ACM-ICPC Team Reference Document
Tula State University (Fursov, Perezyabov, Vasin)

Contents

1 Templates **1**
- 1.1 Cpp Template . . . 1
- 1.2 Cpp Includes . . . 2
- 1.3 Py Template . . . 2

2 Data Structures **2**
- 2.1 Disjoint Set Union . . . 2
- 2.2 Segtree Sum . . . 2
- 2.3 Segtree Min Count . . . 2
- 2.4 Segtree First Above . . . 3
- 2.5 Segtree First Above Left . . . 3
- 2.6 Segtree K Ones . . . 3
- 2.7 Segtree Intersecting Segments . . . 3
- 2.8 Segtree Max Sum . . . 3
- 2.9 Segrtree Nested Segments . . . 4
- 2.10 Segtree Inversions . . . 4
- 2.11 Segtree Inversions II . . . 4
- 2.12 Segtree Seg Adding . . . 4
- 2.13 Segtree Seg Adding Diff . . . 4
- 2.14 Segtree Addsum . . . 4
- 2.15 Segment Tree With Lazy Propagation . . . 5
- 2.16 Segtree Propagate . . . 5
- 2.17 Segtree Propagatesum . . . 5

3 Graphs **6**
- 3.1 Articulation Point . . . 6
- 3.2 Bfs . . . 6
- 3.3 Bridges . . . 6
- 3.4 Components Of Strong Connectivity . . . 7
- 3.5 Connected Component . . . 7
- 3.6 Cycles . . . 7
- 3.7 Eulerian Cycle Path . . . 8
- 3.8 Dijkstra . . . 8
- 3.9 Prim . . . 8
- 3.10 Kruscal . . . 9
- 3.11 Topological Sort . . . 9
- 3.12 Dfs With Timestamps . . . 9
- 3.13 Bellman Ford Algorithm . . . 9
- 3.14 Lowest Common Ancestor . . . 10
- 3.15 Bipartite Graph . . . 10
- 3.16 Floyd's Algorithm . . . 11
- 3.17 Max Flow With Dinic . . . 11
- 3.18 Kuhn . . . 11

4 Geometry **12**
- 4.1 Graham . . . 12
- 4.2 2d Vector . . . 12
- 4.3 Line . . . 12
- 4.4 Circle Line Intersection . . . 13
- 4.5 7zip Cord . . . 13
- 4.6 Formulae . . . 13

5 Algebra **14**
- 5.1 Combinations . . . 14
- 5.2 Eratosthenes . . . 14
- 5.3 Fft . . . 15
- 5.4 Fibonacci . . . 15
- 5.5 Gcd . . . 16
- 5.6 Extended Euclidean Algorithm . . . 16
- 5.7 Euler Totient Function . . . 16
- 5.8 Factorization . . . 16
- 5.9 Binary Mult Pow . . . 16
- 5.10 Matrices . . . 17
- 5.11 Catalan . . . 17
- 5.12 Formulae . . . 17

6 Strings **17**
- 6.1 Manaker . . . 17
- 6.2 Suffixarray . . . 17
- 6.3 Z Function . . . 18
- 6.4 Bor . . . 18

7 Dynamic Programming **19**
- 7.1 Backpack . . . 19
- 7.2 Coins . . . 19
- 7.3 Increasing Sequence . . . 19
- 7.4 Palindromes . . . 19
- 7.5 Pyramid . . . 20
- 7.6 Ribbon . . . 20
- 7.7 Route . . . 20
- 7.8 Upstairs . . . 20

8 Misc **21**
- 8.1 Ternary Search . . . 21

1 Templates

1.1 Cpp Template

```
#include<...>
using namespace std;
#define fast cin.tie(0); cout.tie(0); ios::sync_with_stdio
    (0);
#define all(x) x.begin(), x.end()
#define rall(x) x.rbegin(), x.rend()
#define ff first
#define ss second
#define nl "\n"
#define sp " "
```

```
#define yes "YES"
#define no "NO"
#define bool_out(x) (x ? yes : no)
#define ll long long
#define int ll
const int inf = 1e18;
const int mod = 1e9 + 7;
const int pow_mod = 1e9 + 6;
void solve() {}
signed main() {
    int t = 1;
    while (t--)
        solve();
}
```

1.2 Cpp Includes

```
#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <climits>
#include <cstring>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <algorithm>
#include <vector>
#include <set>
#include <unordered_set>
#include <map>
#include <unordered_map>
#include <queue>
#include <stack>
#include <string>
#include <list>
#include <bitset>
#include <sstream>
#include <functional>
#include <complex>
#include <random>
```

1.3 Py Template

```
from math import ceil, log, sqrt, floor
def solve():
    pass
def main():
    t = 1
    t = int(input())
    for _ in range(t):
        solve()
```

2 Data Structures

2.1 Disjoint Set Union

```
struct dsu {
    vector<int> p, size;
    dsu(int n) {
        p.assign(n, 0); size.assign(n, 0);
        for (int i = 0; i < n; i++) {
            p[i] = i;
            size[i] = 1;
        }
    }
    int get(int v) {
        if (p[v] != v) p[v] = get(p[v]);
        return p[v];
    }
    void unite(int u, int v) {
        auto x = get(u), y = get(v);
        if (size[x] > size[y]) swap(x, y);
        p[x] = y;
        size[x] += size[y];
    }
};
```

2.2 Segtree Sum

```
struct sum_tree {
    vector<int> tree;
    int size;
    void init(int n) {
        size = 1;
        while (size < n) size <<= 1;
        tree.assign(2 * size - 1, 0);
    }
    void build(vector<int> &a, int x, int lx, int rx) {
        if (rx - lx == 1) {
            if (lx < a.size())
                tree[x] = a[lx];
            return;
        }
        int m = (lx + rx) / 2;
        build(a, 2 * x + 1, lx, m);
        build(a, 2 * x + 2, m, rx);
        tree[x] = tree[2 * x + 1] + tree[2 * x + 2];
    }
    void build(vector<int> &a) {
        init(a.size());
        build(a, 0, 0, size);
    }
    void set(int i, int v, int x, int lx, int rx) {
        if (rx - lx == 1) {
            tree[x] = v;
            return;
        }
        int m = (lx + rx) / 2;
        if (i < m) set(i, v, 2 * x + 1, lx, m);
        else set(i, v, 2 * x + 2, m, rx);
        tree[x] = tree[2 * x + 1] + tree[2 * x + 2];
    }
    void set(int i, int v) {
        set(i, v, 0, 0, size);
    }
    int sum(int l, int r, int x, int lx, int rx) {
        if (rx <= l || r <= lx) return 0;
        if (l <= lx && rx <= r) return tree[x];
        int m = (lx + rx) / 2;
        int sum1 = sum(l, r, 2 * x + 1, lx, m);
        int sum2 = sum(l, r, 2 * x + 2, m, rx);
        return sum1 + sum2;
    }
    int sum(int l, int r) {
        return sum(l, r, 0, 0, size);
    }
};
```

2.3 Segtree Min Count

```
struct min_count_tree {
    struct node {
        int min;
        int count;
    };
    node combine(node a, node b) {
        if (a.min < b.min) return a;
        if (a.min > b.min) return b;
        return { a.min, a.count + b.count };
    }
    const node zero = { inf, 0 };
    vector<node> tree;
    int size;
    void init(int n) {
        size = 1;
        while (size < n) size <<= 1;
        tree.assign(2 * size - 1, { 0, 0 });
    }
    void build(vector<int> &a, int x, int lx, int rx) {
        if (rx - lx == 1) {
            if (lx < a.size())
                tree[x] = { a[lx], 1 };
            return;
        }
```

```
        int m = (lx + rx) / 2;
        build(a, 2 * x + 1, lx, m);
        build(a, 2 * x + 2, m, rx);
        tree[x] = combine(tree[2 * x + 1], tree[2 * x + 2]);
    }
    void build(vector<int> &a) {
        init(a.size());
        build(a, 0, 0, size);
    }
    void set(int i, int v, int x, int lx, int rx) {
        if (rx - lx == 1) {
            tree[x] = { v, 1 };
            return;
        }
        int m = (lx + rx) / 2;
        if (i < m) set(i, v, 2 * x + 1, lx, m);
        else set(i, v, 2 * x + 2, m, rx);
        tree[x] = combine(tree[2 * x + 1], tree[2 * x + 2]);
    }
    void set(int i, int v) {
        set(i, v, 0, 0, size);
    }
    node calc(int l, int r, int x, int lx, int rx) {
        if (rx <= l || r <= lx) return zero;
        if (l <= lx && rx <= r) return tree[x];
        int m = (lx + rx) / 2;
        node calc1 = calc(l, r, 2 * x + 1, lx, m);
        node calc2 = calc(l, r, 2 * x + 2, m, rx);
        return combine(calc1, calc2);
    }
    node calc(int l, int r) {
        return calc(l, r, 0, 0, size);
    }
};
```

2.4 Segtree First Above

```
struct first_above_tree {
    //tree_max
    int first_above(int v, int x, int lx, int rx) {
        if (tree[x] < v) return -1;
        if (rx - lx == 1) return lx;
        int m = (lx + rx) / 2;
        int res = first_above(v, 2 * x + 1, lx, m);
        if (res == -1) res = first_above(v, 2 * x + 2, m, rx
            );
        return res;
    }
    int first_above(int v) {
        return first_above(v, 0, 0, size);
    }
};
```

2.5 Segtree First Above Left

```
struct first_above_left_tree {
    //tree_max
    int first_above(int v, int l, int x, int lx, int rx) {
        if (tree[x] < v || rx <= l) return -1;
        if (rx - lx == 1) return lx;
        int m = (lx + rx) / 2;
        int res = first_above(v, l, 2 * x + 1, lx, m);
        if (res == -1) res = first_above(v, l, 2 * x + 2, m,
            rx);
        return res;
    }
    int first_above(int v, int l) {
        return first_above(v, l, 0, 0, size);
    }
};
```

2.6 Segtree K Ones

```
struct k_ones_tree {
    //tree_sum
    int find(int k, int x, int lx, int rx) {
        if (rx - lx == 1) return lx;
        int m = (rx + lx) / 2;
        if (k < tree[2 * x + 1]) return find(k, 2 * x + 1,
            lx, m);
        else return find(k - tree[2 * x + 1], 2 * x + 2, m,
            rx);
    }
    int find(int k) {
        return find(k, 0, 0, size);
    }
};
```

2.7 Segtree Intersecting Segments

```
struct sum_tree {};
signed main() {
    sum_tree tr;
    int n; cin >> n;
    tr.init(2 * n);
    vector<int> pos(n, -1), ans(n, 0), A(2 * n);
    for (int i = 0; i < 2 * n; i++) cin >> A[i];
    for (int i = 0; i < 2 * n; i++) {
        int a = A[i] - 1;
        if (pos[a] == -1) {
            pos[a] = i;
            tr.set(pos[a], 1);
        }
        else {
            ans[a] = tr.sum(pos[a] + 1, i);
            tr.set(pos[a], 0);
            pos[a] = 0;
        }
    }
    pos.assign(n, -1); reverse(A.begin(), A.end());
    for (int i = 0; i < 2 * n; i++) {
        int a = A[i] - 1;
        if (pos[a] == -1) {
            pos[a] = i;
            tr.set(pos[a], 1);
        }
        else {
            ans[a] += tr.sum(pos[a] + 1, i);
            tr.set(pos[a], 0);
            pos[a] = 0;
        }
    }
    for (int i = 0; i < n; i++)
        cout << ans[i] << " ";
}
```

2.8 Segtree Max Sum

```
struct max_sum_tree {
    //tree_min_count
    struct node {
        long long seg, pref, suf, sum;
    };
    node one_element(int x) {
        return {
            max(x,0LL), //seg
            max(x,0LL), //pref
            max(x,0LL), //suf
            x //sum
        };
    }
    node combine(node a, node b) {
        return {
            /*seg*/ max(a.seg, max(b.seg, a.suf + b.pref)),
            /*pref*/ max(a.pref, a.sum + b.pref),
            /*suf*/ max(b.suf, b.sum + a.suf),
            /*sum*/ a.sum + b.sum
        };
    }
    const node zero = { 0, 0, 0, 0 };
};
```

2.9 Segrtree Nested Segments

```
struct tree_sum {};
signed main() {
   tree_sum tree;
   int n, m;
   cin >> n;
   tree.init(2 * n);
   vector<int> pos(n, -1), otv(n, 0);
   for (int i = 0; i < 2 * n; i++) {
      int a; cin >> a; --a;
      if (pos[a] == -1) {
         pos[a] = i;
      }
      else {
         otv[a] = tree.sum(pos[a], i);
         tree.set(pos[a], 1);
      }
   }
   for (int i = 0; i < n; i++) {
      cout << otv[i] << " ";
   }
}
```

2.10 Segtree Inversions

```
struct sum_tree { };
signed main() {
   sum_tree tree;
   int n; cin >> n;
   tree.init(n);
   for (int i = 0; i < n; i++) {
      int a; cin >> a;
      cout << tr.sum(a, n) << endl;
      tr.set(a - 1, 1);
   }
}
```

2.11 Segtree Inversions II

```
struct inversion_tree {
   // sum_tree
   int find(int k, int x, int lx, int rx) {
      if (rx == lx + 1) {
         return lx;
      }
      int m = (rx + lx) / 2;
      if (k < tree[2 * x + 2]) return find(k, 2 * x + 2, m
           , rx);
      else return find(k - tree[2 * x + 2], 2 * x + 1, lx,
            m);
   }
   int find(int k) {
      return find(k, 0, 0, size);
   }
};
signed main() {
   inversion_tree tree;
   int n, m;
   cin >> n;
   vector<int> A(n), P(n), E(n, 1);
   tree.build(E);
   for (int i = 0; i < n; i++) {
      cin >> A[i];
   }
   int pos = 0;
   for (int i = n - 1; i >= 0; i--) {
      pos = tree.find(A[i]);
      P[i] = pos + 1;
      tree.set(pos, 0);
   }
   for (int i = 0; i < n; i++) {
      cout << P[i] << " ";
   }
}
```

2.12 Segtree Seg Adding

```
struct seg_adding_tree_diff {
   long long get(int i, int x, int lx, int rx) {
      if (rx - lx == 1) return tree[x];
      int m = (lx + rx) / 2;
      if (i < m) return get(i, 2 * x + 1, lx, m) + tree[x
           ];
      else return get(i, 2 * x + 2, m, rx) + tree[x];
   }
   long long get(int i) {
      return get(i, 0, 0, size);
   }
   void add(int l, int r, int v, int x, int lx, int rx) {
      if (l >= rx || lx >= r) return;
      if (lx >= l && rx <= r) {
         tree[x] += v;
         return;
      }
      int m = (lx + rx) / 2;
      add(l, r, v, 2 * x + 1, lx, m);
      add(l, r, v, 2 * x + 2, m, rx);
   }
   void add(int l, int r, int v) {
      return add(l, r, v, 0, 0, size);
   }
};
```

2.13 Segtree Seg Adding Diff

```
struct seg_adding_tree_diff {
   // sum_tree + difference array
   void add(int i, int v, int x, int lx, int rx) {
      if (rx - lx == 1) {
         tree[x] += v;
         return;
      }
      int m = (lx + rx) / 2;
      if (i < m) add(i, v, 2 * x + 1, lx, m);
      else add(i, v, 2 * x + 2, m, rx);
      tree[x] = tree[2 * x + 1] + tree[2 * x + 2];
   }
   void add(int l, int r, int v) {
      add(l, v, 0, 0, size);
      add(r, -v, 0, 0, size);
      return;
   }
   long long get(int i) {
      return sum(0, i + 1);
   }
};
```

2.14 Segtree Addsum

```
struct addsum_tree {
   struct node {
      int set;
      int sum;
   };
   const int MOD = 1e9 + 7;
   const int NETRAL = 0;
   int operat_modify(int a, int b, int len) {
      return a + len * b;
   }
   int operat_min(int a, int b) {
      return (a + b);
   }
   void init(int n) {
      size = 1;
      while (size < n) {
         size *= 2;
      }
      tree.assign(2 * size - 1, { 0, 0 });
   }
   int suma(int l, int r, int x, int lx, int rx) {
      if (l >= rx || lx >= r) {
         return NETRAL;
      }
```

```
        if (lx >= l && rx <= r) {
            return tree[x].sum;
        }
        int m = (lx + rx) / 2;
        int m1 = suma(l, r, 2 * x + 1, lx, m);
        int m2 = suma(l, r, 2 * x + 2, m, rx);
        int res = operat_min(m1, m2);
        return operat_modify(res, tree[x].set, min(rx, r) -
             max(lx, l));
    }

    int suma(int l, int r) {
        return suma(l, r, 0, 0, size);
    }
    void modify(int l, int r, int v, int x, int lx, int rx)
         {
        if (l >= rx || lx >= r) {
            return;
        }
        if (lx >= l && rx <= r) {
            tree[x].set = operat_modify(tree[x].set, v, 1);
            tree[x].sum = operat_modify(tree[x].sum, v, (rx -
                 lx));
            return;
        }
        int m = (lx + rx) / 2;
        modify(l, r, v, 2 * x + 1, lx, m);
        modify(l, r, v, 2 * x + 2, m, rx);
        tree[x].sum = operat_min(tree[2 * x + 1].sum, tree[2
             * x + 2].sum);
        tree[x].sum = operat_modify(tree[x].sum, tree[x].set
             , (rx - lx));
    }
    void modify(int l, int r, int v) {
        return modify(l, r, v, 0, 0, size);
    }
};
```

2.15 Segment Tree With Lazy Propagation

```
// mass assignment
struct lazy_seg_tree {
    vector<int> tree, lazy;
    int size;
    init(int n) {
        size = 1;
        while (size < n) size <<= 1;
        tree.assign(2 * size - 1, 0);
        lazy.assign(2 * size - 1, 0);
    }
    void push(int x) {
        tree[2 * x + 1] = lazy[x];
        lazy[2 * x + 1] = lazy[x];
        tree[2 * x + 2] = lazy[x];
        lazy[2 * x + 2] = lazy[x];
        lazy[x] = -1;
    }
    void update(int v, int l, int r, int x, int lx, int rx)
         {
        if (rx <= l && r <= lx) return;
        if (l <= lx && rx <= r) {
            push(x);
            tree[x] = v;
            lazy[x] = v;
            return;
        }
        int m = (lx + rx) / 2;
        tree[x] = v;
        lazy[x] = v;
        update(v, l, r, 2 * x + 1, lx, m);
        update(v, l, r, 2 * x + 2, m, rx);
    }
    void update(int v, int l, int r) {
        update(v, l, r, 0, 0, size);
    }
    int get(int i, int x, int lx, int rx) {
        if (rx - lx == 1) return tree[x];
        int m = (lx + rx) / 2;
        if (i < m) get(i, 2 * x + 1, lx, m);
        else get(i, 2 * x + 2, m, rx);
    }
    int get(int i) {
        return get(i, 0, 0, size);
    }
};
```

2.16 Segtree Propagate

```
struct propagate_tree {
    long long get(int i, int x, int lx, int rx) {
        propagate(x, lx, rx);
        if (rx - lx == 1) return tree[x];
        int m = (lx + rx) / 2;
        if (i < m) return get(i, 2 * x + 1, lx, m);
        else return get(i, 2 * x + 2, m, rx);
    }
    long long get(int i) {
        return get(i, 0, 0, size);
    }
    void propagate(int x, int lx, int rx) {
        if (tree[x] == NO_OPERATION) return;
        if (rx - lx == 1) return;
        tree[2 * x + 1] = tree[x];
        tree[2 * x + 2] = tree[x];
        tree[x] = NO_OPERATION;
    }
    void modify(int l, int r, int v, int x, int lx, int rx)
         {
        propagate(x, lx, rx);
        if (l >= rx || lx >= r) return;
        if (lx >= l && rx <= r) {
            tree[x] = v;
            return;
        }
        int m = (lx + rx) / 2;
        modify(l, r, v, 2 * x + 1, lx, m);
        modify(l, r, v, 2 * x + 2, m, rx);
    }

    void modify(int l, int r, int v) {
        return modify(l, r, v, 0, 0, size);
    }
};
```

2.17 Segtree Propagatesum

```
struct propagatesum_tree {
    struct node {
        int set;
        int sum;
    };
    int MOD = 1e9 + 7;
    int NETRAL = 0;
    int NO_OPERATION = LLONG_MIN;
    int operat_modify(int a, int b, int len) {
        if (b == NO_OPERATION) return a;
        return b * len;
    }
    int operat_min(int a, int b) {
        return a + b;
    }
    void propagate(int x, int lx, int rx) {
        if (tree[x].set == NO_OPERATION || rx - lx == 1)
             return;
        int m = (lx + rx) / 2;
        tree[2 * x + 1].set = operat_modify(tree[2 * x + 1].
             set, tree[x].set, 1);
        tree[2 * x + 1].sum = operat_modify(tree[2 * x + 1].
             sum, tree[x].set, m - lx);
        tree[2 * x + 2].set = operat_modify(tree[2 * x + 1].
             set, tree[x].set, 1);
        tree[2 * x + 2].sum = operat_modify(tree[2 * x + 1].
             sum, tree[x].set, rx - m);
        tree[x].set = NO_OPERATION;
    }
    int suma(int l, int r, int x, int lx, int rx) {
        propagate(x, lx, rx);
        if (l >= rx || lx >= r) return NETRAL;
        if (lx >= l && rx <= r) return tree[x].sum;
        int m = (lx + rx) / 2;
        int m1 = suma(l, r, 2 * x + 1, lx, m);
        int m2 = suma(l, r, 2 * x + 2, m, rx);
```

```
        int res = operat_min(m1, m2);
        return res;
    }

    int suma(int l, int r) {
        return suma(l, r, 0, 0, size);
    }
    void modify(int l, int r, int v, int x, int lx, int rx)
         {
        propagate(x, lx, rx);
        if (l >= rx || lx >= r) return;
        if (lx >= l && rx <= r) {
            tree[x].set = operat_modify(tree[x].set, v, 1);
            tree[x].sum = operat_modify(tree[x].sum, v, (rx -
                  lx));
            return;
        }
        int m = (lx + rx) / 2;
        modify(l, r, v, 2 * x + 1, lx, m);
        modify(l, r, v, 2 * x + 2, m, rx);
        tree[x].sum = operat_min(tree[2 * x + 1].sum, tree[2
             * x + 2].sum);
    }

    void modify(int l, int r, int v) {
        return modify(l, r, v, 0, 0, size);
    }
};
```

3 Graphs

3.1 Articulation Point

```
vector<vector<int>> g;
vector<bool> used;
int timer = 0;
vector<int> tin, fup;
set<int> result;

void dfs(int v, int p = -1) {
    used[v] = true;
    tin[v] = fup[v] = timer++;
    int children = 0;

    for (size_t i = 0; i < g[v].size(); ++i) {
        int to = g[v][i];
        if (to == p) continue;
        if (used[to]) fup[v] = min(fup[v], tin[to]);
        else {
            dfs(to, v);
            fup[v] = min(fup[v], fup[to]);
            if (fup[to] >= tin[v] && p != -1) result.insert(v
                 );
            children++;
        }
    }

    if (p == -1 && children > 1) result.insert(v);
}

signed main() {
    int n, m, k;
    cin >> n >> m;

    g.resize(n);
    used.assign(n, false);
    tin.resize(n);
    fup.resize(n);

    for (int i = 0; i < m; i++) {
        int first, second;
        cin >> first >> second;
        first--; second--;
        g[first].push_back(second);
        g[second].push_back(first);
    }

    for (int i = 0; i < n; i++)
        dfs(i);

    for (auto it = result.begin(); it != result.end(); it++)
        cout << *it + 1 << " ";
}
```

3.2 Bfs

```
vector<vector<int>> v;
vector<int> u;
queue<int> q;

void bfs(int i, int n) {
    q.push(i);
    u[i] = 1;

    while (!q.empty()) {
        int j = q.front(); q.pop();

        for (auto &x : v[j]) {
            if (!u[x]) {
                u[x] = 1;
                q.push(x);
            }
        }

        cout << j << ' ';
    }
}

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

    v.resize(n);

    for (int i = 0; i < m; i++) {
        int x, y;
        cin >> x >> y;
        v[--x].push_back(--y);
        v[y].push_back(x);
    }

    for (int i = 0; i < n; i++) {
        u.assign(n, 0);
        bfs(i, n);
        cout << '\n';
    }
}
```

3.3 Bridges

```
vector<vector<int>> g;
vector<bool> used;
int timer = 0;
vector<int> tin, fup;
vector<pair<int, int>> result;

void dfs(int v, int p = -1) {
    used[v] = true;
    tin[v] = fup[v] = timer++;

    for (int i = 0; i < g[v].size(); i++) {
        int to = g[v][i];
        if (to == p) continue;
        if (used[to]) fup[v] = min(fup[v], tin[to]);
        else {
            dfs(to, v);
            fup[v] = min(fup[v], fup[to]);
            if (fup[to] > tin[v] && count(all(g[v]), to) ==
                 1)
                result.push_back({ min(v, to), max(to, v) });

        }
    }
}

void find_bridges(int n) {
    timer = 0;
    for (int i = 0; i < n; i++) {
        if (!used[i]) dfs(i);
    }
}
```

```
signed main() {
    int n;
    cin >> n;

    g.resize(n);
    used.assign(n, false);
    tin.resize(n);
    fup.resize(n);
    cin.ignore();

    for (int i = 0; i < n; i++) {
        int current = 0, count = 0;
        cin >> current >> count;
        for (int j = 0; j < count; j++) {
            int temp = 0;
            cin >> temp;
            g[current].push_back(temp);
        }
    }

    find_bridges(n);

    if (result.size()) {
        sort(all(result));
        for (int i = 0; i < result.size(); i++) {
            cout << result[i].first << " " << result[i].
                 second << endl;
        }
    }
    else cout << "Empty" << endl;
}
```

3.4 Components Of Strong Connectivity

```
vector < vector<int> > g, gr;
vector<char> used;
vector<int> order, component;

void dfs1(int v) {
    used[v] = true;
    for (size_t i = 0; i < g[v].size(); ++i)
        if (!used[g[v][i]]) dfs1(g[v][i]);
    order.push_back(v);
}

void dfs2(int v) {
    used[v] = true;
    component.push_back(v);
    for (size_t i = 0; i < gr[v].size(); ++i)
        if (!used[gr[v][i]]) dfs2(gr[v][i]);
}

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

    for (int i = 0; i < n; i++) {
        int a = 0, b = 0;
        cin >> a >> b;
        g[a].push_back(b);
        gr[b].push_back(a);
    }

    used.assign(n, false);
    for (int i = 0; i < n; ++i)
        if (!used[i]) dfs1(i);

    used.assign(n, false);
    for (int i = 0; i < n; ++i) {
        int v = order[n - 1 - i];
        if (!used[v]) {
            dfs2(v);
            for (int j = 0; j < component.size(); j++)
                cout << component[i] << " ";
            cout << '\n';
            component.clear();
        }
    }
}
```

3.5 Connected Component

```
void dfs(vector<vector<int>> &mass, vector<bool> &used, int
      vertex) {
    used[vertex] = true;
    for (int i = 0; i < mass[vertex].size(); i++)
        if (used[mass[vertex][i]] == false)
            dfs(mass, used, mass[vertex][i]);
}

signed main() {
    int n = 0, m = 0, second = 0, first = 0, result = 0;
    cin >> n >> m;
    vector<vector<int>> mass(n);
    vector<bool> used(n, false);

    for (int i = 0; i < m; i++) {
        cin >> first >> second;
        first--;
        second--;
        mass[first].push_back(second);
        mass[second].push_back(first);
    }

    for (int i = 0; i < n; i++) {
        if (!used[i]) {
            dfs(mass, used, i);
            result++;
        }
    }

    cout << result << '\n';
}
```

3.6 Cycles

```
int cycle_start = -1, cycle_end = 0;
vector<int> p;

bool dfs(vector<vector<int>> g, vector<bool> used, vector<
     int> color, int vertex) {
    color[vertex] = 1;
    for (int i = 0; i < g[vertex].size(); i++) {
        int to = g[vertex][i];
        if (color[to] == 0) {
            if (dfs(g, used, color, to)) {
                p[to] = vertex;
                return true;
            }
        }
        else if (color[to] == 1) {
            cycle_start = to;
            cycle_end = vertex;
            return true;
        }
    }
    color[vertex] = 2;
    return false;
}

signed main() {
    int n = 0, m = 0, second = 0, first = 0, req = 0;
    cin >> m >> n;
    vector<vector<int>> mass(n);
    vector<bool> used(n, false);
    vector<int> color(n, 0);
    vector<int> cycle;

    p.assign(n, -1);

    for (int i = 0; i < m; i++) {
        cin >> first >> second;
        first--;
        second--;
        mass[first].push_back(second);
    }

    for (int i = 0; i < n; i++)
        if (dfs(mass, used, color, i))
            break;

    if (cycle_start == -1)
```

```
        cout << "No" << endl;
    else {
        cout << "Yes" << endl;

        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = p[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);

        reverse(cycle.begin(), cycle.end());

        for (int i = 0; i < cycle.size(); i++)
            cout << cycle[i] + 1 << " ";
        cout << endl;
    }
}
```

3.7 Eulerian Cycle Path

```
signed main() {
    int n = 0;
    cin >> n;
    vector <vector<int>> g(n, vector<int>(n));
    vector<int> deg(n);

    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            deg[i] += g[i][j];

    int first = 0;
    while (!deg[first]) ++first;
    int v1 = -1, v2 = -1;
    bool bad = false;

    for (int i = 0; i < n; ++i)
        if (deg[i] & 1)
            if (v1 == -1) v1 = i;
            else if (v2 == -1) v2 = i;
            else bad = true;

    if (v1 != -1) {
        ++g[v1][v2];
        ++g[v2][v1];
    }

    stack<int> st;
    st.push(first);

    vector<int> res;

    while (!st.empty()) {
        int v = st.top();
        int i = 0;
        for (i = 0; i < n; ++i)
            if (g[v][i]) break;
        if (i == n) {
            res.push_back(v);
            st.pop();
        }
        else {
            --g[v][i];
            --g[i][v];
            st.push(i);
        }
    }

    if (v1 != -1) {
        for (size_t i = 0; i + 1 < res.size(); ++i) {
            if (res[i] == v1 && res[i + 1] == v2 || res[i] ==
                  v2 && res[i + 1] == v1) {
                vector<int> res2;
                for (size_t j = i + 1; j < res.size(); ++j)
                     res2.push_back(res[j]);
                for (size_t j = 1; j <= i; ++j) res2.push_back
                     (res[j]);
                res = res2;
                break;
            }
        }
    }

    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (g[i][j]) bad = true;

    if (bad)
        cout << -1 << endl;
    else
        for (size_t i = 0; i < res.size(); ++i)
            printf("%d ", res[i] + 1);
}
```

3.8 Dijkstra

```
signed main() {
    int inf = 1e18;
    int n = 0, m = 0;
    cin >> n >> m;
    vector<vector<pair<int, int>>> g(n);

    for (int i = 0; i < m; i++) {
        int to = 0, from = 0, len = 0;
        cin >> from >> to >> len;
        g[from - 1].push_back({ to - 1, len });
        g[to - 1].push_back({ from - 1, len });
    }

    vector<int> d(n, inf);
    vector<int> p(n);
    int from = 0, to = 0;
    d[0] = 0;
    priority_queue<pair<int, int>> q;
    q.push({ 0, 0 });

    while (!q.empty()) {
        int v = q.top().second;
        int cur_d = -q.top().first;
        q.pop();
        if (cur_d > d[v]) continue;
        for (int i = 0; i < g[v].size(); i++) {
            int to = g[v][i].first;
            int length = g[v][i].second;
            if (d[v] + length < d[to]) {
                d[to] = d[v] + length;
                p[to] = v;
                q.push({ -d[to], to });
            }

        }
    }

    if (d[n - 1] == inf) {
        cout << -1 << endl;
        return 0;
    }
    if (!d[n - 1]) {
        cout << 0 << endl;
        return 0;
    }
    vector<int> way;
    for (int v = n - 1; v != 0; v = p[v])
        way.push_back(v + 1);
    way.push_back(1);

    for (int i = way.size() - 1; i >= 0; i--)
        cout << way[i] << " ";
}
```

3.9 Prim

```
int main() {
    map <int, vector <pair<int, int>>> mass;
    vector <int> check;
    vector <int> result;
    vector <pair<int, int>> way;
    int n = 0, m = 0, third = 0, temp = 10e5, top = 10e5,
         sum = 0, count = 0, first = 0, second = 0, parent
         = 0, child = 0;
    cin >> n >> m;

    check.resize(n - 1);
    result.push_back(0);

    for (int i = 0; i < n - 1; i++)
```

```
        check[i] = i + 1;

    for (int i = 0; i < m; i++) {
        cin >> first >> second >> third;
        first--;
        second--;
        mass[first].push_back(make_pair(second, third));
        mass[second].push_back(make_pair(first, third));
    }

    while (!check.empty()) {
        for (int i = 0; i < result.size(); i++) {
            for (int j = 0; j < mass[result[i]].size(); j++)
                 {
                if (mass[result[i]][j].second < temp && find(
                     check.begin(), check.end(), mass[result[
                     i]][j].first) != check.end()) {
                    temp = mass[result[i]][j].second;
                    top = mass[result[i]][j].first;
                    parent = result[i];
                }
            }
        }

        result.push_back(top);
        for (int k = 0; k < check.size(); k++) {
            if (check[k] == top) {
                count = k;
                break;
            }
        }

        check.erase(check.begin() + count);
        sum += temp;
        count = 0;
        temp = 10e5;
        top = 10e5;
    }
    cout << sum << endl;
}
```

3.10 Kruscal

```
vector<vector<pair<int, int>>> mst;
vector<int> parent;
vector<pair<int, pair<int, int>>> G;

int findRoot(int v) {
    return parent[v] == v ? v : parent[v] = findRoot(parent[
        v]);
}

bool connected(int v1, int v2) {
    return findRoot(v1) == findRoot(v2);
}

void merge(int v1, int v2) {
    int r1 = findRoot(v1), r2 = findRoot(v2);
    if (r1 == r2)
        return;
    if (rand() % 2)
        parent[r1] = r2;
    else
        parent[r2] = r1;
}

signed main() {
    int n = 0, m = 0;
    cin >> n >> m;

    mst.resize(n);
    for (int i = 0; i < m; i++) {
        int v = 0, u = 0, cost = 0;
        cin >> v >> u >> cost;
        v--; u--;
        G.push_back({ cost, {v, u} });
    }

    int cost = 0;
    int all_sum = 0;

    sort(all(G));

    parent.resize(n);
    for (int i = 0; i < n; ++i)
        parent[i] = i;

    for (int i = 0; i < m; ++i) {
        int a = G[i].second.first, b = G[i].second.second, l
            = G[i].first;
        if (!connected(a, b)) {
            mst[G[i].second.first].push_back({ G[i].second.
                second, G[i].first });
            // mst[G[i].second.second].push_back({ G[i].
                second.first, G[i].first });
            merge(a, b);
            all_sum += G[i].first;
        }
    }

    cout << all_sum << endl;
    for (int i = 0; i < mst.size(); i++) {
        for (int j = 0; j < mst[i].size(); j++) {
            cout << i + 1 << " " << mst[i][j].first + 1 <<
                endl;
        }
    }
}
```

3.11 Topological Sort

```
vector<bool> used;
vector<int> ans;
vector<vector<int>>g;

void dfs(int v) {
    used[v] = true;
    for (size_t i = 0; i < g[v].size(); ++i) {
        int to = g[v][i];
        if (!used[to]) dfs(to);
    }
    ans.push_back(v);
}

void topological_sort(int n) {
    used.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i)
        if (!used[i]) dfs(i);
    reverse(ans.begin(), ans.end());
}

signed main() {
    int n; // числовершин
    cin >> n;
    used.assign(n, false);
    g.resize(n);
    topological_sort(n);
}
```

3.12 Dfs With Timestamps

```
vector<vector<int>> adj;
vector<int> tIn, tOut, color;
int dfs_timer = 0;

void dfs(int v) {
    tIn[v] = dfs_timer++;
    color[v] = 1;
    for (int u : adj[v])
        if (color[u] == 0)
            dfs(u);
    color[v] = 2;
    tOut[v] = dfs_timer++;
}
```

3.13 Bellman Ford Algorithm

```
struct Edge {
    int a, b, cost;
};

int n, m, v; // v - starting vertex
vector<Edge> e;

/* Finds SSSP with negative edge weights.
 * Possible optimization: check if anything changed in a
     relaxation step. If not - you can break early.
 * To find a negative cycle: perform one more relaxation
     step. If anything changes - a negative cycle exists.
 */
void solve() {
    vector<int> d(n, oo);
    d[v] = 0;
    for (int i = 0; i < n - 1; ++i)
        for (int j = 0; j < m; ++j)
            if (d[e[j].a] < oo)
                d[e[j].b] = min(d[e[j].b], d[e[j].a] + e[j].
                     cost);
}
```

3.14 Lowest Common Ancestor

```
int n, l; // l == logN (usually about ~20)
vector<vector<int>> adj;

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p) {
    tin[v] = ++timer;
    up[v][0] = p;
    // wUp[v][0] = weight[v][u]; // <- path weight sum to 2^
         i-th ancestor
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i - 1]][i - 1];
        // wUp[v][i] = wUp[v][i-1] + wUp[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

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

int lca(int u, int v) {
    if (isAncestor(u, v))
        return u;
    if (isAncestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!isAncestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

3.15 Bipartite Graph

```
class BipartiteGraph {
private:
    vector<int> _left, _right;
    vector<vector<int>> _adjList;
    vector<int> _matchR, _matchL;
    vector<bool> _used;

    bool _kuhn(int v) {
        if (_used[v]) return false;
        _used[v] = true;
        FOR(i, 0, (int) _adjList[v].size()) {
            int to = _adjList[v][i] - _left.size();
            if (_matchR[to] == -1 || _kuhn(_matchR[to])) {
                _matchR[to] = v;
                _matchL[v] = to;
                return true;
            }
        }
        return false;
    }
    void _addReverseEdges() {
        FOR(i, 0, (int) _right.size()) {
            if (_matchR[i] != -1) {
                _adjList[_left.size() + i].pb(_matchR[i]);
            }
        }
    }
    void _dfs(int p) {
        if (_used[p]) return;
        _used[p] = true;
        for (auto x : _adjList[p]) {
            _dfs(x);
        }
    }
    vector<pii> _buildMM() {
        vector<pair<int, int> > res;
        FOR(i, 0, (int) _right.size()) {
            if (_matchR[i] != -1) {
                res.push_back(make_pair(_matchR[i], i));
            }
        }

        return res;
    }
public:
    void addLeft(int x) {
        _left.pb(x);
        _adjList.pb({});
        _matchL.pb(-1);
        _used.pb(false);
    }
    void addRight(int x) {
        _right.pb(x);
        _adjList.pb({});
        _matchR.pb(-1);
        _used.pb(false);
    }
    void addForwardEdge(int l, int r) {
        _adjList[l].pb(r + _left.size());
    }
    void addMatchEdge(int l, int r) {
        if (l != -1) _matchL[l] = r;
        if (r != -1) _matchR[r] = l;
    }
    // Maximum Matching
    vector<pii> mm() {
        _matchR = vector<int>(_right.size(), -1);
        _matchL = vector<int>(_left.size(), -1);
        // ^ these two can be deleted if performing MM on
             already partially matched graph
        _used = vector<bool>(_left.size() + _right.size(),
             false);


        bool path_found;
        do {
            fill(_used.begin(), _used.end(), false);
            path_found = false;
            FOR(i, 0, (int) _left.size()) {
                if (_matchL[i] < 0 && !_used[i]) {
                    path_found |= _kuhn(i);
                }
            }
        }
        while (path_found);

        return _buildMM();
    }

    // Minimum Edge Cover
    // Algo: Find MM, add unmatched vertices greedily.
    vector<pii> mec() {
```

```
        auto ans = mm();
        FOR(i, 0, (int) _left.size()) {
            if (_matchL[i] != -1) {
                for (auto x : _adjList[i]) {
                    int ridx = x - _left.size();
                    if (_matchR[ridx] == -1) {
                        ans.pb({ i, ridx });
                        _matchR[ridx] = i;
                    }
                }
            }
        }
        FOR(i, 0, (int) _left.size()) {
            if (_matchL[i] == -1 && (int) _adjList[i].size()
                 > 0) {
                int ridx = _adjList[i][0] - _left.size();
                _matchL[i] = ridx;
                ans.pb({ i, ridx });
            }
        }
        return ans;
    }

    // Minimum Vertex Cover
    // Algo: Find MM. Run DFS from unmatched vertices from
         the left part.
    // MVC is composed of unvisited LEFT and visited RIGHT
         vertices.
    pair<vector<int>, vector<int>> mvc(bool runMM = true) {
        if (runMM) mm();
        _addReverseEdges();
        fill(_used.begin(), _used.end(), false);
        FOR(i, 0, (int) _left.size()) {
            if (_matchL[i] == -1) {
                _dfs(i);
            }
        }
        vector<int> left, right;
        FOR(i, 0, (int) _left.size()) {
            if (!_used[i]) left.pb(i);
        }
        FOR(i, 0, (int) _right.size()) {
            if (_used[i + (int) _left.size()]) right.pb(i);
        }
        return { left,right };
    }

    // Maximal Independant Vertex Set
    // Algo: Find complement of MVC.
    pair<vector<int>, vector<int>> mivs(bool runMM = true) {
        auto m = mvc(runMM);
        vector<bool> containsL(_left.size(), false),
             containsR(_right.size(), false);
        for (auto x : m.first) containsL[x] = true;
        for (auto x : m.second) containsR[x] = true;
        vector<int> left, right;
        FOR(i, 0, (int) _left.size()) {
            if (!containsL[i]) left.pb(i);
        }
        FOR(i, 0, (int) _right.size()) {
            if (!containsR[i]) right.pb(i);
        }
        return { left, right };
    }

};
```

3.16 Floyd's Algorithm

```
//floyd

for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (d[i][k] == inf || d[k][j] == inf) continue;
            d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
        }
    }
}
```

3.17 Max Flow With Dinic

```
struct Edge {
    int f, c;
    int to;
    pii revIdx;
    int dir;
    int idx;
};

int n, m;
vector<Edge> adjList[MAX_N];
int level[MAX_N];

void addEdge(int a, int b, int c, int i, int dir) {
    int idx = adjList[a].size();
    int revIdx = adjList[b].size();
    adjList[a].pb({ 0,c,b, {b, revIdx} ,dir,i });
    adjList[b].pb({ 0,0,a, {a, idx} ,dir,i });
}

bool bfs(int s, int t) {
    FOR(i, 0, n) level[i] = -1;
    level[s] = 0;
    queue<int> Q;
    Q.push(s);
    while (!Q.empty()) {
        auto t = Q.front(); Q.pop();
        for (auto x : adjList[t]) {
            if (level[x.to] < 0 && x.f < x.c) {
                level[x.to] = level[t] + 1;
                Q.push(x.to);
            }
        }
    }
    return level[t] >= 0;
}

int send(int u, int f, int t, vector<int> &edgeIdx) {
    if (u == t) return f;
    for (; edgeIdx[u] < adjList[u].size(); edgeIdx[u]++) {
        auto &e = adjList[u][edgeIdx[u]];
        if (level[e.to] == level[u] + 1 && e.f < e.c) {
            int curr_flow = min(f, e.c - e.f);
            int next_flow = send(e.to, curr_flow, t, edgeIdx)
                 ;
            if (next_flow > 0) {
                e.f += next_flow;
                adjList[e.revIdx.first][e.revIdx.second].f -=
                     next_flow;
                return next_flow;
            }
        }
    }
    return 0;
}

int maxFlow(int s, int t) {
    int f = 0;
    while (bfs(s, t)) {
        vector<int> edgeIdx(n, 0);
        while (int extra = send(s, oo, t, edgeIdx)) {
            f += extra;
        }
    }
    return f;
}

void init() {
    cin >> n >> m;
    FOR(i, 0, m) {
        int a, b, c;
        cin >> a >> b >> c;
        a--; b--;
        addEdge(a, b, c, i, 1);
        addEdge(b, a, c, i, -1);
    }
}
```

3.18 Kuhn

```
int n, k;
```

```
vector < vector<int> > g;
vector<int> mt;
vector<bool> used;
bool kuhn(int v) {
    if (used[v]) return false;
    used[v] = true;
    for (auto i : g[v]) {
        int to = i;
        if (mt[to] == -1 || kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}
signed main() {
    cin >> n >> m >> k;
    g.resize(n);
    for (int i = 0; i < k; i++) {
        int temp1, temp2;
        cin >> temp1 >> temp2;
        g[temp1 - 1].push_back(temp2 - 1);

    }
    mt.assign(m, -1);
    for (int v = 0; v < n; ++v) {
        used.assign(n, false);
        kuhn(v);
    }
}
```

4 Geometry

4.1 Graham

```
struct point {
    int x, y;
};
point operator-(point a, point b) {
    return {
        a.x - b.x,
        a.y - b.y
    };
};
bool operator==(point a, point b) {
    return (a.x == b.x) && (a.y == b.y);
}
int operator^(point a, point b) {
    return a.x * b.y - a.y * b.x;
};



bool comp(point &a, point &b) {
    return ((a ^ b) > 0) || ((a ^ b) == 0 && a.x * a.x + a.y
         * a.y > b.x * b.x + b.y * b.y);
};

vector<point> graham(vector<point> points) {
    point p0 = points[0];
    for (point p : points)
        if (p.y < p0.y || (p.y == p0.y && p.x > p0.x)) p0 =
            p;

    for (point &p : points) {
        p.x -= p0.x;
        p.y -= p0.y;
    }

    sort(all(points), comp);

    vector<point> hull;
    for (point p : points) {
        while (hull.size() >= 2 && ((p - hull.back()) ^ (
             hull[hull.size() - 2] - hull.back())) <= 0)
            hull.pop_back();
        hull.push_back(p);
    }

    for (point &p : hull) {
        p.x += p0.x;
        p.y += p0.y;
    }

    return hull;
}
```

4.2 2d Vector

```
template <typename T>
struct vec {
    T x, y;
    vec() : x(0), y(0) { }
    vec(T _x, T _y) : x(_x), y(_y) { }

    vec operator+(const vec &b) {
        return vec<T>(x + b.x, y + b.y);
    }
    vec operator-(const vec &b) {
        return vec<T>(x - b.x, y - b.y);
    }
    vec operator*(T c) {
        return vec(x * c, y * c);
    }
    T operator*(const vec &b) {
        return x * b.x + y * b.y;
    }
    T operator^(const vec &b) {
        return x * b.y - y * b.x;
    }
    bool operator<(const vec &other) const {
        if (x == other.x) return y < other.y;
        return x < other.x;
    }
    bool operator==(const vec &other) const {
        return x == other.x && y == other.y;
    }
    bool operator!=(const vec &other) const {
        return !(*this == other);
    }
    friend ostream &operator<<(ostream &out, const vec &v) {
        return out << "(" << v.x << ", " << v.y << ")";
    }
    friend istream &operator>>(istream &in, vec<T> &v) {
        return in >> v.x >> v.y;
    }
    T norm() { // squared length
        return (*this) * (*this);
    }
    ld len() {
        return sqrt(norm());
    }
    ld angle(const vec &other) { // angle between this and
         other vector
        return acosl((*this) * other / len() / other.len());
    }
    vec perp() {
        return vec(-y, x);
    }
};
```

4.3 Line

```
template <typename T>
struct line { // expressed as two vectors
    vec<T> start, dir;
    line() { }
    line(vec<T> a, vec<T> b) : start(a), dir(b - a) { }

    vec<ld> intersect(line l) {
        ld t = ld((l.start - start) ^ l.dir) / (dir ^ l.dir)
            ;
        // For segment-segment intersection this should be
             in range [0, 1]
        vec<ld> res(start.x, start.y);
        vec<ld> dirld(dir.x, dir.y);
        return res + dirld * t;
    }
};
```

4.4 Circle Line Intersection

```
// ax + by + c = 0, radius is at (0, 0)
double r, a, b, c;

// If the center is not at (0, 0), fix the constant c to
     translate everything so that center is at (0, 0)
double x0 = -a * c / (a * a + b * b), y0 = -b * c / (a * a
     + b * b);
if (c*c > r*r*(a*a+b*b)+eps)
   puts ("no points");
else if (abs (c*c - r*r*(a*a+b*b)) < eps) {
   puts ("1 point");
   cout << x0 << ' ' << y0 << '\n';
}
else {
   double d = r*r - c*c/(a*a+b*b);
   double mult = sqrt (d / (a*a+b*b));
   double ax, ay, bx, by;
   ax = x0 + b * mult;
   bx = x0 - b * mult;
   ay = y0 - a * mult;
   by = y0 + a * mult;
   puts ("2 points");
   cout << ax << ' ' << ay << '\n' << bx << ' ' << by << '\
        n';
}
```

4.5 7zip Cord

```
ll dfs(vector<vector<int>> &Map, int i, int j, vector<
     vector<bool>> &used, vector<int> &Xvalue, vector<int>
     &Yvalue) {
   used[i][j] = true;
   bool flag = false;
   ll sum = Xvalue[i] * Yvalue[j];
   int a[] = { 0, -1, 1, 0 };
   int b[] = { -1, 0, 0, 1 };
   for (int h = 0; h < 4; h++)
      if (Map[i + a[h]][j + b[h]] == 0 && !used[i + a[h]][
           j + b[h]]) {
         flag = true;
         sum += dfs(Map, i + a[h], j + b[h], used, Xvalue,
              Yvalue);
      }
   if (!flag) {
      return Xvalue[i] * Yvalue[j];
   }
   return sum;
}

int main() {
   int w, h, n;
   cin >> w >> h >> n;
   set<int> x, y;
   unordered_map<int, int> X, Y;
   vector<vector<int>> lines;
   vector<int> Xvalue, Yvalue;
   x.insert(0);
   y.insert(0);
   x.insert(w);
   y.insert(h);
   for (int i = 0; i < n; i++) {
      int x1, y1, x2, y2;
      cin >> x1 >> y1 >> x2 >> y2;
      if (x1 < 0)
         x1 = 0;
      if (x1 > w)
         x1 = w;
      if (y1 < 0)
         y1 = 0;
      if (y1 > h)
         y1 = h;
      if (x2 < 0)
         x2 = 0;
      if (x2 > w)
         x2 = w;
      if (y2 < 0)
         y2 = 0;
      if (y2 > h)
         y2 = h;
      lines.push_back({ x1, y1, x2, y2 });
      x.insert(x1);
      x.insert(x2);
      y.insert(y1);
      y.insert(y2);
   }
   int index = 0;
   for (auto _x : x) {
      X[_x] = index;
      index += 2;
   }
   index = 0;
   for (auto _y : y) {
      Y[_y] = index;
      index += 2;
   }
   int prev = 0;
   for (auto _x = ++x.begin(); _x != x.end(); _x++) {
      Xvalue.push_back(0);
      Xvalue.push_back(*_x - prev);
      prev = *_x;
   }
   Xvalue.push_back(0);
   prev = 0;
   for (auto _y = ++y.begin(); _y != y.end(); _y++) {
      Yvalue.push_back(0);
      Yvalue.push_back(*_y - prev);
      prev = *_y;
   }
   Yvalue.push_back(0);
   int Xs = Xvalue.size();
   int Ys = Yvalue.size();
   vector<vector<int>> Map(Xs, vector<int>(Ys, 0));
   for (int i = 0; i < Xs; i++) {
      Map[i][0] = 1;
      Map[i][Ys - 1] = 1;
   }
   for (int i = 0; i < Ys; i++) {
      Map[0][i] = 1;
      Map[Xs - 1][i] = 1;
   }
   for (int i = 0; i < n; i++) {
      if (lines[i][0] == lines[i][2]) {
         int x = X[lines[i][0]];
         int y1 = Y[lines[i][1]];
         int y2 = Y[lines[i][3]];
         if (y1 > y2)
            y1 ^= y2 ^= y1 ^= y2;
         for (int i = y1; i <= y2; i++)
            Map[x][i] = 1;
      }
      else {
         int y = Y[lines[i][1]];
         int x1 = X[lines[i][0]];
         int x2 = X[lines[i][2]];
         if (x1 > x2)
            x1 ^= x2 ^= x1 ^= x2;
         for (int i = x1; i <= x2; i++)
            Map[i][y] = 1;
      }
   }
   vector<ll> s;
   vector<vector<bool>> used(Xs, vector<bool>(Ys, false));
   for (int i = 1; i < Xs - 1; i++) {
      for (int j = 1; j < Ys - 1; j++) {
         if (Map[i][j] == 0 && !used[i][j])
            s.push_back(dfs(Map, i, j, used, Xvalue,
                 Yvalue));
      }
   }
   sort(s.rbegin(), s.rend());
   for (auto _s : s)
      cout << _s << "\n";
}
```

4.6 Formulae

Triangles.
Radius of circumscribed circle:
$R = \frac{abc}{4S}$.
Radius of inscribed circle:
$r = \frac{S}{p}$.
Side via medians:

$a = \frac{2}{3}\sqrt{2(m_b^2 + m_c^2) - m_a^2}$.
Median via sides:
$m_a = \frac{1}{2}\sqrt{2(b^2 + c^2) - a^2}$.
Bisector via sides:
$l_a = \frac{2\sqrt{bcp(p-a)}}{b+c}$.
Bisector via two sides and angle:
$l_a = \frac{2bc\cos\frac{\alpha}{2}}{b+c}$.
Bisector via two sides and divided side:
$l_a = \sqrt{bc - a_b a_c}$.

Right triangles.
Let a, b and c - cathets and hypotenuse, h - height to hypotenuse, dividing c to c_a and c_b. Then
$h^2 = c_a \cdot c_b$,
$a^2 = c_a \cdot c$,
$b^2 = c_b \cdot c$.

Quadrangles.
Sides of circumscribed quadrangle:
$a + c = b + d$.
Square of circumscribed quadrangle:
$S = \frac{Pr}{2} = pr$.
Angles of inscribed quadrangle:
$\alpha + \gamma = \beta + \delta = 180^\circ$.
Square of inscribed quadrangle:
$S = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

Circles.
Intersection of circle and line:
$\begin{cases}(x-x_0)^2 + (y-y_0)^2 = R^2 \\ y = ax + b\end{cases}$
$(x-x_0)^2 + (ax+b-y_0)^2 = R^2$
$(1+a^2)x^2+(2a(b-y_0)-2x_0)x+(x_0^2+(b-y_0)^2-R^2) = 0$
Intersection points are solution of equation. If discriminant D < 0 then there are no intesection points. If discriminant D = 0 then there is one intersection point. If discriminant D > 0 then there are two intersection point.
Intersection of circle and circle:
$\begin{cases}(x-x_0)^2 + (y-y_0)^2 = R_0^2 \\ (x-x_1)^2 + (y-y_1)^2 = R_1^2\end{cases}$
$2(x_0-x_1)x + 2(y_0-y_1)y = (R_1^2 - R_0^2) + (x_0^2 - x_1^2) + (y_0^2 - y_1^2)$
$y = \frac{1}{2}\frac{(R_1^2-R_0^2)+(x_0^2-x_1^2)+(y_0^2-y_1^2)}{y_0-y_1} - \frac{x_0-x_1}{y_0-y_1}x$
Task comes to intersection of circle and line.

5 Algebra

5.1 Combinations

```
int c(int n, int k) {
   int result = 1;
   for (int i = 1; i <= k; i++) {
      result *= n - i + 1;
      result /= i;
   }
   return result;
}

const int N = 20;
vector<vector<int>> C(N + 1, vector<int>(N + 1, 1));

for (int i = 1; i < N + 1; i++)
   for (int j = 1; j < N + 1; j++)
      C[i][j] = C[i - 1][j] + C[i][j - 1];
```

5.2 Eratosthenes

```
template<class T>
class prime {
private:
   std::vector<unsigned>pr;
   std::vector<T>lp;
public:
   prime() {};
   prime(unsigned limit) {
      lp.resize(++limit, 0);
      pr.clear();
      pr.push_back(1);
      for (unsigned i = 2;i < limit;++i) {
         register unsigned max_index = lp[i];
         if (max_index == 0) {
            max_index = pr.size();
            pr.push_back(i);
         }
         register unsigned d;
         for (unsigned j = 1;j <= max_index && (d = i * pr
              [j]) < limit;++j)
            lp[d] = j;
      }
   }
   bool is_prime(unsigned number) {
      return number < lp.size() && !lp[number];
   }
   unsigned sequence_number(unsigned prime_number) {
      if (!is_prime(prime_number))
         return 0;
      return std::lower_bound(pr.begin(), pr.end(),
           prime_number) - pr.begin();
   }
   unsigned return_prime(unsigned sequence_number) {
      if (sequence_number && sequence_number < pr.size())
         return pr[sequence_number];
      return 0;
   }
   unsigned least_divisor(unsigned number) {
      if (number >= lp.size())
         return 0;
      if (is_prime(number))
         return number;
      return pr[lp[number]];
   }
   unsigned limit() {
      return lp.size() - 1;
   }
   std::vector<unsigned> factorize(unsigned number) {
      std::vector<unsigned> v;
      if (number < lp.size()) {
         while (!is_prime(number)) {
            v.push_back(pr[lp[number]]);
            number /= pr[lp[number]];
         }
         v.push_back(number);
      }
      return v;
   }
};
//
     ..................................................................

signed main()
{
   int t = 1;
   cin >> t;
```

```
    int n = 1e7;
    vector<int>p(1e7 + 1, 0);
    vector<int>mass;
    for (int i = 2; i * i <= n; i++) {
        if (p[i] == 1) continue;
        if (i * i <= n) {
            for (int j = i * i; j <= n; j += i) {
                p[j] = 1;
            }
        }
    }
    for (int i = 2; i <= 1e7; ++i) {
        if (p[i] == 0) { // если0 - числопростое
            mass.push_back(i);
        }
    }
    while (t--)
    {
        int l = 0, r = 0;
        cin >> l >> r;
        int lx = lower_bound(mass.begin(), mass.end(), l) -
             mass.begin();
        int rx = lower_bound(mass.begin(), mass.end(), r +
             1) - mass.begin();
        cout << rx - lx << endl;
    }
}
//
     ....................................................................

const int SQRT_MAXN = 10000; //
     кореньизмаксимальногозначенияN
const int S = 1e7+1;
bool nprime[SQRT_MAXN], bl[S];
int primes[SQRT_MAXN], cnt;

signed main() {

    int t = 1;
    cin >> t;
    int n = 1e7 + 1;
    int nsqrt = (int)sqrt(n + .0);
    for (int i = 2; i <= nsqrt; ++i)
        if (!nprime[i]) {
            primes[cnt++] = i;
            if (i * 1ll * i <= nsqrt)
                for (int j = i * i; j <= nsqrt; j += i)
                    nprime[j] = true;
        }

    int result = 0;
    vector<int> mass;
    for (int k = 0, maxk = n / S; k <= maxk; ++k) {
        memset(bl, 0, sizeof bl);
        int start = k * S;
        for (int i = 0; i < cnt; ++i) {
            int start_idx = (start + primes[i] - 1) / primes[
                 i];
            int j = max(start_idx, (long long)2) * primes[i]
                 - start;
            for (; j < S; j += primes[i])
                bl[j] = true;
        }
        if (k == 0)
            bl[0] = bl[1] = true;
        for (int i = 0; i < S && start + i <= n; ++i)
            if (!bl[i])
                mass.push_back(i);
    }
    while (t--) {
        int l = 0, r = 0;
        cin >> l >> r;
        int lx = lower_bound(mass.begin(), mass.end(), l) -
             mass.begin();
        int rx = lower_bound(mass.begin(), mass.end(), r +
             1) - mass.begin();
        cout << rx - lx << endl;
    }
}
```

5.3 Fft

```
const int fft_mod = 7340033; // 7 * 2^20 + 1
const int fft_root = 5; // 5 ^ (2^20) == 1 mod 7340033
const int fft_root_1 = 4404020; // 5 * 4404020 == 1 mod
     7340033
const int fft_pw = 1 << 20; // 2 ^ 20

vector<int> fft(vector<int> a, bool invert = 0) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j >= bit; bit >>= 1) j -= bit;
        j += bit;
        if (i < j) swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        int root_len = invert ? fft_root_1 : fft_root;

        for (int i = len; i < fft_pw; i <<= 1)
            root_len = (root_len * root_len) % fft_mod;

        for (int i = 0; i < n; i += len) {
            int root = 1;
            for (int j = 0; j < len / 2; j++) {
                int u = a[i + j], v = a[i + j + len / 2] *
                     root % fft_mod;
                a[i + j] = (u + v) % fft_mod;
                a[i + j + len / 2] = (u - v + fft_mod) %
                     fft_mod;
                root = (root * root_len) % fft_mod;
            }
        }
    }

    if (invert) {
        int _n = 1;
        for (int i = 1; i <= fft_mod - 2; i++) _n = (_n * n)
             % fft_mod;
        for (int i = 0; i < n; i++) a[i] = (a[i] * _n) %
             fft_mod;
    }
}
signed fast_fourier_transform() {
    int n, m, s;
    cin >> n >> m;
    vector<int> a(n), b(m);

    for (int i = 0; i < n; i++) cin >> a[i];
    for (int i = 0; i < m; i++) cin >> b[i];
    s = 1; while (s < n + m - 1) s <<= 1;
    a.resize(s); b.resize(s);

    vector<int> fa = fft(a), fb = fft(b), fc(fa.size());
    for (int i = 0; i < fa.size(); i++) fc[i] = fa[i] * fb[i
         ];

    vector<int> c = fft(fc, 1);
    for (int i = 0; i < s; i++) cout << a[i] << ' '; cout <<
         '\n';
    for (int i = 0; i < s; i++) cout << b[i] << ' '; cout <<
         '\n';
    for (int i = 0; i < s; i++) cout << c[i] << ' '; cout <<
         '\n';
}
```

5.4 Fibonacci

```
signed fibonacci() {
    int n = 0, m = 0;
    cin >> n >> m;
    vector<vector<int>> mass(2, vector<int>(2));

    mass[0][0] = 0;
    mass[0][1] = 1;
    mass[1][0] = 1;
    mass[1][1] = 1;

    if (n == 1) {
        cout << 1 << endl;
        return 0;
    }
    if (n == 2) {
        cout << 1 << endl;
        return 0;
    }
```

```
    if (n == 3) {
        cout << 2 << endl;
        return 0;
    }

    vector<vector<int>> powed = fast_pow(mass, n - 3, m);

    int result = 0;
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++)
            result += powed[i][j];
    }

    cout << result % m << endl;
}
```

5.5 Gcd

```
// simple gcd
int gcd(int a, int b) {
    while (b) {
        a %= b;
        swap(a, b);
    }
    return a;
}

// euclidean algorithm
int gcd(int a, int b, int &x, int &y) {
    if (a == 0) {
        x = 0; y = 1;
        return b;
    }

    int x1, y1;
    int d = gcd(b % a, a, x1, y1);

    x = y1 - (b / a) * x1;
    y = x1;

    return d;
}
```

5.6 Extended Euclidean Algorithm

```
// ax + by = gcd(a,b)
void solve_eq(int a, int b, int &x, int &y, int &g) {
    if (b == 0) {
        x = 1;
        y = 0;
        g = a;
        return;
    }

    int xx, yy;
    solve_eq(b, a % b, xx, yy, g);

    x = yy;
    y = xx - yy * (a / b);
}

// ax + by = c
bool solve_eq(int a, int b, int c, int &x, int &y, int &g)
    {
    solve_eq(a, b, x, y, g);

    if (c % g != 0)
        return false;

    x *= c / g; y *= c / g;

    return true;
}

// finds a solution (x, y) so that x >= 0 and x is minimal
bool solve_eq_non_neg_x(int a, int b, int c, int &x, int &y
    , int &g) {
    if (!solve_eq(a, b, c, x, y, g))
        return false;

    int k = x * g / b;
    x = x - k * b / g;
    y = y + k * a / g;

    if (x < 0) {
        x += b / g;
        y -= a / g;
    }

    return true;
}
```

5.7 Euler Totient Function

```
// number of numbers x < n so that gcd(x, n) = 1
int phi(int n) {
    if (n == 1)
        return 1;

    // f = vector<pair<prime, count>>
    auto f = factorize(n);

    int res = n;
    for(auto p : f) {
        res = res - res/p.first;
    }

    return res;
}
```

5.8 Factorization

```
vector<int> factorization(int n) {
    vector<int> result;
    for (int i = 2; i * i <= n; i++)
        while (n % i == 0) {
            result.push_back(i);
            n /= i;
        }
    if (n != 1)
        result.push_back(n);
    return result;
}
```

5.9 Binary Mult Pow

```
int binmult(int a, int b) {
    int res = 0;
    while (b) {
        if (b & 1)
            res += a;
        a *= 2;
        b >>= 1;
    }
    return res;
}

int binpow(int a, int n) {
    int res = 1;
    while (n) {
        if (n & 1)
            res *= a;
        a *= a;
        n >>= 1;
    }
    return res;
}
```

5.10 Matrices

```
vector<vector<int>> matrix_production(vector<vector<int>> &
     a, vector<vector<int>> &b, int mod=0) {
   vector<vector<int>> result(a.size(), vector<int>(b[0].
        size()));

   for (int i = 0; i < a.size(); i++) {
      for (int j = 0; j < b[0].size(); j++) {
         for (int k = 0; k < b.size(); k++) {
            if (mod) result[i][j] = (result + a[i][k] * b[
                 k][j] % mod) % mod;
            else result[i][j] += a[i][k] * b[k][j];
         }
      }
   }

   return result;
}

// recursive pow
vector<vector<int>> fast_pow(vector<vector<int>> &a, int n,
      int mod) {
   if (n == 0) {
      vector<vector<int>> temp(a.size(), vector<int>(a[0].
           size()));
      for (int i = 0; i < a.size(); i++) {
         temp[i][i] = 1;
      }
      return temp;
   }
   if (n % 2 == 1) {
      vector<vector<int>> temp = fast_pow(a, n - 1, mod);
      return matrix_production(temp, a, mod);
   }
   else {
      vector<vector<int>> b = fast_pow(a, n / 2, mod);
      return matrix_production(b, b, mod);
   }
}

// iterative pow
vector<vector<int>> fast_pow(vector<vector<int>> &a, int n,
      int mod = 0) {
   vector<vector<int>> res(n, vector<int>(n, 0));
   for (int i = 0; i < n; i++) res[i] = 1;

   while (n) {
      if (n & 1) res = matrix_production(res, a, mod);
      a = matrix_production(a, a, mod);
      n >>= 1;
   }

   return res;
}
```

5.11 Catalan

```
//Catalan(n) = (1 / (n + 1)) * C[2n][n]
//Catalan(n) = Sum[i=0...n-1](Catalan(i)*Catalan(n-1-i)))
```

5.12 Formulae

Combinations.
$C_n^k = \frac{n!}{(n-k)!k!}$
$C_n^0 + C_n^1 + ... + C_n^n = 2^n$
$C_{n+1}^{k+1} = C_n^{k+1} + C_n^k$
$C_n^k = \frac{n}{k} C_{n-1}^{k-1}$

Striling approximation.
$n! \approx \sqrt{2\pi n}\frac{n}{e}^n$

Euler's theorem.
$a^{\phi(m)} \equiv 1 \bmod m$, $gcd(a, m) = 1$

Ferma's little theorem.
$a^{p-1} \equiv 1 \bmod p$, $gcd(a, p) = 1$, p - prime.

Catalan number.
$C_0 = 0, C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$
$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$

Arithmetic progression.
$S_n = \frac{a_1+a_n}{2} n = \frac{2a_1+d(n-1)}{2} n$

Geometric progression.
$S_n = \frac{b_1(1-q^n)}{1-q} n$

Infinitely decreasing geometric progression.
$S_n = \frac{b_1}{1-q} n$

6 Strings

6.1 Manaker

```
signed manaker() {
   string s;
   cin >> s;
   int n = s.length();

   vector<int> d1(n);
   int l = 0, r = -1;

   for (int i = 0; i < n; ++i) {
      int k = i > r ? 1 : min(d1[l + r - i], r - i + 1);
      while (i + k < n && i - k >= 0 && s[i + k] == s[i -
           k])
         ++k;
      d1[i] = k;
      if (i + k - 1 > r)
         l = i - k + 1, r = i + k - 1;
   }

   vector<int> d2(n);
   l = 0, r = -1;

   for (int i = 0; i < n; ++i) {
      int k = i > r ? 0 : min(d2[l + r - i + 1], r - i +
           1);
      while (i + k < n && i - k - 1 >= 0 && s[i + k] == s[
           i - k - 1])
         ++k;
      d2[i] = k;
      if (i + k - 1 > r)
         l = i - k, r = i + k - 1;
   }

   int sum = 0;
   for (int i = 0; i < n; i++) {
      sum += ((d1[i] > 1) ? d1[i] - 1 : 0) + d2[i];
   }
   cout << sum << '\n';
}
```

6.2 Suffixarray

```
void count_sort(vector<int> &p, vector<int> &c) {
   int n = p.size();
   vector<int> cnt(n), p_new(n), pos(n);

   for (auto x : c) cnt[x]++;

   pos[0] = 0;
   for (int i = 1; i < n; i++)
      pos[i] = pos[i - 1] + cnt[i - 1];

   for (auto x : p) {
      int i = c[x];
      p_new[pos[i]] = x;
      pos[i]++;
   }

   p = p_new;
}

signed suffix_array() {
   string str;
   cin >> str;
   str += "&";
   int len = str.length();
   vector<int> p(len), c(len);
   vector<pair<char, int>> a(len);

   for (int i = 0; i < len; i++)
      a[i] = { str[i], i };

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

   for (int i = 0; i < len; i++)
      p[i] = a[i].second;

   c[p[0]] = 0;
   for (int i = 1; i < len; i++)
      if (a[i].first == a[i - 1].first)
         c[p[i]] = c[p[i - 1]];
      else
         c[p[i]] = c[p[i - 1]] + 1;

   int k = 0;
   while ((1 << k) < len) {
      for (int i = 0; i < len; i++)
         p[i] = (p[i] - (1 << k) + len) % len;

      count_sort(p, c);

      vector<int> c_new(len);
      c_new[p[0]] = 0;

      for (int i = 1; i < len; i++) {
         pair<int, int> prev = { c[p[i - 1]], c[(p[i - 1]
              + (1 << k)) % len] };
         pair<int, int> now = { c[p[i]], c[(p[i] + (1 << k
              )) % len] };
         if (now == prev)
            c_new[p[i]] = c_new[p[i - 1]];
         else
            c_new[p[i]] = c_new[p[i - 1]] + 1;
      }

      c = c_new;
      k++;
   }

   for (int i = 0; i < len; i++)
      cout << p[i] << " ";
}
```

6.3 Z Function

```
signed z_func() {
   string s = "";
   cin >> s;
   vector<int> z(s.size());

   for (int i = 1, l = 0, r = 0; i < s.size(); i++) {
      if (i <= r)
         z[i] = min(r - i + 1, z[i - l]);

      while (z[i] + i < s.size() && s[z[i]] == s[z[i] + i
           ])
         z[i]++;

      if (z[i] + i - 1 > r) {
         l = i;
         r = z[i] + i - 1;
      }
   }
}
```

6.4 Bor

```
// построениебораипоискk
     посчетулексиграфическойнаименьшейстрокичерез(
     спускподереву)
int K = 26;
// int MAXN = 10;
int MAXN = 2 * 1e5 + 1;

struct vertex {
   vector<int> next;
   vector<int> count_v;
   bool leaf;
};

vector<vertex> t(MAXN);
int sz;

void add_string(string &s) {
   int v = 0;
   for (size_t i = 0; i < s.length(); ++i) {
      char c = s[i] - 'a';
      if (t[v].next[c] == -1) {
         t[sz].next.assign(K, -1);
         t[sz].count_v.assign(K, 0);
         t[v].next[c] = sz++;
      }
      t[v].count_v[c]++;
      v = t[v].next[c];
   }

   t[v].leaf = true;
}

string dfs(int k) {
   string result = "";
   int init = 0;
   while (k != 0) {
      int temp = 0;
      for (int i = 0; i < t[init].next.size(); i++) {
         if (t[init].count_v[i] && t[init].count_v[i] +
              temp >= k) {
            init = t[init].next[i];
            k -= temp;
            if (t[init].leaf) {
               k--;
            }
            result += char(i + 'a');
            break;
         }
         else if (t[init].count_v[i]) {
            temp += t[init].count_v[i];
         }
      }
   }

   return result;
}

signed main() {
   t[0].next.assign(K, -1);
   t[0].count_v.assign(K, 0);
   sz = 1;

   int n = 0;
   cin >> n;

   for (int i = 0; i < n; i++) {
      string s = "";
      cin >> s;
      bool flag = true;
      for (int i = 0; i < s.size(); i++) {
         if (!isdigit(s[i])) {
            flag = false;
            break;
```

```
            }
        }
        if (flag) {
            int k = stoi(s);
            cout << dfs(k) << endl;
        }
        else {
            add_string(s);
        }
    }
}
```

7 Dynamic Programming

7.1 Backpack

```
vector <int> result;

void getResult(int k, int s, vector<vector<int>> &dp,
     vector<int> &mass) {
    if (dp[k][s] == 0)
        return;
    if (dp[k][s - 1] == dp[k][s])
        getResult(k, s - 1, dp, mass);
    else {
        getResult(k - mass[s - 1], s - 1, dp, mass);
        result.push_back(s);
    }
}

signed backpack() {
    int n, w;
    cin >> n >> w;
    vector<int> m(n + 1), c(n + 1);

    int min_w = 0;
    for (int i = 0; i < n; ++i) {
        cin >> m[i] >> c[i];
        min_w += m[i];
    }
    min_w = min(min_w, w); // чтобынебыломl

    vector<vector<int>> dp(min_w + 1, vector<int>(n + 1, 0))
        ;
    for (int k = 1; k < n + 1; k++) {
        for (int i = 0; i < min_w + 1; i++) {
            // циклдосount <= (min(kolvo_predm[k возможно(
                 сделать- 1)], min_w / c[i]) + 1) - сограничн.
                  количествомпредметов
            for (int count = 0; count <= 1; count++) {
                // еслинеограниченноеколичествопредметовза0(nW
                     )
                // dp[i][k] = dp[i][k - 1]
                // if (m[k] <= i) {
                // dp[i][k] = max(dp[i][k], dp[i][j - m[k]] +
                     c[k - 1]);
                // }
                if (i - m[k - 1] * count >= 0) {
                    dp[i][k] = max(dp[i][k], dp[i - m[k - 1] *
                          count][k - 1] + c[k - 1] * count);
                }
            }
        }
    }

    cout << dp[min_w][n] << endl;
    getResult(min_w, n, dp, m);
    for (int i = 0; i < result.size(); i++) {
        if (result[i] != 0) {
            cout << result[i] << ' ';
        }
    }
}
```

7.2 Coins

```
signed coins() {
    int sum = 0, n = 0;
    cin >> sum >> n;
    vector <int> coin(n), res(sum + 1, 1e9);

    res[0] = 0;
    for (int i = 0; i < n; i++) cin >> coin[i];

    for (int i = 1; i <= sum; i++) {
        for (int j = 0; j < n; j++) {
            if (i - coin[j] >= 0)
                res[i] = min(res[i], res[i - coin[j]] + 1);
        }
    }

    if (res[sum] == 1e9)
        cout << '0';
    else
        cout << res[sum];
}
```

7.3 Increasing Sequence

```
signed increasing_sequence() {
    int n = 0;
    cin >> n;
    vector <int> mass(n, 0), dp(n + 1, 10e8), path, pos(n),
         prev(n);

    for (int i = 0; i < n; i++) cin >> mass[i];
    dp[0] = -10e8;
    pos[0] = -1;
    int len = 0;

    for (int i = 0; i < n; i++) {
        int j = upper_bound(dp.begin(), dp.end(), mass[i]) -
              dp.begin();
        if (dp[j - 1] < mass[i] && mass[i] < dp[j]) {
            dp[j] = mass[i];
            pos[j] = i;
            prev[i] = pos[j - 1];
            len = max(len, j);
        }
    }

    cout << len << endl;
    int p = pos[len];
    while (p != -1) {
        path.push_back(mass[p]);
        p = prev[p];
    }
    reverse(path.begin(), path.end());
    for (int i = 0; i < path.size(); i++) {
        cout << path[i] << ' ';
    }
    cout << '\n';
}
```

7.4 Palindromes

```
signed palindromes() {
    string s = "";
    cin >> s;
    vector <vector<int>> dp(s.length(), vector<int>(s.length
         (), 0));
    vector <vector<int>> pal(s.length(), vector<int>(s.
         length(), 1));

    for (int i = s.length() - 1; i >= 0; i--) {
        for (int j = i; j < s.length(); j++)
            if (i == j) {
                dp[i][j] = 1;
            }
            else {
                pal[i][j] = pal[i + 1][j - 1] & (s[i] == s[j])
                     ;
                dp[i][j] = dp[i][j - 1] + dp[i + 1][j] - dp[i
                     + 1][j - 1] + pal[i][j];
            }
    }
}
```

7.5 Pyramid

```
signed pyramid() {
    int n = 0, m = 0, result = 0;
    cin >> n;
    vector<vector<int>>mass(n + 1, vector<int>(n + 1, 0));

    mass[0][0] = 1;

    for (int i = 1; i < n + 1; i++) {
        for (int j = 1; j < n + 1; j++) {
            if (j > i)
                continue;
            for (int m = 0; m < j; m++)
                mass[i][j] += mass[i - j][m];
        }
    }

    for (int i = 1; i < n + 1; i++) {
        result += mass[n][i];
    }
    cout << result << endl;
}
```

7.6 Ribbon

```
signed ribbon() {
    int n;
    cin >> n;
    int count_a = 0, count_b = 0, count_c = 0;
    vector<int> path(3), dp(n + 1, -10e8), mass(n + 1, -1);

    cin >> path[0] >> path[1] >> path[2];
    dp[0] = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 0; j < 3; j++) {
            if (i - path[j] >= 0) {
                dp[i] = max(dp[i], dp[i - path[j]] + 1);
                if (dp[i] == (dp[i - path[j]] + 1))
                    mass[i] = path[j];
            }
        }
    }
    if (dp[n] != -10e8)
        cout << dp[n] << endl;
    else {
        cout << 0 << endl;
        cout << 0 << " " << 0 << " " << 0 << endl;
        return 0;
    }
    for (int i = n; i >= 1;) {
        if (n <= 0)
            break;
        if (mass[i] == path[0])
            count_a++;
        if (mass[i] == path[1])
            count_b++;
        if (mass[i] == path[2])
            count_c++;

        n -= mass[i];
        i -= mass[i];
    }
    cout << count_a << " " << count_b << " " << count_c << '
        \n';
}
```

7.7 Route

```
signed route() {
    int n = 0;
    cin >> n;
    int last_i = 300, last_j = 300;
    char temp = ' ';

    vector<vector<int>> array(n, vector<int>(n));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> temp;
            array[i][j] = int(temp) - 48;
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == 0 && j == 0)
                continue;
            if (i == 0) {
                array[i][j] = array[i][j - 1] + array[i][j];
                continue;
            }
            if (j == 0) {
                array[i][j] = array[i - 1][j] + array[i][j];
                continue;
            }
            array[i][j] = min(array[i][j] + array[i][j - 1],
                  array[i][j] + array[i - 1][j]);
        }
    }

    for (int i = n - 1; i >= 0; i--) {
        for (int j = n - 1; j >= 0; j--) {
            if (i > last_i || j > last_j)
                continue;
            if (i == n - 1 && j == n - 1) {
                array[i][j] = -1;
                last_i = i;
                last_j = j;
            }
            if (i == 0) {
                array[i][j] = -1;
                last_i = i;
                last_j = j;
                continue;
            }
            if (j == 0 && (array[i][j + 1] == -1 || array[i +
                1][j] == -1)) {
                array[i][j] = -1;
                last_i = i;
                last_j = j;
                continue;
            }
            if (array[i - 1][j] < array[i][j - 1]) {
                array[i - 1][j] = -1;
                last_i = i;
                last_j = j;
                break;
            }
            else {
                array[i][j - 1] = -1;
                last_i = i;
                last_j = j;
            }
        }

    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == 0 && j == 0) {
                cout << "#";
                continue;
            }
            if (array[i][j] == -1)
                cout << "#";
            else
                cout << "-";
        }
        cout << '\n';
    }
}
```

7.8 Upstairs

```
signed upstairs() {
    int n; cin >> n;
    vector <int> coin(n, 0);
    vector <int> dp(n + 1, 0);

    for (int i = 0; i < n; i++) cin >> coin[i];
```

```
    for (int i = 1; i <= n; i++) {
        dp[i] = dp[i - 1];
        if (i >= 2)
            dp[i] = max(dp[i], dp[i - 2]);
        dp[i] += coin[i - 1];
    }

    cout << dp[n];
}
```

8 Misc

8.1 Ternary Search

```
double phi = 1 + (1 + sqrt(5)) / 2;

// continuous ternary search
double cont_ternary_search(double l, double r) {
  double m1 = l + (r - l) / phi, m2 = r - (r - l) / phi;
  double f1 = f(m1), f2 = f(m2);
  int count = 200;
  while (count--) {
    if (f1 < f2) {
      r = m2;
      m2 = m1;
      f2 = f1;
      m1 = l + (r - l) / phi;
      f1 = f(m1);
    }
    else {
      l = m1;
      m1 = m2;
      f1 = f2;
      m2 = r - (r - l) / phi;
      f2 = f(m2);
    }
  }
  return f((l + r) / 2);
}

// discrete ternary search
double discr_ternary_search(int l, int r) {
  int m1 = l + (r - l) / 3, m2 = r - (r - l) / 3;
  while (r - l > 2) {
    if (f(m1) < f(m2))
      r = m2;
    else
      l = m1;
    m1 = l + (r - l) / 3;
    m2 = r - (r - l) / 3;
  }
  return min(f(l), min(f(l + 1), f(r)));
}
```