XCPC-Template

CREATED BY

Luliet Lyan & Bleu Echo

NSCC-GZ
School of Computer Science & Engineering
Sun Yat-Sen University

Supervisor: Dr Dan Huang
Co-Supervisor: Dr Zhiguang Chen

Friday 16th May, 2025

Contents

Part I: Basic Template

CREATED BY

Luliet Lyan & Bleu Echo

NSCC-GZ
School of Computer Science & Engineering
Sun Yat-Sen University

Supervisor: Dr Dan Huang
Co-Supervisor: Dr Zhiguang Chen

0 ⋆ Preface

0.1 Template

```cpp
#define itn int
#define nit int
#define nti int
#define tin int
#define tni int
#define retrun return
#define reutrn return
#define rutren return
#define fastin                             \
    ios_base::sync_with_stdio(0); \
    cin.tie(0), cout.tie(0);
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef long double LD;
typedef pair<int, int> PII;
typedef pair<long long, long long> PLL;
typedef pair<double, double> PDD;
typedef vector<int> VI;
#ifndef ONLINE_JUDGE
#define dbg(args...)     \
    do                   \
    {                    \
        cout << "\033[32;1m" << #args << " -> "; \
        err(args);       \
    } while (0)
#else
#define dbg(...)
#endif
void err()
{ cout << "\033[39;0m" << endl; }
template <template <typename...> class T, typename t, typename... Args>
void err(T<t> a, Args... args)
{
    for (auto x : a) cout << x << ' ';
    err(args...);
}
template <typename T, typename... Args>
void err(T a, Args... args)
{ cout << a << ' '; err(args...); }
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
const double eps = 1e-6;
int main()
{
#ifndef ONLINE_JUDGE
    freopen("test.in", "r", stdin);
    freopen("test.out", "w", stdout);
#endif
    fastin;

    return 0;
}
```

0.2 Operator Precedence

- **括号成员排第一；全体单目排第二；**
- **乘除余三加减四；移位五，关系六；**
- **等于不等排第七；位与异或和位或；**
- **三分天下八九十；逻辑与或十一二；**
- **条件赋值十三四；逗号十五最末尾。**

0.3 Time Complexity

- In most ACM or coding interview problems, the time limit is usually 1 or 2 seconds. Under such constraints, C++ programs should aim to stay within about $\mathbf{10^7} \sim \mathbf{10^8}$ operations.
- Below is a guide on how to choose algorithms based on different input size ranges:

  1. $\mathbf{n \le 30} \rightarrow$ Exponential complexity: DFS with pruning, State Compression DP
  2. $\mathbf{n \le 100} \rightarrow \mathbf{O(n^3)}$: Floyd, DP, Gaussian Elimination
  3. $\mathbf{n \le 1000} \rightarrow \mathbf{O(n^2)}$, $\mathbf{O(n^2 \log n)}$: DP, Binary Search, Naive Dijkstra, Naive Prim, Bellman-Ford
  4. $\mathbf{n \le 10000} \rightarrow \mathbf{O(n^{\frac{3}{2}})}$: Block Linked List, Mo's Algorithm
  5. $\mathbf{n \le 100000} \rightarrow \mathbf{O(n \log n)}$: sort, Segment Tree, Fenwick Tree (BIT), set/map, Heap, Topological Sort, Dijkstra (heap optimized), Prim (heap optimized), Kruskal, SPFA, Convex Hull, Half Plane Intersection, Binary Search, CDQ Divide and Conquer, Overall Binary Search, Suffix Array, Heavy-Light Decomposition, Dynamic Trees
  6. $\mathbf{n \le 1000000} \rightarrow \mathbf{O(n)}$, or small-constant $\mathbf{O(n \log n)}$: Monotonic Queue, Hashing, Two Pointers, BFS, Union Find, KMP, Aho-Corasick Automaton
  7. $\mathbf{n \le 10000000} \rightarrow \mathbf{O(n)}$: Two Pointers, KMP, Aho-Corasick Automaton, Linear Sieve for Primes
  8. $\mathbf{n \le 10^9} \rightarrow \mathbf{O(\sqrt{n})}$: Primality Testing
  9. $\mathbf{n \le 10^{18}} \rightarrow \mathbf{O(\log n)}$: GCD, Fast Exponentiation, Digit DP
  10. $\mathbf{n \le 10^{1000}} \rightarrow \mathbf{O((\log n)^2)}$: Big Integer Arithmetic (Add/Subtract/Multiply/Divide)
  11. $\mathbf{n \le 10^{100000}} \rightarrow \mathbf{O(\log k \cdot \log\log k)}$, where k is the number of digits: Big Integer Add/Subtract, FFT/NTT

0.4 If <bits/stdc++.h> Failed

Replace it with:

```
#include <algorithm>
#include <bitset>
#include <complex>
#include <deque>
#include <exception>
#include <fstream>
#include <functional>
#include <iomanip>
#include <ios>
#include <iosfwd>
#include <iostream>
#include <istream>
#include <iterator>
#include <limits>
#include <list>
#include <locale>
#include <map>
#include <memory>
#include <numeric>
#include <ostream>
#include <queue>
#include <set>
#include <sstream>
#include <stack>
#include <stdexcept>
#include <streambuf>
#include <string>
#include <typeinfo>
#include <utility>
#include <valarray>
#include <vector>
```

1 ⋆ Basic Algorithm

1.1 Quick Sort

Sort the given array from index 1 to n.

```
void quick_sort(int l, int r)
{
    if (l >= r) return;
    int x = a[(l + r) >> 1], i = l - 1, j = r + 1;
    while (i < j)
    {
        do i++; while (a[i] < x);
        do j--; while (a[j] > x);
        if (i < j) swap(a[i], a[j]);
    }
    quick_sort(l, j);
    quick_sort(j + 1, r);
    return;
}
```

1.2 Binary Search

```
// 区间 [l, r] 被划分成 [l, mid] 和 [mid + 1, r] 时使用
// 大于等于区间的最小值，check 应为 target <= a[mid]
int bsearch_1(int l, int r)
{
    while (l < r)
    {
        int mid = l + r >> 1;
        if (check(mid)) r = mid;
        else l = mid + 1;
    }
    return l;
}
// 区间 [l, r] 被划分成 [l, mid - 1] 和 [mid, r] 时使用
// 小于等于区间的最大值，check 应为 target >= a[mid]
int bsearch_2(int l, int r)
{
    while (l < r)
    {
        // 为什么要 l + r + 1：因为 l 的更新条件是 mid 本身
        // 当 r == l + 1 时 mid 向下取整必定取 l，有可能在满足 check(mid) 时导致无限循环
        int mid = l + r + 1 >> 1;
        if (check(mid)) l = mid;
        else r = mid - 1;
    }
    return l;
}
// 浮点数二分
double bsearch_3(double l, double r)
{
    // eps 表示精度，取决于题目对精度的要求
    const double eps = 1e-6;
    while (r - l > eps)
    {
        double mid = (l + r) / 2;
        if (check(mid))  r = mid;
        else l = mid;
    }
    return l;
}
```

1.3 High Precision

1.3.1 High Precision Add

```
string s1, s2;
vector<int> a, b, c;
void add(vector<int> &a, vector<int> &b)
{
    if (a.size() < b.size())
    { add(b, a); return; }
    int t = 0;
    for (int i = 0; i < a.size(); i++)
    {
        t += a[i];
        if (i < b.size()) t += b[i];
        c.push_back(t % 10);
        t /= 10;
    }
    while (t)
        c.push_back(t % 10), t /= 10;
}
int main()
{
    cin >> s1 >> s2;
    for (int i = s1.size() - 1; i >= 0; i--)
        a.push_back(s1[i] - '0');
    for (int i = s2.size() - 1; i >= 0; i--)
        b.push_back(s2[i] - '0');
    add(a, b);
    for (int i = c.size() - 1; i >= 0; i--)
        cout << c[i];
    return 0;
}
```

1.3.2 High Precision Subsection

```
vector<int> a, b, c;
string s1, s2;
void sub(vector<int> &a, vector<int> &b)
{
    int t = 0;
    for (int i = 0; i < a.size(); i++)
    {
        t = a[i] - t;
        if (i < b.size()) t -= b[i];
        c.push_back((t + 10) % 10);
        if (t < 0) t = 1;
        else t = 0;
    }
```

```cpp
    while (c.size() > 1 && c.back() == 0)
        c.pop_back();
}
int main()
{
    cin >> s1 >> s2;
    for (int i = s1.size() - 1; i >= 0; i--)
        a.push_back(s1[i] - '0');
    for (int i = s2.size() - 1; i >= 0; i--)
        b.push_back(s2[i] - '0');
    if (s1.size() < s2.size())
        cout << '-', sub(b, a);
    else if (s1.size() == s2.size() && s1 < s2)
        cout << '-', sub(b, a);
    else sub(a, b);
    for (int i = c.size() - 1; i >= 0; i--)
        cout << c[i];
    return 0;
}
```

1.3.3 High Precision Multiply

```cpp
string s1, s2;
vector<int> a, c;
int b;
void mul(vector<int> &a, int b)
{
    for (int i = 0, t = 0; i < a.size() || t; i++)
    {
        if (i < a.size()) t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }
    while (c.size() > 1 && c.back() == 0)
        c.pop_back();
}
int main()
{
    cin >> s1 >> b;
    for (int i = s1.size() - 1; i >= 0; i--)
        a.push_back(s1[i] - '0');
    mul(a, b);
    for (int i = c.size() - 1; i >= 0; i--)
        cout << c[i];
    return 0;
}
```

1.3.4 High Precision Divide

```cpp
string s1, s2;
vector<int> a, c;
int b, r;
void divide(vector<int> &a, int b, int &r)
{
    r = 0;
    for (int i = a.size() - 1; i >= 0; i--)
    {
        r = r * 10 + a[i];
        c.push_back(r / b);
        r %= b;
    }
    reverse(c.begin(), c.end());
    while (c.size() > 1 && c.back() == 0)
        c.pop_back();
}
int main()
{
    cin >> s1 >> b;
    for (int i = s1.size() - 1; i >= 0; i--)
        a.push_back(s1[i] - '0');
    divide(a, b, r);
    for (int i = c.size() - 1; i >= 0; i--)
        cout << c[i];
    cout << '\n' << r;
    return 0;
}
```

1.4 Prefix Sum & Difference Array

1.4.1 1D Prefix Sum

```
S[i] = a[1] + a[2] + ... a[i]
a[l] + ... + a[r] = S[r] - S[l - 1]
```

1.4.2 2D Prefix Sum

```
// S[i, j] = i 行 j 列左上部分所有元素和为:
s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1] + a[i][j]
// 以 (x1, y1) 为左上角，(x2, y2) 为右下角的子矩阵的和为:
S[x2][y2] - S[x1 - 1][y2] - S[x2][y1 - 1] + S[x1 - 1][y1 - 1]
```

1.4.3 1D Difference Array

```cpp
const int N = 100010;
int n, m;
int a[N], b[N];
void insert(int l, int r, int c)
{ b[l] += c; b[r + 1] -= c; }
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i <= n; i++)
        insert(i, i, a[i]);
    while (m--)
```

```
    {
        int l, r, c;
        cin >> l >> r >> c;
        insert(l, r, c);
    }
    for (int i = 1; i <= n; i++)
        b[i] += b[i - 1],
        cout << b[i] << ' ';
    return 0;
}
```

1.4.4 2D Difference Array

```
const int N = 1010;
int n, m, q, a[N][N], b[N][N];
void insert(int x1, int y1, int x2, int y2
    , int c)
{
    b[x1][y1] += c;
    b[x2 + 1][y2 + 1] += c;
    b[x1][y2 + 1] -= c;
    b[x2 + 1][y1] -= c;
}
int main()
{
    cin >> n >> m >> q;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> a[i][j];
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            insert(i, j, i, j, a[i][j]);
    while (q--)
    {
        int x1, x2, y1, y2, c;
        cin >> x1 >> y1 >> x2 >> y2 >> c;
        insert(x1, y1, x2, y2, c);
    }
    // 其他过程略
}
```

2 ⋆ Basic Data Structures

2.1 Linked List

2.1.1 Singly Linked List

```
const int N = 100010;
int n, h[N], e[N], ne[N], idx = 1;
void init() { ne[0] = -1; }
void insert(int k, int x) //  第 k 个节点后插入
{ e[idx] = x, ne[idx] = ne[k], ne[k] = idx++; }
void del(int k) // 第 k 个节点后删除
{ ne[k] = ne[ne[k]]; }
```

2.1.2 Bidirectional Linked List

```
const int N = 100010;
int n, r[N], l[N], e[N], idx = 2;
void init() { r[0] = 1; l[1] = 0; }
void insert(int k, int x) // 第 k 个节点后插入
{
    e[idx] = x;
    r[idx] = r[k];
    l[idx] = k;
    l[r[k]] = idx;
    r[k] = idx++;
}
void remove(int k) // 删除 k 本身
{ r[l[k]] = r[k]; l[r[k]] = l[k]; }
```

2.2 Stack & Queue

2.2.1 Monotonic Stack

```
// 常见模型：找出每个数左边离它最近的比它大/小的数
int tt = 0;
for (int i = 1; i <= n; i ++ )
{
    while (tt && check(stk[tt], i)) tt --;
    stk[++tt] = i;
}
```

2.2.2 Monotonic Queue

```
// 常见模型：找出滑动窗口中的最大值/最小值
int hh = 0, tt = -1;
for (int i = 0; i < n; i ++ )
{
    while (hh <= tt && check_out(q[hh]))
        hh++; // 判断队头是否滑出窗口
    while (hh <= tt && check(q[tt], i))
        tt-- ;
    q[++tt] = i;
}
```

2.3 KMP

```
const int N = 100010, M = 1000010;
int n, m;
char p[N], s[M];
void getNext(int ne[])
{
    for (int i = 2, j = 0; i <= n; i++)
    {
        while (j && p[j + 1] != p[i])
            j = ne[j];
        if (p[j + 1] == p[i]) j++;
        ne[i] = j;
    }
}
int KMP()
{
    int *ne = new int[n + 1];
    getNext(ne);
    for (int i = 1, j = 0; i <= m; i++)
    {
        while (j && p[j + 1] != s[i])
            j = ne[j];
        if (p[j + 1] == s[i]) j++;
        if (j == n) cout << i - n << ' ';
    }
    return -1;
}
```

2.4 Trie

```
const int N = 100010;
int trie[N][26], cnt[N], idx = 0;
void insert(string &str)    // 插入到 Trie 数组
{
    int p = 0;
    for (auto c : str)
    {
        int u = c - 'a';
        if (!trie[p][u])
            trie[p][u] = ++idx;
        p = trie[p][u];
    }
    cnt[p]++;
}
int query(string &str)      // 查询字符串出现的次数
{
    int p = 0;
    for (auto c : str)
    {
        int u = c - 'a';
        if (!trie[p][u]) return 0;
        p = trie[p][u];
    }
```

```
    return cnt[p];
}
```

2.5 Disjoint-Set

```
const int N = 100010;
int n, m, p[N], Size[N], D[N];
void init()
{
    for (int i = 1; i <= n; i ++ )
        p[i] = i, Size[i] = 1, D[i] = 0;
}
int find(int x)
{
    if (p[x] != x)
    {
        int u = find(p[x]);
        D[x] += D[p[x]];  // 视具体情况计算
        p[x] = u;
    }
    return p[x];
}
void merge(int a, int b, int distance)
{
    int x = find(a), y = find(b);
    if(x != y)
    {
        p[x] = y;
        D[x] = distance;  // 视具体情况计算
        Size[y] += Size[x];
    }
}
```

2.6 Hash

2.6.1 Simple Hash

```
// (1) 拉链法
int h[N], e[N], ne[N], idx;
void insert(int x)
{
    int k = (x % N + N) % N;
    e[idx] = x, ne[idx] = h[k], h[k] = idx ++ ;
}
bool find(int x)
{
    for (int i = h[(x % N + N) % N]; i != -1; i = ne[i])
        if (e[i] == x) return true;
    return false;
}
// (2) 开放寻址法
int find(int x)
{
    int t = (x % N + N) % N;
    while (h[t] != null && h[t] != x)
    { t ++ ; if (t == N) t = 0; }
    return t;
}
```

2.6.2 String Hash

```
typedef unsigned long long ULL;
ULL h[N], p[N];
void init()
{
    p[0] = 1;
    for (int i = 1; i <= n; i ++ ) { h[i] = h[i - 1] * P + str[i]; p[i] = p[i - 1] * P; }
}
ULL get(int l, int r) { return h[r] - h[l - 1] * p[r - l + 1]; }
```

2.7 STL

```
// vector
size()      返回元素个数
empty()     返回是否为空
clear()     清空
front()/back()
push_back()/pop_back()
begin()/end()
[]
支持比较运算，按字典序
// pair<int, int>
first       第一个元素
second      第二个元素
支持比较运算，以first为第一关键字，以second为第二关键字（字典序）
// string
size()/length()  返回字符串长度
empty()
clear()
substr(起始下标，(子串长度))  返回子串
c_str()  返回字符串所在字符数组的起始地址
// queue
size()
empty()
push()      向队尾插入一个元素
front()     返回队头元素
back()      返回队尾元素
pop()       弹出队头元素
// priority_queue
size()
empty()
push()      插入一个元素
top()       返回堆顶元素
pop()       弹出堆顶元素
定义成小根堆的方式：priority_queue<int, vector<int>, greater<int>> q;
// stack
size()
empty()
push()      向栈顶插入一个元素
top()       返回栈顶元素
pop()       弹出栈顶元素
// deque
size()
empty()
clear()
front()/back()
push_back()/pop_back()
```

```
push_front()/pop_front()
begin()/end()
[]
// set, map, multiset, multimap：基于平衡二叉树（红黑树）动态维护有序序列
size()
empty()
clear()
begin()/end()
++, -- 返回前驱和后继，时间复杂度 O(logn)
// set/multiset
    insert()  插入一个数
    find()    查找一个数
    count()   返回某一个数的个数
    erase()
        (1) 输入是一个数x，删除所有x，O(k + logn)
        (2) 输入一个迭代器，删除这个迭代器
    lower_bound()/upper_bound()
        lower_bound(x)  返回大于等于x的最小的数的迭代器
        upper_bound(x)  返回大于x的最小的数的迭代器
// map/multimap
    insert()  插入的数是一个pair
    erase()   输入的参数是pair或者迭代器
    find()
    []        注意multimap不支持此操作。 时间复杂度是 O(logn)
    lower_bound()/upper_bound()
// unordered_set, unordered_map, unordered_multiset, unordered_multimap
增删改查的时间复杂度是 O(1)
不支持 lower_bound()/upper_bound()， 迭代器的++，--
// bitset
bitset<10000> s;
~, &, |, ^
>>, <<
==, !=
[]
count()     返回有多少个1
any()       判断是否至少有一个1
none()      判断是否全为0
set()       把所有位置成1
set(k, v)   将第k位变成v
reset()     把所有位变成0
flip()      等价于~
flip(k)     把第k位取反
```

3 ⋆ Search & Graph Theory

3.1 Representation of Tree & Graph

3.1.1 Adjacency Matrix

```
// g[a][b] = a->b
```

3.1.2 Adjacency List

```
int h[N], e[N], ne[N], idx;
void init() { memeset(h, -1, sizeof h); }
void add(int a, int b) { e[idx] = b, ne[idx] = h[a], h[a] = idx++ ; }
```

3.2 DFS & BFS

3.2.1 DFS

```
int dfs(int u)
{
    st[u] = true; // 表示点 u 已经被遍历过
    for (int i = h[u]; i != -1; i = ne[i])
    { int j = e[i]; if (!st[j]) dfs(j); }
}
```

3.2.2 BFS

```
queue<int> q;
st[1] = true; q.push(1);
while (q.size())
{
    int t = q.front(); q.pop();
    for (int i = h[t]; i != -1; i = ne[i])
        if (!st[e[i]]) { st[e[i]] = true; q.push(e[i]); }
}
```

3.3 Topological Sort

```
const int N = 100010;
int e[2 * N], ne[2 * N], h[N], d[N], idx;
int n, m, q[N];
void init() { memset(h, -1, sizeof h); }
void add(int a, int b) { e[idx] = b, ne[idx] = h[a], h[a] = idx++, d[b]++; }
bool topSort()
{
    int hh = 0, tt = -1;
    for (int i = 1; i <= n; i++)
        if (!d[i]) q[++tt] = i;
    while (hh <= tt)
        for (int i = h[q[hh++]]; ~i; i = ne[i])
            if (--d[e[i]] == 0) q[++tt] = e[i];
    return tt == n - 1;
}
```

3.4 Shortest Path

3.4.1 Dijkstra

```
const int N = 1010;
int n, dist[N];
int h[N], w[N], e[N], ne[N], idx;
bool st[N];
void add(int a, int b, int c) { e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++; }
int dijkstra()      // 需要初始化 dist 与 h
{
    dist[1] = 0;
    priority_queue<PII, vector<PII>, greater<PII>> heap;
    heap.push({0, 1});
    while (heap.size())
    {
        auto t = heap.top();
        heap.pop();
        int ver = t.second, distance = t.first;
        if (st[ver]) continue;
        st[ver] = true;
        for (int i = h[ver]; i != -1; i = ne[i])
            if (dist[e[i]] > distance + w[i])
            {
                dist[e[i]] = distance + w[i];
                heap.push({dist[e[i]], e[i]});
            }
    }
    if (dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}
```

3.4.2 Bellman-Ford

```
const int N = 100010;
int n, m, dist[N], backup[N];
struct Edge
{
    int a, b, w;
}edges[N];
int bellman_ford()
{
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    for (int i = 0; i < n; i ++ )
    {
```

```cpp
        memcpy(backup, dist, sizeof dist);
        for (int j = 0; j < m; j++)
        {
            int a = edges[j].a, b = edges[j].b, w = edges[j].w;
            dist[b] = min(dist[b], backup[a] + w);
        }
    }
    if (dist[n] > 0x3f3f3f3f / 2) return -1;
    return dist[n];
}
```

3.4.3 SPFA

```cpp
const int N = 100010;
int n, m, dist[N];
int e[2 * N], ne[2 * N], w[2 * N], h[N], idx;
bool vis[N];
void spfa()     // 需要初始化 dist 与 h
{
    queue<int> q;
    q.push(1); vis[1] = true;
    while (q.size())
    {
        int t = q.front();
        q.pop();
        vis[t] = false;
        for (int i = h[t]; ~i; i = ne[i])
            if (dist[e[i]] > dist[t] + w[i])
            {
                dist[e[i]] = dist[t] + w[i];
                if (!vis[e[i]]) vis[e[i]] = true, q.push(j);
            }
    }
    dist[n] > INF / 2 ? cout << "impossible" : cout << dist[n];
}
```

3.4.4 Detecting Negative Circle in SPFA

```cpp
void spfa()     // 只需要初始化 h
{
    queue<int> q;
    // 基于虚拟原点假设，所有点放入队列
    for (int i = 1; i <= n; i++) q.push(i), st[i] = true;
    while (q.size())
    {
        int t = q.front();
        q.pop();
        vis[t] = false;
        for (int i = h[t]; ~i; i = ne[i])
            if (dist[e[i]] > dist[t] + w[i])
            {
                dist[e[i]] = dist[t] + w[i];
                // 新增
                cnt[j] = cnt[t] + 1;
                if (cnt[j] >= n) return true;
                if (!st[j]) q.push(j), st[j] = true;
            }
    }
    return false;
}
```

3.4.5 Floyd

```cpp
const int N = 210;
int g[N][N], n, m, k;
int main()
{
    cin >> n >> m >> k;
    memset(g, 0x3f, sizeof g);
    for (int i = 1; i <= n; i++) g[i][i] = 0;
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        g[a][b] = min(g[a][b], c);
    }
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                g[i][j] = min(g[i][k] + g[k][j], g[i][j]);
    // 后续代码略
    return 0;
}
```

3.5 Minimum Spanning Tree

3.5.1 Prim

```cpp
const int N = 510;
int n, m, g[N][N], dist[N];
bool vis[N];
void prim()
{
    int res = 0;
    for (int i = 0; i < n; i++)
    {
        int t = -1;
        for (int j = 1; j <= n; j++)
            if (!vis[j] && (t == -1 || dist[j] < dist[t])) t = j;
        if (i && dist[t] == INF) { res = INF; break; }
        if (i) res += dist[t];
        vis[t] = true;
        for (int j = 1; j <= n; j++) dist[j] = min(dist[j], g[t][j]);
```

```
    }
    res == INF ? cout << "impossible" :
    cout << res;
}
int main()
{
    memset(g, 0x3f, sizeof g);
    memset(dist, 0x3f, sizeof dist);
    cin >> n >> m;
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        g[a][b] = min(g[a][b], c);
        g[b][a] = min(g[b][a], c);
    }
    prim();
    return 0;
}
```

3.5.2 Kruskal

```
const int N = 100010;
int n, m;
int p[N];
struct Edge
{
    int a, b, w;
    bool operator<(const Edge &e) const {
    return w < e.w; };
} edge[2 * N];
void init() { for (int i = 1; i <= n; i++)
     p[i] = i; }
int find(int x)
{
    if (x != p[x]) p[x] = find(p[x]);
    return p[x];
}
void merge(int x, int y) { p[find(x)] =
    find(y); }
void kruskal()
{
    int res = 0, cnt = 0;
    for (int i = 1; i <= m; i++)
        if (find(edge[i].a) != find(edge[i
    ].b))
        {
            merge(edge[i].a, edge[i].b);
            res += edge[i].w;
            cnt++;
        }
    if (cnt < n - 1) res = INF;
    res == INF ? cout << "impossible" :
    cout << res;
}
int main()
{
    init();
    cin >> n >> m;
    for (int i = 1; i <= m; i++) cin >>
    edge[i].a >> edge[i].b >> edge[i].w;
    sort(edge + 1, edge + m + 1);
    kruskal();
    return 0;
}
```

3.6 Bipartite Graph

3.6.1 Coloring Method

To check if a given graph is bipartite.

```
const int N = 100010, M = 200010;
int n, m;
int e[M], ne[M], h[N], color[N], idx;
bool dfs(int u, int c)
{
color[u] = c;
for (int i = h[u]; ~i; i = ne[i])
    if (color[e[i]] == -1)
    {
        if (!dfs(e[i], !c)) return false;
    }
    else if (color[e[i]] == c) return
    false;
return true;
}
bool check()
{
for (int i = 1; i <= n; i++)
    if (color[i] == -1)
        if (!dfs(i, 0)) return false;
return true;
}
int main()
{
// 注意另外初始化 h 与 color
cin >> n >> m;
while (m--)
{
    int a, b;
    cin >> a >> b;
    add(a, b), add(b, a);
}
// 其余过程略
}
```

3.6.2 Hungarian Algorithm

To find the maximum matching for a given graph.

```cpp
const int N = 510, M = 100010;
int n1, n2, m;
int e[M], ne[M], h[N], match[N], idx;
bool vis[N];
bool find(int x)
{
for (int i = h[x]; ~i; i = ne[i])
    if (!vis[e[i]])
    {
        vis[e[i]] = true;
        if (match[e[i]] == 0 || find(match[e[i]]))
        {
            match[e[i]] = x;
            return true;
        }
    }
return false;
}
int main()
{
// 注意初始化 h
cin >> n1 >> n2 >> m;
while (m--)
{
    int a, b;
    cin >> a >> b;
    add(a, b);
}
int res = 0;
for (int i = 1; i <= n1; i++)
{
    memset(vis, false, sizeof vis);
    if (find(i)) res++;
}
cout << res;
return 0;
}
```

4 ⋆ Basic Math

4.1 Prime Numbers

4.1.1 Judging Prime Numbers

$O(\sqrt{n})$

```
bool is_prime(int x)
{
    if (x < 2) return false;
    for (int i = 2; i <= x / i; i ++ )
        if (x % i == 0) return false;
    return true;
}
```

4.1.2 Prime Factorization

```
void divide(int x)
{
    for (int i = 2; i <= x / i; i ++ )
        if (x % i == 0)
        {   // 此条件成立时 i 一定是质数
            int s = 0;
            while (x % i == 0) x /= i, s ++ ;
            cout << i << ' ' << s << '\n';
        }
    if (x > 1) cout << x << ' ' << 1 << '\n'
}
```

4.1.3 Euler's Sieve

```
int primes[N], cnt;
bool st[N];
void get_primes(int n)
{
    for (int i = 2; i <= n; i ++ )
    {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] <= n / i; j ++ )
        {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}
```

4.2 Divisor

4.2.1 Find All Divisors

```
vector<int> get_divisors(int x)
{
    vector<int> res;
    for (int i = 1; i <= x / i; i ++ )
        if (x % i == 0)
        {
            res.push_back(i);
            if (i != x / i) res.push_back(x / i);
        }
    sort(res.begin(), res.end());
    return res;
}
```

4.2.2 The Number of Divisors

```
const int mod = 1e9 + 7;
int n;
int main()
{
    cin >> n;
    unordered_map<int, int> h;
    while (n--)
    {
        int x;
        cin >> x;
        for (int i = 2; i <= x / i; i++)
            while (x % i == 0) { h[i]++; x = x / i; }
        if (x > 1) h[x]++;
    }
    long long res = 1;
    for (auto iter = h.begin(); iter != h.end(); iter++)
        res = res * (iter->second + 1) % mod;
    cout << res;
    return 0;
}
```

4.2.3 The Sum of Divisors

```
const int mod = 1e9 + 7;
int n;
long long getSum(int x, int c)
{
    long long s = 1;
    while(c--) s = (s * x + 1) % mod;
    return s;
}
int main()
{
    cin >> n;
    unordered_map<int, int> h;
    while (n--)
    {
        int x;
        cin >> x;
        for (int i = 2; i <= x / i; i++)
            while (x % i == 0) { h[i]++; x = x / i; }
        if (x > 1) h[x]++;
    }
    long long res = 1;
```

```
    for (auto iter = h.begin(); iter != h.end(); iter++)
        res = res * getSum(iter->first, iter->second) % mod;
    cout << res;
    return 0;
}
```

4.2.4 Euclidean Algorithm

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

4.3 Euler Function

4.3.1 Simple Method

```
int phi(int x)
{
    int res = x;
    for (int i = 2; i <= x / i; i ++ )
        if (x % i == 0)
        {
            res = res / i * (i - 1);
            while (x % i == 0) x /= i;
        }
    if (x > 1) res = res / x * (x - 1);
    return res;
}
```

4.3.2 Euler's Sieve Method

```
const int N = 1000010;
int n, primes[N], phi[N], cnt;
bool st[N];
void getEuler()
{
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        if (!st[i])
        {
            primes[cnt++] = i;
            // i 是质数，它只会被本身整除，所以直接赋值 i - 1
            phi[i] = i - 1;
        }
        for (int j = 0; primes[j] <= n / i; j++)
        {
            st[i * primes[j]] = true;
            if (i % primes[j] == 0)
            {
                // 如果 i % primes[j] == 0 成立表示 primes[j] 是 i 的最小质因子
                // 也是 primes[j] * i 的最小质因子
                // 1 - 1 / primes[j] 这一项在 phi[i] 中计算过了，只需将基数 N 修正为 primes[j] 倍
                phi[primes[j] * i] = phi[i] * primes[j];
                break;
            }
            // 否则，primes[j] 不是 i 的质因子，只是 primes[j] * i 的最小质因子
            // 不仅需要将基数 N 修正为 primes[j] 倍
            // 还需要补上 1 - 1 / primes[j] 的分子项，因此最终结果为 phi[i] * (primes[j] - 1)
            phi[primes[j] * i] = phi[i] * (primes[j] - 1);
        }
    }
}
```

4.4 Exponentiating by Squaring

```
LL qmi(int m, int k, int p)
{
    LL res = 1 % p, t = m;
    while (k)
    {
        if (k&1) res = res * t % p;
        t = t * t % p;
        k >>= 1;
    }
    return res;
}
```

4.5 Extended Euclidean Algorithm

```
int exgcd(int a, int b, int &x, int &y)
{
    if (!b)
    {
        x = 1;
        y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return d;
}
```

4.6 Chinese Remainder Theorem

```
LL exgcd(LL a, LL b, LL &x, LL &y)
{
    if (!b) { x = 1, y = 0; return a; }
```

```cpp
    LL d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
int main()
{
    int n;
    cin >> n;
    LL x = 0, m1, a1;
    cin >> m1 >> a1;
    for (int i = 0; i < n - 1; i++)
    {
        LL m2, a2;
        cin >> m2 >> a2;
        LL k1, k2;
        LL d = exgcd(m1, m2, k1, k2);
        if ((a2 - a1) % d) { x = -1; break; }
        k1 *= (a2 - a1) / d;
        k1 = (k1 % (m2 / d) + m2 / d) % (m2 / d);
        x = k1 * m1 + a1;
        LL m = abs(m1 / d * m2);
        a1 = k1 * m1 + a1;
        m1 = m;
    }
    if (x != -1)
        x = (a1 % m1 + m1) % m1;
    cout << x << '\n';
    return 0;
}
```

4.7 Gauss-Jordan Elimination

4.7.1 Linear Equation Group

```cpp
int gauss()
{
    int c, r;
    for (c = 0, r = 0; c < n; c++)
    {
        int t = r;
        for (int i = r; i < n; i++)      // 找绝对值最大的行
            if (fabs(a[i][c]) > fabs(a[t][c]))
                t = i;
        if (fabs(a[t][c]) < eps)         // 此时没必要对该列该行处理
            continue;
        for (int i = c; i <= n; i++)
            swap(a[t][i], a[r][i]);      // 将绝对值最大的行换到最顶端
        for (int i = n; i >= c; i--)
            a[r][i] /= a[r][c];          // 将当前行的首位变成1
        for (int i = r + 1; i < n; i++) // 用当前行将下面所有的列消成0
            if (fabs(a[i][c]) > eps)
                for (int j = n; j >= c; j--)
                    a[i][j] -= a[r][j] * a[i][c];
        r++;
    }
    if (r < n)
    {
        for (int i = r; i < n; i++)
            if (fabs(a[i][n]) > eps)
                return 2; // 无解
        return 1;         // 有无穷多组解
    }
    for (int i = n - 1; i >= 0; i--)
        for (int j = i + 1; j < n; j++)
            a[i][n] -= a[i][j] * a[j][n];
    return 0;            // 有解
}
```

4.7.2 XOR Linear Equation Group

```cpp
int gauss()
{
    int c, r;
    for (c = 0, r = 0; c < n; c++)
    {
        int t = r;
        for (int i = r; i < n; i++)
            if (a[i][c])
                t = i;
        if (!a[t][c])
            continue;
        for (int i = c; i <= n; i++)
            swap(a[r][i], a[t][i]);
        for (int i = r + 1; i < n; i++)
            if (a[i][c])
                for (int j = n; j >= c; j--)
                    a[i][j] ^= a[r][j];
        r++;
    }
    if (r < n)
    {
        for (int i = r; i < n; i++)
            if (a[i][n])
                return 2;
        return 1;
    }
    for (int i = n - 1; i >= 0; i--)
        for (int j = i + 1; j < n; j++)
            a[i][n] ^= a[i][j] * a[j][n];
    return 0;
}
```

4.8 Combinatorial Counting

4.8.1 Recurrence Relation

```cpp
void init()
{
    for (int i = 0; i < N; i++)
        for (int j = 0; j <= i; j++)
            if (!j) c[i][j] = 1;
            else c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % mod;
```

```
}
```

4.8.2 Preprocessing & Inverse Element

```
const int N = 100010, mod = 1e9 + 7;
int n, fact[N], infact[N];
int qmi(int a, int b, int p)
{
    int res = 1;
    while (b)
    {
        if (b & 1)
            res = (LL)res * a % p;
        a = (LL)a * a % p;
        b >>= 1;
    }
    return res;
}
int main()
{
    fact[0] = infact[0] = 1;
    for (int i = 1; i < N; i++)
    {
        fact[i] = (LL)fact[i - 1] * i % mod;
        infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
    }
    // 此后 C(a, b) = (LL)fact[a] * infact[b] % mod * infact[a - b] % mod
}
```

4.8.3 Lucas Theorem

```
int qmi(int a, int k, int p)
{
    int res = 1 % p;
    while (k)
    {
        if (k & 1)
            res = (LL)res * a % p;
        a = (LL)a * a % p;
        k >>= 1;
    }
    return res;
}
int C(int a, int b, int p)
{
    if (a < b) return 0;
    LL x = 1, y = 1;
    // x = a * (a - 1) * (a - 2) * ... * (a - b + 1) = a! / (a - b)! (mod p)
    // y = 1 * 2 * ... * b = b! (mod p)
    for (int i = a, j = 1; j <= b; i--, j++)
    { x = (LL)x * i % p; y = (LL)y * j % p; }
    return x * (LL)qmi(y, p - 2, p) % p;
}
int lucas(LL a, LL b, int p)
{
    if (a < p && b < p)
        return C(a, b, p);
    return (LL)C(a % p, b % p, p) * lucas(a / p, b / p, p) % p;
}
```

4.8.4 Factorization Method

```
const int N = 5010;
int n, primes[N], sum[N], cnt;
bool st[N];
void getPrimes(int n) { // 略 }
// 求 n! 中 p 的幂次
int get(int n, int p)
{
    int res = 0;
    while (n) { res += n / p; n /= p; }
    return res;
}
void mul(vector<int> &a, int b) { // 高精度乘，略 }
int main()
{
    int a, b;
    cin >> a >> b;
    getPrimes(a);
    for (int i = 0; i < cnt; i++)
    {
        int p = primes[i];
        sum[i] = get(a, p) - get(b, p) - get(a - b, p);
    }
    vector<int> res;
    res.push_back(1);
    for (int i = 0; i < cnt; i++)
        for (int j = 0; j < sum[i]; j++)
            mul(res, primes[i]);
    for (int i = res.size() - 1; i >= 0; i--)
        cout << res[i];
}
```

4.8.5 Catalan Number

```
const int N = 100010, mod = 1e9 + 7;
int qmi(int a, int k, int p) { // 略 }
int main()
{
    int n;
    cin >> n;
    int a = n * 2, b = n, res = 1;
    for (int i = a; i > a - b; i--)
        res = (LL)res * i % mod;
    for (int i = 1; i <= b; i++)
        res = (LL)res * qmi(i, mod - 2, mod) % mod;
    res = (LL)res * qmi(n + 1, mod - 2, mod) % mod;
}
```

4.9 Inclusion-Exclusion Principle

```
const int N = 20;
int n, m, res = 0, p[N];
int main()
{
    cin >> n >> m;
    for (int i = 0; i < m; i++)
        cin >> p[i];
    // 使用二进制数字表示数字选取情况
    for (int i = 1; i < 1 << m; i++)
    {
        int t = 1, cnt = 0;
        // 遍历每个被选取的质数
        for (int j = 0; j < m; j++)
            if (i >> j & 1)
            {
                cnt++;
                // 一个质数能被选取的条件应该是其累乘积不超过目标数字
                if ((LL)t * p[j] > n)
                { t = -1; break; }
                t *= p[j];
            }
        if (t != -1)
            // 容斥原理公式中奇数个并集系数为 1，反之为 -1
            if (cnt % 2) res += n / t;
            else res -= n / t;
    }
    cout << res;
}
```

4.10 Game Theory

4.10.1 NIM Game

```
const int N = 110, M = 100010;
int k, n,  s[N], f[M];
int sg(int x)
{
    if (f[x] != -1) return f[x];
    // 到达节点得 SG 函数集合
    unordered_set<int> S;
    // 能取走石子就说明能到达，并且递归向下求解
    for (int i = 0; i < k; i++)
    {
        int sum = s[i];
        if (x >= sum) S.insert(sg(x - sum));
    }
    // SG 从小到达遍历并返回，找到最小的、不包含在 SG 函数集合中的自然数
    for (int i = 0;; i++)
        if (!S.count(i))
            return f[x] = i;
}

int main()
{
    cin >> k;
    for (int i = 0; i < k; i++) cin >> s[i];
    cin >> n;
    memset(f, -1, sizeof f);
    int res = 0;
    // 每一堆石子都是一个入度为 0 的起始点
    for (int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
        res ^= sg(x);
    }
    res ? cout << "Yes" : cout << "No";
    return 0;
}
```

5 ⋆ Basic DP

5.1 Knapsack Problem

5.1.1 01 Knapsack

```
const int N = 1010;
int n, m, v[N], w[N], f[N];
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> v[i] >> w[i];
    for (int i = 1; i <= n; i++)
        for (int j = m; j >= v[i]; j++)
            f[j] = max(f[j], f[j - v[i]] + w[i]);
    cout << f[m];
}
```

5.1.2 Complete Knapsack

```
const int N = 1010;
int n, m, v[N], w[N], f[N];
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> v[i] >> w[i];
    for (int i = 1; i <= n; i++)
        for (int j = v[i]; j <= m; j++)
            f[j] = max(f[j], f[j - v[i]] + w[i]);
    cout << f[m];
}
```

5.1.3 Mutiple Knapsack

```
const int N = 25000;
int n, m, v[N], w[N], f[N];
int main()
{
    cin >> n >> m;
    int cnt = 0;
    for (int i = 1; i <= n; i++)
    {
        int a, b, s;
        cin >> a >> b >> s;
        int k = 1;
        while (k <= s)
        {
            cnt++;
            v[cnt] = a * k, w[cnt] = b * k;
            s -= k, k *= 2;
        }
        if (s > 0)
        {
            cnt++;
            v[cnt] = a * s, w[cnt] = b * s;
        }
    }
    n = cnt;
    for (int i = 1; i <= n; i++)
        for (int j = m; j >= v[i]; j--)
            f[j] = max(f[j], f[j - v[i]] + w[i]);
    cout << f[m];
}
```

5.1.4 Grouped Knapsack

```
const int N = 120;
int n, m, s[N], v[N][N], w[N][N], f[N];
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        cin >> s[i];
        for (int j = 1; j <= s[i]; j++)
            cin >> v[i][j] >> w[i][j];
    }
    for (int i = 1; i <= n; i++)
        for (int j = m; j >= 0; j--)
            for (int k = 1; k <= s[i]; k++)
                if (v[i][k] <= j)
                    f[j] = max(f[j], f[j - v[i][k]] + w[i][k]);
    cout << f[m];
}
```

5.2 Linear DP

5.2.1 LIS

Here is an $O(n^2)$ solution:

```
const int N = 1010;
int n, a[N], f[N];
int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i <= n; i++)
    {
        f[i] = 1;
        for (int j = 1; j < i; j++)
            if (a[j] < a[i])
                f[i] = max(f[i], f[j] + 1);
    }
    int res = 0;
    for (int i = 1; i <= n; i++)
        res = max(res, f[i]);
    cout << res;
}
```

Another is an $O(nlogn)$ solution:

```
const int N = 100010;
int n, a[N], q[N];
int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    int len = 0;
    q[len] = -INF;
    for (int i = 1; i <= n; i++)
    {
        int l = 0, r = len;
        while (l < r)
        {
            int mid = l + r + 1 >> 1;
            if (q[mid] < a[i]) l = mid;
            else r = mid - 1;
        }
        len = max(r + 1, len);
        q[r + 1] = a[i];
    }
    cout << len;
}
```

5.2.2 LCS

```
const int N = 1010;
int n, m, f[N][N];
char a[N], b[N];
int main()
{
    cin >> n >> m >> (a + 1) >> (b + 1);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
        {
            f[i][j] = max(f[i - 1][j], f[i][j - 1]);
            if (a[i] == b[j])
                f[i][j] = max(f[i][j], f[i - 1][j - 1] + 1);
        }
    cout << f[n][m];
}
```

5.3 Interval DP

In this case we focus on an interval, whose sum of its elements can represent the answer we want to find:

```
const int N = 310;
int n, s[N], f[N][N];
int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> s[i], s[i] += s[i - 1];
    for (int len = 2; len <= n; len++)
        for (int i = 1; i + len - 1 <= n; i++)
        {
            int l = i, r = i + len - 1;
            f[l][r] = INF;
            for (int k = l; k < r; k++)
                f[l][r] = min(f[l][r], f[l][k] + f[k + 1][r] + s[r] - s[l - 1]);
        }
    cout << f[1][n];
}
```

5.4 Counting DP

```
const int N = 1010, M = 1e9 + 7;
int n, f[N][N];
int main()
{
    cin >> n;
    f[0][0] = 1;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= i; j++)
            f[i][j] = (f[i - 1][j - 1] + f[i - j][j]) % M;
    int ans = 0;
    for (int i = 1; i <= n; i++)
        ans = (ans + f[n][i]) % M;
    cout << ans;
}
```

5.5 Digit DP

```
// 求数 n 的位数
int get(int n)
{
    int res = 0;
    while (n) n /= 10, res++;
    return res;
}
int count(int n, int i)
{
    int res = 0, dgt = get(n);
    for (int j = 1; j <= dgt; j++)
    {
        // p 为当前遍历位次(第 j 位)的数大小 <10^(右边的数的位数)>, Ps: 从左往右(从高位到低位)
        // l 为第 j 位的左边的数, r 为右边的数, dj 为第 j 位上的数
        int p = pow(10, dgt - j), l = n / p / 10, r = n % p, dj = n / p % 10;
        // 求要选的数在 i 的左边的数小于 l 的情况:
        //      1)、当 i 不为 0 时 xxx : 0...0 ~ l - 1, 即 l * (右边的数的位数) == l * p 种选法
        //      2)、当 i 为 0 时 由于不能有前导零 故 xxx: 0....1 ~ l - 1, 即 (l - 1) * (右边的数的位数) == (l - 1) * p 种选法
        if (i) res += l * p;
        else res += (l - 1) * p;
        // 求要选的数在 i 的左边的数等于 l 的情况: (即视频中的xxx == l 时)
```

```
        //      1)、i > dj 时 0 种选法
        //      2)、i == dj 时 yyy : 0...0 ~ r 即 r + 1 种选法
        //      3)、i < dj 时 yyy : 0...0 ~ 9...9 即 10^(右边的数的位数) == p 种选法 */
        if (i == dj) res += r + 1;
        if (i < dj) res += p;
    }
    return res;
}
int main()
{
    int a, b;
    while (cin >> a >> b, a)
    {
        if (a > b) swap(a, b);
        for (int i = 0; i <= 9; ++i)
            cout << count(b, i) - count(a - 1, i) << ' ';
        // 利用前缀和思想： [l, r] 的和 = s[r] - s[l - 1]
        cout << '\n';
    }
}
```

5.6 State Compression DP

```
const int N = 12, M = 1 << 12;
int n, m;
LL f[N][M];
bool st[M];
int main()
{
    while (cin >> n >> m, n || m)
    {
        memset(f, 0, sizeof f);
        for (int i = 0; i < 1 << n; i++)
        {
            st[i] = true;
            // 统计连续 0 的个数，若连续 0 为奇数个就不能正好放得下竖放的方格
            int cnt = 0;
            for (int j = 0; j < n && st[i]; j++)
                if (i >> j & 1)
                {
                    // 当前格子被使用
                    // 如果连续 0 的数量为奇数个，当前格子被使用的后果就是导致格子重合，所以不可取
                    if (cnt & 1)
                        st[i] = false;
                    // 刷新状态
                    cnt = 0;
                }
                else cnt++;
            // 最后再判断一次，防止漏判
            if (cnt & 1)
                st[i] = false;
        }
        // 没有摆放任何棋子的状态默认只有 1 种取法
        f[0][0] = 1;
        // 遍历每一列
        for (int i = 1; i <= m; i++)
            // 遍历当前列的每一种用二进制数字表示的摆放状态： 1 指横向摆放， 0 指空位
            for (int j = 0; j < 1 << n; j++)
                // 遍历上一列的每一种用二进制数字表示的摆放状态： 1 指横向摆放， 0 指空位
                for (int k = 0; k < 1 << n; k++)
                    // 满足两个条件： 两列的摆放互不冲突； 两列摆放状态的结合状态是一个可取的状态则累加情况数
                    if (!(j & k) && st[j | k])
                        f[i][j] += f[i - 1][k];
        // 输出摆放好第 m 列且第 (m + 1) 列没有任何方格的状态数
        cout << f[m][0] << '\n';
    }
}
```

5.7 Tree DP

```
// Don't use I/O functions from stdio.h!!!
#define itn int
#define nit int
#define nti int
#define tin int
#define tni int
#define retrun return
#define reutrn return
#define rutren return
#define INF 0x3f3f3f3f
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> PII;
typedef long long LL;

const int N = 6010;

int n;
int e[N], ne[N], happy[N], h[N], idx;
int f[N][2];
bool has_father[N];
void add(int a, int b)
{ e[idx] = b, ne[idx] = h[a], h[a] = idx++; }
void dfs(int u)
{
    f[u][1] = happy[u];
    for (int i = h[u]; ~i; i = ne[i])
    {
        dfs(e[i]);
        f[u][0] += max(f[e[i]][0], f[e[i]][1]);
        f[u][1] += f[e[i]][0];
    }
}
```

```cpp
int main()
{
    memset(h, -1, sizeof h);
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> happy[i];
    for (int i = 0; i < n - 1; i++)
    {
        int a, b;
        cin >> a >> b;
        has_father[a] = true;
        add(b, a);
    }
    int root = 1;
    while (has_father[root]) root++;
    dfs(root);
    cout << max(f[root][0], f[root][1]);
}
```

5.8 Memoized Search

```cpp
const int N = 310;
int n, m,
h[N][N], f[N][N],
dx[4] = {0, 1, 0, -1}, dy[4] = {1, 0, -1, 0};
int dp(int x, int y)
{
    int &v = f[x][y];
    if (v != -1) return v;
    v = 1;
    for (int i = 0; i < 4; i++)
    {
        int a = x + dx[i], b = y + dy[i];
        if (a >= 1 && a <= n && b >= 1 && b <= m && h[a][b] < h[x][y])
            v = max(v, dp(a, b) + 1);
    }
    return v;
}
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> h[i][j];
    memset(f, -1, sizeof f);
    int res = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            res = max(res, dp(i, j));
    cout << res;
}
```

Part II: Advanced Template

Created by

Luliet Lyan & Bleu Echo

NSCC-GZ
School of Computer Science & Engineering
Sun Yat-Sen University

Supervisor: Dr Dan Huang
Co-Supervisor: Dr Zhiguang Chen

6 ⋆ Advanced Basic

6.1 Slow Multiplication

```
LL mul(LL a, LL b, LL p)
{
    LL ans = 0;
    while (b)
    {
        if (b & 1) ans = (ans + a) % p;
        a = a * 2 % p; b >>= 1;
    }
    return ans;
}
```

6.2 Sum of Geometric Series

```
const int mod = 9901;
int a, b;
int qmi(int a, int k)
{
    int res = 1;
    a %= mod;
    while (k)
    {
        if (k & 1)
            res = res * a % mod;
        a = a * a % mod;
        k >>= 1;
    }
    return res;
}
int sum(int p, int k)
{
    if (k == 1) return 1;
    if (k % 2 == 0)
        return (1 + qmi(p, k / 2)) * sum(p, k / 2) % mod;
    return (sum(p, k - 1) + qmi(p, k - 1)) % mod;
}
int main()
{
    // 以 a^b 约数之和为例求等比数列和
    cin >> a >> b;
    int res = 1;
    for (int i = 2; i <= a / i; i++)
        if (a % i == 0)
        {
            int s = 0;
            while (a % i == 0) a /= i, s++;
            res = res * sum(i, b * s + 1) % mod;
        }
    if (a > 1) res = res * sum(a, b + 1) % mod;
}
```

6.3 Sort

6.3.1 Card Balancing Problem

```
cin >> n;
for (int i = 1; i <= n; i++)
    cin >> a[i], avg += a[i];
avg /= n;
for (int i = 1; i <= n; i++)
    if (a[i] != avg)
        a[i + 1] += a[i] - avg, ans++;
cout << ans;
```

6.3.2 2D Card Balancing Problem

```
const int N = 100010;
int row[N], col[N], c[N], s[N];
LL work(int n, int a[])
{
    for (int i = 1; i <= n; i++)
        s[i] = s[i - 1] + a[i];
    if (s[n] % n) return -1;
    int avg = s[n] / n;
    c[1] = 0;
    for (int i = 2; i <= n; i++)
        c[i] = s[i - 1] - (i - 1) * avg;
    sort(c + 1, c + n + 1);
    LL res = 0;
    for (int i = 1; i <= n; i++)
        res += abs(c[i] - c[(n + 1) / 2]);
    return res;
}
int main()
{
    int n, m, cnt;
    cin >> n >> m >> cnt;
    while (cnt--)
    {
        int x, y;
        cin >> x >> y;
        row[x]++; col[y]++;
    }
    LL r = work(n, row);
    LL c = work(m, col);
    if (r != -1 && c != -1)
        cout << "both " << r + c;
    else if (r != -1)
        cout << "row " << r;
    else if (c != -1)
        cout << "column " << c;
    else cout << "impossible";
}
```

6.3.3 Dual Heaps

```
if (down.empty() || x <= down.top())
          down.push(x);
else up.push(x);
if (down.size() > up.size() + 1)
    up.push(down.top()), down.pop();
```

```cpp
if (up.size() > down.size())
    down.push(up.top()), up.pop();
if (i % 2)
{
    cout << down.top() << ' ';
    if (++cnt % 10 == 0) cout << '\n';
}
```

6.4 RMQ

```cpp
const int N = 200010, M = 18;
int n, m, w[N], f[N][M];
void init()
{
    for (int j = 0; j < M; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            if (!j) f[i][j] = w[i];
            else    // 也可以是最小值
                f[i][j] = max(f[i][j - 1], f[i + (1 << j - 1)][j - 1]);
}
int query(int l, int r)
{
    int len = r - l + 1;
    int k = log(len) / log(2);
    return max(f[l][k], f[r - (1 << k) + 1][k]);
}
```

7 ⋆ Advanced Data Structures

7.1 Binary Indexed Tree

```
// 支持区间修改、区间查询
// 利用变差分求二阶区间和
const int N = 100010;
int n, m, a[N];
LL tr1[N], tr2[N];
int lowbit(int x) { return x & -x; }
void add(LL tr[], LL x, LL c)
{
    for (int i = x; i <= n; i += lowbit(i))
        tr[i] += c;
}
LL sum(LL tr[], LL x)
{
    LL res = 0;
    for (int i = x; i; i -= lowbit(i))
        res += tr[i];
    return res;
}
LL prefix_sum(LL x)
{ return sum(tr1, x) * (x + 1) - sum(tr2, x); }
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i <= n; i++)
    {
        int b = a[i] - a[i - 1];
        add(tr1, i, b);
        add(tr2, i, (LL)i * b);
    }
    while (m--)
    {
        char op[2];
        int l, r, d;
        cin >> op >> l >> r;
        if (*op == 'Q')
            cout << prefix_sum(r) - prefix_sum(l - 1) << '\n';
        else
        {
            cin >> d;
            add(tr1, l, d), add(tr2, l, (LL)l * d),
            add(tr1, r + 1, -d),
            add(tr2, r + 1, (LL)-(r + 1) * d);
        }
    }
}
```

7.2 Segment Tree

```
struct Node
{   // 可以维护任何满足区间加法的信息
    int l, r; LL sum, add; // 区间和/懒标记
} tr[N * 4];
void pushup(int u)  // 从上至下传递
{ tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum; }
void pushdown(int u)
{   // 从下至上传递
    auto &root = tr[u],
         &left = tr[u << 1],
         &right = tr[u << 1 | 1];
    if (root.add)
    {
        left.add += root.add,
            left.sum += (LL)(left.r - left.l + 1) * root.add;
        right.add += root.add,
            right.sum += (LL)(right.r - right.l + 1) * root.add;
        root.add = 0;
    }
}
void build(int u, int l, int r)
{   // 建树
    if (l == r) tr[u] = {l, r, w[r], 0};
    else
    {
        tr[u] = {l, r};
        int mid = l + r >> 1;
        build(u << 1, l, mid); // 左儿子
        build(u << 1 | 1, mid + 1, r); // 右儿子
        pushup(u); // 从下往上传递区间值
    }
}
void modify(int u, int l, int r, int d)
{   // 区间修改
    if (tr[u].l >= l && tr[u].r <= r)
    {
        tr[u].sum += (LL)(tr[u].r - tr[u].l + 1) * d;
        tr[u].add += d;
    }
    else
    {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid)
            modify(u << 1, l, r, d);
        if (r > mid)
            modify(u << 1 | 1, l, r, d);
        pushup(u);
    }
}
LL query(int u, int l, int r)
{   // 区间查询
    if (tr[u].l >= l && tr[u].r <= r)
        return tr[u].sum;
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    LL sum = 0;
    if (l <= mid)
        sum += query(u << 1, l, r);
    if (r > mid)
        sum += query(u << 1 | 1, l, r);
    return sum;
}
```

8 $\star$ Advanced Search

9 ⋆ Advanced Graph Theory

10 $\star$ Advanced Math

11 $\star$ Advanced DP