

Bitmasking & Bit Manipulation

Kaun Karthik

// unique numbers — where every no. occurs 2 times except 1
to find unique no. do XOR sum of all nos.

// odd-check.

bool isOdd (int n) {

return ~~n & 1~~ n & 1;

if n & 1.
1 then odd }
0 then even

// to get i^{th} bit from right.

N = 5

i = 2.

2 1 0
←
0 0 0 1 0 1
=

// to mask by i^{th} bit use $1 \ll i$

=> $(N \& (1 \ll i)) > 0$

└─ true ; 1 bit
└─ false ; 0 bit.

int getBit (int n, int i) {

int mask = (1 << i);

int bit = (n & mask > 0) ? 1 : 0;

return bit;

}

Set ith bit

```
int setBit(int n, int i) {
```

```
    int mask = (1 << i);
```

```
    int ans = n | mask;
```

```
    return ans;
```

```
}
```

clear bit.

```
void clearBit(int &n, int i) {
```

```
    int mask = ~(1 << i);
```

```
    n = n & mask;
```

update a bit

```
void updateBit(int &n, int i, int v) {
```

```
    int mask = ~(1 << i);
```

```
    int cleared_n = n & mask;
```

```
    n = cleared_n | (v << i);
```

```
}
```

clear last i bits.

```
int clearLastIBits(int n, int i) {
```

```
    int mask = (-1 << i);
```

```
    return n & mask;
```

```
}
```

Clear a range of bits (i to j)

```
int clearRangeItoJ (int n, int i, int j) {
```

```
    int ones = (~0);
```

```
    int a = ones << (j+1);
```

```
    int b = (1 << i) - 1;
```

```
    int mask = a | b;
```

```
    int ans = n & mask;
```

```
    return ans;
```

```
}
```

$\Rightarrow 111000111$
 \downarrow
 $\underline{a} \quad 111000000$
 $+$
 $\underline{b} \quad 000000111$
 \hline
 $\underline{\text{mask}} \quad 111000111$

// Replace bits in N by M from i to j.

10 9 8 7 6 5 4 3 2 1 0

eg N = 1000 000 0000

M = 10101, i = 2, j = 6

O/P \rightarrow N = 10001010100

```
int replaceBits (int n, int m, int i, int j) {
```

```
    int n = clearRangeItoJ(n, i, j);
```

```
    int ans = n | (m << i);
```

```
    return ans;
```

```
}
```

i) Clear n from i to j
(use fn above)

ii) left shift m by
i places.

iii) then do or operation

// Count number of set bits.

Given $n = 13$, binary of $13 = 1101 \therefore O/P = 3$.

3 methods

① to get last bit use $(n \& 1)$ then add 1 to count
then to go to next bit use $N = N \gg 1$
(i.e. discard right bit).

& continue till $N > 0$.

```
{  
    int ans = 0;  
    while (n > 0) {
```

```
        ans = ans + (n & 1);
```

```
        n = n >> 1;
```

```
    }  
    return ans;
```

complexity $\Rightarrow O(\log N + 1)$

long long int $\rightarrow 10^{18}$

```
② int ans = 0;  
   while (n > 0) {
```

```
       n = n & (n - 1);
```

```
       ans++;
```

```
   }
```

// Ans is no. of time
while loop runs.

$n = n \& (n - 1)$

\hookrightarrow removes bits from
right to left

$n = 1001$ $ans = 1$
 $n - 1 = 1000$

$n = 1000$ $ans = 2$

$n - 1 = 0111$
 $\underline{0000} \Rightarrow 2 \text{ bits}$

Complexity $\rightarrow O(\text{no. of set bits})$
 worst case $\rightarrow O(\log n)$

③ `__builtin_popcount(n)` } directly gives you the no. of set bits.
 $1 \leq n \leq 10^9$

decimal to binary

```
int decimalToBinary(int n) {
    int ans = 0;
    int p = 1;
    while (n > 0) {
        int last_bit = (n & 1);
        ans = ans + (p * last_bit);
        p = p * 10;
    }
}
```

`(n = n >> 1; n)`

`return ans;`

}

$1 = 100$ $1001 = n$
 $0001 = 1 - n$

$2 = 100$ $0001 = n$
 $1100 = 1 - n$

$n = 13$ 1101

① last bit = 1
 $ans = 0 + 1 \times 1 = 1$
 $p = 10$

② last bit = 0
 $ans = 1 + 0$
 $p = 100$

③ last bit = 1
 $ans = 101$
 $p = 1000$

④ last-bit = 1.
ans = 1101.
p = 10000

⑤ $n = 0$.

unique no - II → find unique no in $O(n)$ & no extra space.

eg $a = [1, 1, 2, 3, 5, 2, 5, 7]$ $\Rightarrow 3, 7$.

① XOR all array, then result = XOR.

↳ can never be zero

eg $5, 1, 2, 1, 2, 3, 5, 7$

⇒ the no. will contain at least 1 set bit.

result = $5 \wedge 1 \wedge 2 \wedge 1 \wedge 2 \wedge 3 \wedge 5 \wedge 7$

$= 3 \wedge 7 =$

$$\begin{array}{r} 0111 \\ 0011 \\ \hline 0100 \\ 3210 \end{array}$$

→ at 2nd pos set bit is present.

→ then we make a list of element which has set bit at second pos.

⇒ $a = \{5 \wedge 5 \wedge 7\} = 7$.

now for second unique no $b = \text{result} \wedge a$

$= 3 \wedge 7 \wedge 7 = 3$.

① to find XOR → $\text{res} = \text{res} \wedge \text{no.}$

② to find unique set bit → while((temp & 1) != 1) {

pos++;

temp = temp >> 1;

③ unique out of 3 (A)

eg. 7
1 1 1 2 2 2 3

O/P → 3.

⇒ write binary rep.

eg: 3, 3, 2, 1, 1, 1, 3.

* if we consider each bit pos, sum of all of these

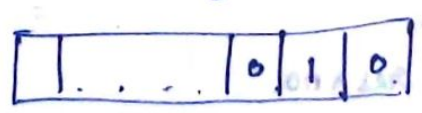
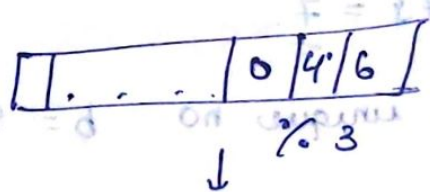
will be $3N$ or $3N+1$.

3 → 0 1 1
3 → 0 1 1
2 → 0 1 0
1 → 0 0 1
1 → 0 0 1
1 → 0 0 1
3 → 0 1 1

0, 4, 6
↓ ↓ ↓
 $3N$ $3N+1$

largest int = 64 bits

so store sum in this array of arr[64];



then convert to decimal

⇒ 2 → unique

```
int main() {
```

```
int cnt[64] = {0};
```

```
int n, no;
```

```
cin >> n;
```

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

```
    cin >> no;
```

```
    // update cnt array by extracting bits.
```

```
    int j = 0;
```

```
    while (no > 0) {
```

```
        int last-bit = (no & 1);
```

```
        cnt[j] = cnt[j] + last-bit;
```

```
        j++;
```

```
        no = no >> 1;
```

```
    }
```

```
}
```

```
// iterate over array & convert it to no.
```

```
int p = 1;
```

```
int ans = 0;
```

```
for (int i = 0; i < 64; i++) {
```

```
    cnt[i] %= 3;
```

```
    ans += (cnt[i] * p);
```

```
    p = p << 1;
```

```
}
```

```
cout << ans;
```

```
return 0;
```


Fast exponentiation

$$a^5 = a^{101} = a^{2^2 \cdot 2^1 \cdot 2^0} = a^{4+0+1} \\ = a^4 \cdot a^0 \cdot a^1 \\ = \underline{a^5}$$

When we

have N then iterating takes $\log_2 N$

eg. $a=3$, $N=5$. initial ans = 1.

$$a^5 = a^{101}$$

↑
k

$$a = 3^1$$

~~ans = 1~~ $k=1$, then ans = 3.

$$a = 3^2$$

$k=0$, then ans = 3.

$$a = (3^2)^2 = 3^4$$

$$k=1, \text{ then } \text{ans} = 3 \times 3^4 \\ = 3^5 = \underline{243}$$

Code

```
int power_optimised (int a, int n) {
```

```
    int ans = 1;
```

```
    while (n > 0) {
```

```
        int last_bit = (n & 1);
```

```
        if (last_bit) { // if set, then multiply
```

```
            ans = ans * a;
```

```
        }
```

```
        a = a * a;
```

// square the val.

```
        n = n >> 1;
```

// Discard last bit.

```
    }
```

```
    return ans;
```

```
}
```

find the subsets of given string

i/p \rightarrow abc

o/p \rightarrow "", a, ab, abc, ac, b, bc, c

approach \rightarrow

Subsequences

a b c \rightarrow every character can be part of result/not.
 $2 \times 2 \times 2$

\hookrightarrow 8 possible subsets

2 components \rightarrow iterating over strings. ①

extracting correct elements ②

from 0 to $2^N - 1$ \leftarrow using a function \rightarrow filter
[N is length of string] of (N, "abc")

$2^N - 1$ \downarrow \downarrow 101 ac (a-c)

$\hookrightarrow (1 \ll N) - 1$

```
void printSubsets(char a[]) {
```

```
    int n = strlen(a);
```

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

```
    { filterChars(i, a);
```

```
    }
```

```
}
```

```
void filterChars(int n,  
char a[]) {
```

```
    int j = 0;
```

```
    while (n > 0) {
```

```
        int last_bit = (n & 1);
```

```
        if (last_bit == 1) {
```

```
            cout << a[j];
```

```
        } j++;
```

```
        n = n >> 1;
```

```
}
```