Contents

1 Basic 1
- 1.1 .vimrc . . . 1
- 1.2 Check . . . 1
- 1.3 Factor Count List . . . 1
- 1.4 Default . . . 1
- 1.5 Pragma . . . 1
- 1.6 Random Number . . . 1
- 1.7 Increase Stack Size . . . 2
- 1.8 FasterIO . . . 2

2 Bitwise Trick 2
- 2.1 Builtin Function . . . 2
- 2.2 Subset Enumeration . . . 2
- 2.3 Next Permutation on Binary . . . 2
- 2.4 SOS DP . . . 2

3 Theorem and Formula 2

4 Data Structure 2
- 4.1 <ext/pb_ds> . . . 2
- 4.2 Unordered Map Hash . . . 2
- 4.3 Rope . . . 3
- 4.4 Disjoint Set . . . 3
- 4.5 Persistent Treap . . . 3
- 4.6 Link Cut Tree . . . 3
- 4.7 Li Chao Tree . . . 4
- 4.8 Dancing Link . . . 4
- 4.9 Range Modify and Query BIT . . . 5

5 Flow 5
- 5.1 ISAP with bound . . . 5
- 5.2 Min Cost Max Flow . . . 5
- 5.3 S-W Global Min Cut . . . 6
- 5.4 Gomory Hu Tree . . . 6

6 Tree 6
- 6.1 Minimum Steiner Tree . . . 6
- 6.2 Zhu Liu Algo . . . 7
- 6.3 Centroid Decomposition . . . 7
- 6.4 Dynamic MST . . . 8
- 6.5 Heavy-Light Decomposition . . . 8
- 6.6 Block tree . . . 9

7 Graph 9
- 7.1 Biconnected Component . . . 9
- 7.2 General Graph Macthing . . . 9
- 7.3 KM . . . 10
- 7.4 Maximum Weighted Matching(General Graph) . . . 10
- 7.5 Minimum Mean Cycle . . . 11
- 7.6 Maximum Clique . . . 12

8 Math 12
- 8.1 Extended Euclidean . . . 12
- 8.2 Gaussian Elimination . . . 12
- 8.3 Linear Basis . . . 12
- 8.4 Build Prime . . . 13
- 8.5 Miller Rabin . . . 13
- 8.6 Pollard Rho . . . 13
- 8.7 Build Phi and Mu . . . 13
- 8.8 Primitive Root . . . 13
- 8.9 Cipolla's Algorithm . . . 13
- 8.10 Discrete Log . . . 14
- 8.11 Integer Partition . . . 14
- 8.12 Meissel-Lehmer Algorithm . . . 14
- 8.13 De Bruijn . . . 14
- 8.14 Simplex Algorithm . . . 14
- 8.15 Middle Speed Linear Recursion . . . 15
- 8.16 Chinese Remainder Theorem . . . 15

9 Convolution 15
- 9.1 FFT . . . 15
- 9.2 NTT . . . 15
- 9.3 FWT . . . 16
- 9.4 Subset Convolution . . . 16
- 9.5 Ternary Xor . . . 16

10 String 17
- 10.1 KMP . . . 17
- 10.2 Z value . . . 17
- 10.3 Longest Palindrome . . . 17
- 10.4 Aho-Corasick Algorithm . . . 17
- 10.5 Suffix Array . . . 17
- 10.6 Palindromic Tree . . . 18
- 10.7 Lexicographically Smallest Rotation . . . 18

11 Geometry 19
- 11.1 Circle . . . 19
- 11.2 Half Plane Intersection . . . 19
- 11.3 Convex Hull 3D . . . 19
- 11.4 Dynamic convexhull . . . 20
- 11.5 Polar Angle Sort . . . 20
- 11.6 Circle and Polygon intersection . . . 20
- 11.7 Segment Intersection . . . 21
- 11.8 Line Intersection Point . . . 21
- 11.9 Rotating Calipers . . . 21

12 Miscellaneous 21
- 12.1 Joseph Problem . . . 21
- 12.2 Segment Max Segment Sum . . . 21
- 12.3 Stone Merge . . . 22
- 12.4 Manhattan Spanning Tree . . . 22
- 12.5 K Cover Tree . . . 22
- 12.6 M Segments' Maximum Sum . . . 23
- 12.7 Minimum Enclosing Cycle . . . 23
- 12.8 Rotating Sweep Line . . . 23
- 12.9 Hilbert Curve . . . 24
- 12.10 Big Integer . . . 24

1 Basic

1.1 .vimrc

```
syntax on
se ru nu ai
se ts=4 sts=4 sw=4 st=4 smarttab laststatus=2 expandtab
inoremap {<ENTER> {}<LEFT><ENTER>;<LEFT><DEL><ENTER><UP><TAB>
"se mouse=a expandtab
```

1.2 Check

```
for i in $(seq 1 10000);
do
    ./gen > input
    ./ac < input > out_ac
    ./wa < input > out_wa
    diff out_ac out_wa || break
done
```

1.3 Factor Count List

```
/*
(i, factor number of i)
10080     72,   50400     108,  110880    144,
221760    168,  332640    192,  498960    200,
554400    216,  665280    224,  720720    240,
1081080   256,  2162160   320,  3603600   360,
4324320   384,  6486480   400,  7207200   432,
8648640   448,  10810800  480,  21621600  576,
32432400  600,  43243200  672,  61261200  720,
73513440  768,  110270160 800,  245044800 1008,
367567200 1152, 551350800 1200, 698377680 1280,
735134400 1344, 1102701600 1440, 1396755360 1536
*/
```

1.4 Default

```
// Compile with "g++ -std=c++11 -Wall -Wextra -Wconversion -
    Wshadow -fsanitize=undefined -Dlawfung"
#ifdef lawfung
#define debug(...) do {\
  fprintf(stderr, "%s - %d : (%s) = ", __PRETTY_FUNCTION__,
      __LINE__, #__VA_ARGS__);\
  _DO(__VA_ARGS__);\
}while(0)
template<typename I> void _DO(I&&x) {cerr << x << '\n';}
template<typename I, typename ...T> void _DO(I&&x,T&&...tail) {
    cerr << x << ", "; _DO(tail...);}
#define IOS
#else
#define debug(...)
#define IOS ios_base::sync_with_stdio(0);cin.tie(0)
#endif
```

1.5 Pragma

```
#pragma GCC optimize("Ofast", "unroll-loops")
#pragma GCC optimize("no-stack-protector")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,sse4.2,popcnt,abm,
    mmx,avx,tune=native")
#pragma GCC diagnostic ignored "-W"
```

1.6 Random Number

```
#include <random>
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
int rand_int(int lb, int ub)
{ return uniform_int_distribution<int>(lb, ub)(rng); }
double rand_double(double lb, double ub)
{ return uniform_real_distribution<double>(lb, ub)(rng); }
```

1.7 Increase Stack Size

```
const int size = 256 << 20;
register long rsp asm("rsp");
char *p = (char*)malloc(size) + size, *bak = (char*)rsp;
__asm__("movq %0, %%rsp\n"::"r"(p));
// main
__asm__("movq %0, %%rsp\n"::"r"(bak));
```

1.8 FasterIO

```
static inline char getRawChar() {
  static char buf[1 << 16], *p = buf, *end = buf;
  if (p == end) {
    if ((end = buf + fread_unlocked(buf, 1, 1 << 16, stdin)) ==
         buf) return '\0';
    p = buf;
  }
  return *p++;
}
while (c = getRawChar() && (unsigned)(c - '0') > 10U) n = n *
    10 + (c - '0');
```

2 Bitwise Trick

2.1 Builtin Function

```
// count left 0s
int __builtin_clz (unsigned int x) // 31 - __builtin_clz is lg
int __builtin_clzll (unsigned long long x) // 63 - clz
// count number of 1's
int __builtin_popcount (unsigned int x)
int __builtin_popcountll (unsigned long long x)
```

2.2 Subset Enumeration

```
int subset_enumeration(int s) {
  for (int now = s; now > 0; now = (now - 1) & s) {
    cout << now << ' ';
  }
  cout << "0\n";
}
```

2.3 Next Permutation on Binary

```
ll next_perm(ll v) {
  ll t = v | (v - 1);
  return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1))
      ;
}
```

2.4 SOS DP

```
// 0 is 0, 1 can be 1 or 0
for (int i = 0; i < n; ++i)
  for (int j = 0; j < (1 << n); ++j)
    if (j & (1 << i))
      a[j] += a[j ^ (1 << i)];
```

3 Theorem and Formula

- Pick's theorem $A = i + \frac{b}{2} - 1$
- Laplacian matrix $L = D - A$
- Derangement $D_n = (n-1)(D_{n-1} + D_{n-2})$
- Möbius function $\sum_{i|n} \mu(i) = [n = 1]$
- Euler's totient function $\sum_{i|n} \phi(i) = n$
- Inversion formula

  $f(n) = \sum_{i=0}^{n} \binom{n}{i} g(i),\ g(n) = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} f(i)$

  $f(n) = \sum_{d|n} g(d),\ g(n) = \sum_{d|n} \mu(\frac{n}{d}) f(d)$
- Sum of powers

  $\sum_{k=1}^{n} k^m = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} B_k^+ \ n^{m+1-k}$

  $\sum_{j=0}^{m} \binom{m+1}{j} B_j^- = 0$

  note : $B_1^+ = -B_1^-\ B_i^+ = B_i^-$
- Cipolla's algorithm

  $\left(\frac{u}{p}\right) = u^{\frac{p-1}{2}}$

  1. $\left(\frac{a^2-n}{p}\right) = -1$
  2. $x = (a + \sqrt{a^2 - n})^{\frac{p+1}{2}}$
- High order residue

  $[d^{\frac{p-1}{(n,p-1)}} \equiv 1]$ (p is odd prime and $p \not| d$)
- Packing and Covering

  |Maximum Independent Set| + |Minimum Vertex Cover| = |V|
- Kőnig's theorem

  |Maximum matching|(easy) = |Minimum vertex cover|
- Dilworth's theorem

  width = |smallest chain decomposition| (vertex split and matching) = |largest antichain| = |maximim clique in Complement| (easy)
- Mirsky's theorem

  height = |longest chain|(easy DP) = |smallest antichain decomposition| = |minimum anticlique partition| (subset DP)
- Triangle center
  - $G : (1, 1, 1)$
  - $O : (a^2(b^2 + c^2 - a^2), \cdots) = (\sin 2A, \sin 2B, \sin 2C)$
  - $I : (a, b, c) = (\sin A, \sin B, \sin C)$
  - $E : (-a, b, c) = (-\sin A, \sin B, \sin C)$
  - $H : (\frac{1}{b^2+c^2-a^2}, \cdots) = (\tan A, \tan B, \tan C)$
- $\lfloor \frac{n}{i} \rfloor$ enumeration $T_0 = 1, T_i = \lfloor \frac{n}{\lfloor \frac{n}{T_{i-1}+1} \rfloor} \rfloor$

4 Data Structure

4.1 <ext/pb_ds>

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/rope>
using namespace __gnu_pbds;
using namespace __gnu_cxx;
using namespace std;

__gnu_pbds::priority_queue<int> pq, pq2;
__gnu_pbds::priority_queue<int>::point_iterator idx[10];
__gnu_pbds::priority_queue<int, less<int>, pairing_heap_tag>
    heap;
/*
pairing_heap_tag, thin_heap_tag, binomial_heap_tag
rc_binomial_heap_tag, binary_heap_tag
*/
idx[0] = pq.push(1);
pq.modify(idx[0], 2); // change the iterator's value to 2
pq.join(pq2);

typedef tree<int,null_type,less<int>,rb_tree_tag,
    tree_order_statistics_node_update> TREE;
TREE name;
*name.find_by_order(0);
name.order_of_key(1);
name.insert(2);
name.delete(3);
name.split(v, b); /// value < v of a split to b
name.join(another TREE);
```

4.2 Unordered Map Hash

```
struct KeyHasher {
  size_t operator()(const Key& k) const {
    return k.first + k.second * 100000;
  }
};
typedef unordered_map<Key, int, KeyHasher> map_t;
```

4.3 Rope

```
#include <ext/rope>
using namespace __gnu_cxx;
int main() {
  rope<int> v;  // can be cout directly if it's char
  rope<int> v1(v);
  rope<int> v2(arr, arr + 10); //int arr[100];
  v.find(3); // return the first positoin of 3
  v.push_back(4); v.pop_back();
  //append not for iterator
  v.insert(pos, s); // pos can be iterator, integer. s can be
      rope, int, array
  v.replace(pos, len, s); // (pos, len) can be (it1, it2). s is
       same as insert.
  v.erase(pos, len); // or v.erase(it1, it2)
  v2 = v.substr(pos, len); // same as erase
  v.copy(pos, len, arr); // int arr[100]; (pos, len) can be
      omitted
  v[0], v[1]
  auto it1 = v.mutable_begin(), it2 = v.mutable_end();
}
```

4.4 Disjoint Set

```
struct DJS{
  int p[N], rk[N];
  vector<pair<int*,int>> memo;
  vector<size_t> stk;
  void save() {
    stk.push_back(memo.size());
  }
  void undo() {
    while (memo.size() > stk.back()) {
      *memo.back().first = memo.back().second;
      memo.pop_back();
    }
    stk.pop_back();
  }
  void assign(int *x, int v) {
    memo.push_back({x, *x});
    *x = v;
  }
  //assign(&a, b); //a = b
} djs;
```

4.5 Persistent Treap

```
#include <bits/stdc++.h>

using namespace std;

struct Treap {
  static Treap mem[P];
  Treap * lc, * rc;
  char c;
  int sz;
  Treap() {}
  Treap(char _c): lc(NULL), rc(NULL), sz(1), c(_c) {}
}
Treap::mem[P], * ptr = Treap::mem;
int Sz(Treap * t) {
  return t ? t -> sz : 0;
}
void pull(Treap * t) {
  if (!t) return;
  t -> sz = Sz(t -> lc) + Sz(t -> rc) + 1;
}
Treap * merge(Treap * a, Treap * b) {
  if (!a || !b) return a ? a : b;
  Treap * ret;
  if (myRnd() % (Sz(a) + Sz(b)) < Sz(a)) {
    ret = new(ptr++) Treap(* a);
    ret -> rc = merge(a -> rc, b);
  } else {
    ret = new(ptr++) Treap(* b);
    ret -> lc = merge(a, b -> lc);
  }
  pull(ret);
  return ret;
}
void split(Treap * t, int k, Treap * & a, Treap * & b) {
  if (!t) a = b = NULL;
  else if (Sz(t -> lc) + 1 <= k) {
    a = new(ptr++) Treap(* t);
    split(t -> rc, k - Sz(t -> lc) - 1, a -> rc, b);
    pull(a);
  } else {
    b = new(ptr++) Treap(* t);
    split(t -> lc, k, a, b -> lc);
    pull(b);
  }
}
int d;
char buf[M];
Treap * ver[N];

ptr = Treap::mem;
v_cnt++;
ver[v_cnt] = ver[v_cnt - 1];
split(ver[v_cnt], p, tl, tr);
tl = merge(tl, new(ptr++) Treap(buf[j]));
```

4.6 Link Cut Tree

```
struct SplayNode {
  static SplayNode HOLE;
  SplayNode *ch[2], *par;
  bool rev;
  SplayNode(): par(&HOLE), rev(false) {
    ch[0] = ch[1] = &HOLE;
  }
  bool isRoot() {
    return (par->ch[0] != this && par->ch[1] != this);
  }
  void push() {
    if (rev) {
      if (ch[0]) ch[0]->rev ^= 1;
      if (ch[1]) ch[1]->rev ^= 1;
      swap(ch[0], ch[1]);
      rev ^= 1;
    }
  }
  void pushFromRoot() {
    if (!isRoot()) par->pushFromRoot();
    push();
  }
  void pull() {
    if (ch[0]) ch[0]->d = d + ch[0]->parLen;
    if (ch[1]) ch[1]->d = d + ch[1]->parLen;
  }
  void rotate() {
    SplayNode *p = par, *gp = p->par;
    bool dir = (p->ch[1] == this);
    par = gp;
    if (!p->isRoot()) gp->ch[gp->ch[1] == p] = this;
    p->ch[dir] = ch[dir ^ 1];
    p->ch[dir]->par = p;
    p->par = this;
    ch[dir ^ 1] = p;
    p->pull(), pull();
  }
  void splay() {
    pushFromRoot();
    while (!isRoot()) {
      if (!par->isRoot()) {
        SplayNode *gp = par->par;
        if ((gp->ch[0] == par) == (par->ch[0] == this))
          rotate();
        else par->rotate();
      }
      rotate();
    }
  }
} SplayNode::HOLE;
namespace LCT {
  SplayNode *access(SplayNode *x) {
    SplayNode *last = &SplayNode::HOLE;
    while (x != &SplayNode::HOLE) {
      x->splay();
      x->ch[1] = last;
      x->pull();
      last = x;
      x = x->par;
    }
    return last;
  }
  void makeRoot(SplayNode *x) {
    access(x);
    x->splay();
    x->rev ^= 1;
  }
  void link(SplayNode *x, SplayNode *y) {
    makeRoot(x);
    x->par = y;
```

```
  }
  void cut(SplayNode *x, SplayNode *y) {
    makeRoot(x);
    access(y);
    y->splay();
    y->ch[0] = &SplayNode::HOLE;
    x->par = &SplayNode::HOLE;
  }
  void cutParent(SplayNode *x) {
    access(x);
    x->splay();
    x->ch[0]->par = &SplayNode::HOLE;
    x->ch[0] = &SplayNode::HOLE;
  }
  SplayNode *findRoot(SplayNode *x) {
    x = access(x);
    while (x->ch[0] != &SplayNode::HOLE) x = x->ch[0];
    x->splay();
    return x;
  }
  SplayNode *query(SplayNode *x, SplayNode *y) {
    makeRoot(x);
    return access(y);
  }
  SplayNode *queryLca(SplayNode *x, SplayNode *y) {
    access(x);
    auto lca = access(y);
    x->splay();
    return lca->data + lca->ch[1]->sum +
                  (x == lca ? 0 : x->sum);
  }
  void modify(SplayNode *x, int data) {
    x->splay();
    x->data = data;
    x->pull();
  }
}
```

4.7 Li Chao Tree

```
struct line {
  ll a, b;
  line(): a(0), b(0) {}
  line(ll a, ll b): a(a), b(b) {}
  ll operator()(ll x) const { return a * x + b; }
};

struct lichao {
  line st[NN];
  int sz, lc[NN], rc[NN];
  int gnode() {
    st[sz] = line(0, -1e18); //min: st[sz] = line(0, 1e18);
    lc[sz] = -1, rc[sz] = -1;
    return sz++;
  }
  void init() {
    sz = 0; gnode();
  }
  void add(int l, int r, line tl, int o) {
    //[l, r)
    bool lcp = st[o](l) < tl(l); //min: change < to >
    bool mcp = st[o]((l + r) / 2) < tl((l + r) / 2); //min:
         change < to >
    if (mcp) swap(st[o], tl);
    if (r - l == 1) return;
    if (lcp != mcp) {
      if (lc[o] == -1) lc[o] = gnode();
      add(l, (l + r) / 2, tl, lc[o]);
    } else {
      if (rc[o] == -1) rc[o] = gnode();
      add((l + r) / 2, r, tl, rc[o]);
    }
  }
  ll query(int l, int r, int x, int o) {
    if (r - l == 1) return st[o](x);
    if (x < (l + r) / 2) {
      if (lc[o] == -1) return st[o](x);
      return max(st[o](x), query(l, (l + r) / 2, x, lc[o]));
    } else {
      if (rc[o] == -1) return st[o](x);
      return max(st[o](x), query((l + r) / 2, r, x, rc[o]));
    }
  }
} solver;
```

4.8 Dancing Link

```
const int MAX = 1050;
const int INF = 0x3f3f3f3f;
struct DLX{
  int n, sz, s[MAX];
  int row[MAX * 100], col[MAX * 100];
  int l[MAX * 100], r[MAX * 100], u[MAX * 100], d[MAX * 100];
  int ans;
  void init(int n) {
    this -> n = n;
    ans = INF;
    for (int i = 0; i <= n; ++ i) {
      u[i] = d[i] = i;
      l[i] = i - 1;
      r[i] = i + 1;
    }
    r[n] = 0, l[0] = n;
    sz = n + 1;
    memset(s, 0, sizeof s);
  }
  void AddRow(int rr, vector<int> sol) {
    int tmp = sz;
    for (auto to : sol) {
      l[sz] = sz - 1;
      r[sz] = sz + 1;
      d[sz] = to;
      u[sz] = u[to];
      d[u[to]] = sz, u[to] = sz;
      row[sz] = rr, col[sz] = to;
      s[to] ++, sz ++;
    }
    r[sz - 1] = tmp, l[tmp] = sz - 1;
  }
#define FOR(i, way, to) for (int i = way[to]; i != to; i = way[
    i])
  void remove(int c) {
    l[r[c]] = l[c];
    r[l[c]] = r[c];
    FOR(i, d, c) FOR(j, r, i) {
      u[d[j]] = u[j];
      d[u[j]] = d[j];
      --s[col[j]];
    }
  }
  int restore(int c) {
    FOR(i, u, c) FOR(j, l, i) {
      ++s[col[j]];
      u[d[j]] = j;
      d[u[j]] = j;
    }
    l[r[c]] = c;
    r[l[c]] = c;
  }
  void DFS(int floor) {
    if (r[0] == 0) {
      ans = min(ans, floor);
      return;
    }
    if (floor >= ans) return;
    int c = r[0];
    FOR(i, r, 0) if (s[i] < s[c]) c = i;
    remove(c);
    FOR(i, d, c) {
      FOR(j, r, i) remove(col[j]);
      DFS(floor + 1);
      FOR(j, l, i) restore(col[j]);
    }
    restore(c);
  }
} solver;
int n, m;
int32_t main() {
  IOS;
  while (cin >> n >> m) {
    solver.init(m);
    for (int i = 0; i < n; ++ i) {
      int nn, in;
      cin >> nn;
      vector<int> sol;
      for (int j = 0; j < nn; ++ j)
        cin >> in, sol.emplace_back(in);
      solver.AddRow(i, sol);
    }
    solver.DFS(0);
    if (solver.ans == INF) cout << "No" << endl;
    else cout << solver.ans << endl;
  }
  return 0;
```

```
}
```

4.9 Range Modify and Query BIT

```
int n, m, k;
int bit[4][MAX][MAX];
void update(int c[MAX][MAX], int a, int b, int val) {
  for (int i = a + 10; i < MAX; i += i & -i)
    for (int j = b + 10; j < MAX; j += j & -j)
      c[i][j] += val;
}
int update(int x, int y, int val) {
  update(bit[0], x, y, val);
  update(bit[1], x, y, -val * x);
  update(bit[2], x, y, -val * y);
  update(bit[3], x, y, val * x * y);
}
void update(int a, int b, int x, int y, int val) {
  update(a, b, val);
  update(a, y + 1, -val);
  update(x + 1, b, -val);
  update(x + 1, y + 1, val);
}
int query(int c[MAX][MAX], int a, int b) {
  int cnt = 0;
  for (int i = a + 10; i > 0; i -= i & -i)
    for (int j = b + 10; j > 0; j -= j & -j)
      cnt += c[i][j];
  return cnt;
}
int query(int x, int y) {
  int cnt = 0;
  cnt += query(bit[0], x, y) * (x + 1) * (y + 1);
  cnt += query(bit[1], x, y) * (y + 1);
  cnt += query(bit[2], x, y) * (x + 1);
  cnt += query(bit[3], x, y);
  return cnt;
}
int query(int a, int b, int x, int y) {
  int cnt = 0;
  cnt += query(a - 1, b - 1);
  cnt -= query(a - 1, y);
  cnt -= query(x, b - 1);
  cnt += query(x, y);
  return cnt;
}
/* usage:
void update(x1, y1, x2, y2, val);
int query(x1, y1, x2, y2);
*/
```

5 Flow

5.1 ISAP with bound

```
/*
Maximum density subgraph (\sum W_e + \sum W_v) / |V|
Binary search on answer:
For a fixed D, construct a Max flow model as follow:
Let S be Sum of all weight(or inf)
1. from source to each node with cap = S
2. For each (u,v,w) in E, (u->v,cap=w), (v->u,cap=w)
3. For each node v, from v to sink with cap = S + 2 * D - deg[v
    ] - 2 * (W of v)
where deg[v] = \sum weight of edge associated with v
If maxflow < S * |V|, D is an answer.
Requiring subgraph: all vertex can be reached from source with
edge whose cap > 0.
*/

//Be careful that it's zero base !!!!!!!!
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define SZ(x) ((int)(x).size())
#define eb emplace_back
const ll INF = 0x3f3f3f3f3f3f3f3f;

const ll N = 5e2 + 5;
struct isap{
  struct edge{
    int t, r;
    ll c;
    edge(int _t, int _r, ll _c) : t(_t), r(_r), c(_c) {}
  };
  int n, S, T;
  vector<edge> adj[N];
  int dis[N], gap[N], ok;
  isap(int _n, int _s, int _t) : n(_n), S(_s), T(_t) {
    for (int i = 0; i < n + 2; ++ i)  adj[i].clear();
  }
  void add(int u, int v, ll c) {
    adj[u].eb(v, adj[v].size(), c);
    adj[v].eb(u, adj[u].size() - 1, 0);
  }
  ll dfs(int now, ll f) {
    if (now == T) return f;
    int mi = n;
    for (edge &e : adj[now]) {
      if (e.c) {
        ll x;
        if (dis[now] == dis[e.t] + 1 && (x = dfs(e.t, min(f, e.
            c)))) {
          e.c -= x;
          adj[e.t][e.r].c += x;
          return x;
        }
        mi = min(mi, dis[e.t]);
      }
    }
    if (--gap[dis[now]] == 0) ok = 0;
    dis[now] = mi + 1;
    gap[ dis[now] ]++;
    return 0;
  }
  ll flow() {
    memset(dis, 0, n * 4);
    memset(gap, 0, n * 4);
    gap[0] = n;
    ok = 1;
    ll r = 0;
    while (dis[S] < n && ok)  r += dfs(S, INF);
    return r;
  }
  // below for bounded only
  ll D[N];
  void bounded_init() {
    memset(D, 0, n * 8);
  }
  void add2(int u, int v, ll b, ll c) {
    add(u, v, c - b);
    D[u] -= b;
    D[v] += b;
  }
  ll bounded_flow() {
    int SS = n, TT = n + 1;
    ll base = 0;
    for (int i = 0; i < n; ++ i) {
      if (D[i] > 0) base += D[i];
      if (D[i] > 0) add(SS, i, D[i]);
      if (D[i] < 0) add(i, TT, -D[i]);
    }
    add(T, S, INF);
    int tmps = S, tmpt = T;
    n += 2; S = SS, T = TT;
    ll f = flow();
    n -= 2; S = tmps; T = tmpt;
    return f == base ? flow() : -1LL;
  }
};
```

5.2 Min Cost Max Flow

```
struct Cost_Flow {
  struct Edge {
    int to, cap, rev, cost;
    Edge(int _to, int _cap, int _rev, int _cost): to(_to), cap(
        _cap), rev(_rev), cost(_cost) {}
  };
  vector<Edge> G[N];
  void add_edge(int from, int to, int cap, int cost) {
    G[from].push_back(Edge(to, cap, (int)G[to].size(), cost));
    G[to].push_back(Edge(from, 0, (int)G[from].size() - 1, -
        cost));
  }
  int n, s, t;
  void init(int _n, int _s, int _t) {
    n = _n, s = _s, t = _t;
    for (int i = 0; i <= n; ++i) {
      G[i].clear();
    }
  }
  bool in_que[N];
  int dis[N], par[N], par_id[N];
```

```
  pair<int, int> flow() {
    int flow = 0, cost = 0;
    while (true) {
      for (int i = 0; i <= n; ++i) {
        dis[i] = INF, in_que[i] = false;
      }
      queue<int> que; que.push(s);
      dis[s] = 0;
      while (!que.empty()) {
        int t = que.front(); que.pop();
        int ptr = 0;
        in_que[t] = false;
        for (Edge e: G[t]) {
          if (e.cap > 0) {
            if (dis[e.to] > dis[t] + e.cost) {
              dis[e.to] = dis[t] + e.cost;
              par[e.to] = t, par_id[e.to] = ptr;
              if (!in_que[e.to]) {
                que.push(e.to);
                in_que[e.to] = true;
              }
            }
          }
          ++ptr;
        }
      }
      if (dis[t] == INF) break;
      int mn_flow = INF;
      for (int i = t; i != s; i = par[i]) {
        mn_flow = min(mn_flow, G[par[i]][par_id[i]].cap);
      }
      flow += mn_flow;
      cost += mn_flow * dis[t];
      for (int i = t; i != s; i = par[i]) {
        G[par[i]][par_id[i]].cap -= mn_flow;
        G[i][G[par[i]][par_id[i]].rev].cap += mn_flow;
      }
    }
    return make_pair(flow, cost);
  }
} flow;
```

5.3 S-W Global Min Cut

```
struct SW {
  //find global min cut in O(V^3)
  //points are ZERO-BASE!!!
  static const int N = 506;
  int adj[N][N], wei[N], n;
  bool vis[N], del[N];
  void init(int _n) {
    n = _n;
    memset(adj, 0, sizeof(adj));
    memset(del, 0, sizeof(del));
  }
  void add_edge(int x, int y, int w) {
    adj[x][y] += w;
    adj[y][x] += w;
  }
  void search(int & s, int & t) {
    memset(wei, 0, sizeof(wei));
    memset(vis, 0, sizeof(vis));
    s = t = -1;
    while (true) {
      int mx = -1, mx_id = 0;
      for (int i = 0; i < n; ++i) {
        if (!del[i] && !vis[i] && mx < wei[i]) {
          mx_id = i;
          mx = wei[i];
        }
      }
      if (mx == -1) break;
      vis[mx_id] = true;
      s = t;
      t = mx_id;
      for (int i = 0; i < n; ++i)
        if (!vis[i] && !del[i])
          wei[i] += adj[mx_id][i];
    }
  }
  int solve() {
    int ret = INF;
    for (int i = 0; i < n - 1; ++i) {
      int x, y;
      search(x, y);
      ret = min(ret, wei[y]);
      del[y] = true;
      for (int j = 0; j < n; ++j) {
        adj[x][j] += adj[y][j];
        adj[j][x] += adj[y][j];
      }
    }
    return ret;
  }
} SW;
```

5.4 Gomory Hu Tree

```
def cut(G,s,t) :
  return minimum s-t cut in G

def gomory_hu(G):
  T = {}
  p = [1] * |V(G)|
  for s in [2,n] :
    t = p[s]
    w(C) = cut(G, s, t)
    add(s, t, w(C)) to T
    for i in [s + 1, n] :
      if p[i] == t and s-i path exists in G\C :
        p[i] = s
  return T;
```

6 Tree

6.1 Minimum Steiner Tree

```
// Minimum Steiner Tree
// O(V 3^T + V^2 2^T)
struct SteinerTree {
  const int V = 33;
  const int T = 8;
  const int INF = 0x3f3f3f3f;

  int n, dst[V][V], dp[1 << T][V], tdst[V];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++)
        dst[i][j] = INF;
      dst[i][i] = 0;
    }
  }
  void add_edge(int ui, int vi, int wi) {
    dst[ui][vi] = min(dst[ui][vi], wi);
    dst[vi][ui] = min(dst[vi][ui], wi);
  }
  void shortest_path() {
    for (int k = 0; k < n; k++)
      for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
          dst[i][j] = min(dst[i][j],
            dst[i][k] + dst[k][j]);
  }
  int solve(const vector<int> & ter) {
    int t = (int) ter.size();
    for (int i = 0; i < (1 << t); i++)
      for (int j = 0; j < n; j++)
        dp[i][j] = INF;
    for (int i = 0; i < n; i++)
      dp[0][i] = 0;
    for (int msk = 1; msk < (1 << t); msk++) {
      if (msk == (msk & (-msk))) {
        int who = __lg(msk);
        for (int i = 0; i < n; i++)
          dp[msk][i] = dst[ter[who]][i];
        continue;
      }
      for (int i = 0; i < n; i++)
        for (int submsk = (msk - 1) & msk; submsk;
          submsk = (submsk - 1) & msk)
          dp[msk][i] = min(dp[msk][i],
            dp[submsk][i] +
            dp[msk ^ submsk][i]);
      for (int i = 0; i < n; i++) {
        tdst[i] = INF;
        for (int j = 0; j < n; j++)
          tdst[i] = min(tdst[i],
            dp[msk][j] + dst[j][i]);
      }
      for (int i = 0; i < n; i++)
        dp[msk][i] = tdst[i];
    }
```

```
    int ans = INF;
    for (int i = 0; i < n; i++)
      ans = min(ans, dp[(1 << t) - 1][i]);
    return ans;
  }
}
solver;
```

6.2 Zhu Liu Algo

```
struct ZL {
  //1 base edge and vertex
  static const int N = 556, M = 2660, MM = M * 10, inf = 1e9;
  //MM = M * log N
  struct bian {
    int u, v, w, use, id;
  }
  b[M], a[MM];
  int n, m = 0, ans, pre[N], id[N], vis[N], root, In[N], h[N],
      len, way[M];
  void init(int _n, int _root) {
    for (int i = 0; i < MM; ++i) {
      a[i] = {0, 0, 0, 0, 0};
    }
    n = _n, m = 0;
    b[0].w = 1e9;
    root = _root;
  }
  void add(int u, int v, int w) {
    b[++m] = (bian) {u, v, w, 0, m};
    a[m] = b[m];
  }
  int work() {
    len = m;
    for (;;) {
      for (int i = 1; i <= n; i++) {
        pre[i] = 0;
        In[i] = inf;
        id[i] = 0;
        vis[i] = 0;
        h[i] = 0;
      }
      for (int i = 1; i <= m; i++)
        if (b[i].u != b[i].v && b[i].w < In[b[i].v]) {
          pre[b[i].v] = b[i].u;
          In[b[i].v] = b[i].w;
          h[b[i].v] = b[i].id;
        }
      for (int i = 1; i <= n; i++)
        if (pre[i] == 0 && i != root) return 0;
      int cnt = 0;
      In[root] = 0;
      for (int i = 1; i <= n; i++) {
        if (i != root) a[h[i]].use++;
        int now = i;
        ans += In[i];
        while (vis[now] == 0 && now != root) {
          vis[now] = i;
          now = pre[now];
        }
        if (now != root && vis[now] == i) {
          cnt++;
          int kk = now;
          while (1) {
            id[now] = cnt;
            now = pre[now];
            if (now == kk) break;
          }
        }
      }
      if (cnt == 0) return 1;
      for (int i = 1; i <= n; i++)
        if (id[i] == 0) id[i] = ++cnt;
      for (int i = 1; i <= m; i++) {
        int k1 = In[b[i].v], k2 = b[i].v;
        b[i].u = id[b[i].u];
        b[i].v = id[b[i].v];
        if (b[i].u != b[i].v) {
          b[i].w -= k1;
          a[++len].u = b[i].id;
          a[len].v = h[k2];
          b[i].id = len;
        }
      }
      n = cnt;
      root = id[root];
    }
    return 1;
  }
  int getway() {
    for (int i = 1; i <= m; i++) way[i] = 0;
    for (int i = len; i > m; i--) {
      a[a[i].u].use += a[i].use;
      a[a[i].v].use -= a[i].use;
    }
    for (int i = 1; i <= m; i++) way[i] = a[i].use;
    int ret = 0;
    for (int i = 1; i <= m; ++i) {
      if (way[i] == 1) {
        ret += a[i].w;
      }
    }
    return ret;
  }
}
zl;
//if zl.work() == 0, then it is not connected
//otherwise, use zl.getway() to check bian is selected or not
```

6.3 Centroid Decomposition

```
const int Mlg = __lg(MAX) + 2;

struct edge {
  int to, weight;
  edge(int _to, int _w): to(_to), weight(_w) {}
};

vector<edge> edg[MAX];

struct Cen {
  ll val;
  int p, sz, dep;
  Cen() {}
  Cen(int _p, int _d): val(0), p(_p), sz(0), dep(_d) {}
}
cen[MAX];
ll dis[Mlg][MAX];

bool visit[MAX];
vector<int> v;
int sz[MAX], mx[MAX];
void dfs_sz(int id) {
  visit[id] = 1;
  v.push_back(id);
  sz[id] = 1;
  mx[id] = 0;
  for (edge i: edg[id]) {
    if (!visit[i.to]) {
      dfs_sz(i.to);
      mx[id] = max(mx[id], sz[i.to]);
      sz[id] += sz[i.to];
    }
  }
}
void dfs_dis(int id, int cen_dep, ll weight) {
  dis[cen_dep][id] = weight;
  visit[id] = 1;
  for (edge i: edg[id])
    if (!visit[i.to])
      dfs_dis(i.to, cen_dep, weight + i.weight);
}
void build(int id, int cen_dep, int p) {
  dfs_sz(id);
  int nn = v.size();
  int ccen = -1;
  for (int i: v) {
    if (max(nn - sz[i], mx[i]) * 2 <= nn)
      ccen = i;
    visit[i] = 0;
  }
  dfs_dis(ccen, cen_dep, 0);
  for (int i: v) visit[i] = 0;
  v.clear();
  visit[ccen] = 1;
  cen[ccen] = Cen(p, cen_dep);
  for (edge i: edg[ccen])
    if (!visit[i.to])
      build(i.to, cen_dep + 1, ccen);
}

void add(int id, int d) {
  for (int p = id; p != -1; p = cen[p].p) {
    cen[p].val += dis[cen[p].dep][id] * d;
```

```
    cen[p].val -= dis[cen[p].dep - 1][id] * d;
    cen[p].sz += d;
  }
}

ll query(int id) {
  ll ret = 0;
  int pre_sz = 0;
  for (int p = id; p != -1; p = cen[p].p) {
    ret += cen[p].val;
    ret += (cen[p].sz - pre_sz) * dis[cen[p].dep][id];
    pre_sz = cen[p].sz;
  }
  return ret;
}
// edg[u].push_back(edge(v,w));
// edg[v].push_back(edge(u,w));
// memset(visit,0,sizeof(visit));
// build(1,1,-1);
// add(u, d)
// query(u)
```

6.4 Dynamic MST

```
/* Dynamic MST O( Q lg^2 Q )
 (qx[i], qy[i])->chg weight of edge No.qx[i] to qy[i]
 delete an edge: (i, \infty)
 add an edge: change from \infty to specific value
 */
const int SZ = M + 3 * MXQ;
int a[N], * tz;
int find(int xx) {
  int root = xx;
  while (a[root]) root = a[root];
  int next;
  while ((next = a[xx])) {
    a[xx] = root;
    xx = next;
  }
  return root;
}
bool cmp(int aa, int bb) {
  return tz[aa] < tz[bb];
}
int kx[N], ky[N], kt, vd[N], id[M], app[M];
bool extra[M];
void solve(int *qx, int *qy, int Q, int n, int *x, int *y, int
    *z, int m1, long long ans) {
  if (Q == 1) {
    for (int i = 1; i <= n; i++) a[i] = 0;
    z[qx[0]] = qy[0];
    tz = z;
    for (int i = 0; i < m1; i++) id[i] = i;
    sort(id, id + m1, cmp);
    int ri, rj;
    for (int i = 0; i < m1; i++) {
      ri = find(x[id[i]]);
      rj = find(y[id[i]]);
      if (ri != rj) {
        ans += z[id[i]];
        a[ri] = rj;
      }
    }
    printf("%lld\n", ans);
    return;
  }
  int ri, rj;
  //contract
  kt = 0;
  for (int i = 1; i <= n; i++) a[i] = 0;
  for (int i = 0; i < Q; i++) {
    ri = find(x[qx[i]]);
    rj = find(y[qx[i]]);
    if (ri != rj) a[ri] = rj;
  }
  int tm = 0;
  for (int i = 0; i < m1; i++) extra[i] = true;
  for (int i = 0; i < Q; i++) extra[qx[i]] = false;
  for (int i = 0; i < m1; i++)
    if (extra[i]) id[tm++] = i;
  tz = z;
  sort(id, id + tm, cmp);
  for (int i = 0; i < tm; i++) {
    ri = find(x[id[i]]);
    rj = find(y[id[i]]);
    if (ri != rj) {
      a[ri] = rj;
      ans += z[id[i]];
      kx[kt] = x[id[i]];
      ky[kt] = y[id[i]];
      kt++;
    }
  }
  for (int i = 1; i <= n; i++) a[i] = 0;
  for (int i = 0; i < kt; i++) a[find(kx[i])] = find(ky[i]);
  int n2 = 0;
  for (int i = 1; i <= n; i++)
    if (a[i] == 0)
      vd[i] = ++n2;
  for (int i = 1; i <= n; i++)
    if (a[i])
      vd[i] = vd[find(i)];
  int m2 = 0, * Nx = x + m1, * Ny = y + m1, * Nz = z + m1;
  for (int i = 0; i < m1; i++) app[i] = -1;
  for (int i = 0; i < Q; i++)
    if (app[qx[i]] == -1) {
      Nx[m2] = vd[x[qx[i]]];
      Ny[m2] = vd[y[qx[i]]];
      Nz[m2] = z[qx[i]];
      app[qx[i]] = m2;
      m2++;
    }
  for (int i = 0; i < Q; i++) {
    z[qx[i]] = qy[i];
    qx[i] = app[qx[i]];
  }
  for (int i = 1; i <= n2; i++) a[i] = 0;
  for (int i = 0; i < tm; i++) {
    ri = find(vd[x[id[i]]]);
    rj = find(vd[y[id[i]]]);
    if (ri != rj) {
      a[ri] = rj;
      Nx[m2] = vd[x[id[i]]];
      Ny[m2] = vd[y[id[i]]];
      Nz[m2] = z[id[i]];
      m2++;
    }
  }
  int mid = Q / 2;
  solve(qx, qy, mid, n2, Nx, Ny, Nz, m2, ans);
  solve(qx + mid, qy + mid, Q - mid, n2, Nx, Ny, Nz, m2, ans);
}
int x[SZ], y[SZ], z[SZ], qx[MXQ], qy[MXQ], n, m, Q;
void init() {
  scanf("%d%d", & n, & m);
  for (int i = 0; i < m; i++) scanf("%d%d%d", x + i, y + i, z +
      i);
  scanf("%d", & Q);
  for (int i = 0; i < Q; i++) {
    scanf("%d%d", qx + i, qy + i);
    qx[i]--;
  }
}
void work() {
  if (Q) solve(qx, qy, Q, n, x, y, z, m, 0);
}
int main() {
  init();
  work();
}
```

6.5 Heavy-Light Decomposition

```
int siz[MAX], son[MAX], dep[MAX], ffa[MAX];
int top[MAX], idx[MAX], idpo = 0;
int n, m;
int e[MAX][3];
vector<int> v[MAX];
struct node {
  int big, sml;
}
st[MAX * 4];
void init() {
  REP(i, 0, MAX) v[i].clear();
  MEM(siz, 0), MEM(son, 0), MEM(dep, 0), MEM(ffa, 0);
  MEM(top, 0), MEM(idx, 0), idpo = 0;
}
void DFS1(int now, int fa, int deep) {
  siz[now] = 1;
  dep[now] = deep;
  ffa[now] = fa;
  int big = 0;
  REP(i, 0, v[now].size()) {
    int to = v[now][i];
```

```
    if (to != fa) {
      DFS1(to, now, deep + 1);
      siz[now] += siz[to];
      if (siz[to] > big) big = siz[to], son[now] = to;
    }
  }
}
void DFS2(int now, int fa, int root) {
  top[now] = root;
  idx[now] = ++idpo;
  if (son[now] != 0) DFS2(son[now], now, root);
  REP(i, 0, v[now].size()) {
    int to = v[now][i];
    if (to != fa && to != son[now]) DFS2(to, now, to);
  }
}
void solveinit() {
  DFS1(1, 0, 0);
  DFS2(1, 0, 1);
  REP(i, 2, n + 1) {
    int a = e[i][0], b = e[i][1], c = e[i][2];
    if (dep[a] < dep[b]) swap(a, b);
    update(1, 1, n, idx[a], c);
  }
}
void query(int a, int b) {
  node ans;
  ans.big = -INF, ans.sml = INF;
  int t1 = top[a], t2 = top[b];
  while (t1 != t2) {
    if (dep[t1] < dep[t2]) swap(t1, t2), swap(a, b);
    ans = pull(ans, query(1, 1, n, idx[t1], idx[a]));
    a = ffa[t1], t1 = top[a];
  }
  if (dep[a] > dep[b]) swap(a, b);
  if (a != b) ans = pull(ans, query(1, 1, n, idx[son[a]], idx[b
      ]));
  return cout << ans.sml << " " << ans.big << endl, void();
}
init();
REP(i, 2, n + 1) {
  int a, b, c;
  cin >> a >> b >> c;
  e[i][0] = a, e[i][1] = b, e[i][2] = c;
  v[a].pb(b);
  v[b].pb(a);
}
solveinit();
query(a, b);
```

6.6 Block tree

```
#include <bits/stdc++.h>

using namespace std;

const int N = 30006;
const int K = 177;

int w[N], sum[N], mx[N];
int root[N], sz[N], fa[N], dep[N];
vector<int> G[N], T[N];

void dfs1(int now, int par, int depth) {
  fa[now] = par;
  dep[now] = depth;
  if (!root[now]) {
    root[now] = now;
    sz[now] = 1;
  }
  for (int i = 0; i < (int) G[now].size(); ++i) {
    int to = G[now][i];
    if (to == par) continue;
    if (sz[root[now]] + 1 < K) {
      T[now].push_back(to);
      root[to] = root[now];
      ++sz[root[now]];
    }
    dfs1(to, now, depth + 1);
  }
}

void dfs2(int now, int pre_sum, int pre_mx) {
  sum[now] = pre_sum, mx[now] = pre_mx;
  for (int i = 0; i < (int) T[now].size(); ++i) {
    int to = T[now][i];
    dfs2(to, pre_sum + w[to], max(pre_mx, w[to]));
  }
}

void change(int pos, int val) {
  w[pos] = val;
  dfs2(root[pos], w[root[pos]], w[root[pos]]);
}

void qmax(int u, int v) {
  // TODO
}

void qsum(int u, int v) {
  int ans = 0;
  while (u != v) {
    if (root[u] == root[v]) {
      if (dep[u] < dep[v]) swap(u, v);
      ans += w[u];
      u = fa[u];
    } else {
      if (dep[root[u]] < dep[root[v]]) swap(u, v);
      ans += sum[u];
      u = fa[root[u]];
    }
  }
  ans += w[u];
  printf("%d\n", ans);
}
```

7 Graph

7.1 Biconnected Component

```
int low[N], dfn[N];
bool vis[N];
int cnt[N], e[N], x[N], y[N]; // e[i] = x[i] ^ y[i]
int stamp, bcc_no = 0;

vector<int> G[N], bcc[N];
stack<int> sta;

void dfs(int now,int par) {
  vis[now] = true;
  dfn[now] = low[now] = (++stamp);
  for (int i : G[now]) {
    int to = (e[i] ^ now);
    if (to == par) continue;
    if (!vis[to]) {
      sta.push(i); dfs(to,now);
      low[now] = min(low[now], low[to]);
      if (low[to] >= dfn[now]) {
        ++bcc_no; int p; // p is edge index
        do {
          p = sta.top(); sta.pop();
          bcc[bcc_no].push_back(p);
        } while (p != i);
      }
    }
    else if (dfn[to] < dfn[now]) {
      sta.push(i);
      low[now] = min(low[now], dfn[to]);
    }
  }
}
```

7.2 General Graph Macthing

```
const int N = 100006,
  E = (2e5) * 2;
struct Graph {
  //1-index
  int to[E], bro[E], head[N], e;
  int lnk[N], vis[N], stp, n;
  int per[N];
  void init(int _n) {
    //remember to set every array to 0
    stp = 0;
    e = 1;
    n = _n;
    for (int i = 1; i <= n; i++)
      head[i] = lnk[i] = vis[i] = 0, per[i] = i;
    //random_shuffle(per+1, per+n+1);
  }
  void add_edge(int u, int v) {
    u = per[u], v = per[v];
    to[e] = v, bro[e] = head[u], head[u] = e++;
    to[e] = u, bro[e] = head[v], head[v] = e++;
```

```
  }
  bool dfs(int x) {
    vis[x] = stp;
    for (int i = head[x]; i; i = bro[i]) {
      int v = to[i];
      if (!lnk[v]) {
        lnk[x] = v, lnk[v] = x;
        return true;
      } else if (vis[lnk[v]] < stp) {
        int w = lnk[v];
        lnk[x] = v, lnk[v] = x, lnk[w] = 0;
        if (dfs(w)) {
          return true;
        }
        lnk[w] = v, lnk[v] = w, lnk[x] = 0;
      }
    }
    return false;
  }
  int solve() {
    int ans = 0;
    for (int i = 1; i <= n; i++)
      if (!lnk[i]) {
        stp++;
        ans += dfs(i);
      }
    return ans;
  }
}
graph;
```

7.3 KM

```
const int INF = 0x3f3f3f3f;
const int maxn = 610;

int n, w[maxn][maxn], lx[maxn], ly[maxn], slk[maxn];
int s[maxn], t[maxn], good[maxn];

int match(int now) {
  s[now] = 1;
  for (int to = 1; to <= n; to ++) {
    if (t[to]) continue;
    if (lx[now] + ly[to] == w[now][to]) {
      t[to] = 1;
      if (good[to] == 0 || match(good[to]))
        return good[to] = now, 1;
    }
    else slk[to] = min(slk[to], lx[now] + ly[to] - w[now][to]);
  }
  return 0;
}
void update() {
  int val = INF;
  for (int i = 1; i <= n; i ++)
    if (t[i] == 0) val = min(val, slk[i]);
  for (int i = 1; i <= n; i ++) {
    if (s[i]) lx[i] -= val;
    if (t[i]) ly[i] += val;
  }
}
void run_km() {
  for (int i = 1; i <= n; i ++) {
    lx[i] = w[i][1];
    for (int j = 1; j <= n; j ++)
      lx[i] = max(lx[i], w[i][j]);
  }
  for (int i = 1; i <= n; i ++)
    ly[i] = 0, good[i] = 0;
  for (int i = 1; i <= n; i ++) {
    for (int j = 1; j <= n; j ++) slk[j] = INF;
    while (1) {
      for (int j = 1; j <= n; j ++)
        s[j] = t[j] = 0;
      if (match(i)) break;
      else update();
    }
  }
}
/* how_to_use:
1. put edge in w[i][j]
2. run_km
3. match: (good[i], i)
*/
```

7.4 Maximum Weighted Matching(General Graph)

```
struct WeightGraph {
  static const int INF = INT_MAX;
  static const int N = 514;
  struct edge {
    int u, v, w;
    edge() {}
    edge(int ui, int vi, int wi): u(ui), v(vi), w(wi) {}
  };
  int n, n_x;
  edge g[N * 2][N * 2];
  int lab[N * 2];
  int match[N * 2], slack[N * 2], st[N * 2], pa[N * 2];
  int flo_from[N * 2][N + 1], S[N * 2], vis[N * 2];
  vector<int> flo[N * 2];
  queue<int> q;
  int e_delta(const edge & e) {
    return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
  }
  void update_slack(int u, int x) {
    if (!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x])
        ) slack[x] = u;
  }
  void set_slack(int x) {
    slack[x] = 0;
    for (int u = 1; u <= n; ++u)
      if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
        update_slack(u, x);
  }
  void q_push(int x) {
    if (x <= n) q.push(x);
    else
      for (size_t i = 0; i < flo[x].size(); i++)
        q_push(flo[x][i]);
  }
  void set_st(int x, int b) {
    st[x] = b;
    if (x > n)
      for (size_t i = 0; i < flo[x].size(); ++i)
        set_st(flo[x][i], b);
  }
  int get_pr(int b, int xr) {
    int pr = find(flo[b].begin(), flo[b].end(), xr) - flo[b].
        begin();
    if (pr % 2 == 1) {
      reverse(flo[b].begin() + 1, flo[b].end());
      return (int) flo[b].size() - pr;
    } else return pr;
  }
  void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n) return;
    edge e = g[u][v];
    int xr = flo_from[u][e.u], pr = get_pr(u, xr);
    for (int i = 0; i < pr; ++i) set_match(flo[u][i], flo[u][i
        ^ 1]);
    set_match(xr, v);
    rotate(flo[u].begin(), flo[u].begin() + pr, flo[u].end());
  }
  void augment(int u, int v) {
    for (;;) {
      int xnv = st[match[u]];
      set_match(u, v);
      if (!xnv) return;
      set_match(xnv, st[pa[xnv]]);
      u = st[pa[xnv]], v = xnv;
    }
  }
  int get_lca(int u, int v) {
    static int t = 0;
    for (++t; u || v; swap(u, v)) {
      if (u == 0) continue;
      if (vis[u] == t) return u;
      vis[u] = t;
      u = st[match[u]];
      if (u) u = st[pa[u]];
    }
    return 0;
  }
  void add_blossom(int u, int lca, int v) {
    int b = n + 1;
    while (b <= n_x && st[b]) ++b;
    if (b > n_x) ++n_x;
    lab[b] = 0, S[b] = 0;
```

```
    match[b] = match[lca];
    flo[b].clear();
    flo[b].push_back(lca);
    for (int x = u, y; x != lca; x = st[pa[y]])
      flo[b].push_back(x), flo[b].push_back(y = st[match[x]]),
          q_push(y);
    reverse(flo[b].begin() + 1, flo[b].end());
    for (int x = v, y; x != lca; x = st[pa[y]])
      flo[b].push_back(x), flo[b].push_back(y = st[match[x]]),
          q_push(y);
    set_st(b, b);
    for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
    for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
    for (size_t i = 0; i < flo[b].size(); ++i) {
      int xs = flo[b][i];
      for (int x = 1; x <= n_x; ++x)
        if (g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][
            x]))
          g[b][x] = g[xs][x], g[x][b] = g[x][xs];
      for (int x = 1; x <= n; ++x)
        if (flo_from[xs][x]) flo_from[b][x] = xs;
    }
    set_slack(b);
  }
  void expand_blossom(int b) {
    for (size_t i = 0; i < flo[b].size(); ++i)
      set_st(flo[b][i], flo[b][i]);
    int xr = flo_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
    for (int i = 0; i < pr; i += 2) {
      int xs = flo[b][i], xns = flo[b][i + 1];
      pa[xs] = g[xns][xs].u;
      S[xs] = 1, S[xns] = 0;
      slack[xs] = 0, set_slack(xns);
      q_push(xns);
    }
    S[xr] = 1, pa[xr] = pa[b];
    for (size_t i = pr + 1; i < flo[b].size(); ++i) {
      int xs = flo[b][i];
      S[xs] = -1, set_slack(xs);
    }
    st[b] = 0;
  }
  bool on_found_edge(const edge & e) {
    int u = st[e.u], v = st[e.v];
    if (S[v] == -1) {
      pa[v] = e.u, S[v] = 1;
      int nu = st[match[v]];
      slack[v] = slack[nu] = 0;
      S[nu] = 0, q_push(nu);
    } else if (S[v] == 0) {
      int lca = get_lca(u, v);
      if (!lca) return augment(u, v), augment(v, u), true;
      else add_blossom(u, lca, v);
    }
    return false;
  }
  bool matching() {
    memset(S + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) * n_x);
    q = queue<int> ();
    for (int x = 1; x <= n_x; ++x)
      if (st[x] == x && !match[x]) pa[x] = 0, S[x] = 0, q_push(
          x);
    if (q.empty()) return false;
    for (;;) {
      while (q.size()) {
        int u = q.front();
        q.pop();
        if (S[st[u]] == 1) continue;
        for (int v = 1; v <= n; ++v)
          if (g[u][v].w > 0 && st[u] != st[v]) {
            if (e_delta(g[u][v]) == 0) {
              if (on_found_edge(g[u][v])) return true;
            } else update_slack(u, st[v]);
          }
      }
      int d = INF;
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b && S[b] == 1) d = min(d, lab[b] / 2);
      for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && slack[x]) {
          if (S[x] == -1) d = min(d, e_delta(g[slack[x]][x]));
          else if (S[x] == 0) d = min(d, e_delta(g[slack[x]][x
              ]) / 2);
        }
      for (int u = 1; u <= n; ++u) {
        if (S[st[u]] == 0) {
          if (lab[u] <= d) return 0;
          lab[u] -= d;
        } else if (S[st[u]] == 1) lab[u] += d;
      }
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b) {
          if (S[st[b]] == 0) lab[b] += d * 2;
          else if (S[st[b]] == 1) lab[b] -= d * 2;
        }
      q = queue<int> ();
      for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && slack[x] && st[slack[x]] != x &&
            e_delta(g[slack[x]][x]) == 0)
          if (on_found_edge(g[slack[x]][x])) return true;
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b && S[b] == 1 && lab[b] == 0)
          expand_blossom(b);
    }
    return false;
  }
  pair < long long, int > solve() {
    memset(match + 1, 0, sizeof(int) * n);
    n_x = n;
    int n_matches = 0;
    long long tot_weight = 0;
    for (int u = 0; u <= n; ++u) st[u] = u, flo[u].clear();
    int w_max = 0;
    for (int u = 1; u <= n; ++u)
      for (int v = 1; v <= n; ++v) {
        flo_from[u][v] = (u == v ? u : 0);
        w_max = max(w_max, g[u][v].w);
      }
    for (int u = 1; u <= n; ++u) lab[u] = w_max;
    while (matching()) ++n_matches;
    for (int u = 1; u <= n; ++u)
      if (match[u] && match[u] < u)
        tot_weight += g[u][match[u]].w;
    return make_pair(tot_weight, n_matches);
  }
  void add_edge(int ui, int vi, int wi) {
    g[ui][vi].w = g[vi][ui].w = wi;
  }
  void init(int _n) {
    n = _n;
    for (int u = 1; u <= n; ++u)
      for (int v = 1; v <= n; ++v)
        g[u][v] = edge(u, v, 0);
  }
}
graph;
```

7.5 Minimum Mean Cycle

```
/* minimum mean cycle O(VE) */
struct MMC {
  struct Edge {
    int v, u;
    double c;
  };
  int n, m, prv[V][V], prve[V][V], vst[V];
  Edge e[E];
  vector<int> edgeID, cycle, rho;
  double d[V][V];
  void init(int _n) {
    n = _n, m = 0;
  }
  // WARNING: TYPE matters
  void addEdge(int vi, int ui, double ci) {
    e[m++] = {vi, ui, ci};
  }
  void bellman_ford() {
    for (int i = 0; i < n; i++) d[0][i] = 0;
    for (int i = 0; i < n; i++) {
      fill(d[i + 1], d[i + 1] + n, inf);
      for (int j = 0; j < m; j++) {
        int v = e[j].v, u = e[j].u;
        if (d[i][v] < inf && d[i + 1][u] > d[i][v] + e[j].c) {
          d[i + 1][u] = d[i][v] + e[j].c;
          prv[i + 1][u] = v;
          prve[i + 1][u] = j;
        }
      }
    }
  }
  double solve() {
    // returns inf if no cycle, mmc otherwise
```

```
    double mmc = inf;
    int st = -1;
    bellman_ford();
    for (int i = 0; i < n; i++) {
      double avg = -inf;
      for (int k = 0; k < n; k++) {
        if (d[n][i] < inf - eps) avg = max(avg, (d[n][i] - d[k
            ][i]) / (n - k));
        else avg = max(avg, inf);
      }
      if (avg < mmc) tie(mmc, st) = tie(avg, i);
    }
    FZ(vst);
    edgeID.clear();
    cycle.clear();
    rho.clear();
    for (int i = n; !vst[st]; st = prv[i--][st]) {
      vst[st]++;
      edgeID.PB(prve[i][st]);
      rho.PB(st);
    }
    while (vst[st] != 2) {
      int v = rho.back();
      rho.pop_back();
      cycle.PB(v);
      vst[v]++;
    }
    reverse(ALL(edgeID));
    edgeID.resize(SZ(cycle));
    return mmc;
  }
}
mmc;
```

7.6 Maximum Clique

```
struct BKB {
  static const int MAX_N = 50;
  typedef bitset < MAX_N > bst;
  bst N[MAX_N];
  int n;
  ll wei[MAX_N], ans, cc;
  BKB(int _n = 0): n(_n), ans(0), cc(0) {
    for (int i = 0; i < _n; ++i)
      N[i].reset();
  }
  void add_edge(int a, int b) {
    N[a][b] = N[b][a] = 1;
  }
  void set_wei(int a, ll w) {
    wei[a] = w;
  }
  ll CNT(bst P) {
    //if vertices have no weight: return P.count();
    ll rt = 0;
    for (int i = P._Find_first(); i < n; i = P._Find_next(i))
      rt += wei[i];
    return rt;
  }
  void pro(bst P, ll cnt = 0) {
    if (!P.any()) {
      if (cnt == ans)
        ++cc;
      else if (cnt > ans) {
        ans = cnt;
        cc = 1;
      }
      return;
    }
    // "<" can be change to "<=" if we don't need to count
    if (CNT(P) + cnt < ans)
      return;
    int u = P._Find_first();
    bst now = P & ~N[u];
    for (int i = now._Find_first(); i < n; i = now._Find_next(i
        )) {
      pro(P & N[i], cnt + wei[i]);
      P[i] = 0;
    }
    return;
  }
  pll solve() {
    bst tmp;
    tmp.reset();
    for (int i = 0; i < n; ++i)
      tmp[i] = 1;
    pro(tmp);
    return pll(ans, cc);
  }
}
ss(0);
```

8 Math

8.1 Extended Euclidean

```
// ax + by = gcd(a, b)
ll exgcd(ll a, ll b, ll & x, ll & y) {
  if (a == 0) return x = 0, y = 1, b;
  ll g = exgcd(b % a, a, y, x);
  x -= b / a * y;
  return g;
}
```

8.2 Gaussian Elimination

```
const int GAUSS_MOD = 100000007LL;
struct GAUSS {
  int n;
  vector<vector<int>> v;
  int ppow(int a, int k) {
    if (k == 0) return 1;
    if (k % 2 == 0) return ppow(a * a % GAUSS_MOD, k >> 1);
    if (k % 2 == 1) return ppow(a * a % GAUSS_MOD, k >> 1) * a
        % GAUSS_MOD;
  }
  vector<int> solve() {
    vector<int> ans(n);
    REP(now, 0, n) {
      REP(i, now, n) if (v[now][now] == 0 && v[i][now] != 0)
        swap(v[i], v[now]); // det = -det;
      if (v[now][now] == 0) return ans;
      int inv = ppow(v[now][now], GAUSS_MOD - 2);
      REP(i, 0, n) if (i != now) {
        int tmp = v[i][now] * inv % GAUSS_MOD;
        REP(j, now, n + 1)(v[i][j] += GAUSS_MOD - tmp * v[now][
            j] % GAUSS_MOD) %= GAUSS_MOD;
      }
    }
    REP(i, 0, n) ans[i] = v[i][n + 1] * ppow(v[i][i], GAUSS_MOD
        - 2) % GAUSS_MOD;
    return ans;
  }
  // gs.v.clear(), gs.v.resize(n, vector<int>(n + 1, 0));
}
gs;
```

8.3 Linear Basis

```
const int MAX_M = 500; //maximum number of variable
typedef bitset<MAX_M + 1> bst;
struct linear_basis {
  int m;
  bst mat[MAX_M];
  linear_basis(int _m): m(_m) {
    for (int i = 0; i < _m; ++i) mat[i].reset();
  }
  // True means "No solution"
  int add_constraint(bst now) {
    for (int j = 0; j < m; ++j) {
      if (now[j]) {
        if (mat[j][j]) now ^= mat[j];
        else {
          mat[j] = now;
          for (int k = j + 1; k < m; ++k)
            if (mat[j][k])
              mat[j] ^= mat[k];
          for (int k = 0; k < j; ++k)
            if (mat[k][j])
              mat[k] ^= mat[j];
          return 0;
        }
      }
    }
    return now[m];
  }
  // get one possible solution
  bst get_ans() {
    bst rt;
    rt.reset();
    for (int i = 0; i < m; ++i)
      if (mat[i][i] && mat[i][m])
        rt[i] = 1;
    return rt;
```

```
  }
};
/* usage :
1. Init it with # of variables
2. Adding constraint with format x1,x2...,xm,C
3. get_ans return one possible solution
*/
```

8.4 Build Prime

```
// MAX, eb
void build_prime(int min_fc[], vector<int> & P) {
  for (int i = 2; i < MAX; ++i) {
    if (min_fc[i] == 0) min_fc[i] = i, P.eb(i);
    for (auto j: P) {
      if (i * j >= MAX) break;
      min_fc[i * j] = j;
      if (i % j == 0) break;
    }
  }
}
```

8.5 Miller Rabin

```
ll mul(ll a, ll b, ll mod) {
  //calculate a*b % mod
  ll r = 0;
  a %= mod;
  b %= mod;
  while (b) {
    if (b & 1) r = (a + r >= mod ? a + r - mod : a + r);
    a = (a + a >= mod ? a + a - mod : a + a);
    b >>= 1;
  }
  return r;
}
ll power(ll a, ll n, ll mod) {
  if (n == 0) return 1 ll;
  else if (n == 1) return a % mod;
  return mul(power(mul(a, a, mod), n / 2, mod), n % 2 ? a : 1,
      mod);
}
const bool PRIME = 1, COMPOSITE = 0;
bool miller_robin(ll n, ll a) {
  if (__gcd(a, n) == n) return PRIME;
  if (__gcd(a, n) != 1) return COMPOSITE;
  ll d = n - 1, r = 0, ret;
  while (d % 2 == 0) {
    r++;
    d /= 2;
  }
  ret = power(a, d, n);
  if (ret == 1 || ret == n - 1) return PRIME;
  while (r--) {
    ret = mul(ret, ret, n);
    if (ret == n - 1) return PRIME;
  }
  return COMPOSITE;
}
bool isPrime(ll n) {
  //for int: 2, 7, 61
  ll as[7] = {2, 325, 9375, 28178, 450775, 9780504,
      1795265022};
  for (int i = 0; 7 > i; i++) {
    if (miller_robin(n, as[i]) == COMPOSITE) return COMPOSITE;
  }
  return PRIME;
}
```

8.6 Pollard Rho

```
// isPrime (miller rabin)
map < ll, int > cnt;
void PollardRho(ll n) {
  if (n == 1) return;
  if (isPrime(n)) return ++cnt[n], void();
  if (n % 2 == 0) return PollardRho(n / 2), ++cnt[2], void();
  ll x = 2, y = 2, d = 1, p = 1;
  auto f = [ & ](auto x, auto n, int p) {
    return (mul(x, x, n) + p) % n;
  }
  while (true) {
    if (d != n && d != 1) {
      PollardRho(n / d);
      PollardRho(d);
      return;
    }
    if (d == n) ++p;
    x = f(x, n, p);
    y = f(f(y, n, p), n, p);
    d = __gcd(abs(x - y), n);
  }
}
```

8.7 Build Phi and Mu

```
void build_phi(int ax[], int n) {
  for (int i = 1; i <= n; ++i)
    ax[i] = i;
  for (int i = 1; i <= n; ++i)
    for (int j = i + i; j <= n; j += i)
      ax[j] -= ax[i];
}
void build_mu(int ax[], int n) {
  for (int i = 1; i <= n; ++i)
    ax[i] = 0;
  ax[1] = 1;
  for (int i = 1; i <= n; ++i)
    for (int j = i + i; j <= n; j += i)
      ax[j] -= ax[i];
}
```

8.8 Primitive Root

```
// build_phi, power, eb
// M has primitive root when M = 2, 4, p^n, 2p^n
ll Primitive_root(ll n) {
  if (n == 2) return 1;
  vector<ll> sol;
  ll val = phi[n];
  for (ll i = 2; i * i <= val; ++i) {
    if (val % i == 0) {
      sol.eb(i);
      while (val % i == 0) val /= i;
    }
  }
  if (val != 1) sol.eb(val);
  for (ll i = 2; i < n; ++i) {
    if (__gcd(i, n) != 1) continue;
    ll ok = 1;
    for (auto to: sol) {
      if (power(i, phi[n] / to, n) == 1) {
        ok = 0;
        break;
      }
    }
    if (ok)
      return i;
  }
  return -1;
}
```

8.9 Cipolla's Algorithm

```
struct Cipolla {
  ll p, n, a, w;
  Cipolla(ll _p, ll _n): p(_p), n(_n) {
    n %= p;
    a = -1;
  }
  ll power(ll a, ll x) {
    if (x == 0) return 1;
    return power(a * a % p, x >> 1) * (x & 1 ? a : 1) % p;
  }
  inline int lgd(ll x) {
    return power(x, (p - 1) / 2);
  }
  ll rnd() {
    return (((ll) rand() << 28) + rand());
  }
  pll mul(pll a, pll b) {
    return pll((a.F * b.F + a.S * b.S % p * w) % p,
      (a.F * b.S + a.S * b.F) % p);
  }
  pll power(pll ii, ll x) {
    if (x == 0) return pll(1, 0);
    return mul(power(mul(ii, ii), x >> 1), (x & 1 ? ii : pll(1,
        0)));
  }
  ll solve() {
    if (p == 2)
      return n & 1;
    if (lgd(n) == p - 1) return -1;
    if (n == 0) return 0;
    while (a = rnd() % p, lgd((a * a - n + p) % p) == 1);
```

```
    w = (a * a - n + p) % p;
    pll ii = power(pll(a, 1), (p + 1) / 2);
    assert(ii.S == 0);
    return ii.F;
  }
};
```

8.10 Discrete Log

```
// power
int DiscreteLog_with_s(int s, int x, int y, int m) {
  int kStep = max((int) sqrt(m), 10);
  unordered_map<int, int> p;
  int b = 1;
  for (int i = 0; i < kStep; ++i) {
    p[y] = i;
    y = 1 LL * y * x % m;
    b = 1 LL * b * x % m;
  }
  for (int i = 0; i < m + 10; i += kStep) {
    s = 1 LL * s * b % m;
    if (p.find(s) != p.end()) return i + kStep - p[s];
  }
  return -1;
}
int DiscreteLog(int x, int y, int m) {
  // x ^ ? === y % m
  if (m == 1) return 0;
  // y %= m;
  int s = 1;
  for (int i = 0; i < 70; ++i) {
    if (s == y) return i;
    s = 1 LL * s * x % m;
  }
  if (s == y) return 70;
  int p = 70 + DiscreteLog_with_s(s, x, y, m);
  if (power(x, p, m) != y) return -1;
  return p;
}
```

8.11 Integer Partition

```
void build_partition(int _dp[], int n, int mod) {
  _dp[0] = 1;
  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= n; ++j) {
      int tmp = j * (j * 3 - 1) / 2;
      if (tmp > i) break;
      else if (j % 2 == 1) _dp[i] = (_dp[i] + _dp[i - tmp]) %
           mod;
      else if (j % 2 == 0) _dp[i] = (_dp[i] - _dp[i - tmp] +
           mod) % mod;
    }
    for (int j = 1; j <= n; ++j) {
      int tmp = j * (j * 3 + 1) / 2;
      if (tmp > i) break;
      else if (j % 2 == 1) _dp[i] = (_dp[i] + _dp[i - tmp]) %
           mod;
      else if (j % 2 == 0) _dp[i] = (_dp[i] - _dp[i - tmp] +
           mod) % mod;
    }
  }
  return;
}
```

8.12 Meissel-Lehmer Algorithm

```
// count number of prime that is <= n
int64_t PrimeCount(int64_t n) {
  if (n <= 1) return 0;
  const int v = sqrt(n);
  vector<int> smalls(v + 1);
  for (int i = 2; i <= v; ++i) smalls[i] = (i + 1) / 2;
  int s = (v + 1) / 2;
  vector<int> roughs(s);
  for (int i = 0; i < s; ++i) roughs[i] = 2 * i + 1;
  vector<int64_t > larges(s);
  for (int i = 0; i < s; ++i) larges[i] = (n / (2 * i + 1) + 1)
       / 2;
  vector<bool> skip(v + 1);
  int pc = 0;
  for (int p = 3; p <= v; ++p) {
    if (smalls[p] > smalls[p - 1]) {
      int q = p * p;
      pc++;
      if (1 LL * q * q > n) break;
      skip[p] = true;
      for (int i = q; i <= v; i += 2 * p) skip[i] = true;
      int ns = 0;
      for (int k = 0; k < s; ++k) {
        int i = roughs[k];
        if (skip[i]) continue;
        int64_t d = 1 LL * i * p;
        larges[ns] = larges[k] - (d <= v ? larges[smalls[d] -
             pc] : smalls[n / d]) + pc;
        roughs[ns++] = i;
      }
      s = ns;
      for (int j = v / p; j >= p; --j) {
        int c = smalls[j] - pc;
        for (int i = j * p, e = min(i + p, v + 1); i < e; ++i)
             smalls[i] -= c;
      }
    }
  }
  for (int k = 1; k < s; ++k) {
    const int64_t m = n / roughs[k];
    int64_t s = larges[k] - (pc + k - 1);
    for (int l = 1; l < k; ++l) {
      int p = roughs[l];
      if (1 LL * p * p > m) break;
      s -= smalls[m / p] - (pc + l - 1);
    }
    larges[0] -= s;
  }
  return larges[0];
}
```

8.13 De Bruijn

```
// sz_lim, MAX, MAX_len
int res[MAX], aux[MAX_len];
void db(int t, int p, int len, int k, int & sz) {
  if (sz >= sz_lim) return;
  if (t > len) {
    if (len % p == 0) {
      for (int i = 1; i <= p && sz < sz_lim; ++i) res[sz++] =
           aux[i];
    }
  } else {
    aux[t] = aux[t - p];
    db(t + 1, p, len, k, sz);
    for (int i = aux[t - p] + 1; i < k; ++i) {
      aux[t] = i;
      db(t + 1, t, len, k, sz);
    }
  }
}
// return cyclic string such that every string of length len
     using k character appears as a substring.
int de_bruijn(int k, int len) {
  if (k == 1) {
    res[0] = 0;
    return 1;
  }
  for (int i = 0; i < k * len; i++) aux[i] = 0;
  int sz = 0;
  db(1, 1, len, k, sz);
  return sz; // k^n
}
```

8.14 Simplex Algorithm

```
/*
maximize Cx under
Ax <=b
x >= 0
b >= 0
n variables
m constraints
A is m by n
*/
const int MAX = 45;
int n, m;
double arr[MAX][MAX];
bool pro() {
  double mi = 0;
  int x = 1;
  for (int i = 1; i <= n + m; i++)
    if (arr[0][i] < mi) {
      mi = arr[0][i];
      x = i;
    }
  if (abs(mi) < eps) return 0; // sigma <= 0
  mi = INF; // theta
```

```
  int y = 0;
  for (int i = 1; i <= m; i++) {
    if (arr[i][x] > eps && arr[i][n + m + 1] / arr[i][x] < mi)
        {
      mi = arr[i][n + m + 1] / arr[i][x];
      y = i;
    }
  }
  assert(y);
  double weed = arr[y][x];
  for (int i = 1; i <= n + m + 1; ++i)
    arr[y][i] /= weed;
  // now arr[y][n + m + 1] == theta
  for (int i = 0; i <= m; i++) {
    if (i == y) continue;
    double f = arr[i][x];
    for (int j = 1; j <= m + n + 1; j++)
      arr[i][j] -= f * arr[y][j];
  }
  return 1;
}
int main() {
  cin >> n;
  cin >> m;
  memset(arr, 0, sizeof arr);
  // input C
  for (int i = 1; i <= n; i++) {
    cin >> arr[0][i];
    arr[0][i] = -arr[0][i];
  }
  for (int i = 1; i <= m; i++) {
    // input A
    for (int j = 1; j <= n; j++)
      cin >> arr[i][j];
    arr[i][n + i] = 1;
    // input b
    cin >> arr[i][n + m + 1];
  }
  while (pro());
  cout << arr[0][n + m + 1] << "\n";
  return 0;
}
```

8.15 Middle Speed Linear Recursion

```
const int MAX = 1e5;
const int INF = 0x3f3f3f3f;
const int mod = 1e4;
int n, k, x[MAX], c[MAX];
vector<int> mul(vector<int> a, vector<int> b) {
  vector<int> ans(n + n + 1);
  REP(i, 1, n + 1) REP(j, 1, n + 1)
  ans[i + j] = (ans[i + j] + (a[i] * b[j])) % mod;
  RREP(i, n + n, n + 1) {
    REP(j, 1, n + 1) ans[i - j] = (ans[i - j] + ans[i] * c[j])
        % mod;
    ans[i] = 0;
  }
  return ans;
}
vector<int> ppow(vector<int> a, int k) {
  if (k == 1) return a;
  if (k % 2 == 0) return ppow(mul(a, a), k >> 1);
  if (k % 2 == 1) return mul(ppow(mul(a, a), k >> 1), a);
}
int main() {
  IOS;
  while (cin >> n && n) {
    REP(i, 1, n + 1) cin >> x[i];
    REP(i, 1, n + 1) cin >> c[i];
    vector<int> v(n + n + 1);
    v[1] = 1;
    cin >> k, k++;
    v = ppow(v, k);
    int ans = 0;
    REP(i, 1, n + 1) ans = (ans + x[i] * v[i]) % mod;
    cout << ans << endl;
  }
  return 0;
}
```

8.16 Chinese Remainder Theorem

```
const int INF = 0x3f3f3f3f
void extgcd(ll a, ll b, ll & d, ll & x, ll & y) {
  if (b == 0) d = a, x = 1, y = 0;
  else extgcd(b, a % b, d, y, x), y -= (a / b) * x;
}
ll n;
vector<ll> v, m;
int main() {
  while (cin >> n) {
    v.clear(), m.clear();
    ll ans, mod, d, x, y;
    REP(i, 0, n) cin >> mod >> ans, m.pb(mod), v.pb(ans);
    mod = m[0], ans = v[0];
    REP(i, 1, n) {
      ll res = ((v[i] - ans) % m[i] + m[i]) % m[i];
      extgcd(mod, m[i], d, x, y);
      if (res % d != 0) {
        ans = -1;
        break;
      }

      res = (res / d * x % m[i] + m[i]) % m[i];
      ans = ans + res * mod;
      mod = mod * m[i] / d;
    }
    if (ans == -1) cout << ans << endl;
    else cout << ans % mod << endl;
  }
  return 0;
}
```

9 Convolution

9.1 FFT

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 2 * 262144;
typedef long double ld;
typedef complex<ld> cplx;
const ld PI = acos(-1);
const cplx I(0, 1);
cplx omega[MAXN + 1];
void pre_fft() {
  for (int i = 0; i <= MAXN; i++) {
    omega[i] = exp(i * 2 * PI / MAXN * I);
  }
}
void fft(int n, cplx a[], bool inv = false) {
  int basic = MAXN/n;
  int theta = basic;
  for (int m = n; m >= 2; m >>= 1) {
    int mh = m >> 1;
    for (int i=0; i<mh; i++) {
      cplx w = omega[inv ? MAXN - (i * theta % MAXN) : i *
          theta % MAXN];
      for (int j = i; j < n; j += m) {
        int k = j + mh;
        cplx x = a[j] - a[k];
        a[j] += a[k];
        a[k] = w * x;
      }
    }
    theta = (theta * 2) % MAXN;
  }
  int i = 0;
  for (int j = 1;j < n - 1; j++) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j<i) swap(a[i],a[j]);
  }
  if (inv) {
    for (int i=0; i<n; i++) a[i] /= n;
  }
}
cplx a[MAXN], b[MAXN], c[MAXN];
//how to use :
/*
pre_fft();
fft(n,a);
fft(n,b);
for (int i = 0; i < n; i++) {
  c[i] = a[i] * b[i];
}
fft(n,c,1);
*/
```

9.2 NTT

```
// Remember coefficient are mod P
/*
```

```
(mod, root)
(65537, 3)
(23068673, 3)
(998244353, 3)
(1107296257, 10)
(2013265921, 31)
(2885681153, 3)
*/
typedef long long ll;
const int maxn = 65536;

struct NTT {
  ll mod = 2013265921, root = 31;
  ll omega[maxn + 1];
  void prentt() {
    ll x = fpow(root, (mod - 1) / maxn);
    omega[0] = 1;
    for (int i = 1; i <= maxn; ++i) {
      omega[i] = omega[i - 1] * x % mod;
    }
  }
  void real_init(ll _mod, ll _root) {
    mod = _mod;
    root = _root;
    prentt();
  }
  ll fpow(ll a, ll n) {
    (n += mod - 1) %= mod - 1;
    ll r = 1;
    for (; n; n >>= 1) {
      if (n & 1)(r *= a) %= mod;
      (a *= a) %= mod;
    }
    return r;
  }
  void bitrev(vector<ll> & v, int n) {
    int z = __builtin_ctz(n) - 1;
    for (int i = 0; i < n; ++i) {
      int x = 0;
      for (int j = 0; j <= z; ++j) x ^= ((i >> j & 1) << (z - j
          ));
      if (x > i) swap(v[x], v[i]);
    }
  }
  void ntt(vector<ll> & v, int n) {
    bitrev(v, n);
    for (int s = 2; s <= n; s <<= 1) {
      int z = s >> 1;
      for (int i = 0; i < n; i += s) {
        for (int k = 0; k < z; ++k) {
          ll x = v[i + k + z] * omega[maxn / s * k] % mod;
          v[i + k + z] = (v[i + k] + mod - x) % mod;
          (v[i + k] += x) %= mod;
        }
      }
    }
  }
  void intt(vector<ll> & v, int n) {
    ntt(v, n);
    reverse(v.begin() + 1, v.end());
    ll inv = fpow(n, mod - 2);
    for (int i = 0; i < n; ++i) {
      (v[i] *= inv) %= mod;
    }
  }
  vector<ll> conv(vector<ll> a, vector<ll> b) {
    int sz = 1;
    while (sz < a.size() + b.size() - 1) sz <<= 1;
    vector<ll> c(sz);
    while (a.size() < sz) a.push_back(0);
    while (b.size() < sz) b.push_back(0);
    ntt(a, sz), ntt(b, sz);
    for (int i = 0; i < sz; ++i) c[i] = (a[i] * b[i]) % mod;
    intt(c, sz);
    while (c.size() && c.back() == 0) c.pop_back();
    return c;
  }
};
ll chinese(ll b1, ll m1, ll b2, ll m2) {
  ll a1 = bigpow(m2, m1 - 2, m1) * b1 % m1;
  ll a2 = bigpow(m1, m2 - 2, m2) * b2 % m2;
  ll ret = (a1 * m2 + a2 * m1) % (m1 * m2);
  assert(ret % m1 == b1 && ret % m2 == b2);
  return ret;
}
```

9.3 FWT

```
void FWT(ll a[],int n) {
  for (int d = 1; d < n; d <<= 1)  // d = half of block size
    for (int i = 0; i < n; i += d + d) // every block
      for (int j = i; j < i + d; j++) {   //processing
        ll x = a[j], y = a[j + d];
        a[j] = x + y;           //FWT XOR
        a[j + d] = x - y;       //FWT XOR
        a[j] = x + y;           //FWT AND
        a[j + d] = y + x;       //FWT OR

        a[j] = (x + y) / 2;     //IFWT XOR
        a[j + d] = (x - y) / 2; //IFWT XOR
        a[j] = x - y;           //IFWT AND
        a[j + d] = y - x;       //IFWT OR
      }
}
```

9.4 Subset Convolution

```
for (int i = 0; i <= n; ++ i) {
  // f[__builtin_popcount(s)][s] = s, otherwise = 0. So is g[i]
  FWT(f[i], n) // OR
  FWT(g[i], n) // OR
  for (int s = 0; s < (1 << n); ++ s)
    for (int j = 0; j <= i; ++ j)
      h[i][s] += f[j][s] * g[i - j][s]
  IFWT(h[i], n) // OR
  for (int s = 0; i < (1 << n); ++ s)
    h[__builtin_popcount(s)][s] // is the real answer
}
```

9.5 Ternary Xor

```
pii operator*(const pii &p1, const pii &p2) {
  return {subb(mull(p1.F, p2.F) - mull(p1.S, p2.S)),
      subb(addd(mull(p1.F, p2.S) + mull(p1.S, p2.F)) - mull(p1.
          S, p2.S))};
}
pii cal1(pii p) {
  return {subb(-p.S), subb(p.F - p.S)};
}
pii cal2(pii p) {
  return {subb(p.S - p.F), subb(-p.F)};
}
//C is the size of a
void DFT(vector<pii> &a) {
  for (int mid = 1; mid < C; mid *= 3) {
    for (int j = 0; j < C; j += mid * 3) {
      for (int k = 0; k < mid; ++k) {
        pii x = a[j + k], y = a[j + k + mid], z = a[j + k + (
            mid << 1)];
        a[j + k] = x + y + z;
        a[j + k + mid] = x + cal1(y) + cal2(z);
        a[j + k + (mid << 1)] = x + cal2(y) + cal1(z);
      }
    }
  }
}
const int invn = ppow(C, mod - 2);
void IDFT(vector<pii> &a) {
  for (int mid = 1; mid < C; mid *= 3) {
    for (int j = 0; j < C; j += mid * 3) {
      for (int k = 0; k < mid; ++k) {
        pii x = a[j + k], y = a[j + k + mid],
          z = a[j + k + (mid << 1)];
        a[j + k] = x + y + z;
        a[j + k + mid] = x + cal2(y) + cal1(z);
        a[j + k + (mid << 1)] = x + cal1(y) + cal2(z);
      }
    }
  }
  for (int i = 0; i < C; ++i) {
    a[i].F = mull(a[i].F, invn);
  }
}
void ff(vector<pii> &a, vector<pii> b) {
  DFT(a); DFT(b);
  for (int i = 0; i < C; ++i) {
    a[i] = a[i] * b[i];
  }
  IDFT(a);
}
```

10 String

10.1 KMP

```
const KMP_SIZE = ;
struct KMP {
  string s;
  int f[KMP_SIZE], pos;
  void solve() {
    f[0] = pos = -1;
    for (int i = 1; i < s.size(); ++i) {
      while (pos != -1 && s[pos + 1] != s[i]) pos = f[pos];
      if (s[pos + 1] == s[i]) pos++;
      f[i] = pos;
    }
  }
};
```

10.2 Z value

```
const int ZVALUE_SIZE = ;
struct Z_VALUE {
  string s;
  int l = 0, r = 0, z[ZVALUE_SIZE];
  void solve() {
    for (int i = 0; i < s.size(); ++i) {
      z[i] = max(min(z[i - l], r - i), 0 LL);
      while (i + z[i] < s.size() && s[z[i]] == s[i + z[i]]) {
        l = i, r = i + z[i];
        z[i]++;
      }
    }
  }
};
```

10.3 Longest Palindrome

```
const int PALINDROME_MAX = 2 * ;
struct Palindrome {
  string s, ss; // ss = input
  int z[PALINDROME_MAX];
  void solve() {
    s.resize(ss.size() + ss.size() + 1, '.');
    for (int i = 0; i < ss.size(); ++i)
      s[i + i + 1] = ss[i];
    int l = 0, r = 0;
    for (int i = 0; i < s.size(); ++i) {
      z[i] = max(min(z[l + l - i], r - i), 1);
      while (i - z[i] >= 0 && i + z[i] < s.size() && s[i - z[i
          ]] == s[i + z[i]]) {
        l = i, r = i + z[i];
        z[i]++;
      }
    }
  }
};
```

10.4 Aho–Corasick Algorithm

```
struct AC_Automata {
  static const int N = 2e4 + 6;
  static const int SIGMA = 26;
  int ch[N][SIGMA], val[N], sz;
  int last[N], fail[N];
  int que[N], qs, qe, cnt[N];
  void init() {
    sz = 1;
    memset(ch[0], 0, sizeof(ch[0]));
    qs = qe = 0;
    memset(cnt, 0, sizeof(cnt));
    memset(val, 0, sizeof(val));
    memset(last, 0, sizeof(last));
  }
  int idx(char c) {
    return c - 'a';
  }
  int insert(string s, int v) {
    int now = 0;
    int n = s.size();
    for (int i = 0; i < n; ++i) {
      int c = idx(s[i]);
      if (!ch[now][c]) {
        memset(ch[sz], 0, sizeof(ch[sz]));
        val[sz] = 0, ch[now][c] = sz++;
      }
      now = ch[now][c];
    }
    val[now] = v;
    return now;
  }
  void print(int j) {
    if (j) {
      //now we match string v[j]
      print(last[j]); //may match multiple strings
    }
  }
  void getFail() {
    qs = 0, qe = 0;
    fail[0] = 0;
    for (int c = 0; c < SIGMA; c++) {
      int now = ch[0][c];
      if (now) {
        fail[now] = 0;
        que[qe++] = now;
        last[now] = 0;
      }
    }
    while (qs != qe) {
      int t = que[qs++];
      for (int c = 0; c < SIGMA; c++) {
        int now = ch[t][c];
        if (!now) continue;
        que[qe++] = now;
        int v = fail[t];
        while (v && !ch[v][c]) v = fail[v];
        fail[now] = ch[v][c];
        last[now] = val[fail[now]] ? fail[now] : last[fail[now
            ]];
      }
    }
  }
  void AC_evolution() {
    for (qs = 0; qs != qe;) {
      int now = que[qs++];
      for (int i = 0; i < SIGMA; i++) {
        if (ch[now][i] == 0) ch[now][i] = ch[fail[now]][i];
      }
    }
  }
  void build() {
    getFail();
    AC_evolution();
  }
  void Find(string s) {
    int n = s.size(), now = 0;
    for (int i = 0; i < n; i++) {
      int c = idx(s[i]);
      while (now && !ch[now][c]) now = fail[now];
      now = ch[now][c];
      cnt[now]++;
    }
    for (int i = qe - 1; i >= 0; i--) {
      cnt[fail[que[i]]] += cnt[que[i]];
    }
  }
}
ac;

const int N = 156;
string s[N];
int ed[N];

ac.init();
ac.insert(s[i], i); // insert small strings
ac.build();
ac.Find(large_string);
ac.cnt[ac.insert(s[i], i)]; // number of small string
```

10.5 Suffix Array

```
const int SA_SIZE = ;
const int logn = 1 + ;
string s;
int sa[SA_SIZE], rk[SA_SIZE], lcp[SA_SIZE];
int tma[2][SA_SIZE], c[SA_SIZE], sp[SA_SIZE][logn];

int getsa() {
  -> update m = ? // how many char
    int * x = tma[0], * y = tma[1], n = s.size(), m = 200;
  for (int i = 0; i < m; ++i) c[i] = 0;
  for (int i = 0; i < n; ++i) c[x[i] = s[i]]++;
  for (int i = 1; i < m; ++i) c[i] += c[i - 1];
  for (int i = n - 1; i >= 0; --i) sa[--c[x[i]]] = i;
```

```
  for (int k = 1; k <= n; k <<= 1) {
    for (int i = 0; i < m; ++i) c[i] = 0;
    for (int i = 0; i < n; ++i) c[x[i]]++;
    for (int i = 1; i < m; ++i) c[i] += c[i - 1];
    int p = 0;
    for (int i = n - k; i < n; ++i) y[p++] = i;
    for (int i = 0; i < n; ++i)
      if (sa[i] >= k) y[p++] = sa[i] - k;
    for (int i = n - 1; i >= 0; --i) sa[--c[x[y[i]]]] = y[i];
    y[sa[0]] = p = 0;
    for (int i = 1; i < n; ++i) {
      if (x[sa[i]] == x[sa[i - 1]] && sa[i] + k < n && sa[i -
          1] + k < n &&
        x[sa[i] + k] == x[sa[i - 1] + k]);
      else p++;
      y[sa[i]] = p;
    }
    swap(x, y);
    if (p + 1 == n) break;
    m = p + 1;
  }
}
void getlcp() {
  int tmp = 0, n = s.size();
  for (int i = 0; i < n; ++i) rk[sa[i]] = i;
  for (int i = 0; i < n; ++i) {
    if (rk[i] == 0) lcp[0] = 0;
    else {
      if (tmp) tmp--;
      int po = sa[rk[i] - 1];
      while (tmp + po < n && tmp + i < n && s[tmp + i] == s[tmp
           + po]) tmp++;
      lcp[rk[i]] = tmp;
    }
  }
}
void getsp() {
  int n = s.size();
  for (int i = 0; i < n; ++i) sp[rk[i]][0] = s.size() - i;
  for (int i = 1; i < n; ++i) sp[i - 1][1] = lcp[i];
  for (int i = 2; i < logn; ++i) {
    for (int j = 0; j < n; ++j) {
      if (j + (1 << (i - 2)) >= s.size()) continue;
      sp[j][i] = min(sp[j][i - 1], sp[j + (1 << (i - 2))][i -
          1]);
    }
  }
}
int Query(int L, int R) {
  int tmp = (L == R) ? 0 : 32 - __builtin_clz(R - L);
  if (tmp == 0) return sp[L][0];
  else return min(sp[L][tmp], sp[R - (1 << (tmp - 1))][tmp]);
}
int Find(string ss) {
  int L = 0, R = s.size(), now;
  while (R - L > 1) {
    now = (L + R) / 2;
    if (s[sa[now]] == ss[0]) break;
    else if (s[sa[now]] > ss[0]) R = now;
    else if (s[sa[now]] < ss[0]) L = now;
  }
  if (s[sa[now]] != ss[0]) return 0;
  for (int i = 1; i < ss.size(); ++i) {
    int pre = now, ty = 0;
    if (sa[now] + i >= s.size()) L = now, ty = 0;
    else if (s[sa[now] + i] == ss[i]) continue;
    else if (s[sa[now] + i] > ss[i]) R = now, ty = 1;
    else if (s[sa[now] + i] < ss[i]) L = now, ty = 0;

    while (R - L > 1) {
      now = (L + R) / 2;
      if (sa[now] + i >= s.size()) {
        if (ty == 0) R = now;
        if (ty == 1) L = now;
      } else if (ty == 0 && Query(pre, now) < i) R = now;
      else if (ty == 1 && Query(now, pre) < i) L = now;
      else if (s[sa[now] + i] == ss[i]) break;
      else if (s[sa[now] + i] > ss[i]) R = now;
      else if (s[sa[now] + i] < ss[i]) L = now;
    }
    if (sa[now] + i >= s.size()) return 0;
    if (s[sa[now] + i] != ss[i]) return 0;
  }
  L = now, R = now;
  for (int i = 19; i >= 0; --i) {
    if (R + (1 << i) >= s.size()) continue;
    else if (Query(L, R + (1 << i)) >= ss.size()) R += (1 << i)
        ;
  }
  for (int i = 19; i >= 0; --i) {
    if (L - (1 << i) < 0) continue;
    else if (Query(L - (1 << i), R) >= ss.size()) L -= (1 << i)
        ;
  }
  return R - L + 1;
}
/*
how to use :
1. cin >> s;
2. getsa(), getlcp(), getsp();
3. string ss;
4. cin >> ss;
5. cout << Find(ss) << endl;
*/
```

10.6 Palindromic Tree

```
//MAXN
const int N = 26;
struct Palindromic_Tree {
  int next[MAXN][N]; //trie tree edge
  int len[MAXN]; //tree edge depth*2 (-1)
  int fail[MAXN]; //fail link
  int num[MAXN]; //fail link depth
  int cnt[MAXN]; //# of this Palindrom
  int S[MAXN]; //string
  int p; //# of different Palindrom + 2
  int n; //string len
  int last;
  int newnode(int l) {
    memset(next[p], 0, N * 4);
    cnt[p] = num[p] = 0;
    len[p] = l;
    return p++;
  }
  void init() {
    p = n = 0;
    last = 1;
    newnode(0);
    newnode(-1);
    S[n] = -1;
    fail[0] = 1;
  }
  int get_fail(int x) {
    while (S[n - len[x] - 1] != S[n]) x = fail[x];
    return x;
  }
  void add(int c) {
    c -= 'a';
    S[++n] = c;
    int cur = get_fail(last);
    if (!next[cur][c]) {
      int now = newnode(len[cur] + 2);
      fail[now] = next[get_fail(fail[cur])][c];
      next[cur][c] = now;
      num[now] = num[fail[now]] + 1;
    }
    last = next[cur][c];
    cnt[last]++;
  }
  void count() {
    for (int i = p - 1; i >= 0; --i) cnt[fail[i]] += cnt[i];
  }
};
```

10.7 Lexicographically Smallest Rotation

```
string s;
const int N = 4000006;
int f[N];
void solve() {
  s = s + s;
  int n = (int) s.size();
  for (int i = 0; i < n; ++i) f[i] = -1;
  int k = 0;
  for (int j = 1; j < n; ++j) {
    char sj = s[j];
    int i = f[j - k - 1];
    while (i != -1 && sj != s[k + i + 1]) {
      if (sj < s[k + i + 1])
        k = j - i - 1;
      i = f[i];
    }
```

```
    if (sj != s[k + i + 1]) {
      if (sj < s[k]) k = j;
      f[j - k] = -1;
    } else f[j - k] = i + 1;
  }
  n >>= 1;
  if (k >= n) k -= n;
  for (int i = k; i < k + n; ++i)
    cout << s[i];
  cout << endl;
}
```

11 Geometry

11.1 Circle

```
//Note that this code will crash if circle A and B are the same
typedef pair<double, double> pdd;
pdd rtcw(pdd p) { return pdd(p.Y, -p.X); }
vector<pdd> circlesintersect(pdd A, pdd B, double r1, double r2
    ) {
  vector<pdd> ret;
  double d = dis(A, B);
  if (d > r1 + r2 || d + min(r1, r2) < max(r1, r2))
    return ret;
  double x = (d * d + r1 * r1 - r2 * r2) / (2 * d);
  double y = sqrt(r1 * r1 - x * x);
  pdd v = (B - A) / d;
  ret.eb(A + v * x + rtcw(v) * y);
  if (y > 0)
    ret.eb(A + v * x - rtcw(v) * y);
  return ret;
}
```

11.2 Half Plane Intersection

```
Pt interPnt(Line l1, Line l2, bool & res) {
  Pt p1, p2, q1, q2;
  tie(p1, p2) = l1;
  tie(q1, q2) = l2;
  double f1 = (p2 - p1) ^ (q1 - p1);
  double f2 = (p2 - p1) ^ (p1 - q2);
  double f = (f1 + f2);
  if (fabs(f) < eps) {
    res = 0;
    return {0, 0};
  }
  res = true;
  return q1 * (f2 / f) + q2 * (f1 / f);
}
bool isin(Line l0, Line l1, Line l2) {
  // Check inter(l1, l2) in l0
  bool res;
  Pt p = interPnt(l1, l2, res);
  return ((l0.SE - l0.FI) ^ (p - l0.FI)) > eps;
}
/* If no solution, check: 1. ret.size() < 3
 * Or more precisely, 2. interPnt(ret[0], ret[1])
 * in all the lines. (use (l.S - l.F) ^ (p - l.F) > 0
 */
/* --^-- Line.FI --^-- Line.SE --^-- */
vector<Line> halfPlaneInter(vector<Line> lines) {
  int sz = lines.size();
  vector<double> ata(sz), ord(sz);
  for (int i = 0; i < sz; i++) {
    ord[i] = i;
    Pt d = lines[i].SE - lines[i].FI;
    ata[i] = atan2(d.Y, d.X);
  }
  sort(ord.begin(), ord.end(), [ & ](int i, int j) {
    if (fabs(ata[i] - ata[j]) < eps)
      return ((lines[i].SE - lines[i].FI) ^
        (lines[j].SE - lines[i].FI)) < 0;
    return ata[i] < ata[j];
  });
  vector<Line> fin;
  for (int i = 0; i < sz; i++)
    if (!i or fabs(ata[ord[i]] - ata[ord[i - 1]]) > eps)
      fin.PB(lines[ord[i]]);
  deque<Line> dq;
  for (int i = 0; i < (int)(fin.size()); i++) {
    while ((int)(dq.size()) >= 2 and not isin(fin[i], dq[(int)(
        dq.size()) - 2],
      dq[(int)(dq.size()) - 1]))
      dq.pop_back();
    while ((int)(dq.size()) >= 2 and not isin(fin[i], dq[0], dq
        [1]))
      dq.pop_front();
    dq.push_back(fin[i]);
  }
  while ((int)(dq.size()) >= 3 and not isin(dq[0], dq[(int)(dq.
      size()) - 2],
    dq[(int)(dq.size()) - 1]))
    dq.pop_back();
  while ((int)(dq.size()) >= 3 and not isin(dq[(int)(dq.size())
      - 1], dq[0], dq[1]))
    dq.pop_front();
  vector<Line> res(dq.begin(), dq.end());
  return res;
}
```

11.3 Convex Hull 3D

```
#define SIZE(X) (int(X.size()))
#define PI 3.14159265358979323846264338327950288
struct Pt{
  Pt cross(const Pt &p) const
  { return Pt(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y
      * p.x); }
} info[N];
int mark[N][N],n, cnt;;
double mix(const Pt &a, const Pt &b, const Pt &c)
{ return a * (b ^ c); }
double area(int a, int b, int c)
{ return norm((info[b] - info[a]) ^ (info[c] - info[a])); }
double volume(int a, int b, int c, int d)
{ return mix(info[b] - info[a], info[c] - info[a], info[d] -
    info[a]); }
struct Face{
  int a, b, c; Face() {}
  Face(int a, int b, int c): a(a), b(b), c(c) {}
  int &operator [](int k)
  { if (k == 0) return a; if (k == 1) return b; return c; }
};
vector<Face> face;
void insert(int a, int b, int c)
{ face.push_back(Face(a, b, c)); }
void add(int v) {
  vector<Face> tmp; int a, b, c; cnt++;
  for (int i = 0; i < SIZE(face); i++) {
    a = face[i][0]; b = face[i][1]; c = face[i][2];
    if (Sign(volume(v, a, b, c)) < 0)
      mark[a][b] = mark[b][a] = mark[b][c] = mark[c][b] = mark[
          c][a] = mark[a][c] = cnt;
    else tmp.push_back(face[i]);
  } face = tmp;
  for (int i = 0; i < SIZE(tmp); i++) {
    a = face[i][0]; b = face[i][1]; c = face[i][2];
    if (mark[a][b] == cnt) insert(b, a, v);
    if (mark[b][c] == cnt) insert(c, b, v);
    if (mark[c][a] == cnt) insert(a, c, v);
  }}
int Find() {
  for (int i = 2; i < n; i++) {
    Pt ndir = (info[0] - info[i]) ^ (info[1] - info[i]);
    if (ndir == Pt()) continue; swap(info[i], info[2]);
    for (int j = i + 1; j < n; j++) if (Sign(volume(0, 1, 2, j)
        ) != 0) {
      swap(info[j], info[3]); insert(0, 1, 2); insert(0, 2, 1);
          return 1;
    } } return 0; }
int main() {
  for (; scanf("%d", &n) == 1; ) {
    for (int i = 0; i < n; i++) info[i].Input();
    sort(info, info + n); n = unique(info, info + n) - info;
    face.clear(); random_shuffle(info, info + n);
    if (Find()) { memset(mark, 0, sizeof(mark)); cnt = 0;
      for (int i = 3; i < n; i++) add(i); vector<Pt> Ndir;
      for (int i = 0; i < SIZE(face); ++i) {
        Pt p = (info[face[i][0]] - info[face[i][1]]) ^
          (info[face[i][2]] - info[face[i][1]]);
        p = p / norm(p); Ndir.push_back(p);
      } sort(Ndir.begin(), Ndir.end());
      int ans = unique(Ndir.begin(), Ndir.end()) - Ndir.begin()
          ;
      printf("%d\n", ans);
    } else printf("1\n");
  } }
double calcDist(const Pt &p, int a, int b, int c)
{ return fabs(mix(info[a] - p, info[b] - p, info[c] - p) / area
    (a, b, c)); }
//compute the minimal distance of center of any faces
double findDist() { //compute center of mass
  double totalWeight = 0; Pt center(.0, .0, .0);
```

```
  Pt first = info[face[0][0]];
  for (int i = 0; i < SIZE(face); ++i) {
    Pt p = (info[face[i][0]]+info[face[i][1]]+info[face[i][2]]+
        first)*.25;
    double weight = mix(info[face[i][0]] - first, info[face[i
        ][1]]
        - first, info[face[i][2]] - first);
    totalWeight += weight; center = center + p * weight;
  } center = center / totalWeight;
  double res = 1e100; //compute distance
  for (int i = 0; i < SIZE(face); ++i)
    res = min(res, calcDist(center, face[i][0], face[i][1],
        face[i][2]));
  return res; }
```

11.4 Dynamic convexhull

```
/* Given a convexhull, answer querys in O(\lg N)
   CH should not contain identical points, the area should
   be > 0, min pair(x, y) should be listed first */
double det(const Pt& p1, const Pt& p2)
{ return p1.X * p2.Y - p1.Y * p2.X; }
struct Conv{
  int n;
  vector<Pt> a;
  vector<Pt> upper, lower;
  Conv(vector<Pt> _a) : a(_a) {
    n = a.size();
    int ptr = 0;
    for (int i=1; i<n; ++i) if (a[ptr] < a[i]) ptr = i;
    for (int i=0; i<=ptr; ++i) lower.push_back(a[i]);
    for (int i=ptr; i<n; ++i) upper.push_back(a[i]);
    upper.push_back(a[0]);
  }
  int sign(LL x) { // fixed when changed to double
    return x < 0 ? -1 : x > 0; }
  pair<LL,int> get_tang(vector<Pt> &conv, Pt vec) {
    int l = 0, r = (int)conv.size() - 2;
    for (; l + 1 < r; ) {
      int mid = (l + r) / 2;
      if (sign(det(conv[mid+1]-conv[mid],vec))>0)r=mid;
      else l = mid;
    }
    return max(make_pair(det(vec, conv[r]), r),
        make_pair(det(vec, conv[0]), 0));
  }
  void upd_tang(const Pt &p, int id, int &i0, int &i1) {
    if (det(a[i0] - p, a[id] - p) > 0) i0 = id;
    if (det(a[i1] - p, a[id] - p) < 0) i1 = id;
  }
  void bi_search(int l, int r, Pt p, int &i0, int &i1) {
    if (l == r) return;
    upd_tang(p, l % n, i0, i1);
    int sl=sign(det(a[l % n] - p, a[(l + 1) % n] - p));
    for (; l + 1 < r; ) {
      int mid = (l + r) / 2;
      int smid=sign(det(a[mid%n]-p, a[(mid+1)%n]-p));
      if (smid == sl) l = mid;
      else r = mid;
    }
    upd_tang(p, r % n, i0, i1);
  }
  int bi_search(Pt u, Pt v, int l, int r) {
    int sl = sign(det(v - u, a[l % n] - u));
    for (; l + 1 < r; ) {
      int mid = (l + r) / 2;
      int smid = sign(det(v - u, a[mid % n] - u));
      if (smid == sl) l = mid;
      else r = mid;
    }
    return l % n;
  }
  // 1. whether a given point is inside the CH
  bool contain(Pt p) {
    if (p.X < lower[0].X || p.X > lower.back().X) return 0;
    int id = lower_bound(lower.begin(), lower.end(), Pt(p.X, -
        INF)) - lower.begin();
    if (lower[id].X == p.X) {
      if (lower[id].Y > p.Y) return 0;
    }else if (det(lower[id-1]-p,lower[id]-p)<0)return 0;
    id = lower_bound(upper.begin(), upper.end(), Pt(p.X, INF),
        greater<Pt>()) - upper.begin();
    if (upper[id].X == p.X) {
      if (upper[id].Y < p.Y) return 0;
    }else if (det(upper[id-1]-p,upper[id]-p)<0)return 0;
    return 1;
  }
  // 2. Find 2 tang pts on CH of a given outside point
  // return true with i0, i1 as index of tangent points
  // return false if inside CH
  bool get_tang(Pt p, int &i0, int &i1) {
    if (contain(p)) return false;
    i0 = i1 = 0;
    int id = lower_bound(lower.begin(), lower.end(), p) - lower
        .begin();
    bi_search(0, id, p, i0, i1);
    bi_search(id, (int)lower.size(), p, i0, i1);
    id = lower_bound(upper.begin(), upper.end(), p, greater<Pt
        >()) - upper.begin();
    bi_search((int)lower.size() - 1, (int)lower.size() - 1 + id
        , p, i0, i1);
    bi_search((int)lower.size() - 1 + id, (int)lower.size() - 1
         + (int)upper.size(), p, i0, i1);
    return true;
  }
  // 3. Find tangent points of a given vector
  // ret the idx of vertex has max cross value with vec
  int get_tang(Pt vec) {
    pair<LL, int> ret = get_tang(upper, vec);
    ret.second = (ret.second+(int)lower.size()-1)%n;
    ret = max(ret, get_tang(lower, vec));
    return ret.second;
  }
  // 4. Find intersection point of a given line
  // return 1 and intersection is on edge (i, next(i))
  // return 0 if no strictly intersection
  bool get_intersection(Pt u, Pt v, int &i0, int &i1) {
    int p0 = get_tang(u - v), p1 = get_tang(v - u);
    if (sign(det(v-u,a[p0]-u))*sign(det(v-u,a[p1]-u))<0) {
      if (p0 > p1) swap(p0, p1);
      i0 = bi_search(u, v, p0, p1);
      i1 = bi_search(u, v, p1, p0 + n);
      return 1;
    }
    return 0;
  }
};
```

11.5 Polar Angle Sort

```
#define is_neg(_k) (_k.Y < 0 || (_k.Y == 0 && _k.X < 0))
bool cmp(pll a,pll b) {
  int A = is_neg(a), B = is_neg(b);
  return (A == B ? (a ^ b) > 0 : A < B);
}
```

11.6 Circle and Polygon intersection

```
struct Circle_and_Segment_Intersection {
  const ld eps = 1e-9;
  vector<pdd> solve(pdd p1, pdd p2, pdd cen, ld r) {
    //please notice that p1 != p2
    //condiser p = p2 + (p1 - p2) * t, 0 <= t <= 1
    vector<pdd> ret;
    p1 = p1 - cen; p2 = p2 - cen;
    ld a = (p1 - p2) * (p1 - p2);
    ld b = 2 * (p2 * (p1 - p2));
    ld c = p2 * p2 - r * r;
    ld bb4ac = b * b - 4 * a * c;
    if (bb4ac < -eps) return ret; //no intersection
    vector<ld> ts;
    if ((bb4ac) <= eps) {
      ts.push_back(-b / 2 / a);
    }
    else {
      ts.push_back((-b + sqrt(bb4ac)) / (a * 2));
      ts.push_back((-b - sqrt(bb4ac)) / (a * 2));
    }
    sort(ts.begin(), ts.end());
    for (ld t: ts) {
      if (-eps <= t && t <= 1 + eps) {
        t = max(t, 0.0);
        t = min(t, 1.0);
        pdd pt = p2 + t * (p1 - p2);
        pt = pt + cen;
        ret.push_back(pt);
      }
    }
    return ret;
  }
} solver;

double f(ld a, ld b) {
  ld ret = b - a;
  while (ret <= -pi - eps) ret += 2 * pi;
```

```
  while (ret >= pi + eps) ret -= 2 * pi;
  return ret;
}

ld solve_small(pdd cen, ld r, pdd p1, pdd p2) {
  p1 = p1 - cen, p2 = p2 - cen;
  cen = {0, 0};
  vector<pdd> inter = solver.solve(p1, p2, cen, r);
  ld ret = 0.0;
  if ((int)inter.size() == 0) {
    if (in_cir(cen, r, p1)) {
      ret = (p1 ^ p2) / 2;
    }
    else {
      ret = (r * r * f(atan2(p1.Y, p1.X), atan2(p2.Y, p2.X))) /
            2;
    }
  }
  else if ((int)inter.size() == 1) {
    if (!in_cir(cen, r, p1) && !in_cir(cen, r, p2)) {
      //outside cut
      ret = (r * r * f(atan2(p1.Y, p1.X), atan2(p2.Y, p2.X))) /
            2;
    }
    else if (!in_cir(cen, r, p1)) {
      pdd _p1 = inter[0];
      ret +=  ((_p1 ^ p2) / 2);
      ret +=  (r * r * f(atan2(p1.Y, p1.X), atan2(_p1.Y, _p1.X)
           )) / 2;
    }
    else if (!in_cir(cen, r, p2)) {
      pdd _p2 = inter[0];
      ret +=  ((p1 ^ _p2) / 2);
      ret +=  (r * r * f(atan2(_p2.Y, _p2.X), atan2(p2.Y, p2.X)
           )) / 2;
    }
  }
  else if ((int)inter.size() == 2) {
    pdd _p2 = inter[0], _p1 = inter[1];
    ret +=  ((_p1 ^ _p2) / 2);
    ret +=  (r * r * f(atan2(_p2.Y, _p2.X), atan2(p2.Y, p2.X)))
          / 2;
    ret +=  (r * r * f(atan2(p1.Y, p1.X), atan2(_p1.Y, _p1.X)))
          / 2;
  }
  return ret;
}

ld solve(pdd cen, ld r, vector<pdd> pts) {
  ld ret = 0;
  for (int i = 0; i < (int)pts.size(); ++i) {
    ret += solve_small(cen, r, pts[i], pts[(i + 1) % int(pts.
         size())]);
  }
  ret = max(ret, -ret);
  return ret;
}
```

11.7 Segment Intersection

```
int intersect(PII a, PII b, PII c, PII d) {
  if (max(a.F, b.F) < min(c.F, d.F)) return 0;
  if (max(c.F, d.F) < min(a.F, b.F)) return 0;
  if (max(a.S, b.S) < min(c.S, d.S)) return 0;
  if (max(c.S, d.S) < min(a.S, b.S)) return 0;
  if (cross(b - a, c - a) * cross(b - a, d - a) == 1) return 0;
  if (cross(d - c, a - c) * cross(d - c, b - c) == 1) return 0;
  return 1;
}
```

11.8 Line Intersection Point

```
pdd intersect(pdd p1, pdd p2, pdd q1, pdd q2) {
  //make sure that p1p2 is not parallel to q1q2
  return p1 + ((q1 - p1) ^ (q2 - q1)) / ((p2 - p1) ^ (q2 - q1))
        * (p2 - p1);
}
```

11.9 Rotating Calipers

```
#define NXT(x)((x + 1) % m)
int main() {
  vector<pii> v; // v is the input points
  sort(v.begin(), v.end());
  vector<pii> up, down;
  for (pii p: v) {
    while (SZ(down) >= 2 && sgn((p - down[SZ(down) - 2]) ^ (p -
          down.back())) >= 0) {
      down.pop_back();
    }
    down.push_back(p);
  }
  reverse(v.begin(), v.end());
  for (pii p: v) {
    while (SZ(up) >= 2 && sgn((p - up[SZ(up) - 2]) ^ (p - up.
         back())) >= 0) {
      up.pop_back();
    }
    up.push_back(p);
  }
  vector<pii> all;
  for (pii p: down) all.push_back(p);
  all.pop_back();
  for (pii p: up) all.push_back(p);
  all.pop_back();
  int m = all.size();
  int ptr = (int) down.size() - 1;
  for (int i = 0; i < m; ++i) {
    while (((all[NXT(ptr)] - all[ptr]) ^ (all[NXT(i)] - all[i])
         ) > 0) {
      ptr = NXT(ptr);
    }
  }
}
```

12 Miscellaneous

12.1 Joseph Problem

```
// O(m + log N)
// n people, k-th dead. Find out the last alive person
int main() {
  long long n, k, i, x = 0, y;
  scanf("%I64d%I64d", &n, &k);
  for (i = 2; i <= k && i <= n; ++i) x = (x + k) % i;
  for (; i <= n; ++i) {
      y = (i - x - 1) / k;
      if (i + y > n) y = n - i;
      i += y;
      x = (x + (y + 1) % i * k) % i;
  }
  printf("%I64d\n", x + 1);
  return 0;
}
```

12.2 Segment Max Segment Sum

```
int n, m, x[MAX];
class N{
public: int tag, sml, sum, none;
} b[MAX * 4];
void Pull(int now, int l, int r) {
  if (l == r) {
    if (b[now].tag) {
      b[now].sum = b[now].tag;
      b[now].none = 0;
      b[now].sml = b[now].tag;
    }
    else{
      b[now].sum = 0;
      b[now].none = 1;
      b[now].sml = INF;
    }
  }
  else {
    b[now].sml = min(b[ls].sml, b[rs].sml);
    if (b[now].tag) b[now].sml = min(b[now].sml, b[now].tag);

    b[now].sum = b[ls].sum + b[rs].sum;
    b[now].none = b[ls].none + b[rs].none;
    if (b[now].tag) b[now].sum += b[now].tag * b[now].none, b[
         now].none = 0;
  }
}
void take_tag(int now, int l, int r, int val) {
  if (b[now].tag && b[now].tag < val) b[now].tag = 0;
  if (l != r && b[ls].sml < val) take_tag(ls, l, mid, val);
  if (l != r && b[rs].sml < val) take_tag(rs, mid + 1, r, val);
  Pull(now, l, r);
}
void Build(int now, int l, int r) {
  b[now].none = 0;
  if (l == r) b[now].tag = b[now].sml = b[now].sum = x[l];
  else {
```

```
    Build(ls, l, mid), Build(rs, mid + 1, r);
    Pull(now, l, r);
  }
}
void update(int now, int l, int r, int ql, int qr, int val) {
  if (b[now].tag >= val) return;
  if (ql <= l && r <= qr) {
    take_tag(now, l, r, val);
    b[now].tag = val;
    Pull(now, l, r);
  }
  else{
    if (qr <= mid) update(ls, l, mid, ql, qr, val);
    else if (mid + 1 <= ql) update(rs, mid + 1, r, ql, qr, val)
        ;
    else update(ls, l, mid, ql, qr, val), update(rs, mid + 1, r
        , ql, qr, val);
    Pull(now, l, r);
  }
}
PII query(int now, int l, int r, int ql, int qr) {
  if (ql <= l && r <= qr) return mp(b[now].sum, b[now].none);
  else {
    PII ans = mp(0, 0);
    if (qr <= mid) ans = query(ls, l, mid, ql, qr);
    else if (mid + 1 <= ql) ans = query(rs, mid + 1, r, ql, qr)
        ;
    else {
      PII a = query(ls, l, mid, ql, qr);
      PII b = query(rs, mid + 1, r, ql, qr);
      ans = mp(a.A + b.A, a.B + b.B);
    }
    if (b[now].tag != 0) ans.A += ans.B * b[now].tag, ans.B =
        0;
    return ans;
  }
}
REP(i, 1, n + 1) cin >> x[i];
Build(1, 1, n);
update(1, 1, n, l, r, v);
cout << query(1, 1, n, l, r).A << endl;
```

12.3 Stone Merge

```
int n, x[MAX], ans = 0;
vector<int> v;
int DFS(int now) {
  int val = v[now] + v[now + 1];
  ans += val;
  v.erase(v.begin() + now);
  v.erase(v.begin() + now);
  int id = 0;
  for (int i = now - 1; i >= 0; -- i)
    if (v[i] >= val) { id = i + 1; break; }
  v.insert(v.begin() + id, val);
  while (id >= 2 && v[id - 2] <= v[id]) {
    int dis = v.size() - id;
    DFS(id - 2);
    id = v.size() - dis;
  }
}
int32_t main() {
  IOS;
  cin >> n;
  for (int i = 0; i < n; ++ i) cin >> x[i];
  for (int i = 0; i < n; ++ i) {
    v.emplace_back(x[i]);
    while (v.size() >= 3 && v[v.size() - 3] <= v[v.size() - 1])
      DFS(v.size() - 3);
  }
  while (v.size() > 1) DFS(v.size() - 2);
  cout << ans << endl;
  return 0;
}
```

12.4 Manhattan Spanning Tree

```
typedef pair<int, PII> edge;
int n, sol[maxn];
PII x[maxn];
vector<edge> v;
class djs{
public:
  int x[maxn];
  void init() { for (int i = 0; i < maxn; ++ i) x[i] = i; }
  int Find(int now) { return x[now] == now ? now : x[now] =
      Find(x[now]); }
  void Union(int a, int b) { x[Find(a)] = Find(b); }
  int operator[](int now) { return Find(now); }
} ds;
PII bit[maxn];
void update(int from, int val, int id) {
  for (int i = from; i < maxn; i += i & -i)
    bit[i] = maxn(bit[i], mp(val, id));
}
int query(int from) {
  PII res = bit[from];
  for (int i = from; i > 0; i -= i & -i)
    res = maxn(res, bit[i]);
  return res.B;
}
int cmp(int a, int b) {
  return x[a] < x[b];
}
int DIS(int q, int w) {
  return abs(x[q].A - x[w].A) + abs(x[q].B - x[w].B);
}
void BuildEdge() {
  vector<int> uni;
  for (int i = 0; i < maxn; ++ i)
    bit[i] = mp(-INF, -1);
  for (int i = 0; i < n; ++ i) sol[i] = i;
  for (int i = 0; i < n; ++ i) uni.pb(x[i].B - x[i].A);
  sort(ALL(uni));
  uni.resize(unique(ALL(uni)) - uni.begin());
  sort(sol, sol + n, cmp);
  for (int i = 0; i < n; ++ i) {
    int now = sol[i];
    int tmp = x[sol[i]].B - x[sol[i]].A;
    int po = lower_bound(ALL(uni), tmp) - uni.begin() + 1;
    int id = query(po);
    if (id >= 0) v.pb(mp(DIS(id, now), mp(id, now)));
    update(po, x[now].A + x[now].B, now);
  }
}
void Build() {
  BuildEdge();
  for (int i = 0; i < n; ++ i) swap(x[i].A, x[i].B);
  BuildEdge();
  for (int i = 0; i < n; ++ i) x[i].A *= -1;
  BuildEdge();
  for (int i = 0; i < n; ++ i) swap(x[i].A, x[i].B);
  BuildEdge();
}
int solveKruskal() {
  ds.init();
  sort(ALL(v));
  int res = 0;
  for (int i = 0; i < v.size(); ++ i) {
    int dis = v[i].A;
    PII tmp = v[i].B;
    if (ds[tmp.A] != ds[tmp.B]) {
      ds.Union(tmp.A, tmp.B);
      res += dis;
    }
  }
  return res;
}
int32_t main() {
  IOS;
  cin >> n;
  for (int i = 0; i < n; ++ i) cin >> x[i].A >> x[i].B;
  Build();
  int ans = solveKruskal();
  cout << ans << endl;
  return 0;
}
```

12.5 K Cover Tree

```
int n, k, dp[MAX], ans;
vector<int> v[MAX];
void DFS(int now, int fa) {
  if (v[now].size() == 1 && v[now][0] == fa)
    return dp[now] = -1, void();
  int sml = INF, big = -INF;
  for (auto to : v[now]) if (to != fa) {
    DFS(to, now);
    sml = min(sml, dp[to]);
    big = max(big, dp[to]);
  }
  if (sml == -k) dp[now] = k, ans ++;
  else if (big - 1 >= abs(sml)) dp[now] = big - 1;
  else dp[now] = sml - 1;
```

```
}
int32_t main() {
  IOS;
  cin >> n >> k;
  REP(i, 2, n + 1) {
    int a, b; cin >> a >> b;
    v[a].pb(b); v[b].pb(a);
  }
  if (k == 0) cout << n << endl;
  else {
    DFS(0, 0), ans += dp[0] < 0;
    cout << ans << endl;
  }
  return 0;
}
```

12.6 M Segments' Maximum Sum

```
----------------------Greedy----------------------
int n, m, fr[MAX], ba[MAX];
int v[MAX], idx = 1;
set<PII> cc;
void erase(int id) {
  if (id == 0) return;
  int f = fr[id], b = ba[id];
  ba[fr[id]] = b, fr[ba[id]] = f;
  cc.erase(mp(abs(v[id]), id));
}
int32_t main() {
  cin >> n >> m;
  int sum = 0, pos = 0, ans = 0;
  for (int i = 0; i < n; ++ i) {
    int tmp; cin >> tmp;
    if (tmp == 0) continue;
    if ((tmp >= 0 && sum >= 0) || (tmp <= 0 && sum <= 0)) {
      sum += tmp;
    }
    else {
      if (sum > 0) ans += sum, pos ++;
      v[idx ++] = sum, sum = tmp;
    }
  }
  if (sum) v[idx ++] = sum;
  if (sum > 0) ans += sum, pos ++;
  REP(i, 0, idx) {
    fr[i + 1] = i;
    ba[i] = i + 1;
    if (i) cc.insert(mp(abs(v[i]), i));
  } ba[idx - 1] = 0;
  while (pos > m) {
    auto tmp = cc.begin();
    int val = (*tmp).A, id = (*tmp).B;
    cc.erase(tmp);
    if (v[id] < 0 && (fr[id] == 0 || ba[id] == 0)) continue;
    if (v[id] == 0) continue;
    ans -= val, pos --;
    v[id] = v[fr[id]] + v[id] + v[ba[id]];
    cc.insert(mp(abs(v[id]), id));
    erase(fr[id]), erase(ba[id]);
  }
  cout << ans << endl;
  return 0;
}
----------------------Aliens----------------------
int n, k, x[MAX];
PII dp[MAX], rd[MAX]; // max value, times, can be buy, times
int judge(int now) {
  dp[1] = mp(0, 0), rd[1] = mp(-x[1], 0);
  REP(i, 2, n + 1) {
    dp[i] = max(dp[i - 1],
      mp(rd[i - 1].A + x[i] - now, rd[i - 1].B + 1));
    rd[i] = max(rd[i - 1],
      mp(dp[i - 1].A - x[i]   , dp[i - 1].B));
  }
  return dp[n].B;
}
int32_t main() {
  IOS;
  cin >> n >> k;
  n ++;
  for (int i = 2; i <= n + 1; ++ i)
    cin >> x[i];
  for (int i = 1; i <= n; ++ i)
    x[i] += x[i - 1];
  if (judge(0) <= k) cout << dp[n].A << endl;
  else {
    int l = 0, r = 1000000000000LL;
    while (r - l > 1) {
      int mid = l + ((r - l) >> 1), res = judge(mid);
      if (res == k)
        return cout << dp[n].A + dp[n].B * mid << endl, 0;
      else if (res < k) r = mid;
      else if (res > k) l = mid;
    }
    judge(l);
    cout << dp[n].A + k * l << endl;
  }
  return 0;
}
```

12.7 Minimum Enclosing Cycle

```
typedef pair<double, double> pdd;
#define F first
#define S second

int n;
pdd a[maxn];
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());

double dis(pdd p1, pdd p2) {
  return hypot(p1.F - p2.F, p1.S - p2.S);
}
inline double sq(double x) {
  return x * x;
}
pdd external(pdd p1, pdd p2, pdd p3) {
  double a1 = p1.F - p2.F, a2 = p1.F - p3.F;
  double b1 = p1.S - p2.S, b2 = p1.S - p3.S;

  double c1 = (sq(p1.F) - sq(p2.F)
            + sq(p1.S) - sq(p2.S)) / 2;
  double c2 = (sq(p1.F) - sq(p3.F)
            + sq(p1.S) - sq(p3.S)) / 2;
  double dd = a1 * b2 - a2 * b1;
  return make_pair((c1 * b2 - c2 * b1) / dd
                 , (a1 * c2 - a2 * c1) / dd);
}
int main() {
  cin >> n;
  for (int i = 0; i < n; ++ i)
    cin >> a[i].F >> a[i].S;
  shuffle(a, a + n, rng);

  pdd center = a[0];
  double r = 0;

  for (int i = 0; i < n; ++ i) {
    if (dis(center, a[i]) <= r) continue;
    center = a[i], r = 0;
    for (int j = 0; j < i; ++ j) {
      if (dis(center, a[j]) <= r) continue;
      center.F = (a[i].F + a[j].F) / 2;
      center.S = (a[i].S + a[j].S) / 2;
      r = dis(center, a[i]);
      for (int k = 0; k < j; ++ k) {
        if (dis(center, a[k]) <= r) continue;
        center = external(a[i], a[j], a[k]);
        r = dis(center, a[i]);
      }
    }
  }

  cout << fixed << setprecision(10) << r << endl;
  cout << center.F << " " << center.S << "\n";
  return 0;
}
```

12.8 Rotating Sweep Line

```
PII p[maxn];
int n, idx[maxn], pos[maxn];
vector<PII> v;
inline PII operator + (PII x, PII y) {
  return make_pair(x.F + y.F, x.S + y.S); }
inline PII operator - (PII x, PII y) {
  return make_pair(x.F - y.F, x.S - y.S); }
inline long long cross(PII x, PII y) {
  return 1ll * x.F * y.S - 1ll * x.S * y.F; }
inline int cmp(PII x, PII y) {
  x = p[x.S] - p[x.F];
  y = p[y.S] - p[y.F];
  return cross(x, y) > 0;
}
```

```
int32_t main() {
  cin.tie(0), cout.sync_with_stdio(0);
  cin >> n >> wnt, wnt += wnt;
  for (int i = 1; i <= n; ++ i)
    cin >> p[i].F >> p[i].S;
  sort(p + 1, p + 1 + n);
  for (int i = 1; i <= n; ++ i)
    idx[i] = i, pos[i] = i;
  for (int i = 1; i <= n; ++ i)
    for (int j = i + 1; j <= n; ++ j)
      v.emplace_back(i, j);
  sort(v.begin(), v.end(), cmp);

  for (auto line : v) {
    int fr = pos[line.F], ba = pos[line.S], now;
    if (fr > ba) swap(fr, ba);
    // [TODO] points:
    // p[idx[    1]] more farther
    // p[idx[    2]] farther
    // p[idx[   fr]] ... p[idx[ba]]
    // p[idx[n - 1]] farther
    // p[idx[n - 0]] more farther
    swap(idx[fr], idx[ba]);
    swap(pos[line.F], pos[line.S]);
  }
    return 0;
}
```

12.9 Hilbert Curve

```
// soring Mo's with hilbert(nn, L, R) can be faster !!
// needed: nn >= n, no need to change n, nn = 2^k
// usage: sort (ql_i, qr_i) by hilbert(nn, ql_i, qr_i)
ll hilbert(int nn, int x, int y) {
  ll res = 0;
  for (int s = nn / 2; s; s >>= 1) {
    int rx = (x & s) > 0;
    int ry = (y & s) > 0;
    res += s * 1ll * s * ((3 * rx) ^ ry);
    if (ry == 0) {
      if (rx == 1) {
        x = s - 1 - x;
        y = s - 1 - y;
      }
      swap(x, y);
    }
  }
  return res;
}
```

12.10 Big Integer

```
struct Bigint {
  static const int LEN = 60;
  static const int BIGMOD = 10000;
  int s;
  int vl, v[LEN];
  //  vector<int> v;
  Bigint(): s(1) {
    vl = 0;
  }
  Bigint(long long a) {
    s = 1;
    vl = 0;
    if (a < 0) {
      s = -1;
      a = -a;
    }
    while (a) {
      push_back(a % BIGMOD);
      a /= BIGMOD;
    }
  }
  Bigint(string str) {
    s = 1;
    vl = 0;
    int stPos = 0, num = 0;
    if (!str.empty() && str[0] == '-') {
      stPos = 1;
      s = -1;
    }
    for (int i = SZ(str) - 1, q = 1; i >= stPos; i--) {
      num += (str[i] - '0') * q;
      if ((q *= 10) >= BIGMOD) {
        push_back(num);
        num = 0;
        q = 1;
      }
    }
    if (num) push_back(num);
    n();
  }
  int len() const {
    return vl; //return SZ(v);
  }
  bool empty() const {
    return len() == 0;
  }
  void push_back(int x) {
    v[vl++] = x; //v.PB(x);
  }
  void pop_back() {
    vl--; //v.pop_back();
  }
  int back() const {
    return v[vl - 1]; //return v.back();
  }
  void n() {
    while (!empty() && !back()) pop_back();
  }
  void resize(int nl) {
    vl = nl; //v.resize(nl);
    fill(v, v + vl, 0); //fill(ALL(v), 0);
  }
  void print() const {
    if (empty()) {
      putchar('0');
      return;
    }
    if (s == -1) putchar('-');
    printf("%d", back());
    for (int i = len() - 2; i >= 0; i--) printf("%.4d", v[i]);
  }
  friend std::ostream & operator << (std::ostream & out,
    const Bigint & a) {
    if (a.empty()) {
      out << "0";
      return out;
    }
    if (a.s == -1) out << "-";
    out << a.back();
    for (int i = a.len() - 2; i >= 0; i--) {
      char str[10];
      snprintf(str, 5, "%.4d", a.v[i]);
      out << str;
    }
    return out;
  }
  int cp3(const Bigint & b) const {
    if (s != b.s) return s - b.s;
    if (s == -1) return -(- * this).cp3(-b);
    if (len() != b.len()) return len() - b.len(); //int
    for (int i = len() - 1; i >= 0; i--)
      if (v[i] != b.v[i]) return v[i] - b.v[i];
    return 0;
  }
  bool operator < (const Bigint & b) const {
    return cp3(b) < 0;
  }
  bool operator <= (const Bigint & b) const {
    return cp3(b) <= 0;
  }
  bool operator == (const Bigint & b) const {
    return cp3(b) == 0;
  }
  bool operator != (const Bigint & b) const {
    return cp3(b) != 0;
  }
  bool operator > (const Bigint & b) const {
    return cp3(b) > 0;
  }
  bool operator >= (const Bigint & b) const {
    return cp3(b) >= 0;
  }
  Bigint operator - () const {
    Bigint r = (* this);
    r.s = -r.s;
    return r;
  }
  Bigint operator + (const Bigint & b) const {
    if (s == -1) return -(-(* this) + (-b));
    if (b.s == -1) return (* this) - (-b);
    Bigint r;
    int nl = max(len(), b.len());
    r.resize(nl + 1);
```

```
    for (int i = 0; i < nl; i++) {
      if (i < len()) r.v[i] += v[i];
      if (i < b.len()) r.v[i] += b.v[i];
      if (r.v[i] >= BIGMOD) {
        r.v[i + 1] += r.v[i] / BIGMOD;
        r.v[i] %= BIGMOD;
      }
    }
    r.n();
    return r;
  }
  Bigint operator - (const Bigint & b) const {
    if (s == -1) return -(-(* this) - (-b));
    if (b.s == -1) return (* this) + (-b);
    if ((* this) < b) return -(b - (* this));
    Bigint r;
    r.resize(len());
    for (int i = 0; i < len(); i++) {
      r.v[i] += v[i];
      if (i < b.len()) r.v[i] -= b.v[i];
      if (r.v[i] < 0) {
        r.v[i] += BIGMOD;
        r.v[i + 1]--;
      }
    }
    r.n();
    return r;
  }
  Bigint operator * (const Bigint & b) {
    Bigint r;
    r.resize(len() + b.len() + 1);
    r.s = s * b.s;
    for (int i = 0; i < len(); i++) {
      for (int j = 0; j < b.len(); j++) {
        r.v[i + j] += v[i] * b.v[j];
        if (r.v[i + j] >= BIGMOD) {
          r.v[i + j + 1] += r.v[i + j] / BIGMOD;
          r.v[i + j] %= BIGMOD;
        }
      }
    }
    r.n();
    return r;
  }
  Bigint operator / (const Bigint & b) {
    Bigint r;
    r.resize(max(1, len() - b.len() + 1));
    int oriS = s;
    Bigint b2 = b; // b2 = abs(b)
    s = b2.s = r.s = 1;
    for (int i = r.len() - 1; i >= 0; i--) {
      int d = 0, u = BIGMOD - 1;
      while (d < u) {
        int m = (d + u + 1) >> 1;
        r.v[i] = m;
        if ((r * b2) > (* this)) u = m - 1;
        else d = m;
      }
      r.v[i] = d;
    }
    s = oriS;
    r.s = s * b.s;
    r.n();
    return r;
  }
  Bigint operator % (const Bigint & b) {
    return (* this) - (* this) / b * b;
  }
};
```