



**国家超级计算广州中心**  
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

---

# 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 16<sup>th</sup> May, 2025*

# Contents

<b>0</b>	<b>Preface</b>	<b>5</b>	<b>4</b>	<b>Basic Math</b>	<b>17</b>
0.1	Template	5	4.1	Prime Numbers	17
0.2	Operator Precedence	5	4.1.1	Judging Prime Numbers	17
0.3	Time Complexity	5	4.1.2	Prime Factorization	17
0.4	If <bits/stdc++.h> Failed	6	4.1.3	Euler's Sieve	17
<b>1</b>	<b>Basic Algorithm</b>	<b>7</b>	4.2	Divisor	17
1.1	Quick Sort	7	4.2.1	Find All Divisors	17
1.2	Binary Search	7	4.2.2	The Number of Divisors	17
1.3	High Precision	7	4.2.3	The Sum of Divisors	17
1.3.1	High Precision Add	7	4.2.4	Euclidean Algorithm	18
1.3.2	High Precision Subsection	7	4.3	Euler Function	18
1.3.3	High Precision Multiply	8	4.3.1	Simple Method	18
1.3.4	High Precision Divide	8	4.3.2	Euler's Sieve Method	18
1.4	Prefix Sum & Difference Array	8	4.4	Exponentiating by Squaring	18
1.4.1	1D Prefix Sum	8	4.5	Extended Euclidean Algorithm	18
1.4.2	2D Prefix Sum	8	4.6	Chinese Remainder Theorem	18
1.4.3	1D Difference Array	8	4.7	Gauss-Jordan Elimination	19
1.4.4	2D Difference Array	9	4.7.1	Linear Equation Group	19
<b>2</b>	<b>Basic Data Structures</b>	<b>10</b>	4.7.2	XOR Linear Equation Group	19
2.1	Linked List	10	4.8	Combinatorial Counting	19
2.1.1	Singly Linked List	10	4.8.1	Recurrence Relation	19
2.1.2	Bidirectional Linked List	10	4.8.2	Preprocessing & Inverse Element	20
2.2	Stack & Queue	10	4.8.3	Lucas Theorem	20
2.2.1	Monotonic Stack	10	4.8.4	Factorization Method	20
2.2.2	Monotonic Queue	10	4.8.5	Catalan Number	20
2.3	KMP	10	4.9	Inclusion-Exclusion Principle	21
2.4	Trie	10	4.10	Game Theory	21
2.5	Disjoint-Set	11	4.10.1	NIM Game	21
2.6	Hash	11	<b>5</b>	<b>Basic DP</b>	<b>22</b>
2.6.1	Simple Hash	11	5.1	Knapsack Problem	22
2.6.2	String Hash	11	5.1.1	01 Knapsack	22
2.7	STL	11	5.1.2	Complete Knapsack	22
<b>3</b>	<b>Search &amp; Graph Theory</b>	<b>13</b>	5.1.3	Mutiple Knapsack	22
3.1	Representation of Tree & Graph	13	5.1.4	Grouped Knapsack	22
3.1.1	Adjacency Matrix	13	5.2	Linear DP	22
3.1.2	Adjacency List	13	5.2.1	LIS	22
3.2	DFS & BFS	13	5.2.2	LCS	23
3.2.1	DFS	13	5.3	Interval DP	23
3.2.2	BFS	13	5.4	Counting DP	23
3.3	Topological Sort	13	5.5	Digit DP	23
3.4	Shortest Path	13	5.6	State Compression DP	24
3.4.1	Dijkstra	13	5.7	Tree DP	24
3.4.2	Bellman-Ford	13	5.8	Memoized Search	25
3.4.3	SPFA	14	<b>6</b>	<b>Advanced Basic</b>	<b>27</b>
3.4.4	Detecting Negative Circle in SPFA	14	6.1	Slow Multiplication	27
3.4.5	Floyd	14	6.2	Sum of Geometric Series	27
3.5	Minimum Spanning Tree	14	6.3	Sort	27
3.5.1	Prim	14	6.3.1	Card Balancing Problem	27
3.5.2	Kruskal	15	6.3.2	2D Card Balancing Problem	27
3.6	Bipartite Graph	15	6.3.3	Dual Heaps	27
3.6.1	Coloring Method	15	6.4	RMQ	28
3.6.2	Hungarian Algorithm	16	<b>7</b>	<b>Advanced Data Structures</b>	<b>29</b>
			7.1	Binary Indexed Tree	29
			7.2	Segment Tree	29
			<b>8</b>	<b>Advanced Search</b>	<b>30</b>
			<b>9</b>	<b>Advanced Graph Theory</b>	<b>31</b>

10 Advanced Math	32
11 Advanced DP	33



---

# 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

```
1 #define itn int
2 #define nit int
3 #define nti int
4 #define tin int
5 #define tni int
6 #define retrun return
7 #define reutrn return
8 #define rutren return
9 #define fastin \
10     ios_base::sync_with_stdio(0); \
11     cin.tie(0), cout.tie(0);
12 #include <bits/stdc++.h>
13 using namespace std;
14 typedef long long LL;
15 typedef long double LD;
16 typedef pair<int, int> PII;
17 typedef pair<long long, long long> PLL;
18 typedef pair<double, double> PDD;
19 typedef vector<int> VI;
20 #ifndef ONLINE_JUDGE
21 #define dbg(args...) \
22     do \
23     { \
24         cout << "\033[32;1m" << #args << " \
25         -> "; \
26         err(args); \
27     } while (0)
28 #define dbg(...)
29 #endif
30 void err()
31 { cout << "\033[39;0m" << endl; }
32 template <template <typename...> class T,
33         typename t, typename... Args>
34 void err(T<t> a, Args... args)
35 {
36     for (auto x : a) cout << x << ' ';
37     err(args...);
38 }
39 template <typename T, typename... Args>
40 void err(T a, Args... args)
41 { cout << a << ' '; err(args...); }
42 const int INF = 0x3f3f3f3f;
43 const int mod = 1e9 + 7;
44 const double eps = 1e-6;
45 int main()
46 {
47     #ifndef ONLINE_JUDGE
48         freopen("test.in", "r", stdin);
49         freopen("test.out", "w", stdout);
50     #endif
51     fastin;
52     return 0;
53 }
```

### 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  $10^7 \sim 10^8$  operations.
- Below is a guide on how to choose algorithms based on different input size ranges:
  1.  $n \leq 30 \rightarrow$  Exponential complexity: DFS with pruning, State Compression DP
  2.  $n \leq 100 \rightarrow O(n^3)$ : Floyd, DP, Gaussian Elimination
  3.  $n \leq 1000 \rightarrow O(n^2)$ ,  $O(n^2 \log n)$ : DP, Binary Search, Naive Dijkstra, Naive Prim, Bellman-Ford
  4.  $n \leq 10000 \rightarrow O(n^3)$ : Block Linked List, Mo's Algorithm
  5.  $n \leq 100000 \rightarrow O(n \log n)$ : 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.  $n \leq 1000000 \rightarrow O(n)$ , or small-constant  $O(n \log n)$ : Monotonic Queue, Hashing, Two Pointers, BFS, Union Find, KMP, Aho-Corasick Automaton
  7.  $n \leq 10000000 \rightarrow O(n)$ : Two Pointers, KMP, Aho-Corasick Automaton, Linear Sieve for Primes
  8.  $n \leq 10^9 \rightarrow O(\sqrt{n})$ : Primality Testing
  9.  $n \leq 10^{18} \rightarrow O(\log n)$ : GCD, Fast Exponentiation, Digit DP
  10.  $n \leq 10^{1000} \rightarrow O((\log n)^2)$ : Big Integer Arithmetic (Add/Subtract/Multiply/Divide)
  11.  $n \leq 10^{100000} \rightarrow O(\log k \cdot \log \log k)$ , where  $k$  is the number of digits: Big Integer Add/Subtract, FFT/NTT

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

Replace it with:

```
1  #include <algorithm>
2  #include <bitset>
3  #include <complex>
4  #include <deque>
5  #include <exception>
6  #include <fstream>
7  #include <functional>
8  #include <iomanip>
9  #include <ios>
10 #include <iosfwd>
11 #include <iostream>
12 #include <istream>
13 #include <iterator>
14 #include <limits>
15 #include <list>
16 #include <locale>
17 #include <map>
18 #include <memory>
19 #include <numeric>
20 #include <ostream>
21 #include <queue>
22 #include <set>
23 #include <sstream>
24 #include <stack>
25 #include <stdexcept>
26 #include <streambuf>
27 #include <string>
28 #include <typeinfo>
29 #include <utility>
30 #include <valarray>
31 #include <vector>
```

# 1 ★ Basic Algorithm

## 1.1 Quick Sort

Sort the given array from index 1 to n.

```
1 void quick_sort(int l, int r)
2 {
3     if (l >= r) return;
4     int x = a[(l + r) >> 1], i = l - 1, j
      = r + 1;
5     while (i < j)
6     {
7         do i++; while (a[i] < x);
8         do j--; while (a[j] > x);
9         if (i < j) swap(a[i], a[j]);
10    }
11    quick_sort(l, j);
12    quick_sort(j + 1, r);
13    return;
14 }
```

## 1.2 Binary Search

```
1 // 区间 [l, r] 被划分成 [l, mid] 和 [mid +
  1, r] 时使用
2 // 大于等于区间的最小值, check 应为 target
  <= a[mid]
3 int bsearch_1(int l, int r)
4 {
5     while (l < r)
6     {
7         int mid = l + r >> 1;
8         if (check(mid)) r = mid;
9         else l = mid + 1;
10    }
11    return l;
12 }
13 // 区间 [l, r] 被划分成 [l, mid - 1] 和 [
  mid, r] 时使用
14 // 小于等于区间的最大值, check 应为 target
  >= a[mid]
15 int bsearch_2(int l, int r)
16 {
17     while (l < r)
18     {
19         // 为什么要 l + r + 1: 因为 l 的更
          新条件是 mid 本身
20         // 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
21         int mid = l + r + 1 >> 1;
22         if (check(mid)) l = mid;
23         else r = mid - 1;
24    }
25    return l;
26 }
27 // 浮点数二分
28 double bsearch_3(double l, double r)
29 {
30     // eps 表示精度, 取决于题目对精度的要求
31     const double eps = 1e-6;
```

```
32 while (r - l > eps)
33 {
34     double mid = (l + r) / 2;
35     if (check(mid)) r = mid;
36     else l = mid;
37 }
38 return l;
39 }
```

## 1.3 High Precision

### 1.3.1 High Precision Add

```
1 string s1, s2;
2 vector<int> a, b, c;
3 void add(vector<int> &a, vector<int> &b)
4 {
5     if (a.size() < b.size())
6     { add(b, a); return; }
7     int t = 0;
8     for (int i = 0; i < a.size(); i++)
9     {
10         t += a[i];
11         if (i < b.size()) t += b[i];
12         c.push_back(t % 10);
13         t /= 10;
14    }
15    while (t)
16        c.push_back(t % 10), t /= 10;
17 }
18 int main()
19 {
20     cin >> s1 >> s2;
21     for (int i = s1.size() - 1; i >= 0; i
      --)
22         a.push_back(s1[i] - '0');
23     for (int i = s2.size() - 1; i >= 0; i
      --)
24         b.push_back(s2[i] - '0');
25     add(a, b);
26     for (int i = c.size() - 1; i >= 0; i
      --)
27         cout << c[i];
28     return 0;
29 }
```

### 1.3.2 High Precision Subsection

```
1 vector<int> a, b, c;
2 string s1, s2;
3 void sub(vector<int> &a, vector<int> &b)
4 {
5     int t = 0;
6     for (int i = 0; i < a.size(); i++)
7     {
8         t = a[i] - t;
9         if (i < b.size()) t -= b[i];
10        c.push_back((t + 10) % 10);
11        if (t < 0) t = 1;
12        else t = 0;
13    }
```

```

14     while (c.size() > 1 && c.back() == 0)
15         c.pop_back();
16 }
17 int main()
18 {
19     cin >> s1 >> s2;
20     for (int i = s1.size() - 1; i >= 0; i --)
21         a.push_back(s1[i] - '0');
22     for (int i = s2.size() - 1; i >= 0; i --)
23         b.push_back(s2[i] - '0');
24     if (s1.size() < s2.size())
25         cout << '-', sub(b, a);
26     else if (s1.size() == s2.size() && s1 < s2)
27         cout << '-', sub(b, a);
28     else sub(a, b);
29     for (int i = c.size() - 1; i >= 0; i --)
30         cout << c[i];
31     return 0;
32 }

```

```

7     for (int i = a.size() - 1; i >= 0; i --)
8     {
9         r = r * 10 + a[i];
10        c.push_back(r / b);
11        r %= b;
12    }
13    reverse(c.begin(), c.end());
14    while (c.size() > 1 && c.back() == 0)
15        c.pop_back();
16 }
17 int main()
18 {
19     cin >> s1 >> b;
20     for (int i = s1.size() - 1; i >= 0; i --)
21         a.push_back(s1[i] - '0');
22    divide(a, b, r);
23    for (int i = c.size() - 1; i >= 0; i --)
24        cout << c[i];
25    cout << '\n' << r;
26    return 0;
27 }

```

### 1.3.3 High Precision Multiply

```

1  string s1, s2;
2  vector<int> a, c;
3  int b;
4  void mul(vector<int> &a, int b)
5  {
6      for (int i = 0, t = 0; i < a.size() || t; i++)
7      {
8          if (i < a.size()) t += a[i] * b;
9          c.push_back(t % 10);
10         t /= 10;
11     }
12     while (c.size() > 1 && c.back() == 0)
13         c.pop_back();
14 }
15 int main()
16 {
17     cin >> s1 >> b;
18     for (int i = s1.size() - 1; i >= 0; i --)
19         a.push_back(s1[i] - '0');
20     mul(a, b);
21     for (int i = c.size() - 1; i >= 0; i --)
22         cout << c[i];
23     return 0;
24 }

```

### 1.3.4 High Precision Divide

```

1  string s1, s2;
2  vector<int> a, c;
3  int b, r;
4  void divide(vector<int> &a, int b, int &r)
5  {
6      r = 0;

```

## 1.4 Prefix Sum & Difference Array

### 1.4.1 1D Prefix Sum

```

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

```

### 1.4.2 2D Prefix Sum

```

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

```

### 1.4.3 1D Difference Array

```

1  const int N = 100010;
2  int n, m;
3  int a[N], b[N];
4  void insert(int l, int r, int c)
5  { b[l] += c; b[r + 1] -= c; }
6  int main()
7  {
8      cin >> n >> m;
9      for (int i = 1; i <= n; i++)
10         cin >> a[i];
11     for (int i = 1; i <= n; i++)
12         insert(i, i, a[i]);
13     while (m--)

```



```

14     {
15         int l, r, c;
16         cin >> l >> r >> c;
17         insert(l, r, c);
18     }
19     for (int i = 1; i <= n; i++)
20         b[i] += b[i - 1],
21         cout << b[i] << ' ';
22     return 0;
23 }

```

#### 1.4.4 2D Difference Array

```

1  const int N = 1010;
2  int n, m, q, a[N][N], b[N][N];
3  void insert(int x1, int y1, int x2, int y2
4  , int c)
5  {
6      b[x1][y1] += c;
7      b[x2 + 1][y2 + 1] += c;
8      b[x1][y2 + 1] -= c;
9      b[x2 + 1][y1] -= c;
10 }
11 int main()
12 {
13     cin >> n >> m >> q;
14     for (int i = 1; i <= n; i++)
15         for (int j = 1; j <= m; j++)
16             cin >> a[i][j];
17     for (int i = 1; i <= n; i++)
18         for (int j = 1; j <= m; j++)
19             insert(i, j, i, j, a[i][j]);
20     while (q--)
21     {
22         int x1, x2, y1, y2, c;
23         cin >> x1 >> y1 >> x2 >> y2 >> c;
24         insert(x1, y1, x2, y2, c);
25     }
26     // 其他过程略

```

## 2 ★ Basic Data Structures

### 2.1 Linked List

#### 2.1.1 Singly Linked List

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

#### 2.1.2 Bidirectional Linked List

```
1 const int N = 100010;
2 int n, r[N], l[N], e[N], idx = 2;
3 void init() { r[0] = 1; l[1] = 0; }
4 void insert(int k, int x) // 第 k 个节点后
   插入
5 {
6     e[idx] = x;
7     r[idx] = r[k];
8     l[idx] = k;
9     l[r[k]] = idx;
10    r[k] = idx++;
11 }
12 void remove(int k) // 删除 k 本身
13 { r[l[k]] = r[k]; l[r[k]] = l[k]; }
```

## 2.2 Stack & Queue

### 2.2.1 Monotonic Stack

```
1 // 常见模型：找出每个数左边离它最近的比它大/
   小的数
2 int tt = 0;
3 for (int i = 1; i <= n; i++)
4 {
5     while (tt && check(stk[tt], i)) tt --
6     ;
7     stk[++tt] = i;
8 }
```

### 2.2.2 Monotonic Queue

```
1 // 常见模型：找出滑动窗口中的最大值/最小值
2 int hh = 0, tt = -1;
3 for (int i = 0; i < n; i++)
4 {
5     while (hh <= tt && check_out(q[hh]))
6         hh++; // 判断队头是否滑出窗口
7     while (hh <= tt && check(q[tt], i))
```

```
8         tt-- ;
9     q[++tt] = i;
10 }
```

### 2.3 KMP

```
1 const int N = 100010, M = 1000010;
2 int n, m;
3 char p[N], s[M];
4 void getNext(int ne[])
5 {
6     for (int i = 2, j = 0; i <= n; i++)
7     {
8         while (j && p[j + 1] != p[i])
9             j = ne[j];
10        if (p[j + 1] == p[i]) j++;
11        ne[i] = j;
12    }
13 }
14 int KMP()
15 {
16     int *ne = new int[n + 1];
17     getNext(ne);
18     for (int i = 1, j = 0; i <= m; i++)
19     {
20         while (j && p[j + 1] != s[i])
21             j = ne[j];
22         if (p[j + 1] == s[i]) j++;
23         if (j == n) cout << i - n << ' ';
24     }
25     return -1;
26 }
```

### 2.4 Trie

```
1 const int N = 100010;
2 int trie[N][26], cnt[N], idx = 0;
3 void insert(string &str) // 插入到 Trie
   数组
4 {
5     int p = 0;
6     for (auto c : str)
7     {
8         int u = c - 'a';
9         if (!trie[p][u])
10             trie[p][u] = ++idx;
11         p = trie[p][u];
12     }
13     cnt[p]++;
14 }
15 int query(string &str) // 查询字符串出
   现的次数
16 {
17     int p = 0;
18     for (auto c : str)
19     {
20         int u = c - 'a';
21         if (!trie[p][u]) return 0;
22         p = trie[p][u];
23     }
```

```

24     return cnt[p];
25 }

```

## 2.5 Disjoint-Set

```

1  const int N = 100010;
2  int n, m, p[N], Size[N], D[N];
3  void init()
4  {
5      for (int i = 1; i <= n; i++)
6          p[i] = i, Size[i] = 1, D[i] = 0;
7  }
8  int find(int x)
9  {
10     if (p[x] != x)
11     {
12         int u = find(p[x]);
13         D[x] += D[p[x]]; // 视具体情况计算
14         p[x] = u;
15     }
16     return p[x];
17 }
18 void merge(int a, int b, int distance)
19 {
20     int x = find(a), y = find(b);
21     if (x != y)
22     {
23         p[x] = y;
24         D[x] = distance; // 视具体情况计算
25         Size[y] += Size[x];
26     }
27 }

```

## 2.6 Hash

### 2.6.1 Simple Hash

```

1  // (1) 拉链法
2  int h[N], e[N], ne[N], idx;
3  void insert(int x)
4  {
5      int k = (x % N + N) % N;
6      e[idx] = x, ne[idx] = h[k], h[k] = idx;
7      ++idx;
8  }
9  bool find(int x)
10 {
11     for (int i = h[(x % N + N) % N]; i != -1; i = ne[i])
12         if (e[i] == x) return true;
13     return false;
14 }
15 // (2) 开放寻址法
16 int find(int x)
17 {
18     int t = (x % N + N) % N;
19     while (h[t] != null && h[t] != x)
20         t++; if (t == N) t = 0; }
21 return t;
22 }

```

### 2.6.2 String Hash

```

1  typedef unsigned long long ULL;
2  ULL h[N], p[N];
3  void init()
4  {
5      p[0] = 1;
6      for (int i = 1; i <= n; i++) { h[i]
7          = h[i - 1] * P + str[i]; p[i] = p[i -
8          1] * P; }
9  }
10 ULL get(int l, int r) { return h[r] - h[l
11     - 1] * p[r - l + 1]; }

```

## 2.7 STL

```

1  // vector
2  size()      返回元素个数
3  empty()     返回是否为空
4  clear()     清空
5  front()/back()
6  push_back()/pop_back()
7  begin()/end()
8  []
9  支持比较运算, 按字典序
10 // pair<int, int>
11 first       第一个元素
12 second      第二个元素
13 支持比较运算, 以first为第一关键字, 以second
14 为第二关键字 (字典序)
15 // string
16 size()/length() 返回字符串长度
17 empty()
18 clear()
19 substr(起始下标, (子串长度)) 返回子串
20 c_str() 返回字符串所在字符数组的起始地址
21 // queue
22 size()
23 empty()
24 push()      向队尾插入一个元素
25 front()     返回队头元素
26 back()      返回队尾元素
27 pop()       弹出队头元素
28 // priority_queue
29 size()
30 empty()
31 push()      插入一个元素
32 top()       返回堆顶元素
33 pop()       弹出堆顶元素
34 定义成小根堆的方式: priority_queue<int,
35     vector<int>, greater<int>> q;
36 // stack
37 size()
38 empty()
39 push()      向栈顶插入一个元素
40 top()       返回栈顶元素
41 pop()       弹出栈顶元素
42 // deque
43 size()
44 empty()
45 clear()
46 front()/back()
47 push_back()/pop_back()

```

```

46 push_front()/pop_front()
47 begin()/end()
48 []
49 // set, map, multiset, multimap: 基于平衡二
    叉树 (红黑树) 动态维护有序序列
50 size()
51 empty()
52 clear()
53 begin()/end()
54 ++, -- 返回前驱和后继, 时间复杂度  $O(\log n)$ 
55 // set/multiset
56     insert() 插入一个数
57     find()   查找一个数
58     count()  返回某一个数的个数
59     erase()
60         (1) 输入是一个数x, 删除所有x,  $O(k + \log n)$ 
61         (2) 输入一个迭代器, 删除这个迭代器
62     lower_bound()/upper_bound()
63         lower_bound(x) 返回大于等于x的最小的
        数的迭代器
64         upper_bound(x) 返回大于x的最小的数
        的迭代器
65 // map/multimap
66     insert() 插入的数是一个pair
67     erase()   输入的参数是pair或者迭代器
68     find()
69     []        注意multimap不支持此操作。时
        间复杂度是  $O(\log n)$ 
70     lower_bound()/upper_bound()
71 // unordered_set, unordered_map,
    unordered_multiset, unordered_multimap
72 增删改查的时间复杂度是  $O(1)$ 
73 不支持 lower_bound()/upper_bound(), 迭代器
    的++, --
74 // bitset
75 bitset<10000> s;
76 ~, &, |, ^
77 >>, <<
78 ==, !=
79 []
80 count()    返回有多少个1
81 any()      判断是否至少有一个1
82 none()     判断是否全为0
83 set()      把所有位置成1
84 set(k, v)  将第k位变成v
85 reset()    把所有位变成0
86 flip()     等价于~
87 flip(k)    把第k位取反

```

## 3 ★ Search & Graph Theory

### 3.1 Representation of Tree & Graph

#### 3.1.1 Adjacency Matrix

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

#### 3.1.2 Adjacency List

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

## 3.2 DFS & BFS

#### 3.2.1 DFS

```
1 int dfs(int u)
2 {
3     st[u] = true; // 表示点 u 已经被遍历过
4     for (int i = h[u]; i != -1; i = ne[i])
5     { int j = e[i]; if (!st[j]) dfs(j); }
6 }
```

#### 3.2.2 BFS

```
1 queue<int> q;
2 st[1] = true; q.push(1);
3 while (q.size())
4 {
5     int t = q.front(); q.pop();
6     for (int i = h[t]; i != -1; i = ne[i])
7     { if (!st[e[i]]) { st[e[i]] = true;
8         q.push(e[i]); }
9 }
```

## 3.3 Topological Sort

```
1 const int N = 100010;
2 int e[2 * N], ne[2 * N], h[N], d[N], idx;
3 int n, m, q[N];
4 void init() { memset(h, -1, sizeof h); }
5 void add(int a, int b) { e[idx] = b, ne[
    idx] = h[a], h[a] = idx++, d[b]++; }
6 bool topSort()
7 {
8     int hh = 0, tt = -1;
9     for (int i = 1; i <= n; i++)
10         if (!d[i]) q[++tt] = i;
11     while (hh <= tt)
```

```
12         for (int i = h[q[hh++]]; ~i; i =
13             ne[i])
14             if (--d[e[i]] == 0) q[++tt] =
15                 e[i];
16     return tt == n - 1;
17 }
```

## 3.4 Shortest Path

#### 3.4.1 Dijkstra

```
1 const int N = 1010;
2 int n, dist[N];
3 int h[N], w[N], e[N], ne[N], idx;
4 bool st[N];
5 void add(int a, int b, int c) { e[idx] = b,
    w[idx] = c, ne[idx] = h[a], h[a] =
    idx++; }
6 int dijkstra() // 需要初始化 dist 与 h
7 {
8     dist[1] = 0;
9     priority_queue<PII, vector<PII>,
10         greater<PII>> heap;
11     heap.push({0, 1});
12     while (heap.size())
13     {
14         auto t = heap.top();
15         heap.pop();
16         int ver = t.second, distance = t.
17             first;
18         if (st[ver]) continue;
19         st[ver] = true;
20         for (int i = h[ver]; i != -1; i =
21             ne[i])
22             if (dist[e[i]] > distance + w[
23                 i])
24             {
25                 dist[e[i]] = distance + w[
26                     i];
27                 heap.push({dist[e[i]], e[i]
28                     });
29             }
30     }
31     if (dist[n] == 0x3f3f3f3f) return -1;
32     return dist[n];
33 }
```

#### 3.4.2 Bellman-Ford

```
1 const int N = 100010;
2 int n, m, dist[N], backup[N];
3 struct Edge
4 {
5     int a, b, w;
6 } edges[N];
7 int bellman_ford()
8 {
9     memset(dist, 0x3f, sizeof dist);
10    dist[1] = 0;
11    for (int i = 0; i < n; i++)
12    {
```

```

13     memcpy(backup, dist, sizeof dist);
14     for (int j = 0; j < m; j++)
15     {
16         int a = edges[j].a, b = edges[
17         j].b, w = edges[j].w;
18         dist[b] = min(dist[b], backup[
19         a] + w);
20     }
21     if (dist[n] > 0x3f3f3f3f / 2) return
22     -1;
23     return dist[n];
24 }

```

```

13     {
14         dist[e[i]] = dist[t] + w[i
15     ];
16         // 新增
17         cnt[j] = cnt[t] + 1;
18         if (cnt[j] >= n) return
19         true
20         if (!st[j]) q.push(j), st[
21         j] = true;
22     }
23     }
24     return false;
25 }

```

### 3.4.3 SPFA

```

1  const int N = 100010;
2  int n, m, dist[N];
3  int e[2 * N], ne[2 * N], w[2 * N], h[N],
4  idx;
5  bool vis[N];
6  void spfa()    // 需要初始化 dist 与 h
7  {
8      queue<int> q;
9      q.push(1); vis[1] = true;
10     while (q.size())
11     {
12         int t = q.front();
13         q.pop();
14         vis[t] = false;
15         for (int i = h[t]; ~i; i = ne[i])
16             if (dist[e[i]] > dist[t] + w[i
17             ])
18             {
19                 dist[e[i]] = dist[t] + w[i
20                 ];
21                 if (!vis[e[i]]) vis[e[i]]
22                 = true, q.push(j);
23             }
24     }
25     dist[n] > INF / 2 ? cout << "
26     impossible" : cout << dist[n];
27 }

```

### 3.4.5 Floyd

```

1  const int N = 210;
2  int g[N][N], n, m, k;
3  int main()
4  {
5      cin >> n >> m >> k;
6      memset(g, 0x3f, sizeof g);
7      for (int i = 1; i <= n; i++) g[i][i] =
8      0;
9      while (m--)
10     {
11         int a, b, c;
12         cin >> a >> b >> c;
13         g[a][b] = min(g[a][b], c);
14     }
15     for (int k = 1; k <= n; k++)
16         for (int i = 1; i <= n; i++)
17             for (int j = 1; j <= n; j++)
18                 g[i][j] = min(g[i][k] + g[
19                 k][j], g[i][j]);
20     // 后续代码略
21     return 0;
22 }

```

## 3.5 Minimum Spanning Tree

### 3.5.1 Prim

### 3.4.4 Detecting Negative Circle in SPFA

```

1  void spfa()    // 只需要初始化 h
2  {
3      queue<int> q;
4      // 基于虚拟原点假设, 所有点放入队列
5      for (int i = 1; i <= n; i++) q.push(i)
6      , st[i] = true;
7      while (q.size())
8      {
9          int t = q.front();
10         q.pop();
11         vis[t] = false;
12         for (int i = h[t]; ~i; i = ne[i])
13             if (dist[e[i]] > dist[t] + w[i
14             ])
15             {
16                 dist[e[i]] = dist[t] + w[i
17                 ];
18                 if (!vis[e[i]]) vis[e[i]]
19                 = true, q.push(j);
20             }
21     }
22 }

```

```

1  const int N = 510;
2  int n, m, g[N][N], dist[N];
3  bool vis[N];
4  void prim()
5  {
6      int res = 0;
7      for (int i = 0; i < n; i++)
8      {
9          int t = -1;
10         for (int j = 1; j <= n; j++)
11             if (!vis[j] && (t == -1 ||
12             dist[j] < dist[t])) t = j;
13         if (i && dist[t] == INF) { res =
14         INF; break; }
15         if (i) res += dist[t];
16         vis[t] = true;
17         for (int j = 1; j <= n; j++) dist[
18         j] = min(dist[j], g[t][j]);
19     }
20 }

```

```

16     }
17     res == INF ? cout << "impossible" :
        cout << res;
18 }
19 int main()
20 {
21     memset(g, 0x3f, sizeof g);
22     memset(dist, 0x3f, sizeof dist);
23     cin >> n >> m;
24     while (m--)
25     {
26         int a, b, c;
27         cin >> a >> b >> c;
28         g[a][b] = min(g[a][b], c);
29         g[b][a] = min(g[b][a], c);
30     }
31     prim();
32     return 0;
33 }

```

### 3.5.2 Kruskal

```

1  const int N = 100010;
2  int n, m;
3  int p[N];
4  struct Edge
5  {
6      int a, b, w;
7      bool operator<(const Edge &e) const {
9          return w < e.w; };
8  } edge[2 * N];
9  void init() { for (int i = 1; i <= n; i++)
10     p[i] = i; }
10 int find(int x)
11 {
12     if (x != p[x]) p[x] = find(p[x]);
13     return p[x];
14 }
15 void merge(int x, int y) { p[find(x)] =
16     find(y); }
16 void kruskal()
17 {
18     int res = 0, cnt = 0;
19     for (int i = 1; i <= m; i++)
20         if (find(edge[i].a) != find(edge[i]
21             .b))
22             {
23                 merge(edge[i].a, edge[i].b);
24                 res += edge[i].w;
25                 cnt++;
26             }
27     if (cnt < n - 1) res = INF;
28     res == INF ? cout << "impossible" :
29         cout << res;
30 }
31 int main()
32 {
33     init();
34     cin >> n >> m;
35     for (int i = 1; i <= m; i++) cin >>
36         edge[i].a >> edge[i].b >> edge[i].w;
37     sort(edge + 1, edge + m + 1);
38     kruskal();
39     return 0;

```

```

37 }

```

## 3.6 Bipartite Graph

### 3.6.1 Coloring Method

To check if a given graph is bipartite.

```

1  const int N = 100010, M = 200010;
2  int n, m;
3  int e[M], ne[M], h[N], color[N], idx;
4  bool dfs(int u, int c)
5  {
6      color[u] = c;
7      for (int i = h[u]; ~i; i = ne[i])
8          if (color[e[i]] == -1)
9              {
10                 if (!dfs(e[i], !c)) return false;
11             }
12     else if (color[e[i]] == c) return
13         false;
14 return true;
15 }
16 bool check()
17 {
18     for (int i = 1; i <= n; i++)
19         if (color[i] == -1)
20             if (!dfs(i, 0)) return false;
21 return true;
22 }
23 int main()
24 {
25     // 注意另外初始化 h 与 color
26     cin >> n >> m;
27     while (m--)
28     {
29         int a, b;
30         cin >> a >> b;
31         add(a, b), add(b, a);
32     }
33     // 其余过程略

```

### 3.6.2 Hungarian Algorithm

To find the maximum matching for a given graph.

```
1  const int N = 510, M = 100010;
2  int n1, n2, m;
3  int e[M], ne[M], h[N], match[N], idx;
4  bool vis[N];
5  bool find(int x)
6  {
7      for (int i = h[x]; ~i; i = ne[i])
8          if (!vis[e[i]])
9              {
10                 vis[e[i]] = true;
11                 if (match[e[i]] == 0 || find(match[e[i]]))
12                     {
13                         match[e[i]] = x;
14                         return true;
15                     }
16             }
17     return false;
18 }
19 int main()
20 {
21     // 注意初始化 h
22     cin >> n1 >> n2 >> m;
23     while (m--)
24     {
25         int a, b;
26         cin >> a >> b;
27         add(a, b);
28     }
29     int res = 0;
30     for (int i = 1; i <= n1; i++)
31     {
32         memset(vis, false, sizeof vis);
33         if (find(i)) res++;
34     }
35     cout << res;
36     return 0;
37 }
```



## 4 ★ Basic Math

### 4.1 Prime Numbers

#### 4.1.1 Judging Prime Numbers

$O(\sqrt{n})$

```
1 bool is_prime(int x)
2 {
3     if (x < 2) return false;
4     for (int i = 2; i <= x / i; i++)
5         if (x % i == 0) return false;
6     return true;
7 }
```

#### 4.1.2 Prime Factorization

```
1 void divide(int x)
2 {
3     for (int i = 2; i <= x / i; i++)
4         if (x % i == 0)
5             { // 此条件成立时 i 一定是质数
6                 int s = 0;
7                 while (x % i == 0) x /= i, s
8                     ++;
9                 cout << i << ' ' << s << '\n';
10            }
11     if (x > 1) cout << x << ' ' << 1 << '\n';
12 }
```

#### 4.1.3 Euler's Sieve

```
1 int primes[N], cnt;
2 bool st[N];
3 void get_primes(int n)
4 {
5     for (int i = 2; i <= n; i++)
6     {
7         if (!st[i]) primes[cnt++] = i;
8         for (int j = 0; primes[j] <= n / i; j++)
9             {
10                 st[primes[j] * i] = true;
11                 if (i % primes[j] == 0) break;
12             }
13     }
14 }
```

## 4.2 Divisor

### 4.2.1 Find All Divisors

```
1 vector<int> get_divisors(int x)
2 {
3     vector<int> res;
```

```
4     for (int i = 1; i <= x / i; i++)
5         if (x % i == 0)
6             {
7                 res.push_back(i);
8                 if (i != x / i) res.push_back(
9                     x / i);
10            }
11     sort(res.begin(), res.end());
12     return res;
```

### 4.2.2 The Number of Divisors

```
1 const int mod = 1e9 + 7;
2 int n;
3 int main()
4 {
5     cin >> n;
6     unordered_map<int, int> h;
7     while (n--)
8     {
9         int x;
10        cin >> x;
11        for (int i = 2; i <= x / i; i++)
12            while (x % i == 0) { h[i]++; x
13                = x / i; }
14        if (x > 1) h[x]++;
15    }
16    long long res = 1;
17    for (auto iter = h.begin(); iter != h.
18        end(); iter++)
19        res = res * (iter->second + 1) %
20        mod;
21    cout << res;
22    return 0;
23 }
```

### 4.2.3 The Sum of Divisors

```
1 const int mod = 1e9 + 7;
2 int n;
3 long long getSum(int x, int c)
4 {
5     long long s = 1;
6     while(c--) s = (s * x + 1) % mod;
7     return s;
8 }
9 int main()
10 {
11     cin >> n;
12     unordered_map<int, int> h;
13     while (n--)
14     {
15         int x;
16         cin >> x;
17         for (int i = 2; i <= x / i; i++)
18             while (x % i == 0) { h[i]++; x
19                 = x / i; }
20         if (x > 1) h[x]++;
21     }
22     long long res = 1;
```

```

22     for (auto iter = h.begin(); iter != h.
        end(); iter++)
23         res = res * getSum(iter->first,
            iter->second) % mod;
24     cout << res;
25     return 0;
26 }

```

#### 4.2.4 Euclidean Algorithm

```

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

```

### 4.3 Euler Function

#### 4.3.1 Simple Method

```

1  int phi(int x)
2  {
3      int res = x;
4      for (int i = 2; i <= x / i; i++)
5          if (x % i == 0)
6              {
7                  res = res / i * (i - 1);
8                  while (x % i == 0) x /= i;
9              }
10     if (x > 1) res = res / x * (x - 1);
11     return res;
12 }

```

#### 4.3.2 Euler's Sieve Method

```

1  const int N = 1000010;
2  int n, primes[N], phi[N], cnt;
3  bool st[N];
4  void getEuler()
5  {
6      phi[1] = 1;
7      for (int i = 2; i <= n; i++)
8          {
9              if (!st[i])
10                 {
11                     primes[cnt++] = i;
12                     // i 是质数, 它只会被本身整除,
13                     // 所以直接赋值 i - 1
14                     phi[i] = i - 1;
15                     for (int j = 0; primes[j] <= n / i; j++)
16                         {
17                             st[i * primes[j]] = true;
18                             if (i % primes[j] == 0)
19                                 {
20                                     // 如果 i % primes[j] == 0
21                                     // 成立表示 primes[j] 是 i 的最小质因子
22                                     // 也是 primes[j] * i 的最
23                                     // 小质因子

```

```

22                                     // 1 - 1 / primes[j] 这一
23                                     // 项在 phi[i] 中计算过了, 只需将基数 N 修
24                                     // 正为 primes[j] 倍
25                                     phi[primes[j] * i] = phi[i]
26                                     * primes[j];
27                                     break;
28                                     }
29                                     // 否则, primes[j] 不是 i 的质
30                                     // 因子, 只是 primes[j] * i 的最小质因子
31                                     // 不仅需要将基数 N 修正为
32                                     // primes[j] 倍
33                                     // 还需要补上 1 - 1 / primes[j]
34                                     // 的分子项, 因此最终结果为 phi[i] * (
35                                     // primes[j] - 1)
36                                     phi[primes[j] * i] = phi[i] *
37                                     (primes[j] - 1);
38                                     }
39     }
40 }

```

### 4.4 Exponentiating by Squaring

```

1  LL qmi(int m, int k, int p)
2  {
3      LL res = 1 % p, t = m;
4      while (k)
5          {
6              if (k & 1) res = res * t % p;
7              t = t * t % p;
8              k >>= 1;
9          }
10     return res;
11 }

```

### 4.5 Extended Euclidean Algorithm

```

1  int exgcd(int a, int b, int &x, int &y)
2  {
3      if (!b)
4          {
5              x = 1;
6              y = 0;
7              return a;
8          }
9      int d = exgcd(b, a % b, y, x);
10     y -= (a / b) * x;
11     return d;
12 }

```

### 4.6 Chinese Remainder Theorem

```

1  LL exgcd(LL a, LL b, LL &x, LL &y)
2  {
3      if (!b) { x = 1, y = 0; return a; }

```

```

4    LL d = exgcd(b, a % b, y, x);
5    y -= a / b * x;
6    return d;
7 }
8 int main()
9 {
10     int n;
11     cin >> n;
12     LL x = 0, m1, a1;
13     cin >> m1 >> a1;
14     for (int i = 0; i < n - 1; i++)
15     {
16         LL m2, a2;
17         cin >> m2 >> a2;
18         LL k1, k2;
19         LL d = exgcd(m1, m2, k1, k2);
20         if ((a2 - a1) % d) { x = -1; break
; }
21         k1 *= (a2 - a1) / d;
22         k1 = (k1 % (m2 / d) + m2 / d) % (
m2 / d);
23         x = k1 * m1 + a1;
24         LL m = abs(m1 / d * m2);
25         a1 = k1 * m1 + a1;
26         m1 = m;
27     }
28     if (x != -1)
29         x = (a1 % m1 + m1) % m1;
30     cout << x << '\n';
31     return 0;
32 }

```

## 4.7 Gauss-Jordan Elimination

### 4.7.1 Linear Equation Group

```

1 int gauss()
2 {
3     int c, r;
4     for (c = 0, r = 0; c < n; c++)
5     {
6         int t = r;
7         for (int i = r; i < n; i++) //
找绝对值最大的行
8             if (fabs(a[i][c]) > fabs(a[t][
c]))
9                 t = i;
10        if (fabs(a[t][c]) < eps) //
此时没必要对该列该行处理
11            continue;
12        for (int i = c; i <= n; i++)
13            swap(a[t][i], a[r][i]); //
将绝对值最大的行换到最顶端
14        for (int i = n; i >= c; i--)
15            a[r][i] /= a[r][c]; //
将当前行的首位变成1
16        for (int i = r + 1; i < n; i++) //
用当前行将下面所有的列消成0
17            if (fabs(a[i][c]) > eps)
18                for (int j = n; j >= c; j
--))
19                    a[i][j] -= a[r][j] * a
[i][c];

```

```

20        r++;
21    }
22    if (r < n)
23    {
24        for (int i = r; i < n; i++)
25            if (fabs(a[i][n]) > eps)
26                return 2; // 无解
27        return 1; // 有无穷多组解
28    }
29    for (int i = n - 1; i >= 0; i--)
30        for (int j = i + 1; j < n; j++)
31            a[i][n] -= a[i][j] * a[j][n];
32    return 0; // 有解
33 }

```

### 4.7.2 XOR Linear Equation Group

```

1 int gauss()
2 {
3     int c, r;
4     for (c = 0, r = 0; c < n; c++)
5     {
6         int t = r;
7         for (int i = r; i < n; i++)
8             if (a[i][c])
9                 t = i;
10        if (!a[t][c])
11            continue;
12        for (int i = c; i <= n; i++)
13            swap(a[r][i], a[t][i]);
14        for (int i = r + 1; i < n; i++)
15            if (a[i][c])
16                for (int j = n; j >= c; j
--))
17                    a[i][j] ^= a[r][j];
18        r++;
19    }
20    if (r < n)
21    {
22        for (int i = r; i < n; i++)
23            if (a[i][n])
24                return 2;
25        return 1;
26    }
27    for (int i = n - 1; i >= 0; i--)
28        for (int j = i + 1; j < n; j++)
29            a[i][n] ^= a[i][j] * a[j][n];
30    return 0;
31 }

```

## 4.8 Combinatorial Counting

### 4.8.1 Recurrence Relation

```

1 void init()
2 {
3     for (int i = 0; i < N; i++)
4         for (int j = 0; j <= i; j++)
5             if (!j) c[i][j] = 1;
6             else c[i][j] = (c[i - 1][j] +
c[i - 1][j - 1]) % mod;

```

```
7 }
```

## 4.8.2 Preprocessing & Inverse Element

```
1 const int N = 100010, mod = 1e9 + 7;
2 int n, fact[N], infact[N];
3 int qmi(int a, int b, int p)
4 {
5     int res = 1;
6     while (b)
7     {
8         if (b & 1)
9             res = (LL)res * a % p;
10        a = (LL)a * a % p;
11        b >>= 1;
12    }
13    return res;
14 }
15 int main()
16 {
17     fact[0] = infact[0] = 1;
18     for (int i = 1; i < N; i++)
19     {
20         fact[i] = (LL)fact[i - 1] * i %
mod;
21         infact[i] = (LL)infact[i - 1] *
qmi(i, mod - 2, mod) % mod;
22     }
23     // 此后 C(a, b) = (LL)fact[a] * infact
[b] % mod * infact[a - b] % mod
24 }
```

## 4.8.3 Lucas Theorem

```
1 int qmi(int a, int k, int p)
2 {
3     int res = 1 % p;
4     while (k)
5     {
6         if (k & 1)
7             res = (LL)res * a % p;
8         a = (LL)a * a % p;
9         k >>= 1;
10    }
11    return res;
12 }
13 int C(int a, int b, int p)
14 {
15     if (a < b) return 0;
16     LL x = 1, y = 1;
17     // x = a * (a - 1) * (a - 2) * ... * (
a - b + 1) = a! / (a - b)! (mod p)
18     // y = 1 * 2 * ... * b = b! (mod p)
19     for (int i = a, j = 1; j <= b; i--, j
++)
20     { x = (LL)x * i % p; y = (LL)y * j % p
; }
21     return x * (LL)qmi(y, p - 2, p) % p;
22 }
23 int lucas(LL a, LL b, int p)
24 {
25     if (a < p && b < p)
```

```
26         return C(a, b, p);
27     return (LL)C(a % p, b % p, p) * lucas(
a / p, b / p, p) % p;
28 }
```

## 4.8.4 Factorization Method

```
1 const int N = 5010;
2 int n, primes[N], sum[N], cnt;
3 bool st[N];
4 void getPrimes(int n) { // 略 }
5 // 求 n! 中 p 的幂次
6 int get(int n, int p)
7 {
8     int res = 0;
9     while (n) { res += n / p; n /= p; }
10    return res;
11 }
12 void mul(vector<int> &a, int b) { // 高精
度乘, 略 }
13 int main()
14 {
15     int a, b;
16     cin >> a >> b;
17     getPrimes(a);
18     for (int i = 0; i < cnt; i++)
19     {
20         int p = primes[i];
21         sum[i] = get(a, p) - get(b, p) -
get(a - b, p);
22     }
23     vector<int> res;
24     res.push_back(1);
25     for (int i = 0; i < cnt; i++)
26         for (int j = 0; j < sum[i]; j++)
27             mul(res, primes[i]);
28     for (int i = res.size() - 1; i >= 0; i
--)
29         cout << res[i];
30 }
```

## 4.8.5 Catalan Number

```
1 const int N = 100010, mod = 1e9 + 7;
2 int qmi(int a, int k, int p) { // 略 }
3 int main()
4 {
5     int n;
6     cin >> n;
7     int a = n * 2, b = n, res = 1;
8     for (int i = a; i > a - b; i--)
9         res = (LL)res * i % mod;
10    for (int i = 1; i <= b; i++)
11        res = (LL)res * qmi(i, mod - 2,
mod) % mod;
12    res = (LL)res * qmi(n + 1, mod - 2,
mod) % mod;
13 }
```

## 4.9 Inclusion-Exclusion Principle

```
1  const int N = 20;
2  int n, m, res = 0, p[N];
3  int main()
4  {
5      cin >> n >> m;
6      for (int i = 0; i < m; i++)
7          cin >> p[i];
8      // 使用二进制数字表示数字选取情况
9      for (int i = 1; i < 1 << m; i++)
10     {
11         int t = 1, cnt = 0;
12         // 遍历每个被选取的质数
13         for (int j = 0; j < m; j++)
14             if (i >> j & 1)
15             {
16                 cnt++;
17                 // 一个质数能被选取的条件应
18                 // 该是其累乘积不超过目标数字
19                 if ((LL)t * p[j] > n)
20                     { t = -1; break; }
21                 t *= p[j];
22             }
23             if (t != -1)
24                 // 容斥原理公式中奇数个并集系数
25                 // 为 1, 反之为 -1
26                 if (cnt % 2) res += n / t;
27                 else res -= n / t;
28     }
29     cout << res;
30 }
```

```
21 {
22     cin >> k;
23     for (int i = 0; i < k; i++) cin >> s[i];
24     cin >> n;
25     memset(f, -1, sizeof f);
26     int res = 0;
27     // 每一堆石子都是一个入度为 0 的起始点
28     for (int i = 0; i < n; i++)
29     {
30         int x;
31         cin >> x;
32         res ^= sg(x);
33     }
34     res ? cout << "Yes" : cout << "No";
35     return 0;
36 }
```

## 4.10 Game Theory

### 4.10.1 NIM Game

```
1  const int N = 110, M = 100010;
2  int k, n, s[N], f[M];
3  int sg(int x)
4  {
5      if (f[x] != -1) return f[x];
6      // 到达节点得 SG 函数集合
7      unordered_set<int> S;
8      // 能取走石子就说明能到达, 并且递归向下
9      // 求解
10     for (int i = 0; i < k; i++)
11     {
12         int sum = s[i];
13         if (x >= sum) S.insert(sg(x - sum));
14     }
15     // SG 从小到达遍历并返回, 找到最小的、不
16     // 包含在 SG 函数集合中的自然数
17     for (int i = 0;; i++)
18         if (!S.count(i))
19             return f[x] = i;
20 }
```

```
21 int main()
22 {
23     cin >> k;
24     for (int i = 0; i < k; i++) cin >> s[i];
25     cin >> n;
26     memset(f, -1, sizeof f);
27     int res = 0;
28     // 每一堆石子都是一个入度为 0 的起始点
29     for (int i = 0; i < n; i++)
30     {
31         int x;
32         cin >> x;
33         res ^= sg(x);
34     }
35     res ? cout << "Yes" : cout << "No";
36     return 0;
37 }
```

## 5 ★ Basic DP

### 5.1 Knapsack Problem

#### 5.1.1 01 Knapsack

```
1  const int N = 1010;
2  int n, m, v[N], w[N], f[N];
3  int main()
4  {
5      cin >> n >> m;
6      for (int i = 1; i <= n; i++)
7          cin >> v[i] >> w[i];
8      for (int i = 1; i <= n; i++)
9          for (int j = m; j >= v[i]; j--)
10             f[j] = max(f[j], f[j - v[i]] +
11                 w[i]);
12     cout << f[m];
13 }
```

#### 5.1.2 Complete Knapsack

```
1  const int N = 1010;
2  int n, m, v[N], w[N], f[N];
3  int main()
4  {
5      cin >> n >> m;
6      for (int i = 1; i <= n; i++)
7          cin >> v[i] >> w[i];
8      for (int i = 1; i <= n; i++)
9          for (int j = v[i]; j <= m; j++)
10             f[j] = max(f[j], f[j - v[i]] +
11                 w[i]);
12     cout << f[m];
13 }
```

#### 5.1.3 Mutiple Knapsack

```
1  const int N = 25000;
2  int n, m, v[N], w[N], f[N];
3  int main()
4  {
5      cin >> n >> m;
6      int cnt = 0;
7      for (int i = 1; i <= n; i++)
8      {
9          int a, b, s;
10         cin >> a >> b >> s;
11         int k = 1;
12         while (k <= s)
13         {
14             cnt++;
15             v[cnt] = a * k, w[cnt] = b * k
16         };
17         s -= k, k *= 2;
18     }
19     if (s > 0)
20     {
21         cnt++;
22         v[cnt] = a * s, w[cnt] = b * s
23     }
24     for (int i = 1; i <= cnt; i++)
25         for (int j = m; j >= v[i]; j--)
26             f[j] = max(f[j], f[j - v[i]] +
27                 w[i]);
28     cout << f[m];
29 }
```

```
21         v[cnt] = a * s, w[cnt] = b * s
22     };
23 }
24 n = cnt;
25 for (int i = 1; i <= n; i++)
26     for (int j = m; j >= v[i]; j--)
27         f[j] = max(f[j], f[j - v[i]] +
28             w[i]);
29     cout << f[m];
30 }
```

#### 5.1.4 Grouped Knapsack

```
1  const int N = 120;
2  int n, m, s[N], v[N][N], w[N][N], f[N];
3  int main()
4  {
5      cin >> n >> m;
6      for (int i = 1; i <= n; i++)
7      {
8          cin >> s[i];
9          for (int j = 1; j <= s[i]; j++)
10             cin >> v[i][j] >> w[i][j];
11     }
12     for (int i = 1; i <= n; i++)
13         for (int j = m; j >= 0; j--)
14             for (int k = 1; k <= s[i]; k++)
15                 if (v[i][k] <= j)
16                     f[j] = max(f[j], f[j -
17                         v[i][k]] + w[i][k]);
18     cout << f[m];
19 }
```

## 5.2 Linear DP

### 5.2.1 LIS

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

```
1  const int N = 1010;
2  int n, a[N], f[N];
3  int main()
4  {
5      cin >> n;
6      for (int i = 1; i <= n; i++)
7          cin >> a[i];
8      for (int i = 1; i <= n; i++)
9      {
10         f[i] = 1;
11         for (int j = 1; j < i; j++)
12             if (a[j] < a[i])
13                 f[i] = max(f[i], f[j] + 1)
14     };
15     int res = 0;
16     for (int i = 1; i <= n; i++)
17         res = max(res, f[i]);
18     cout << res;
19 }
```

Another is an  $O(n\log n)$  solution:

```
1  const int N = 100010;
2  int n, a[N], q[N];
3  int main()
4  {
5      cin >> n;
6      for (int i = 1; i <= n; i++) cin >> a[i];
7      int len = 0;
8      q[len] = -INF;
9      for (int i = 1; i <= n; i++)
10     {
11         int l = 0, r = len;
12         while (l < r)
13         {
14             int mid = l + r + 1 >> 1;
15             if (q[mid] < a[i]) l = mid;
16             else r = mid - 1;
17         }
18         len = max(r + 1, len);
19         q[r + 1] = a[i];
20     }
21     cout << len;
22 }
```

## 5.2.2 LCS

```
1  const int N = 1010;
2  int n, m, f[N][N];
3  char a[N], b[N];
4  int main()
5  {
6      cin >> n >> m >> (a + 1) >> (b + 1);
7      for (int i = 1; i <= n; i++)
8          for (int j = 1; j <= m; j++)
9              {
10                 f[i][j] = max(f[i - 1][j], f[i][j - 1]);
11                 if (a[i] == b[j])
12                     f[i][j] = max(f[i][j], f[i - 1][j - 1] + 1);
13             }
14     cout << f[n][m];
15 }
```

## 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:

```
1  const int N = 310;
2  int n, s[N], f[N][N];
3  int main()
4  {
5      cin >> n;
6      for (int i = 1; i <= n; i++)
7          cin >> s[i], s[i] += s[i - 1];
8      for (int len = 2; len <= n; len++)
9          for (int i = 1; i + len - 1 <= n; i++)
10             {
```

```
11                 int l = i, r = i + len - 1;
12                 f[l][r] = INF;
13                 for (int k = l; k < r; k++)
14                     f[l][r] = min(f[l][r], f[l][k] + f[k + 1][r] + s[r] - s[l - 1]);
15             }
16     cout << f[1][n];
17 }
```

## 5.4 Counting DP

```
1  const int N = 1010, M = 1e9 + 7;
2  int n, f[N][N];
3  int main()
4  {
5      cin >> n;
6      f[0][0] = 1;
7      for (int i = 1; i <= n; i++)
8          for (int j = 1; j <= i; j++)
9              f[i][j] = (f[i - 1][j - 1] + f[i - j][j]) % M;
10     int ans = 0;
11     for (int i = 1; i <= n; i++)
12         ans = (ans + f[n][i]) % M;
13     cout << ans;
14 }
```

## 5.5 Digit DP

```
1  // 求数 n 的位数
2  int get(int n)
3  {
4      int res = 0;
5      while (n) n /= 10, res++;
6      return res;
7  }
8  int count(int n, int i)
9  {
10     int res = 0, dgt = get(n);
11     for (int j = 1; j <= dgt; j++)
12     {
13         // 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)
14         // l 为第 j 位的左边的数, r 为右边的数, dj 为第 j 位上的数
15         int p = pow(10, dgt - j), l = n / p / 10, r = n % p, dj = n / p % 10;
16         // 求要选的数在 i 的左边的数小于 1 的情况:
17         // 1)、当 i 不为 0 时 xxx : 0...0 ~ 1 - 1, 即 1 * (右边的数的位数) == 1 * p 种选法
18         // 2)、当 i 为 0 时 由于不能有前导零 故 xxx: 0...1 ~ 1 - 1, 即 (1 - 1) * (右边的数的位数) == (1 - 1) * p 种选法
19         if (i) res += 1 * p;
20         else res += (1 - 1) * p;
21         // 求要选的数在 i 的左边的数等于 1 的情况:(即视频中的 xxx == 1 时)
```

```

22         //      1)、i > dj 时 0 种选法
23         //      2)、i == dj 时 yyy : 0...0
        ~ r 即 r + 1 种选法
24         //      3)、i < dj 时 yyy : 0...0
        ~ 9...9 即 10^(右边的数的位数) == p 种
        选法 */
25         if (i == dj) res += r + 1;
26         if (i < dj) res += p;
27     }
28     return res;
29 }
30 int main()
31 {
32     int a, b;
33     while (cin >> a >> b, a)
34     {
35         if (a > b) swap(a, b);
36         for (int i = 0; i <= 9; ++i)
37             cout << count(b, i) - count(a
        - 1, i) << ' ';
38         // 利用前缀和思想: [l, r] 的和 = s[
        r] - s[l - 1]
39         cout << '\n';
40     }
41 }

```

```

31     f[0][0] = 1;
32     // 遍历每一列
33     for (int i = 1; i <= m; i++)
34         // 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
35         for (int j = 0; j < 1 << n; j
        ++))
36             // 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
37             for (int k = 0; k < 1 << n
        ; k++)
38                 // 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
39                 if (!(j & k) && st[j |
        k])
40                     f[i][j] += f[i -
        1][k];
41                 // 输出摆放好第 m 列且第 (m + 1) 列
        没有任何方格的状态数
42                 cout << f[m][0] << '\n';
43             }
44 }

```

## 5.6 State Compression DP

```

1  const int N = 12, M = 1 << 12;
2  int n, m;
3  LL f[N][M];
4  bool st[M];
5  int main()
6  {
7      while (cin >> n >> m, n || m)
8      {
9          memset(f, 0, sizeof f);
10         for (int i = 0; i < 1 << n; i++)
11             {
12                 st[i] = true;
13                 // 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
14                 int cnt = 0;
15                 for (int j = 0; j < n && st[i
                    ]; j++)
16                     if (i >> j & 1)
17                     {
18                         // 当前格子被使用
19                         // 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
20                         if (cnt & 1)
21                             st[i] = false;
22                         // 刷新状态
23                         cnt = 0;
24                     }
25                     else cnt++;
26                 // 最后再判断一次, 防止漏判
27                 if (cnt & 1)
28                     st[i] = false;
29             }
30             // 没有摆放任何棋子的状态默认只有 1
                种取法

```

## 5.7 Tree DP

```

1  // Don't use I/O functions from stdio.h!!!
2  #define itn int
3  #define nit int
4  #define nti int
5  #define tin int
6  #define tni int
7  #define retrun return
8  #define reutrn return
9  #define rutren return
10 #define INF 0x3f3f3f3f
11 #include <bits/stdc++.h>
12 using namespace std;
13 typedef pair<int, int> PII;
14 typedef long long LL;
15
16 const int N = 6010;
17
18 int n;
19 int e[N], ne[N], happy[N], h[N], idx;
20 int f[N][2];
21 bool has_father[N];
22 void add(int a, int b)
23 { e[idx] = b, ne[idx] = h[a], h[a] = idx
    ++; }
24 void dfs(int u)
25 {
26     f[u][1] = happy[u];
27     for (int i = h[u]; ~i; i = ne[i])
28         dfs(e[i]);
29     f[u][0] += max(f[e[i]][0], f[e[i]
        ][1]);
31     f[u][1] += f[e[i]][0];
32 }
33 }

```



```

34 int main()
35 {
36     memset(h, -1, sizeof h);
37     cin >> n;
38     for (int i = 1; i <= n; i++) cin >>
happy[i];
39     for (int i = 0; i < n - 1; i++)
40     {
41         int a, b;
42         cin >> a >> b;
43         has_father[a] = true;
44         add(b, a);
45     }
46     int root = 1;
47     while (has_father[root]) root++;
48     dfs(root);
49     cout << max(f[root][0], f[root][1]);
50 }

```

## 5.8 Memoized Search

```

1  const int N = 310;
2  int n, m,
3  h[N][N], f[N][N],
4  dx[4] = {0, 1, 0, -1}, dy[4] = {1, 0, -1,
0};

```

```

5  int dp(int x, int y)
6  {
7      int &v = f[x][y];
8      if (v != -1) return v;
9      v = 1;
10     for (int i = 0; i < 4; i++)
11     {
12         int a = x + dx[i], b = y + dy[i];
13         if (a >= 1 && a <= n && b >= 1 &&
b <= m && h[a][b] < h[x][y])
14             v = max(v, dp(a, b) + 1);
15     }
16     return v;
17 }
18 int main()
19 {
20     cin >> n >> m;
21     for (int i = 1; i <= n; i++)
22         for (int j = 1; j <= m; j++)
23             cin >> h[i][j];
24     memset(f, -1, sizeof f);
25     int res = 0;
26     for (int i = 1; i <= n; i++)
27         for (int j = 1; j <= m; j++)
28             res = max(res, dp(i, j));
29     cout << res;
30 }

```



**国家超级计算广州中心**  
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

---

## 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

```
1 LL mul(LL a, LL b, LL p)
2 {
3     LL ans = 0;
4     while (b)
5     {
6         if (b & 1) ans = (ans + a) % p;
7         a = a * 2 % p; b >>= 1;
8     }
9     return ans;
10 }
```

### 6.2 Sum of Geometric Series

```
1 const int mod = 9901;
2 int a, b;
3 int qmi(int a, int k)
4 {
5     int res = 1;
6     a %= mod;
7     while (k)
8     {
9         if (k & 1)
10            res = res * a % mod;
11        a = a * a % mod;
12        k >>= 1;
13    }
14    return res;
15 }
16 int sum(int p, int k)
17 {
18     if (k == 1) return 1;
19     if (k % 2 == 0)
20         return (1 + qmi(p, k / 2)) * sum(p, k / 2) % mod;
21     return (sum(p, k - 1) + qmi(p, k - 1)) % mod;
22 }
23 int main()
24 {
25     // 以  $a^b$  约数之和为例求等比数列和
26     cin >> a >> b;
27     int res = 1;
28     for (int i = 2; i <= a / i; i++)
29         if (a % i == 0)
30         {
31             int s = 0;
32             while (a % i == 0) a /= i, s++;
33             res = res * sum(i, b * s + 1) % mod;
34         }
35     if (a > 1) res = res * sum(a, b + 1) % mod;
36 }
```

## 6.3 Sort

### 6.3.1 Card Balancing Problem

```
1 cin >> n;
2 for (int i = 1; i <= n; i++)
3     cin >> a[i], avg += a[i];
4 avg /= n;
5 for (int i = 1; i <= n; i++)
6     if (a[i] != avg)
7         a[i + 1] += a[i] - avg, ans++;
8 cout << ans;
```

### 6.3.2 2D Card Balancing Problem

```
1 const int N = 100010;
2 int row[N], col[N], c[N], s[N];
3 LL work(int n, int a[])
4 {
5     for (int i = 1; i <= n; i++)
6         s[i] = s[i - 1] + a[i];
7     if (s[n] % n) return -1;
8     int avg = s[n] / n;
9     c[1] = 0;
10    for (int i = 2; i <= n; i++)
11        c[i] = s[i - 1] - (i - 1) * avg;
12    sort(c + 1, c + n + 1);
13    LL res = 0;
14    for (int i = 1; i <= n; i++)
15        res += abs(c[i] - c[(n + 1) / 2]);
16    return res;
17 }
18 int main()
19 {
20     int n, m, cnt;
21     cin >> n >> m >> cnt;
22     while (cnt--)
23     {
24         int x, y;
25         cin >> x >> y;
26         row[x]++; col[y]++;
27     }
28     LL r = work(n, row);
29     LL c = work(m, col);
30     if (r != -1 && c != -1)
31         cout << "both " << r + c;
32     else if (r != -1)
33         cout << "row " << r;
34     else if (c != -1)
35         cout << "column " << c;
36     else cout << "impossible";
37 }
```

### 6.3.3 Dual Heaps

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

```

6  if (up.size() > down.size())
7      down.push(up.top()), up.pop();
8  if (i % 2)
9  {
10     cout << down.top() << ' ';
11     if (++cnt % 10 == 0) cout << '\n';
12 }

```

## 6.4 RMQ

```

1  const int N = 200010, M = 18;
2  int n, m, w[N], f[N][M];
3  void init()
4  {
5      for (int j = 0; j < M; j++)
6          for (int i = 1; i + (1 << j) - 1
7              <= n; i++)
8              if (!j) f[i][j] = w[i];
9              else // 也可以是最小值
10                 f[i][j] = max(f[i][j - 1],
11                               f[i + (1 << j - 1)][j - 1]);
12 }
13 int query(int l, int r)
14 {
15     int len = r - l + 1;
16     int k = log(len) / log(2);
17     return max(f[l][k], f[r - (1 << k) + 1][k]);
18 }

```

## 7 ★ Advanced Data Structures

### 7.1 Binary Indexed Tree

```
1 // 支持区间修改、区间查询
2 // 利用变差分求二阶区间和
3 const int N = 100010;
4 int n, m, a[N];
5 LL tr1[N], tr2[N];
6 int lowbit(int x) { return x & -x; }
7 void add(LL tr[], LL x, LL c)
8 {
9     for (int i = x; i <= n; i += lowbit(i))
10         tr[i] += c;
11 }
12 LL sum(LL tr[], LL x)
13 {
14     LL res = 0;
15     for (int i = x; i; i -= lowbit(i))
16         res += tr[i];
17     return res;
18 }
19 LL prefix_sum(LL x)
20 { return sum(tr1, x) * (x + 1) - sum(tr2, x); }
21 int main()
22 {
23     cin >> n >> m;
24     for (int i = 1; i <= n; i++)
25         cin >> a[i];
26     for (int i = 1; i <= n; i++)
27     {
28         int b = a[i] - a[i - 1];
29         add(tr1, i, b);
30         add(tr2, i, (LL)i * b);
31     }
32     while (m--)
33     {
34         char op[2];
35         int l, r, d;
36         cin >> op >> l >> r;
37         if (*op == 'Q')
38             cout << prefix_sum(r) -
39             prefix_sum(l - 1) << '\n';
40         else
41         {
42             cin >> d;
43             add(tr1, l, d), add(tr2, l, (
44             LL)l * d),
45             add(tr1, r + 1, -d),
46             add(tr2, r + 1, (LL)-(r + 1) *
47             d);
48         }
49     }
50 }
```

### 7.2 Segment Tree

```
1 struct Node
2 { // 可以维护任何满足区间加法的信息
```

```
3     int l, r; LL sum, add; // 区间和/懒标记
4 } tr[N * 4];
5 void pushup(int u) // 从上至下传递
6 { tr[u].sum = tr[u << 1].sum + tr[u << 1 |
7   1].sum; }
8 void pushdown(int u)
9 { // 从下至上传递
10     auto &root = tr[u],
11     &left = tr[u << 1],
12     &right = tr[u << 1 | 1];
13     if (root.add)
14     {
15         left.add += root.add,
16         left.sum += (LL)(left.r - left
17         .l + 1) * root.add;
18         right.add += root.add,
19         right.sum += (LL)(right.r -
20         right.l + 1) * root.add;
21         root.add = 0;
22     }
23 }
24 void build(int u, int l, int r)
25 { // 建树
26     if (l == r) tr[u] = {l, r, w[r], 0};
27     else
28     {
29         tr[u] = {l, r};
30         int mid = l + r >> 1;
31         build(u << 1, l, mid); // 左儿子
32         build(u << 1 | 1, mid + 1, r); //
33         右儿子
34         pushup(u); // 从下往上传递区间值
35     }
36 }
37 void modify(int u, int l, int r, int d)
38 { // 区间修改
39     if (tr[u].l >= l && tr[u].r <= r)
40     {
41         tr[u].sum += (LL)(tr[u].r - tr[u].
42         l + 1) * d;
43         tr[u].add += d;
44     }
45     else
46     {
47         pushdown(u);
48         int mid = tr[u].l + tr[u].r >> 1;
49         if (l <= mid)
50             modify(u << 1, l, r, d);
51         if (r > mid)
52             modify(u << 1 | 1, l, r, d);
53         pushup(u);
54     }
55 }
56 LL query(int u, int l, int r)
57 { // 区间查询
58     if (tr[u].l >= l && tr[u].r <= r)
59         return tr[u].sum;
60     pushdown(u);
61     int mid = tr[u].l + tr[u].r >> 1;
62     LL sum = 0;
63     if (l <= mid)
64         sum += query(u << 1, l, r);
65     if (r > mid)
66         sum += query(u << 1 | 1, l, r);
67     return sum;
68 }
```

## 8 ★ Advanced Search

## 9 ★ Advanced Graph Theory





## 11 ★ Advanced DP