A_21 비트마스크 뽀개기

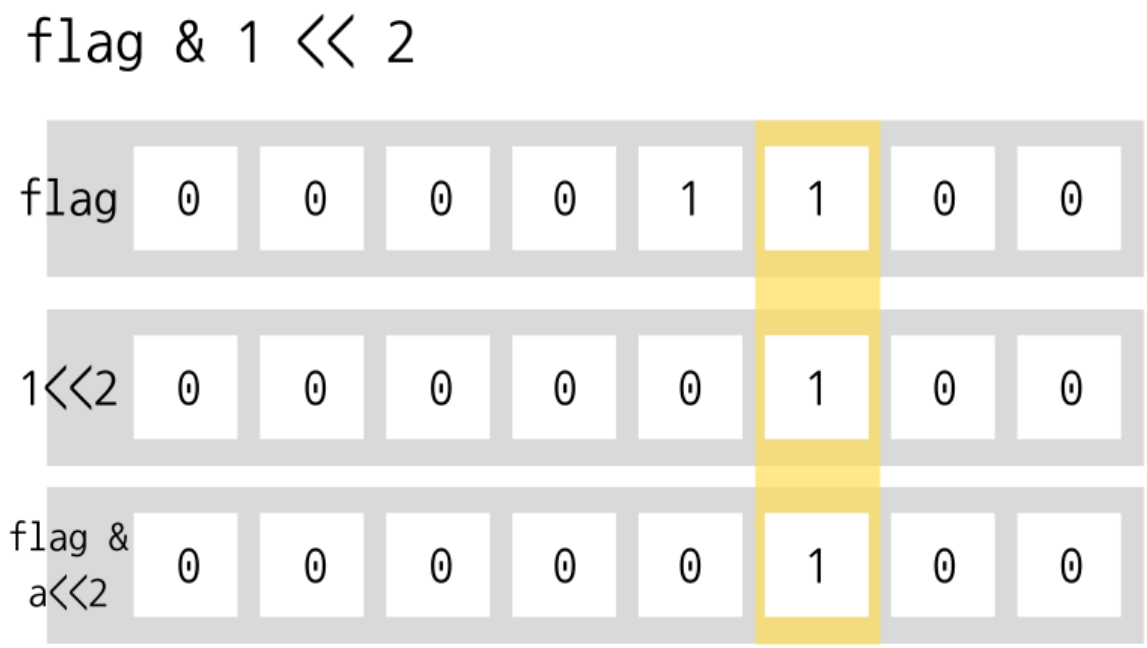
개념

- 정수의 이진수 표현을 활용한 기법
- 선택한 것과 그렇지 않은 것을 구분할 때, 비트를 활용하여 0,1으로 표현하는 것
    - 대체로 0은 비 선택, 1은 선택으로 나타낸다.

비트 연산

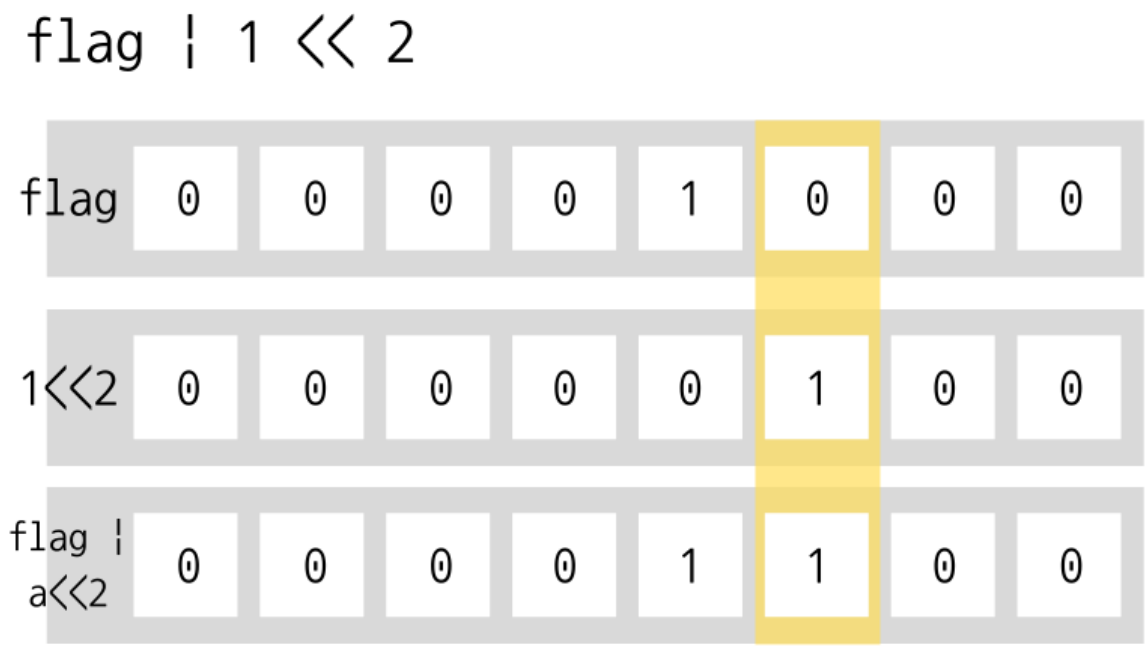
& (AND)	둘 다 1이면 1	A & B
| (OR)	둘 중 하나 1이면 1	A | B
^ (XOR)	같으면 0 다르면 1	A ^ B
~ (NOT)	피연산자의 모든 비트를 반전	~A
<< (LEFT SHIFT)	비트 왼쪽 이동 (빈 자리는 0으로 채움)	A<<i
>> (RIGHT SHIFT)	비트 오른쪽 이동(빈 자리는 최상위 비트값으로 채움)	A>>i

특정 자리 비트의 값 확인



1. 확인을 원하는 자리만 1로 둔 값 구하기(1<<2)
2. 확인을 원하는 값(flag)과 1에서 구한 값을 & 하기(둘 다 1 → 1)
3. 자리 값이 1면 1, 0이면 0이 나오게 됨 ⇒ 실제로는 4가 나오게 됨
4. 0이 아니라면 해당 비트 값은 1이었음을 알 수 있음

특정 자리 비트 값 변경



1. 변경을 원하는 자리만 1로 둔 값 구하기(1<<2)
2. 변경을 원하는 값(flag)과 1에서 구한 값을 | 하기
3. 내가 원하는 특정 비트를 1로 만들어 합한 결과가 나옴

예시(순열)

```
public static void permutation(int cnt, int flag){ // 직전까지 뽑은 수의 개수
    if(cnt == R){
        System.out.println(Arrays.toString(numbers));
        return;
    }
    //입력받은 모든 수를 현재 자리에 넣어보기
    for (int i = 0; i < N; i++) {
        if((flag & 1 << i) != 0)continue; //기존 자리 수들과 중복되는지 체크

        numbers[cnt] = input[i];
        permutation(cnt+1, flag | 1 << i); //다음 수 뽑으러 가기
    }
}
```

장단점

장점

- 간결한 코드
    - 다양한 집합 연산들을 비트 연산자로 작성하기 때문에 반복문, 조건문으로 비교했을 때보다 간결하다.
- 빠른 연산 속도
    - 비트 연산이기 때문에 O(1)에 구현되는 것들이 많아, 다른 자료 구조를 이용할 때보다 빠르게 동작한다.
- 적은 메모리 사용
    - 예시로 비트가 10개인 경우 2^10가지의 경우를 10bit 이진수로 표현이 가능하다.

단점

- 선택/미선택으로 해결되지 않는 문제는 사용 불가

주의 사항

- 일반 수학연산자보다 우선순위가 낮아 원하는 답이 나오지 않을 수 있다.
    - 비트연산자를 사용할 때는 괄호를 사용하여 실수 방지

응용 사항

- 집합
    - “하나의 bit가 하나의 원소"를 의미하게 된다.
    - bit가 1이면 해당 원소가 집합에 포함되어 있다는 의미이고, 0이면 포함되어 있지 않다는 의미이다.
    - N비트 정수 변수라면 N개의 원소를 갖는 집합의 부분집합들을 모두 표현할 수 있게 된다.
- 순열