

Contents

1 Basic	1	4 String	9
1.1 Default Code	1	4.1 trie	9
1.2 PBDS	1	4.2 KMP	9
1.3 int128 Input Output	1	4.3 Hash	9
1.4 Python	1	4.4 Zvalue	9
1.5 bitset	2	4.5 最長迴文子字串	9
2 Math	2	4.6 Suffix Array	10
2.1 質數表	2	4.7 AC-Automatan	10
2.2 快速冪	2	5 Geometry	10
2.3 擴展歐幾里得	2	5.1 Point	10
2.4 矩陣	2	5.2 內積, 外積, 距離	10
2.5 Miller rabin Prime test	2	5.3 向量應用	11
2.6 Pollard's Rho	3	5.4 Static Convex Hull	11
2.7 皮薩諾定理	3	5.5 外心, 最小覆蓋圓	11
2.8 高斯消去法	3	5.6 四邊形旋轉	12
2.9 卡特蘭數	4	5.7 旋轉	12
2.10 中國剩餘定理	4	5.8 極座標轉直角座標	12
2.11 Theorem	4	5.9 直角座標轉極座標	12
2.12 Estimation	5	6 Data Structure	12
2.13 Euclidean Algorithms	5	6.1 Sparse Table	12
2.14 General Purpose Numbers	5	6.2 Segement Tree	12
2.15 Tips for Generating Functions	5	6.3 Discrete Segement Tree	13
3 Graph	5	6.4 Link Cut Tree	14
3.1 DSU	5	6.5 BIT	15
3.2 Dijkstra	5	6.6 2D BIT	15
3.3 SPFA	6	6.7 undo DSU	15
3.4 Floyd Warshell	6	7 Dynamic Programing	15
3.5 Tarjan SCC	6	7.1 LCS	15
3.6 2 SAT	6	7.2 LIS	16
3.7 Euler Path	7	7.3 Knapsack	16
3.8 Bridge	7	7.4 位元 dp	16
3.9 Max flow min cut	7	7.5 經典 dp 轉移式	16
3.10 Minimum cost maximum flow	7	8 Divide and conquer	16
3.11 二分圖	8	8.1 逆序數對	16
3.12 Check cycle	8	8.2 Mo's algorithm	17
3.13 BCC	8	9 Tree	17
		9.1 樹直徑	17
		9.2 LCA	17
		9.3 樹壓平	17
		9.4 樹鏈剖分	18
		10 Else	19
		10.1 Big Number	19
		10.2 Tenary Search	19
		10.3 Duipai	19
		10.4 Random Generator	19

1 Basic

1.1 Default Code

```
#include <bits/stdc++.h>
#define int long long
#define endl '\n' // 如果是互動題要把這個註解掉
#define de(x) cout << #x << '=' << x << ", "
#define dd cout << '\n';
// #pragma GCC target("popcnt")
// #pragma GCC optimize("O3")
using namespace std;
int tt = 1;

void pre() {
    cout.tie(nullptr); // 輸出加速
    cin >> tt; // 多筆輸入
}

void solve() {}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    #ifdef LOCAL
        // g++ -DLOCAL -std=c++17 <filename> && ./a.out
        freopen("input.txt", "r", stdin);
        // freopen("output.txt", "w", stdout);
    #endif // LOCAL
    pre();
    while (tt--) { solve(); }
    return 0;
}
```

1.2 PBDS

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
```

```
using namespace std;

template
<class T> using Tree = tree<T, null_type, less<T>,
>, rb_tree_tag, tree_order_statistics_node_update>;

/*
如果有 define int long long 記得拿掉
Tree<int> t 就跟 set<int> t 一樣, 有包好 template
rb_tree_tag 使用紅黑樹
第三個參數 less<T> 為由小到大, greater<T> 為由大到小
插入 t.insert(); 刪除 t.erase();
t.order_of_key
(k); 從前往後數 k 是第幾個 (0-base 且回傳 int 型別)
t.find_by_order(k);
從前往後數第 k 個元素 (0-base 且回傳 iterator 型別)
t.lower_bound
(); t.upper_bound(); 用起來一樣 回傳 iterator
可以用 Tree<pair<int, int>> T 來模擬 multiset
*/
```

1.3 int128 Input Output

```
// 抄 BBuf github 的

#include <bits/stdc++.h>
using namespace std;

void scan(__int128 &x) // 輸入
{
    x = 0;
    int f = 1;
    char ch;
    if((ch = getchar()) == '-') f = -f;
    else x = x*10 + ch - '0';
    while((ch = getchar()) >= '0' && ch <= '9')
        x = x*10 + ch - '0';
    x *= f;
}

void print(__int128 x) // 輸出
{
    if(x < 0)
    {
        x = -x;
        putchar('-');
    }
    if(x > 9) print(x/10);
    putchar(x%10 + '0');
}

int main()
{
    __int128 a, b;
    scan(a);
    scan(b);
    print(a + b);
    puts("");
    print(a*b);
    return 0;
}
```

1.4 Python

```
## Input
# p q 都是整數, 中間以空白分開輸入
p, q = map(int, input().split())

# 輸入很多個用空
白隔開的數字, 轉成 float 放進陣列, s 是 input 字串
arr = list(map(float, s.split()))

# 分數用法 Fraction(被除數, 除數)
from fractions import Fraction

frac = Fraction(3, 4)
numerator = frac.numerator # 取出分子
denominator = frac.denominator # 取出分母

arr = [Fraction
(0), Fraction(1, 6), Fraction(1, 2), Fraction(5,
12), Fraction(0), Fraction(-1, 12), Fraction(0)]
```

```
# 可以直接做乘除
def fx(x):
    x = Fraction(x)
    ans = Fraction(0)
    for i in range(1, 7):
        ans += arr[i] * x ** (7 - i)
    return ans
```

1.5 bitset

bitset<size> b(a): 長度為size，初始化為a
 b[i]: 第i位元的值(0 or 1)
 b.size(): 有幾個位元
 b.count(): 有幾個1
 b.set(): 所有位元設為1
 b.reset(): 所有位元設為0
 b.flip(): 所有位元反轉

2 Math

2.1 質數表

```
vector<int> prime_table(int n){
    vector<int> table(n + 1, 0);
    for(int i = 1; i <= n; i++){
        for(int j = i; j <= n; j += i){
            table[j]++;
        }
    }
    return table;
}
```

2.2 快速冪

```
#define int long long

// 根據費馬小定
// 理，若 a, p 互質，a^(p-2) 為 a 在 mod p 時的乘法逆元
// a ^ (b ^ c)
// ) % mod = fast_pow(a, fast_pow(b, c, mod - 1), mod)
typedef unsigned long long ull;
inline int ksc(ull
    x, ull y, int p) { // 0(1)快速乘 (防爆 long long)
    return (x
        * y - (ull)((long double)x / p * y) * p + p) % p;
}

inline int fast_pow(int a, int b, int mod)
{
    // a^b % mod
    int res = 1;
    while(b)
    {
        if(b & 1) res = ksc(res, a, mod);
        a = ksc(a, a, mod);
        b >>= 1;
    }
    return res;
}
```

2.3 擴展歐幾里得

```
int gcd(int a, int b)
{
    return b == 0 ? a : gcd(b, a % b);
}

int lcm(int a, int b)
{
    return a * b / gcd(a, b);
}

pair<int, int> ext_gcd
    (int a, int b) //擴展歐幾里德 ax+by = gcd(a,b)
{
    if (b == 0)
        return {1, 0};
    if (a == 0)
        return {0, 1};
    int x, y;
    tie(x, y) = ext_gcd(b % a, a);
    return make_pair(y - (b / a) * x, x);
}
```

2.4 矩陣

```
template<typename T>
struct Matrix{
    using rt = std::vector<T>;
    using mt = std::vector<rt>;
    using matrix = Matrix<T>;
    int r, c;
    mt m;
    Matrix(int r, int c): r(r), c(c), m(r, rt(c)){}
    rt& operator[](int i){return m[i];}
    matrix operator+(const matrix &a){
        matrix rev(r, c);
        for(int i=0; i<r; ++i)
            for(int j=0; j<c; ++j)
                rev[i][j] = m[i][j] + a.m[i][j];
        return rev;
    }
    matrix operator-(const matrix &a){
        matrix rev(r, c);
        for(int i=0; i<r; ++i)
            for(int j=0; j<c; ++j)
                rev[i][j] = m[i][j] - a.m[i][j];
        return rev;
    }
    matrix operator*(const matrix &a){
        matrix rev(r, a.c);
        matrix tmp(a.c, a.r);
        for(int i=0; i<a.r; ++i)
            for(int j=0; j<a.c; ++j)
                tmp[j][i] = a.m[i][j];
        for(int i=0; i<r; ++i)
            for(int j=0; j<a.c; ++j)
                for(int k=0; k<c; ++k)
                    rev.m[i][j] += m[i][k] * tmp[j][k];
        return rev;
    }
    bool inverse(){
        Matrix t(r, r+c);
        for(int y=0; y<r; y++){
            t.m[y][c+y] = 1;
            for(int x=0; x<c; ++x)
                t.m[y][x] = m[y][x];
        }
        if(!t.gas())
            return false;
        for(int y=0; y<r; y++){
            for(int x=0; x<c; ++x)
                m[y][x] = t.m[y][c+x] / t.m[y][y];
            return true;
        }
    }
    T gas(){
        vector<T> lazy(r, 1);
        bool sign=false;
        for(int i=0; i<r; ++i){
            if(m[i][i]==0){
                int j=i+1;
                while(j<r && !m[j][i]) j++;
                if(j==r) continue;
                m[i].swap(m[j]);
                sign=!sign;
            }
            for(int j=0; j<r; ++j){
                if(i==j) continue;
                lazy[j] = lazy[j] * m[i][i];
                T mx = m[j][i];
                for(int k=0; k<c; ++k)
                    m[j][k] = m[j][k] * m[i][i] - m[i][k] * mx;
            }
        }
        T det = sign ? -1 : 1;
        for(int i=0; i<r; ++i){
            det = det * m[i][i];
            det = det / lazy[i];
            for(auto &j: m[i]) j /= lazy[i];
        }
        return det;
    }
};
```

2.5 Miller rabin Prime test

```
// fast_pow 去前面抄，需要處理防暴乘法
// 記得 #define int long long 也要放
// long long 範圍內測試過答案正確
// time: O(logn)
```

```

inline bool mr(int x, int p) {
    if (fast_pow(x, p - 1, p) != 1) return 0;
    int y = p - 1, z;
    while (!(y & 1)) {
        y >>= 1;
        z = fast_pow(x, y, p);
        if (z != 1 && z != p - 1) return 0;
        if (z == p - 1) return 1;
    }
    return 1;
}

inline bool prime(int x) {
    if (x < 2) return 0;
    if (x == 2 ||
        x == 3 || x == 5 || x == 7 || x == 43) return 1;
    // 如果把 2
    // 到 37 前 12 個質數都檢查一遍 可以保證 2^78 皆可用
    return mr(2, x)
        && mr(3, x) && mr(5, x) && mr(7, x) && mr(43, x);
}

```

2.6 Pollard's Rho

```

// 主函數記得放 srand(time(nullptr))
// prime 檢測以及快速冪, gcd 等請從前面抄

// 輸入一個數字 p, 隨
// 機回傳一個 非 1 非 p 的因數, 若 p 是質數會無窮迴圈
#define rg register int
inline int rho(int p) {
    int x, y, z, c, g;
    rg i, j;
    while (1) {
        y = x = rand() % p;
        z = 1;
        c = rand() % p;
        i = 0, j = 1;
        while (++i) {
            x = (ksc(x, x, p) + c) % p;
            z = ksc(z, abs(y - x), p);
            if (x == y || !z) break;
            if (!(i % 127) || i == j) {
                g = gcd(z, p);
                if (g > 1) return g;
                if (i == j) y = x, j <= 1;
            }
        }
    }
}

// 回傳隨機一個質因數, 若 input 為質數, 則直接回傳
int prho(int p) {
    if (prime(p)) return p;
    int m = rho(p);
    if (prime(m)) return m;
    return prho(p / m);
}

// 回傳將 n 質因數分解的結果, 由小到大排序
// ex: input: 48, output: 2 2 2 2 3
vector<int> prime_factorization(int n) {
    vector<int> ans;
    while (n != 1) {
        int m = prho(n);
        ans.push_back(m);
        n /= m;
    }
    sort(ans.begin(), ans.end());
    return ans;
}

```

2.7 皮薩諾定理

```

// fib(x) % m = fib(x + kn) % m 當 k >= 1, 求 n
// n 為費式數列 % m 會重複的週期
// pisano_period(m) <= 6m
// 通常這都要本地跑

#define int long long

int pisano_period(int m) {
    int pre = 0, cur = 1;
    int temp;

```

```

for (int i = 0; i < m * m; i++) {
    temp = pre;
    pre = cur;
    cur = (temp + cur) % m;
    if (pre == 0 && cur == 1) return i + 1;
}
return 0;
}

```

2.8 高斯消去法

```

from fractions import Fraction
def gauss_elimination(matrix, results):
    # 將所有數字轉換為分數
    n = len(matrix)
    augm = [[Fraction(matrix
        [i][j]) for j in range(n)] for i in range(n)]
    augr = [Fraction(results[i]) for i in range(n)]

    # 高斯消去法
    for i in range(n):
        # 尋找主元
        if augm[i][i] == 0:
            for j in range(i + 1, n):
                if augm[j][i] != 0:
                    augm[i], augm[j] = augm[j], augm[i]
                    augr[i], augr[j] = augr[j], augr[i]
                    break

        pivot = augm[i][i]
        if pivot == 0:
            # 如果主元為0, 繼續檢查該行是否全為 0
            if all(augm[i][j] == 0 for j in range(n)):
                if augr[i] != 0:
                    return None # 無解
                else:
                    continue
            # 可能有無限多解, 繼續檢查

        # 將主元行的數字規一化
        for j in range(i, n):
            augm[i][j] /= pivot
            augr[i] /= pivot

        # 將其他行的數字變為0
        for j in range(n):
            if i != j:
                factor = augm[j][i]
                for k in range(i, n):
                    augm[j][k] -= factor * augm[i][k]
                augr[j] -= factor * augr[i]

    # 檢查是否存在無限多解的情況
    for i in range(n):
        if all(augm[i][j]
            [j] == 0 for j in range(n)) and augr[i] != 0:
            return [] # 無限多組解

    return augr

# matrix = [
#     [2, -1, 1],
#     [3, 3, 9],
#     [3, 3, 5]
# ]
# results = [8, -42, 0]
# output = [
#     Fraction(12, 1), Fraction(11, 2), Fraction(-21, 2)]
# Fraction 可以強轉 float

import numpy as np

def gauss_elimination(matrix, ans):
    matrix = np.array(matrix)
    ans = np.array(ans)

    try:
        solution = np.linalg.solve(matrix, ans)
        return [f"{value:.2f}" for value in solution]
    except np.linalg.LinAlgError:
        # 無解或者無限多組解
        return "No Solution"

```

- # 有開放 numpy 可以用
- # 優點：行數短，執行速度快
- # 缺點：只能用浮點數，無法區分無解及無限多組解

2.9 卡特蘭數

```

"""
卡特蘭數 Catalan
公式：H(n) = C(2 * n, n) // (n + 1), n >= 2, n 為正整數
快速計算方式：
1. H(0) = H(1) = 1, H(n)
   = sum(H(i - 1) * H(n - i) for i in range(1, n + 1))
2. H(n) = H(n - 1) * (4 * n - 2) // (n + 1)
3. H(n) = C(2 * n, n) - C(2 * n, n - 1)
"""

"""
可解問題：
有效括號匹配問題：
    給定 n 個左括號與右括號，求有幾種不同的正確括號匹配
二元樹結構問題：給定 n 個節點，求有幾種不同的二元樹結構
將一個凸
    n + 2 邊形劃分成多個三角形，求有幾種不同的劃分方式
狄克路徑：給定 n * n 的網格，
    從左下到右上的路徑中，永不超過對角線的路徑有幾種
一個 stack 在 push 順
    序不變的情況下 (1, 2, 3, ..., n)，有幾種 pop 的方式
在圖上選擇 2 * n 個
    點，將這些點兩兩連接使得 n 條線段不相交的方法有幾種
"""

n = int(input())

catalan = [1 for _ in range(n + 1)]
for i in range(1, n + 1):
    catalan[i] = catalan[i - 1] * (4 * i - 2) // (i + 1)

ans = 0
for i in range(0, n + 1): # 卡特蘭數的平方
    ans += catalan[i] * catalan[n - i]

print(ans)
# 185ms in codeforces, n <= 5000

```

2.10 中國剩餘定理

```

// vec[i] = {m_i, x_i}, 求最小非負 x
// 使得 x ≡ x_i (mod m_i) 對所有 i 同時成立；無解回 -1
// 注意 overflow
int CRT(vector<pair<int, int>> &v)
{
    int m = v[0].first, x = (v[0].second % m + m) % m;
    for (int i = 1; i < (int)v.size(); ++i)
    {
        int mi =
            v[i].first, xi = (v[i].second % mi + mi) % mi;
        int g = gcd(m, mi), d = xi - x;
        if (d % g) return -1;
        int m1 = m / g, m2 = mi / g;
        auto ab = ext_gcd((int)m1, (int)m2);
        int inv = ((int)ab.first % m2 + m2) % m2;
        int k = ((d / g) % m2 + m2) % m2;
        k = (k * inv) % m2;
        x = (x + m * k) % (m * m2);
        m *= m2;
        x = (x + m) % m;
    }
    return x;
}

```

2.11 Theorem

- Cramer 法則

$$\begin{aligned} ax+by &= e & x &= \frac{ed-bf}{ad-bc} \\ cx+dy &= f & y &= \frac{af-ec}{ad-bc} \end{aligned}$$

- Vandermonde 恆等式

$$C(n+m, k) = \sum_{i=0}^k C(n, i) C(m, k-i)$$

- Kirchhoff 定理
設 L 為圖 G 的 $n \times n$ Laplacian 矩陣，其中 $L_{ii} = d(i)$ ， $L_{ij} = -c$ ， c 為邊 (i, j) 的條數。
 - 無向圖的生成樹數為 $|\det(\tilde{L}_{11})|$ 。
 - 以 r 為根的有向生成樹數為 $|\det(\tilde{L}_{rr})|$ 。
- Tutte 矩陣
令 D 為 $n \times n$ 矩陣，當 $i < j$ 且 $(i, j) \in E$ 時令 $d_{ij} = x_{ij}$ (x_{ij} 由均勻隨機選取)，否則令 $d_{ij} = -d_{ji}$ 。則 $\frac{\text{rank}(D)}{2}$ 為圖 G 的最大匹配數。

- Cayley 公式
對於帶標號頂點、度數序列為 d_1, d_2, \dots, d_n 的情形，生成樹數為 $\frac{(d_1-1)!(d_2-1)!\dots(d_n-1)!}{(n-2)!}$ 。
 - 設 $T_{n,k}$ 為在 n 頂點、 k 個連通分量的帶標號森林數，且頂點 $1, 2, \dots, k$ 分屬不同分量，則 $T_{n,k} = kn^{n-k-1}$ 。
- Erdős–Gallai 定理
非負整數序列 $d_1 \geq \dots \geq d_n$ 可表示為 n 頂點簡單圖的度數序列，當且僅當 $d_1 + \dots + d_n$ 為偶數，且對每個 $1 \leq k \leq n$ 皆有

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k).$$

- Gale–Ryser 定理
非負整數序列 $a_1 \geq \dots \geq a_n$ 與 b_1, \dots, b_n 構成雙圖度數序列 (bigraphic) 的充要條件為

$$\sum_{i=1}^n a_i = \sum_{i=1}^n b_i \quad \text{且} \quad \sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k) \quad \text{對所有} 1 \leq k \leq n.$$

- Fulkerson–Chen–Anstee 定理
一序列 $(a_1, b_1), \dots, (a_n, b_n)$ (非負整數對，且 $a_1 \geq \dots \geq a_n$) 為有向圖可實現之入出度序列，當且僅當

$$\sum_{i=1}^n a_i = \sum_{i=1}^n b_i \quad \text{且} \quad \sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k) \quad \text{對所有} 1 \leq k < n.$$

- Pick 定理
對所有頂點為整數點的單純多邊形，有

$$A = \#\{\text{內部格點}\} + \frac{\#\{\text{邊界格點}\}}{2} - 1.$$

- 莫比烏斯反演公式
 - $f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right)$
 - $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) f(d)$
- 球冠 (Spherical cap)
 - 球面被平面切下的一部分。
 - r : 球半徑, a : 球冠底面半徑, h : 球冠高度, $\theta = \arcsin(a/r)$ 。
 - 體積 $= \frac{\pi h^2(3r-h)}{3} = \frac{\pi h(3a^2+h^2)}{6} = \frac{\pi r^3(2+\cos\theta)(1-\cos\theta)^2}{3}$ 。
 - 表面積 $= 2\pi r h = \pi(a^2+h^2) = 2\pi r^2(1-\cos\theta)$ 。
- 拉格朗日乘數法
 - 在 k 個約束 $g_i(x_1, \dots, x_n) = 0$ 下優化 $f(x_1, \dots, x_n)$ 。
 - 拉格朗日函數 $\mathcal{L}(x_1, \dots, x_n, \lambda_1, \dots, \lambda_k) = f(x_1, \dots, x_n) - \sum_{i=1}^k \lambda_i g_i(x_1, \dots, x_n)$ 。
 - 對應原始受限問題的解是拉格朗日函數的一個鞍點。
- 兩條錯線的最近點
 - Line 1: $v_1 = p_1 + t_1 d_1$
 - Line 2: $v_2 = p_2 + t_2 d_2$
 - $n = d_1 \times d_2$
 - $n_1 = d_1 \times n$
 - $n_2 = d_2 \times n$
 - $c_1 = p_1 + \frac{(p_2 - p_1) \cdot n_2}{d_1 \cdot n_2} d_1$
 - $c_2 = p_2 + \frac{(p_1 - p_2) \cdot n_1}{d_2 \cdot n_1} d_2$

- 導數/積分
 分部積分： $\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$
 $\left| \begin{aligned} \frac{d}{dx} \sin^{-1}x &= \frac{1}{\sqrt{1-x^2}} & \frac{d}{dx} \cos^{-1}x &= -\frac{1}{\sqrt{1-x^2}} & \frac{d}{dx} \tan^{-1}x &= \frac{1}{1+x^2} \\ \frac{d}{dx} \tan x &= 1 + \tan^2 x & \int \tan(ax) dx &= -\frac{\ln|\cos(ax)|}{a} \\ \int e^{-x^2} dx &= \frac{\sqrt{\pi}}{2} \text{erf}(x) & \int x e^{ax} dx &= \frac{e^{ax}}{a^2} (ax-1) \\ \int \sqrt{a^2+x^2} dx &= \frac{1}{2} \left(x\sqrt{a^2+x^2} + a^2 \text{asinh}(x/a) \right) \end{aligned} \right|$
- 球座標 (Spherical Coordinate)

$$(x, y, z) = (r \sin \theta \cos \phi, r \sin \theta \sin \phi, r \cos \theta)$$

$$(r, \theta, \phi) = (\sqrt{x^2 + y^2 + z^2}, \arccos(z/\sqrt{x^2 + y^2 + z^2}), \text{atan2}(y, x))$$

- 旋轉矩陣

$$M(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}, \quad R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x \\ 0 & \sin \theta_x & \cos \theta_x \end{bmatrix}$$

2.12 Estimation

n	2	3	4	5	6	7	8	9	20	30	40	50	100		
$p(n)$	2	3	5	7	11	15	22	30	627	5604	4e4	2e5	2e8		
n	100	1e3	1e6	1e9	1e12	1e15	1e18								
$d(i)$	12	32	240	1344	6720	26880	103680								
n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\binom{2n}{n}$	2	6	20	70	252	924	3432	12870	48620	184756	7e5	2e6	1e7	4e7	1.5e8
n	2	3	4	5	6	7	8	9	10	11	12	13			
B_n	2	5	15	52	203	877	4140	21147	115975	7e5	4e6	3e7			

2.13 Euclidean Algorithms

- $m = \lfloor \frac{an+b}{c} \rfloor$
- Time complexity: $O(\log n)$

$$f(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor$$

$$= \begin{cases} \lfloor \frac{a}{c} \rfloor \cdot \frac{n(n+1)}{2} + \lfloor \frac{b}{c} \rfloor \cdot (n+1) + f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ nm - f(c, c-b-1, a, m-1), & \text{otherwise} \end{cases}$$

$$g(a, b, c, n) = \sum_{i=0}^n i \lfloor \frac{ai+b}{c} \rfloor$$

$$= \begin{cases} \lfloor \frac{a}{c} \rfloor \cdot \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor \cdot \frac{n(n+1)}{2} + g(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ \frac{1}{2} \cdot (n(n+1)m - f(c, c-b-1, a, m-1) - h(c, c-b-1, a, m-1)), & \text{otherwise} \end{cases}$$

$$h(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor^2$$

$$= \begin{cases} \lfloor \frac{a}{c} \rfloor^2 \cdot \frac{n(n+1)(2n+1)}{6} + 2 \lfloor \frac{a}{c} \rfloor \cdot \lfloor \frac{b}{c} \rfloor \cdot \frac{n(n+1)}{2} + h(a \bmod c, b \bmod c, c, n) + 2 \lfloor \frac{a}{c} \rfloor \cdot g(a \bmod c, b \bmod c, c, n) + 2 \lfloor \frac{b}{c} \rfloor \cdot f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ nm(m+1) - 2g(c, c-b-1, a, m-1) - 2f(c, c-b-1, a, m-1) - f(a, b, c, n), & \text{otherwise} \end{cases}$$

2.14 General Purpose Numbers

- 白努力數 (Bernoulli numbers)

$$B_0 = 1, B_1^{\pm} = \pm \frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$$

滿足關係式 $\sum_{j=0}^m \binom{m+1}{j} B_j = 0$ 。其指數生成函數 (EGF) 為

$$B(x) = \frac{x}{e^x - 1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}.$$

且

$$S_m(n) = \sum_{k=1}^n k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k^+ n^{m+1-k}.$$

- 第二類史特林數 (Stirling numbers of the second kind)

表示將 n 個不同元素分成恰好 k 個非空集合的劃分數。遞迴式：

$$S(n, k) = S(n-1, k-1) + kS(n-1, k), \quad S(n, 1) = S(n, n) = 1.$$

顯式公式：

$$S(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n.$$

轉換關係：

$$x^n = \sum_{i=0}^n S(n, i) (x)_i.$$

- 五角數定理 (Pentagonal number theorem)

$$\prod_{n=1}^{\infty} (1 - x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k \left(x^{k(3k+1)/2} + x^{k(3k-1)/2} \right).$$

此恆等式由歐拉提出，與分割函數的生成函數密切相關。

- 卡塔蘭數 (Catalan numbers)

$$C_n^{(k)} = \frac{1}{(k-1)n+1} \binom{kn}{n}, \quad C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k.$$

其中 $C_n^{(2)}$ 為經典卡塔蘭數，常出現在二元樹、括號配對、Dyck path 等組合結構中。

- 歐拉數 (Eulerian numbers)

表示排列 $\pi \in S_n$ 中，恰有 k 個元素比前一個元素大的排列數。滿足遞迴式：

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k), \quad E(n, 0) = E(n, n-1) = 1.$$

顯式公式：

$$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n.$$

2.15 Tips for Generating Functions

- Ordinary Generating Function $A(x) = \sum_{i \geq 0} a_i x^i$

- $A(rx) \Rightarrow r^n a_n$
- $A(x) + B(x) \Rightarrow a_n + b_n$
- $A(x)B(x) \Rightarrow \sum_{i=0}^n a_i b_{n-i}$
- $A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k}$
- $x A(x)' \Rightarrow n a_n$
- $\frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^n a_i$

- Exponential Generating Function $A(x) = \sum_{i \geq 0} \frac{a_i}{i!} x^i$

- $A(x) + B(x) \Rightarrow a_n + b_n$
- $A^{(k)}(x) \Rightarrow a_{n+k}$
- $A(x)B(x) \Rightarrow \sum_{i=0}^n \binom{n}{i} a_i b_{n-i}$
- $A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} \binom{n}{i_1, i_2, \dots, i_k} a_{i_1} a_{i_2} \dots a_{i_k}$
- $x A(x) \Rightarrow n a_n$

- Special Generating Function

- $(1+x)^n = \sum_{i \geq 0} \binom{n}{i} x^i$
- $\frac{1}{(1-x)^n} = \sum_{i \geq 0} \binom{n-1}{i} x^i$

3 Graph

3.1 DSU

```
class dsu{
public:
    vector<int> parent;
    dsu(int num){
        parent.resize(num);
        for(int i = 0; i < num; i++) parent[i] = i;
    }
    int find(int x){
        if(parent[x] == x) return x;
        return parent[x] = find(parent[x]);
    }
    bool same(int a, int b){
        return find(a) == find(b);
    }
    void Union(int a, int b){
        parent[find(a)] = find(b);
    }
};
```

3.2 Dijkstra

```
// 傳入圖的 pair 為 {權重, 點}, 無限大預設 1e9 是情況改
#define pii pair<int, int>
vector<
    int> dijkstra(vector<vector<pii>> &graph, int src){
    int n = graph.size();
    vector<int> dis(n, 1e9);
    vector<bool> vis(n, false);
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push({0, src});
    dis[src] = 0;
    while(!pq.empty()){
        auto [w, node] = pq.top();
        pq.pop();
        if(vis[node]) continue;
        vis[node] = true;
        for(auto [nw, nn]: graph[node]){
            if(w + nw < dis[nn]){
                dis[nn] = w + nw;
                pq.push({dis[nn], nn});
            }
        }
    }
    return dis;
}
```


3.3 SPFA

```
#define pii pair<int, int>
// {在 src 可到達
  的點中是否存在負環，最短路徑}, arg 中 n 為點的數量
// arg 中 pair 裡的第一個值為權重，第二個為點
pair<bool, vector<int>>
SPFA(vector<vector<pii>> &graph, int n, int src){
    vector<int> dis(n + 1, 1e9);
    vector<int> cnt(n + 1, 0);
    vector<bool> vis(n + 1, false);
    queue<int> q;
    vis[src] = true; q.push(src); dis[src] = 0;
    while(!q.empty()){
        auto node = q.front(); vis[node] = false; q.pop();
        for(auto [w, nn]:graph[node]){
            if(w + dis[node] < dis[nn]){
                dis[nn] = w + dis[node];
                if(!vis[nn]){
                    if(++cnt[nn] >= n) return {true, {}};
                    q.push(nn);
                    vis[nn] = true;
                }
            }
        }
    }
    return {false, dis};
}
```

3.4 Floyd Warshell

```
// 中繼點放外面
for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
        }
    }
}
```

3.5 Tarjan SCC

```
class tarjan{
    // 1-base
    int time = 1;
    int id = 1;
    stack<int> s;
    vector<int> low;
    vector<int> dfn;
    vector<bool> in_stack;
    void dfs(int node, vector<vector<int>> &graph){
        in_stack[node] = true;
        s.push(node);
        dfn[node] = low[node] = time++;
        for(auto &j : graph[node]){
            if(dfn[j] == 0){
                dfs(j, graph);
                // 看看往下有沒有辦法回到更上面的點
                low[node] = min(low[node], low[j]);
            }
            else if(in_stack[j]){
                low[node] = min(low[node], low[j]);
            }
        }
        vector<int> t; // 儲存這個強連通分量
        if(dfn[node] == low[node]){
            while(s.top() != node){
                t.push_back(s.top());
                in_stack[s.top()] = false;
                scc_id[s.top()] = id;
                s.pop();
            }
            t.push_back(s.top());
            scc_id[s.top()] = id;
            in_stack[s.top()] = false;
            s.pop();
            id++;
        }
        if(!t.empty()) ans.push_back(t);
    }
public:
    vector<int> scc_id;
    vector<vector<int>> ans;
}
```

```
// ans ans[i] 代表第 i 個強連通分量裡面包涵的點
// scc_id[i] 代表第 i 個點屬於第幾個強連通分量
vector
vector<vector<int>> scc(vector<vector<int>> &graph){
    int num = graph.size();
    scc_id.resize(num, -1);
    dfn.resize(num, 0);
    low.resize(num, 0);
    in_stack.resize(num, false);
    for(int i = 1; i < num; i++){
        if(dfn[i] == 0) dfs(i, graph);
    }
    return ans;
}
```

3.6 2SAT

```
// (a || b) && (c || d) && (e || f) .....
```

```
// 用
```

下面的 tarjan scc 算法來解 2 sat 問題，若事件 a 發生時，事件 b 必然發生，我們須在 a -> b 建立一條有向

```
// 用
```

cses 的 Giant Pizza 來舉例子，給定 n 個人 m 個配料表，每個人可以提兩個要求，兩個要求至少要被滿足一個

```
// 3 5
// + 1 + 2
// - 1 + 3
// + 4 - 2
```

```
// 以這
```

個例子來說，第一個人要求要加 配料1 或者 配料2 其中一項，第二個人要求不要 配料1 或者 要配料3 其中一項
 // 試問能不能滿足所有人的要求，我們可以把 要加 配料 i 當作點 i，不加配料 i 當作點 i + m(配料數量)
 // 關於第一個人的要求 我們可以看成若不加 配料1 則必定要 配料2 以及 若不加 配料2 則必定要 配料1
 // 關於第二個人要求 可看做加了 配料1 就必定要加 配料3 以及 不加 配料3 就必定不加 配料1

```
// 以這些條件建立有向圖，並且
```

找尋 scc，若 i 以及 i + m 在同一個 scc 中代表無解
 // 若要求解，則若 i 的 scc_id 小於 i + m 的 scc_id 則 i 為 true，反之為 false

```
// tarjan 的模板在上面
```

```
cin >> n >> m;
```

```
vector<vector<int>> graph(m * 2 + 1);
function<int(int)> tr = [&](int x){
    if(x > m) return x - m;
    return x + m;
};
```

```
for(int i = 0; i < n; i++){
    char c1, c2;
    int a, b;
    cin >> c1 >> a >> c2 >> b;
    // a 代表 a 為真，m + a 代表 a 為假
    if(c1 == '-') a += m;
    if(c2 == '-') b += m;
    graph[tr(a)].push_back(b);
    graph[tr(b)].push_back(a);
}
```

```
tarjan t;
auto scc = t.scc(graph);

for(int i = 1; i <= m; i++){
    if(t.scc_id[i] == t.scc_id[tr(i)]){
        cout << "IMPOSSIBLE\n";
        return 0;
    }
}
```

```
for(int i = 1; i <= m; i++){
    if(t.scc_id[i] < t.scc_id[tr(i)]){
        cout << '+';
    }
    else cout << '-';
    cout << ' ';
}
```

```
cout << '\n';
```

3.7 Euler Path

```
// 1. 無向圖是歐拉圖：
// 非零度頂點是連通的
// 頂點的度數都是偶數

// 2. 無向圖是半歐拉圖(有路沒有環)：
// 非零度頂點是連通的
// 恰有 2 個奇度頂點

// 3. 有向圖是歐拉圖：
// 非零度頂點是強連通的
// 每個頂點的入度和出度相等

// 4. 有向圖是半歐拉圖(有路沒有環)：
// 非零度頂點是弱連通的
// 至多一個頂點的出度與入度之差為 1
// 至多一個頂點的入度與出度之差為 1
// 其他頂點的入度和出度相等
vector<set<int>> adj;
vector<int> ans;

void dfs(int x) { // Hierholzer's Algorithm
    while (!adj[x].empty()) {
        auto next = *(adj[x].begin());
        adj[x].erase(next);
        adj[next].erase(x);
        dfs(next);
    }
    ans.emplace_back(x);
}

void solve() {
    // 建立雙向邊，set用來防重邊，點數n，邊數m
    for (int i = 1; i <= n; i++)
        if (adj[i].size() & 1) return; /* impossible */
    dfs(1);
    if (ans.size() != m + 1) return; /* impossible */
    reverse(ans.begin(), ans.end()); /* then print it */
}
```

3.8 Bridge

```
// 橋：若移除邊會使連通分量變多
// [USAGE] ECC ecc(n); ecc.add_edge(u, v); ecc.solve();
// is_bridge[i]; necc; bln[i];
// 邊是否為橋；橋連通分量數量；頂點所屬橋連通分量編號
struct ECC { // 0-base
    int n, dft, ecnt, necc;
    vector<int> low, dfn, bln, is_bridge, stk;
    vector<vector<pii>> G;
    void dfs(int u, int f) {
        dfn[u] = low[u] = ++dft, stk.pb(u);
        for (auto [v, e] : G[u])
            if (!dfn[v])
                dfs(v, e), low[u] = min(low[u], low[v]);
            else if (e != f)
                low[u] = min(low[u], dfn[v]);
        if (low[u] == dfn[u]) {
            if (f != -1) is_bridge[f] = 1;
            for (; stk.back() != u; stk.pop_back())
                bln[stk.back()] = necc;
            bln[u] = necc++, stk.pop_back();
        }
    }
    ECC(int _n): n(_n), dft(),
        , ecnt(), necc(), low(n), dfn(n), bln(n), G(n) {}
    void add_edge(int u, int v) {
        G[u].pb(pii(v, ecnt)), G[v].pb(pii(u, ecnt++));
    }
    void solve() {
        is_bridge.resize(ecnt);
        for (int i = 0; i < n; ++i)
            if (!dfn[i]) dfs(i, -1);
    }
}; // 8BQube
```

3.9 Max flow min cut

```
#define int long long

// dicnic Algorithm Time:  $O(V^2E)$  實際上會快一點
```

```
// 記得在 main 裡面 resize graph
// 最小割，找
// 到最少條的邊切除，使得從 src 到 end 的 maxflow 為 0
// 枚舉所有邊  $i \rightarrow j$ ，src 可
// 以到達 i 但無法到達 j，那這條邊為最小割裡的邊之一
// 無向圖最大流：修改 add_edge，反向邊建為 capacity
// 使用時只要 add_edge 一次
```

```
class edge{
public:
    int next;
    int capacity;
    int rev;
    bool is_rev;
    edge(int _n, int _c, int _r, int _ir) :
        next(_n), capacity(_c), rev(_r), is_rev(_ir){};
};

vector<vector<edge>> graph;
vector<int> level, iter;

void add_edge(int a, int b, int capacity){
    graph[a].push_back
        (edge(b, capacity, graph[b].size(), false));
    graph[b].
        push_back(edge(a, 0, graph[a].size() - 1, true));
}

void bfs(int start) {
    fill(level.begin(), level.end(), -1);
    queue<int> q;
    level[start] = 0;
    q.push(start);
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (auto& e : graph[v]) {
            if (e.capacity > 0 && level[e.next] < 0) {
                level[e.next] = level[v] + 1;
                q.push(e.next);
            }
        }
    }
}

int dfs(int v, int end, int flow) {
    if (v == end) return flow;
    for (int &i = iter[v]; i < graph[v].size(); i++) {
        edge &e = graph[v][i];
        if (e.capacity > 0 && level[v] < level[e.next]) {
            int d = dfs(e.next, end, min(flow, e.capacity));
            if (d > 0) {
                e.capacity -= d;
                graph[e.next][e.rev].capacity += d;
                return d;
            }
        }
    }
    return 0;
}

int maxflow(int start, int end) {
    int flow = 0;
    level.resize(graph.size() + 1);
    while (true) {
        bfs(start);
        if (level[end] < 0) return flow;
        iter.assign(graph.size() + 1, 0);
        int f;
        while ((f = dfs(start, end, 1e9)) > 0) {
            flow += f;
        }
    }
}
```

3.10 Minimum cost maximum flow

```
#define int long long
#define pii pair<int, int>

// Edmonds-Karp Algorithm Time:  $O(VE^2)$  實際上會快一點
// 一條邊的費用為 單位花費 * 流過流量
// 把原本的 BFS 換成 SPFA 而已
// 記得在 main 裡面 resize graph
// MCMF 回傳 {flow, cost}
```

```
// 無向圖：add_edge(u,v,C,W), add_edge(v,u,C,W);

class edge{
public:
    int next;
    int capacity;
    int rev;
    int cost;
    bool is_rev;
    edge(int _n, int _c,
         int _r, int _co, int _ir) : next(_n), capacity
         (_c), rev(_r), cost(_co), is_rev(_ir){};
};

vector<vector<edge>> graph;

void add_edge(int a, int b, int capacity, int cost){
    graph[a].push_back(
        edge(b, capacity, graph[b].size(), cost, false));
    graph[b].push_back
        (edge(a, 0, graph[a].size() - 1, -cost, true));
}

pii dfs(int now
        , int end, pii data, vector<pii> &path, int idx){
    auto [flow, cost] = data;
    if(now == end) return {flow, 0};
    auto &e = graph[now][path[idx + 1].second];
    if(e.capacity > 0){
        auto [ret, nc] = dfs(e.next, end, {min(flow
            , e.capacity), cost + e.cost}, path, idx + 1);
        if(ret > 0){
            e.capacity -= ret;
            graph[e.next][e.rev].capacity += ret;
            return {ret, nc + ret * e.cost};
        }
    }
    return {0, 0};
}

vector<pii> search_path(int start, int end){
    int n = graph.size() + 1;
    vector<int> dis(n + 1, 1e9);
    vector<bool> vis(n + 1, false);
    vector<pii> ans; queue<int> q;
    vis[start] = true; q.push(start); dis[start] = 0;
    vector<pii> parent(graph.size(), {-1, -1});
    q.push(start);
    while(!q.empty()){
        auto node = q.front(); vis[node] = false; q.pop();
        for(int i = 0; i < graph[node].size(); i++){
            auto &e = graph[node][i];
            if(e.capacity
                > 0 and e.cost + dis[node] < dis[e.next]){
                dis[e.next] = e.cost + dis[node];
                parent[e.next] = {node, i};
                if(!vis[e.next]){
                    q.push(e.next);
                    vis[e.next] = true;
                }
            }
        }
    }
    if(parent[end].first == -1) return ans;
    int now = end;
    while(now != start){
        auto [node, idx] = parent[now];
        ans.emplace_back(node, idx);
        now = node;
    }
    ans.emplace_back(start, -1);
    reverse(ans.begin(), ans.end());
    return ans;
}

pii MCMF(int start, int end){
    int ans = 0, cost = 0;
    while(1){
        vector<bool> visited(graph.size() + 1, false);
        auto tmp = search_path(start, end);
        if(tmp.size() == 0) break;
        auto [flow, c] = dfs(start, end, {1e9, 0}, tmp, 0);
        ans += flow;
        cost += c;
    }
    return {ans, cost};
}
```

3.11 二分圖

/*
判定二分圖：著色法 dfs 下去，顏色相撞非二分圖

二分圖最大匹配：用 maxflow 去做，一個 src
點聯通所有左圖，左圖建邊向右圖，右圖再建邊向 end
點，計算 src 跟 end 的最大流，若要還原，找出左圖
通往右圖中 capacity 為 0 的邊，他的兩個端點就是答案

最小點覆蓋：選最少的點，保證每條邊
至少有一個端點被選到， 最小點覆蓋 = 二分圖最大匹配

最大獨立集：選最多的點，滿足這些
點兩兩間互不相連， 最大獨立集 = n - 二分圖最大匹配
*/

3.12 Check cycle

```
vector<int> G[MAXN];
bool visit[MAXN];
/* return if the connected component where u is
contains a cycle*/
bool dfs(int u, int pre) {
    if(visit[u]) return true;
    visit[u] = true;
    for(int v : G[u])
        if(v != pre && dfs(v, u))
            return true;
    return false;
}
```

//check if a graph contains a cycle

```
bool checkCycle(int n) {
    for(int i = 1; i <= n; i++)
        if(!visit[i] && dfs(i, -1))
            return true;
    return false;
}
```

3.13 BCC

// [USAGE] BCC bcc(n); bcc.add_edge(u, v); bcc.solve();
// bcc.is_ap[i]; // i 是否為割點
// bcc.bcc[j]; // 第 j 個點雙連通分量中包含的所有頂點
// bcc.nbcc; // 點雙連通分量數量

// [USAGE] bcc.block_cut_tree();
// bcc.ng[i]; // 新圖頂點 i 的所有鄰居
// bcc
.bln[i]; // 原圖中的 i 在新圖上的編號(i可以是割點)
// bcc.cir[j]; // 新圖上的頂點 j 是否為割點

```
struct BCC { // 0-base
    int n, dft, nbcc;
    vector<int> low, dfn, bln, stk, is_ap, cir;
    vector<vector<int>> G, bcc, ng;
    void make_bcc(int u) {
        bcc.emplace_back(1, u);
        for (; stk.back() != u; stk.pop_back())
            bln[stk.back()] = nbcc, bcc[nbcc].pb(stk.back());
        stk.pop_back(), bln[u] = nbcc++;
    }
    void dfs(int u, int f) {
        int child = 0;
        low[u] = dfn[u] = ++dft, stk.pb(u);
        for (int v : G[u])
            if (!dfn[v]) {
                dfs(v, u), ++child;
                low[u] = min(low[u], low[v]);
                if (dfn[u] <= low[v]) {
                    is_ap[u] = 1, bln[u] = nbcc;
                    make_bcc(v), bcc.back().pb(u);
                }
            } else if (dfn[v] < dfn[u] && v != f)
                low[u] = min(low[u], dfn[v]);
        if (f == -1 && child < 2) is_ap[u] = 0;
        if (f == -1 && child == 0) make_bcc(u);
    }
    BCC(int _n): n(_n), dft(),
        nbcc(), low(n), dfn(n), bln(n), is_ap(n), G(n) {}
}
```



```

void add_edge(int u, int v) {
    G[u].pb(v), G[v].pb(u);
}
void solve() {
    for (int i = 0; i < n; ++i)
        if (!dfn[i]) dfs(i, -1);
}
void block_cut_tree() {
    cir.resize(nbcc);
    for (int i = 0; i < n; ++i)
        if (is_ap[i])
            bln[i] = nbcc++;
    cir.resize(nbcc, 1), nG.resize(nbcc);
    for (int i = 0; i < nbcc && !cir[i]; ++i)
        for (int j : bcc[i])
            if (is_ap[j])
                nG[i].pb(bln[j]), nG[bln[j]].pb(i);
} // up to 2 * n - 2 nodes!! bln[i] for id
}; // 8BQube

```

4 String

4.1 trie

```

class trie{
public:
    class node{
    public:
        int count;
        vector<trie::node*> child;
        node(){
            child.resize(26, nullptr);
            count = 0;
        }
        ~node() {
            for (auto c : child)
                if (c) delete c;
        }
    };
    node* root;
    trie(){
        root = new node;
    }
    ~trie() {
        delete root;
    }
    void insert(string s){
        auto temp = root;
        for(int i = 0; i < s.size(); i++){
            if(!temp -> child[s[i] - 'a']) temp -> child[s[i] - 'a'] = new node;
            temp = temp -> child[s[i] - 'a'];
        }
        temp -> count++;
    }
    bool search(string &s){
        auto temp = root;
        for(int i = 0; i < s.size(); i++){
            temp = temp -> child[s[i] - 'a'];
            if(!temp) return false;
        }
        if(temp -> count > 0) return true;
        return false;
    }
};

```

4.2 KMP

```

vector<int> build(string &s){
    vector<int> next = {0, 0};
    // 匹配失敗跳去哪 (最長共同前後綴)
    int length = s.size(), j = 0;
    for(int i = 1; i < length; i++){
        while(j > 0 and s[j] != s[i]){
            j = next[j];
        }
        if(s[j] == s[i]) j++;
        next.push_back(j);
    }
    return next;
}

int match(string &a, string &b){
    auto next = build(b);
    int length
        = a.size(), length2 = b.size(), j = 0, count = 0;

```

```

for(int i = 0; i < length; i++){
    while(j > 0 and a[i] != b[j]){
        j = next[j];
    }
    if(a[i] == b[j]) j++;
    if(j == length2){
        count++;
        j = next[j];
    }
}
return count;
}

```

4.3 Hash

```

vector<int> Pow(int num){
    int p = 1e9 + 7;
    vector<int> ans = {1};
    for(int i = 0; i < num; i++){
        ans.push_back(ans.back() * b % p);
    }
    return ans;
}

vector<int> Hash(string s){
    int p = 1e9 + 7;
    vector<int> ans = {0};
    for(char c:s){
        ans.push_back((ans.back() * b + c) % p);
    }
    return ans;
}

// 閉區間[l, r]
int query
    (vector<int> &vec, vector<int> &pow, int l, int r){
    int p = 1e9 + 7;
    int length = r - l + 1;
    return
        (vec[r + 1] - vec[l] * pow[length] % p + p) % p;
}

```

4.4 Zvalue

```

vector<int> z_func(string s1){
    int l = 0, r = 0, n = s1.size();
    vector<int> z(n, 0);
    for(int i = 1; i < n; i++){
        if(i
            <= r and z[i - l] < r - i + 1) z[i] = z[i - l];
        else{
            z[i] = max(z[i], r - i + 1);
            while(i + z
                [i] < n and s1[i + z[i]] == s1[z[i]]) z[i]++;
        }
        if(i + z[i] - 1 > r){
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}

```

4.5 最長迴文子字串

```

// 找到對於每個位置的迴文半徑
vector<int> manacher(string s) {
    string t = "#";
    for (auto c : s) {
        t += c;
        t += '#';
    }
    int n = t.size();
    vector<int> r(n);
    for (int i = 0, j = 0; i
        < n; i++) { // i 是中心, j 是最長回文字串中心
        if (2 * j - i >= 0 && j + r[j] > i) {
            r[i] = min(r[2 * j - i], j + r[j] - i);
        }
        while (i - r[i] >= 0 &&
            i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
            r[i] += 1;
        }
        if (i + r[i] > j + r[j]) {
            j = i;
        }
    }
}

```

```

return r;
// # a # b # a #
// 1 2 1 4 1 2 1
// # a # b # b # a #
// 1 2 1 2 5 2 1 2 1
// 值 -1 代表原回文字串長度
// (id - val + 1) / 2 可得原字串回文開頭
}

```

4.6 Suffix Array

```

struct SuffixArray {
    int n; string s;
    vector<int> sa, rk, lc;
    // 想法：
    // 排序過了，因此前綴長得像的會距離很近在差不多位置
    // n: 字串長度
    // sa: 後綴數組，sa[i] 表示第 i 小的後綴的起始位置
    // rk: 排名數組，rk[i] 表示從位置 i 開始的後綴的排名
    // lc: LCP 數組，
    // lc[i] 表示 sa[i] 和 sa[i + 1] 的最長公共前綴長度
    // 求 sa[i] 跟 sa[j] 的
    // LCP 長度 當 i < j : min(lc[i] ..... lc[j - 1])
    // 求 longest common substring : A +
    // "B" + B 建立 SA，找到 sa 相鄰但不同組中 lc 最大的
    SuffixArray(const string &s_) {
        s = s_; n = s.length();
        sa.resize(n);
        lc.resize(n - 1);
        rk.resize(n);
        iota(sa.begin(), sa.end(), 0);
        sort(sa.begin(), sa.end(),
            [&](int a, int b) { return s[a] < s[b]; });
        rk[sa[0]] = 0;
        for (int i = 1; i < n; ++i)
            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
        int k = 1;
        vector<int> tmp, cnt(n);
        tmp.reserve(n);
        while (rk[sa[n - 1]] < n - 1) {
            tmp.clear();
            for (int i = 0; i < k; ++i)
                tmp.push_back(n - k + i);
            for (auto i : sa)
                if (i >= k)
                    tmp.push_back(i - k);
            fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; ++i)
                ++cnt[rk[i]];
            for (int i = 1; i < n; ++i)
                cnt[i] += cnt[i - 1];
            for (int i = n - 1; i >= 0; --i)
                sa[--cnt[rk[tmp[i]]]] = tmp[i];
            swap(rk, tmp);
            rk[sa[0]] = 0;
            for (int i = 1; i < n; ++i)
                rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i] - 1] < tmp[sa[i]] || sa[i - 1] + k ==
                    n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
            k *= 2;
        }
        for (int i = 0, j = 0; i < n; ++i) {
            if (rk[i] == 0) {
                j = 0;
            } else {
                for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j
                    < n && s[i + j] == s[sa[rk[i] - 1] + j]; )
                    ++j;
                lc[rk[i] - 1] = j;
            }
        }
    }
};

```

4.7 AC-Automatan

```

struct AC_Automatan {
    int nx[len][sigma], fl[len], cnt[len], ord[len], top;
    int rxn[len][sigma]; // node actually be reached
    int newnode() {
        fill_n(nx[top], sigma, -1);
        return top++;
    }
};

```

```

void init() { top = 1, newnode(); }
int input(string &s) {
    int X = 1;
    for (char c : s) {
        if (!nx[X][c - 'A']) nx[X][c - 'A'] = newnode();
        X = nx[X][c - 'A'];
    }
    return X; // return the end node of string
}
void make_fl() {
    queue<int> q;
    q.push(1), fl[1] = 0;
    for (int t = 0; !q.empty(); ) {
        int R = q.front();
        q.pop(), ord[t++] = R;
        for (int i = 0; i < sigma; ++i)
            if (~nx[R][i]) {
                int X = rxn[R][i] = nx[R][i], Z = fl[R];
                for (; Z && !nx[Z][i]; ) Z = fl[Z];
                fl[X] = Z ? nx[Z][i] : 1, q.push(X);
            }
            else rxn[R][i] = R > 1 ? rxn[fl[R]][i] : 1;
    }
}
void solve() {
    for (int i = top - 2; i > 0; --i)
        cnt[fl[ord[i]]] += cnt[ord[i]];
}
} ac;

```

5 Geometry

5.1 Point

```

template<typename T>
class point {
public:
    T x;
    T y;
    point() {}
    point(T _x, T _y) {
        x = _x;
        y = _y;
    }
    point<T> operator+(const point<T> &a);
    point<T> operator-(const point<T> &a);
    point<T> operator/(const point<T> &a);
    point<T> operator/(T a);
    point<T> operator*(const T &a);
    bool operator<(const point<T> &a);
};

template<typename T>
point<T> point<T>::operator+(const point<T> &a) {
    return point<T>(x + a.x, y + a.y);
}

template<typename T>
point<T> point<T>::operator-(const point<T> &a) {
    return point<T>(x - a.x, y - a.y);
}

template<typename T>
point<T> point<T>::operator/(const point<T> &a) {
    return point<T>(x / a.x, y / a.y);
}

template<typename T>
point<T> point<T>::operator/(T a) {
    return point<T>(x / a, y / a);
}

template<typename T>
point<T> point<T>::operator*(const T &a) {
    return point<T>(x * a, y * a);
}

template<typename T>
bool point<T>::operator<(const point<T> &a) {
    if (x != a.x) return x < a.x;
    return y < a.y;
}

```

5.2 內積, 外積, 距離

```

template<typename T>
T dot(const point<T> &a, const point<T> &b){
    return a.x * b.x + a.y * b.y;
}

template<typename T>
T cross(const point<T> &a, const point<T> &b){
    return a.x * b.y - a.y * b.x;
}

template<typename T>
T len(point<T> p){
    return sqrt(dot(p, p));
}

template<typename T>
int sign(T x){
    return x == 0 ? 0 : x > 0 ? 1 : -1;
}

template<typename T>
T pointSegDist(point<T> q0, point<T> q1, point<T> p) {
    if (sign(len(q0 - q1)) == 0) return len(q0 - p);
    if (sign(dot(q1 - q0, p - q0)) >= 0 && sign(dot(q0 - q1, p - q1)) >= 0)
        return
            abs(cross(q1 - q0, p - q0) / len(q0 - q1));
    return min(len(p - q0), len(p - q1));
}

```

5.3 向量應用

```

template<typename T>
bool collinearity
(point<T> p1, point<T> p2, point<T> p3){
    //檢查三點是否共線
    return cross(p2 - p1, p2 - p3) == 0;
}

template<typename T>
bool inLine(point<T> a, point<T> b, point<T> p){
    //檢查 p 點是否在ab線段
    return collinearity
        (a, b, p) && dot(a - p, b - p) <= 0;
}

template<typename T>
bool intersect
(point<T> a, point<T> b, point<T> c, point<T> d){
    //ab線段跟cd線段是否相交
    return (cross(b - a, c - a) * \
        cross(b - a, d - a) < 0 && \
        cross(d - c, a - c) * \
        cross(d - c, b - c) < 0) \
        || inLine(a, b, c) || \
        inLine(a, b, d) || inLine(c, d, a) \
        || inLine(c, d, b);
}

template<typename T>
point<T> intersection
(point<T> a, point<T> b, point<T> c, point<T> d){
    //ab線段跟cd線段相交的點
    assert(intersect(a, b, c, d));
    return a + (b -
        a) * cross(a - c, d - c) / cross(d - c, b - a);
}

template<typename T>
bool inPolygon(vector<point<T>> polygon, point<T> p){
    //判斷點
    //p是否在多邊形polygon裡，vector裡的點要連續填對
    for(int i = 0; i < polygon.size(); i++){
        if(cross(p - polygon[i], \
            polygon[(i - 1 + polygon.size()) % \
            polygon.size()] - polygon[i]) * \
            cross(p - polygon[i], \
            polygon[(i +
                1) % polygon.size()] - polygon[i]) > 0)
            return false;
        return true;
    }
}

template<typename T>
T triangleArea(point<T> a, point<T> b, point<T> c){

```

```

//三角形頂點，求面積
return abs(cross(b - a, c - a)) / 2;
}

template<typename T, typename F, typename S>
long double triangleArea_Herons_formula(T a, F b, S c){
    //三角形頂點，求面積(給邊長)
    auto p = (a + b + c) / 2;
    return sqrt(p * (p - a) * (p - b) * (p - c));
}

template<typename T>
T area(vector<point<T>> &p){
    //多邊形頂點，求面積
    T ans = 0;
    for(int i = 0; i < p.size(); i++){
        ans += cross(p[i], p[(i + 1) % p.size()]);
    }
    return ans / 2 > 0 ? ans / 2 : -ans / 2;
}

```

5.4 Static Convex Hull

```

// 需要使
// 用前一個向量模板的 point，需要 operator - 以及 <
// 需要前面向量模板的 cross

template<typename T>
vector<point<T>> getConvexHull(vector<point<T>>& pnts){
    sort(pnts.begin(), pnts.end());
    auto cmp = [&](point<T> a, point<T> b)
    { return a.x == b.x && a.y == b.y; };
    pnts.erase(unique
        (pnts.begin(), pnts.end(), cmp), pnts.end());
    if(pnts.size() <= 1) return pnts;
    vector<point<T>> hull;
    for(int i = 0; i < 2; i++){
        int t = hull.size();
        for(point<T> pnt : pnts){
            while(hull.size() - t >= 2 &&
                cross(hull.back() - hull[hull.size() - 2], pnt - hull[hull.size() - 2]) < 0)
                // <= 0 或者 < 0 要看點有沒有在邊上
                hull.pop_back();
            hull.push_back(pnt);
        }
        hull.pop_back();
        reverse(pnts.begin(), pnts.end());
    }
    return hull;
}

```

5.5 外心,最小覆蓋圓

```

int sign(double a)
{
    // 小於 eps
    // 回傳 0，否則正回傳 1，負回傳 -1 應付浮點數誤差用
    const double eps = 1e-10;
    return fabs(a) < eps ? 0 : a > 0 ? 1 : -1;
}

// 輸入三個點求外心
template<typename T>
point<T> findCircumcenter(point<T> A, point<T> B, point<T> C, const T eps = 1e-10){
    point<T> AB = B - A;
    point<T> AC = C - A;
    T AB_len_sq = AB.x * AB.x + AB.y * AB.y;
    T AC_len_sq = AC.x * AC.x + AC.y * AC.y;
    T D = AB.x * AC.y - AB.y * AC.x;
    // 若三點接近共線
    assert(fabs(D) < eps);
    // 外心的座標
    T circumcenterX = A.x + (
        AC.y * AB_len_sq - AB.y * AC_len_sq) / (2 * D);
    T circumcenterY = A.y + (
        AB.x * AC_len_sq - AC.x * AB_len_sq) / (2 * D);
    return point<T>(circumcenterX, circumcenterY);
}

template<typename T>
pair<T, point<T>> MinCircleCover(vector<point<T>> &p) {
    // 引入前面的 len 跟 point
    // 回傳最小覆蓋圓{半徑, 中心}
}

```

```

random_shuffle(p.begin(), p.end());
int n = p.size();
point<T> c = p[0]; T r = 0;
for(int i=1; i<n; i++) {
    if(sign(len(c-p[i])-r) > 0) { // 不在圓內
        c = p[i], r = 0;
        for(int j=0; j<i; j++) {
            if(sign(len(c-p[j])-r) > 0) {
                c = (p[i]+p[j])/2.0;
                r = len(c-p[i]);
                for(int k=0; k<j; k++) {
                    if(sign(len(c-p[k])-r) > 0) {
                        c = findCircumcenter(p[i], p[j], p[k]);
                        r = len(c-p[i]);
                    }
                }
            }
        }
    }
}
return make_pair(r, c);
}

```

5.6 四邊形旋轉

```

const long double PI = acos(-1);

// 寬 w 高 h 的四邊形，旋轉一個 pi 後在每個角度的寬高
vector
<pair<double, double>> rotate(double w, double h){
    int freq = 1000; // 自己調整精度
    vector<pair<double, double>> res;
    for (int i = 0; i <= 5; ++i) {
        double theta = (PI * i) / 5;
        double nw
            = c * fabs(cos(theta)) + d * fabs(sin(theta));
        double nh
            = c * fabs(sin(theta)) + d * fabs(cos(theta));
        res.push_back({nw, nh});
    }
    return res;
}

```

5.7 旋轉

```

const long double PI = acos(-1);
// 逆時針旋轉
// angle_rad 為弧度
pair<double, double> rotate_point
    (double x, double y, double angle_rad) {
    angle_rad *= PI;
    double
        new_x = x * cos(angle_rad) - y * sin(angle_rad);
    double
        new_y = x * sin(angle_rad) + y * cos(angle_rad);
    return {new_x, new_y};
}

int main() {
    double x = 5, y = 0;
    double angle = 0.5; // 逆時針旋轉 90 度
    auto result = rotate_point(x, y, angle);
    cout << result.first << " " << result.second << endl;
    // 0, 5
    return 0;
}

```

5.8 極座標轉直角座標

```

// 極座標轉換為直角座標函數，theta 單位為弧度
const long double PI = acos(-1);
pair<double, double>
    > polar_to_cartesian(double r, double theta) {
    double theta_radians = theta * PI;
    double x = r * cos(theta_radians);
    double y = r * sin(theta_radians);
    return {x, y};
}

int main() {
    double r = 5, theta = 0.5; // 極座標
    auto result = polar_to_cartesian(r, theta);
    cout << result.first << " " << result.second << endl;
}

```

```

// 0, 5
return 0;
}

```

5.9 直角座標轉極座標

```

// 直角座標轉換為極座標
const long double PI = acos(-1);
std::pair<double
    , double> cartesian_to_polar(double x, double y) {
    double r = sqrt(x * x + y * y);
    double theta = atan2(y, x) / PI;
    return {r, theta};
}

int main() {
    double x = 3, y = 4; // 直角座標
    auto result = cartesian_to_polar(x, y);
    cout << "r = " << result
        .first << ", theta = " << result.second << endl;
    // 5, 0.295167
    return 0;
}

```

6 Data Structure

6.1 Sparse Table

```

class Sparse_Table{
    // 0-base
    // 要改成找最大把min換成max就好
private:
public:
    int spt[500005][22][2];
    Sparse_Table(vector<int> &ar){
        int n = ar.size();
        for (int i = 0; i < n; i++){
            spt[i][0][0] = ar[i];
            // spt[i][0][1] = ar[i];
        }
        for (int j = 1; (1 << j) <= n; j++) {
            for (int i = 0; (i + (1 << j) - 1) < n; i++) {
                spt[i][j][0] = min(spt[i + (1 <<
                    (j - 1))][j - 1][0], spt[i][j - 1][0]);
                // spt[i][j][1] = max(spt[i + (1 <<
                    (j - 1))][j - 1][1], spt[i][j - 1][1]);
            }
        }
    }
    int query_min(int l, int r)
    {
        if(l>r) return INT_MAX;
        int j = (int)__lg(r - l + 1);
        ///j = 31 - __builtin_clz(r - l + 1);
        return min
            (spt[l][j][0], spt[r - (1 << j) + 1][j][0]);
    }
    int query_max(int l, int r)
    {
        if(l>r) return INT_MAX;
        int j = (int)__lg(r - l + 1);
        ///j = 31 - __builtin_clz(r - l + 1);
        return max
            (spt[l][j][1], spt[r - (1 << j) + 1][j][1]);
    }
};

```

6.2 Segement Tree

```

// 不想要區間加值就把每個函數裡面的 push 都移除
// 最外層呼叫時，每個 id 都傳 1

const int N = 200000 + 9;
int a[N];
int seg[4 * N];
int lazy[4 * N];

inline void pull(
    int id){ seg[id] = seg[id * 2] + seg[id * 2 + 1]; }

inline void apply(int id, int l, int r, int v){
    seg[id] += v * (r - l + 1);
    lazy[id] += v;
}

```

```

inline void push(int id, int l, int r){
    if (!lazy[id] || l == r) return;
    int mid = (l + r) / 2;
    apply(id * 2, l, mid, lazy[id]);
    apply(id * 2 + 1, mid + 1, r, lazy[id]);
    lazy[id] = 0;
}

void build(int id, int
    l, int r) { // 編號為 id 的節點，存的區間為 [l, r]
    if (l
        == r) { seg[id] = a[l]; return; } // 葉節點的值
    int mid =
        (l + r) / 2; // 將區間切成兩半
    build(id * 2, l, mid); // 左子節點
    build(id * 2 + 1, mid + 1, r); // 右子節點
    pull(id);
}

// 區間查詢：回傳 [ql, qr] 的區間和
int query(int id, int l, int r, int ql, int qr) {
    if (r < ql || qr < l) return 0; // 交集為空
    if (ql <= l && r <= qr) return seg[id]; // 完全覆蓋
    push(
        id, l, r); // 下傳 lazy
    int mid = (l + r) / 2;
    return query(id * 2, l, mid, ql, qr) // 左
        + query(id * 2 + 1, mid + 1, r, ql, qr); // 右
    // 否則，往左、右進行遞迴
}

// 區間加值：將 [ql, qr] 每個位置都加上 x
void range_add
    (int id, int l, int r, int ql, int qr, int x) {
    if (r < ql || qr < l) return; // 交集為空
    if (ql <= l && r <=
        qr) { apply(id, l, r, x); return; } // 完全覆蓋
    push(id, l, r) // 下傳 lazy 再往下走
    ;
    int mid = (l + r) / 2;
    range_add
        (id * 2, l, mid, ql, qr, x); // 左
    range_add
        (id * 2 + 1, mid + 1, r, ql, qr, x); // 右
    pull(id);
}

// 單點修改 (設值版)：將 a[i] 改成 x
void modify(int id, int l, int r, int i, int x) {
    if (l == r) { seg[id] = x; return; }
    push(id, l, r); // 確保往下的值正確
    int mid = (l + r) / 2;
    if (i
        <= mid) modify(id * 2, l, mid, i, x); // 左
    else modify
        (id * 2 + 1, mid + 1, r, i, x); // 右
    pull(id);
}

```

6.3 Discrete Segement Tree

```

#include <bits/stdc++.h>
#define int long long
#define de(x) cout << #x << ' ' << x << ", "
#define dd cout << '\n';

/*
本題給n個長方形 0 <= x1, y1, x2, y2 <= 10^9
求被奇數個長方形覆蓋的面積

想法：掃描線+離散化線段樹
區間xor加值 + 區間sum查詢
*/

/*
先將輸入點離散化 [a1, a2, ..., an] => [1, 2, ..., n]
將每個相鄰離散點區間重新編號 [1, 2] => 編號 1 [2, 3] => 編號 2, ..., [n - 1, n] 編號 n - 1
將每個區間的長度都記下 pre[1] = a2 - a1
, pre[2] = a3 - a2, ..., pre[n - 1] = an - an-1

```

```

若要加值區間[a, b]，則加線段樹上的區間[a, b - 1]
和一般線段樹唯一差別在 void apply
    ()，區間長度從(r - l + 1)變成(pre[r] - pre[l - 1])
*/

/*
testcase 1 :
2
0 0 4 4
1 1 3 3

answer : 12

testcase 2 :
4
0 0 10 10
1 1 11 11
2 2 12 12
3 3 13 13

answer : 72
*/

using namespace std;
int tt = 1;

int a[600009];
int seg[4 * 600009];
int lazy[4 * 600009];
int mp[600009];
int pre[600009];

map<int, int> mp2;

void pull(int id){
    seg[id] = seg[id * 2] + seg[id * 2 + 1];
}

void apply(int id, int l, int r, int v){
    seg[id] = (pre[r] - pre[l - 1]) - seg[id];
    lazy[id] ^= 1;
}

void push(int id, int l, int r){
    if(!lazy[id] || l == r) return;
    int mid = (l + r) / 2;
    apply(id * 2, l, mid, lazy[id]);
    apply(id * 2 + 1, mid + 1, r, lazy[id]);
    lazy[id] = 0;
}

int query(int id, int l, int r, int ql, int qr){
    if(r < ql || qr < l) return 0;
    if(ql <= l && r <= qr) return seg[id];
    push(id, l, r);
    int mid = (l + r) / 2;
    return query(id * 2, l, mid, ql, qr)
        + query(id * 2 + 1, mid + 1, r, ql, qr);
}

void range_add
    (int id, int l, int r, int ql, int qr, int x){
    if(r < ql || qr < l) return;
    if(ql <= l && r <= qr) {
        apply(id, l, r, x);
        return;
    }
    push(id, l, r);
    int mid = (l + r) / 2;
    range_add(id * 2, l, mid, ql, qr, x);
    range_add(id * 2 + 1, mid + 1, r, ql, qr, x);
    pull(id);
}

void _pre(){
    cout.tie(nullptr);
    //cin >> tt;
}

struct rec{
    int loc, down, top;
};

bool comp(rec a, rec b){

```



```

    return a.loc < b.loc;
}

void swap(int &a, int &b){
    int tmp = a;
    a = b;
    b = tmp;
}

void solve(){
    int n, a, b, c, d, i, j, k;
    vector<rec> v;
    cin >> n;
    set<int> st;
    for(i = 1; i <= n; i++){
        cin >> a >> b >> c >> d;
        int high = max(b, d);
        int low = min(b, d);
        v.push_back({a, low, high});
        v.push_back({c, low, high});
        st.insert(low);
        st.insert(high);
    }
    int N = 0;
    pre[0] = 0;
    mp[0] = 0;
    vector<int> vv;

    for(auto it = st.begin(); it != st.end(); it++){
        mp[++N] = *it;
        mp2[*it] = N;
        vv.push_back(*it);
    }

    for(i = 0; i < (int)vv.size() - 1; i++){
        pre[i + 1] = pre[i] + (vv[i + 1] - vv[i]);
    }

    for(i = 0; i < (int)v.size(); i++){
        v[i] = {v[i].low, mp2[v[i].down], mp2[v[i].top] - 1};
    }
    sort(v.begin(), v.end(), comp);
    int ans = 0;

    int lastloc = v[0].loc;
    range_add(1, 1, N - 1, v[0].down, v[0].top, 1);

    for(i = 1; i < (int)v.size(); i++){
        int down = v[i].down;
        int top = v[i].top;
        int loc = v[i].loc;

        int q = query(1, 1, N - 1, 1, N - 1);

        ans += (loc - lastloc) * q;
        range_add(1, 1, N - 1, down, top, 1);
        lastloc = loc;
    }

    cout << ans << '\n';
}

signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    _pre();
    while(tt--){
        solve();
    }
}

```

6.4 Link Cut Tree

// 通常用於對樹上任兩點間的路徑做加值、修改、查詢等工作
 // 與線段樹相同，要修改 LCT 的功能只需更改
 // pull、push、fix、query 等函數，再加上需要的懶標即可
 // 範例為樹上任兩點 x, y 路徑上的權值 xor
 // 和，樹上任意點單點改值

```

const int N = 300005;
class LinkCutTree {
private:
#define lc(x) node[x].ch[0]
#define rc(x) node[x].ch[1]
#define fa(x) node[x].fa
#define rev(x) node[x].rev
#define val(x) node[x].val

```

```

#define sum(x) node[x].sum
struct Tree {
    int val, sum, fa, rev, ch[2];
} node[N];
inline void pull(int x) {
    sum(x) = val(x) ^ sum(lc(x)) ^ sum(rc(x));
}
inline void reverse(int x) {
    swap(lc(x), rc(x));
    rev(x) ^= 1;
}
inline void push(int x) {
    if (rev(x)) {
        reverse(lc(x));
        reverse(rc(x));
        rev(x) ^= 1;
    }
}
inline bool get(int x) { return rc(fa(x)) == x; }
inline bool isroot(int x) {
    return (lc(fa(x)) ^ x) && (rc(fa(x)) ^ x);
}
inline void update(int x) {
    if (!isroot(x)) update(fa(x));
    push(x);
}
void rotate(int x) {
    int y = fa(x), z = fa(y), d = get(x);
    if (!isroot(y))
        node[z].ch[get(y)] = x; // 重要，不能更換順序
    fa(x) = z;
    node[fa(x)].ch[d ^ 1] = y;
    node[x].ch[d ^ 1] = y;
    pull(y), pull(x); // 先 y 再 x
}
void splay(int x) {
    update(x);
    for (int y = fa(x); !isroot(x);
        rotate(x), y = fa(x)) {
        if (!isroot(y)) rotate(get(x) == get(y) ? y : x);
    }
    pull(x);
}
int access(int x) {
    int p = 0;
    for (; x; x = fa(p = x)) {
        splay(x), rc(x) = p, pull(x);
    }
    return p;
}
inline void makeroot(int x) {
    access(x), splay(x), reverse(x);
}
inline int findroot(int x) {
    access(x), splay(x);
    while (lc(x)) { push(x), x = lc(x); }
    return splay(x), x;
}
inline void split(int x, int y) {
    makeroot(x), access(y), splay(y);
}

public:
inline void init(int len, int *data) {
    for (int i = 1; i <= len; ++i) {
        node[i].val = data[i];
    }
}
inline void link(int x, int y) { // 連邊
    makeroot(x);
    if (findroot(y) == x) return;
    fa(x) = y;
}
inline void cut(int x, int y) { // 斷邊
    makeroot(x);
    if (findroot(y) != x || fa(y) != x || lc(y))
        return;
    fa(y) = rc(x) = 0;
    pull(x);
}
inline void fix(int x, int v) { // 單點改值
    splay(x);
    val(x) = v;
}
}

```

```
// 區間查詢
inline int query(int x, int y) {
    return split(x, y), sum(y);
}
};

LinkCutTree LCT;
int n, a[N];

signed main() {
    int n, q, op, x, y;
    cin >> n >> q;
    for (int i = 1; i <= n; ++i) { cin >> a[i]; }
    LCT.init(n, a);
    while (q--) {
        cin >> op >> x >> y;
        if (op == 0) {
            cout << LCT.query(x, y) << endl;
        } else if (op == 1) {
            LCT.link(x, y);
        } else if (op == 2) {
            LCT.cut(x, y);
        } else {
            LCT.fix(x, y);
        }
    }
    return 0;
}
```

6.5 BIT

```
#define lowbit(x) x & -x

void modify(vector<int> &bit, int idx, int val) {
    for(int i = idx; i <= bit.size(); i += lowbit(i)) bit[i] += val;
}

int query(vector<int> &bit, int idx) {
    int ans = 0;
    for(int i = idx; i > 0; i -= lowbit(i)) ans += bit[i];
    return ans;
}

// the first i s.t. a[1]+...+a[i] >= k
int findK(vector<int> &bit, int k) {
    int idx = 0, res = 0;
    int mx = __lg(bit.size()) + 1;
    for(int i = mx; i >= 0; i--) {
        if((idx | (1<<i)) > bit.size()) continue;
        if(res + bit[idx | (1<<i)] < k) {
            idx = (idx | (1<<i));
            res += bit[idx];
        }
    }
    return idx + 1;
}

// O(n)建bit
for (int i = 1; i <= n; ++i) {
    bit[i] += a[i];
    int j = i + lowbit(i);
    if (j <= n) bit[j] += bit[i];
}
}
```

6.6 2D BIT

```
// 2維 BIT
#define lowbit(x) (x&-x)

class BIT {
    int n;
    vector<int> bit;

public:
    void init(int _n) {
        n = _n;
        bit.resize(n + 1);
        for(auto &b : bit) b = 0;
    }
    int query(int x) const {
        int sum = 0;
        for(; x; x -= lowbit(x))
            sum += bit[x];
        return sum;
    }
}
```

```
void modify(int x, int val) {
    for(; x <= n; x += lowbit(x))
        bit[x] += val;
}

class BIT2D {
    int m;
    vector<BIT> bit1D;

public:
    void init(int _m, int _n) {
        m = _m;
        bit1D.resize(m + 1);
        for(auto &b : bit1D) b.init(_n);
    }
    int query(int x, int y) const {
        int sum = 0;
        for(; x; x -= lowbit(x))
            sum += bit1D[x].query(y);
        return sum;
    }
    void modify(int x, int y, int val) {
        for(; x <= m; x += lowbit(x))
            bit1D[x].modify(y, val);
    }
};
```

6.7 undo DSU

```
struct dsu_undo{
    vector<int> sz, p;
    int comps;
    dsu_undo(int n){
        sz.assign(n+5, 1);
        p.resize(n+5);
        for(int i = 1; i <= n; ++i) p[i] = i;
        comps = n;
    }
    vector<pair<int, int>> opt;
    int Find(int x){
        return x == p[x] ? x : Find(p[x]);
    }
    bool Union(int a, int b){
        int pa = Find(a), pb = Find(b);
        if(pa == pb) return 0;
        if(sz[pa] < sz[pb]) swap(pa, pb);
        sz[pa] += sz[pb];
        p[pb] = pa;
        opt.push_back({pa, pb});
        comps--;
        return 1;
    }
    void undo(){
        auto [pa, pb] = opt.back();
        opt.pop_back();
        p[pb] = pb;
        sz[pa] -= sz[pb];
        comps++;
    }
};
```

7 Dynamic Programming

7.1 LCS

```
// O(n^2)
int LCS(string t1, string t2) {
    if(t1.size() < t2.size()) swap(t1, t2);
    int len = t1.size();
    vector<vector<int>> dp(2, vector<int>(len + 1, 0));
    for(int j = 1; j <= t2.size(); j++){
        for(int i = 1; i <= len; i++){
            if(t2[j - 1] == t1[i - 1])
                dp[j % 2][i] = dp[(j + 1) % 2][i - 1] + 1;
            else
                dp[j % 2][i] = max(dp[(j + 1) % 2][i], dp[j % 2][i - 1]);
        }
    }
    return dp[t2.size() % 2][t1.size()];
}

// O(nlogn)
// 這裡 string 要以 1 base index 所以開頭要補個字元
// d: 記住此數字的前一個數字
// t: 當前 LIS 位置, num: 根據 t2 生成出 string 來找 LIS 長度
```

```
// N: 最大字串長度
#define N 120
int t[N*N], d[N*N], num[N*N];
map<char, vector<int>> dict; // 每個字串出現的index位置
int binarySearch(int l, int r, int v){
    int m;
    while(r>l){
        m = (l+r)/2;
        if(num[v] > num[t[m]]) l = m+1;
        else if(num[v] < num[t[m]]) r = m;
        else return m;
    }
    return r;
}
int LCS(string t1, string t2){
    dict.clear();
    //i = strA.length() - 1 才可以逆序
    for(int i = t1.length() - 1; i > 0; i--) dict[t1[i]].push_back(i);
    int k = 0; //生成數列的長度的最長長度
    for(int i = 1; i < t2.length(); i++){
        // 依據 strB 的每個字元來生成數列
        for(int j = 0; j < dict[t2[i]].size(); j++){
            //將此字元在 strA 出現的位置放入數列
            num[i+k] = dict[t2[i]][j];
        }
        if(k==0) return 0;
        d[1] = -1, t[1] = 1; //LIS init
        int len = 1, cur; // len 由於前面已經把 LCS = 0 的機會排除，於是這裡則從 1 開始

        // 標準的 LIS 作法，不斷嘗試將 LCS 生長
        for(int i = 1; i <= k; i++){
            if(num[i] > num[t[len]]) t[++len] = i, d[i] = t[len-1];
            else{
                cur = binarySearch(1, len, i);
                t[cur] = i;
                d[i] = t[cur-1];
            }
        }
        return len;
    }
}
```

7.2 LIS

```
int LIS(vector<int>& save) {
    vector<int> dp;
    int n = save.size();
    for (int i = 0; i < n; i++) {
        auto it = lower_bound(dp.begin(), dp.end(), save[i]);
        if(it == dp.end()) dp.push_back(save[i]);
        else *it = save[i];
    }
    return dp.size();
}
```

7.3 Knapsack

```
/**
 * 背包問題：
 * 1. dp[i][j]: 考慮 1~i 個物品，重量為 j 時的最大價格
 * 2. dp[i][j]: 考慮 1~i 個物品，價值為 j 時的最小重量
 */

// 當重量比較輕時 O(nw)
vector<int> dp(sum + 1, 0);
for (int i = 1; i <= n; ++i) {
    for (int j = sum /* bound */; j >= weight[i]; --j) {
        if (dp[j] < dp[j - weight[i]] + price[i]) {
            dp[j] = dp[j - weight[i]] + price[i];
            backtrack[i][j] = 1;
        }
    }
}

// 當重量比較重時 O(nc)
vector<int> dp(sum + 1, 1e9 + 7);
dp[0] = 0;
for (int i = 1; i <= n; ++i) {
    for (int j = sum /* bound */; j >= price[i]; --j) {
        if (dp[j] > dp[j - price[i]] + weight[i]) {
            dp[j] = dp[j - price[i]] + weight[i];
            backtrack[i][j] = 1;
        }
    }
}
```

```
}
}
}

// backtrack: 找到當 bound 為 k 時，背包內有哪些東西
// 註：只找到其中一種
int l = n, r = k;
vector<int> ans;
while (l != 0 && r != 0) {
    if (backtrack[l][r]) {
        ans.push_back(l);
        r -= weight[l]; // 當用方法一時，用這行
        r -= price[l]; // 當用方法二時，用這行
    }
    l--;
}
```

7.4 位元 dp

```
// 檢查第 n 位是否為 1
if(a & (1 << n))

// 強制將第 n 位變成 1
a |= (1 << n)

// 強制將第 n 位變成 0
a &= ~(1 << n)

// 將第 n 位反轉(1變0, 0變1)
a ^= (1 << n)

// 第 0 ~ n - 1 位 全部都是 1
(1 << n) - 1

// 兩個集合的聯集
S = a | b

// 兩個集合的交集
S = a & b
```

7.5 經典 dp 轉移式

```
/*
最大區間和：

dp[i] 代表 由第 i 項結尾時的最大區間和
dp[0] = arr[0]
dp[i] = max(dp[i - 1], arr[i])
ans = max_element(dp)
*/
```

8 Divide and conquer

8.1 逆序數對

```
int merge(
    vector<pair<int, int>>& v, int l, int mid, int r) {
    vector<pair<int, int>> temp(r - l + 1);
    int i = l, j = mid + 1, k = 0, inv_count = 0;
    while (i <= mid && j <= r) {
        if (v[i].second <= v[j].second) {
            temp[k++] = v[i++];
        } else {
            temp[k++] = v[j++];
            inv_count += (mid - i + 1);
        }
    }
    while (i <= mid) temp[k++] = v[i++];
    while (j <= r) temp[k++] = v[j++];
    for (int i = l; i <= r; i++) {
        v[i] = temp[i - l];
    }
    return inv_count;
}

int mergeSort(
    vector<pair<int, int>>& v, int l, int r) {
    int count = 0;
    if (l < r) {
        int mid = l + (r - l) / 2;
        count += mergeSort(v, l, mid);
        count += mergeSort(v, mid + 1, r);
        count += merge(v, l, mid, r);
    }
}
```

```

    }
    return count;
}

signed main()
{
    int n;
    cin >> n;
    vector<pair<int, int>> arr(n);
    for(int i = 0; i < n; i++){
        arr[i].first = i;
        cin >> arr[i].second;
    }
    cout << mergeSort(arr, 0, n - 1) << '\n';
}

```

8.2 Mo's algorithm

```

// time complexity: n * sqrt(q) * O(p)
// O(p) 為 add, remove 的時間複雜度
// 若知道 [l, r] 的答案 需要快速知道 [l
- 1, r], [l + 1, r], [l, r - 1], [l, r + 1] 的答案

int n, q, k, l = 0, r = 0;

array queries = 詢問們;
type ans; //目前答案
void add(type v){/*...*/} //增加一個數字，算新答案
void remove(type v){/*...*/} //移除一個數字，算新答案

vector<tuple<int, int, int, int>> queries(q);
k = sqrt(n);
for(int i = 0; i < q; i++){
    int l, r;
    cin >> l >> r;
    queries[i] = {l / k, r, l, i};
    // 先對 l 的塊，再對 r 排序
}

sort(queries.begin(), queries.end());

add(a[0]);

for(int i = 0; i < q; i++){
    auto [_ , rp, lp, id] = queries[i];
    lp--; rp--;
    while(l > lp) add(a[--l]);
    while(l < lp) remove(a[l++]);
    while(r < rp) add(a[++r]);
    while(r > rp) remove(a[r--]);
    ans_v[id] = ans;
}

```

9 Tree

9.1 樹直徑

```

int d1[200005], d2[200005], ans;

void dfs(int now, int fa, vector<vector<int>> &graph){
    for(auto i: graph[now]){
        if(i != fa){
            dfs(i, now, graph);
            if(d1[i] + 1 > d1[now]){
                d2[now] = d1[now];
                d1[now] = d1[i] + 1;
            }
            else if(d1[i] + 1 > d2[now]){
                d2[now] = d1[i] + 1;
            }
        }
    }
    ans = max(ans, d1[now] + d2[now]);
}

signed main()
{
    int n;
    cin >> n;
    vector<vector<int>> graph(n + 1);
    for(int i = 0; i < n - 1; i++){
        int a, b;
        cin >> a >> b;
        graph[a].push_back(b);
        graph[b].push_back(a);
    }
}

```

```

    }
    dfs(1, 0, graph);
    cout << ans << '\n';
}

```

9.2 LCA

// n 為點數，graph 由子節點往父節點建有向邊
// graph 要 resize

```

int n, q;
int fa[20][200001];
int dep[200001];

vector<vector<int>> graph;

void dfs(int now, int lst){
    fa[0][now] = lst;
    for(int &i: graph[now]){
        dep[i] = dep[now] + 1;
        dfs(i, now);
    }
}

void build_lca(int root){
    dep[root] = 1;
    dfs(root, root);
    for(int i = 1; i < 18; i++){
        for(int j = 1; j < n + 1; j++){
            fa[i][j] = fa[i - 1][fa[i - 1][j]];
        }
    }
}

int lca(int a, int b){
    // 預設 a 比 b 淺
    if(dep[a] > dep[b]) return lca(b, a);
    // 讓 a 和 b 跳到同一個地方
    int step = dep[b] - dep[a];
    for(int i = 0; i < 18; i++){
        if(step >> i & 1){
            b = fa[i][b];
        }
    }
    if(a == b) return a;
    for(int i = 17; i >= 0; i--){
        if(fa[i][a] != fa[i][b]){
            a = fa[i][a];
            b = fa[i][b];
        }
    }
    return fa[0][a];
}

```

9.3 樹壓平

```

//紀錄 in & out
vector<int> Arr;
vector<int> In, Out;
void dfs(int u) {
    Arr.push_back(u);
    In[u] = Arr.size() - 1;
    for (auto v : Tree[u]) {
        if (v == parent[u])
            continue;
        parent[v] = u;
        dfs(v);
    }
    Out[u] = Arr.size() - 1;
}

//進去出來都紀錄
vector<int> Arr;
void dfs(int u) {
    Arr.push_back(u);
    for (auto v : Tree[u]) {
        if (v == parent[u])
            continue;
        parent[v] = u;
        dfs(v);
    }
    Arr.push_back(u);
}

```

```

//用 Treap 紀錄
Treap *root = nullptr;
vector<Treap *> In, Out;
void dfs(int u) {
    In[u] = new Treap(cost[u]);
    root = merge(root, In[u]);
    for (auto v : Tree[u]) {
        if (v == parent[u])
            continue;
        parent[v] = u;
        dfs(v);
    }
    Out[u] = new Treap(0);
    root = merge(root, Out[u]);
}

//Treap 紀錄 Parent
struct Treap {
    Treap *lc = nullptr, *rc = nullptr;
    Treap *pa = nullptr;
    unsigned pri, size;
    long long Val, Sum;
    Treap(int Val):
        pri(rand()), size(1),
        Val(Val), Sum(Val) {}
    void pull();
};

void Treap::pull() {
    size = 1;
    Sum = Val;
    pa = nullptr;
    if (lc) {
        size += lc->size;
        Sum += lc->Sum;
        lc->pa = this;
    }
    if (rc) {
        size += rc->size;
        Sum += rc->Sum;
        rc->pa = this;
    }
}

//找出節點在中序的編號
size_t getIdx(Treap *x) {
    assert(x);
    size_t Idx = 0;
    for (Treap *child = x->rc; x;) {
        if (child == x->rc)
            Idx += 1 + size(x->lc);
        child = x;
        x = x->pa;
    }
    return Idx;
}

//切出想要的東西
void move(Treap *&root, int a, int b) {
    size_t a_in = getIdx(In[a]), a_out = getIdx(Out[a]);
    auto [L, tmp] = splitK(root, a_in - 1);
    auto [tree_a, R] = splitK(tmp, a_out - a_in + 1);
    root = merge(L, R);
    tie(L, R) = splitK(root, getIdx(In[b]));
    root = merge(L, merge(tree_a, R));
}

```

9.4 樹鏈剖分

```

// 從 top 的深度較低的那個節點開始跳
// 跳到同一條鏈上 (top 相同)
// 一邊跳一邊更新線段樹
// in 是 dfn

```

```

struct HLD {
    // 0-base
    int n, cur;
    vector<int> siz, top, dep, parent, in, out, seq;
    vector<vector<int>>> adj;
    HLD(int n) : n(n), cur(0) {
        siz.resize(n); top.resize(n); dep.resize(n);
        parent.resize(n); in.resize(n); out.resize(n);
        seq.resize(n); adj.assign(n, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
}

```

```

void work(int rt = 0) {
    top[rt] = rt;
    dep[rt] = 0;
    parent[rt] = -1;
    dfs1(rt); dfs2(rt);
}

void dfs1(int u) {
    if (parent[u] != -1)
        adj[u].erase(find(adj[u].begin(), adj[u].end(), parent[u]));
    siz[u] = 1;
    for (auto &v : adj[u]) {
        parent[v] = u, dep[v] = dep[u] + 1;
        dfs1(v);
        siz[u] += siz[v];
        if (siz[v] > siz[adj[u][0]]) {
            swap(v, adj[u][0]);
        } // 讓 adj[u][0] 是重子節點
    }
}

void dfs2(int u) {
    in[u] = cur++;
    seq[in[u]] = u; // dfn 對應的編號
    for (auto v : adj[u]) {
        top[v] = v == adj[u][0] ? top[u] : v;
        dfs2(v);
    }
    out[u] = cur;
}

int lca(int u, int v) {
    while (top[u] != top[v]) {
        if (dep[top[u]] > dep[top[v]]) {
            u = parent[top[u]];
        } else {
            v = parent[top[v]];
        }
    }
    return dep[u] < dep[v] ? u : v;
}

int dist(int u, int v) {
    return dep[u] + dep[v] - 2 * dep[lca(u, v)];
}

int jump(int u, int k) {
    if (dep[u] < k) return -1;
    int d = dep[u] - k;
    while (dep[top[u]] > d) u = parent[top[u]];
    return seq[in[u] - dep[u] + d];
}

bool isAncestor(int u, int v) {
    return in[u] <= in[v] && in[v] < out[u];
}

int rootedParent(int rt, int v) {
    if (rt == v) return rt;
    if (!isAncestor(v, rt)) return parent[v];
    auto it = upper_bound(
        adj[v].begin(), adj[v].end(), rt,
        [&](int x, int y) {
            return in[x] < in[y];
        }) - 1;
    return *it;
}

int rootedSize(int rt, int v) {
    if (rt == v) return n;
    if (!isAncestor(v, rt)) return siz[v];
    return n - siz[rootedParent(rt, v)];
}

int rootedLca(int rt, int a, int b) {
    return lca(rt, a) ^ lca(a, b) ^ lca(b, rt);
}
}

```

// example 找兩個子節點間的 max 以及改值

```

signed main() {
    int n, q;
    cin >> n >> q;

    vector<int> tp(n);
    for (int i = 0; i < n; i++) cin >> tp[i];

    HLD t(n);
    for (int i = 0; i < n - 1; i++) {
        int u, v;
        cin >> u >> v;
        --u; --v;
    }
}

```



```

    t.addEdge(u, v);
}

t.work(0); // 以 0 (節點1) 為根

// 將點值依照 dfn 映射到線段樹底層
for (int i = 0; i < n; i++) {
    a[t.in[i]] = tp[i];
}
build(1, 0, n - 1); // 根 id = 1

auto path_max = [&](int u, int v) {
    int res = LLONG_MIN;
    // 往上跳鏈 邊跳邊維護答案
    while (t.top[u] != t.top[v]) {
        if (t
            .dep[t.top[u]] < t.dep[t.top[v]]) swap(u, v);
        res = max(res,
            query(1, 0, n - 1, t.in[t.top[u]], t.in[u]));
        u = t.parent[t.top[u]];
    }
    int l = t.in[u], r = t.in[v];
    if (l > r) swap(l, r);
    res = max(res, query(1, 0, n - 1, l, r));
    return res;
};

while (q--) {
    int mode;
    cin >> mode;
    if (mode == 1) {
        int s, x;
        cin >> s >> x;
        --s;
        modify(1, 0, n - 1, t.in[s], x);
    } else if (mode == 2) {
        int a, b;
        cin >> a >> b;
        --a; --b;
        cout << path_max(a, b) << '\n';
    }
}
}

```

10 Else

10.1 Big Number

```

string Add(const string &a, const string &b) {
    int n
        = a.length() - 1, m = b.length() - 1, car = 0;
    string res;
    while (n >= 0 || m >= 0 || car) {
        int x = (n >= 0 ? a[n] -
            '0' : 0) + (m >= 0 ? b[m] - '0' : 0) + car;
        res += (x % 10) + '0';
        car = x / 10;
        n--, m--;
    }
    while (res.length() > 1 && res.back() == '0') {
        res.pop_back();
    }
    reverse(res.begin(), res.end());
    return res;
}

string Minus(const string &a, const string &b) {
    // Assume a >= b
    int n
        = a.length() - 1, m = b.length() - 1, bor = 0;
    string res;
    while (n >= 0) {
        int x = a[n] - '0' - bor;
        int y = m >= 0 ? b[m] - '0' : 0;
        bor = 0;
        if (x < y) {
            x += 10;
            bor = 1;
        }
        res += x - y + '0';
        n--, m--;
    }
    while (res.length() > 1 && res.back() == '0') {
        res.pop_back();
    }
    reverse(res.begin(), res.end());
}

```

```

    return res;
}

string Multiple(const string &a, const string &b) {
    string res = "0";
    int n = a.length() - 1, m = b.length() - 1;
    for (int i = m; i >= 0; i--) {
        string add;
        int car = 0;
        for (int j = n; j >= 0 || car; j--) {
            int x = (j >= 0
                ? a[j] - '0' : 0) * (b[i] - '0') + car;
            add += (x % 10) + '0';
            car = x / 10;
        }
        while (add.length() > 1 && add.back() == '0') {
            add.pop_back();
        }
        reverse(add.begin(), add.end());
        res = Add(res, add + string(m - i, '0'));
    }
    return res;
}

```

10.2 Ternary Search

```

// return the maximum of $f(x)$ in $[l, r]$
double ternary_search(double l, double r) {
    while (r - l > EPS) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1), f2 = f(m2);
        if (f1 < f2) l = m1;
        else r = m2;
    }
    return f(l);
}

// return the maximum of $f(x)$ in $(l, r]$
int ternary_search(int l, int r) {
    while (r - l > 1) {
        int mid = (l + r) / 2;
        if (f(m) > f(m + 1)) r = m;
        else l = m;
    }
    return r;
}

```

10.3 Duipai

```

#include <iostream>
using namespace std;
int main() {
    for (int T=1; T++;) {
        if (system("./random > test.in")) {
            cout << "random RE on " << T << '\n';
            return 0;
        }
        if (system("./sol < test.in > test.out")) {
            cout << "sol RE on " << T << '\n';
            return 0;
        }
        if (system("./bf < test.in > test.ans")) {
            cout << "bf RE on " << T << '\n';
            return 0;
        }
        if (system("diff -Z test.out test.ans")) {
            cout << "WA on " << T << '\n';
            return 0;
        } else {
            cout << "AC on " << T << '\n';
        }
    }
}

```

10.4 Random Generator

```

#include <iostream>
#include <random>
int main() {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distrib(1, 100);
    std::cout << "Get Rand: " << distrib(gen) << '\n';
}

```