Contents

```
set nocompatible

set enc=utf-8
set fenc=utf-8
set tenc=utf-8

set backspace=2
set autoindent
set cindent
set tabstop=4
set softtabstop=4
set shiftwidth=4

syntax on
set t_Co=256
set number
set showmatch
set hls

autocmd filetype cpp nnoremap <F9> :w <bar> :!g++ -std=c
    ++11 -O2 % && ./a.out<CR>
```

1 Misc

1.1 JavaCheatSheet

```
import java.util.*;
import java.io.*;
import java.math.*;
public class JavaCheetSheet {
  public static void main(String[] args)throws Exception
    {
    Integer[] v = {1, 5, 2, 4, 3};
    Arrays.sort(v, new Comp());

    LinkedList<Integer> linkedlist = new LinkedList<
    Integer>();
    linkedlist.addLast(1); linkedlist.addFirst(2);
    linkedlist.addFirst(3);
    System.out.println(linkedlist.peekFirst());
    linkedlist.removeLast();

    ArrayList<Integer> list = new ArrayList<Integer>();
    list.add(1); list.add(2); list.add(3); list.add(5);
    list.remove(list.size() - 1); list.remove((Integer)1)
    ;
    for (int i = 0; i < list.size(); i++)
      System.out.print(list.get(i));
    Collections.sort(list, new Comp());
    for (int i : list)
      System.out.print(i);

    Set<String> set = new TreeSet<String>(); // or
    HashSet
    set.add("abc"); set.add("ghi"); set.add("def");
    for (String s : set)
      System.out.println(s);
    System.out.println(set.contains("abc"));
    set.clear();

    Map<String, String> map = new TreeMap<String, String
    >(); // or HashMap
    map.put("k1", "v1"); map.put("k2", "v2");
    System.out.println(map.containsKey("k1"));
    System.out.println(map.get("k2"));
    for (Map.Entry<String, String> entry : map.entrySet()
    ) {
      System.out.println(entry.getKey() + " " + entry.
    getValue());
    }

    BigInteger i1 = new BigInteger("1234567");
    BigInteger i2 = BigInteger.valueOf(23456);
    System.out.println(i1.add(i2));
    System.out.println(i1.isProbablePrime(32));
    System.out.println(i1.modInverse(i2));

    StringBuilder sb = new StringBuilder();
    sb.append("abc"); sb.append(123);
    System.out.println(sb.toString());
  }
}
class Comp implements Comparator<Integer> {
  public int compare(Integer lhs, Integer rhs) {
    return rhs - lhs;
  }
}
class Scan {
  BufferedReader buffer;
  StringTokenizer tok;
  Scan() {
    buffer = new BufferedReader(new InputStreamReader(
    System.in));
  }
  boolean hasNext() {
    while (tok == null || !tok.hasMoreElements()) {
      try {
        tok = new StringTokenizer(buffer.readLine());
      } catch (Exception e) {
        return false;
      }
    }
    return true;
  }
  String next() {
    if (hasNext()) return tok.nextToken();
    return null;
  }
  String nextLine() {
    if (hasNext()) return tok.nextToken("\n");
    return null;
  }
  int nextInt() {
    return Integer.parseInt(next());
  }
}
```

2 BasicDS

2.1 Doubly Linked List

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <algorithm>

using namespace std;

template <class T>
struct node
{
  T val;
  node *prev, *next;
  node(T _val): val(_val), prev(NULL), next(NULL) {}
  node(): prev(NULL), next(NULL) {}
};

template <class T>
struct list
{
  node<T> *first, *last;
  list(): first(NULL), last(NULL) {}

  void insert(const T &val)
  {
    if (first == NULL)
    {
      first = new node<T>(val);
      last = first;
    }
    else
    {
      last->next = new node<T>(val);
      last->next->prev = last;
      last = last->next;
    }
  }
  void insertFront(const T &val)
  {
    if (first == NULL)
      insert(val);
    else
    {
      first->prev = new node<T>(val);
      first->prev->next = first;
      first = first->prev;
    }
  }
  void insertAfter(const T &val, node<T> *nd)
  {
    if (nd == last)
      insert(val);
    else
    {
      node<T> *tmp = nd->next;
      nd->next = new node<T>(val);
      nd->next->prev = nd;
      nd = nd->next;
      nd->next = tmp;
      tmp->prev = nd;
    }
  }
  void move(node<T> *front, node<T> *rear, node<T> *pos)
    // assume front, rear != NULL
  {
    // split
    if (front == first)
    {
      first = rear->next;
      if (first) first->prev = NULL;
    }
    else
    {
      front->prev->next = rear->next;
      if (rear->next) rear->next->prev = front->prev;
    }
    // merge
    if (pos)
    {
      node<T> *nxt = pos->next;
      pos->next = front;
      front->prev = pos;
      rear->next = nxt;
      if (nxt) nxt->prev = rear;
    }
    else
    {
      node<T> *nxt = first;
      first = front;
      first->prev = NULL;
      rear->next = nxt;
      if (nxt) nxt->prev = rear;
    }
    if (pos == last) last = rear;
  }
};

int main()
{
  list<int> l;
  l.insert(4);
  l.insert(5);
  l.insert(2);
  l.insertAfter(7, l.first);
  l.insertFront(1);

  node<int> *it = l.first;
  while (it)
  {
    printf("%d\n", it->val);
    it = it->next;
  }
  puts("-");

  l.move(l.first, l.first->next, l.last->prev);
  it = l.first;
  while (it)
  {
    printf("%d\n", it->val);
    it = it->next;
  }
  return 0;
}
```

2.2 Hashmap_Unordered Map_umap

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <string>
#include <algorithm>
#include <vector>
#include <cstdlib>

using namespace std;

template <typename S>
class hasher
{
private:
    const int hmask;
    vector<vector<int> > lk; // lookup table

    inline int get_rand()
    {
        return ((rand() & 0xffff) << 16 | (rand() & 0
xffff)) & hmask; // [OR_1] return rand(); // for
unix/unix-like
    }
public:
    hasher(const int &hbits)
        : hmask((1 << hbits) - 1)
        , lk((sizeof(S) + 1) >> 1)
    {
        srand(12345678);
        int i, segs = (sizeof(S) + 1) >> 1;
        for (i = 0; i < segs; i++)
        {
            lk[i].resize(1 << 16);
            for (auto &lk_ij: lk[i])
                lk_ij = get_rand();
        }
    }

    inline int get_hash(const S &key)
    {
        int i, ret = 0;
        unsigned short *it = (unsigned short*) &key;
        for (i = 0; i < (sizeof(S) >> 1); i++, it++) ret
^= lk[i][*it];
        if (sizeof(S) & 1) ret ^= lk[i][*((unsigned char
*)it)]; // is not a multiple of 16bits
        return ret;
    }
/* for integer types
    inline int get_hash(S key)
    {
        int ret = 0;
        for ( ; key; key >>= 16) ret ^= lk[key & 0xffff];
        return ret;
    }
*/
};

template <typename S, typename T>
class umap
{
private:
    const int ubits; // table size => (1 << ubits)
    const int umask;
    hasher<S> *hshr;
    vector<int> hkey; // hkey[i] = hash(tb[i].first);
    vector<pair<S, T> > tb; // key[i] => tb[i].first, val
    [i] => tb[i].second;

public:
    typedef pair<S, T> *iterator;

    umap(const int &_ubits = 20, hasher<S> *_hshr = NULL)
        : hkey(1 << _ubits, -1)
        , tb(1 << _ubits)
        , ubits(_ubits)
        , umask((1 << _ubits) - 1)
    {
        hshr = _hshr ? _hshr : (new hasher<S>(ubits));
    }

    iterator end() { return NULL; }

    iterator find(const S &key)
    {
        int i;
        int ha = hshr->get_hash(key);
        for (i = ha; hkey[i] != -1; i = (i + 1) ^ umask)
            if (hkey[i] == ha)
                return (&tb[i]);
        return NULL;
    }

    pair<iterator, bool> insert(const pair<S, T> &p)
    {
        int i;
        int ha = hshr->get_hash(p.first);
        for (i = ha; hkey[i] != -1; i = (i + 1) ^ umask)
            if (hkey[i] == ha)
                return make_pair(&tb[i], false);
        hkey[i] = ha;
        tb[i] = p;
        return make_pair(&tb[i], true);
    }

    T & operator [](const S &key)
    {
        int i;
        int ha = hshr->get_hash(key);
        for (i = ha; hkey[i] != -1; i = (i + 1) ^ umask)
            if (hkey[i] == ha)
                return tb[i].second;
        hkey[i] = ha;
        tb[i].first = key;
        return tb[i].second;
    }

    void clear()
    {
        fill(hkey.begin(), hkey.end(), -1);
    }

};

int main()
{
    umap<int, int> mp;
    umap<int, int>::iterator it;
    mp.insert(make_pair(123, 1)); cout << "(123, 1)
    inserted\n";
    mp.insert(make_pair(456, 2)); cout << "(456, 2)
    inserted\n";
    mp.insert(make_pair(789, 3)); cout << "(789, 3)
    inserted\n";

    it = mp.find(123456);
    if (it != mp.end())
        cout << it->first << " " << it->second << endl;
    else
        cout << "Not Found: 123456\n";

    it = mp.find(456);
    if (it != mp.end())
        cout << it->first << " " << it->second << endl;
    else
        cout << "456 Not Found\n";


    cout << "mp[789] = " << mp[789] << endl;

    // clear the map
    mp.clear();
    cout << "\nmp cleared\n";

    mp[159] = 4; cout << "(159, 4) inserted\n";
    mp[753] = 5; cout << "(753, 5) inserted\n";

    it = mp.find(123);
    if (it != mp.end())
        cout << it->first << " " << it->second << endl;
    else
        cout << "Not Found: 123\n";

    it = mp.find(753);
    if (it != mp.end())
        cout << it->first << " " << it->second << endl;
    else
        cout << "Not Found\n";
    return 0;
}
```

3 Graph

3.1 Strongly Connected Component

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cstdlib>
#include <vector>

using namespace std;

const int MAX_V = 2002;

int vs, es;
vector<int> g[MAX_V];
int t, low[MAX_V], dfn[MAX_V];
char instk[MAX_V];
int stk[MAX_V], top;
int scc_cnt;

void dfs(int vi)
{
  dfn[vi] = t++;
  low[vi] = dfn[vi];
  stk[++top] = vi;
  instk[vi] = 1;
  for (auto &vj: g[vi])
    if (dfn[vj] == -1)
    {
      dfs(vj);
      low[vi] = min(low[vi], low[vj]);
    }
    else if (instk[vj])
      low[vi] = min(low[vi], dfn[vj]);
  if (low[vi] == dfn[vi])
  {
    scc_cnt++;
    while (stk[top] != vi)
    {
      instk[stk[top]] = 0;
      --top;
    }
    instk[stk[top]] = 0;
    --top;
  }
}

int main()
{
  int i;
  int v1, v2, dir;
  while (scanf("%d%d", &vs, &es) == 2 && (vs | es))
  {
    for (i = 1; i <= vs; i++) g[i].clear();
    t = 0;
    memset(dfn, -1, sizeof(dfn));
    memset(instk, 0, sizeof(instk));
    top = -1;
    scc_cnt = 0;
    for (i = 0; i < es; i++)
    {
      scanf("%d%d%d", &v1, &v2, &dir);
      g[v1].push_back(v2);
      if (dir == 2) g[v2].push_back(v1);
    }
    for (i = 1; i <= vs; i++)
      if (dfn[i] == -1)
        dfs(i);
    printf("%d\n", scc_cnt == 1);
  }
  return 0;
}
```

3.2 Articulation Point

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
#include <cstdlib>
#include <cmath>

using namespace std;

const int MAX_V = 10002;

pair<int, int> ans[MAX_V];
vector<int> g[10002];
int vs, es, t;
int low[MAX_V], dfn[MAX_V];

bool cmp(pair<int, int> a, pair<int, int> b)
{
  if (a.second != b.second)
    return a.second > b.second;
  else
    return a.first < b.first;
}

int dfs(int p, int vi)
{
  dfn[vi] = ++t;
  low[vi] = dfn[vi];
  int ch = 0, cnt = 0;
  for (auto &vj: g[vi])
    if (dfn[vj] == -1)
    {
      ++ch;
      dfs(vi, vj);
      if (low[vj] >= dfn[vi]) cnt++;
      low[vi] = min(low[vi], low[vj]);
    }
    else if (vj != p)
      low[vi] = min(low[vi], dfn[vj]);
  if (p != -1) // not root
  {
    if (cnt) ans[vi].second = cnt+1;
  }
  else
  {
    if (ch > 1) ans[vi].second = ch;
  }
  return low[vi];
}

int main()
{
  int i, m, v1, v2;
  while (scanf("%d%d", &vs, &m) == 2 && (vs | m))
  {
    t = 0;
    memset(dfn, -1, sizeof(dfn));
    for (i = 0; i < vs; i++)
    {
      g[i].clear();
      ans[i].first = i;
      ans[i].second = 1;
    }
    while (scanf("%d%d", &v1, &v2) == 2 && !(v1 == -1 &&
    v2 == -1))
    {
      g[v1].push_back(v2);
      g[v2].push_back(v1);
    }
    for (i = 0; i < vs; i++)
      if (dfn[i] == -1)
        dfs(-1, i);
    sort(ans, ans+vs, cmp);
    for (i = 0; i < m; i++)
      printf("%d %d\n", ans[i].first, ans[i].second);
    puts("");
  }
  return 0;
}
```

3.3 Eulerian Circuit_Trail

```
/* Eulerian trail/circuit properties

Undirected graph:
all vertices with nonzero degree belong to a single
    connected component
[Circuit] every vertex has even degree.
[Trail] at most two vertices have odd degree

Directed graph:
[Circuit] every vertex has equal in degree and out degree
    (all vertices with nonzero degree belong to a single
    strongly connected component)
[Trail] at most one vertex has (out-degree) - (in-degree)
     = 1
    , at most one vertex has (in-degree) - (out-degree) =
     1
    , every other vertex has equal in-degree and out-
    degree
    (all vertices with nonzero degree belong to a single
    connected component of the underlying undirected
    graph)
*/

#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

const int V = 52;

int g[V][V];
int deg[V];
int v;

void dfs(int vi)
{
  int vj;
  for (vj = 1; vj <= v; vj++)
    if (g[vi][vj])
    {
      g[vi][vj]--; g[vj][vi]--;
      dfs(vj);
      printf("%d %d\n", vj, vi);
    }
}

int main()
{
  int i, j, t, e;
  int v1, v2;
  scanf("%d", &t);
  for (int c = 1; c <= t; c++)
  {
    v = 0;
    memset(g, 0, sizeof(g));
    memset(deg, 0, sizeof(deg));

    scanf("%d", &e);
    while (e--)
    {
      scanf("%d%d", &v1, &v2);
      v = max(v, v1); v = max(v, v2);
      g[v1][v2]++; g[v2][v1]++;
      deg[v1]++; deg[v2]++;
    }
    if (c-1) puts("");
    printf("Case #%d\n", c);
    for (i = 1; i <= v; i++)
      if (deg[i] & 1)
        break;
    if (i <= v)
      puts("some beads may be lost");
    else
      for (i = 1; i <= v; i++)
        dfs(i);
  }
  return 0;
}
```

3.4 LCA_Tarjan's Algorithm

```
#include <cstdio>
#include <cstring>
#include <vector>
#define MAX 905

using namespace std;

int vs, es;
vector<int> g[MAX];
int pre[MAX];
int f[MAX];
int vleft[MAX], top;
int lca[MAX][MAX];
int cnt[MAX];

inline int input()
{
  char c;
  for (c = getchar(); c < 48 || c > 57; c = getchar()) ;
  int x = c - 48;
  for (c = getchar(); c > 47 && c < 58; c = getchar())
    x = x * 10 + c - 48;
  return x;
}

int findF(int idx)
{
  if (f[idx] == idx) return idx;
  return f[idx] = findF(f[idx]);
}

void unionF(int idx1, int idx2)
{
  int f1 = findF(idx1), f2 = findF(idx2);
  f[idx2] = idx1;
}

void dfs(int idx)
{
  int i, sz = g[idx].size();
  for (i = 0; i < sz; i++)
  {
    int &idx2 = g[idx][i];
    dfs(idx2);
    vleft[++top] = idx2;
    unionF(idx, idx2);
  }
  for (i = 0; i <= top; i++)
    lca[ vleft[i] ][idx] =
    lca[idx][ vleft[i] ] = findF( vleft[i] );
}


int main()
{
  int i, j, vi, vj, ps;
  while (scanf(" %d", &vs) == 1)
  {
    top = -1;
    for (i = 1; i <= vs; i++)
    {
      g[i].clear();
      pre[i] = i;
      f[i] = i;
      cnt[i] = 0;
    }

    for (i = 0; i < vs; i++)
    {
      vi = input();
      es = input();
      while (es--)
      {
        vj = input();
        g[vi].push_back(vj);
        pre[vj] = vi;
      }
    }
    while (pre[vi] != vi)
      vi = pre[vi];

    dfs(vi);

    ps = input();
    while (ps--)
    {
      vi = input();
      vj = input();
      cnt[ lca[vi][vj] ]++;
    }

    for (i = 1; i <= vs; i++)
      if (cnt[i] > 0)
        printf("%d:%d\n", i, cnt[i]);
  }
  return 0;
}
```

3.5 LCAtoRMQ_LA

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <algorithm>
#include <vector>

using namespace std;

const int N = 100005;
const int lgN = 25;

int rmq[N << 1][lgN];
void build_rmq(int a[], int len)
{
  int i, j, i1, i2;
  for (i = 0; i < len; i++)
  {
    rmq[i][0] = i;
    for (j = 1; i - (1 << j) + 1 >= 0; j++)
    {
      i1 = rmq[i][j - 1];
      i2 = rmq[i - (1 << (j - 1))][j - 1];
      rmq[i][j] = a[i1] < a[i2] ? i1 : i2;
    }
  }
}
int get_rmq(int a[], int i, int j)
{
  if (i > j) swap(i, j);
  int i1, i2;
  int k, l = j - i + 1;
  for (k = 0; (1 << k) <= l; k++) ; --k;
  i1 = rmq[i + (1 << k) - 1][k];
  i2 = rmq[j][k];
  return a[i1] < a[i2] ? i1 : i2;
}

vector<int> g[N]; // 0-based
int t;
int dfn[N], lst[N << 1], dep[N << 1]; // o[i]: the i'th
    vertex visited, dfn[i]: the visited time of vertex i
    , dth[i]: the depth of vertex i
void dfs(int d, int vi)
{
  dfn[vi] = t;
  lst[t] = vi;
  dep[t] = d;
  t++;
  for (auto &vj: g[vi])
    if (dfn[vj] == -1) // not needed if directed tree
    {
      dfs(d + 1, vj);
      lst[t] = vi;
      dep[t] = d;
      t++;
    }
}
void build_lca(int n)
{
  fill(dfn, dfn + n, -1);
  t = 0;
  dfs(0, 0); // second paramter = root
  build_rmq(dep, t);
}
int get_lca(int vi, int vj)
{
  return lst[get_rmq(dep, dfn[vi], dfn[vj])];
}

// vector<int> g[N];
char vst[N];
int anc[N][lgN];
void dfs2(int d, int va, int vi)
{
  int i;
  vst[vi] = 1; // not needed if directed tree
  anc[vi][0] = va;
  for (i = 1; d - (1 << i) >= 0; i++)
    anc[vi][i] = anc[ anc[vi][i - 1] ][i - 1];
  for (auto &vj: g[vi])
    if (!vst[vj]) // not needed if directed tree
      dfs2(d + 1, vi, vj);
}
void build_la(int n)
{
  fill(vst, vst + n, 0);
  dfs2(0, -1, 0); // third parameter = root
}
int get_la(int vi, int k)
{
  int p;
  for (p = 0; k; k >>= 1, p++)
    if (k & 1)
      vi = anc[vi][p];
  return vi;
}

// vector<int> g[N];
// char vst[N];
int sz[N];
int dfs3(int vi)
{
  vst[vi] = 1;
  sz[vi] = 1;
  for (auto &vj: g[vi])
    if(!vst[vj])
      sz[vi] += dfs3(vj);
  return sz[vi];
}
void build_sz(int n)
{
  fill(vst, vst + n, 0);
  dfs3(0); // first parameter is the root
}

int main()
{
  int i, n, q;
  int vi, vj, an;
  int di, dj;
  int ani, anj;
  scanf("%d", &n);
  for (i = 1; i < n; i++)
  {
    scanf("%d%d", &vi, &vj); --vi; --vj;
    g[vi].push_back(vj);
    g[vj].push_back(vi);
  }
  build_lca(n);
  build_la(n);
  build_sz(n);
  scanf("%d", &q);
  while (q--)
  {
    scanf("%d%d", &vi, &vj); --vi; --vj;
    an = get_lca(vi, vj);
    di = dep[dfn[vi]] - dep[dfn[an]];
    dj = dep[dfn[vj]] - dep[dfn[an]];
    if ((di + dj) & 1)
      puts("0");
    else if (vi == vj)
      printf("%d\n", n);
    else if (di == dj)
    {
      ani = get_la(vi, di - 1);
      anj = get_la(vj, dj - 1);
      printf("%d\n", n - sz[ani] - sz[anj]);
    }
    else
    {
      if (di < dj)
        swap(di, dj),
        swap(vi, vj);
      ani = get_la(vi, (di + dj) >> 1);
      anj = get_la(vi, ((di + dj) >> 1) - 1);
      printf("%d\n", sz[ani] - sz[anj]);
    }
  }
//  int i, j;
//  g[0].push_back(1); g[1].push_back(0);
//  g[1].push_back(2); g[2].push_back(1);
//  g[1].push_back(3); g[3].push_back(1);
//  g[2].push_back(4); g[4].push_back(2);
//  build_lca(5);
//  cout << get_lca(1, 4) << endl;
//  cout << get_lca(3, 4) << endl;
//  cout << get_lca(2, 3) << endl;
//
//  build_la(5);
//  cout << get_la(3, 2) << endl;
//  cout << get_la(4, 2) << endl;
//  cout << get_la(4, 3) << endl;
//
//  build_sz(5);
//  cout << sz[0] << endl;
//  cout << sz[2] << endl;
//
//  while (1);
//
//  int a[] = {2, 5, 6, 1, 2, 3, 5, 0, 4};
//  int n = sizeof(a) / sizeof(int);
//  build_rmq(a, n);
//  for (i = 0; i < n; printf(", i = %d\n", i), i++)
//    for (j = 0; j < 3; j++)
//      printf("%d ", rmq[i][j]);
//  while (cin >> i >> j)
//    cout << get_rmq(a, i, j) << endl;
  return 0;
}
```

3.6 Min Path Cover_Max Bipartite Matching

```
/*
Max Bipartite Matching (call it MA)
Min Vertex Cover = MA
Max Independent Set = V - MA

Directed graph:
Minimum Disjoint Path Cover
    => Build a new graph, Edge (Vi, Vj) = Edge (Out(Vi),
    In(Vj))
    = V - MA(new graph)
Minimum Path Cover
    => Floyd Warshall to obtain transitive closure
    => Build a new graph, Vi can reach Vj  = Edge (Out(Vi
    ), In(Vj))\
    = V - MA(new graph)
*/

#include <cstdio>
#include <algorithm>
#include <vector>
#include <queue>

using namespace std;

const int N = 1005;
const int NN = N + N;

vector<int> g[NN];
char vst[NN];
int match[NN];
int ma;

bool dfs(int vi)
{
  for (auto &vj: g[vi])
  {
    if (vst[vj]) continue;
    vst[vj] = 1;
    if (match[vj] == -1 || dfs(match[vj]))
    {
      match[vi] = vj;
      match[vj] = vi;
      return true;
    }
  }
  return false;
}

int main()
{
  int i, j, n, d;
  int vi, vj;
  scanf("%d", &n);
  ma = 0; // vst = 0
  fill(match, match + n * 2, -1);
  for (i = 0; i < n; i++)
  {
    scanf("%d", &d);
    for (j = 0; j < d; j++)
    {
      scanf("%d", &vj);
      g[n + i].push_back(vj);
      g[vj].push_back(n + i);
    }
  }
  for (i = 0; i < n * 2; i++)
    if (match[i] == -1)
    {
      fill(vst, vst + n * 2, 0);
      ma += dfs(i);
    }
  printf("%d\n", n - ma);
  return 0;
}
```

3.7 Maximum Flow_Dinic's Algorithm

```
#include <cstdio>
#include <cstring>
#include <vector>
#include <queue>
#include <algorithm>
#define MAX_V 102
#define INF 1 << 30

using namespace std;

struct edge{ int to, res, rev; };
vector<edge> g[MAX_V];
int lvl[MAX_V];
int iter[MAX_V];

void addEdge(int from, int to, int fcap, int bcap)
{
  g[from].push_back( edge{ to, fcap, g[to].size() } );
  g[to].push_back( edge{ from, bcap, g[from].size()-1 } )
    ;
}

void doFlow(edge &e, int f)
{
  e.res -= f;
  g[e.to][e.rev].res += f;
}

int bfs(const int &src, const int &sink)
{
  int qi;
  queue<int> q;
  memset(lvl, -1, sizeof(lvl));
  lvl[src] = 0;
  q.push(src);
  while (!q.empty())
  {
    qi = q.front(); q.pop();
    for (auto &e: g[qi])
      if (e.res > 0 && lvl[e.to] < 0)
      {
        lvl[e.to] = lvl[qi] + 1;
        q.push(e.to);
      }
  }
  return lvl[sink];
}

int dfs(int idx, int minF, const int &dest)
{
  if (idx == dest) return minF;
  int sz = g[idx].size(), ret;
  for (int &i = iter[idx]; i < sz; i++)
  {
    edge &e = g[idx][i];
    if (e.res > 0 && lvl[idx] < lvl[e.to])
    {
      ret = dfs(e.to, min(minF, e.res), dest);
      if (ret > 0)
      {
        doFlow(e, ret);
        return ret;
      }
    }
  }
  return 0;
}

int maxFlow(const int &src, const int &sink)
{
  int ret, f = 0;
  while ( bfs(src, sink) >= 0 )
  {
    memset(iter, 0, sizeof(iter));
    while ( (ret = dfs(src, INF, sink)) > 0)
      f += ret;
  }
  return f;
}

int main()
{
  int i, cases = 0, vs, src, sink, es, v1, v2, cap;
  while (scanf("%d", &vs) == 1 && vs > 0)
  {
    for (i = 1; i <= vs; i++) g[i].clear();
    scanf("%d%d%d", &src, &sink, &es);
    while (es--)
    {
      scanf("%d%d%d", &v1, &v2, &cap);
      addEdge(v1, v2, cap, cap);
    }
    printf("Network %d\n", ++cases);
    printf("The bandwidth is %d.\n\n", maxFlow(src, sink)
    );
  }
  return 0;
}
```

4 Math

4.1 ExGCD_Lucas_CRT

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <algorithm>
#include <utility>

using namespace std;

typedef unsigned long long ull;

const int K = 10;
const int M = 100002;

int n, m, k;
int pt;
int p[K], pi[M]; // pi[prime] = index
int inv[K][M];
ull r[K];

int ex_gcd(int a, int b, int &x, int &y)
{
  if (b == 0) { x = 1; y = 0; return a; }
  int x1, y1, g = ex_gcd(b, a % b, y1, x1);
  x = x1;
  y = y1 - a / b * x1;
  return g;
}
int mod_inv(int a, int p)
{
  int x, y, g;
  g = ex_gcd(a, p, x, y);
  if (x < 0)
  {
    int dx = p / g;
    x = (x + dx * ((-x) / dx + 1)) % dx;
  }
  return x;
}
ull c_small(int n, int m, int p) // pi = the ith prime in
     the input
{
  if (n < m) return 0;
  int i; ull ret = 1;
  if (m >= (n >> 1)) m = n - m;
  for (i = 1; i <= m; i++)
  {
    ret = (ret * (n - i + 1)) % p;
    ret = (ret * inv[pi[p]][i]) % p;
  }
  return ret;
}
ull lucas(int n, int m, int p)
{
  ull ret = 1;
  for ( ; n | m; n /= p, m /= p)
    ret = (ret * c_small(n % p, m % p, p)) % p;
  return ret;
}
void cal_inv()
{
  int i, j;
  for (i = 0; i < k; i++)
    for (j = 1; j <= p[i]; j++)
      inv[i][j] = mod_inv(j, p[i]);
}
ull crt() // (int r[], int k, int p[])
{
  int i;
  ull ret = 0;
  int pt = 1, pti;
  // calculate pi
  for (i = 0; i < k; i++) pt *= p[i];
  // crt start
  for (i = 0; i < k; i++)
  {
    pti = pt / p[i];
    ret += (ull)r[i] * pti * mod_inv(pti, p[i]);
    ret %= pt;
  }
  return ret;
}
int main()
{
  int i, tt;
  scanf("%d", &tt);
  while (tt--)
  {
    scanf("%d%d%d", &n, &m, &k);
    for (i = 0; i < k; pi[p[i]] = i, i++) scanf("%d", &p[
    i]);
    cal_inv();
    for (i = 0; i < k; i++) r[i] = lucas(n, m, p[i]);
    printf("%llu\n", crt());
  }
  return 0;
}
```

4.2 Pollard's rho_Miller Rabin

```
#include <cstdio>
#include <map>
#include <algorithm>

using namespace std;
typedef long long ll;

map<ll, int> fact;
ll mul(ll a, ll b, ll n) { // a*b%n
    ll r = 0;
    a %= n, b %= n;
    while(b){
        if( b&1 ) r = a+r>=n ? a+r-n : a+r;
        a = a+a>=n ? a+a-n : a+a;
        b >>= 1;
    }
    return r;
}
ll powmod(ll a, ll d, ll n) { // a^d%n
    if(d==0) return 1ll;
    if(d==1) return a%n;
    return mul(powmod(mul(a, a, n), d>>1, n), d%2?a:1, n)
    ;
}
bool miller_rabin(ll n, ll a) {
    if(__gcd(a,n)==n) return true;
    if(__gcd(a,n)!=1) return false;
    ll d = n-1, r = 0, res;
    while(d%2==0) { ++r; d>>=1; }
    res = powmod(a, d, n);
    if(res==1||res==n-1) return true;
    while(r--) {
        res = mul(res, res, n);
        if(res==n-1) return true;
    }
    return false;
}
bool isPrime(ll n) {
    ll as[7]={2, 325, 9375, 28178, 450775, 9780504,
    1795265022}; //  2, 7, 61
    for(int i=0; i<7; i++)
        if( !miller_rabin(n, as[i]) )
            return false;
    return true;
}
void pollardrho(long long n)
{
    if(n==1) return;
    if(isPrime(n))
  {
        fact[n]++;
        return;
    }
    if(!(n&1))
  {
        fact[2]++;
        pollardrho(n>>1);
        return;
    }
    while(1)
    {
        long long a = rand()%n;
        long long b = a;
        long long c = rand()%(n-1)+1;
        while(1)
        {
            a = (mul(a, a, n)+c)%n;
            b = (mul(b, b, n)+c)%n;
            b = (mul(b, b, n)+c)%n;
            long long g = __gcd(abs(a-b),n);
            if(g==n) break;
            if(g>1)
            {
                if (isPrime(g)) fact[g]++;
                else pollardrho(g);
                pollardrho(n/g);
                return;
            }
        }
    }
}
int main()
{
    long long n;
    scanf("%lld",&n);
    pollardrho(n);
    int ans=1;
    for(map<ll, int>::iterator it=fact.begin();it!=fact.
    end();it++)
        ans*=it->second+1;
    printf("%d\n",ans);
    return 0;
}
```

5 RangeQuery

5.1 RMQ

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <algorithm>
#include <vector>

using namespace std;

const int N = 100005;
const int lgN = 25;

int rmq[N][lgN];
void build_rmq(int a[], int len)
{
  int i, j;
  for (i = 0; i < len; i++)
  {
    rmq[i][0] = a[i];
    for (j = 1; i - (1 << j) + 1 >= 0; j++)
      rmq[i][j] = min(rmq[i][j - 1], rmq[i - (1 << (j -
    1))][j - 1]);
  }
}
int get_rmq(int i, int j)
{
  if (i > j) swap(i, j);
  int k, l = j - i + 1;
  for (k = 0; (1 << k) <= l; k++) ; --k;
  return min(rmq[i + (1 << k) - 1][k], rmq[j][k]);
}

int main()
{
  int i, j;
  int a[] = {2, 5, 6, 1, 2, 3, 5, 0, 4};
  int n = sizeof(a) / sizeof(int);
  build_rmq(a, n);
  for (i = 0; i < n; printf(", i = %d\n", i), i++)
    for (j = 0; j < 3; j++)
      printf("%d ", rmq[i][j]);
  while (cin >> i >> j)
    cout << get_rmq(i, j) << endl;
  return 0;
}

/*
 * RMQ that returns the index of the minimum item
 *
int rmq[N][lgN];
void build_rmq(int a[], int len)
{
  int i, j, i1, i2;
  for (i = 0; i < len; i++)
  {
    rmq[i][0] = i;
    for (j = 1; i - (1 << j) + 1 >= 0; j++)
    {
      i1 = rmq[i][j - 1];
      i2 = rmq[i - (1 << (j - 1))][j - 1];
      rmq[i][j] = a[i1] < a[i2] ? i1 : i2;
    }
  }
}
int get_rmq(int a[], int i, int j)
{
  if (i > j) swap(i, j);
  int i1, i2;
  int k, l = j - i + 1;
  for (k = 0; (1 << k) <= l; k++) ; --k;
  i1 = rmq[i + (1 << k) - 1][k];
  i2 = rmq[j][k];
  return a[i1] < a[i2] ? i1 : i2;
}

int main()
{
  int i, j;
  int a[] = {2, 5, 6, 1, 2, 3, 5, 0, 4};
  int n = sizeof(a) / sizeof(int);
  build_rmq(a, n);
  for (i = 0; i < n; printf(", i = %d\n", i), i++)
    for (j = 0; j < 3; j++)
      printf("%d ", rmq[i][j]);
  while (cin >> i >> j)
    cout << get_rmq(a, i, j) << endl;
  return 0;
}*/
```

5.2 Fenwick Tree

```
/* Arrays numbered from 1 */

#include <cstdio>
#include <cstring>
#define MAX 100002

using namespace std;

int a[MAX], fenwick[MAX], maxIdx;

int query(int idx)
{
  int sum = 0;
  while (idx > 0)
  {
    sum += fenwick[idx];
    idx -= idx & (-idx);
  }
  return sum;
}

void update(int idx, int n)
{
  while (idx <= maxIdx)
  {
    fenwick[idx] += n;
    idx += idx & (-idx);
  }
  return ;
}

int main()
{
  int i;
  memset(fenwick, 0, sizeof(fenwick));
  maxIdx = 5;
  for (i = 1; i <= 5; i++)
    update(i, +i);
  for (i = 1; i <= 5; i++)
    printf("%d\n", query(i));
  return 0;
}
```

5.3 Fenwick Tree_2D

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <algorithm>

using namespace std;
```

```
const int R = 502;
const int C = 502;

int r, c;
int a[R][C], s[R][C];

int query(int x, int y)
{
  int yy, ret = 0;
  while (x >= 1)
  {
    yy = y;
    while (yy >= 1)
    {
      ret += s[x][yy];
      yy -= yy & (-yy);
    }
    x -= x & (-x);
  }
  return ret;
}
void update(int x, int y, int v)
{
  int yy;
  while (x <= r)
  {
    yy = y;
    while (yy <= c)
    {
      s[x][yy] += v;
      yy += yy & (-yy);
    }
    x += x & (-x);
  }
}
void init()
{
  memset(s, 0, sizeof(s));
}

int main()
{
  int i, j;
  while (cin >> r >> c)
  {
    init();
    for (i = 1; i <= r; i++)
      for (j = 1; j <= c; j++)
      {
        scanf("%d", &a[i][j]);
        update(i, j, a[i][j]);
      }
    cout << query(r, c) << endl;
  }
  return 0;
}
```

5.4 Mo's Algorithm

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <utility>
#include <algorithm>
#include <vector>

using namespace std;

typedef pair<int, int> pii;
typedef unsigned long long ull;

int sq;
ull a[200005];
pair<pii, int> rng[200005];
ull cnt[1000005];
ull ans;
ull que[200005];

bool mo_cmp(pair<pii, int> l, pair<pii, int> r)
{
  const pii &a = l.first, &b = r.first;
  int asq = a.first / sq, bsq = b.first / sq;
  return asq != bsq ? asq < bsq : ((asq & 1) ? a.second <
     b.second : a.second > b.second);
}

void add(int i, int j)
{
  int k;
  for (k = i; k <= j; k++)
    cnt[a[k]]++, ans += (2 * cnt[a[k]] - 1) * a[k];
}

void rem(int i, int j)
{
  int k;
  for (k = i; k <= j; k++)
    cnt[a[k]]--, ans -= (2 * cnt[a[k]] + 1) * a[k];
}

int main()
{
  int i, n, q;
  int prev_l, prev_r;
  ans = 0;
  scanf("%d%d", &n, &q); sq = sqrt(n);
  memset(cnt, 0, sizeof(cnt));
  for (i = 0; i < n; i++) scanf("%d", &a[i]);
  for (i = 0; i < q; rng[i].second = i, rng[i].first.
    first--, rng[i].first.second--, i++) scanf("%d%d", &
    rng[i].first.first, &rng[i].first.second);
  sort(rng, rng + q, mo_cmp);
  prev_l = prev_r = 0; add(0, 0);
  for (i = 0; i < q; prev_l = rng[i].first.first, prev_r
    = rng[i].first.second, i++)
  {
    int &l = rng[i].first.first, &r = rng[i].first.second
    ;
    if (r >= prev_r) add(prev_r + 1, r); else rem(r + 1,
    prev_r);
    if (l >= prev_l) rem(prev_l, l - 1); else add(l,
    prev_l - 1);
    que[rng[i].second] = ans;
  }
  for (i = 0; i < q; i++)
    printf("%I64u\n", que[i]);
  return 0;
}
```

5.5 Kth Element_Partition Tree_Fractional Cascading

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <algorithm>
#include <vector>
#include <utility>

using namespace std;

class PartitionTreeNode;
class PartitionTree;


class PartitionTreeNode
```

```cpp
{
public:
  vector<int> y;
  vector<int> fl, fr;

  PartitionTreeNode() {}
  void resize(int _sz)
  {
    y.resize(_sz);
    fl.resize(_sz);
    fr.resize(_sz);
  }
};

class PartitionTree
{
public:
  PartitionTreeNode *nd;
  vector<pair<int, int> > ref;

  PartitionTree() {}
  ~PartitionTree() { delete [] nd; }
  PartitionTreeNode * get_root() { return (&nd[1]); }
  void ptree_merge(int idx, int l, int r);
  int ptree_get_kth(int idx, int ql, int qr, int k);
  int get_kth(int ql, int qr, int k)
  {
    return ptree_get_kth(1, ql, qr, k);
  }
  void build(int a[], int len)
  {
    int i;
    ref.resize(len);
    nd = new PartitionTreeNode[len << 2 | 3];
    for (i = 0; i < len; i++)
    {
      ref[i].first = a[i];
      ref[i].second = i;
    }
    sort(ref.begin(), ref.end());
    ptree_merge(1, 0, len - 1);
  }

};

void PartitionTree::ptree_merge(int idx, int l, int r)
{
  PartitionTreeNode *t = &nd[idx];
  t->resize(r - l + 1);
  if (l == r) { t->y[0] = ref[l].second; return ; }
  PartitionTreeNode *tl = &nd[idx << 1], *tr = &nd[idx <<
     1 | 1];
  int i, j, k, m = (l + r) >> 1;
  int llen = m - l + 1, rlen = r - m;
  ptree_merge(idx << 1, l, m);
  ptree_merge(idx << 1 | 1, m + 1, r);
  for (i = j = k = 0; i < llen && j < rlen; k++)
  {
    t->fl[k] = i; t->fr[k] = j;
    if (tl->y[i] < tr->y[j])
      t->y[k] = tl->y[i++];
    else
      t->y[k] = tr->y[j++];
  }
  for ( ; i < llen; k++) { t->fl[k] = i; t->fr[k] = j; t
    ->y[k] = tl->y[i++]; }
  for ( ; j < rlen; k++) { t->fl[k] = i; t->fr[k] = j; t
    ->y[k] = tr->y[j++]; }
}

int PartitionTree::ptree_get_kth(int idx, int ql, int qr,
     int k)
{
  PartitionTreeNode *t = &nd[idx];
  if (ql == qr) return t->y[ql];
  PartitionTreeNode *tl = &(nd[idx << 1]), *tr = &(nd[idx
      << 1 | 1]);
  int l1 = t->fl[ql], l2 = t->fr[ql];
  int r1 = t->fl[qr], r2 = t->fr[qr];
  if (r1 < tl->y.size() && tl->y[r1] == t->y[qr]) --r2;
    else --r1;
  if (k <= r1 - l1 + 1)
    return ptree_get_kth(idx << 1, l1, r1, k);
  else
    return ptree_get_kth(idx << 1 | 1, l2, r2, k - (r1 -
    l1 + 1));
}

int a[100005];
int main()
{
  int i, n, q;
  int ql, qr, k;
  while (scanf("%d%d", &n, &q) == 2)
  {
    PartitionTree ptree;
    for (i = 0; i < n; i++)
      scanf("%d", &a[i]);
    ptree.build(a, n);
    while (q--)
    {
      scanf("%d%d%d", &ql, &qr, &k); --ql; --qr;
      printf("%d\n", a[ptree.get_kth(ql, qr, k)]);
    }
  }
  return 0;
}
```

5.6 Discrete Segment Tree

```cpp
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
#include <map>
#include <queue>

using namespace std;
typedef long long ll;

const int N = 1e6 + 5;
const int X = 2e6 + 5;
const int S = 1e6;

vector<int> x, y;
int lx[X], ly[X];

struct node
{
  int mx, mxs, cov;
} s[N << 3];

struct event
{
  int x;
  int y1, y2;
  int u; // 1 for add, 0 for substract
  bool operator < (const event &rhs)const
  {
    if (x != rhs.x)
      return x < rhs.x;
    else
      return u > rhs.u;
  }
};
vector<event> a, m;

void build(int i, int l, int r)
{
  s[i].mx = s[i].cov = 0;
  s[i].mxs = y[r] - y[l]; // change
  if (r - l == 1) return ;
  int m = (l + r) >> 1;
  build(i << 1, l, m);
  build(i << 1 | 1, m, r);
}

int upd(int i, int l, int r, int ql, int qr, int u)
{
  if (qr <= l || ql >= r) return s[i].mx + s[i].cov;
  if (ql <= l && r <= qr)
  {
    s[i].cov += u;
    return s[i].mx + s[i].cov;
  }
  int m = (l + r) >> 1;
  s[i].mx = max(upd(i << 1, l, m, ql, qr, u), upd(i << 1
    | 1, m, r, ql, qr, u));
  // printf("[%d, %d]\n", l, r);
  // printf("%d %d\n", s[i << 1].mx + s[i << 1].cov, s[i
     << 1].mxs);
  // printf("%d %d\n", s[i<<1|1].mx + s[i<<1|1].cov, s[i
     <<1|1].mxs);
  s[i].mxs = s[i << 1].mx + s[i << 1].cov == s[i].mx ? s[
    i << 1].mxs : 0;
  s[i].mxs += s[i<<1|1].mx + s[i<<1|1].cov == s[i].mx ? s
    [i<<1|1].mxs : 0;
  // printf("s[%d].mx = %d, s[%d].mxs = %d, s[%d].cov = %
     d\n", i, s[i].mx, i, s[i].mxs, i, s[i].cov);
  return s[i].mx + s[i].cov;
}

int main()
{
  int i, tt, n, x1, y1, x2, y2, c;
  int xx, ai, mi;
  ll ans, anss;
  scanf("%d", &tt);
  while (tt--)
  {
    scanf("%d", &n);
    x.clear(); x.reserve(n << 1);
    y.clear(); y.reserve(n << 1);
    a.clear(); a.reserve(n << 1);
    m.clear(); m.reserve(n << 1);
    ai = mi = 0;
    ans = 0;
    for (i = 0; i < n; i++)
    {
      scanf("%d%d%d%d%d", &x1, &y1, &x2, &y2, &c);
      x.push_back(x1); x.push_back(x2);
      y.push_back(y1); y.push_back(y2);
      if (x1 > x2) swap(x1, x2);
      if (y1 > y2) swap(y1, y2);
      a.push_back( (event){x1, y1, y2, c} );
      m.push_back( (event){x2, y1, y2, -c} );
    }
    sort(x.begin(), x.end());
    sort(y.begin(), y.end());
    unique(x.begin(), x.end());
    unique(y.begin(), y.end());
    for (i = 0; i < x.size(); i++)
      lx[x[i] + S] = i;
    for (i = 0; i < y.size(); i++)
      ly[y[i] + S] = i;
    // for (i = 0; i < x.size(); i++)
    //  printf("%d %d\n", i, x[i]);
    sort(a.begin(), a.end());
    sort(m.begin(), m.end());
    build(1, 0, y.size() - 1);
    for (xx = 1; xx < x.size(); xx++)
    {
//      puts("*");
      while (ai < a.size() && lx[a[ai].x + S] < xx)
      {
        upd(1, 0, y.size() - 1, ly[a[ai].y1 + S], ly[a[ai
      ].y2 + S], a[ai].u);
        ai++;
      }
      while (mi < m.size() && lx[m[mi].x + S] < xx)
      {
        upd(1, 0, y.size() - 1, ly[m[mi].y1 + S], ly[m[mi
      ].y2 + S], m[mi].u);
        mi++;
      }
      if (s[1].mx + s[1].cov > ans)
      {
        ans = s[1].mx + s[1].cov;
        anss = (ll)s[1].mxs * (x[xx] - x[xx - 1]);
      }
      else if (s[1].mx + s[1].cov == ans)
        anss += (ll)s[1].mxs * (x[xx] - x[xx - 1]);
    }
    printf("%lld %lld\n", ans, anss);
  }
  return 0;
}
```

5.7 Treap

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#define MAX_N 100005
#define MAX_M 50005

using namespace std;

struct treap;
int size(treap* &);

struct treap
{
  int pri;
  int key, size;
  treap *chi[2];
  treap(): pri(rand()), size(1) { chi[0] = chi[1] = NULL;
     }

  void pull()
  {
    size = ::size(chi[0]) + ::size(chi[1]) + 1;
  }
} *root = NULL;

struct range
{
  int i;
  int l, r, rk;
  bool operator < (const range &rhs)const
  {
    return l < rhs.l;
  }
} r[MAX_M];
int out[MAX_M];

int size(treap* &cur)
{
  return cur ? cur -> size : 0;
}

void split(treap *cur, const int &key, treap* &l, treap*
    &r)
{
  if (!cur) { l = r = NULL; return ; }
  if (cur->key <= key)
  {
    l = cur;
    split(l->chi[1], key, l->chi[1], r);
    l->pull();
  }
  else
  {
    r = cur;
    split(r->chi[0], key, l, r->chi[0]);
    r->pull();
  }
}

treap * merge(treap *l, treap *r) // l->key < r->key, for
     all children
{
  if (!l) return r;
  if (!r) return l;
  if (l->pri > r->pri)
  {
    l->chi[1] = merge(l->chi[1], r);
    l->pull();
    return l;
  }
  else
  {
    r->chi[0] = merge(l, r->chi[0]);
    r->pull();
    return r;
  }
}

void insert(const int &key)
{
  if (root == NULL) { root = new treap; root->key = key;
    return ; }
  treap *l, *r, *n = new treap;
  n->key = key;
  split(root, key, l, r);
  root = merge(l, n);
  root = merge(root, r);
}

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

void del(const int &key)
{
  treap *l, *r, *ll, *lr, *n;
  split(root, key, l, r);
  if (!l) return ;
  // if exists key
  split(l, key-1, ll, lr);
  // drop 1 from lr
  n = merge(lr->chi[0], lr->chi[1]); // can be null
  n = merge(ll, n); // can be null
  root = merge(n, r);
  delete lr;
}

treap *getRk(int rk, treap* &cur)
{
  int myRk = size(cur->chi[0]) + 1;
  if (myRk == rk) return cur;
  if (myRk < rk)
    return getRk(rk - myRk, cur->chi[1]);
  else
    return getRk(rk, cur->chi[0]);
}

void pre(treap *n)
{
  if (!n) return ;
  printf("%d(%d) ", n->key, size(n));
  pre(n->chi[0]);
  pre(n->chi[1]);
}

void in(treap *n)
{
  if (!n) return ;
  in(n->chi[0]);
  printf("%d(%d) ", n->key, size(n));
  in(n->chi[1]);
}

int a[MAX_N];
int main()
{
  int i, j, n, m, prev_l, prev_r, lb, ub;
  treap *ans;
  srand(time(0));

  scanf("%d%d", &n, &m);
  if (!m) return 0;

  for (i = 1; i <= n; i++)
    scanf("%d", &a[i]);
  // root = NULL, declared above
  for (i = 0; i < m; i++)
  {
    r[i].i = i;
    scanf("%d%d%d", &r[i].l, &r[i].r, &r[i].rk);
  }
  sort(r, r + m);

  for (i = r[0].l; i <= r[0].r; i++)
    insert(a[i]);
  ans = getRk(r[0].rk, root);
  // in(root); puts("");
  out[r[0].i] = ans->key;
  prev_l = r[0].l;
  prev_r = r[0].r;
  for (i = 1; i < m; prev_l = r[i].l, prev_r = r[i].r, i
    ++)
  {
    range &ri = r[i];
    ub = min(ri.l - 1, prev_r);
    for (j = prev_l; j <= ub; j++)
      del(a[j]);
    lb = max(ri.l, prev_r + 1);
    for (j = lb; j <= ri.r; j++)
      insert(a[j]);
    ans = getRk(ri.rk, root);
    out[ri.i] = ans->key;
  }
  for (i = 0; i < m; i++)
    printf("%d\n", out[i]);
  return 0;
}
```

5.8 Treap,NoKey

```
#include <cstdio>
#include <cstring>
#include <ctime>
#include <cstdlib>
#include <cmath>
#include <algorithm>

using namespace std;

struct treap;
int size(treap *&);

#define L(X) X->c[0^accu_rev]
#define R(X) X->c[1^accu_rev]
struct treap
```

```
{
  int val, pri, sz;
  int mi, add;
  char rev;
  treap *c[2];
  treap(int _val = 0): val(_val), sz(1), pri(rand()), mi(
    _val), add(0), rev(0)
  {
    c[0] = c[1] = NULL;
  }
  void push(char &_rev)
  {
    _rev ^= rev;
  }
  void pull()
  {
    sz = size(c[0]) + size(c[1]) + 1;
    mi = val;
    if (c[0]) mi = min(mi, c[0]->mi + c[0]->add);
    if (c[1]) mi = min(mi, c[1]->mi + c[1]->add);
  }
} *root = NULL;

int size(treap *&t) { return t ? t->sz : 0; }

void split(treap *t, int rk, treap *&l, treap *&r, char
    accu_rev = 0)
{
  if (!t) { l = r = NULL; return ; }
  t->push(accu_rev);
  treap *&tl = L(t), *&tr = R(t);
  if (size(tl) < rk)
  {
    l = t;
    split(tr, rk - size(tl) - 1, tr, r, accu_rev);
    if (r) {
      r->add += t->add;
      r->rev ^= t->rev;
    }
  }
  else
  {
    r = t;
    split(tl, rk, l, tl, accu_rev);
    if (l) {
      l->add += t->add;
      l->rev ^= t->rev;
    }
  }
  t->pull();
}

treap * merge(treap *&l, treap *&r, char accu_rev_l = 0,
    char accu_rev_r = 0)
{
  if (!l) return r;
  if (!r) return l;
  l->push(accu_rev_l);
  r->push(accu_rev_r);
  if (l->pri > r->pri)
  {
    r->add -= l->add;
    r->rev ^= l->rev;
    char &accu_rev = accu_rev_l;
    r->push(accu_rev_r);
    R(l) = merge(R(l), r, accu_rev_l, accu_rev_r);
    l->pull();
    return l;
  }
  else
  {
    l->add -= r->add;
    l->rev ^= r->rev;
    char &accu_rev = accu_rev_r;
    l->push(accu_rev_l);
    L(r) = merge(l, L(r), accu_rev_l, accu_rev_r);
    r->pull();
    return r;
  }
}

void insert(const int &val, const int &pos) // pos = [1,]
{
  treap *l, *r, *n = new treap;
  n->val = n->mi = val;
  split(root, pos-1, l, r);
  root = merge(l, n);
  root = merge(root, r);
}

void del(const int &rk)
{
  treap *l, *r, *ll, *lr;
  split(root, rk, l, r);
  split(l, rk-1, ll, lr);
  root = merge(ll, r);
  delete lr;
}

int query(int type, int i, int j, int n = 0) // i, j =
    [1,]
{
  int ret;
  treap *l, *r, *rl, *rr;
  if (i > j) swap(i, j);
  split(root, i-1, l, r);
  split(r, j-i+1, rl, rr);
  switch (type)
  {
    case 0: // min
      ret = rl->mi + rl->add;
      break;
    case 1: // add
      rl->add += n;
      ret = 1; //meaningless
      break;
    case 2: // reverse
      rl->rev ^= 1;
      ret = 1;
      break;
    case 3: // revolve
      n %= (j - i + 1);
      treap *ql, *qr;
      split(rl, j - i + 1 - n, ql, qr);
      rl = merge(qr, ql);
      ret = 1;
      break;
  }
  root = merge(rl, rr);
  root = merge(l, root);
  return ret;
}

void in(treap *t, char accu_rev = 0)
{
  if (!t) return ;
  t->push(accu_rev);
  in(L(t), accu_rev);
  printf("%d(%d) ", t->val, size(t));
  in(R(t), accu_rev);
  return ;
}

int main()
{
  int i, j, n, m, ai, qi, qj, qn;
  char s[100], cmd[20];
  srand(time(0));
  scanf("%d", &n);
  for (i = 1; i <= n; i++)
  {
    scanf("%d", &ai);
    insert(ai, i);
  }
  scanf("%d", &m);
  while (m--)
  {
    scanf(" ");
    gets(s);
    sscanf(s, "%s", cmd);
    if (strcmp(cmd, "ADD") == 0)
    {
      sscanf(s, "%s %d %d %d", cmd, &qi, &qj, &qn);
      query(1, qi, qj, qn);
    }
    else if (strcmp(cmd, "REVERSE") == 0)
    {
      sscanf(s, "%s %d %d", cmd, &qi, &qj);
      query(2, qi, qj);
    }
    else if (strcmp(cmd, "REVOLVE") == 0)
    {
      sscanf(s, "%s %d %d %d", cmd, &qi, &qj, &qn);
      query(3, qi, qj, qn);
    }
    else if (strcmp(cmd, "INSERT") == 0)
    {
      sscanf(s, "%s %d %d", cmd, &qi, &qn);
      insert(qn, qi+1);
    }
    else if (strcmp(cmd, "DELETE") == 0)
    {
      sscanf(s, "%s %d", cmd, &qn);
      del(qn);
    }
    else if (strcmp(cmd, "MIN") == 0)
    {
      sscanf(s, "%s %d %d", cmd, &qi, &qj);
      printf("%d\n", query(0, qi, qj));
    }
  }
  return 0;
}
```

5.9 Persistent Treap

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <cmath>
#define ROOT root[root_iter]

using namespace std;

const int MAX_VER = 50002;
const int MAX_LEN = 105;

struct treap;
int size(treap *);

struct treap
{
  char val;
  int pri, sz;
  treap *l, *r;
  treap(int _val = 0): val(_val), pri(rand()), sz(1), l(
    NULL), r(NULL){}

  void pull()
  {
    sz = size(l) + size(r) + 1;
  }
};
```

```
treap *root[MAX_VER];
int root_iter;
int cnt_c;
char s[MAX_LEN];

int size(treap *t) { return t ? t->sz : 0; }

treap * copy(treap *t)
{
  treap *n = new treap;
  *n = *t;
  return n;
}

void in(treap *t)
{
  if (!t) return ;
  in(t->l);
  putchar(t->val);
  if (t->val == 'c') cnt_c++;
  in(t->r);
}

void split(treap *t, const int &rk, treap * &l, treap * &
    r)
{
  if (!t) { l = r = NULL; return ; }
  treap *t2 = copy(t);
  if (size(t->l) < rk)
  {
    l = t2;
    split(t2->r, rk - size(t->l) - 1, t2->r, r);
  }
  else
  {
    r = t2;
    split(t2->l, rk, l, t2->l);
  }
  t2->pull();
}

treap * merge(treap *l, treap *r)
{
  if (!l) return r;
  if (!r) return l;
  if (l->pri > r->pri)
  {
    treap *n = copy(l);
    n->r = merge(n->r, r);
    n->pull();
    return n;
  }
  else
  {
    treap *n = copy(r);
    n->l = merge(l, n->l);
    n->pull();
    return n;
  }
}

void _split(treap *t, const int &rk, treap * &l, treap *
    &r)
{
  if (!t) { l = r = NULL; return ; }
  if (size(t->l) < rk)
  {
    l = t;
    split(t->r, rk - size(t->l) - 1, t->r, r);
  }
  else
  {
    r = t;
    split(t->l, rk, l, t->l);
  }
  t->pull();
}

treap * _merge(treap *l, treap *r)
{
  if (!l) return r;
  if (!r) return l;
  if (l->pri > r->pri)
  {
    l->r = merge(l->r, r);
    l->pull();
    return l;
  }
  else
  {
    r->l = merge(l, r->l);
    r->pull();
    return r;
  }
}

void insert(char s[], int len, int pos) // insert after
    pos
{
  int i;
  treap *l, *r, *n, *ins = NULL;
  split(ROOT, pos, l, r);
  ++root_iter;
  for (i = 0; i < len; i++)
  {
    n = new treap;
    n->val = s[i];
    ins = _merge(ins, n);
  }
  ROOT = merge(l, ins);
  ROOT = merge(ROOT, r);
}

void remove(const int &st, const int &len)
{
  treap *l, *r, *ll, *lr;
  split(ROOT, st+len-1, l, r);
  split(l, st-1, ll, lr);
  ++root_iter;
  ROOT = merge(ll, r);
}

void print(const int &ver, const int &st, const int &len)
{
  treap *l, *r, *ll, *lr;
  treap *&rt = root[ver];
  _split(rt, st+len-1, l, r);
  _split(l, st-1, ll, lr);
  in(lr); puts("");
  rt = _merge(ll, lr);
  rt = _merge(rt, r);
}

void init()
{
  srand(time(0));
  memset(root, 0, sizeof(root));
  root_iter = 0; // can be -1
  cnt_c = 0;
}

int main()
{
  int i, qs, cmd, pos, len, ver, l;
  init();
  scanf("%d", &qs);
  while (qs--)
  {
    scanf("%d", &cmd);
    switch (cmd)
    {
      case 1:
        scanf("%d %s", &pos, s);
        l = strlen(s);
        pos -= cnt_c;
        insert(s, l, pos + i);
        break;
      case 2:
        scanf("%d %d", &pos, &len);
        pos -= cnt_c;
        len -= cnt_c;
        remove(pos, len);
        break;
      case 3:
        scanf("%d %d %d", &ver, &pos, &len);
        ver -= cnt_c;
        pos -= cnt_c;
        len -= cnt_c;
        print(ver, pos, len);
        break;
    }
  }
  return 0;
}
```

6 String

6.1 KMP Algorithm

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <algorithm>

using namespace std;

const int L = 200005;

void cal_fail(char s[], int len, int f[])
{
  int i, j;
  for (i = 0; i < len; i++)
  {
    j = i - 1;
    while (j > -1 && s[f[j] + 1] != s[i]) j = f[j];
    f[i] = j > -1 ? f[j] + 1 : -1;
  }
}

int kmp(char s[], int len, char p[], int plen, int f[])
{
  int i, j;
  for (i = 0, j = 0; i < len; i++)
  {
    while (j > -1 && s[i] != p[j]) j = j ? f[j - 1] + 1 :
     -1;
    if (++j == plen) return i; // j = f[j - 1] + 1
  }
  return -1;
}

char s[L], *p;
int f[L];

int main()
{
  int i, len;
  while (gets(s))
  {
    len = strlen(s);
    for (i = 0; i < len; i++)
      s[len + i] = s[len - i - 1];
    p = &s[len]; p[len] = '\0';
    puts(p);
    cal_fail(p, len, f);
    for (i = 0; i < len; i++)
      printf("%d ", f[i]);
    puts("");
//    len <<= 1;
//    s[len] = '\0';
//    puts(s);

  }
  return 0;
}
```

6.2 Suffix Array_LCP Array

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <utility>
#include <vector>
#include <string>
#include <unordered_set>
#include <algorithm>

using namespace std;

const int L = 605;
const int I = 20; // log2(MAX_LEN) + 2, because 0 to ceil
     (log2(MAX_LEN)) is needed

struct suf
{
  int rk1, rk2;
  int i;
  bool operator < (const suf &rhs)const
  {
    if (rk1 != rhs.rk1)
      return rk1 < rhs.rk1;
    else
      return rk2 < rhs.rk2;
  }
  bool operator == (const suf &rhs)const
  {
    return rk1 == rhs.rk1 && rk2 == rhs.rk2;
  }
};

int rk[I][L];
int it; // number of iterations
suf tmp[L];
suf real_suf[L];
unordered_set<string> pr;

void build(const char *s, const int &len)
{
  int i, l;
  for (i = 0; i < len; i++)
    rk[0][i] = s[i];
  for (it = 1, l = 1; l < len; it++, l <<= 1)
  {
    for (i = 0; i < len; i++)
    {
      tmp[i].i = i;
      tmp[i].rk1 = rk[it - 1][i];
      tmp[i].rk2 = i + l < len ? rk[it - 1][i + l] : -1;
    }
    sort(tmp, tmp + len);
    for (i = 0; i < len; i++)
      rk[it][tmp[i].i] = tmp[i] == tmp[i - 1] ? rk[it][
    tmp[i - 1].i] : i;
  }
  for (i = 0; i < len; i++)
    real_suf[i] = (suf){ rk[it - 1][i], 0, i };
  sort(real_suf, real_suf + len);
}

int lcp(const char *s, const int &len, int i, int j)
{
  int k, ret = 0;
  for (k = it - 1; k >= 0 && i < len && j < len; k--)
    if (rk[k][i] == rk[k][j])
    {
      i += 1 << k;
      j += 1 << k;
      ret += 1 << k;
    }
  return ret;
}

char s[L];
int main()
{
  int i, j, l1, len, lans, cp;
  vector<int> ians;
  int tt = 0;
  while (gets(s))
  {
    pr.clear();
    if (tt++) puts("");
    lans = 0;
    l1 = strlen(s); s[l1] = -2;
    gets(&s[l1 + 1]);
    len = l1 + 1 + strlen(&s[l1 + 1]);
    build(s, len);
    for (i = 0; i < len - 1; i++)
    {
      if (s[real_suf[i].i] <= 0) continue;
      int i1 = real_suf[i].i, i2 = real_suf[i + 1].i;
      if ( (i1 < l1 && i2 < l1) || (i1 > l1 && i2 > l1) )
     continue;
      if (i1 > i2) swap(i1, i2);
      cp = lcp(s, len, i1, i2);
      if (cp > lans)
      {
        lans = cp;
        ians.clear();
        ians.push_back(i1);
      }
      else if (cp == lans)
        ians.push_back(i1);
    }
    if (lans == 0)
      puts("No common sequence.");
    else
    {
//        printf("%d\n", lans);
      for (i = 0; i < ians.size(); i++)
      {
        char tmp = s[ians[i] + lans];
        s[ians[i] + lans] = '\0';
        auto it = pr.find(string(&s[ians[i]]));
        if (it == pr.end())
        {
          for (j = 0; j < lans; j++)
            putchar(s[ians[i] + j]);
          puts("");
          pr.insert(string(&s[ians[i]]));
        }
        s[ians[i] + lans] = tmp;
      }
    }
    gets(s);
  }
  return 0;
}
```

7 Geometry

7.1 Convex Hull_Monotone Chain

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <cstdlib>
#include <algorithm>
#include <vector>

using namespace std;

const int N = 5005;

struct pt
{
  int x, y;
} a[N];
int n;
int stk[N], top;

bool cmp(pt a, pt b)
{
  if (a.x != b.x)
    return a.x < b.x;
  else
    return a.y < b.y;
}

int cross(pt c, pt b, pt a)
{
  int dx1 = c.x - b.x, dy1 = c.y - b.y;
  int dx2 = b.x - a.x, dy2 = b.y - a.y;
  return dx1 * dy2 - dy1 * dx2;
}

double dist(pt a, pt b)
{
  int dx = a.x - b.x, dy = a.y - b.y;
  return sqrt(dx * dx + dy * dy);
}

void mtone()
{
  int i, j, fhalf;
  top = 0;
  sort(a, a + n, cmp);
  for (i = 0; i < n; i++)
  {
    if (top < 2)
      stk[++top] = i;
    else
    {
      while (top >= 2 && cross(a[i], a[stk[top]], a[stk[
    top - 1]]) >= 0) --top;
      stk[++top] = i;
    }
  }
  fhalf = top;
  for (i = n - 1; i >= 0; i--)
  {
    if (top - fhalf < 2)
      stk[++top] = i;
    else
    {
      while (top - fhalf >= 2 && cross(a[i], a[stk[top]],
     a[stk[top - 1]]) >= 0) --top;
      stk[++top] = i;
    }
  }
}

int main()
```

```
{
  int i, t;
  double ans;
  scanf("%d", &t);
  while (t--)
  {
    ans = 0;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
      scanf("%d%d", &a[i].x, &a[i].y);
    mtone();
    for (i = 1; i < top; i++)
      ans += dist(a[stk[i]], a[stk[i + 1]]);
    printf("%.3f\n", ans);
  }
  return 0;
}
```

7.2 Intersecting Halfplanes

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <queue>
#include <cmath>
#include <algorithm>
#define EPS 1e-9

using namespace std;

typedef long long LL;

const int N = 505;

struct vec;
struct pt;
struct line;

struct vec
{
  double x, y;
  vec(){}
  vec(double _x, double _y): x(_x), y(_y){}
};

struct pt
{
  double x, y;
  pt(){}
  pt(double _x, double _y): x(_x), y(_y){}
  vec operator - (const pt &rhs)
  {
    vec tmp(x - rhs.x, y - rhs.y);
    return tmp;
  }
};

struct line
{
  double a, b; // expressed as y = ax + b
  char v;
  static double inf;
  line(){}
  line(double _a, double _b, char _v = 0): a(_a), b(_b),
    v(_v){}
};
double line::inf = 1e9;

// l is a line
inline bool above(const int &x, const int &y, const pt &l
    )
{
  return y > l.x * x - l.y + EPS;
}

double cross(const vec &a, const vec &b)
{
  return a.x * b.y - a.y * b.x;
}

bool pt_cmp(pt a, pt b)
{
  if (a.x != b.x)
    return a.x < b.x;
  else
    return a.y < b.y;
}

bool pt_cmp_rev(pt a, pt b)
{
  if (a.x != b.x)
    return a.x > b.x;
  else
    return a.y > b.y;
}

//bool parallel(const line &a, const line &b)
//{
//  return abs(a.a - b.a) < EPS;
//}

// negate y
// a = x, b = -y
pt intersect(const pt &a, const pt &b)
{
  double x, y;
  if (a.x == b.x)
    return pt(line::inf, line::inf);
  x = (a.y - b.y) / (a.x - b.x);
  y = a.x * x - a.y;

  return pt(x, y);
}

int stk[N], t[N];
void monotone_half(pt p[], int &sz, char rev = 0)
{
  int i, t = -1;
  if (!rev) sort(p, p + sz, pt_cmp);
  else sort(p, p + sz, pt_cmp_rev);
  for (i = 0; i < sz; i++)
  {
    if (t < 1){ stk[++t] = i; continue; }
    while (t >= 1 && cross(p[i] - p[stk[t]], p[stk[t]] -
    p[stk[t-1]]) >= 0) --t;
    stk[++t] = i;
  }
  for (i = 0; i <= t; i++) p[i] = p[stk[i]];
  sz = t + 1;
}

inline int input()
{
  char c;
  for (c = getchar(); c < 48 || c > 57; c = getchar() ) ;
  int x = c - 48;
  for (c = getchar(); c > 47 && c < 58; c = getchar() ) x
     = x * 10 + c - 48;
  return x;
}


pt pt_l[N], pt_u[N]; int sz_l, sz_u;
int lns_v[N]; int sz_v; // vertical lines
LL w, h;

int main()
{
```

```cpp
//  freopen("11265.in", "r", stdin);
//  freopen("11265.txt", "w", stdout);
  int i, j, tt = 0, n;
  LL x1, y1, x2, y2;
  int ref_x, ref_y;
  int it_l, it_u;
  pt i_i, i_l, i_u;
  double prev_dy, dy, nxt_x, ans;
  char st;
  pt tmp;
  while (scanf("%d", &n) == 1)
  {
    w = input(); h = input();
    sz_l = sz_u = sz_v = 0;
    it_l = it_u = 0;
    st = 0;
    ans = 0.0;
    ref_x = input(); ref_y = input();
    pt_l[sz_l++] = pt(0, 0);
    pt_u[sz_u++] = pt(0, -h);
    lns_v[sz_v++] = 0;
    lns_v[sz_v++] = w;
    for (i = 0; i < n; i++)
    {
      x1 = input(); y1 = input(); x2 = input(); y2 =
    input();
      if (x1 == x2) // vertical line
        lns_v[sz_v++] = x1;
      else
      {
        tmp = pt(double(y2 - y1) / (x2 - x1), double(y1 *
     x2 - y2 * x1) / (x1 - x2));
        if (above(ref_x, ref_y, tmp))
          pt_l[sz_l++] = tmp;
        else
          pt_u[sz_u++] = tmp;
      }
    }
    monotone_half(pt_l, sz_l, 0);
    monotone_half(pt_u, sz_u, 1);

    if (sz_l >= 2 && pt_l[sz_l - 1].x == pt_l[sz_l - 2].x
    ) --sz_l;
    if (sz_u >= 2 && pt_u[sz_u - 1].x == pt_u[sz_u - 2].x
    ) --sz_u;

    sort(lns_v, lns_v + sz_v);
    int idx = upper_bound(lns_v, lns_v + sz_v, ref_x) -
    lns_v;

    for ( ; it_l < sz_l - 1; it_l++) if (intersect(pt_l[
    it_l], pt_l[it_l + 1]).x >= lns_v[idx - 1]) break;
    for ( ; it_u < sz_u - 1; it_u++) if (intersect(pt_u[
    it_u], pt_u[it_u + 1]).x >= lns_v[idx - 1]) break;
    for (double x = lns_v[idx - 1]; x < lns_v[idx] - EPS;
     )
    {
      i_i = intersect(pt_l[it_l], pt_u[it_u]);
      i_l = it_l < sz_l - 1 ? intersect(pt_l[it_l], pt_l[
    it_l + 1]) : pt(1e9, 1e9);
      i_u = it_u < sz_u - 1 ? intersect(pt_u[it_u], pt_u[
    it_u + 1]) : pt(1e9, 1e9);
      if (st == 0 && i_i.x >= x && pt_u[it_u].x * x -
    pt_u[it_u].y - (pt_l[it_l].x * x - pt_l[it_l].y) >=
    0.0) i_i = pt(-1e9, -1e9);
      if (st == 0 && i_i.x <= i_l.x && i_i.x <= i_u.x)
      {
        nxt_x = max(x, i_i.x);
        dy = i_i.x <= x ? pt_u[it_u].x * x - pt_u[it_u].y
     - (pt_l[it_l].x * x - pt_l[it_l].y) : 0.0;
        st = 1;
        x = nxt_x; // not counting the first edge (the
    area between the previous edge and the first edge)
      }
      else if (i_i.x > x + EPS && i_i.x <= i_l.x && i_i.x
       <= i_u.x)
      {
        nxt_x = min(i_i.x, (double)lns_v[idx]);
        dy = i_i.x <= lns_v[idx] ? 0 : pt_u[it_u].x *
    lns_v[idx] - pt_u[it_u].y - (pt_l[it_l].x * lns_v[
    idx] - pt_l[it_l].y);
        st = 2;
      }
      else if (i_l.x <= i_u.x)
      {
        nxt_x = min(i_l.x, (double)lns_v[idx]);
        dy = pt_u[it_u].x * nxt_x - pt_u[it_u].y - (pt_l[
    it_l].x * nxt_x - pt_l[it_l].y);
        it_l++;
      }
      else // if (i_u.x <= i_l.x)
      {
        nxt_x = min(i_u.x, (double)lns_v[idx]);
        dy = pt_u[it_u].x * nxt_x - pt_u[it_u].y - (pt_l[
    it_l].x * nxt_x - pt_l[it_l].y);
        it_u++;
      }

      if (st && dy < -EPS) break;

      if (st) ans += (dy + prev_dy) * (nxt_x - x) * 0.5;
      if (st == 2) break;
      prev_dy = dy;
      x = nxt_x;
    }
    printf("Case #%d: %.3f\n", ++tt, ans);
  }
  return 0;
}
```