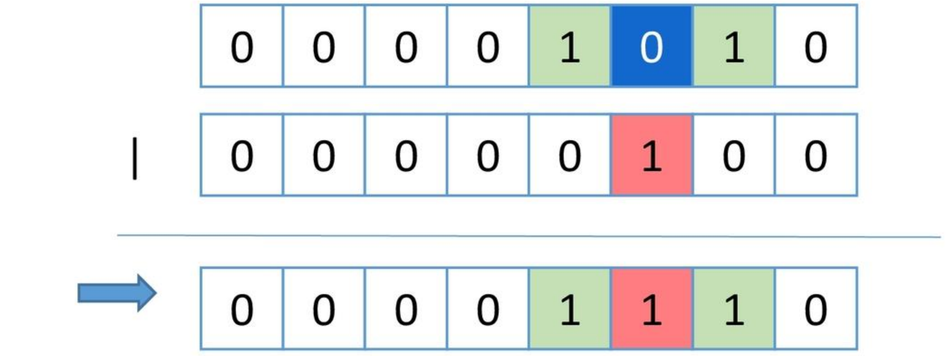
- 10 &1<<2



- 10 | 1<<3



- 10 | 1<<2



SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

부분집합 응용 - 바이너리 카운팅

부분 집합 응용 - 바이너리 카운팅

- **바이너리 카운팅을 통한 사전적 순서(Lexicographical Order)로 생성하는 방법**
  - 부분집합을 생성하기 위한 가장 자연스러운 방법이다.
  - 바이너리 카운팅(Binary Counting)은 사전적 순서로 생성하기 위한 가장 간단한 방법이다.

부분 집합 응용 - 바이너리 카운팅

바이너리 카운팅(Binary Counting)

- 원소 수에 해당하는 N개의 비트열을 이용한다.
- n번째 비트값이 1이면 n번째 원소가 포함되었음을 의미한다.

10진수	이진수	{A, B, C, D}
0	0000	{}
1	0001	{A}
2	0010	{B}
3	0011	{B,A}
4	0100	{C}
5	0101	{C,A}
6	0110	{C,B}
7	0111	{C,B,A}
8	1000	{D}
9	1001	{D,A}
10	1010	{D,B}
11	1011	{D,B,A}
12	1100	{D,C}
13	1101	{D,C,A}
14	1110	{D,C,B}
15	1111	{D,C,B,A}

부분 집합 응용 - 바이너리 카운팅

- 바이너리 카운팅을 통한 부분집합 생성 코드 예

```
int arr[] = {3, 6, 7, 1, 5, 4};
int n = arr.length;

for(int i = 0; i < (1 << n); i++)        // 1 << n : 부분집합의 개수
{
    for(int j = 0; j < n; j++)           // 원소의 수만큼 비트를 비교함
    {
        if( i & (1<<j) ! = 0)            // i의 j번째 비트가 1이면 j번째 원소 출력
            System.out.print(arr[j]+" ");
    }
    System.out.println();
}
```

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 방송에서 만나요!

삼성 청년 SW 아카데미