Bits & Bitmasking

Sogang ACM-ICPC Team 2012-2013 Winter Study

Initiating Class

2013. 02. 14

Sweet Valentine's Day :)

Presentation by 선0

Powered by LaTeX

Bits and Integers

- 이진수의 한 자리
- 0 또는 1
- 1 byte = 8 bit
- (signed/unsigned) int = 4 byte = 4 * 8 bit = 32 bit

Bits and Integers

- Integers in C/C++
  - int : 32 bit integer (-2147483648 ~ 2147483647)
  - unsigned int : 32 bit unsigned integer (0 ~ 4294967296)
  - long long int : 64 bit integer (- 2^{63} ~ 2^{63} - 1)
  - unsigned long long int : 64 bit unsigned integer (0 ~ 2^{64} - 1)

Bits Operation

- & (AND) : bitwise and
  - 두 비트가 모두 1일 때만 1

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

```
a     0111 0001
b     1000 1011
-------------------
a & b 0000 0001
```

- 주의: 연산자 우선순위

$$a \,\&\, b \,\&\&\, c \,\&\, d \quad \Leftrightarrow \quad a \,\&\, (b \,\&\&\, c) \,\&\, d$$

$$(a \,\&\, b) \,\&\&\, (c \,\&\, d)$$

Bits Operation

- | (OR) : bitwise or
  - 두 비트 중 하나라도 1이면 1

x	y	x | y
0	0	0
0	1	1
1	0	1
1	1	1

```
a     0111 0001
b     1000 1011
-------------------
a | b 1111 1111
```

Bits Operation

- ^ (XOR) : bitwise eXclusive OR
  - 두 비트가 다르면 1

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

```
a     0111 0001
b     1000 1011
-----------------
a^ b  1111 1110
```

Bits Operation

- ~ (NOT) : bitwise not
  - 모든 비트 반전

x	~x
0	1
1	0

a 00000000000000000000000010100110 166

~a 11111111111111111111111101011001 ~~89~~ -167

Bits Operation

- << : left shift
  - 왼쪽으로 비트를 이동
- >> : right shift
  - 오른쪽으로 비트를 이동

```
a      0111 0001
-------------------
a<<1   1110 0010
a>>3   0000 1110
```

$1 << x == 2^x, a << x == 2^x a$

$1 >> x == 2^{-x}, a >> x == 2^{-x} a$

Bits Operation

- x &= y ⇔ x = x & y
- x |= y ⇔ x = x | y
- x ^= y ⇔ x = x ^ y
- x <<= y ⇔ x = x << y
- x >>= y ⇔ x <<= y

Fun with Bits

- i번째 비트가 무엇인가 알아보기
  x & (1 << i)
- i번째 비트를 1로 만들기
  x |= (1 << i)
- i번째 비트를 0으로 만들기
  x &= ~(1 << i)
- i번째 비트를 토글하기
  x ^= (1 << i)

...@#%^@#%@$

- 꼭 알아야 하나요?
  - 빠름
  - 짧음
  - 메모리 효율
- 어디에 쓰나요?

Application: Representing Sets

- 집합의 표현
  - 집합에 포함된 원소를 1, 포함되지 않은 원소를 0으로 표시
    - 공집합
    - 전체집합
    - 원소 존재 여부
    - 원소 추가
    - 원소 삭제
    - 원소 토글

Application: Representing Sets

- 합집합
- 교집합
- 차집합

```
a    0111 0001
b    1000 1011
-------------
a | b 1111 1011
a & b 0000 0001
a & (~b) 0111 0000
```

a union b →

a intersection b →

a - b →

- Source Code
  - when $size \leq 60$

```cpp
long long stat = 0;
//Empty set stat

//Insert the xth element
long long insertElement(int x) { return stat | (1LL << x); }

//Delete the xth element
long long deleteElement(int x) { return stat & (~(1LL << x)); }

//Exam if the xth element is in the set
bool isElement(int x) { return (stat &(1LL<< x)) != 0; }
```

Application: Representing Sets

- Source Code
  - when $size \leq 60$, counting the size of the set

```
14 int sizeOfSet() {
15
16     int cnt = 0;
17     for(int i=0; i<(1LL<< n); ++i)
18         if(stat&(1LL<<i)) ++cnt; //or if(isElement(i)) ++cnt;
19
20     return cnt;
21
22 }
```

```
24 //Recursive Version
25 int sizeOfSet(int stat) {
26     if(stat == 0) return 0;
27     return stat % 2 + sizeOfSet(stat / 2);
28 }
```

code by Jongman Koo, 알고리즘 문제 해결 전략2, p582-583

- gcc/g++: _ _builtin_popcount(unsigned int stat)
- Visual C++: _ _popcnt(unsigned int stat)

Application: Representing Sets

- Source Code
  - 켜져 있는 최하위 비트 찾기. i번째 비트가 최하위 비트라면 2^i을 구한다.
  - 최하위 원소 삭제

```
30 //Get the bit of the first element
31 long long firstElement() { return stat & -stat; }
32
33 //Delete the first element
34 long long deleteFirstElement() { return stat & (stat-1) }
35
```

  - 최하위 원소의 번호
    - gcc/g++: __builtin_ctz(unsigned int stat)
    - Visual C++: _BitScanForward(int* idx, unsigned int stat)

Application: Representing Sets

- Source Code
  - Get every subset of the set (except the empty set)

```
36 for(long long int subset = stat; subset; subset = ((subset-1) & stat)) {
37     //...
38 }
39
```

code by Jongman Koo, 알고리즘 문제 해결 전략2, p585

Application: 2^n State

- 2^n가지의 상태 구현
  - 2^n 구현(Brute-Force)문제
  - 2^n Backtracking
  - 2^n이 들어가는 상태공간 탐색 (with BFS, DFS, ...)
- Exponential DP (지수시간 동적계획법)

Application: 2^n State

- Source Code: 2^n Backtracking

```
1 int n = 10;
2
3 bool on[N_MAX];
4 for(int stat=0; stat<(1 <<n); ++stat) {
5
6     for(int j=0; j<n; ++j) {
7         if(stat & (1 << j)) on[j] = true;
8         else on[j] = false;
9     }
10
11    //...
12
13 }
```

code by Jongkwook Choi, 2009 im4u Winter Camp Advanced Class, 2009. 1. 11

Application: 2^n State

- Source Code: 2^n DP - TSP

```
for(int state=0; state<(1 << n); ++state) {

    int ones=0;
    for(int i=0; i<n; ++i)
        if((1 << i) & st) ++ones;

    for(int i=0; i<n; ++i) {

        T[state][i]=0;
        if( ((1 << i) & state) == 0 ) continue;

        if(ones == 1)
            if(i == 0) T[state][i] = 0;

        for(int j=0; j<n; ++j) {

            if( ((1<<j) & state ) == 0 ) continue;
            if(j == i) continue;

            if(T[state][i] > T[state ^ (1 << i)][j] + dist[j][i])
                T[state][i] = T[state ^ (1 << i)][j] + dist[j][i];

        }

    }

}

int ans = 0;
for(int e=0; e<n; ++e)
    if(ans > T[(1 << n) - 1][e] + dist[e][0])
        ans = T[(1 << n) - 1][e] + dist[e][0];

printf("%d", ans);
```

Application: Sieve of Eratosthenes

- Source Code

```
1 int n;
2 unsigned char sieve[(MAX_N + 7) / 8];
3
4 inline bool isPrime(int k) {
5     return sieve[k >> 3] & (1 << (k & 7));
6 }
7
8 inline void setCoomposite(int k) {
9     sieve[k >> 3] &= ~(1 << (k & 7));
10 }
11
12 void eratosthenes() {
13     memset(sieve, 255, sizeof(sieve));
14     setComposite(0);
15     setComposite(1);
16     int sqrtn = int(sqrt(n));
17     for(int i=2; i<=sqrtn; ++i)
18         if(isPrime(i))
19             for(int j=i*i; j<=n; j+=i)
20                 setComposite(j);
21 }
22
23
```

code by Jongman Koo, 알고리즘 문제 해결 전략2, p587