Bitmask

Nafis Ul Haque Shifat and Debojoti Das Soumya

4th February 2022

সূচীপত্র

1 Base Conversion and Bitwise Operations

Debojoti Das Soumya

1.1 Decimal to other base

> There are 10 kinds of people. Those who understand binary and those who don't.

আমরা যে সংখ্যা লিখি আমরা ১০ টি অঙ্ক (digit) ব্যবহার করি। কিন্তু আমাদেরকে কি সবসময় ১০ টি অঙ্কই ব্যবহার করতে হবে? উত্তর অবশ্যই না। আমরা চাইলে যত ইচ্ছা অঙ্ক ব্যবহার করতে পারি এবং যত ইচ্ছা সংখ্যা পদ্ধতি ব্যবহার করতে পারি। একটি সংখ্যা পদ্ধতিতে যতগুলো অঙ্ক থাকে তাকে ওই সংখ্যা পদ্ধতির ভিত্তি (base) বলে। আমরা যে সংখ্যা পদ্ধতি ব্যবহার করি সেটায় সর্বোচ্চ ১০ টি অঙ্ক থাকে। তাই এই সংখ্যা পদ্ধতিকে ১০ ভিত্তিক সংখ্যা পদ্ধতি (decimal number system) বলে। এখন যদি আমরা ২ টি অঙ্ক ব্যবহার করি তাহলে সেটাকে ২ (binary) ভিত্তিক সংখ্যা পদ্ধতি বলে। ৩ টি অঙ্ক থাকলে ৩ (ternary) ভিত্তিক সংখ্যা পদ্ধতি বলে। এক এক ভিত্তিতে একই সংখ্যা এক এক রকম দেখায়। তাই একটি সংখ্যাকে ঠিক মতো বুঝতে তার ভিত্তি ও বলে দেওয়া হয়। যেমন decimal 100 কে $(100)_{10}$ লিখা হয় এবং binary 100 কে $(100)_2$ লিখা হয়। দুইটি সংখ্যাই দেখতে 100 হলেও তারা এক নয় কারণ $(100)_2 = (4)_{10} \neq (100)_{10}$। অর্থাৎ বাইনারিতে 100 মানে হলে decimal এ 4। এটি কিভাবে বের করলাম তা একটু পরেই জানতে পারবে।

আমরা ছোট বেলায় স্থানীয় মান (place value) বের করতাম। মনে আছে? যেমন 1023 এর স্থানীয় মান হবেঃ

$$1023 = 1 \cdot 10^3 + 0 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$$

আমরা কিন্তু 1023 এর সামনে আর কিছু 0 যোগ করতে পারি। তাহলে সংখ্যাটির স্থানীয় মান হবেঃ

$$001023 = \cdots + 0 \cdot 10^5 + 0 \cdot 10^4 + 1 \cdot 10^3 + 0 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$$

দুইটি স্থানীয় মানই কিন্তু আসলে এক।

স্থানীয় মান দিয়ে আমরা আসলে কী বের করি? একটি সংখ্যায় প্রতিটি 10 এর power কয়বার আছে যদি আমরা এই 10 এর power এর চেয়ে বড় সব 10 এর power বাদ দিয়ে দেই। তাই না? যেমন এখানে আমরা দেখলাম 10^∞ থেকে 10^4 পর্যন্ত সবগুলো power ই 0 বার এসেছে। তার পর 10^3 একবার। সবগুলো $10^3, 10^4, 10^5...$ বাদ দিয়ে দিলে সংখ্যাটি হবে 023 এখানে 10^2 আছে 0 টি। সব $10^2, 10^3, 10^4...$ বাদ দিলে সংখ্যাটি হবে 23 এখানে দুইটি 10^1 আছে। এবং সব $10^1, 10^2, 10^3...$ বাদ দিলে সংখ্যাটি হবে 3 এখানে শুধু তিনটি 10^0 আছে।

এভাবে সব power কয়বার আছে তা বের করেই আমরা একটি সংখ্যার অঙ্ক বের করতে পারি। একটি সংখ্যার সবচেয়ে ডানের অঙ্ক কে 0th digit, এর আগের অঙ্ক কে 1st digit এমন ভাবে পড়া হয়। সবচেয়ে ডানের অঙ্ক কে Least Significant Digit (LSD) এবং সবচেয়ে বামের ধনাত্মক অঙ্ক কে Most Significant Digit (MSB) বলা হয়।

এখানে 10^0 আছে 3 টি তাই 1023 এর 0th digit হবে 3, 10^1 আছে 2 টি তাই 1023 এর 1st digit হবে 2, 10^2 আছে 0 টি তাই 2nd ডিজিট হবে 0, 10^3 আছে 1 টি তাই 3rd digit হবে 1, 10^4 থেকে 10^∞ পর্যন্ত সব power ই 0 বার এসেছে তাই এই সংখ্যার সব বামের ডিজিটগুলো হবে 0। 001023 র LSD হবে 3 এবং MSD হবে 1।

এখানে 10 এর power ব্যবহার না করে অন্য সংখ্যার power ব্যবহার করলে আমরা যে সংখ্যাটি পেতাম সেটি হবে যে সংখ্যার power ব্যহহার করেছি তার ভিত্তির সংখ্যা। যেমন $(140)_{10}$ কে এখন আমরা 3 এর power আকারে লিখবঃ

$$\begin{aligned} 140 &= 1 \cdot 81 + 2 \cdot 27 + 1 \cdot 3 + 2 \\ &= \dots + 0 \cdot 3^6 + 0 \cdot 3^5 + 1 \cdot 3^4 + 2 \cdot 3^3 + 0 \cdot 3^2 + 1 \cdot 3^1 + 2 \cdot 3^0 \end{aligned}$$

এখানে ঠিক আগের মতো power গুলো কয়বার আছে তা বের করেছি। এখন আমরা $(140)_{10}$ কে ternary তে লিখতে পারি $(00012012)_3$।

আরেকটা উদাহরণ দেওয়া যাক। যেমন $(43)_{10}$ কে 2 এর power আকারে লিখলে পাবঃ

$$\begin{aligned} 43 &= 32 + 8 + 2 + 1 \\ &= \dots + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \end{aligned}$$

তাই $(43)_{10}$ কে binary তে লিখলে হবে $(00101011)_2$।

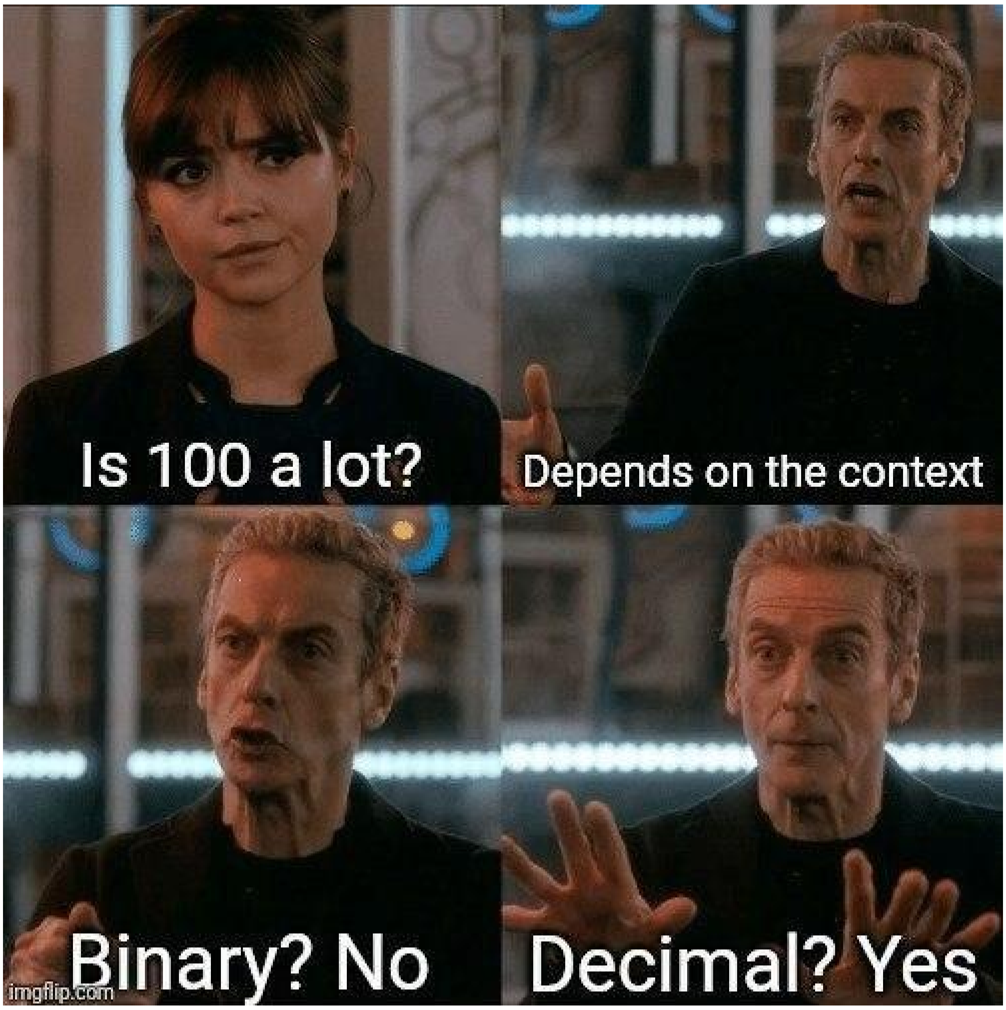
যেকোন ভিত্তির জন্য এই বিষয়টা generalize করা যাক। যদি কোনো সংখ্যা n কে b ভিত্তিতে রূপান্তর করতে চাইলে প্রথমে আমরা n কে b এর power আকারে লিখলে যদি নিচের মতো হয় তাহলে $n = (a_m a_{m-1} a_{m-2} \dots a_2 a_1 a_0)_b$।

$$n = a_m \cdot b^m + a_{m-1} \cdot b^{m-1} + \dots + a_2 \cdot b^2 + a_1 \cdot b^1 + a_0 \cdot b^0$$

। $b > 1$ হলে সবসময় সব i এর জন্য $a_i < b$ হবে। এখন আমরা যেকোন সংখ্যা কে decimal থেকে অন্য ভিত্তিতে রূপান্তর করতে পারব। নিচে কোডটি দেওয়া হলোঃ

```
string base_convert(int n, int b) {
  if (n == 0) return "0";
  int x = 0;
  int power = 1;
  while (power * b <= n) {
    power *= b;
    x++;
  }
  string d;
  while (n > 0) {
    int k = n / power;
    d += k + '0';
    n -= k * power;
    power /= b;
  }
  return d;
}
```

কোডটি কিভাবে কাজ করছে তা একটু উদাহরণের মাধ্যমে বুঝা যাক। যেমন $n = 19, b = 3$। প্রথমেই আমরা একটি string, d রাখব যা b base এ n এর অঙ্ক গুলো রাখবে। এখন বের করব সবচেয়ে বড় x যেন $b^x \leq n$। এই ক্ষেত্রে $x = 2$। একটা জিনিস খেয়াল কর যে 3^3 বা তার চেয়ে কোন বড় সংখ্যা 0 বার থাকবে তাই 3rd digit থেকে তার আগে পর্যন্ত সব ডিজিটি 0 হবে। এখন 2nd digit হবে, $d_2 = \left\lfloor \frac{19}{3^2} \right\rfloor = 2$ তারপর আমরা $n = 19 - 2 \cdot 3^2 = 1$ বাদ দিয়ে দিব। এখন 1st digit হবে, $d_1 = \left\lfloor \frac{1}{3^1} \right\rfloor = 0$ তারপর $n = 1 - 0 \cdot 3^1 = 1$। এখন 0th digit hobe, $d_0 = \left\lfloor \frac{1}{3^0} \right\rfloor = 1$ তারপর $n = 1 - 1 = 0$। এখন যেহেত $n \not> 0$ লুপ বন্ধ হয়ে যাবে।



1.2 Other base to decimal

এতক্ষণ আমরা শুধু decimal থেকে অন্য base এ রূপান্তরক করেছি। এখন আমরা অন্য base থেকে decimal এ রূপান্তর করব। একটি সংখ্যা n কে b base এ দেওয়া আছে $(n)_{10} = (a_m a_{m-1} a_{m-2} ... a_2 a_1 a_0)_b$ । আমরা জানি $n = a_m \cdot b^m + a_{m-1} \cdot b^{m-1} + \cdots + a_2 \cdot b^2 + a_1 \cdot b^1 + a_0 \cdot b^0$ । এখন এই সূত্র থেকেই আমরা n বের করে ফেলতে পারি। নিচে কোডটি দেয়া হলোঃ

```
int convert_to_decimal(string a, int b) {
  int n = 0;
  int power = 1;
  for (int i = (int) a.size() - 1; i >= 0; i--) {
    n += (c - '0') * power;
    power *= b;
  }
  return n;
}
```

উপরের কোড দুটি $b \leq 10$ এর জন্য কাজ করবে কারণ $b > 10$ হলে আমরা আর $0, 1, ..., 8, 9$ দিয়ে সংখ্যাটি লিখতে পারব না, আমাদের আর বেশি digit লাগবে। সাধারণত $A, B, C, ...$ ব্যবহার করা হয়। তাই কিছু case work করেই $b > 10$ হলেও ওই base এ রূপান্তর করা যাবে। যদিও $b > 10$ এ ভিত্তি পরিবর্তন করার খুব একটা দরকার হয় না।

1.3 Any base to any other base

এখন আমরা একটি সংখ্যাকে b_1 base থেকে b_2 তে রূপান্তর করতে চাইলে প্রথমে b_1 থেকে decimal এ রূপান্তর করব তারপর decimal থেকে b_2 base এ রূপান্তর করতে পারব।

1.4 Bitwise operations

1.4.1 Bitwise and

কোন সংখ্যা $(n)_{10} = (a_m a_{m-1} ... a_2 a_1 a_0)_2, (m)_{10} = (b_m b_{m-1} ... b_2 b_1 b_0)_2$ হলে $n \& m = (c_m c_{m-1} ... c_2 c_1 c_0)_2$ হবে যেখানেঃ

$$c_i = \begin{cases} 1, & \text{if } a_i = b_i = 1 \\ 0, & \text{otherwise} \end{cases}$$

যেমন $a = (10)_{10} = (1010)_2, b = (12)_{10} = (1100)_2$, $a \& b = 1010 \& 1100 = (1000)_2 = (8)_{10}$।

1.4.2 Bitwise or

কোন সংখ্যা $(n)_{10} = (a_m a_{m-1} ... a_2 a_1 a_0)_2, (m)_{10} = (b_m b_{m-1} ... b_2 b_1 b_0)_2$ হলে $n | m = (c_m c_{m-1} ... c_2 c_1 c_0)_2$ হবে যেখানেঃ

$$c_i = \begin{cases} 1, & \text{if } a_i = 1 \text{ or } b_i = 1 \\ 0, & \text{otherwise} \end{cases}$$

যেমন $a = (10)_{10} = (1010)_2, b = (12)_{10} = (1100)_2$, $a | b = 1010 | 1100 = (1110)_2 = (14)_{10}$।

1.4.3 Bitwise exclusive or (xor)

কোন সংখ্যা $(n)_{10} = (a_m a_{m-1} ... a_2 a_1 a_0)_2, (m)_{10} = (b_m b_{m-1} ... b_2 b_1 b_0)_2$ হলে $n \oplus m = (c_m c_{m-1} ... c_2 c_1 c_0)_2$ হবে যেখানেঃ

$$c_i = \begin{cases} 0, & \text{if } a_i = b_i \\ 1, & \text{otherwise} \end{cases}$$

যেমন $a = (10)_{10} = (1010)_2, b = (12)_{10} = (1100)_2$, $a \oplus b = 1010 \oplus 1100 = (0110)_2 = (6)_{10}$ ।

1.4.4 Left Shift

কোন সংখ্যা n কে x ঘর left shift করা অর্থ n এর binary র ডানে x টি শূন্য যোগ করা। যেমন $n = (3)_{10} = (11)_2$, $x = 2$ হলে, $(n << x) = (1100)_2 = (12)_{10}$ । আবার x ঘর left shift করা অর্থ 2^x দিয়ে গুন করা ও বুঝায়। কিন্তু সাবধান C++ এ কোন int কে x ঘর left shift করলে ওই int বাম দিকে থেকে প্রথম x bit মুছে দেয়া হয়।

1.4.5 Right Shift

কোন সংখ্যা n কে x ঘর right shift করা অর্থ n এর binary র ডানের x বিট মুছে দেয়া। যেমন $n = (9)_{10} = (1001)_2, x = 2$, হলে, $(n >> x) = (10)_2 = (2)_{10}$ । আবার x ঘর right shift করা অর্থ 2^x দিয়ে ভাগ করে floor নেয়া।

1.4.6 Bitwise not

Bitwise not operation একটি সংখ্যার সব বিট invert করে দেয়। যেমন $a = (1001)_2$ হলে $\neg a = (0110)_2$ । C++ এ $\neg a = (11111111111111111111111111111111001)_2$ কারণ int এ 32 টি বিট থাকে, long long ব্যবহার করলে 64 টি বিট। আগে ওই বিটগুলোয় 0 ছিল তাই bitwise not নেয়ার কারণে ওই বিটগুলো ১ হয়ে গেছে।

1.4.7 Builtin functions

C++ এ কিছু builtin function আছে যা bit manupulation এ খুব কাজে লাগে। যেমনঃ

1. __builtin_popcount(n): n এ কয়টি 1 বিট আছে তা return করে। যেমন $n = (1010)_2$ হলে function টি 2 return করবে।
2. __builtin_clz(n): n এর শুরুতে কয়টি 0 (leading zero) আছে তা return করে। যেমন $n = (101110)_2$ হলে function টি 26 return করবে কারণ C++ int এ যেকোন সংখ্যাকে 32 বিটে রাখা হয়, এখানে ৬ টি বিট বাদ গেলে সামনে আর ২৬ টি 0 রয়েছে।
3. __builtin_ctz(n): n এর শেষে কয়টি 0 (tailing zero) আছে তা return করে। যেমন $n = (1100)_2$ হলে function টি 2 return করবে কারণ n এর শেষে 2 টি zero আছে।

উপরের সব function শুধু int এর জন্য কাজ করবে। long long এর জন্য function গুলার শেষে ll যোগ করতে হবে। যেমন __builtin_popcountll(n)।

নিচে C++ এ function / operations গুলোর implementation গুলো দেখানো হলোঃ

```
int and_value = (a & b);
int or_value = (a | b);
int xor_value = (a ^ b);
int right_shift = (a >> b);
int left_shift = (a << b);
int bitwise_not = (~a);
int ones = __builtin_popcount(a); // __buitlin_popcountll(a) in case of long long
int lz = __builtin_clz(a); // __builtin_clzll(a) in case of long long
int tz = __builtin_ctz(a); // __builtin_ctzll(a) in case of long long
```

একটি সংখ্যা x এর LSB (Least Significant Bit) কিভাবে বের করা যায়? আমরা যদি $x\&1$ করেই আমরা x এর LSB বের করে ফেলতে পারি।

একটি সংখ্যা a এর i-th bit বের কিভাবে করব? এটি করতে আমরা i এর আগের সব বিট বাদ দিয়ে দিতে পারি এবং শেষে LSB কি তা বের করলেই হয়ে যাচ্ছে। যেমন $x = (10110)_2$ এর ক্ষেত্রে তৃতীয় (0 based indexing) বিট হবে 0। এখন যদি আমরা তৃতীয় বিটটি বের করতে চাই তাহলে আমরা শেষের তিনটি বিট বাদ দিয়ে দিব তাহলে $y = 10$ এখন y এর LSB ই হলো x এর তৃতীয় বিট। আমরা কিন্তু জানি right shift ব্যবহার করে একটি সংখ্যার k টি বিট বাদ দেওয়া যায়। তাই C++ এ i-th bit বের করতে হলে আমরা শুধু $(x >> i)\&1$ করব।

2 Problems

Nafis Ul Haque Shifat

2.1 Bacteria

Statement

তোমার ব্যাকটেরিয়া খুব পছন্দ, প্রতিদিন সকালে তুমি একটি বাক্সে কিছু সংখ্যক ব্যাকটেরিয়া রাখতে পারবে, প্রতি রাতে ওই বাক্সের প্রতিটি ব্যাকটেরিয়া ভেঙে দুটি ব্যাকটেরিয়া হয়ে যাবে!(অর্থাৎ বাক্সে n টি ব্যাকটেরিয়া থাকলে রাত শেষে বাক্সে $2n$ টি ব্যাকটেরিয়া থাকবে)। তোমার কাজ হচ্ছে প্রতিদিন সকালে এমন ভাবে বাক্সে নতুন ব্যাকটেরিয়া রাখা যেন কোনো এক মুহূর্তে তুমি বাক্সে x টি ব্যাকটেরিয়া পাও!
সর্বনিম্ন কতটি ব্যাকটেরিয়া তোমার বাক্সে রাখতে হবে?

একটা উদাহরণ দেখা যাক, $x = 5$ হলে তুমি প্রথম দিন সকালে বাক্সে 1 টি ব্যাকটেরিয়া রাখলে, প্রথম রাতে সেটি ভেঙে 2 টি এবং পরের রাতে এই 2টি ব্যাকটেরিয়া ভেঙে 4 টি হবে। এবার পরের দিন সকালে তুমি আরেকটি ব্যাকটেরিয়া বাক্সে রাখলেই সব মিলিয়ে 5টি ব্যাকটেরিয়া পেয়ে গেলে!
এবার সমাধানের দিকে যাওয়া যাক। ধরি বাক্সে এখন y টি ব্যাকটেরিয়া আছে (শুরুতে $y = 0$)। তাহলে ওইদিন রাত শেষে ব্যাকটেরিয়া থাকবে $2y$ টি, তার পরের রাত শেষে থাকবে $4y$ টি, তার পরের রাত শেষে $8y$ ... যদি আমরা y এর বাইনারি রিপ্রেজেন্টেশনের দিকে তাকাই, তবে দেখব প্রতি রাত শেষে আসলে y এর শেষে একটি করে শূন্য যোগ হচ্ছে, যেমন $y = (10011)_2$ হলে রাত শেষে y হবে $(100110)_2$, তার

পরের রাত শেষে হবে $(1001100)_2$! আমাদের লক্ষ্য হচ্ছে $y = x$ বানানো! যেহেতু 0 গুলো রাত শেষে এমনিতেই যোগ হয়ে যায়, তাই আমাদের x এর বাইনারি তে যেরকম ভাবে 1 আছে, y তেও সেরকম করে 1 বসাতে হবে।

ধরা যাক, $x = (1001101)_2$, শুরুতে $y = 0$। প্রথম দিন সকালে বাক্সে আমরা 1টি ব্যাকটেরিয়া রাখলাম, অর্থাৎ এখন $y = 1_2$। প্রথম তিন রাত শেষে y এর শেষে তিনটি শূন্য বসবে, অর্থাৎ y হবে $(1000)_2$। কিন্তু বামদিক থেকে x এর চতুর্থ বিট হচ্ছে 1, তাই পরের দিন সকালে আমরা বাক্সে নতুন ব্যাকটেরিয়া রাখব, এবার $y = (1001)_2$ হবে! দেখ বাম দিক থেকে x এর 4টি বিট ও y এর 4টি বিট মিলে গেছে, এবার বাকি গুলো ও মিলিয়ে ফেলি! রাত শেষে y তে আরেকটি শূন্য বসবে, y হবে $(10010)_2$, x এ সেখানে 1 হওয়ার কথা, তাই আমরা পরের দিন সকালে আবার 1টি ব্যাকটেরিয়া বাক্সে রাখব, তবেই $y = (10011)_2$ হয়ে যাবে! আবার দুই রাত শেষে $y = (1001100)_2$ হবে, পরের দিন সকালে আরেকটি ব্যাকটেরিয়া রাখলেই $y = (1001101)_2$ হয়ে যাবে, ঠিক x এর সমান! বুঝাই যাচ্ছে x এ যতগুলো 1 আছে আমাদের ঠিক ততগুলো ব্যাকটেরিয়া ই বাক্সে রাখতে হবে!

x এ কয়টি বিট 1 বের করার জন্য C++ এ খুব সহজ একটি ফাংশন আছে, সেটি হচ্ছে __builtin_popcount()!

```
cout << __builtin_popcount(x) << endl;
```

2.2 Find Special

Statement

তোমাকে n সাইজের একটি array দেয়া হবে, যেখানে $1 \leq n \leq 10^6$। এই array এর একটি বাদে বাকি সব গুলো সংখ্যাই ঠিক দুবার করে থাকবে, বাকি ওই সংখ্যাটি থাকবে ঠিক একবার, এটি একটি special সংখ্যা! আমাদের এই special সংখ্যা টি খুঁজে বের করতে হবে। তবে এই সমস্যায় **মেমরি লিমিট** হচ্ছে 1MB।

এই প্রব্লেম এর আসল চ্যালেঞ্জ হচ্ছে এর মেমরি লিমিট, 1MB এতই কম যে আমরা 10^6 সাইজের একটি array মেমরি তে রাখতে পারব না (10^6 সাইজের একটি array স্টোর করতে মোটামোটি 4MB লাগে)। তাই আমাদের যা করার ইনপুট নেয়ার সাথে সাথেই করতে হবে, যেন কাজটা $O(1)$ মেমরি তে করা যায়। মজার ব্যাপার হচ্ছে আমাদের এই special সংখ্যাটি আসলে পুরো array টির Bitwise Xor এর সমান! অর্থাৎ যদি $x = a_1 \oplus a_2 ... \oplus a_{n-1} \oplus a_n$ হয় তবে x ই হচ্ছে সেই special সংখ্যাটির সমান যা array তে ঠিক একবার আছে! কিন্তু কেন?

কারণ হচ্ছে array এর বাকি উপাদান গুলো প্রত্যেকে ঠিক দুবার করে আছে, আর Bitwise Xor অপারেশনে দুটি সংখ্যা সমান হলে তারা একে অপরকে কাটাকাটি করে দেয়! যেমন $5 \oplus 5 = 0$, $5 \oplus 13 \oplus 5 = (5 \oplus 5) \oplus 13 = 0 \oplus 13 = 13$। কাজেই পুরো array এর xor নিলে যেসব সংখ্যা দুবার আছে তারা একে অপরকে কাটাকাটি করে দিচ্ছে, যেই সংখ্যাটি একবার ছিল তাকে কেও কাটাকাটি করতে পারছে না! কাজেই পুরো array টির Bitwise Xor ই হচ্ছে আমাদের special সংখ্যা টি!

Array এর xor নির্ণয় করতে আমাদের array টি স্টোর করতে হবে না, কোনো সংখ্যা ইনপুট নেয়ার সাথে সাথেই আমরা xor নিয়ে নিতে পারব! তাই মেমরি ও লাগছে $O(1)$।

```
int n;
cin >> n;
int XOR = 0;
for(int i = 1; i <= n; i++) {
    int x;
    cin >> x;
```

```
    XOR = XOR ^ x;
}
cout << XOR << endl;
```

2.3 Xor and And

Statement

একটি n সাইজের পুর্নসংখ্যার array a দেওয়া আছে($1 \le n \le 10^5$, $a_i \le 10^9$)। এমন কয়টি pair (i, j) আছে যেন $i < j$ এবং $a_i \& a_j \ge a_i \oplus a_j$ হয়।

দুটি সংখ্যা x আর y এর জন্য $x \& y \ge x \oplus y$ হবে কিনা তা আসলে x আর y এর Most Significant Bit (MSB) এর দিকে তাকিয়েই বলে দেয়া যায়।
Most Significant Bit বা MSB হচ্ছে কোনো সংখ্যা x এর বাইনারি রিপ্রেজেন্টেশনে সবচেয়ে বামের বিট যেটি on, অর্থাৎ 1। যেমন $x = (100110)_2$ MSB হচ্ছে 5 তম বিটটি। এখানে সবচেয়ে ডানের বিট টি হচ্ছে 0-তম বিট, তার পরেরটি হচ্ছে 1-তম বিট এবং সবচেয়ে বামের বিটটি হচ্ছে 5-তম বিট বা MSB।
যদি x এর MSB p তম বিট হয়, আর y এর MSB q তম বিট হয় এবং $p > q$ হয়, তবে $x \oplus y$ এর p তম বিট 1 হবে, আর $x \& y$ এর p তম বিট 0 হবে (কারণ y এর p তম বিট 0)। এতে করে $x \oplus y > x \& y$ হবে। $q > p$ এর জন্য ও একই যুক্তি প্রযোজ্য।
যদি $p = q$ হয় তবে $x \oplus y$ এর p তম বিট 0 হবে (কারণ x ও y উভয়ের ই p তম বিট 1)। একই কারণে $x \& y$ এর p তম বিট হবে 1। কাজেই $x \& y > x \oplus y$ হবে! আমাদের এমন pair ই চাই!
কাজেই আমাদের দেখতে হবে কতগুলো pair (i, j) আছে যেন a_i আর a_j এর MSB সমান হয়। কোনো সংখ্যার MSB বের করতে পারলেই বাকি কাজটা খুব সহজ। MSB নির্নয় করতে আমরা x এর বাম থেকে ডানে যেতে থাকব, যতক্ষণ না পর্যন্ত কোনো বিট পাই যেটি on, অর্থাৎ 1 ।

```
int getMSB(int x) {
    for(int i = 29; i >= 0; i--) {
        if((x >> i) & 1) {
            // (x >> i) & 1 is non-zero if i-th bit of x is 1
            return i;
        }
    }

    return -1; // x = 0, hence no MSB exists
}
```

এবার যদি আমরা k টি সংখ্যা পাই যাদের MSB i, তবে আমরা $\binom{k}{2} = \frac{k \times (k-1)}{2}$ টি pair পাব, এভাবে সব বিট এর জন্য মোট pair কতটি আছে তা নির্নয় করে নিবো।

```
int cnt[30] = {};
for(int i = 1; i <= n; i++) cnt[getMSB(a[i])]++;

ll result = 0;

for(int i = 0; i < 30; i++) {
    res += 1ll * cnt[i] * (cnt[i] - 1) / 2;
}
cout << res << endl;
```

2.4 Sum

Statement

তোমার বন্ধু তোমার থেকে একটি array $a_1, a_2, ..a_n$ লুকিয়ে রেখেছে, যেখানে n জোড়। তুমি তাকে দুই ধরনের প্রশ্ন করতে পারবে।
১. তুমি তাকে দুটি index i, j দিবে, উত্তরে সে তোমাকে $(a_i \mid a_j)$ এর মান কত তা বলবে। এখানে $x \mid y$ দ্বারা x ও y এর bitwise or বুঝানো হচ্ছে।
২. তুমি তাকে দুটি index i, j দিবে, উত্তরে সে তোমাকে $(a_i \& a_j)$ কত তা বলবে।
তোমার কাজ হচ্ছে n টি প্রশ্নের মধ্যে array এর সব গুলো সংখ্যার যোগফল নির্নয় করা।

এখানে আমাদের শুধু bitwise or এবং and দেওয়া হবে, তার থেকে পুরো array এর যোগফল বের করতে হবে। তবে কি bitwise or এবং and এর সাথে যোগফল এর কোনো সম্পর্ক রয়েছে? হ্যাঁ নিশ্চয়ই আছে!

মজার ব্যাপার হচ্ছে দুটি সংখ্যা a আর b হলে $a + b = (a \mid b) + (a\&b)$। কিভাবে? a আর b এর প্রতিটি বিট $(a + b)$ তে কতখানি ভূমিকা রাখছে তা দেখার চেষ্টা করি। যদি a আর b উভয়ের i তম বিট 1 হয়, তার মানে a তেও একটি 2^i আছে, b তেও একটি 2^i আছে, তাহলে $(a + b)$ তে নিশ্চয়ই দুটি 2^i থাকবে। অর্থাৎ $(a + b)$ তে i তম বিটের ভূমিকা হবে 2×2^i। আর যদি a আর b এর যেকোনো একটির i তম বিট 1 হয় তবে $(a + b)$ তে i তম বিটের ভূমিকা হবে 1×2^i, আর a, b কারোই i তম বিট 1 না হলে ভূমিকা হবে শূন্য। (লক্ষ্য কর আমি কিন্তু $(a + b)$ এর বাইনারি রিপ্রেজেন্টেশনের কথা বলছি না, a আর b এর বিট গুলো $(a + b)$ তে কেমন ভূমিকা রাখছে তা বলছি)।

a আর b উভয়ের ই i তম বিট 1 হলে, $(a \mid b)$ এবং $(a\&b)$ উভয়ের ই i তম বিট 1 হবে, তাই তাদের যোগফলে i তম বিটের ভূমিকা হবে 2×2^i। আর a আর b এর যেকোনো একটির i তম বিট 1 হলে $(a \mid b)$ এর i তম বিট 1 হলেও $(a\&b)$ এর i তম বিট 0 হবে, কাজেই তাদের যোগফল এ i তম বিটের ভূমিকা 1×2^i। বুঝাই যাচ্ছে সব ক্ষেত্রেই প্রতি বিটের ভূমিকা উভয় পক্ষে একই থাকছে!

কাজেই আমরা দুটি সংখ্যার bitwise or এবং and থেকে যোগফল বের করে ফেলতে পারি! যোগফল $S = a_1 + a_2 + ... + a_n = (a_1 + a_2) + (a_3 + a_4) + ... + (a_{n-1} + a_n)$। এভাবে আমরা পাশাপাশি সংখ্যা গুলো কে নিয়ে $\frac{n}{2}$ টি pair বানাতে পারি। কাজেই আমরা $\frac{n}{2}$ টি pair এর প্রতিটির যোগফল আলাদা করে নির্নয় করে নিলেই তা থেকে S পেয়ে যাব। আর একটি pair এর জন্য খরচ হচ্ছে দুটি প্রশ্ন (একটি bitwise or এর জন্য, আরেকটি and এর জন্য)। সব মিলিয়ে প্রশ্ন লাগছে $\frac{n}{2} \times 2 = n$ টি প্রশ্ন!

বোনাস 1 : $2 \times n$ টি প্রশ্নের মধ্যে array এর প্রতিটি উপাদানের মান বের করতে পারবে?
বোনাস 2 : প্রমাণ কর $a + b = (a \oplus b) + 2 \times (a\&b)$।