



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

Monday 16th June, 2025

Contents

| | | | | | |
|----------|-----------------------------------|-----------|----------|---------------------------------|-----------|
| 0 | Preface | 5 | 4 | Basic Math | 17 |
| 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 |
| 1 | Basic Algorithm | 7 | 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 | Ternary Search | 7 | 4.2.3 | The Sum of Divisors | 17 |
| 1.4 | High Precision | 7 | 4.2.4 | Euclidean Algorithm | 18 |
| 1.4.1 | High Precision Add | 7 | 4.3 | Euler Function | 18 |
| 1.4.2 | High Precision Subsection | 8 | 4.3.1 | Simple Method | 18 |
| 1.4.3 | High Precision Multiply | 8 | 4.3.2 | Euler's Sieve Method | 18 |
| 1.4.4 | High Precision Divide | 8 | 4.4 | Exponentiating by Squaring | 18 |
| 1.5 | Prefix Sum & Difference Array | 9 | 4.5 | Extended Euclidean Algorithm | 18 |
| 1.5.1 | 1D Prefix Sum | 9 | 4.6 | Chinese Remainder Theorem | 18 |
| 1.5.2 | 2D Prefix Sum | 9 | 4.7 | Gauss-Jordan Elimination | 19 |
| 1.5.3 | 1D Difference Array | 9 | 4.7.1 | Linear Equation Group | 19 |
| 1.5.4 | 2D Difference Array | 9 | 4.7.2 | XOR Linear Equation Group | 19 |
| 2 | Basic Data Structures | 10 | 4.8 | Combinatorial Counting | 19 |
| 2.1 | Linked List | 10 | 4.8.1 | Recurrence Relation | 19 |
| 2.1.1 | Singly Linked List | 10 | 4.8.2 | Preprocessing & Inverse Element | 20 |
| 2.1.2 | Bidirectional Linked List | 10 | 4.8.3 | Lucas Theorem | 20 |
| 2.2 | Stack & Queue | 10 | 4.8.4 | Factorization Method | 20 |
| 2.2.1 | Monotonic Stack | 10 | 4.8.5 | Catalan Number | 20 |
| 2.2.2 | Monotonic Queue | 10 | 4.9 | Inclusion-Exclusion Principle | 21 |
| 2.3 | KMP | 10 | 4.10 | Game Theory | 21 |
| 2.4 | Trie | 10 | 4.10.1 | NIM Game | 21 |
| 2.5 | Disjoint-Set | 11 | 5 | Basic DP | 22 |
| 2.6 | Hash | 11 | 5.1 | Knapsack Problem | 22 |
| 2.6.1 | Simple Hash | 11 | 5.1.1 | 01 Knapsack | 22 |
| 2.6.2 | String Hash | 11 | 5.1.2 | Complete Knapsack | 22 |
| 2.7 | STL | 11 | 5.1.3 | Mutiple Knapsack | 22 |
| 3 | Search & Graph Theory | 13 | 5.1.4 | Grouped Knapsack | 22 |
| 3.1 | Representation of Tree & Graph | 13 | 5.2 | Linear DP | 22 |
| 3.1.1 | Adjacency Matrix | 13 | 5.2.1 | LIS | 22 |
| 3.1.2 | Adjacency List | 13 | 5.2.2 | LCS | 23 |
| 3.2 | DFS & BFS | 13 | 5.3 | Interval DP | 23 |
| 3.2.1 | DFS | 13 | 5.4 | Counting DP | 23 |
| 3.2.2 | BFS | 13 | 5.5 | Digit DP | 23 |
| 3.3 | Topological Sort | 13 | 5.6 | State Compression DP | 24 |
| 3.4 | Shortest Path | 13 | 5.7 | Tree DP | 24 |
| 3.4.1 | Dijkstra | 13 | 5.8 | Memoized Search | 25 |
| 3.4.2 | Bellman-Ford | 13 | 6 | Advanced Basic | 27 |
| 3.4.3 | SPFA | 14 | 6.1 | Slow Multiplication | 27 |
| 3.4.4 | Detecting Negative Circle in SPFA | 14 | 6.2 | Sum of Geometric Series | 27 |
| 3.4.5 | Floyd | 14 | 6.3 | Sort | 27 |
| 3.5 | Minimum Spanning Tree | 14 | 6.3.1 | Card Balancing Problem | 27 |
| 3.5.1 | Prim | 14 | 6.3.2 | 2D Card Balancing Problem | 27 |
| 3.5.2 | Kruskal | 15 | 6.3.3 | Dual Heaps | 27 |
| 3.6 | Bipartite Graph | 15 | 6.4 | RMQ | 28 |
| 3.6.1 | Coloring Method | 15 | | | |
| 3.6.2 | Hungarian Algorithm | 16 | | | |

| | | | | | |
|-----------|---|-----------|-----------|---|-----------|
| 7 | Advanced Data Structures | 29 | 11 | Advanced DP | 47 |
| 7.1 | Binary Indexed Tree | 29 | 11.1 | Advanced LIS | 47 |
| 7.2 | Segment Tree | 29 | 11.1.1 | MSIS | 47 |
| 7.2.1 | Maintain the Maximum | 29 | 11.1.2 | LCIS | 47 |
| 7.2.2 | Maintain the Maximum Subarray Sum | 29 | 11.2 | Knapsack Problem | 47 |
| 7.2.3 | Maintain the GCD | 30 | 11.2.1 | Multiple Knapsack Problem | 47 |
| 7.2.4 | Optimize Range Updates | 30 | 11.2.2 | Two-Dimensional Cost Knapsack Problem | 47 |
| 7.3 | Persistent Data Structure | 31 | 11.2.3 | Finding the Actual Solution Set | 47 |
| 7.3.1 | Persistent Trie | 31 | 11.2.4 | Maximum Linearly Independent Subset | 48 |
| 7.3.2 | Persistent Segment Tree | 31 | 11.2.5 | Mixed Knapsack Problem | 48 |
| 7.4 | Treap | 32 | 11.2.6 | Dependent Knapsack Problem | 48 |
| 7.5 | AC Automaton | 33 | 11.2.7 | Number of Solutions | 49 |
| 8 | Advanced Search | 34 | 11.3 | FSM | 49 |
| 8.1 | Flood-Fill | 34 | 11.4 | Digit DP | 49 |
| 8.2 | Multi-source BFS | 34 | 11.5 | Queue Optimization for DP | 49 |
| 8.3 | BFS with Deque | 34 | | | |
| 8.4 | Bidirectional BFS | 35 | | | |
| 8.5 | A* | 35 | | | |
| 8.6 | DFS Connectivity Model | 35 | | | |
| 8.7 | Iterative Deepening | 35 | | | |
| 8.8 | Bidirectional DFS | 36 | | | |
| 8.9 | IDA* | 36 | | | |
| 9 | Advanced Graph Theory | 37 | | | |
| 9.1 | Detecting Negative Cycles | 37 | | | |
| 9.2 | SPFA-SLF | 37 | | | |
| 9.3 | SPFA-Stack | 37 | | | |
| 9.4 | SPFA & MIN & MAX | 37 | | | |
| 9.5 | Second Shortest Path | 38 | | | |
| 9.6 | Second Minimum Spanning Tree | 38 | | | |
| 9.6.1 | brute-force | 38 | | | |
| 9.6.2 | LCA | 39 | | | |
| 9.7 | Difference Constraints | 40 | | | |
| 9.7.1 | Maximum-Shortest Path | 40 | | | |
| 9.7.2 | Minimum-Longest Path | 41 | | | |
| 9.8 | LCA | 41 | | | |
| 9.9 | SCC | 41 | | | |
| 9.10 | DCC | 42 | | | |
| 9.10.1 | e-DCC | 42 | | | |
| 9.10.2 | v-DCC | 42 | | | |
| 9.11 | Bipartite Graph | 42 | | | |
| 9.11.1 | maximum matching | 42 | | | |
| 9.11.2 | minimum vertex cover | 43 | | | |
| 9.11.3 | maximum independent set | 43 | | | |
| 9.11.4 | minimum path cover | 44 | | | |
| 9.12 | Eulerian Circuit & Eulerian Path | 44 | | | |
| 9.12.1 | Eulerian Circuit | 44 | | | |
| 9.12.2 | Eulerian Path | 44 | | | |
| 10 | Advanced Math | 46 | | | |
| 10.1 | Euler's Totient Function | 46 | | | |
| 10.1.1 | GCD | 46 | | | |
| 10.2 | Matrix Multiplication | 46 | | | |



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. $n \leq 10000 \rightarrow O(n^{\frac{3}{2}})$: 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>
32 #include <unordered_map>
33 #include <unordered_set>
```

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 Ternary Search

```
1 // 整数三分
2 void tsearch_1(int l, int r)
3 {
4     while (l < r)
5     {
6         int lmid = l + (r - l) / 3, rmid = r
          - (r - l) / 3;
7         lans = cal(lmid), rans = cal(rmid);
8         if (lans <= rans) r = rmid - 1;
9         else l = lmid + 1;
10        if (lans <= rans) l = lmid + 1;
11        else r = rmid - 1;
12    }
13    // 求凹函数的极小值
14    cout << min(lans, rans) << endl;
15    // 求凸函数的极大值
16    cout << max(lans, rans) << endl;
17 }
18 // 浮点数三分
19 void tsearch_2(int l, int r)
20 {
21     const double eps = 1e-6;
22     while (r - l < eps)
23     {
24         double lmid = l + (r - l) / 3;
25         double rmid = r - (r - l) / 3;
26         lans = cal(lmid), rans = cal(rmid);
27         // 求凹函数的极小值
28         if (lans <= rans) r = rmid;
29         else l = lmid;
30         // 求凸函数的极大值
31         if (lans <= rans) l = lmid;
32         else r = rmid;
33    }
34 }
```

1.4 High Precision

1.4.1 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 }

```

```

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

```

1.4.2 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 <
27        s2) 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 }

```

1.4.4 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;
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 }

```

1.4.3 High Precision Multiply

```

1 string s1, s2;
2 vector<int> a, c;
3 int b;

```

1.5 Prefix Sum & Difference Array

1.5.1 1D Prefix Sum

```

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

```


1.5.2 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]
```

```
23         insert(x1, y1, x2, y2, c);
24     }
25     // 其他过程略
26 }
```

1.5.3 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 }
```

1.5.4 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,
  int c)
4 {
5     b[x1][y1] += c;
6     b[x2 + 1][y2 + 1] += c;
7     b[x1][y2 + 1] -= c;
8     b[x2 + 1][y1] -= c;
9 }
10 int main()
11 {
12     cin >> n >> m >> q;
13     for (int i = 1; i <= n; i++)
14         for (int j = 1; j <= m; j++)
15             cin >> a[i][j];
16     for (int i = 1; i <= n; i++)
17         for (int j = 1; j <= m; j++)
18             insert(i, j, i, j, a[i][j]);
19     while (q--)
20     {
21         int x1, x2, y1, y2, c;
22         cin >> x1 >> y1 >> x2 >> y2 >> c;
```

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     stk[++tt] = i;
7 }
```

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];
```

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  }
8  bool find(int x)
9  {
10     for (int i = h[(x % N + N) % N]; i != -1; i = ne[i])
11         if (e[i] == x) return true;
12     return false;
13 }
14 // (2) 开放寻址法
15 int find(int x)
16 {
17     int t = (x % N + N) % N;
18     while (h[t] != null && h[t] != x)
19         t++; if (t == N) t = 0; }
20     return t;
21 }
```

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 - 1] *
8          P; }
9  }
10 ULL get(int l, int r) { return h[r] - h[l - 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为
14             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; q.
8         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 = ne[
13             i])
14             if (--d[e[i]] == 0) q[++tt] = e[
15                 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>, greater
10         <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 = ne
21             [i])
22             if (dist[e[i]] > distance + w[i]
23                 )
24             {
25                 dist[e[i]] = distance + w[i]
26                 ;
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[j]
17         ].b, w = edges[j].w;
18         dist[b] = min(dist[b], backup[a]
19         + w);
20     }
21     if (dist[n] > 0x3f3f3f3f / 2) return -1;
22     return dist[n];
23 }

```

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], idx
4  ;
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                 dist[e[i]] = dist[t] + w[i];
19                 if (!vis[e[i]]) vis[e[i]] =
20                 true, q.push(j);
21             }
22     }
23     dist[n] > INF / 2 ? cout << "impossible"
24     : cout << dist[n];
25 }

```

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                 dist[e[i]] = dist[t] + w[i];
16                 // 新增
17                 cnt[j] = cnt[t] + 1;

```

```

17         if (cnt[j] >= n) return true
18         if (!st[j]) q.push(j), st[j]
19         = true;
20     }
21     return false;
22 }

```

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[k
19                 ][j], g[i][j]);
20     // 后续代码略
21     return 0;
22 }

```

3.5 Minimum Spanning Tree

3.5.1 Prim

```

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 || dist[
12             j] < dist[t])) t = j;
13         if (i && dist[t] == INF) { res = INF
14         ; break; }
15         if (i) res += dist[t];
16         vis[t] = true;
17         for (int j = 1; j <= n; j++) dist[j]
18         = min(dist[j], g[t][j]);
19     }
20     res == INF ? cout << "impossible" : cout
21     << res;
22 }
23 int main()
24 {

```

```

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 {
8          return w < e.w; };
9  } edge[2 * N];
10 void init() { for (int i = 1; i <= n; i++) p[i] = i; }
11 int find(int x)
12 {
13     if (x != p[x]) p[x] = find(p[x]);
14     return p[x];
15 }
16 void merge(int x, int y) { p[find(x)] = find(y); }
17 void kruskal()
18 {
19     int res = 0, cnt = 0;
20     for (int i = 1; i <= m; i++)
21         if (find(edge[i].a) != find(edge[i].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" : cout << res;
29 }
30 int main()
31 {
32     init();
33     cin >> n >> m;
34     for (int i = 1; i <= m; i++) cin >> edge[i].a >> edge[i].b >> edge[i].w;
35     sort(edge + 1, edge + m + 1);
36     kruskal();
37     return 0;
38 }

```

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 false;
13     return true;
14 }
15 bool check()
16 {
17     for (int i = 1; i <= n; i++)
18         if (color[i] == -1)
19             if (!dfs(i, 0)) return false;
20     return true;
21 }
22 int main()
23 {
24     // 注意另外初始化 h 与 color
25     cin >> n >> m;
26     while (m--)
27     {
28         int a, b;
29         cin >> a >> b;
30         add(a, b), add(b, a);
31     }
32     // 其余过程略
33 }

```

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;
9             j++)
10            {
11                st[primes[j] * i] = true;
12                if (i % primes[j] == 0) break;
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(x
9                 / i);
10        }
11    sort(res.begin(), res.end());
12    return res;
13 }
```

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) % mod
20    ;
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;
23     for (auto iter = h.begin(); iter != h.
24         end(); iter++)
```

```

23         res = res * getSum(iter->first, iter
24         ->second) % mod;
25     cout << res;
26     return 0;
27 }

```

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，所以直接赋值 i - 1
13                phi[i] = i - 1;
14            }
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 的最小质因子
23                    // 1 - 1 / primes[j] 这一项
24                    // 在 phi[i] 中计算过了，只需将基数 N 修正为
25                    // primes[j] 倍

```

```

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

```

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
22         k1 *= (a2 - a1) / d;
23         k1 = (k1 % (m2 / d) + m2 / d) % (m2 / d);
24         x = k1 * m1 + a1;
25         LL m = abs(m1 / d * m2);
26         a1 = k1 * m1 + a1;
27         m1 = m;
28     }
29     if (x != -1)
30         x = (a1 % m1 + m1) % m1;
31     cout << x << '\n';
32     return 0;
}

```

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(
22                 i, mod - 2, mod) % mod;
23             // 此后 C(a, b) = (LL)fact[a] * infact[b]
24             // % mod * infact[a - b] % mod
25         }
26     }

```

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
13     // 乘, 略 }
14     int main()
15     {
16         int a, b;
17         cin >> a >> b;
18         getPrimes(a);
19         for (int i = 0; i < cnt; i++)
20         {
21             int p = primes[i];
22             sum[i] = get(a, p) - get(b, p) - get(
23                 a - b, p);
24         }
25         vector<int> res;
26         res.push_back(1);
27         for (int i = 0; i < cnt; i++)
28             for (int j = 0; j < sum[i]; j++)
29                 mul(res, primes[i]);
30         for (int i = res.size() - 1; i >= 0; i--)
31             cout << res[i];
32     }

```

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)
12         % mod;
13     res = (LL)res * qmi(n + 1, mod - 2, mod)
14     % mod;
15 }

```

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该
                是其累乘积不超过目标数字
18                if ((LL)t * p[j] > n)
19                    { t = -1; break; }
20                t *= p[j];
21            }
22            if (t != -1)
23                // 容斥原理公式中奇数个并集系数为
                1, 反之为 -1
24                if (cnt % 2) res += n / t;
25                else res -= n / t;
26    }
27    cout << res;
28 }

```

```

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, 并且递归向下求解
9      for (int i = 0; i < k; i++)
10     {
11         int sum = s[i];
12         if (x >= sum) S.insert(sg(x - sum));
13     }
14     // 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
15     for (int i = 0;; i++)
16         if (!S.count(i))
17             return f[x] = i;
18 }
19
20 int main()
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     {

```

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]] + w
11             [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]] + w
11             [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             s -= k, k *= 2;
17         }
18         if (s > 0)
19         {
20             cnt++;
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]] + w
28             [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 - v
17                     [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]
15                     + f[k + 1][r] + s[r] - s[l - 1]);
16             }
17     cout << f[1][n];
18 }

```

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 - 1][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
14         // <10^(右边的数的位数)>, Ps: 从左往右(从高位到低位)
15         // l 为第 j 位的左边的数, r 为右边的数, dj 为第 j 位上的数
16         int p = pow(10, dgt - j), l = n / p / 10, r = n % p, dj = n / p % 10;
17         // 求要选的数在 i 的左边的数小于 l 的情况:
18         // 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
19         // 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
20         if (i) res += l * p;
21         else res += (l - 1) * p;
22         // 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)
23         // 1)、i > dj 时 0 种选法
24         // 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 -
38             1, i) << ' ';
39         // 利用前缀和思想: [1, r] 的和 = s[r]
39         // - s[1 - 1]
40         cout << '\n';
41     }
42 }

```

```

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;
38              k++)
39              // 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
40              if (!(j & k) && st[j | k
41                  ])
42                  f[i][j] += f[i - 1][
43                  k];
44              // 输出摆放好第 m 列且第 (m + 1) 列没
      有任何方格的状态数
45              cout << f[m][0] << '\n';
46          }
47      }

```

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
14             奇数个就不能正好放下竖放的方格
15             int cnt = 0;
16             for (int j = 0; j < n && st[i];
17                 j++)
18                 if (i >> j & 1)
19                 {
20                     // 当前格子被使用
21                     // 如果连续 0 的数量为奇
22                     数个, 当前格子被使用的后果就是导致格子重
23                     合, 所以不可取
24                     if (cnt & 1)
25                         st[i] = false;
26                     // 刷新状态
27                     cnt = 0;
28                 }
29                 else cnt++;
30             // 最后再判断一次, 防止漏判
31             if (cnt & 1)
32                 st[i] = false;
33         }
34         // 没有摆放任何棋子的状态默认只有 1
35         种取法
36         f[0][0] = 1;
37         // 遍历每一列
38         for (int i = 1; i <= m; i++)

```

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 }
25 void dfs(int u)
26 {
27     f[u][1] = happy[u];
28     for (int i = h[u]; ~i; i = ne[i])
29         dfs(e[i]);
30     f[u][0] += max(f[e[i]][0], f[e[i]
31     ][1]);
32     f[u][1] += f[e[i]][0];
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,
5  0};
6  int dp(int x, int y)
7  {

```

```

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
14         <= m && h[a][b] < h[x][y])
15         v = max(v, dp(a, b) + 1);
16 }
17 return v;
18 }
19 int main()
20 {
21     cin >> n >> m;
22     for (int i = 1; i <= n; i++)
23         for (int j = 1; j <= m; j++)
24             cin >> h[i][j];
25     memset(f, -1, sizeof f);
26     int res = 0;
27     for (int i = 1; i <= n; i++)
28         for (int j = 1; j <= m; j++)
29             res = max(res, dp(i, j));
30     cout << res;
31 }

```



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 <=
              n; i++)
7              if (!j) f[i][j] = w[i];
8              else // 也可以是最小值
9                  f[i][j] = max(f[i][j - 1], f
                                [i + (1 << j - 1)][j - 1]);
10 }
11 int query(int l, int r)
12 {
13     int len = r - l + 1;
14     int k = log(len) / log(2);
15     return max(f[l][k], f[r - (1 << k) + 1][
16               k]);

```

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, (LL)
44             l * d),
45             add(tr1, r + 1, -d),
46             add(tr2, r + 1, (LL)-(r + 1) * d
47             );
48         }
49     }
50 }
```

7.2 Segment Tree

7.2.1 Maintain the Maximum

```
1 struct Node
```

```
2 { int l, r, v; } tr[N * 4];
3 void pushup(int u)
4 {
5     tr[u].v = max(tr[u << 1].v, tr[u << 1 |
6     1].v);
7 }
8 void build(int u, int l, int r)
9 {
10     tr[u] = {l, r};
11     if (l == r) return;
12     int mid = l + r >> 1;
13     build(u << 1, l, mid),
14     build(u << 1 | 1, mid + 1, r);
15 }
16 int query(int u, int l, int r)
17 {
18     if (tr[u].l >= l && tr[u].r <= r)
19         return tr[u].v;
20     int mid = tr[u].l + tr[u].r >> 1;
21     int v = 0;
22     if (l <= mid)
23         v = query(u << 1, l, r);
24     if (r > mid)
25         v = max(v, query(u << 1 | 1, l, r));
26     return v;
27 }
28 void modify(int u, int x, int v)
29 {
30     if (tr[u].l == x && tr[u].r == x)
31         tr[u].v = v;
32     else
33     {
34         int mid = tr[u].l + tr[u].r >> 1;
35         if (x <= mid)
36             modify(u << 1, x, v);
37         else
38             modify(u << 1 | 1, x, v);
39     }
40     pushup(u);
41 }
```

7.2.2 Maintain the Maximum Subarray Sum

```
1 struct Node
2 { int l, r, sum, lmax, rmax, tmax; } tr[N *
3     4];
4 void pushup(Node &u, Node &l, Node &r)
5 {
6     u.sum = l.sum + r.sum;
7     u.lmax = max(l.lmax, l.sum + r.lmax);
8     u.rmax = max(r.rmax, r.sum + l.rmax);
9     u.tmax = max(max(l.tmax, r.tmax), l.rmax
10     + r.lmax);
11 }
12 void pushup(int u)
13 { pushup(tr[u], tr[u << 1], tr[u << 1 | 1]); }
14 void build(int u, int l, int r)
15 {
16     if (l == r)
17         tr[u] = {l, r, w[r], w[r], w[r], w[r]
18         };
19     else
20         build(u << 1, l, mid),
21         build(u << 1 | 1, mid + 1, r);
22 }
```

```

17     {
18         tr[u] = {l, r};
19         int mid = l + r >> 1;
20         build(u << 1, l, mid),
21         build(u << 1 | 1, mid + 1, r);
22         pushup(u);
23     }
24 }
25 void modify(int u, int x, int v)
26 {
27     if (tr[u].l == x && tr[u].r == x)
28         tr[u] = {x, x, v, v, v, v};
29     else
30     {
31         int mid = tr[u].l + tr[u].r >> 1;
32         if (x <= mid)
33             modify(u << 1, x, v);
34         else
35             modify(u << 1 | 1, x, v);
36         pushup(u);
37     }
38 }
39 Node query(int u, int l, int r)
40 {
41     if (tr[u].l >= l && tr[u].r <= r)
42         return tr[u];
43     else
44     {
45         int mid = tr[u].l + tr[u].r >> 1;
46         if (r <= mid)
47             return query(u << 1, l, r);
48         else if (l > mid)
49             return query(u << 1 | 1, l, r);
50         else
51         {
52             auto left = query(u << 1, l, r);
53             auto right = query(u << 1 | 1, l
54             , r);
55             Node res;
56             pushup(res, left, right);
57             return res;
58         }
59     }

```

7.2.3 Maintain the GCD

```

1  struct Node
2  { int l, r; LL sum, d; } tr[N * 4];
3  LL gcd(LL a, LL b)
4  { return b ? gcd(b, a % b) : a; }
5  void pushup(Node &u, Node &l, Node &r)
6  {
7      u.sum = l.sum + r.sum;
8      u.d = gcd(l.d, r.d);
9  }
10 void pushup(int u)
11 { pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
12 }
13 void build(int u, int l, int r)
14 {
15     if (l == r)
16     {
17         LL b = w[r] - w[r - 1];

```

```

17         tr[u] = {l, r, b, b};
18     }
19     else
20     {
21         tr[u].l = l, tr[u].r = r;
22         int mid = l + r >> 1;
23         build(u << 1, l, mid),
24         build(u << 1 | 1, mid + 1, r);
25         pushup(u);
26     }
27 }
28 void modify(int u, int x, LL v)
29 {
30     if (tr[u].l == x && tr[u].r == x)
31     {
32         LL b = tr[u].sum + v;
33         tr[u] = {x, x, b, b};
34     }
35     else
36     {
37         int mid = tr[u].l + tr[u].r >> 1;
38         if (x <= mid)
39             modify(u << 1, x, v);
40         else
41             modify(u << 1 | 1, x, v);
42         pushup(u);
43     }
44 }
45 Node query(int u, int l, int r)
46 {
47     if (tr[u].l >= l && tr[u].r <= r)
48         return tr[u];
49     else
50     {
51         int mid = tr[u].l + tr[u].r >> 1;
52         if (r <= mid)
53             return query(u << 1, l, r);
54         else if (l > mid)
55             return query(u << 1 | 1, l, r);
56         else
57         {
58             auto left = query(u << 1, l, r);
59             auto right = query(u << 1 | 1, l
60             , r);
61             Node res;
62             pushup(res, left, right);
63             return res;
64         }
65     }

```

7.2.4 Optimize Range Updates

Use this when you need to get summary of a specific range of an array but you also need to modify a specific range of an array:

```

1  struct Node
2  { int l, r; LL sum, add; } tr[N * 4];
3  void pushup(int u)
4  { tr[u].sum = tr[u << 1].sum + tr[u << 1 |
5  1].sum; }
6  void pushdown(int u)
7  {
8      auto &root = tr[u],

```

```

8      &left = tr[u << 1],
9      &right = tr[u << 1 | 1];
10     if (root.add)
11     {
12         left.add += root.add,
13         left.sum += (LL)(left.r - left.l +
14         1) * root.add;
15         right.add += root.add,
16         right.sum += (LL)(right.r - right.l
17         + 1) * root.add;
18         root.add = 0;
19     }
20 }
21 void build(int u, int l, int r)
22 {
23     if (l == r) tr[u] = {l, r, w[r], 0};
24     else
25     {
26         tr[u] = {l, r};
27         int mid = l + r >> 1;
28         build(u << 1, l, mid);
29         build(u << 1 | 1, mid + 1, r);
30         pushup(u);
31     }
32 }
33 void modify(int u, int l, int r, int d)
34 {
35     if (tr[u].l >= l && tr[u].r <= r)
36     {
37         tr[u].sum += (LL)(tr[u].r - tr[u].l
38         + 1) * d;
39         tr[u].add += d;
40     }
41     else
42     {
43         pushdown(u);
44         int mid = tr[u].l + tr[u].r >> 1;
45         if (l <= mid)
46             modify(u << 1, l, r, d);
47         if (r > mid)
48             modify(u << 1 | 1, l, r, d);
49         pushup(u);
50     }
51 }
52 LL query(int u, int l, int r)
53 {
54     if (tr[u].l >= l && tr[u].r <= r)
55         return tr[u].sum;
56     pushdown(u);
57     int mid = tr[u].l + tr[u].r >> 1;
58     LL sum = 0;
59     if (l <= mid)
60         sum += query(u << 1, l, r);
61     if (r > mid)
62         sum += query(u << 1 | 1, l, r);
63     return sum;
64 }

```

7.3 Persistent Data Structure

7.3.1 Persistent Trie

```
1  const int N = 600010, M = N * 25;
```

```

2  int n, m, s[N], root[N], idx;
3  int trie[M][2], max_id[M];
4  void insert(int i, int k, int p, int q)
5  {
6      if (k < 0)
7      {
8          max_id[q] = i;
9          return;
10     }
11     int v = s[i] >> k & 1;
12     if (p)
13         trie[q][v ^ 1] = trie[p][v ^ 1];
14     trie[q][v] = ++idx;
15     insert(i, k - 1, trie[p][v], trie[q][v]);
16     max_id[q] = max(max_id[trie[q][0]],
17                     max_id[trie[q][1]]);
18 }
19 int query(int root, int C, int L)
20 {
21     int p = root;
22     for (int i = 23; i >= 0; i--)
23     {
24         int v = C >> i & 1;
25         if (max_id[trie[p][v ^ 1]] >= L)
26             p = trie[p][v ^ 1];
27         else
28             p = trie[p][v];
29     }
30     return C ^ s[max_id[p]];
31 }
32 // insert(i, 23, root[i - 1], root[i]);
33 // query(root[r - 1], l - 1, x ^ s[n]);

```

7.3.2 Persistent Segment Tree

```

1  const int N = 100010, M = 10010;
2  int n, m, a[N], root[N], idx;
3  vector<int> nums;
4  struct Node
5  {
6      int l, r;
7      int cnt;
8  } tr[N * 4 + N * 17];
9  int find(int x)
10 {
11     return lower_bound(nums.begin(), nums.
12     end(), x) - nums.begin();
13 }
14 int build(int l, int r)
15 {
16     int p = ++idx;
17     if (l == r)
18         return p;
19     int mid = l + r >> 1;
20     tr[p].l = build(l, mid), tr[p].r = build
21     (mid + 1, r);
22     return p;
23 }
24 int insert(int p, int l, int r, int x)
25 {
26     int q = ++idx;
27     tr[q] = tr[p];
28     if (l == r)

```

```

27     {
28         tr[q].cnt++;
29         return q;
30     }
31     int mid = l + r >> 1;
32     if (x <= mid)
33         tr[q].l = insert(tr[p].l, l, mid, x)
34     ;
35     else
36         tr[q].r = insert(tr[p].r, mid + 1, r, x);
37     tr[q].cnt = tr[tr[q].l].cnt + tr[tr[q].r].cnt;
38     return q;
39 }
40 int query(int q, int p, int l, int r, int k)
41 {
42     if (l == r)
43         return r;
44     int cnt = tr[tr[q].l].cnt - tr[tr[p].l].cnt;
45     int mid = l + r >> 1;
46     if (k <= cnt)
47         return query(tr[q].l, tr[p].l, l, mid, k);
48     else
49         return query(tr[q].r, tr[p].r, mid + 1, r, k - cnt);
50 }

```

7.4 Treap

```

1  const int N = 100010, INF = 1e8;
2  int n, root, idx;
3  struct Node
4  { int l, r, key, val, cnt, size; } tr[N];
5  void pushup(int p)
6  {
7      tr[p].size = tr[tr[p].l].size +
8      tr[tr[p].r].size + tr[p].cnt;
9  }
10 int get_node(int key)
11 {
12     tr[++idx].key = key;
13     tr[idx].val = rand();
14     tr[idx].cnt = tr[idx].size = 1;
15     return idx;
16 }
17 void zig(int &p)
18 {
19     int q = tr[p].l;
20     tr[p].l = tr[q].r, tr[q].r = p, p = q;
21     pushup(tr[p].r), pushup(p);
22 }
23 void zag(int &p)
24 {
25     int q = tr[p].r;
26     tr[p].r = tr[q].l, tr[q].l = p, p = q;
27     pushup(tr[p].l), pushup(p);
28 }
29 void build()
30 {

```

```

31     get_node(-INF), get_node(INF);
32     root = 1, tr[1].r = 2;
33     pushup(root);
34     if (tr[1].val < tr[2].val) zag(root);
35 }
36 void insert(int &p, int key)
37 {
38     if (!p) p = get_node(key);
39     else if (tr[p].key == key) tr[p].cnt++;
40     else if (tr[p].key > key)
41     {
42         insert(tr[p].l, key);
43         if (tr[tr[p].l].val > tr[p].val)
44             zig(p);
45     }
46     else
47     {
48         insert(tr[p].r, key);
49         if (tr[tr[p].r].val > tr[p].val)
50             zag(p);
51     }
52     pushup(p);
53 }
54 void remove(int &p, int key)
55 {
56     if (!p) return;
57     if (tr[p].key == key)
58     {
59         if (tr[p].cnt > 1) tr[p].cnt--;
60         else if (tr[p].l || tr[p].r)
61         {
62             if (!tr[p].r || tr[tr[p].l].val > tr[tr[p].r].val)
63             {
64                 zig(p);
65                 remove(tr[p].r, key);
66             }
67             else
68             {
69                 zag(p);
70                 remove(tr[p].l, key);
71             }
72         }
73         else p = 0;
74     }
75     else if (tr[p].key > key)
76         remove(tr[p].l, key);
77     else remove(tr[p].r, key);
78     pushup(p);
79 }
80 int get_rank_by_key(int p, int key)
81 {
82     if (!p) return 0;
83     if (tr[p].key == key)
84         return tr[tr[p].l].size + 1;
85     if (tr[p].key > key)
86         return get_rank_by_key(tr[p].l, key);
87     return tr[tr[p].l].size + tr[p].cnt + get_rank_by_key(tr[p].r, key);
88 }
89 int get_key_by_rank(int p, int rank)
90 {
91     if (!p) return INF;
92     if (tr[tr[p].l].size >= rank)
93         return get_key_by_rank(tr[p].l, rank);

```



```

    );
94     if (tr[tr[p].l].size + tr[p].cnt >= rank
    )
95         reutrn tr[p].key;
96     return get_key_by_rank(tr[p].r, rank -
    tr[tr[p].l].size - tr[p].cnt);
97 }
98 int get_prev(int p, int key)
99 {
100     if (!p) return -INF;
101     if (tr[p].key >= key)
102         reutrn get_prev(tr[p].l, key);
103     return max(tr[p].key, get_prev(tr[p].r,
    key));
104 }
105 int get_next(int p, int key)
106 {
107     if (!p) reutrn INF;
108     if (tr[p].key <= key)
109         return get_next(tr[p].r, key);
110     return min(tr[p].key, get_next(tr[p].l,
    key));
111 }

```

7.5 AC Automaton

```

1  const int N = 10010, M = 1000010, S = 55;
2  int n, tr[N * S][26], cnt[N * S], idx;
3  int q[N * S], ne[N * S];
4  char str[M];
5  void insert()
6  {
7      int p = 0;
8      for (int i = 0; str[i]; i++)
9      {
10         int t = str[i] - 'a';
11         if (!tr[p][t]) tr[p][t] = ++idx;
12         p = tr[p][t];
13     }
14     cnt[p]++;
15 }
16 void build()
17 {
18     int hh = 0, tt = -1;
19     for (int i = 0; i < 26; i++)
20         if (tr[0][i]) q[++tt] = tr[0][i];
21     while (hh <= tt)
22     {
23         int t = q[hh++];
24         for (int i = 0; i < 26; i++)
25         {
26             int p = tr[t][i];
27             if (!p) tr[t][i] = tr[ne[t]][i];
28             else
29             {
30                 ne[p] = tr[ne[t]][i];
31                 q[++tt] = p;
32             }
33         }
34     }
35 }

```

8.1 Flood-Fill

```

1  const int N = 1010, M = N * N;
2  int n, m;
3  char g[N][N];
4  PII q[M];
5  bool st[N][N];
6  void bfs(int sx, int sy)
7  {
8      int hh = 0, tt = 0;
9      q[0] = {sx, sy}; st[sx][sy] = true;
10     while (hh <= tt)
11     {
12         PII t = q[hh++];
13         for (int i = t.first - 1; i <= t.first + 1; i++)
14             for (int j = t.second - 1; j <= t.second + 1; j++)
15             {
16                 if (i == t.first && j == t.second)
17                     continue;
18                 if (i < 0 || i >= n || j < 0 || j >= m)
19                     continue;
20                 if (g[i][j] == '.' || st[i][j])
21                     continue;
22                 q[++tt] = {i, j};
23                 st[i][j] = true;
24             }
25     }
26 }
27 int main()
28 {
29     int cnt = 0;
30     for (int i = 0; i < n; i++)
31         for (int j = 0; j < m; j++)
32             if (g[i][j] == 'W' && !st[i][j])
33                 { bfs(i, j); cnt++; }
34 }

```

8.2 Multi-source BFS

```

1  const int N = 1010, M = N * N;
2  int n, m, dist[N][N];
3  char g[N][N];
4  PII q[M];
5  int dx[4] = {-1, 0, 1, 0},
6      dy[4] = {0, 1, 0, -1};
7  void bfs()
8  {
9      memset(dist, -1, sizeof dist);
10     int hh = 0, tt = -1;
11     for (int i = 1; i <= n; i++)
12         for (int j = 1; j <= m; j++)
13             if (g[i][j] == '1')
14                 {
15                     dist[i][j] = 0;

```

```

16         q[++tt] = {i, j};
17     }
18     while (hh <= tt)
19     {
20         auto t = q[hh++];
21         for (int i = 0; i < 4; i++)
22         {
23             int a = t.x + dx[i], b = t.y +
dy[i];
24             if (a < 1 || a > n | b < 1 || b
> m) continue;
25             if (dist[a][b] != -1) continue;
26             dist[a][b] = dist[t.x][t.y] + 1;
27             q[++tt] = {a, b};
28         }
29     }
30 }

```

8.3 BFS with Deque

```

1  const int N = 510, M = N * N;
2  int n, m, dist[N][N];
3  char g[N][N];
4  bool st[N][N];
5  int dx[4] = {-1, -1, 1, 1},
6      dy[4] = {-1, 1, 1, -1},
7      ix[4] = {-1, -1, 0, 0},
8      iy[4] = {-1, 0, 0, -1};
9  int bfs()
10 {
11     memset(dist, 0x3f, sizeof dist);
12     memset(st, 0, sizeof st);
13     dist[0][0] = 0;
14     deque<PII> q;
15     q.push_back({0, 0});
16     char cs[] = "\\\/\\\/";
17     while (q.size())
18     {
19         PII t = q.front();
20         q.pop_front();
21         if (st[t.x][t.y]) continue;
22         st[t.x][t.y] = true;
23         for (int i = 0; i < 4; i++)
24         {
25             int a = t.x + dx[i], b = t.y +
dy[i];
26             if (a < 0 || a > n || b < 0 || b
> m) continue;
27             int ca = t.x + ix[i], cb = t.y +
iy[i];
28             int d = dist[t.x][t.y] +
(g[ca][cb] != cs[i]);
29             if (d < dist[a][b])
30             {
31                 dist[a][b] = d;
32                 if (g[ca][cb] != cs[i])
33                     q.push_back({a, b});
34                 else
35                     q.push_front({a, b});
36             }
37         }
38     }
39 }
40 return dist[n][m];

```

```
41 }
```

8.4 Bidirectional BFS

```
1 int bfs()
2 {
3     if (A == B) return 0;
4     queue<string> qa, qb;
5     unordered_map<string, int> da, db;
6     qa.push(A), qb.push(B);
7     da[A] = db[B] = 0;
8     int step = 0;
9     while (qa.size() && qb.size())
10    {
11        int t;
12        if (qa.size() < qb.size())
13            // PROCESS
14        else
15            // PROCESS
16        if (t <= 10) return t;
17        if (++step == 10) return -1;
18    }
19    return -1;
20 }
```

8.5 A*

```
1 const int N = 1010, M = 200010;
2 int n, m, S, T, K;
3 int h[N], rh[N], e[M], w[M], ne[M], idx;
4 int dist[N], cnt[N];
5 bool st[N];
6 void dijkstra()
7 {
8     priority_queue<PII, vector<PII>, greater<PII>> heap;
9     heap.push({0, T});
10    memset(dist, 0x3f, sizeof dist);
11    dist[T] = 0;
12    while (heap.size())
13    {
14        auto t = heap.top();
15        heap.pop();
16        int ver = t.y;
17        if (st[ver]) continue;
18        st[ver] = true;
19        for (int i = rh[ver]; ~i; i = ne[i])
20        {
21            int j = e[i];
22            if (dist[j] > dist[ver] + w[i])
23            {
24                dist[j] = dist[ver] + w[i];
25                heap.push({dist[j], j});
26            }
27        }
28    }
29 }
30
31 int astar()
32 {
```

```
33 priority_queue<PIII, vector<PIII>,
34 greater<PIII>> heap;
35 heap.push({dist[S], {0, S}});
36 while (heap.size())
37 {
38     auto t = heap.top();
39     heap.pop();
40     int ver = t.y.y, distance = t.y.x;
41     cnt[ver]++;
42     if (cnt[T] == K) return distance;
43     for (int i = h[ver]; ~i; i = ne[i])
44     {
45         int j = e[i];
46         if (cnt[j] < K)
47             heap.push({distance + w[i] +
48 dist[j], {distance + w[i], j}});
49     }
50 }
51 int main()
52 {
53     // PROCESS
54     dijkstra(); cout << astar();
55     // PROCESS
56 }
```

8.6 DFS Connectivity Model

```
1 char g[N][N];
2 int xa, ya, xb, yb;
3 int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};
4 bool st[N][N];
5 bool dfs(int x, int y)
6 {
7     if (g[x][y] == '#') return false;
8     if (x == xb && y == yb) return true;
9     st[x][y] = true;
10    for (int i = 0; i < 4; i++)
11    {
12        int a = x + dx[i], b = y + dy[i];
13        if (a < 0 || a >= n || b < 0 || b >= n) continue;
14        if (st[a][b]) continue;
15        if (dfs(a, b)) return true;
16    }
17    return false;
18 }
```

8.7 Iterative Deepening

```
1 const int N = 110;
2 int n, path[N];
3 bool dfs(int u, int k)
4 {
5     if (u == k)
6         return path[u - 1] == n;
7     bool st[N] = {0};
8     for (int i = u - 1; i >= 0; i--)
```

```

9         for (int j = i; j >= 0; j--)
10         {
11             int s = path[i] + path[j];
12             if (s > n || s <= path[u - 1] ||
13                 st[s]) continue;
14             st[s] = true;
15             path[u] = s;
16             if (dfs(u + 1, k)) return true;
17         }
18     return false;
19 }

```

8.8 Bidirectional DFS

```

1  const int N = 1 << 24;
2  int n, m, k, cnt = 0, ans;
3  int g[50], weights[N];
4  void dfs(int u, int s)
5  {
6      if (u == k)
7      {
8          weights[cnt++] = s;
9          return;
10     }
11     if ((LL)s + g[u] <= m)
12         dfs(u + 1, s + g[u]);
13     dfs(u + 1, s);
14 }
15 void dfs2(int u, int s)
16 {
17     if (u == n)
18     {
19         int l = 0, r = cnt - 1;
20         while (l < r)
21         {
22             int mid = l + r + 1 >> 1;
23             if (weights[mid] + (LL)s <= m)
24                 l = mid;
25             else r = mid - 1;
26         }
27         if (weights[l] + (LL)s <= m)
28             ans = max(ans, weights[l] + s);
29         return;
30     }
31     if ((LL)s + g[u] <= m)
32         dfs2(u + 1, s + g[u]);
33     dfs2(u + 1, s);
34 }

```

```

9         tot++;
10        return (tot + 2) / 3;
11    }
12    bool IDAstar(int u, int maxn)
13    {
14        if (f() > maxn - u) return false;
15        if (u == maxn) return true;
16        string temp = t;
17        for (int len = 1; len <= n - 1; len++)
18        {
19            for (int l = 0; l <= n - len; l++)
20            {
21                int r = l + len - 1;
22                string substr = temp.substr(l, r
23                    - l + 1);
24                for (int k = r + 1; k < n; k++)
25                {
26                    t.insert(k + 1, substr);
27                    t.erase(l, r - l + 1);
28                    if (IDAstar(u + 1, maxn))
29                        return true;
30                    t = temp;
31                }
32            }
33            return false;
34        }

```

8.9 IDA*

```

1  const int N = 1e2;
2  int n, a[N];
3  string t;
4  int f()
5  {
6      int tot = 0;
7      for (int i = 1; i < n; i++)
8          if (t[i - 1] != t[i] - 1)

```

9 ★ Advanced Graph Theory

9.1 Detecting Negative Cycles

```
1  int n, m1, m2;
2  int h[N], e[M], w[M], ne[M], idx;
3  int dist[N], q[N], cnt[N];
4  bool st[N];
5  bool spfa()
6  {
7      memset(dist, 0, sizeof dist);
8      memset(cnt, 0, sizeof cnt);
9      memset(st, 0, sizeof st);
10     int hh = 0, tt = 0;
11     for (int i = 1; i <= n; i++)
12     { q[tt++] = i; st[i] = true; }
13     while (hh != tt)
14     {
15         int t = q[hh++];
16         if (hh == N) hh = 0;
17         st[t] = false;
18         for (int i = h[t]; ~i; i = ne[i])
19         {
20             int j = e[i];
21             if (dist[j] > dist[t] + w[i])
22             {
23                 dist[j] = dist[t] + w[i];
24                 cnt[j] = cnt[t] + 1;
25                 if (cnt[j] >= n)
26                     return true;
27                 if (!st[j])
28                 {
29                     q[tt++] = j;
30                     if (tt == N) tt = 0;
31                     st[j] = true;
32                 }
33             }
34         }
35     }
36     return false;
37 }
```

9.2 SPFA-SLF

Using deque to solve SPFA question.

```
1  void spfa()
2  {
3      memset(dist, 0x3f, sizeof dist);
4      memset(st, 0, sizeof st);
5      deque<int> q;
6      q.push_back(s);
7      st[s] = 1, dist[s] = 0;
8      while (q.size())
9      {
10         int t = q.front();
11         q.pop_front();
12         st[t] = 0;
13         for (int i = h[t]; ~i; i = ne[i])
14         {
15             int j = e[i];
16             if (dist[j] > dist[t] + w[i])
```

```
17         {
18             dist[j] = dist[t] + w[i];
19             if (!st[j])
20             {
21                 st[j] = true;
22                 if (q.size() && dist[j]
23                     < dist[q.front()])
24                     q.push_front(j);
25                 else q.push_back(j);
26             }
27         }
28     }
29 }
```

9.3 SPFA-Stack

```
1  bool spfa()
2  {
3      int hh = 0, tt = 1;
4      memset(dist, -0x3f, sizeof dist);
5      dist[0] = 0; q[0] = 0;
6      while (hh != tt)
7      {
8          int t = q[--tt];
9          st[t] = false;
10         for(int i = h[t]; ~i; i = ne[i])
11         {
12             int j = e[i];
13             if (dist[j] < dist[t] + w[i])
14             {
15                 dist[j] = dist[t] + w[i];
16                 cnt[j] = cnt[t] + 1;
17                 if (cnt[j] >= n + 1) return
18                     true;
19                 if (!st[j])
20                 {
21                     st[j] = true; q[tt++] =
22                         j;
23                 }
24             }
25         }
26     }
```

9.4 SPFA & MIN & MAX

Using SPFA to maintain the minimum and maximum. In this case we need **Original Graph** and **Reverse Graph**, in which we can use `type == 0` or `type == 1` to describe.

```
1  void spfa(int h[], int dist[], int type)
2  {
3      int hh = 0, tt = 1;
4      if (type == 0)
5      {
6          memset(dist, 0x3f, sizeof dmin);
7          dist[1] = w[1]; q[0] = 1;
8      }
```

```

9     else
10    {
11        memset(dist, -0x3f, sizeof dmax);
12        dist[n] = w[n]; q[0] = n;
13    }
14    while (hh != tt)
15    {
16        int t = q[hh++];
17        if (hh == N) hh = 0;
18        st[t] = false;
19        for (int i = h[t]; ~i; i = ne[i])
20        {
21            int j = e[i];
22            if (type == 0 && dist[j] > min(
dist[t], w[j]) || type == 1 && dist[j] <
max(dist[t], w[j]))
23            {
24                if (type == 0)
25                    dist[j] = min(dist[t], w
[j]);
26                else
27                    dist[j] = max(dist[t], w
[j]);
28                if (!st[j])
29                {
30                    q[tt++] = j;
31                    if (tt == N) tt = 0;
32                    st[j] = true;
33                }
34            }
35        }
36    }
37 }

```

9.5 Second Shortest Path

```

1  const int N = 1010, M = 20010;
2  struct Ver
3  {
4      int id, type, dist;
5      bool operator>(const Ver &W) const
6      { return dist > W.dist; }
7  };
8  int n, m, S, T, dist[N][2], cnt[N][2];
9  int h[N], e[M], w[M], ne[M], idx;
10 bool st[N][2];
11 void add(int a, int b, int c)
12 {
13     e[idx] = b, w[idx] = c, ne[idx] = h[a],
h[a] = idx++;
14 }
15 int dijkstra()
16 {
17     memset(st, 0, sizeof st);
18     memset(dist, 0x3f, sizeof dist);
19     memset(cnt, 0, sizeof cnt);
20     dist[S][0] = 0, cnt[S][0] = 1;
21     priority_queue<Ver, vector<Ver>, greater
<Ver>> heap;
22     heap.push({S, 0, 0});
23     while (heap.size())
24     {
25         Ver t = heap.top();

```

```

26         heap.pop();
27         int ver = t.id, type = t.type,
distance = t.dist, count = cnt[ver][type
];
28         if (st[ver][type]) continue;
29         st[ver][type] = true;
30         for (int i = h[ver]; ~i; i = ne[i])
31         {
32             int j = e[i];
33             if (dist[j][0] > distance + w[i
])
34                 {
35                     dist[j][1] = dist[j][0], cnt
[j][1] = cnt[j][0];
36                     heap.push({j, 1, dist[j
][1]});
37                     dist[j][0] = distance + w[i
], cnt[j][0] = count;
38                     heap.push({j, 0, dist[j
][0]});
39                 }
40             else if (dist[j][0] == distance
+ w[i])
41                 cnt[j][0] += count;
42             else if (dist[j][1] > distance +
w[i])
43                 {
44                     dist[j][1] = distance + w[i
], cnt[j][1] = count;
45                     heap.push({j, 1, dist[j
][1]});
46                 }
47             else if (dist[j][1] == distance
+ w[i])
48                 cnt[j][1] += count;
49             }
50         }
51         int res = cnt[T][0];
52         if (dist[T][0] + 1 == dist[T][1])
53             res += cnt[T][1];
54         return res;
55     }

```

9.6 Second Minimum Spanning Tree

9.6.1 brute-force

```

1  const int N = 510, M = 10010;
2  int n, m, p[N], dist1[N][N], dist2[N][N];
3  int h[N], e[N * 2], w[N * 2], ne[N * 2], idx
;
4  struct Edge
5  {
6      int a, b, w;
7      bool f;
8      bool operator<(const Edge &e) const
9      { return w < e.w; }
10 } edge[M];
11 void add(int a, int b, int c)
12 {
13     e[idx] = b, w[idx] = c, ne[idx] = h[a], h[
a] = idx++;

```

```

14 }
15 int find(int x)
16 {
17     if (p[x] != x) p[x] = find(p[x]);
18     return p[x];
19 }
20 void dfs(int u, int fa, int maxd1, int maxd2,
21         , int d1[], int d2[])
22 {
23     d1[u] = maxd1, d2[u] = maxd2;
24     for (int i = h[u]; ~i; i = ne[i])
25     {
26         int j = e[i];
27         if (j != fa)
28         {
29             int td1 = maxd1, td2 = maxd2;
30             if (w[i] > td1)
31                 td2 = td1, td1 = w[i];
32             else if (w[i] < td1 && w[i] > td2)
33                 td2 = w[i];
34             dfs(j, u, td1, td2, d1, d2);
35         }
36     }
37 }
38 int main()
39 {
40     cin >> n >> m;
41     memset(h, -1, sizeof h);
42     for (int i = 0; i < m; i++)
43         cin >> edge[i].a >> edge[i].b >> edge[i].w;
44     sort(edge, edge + m);
45     for (int i = 1; i <= n; i++) p[i] = i;
46     LL sum = 0;
47     for (int i = 0; i < m; i++)
48     {
49         int a = edge[i].a, b = edge[i].b, w = edge[i].w;
50         int pa = find(a), pb = find(b);
51         if (pa != pb)
52         {
53             p[pa] = pb;
54             sum += w;
55             add(a, b, w), add(b, a, w);
56             edge[i].f = true;
57         }
58     }
59     for (int i = 1; i <= n; i++)
60         dfs(i, -1, -1e9, -1e9, dist1[i], dist2[i]);
61     LL res = 1e18;
62     for (int i = 0; i < m; i++)
63         if (!edge[i].f)
64         {
65             int a = edge[i].a, b = edge[i].b, w = edge[i].w;
66             LL t;
67             if (w > dist1[a][b])
68                 t = sum + w - dist1[a][b];
69             else if (w > dist2[a][b])
70                 t = sum + w - dist2[a][b];
71             res = min(res, t);
72         }
73 }

```

9.6.2 LCA

```

1  const int N = 100010, M = 300010;
2  int n, m, p[N], q[N];
3  int h[N], e[M], w[M], ne[M], idx;
4  int depth[N], fa[N][17], d1[N][17], d2[N][17];
5  struct Edge
6  {
7      int a, b, w;
8      bool used;
9      bool operator<(const Edge &t) const
10     { return w < t.w; }
11 } edge[M];
12 void add(int a, int b, int c)
13 {
14     e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
15 }
16 int find(int x)
17 {
18     if (p[x] != x) p[x] = find(p[x]);
19     return p[x];
20 }
21 LL kruskal()
22 {
23     for (int i = 1; i <= n; i++) p[i] = i;
24     sort(edge, edge + m);
25     LL res = 0;
26     for (int i = 0; i < m; i++)
27     {
28         int a = find(edge[i].a), b = find(edge[i].b), w = edge[i].w;
29         if (a != b)
30         {
31             p[a] = b; res += w;
32             edge[i].used = true;
33         }
34     }
35     return res;
36 }
37 void build()
38 {
39     memset(h, -1, sizeof h);
40     for (int i = 0; i < m; i++)
41         if (edge[i].used)
42         {
43             int a = edge[i].a, b = edge[i].b, w = edge[i].w;
44             add(a, b, w), add(b, a, w);
45         }
46 }
47 void bfs()
48 {
49     memset(depth, 0x3f, sizeof depth);
50     depth[0] = 0, depth[1] = 1, q[0] = 1;
51     int hh = 0, tt = 0;
52     while (hh <= tt)
53     {
54         int t = q[hh++];
55         for (int i = h[t]; ~i; i = ne[i])
56         {
57             int j = e[i];
58             if (depth[j] > depth[t] + 1)
59                 depth[j] = depth[t] + 1;
60         }
61     }
62 }

```

```

60     depth[j] = depth[t] + 1;
61     q[++tt] = j;
62     fa[j][0] = t;
63     d1[j][0] = w[i], d2[j][0] = -INF;
64     for (int k = 1; k <= 16; k++)
65     {
66         int anc = fa[j][k - 1];
67         fa[j][k] = fa[anc][k - 1];
68         int distance[4] = {d1[j][k - 1],
69                             d2[j][k - 1],
70                             d1[anc][k - 1],
71                             d2[anc][k -
72     1]};
73     d1[j][k] = d2[j][k] = -INF;
74     for (int u = 0; u < 4; u++)
75     {
76         int d = distance[u];
77         if (d > d1[j][k])
78             d2[j][k] = d1[j][k], d1[j][k]
79             = d;
80         else if (d != d1[j][k] && d > d2
81             [j][k])
82             d2[j][k] = d;
83     }
84 }
85 }
86 int lca(int a, int b, int w)
87 {
88     static int distance[N * 2];
89     int cnt = 0;
90     if (depth[a] < depth[b])
91         swap(a, b);
92     for (int k = 16; k >= 0; k--)
93         if (depth[fa[a][k]] >= depth[b])
94         {
95             distance[cnt++] = d1[a][k];
96             distance[cnt++] = d2[a][k];
97             a = fa[a][k];
98         }
99     if (a != b)
100     {
101         for (int k = 16; k >= 0; k--)
102             if (fa[a][k] != fa[b][k])
103             {
104                 distance[cnt++] = d1[a][k];
105                 distance[cnt++] = d2[a][k];
106                 distance[cnt++] = d1[b][k];
107                 distance[cnt++] = d2[b][k];
108                 a = fa[a][k], b = fa[b][k];
109             }
110         distance[cnt++] = d1[a][0];
111         distance[cnt++] = d1[b][0];
112     }
113     int dist1 = -INF, dist2 = -INF;
114     for (int i = 0; i < cnt; i++)
115     {
116         int d = distance[i];
117         if (d > dist1)
118             dist2 = dist1, dist1 = d;
119         else if (d != dist1 && d > dist2)
120             dist2 = d;
121     }
122     if (w > dist1) return w - dist1;

```

```

123     if (w > dist2) return w - dist2;
124     return INF;
125 }
126 int main()
127 {
128     cin >> n >> m;
129     for (int i = 0; i < m; i++)
130     {
131         int a, b, c;
132         cin >> a >> b >> c;
133         edge[i] = {a, b, c};
134     }
135     LL sum = kruskal();
136     build();
137     bfs();
138     LL res = 1e18;
139     for (int i = 0; i < m; i++)
140         if (!edge[i].used)
141         {
142             int a = edge[i].a, b = edge[i].b, w =
143             edge[i].w;
144             res = min(res, sum + lca(a, b, w));
145         }
146     cout << res;

```

9.7 Difference Constraints

- size == N: Feasible Solution
- size == 1: Maximum/Minimum
- Maximum: Shortest Path
- Minimum: Longest Path

9.7.1 Maximum-Shortest Path

```

1 bool spfa(int size)
2 {
3     int hh = 0, tt = 0;
4     memset(dist, 0x3f, sizeof dist);
5     memset(st, 0, sizeof st);
6     memset(cnt, 0, sizeof cnt);
7     for (int i = 1; i <= size; i++)
8     {
9         q[tt++] = i;
10        dist[i] = 0;
11        st[i] = true;
12    }
13    while (hh != tt)
14    {
15        int t = q[hh++];
16        if (hh == N) hh = 0;
17        st[t] = false;
18        for (int i = h[t]; ~i; i = ne[i])
19        {
20            int j = e[i];
21            if (dist[j] > dist[t] + w[i])
22            {
23                dist[j] = dist[t] + w[i];
24                cnt[j] = cnt[t] + 1;
25                if (cnt[j] >= n)
26                    return true;

```



```

27         if (!st[j])
28         {
29             st[j] = true;
30             q[tt++] = j;
31             if (tt == N) tt = 0;
32         }
33     }
34 }
35 }
36 return false;
37 }
38 int main()
39 {
40     // add(a, b, k) means x_b <= x_a + k
41     // PROCESS
42 }

```

9.7.2 Minimum-Longest Path

```

1 bool spfa(int size)
2 {
3     int hh = 0, tt = 0;
4     memset(dist, -0x3f, sizeof dist);
5     memset(st, 0, sizeof st);
6     memset(cnt, 0, sizeof cnt);
7     for (int i = 1; i <= size; i++)
8     {
9         q[tt++] = i;
10        dist[i] = 0;
11        st[i] = true;
12    }
13    while (hh != tt)
14    {
15        int t = q[hh++];
16        if (hh == N) hh = 0;
17        st[t] = false;
18        for (int i = h[t]; ~i; i = ne[i])
19        {
20            int j = e[i];
21            if (dist[j] < dist[t] + w[i])
22            {
23                dist[j] = dist[t] + w[i];
24                cnt[j] = cnt[t] + 1;
25                if (cnt[j] >= n)
26                    return false;
27                if (!st[j])
28                {
29                    st[j] = true;
30                    q[tt++] = j;
31                    if (tt == N) tt = 0;
32                }
33            }
34        }
35    }
36    return true;
37 }
38 int main()
39 {
40     // add(a, b, k) means x_a + k <= x_b
41     // PROCESS
42 }

```

9.8 LCA

```

1 int n, m, h[N], e[M], ne[M], idx;
2 int depth[N], fa[N][16], q[N];
3 void bfs(int root)
4 {
5     memset(depth, 0x3f, sizeof depth);
6     depth[0] = 0;
7     depth[root] = 1;
8     int hh = 0, tt = 0;
9     q[0] = root;
10    while (hh <= tt)
11    {
12        int t = q[hh++];
13        for (int i = h[t]; ~i; i = ne[i])
14        {
15            int j = e[i];
16            if (depth[j] > depth[t] + 1)
17            {
18                depth[j] = depth[t] + 1;
19                q[++tt] = j;
20                fa[j][0] = t;
21                for (int k = 1; k <= 15; k++)
22                    fa[j][k] = fa[fa[j][k-1]][k-1];
23            }
24        }
25    }
26 }
27 int lca(int a, int b)
28 {
29     if (depth[a] < depth[b])
30         swap(a, b);
31     for (int k = 15; k >= 0; k--)
32         if (depth[fa[a][k]] >= depth[b])
33             a = fa[a][k];
34     if (a == b)
35         return a;
36     for (int k = 15; k >= 0; k--)
37         if (fa[a][k] != fa[b][k])
38         {
39             a = fa[a][k];
40             b = fa[b][k];
41         }
42     return fa[a][0];
43 }

```

9.9 SCC

```

1 void tarjan(int u)
2 {
3     dfn[u] = low[u] = ++timestamp;
4     stack[++top] = u, in_stk[u] = true;
5     for (int i = h[u]; ~i; i = ne[i])
6     {
7         int j = e[i];
8         if (!dfn[j])
9         {
10            tarjan(j);
11            low[u] = min(low[u], low[j]);
12        }

```

```

13         else if(in_stk[j])
14             low[u] = min(low[u], dfn[j]);
15     }
16     if(dfn[u] == low[u])
17     {
18         int y;
19         ++scc_cnt;
20         do
21         {
22             y = stk[top--];
23             in_stk[y] = false;
24             id[y] = scc_cnt;
25         } while(y != u);
26     }
27 }

```

9.10 DCC

9.10.1 e-DCC

```

1  const int N = 5010, M = 20010;
2  int n, m, h[N], e[M], ne[M], idx;
3  int dfn[N], low[N], timestamp;
4  int stk[N], top, id[N], dcc_cnt, d[N];
5  bool is_bridge[M];
6  void tarjan(int u, int from)
7  {
8      dfn[u] = low[u] = ++timestamp;
9      stk[++top] = u;
10     for (int i = h[u]; ~i; i = ne[i])
11     {
12         int j = e[i];
13         if (!dfn[j])
14         {
15             tarjan(j, i);
16             low[u] = min(low[u], low[j]);
17             if (dfn[u] < low[j])
18                 is_bridge[i] = is_bridge[i ^ 1] =
19                 true;
20         }
21         else if (i != (from ^ 1))
22             low[u] = min(low[u], dfn[j]);
23     }
24     if (dfn[u] == low[u])
25     {
26         ++dcc_cnt;
27         int y;
28         do
29         {
30             y = stk[top--];
31             id[y] = dcc_cnt;
32         } while (y != u);
33     }

```

9.10.2 v-DCC

```

1  const int N = 1010, M = 1010;
2  int n, m, h[N], e[M], ne[M], idx;
3  int dfn[N], low[N], timestamp;
4  int stk[N], top, dcc_cnt, root;

```

```

5  vector<int> dcc[N];
6  bool cut[N];
7  void init()
8  {
9      for (int i = 1; i <= dcc_cnt; i++)
10         dcc[i].clear();
11     idx = n = timestamp = top = dcc_cnt = 0;
12     memset(h, -1, sizeof h);
13     memset(dfn, 0, sizeof dfn);
14     memset(cut, 0, sizeof cut);
15 }
16 void tarjan(int u)
17 {
18     dfn[u] = low[u] = ++timestamp;
19     stk[++top] = u;
20     if (u == root && h[u] == -1)
21     {
22         dcc_cnt++;
23         dcc[dcc_cnt].push_back(u);
24         return;
25     }
26     int cnt = 0;
27     for (int i = h[u]; ~i; i = ne[i])
28     {
29         int j = e[i];
30         if (!dfn[j])
31         {
32             tarjan(j);
33             low[u] = min(low[u], low[j]);
34             if (dfn[u] <= low[j])
35             {
36                 cnt++;
37                 if (u != root || cnt > 1)
38                     cut[u] = true;
39                 ++dcc_cnt;
40                 int y;
41                 do
42                 {
43                     y = stk[top--];
44                     dcc[dcc_cnt].push_back(y);
45                 } while (y != j);
46                 dcc[dcc_cnt].push_back(u);
47             }
48         }
49         else
50             low[u] = min(low[u], dfn[j]);
51     }
52 }

```

9.11 Bipartite Graph

The maximum matching
(by the Hungarian algorithm) =
the minimum vertex cover =
total number of vertices -
maximum independent set =
total number of vertices -
minimum path cover.

9.11.1 maximum matching

```

1  const int N = 110;

```

```

2  int n, m;
3  int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};
4  PII match[N][N];
5  bool g[N][N], st[N][N];
6  bool find(int x, int y)
7  {
8      for (int i = 0; i < 4; i++)
9      {
10         int a = x + dx[i], b = y + dy[i];
11         if (a && a <= n && b && b <= n && !g[a][b] && !st[a][b])
12         {
13             st[a][b] = true;
14             PII t = match[a][b];
15             if (t.x == -1 || find(t.x, t.y))
16             {
17                 match[a][b] = {x, y};
18                 return true;
19             }
20         }
21     }
22     return false;
23 }
24 int main()
25 {
26     // PROCESS
27     memset(match, -1, sizeof match);
28     int res = 0;
29     for (int i = 1; i <= n; i++)
30         for (int j = 1; j <= n; j++)
31             if ((i + j) % 2 && !g[i][j])
32             {
33                 memset(st, 0, sizeof st);
34                 if (find(i, j)) res++;
35             }
36     // PROCESS
37 }

```

9.11.2 minimum vertex cover

```

1  const int N = 110;
2  int n, m, k, match[N];
3  bool g[N][N], st[N];
4  bool find(int x)
5  {
6      for (int i = 0; i < m; i++)
7          if (!st[i] && g[x][i])
8          {
9              st[i] = true;
10             if (match[i] == -1 || find(match[i]))
11             {
12                 match[i] = x;
13                 return true;
14             }
15         }
16     return false;
17 }
18 int main()
19 {
20     while (cin >> n, n)
21     {
22         cin >> m >> k;

```

```

23         memset(g, 0, sizeof g);
24         memset(match, -1, sizeof match);
25         while (k--)
26         {
27             int t, a, b;
28             cin >> t >> a >> b;
29             if (!a || !b) continue;
30             g[a][b] = true;
31         }
32         int res = 0;
33         for (int i = 0; i < n; i++)
34         {
35             memset(st, 0, sizeof st);
36             if (find(i)) res++;
37         }
38         cout << res << '\n';
39     }
40 }

```

9.11.3 maximum independent set

```

1  const int N = 110;
2  int n, m, k;
3  PII match[N][N];
4  bool g[N][N], st[N][N];
5  int dx[8] = {-2, -1, 1, 2, 2, 1, -1, -2};
6  int dy[8] = {1, 2, 2, 1, -1, -2, -2, -1};
7  bool find(int x, int y)
8  {
9      for (int i = 0; i < 8; i++)
10      {
11         int a = x + dx[i], b = y + dy[i];
12         if (a < 1 || a > n || b < 1 || b > m)
13             continue;
14         if (g[a][b]) continue;
15         if (st[a][b]) continue;
16         st[a][b] = true;
17         PII t = match[a][b];
18         if (t.x == 0 || find(t.x, t.y))
19         {
20             match[a][b] = {x, y};
21             return true;
22         }
23     }
24     return false;
25 }
26 int main()
27 {
28     // PROCESS
29     int res = 0;
30     for (int i = 1; i <= n; i++)
31         for (int j = 1; j <= m; j++)
32         {
33             if (g[i][j] || (i + j) % 2)
34                 continue;
35             memset(st, 0, sizeof st);
36             if (find(i, j)) res++;
37         }
38     cout << n * m - k - res << '\n';
39 }

```

9.11.4 minimum path cover

- Only for DAG.
- If you need to compute the **minimum path cover with repeated nodes**, you need to perform transitive closure as shown in the following code.

```
1  const int N = 210, M = 30010;
2  int n, m, match[N];
3  bool d[N][N], st[N];
4  bool find(int x)
5  {
6      for (int i = 1; i <= n; i++)
7          if (d[x][i] && !st[i])
8              {
9                  st[i] = true;
10                 int t = match[i];
11                 if (t == 0 || find(t))
12                     {
13                         match[i] = x;
14                         return true;
15                     }
16             }
17     return false;
18 }
19 int main()
20 {
21     // 传递闭包
22     for (int k = 1; k <= n; k++)
23         for (int i = 1; i <= n; i++)
24             for (int j = 1; j <= n; j++)
25                 d[i][j] |= d[i][k] & d[k][j];
26     int res = 0;
27     for (int i = 1; i <= n; i++)
28     {
29         memset(st, 0, sizeof st);
30         if (find(i)) res++;
31     }
32     cout << n - res;
33 }
```

9.12 Eulerian Circuit & Eulerian Path

9.12.1 Eulerian Circuit

```
1  int type, n, m;
2  int h[N], e[M], ne[M], idx;
3  bool used[M];
4  int ans[M], cn, din[N], dout[N];
5  void add(int a, int b)
6  {
7      e[idx] = b, ne[idx] = h[a], h[a] = idx++;
8  }
9  void dfs(int u)
10 {
11     for (int &i = h[u]; ~i;)
12     {
```

```
13         if (used[i])
14             { i = ne[i]; continue; }
15         used[i] = true;
16         if (type == 1) used[i ^ 1] = true;
17         int t;
18         if (type == 1)
19             {
20                 t = i / 2 + 1;
21                 if (i & 1) t = -t;
22             }
23         else t = i + 1;
24         int j = e[i];
25         i = ne[i];
26         dfs(j);
27         ans[++cnt] = t;
28     }
29 }
30 int main()
31 {
32     cin >> type >> n >> m;
33     memset(h, -1, sizeof h);
34     for (int i = 0; i < m; i++)
35     {
36         int a, b;
37         cin >> a >> b;
38         add(a, b);
39         if (type == 1) add(b, a);
40         din[b]++, dout[a]++;
41     }
42     if (type == 1)
43     {
44         for (int i = 1; i <= n; i++)
45             if (din[i] + dout[i] & 1)
46                 {
47                     cout << "NO\n";
48                     return 0;
49                 }
50     }
51     else
52     {
53         for (int i = 1; i <= n; i++)
54             if (din[i] != dout[i])
55                 {
56                     cout << "NO\n";
57                     return 0;
58                 }
59     }
60     for (int i = 1; i <= n; i++)
61         if (h[i] != -1) { dfs(i); break; }
62 }
```

9.12.2 Eulerian Path

```
1  const int N = 510;
2  int n = 500, m, g[N][N];
3  int ans[1100], cnt, d[N];
4  void dfs(int u)
5  {
6      for (int i = 1; i <= n; i++)
7          if (g[u][i])
8              {
9                  g[u][i]--, g[i][u]--;
10                 dfs(i);
11             }
```

```
12     ans[++cnt] = u;
13 }
14 int main()
15 {
16     cin >> m;
17     while (m--)
18     {
19         int a, b;
20         cin >> a >> b;
21         g[a][b]++, g[b][a]++;
22         d[a]++, d[b]++;
23     }
24     int start = 1;
25     while (!d[start]) ++start;
26     for (int i = 1; i <= 500; i++)
27         if (d[i] % 2)
28             { start = i; break; }
29     dfs(start);
30 }
```

10 ★ Advanced Math

10.1 Euler's Totient Function

10.1.1 GCD

```
1  const int N = 1e7 + 10;
2  int primes[N], cnt, phi[N];
3  bool st[N];
4  LL s[N];
5  void init(int n)
6  {
7      for (int i = 2; i <= n; i++)
8      {
9          if (!st[i])
10             {
11                 primes[cnt++] = i;
12                 phi[i] = i - 1;
13             }
14             for (int j = 0; primes[j] * i <= n;
15                 j++)
16             {
17                 st[primes[j] * i] = true;
18                 if (i % primes[j] == 0)
19                     {
20                         phi[i * primes[j]] = phi[i]
21                         * primes[j];
22                         break;
23                     }
24                     phi[i * primes[j]] = phi[i] * (
25                     primes[j] - 1);
26             }
27     }
28     for (int i = 1; i <= n; i++)
29         s[i] = s[i - 1] + phi[i];
30 }
31 int main()
32 {
33     int n; cin >> n;
34     init(n);
35     LL res = 0;
36     for (int i = 0; i < cnt; i++)
37     {
38         int p = primes[i];
39         res += s[n / p] * 2 + 1;
40     }
41 }
```

```
11 void mul(int c[][N], int a[][N], int b[][N])
12 {
13     int temp[N][N] = {0};
14     for (int i = 0; i < N; i++)
15         for (int j = 0; j < N; j++)
16             for (int k = 0; k < N; k++)
17                 temp[i][j] = (temp[i][j] + (
18                 LL)a[i][k] * b[k][j]) % m;
19     memcpy(c, temp, sizeof temp);
20 }
21 int main()
22 {
23     while (n)
24     {
25         if (n & 1) mul(f1, f1, a);
26         mul(a, a, a); n >>= 1;
27     }
28 }
```

10.2 Matrix Multiplication

```
1  const int N = 3;
2  int n, m;
3  void mul(int c[], int a[], int b[][N])
4  {
5      int temp[N] = {0};
6      for (int i = 0; i < N; i++)
7          for (int j = 0; j < N; j++)
8              temp[i] = (temp[i] + (LL)a[j] *
9              b[j][i]) % m;
10     memcpy(c, temp, sizeof temp);
11 }
```

11 ★ Advanced DP

11.1 Advanced LIS

11.1.1 MSIS

MSIS means Maximum Sum Increasing Subsequence

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

11.1.2 LCIS

LCIS means Longest Common Increasing Subsequence

```
1  const int N = 3010;
2  int n, a[N], b[N], f[N][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          cin >> b[i];
10     for (int i = 1; i <= n; i++)
11     {
12         int maxv = 1;
13         for (int j = 1; j <= n; j++)
14         {
15             f[i][j] = f[i - 1][j];
16             if (a[i] == b[j])
17                 f[i][j] = max(f[i][j], maxv);
18             ;
19             if (a[i] > b[j])
20                 maxv = max(maxv, f[i - 1][j]
21                             + 1);
22         }
23     }
24     int res = 0;
25     for (int i = 1; i <= n; i++)
26         res = max(res, f[n][i]);
27     cout << res;
```

11.2 Knapsack Problem

11.2.1 Multiple Knapsack Problem

```
1  const int N = 20010;
2  int n, m, f[N], g[N], q[N];
3  int main()
4  {
5      cin >> n >> m;
6      for (int i = 0; i < n; i++)
7      {
8          int v, w, s;
9          cin >> v >> w >> s;
10         memcpy(g, f, sizeof f);
11         for (int j = 0; j < v; j++)
12         {
13             int hh = 0, tt = -1;
14             for (int k = j; k <= m; k += v)
15             {
16                 if (hh <= tt && q[hh] < k -
17                     s * v)
18                     hh++;
19                 while (hh <= tt && g[q[tt]]
20                     - (q[tt] - j) / v * w <= g[k] - (k - j)
21                     / v * w)
22                     tt--;
23                 q[++tt] = k;
24                 f[k] = g[q[hh]] + (k - q[hh]
25                     ) / v * w;
26             }
27         }
28     }
29     cout << f[m] << '\n';
```

11.2.2 Two-Dimensional Cost Knapsack Problem

```
1  const int N = 110;
2  int n, V, M, f[N][N];
3  int main()
4  {
5      cin >> n >> V >> M;
6      for (int i = 0; i < n; i++)
7      {
8          int v, m, w;
9          cin >> v >> m >> w;
10         for (int j = V; j >= v; j--)
11             for (int k = M; k >= m; k--)
12                 f[j][k] = max(f[j][k], f[j -
13                     v][k - m] + w);
14     }
15     cout << f[V][M] << '\n';
```

11.2.3 Finding the Actual Solution Set

```
1  const int N = 1010;
2  int n, m;
3  int v[N], w[N], f[N][N];
```

```

4 int main()
5 {
6     cin >> n >> m;
7     for (int i = 1; i <= n; i++)
8         cin >> v[i] >> w[i];
9     for (int i = n; i >= 1; i--)
10        for (int j = 0; j <= m; j++)
11            {
12                f[i][j] = f[i + 1][j];
13                if (j >= v[i])
14                    f[i][j] = max(f[i][j], f[i + 1][j - v[i]] + w[i]);
15            }
16    int j = m;
17    for (int i = 1; i <= n; i++)
18        if (j >= v[i] && f[i][j] == f[i + 1][j - v[i]] + w[i])
19            {
20                cout << i << ' ';
21                j -= v[i];
22            }
23 }

```

11.2.4 Maximum Linearly Independent Subset

```

1 const int N = 110, M = 25010;
2 int n, v[N];
3 bool f[M];
4 int main()
5 {
6     int T; cin >> T;
7     while (T--)
8     {
9         cin >> n;
10        for (int i = 1; i <= n; ++i)
11            cin >> v[i];
12        sort(v + 1, v + n + 1);
13        int m = v[n], res = 0;
14        memset(f, 0, sizeof f);
15        f[0] = true; // 状态的初值
16        for (int i = 1; i <= n; ++i)
17            {
18                if (f[v[i]]) continue;
19                res++;
20                for (int j = v[i]; j <= m; ++j)
21                    f[j] |= f[j - v[i]];
22            }
23        cout << res << '\n';
24    }
25 }

```

11.2.5 Mixed Knapsack Problem

```

1 const int N = 1010;
2 int n, m, f[N];
3 int main()
4 {
5     cin >> n >> m;
6     for (int i = 0; i < n; i++)
7     {

```

```

8         int v, w, s;
9         cin >> v >> w >> s;
10        if (!s)
11            {
12                for (int j = v; j <= m; j++)
13                    f[j] = max(f[j], f[j - v] + w);
14            }
15        else
16            {
17                if (s == -1)
18                    s = 1;
19                for (int k = 1; k <= s; k *= 2)
20                {
21                    for (int j = m; j >= k * v; j--)
22                        f[j] = max(f[j], f[j - k * v] + k * w);
23                    s -= k;
24                }
25                if (s)
26                {
27                    for (int j = m; j >= s * v; j--)
28                        f[j] = max(f[j], f[j - s * v] + s * w);
29                }
30            }
31        cout << f[m] << '\n';
32    }
33 }

```

11.2.6 Dependent Knapsack Problem

```

1 const int N = 110;
2 int n, m, root;
3 int h[N], e[N], ne[N], idx;
4 int v[N], w[N], f[N][N];
5 void add(int a, int b)
6 {
7     e[idx] = b, ne[idx] = h[a], h[a] = idx++;
8 }
9 void dfs(int u)
10 {
11     for (int i = h[u]; ~i; i = ne[i])
12     {
13         int son = e[i];
14         dfs(son);
15         for (int j = m - v[u]; j >= 0; --j)
16             for (int k = 0; k <= j; ++k)
17                 f[u][j] = max(f[u][j], f[u][j - k] + f[son][k]);
18     }
19     for (int j = m; j >= v[u]; --j)
20         f[u][j] = f[u][j - v[u]] + w[u];
21     for (int j = 0; j < v[u]; ++j)
22         f[u][j] = 0;
23 }
24 int main()
25 {
26     memset(h, -1, sizeof h);
27     cin >> n >> m;
28     for (int i = 1; i <= n; ++i)

```



```

29     {
30         int p;
31         cin >> v[i] >> w[i] >> p;
32         if (p == -1) root = i;
33         else add(p, i);
34     }
35     dfs(root);
36     cout << f[root][m] << '\n';
37 }

```

11.2.7 Number of Solutions

```

1  const int N = 1010, mod = 1e9 + 7;
2  int n, m;
3  int w[N], v[N], f[N], g[N];
4  int main()
5  {
6      cin >> n >> m;
7      for (int i = 1; i <= n; ++i)
8          cin >> v[i] >> w[i];
9      g[0] = 1;
10     for (int i = 1; i <= n; ++i)
11     {
12         for (int j = m; j >= v[i]; --j)
13         {
14             int temp = max(f[j], f[j - v[i]]
15 + w[i]), c = 0;
16             if (temp == f[j])
17                 c = (c + g[j]) % mod;
18             if (temp == f[j - v[i]] + w[i])
19                 c = (c + g[j - v[i]]) % mod;
20             f[j] = temp, g[j] = c;
21         }
22     }
23     int res = 0;
24     for (int j = 0; j <= m; ++j)
25         if (f[j] == f[m])
26             res = (res + g[j]) % mod;
27     cout << res << '\n';
28 }

```

11.3 FSM

```

1  const int N = 100010;
2  int n, w[N], f[N][2];
3  int main()
4  {
5      int T; cin >> T;
6      while (T--)
7      {
8          cin >> n;
9          for (int i = 1; i <= n; ++i)
10             cin >> w[i];
11         for (int i = 1; i <= n; ++i)
12         {
13             // YOUR_FSM_RULES
14             // f[i][0] =
15             // f[i][1] =

```

```

16         }
17         cout << max(f[n][0], f[n][1]) << '\n';
18     }
19 }

```

11.4 Digit DP

```

1  const int N = 35;
2  int l, r, k, b, a[N], al, f[N][N];
3  int dp(int pos, int st, int op)
4  {
5      if (!pos) return st == k;
6      if (!op && ~f[pos][st])
7          return f[pos][st];
8      int res = 0, maxx = op ? min(a[pos], 1)
9 : 1;
10     for (int i = 0; i <= maxx; ++i)
11     {
12         if (st + i > k) continue;
13         res += dp(pos - 1, st + i, op && i
14 == a[pos]);
15     }
16     return op ? res : f[pos][st] = res;
17 }
18 int calc(int x)
19 {
20     al = 0;
21     memset(f, -1, sizeof f);
22     while (x) a[++al] = x % b, x /= b;
23     return dp(al, 0, 1);
24 }
25 int main()
26 {
27     cin >> l >> r >> k >> b;
28     cout << calc(r) - calc(l - 1) << '\n';
29 }

```

11.5 Queue Optimization for DP

```

1  int n, m, s[300010], q[300010];
2  int main()
3  {
4      cin >> n >> m;
5      for (int i = 1; i <= n; ++i)
6          cin >> s[i], s[i] += s[i - 1];
7      int res = INT_MIN, hh = 0, tt = 0;
8      for (int i = 1; i <= n; ++i)
9      {
10         if (q[hh] < i - m) hh++;
11         res = max(res, s[i] - s[q[hh]]);
12         while (hh <= tt && s[q[tt]] >= s[i])
13             tt--;
14         q[++tt] = i;
15     }

```