피자, 20개토핑 (idx 0~19)

31케이스 까지 경우의수로... → 시간복잡도.

NOT	~		
AND	&	공집합	0 //토핑이 하나도 없는 피자
OR	|	꽉찬집합	int fullPizza = (1<<20)-1; //(1<<n)-1; //full (크기가 n인 모든 비트켜기!)
XOR	^	원소추가	toppings |= (1<<P); //P번째 불켜기! (P가 1이면 값변동없음)
a<<b		원소삭제	toppings &= ~(1<<P);
a>>b (a를 b비트만큼)		원소토글	toppings ^= (1<<P); //idx가 P인 원소 켜져있으면 끄고, 꺼져

두집합연산
```
int added = (A | B);        //합집합
int intersect = (A & B);    //교집합
int removed = (A & ~B);     //차집합A-B
int toggled = (A ^ B);      //A와 B중에 한곳에만 포함되어 있는 원소의 집합
```

집합크기 구하기
① __popcnt(toppings);
#include <intirin.h>

②
```
int bitCount(int x){
    if(x==0)return 0;
    return x%2 + bitCount(x/2);
}
```

최소원소 찾기 (펜윅트리)
① 켜져있는 최하위 비트 번호 반환 `__BitScanForward(&index, toppings);`
② 켜져있는 최하위 비트 직접구하기 `int firstTopping = (toppings & -toppings);`

1110
+ 0001 +1
※ 보수개념

최소원소 지우기 `toppings &= (toppings -1);`

★모든 부분 집합 순회★ `for(int i = pizza ; i ; i = ((i-1) & pizza)){ }`

경우의 수 체크 2^n

ex) {포도, 사과, 딸기, 배}의 모든 경우의 수 2^4

```
for (int i = 0; i < (1<<4) ; i++){      모든경우의수순회
    for(int j = 0; j<4; j++){
        if( i & (1<<j)) //있는지 없는지 확인
```

idx번째 비트 끄기	S &= ~(1<<idx);
idx번째 대한 XOR	S ^= (1<<idx);
최하위 켜져 있는 idx찾기	idx = (S & -S);
n인 집합의 모든비트 켜기	int pizza = (1<<n)-1
idx번째 비트 켜기	S |= (1<<idx);
idx번째 비트유무확인	if(S& (1<<idx))

```
string s = tmp & (1<<idx) ? "Yes" : "No";
```

비트마스킹의 한계 : 31까지 가능하다 (32 bit부터는 unsigned 붙여야함)
... 2^30 승하면, 10억 ∴ 시간복잡도 초과 (30~31 개정도.)

따라서 일반 PS에서 30~31승까지 비트마스킹으로 표현할 수 있다고 생각하면 됨.