Seoul National University

Omogen Heap

sharaelong, whqkrkt04, serverrepairman

2023-11-24

Contest (1)

template.cpp

37 lines

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<int, pii> piii;
typedef pair<ll, ll> pll;
typedef pair<ll, pll> plll;
#define fi first
#define se second
const int INF = 1e9+1;
const int P = 1000000007;
const ll LLINF = (ll)1e18+1;
template <typename T>
ostream& operator<<(ostream& os, const vector<T>& v) { for(auto
    i : v) os << i << " "; os << "\n"; return os; }
template <typename T1, typename T2>
ostream& operator<<(ostream& os, const pair<T1, T2>& p) { os <<
    p.fi << " " << p.se; return os; }
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
#define rnd(x, y) uniform_int_distribution<int>(x, y)(rng)

ll mod(ll a, ll b) { return ((a%b) + b) % b; }
ll ext_gcd(ll a, ll b, ll &x, ll &y) {
    ll g = a; x = 1, y = 0;
    if(b) g = ext_gcd(b, a % b, y, x), y -= a / b * x;
    return g;
}
ll inv(ll a, ll m) {
    ll x, y; ll g = ext_gcd(a, m, x, y);
    if(g > 1) return -1;
    return mod(x, m);
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    return 0;
}
```

.bashrc

3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14 \
  -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <>
```

hash.sh

3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

General Tips

- 뭐라도 하자. 조급해지지 말기.
- 진전이 없다면 버려라. 설령 아무리 예쁜 아이디어처럼 보여도!
- 현재 상황을 파악: 풀 수 있는가? 다항시간에 되는가? 어느 풀이까지 아는가?
- 지문 다시 읽기, 예제와 이해한 내용 대조하기
- 관찰을 시도 중인지 / 계산의 개선을 시도 중인지 구분하기
- 지금까지 고민한 걸로 문제가 이미 풀려있을 수도 있다.
- mathematically formulate, bold hypothesis

Ideas.

- 문제가 매우 나이브해졌다고 생각되는 시점까지 왔는가?
  - dnc, 루트질, 오프라인, pbs을 생각해보자
- 과정이 어렵다면, 결과를 생각하자.
  - 거꾸로 생각하기. 연산을 뒤집든, 순서를 뒤집든, 문자열을 뒤집든, ...
  - 뭐든지 어려운 걸 붙잡고 있지 말자. 그리디일 수도 있으니까.
- 추가적인 성질을 줘도 최적해를 찾을 수 있을까?
  - exchange, 불필요한 이동, 상태에 제약조건 걸기, ...
  - 혹은 특정 성질을 가지는 구간들로 해를 나눠서 생각하기
- 최대, 최소, 불변, 맨 왼쪽(아래쪽)인 object를 생각했을 때 어떤 성질이 있나?
- 불변량이 있는가?
- 경계에 위치한 무언가가 더 특수한가?
  - 선형적으로 연산을 합칠 수 없는 경우
  - 범위에 한계가 있는 경우
- 최초로 무언가가 아닌, 달라지는 등등의 위치나 값을 생각하기
  - 문제의 제약조건을 완화시켜보자
- 본질적인 다양성인가? closed form을 찾을 수 있는 기대를 할 수 있나?
- (특히 조합론) 자동 결정되는 요소가 있나?
- 상태공간을 더 넓게 보기, 조건을 떼고 생각하기
- 중요한 상태는 전체 상태에 비해 매우 작을 수 있다
- 가설, 추정이라도 마찬가지. 더 강화하기, 더 약화하기.
- 이미 계산한 위치이면 (비슷한 상태이면) 다른 경우에도 계산하지 않아도 되는가?
- 답이나 어떤 변수가 취할 수 있는 범위가 알고 보면 작은가?
- 단조성이 있는지, opt를 적용할 수 있는지?
- 볼록성이 있는지?
  - slope trick은 볼록함수를 쉽게 합쳐주게 해주는 무언가이다.
  - 볼록성과 greedy의 관련성: 비효율적이더라도 flow로 모델링이 되는가?
- 어떤 형태의 정규화가 가능한지, 그러한 정규화가 문제를 더 단순하게 바꿀 수 있나?
- 재귀적으로 접근할 수 있을 여지가 있나?
- log개로 쪼개기, 절반씩 쪼개기, 2진법, euclid 호제법, chessboard
- Maximal is exclusive

Data structures (2)

LazySegmentTree.h

Description: 0-index, [l, r] interval
Usage: `SegmentTree seg(n); seg.query(l, r); seg.update(l, r, val);`

d41d8c, 64 lines

```
struct SegmentTree {
  int n, h;
  vector<int> arr;
  vector<int> lazy;
  SegmentTree(int _n) : n(_n) {
  h = Log2(n);
  n = 1 << h;
  arr.resize(2*n, 0);
  lazy.resize(2*n, 0);
}

  void update(int l, int r, int c) {
    l += n, r += n;
    for (int i=h; i>=1; --i) {
      if (l >> i << i != l) push(l >> i);
      if ((r+1) >> i << i != (r+1)) push(r >> i);
    }
    for (int L=l, R=r; L<=R; L/=2, R/=2) {
      if (L & 1) apply(L++, c);
      if (~R & 1) apply(R--, c);
    }
    for (int i=1; i<=h; ++i) {
      if (l >> i << i != l) pull(l >> i);
      if ((r+1) >> i << i != (r+1)) pull(r >> i);
    }
  }

  int query(int l, int r) {
    l += n, r += n;
    for (int i=h; i>=1; --i) {
      if (l >> i << i != l) push(l >> i);
      if ((r+1) >> i << i != (r+1)) push(r >> i);
    }
    int ret = 0;
    for (; l <= r; l/=2, r/=2) {
      if (l & 1) ret = max(ret, arr[l++]);
      if (~r & 1) ret = max(ret, arr[r--]);
    }
    return ret;
  }

  void push(int x) {
    if (lazy[x] != 0) {
      apply(2*x, lazy[x]);
      apply(2*x+1, lazy[x]);
      lazy[x] = 0;
    }
  }

  void apply(int x, int c) {
    arr[x] = max(arr[x], c);
    if (x < n) lazy[x] = c;
  }

  void pull(int x) {
    arr[x] = max(arr[2*x], arr[2*x+1]);
  }

  static int Log2(int x){
    int ret = 0;
    while (x > (1 << ret)) ret++;
    return ret;
  }
};
```

ConvexHullTrick.h

Description: Max query, call init() before use.

d41d8c, 55 lines

```
struct Line{
  ll a, b, c; // y = ax + b, c = line index
  Line(ll a, ll b, ll c) : a(a), b(b), c(c) {}
  ll f(ll x){ return a * x + b; }
};
vector<Line> v; int pv;
void init(){ v.clear(); pv = 0; }
int chk(const Line &a, const Line &b, const Line &c) const {
```

```
  return (__int128_t)(a.b - b.b) * (b.a - c.a) <=
    (__int128_t)(c.b - b.b) * (b.a - a.a);
}
void insert(Line l){
  if(v.size() > pv && v.back().a == l.a){
    if(l.b < v.back().b) l = v.back(); v.pop_back();
  }
  while(v.size() >= pv+2 && chk(v[v.size()-2], v.back(), l))
    v.pop_back();
  v.push_back(l);
}
p query(ll x){ // if min query, then v[pv].f(x) >= v[pv+1].f(x)
  while(pv+1 < v.size() && v[pv].f(x) <= v[pv+1].f(x)) pv++;
  return {v[pv].f(x), v[pv].c};
}

// Container where you can add lines of the form kx+m, and
    query maximum values at points x.
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

FenwickTree.h

Description: 0-indexed. (1-index for internal bit trick)
Usage: `FenwickTree fen(n); fen.add(x, val); fen.sum(x);`

d41d8c, 12 lines

```
struct FenwickTree {
  vector<int> tree;
  FenwickTree(int size) { tree.resize(size+1, 0); }
  int sum(int pos) {
    int ret = 0;
    for (int i=pos+1; i>0; i &= (i-1)) ret += tree[i];
    return ret;
  }
  void add(int pos, int val) {
    for (int i=pos+1; i<tree.size(); i+=(i & -i)) tree[i] +=
        val;
  }
};
```

HLD.h

d41d8c, 54 lines

```
class HLD {
private:
  vector<vector<int>> adj;
  vector<int> in, sz, par, top, depth;
  void traverse1(int u) {
    sz[u] = 1;
    for (int &v: adj[u]) {
      adj[v].erase(find(adj[v].begin(), adj[v].end(), u));
      depth[v] = depth[u] + 1;
      traverse1(v);
      par[v] = u;
      sz[u] += sz[v];
      if (sz[v] > sz[adj[u][0]]) swap(v, adj[u][0]);
    }
  }
  void traverse2(int u) {
    static int n = 0;
    in[u] = n++;
    for (int v: adj[u]) {
      top[v] = (v == adj[u][0] ? top[u] : v);
      traverse2(v);
    }
  }
public:
  void link(int u, int v) { // u and v is 1-based
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  void init() { // have to call after linking
    top[1] = 1;
    traverse1(1);
    traverse2(1);
  }
  // u is 1-based and returns dfs-order [s, e) 0-based index
  pii subtree(int u) {
    return {in[u], in[u] + sz[u]};
  }
  // u and v is 1-based and returns array of dfs-order [s, e)
      0-based index
  vector<pii> path(int u, int v) {
    vector<pii> res;
    while (top[u] != top[v]) {
      if (depth[top[u]] < depth[top[v]]) swap(u, v);
      res.emplace_back(in[top[u]], in[u] + 1);
      u = par[top[u]];
    }
    res.emplace_back(min(in[u], in[v]), max(in[u], in[v]) + 1);
    return res;
  }
  HLD(int n) { // n is number of vertexes
    adj.resize(n+1); depth.resize(n+1);
    in.resize(n+1); sz.resize(n+1);
    par.resize(n+1); top.resize(n+1);
  }
};
```

PBDS.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change `null_type`.
Time: $\mathcal{O}(\log N)$

d41d8c, 13 lines

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
template<class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
                         tree_order_statistics_node_update>;
int main() {
  ordered_set<int> X;
  for (int i=1; i<10; i+=2) X.insert(i); // 1 3 5 7 9
  cout << *X.find_by_order(2) << endl; // 5
  cout << X.order_of_key(6) << endl; // 3
  cout << X.order_of_key(7) << endl; // 3
  X.erase(3);
}
```

Rope.h

Description: 1 x y: Move SxSx+1...Sy to front of string. $(0 \le x \le y < N)$ 2 x y: Move SxSx+1...Sy to back of string. $(0 \le x \le y < N)$ 3 x: Print Sx. $(0 \le x < N)$ cf. rope.erase(index, count) : erase [index, index+count)

<ext/rope> d41d8c, 23 lines

```
using namespace __gnu_cxx;
int main() {
  string s; cin >> s;
  rope<char> R;
  R.append(s.c_str());
  int q; cin >> q;
  while(q--) {
    int t, x, y; cin >> t;
    switch(t) {
    case 1:
      cin >> x >> y; y++;
      R = R.substr(x, y-x) + R.substr(0, x) + R.substr(y, s.
          size());
      break;
    case 2:
      cin >> x >> y; y++;
      R = R.substr(0, x) + R.substr(y, s.size()) + R.substr(x,
          y-x);
      break;
    default:
      cin >> x;
      cout << R[x] << "\n";
    }
  }
}
```

PersistentSegmentTree.h

Description: Point update (addition), range sum query
Usage: Unknown, but just declare sufficient size. You should achieve root number manually after every query/update.

d41d8c, 69 lines

```
struct PersistentSegmentTree {
  int size;
  int last_root;
  vector<ll> tree, l, r;

  PersistentSegmentTree(int _size) {
    size = _size;
    init(0, size-1);
    last_root = 0;
  }

  void add_node() {
    tree.push_back(0);
    l.push_back(-1);
    r.push_back(-1);
  }

  int init(int nl, int nr) {
    int n = tree.size();
    add_node();
    if (nl == nr) {
      tree[n] = 0;
      return n;
    }
    int mid = (nl + nr) / 2;
    l[n] = init(nl, mid);
```

```
    r[n] = init(mid+1, nr);
    return n;
  }

  void update(int ori, int pos, int val, int nl, int nr) {
    int n = tree.size();
    add_node();
    if (nl == nr) {
      tree[n] = tree[ori] + val;
      return;
    }

    int mid = (nl + nr) / 2;
    if (pos <= mid) {
      l[n] = tree.size();
      r[n] = r[ori];
      update(l[ori], pos, val, nl, mid);
    } else {
      l[n] = l[ori];
      r[n] = tree.size();
      update(r[ori], pos, val, mid+1, nr);
    }
    tree[n] = tree[l[n]] + tree[r[n]];
  }

  void update(int pos, int val) {
    int new_root = tree.size();
    update(last_root, pos, val, 0, size-1);
    last_root = new_root;
  }

  ll query(int a, int b, int n, int nl, int nr) {
    if (n == -1) return 0;
    if (b < nl || nr < a) return 0;
    if (a <= nl && nr <= b) return tree[n];
    int mid = (nl + nr) / 2;
    return query(a, b, l[n], nl, mid) + query(a, b, r[n], mid
        +1, nr);
  }

  ll query(int x, int root) {
    return query(0, x, root, 0, size-1);
  }
};
```

SplayTree.h

d41d8c, 152 lines

```
typedef ll TCON; // content
const TCON initval = 0;
typedef ll TV; // subtree value
const TV id = 0;
typedef ll TLAZ; // lazy value
const TLAZ S_unused = 0;

struct Snode{
  Snode *l, *r, *p;
  int cnt;

  TCON content = initval;
  TV val;
  TLAZ lazy = S_unused;

  void init(){
    // Initialize value using CONTENT
    val = content;
  }
  TV combine(TV a, TV b){
    // Real value when a <-- b
    return a+b;
  }
  TLAZ combineL(TLAZ a, TLAZ b){
    // Lazy value when a <-- b
    return a+b;
  }
  void unlazy_inner(){
    // Update CONTENT and VAL using LAZY
    content+= lazy;
    val+= lazy * cnt;
  }

  void update(){
    cnt = 1;
    init();
    if(l) l->unlazy(), cnt+= l->cnt,
            val = combine(l->val, val);
    if(r) r->unlazy(), cnt+= r->cnt,
            val = combine(val, r->val);
  }

  void lazy_add(TLAZ x){lazy = combineL(lazy, x);}
  void unlazy(){
    if(lazy == S_unused) return;
    unlazy_inner();
    if(l) l->lazy_add(lazy);
    if(r) r->lazy_add(lazy);
    lazy = S_unused;
  }

  void debug_inorder(){
    unlazy();
    if(l) l->debug_inorder();
    //cout << content << ' ';
    if(r) r->debug_inorder();
  }
};

// 1-indexed; has sentinel nodes on both ends
struct Splay{
  Snode *root;

  Splay(int n){
    Snode *x;
    root = x = new Snode;
    x->l = x->r = x->p = NULL;
    x->cnt = n, x->lazy = S_unused;
    x->init();
    for(int i=1; i<n+2; i++){
      x->r = new Snode;
      x->r->p = x; x = x->r;
      x->l = x->r = NULL;
      x->cnt = n-i, x->lazy = S_unused;
      x->init();
    }
  }

  void rotate(Snode *x){
    // x goes to parent of x
    Snode *p = x->p, *b = NULL;
    if(x == p->l) p->l = b = x->r, x->r = p;
    else p->r = b = x->l, x->l = p;
    x->p = p->p, p->p = x;
    if(b) b->p = p;
    (x->p ? p == x->p->l ? x->p->l : x->p->r : root) = x;
    p->update(), x->update();
  }

  Snode* splay(Snode *x){
    // x becomes the root
    while(x->p){
      Snode *p = x->p, *g = p->p;
      if(g) rotate((x == p->l) == (p == g->l) ? p : x);
      rotate(x);
    }
    return root = x;
  }

  Snode* kth(int k){
    // kth becomes the root
    // DO NOT USE IT FOR POINT UPDATE!! USE INTERVAL(k,k)!!!!
    Snode *x = root; x->unlazy();
    while(1){
      while(x->l && x->l->cnt > k) (x = x->l)->unlazy();
      if(x->l) k-= x->l->cnt;
      if(!k--) break;
      (x = x->r)->unlazy();
    }
    return splay(x);
  }

  Snode* interval(int l, int r){
    // l to r goes to root->r->l
    kth(l-1);
    Snode *x = root;
    root = x->r; root->p = NULL;
    kth(r-l+1);
    x->r= root; root->p = x; root = x;
    (x = root->r->l)->unlazy();
    return x;
  }

  void insert(int k, TCON v){
    // insert CONTENT v at index k, which becomes root
    kth(k);
    Snode *x = new Snode;
    if(root->l) root->l->p = x;
    x->l = root->l; root->l = x; x->p = root; x->r = NULL;
    x->content = v; x->init();
    splay(x);
  }

  void remove(int k){
    // remove k-th node
    kth(k);
    Snode *p = root;
    p->unlazy();
    if(p->l){
      if(p->r){
        root = p->l; root->p = NULL;
        Snode *cur = root;
        cur->unlazy();
        while (cur->r) cur = cur->r, cur->unlazy();
        cur->r = p->r; p->r->p = cur;
        splay(cur); delete p;
      }
      else{root = p->l; root->p = NULL; delete p;}
    }
    else{root = p->r; if(root) root->p = NULL; delete p;}
  }
};
```

Graph (3)

3.1 Fundamentals

Bridge.h

Description: Undirected connected graph, no self-loop. Find every bridges. Usual graph representation. dfs(here, par): returns fastest vertex which connected by some node in subtree of here, except here-parent edge.
Time: $\mathcal{O}(V+E)$, 180ms for $V=10^5$ and $E=10^6$ graph.

d41d8c, 23 lines

```cpp
const int MAX_N = 1e5 + 1;

vector<int> adj[MAX_N];
vector<pii> bridges;
int in[MAX_N];
int cnt = 0;

int dfs(int here, int parent = -1) {
  in[here] = cnt++;
  int ret = 1e9;
  for (int there: adj[here]) {
    if (there != parent) {
      if (in[there] == -1) {
        int subret = dfs(there, here);
        if (subret > in[here]) bridges.push_back({here, there});
        ret = min(ret, subret);
      } else {
        ret = min(ret, in[there]);
      }
    }
  }
  return ret;
}
```

KthShortestPath.h

Description: Calculate Kth shortest path from s to t.
Usage: 0-base index. Vertex is 0 to n-1. `KthShortestPath g(n); g.add_edge(s, e, cost); g.run(s, t, k);`
Time: $\mathcal{O}(E\log V + K\log K)$, $V=E=K=3\times10^5$ in 312ms, 144MB at yosupo.

d41d8c, 75 lines

```cpp
struct KthShortestPath {
  struct node{
    array<node*, 2> son; pair<ll, ll> val;
    node() : node(make_pair(-1e18, -1e18)) {}
    node(pair<ll, ll> val) : node(nullptr, nullptr, val) {}
    node(node *l, node *r, pair<ll, ll> val) : son({l,r}), val(val) {}
  };
  node* copy(node *x){ return x ? new node(x->son[0], x->son[1], x->val) : nullptr; }
  node* merge(node *x, node *y){ // precondition: x, y both points to new entity
    if(!x || !y) return x ? x : y;
    if(x->val > y->val) swap(x, y);
    int rd = rnd(0, 1);
    if(x->son[rd]) x->son[rd] = copy(x->son[rd]);
    x->son[rd] = merge(x->son[rd], y); return x;
  }

  struct edge{
    ll v, c, i; edge() = default;
    edge(ll v, ll c, ll i) : v(v), c(c), i(i) {}
  };

  vector<vector<edge>> gph, rev;
  int idx;
  vector<int> par, pae; vector<ll> dist; vector<node*> heap;

  KthShortestPath(int n) {
    gph = rev = vector<vector<edge>>(n);
    idx = 0;
  }

  void add_edge(int s, int e, ll x){
    gph[s].emplace_back(e, x, idx);
    rev[e].emplace_back(s, x, idx);
    assert(x >= 0); idx++;
  }

  void dijkstra(int snk){ // replace this to SPFA if edge weight is negative
    int n = gph.size();
    par = pae = vector<int>(n, -1);
    dist = vector<ll>(n, 0x3f3f3f3f3f3f3f3f);
    heap = vector<node*>(n, nullptr);
    priority_queue<pair<ll,ll>, vector<pair<ll,ll>>, greater<>> pq;
    auto enqueue = [&](int v, ll c, int pa, int pe){
      if(dist[v] > c) dist[v] = c, par[v] = pa, pae[v] = pe, pq.emplace(c, v);
    }; enqueue(snk, 0, -1, -1); vector<int> ord;
    while(!pq.empty()){
      auto [c,v] = pq.top(); pq.pop(); if(dist[v] != c) continue;
      ord.push_back(v); for(auto e : rev[v]) enqueue(e.v, c+e.c, v, e.i);
    }
    for(auto &v : ord){
      if(par[v] != -1) heap[v] = copy(heap[par[v]]);
      for(auto &e : gph[v]){
        if(e.i == pae[v]) continue;
        ll delay = dist[e.v] + e.c - dist[v];
        if(delay < 1e18) heap[v] = merge(heap[v], new node(make_pair(delay, e.v)));
      }
    }
  }
  vector<ll> run(int s, int e, int k){
    using state = pair<ll, node*>; dijkstra(e); vector<ll> ans;
    priority_queue<state, vector<state>, greater<state>> pq;
    if(dist[s] > 1e18) return vector<ll>(k, -1);
    ans.push_back(dist[s]);
    if(heap[s]) pq.emplace(dist[s] + heap[s]->val.first, heap[s]);
    while(!pq.empty() && ans.size() < k){
      auto [cst, ptr] = pq.top(); pq.pop(); ans.push_back(cst);
      for(int j=0; j<2; j++) if(ptr->son[j])
                            pq.emplace(cst-ptr->val.first + ptr->son[j]->val.first, ptr->son[j]);
      int v = ptr->val.second;
      if(heap[v]) pq.emplace(cst + heap[v]->val.first, heap[v]);
    }
    while(ans.size() < k) ans.push_back(-1);
    return ans;
  }
};
```

TreeIsomorphism.h

Description: Calculate hash of given tree.
Usage: 1-base index. `t.init(n); t.add_edge(a, b); (size, hash) = t.build(void); // size may contain dummy centroid.`
Time: $\mathcal{O}(N\log N)$, $N=30$ and $\sum N\le10^6$ in 256ms.

d41d8c, 74 lines

```cpp
const int MAX_N = 33;
ull A[MAX_N], B[MAX_N];

struct Tree {
  int n;
  vector<int> adj[MAX_N];
  int sz[MAX_N];
  vector<int> cent; // sz(cent) <= 2
  Tree() {}

  void init(int n) {
    this->n = n;
    for (int i=0; i<n+2; ++i) adj[i].clear();
    fill(sz, sz+n+2, 0);
    cent.clear();
  }

  void add_edge(int s, int e) {
    adj[s].push_back(e);
    adj[e].push_back(s);
  }

  int get_cent(int v, int b = -1) {
    sz[v] = 1;
    for (auto i: adj[v]) {
      if (i != b) {
        int now = get_cent(i, v);
        if (now <= n/2) sz[v] += now;
        else break;
      }
    }
    if (n - sz[v] <= n/2) cent.push_back(v);
    return sz[v];
  }

  int init() {
    get_cent(1);
    if (cent.size() == 1) return cent[0];
    int u = cent[0], v = cent[1], add = ++n;
    adj[u].erase(find(adj[u].begin(), adj[u].end(), v));
    adj[v].erase(find(adj[v].begin(), adj[v].end(), u));
    adj[add].push_back(u); adj[u].push_back(add);
    adj[add].push_back(v); adj[v].push_back(add);
    return add;
  }

  pair<int, ull> build(int v, int p = -1, int d = 1) {
    vector<pair<int, ull>> ch;
    for (auto i: adj[v]) {
      if (i != p) ch.push_back(build(i, v, d+1));
    }
    if (ch.empty()) return { 1, d };

    sort(ch.begin(), ch.end());
    ull ret = d;
    int tmp = 1;
    for (int j=0; j<ch.size(); ++j) {
      ret += A[d] ^ B[j] ^ ch[j].second;
      tmp += ch[j].first;
    }
    return { tmp, ret };
  }

  pair<int, ull> build() {
    return build(init());
  }
};

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
```

```
uniform_int_distribution<ull> urnd;

void solve() {
  for (int i=0; i<MAX_N; ++i) A[i] = urnd(rng), B[i] = urnd(rng
       );
}
```

CentroidDecomposition.h

Description: Centroid decomposition.
Usage: check for structure of decomposition.

d41d8c, 39 lines

```
vector<int> adj[MAX_N];
int sz[MAX_N];
bool decomposed[MAX_N];
int ctpar[MAX_N];

int dfs(int here, int par = -1) {
  sz[here] = 1;
  for (int there: adj[here]) {
    if (there != par && !decomposed[there]) sz[here] += dfs(
         there, here);
  }
  return sz[here];
}

int get_cent(int here, int par, int capa) {
  for (int there: adj[here]) {
    if (there != par && !decomposed[there] && sz[there] > capa)
         return get_cent(there, here, capa);
  }
  return here;
}

void init(int here, int prev_cent = -1) {
  int size = dfs(here);
  int cent = get_cent(here, -1, size/2);
  decomposed[cent] = true;
  ctpar[cent] = prev_cent;
  for (int there: adj[cent]) {
    if (!decomposed[there]) {
      init(there, cent);
    }
  }
}

void update(int v) {
  for (int vp = v; vp != -1; vp = ctpar[vp]) { // do sth }
  }

  void solve() {
    init(1);
  }
```

3.2 Network flow

MinCostMaxFlow.h

Description: Set MAXN. Overflow is not checked.
Usage: MCMF g; g.add_edge(s, e, cap, cost); g.solve(src, sink, total_size);
Time: 216ms on almost K_n graph, for $n = 300$.

d41d8c, 90 lines

```
const int MAXN = 800 + 5;

struct MCMF {
  struct Edge{ int pos, cap, rev; ll cost; };
  vector<Edge> gph[MAXN];
  void clear(){
    for(int i=0; i<MAXN; i++) gph[i].clear();
  }
  void add_edge(int s, int e, int x, ll c){
    gph[s].push_back({e, x, (int)gph[e].size(), c});
    gph[e].push_back({s, 0, (int)gph[s].size()-1, -c});
  }
  ll dist[MAXN];
  int pa[MAXN], pe[MAXN];
  bool inque[MAXN];
  bool spfa(int src, int sink, int n){
    memset(dist, 0x3f, sizeof(dist[0]) * n);
    memset(inque, 0, sizeof(inque[0]) * n);
    queue<int> que;
    dist[src] = 0;
    inque[src] = 1;
    que.push(src);
    bool ok = 0;
    while(!que.empty()){
      int x = que.front();
      que.pop();
      if(x == sink) ok = 1;
      inque[x] = 0;
      for(int i=0; i<gph[x].size(); i++){
        Edge e = gph[x][i];
        if(e.cap > 0 && dist[e.pos] > dist[x] + e.cost){
          dist[e.pos] = dist[x] + e.cost;
          pa[e.pos] = x;
          pe[e.pos] = i;
          if(!inque[e.pos]){
            inque[e.pos] = 1;
            que.push(e.pos);
          }
        }
      }
    }
    return ok;
  }
  ll new_dist[MAXN];
  pair<bool, ll> dijkstra(int src, int sink, int n){
    priority_queue<pii, vector<pii>, greater<pii> > pq;
    memset(new_dist, 0x3f, sizeof(new_dist[0]) * n);
    new_dist[src] = 0;
    pq.emplace(0, src);
    bool isSink = 0;
    while(!pq.empty()) {
      auto tp = pq.top(); pq.pop();
      if(new_dist[tp.second] != tp.first) continue;
      int v = tp.second;
      if(v == sink) isSink = 1;
      for(int i = 0; i < gph[v].size(); i++){
        Edge e = gph[v][i];
        ll new_weight = e.cost + dist[v] - dist[e.pos];
        if(e.cap > 0 && new_dist[e.pos] > new_dist[v] +
             new_weight){
          new_dist[e.pos] = new_dist[v] + new_weight;
          pa[e.pos] = v;
          pe[e.pos] = i;
          pq.emplace(new_dist[e.pos], e.pos);
        }
      }
    }
    return make_pair(isSink, new_dist[sink]);
  }

  pair<ll, ll> solve(int src, int sink, int n){
    spfa(src, sink, n);
    pair<bool, ll> path;
    pair<ll,ll> ret = {0,0};
    while((path = dijkstra(src, sink, n)).first){
      for(int i = 0; i < n; i++) dist[i] += min(ll(2e15),
           new_dist[i]);
      ll cap = 1e18;
      for(int pos = sink; pos != src; pos = pa[pos]){
        cap = min(cap, (ll)gph[pa[pos]][pe[pos]].cap);
      }
      ret.first += cap;
      ret.second += cap * (dist[sink] - dist[src]);
      for(int pos = sink; pos != src; pos = pa[pos]){
        int rev = gph[pa[pos]][pe[pos]].rev;
        gph[pa[pos]][pe[pos]].cap -= cap;
        gph[pos][rev].cap += cap;
      }
    }
    return ret;
  }
};
```

Dinic.h

Description: 0-indexed. cf) $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching.
Usage: Dinic g(n); g.add_edge(u, v, cap_uv, cap_vu); g.max_flow(s, t); g.clear_flow();

d41d8c, 73 lines

```
struct Dinic {
  struct Edge { int a; ll flow; ll cap; int rev; };
  int n, s, t;
  vector<vector<Edge>> adj;
  vector<int> level;
  vector<int> cache;
  vector<int> q;
Dinic(int _n) : n(_n) {
  adj.resize(n);
  level.resize(n);
  cache.resize(n);
  q.resize(n);
}

  bool bfs() {
    fill(level.begin(), level.end(), -1);
    level[s] = 0;
    int l = 0, r = 1;
    q[0] = s;
    while (l < r) {
      int here = q[l++];
      for (auto[there, flow, cap, rev]: adj[here]) {
        if (flow < cap && level[there] == -1) {
          level[there] = level[here] + 1;
          if (there == t) return true;
          q[r++] = there;
        }
      }
    }
    return false;
  }

  ll dfs(int here, ll extra_capa) {
    if (here == t) return extra_capa;
    for (int& i=cache[here]; i<adj[here].size(); ++i) {
      auto[there, flow, cap, rev] = adj[here][i];
      if (flow < cap && level[there] == level[here] + 1) {
        ll f = dfs(there, min(extra_capa, cap-flow));
        if (f > 0) {
          adj[here][i].flow += f;
          adj[there][rev].flow -= f;
          return f;
        }
      }
    }
    return 0;
  }
```

```
  void clear_flow() {
    for (auto& v: adj) {
      for (auto& e: v) e.flow = 0;
    }
  }

  ll max_flow(int _s, int _t) {
    s = _s, t = _t;
    ll ret = 0;
    while (bfs()) {
      fill(cache.begin(), cache.end(), 0);
      while (true) {
        ll f = dfs(s, 2e18);
        if (f == 0) break;
        ret += f;
      }
    }
    return ret;
  }

  void add_edge(int u, int v, ll uv, ll vu) {
    adj[u].push_back({ v, 0, uv, (int)adj[v].size() });
    adj[v].push_back({ u, 0, vu, (int)adj[u].size()-1 });
  }
};
```

Hungarian.h

Description: Bipartite minimum weight matching. 1-base indexed. A[1..n][1..m] and $n \le m$ needed. pair(cost, matching) will be returned.
Usage: `auto ret = hungarian(A);`
Time: $\mathcal{O}\left(n^2m\right)$, and 100ms for n = 500.

d41d8c, 40 lines

```
const ll INF = 1e18;
pair<ll, vector<int>> hungarian(const vector<vector<ll>>& A) {
  int n = (int)A.size()-1;
  int m = (int)A[0].size()-1;
  vector<ll> u(n+1), v(m+1), p(m+1), way(m+1);
  for (int i=1; i<=n; ++i) {
    p[0] = i;
    int j0 = 0;
    vector<ll> minv (m+1, INF);
    vector<char> used (m+1, false);
    do {
      used[j0] = true;
      int i0 = p[j0], j1;
      ll delta = INF;
      for (int j=1; j<=m; ++j) {
        if (!used[j]) {
          ll cur = A[i0][j]-u[i0]-v[j];
          if (cur < minv[j])
            minv[j] = cur, way[j] = j0;
          if (minv[j] < delta)
            delta = minv[j], j1 = j;
        }
      }
      for (int j=0; j<=m; ++j)
        if (used[j])
          u[p[j]] += delta, v[j] -= delta;
        else
          minv[j] -= delta;
      j0 = j1;
    } while (p[j0] != 0);
    do {
      int j1 = way[j0];
      p[j0] = p[j1];
      j0 = j1;
    } while (j0);
  }
  vector<int> match(n+1);
  for (int i=1; i<=m; ++i) match[p[i]] = i;
  return { -v[0], match };
}
```

GlobalMinCut.h

Description: Undirected graph with adj matrix. No edge means $adj[i][j] = 0$. 0-based index, and expect $N \times N$ adj matrix.
Time: $\mathcal{O}\left(V^3\right)$, $\sum V^3 = 5.5 \times 10^8$ in 640ms.

d41d8c, 24 lines

```
const int INF = 1e9;
int getMinCut(vector<vector<int>> &adj) {
  int n = adj.size();
  vector<int> used(n);
  int ret = INF;
  for (int ph=n-1; ph>=0; --ph) {
    vector<int> w = adj[0], added = used;
    int prev, k = 0;
    for (int i=0; i<ph; ++i) {
      prev = k;
      k = -1;
      for (int j = 1; j < n; j++) {
        if (!added[j] && (k == -1 || w[j] > w[k])) k = j;
      }
      if (i+1 == ph) break;
      for (int j = 0; j < n; j++) w[j] += adj[k][j];
      added[k] = 1;
    }
    for (int i=0; i<n; ++i) adj[i][prev] = (adj[prev][i] += adj
        [k][i]);
    used[k] = 1;
    ret = min(ret, w[k]);
  }
  return ret;
}
```

GomoryHu.h

Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.
Usage: 0-base index. `GomoryHuTree t; auto ret = t.solve(n, edges);` 0 is root, `ret[i]` for $i > 0$ contains (cost, par)
Time: $\mathcal{O}(V)$ Flow Computations, $V = 3000, E = 4500$ and special graph that flow always terminate in $\mathcal{O}(3(V+E))$ time in 4036ms.

d41d8c, 29 lines

```
struct Edge { int s, e, x; };
const int MAX_N = 500 + 1;
bool vis[MAX_N];

struct GomoryHuTree {
  vector<pii> solve(int n, const vector<Edge>& edges) { // i -
      j cut : i - j minimum edge cost. 0 based.
    vector<pii> ret(n); // if i > 0, stores pair(cost, parent)
    for(int i=1; i<n; i++){
      Dinic g(n);
      for (auto[s, e, x]: edges) g.add_edge(s, e, x, x);
      ret[i].first = g.max_flow(i, ret[i].second);

      memset(vis, 0, sizeof(vis));
      function<void(int)> dfs = [&](int x) {
        if (vis[x]) return;
        vis[x] = 1;
        for (auto& i: g.adj[x]) {
          if (i.cap - i.flow > 0) dfs(i.a);
        }
      };

      dfs(i);
      for (int j=i+1; j<n; j++) {
        if (ret[j].second == ret[i].second && vis[j]) ret[j].
            second = i;
      }
    }
    return ret;
  }
};
```

3.3 Matching

3.3.1 Random notes on matching (and bipartite)

In general graph, complement of independent set is vertex cover, and reverse holds too.
In bipartite graph, cardinality of minimum vertex cover is equal to card of maximum matching (konig).
In poset (DAG), card of maximum anti chain is equal to minimum path cover (dilworth).
Poset is DAG which satisfy i-¿j and j-¿k edge means i-¿k (transitivity).

hopcroftKarp.h

Description: It contains several application of bipartite matching.
Usage: Both left and right side of node number starts with 0. `HopcraftKarp(n, m); g.add_edge(s, e);`
Time: $\mathcal{O}\left(E\sqrt{V}\right)$, min path cover $V = 10^4, E = 10^5$ in 20ms.

d41d8c, 92 lines

```
struct HopcroftKarp{
  int n, m;
  vector<vector<int>> g;
  vector<int> dst, le, ri;
  vector<char> visit, track;
HopcroftKarp(int n, int m) : n(n), m(m), g(n), dst(n), le(n,
    -1), ri(m, -1), visit(n), track(n+m) {}

  void add_edge(int s, int e){ g[s].push_back(e); }
  bool bfs(){
    bool res = false; queue<int> que;
    fill(dst.begin(), dst.end(), 0);
    for(int i=0; i<n; i++)if(le[i] == -1)que.push(i),dst[i]=1;
    while(!que.empty()){
      int v = que.front(); que.pop();
      for(auto i : g[v]){
        if(ri[i] == -1) res = true;
        else if(!dst[ri[i]])dst[ri[i]]=dst[v]+1,que.push(ri[i])
            ;
      }
    }
    return res;
  }
  bool dfs(int v){
    if(visit[v]) return false; visit[v] = 1;
    for(auto i : g[v]){
      if(ri[i] == -1 || !visit[ri[i]] && dst[ri[i]] == dst[v] +
          1 && dfs(ri[i])){
        le[v] = i; ri[i] = v; return true;
      }
    }
    return false;
  }
  int maximum_matching(){
    int res = 0; fill(le.begin(), le.end(), -1); fill(ri.begin
        (), ri.end(), -1);
    while(bfs()){
      fill(visit.begin(), visit.end(), 0);
      for(int i=0; i<n; i++) if(le[i] == -1) res += dfs(i);
```

```
    }
    return res;
  }
  vector<pair<int,int>> maximum_matching_edges(){
    int matching = maximum_matching();
    vector<pair<int,int>> edges; edges.reserve(matching);
    for(int i=0; i<n; i++) if(le[i] != -1) edges.emplace_back(i
        , le[i]);
    return edges;
  }
  void dfs_track(int v){
    if(track[v]) return; track[v] = 1;
    for(auto i : g[v]) track[n+i] = 1, dfs_track(ri[i]);
  }
  tuple<vector<int>, vector<int>, int> minimum_vertex_cover(){
    int matching = maximum_matching(); vector<int> lv, rv;
    fill(track.begin(), track.end(), 0);
    for(int i=0; i<n; i++) if(le[i] == -1) dfs_track(i);
    for(int i=0; i<n; i++) if(!track[i]) lv.push_back(i);
    for(int i=0; i<m; i++) if(track[n+i]) rv.push_back(i);
    return {lv, rv, lv.size() + rv.size()}; // s(lv)+s(rv)=mat
  }
  tuple<vector<int>, vector<int>, int> maximum_independent_set
      (){
    auto [a,b,matching] = minimum_vertex_cover();
    vector<int> lv, rv; lv.reserve(n-a.size()); rv.reserve(m-b.
        size());
    for(int i=0, j=0; i<n; i++){
      while(j < a.size() && a[j] < i) j++;
      if(j == a.size() || a[j] != i) lv.push_back(i);
    }
    for(int i=0, j=0; i<m; i++){
      while(j < b.size() && b[j] < i) j++;
      if(j == b.size() || b[j] != i) rv.push_back(i);
    } // s(lv)+s(rv)=n+m-mat
    return {lv, rv, lv.size() + rv.size()};
  }
  vector<vector<int>> minimum_path_cover(){ // n == m
    int matching = maximum_matching();
    vector<vector<int>> res; res.reserve(n - matching);
    fill(track.begin(), track.end(), 0);
    auto get_path = [&](int v) -> vector<int> {
      vector<int> path{v}; // ri[v] == -1
      while(le[v] != -1) path.push_back(v=le[v]);
      return path;
    };
    for(int i=0; i<n; i++) if(!track[n+i] && ri[i] == -1) res.
        push_back(get_path(i));
    return res; // sz(res) = n-mat
  }
  vector<int> maximum_anti_chain(){ // n = m
    auto [a,b,matching] = minimum_vertex_cover();
    vector<int> res; res.reserve(n - a.size() - b.size());
    for(int i=0, j=0, k=0; i<n; i++){
      while(j < a.size() && a[j] < i) j++;
      while(k < b.size() && b[k] < i) k++;
      if((j == a.size() || a[j] != i) && (k == b.size() || b[k]
          != i)) res.push_back(i);
    }
    return res; // sz(res) = n-mat
  }
};
```

GeneralMatching.h

Description: Matching for general graphs.
Usage: `1-base index. match[] has real matching (maybe). GeneralMatching g(n); g.add_edge(a, b); int ret = g.run(void);`
Time: $\mathcal{O}(N^3)$, $N = 500$ in 20ms.
d41d8c, 93 lines

```
const int MAX_N = 500 + 1;

struct GeneralMatching {
  int n, cnt;
  int match[MAX_N], par[MAX_N], chk[MAX_N], prv[MAX_N], vis[
      MAX_N];
  vector<int> g[MAX_N];
GeneralMatching(int n): n(n) {
  // init
  cnt = 0;
  for (int i=0; i<=n; ++i) g[i].clear();
  memset(match, 0, sizeof match);
  memset(vis, 0, sizeof vis);
  memset(prv, 0, sizeof prv);
}

  int find(int x) { return x == par[x] ? x : par[x] = find(par[
      x]); }

  int lca(int u, int v) {
    for (cnt++; vis[u] != cnt; swap(u, v)) {
      if (u) vis[u] = cnt, u = find(prv[match[u]]);
    }
    return u;
  }

  void add_edge(int u, int v) {
    g[u].push_back(v);
    g[v].push_back(u);
  }

  void blossom(int u, int v, int rt, queue<int> &q) {
    for (; find(u) != rt; u = prv[v]) {
      prv[u] = v;
      par[u] = par[v = match[u]] = rt;
      if (chk[v] & 1) q.push(v), chk[v] = 2;
    }
  }

  bool augment(int u) {
    iota(par, par + MAX_N, 0);
    memset(chk, 0, sizeof chk);
    queue<int> q;
    q.push(u);
    chk[u] = 2;

    while (!q.empty()) {
      u = q.front();
      q.pop();
      for (auto v : g[u]) {
        if (chk[v] == 0) {
          prv[v] = u;
          chk[v] = 1;
          q.push(match[v]);
          chk[match[v]] = 2;
          if (!match[v]) {
            for (; u; v = u) {
              u = match[prv[v]];
              match[match[v] = prv[v]] = v;
            }
            return true;
          }
        } else if (chk[v] == 2) {
          int l = lca(u, v);
          blossom(u, v, l, q);
          blossom(v, u, l, q);
        }
      }
    }
    return false;
  }

  int run() {
    int ret = 0;
    vector<int> tmp(n-1); // not necessary, just for constant
        optimization
    iota(tmp.begin(), tmp.end(), 0);
    shuffle(tmp.begin(), tmp.end(), mt19937(0x1557));
    for (auto x: tmp) {
      if (!match[x]) {
        for (auto y: g[x]) {
          if (!match[y]) {
            match[x] = y;
            match[y] = x;
            ret++;
            break;
          }
        }
      }
    }
    for (int i=1; i<=n; i++) {
      if (!match[i]) ret += augment(i);
    }
    return ret;
  }
};
```

GeneralWeightedMatching.h

Description: Given a weighted undirected graph, return maximum matching.
Usage: `1-base index. init(n); add_edge(a, b, w); (tot_weight, n_matches) = _solve(void); Note that get_lca function have a static variable.`
Time: $\mathcal{O}(N^3)$, $N = 500$ in 317ms at yosupo.
d41d8c, 228 lines

```
static const int INF = INT_MAX;
static const int N = 500 + 1;

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);
  }
}
```

3.4 DFS algorithms

2sat.h

Description: Every variable x is encoded to 2i, !x is 2i+1. n of TwoSAT means number of variables.

Usage: `TwoSat g(number of vars);`
`g.addCNF(x, y); // x or y`
`g.atMostOne({ a, b, ... });`
`auto ret = g.solve(void); if impossible empty`

Time: $\mathcal{O}(V+E)$, note that sort in atMostOne function. 10^5 simple cnf clauses 56ms.

d41d8c, 91 lines

```
struct TwoSAT {
  struct SCC {
    int n;
    vector<bool> chk;
    vector<vector<int>> E, F;
    SCC() {}

    void dfs(int x, vector<vector<int>> &E, vector<int> &st) {
      if(chk[x]) return;
      chk[x] = true;
      for(auto i : E[x]) dfs(i, E, st);
      st.push_back(x);
    }

    void init(vector<vector<int>> &E) {
      n = E.size();
      this->E = E;
      F.resize(n);
      chk.resize(n, false);
      for(int i = 0; i < n; i++)
        for(auto j : E[i]) F[j].push_back(i);
    }

    vector<vector<int>> getSCC() {
      vector<int> st;
      fill(chk.begin(), chk.end(), false);
      for(int i = 0; i < n; i++) dfs(i, E, st);
      reverse(st.begin(), st.end());
      fill(chk.begin(), chk.end(), false);
      vector<vector<int>> scc;
      for(int i = 0; i < n; i++) {
        if(chk[st[i]]) continue;
        vector<int> T;
        dfs(st[i], F, T);
        scc.push_back(T);
      }
      return scc;
    }
  };

  int n;
  vector<vector<int>> adj;
  TwoSAT(int n): n(n) {
  adj.resize(2*n);
}

  int new_node() {
    adj.push_back(vector<int>());
    adj.push_back(vector<int>());
    return n++;
  }
  void add_edge(int a, int b) {
    adj[a].push_back(b);
  }
  void add_cnf(int a, int b) {
    add_edge(a^1, b);
    add_edge(b^1, a);
  }
  // arr elements need to be unique
  // Add n dummy variable, 3n-2 edges
  // yi = x1 | x2 | ... | xi, xi->yi, yi->y(i+1), yi->!x(i+1)
  void at_most_one(vector<int> arr) {
    sort(arr.begin(), arr.end());
    assert(unique(arr.begin(), arr.end()) == arr.end());
    for (int i=0; i<arr.size(); ++i) {
      int now = new_node();
      add_cnf(arr[i]^1, 2*now);
      if (i == 0) continue;
      add_cnf(2*(now-1)+1, 2*now);
      add_cnf(2*(now-1)+1, arr[i]^1);
    }
  }

  vector<int> solve() {
    SCC g;
    g.init(adj);
    auto scc = g.getSCC();

    vector<int> rev(2*n, -1);
    for (int i=0; i<scc.size(); ++i) {
      for (int x: scc[i]) rev[x] = i;
    }
    for (int i=0; i<n; ++i) {
      if (rev[2*i] == rev[2*i+1]) return vector<int>();
    }

    vector<int> ret(n);
    for (int i=0; i<n; ++i) ret[i] = (rev[2*i] > rev[2*i+1]);
    return ret;
  }
};
```

3.5 Coloring

EdgeColoring.h

Description: Given a simple, undirected graph with max degree D, computes a $(D+1)$-coloring of the edges such that no neighboring edges share a color.

Usage: `1-base index. Vizing g; g.clear(V); g.solve(edges, V); answer saved in G.`

Time: $\mathcal{O}(VE)$, $\sum VE = 1.1\times 10^6$ in 24ms.

d41d8c, 60 lines

```
const int MAX_N = 444 + 1;
struct Vizing { // returns edge coloring in adjacent matrix G.
    1 - based
  int C[MAX_N][MAX_N], G[MAX_N][MAX_N];

  void clear(int n){
    for (int i=0; i<=n; i++){
      for (int j=0; j<=n; j++) C[i][j] = G[i][j] = 0;
    }
  }

  void solve(vector<pii> &E, int n){
    int X[MAX_N] = {}, a;

    auto update = [&](int u) {
      for (X[u] = 1; C[u][X[u]]; X[u]++);
    };

    auto color = [&](int u, int v, int c) {
      int p = G[u][v];
      G[u][v] = G[v][u] = c;
      C[u][c] = v;
      C[v][c] = u;
      C[u][p] = C[v][p] = 0;
      if (p) X[u] = X[v] = p;
      else update(u), update(v);
      return p;
    };

    auto flip = [&](int u, int c1, int c2){
      int p = C[u][c1]; swap(C[u][c1], C[u][c2]);
      if (p) G[u][p] = G[p][u] = c2;
      if (!C[u][c1]) X[u] = c1;
      if (!C[u][c2]) X[u] = c2;
      return p;
    };

    for (int i=1; i <= n; i++) X[i] = 1;
    for (int t=0; t<E.size(); ++t) {
      auto[u, v0] = E[t];
      int v = v0, c0 = X[u], c=c0, d;
      vector<pii> L;
      int vst[MAX_N] = {};
      while (!G[u][v0]) {
        L.emplace_back(v, d = X[v]);
        if(!C[v][c]) for(a = (int)L.size()-1; a >= 0; a--) c =
            color(u, L[a].first, c);
        else if (!C[u][d]) for(a=(int)L.size()-1;a>=0;a--)
            color(u,L[a].first,L[a].second);
        else if (vst[d]) break;
        else vst[d] = 1, v = C[u][d];
      }

      if(!G[u][v0]) {
        for (; v; v = flip(v, c, d), swap(c, d));
        if(C[u][c0]){
          for(a = (int)L.size()-2; a >= 0 && L[a].second != c;
              a--);
          for(; a >= 0; a--) color(u, L[a].first, L[a].second);
        } else t--;
      }
    }
  }
};
```

3.6 Heuristics

3.7 Trees

DirectedMST.h

Description: Directed MST for given root node. If no MST exists, returns -1.

Usage: `0-base index. Vertex is 0 to n-1. typedef ll cost_t.`

Time: $\mathcal{O}(E\log V)$, $V=E=2\times 10^5$ in 90ms at yosupo.

d41d8c, 87 lines

```
struct Edge{
  int s, e; cost_t x;
  Edge() = default;
Edge(int s, int e, cost_t x) : s(s), e(e), x(x) {}
  bool operator < (const Edge &t) const { return x < t.x; }
};
struct UnionFind{
  vector<int> P, S;
  vector<pair<int, int>> stk;
UnionFind(int n) : P(n), S(n, 1) { iota(P.begin(), P.end(), 0);
    }
  int find(int v) const { return v == P[v] ? v : find(P[v]); }
  int time() const { return stk.size(); }
  void rollback(int t){
    while(stk.size() > t){
      auto [u,v] = stk.back(); stk.pop_back();
      P[u] = u; S[v] -= S[u];
```

```
    }
  }
  bool merge(int u, int v){
    u = find(u); v = find(v);
    if(u == v) return false;
    if(S[u] > S[v]) swap(u, v);
    stk.emplace_back(u, v);
    S[v] += S[u]; P[u] = v;
    return true;
  }
};
struct Node{
  Edge key;
  Node *l, *r;
  cost_t lz;
Node() : Node(Edge()) {}
Node(const Edge &edge) : key(edge), l(nullptr), r(nullptr), lz
    (0) {}
  void push(){
    key.x += lz;
    if(l) l->lz += lz;
    if(r) r->lz += lz;
    lz = 0;
  }
  Edge top(){ push(); return key; }
};
Node* merge(Node *a, Node *b){
  if(!a || !b) return a ? a : b;
  a->push(); b->push();
  if(b->key < a->key) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node* &a){ a->push(); a = merge(a->l, a->r); }

// 0-based
pair<cost_t, vector<int>> DirectMST(int n, int rt, vector<Edge>
    &edges){
  vector<Node*> heap(n);
  UnionFind uf(n);
  for(const auto &i : edges) heap[i.e] = merge(heap[i.e], new
      Node(i));
  cost_t res = 0;
  vector<int> seen(n, -1), path(n), par(n);
  seen[rt] = rt;
  vector<Edge> Q(n), in(n, {-1,-1, 0}), comp;
  deque<tuple<int, int, vector<Edge>>> cyc;
  for(int s=0; s<n; s++){
    int u = s, qi = 0, w;
    while(seen[u] < 0){
      if(!heap[u]) return {-1, {}};
      Edge e = heap[u]->top();
      heap[u]->lz -= e.x; pop(heap[u]);
      Q[qi] = e; path[qi++] = u; seen[u] = s;
      res += e.x; u = uf.find(e.s);
      if(seen[u] == s){ // found cycle, contract
        Node* nd = 0;
        int end = qi, time = uf.time();
        do nd = merge(nd, heap[w = path[--qi]]); while(uf.merge
            (u, w));
        u = uf.find(u); heap[u] = nd; seen[u] = -1;
        cyc.emplace_front(u, time, vector<Edge>{&Q[qi], &Q[end]
            });
      }
    }
    for(int i=0; i<qi; i++) in[uf.find(Q[i].e)] = Q[i];
  }
  for(auto& [u,t,comp] : cyc){
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.e)] = e;
    in[uf.find(inEdge.e)] = inEdge;
  }
  for(int i=0; i<n; i++) par[i] = in[i].s;
  return {res, par};
}
```

ManhattanMST.h

Description: Given 2d points, find MST with taxi distance.
Usage: `0-base index internally. taxiMST(pts); Returns mst's tree edges with (length, a, b); Note that union-find need return value.`
Time: $\mathcal{O}(N \log N)$, $N = 2 \times 10^5$ in 363ms at yosupo.

d41d8c, 26 lines

```
struct point { ll x, y; };

vector<tuple<ll, int, int>> taxiMST(vector<point> a){
  int n = a.size();
  vector<int> ind(n);
  iota(ind.begin(), ind.end(), 0);
  vector<tuple<ll, int, int>> edge;
  for(int k=0; k<4; k++){
    sort(ind.begin(), ind.end(), [&](int i,int j){return a[i].x
        -a[j].x < a[j].y-a[i].y;});
    map<ll, int> mp;
    for(auto i: ind){
      for(auto it=mp.lower_bound(-a[i].y); it!=mp.end(); it=mp.
          erase(it)){
        int j = it->second; point d = {a[i].x-a[j].x, a[i].y-a[
            j].y};
        if(d.y > d.x) break;
        edge.push_back({d.x + d.y, i, j});
      }
      mp.insert({-a[i].y, i});
    }
    for(auto &p: a) if(k & 1) p.x = -p.x; else swap(p.x, p.y);
  }
  sort(edge.begin(), edge.end());
  DisjointSet dsu(n);
  vector<tuple<ll, int, int>> res;
  for(auto [x, i, j]: edge) if(dsu.merge(i, j)) res.push_back({
      x, i, j});
  return res;
}
```

DominatorTree.h

d41d8c, 46 lines

```
vector<int> E[MAXN], RE[MAXN], rdom[MAXN];

int S[MAXN], RS[MAXN], cs;
int par[MAXN], val[MAXN], sdom[MAXN], rp[MAXN], dom[MAXN];

void clear(int n) {
  cs = 0;
  for(int i=0;i<=n;i++) {
    par[i] = val[i] = sdom[i] = rp[i] = dom[i] = S[i] = RS[i] =
        0;
    E[i].clear(); RE[i].clear(); rdom[i].clear();
  }
}
void add_edge(int x, int y) { E[x].push_back(y); }
void Union(int x, int y) { par[x] = y; }
int Find(int x, int c = 0) {
  if(par[x] == x) return c ? -1 : x;
  int p = Find(par[x], 1);
  if(p == -1) return c ? par[x] : val[x];
  if(sdom[val[x]] > sdom[val[par[x]]]) val[x] = val[par[x]];
  par[x] = p;
  return c ? p : val[x];
}
void dfs(int x) {
  RS[ S[x] = ++cs ] = x;
  par[cs] = sdom[cs] = val[cs] = cs;
  for(int e : E[x]) {
    if(S[e] == 0) dfs(e), rp[S[e]] = S[x];
    RE[S[e]].push_back(S[x]);
  }
}
int solve(int s, int *up) { // Calculate idoms
  dfs(s);
  for(int i=cs;i;i--) {
    for(int e : RE[i]) sdom[i] = min(sdom[i], sdom[Find(e)]);
    if(i > 1) rdom[sdom[i]].push_back(i);
    for(int e : rdom[i]) {
      int p = Find(e);
      if(sdom[p] == i) dom[e] = i;
      else dom[e] = p;
    }
    if(i > 1) Union(i, rp[i]);
  }
  for(int i=2;i<=cs;i++) if(sdom[i] != dom[i]) dom[i] = dom[dom
      [i]];
  for(int i=2;i<=cs;i++) up[RS[i]] = RS[dom[i]];
  return cs;
}
```

3.8 Math

3.8.1 Number of Spanning Trees

Create an $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]--`, `mat[b][b]++` (and `mat[b][a]--`, `mat[a][a]++` if G is undirected). Remove the ith row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

3.8.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k).$$

Mathematics (4)

FFT.h

d41d8c, 51 lines

```
// multiply: input format - [x^0 coeff, x^1 coeff, ...], [same
    ], [anything]
typedef complex<double> base;

const double PI = acos(-1);
void fft(vector<base>& a, bool inv) {
  int n = a.size();
  for (int dest=1, src=0; dest<n; ++dest) {
    int bit = n / 2;
    while (src >= bit) {
      src -= bit;
      bit /= 2;
    }
    src += bit;
```

```
    if (dest < src) { swap(a[dest], a[src]); }
  }
  for (int len=2; len <= n; len *= 2) {
    double ang = 2 * PI / len * (inv ? -1 : 1);
    base unity(cos(ang), sin(ang));
    for (int i=0; i<n; i+=len) {
      base w(1, 0);
      for (int j=0; j<len/2; ++j) {
        base u = a[i+j], v = a[i+j+len/2] * w;
        a[i+j] = u+v;
        a[i+j+len/2] = u-v;
        w *= unity;
      }
    }
  }
  if (inv) {
    for (int i=0; i<n; ++i) { a[i] /= n; }
  }
}

void multiply(const vector<int>& a, const vector<int>& b,
    vector<int>& result) {
  int n = 2;
  while (n < a.size() + b.size()) { n *= 2; }

  vector<base> p(a.begin(), a.end());
  p.resize(n);
  for (int i=0; i<b.size(); ++i) { p[i] += base(0, b[i]); }
  fft(p, false);

  result.resize(n);
  for (int i=0; i<=n/2; ++i) {
    base u = p[i], v = p[(n-i) % n];
    p[i] = (u * u - conj(v) * conj(v)) * base(0, -0.25);
    p[(n-i) % n] = (v * v - conj(u) * conj(u)) * base(0, -0.25)
        ;
  }
  fft(p, true);
  for (int i=0; i<n; ++i) { result[i] = (int)round(p[i].real())
      ; }
}
```

NTT.h

d41d8c, 48 lines

```
// Caution! prim needs to be initialized with prim = power(
    primitive root of MOD, A) before use;
// multiply: input format - [x^0 coeff, x^1 coeff, ...], [same
    ], [anything]
const int MOD = 998244353;
const int A = 119, B = 23;
ll prim;
ll power(ll a, int pow) {
  ll ret = 1;
  while (pow > 0) {
    if (pow & 1) ret = ret * a % MOD;
    a = a * a % MOD;
    pow /= 2;
  }
  return ret;
}
void fft(vector<ll>& a, bool inv) {
  int n = a.size();
  for (int dest=1, src=0; dest<n; ++dest) {
    int bit = n / 2;
    while (src >= bit) {
      src -= bit;
      bit /= 2;
    }
    src += bit;
    if (dest < src) { swap(a[dest], a[src]); }
  }
  for (int len=2; len <= n; len *= 2) {
    ll unity = power(inv ? power(prim, MOD-2) : prim, (1 << B)
        / len);
    for (int i=0; i<n; i+=len) {
      ll w = 1;
      for (int j=0; j<len/2; ++j) {
        ll u = a[i+j], v = a[i+j+len/2] * w % MOD;
        a[i+j] = u+v;
        if (a[i+j] >= MOD) a[i+j] -= MOD;
        a[i+j+len/2] = u-v;
        if (a[i+j+len/2] < 0) a[i+j+len/2] += MOD;
        w = w * unity % MOD;
      }
    }
  }
  if (inv) {
    ll tmp = power(n, MOD-2);
    for (int i=0; i<n; ++i) a[i] = a[i] * tmp % MOD;
  }
}
void conv(const vector<ll>& a, const vector<ll>& b, vector<ll>&
    result) {
  result.resize(a.size(), 0);
  for (int i=0; i<result.size(); ++i) result[i] = (result[i] +
      a[i] * b[i]) % MOD;
}
```

BerlekampKitamasa.h

d41d8c, 294 lines

```
vector<int> berlekamp_massey(vector<int> x){
  vector<int> ls, cur;
  int lf, ld;
  for(int i=0; i<x.size(); i++){
    ll t = 0;
    for(int j=0; j<cur.size(); j++){
      t = (t + 1ll * x[i-j-1] * cur[j]) % mod;
    }
    if((t - x[i]) % mod == 0) continue;
    if(cur.empty()){
      cur.resize(i+1);
      lf = i;
      ld = (t - x[i]) % mod;
      continue;
    }
    ll k = -(x[i] - t) * ipow(ld, mod - 2) % mod;
    vector<int> c(i-lf-1);
    c.push_back(k);
    for(auto &j : ls) c.push_back(-j * k % mod);
    if(c.size() < cur.size()) c.resize(cur.size());
    for(int j=0; j<cur.size(); j++){
      c[j] = (c[j] + cur[j]) % mod;
    }
    if(i-lf+(int)ls.size()>=(int)cur.size()){
      tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
    }
    cur = c;
  }
  for(auto &i : cur) i = (i % mod + mod) % mod;
  return cur;
}

struct elem{int x, y, v;}; // A_(x, y) <- v, 0-based. no
    duplicate please..
vector<int> get_min_poly(int n, vector<elem> M){
  // smallest poly P such that A^i = sum_{j < i} {A^j \times
      P_j}
  vector<int> rnd1, rnd2;
  mt19937 rng(0x14004);
  auto randint = [&rng](int lb, int ub){
    return uniform_int_distribution<int>(lb, ub)(rng);
  };
  for(int i=0; i<n; i++){
    rnd1.push_back(randint(1, mod - 1));
    rnd2.push_back(randint(1, mod - 1));
  }
  vector<int> gobs;
  for(int i=0; i<2*n+2; i++){
    int tmp = 0;
    for(int j=0; j<n; j++){
      tmp += 1ll * rnd2[j] * rnd1[j] % mod;
      if(tmp >= mod) tmp -= mod;
    }
    gobs.push_back(tmp);
    vector<int> nxt(n);
    for(auto &i : M){
      nxt[i.x] += 1ll * i.v * rnd1[i.y] % mod;
      if(nxt[i.x] >= mod) nxt[i.x] -= mod;
    }
    rnd1 = nxt;
  }
  auto sol = berlekamp_massey(gobs);
  reverse(sol.begin(), sol.end());
  return sol;
}
lint det(int n, vector<elem> M){
  vector<int> rnd;
  mt19937 rng(0x14004);
  auto randint = [&rng](int lb, int ub){
    return uniform_int_distribution<int>(lb, ub)(rng);
  };
  for(int i=0; i<n; i++) rnd.push_back(randint(1, mod - 1));
  for(auto &i : M){
    i.v = 1ll * i.v * rnd[i.y] % mod;
  }
  auto sol = get_min_poly(n, M)[0];
  if(n % 2 == 0) sol = mod - sol;
  for(auto &i : rnd) sol = 1ll * sol * ipow(i, mod - 2) % mod;
  return sol;
}

using uint = unsigned;
using ll = long long;
using ull = unsigned long long;

template<int M>
struct MINT{
  int v;
  MINT() : v(0) {}
  MINT(ll val){
    v = (-M <= val && val < M) ? val : val % M;
    if(v < 0) v += M;
  }

  friend istream& operator >> (istream &is, MINT &a) { ll t; is
      >> t; a = MINT(t); return is; }
  friend ostream& operator << (ostream &os, const MINT &a) {
      return os << a.v; }
  friend bool operator == (const MINT &a, const MINT &b) {
      return a.v == b.v; }
  friend bool operator != (const MINT &a, const MINT &b) {
      return a.v != b.v; }
  friend MINT pw(MINT a, ll b){
    MINT ret= 1;
    while(b){
      if(b & 1) ret *= a;
      b >>= 1; a *= a;
```

```
    }
    return ret;
  }
  friend MINT inv(const MINT a) { return pw(a, M-2); }
  MINT operator - () const { return MINT(-v); }
  MINT& operator += (const MINT m) { if((v += m.v) >= M) v -= M
      ; return *this; }
  MINT& operator -= (const MINT m) { if((v -= m.v) < 0) v += M;
       return *this; }
  MINT& operator *= (const MINT m) { v = (ll)v*m.v%M; return *
      this; }
  MINT& operator /= (const MINT m) { *this *= inv(m); return *
      this; }
  friend MINT operator + (MINT a, MINT b) { a += b; return a; }
  friend MINT operator - (MINT a, MINT b) { a -= b; return a; }
  friend MINT operator * (MINT a, MINT b) { a *= b; return a; }
  friend MINT operator / (MINT a, MINT b) { a /= b; return a; }
  operator int32_t() const { return v; }
  operator int64_t() const { return v; }
};

namespace fft{
  template<int W, int M>
  static void NTT(vector<MINT<M>> &f, bool inv_fft = false){
    using T = MINT<M>;
    int N = f.size();
    vector<T> root(N >> 1);
    for(int i=1, j=0; i<N; i++){
      int bit = N >> 1;
      while(j >= bit) j -= bit, bit >>= 1;
      j += bit;
      if(i < j) swap(f[i], f[j]);
    }
    T ang = pw(T(W), (M-1)/N); if(inv_fft) ang = inv(ang);
    root[0] = 1; for(int i=1; i<N>>1; i++) root[i] = root[i-1]
        * ang;
    for(int i=2; i<=N; i<<=1){
      int step = N / i;
      for(int j=0; j<N; j+=i){
        for(int k=0; k<i/2; k++){
          T u = f[j+k], v = f[j+k+(i>>1)] * root[k*step];
          f[j+k] = u + v;
          f[j+k+(i>>1)] = u - v;
        }
      }
    }
    if(inv_fft){
      T rev = inv(T(N));
      for(int i=0; i<N; i++) f[i] *= rev;
    }
  }
  template<int W, int M>
  vector<MINT<M>> multiply_ntt(vector<MINT<M>> a, vector<MINT<M
      >> b){
    int N = 2; while(N < a.size() + b.size()) N <<= 1;
    a.resize(N); b.resize(N);
    NTT<W, M>(a); NTT<W, M>(b);
    for(int i=0; i<N; i++) a[i] *= b[i];
    NTT<W, M>(a, true);
    return a;
  }
}

template<int W, int M>
struct PolyMod{
  using T = MINT<M>;
  vector<T> a;

  // constructor
  PolyMod(){}
  PolyMod(T a0) : a(1, a0) { normalize(); }
  PolyMod(const vector<T> a) : a(a) { normalize(); }

  // method from vector<T>
  int size() const { return a.size(); }
  int deg() const { return a.size() - 1; }
  void normalize(){ while(a.size() && a.back() == T(0)) a.
      pop_back(); }
  T operator [] (int idx) const { return a[idx]; }
  typename vector<T>::const_iterator begin() const { return a.
      begin(); }
  typename vector<T>::const_iterator end() const { return a.end
      (); }
  void push_back(const T val) { a.push_back(val); }
  void pop_back() { a.pop_back(); }

  // basic manipulation
  PolyMod reversed() const {
    vector<T> b = a;
    reverse(b.begin(), b.end());
    return b;
  }
  PolyMod trim(int n) const {
    return vector<T>(a.begin(), a.begin() + min(n, size()));
  }
  PolyMod inv(int n){
    PolyMod q(T(1) / a[0]);
    for(int i=1; i<n; i<<=1){
      PolyMod p = PolyMod(2) - q * trim(i * 2);
      q = (p * q).trim(i * 2);
    }
    return q.trim(n);
  }

  // operation with scala value
  PolyMod operator *= (const T x){
    for(auto &i : a) i *= x;
    normalize();
    return *this;
  }
  PolyMod operator /= (const T x){
    return *this *= (T(1) / T(x));
  }

  // operation with poly
  PolyMod operator += (const PolyMod &b){
    a.resize(max(size(), b.size()));
    for(int i=0; i<b.size(); i++) a[i] += b.a[i];
    normalize();
    return *this;
  }
  PolyMod operator -= (const PolyMod &b){
    a.resize(max(size(), b.size()));
    for(int i=0; i<b.size(); i++) a[i] -= b.a[i];
    normalize();
    return *this;
  }
  PolyMod operator *= (const PolyMod &b){
    *this = fft::multiply_ntt<W, M>(a, b.a);
    normalize();
    return *this;
  }
  PolyMod operator /= (const PolyMod &b){
    if(deg() < b.deg()) return *this = PolyMod();
    int sz = deg() - b.deg() + 1;
    PolyMod ra = reversed().trim(sz), rb = b.reversed().trim(sz
        ).inv(sz);
    *this = (ra * rb).trim(sz);
    for(int i=sz-size(); i; i--) push_back(T(0));
    reverse(all(a));
    normalize();
    return *this;
  }
  PolyMod operator %= (const PolyMod &b){
    if(deg() < b.deg()) return *this;
    PolyMod tmp = *this; tmp /= b; tmp *= b;
    *this -= tmp;
    normalize();
    return *this;
  }

  // operator
  PolyMod operator * (const T x) const { return PolyMod(*this)
      *= x; }
  PolyMod operator / (const T x) const { return PolyMod(*this)
      /= x; }
  PolyMod operator + (const PolyMod &b) const { return PolyMod
      (*this) += b; }
  PolyMod operator - (const PolyMod &b) const { return PolyMod
      (*this) -= b; }
  PolyMod operator * (const PolyMod &b) const { return PolyMod
      (*this) *= b; }
  PolyMod operator / (const PolyMod &b) const { return PolyMod
      (*this) /= b; }
  PolyMod operator % (const PolyMod &b) const { return PolyMod
      (*this) %= b; }
};

constexpr int W = 3, MOD = 104857601;
using mint = MINT<MOD>;
using poly = PolyMod<W, MOD>;

mint kitamasa(poly c, poly a, ll n){
  poly x = vector<mint>{0,1};
  poly res = vector<mint>{1};
  while (n > 0) {
    if (n & 1) res = res * x % c;
    x = x * x % c;
    n /= 2;
  }

  mint ret(0);
  for (int i=0; i<a.size(); ++i) ret += res[i] * a[i];
  return ret;
}

int main(){
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  int k;
  ll n;
  cin >> k >> n;
  vector<mint> A(k), C(k);
  for (int i=0; i<k; ++i) {
    int x;
    cin >> x;
    A[i] = mint(x);
  }
  for (int i=0; i<k; ++i) {
    int x;
    cin >> x;
    C[i] = mint(x);
  }

  reverse(C.begin(), C.end());
  for (int i=0; i<C.size(); ++i) C[i] = -C[i];
```

```
  C.push_back(1);
  cout << kitamasa(C, A, n-1);
}
```

Simplex.h

Description: Solve $Ax <= b$, max $c^T x$. Maximal value store in v, answer backtracking via sol[i]. 1-base index.
Time: exponential. fast $\mathcal{O}\left(MN^2\right)$ in experiment. dependent on the modeling.

d41d8c, 62 lines

```
using T = long double;
const int N = 410, M = 30010;
const T eps = 1e-7;
int n, m;
int Left[M], Down[N];
T a[M][N], b[M], c[N], v, sol[N];
bool eq(T a, T b) { return fabs(a - b) < eps;  }
bool ls(T a, T b) { return a < b && !eq(a, b); }
void init(int p, int q) {
  n = p; m = q; v = 0;
  for(int i = 1; i <= m; i++){
    for(int j = 1; j <= n; j++) a[i][j]=0;
  }
  for(int i = 1; i <= m; i++) b[i]=0;
  for(int i = 1; i <= n; i++) c[i]=sol[i]=0;
}
void pivot(int x,int y) {
  swap(Left[x], Down[y]);
  T k = a[x][y]; a[x][y] = 1;
  vector<int> nz;
  for(int i = 1; i <= n; i++){
    a[x][i] /= k;
    if(!eq(a[x][i], 0)) nz.push_back(i);
  }
  b[x] /= k;

  for(int i = 1; i <= m; i++){
    if(i == x || eq(a[i][y], 0)) continue;
    k = a[i][y]; a[i][y] = 0;
    b[i] -= k*b[x];
    for(int j : nz) a[i][j] -= k*a[x][j];
  }
  if(eq(c[y], 0)) return;
  k = c[y]; c[y] = 0;
  v += k*b[x];
  for(int i : nz) c[i] -= k*a[x][i];
}
// 0: found solution, 1: no feasible solution, 2: unbounded
int solve() {
  for(int i = 1; i <= n; i++) Down[i] = i;
  for(int i = 1; i <= m; i++) Left[i] = n+i;
  while(1) { // Eliminating negative b[i]
    int x = 0, y = 0;
    for(int i = 1; i <= m; i++) if (ls(b[i], 0) && (x == 0 || b
        [i] < b[x])) x = i;
    if(x == 0) break;
    for(int i = 1; i <= n; i++) if (ls(a[x][i], 0) && (y == 0
        || a[x][i] < a[x][y])) y = i;
    if(y == 0) return 1;
    pivot(x, y);
  }
  while(1) {
    int x = 0, y = 0;
    for(int i = 1; i <= n; i++)
      if (ls(0, c[i]) && (!y || c[i] > c[y])) y = i;
    if(y == 0) break;
    for(int i = 1; i <= m; i++)
      if (ls(0, a[i][y]) && (!x || b[i]/a[i][y] < b[x]/a[x][y])
          ) x = i;
    if(x == 0) return 2;
    pivot(x, y);
  }
  for(int i = 1; i <= m; i++) if(Left[i] <= n) sol[Left[i]] = b
      [i];
  return 0;
}
```

MillerRabinPollardRho.h

d41d8c, 88 lines

```
// Usage: NT::factorize(n, res);
// Caution! res may not be sorted.
mt19937 rng(1010101);
ll randInt(ll l, ll r) {
  return uniform_int_distribution<ll>(l, r)(rng);
}
namespace NT {
  const ll Base[12] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
       37 };
  const ll NAIVE_MAX = 1'000'000'000;

  ll add(ll a, ll b, const ll mod) {
    if (a + b >= mod) return a + b - mod;
    return a + b;
  }
  ll mul(ll a, ll b, const ll mod) {
    return (__int128_t)a * b % mod;
  }
  ll _pow(ll a, ll b, const ll mod) {
    ll ret = 1;
    while (b) {
      if (b & 1) ret = mul(ret, a, mod);
      a = mul(a, a, mod); b /= 2;
    }
    return ret;
  }
  bool naive_prime(ll n) {
    for (int i = 2; i * i <= n; i++) {
      if (n % i == 0) return false;
    }
    return true;
  }
  bool is_prime(ll n) {
    if (n <= NAIVE_MAX) {
      return naive_prime(n);
    }
    if (n % 2 == 0) return false;
    // Miller-Rabin Primality test
    ll s = 0, d = n - 1;
    while (d % 2 == 0) {
      s += 1; d /= 2;
    }

    // When n < 2^64, it is okay to test only prime bases <= 37
    for (ll base : Base) {
      ll x = _pow(base, d, n), f = 0;
      if (x == 1) f = 1;
      for (int i = 0; i < s; i++) {
        if (x == n - 1) {
          f = 1;
        }
        x = mul(x, x, n);
      }
      if (!f) return false;
    }
    return true;
  }
  ll run(ll n, ll x0, ll c) {
    function<ll(ll)> f = [c, n](ll x) {
      return NT::add(NT::mul(x, x, n), c, n);
    };
    ll x = x0, y = x0, g = 1;
    while (g == 1) {
      x = f(x);
      y = f(y); y = f(y);
      g = gcd(abs(x - y), n);
    }
    return g;
  }
  // Res is NOT sorted after this call
  void factorize(ll n, vector<ll> &Res) {
    if (n == 1) return;
    if (n % 2 == 0) {
      Res.push_back(2); factorize(n / 2, Res);
      return;
    }
    if (is_prime(n)) {
      Res.push_back(n); return;
    }
    while (1) {
      ll x0 = randInt(1, n - 1), c = randInt(1, 20) % (n - 1) +
          1;
      ll g = run(n, x0, c);
      if (g != n) {
        factorize(n / g, Res); factorize(g, Res);
        return;
      }
    }
  }
};
```

LinearSieve.h

d41d8c, 27 lines

```
void linear_sieve() {
  vector<int> p(M), pr;
  vector<int> mu(M), phi(M);
  for (int i = 2; i < M; i++) {
    if (!p[i]) {
      pr.push_back(i);
      mu[i] = -1;
      phi[i] = i - 1; // value of multiplicative function for
          prime
    }
    for (int j = 0; j < pr.size() && i * pr[j] < M; j++) {
      p[i * pr[j]] = 1;
      if (i % pr[j] == 0) {
        mu[i * pr[j]] = 0;
        phi[i * pr[j]] = phi[i] * pr[j];
        break;
      }
      else {
        mu[i * pr[j]] = mu[i] * mu[pr[j]];
        phi[i * pr[j]] = phi[i] * phi[pr[j]];
      }
    }
  }
  for (int i = 2; i < 50; i++) {
    cout << "mu(" << i << ") = " << mu[i] << ' ';
    cout << "phi(" << i << ") = " << phi[i] << '\n';
  }
}
```

CRTDiophantine.h

d41d8c, 40 lines

```
typedef long long lint;
typedef pair<lint, lint> pint;
// return: [g, x, y], g = gcd(a, b), solution of ax+by=g.
std::array<ll, 3> exgcd(ll a, ll b) {
```

```
  if (b == 0) {
    return {a, 1, 0};
  }
  auto [g, x, y] = exgcd(b, a % b);
  return {g, y, x - a / b * y};
}
// returns (x0, y0) where x0 >= 0, x0 = -1 if solution does not
      exist
pii solve(ll a, ll b, ll c) {
  ll g = __gcd(a, b);
  if (c % g != 0) return pii(-1, 0);
  c /= g; a /= g; b /= g;

  vector<ll> V;
  while (b != 0) {
    ll q = a / b, r = a % b;
    V.push_back(q);
    a = b; b = r;
  }
  ll x = c, y = 0;
  while (!V.empty()) {
    ll q = V.back(); V.pop_back();
    b += q * a; swap(a, b);
    x -= q * y; swap(x, y);
  }
  ll r = (x - (b + x % b) % b) / b;
  x -= b * r; y += a * r;

  return pii(x, y);
}
// returns (x, period of x), x = -1 if solution doesn't exist
pii CRT(ll a1, ll m1, ll a2, ll m2) {
  auto sol = solve(m1, m2, a2 - a1);
  if (sol.va == -1) return pii(-1, 0);
  ll g = __gcd(m1, m2); m2 /= g;
  return pii((m1 * sol.va + a1) % (m1 * m2), m1 * m2);
}
```

FancyDiophantine.h

Description: Not tested!

d41d8c, 23 lines

```
// solutions to ax + by = c where x in [xlow, xhigh] and y in [
      ylow, yhigh]
// cnt, leftsol, rightsol, gcd of a and b
template<class T> array<T, 6> solve_linear_diophantine(T a, T b
    , T c, T xlow, T xhigh, T ylow, T yhigh){
  T g, x, y = euclid(a >= 0 ? a : -a, b >= 0 ? b : -b, x, y);
      array<T, 6> no_sol{0, 0, 0, 0, 0, g};
  if(c % g) return no_sol; x *= c / g, y *= c / g;
  if(a < 0) x = -x; if(b < 0) y = -y;
  a /= g, b /= g, c /= g;
  auto shift = [&](T &x, T &y, T a, T b, T cnt){ x += cnt * b,
      y -= cnt * a; };
  int sign_a = a > 0 ? 1 : -1, sign_b = b > 0 ? 1 : -1; shift(x
      , y, a, b, (xlow - x) / b);
  if(x < xlow) shift(x, y, a, b, sign_b);
  if(x > xhigh) return no_sol;
  T lx1 = x; shift(x, y, a, b, (xhigh - x) / b);
  if(x > xhigh) shift(x, y, a, b, -sign_b);
  T rx1 = x; shift(x, y, a, b, -(ylow - y) / a);
  if(y < ylow) shift(x, y, a, b, -sign_a);
  if(y > yhigh) return no_sol;
  T lx2 = x; shift(x, y, a, b, -(yhigh - y) / a);
  if(y > yhigh) shift(x, y, a, b, sign_a);
  T rx2 = x; if(lx2 > rx2) swap(lx2, rx2);
  T lx = max(lx1, lx2), rx = min(rx1, rx2);
  if(lx > rx) return no_sol;
  return {(rx - lx) / (b >= 0 ? b : -b) + 1, lx, (c - lx * a) /
       b, rx, (c - rx * a) / b, g};
}
```

DiscreteLog.h

Description: Not tested!

Time: $\mathcal{O}\left(\sqrt{P}\log P\right)$, $\mathcal{O}\left(\log^2 P\right)$ in random data. With hash table one log factor removed.

d41d8c, 34 lines

```
// Given A, B, P, solve A^x === B mod P
ll DiscreteLog(ll A, ll B, ll P){
  __gnu_pbds::gp_hash_table<ll,__gnu_pbds::null_type> st;
  ll t = ceil(sqrt(P)), k = 1; // use binary search?
  for(int i=0; i<t; i++) st.insert(k), k = k * A % P;
  ll inv = Pow(k, P-2, P);
  for(int i=0, k=1; i<t; i++, k=k*inv%P){
    ll x = B * k % P;
    if(st.find(x) == st.end()) continue;
    for(int j=0, k=1; j<t; j++, k=k*A%P){
      if(k == x) return i * t + j;
    }
  }
  return -1;
}
// Given A, P, solve X^2 === A mod P
ll DiscreteSqrt(ll A, ll P){
  if(A == 0) return 0;
  if(Pow(A, (P-1)/2, P) != 1) return -1;
  if(P % 4 == 3) return Pow(A, (P+1)/4, P);
  ll s = P - 1, n = 2, r = 0, m;
  while(~s & 1) r++, s >>= 1;
  while(Pow(n, (P-1)/2, P) != P-1) n++;
  ll x = Pow(A, (s+1)/2, P), b = Pow(A, s, P), g = Pow(n, s, P)
      ;
  for(;; r=m){
    ll t = b;
    for(m=0; m<r && t!=1; m++) t = t * t % P;
    if(!m) return x;
    ll gs = Pow(g, 1LL << (r-m-1), P);
    g = gs * gs % P;
    x = x * gs % P;
    b = b * g % P;
  }
}
```

PowerTower.h

Description: Not tested!

d41d8c, 23 lines

```
bool PowOverflow(ll a, ll b, ll c){
  __int128_t res = 1;
  bool flag = false;
  for(; b; b >>= 1, a = a * a){
    if(a >= c) flag = true, a %= c;
    if(b & 1){
      res *= a;
      if(flag || res >= c) return true;
    }
  }
  return false;
}
ll Recursion(int idx, ll mod, const vector<ll> &vec){
  if(mod == 1) return 1;
  if(idx + 1 == vec.size()) return vec[idx];
  ll nxt = Recursion(idx+1, phi[mod], vec);
  if(PowOverflow(vec[idx], nxt, mod)) return Pow(vec[idx], nxt,
      mod) + mod;
  else return Pow(vec[idx], nxt, mod);
}
ll PowerTower(const vector<ll> &vec, ll mod){ // vec[0]^(vec[1]
    ^(vec[2]^(...)))
  if(vec.size() == 1) return vec[0] % mod;
  else return Pow(vec[0], Recursion(1, phi[mod], vec), mod);
}
```

GaussJordanElimination.h

Description: Not tested! Seems like we have to make Mul, Div, and sth else according to problem. square option seems like square matrix.

d41d8c, 30 lines

```
template<typename T> // return {rref, rank, det, inv}
tuple<vector<vector<T>>, T, T, vector<vector<T>>> Gauss(vector<
    vector<T>> a, bool square=true){
  int n = a.size(), m = a[0].size(), rank = 0;
  vector<vector<T>> out(n, vector<T>(m, 0)); T det = T(1);
  for(int i=0; i<n; i++) if(square) out[i][i] = T(1);
  for(int i=0; i<m; i++){
    if(rank == n) break;
    if(IsZero(a[rank][i])){
      T mx = T(0); int idx = -1; // fucking precision error
      for(int j=rank+1; j<n; j++) if(mx < abs(a[j][i])) mx =
          abs(a[j][i]), idx = j;
      if(idx == -1 || IsZero(a[idx][i])){ det = 0; continue; }
      for(int k=0; k<m; k++){
        a[rank][k] = Add(a[rank][k], a[idx][k]);
        if(square) out[rank][k] = Add(out[rank][k], out[idx][k
            ]);
      }
    }
    det = Mul(det, a[rank][i]);
    T coeff = Div(T(1), a[rank][i]);
    for(int j=0; j<m; j++) a[rank][j] = Mul(a[rank][j], coeff);
    for(int j=0; j<m; j++) if(square) out[rank][j] = Mul(out[
        rank][j], coeff);
    for(int j=0; j<n; j++){
      if(rank == j) continue;
      T t = a[j][i]; // Warning: [j][k], [rank][k]
      for(int k=0; k<m; k++) a[j][k] = Sub(a[j][k], Mul(a[rank
          ][k], t));
      for(int k=0; k<m; k++) if(square) out[j][k] = Sub(out[j][
          k], Mul(out[rank][k], t));
    }
    rank++;
  }
  return {a, rank, det, out};
}
```

DeBruijnSequence.h

Description: Calculate length-L DeBruijn sequence.

Usage: Returns 1-base index. K is the number of alphabet, N is the length of different substring, L is the return length ($0 <= L <= K\hat{}N$). vector<int> seq = de_bruijn(K, N, L);

Time: $\mathcal{O}(L)$, $N = L = 10^5, K = 10$ in 12ms.

d41d8c, 23 lines

```
vector<int> de_bruijn(int K, int N, int L) {
  vector<int> ans, tmp;
  function<void(int)> dfs = [&](int T) {
    if((int)ans.size() >= L) return;
    if((int)tmp.size() == N) {
      if(N%T == 0)
        for(int i = 0; i < T && (int)ans.size() < L; i++)
          ans.push_back(tmp[i]);
    } else {
      int k = ((int)tmp.size()-T >= 0 ? tmp[(int)tmp.size()-T]
          : 1);
      tmp.push_back(k);
      dfs(T);
      tmp.pop_back();
      for(int i = k+1; i <= K; i++) {
        tmp.push_back(i);
        dfs((int)tmp.size());
        tmp.pop_back();
```

```
      }
    }
  };
  dfs(1);
  return ans;
}
```

4.1 Equations

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A_i'}{\det A}$$

where A_i' is A with the i'th column replaced by b.

4.2 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let $X_1, X_2, \ldots$ be a sequence of random variables generated by the Markov process. Then there is a transition matrix $\mathbf{P} = (p_{ij})$, with $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$, and $\mathbf{p}^{(n)} = \mathbf{P}^n \mathbf{p}^{(0)}$ is the probability distribution for X_n (i.e., $p_i^{(n)} = \Pr(X_n = i)$), where $\mathbf{p}^{(0)}$ is the initial distribution.

π is a stationary distribution if $\pi = \pi \mathbf{P}$. If the Markov chain is *irreducible* (it is possible to get to any state from any state), then $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i. π_j / π_i is the expected number of visits in state j between two visits in state i.

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors, π_i is proportional to node i's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k\to\infty} \mathbf{P}^k = \mathbf{1}\pi$.

A Markov chain is an A-chain if the states can be partitioned into two sets $\mathbf{A}$ and $\mathbf{G}$, such that all states in $\mathbf{A}$ are absorbing ($p_{ii} = 1$), and all states in $\mathbf{G}$ leads to an absorbing state in $\mathbf{A}$. The probability for absorption in state $i \in \mathbf{A}$, when the initial state is j, is $a_{ij} = p_{ij} + \sum_{k \in \mathbf{G}} a_{ik} p_{kj}$. The expected time until absorption, when the initial state is i, is $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki} t_k$.

Strings (5)

joint eertree로 다음과 같은 문제를 해결할 수 있습니다.

문제	풀이
문자열 k개가 주어졌을 때, 모든 문자열의 substring인 palindrome의 개수를 구하여라.	$eertree(S_1, S_2, \ldots, S_k)$에서 flag의 값이 모두 1인 정점의 개수를 구하면 됩니다.
문자열 k개가 주어졌을 때, 모든 문자열의 substring인 palindrome 중 가장 긴 것을 구하여라.	$eertree(S_1, S_2, \ldots, S_k)$에서 flag의 값이 모두 1인 가장 긴 길이의 정점을 구하면 됩니다.
두 문자열 S, T에 대해 T보다 S에서 더 많이 나타나는 palindrome의 개수를 구하여라.	$eertree(S, T)$를 만들고 occ_S와 occ_T를 계산합니다. (occ에 대해서는 위 문제 팰린드롬에 설명되어 있습니다.) $occ_S[v] > occ_T[v]$인 정점 v의 개수가 답이 됩니다.
두 문자열 S, T에 대해 $S[i..i+k] = T[j..j+k]$인 (i, j, k)의 개수를 구하여라.	$eertree(S, T)$를 만들고 occ_S와 occ_T를 계산합니다. $sum_v\ occ_S[v] * occ_T[v]$ 의 값이 답이 됩니다.

KMP.h

Usage: 0-base. `pmt[i] = s[0..i]'s common longest prefix and suffix. kmp[i] = ith matched begin position.`
Time: $\mathcal{O}(n)$ d41d8c, 40 lines

```
vector<int> get_pmt(const string& s) {
  int n = s.size();
  vector<int> pmt(n, 0);
  // except finding itself by searching from s[0]
  int b = 1, m = 0;
  // s[b + m]: letter to compare
  while (b + m < n) {
    if (s[b+m] == s[m]) {
      pmt[b+m] = m + 1;
      m++;
    } else {
      if (m > 0) {
        b += m - pmt[m-1];
        m = pmt[m-1];
      } else {
        b++;
      }
    }
  }
  return pmt;
}
vector<int> KMP(const string& hay, const string& needle) {
  vector<int> pmt = get_pmt(needle);
  vector<int> ret;
  int b = 0, m = 0;
  while (b <= (int)hay.size() - needle.size()) {
    if (m < needle.size() && hay[b+m] == needle[m]) {
      m++;
      if (m == needle.size()) ret.push_back(b);
    } else {
      if (m > 0) {
        b += m - pmt[m-1];
        m = pmt[m-1];
      } else {
        b++;
      }
    }
  }
  return ret;
}
```

Zfunc.h

Usage: `Z[i] stores lcp of s[0..] and s[i..]`
Time: $\mathcal{O}(n)$, $N = 10^5$ in 20ms. d41d8c, 9 lines

```
vector<int> Z(string &s) {
  vector<int> ret(s.size(), 0); ret[0] = s.size();
  for(int i = 1, l = 0, r = 1; i < s.size(); i++) {
    ret[i] = max(0, min(ret[i-l], r-i));
    while(ret[i]+i < s.size() && s[i+ret[i]] == s[ret[i]]) ret[
        i]++;
    if(i+ret[i] > r) r = i+ret[i], l = i;
  }
  return ret;
}
```

Manacher.h

Usage: `mana[i] stores radius of maximal palindrome of intervened string. Single char is radius 0. Max element of mana is equal to real longest palindrome length.`
Time: $\mathcal{O}(N)$, $N = 10^5$ in 4ms. d41d8c, 11 lines

```
vector<int> mana(string &s) {
  string t = ".";
  for(auto i : s) { t += i; t += '.'; }
  vector<int> ret(t.size(), 0);
  for(int i = 0, c = 0, r = 0; i < (int)t.size(); i++) {
    if(i < r) ret[i] = min(r-i, ret[2*c-i]);
    while(i-ret[i]-1 >= 0 && i+ret[i]+1 < (int)t.size() && t[i-
        ret[i]-1] == t[i+ret[i]+1]) ret[i]++;
    if(r < i+ret[i]) r = i+ret[i], c = i;
  }
  return ret;
}
```

Duval.h

Description: Return lyndon decomposition start positions and n.
Time: $\mathcal{O}(n)$, no data. d41d8c, 19 lines

```
vector<int> duval(string const& s) {
  vector<int> ret;
  int n = s.size();
  int i = 0;
  while(i < n) {
    int j = i+1, k = i;
    while(j < n && s[k] <= s[j]) {
      if(s[k] < s[j]) k = i;
      else k++;
      j++;
    }
    while(i <= k) {
      ret.push_back(i);
      i += j-k;
    }
  }
  ret.push_back(n);
  return ret;
}
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.
Time: $\mathcal{O}(N)$, no data. d41d8c, 16 lines

```
string min_cyclic_string(string s) {
  int n = s.size();
  s += s;
  int i = 0, ans = 0;
  while(i < n) {
    ans = i;
    int j = i+1, k = i;
    while(j < 2*n && s[k] <= s[j]) {
      if(s[k] < s[j]) k = i;
      else k++;
      j++;
    }
    while(i <= k) i += j-k;
  }
  return s.substr(ans, n);
}
```

SuffixArray.h

Usage: `0-base index. sa[i]: lexicographically (i+1)'th suffix (of d letters). lcp[i]: lcp between sa[i] and sa[i+1]. r[i]: rank of s[i..n-1] when only consider first d letters. nr: temp array for next rank. cnt[i]: number of positions which has i of next rank. rf[r]: lexicographically first position which suffixes (of d letters) has rank r. rdx[i]: lexicographically (i+1)'th suffix when only consider (d+1)'th 2d'th letters.`
Time: $\mathcal{O}(n \log n)$, $N = 5 \times 10^5$ in 176ms. d41d8c, 41 lines

```
void suffix_array(string S, vector<int> &sa, vector<int> &lcp)
    {
  int n = S.size();
```

```
  vector<int> r(n), nr(n), rf(n), rdx(n);
  sa.resize(n); lcp.resize(n);

  for (int i = 0; i < n; i++) sa[i] = i;
  sort(sa.begin(), sa.end(), [&](int a, int b) { return S[a] <
       S[b]; });
  for (int i = 1; i < n; i++) r[sa[i]] = r[sa[i - 1]] + (S[sa[i
       - 1]] != S[sa[i]]);

  for (int d = 1; d < n; d <<= 1) {
    for (int i = n - 1; i >= 0; i--) {
      rf[r[sa[i]]] = i;
    }
    int j = 0;
    for (int i = n - d; i < n; i++) rdx[j++] = i;
    for (int i = 0; i < n; i++) {
      if (sa[i] >= d) rdx[j++] = sa[i] - d;
    }
    for (int i = 0; i < n; i++) {
      sa[rf[r[rdx[i]]]++] = rdx[i];
    }
    nr[sa[0]] = 0;
    for (int i = 1; i < n; i++) {
      if (r[sa[i]] != r[sa[i - 1]]) {
        nr[sa[i]] = nr[sa[i - 1]] + 1;
      }
      else {
        int prv = (sa[i - 1] + d >= n ? -1 : r[sa[i - 1] + d]);
        int cur = (sa[i] + d >= n ? -1 : r[sa[i] + d]);
        nr[sa[i]] = nr[sa[i - 1]] + (prv != cur);
      }
    }
    swap(r, nr);
    if (r[sa[n-1]] == n-1) break;
  }
  for (int i = 0, len = 0; i < n; ++i, len = max(len - 1, 0)) {
    if (r[i] == n - 1) continue;
    for (int j = sa[r[i] + 1]; S[i + len] == S[j + len]; ++len)
        ;
    lcp[r[i]] = len;
  }
}
```

Hashing.h

Usage: Hashing<base, mod> hsh; hsh.Build(s); query is 1-base (but s is not modified).

d41d8c, 16 lines

```
// 1e5+3, 1e5+13, 131'071, 524'287, 1'299'709, 1'301'021
// 1e9-63, 1e9+7, 1e9+9, 1e9+103
template<ll P, ll M> struct Hashing {
  vector<ll> H, B;
  void Build(const string &S){
    H.resize(S.size()+1);
    B.resize(S.size()+1);
    B[0] = 1;
    for(int i=1; i<=S.size(); i++) H[i] = (H[i-1] * P + S[i-1])
          % M;
    for(int i=1; i<=S.size(); i++) B[i] = B[i-1] * P % M;
  }
  ll sub(int s, int e){
    ll res = (H[e] - H[s-1] * B[e-s+1]) % M;
    return res < 0 ? res + M : res;
  }
};
```

AhoCorasick.h

Description: Aho-Corasick automaton, used for multiple pattern matching.
Time: Build $\mathcal{O}(26n)$, find $\mathcal{O}(n)$. Build 10^5, find 10^7 in 132ms.

d41d8c, 55 lines

```
struct Node {
  Node *go[26], *fail;
  bool end;
Node() : fail(nullptr), end(false) { fill(go, go + 26, nullptr)
    ; }
  ~Node() {
    for (Node *next: go)
      if (next) delete next;
  }
};

Node * build_trie(vector<string> &patterns) {
  Node *root = new Node();

  for (string &p: patterns) {
    Node *curr = root;
    for (char c: p) {
      if (!curr->go[c - 'a']) curr->go[c - 'a'] = new Node();
      curr = curr->go[c - 'a'];
    }
    curr->end = true;
  }

  queue<Node *> q; q.push(root);
  root->fail = root;

  while (!q.empty()) {
    Node *curr = q.front(); q.pop();
    for (int i = 0; i < 26; i++) {
      Node *next = curr->go[i];
      if (!next) continue;
      q.push(next);

      if (curr == root) next->fail = root;
      else {
        Node *dest = curr->fail;
        while (dest != root && !dest->go[i]) dest = dest->fail;
        if (dest->go[i]) dest = dest->go[i];
        next->fail = dest;
        next->end |= dest->end;
      }
    }
  }

  return root;
}

bool find_trie(Node *trie, string &s) {
  Node *curr = trie;
  for (char c: s) {
    while (curr != trie && !curr->go[c - 'a']) curr = curr->
        fail;
    if (curr->go[c - 'a']) curr = curr->go[c - 'a'];
    if (curr->end) return true;
  }
  return false;
}
```

SuffixAutomaton.h

Usage: add(c) adds c at the end of string. topo(f) executes f while topological sort when erase edges. f(x, y, c): y->x edge marked as c. Note that c is 0-base.
Time: add is amortized $\mathcal{O}(1)$, topo is $\mathcal{O}(n)$, no data. 10^6 add call and one topo call in 668ms.

d41d8c, 65 lines

```
template <int MAXN>
struct SuffixAutomaton {
  struct Node {
    int nxt[MAXN];
    int len = 0, link = 0;
    Node() { memset(nxt, -1, sizeof nxt); }
  };

  int root = 0;
  vector<Node> V;

  SuffixAutomaton() {
    V.resize(1);
    V.back().link = -1;
  }

  void add(int c) {
    V.push_back(Node());
    V.back().len = V[root].len+1;
    int tmp = root;
    root = (int)V.size()-1;
    while(tmp != -1 && V[tmp].nxt[c] == -1) {
      V[tmp].nxt[c] = root;
      tmp = V[tmp].link;
    }
    if(tmp != -1) {
      int x = V[tmp].nxt[c];
      if(V[tmp].len+1 < V[x].len) {
        int y = x;
        x = (int)V.size();
        V.push_back(V[y]);
        V.back().len = V[tmp].len+1;
        V[y].link = x;
        while(tmp != -1 && V[tmp].nxt[c] == y) {
          V[tmp].nxt[c] = x;
          tmp = V[tmp].link;
        }
      }
      V[root].link = x;
    }
  }

  void topo(function<void(int, int, int)> f) {
    vector<int> indeg(V.size(), 0);
    for(auto &node : V) {
      for(auto j : node.nxt) {
        if(j == -1) continue;
        indeg[j]++;
      }
    }
    queue<int> Q;
    for(int i = 0; i < (int)indeg.size(); i++)
      if(indeg[i] == 0) Q.push(i);
    while(Q.size()) {
      int tmp = Q.front(); Q.pop();
      auto &node = V[tmp];
      for(int j = 0; j < MAXN; j++) {
        if(node.nxt[j] == -1) continue;
        f(node.nxt[j], tmp, j);
        if(--indeg[node.nxt[j]] == 0)
          Q.push(node.nxt[j]);
      }
    }
  }
};
```

eertree.h

Description: eertree.
Usage: add is same as suffix automaton. Note that c is 0-base.
Time: add is amortized $\mathcal{O}(1)$, 10^6 add in 212ms.

d41d8c, 43 lines

```
template <int MAXN>
struct eertree {
```

```
  struct Node {
    int len = 0, link = 0, cnt = 0;
    array<int, MAXN> nxt;
    Node() { fill(nxt.begin(), nxt.end(), -1); }
  };

  vector<int> S;
  vector<Node> V;
  int root = 0;

  eertree() {
    V.resize(2);
    V[0].len = -1;
  }

  void add(int c) {
    S.push_back(c);
    for(int tmp = root; ; tmp = V[tmp].link) {
      auto iter = S.rbegin()+V[tmp].len+1;
      if(iter < S.rend() && *iter == c) {
        if(V[tmp].nxt[c] == -1) {
          root = V.size();
          V[tmp].nxt[c] = root;
          V.push_back(Node());
          V.back().len = V[tmp].len+2;
          tmp = V[tmp].link;
          iter = S.rbegin()+V[tmp].len+1;
          while(iter >= S.rend() || *iter != c) {
            tmp = V[tmp].link;
            iter = S.rbegin()+V[tmp].len+1;
          }
          tmp = V[tmp].nxt[c];
          if(V.back().len == 1 || tmp <= 0) V.back().link = 1;
          else V.back().link = tmp;
        } else root = V[tmp].nxt[c];
        V[root].cnt++;
        break;
      }
    }
  }
};
```

RunEnumeration.h

Description: Run enumeration.
Time: $\mathcal{O}(N \log N)$, $N = 2 \times 10^5$ in 457ms.

d41d8c, 83 lines

```
struct runs{
  int t, l, r;
  bool operator<(const runs &x)const{
    return make_tuple(t, l, r) < make_tuple(x.t, x.l, x.r);
  }
  bool operator==(const runs &x)const{
    return make_tuple(t, l, r) == make_tuple(x.t, x.l, x.r);
  }
};

namespace ds{
  const int MAXN = 400005;

  vector<int> sfx, rev, lcp;
  int spt[19][MAXN], lg[MAXN], n;

  int get_lcp(int s, int e){
    if(s == 2 * n + 1 || e == 2 * n + 1) return 0;
    s = rev[s]; e = rev[e];
    if(s > e) swap(s, e);
    int l = lg[e - s];
    return min(spt[l][e - 1], spt[l][s + (1<<l) - 1]);
  }

  int get_lcp_rev(int s, int e){
    return get_lcp(2*n+1-s, 2*n+1-e);
  }

  void prep(string str){
    n = str.size();
    string s = str;
    string r = s; reverse(r.begin(), r.end());
    s = s + "#" + r;
    suffix_array(s, sfx, lcp);
    rev.resize(sfx.size());
    for(int i=0; i<sfx.size(); i++) rev[sfx[i]] = i;
    for(int i=1; i<MAXN; i++){
      lg[i] = lg[i-1];
      while((2 << lg[i]) <= i) lg[i]++;
    }
    for(int i=0; i<sfx.size()-1; i++) spt[0][i] = lcp[i];
    for(int i=1; i<19; i++){
      for(int j=0; j<sfx.size(); j++){
        spt[i][j] = spt[i-1][j];
        if(j >= (1<<(i-1))) spt[i][j] = min(spt[i][j], spt[i
            -1][j-(1<<(i-1))]);
      }
    }
  }
}

vector<runs> run_enumerate(string s){
  int n = s.size();
  vector<pii> v;
  auto get_interval = [&](string t){
    vector<int> sfx, lcp;
    suffix_array(t, sfx, lcp);
    vector<int> rev(n + 1);
    for(int i = 0; i < n; i++) rev[sfx[i]] = i;
    rev[n] = -1;
    vector<int> stk = {n}, ans(n);
    for(int i = n - 1; i >= 0; i--){
      while(stk.size() && rev[stk.back()] > rev[i]) stk.
          pop_back();
      v.emplace_back(i, stk.back());
      stk.push_back(i);
    }
  };
  ds::prep(s);
  get_interval(s);
  for(auto &i : s) i = 'a' + 'z' - i;
  get_interval(s);
  vector<runs> ans;
  for(auto &[x, y] : v){
    int s = x - ds::get_lcp_rev(x, y);
    int e = y + ds::get_lcp(x, y);
    int p = y - x;
    if(e - s >= 2 * p){
      ans.push_back({p, s, e});
    }
  }
  sort(ans.begin(), ans.end());
  ans.resize(unique(ans.begin(), ans.end()) - ans.begin());
  return ans;
}
```

Geometry (6)

6.1 Analytic Geometry

Area $A = \sqrt{p(p-a)(p-b)(p-c)}$ when $p = (a+b+c)/2$
Circumscribed circle $R = abc/4A$, inscribed circle $r = A/p$
Middle line length $m_a = \sqrt{2b^2 + 2c^2 - a^2}/2$
Bisector line length $s_a = \sqrt{bc[1 - (\frac{a}{b+c})^2]}$

Name	α	β	γ	
R	$a^2\mathcal{A}$	$b^2\mathcal{B}$	$c^2\mathcal{C}$	$\mathcal{A} = b^2 + c^2 - a^2$
r	a	b	c	$\mathcal{B} = a^2 + c^2 - b^2$
G	1	1	1	$\mathcal{C} = a^2 + b^2 - c^2$
H	$\mathcal{BC}$	$\mathcal{CA}$	$\mathcal{AB}$	
Excircle(A)	$-a$	b	c	

With side lengths a, b, c, d, diagonals e, f, diagonals angle θ, area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:
$4A = 2ef \cdot \sin\theta = F \tan\theta = \sqrt{4e^2f^2 - F^2}$
For cyclic quadrilaterals the sum of opposite angles is $180°$, $ef = ac + bd$, and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.
$HG : GO = 1 : 2$. H of triangle made by middle point on arc of circumscribed circle is equal to inscribed circle center of original triangle.

PointInteger.h

d41d8c, 9 lines

```
struct Point {
  ll x, y;
  Point operator-(const Point& r) const { return Point{ x-r.x,
        y-r.y }; }
  ll operator^(const Point& r) const { return x * r.y - y * r.x
        ; }
  bool operator<(const Point& r) const { return (x == r.x ? y <
        r.y : x < r.x); }
  bool operator==(const Point& r) const { return (x == r.x && y
        == r.y); }
  friend istream& operator>>(istream& is, Point& p) { is >> p.x
        >> p.y; return is; }
  friend ostream& operator<<(ostream& os, const Point& p) { os
        << '(' << p.x << ' ' << p.y << ')'; return os; }
};
```

PointDouble.h

d41d8c, 17 lines

```
int sgn(ld x) {
  return abs(x) < 1e-16L ? 0 : (x > 0 ? 1 : -1);
}

struct Point {
  ld x, y;
  Point operator-(const Point& r) const { return Point{ x-r.x,
        y-r.y }; }
  Point operator*(ld a) const { return Point{ x*a, y*a }; }
  ld operator*(const Point& r) const { return x * r.x + y * r.y
        ; }
  ld operator^(const Point& r) const { return x * r.y - y * r.x
        ; }
  bool operator<(const Point& r) const { return (sgn(x-r.x) < 0
        || (sgn(x-r.x) == 0 && sgn(y-r.y) < 0)); }
  bool operator==(const Point& r) const { return (sgn(x-r.x) ==
        0 && sgn(y-r.y) == 0); }
  ld norm() const { return sqrtl(x*x + y*y); }
  ld sqnorm() const { return x*x + y*y; }
  friend istream& operator>>(istream& is, Point& p) { is >> p.x
        >> p.y; return is; }
  friend ostream& operator<<(ostream& os, const Point& p) { os
        << '(' << p.x << ' ' << p.y << ')'; return os; }
};
```

SegmentDistance.h

d41d8c, 18 lines

```
ld proj_height(Point a, Point b, Point x) {
  ld t1 = (b-a) * (x-a), t2 = (a-b) * (x-b);
  if (sgn(t1*t2) >= 0) return abs((b-a)^(x-a)) / (b-a).norm();
  else return 1e18;
}

ld segment_dist(Point s1, Point e1, Point s2, Point e2) {
  ld ans = 1e18;
  ans = min(ans, (s2-s1).norm());
  ans = min(ans, (e2-s1).norm());
  ans = min(ans, (s2-e1).norm());
  ans = min(ans, (e2-e1).norm());
  ans = min(ans, proj_height(s1, e1, s2));
  ans = min(ans, proj_height(s1, e1, e2));
  ans = min(ans, proj_height(s2, e2, s1));
  ans = min(ans, proj_height(s2, e2, e1));
  return ans;
}
```

SegmentIntersection.h

Usage: `intersect(..)` returns type of segment intersection defined in enum.

`find_point(..)` 0: not intersect, -1: infinity, 1: cross. Return value is flag, xp, xq, yp, yq given in fraction. xp is numer, xq is domi.

d41d8c, 46 lines

```
int sgn(ll x) { return (x > 0 ? 1 : (x < 0 ? -1 :0)); }

enum Intersection {
  NONE, ENDEND, ENDMID, MID, INF
};

int intersect(Point s1, Point e1, Point s2, Point e2) {
  int t1 = sgn((e1-s1) ^ (s2-e1));
  int t2 = sgn((e1-s1) ^ (e2-e1));
  if (t1 == 0 && t2 == 0) {
    if (e1 < s1) swap(s1, e1);
    if (e2 < s2) swap(s2, e2);
    if (e1 == s2 || s1 == e2) return ENDEND;
    else return (e1 < s2 || e2 < s1) ? NONE : INF;
  } else {
    int t3 = sgn((e2-s2) ^ (s1-e2));
    int t4 = sgn((e2-s2) ^ (e1-e2));
    if (t1*t2 == 0 && t3*t4 == 0) return ENDEND;
    if (t1 != t2 && t3 != t4) {
      return (t1*t2 == 0 || t3*t4 == 0 ? ENDMID : MID);
    } else {
      return NONE;
    }
  }
}

using T = __int128_t; // T <= O(COORD^3)
tuple<int, T, T, T, T> find_point(Point s1, Point e1, Point s2, Point e2) {
  int res = intersect(s1, e1, s2, e2);
  if (res == NONE) return {0, 0, 0, 0, 0};
  if (res == INF) return {-1, 0, 0, 0, 0};
  auto det = (e1-s1)^(e2-s2);
  if (!det) {
    if(s1 > e1) swap(s1, e1);
    if(s2 > e2) swap(s2, e2);
    if(e1 == s2) return {1, e1.x, 1, e1.y, 1};
    else return {1, e2.x, 1, e2.y, 1};
  }
  T p = (s2-s1)^(e2-s2), q = det;
  T xp = s1.x*q + (e1.x-s1.x)*p, xq = q;
  T yp = s1.y*q + (e1.y-s1.y)*p, yq = q;
  if (xq < 0) xp = -xp, xq = -xq;
  if (yq < 0) yp = -yp, yq = -yq;
  T xg = gcd(abs(xp), xq), yg = gcd(abs(yp), yq);
  return {1, xp/xg, xq/xg, yp/yg, yq/yg};
}
```

AngleSort.h

Description: Possible to sort without consider quadrant. If angle is same then sorted by distance.

d41d8c, 5 lines

```
sort(a, a+n, [&](const pnt &a, const pnt &b){
    if((pi(a.x, a.y) > pi(0, 0)) ^ (pi(b.x, b.y) > pi(0, 0)))
        return pi(a.x, a.y) > pi(b.x, b.y);
    if(ccw(a, b) != 0) return ccw(a, b) > 0;
    return hypot(a) < hypot(b);
});
```

ShamosHoey.h

Description: Check whether segments are intersected at least once or not. Strict option available, and it depends on the segment intersection function.
Usage: 0-base index. `vector<array<Point, 2>> pts(n); auto ret = ShamosHoey(pts);`
Time: $\mathcal{O}(N \log N)$, $N = 2 \times 10^5$ in 320ms.

d41d8c, 75 lines

```
struct Line{
  static ll CUR_X; ll x1, y1, x2, y2, id;
Line(Point p1, Point p2, int id) : id(id) {
  if(p2 < p1) swap(p1, p2);
  x1 = p1.x, y1 = p1.y;
  x2 = p2.x, y2 = p2.y;
} Line() = default;
  int get_k() const { return y1 != y2 ? (x2-x1)/(y1-y2) : -1; }
  void convert_k(int k){ // x1,y1,x2,y2 = O(COORD^2), use i128 in ccw
    Line res; res.x1=x1+y1*k;res.y1=-x1*k+y1; res.x2=x2+y2*k; res.y2=-x2*k+y2;
    x1 = res.x1; y1 = res.y1; x2 = res.x2; y2 = res.y2; if(x1 > x2) swap(x1, x2), swap(y1, y2);
  }
  ld get_y(ll offset=0) const { // OVERFLOW
    ld t = ld(CUR_X-x1+offset) / (x2-x1);
    return t * (y2 - y1) + y1;
  }
  bool operator < (const Line &l) const {
    return get_y() < l.get_y();
  }
  // strict
  // bool operator < (const Line &l) const {
  //     auto le = get_y(), ri = l.get_y();
  //     if(abs(le-ri) > 1e-7) return le < ri;
  //     if(CUR_X == x1 || CUR_X == l.x1) return get_y(1) < l.get_y(1);
  //     else return get_y(-1) < l.get_y(-1);
  // }
}; ll Line::CUR_X = 0;
struct Event{ // f=0 st, f=1 ed
  ll x, y, i, f; Event() = default;
Event(Line l, ll i, ll f) : i(i), f(f) {
  if(f==0) tie(x,y) = tie(l.x1,l.y1);
  else tie(x,y) = tie(l.x2,l.y2);
}
  bool operator < (const Event &e) const {
    return tie(x,f,y) < tie(e.x,e.f,e.y);
    // strict
    // return make_tuple(x,-f,y) < make_tuple(e.x,-e.f,e.y);
  }
};
bool intersect(Line l1, Line l2) {
  Point p1{l1.x1,l1.y1}, p2{l1.x2,l1.y2};
  Point p3{l2.x1,l2.y1}, p4{l2.x2,l2.y2};
  // @TODO Intersection logic depends on problem
}

tuple<bool,int,int> ShamosHoey(vector<array<Point,2>> v){
  int n = v.size(); vector<int> use(n+1);
  vector<Line> lines; vector<Event> E; multiset<Line> T;
  for(int i=0; i<n; i++){
    lines.emplace_back(v[i][0], v[i][1], i);
    if(int t=lines[i].get_k(); 0<=t && t<=n) use[t] = 1;
  }
  int k = find(use.begin(), use.end(), 0) - use.begin();
  for(int i=0; i<n; i++){
    lines[i].convert_k(k);
    E.emplace_back(lines[i], i, 0);
    E.emplace_back(lines[i], i, 1);
  }
  sort(E.begin(), E.end());
  for(auto &e : E){
    Line::CUR_X = e.x;
    if(e.f == 0){
      auto it = T.insert(lines[e.i]);
      if(next(it) != T.end() && intersect(lines[e.i], *next(it)
          )) return {true, e.i, next(it)->id};
      if(it != T.begin() && intersect(lines[e.i], *prev(it)))
          return {true, e.i, prev(it)->id};
    } else{
      auto it = T.lower_bound(lines[e.i]);
      if(it != T.begin() && next(it) != T.end() &&
        intersect(*prev(it), *next(it))) return {true, prev(it)->id, next(it)->id};
      T.erase(it);
    }
  }
  return {false, -1, -1};
}
```

HalfPlaneIntersection.h

Description: Calculate intersection of left half plane of line (s->t).
Usage: 0-base index. `vector<Point> ret = HPI(lines);`
Time: $\mathcal{O}(N \log N)$, no data.

d41d8c, 49 lines

```
struct Line { Point s, t; };
const ld eps = 1e-9;
bool equals(ld a, ld b) { return abs(a-b) < eps; }

bool line_intersect(Point& s1, Point& e1, Point& s2, Point& e2, Point& v) {
  ld det = (e2-s2) ^ (e1-s1);
  if (equals(det, 0)) return 0;
  ld s = (ld)((s2.x-s1.x) * (s2.y-e2.y) + (s2.y-s1.y) * (e2.x-s2.x)) / det;
  v.x = s1.x + (e1.x-s1.x) * s;
  v.y = s1.y + (e1.y-s1.y) * s;
  return 1;
}

bool bad(Line& a, Line& b, Line& c) {
  Point v;
  if (!line_intersect(a.s, a.t, b.s, b.t, v)) return 0;
  ld crs = (c.t-c.s) ^ (v-c.s);
  return crs < 0 || equals(crs, 0);
}

vector<Point> HPI(vector<Line>& ln) {
  auto lsgn = [&](const Line& a) {
    if(a.s.y == a.t.y) return a.s.x > a.t.x;
```

```
    return a.s.y > a.t.y;
  };
  sort(ln.begin(), ln.end(), [&](const Line& a, const Line& b) {
    if(lsgn(a) != lsgn(b)) return lsgn(a) < lsgn(b);
    return (a.t.x-a.s.x)*(b.t.y-b.s.y)-(a.t.y-a.s.y)*(b.t.x-b.s.x)>0;
  });
  deque<Line> dq;
  for(int i=0; i<ln.size(); i++) {
    while(dq.size() >= 2 && bad(dq[dq.size()-2], dq.back(), ln[i]))
      dq.pop_back();
    while(dq.size() >= 2 && bad(dq[0], dq[1], ln[i]))
      dq.pop_front();
    if(dq.size() < 2 || !bad(dq.back(), ln[i], dq[0]))
      dq.push_back(ln[i]);
  }
  vector<Point> res;
  if(dq.size() >= 3) {
    for(int i=0; i<dq.size(); i++) {
      int j=(i+1)%dq.size();
      Point v;
      if(!line_intersect(dq[i].s, dq[i].t, dq[j].s, dq[j].t, v)) continue;
      res.push_back(v);
    }
  }
  return res;
}
```

FastDelaunay.h

Description: Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], . . .}, all counter-clockwise.
Usage: `vector<P> tris = triangulate(pts);`
Time: $\mathcal{O}(n \log n)$, $\sum n \log n = 1.3 \times 10^7$ in 2500ms. d41d8c, 157 lines

```
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;

template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
  friend istream& operator>>(istream& is, P& p) {
    return is >> p.x >> p.y; }
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};

typedef Point<ll> P;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
  int rot, o; P p = arb; bool mark;
  Quad(): rot(-1), o(-1), mark(false) {}
};

vector<Quad> qs;
int H = -1;

int& r(const Quad& q) { return qs[q.rot].rot; }
P& F(const Quad& q) { return qs[r(q)].p; }
int prev(const Quad& q) { return qs[qs[q.rot].o].rot; }
int next(const Quad& q) { return prev(qs[r(q)]); }

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
    B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
int makeEdge(P orig, P dest) {
  int rr;
  if (H != -1) {
    rr = H;
  } else {
    qs.push_back(Quad());
    int sz = qs.size()-1;
    qs.push_back(Quad());
    qs.back().rot = sz;
    sz = qs.size()-1;
    qs.push_back(Quad());
    qs.back().rot = sz;
    sz = qs.size()-1;
    qs.push_back(Quad());
    qs.back().rot = sz;
    rr = qs.size()-1;
  }
  H = qs[rr].o; r(qs[r(qs[rr])]) = rr;
  rep(i,0,4) rr = qs[rr].rot, qs[rr].p = arb, qs[rr].o = i & 1 ? rr : r(qs[rr]);
  qs[rr].p = orig; F(qs[rr]) = dest;
  return rr;
}

void splice(int a, int b) {
  swap(qs[qs[qs[a].o].rot].o, qs[qs[qs[b].o].rot].o); swap(qs[a].o, qs[b].o);
}

int connect(int a, int b) {
  int q = makeEdge(F(qs[a]), qs[b].p);
  splice(q, next(qs[a]));
  splice(r(qs[q]), b);
  return q;
}

pair<int,int> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    int a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, r(qs[a]) };
    splice(r(qs[a]), b);
    auto side = s[0].cross(s[1], s[2]);
    int c = side ? connect(b, a) : 0;
    return {side < 0 ? r(qs[c]) : a, side < 0 ? c : r(qs[b])};
  }

#define H(e) F(qs[e]), qs[e].p
#define valid(e) (F(qs[e]).cross(H(base)) > 0)
  int A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((qs[B].p.cross(H(A)) < 0 && (A = next(qs[A]))) ||
         (qs[A].p.cross(H(B)) > 0 && (B = qs[r(qs[B])].o)));
  int base = connect(r(qs[B]), A);
  if (qs[A].p == qs[ra].p) ra = r(qs[base]);
  if (qs[B].p == qs[rb].p) rb = base;

  for (;;) {
    int LC = qs[r(qs[base])].o;
    if (valid(LC)) {
      while (circ(F(qs[qs[LC].o]), H(base), F(qs[LC]))) {
        int t = qs[LC].o;
        splice(LC, prev(qs[LC]));
        splice(r(qs[LC]), prev(qs[r(qs[LC])]));
        qs[LC].o = H; H = LC; LC = t;
      }
    }
    int RC = prev(qs[base]);
    if (valid(RC)) {
      while (circ(F(qs[prev(qs[RC])]), H(base), F(qs[RC]))) {
        int t = prev(qs[RC]);
        splice(RC, prev(qs[RC]));
        splice(r(qs[RC]), prev(qs[r(qs[RC])]));
        qs[RC].o = H; H = RC; RC = t;
      }
    }
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, r(qs[base]));
    else
      base = connect(r(qs[base]), r(qs[LC]));
  }
  return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  int e = rec(pts).first;
  vector<int> q = {e};
  int qi = 0;
  while (F(qs[qs[e].o]).cross(F(qs[e]), qs[e].p) < 0) e = qs[e].o;
#define ADD { int c = e; do { qs[c].mark = 1; pts.push_back(qs[c].p);   \
    q.push_back(r(qs[c])); c = next(qs[c]); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!qs[(e = q[qi++])].mark) ADD;
  return pts;
}
```

BulldozerTrick.h

Description: Bulldozer trick. At first points need to be sorted. If collinear point exists, comparison operator must consider original index of point.
Usage: `vector<Line> V;`

Time: $\mathcal{O}\left(n^2 \log n\right)$, no data but relatively fast.
d41d8c, 27 lines

```
struct Line {
  ll i, j, dx, dy; // dx >= 0
Line(int i, int j, const Point &pi, const Point &pj)
  : i(i), j(j), dx(pj.x-pi.x), dy(pj.y-pi.y) {}
  bool operator < (const Line &l) const {
    return make_tuple(dy*l.dx, i, j) < make_tuple(l.dy*dx, l.i,
         l.j); }
  bool operator == (const Line &l) const {
    return dy * l.dx == l.dy * dx;
  }
};
void Solve(){
  sort(A+1, A+N+1); iota(P+1, P+N+1, 1);
  vector<Line> V; V.reserve(N*(N-1)/2);
  for(int i=1; i<=N; i++)
    for(int j=i+1; j<=N; j++)
      V.emplace_back(i, j, A[i], A[j]);
  sort(V.begin(), V.end());
  for(int i=0, j=0; i<V.size(); i=j){
    while(j < V.size() && V[i] == V[j]) j++;
    for(int k=i; k<j; k++){
      int u = V[k].i, v = V[k].j; // point id, index -> Pos[id]
      swap(Pos[u], Pos[v]); swap(A[Pos[u]], A[Pos[v]]);
      if(Pos[u] > Pos[v]) swap(u, v);
      // @TODO
    }
  }
}
```

RotatingCallipers.h

Description: A[0] should be mininum element, A should be a convex polygon and sorted in ccw.
d41d8c, 11 lines

```
ll rotating_calipers(vector<Point> A) {
  int n = A.size(); A.push_back(A[0]);
  int l = 0, r = 0;
  ll ret = 0;
  while(l < n) {
    // A[l], A[r] are antipodal points at this moment
    if(r+1 == l || ((A[l+1]-A[l])^(A[r+1]-A[r])) <= 0) l++;
    else r++;
  }
  return ret;
}
```

DualGraph.h

Description: Return compressed node index of dual graph of each edge's both side. Second index is inf out face of dual graph. Even number is left side of edge oriented to dictionary order increasing.
Time: $\mathcal{O}(n \log n)$
d41d8c, 36 lines

```
constexpr int quadrant_id(const Point p){
  constexpr int arr[9] = { 5, 4, 3, 6, -1, 2, 7, 0, 1 };
  return arr[sign(p.x)*3+sign(p.y)+4];
}
pair<vector<int>, int> dual_graph(const vector<Point> &points,
    const vector<pair<int,int>> &edges){
  int n = points.size(), m = edges.size();
  vector<int> uf(2*m); iota(uf.begin(), uf.end(), 0);
  function<int(int)> find = [&](int v){ return v == uf[v] ? v :
       uf[v] = find(uf[v]); };
  function<bool(int,int)> merge = [&](int u, int v){ return
       find(u) != find(v) && (uf[uf[u]]=uf[v], true); };
  vector<vector<pair<int,int>>> g(n);
  for(int i=0; i<m; i++){
    g[edges[i].first].emplace_back(edges[i].second, i);
    g[edges[i].second].emplace_back(edges[i].first, i);
  }
  for(int i=0; i<n; i++){
    const auto base = points[i];
    sort(g[i].begin(), g[i].end(), [&](auto a, auto b){
        auto p1 = points[a.first] - base, p2 = points[b.first]
          - base;
        return quadrant_id(p1) != quadrant_id(p2) ? quadrant_id
            (p1) < quadrant_id(p2) : (p1 ^ p2) > 0;
    });
    for(int j=0; j<g[i].size(); j++){
      int k = j ? j - 1 : g[i].size() - 1;
      int u = g[i][k].second << 1, v = g[i][j].second << 1 | 1;
      auto p1 = points[g[i][k].first], p2 = points[g[i][j].
          first];
      if(p1 < base) u ^= 1; if(p2 < base) v ^= 1;
      merge(u, v);
    }
  }
  vector<int> res(2*m);
  for(int i=0; i<2*m; i++) res[i] = find(i);
  auto comp = res; sort(comp.begin(), comp.end());
  comp.erase(unique(comp.begin(), comp.end()), comp.end());
  for(auto &i : res) i = lower_bound(comp.begin(), comp.end(),
      i) - comp.begin();
  int mx_idx = max_element(points.begin(), points.end()) -
      points.begin();
  return {res, res[g[mx_idx].back().second << 1 | 1]};
}
```

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points. Expect that point list is shuffled.
Time: expected $\mathcal{O}(n)$
d41d8c, 28 lines

```
struct Circle{ Point p; double r; };

long double dst(Point a, Point b); // root of square dist
Point getCenter(Point a, Point b){ return {(a.x+b.x)/2, (a.y+b.
     y)/2}; } // center of two points
Point getCenter(Point a, Point b, Point c){ // center of
     circumcircle of triangle
  Point aa = b - a, bb = c - a;
  auto c1 = aa*aa * 0.5, c2 = bb*bb * 0.5;
  auto d = aa ^ bb;
  auto x = a.x + (c1 * bb.y - c2 * aa.y) / d;
  auto y = a.y + (c2 * aa.x - c1 * bb.x) / d;
  return {x, y};
}

Circle solve(vector<Point> v){
  Point p = {0, 0};
  double r = 0; int n = v.size();
  for(int i=0; i<n; i++) if(dst(p, v[i]) > r){ //break point 1
      p = v[i]; r = 0;
      for(int j=0; j<i; j++) if(dst(p, v[j]) > r){ //break
           point 2
          p = getCenter(v[i], v[j]); r = dst(p, v[i]);
          for(int k=0; k<j; k++) if(dst(p, v[k]) > r){ //break
               point 3
              p = getCenter(v[i], v[j], v[k]);
              r = dst(v[k], p);
            }
        }
    }
  return {p, r};
}
```

UnionOfCircle.h

Description: Calculate the area of union of circle.
d41d8c, 166 lines

```
inline ld sqr(ld x) {return x*x;}
inline int sgn(ld x) {return abs(x)<1e-19L? 0: x > 0? 1 : -1;}
struct vec2{
  ld x,y;
  vec2(){}
  vec2(ld _x, ld _y): x(_x), y(_y){}
  ld norm()const{return sqrtl(sqr(x)+sqr(y)); }
  ld angle()const{return atan2l(y,x);}

  friend vec2 operator+(vec2 a, vec2 b){return vec2(a.x+b.x, a.
       y+b.y); }
  friend vec2 operator-(vec2 a, vec2 b){return vec2(a.x-b.x, a.
       y-b.y); }
  friend vec2 operator*(vec2 a, ld b){return vec2(a.x*b, a.y*b)
       ; }
  friend vec2 operator / (vec2 a, ld b){return vec2(a.x / b, a.
       y / b); }
  friend bool operator == (vec2 a, vec2 b){return sgn(a.x-b.x)
       == 0 && sgn(a.y-b.y) == 0; }
  friend bool operator < (vec2 a, vec2 b){return sgn(a.x-b.x) <
        0 || (sgn(a.x-b.x) == 0 && sgn(a.y-b.y) < 0); }

  vec2 rotate(vec2 p, ld ang) {
    vec2 v=(*this)-p;
    vec2 ret;
    ret.x=v.x*cosl(ang)-v.y*sinl(ang);
    ret.y=v.y*cosl(ang)+v.x*sinl(ang);
    return ret+p;
  }
};
int circle_inter_circle(vec2 c1, ld r1, vec2 c2, ld r2, vec2 *
    res)
{
  ld d=(c1-c2).norm();
  if (sgn(d) == 0) {
    if (sgn(r1-r2) == 0) return -1;
    return 0;
  }
  if (sgn(r1+r2-d) < 0) return 0;
  if (sgn(fabs(r1-r2)-d) > 0) return 0;

  ld ang=atan2((c2-c1).y, (c2-c1).x);
  ld vang=acosl( (sqr(r1)+sqr(d)-sqr(r2)) / (2*r1*d) );
  res[0]=vec2(c1.x+r1, c1.y).rotate(c1, ang+vang);
  res[1]=vec2(c1.x+r1, c1.y).rotate(c1, ang-vang);
  if (res[0] == res[1]) return 1;
  return 2;
}

struct region{
  ld st, ed;
  region(){}
  region(ld _st, ld _ed): st(_st), ed(_ed) {}
  bool operator < (const region &a)const {
    return sgn(st-a.st) < 0 || (sgn(st-a.st) == 0 && sgn(ed-a.
         ed) < 0);
  }
};

struct Circle{
  vec2 c;
  ld r;
  vector<region> reg;
  Circle(){}
  Circle(vec2 _c, ld _r): c(_c), r(_r) {}
```

```
  void add(const region r={}){reg.emplace_back(r); }
  ld area(ld ang=M_PI){return ang*sqr(r); }
  vec2 makepoint(ld ang){return vec2(c.x+r*cosl(ang) , c.y+r*
       sinl(ang)); }

  bool operator<(const Circle &a)const {
    return sgn(r-a.r) < 0 || (sgn(r-a.r) == 0 && c < a.c);
  }
  bool operator==(const Circle &a)const {
    return sgn(r-a.r) == 0 && c == a.c;
  }
};

ld area_of_circles(Circle *cir, int n){
  bool ok[n+5];
  memset(ok, true, sizeof ok);
  ld ans=0;
  for (int i=0; i < n; i++) {
    for (int j=0; j < n; j++) if (ok[j]) {
      if (i == j) continue;
      ld d=(cir[i].c-cir[j].c).norm();
      if (sgn(d+cir[i].r-cir[j].r) <= 0){//!!!!
        ok[i]=false;
        break;
      }
    }
  }
  for (int i=0; i < n; i++) if (ok[i]) {
    vec2 p[2];
    bool flag=false;
    for (int j=0; j < n; j++) if(ok[j]) {
      if (i == j) continue;
      int k=circle_inter_circle(cir[i].c, cir[i].r, cir[j].c,
           cir[j].r, p);
      if (k != 2) continue;
      flag=true;

      ld ang1=(p[1]-cir[i].c).angle(), ang2=(p[0]-cir[i].c).
           angle();
      if (sgn(ang1) < 0) ang1+=2*M_PI;
      if (sgn(ang2) < 0) ang2+=2*M_PI;
      if (sgn(ang1-ang2) > 0) cir[i].add(region(ang1, 2*M_PI)),
           cir[i].add(region(0, ang2));
      else cir[i].add(region(ang1, ang2));
    }
    if (!flag) {
      ans+=cir[i].area();
      continue;
    }
    sort(cir[i].reg.begin(), cir[i].reg.end());
    int cnt=1;
    for (int j=1; j < int(cir[i].reg.size()); j++) {
      if (sgn(cir[i].reg[cnt-1].ed-cir[i].reg[j].st) >= 0) {
        cir[i].reg[cnt-1].ed=max(cir[i].reg[cnt-1].ed, cir[i].
             reg[j].ed);
      }
      else {
        cir[i].reg[cnt++]=cir[i].reg[j];
      }
    }
    cir[i].add();
    cir[i].reg[cnt]=cir[i].reg[0];
    for (int j=0; j < cnt; j++) {
      p[0]=cir[i].makepoint(cir[i].reg[j].ed);
      p[1]=cir[i].makepoint(cir[i].reg[j+1].st);
      ans+=(p[0].x*p[1].y-p[1].x*p[0].y)/2.L;
      ld ang=cir[i].reg[j+1].st-cir[i].reg[j].ed;
      if (sgn(ang) < 0) ang+=2*M_PI;
      ans+=0.5*sqr(cir[i].r)*(ang-sinl(ang));
    }
  }
  return ans;
}

ld total_area(vector<ld> cx,vector<ld> cy,vector<ld> cr){
  size_t const n=cx.size();
  vector<Circle> c(n);
  for (int i=0; i<n; i++){
    c[i].c.x=cx[i];
    c[i].c.y=cy[i];
    c[i].r=cr[i];
  }
  return area_of_circles(&c[0],n);
}

int main(){
  ld angle=0.45692586256L;
  int n; ld area; cin>>n>>area;
  vector<ld> cx(n),cy(n),cr(n);
  for (int i=n; i--;){
    ld x,y,r; cin>>x>>y>>r;
    cx[i]=cosl(angle)*x+sinl(angle)*y;
    cy[i]=cosl(angle)*y-sinl(angle)*x;
    cr[i]=r;
  }

  ld min_rad=cr[0];
  for (auto const &i: cr) min_rad=min(min_rad,i);
  ld lef=min_rad;
  ld rgt=min_rad+sqrtl(area/M_PI);
  for (int iter=200; iter--;){
    ld const mid=(lef+rgt)/2.L;
    vector<ld> tcr=cr;
    for (int i=0; i<n; i++) tcr[i]=max(mid-cr[i],0.L);
    ld const total=total_area(cx,cy,tcr);
    if (total<area){
      lef=mid;
    }else{
      rgt=mid;
    }
  }
  cout<<lef<<endl;
}
```

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.
Usage: `vector<P> v = {P{4,4}, P{1,2}, P{2,1}};`
`bool in = inPolygon(v, P{3, 3}, false);`
Time: $\mathcal{O}(n)$

d41d8c, 15 lines

```
// True when p is endpoint
bool onSegment(Point s, Point e, Point p) {
  return ((s-p) ^ (e-p)) == 0 && (s - p) * (e - p) <= 0;
}

bool inPolygon(vector<Point> &p, Point a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    Point q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * ((p[i]-a) ^ (q-a)) > 0;
  }
  return cnt;
}
```

PolygonCut.h

Description: Returns the polygon on the left of line l.

d41d8c, 13 lines

```
// l = p + d*t, l.q() = l + d
// doubled_signed_area(p,q,r) = (q-p) ^ (r-p)
vector<Point> polygon_cut(const vector<Point> &a, const line<T>
     &l){
  vector<Point> res;
  for(auto i = 0; i < (int)a.size(); ++ i){
    auto cur = a[i], prev = i ? a[i - 1] : a.back();
    bool side = doubled_signed_area(l.p, l.q(), cur) > 0;
    if(side != (doubled_signed_area(l.p, l.q(), prev) > 0))
      res.push_back(l.p + ((cur - l.p) ^ (prev - cur)) / ((l.d
           ^ prev) - cur) * l.d); // line intersection
    if(side) res.push_back(cur);
  }
  return res;
}
```

PolygonUnion.h

Description: Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.) Cross exists on FastDelaunay, sideOf exists in InsidePolygon or somewhere else.
Time: $\mathcal{O}\left(N^2\right)$, where N is the total number of points

"Point.h", "sideOf.h" d41d8c, 33 lines

```
typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
  double ret = 0;
  rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
    P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
    vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
    rep(j,0,sz(poly)) if (i != j) {
      rep(u,0,sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
        if (sc != sd) {
          double sa = C.cross(D, A), sb = C.cross(D, B);
          if (min(sc, sd) < 0)
            segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
        } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0){
          segs.emplace_back(rat(C - A, B - A), 1);
          segs.emplace_back(rat(D - A, B - A), -1);
        }
      }
    }
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j - 1].first;
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  }
  return ret / 2;
}
```

ConvexHull.h

Description: collinear points removal is optional. Return points are sorted in ccw.
Time: $\mathcal{O}(n \log n)$

d41d8c, 12 lines

```
vector <Point> convex_hull(vector <Point> P){ // collinear
     points are not removed
  vector <Point> up, down;
```

```
  sort(P.begin(), P.end());
  for (Point now : P){
    while (up.size() >= 2 && ((up.back() - up[up.size() - 2])^(
         now - up[up.size() - 2])) > 0) up.pop_back(); // >=0 :
         remove collinear points
    up.push_back(now);
    while (down.size() >= 2 && ((down.back() - down[down.size()
          - 2])^(now - down[down.size() - 2])) < 0) down.
         pop_back(); // <=0 : remove collinear points
    down.push_back(now);
  }
  down.insert(down.end(), up.rbegin()+1, up.rend()-1);
  return down;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Usage: `bool left = sideOf(p1,p2,q)==1;`
Time: $\mathcal{O}(\log N)$

d41d8c, 19 lines

```
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}

bool inHull(const vector<P>& l, P p, bool strict = true) {
  int a = 1, b = sz(l) - 1, r = !strict;
  if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
  if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
  if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
```

kdTree.h

Description: Unknown usage

d41d8c, 63 lines

```
T GetDist(const P &a, const P &b){ return (a.x-b.x) * (a.x-b.x)
      + (a.y-b.y) * (a.y-b.y); }
struct Node{
  P p; int idx;
  T x1, y1, x2, y2;
  Node(const P &p, const int idx) : p(p), idx(idx), x1(1e9), y1
       (1e9), x2(-1e9), y2(-1e9) {}
  bool contain(const P &pt)const{ return x1 <= pt.x && pt.x <=
       x2 && y1 <= pt.y && pt.y <= y2; }
  T dist(const P &pt) const { return idx == -1 ? INF : GetDist(
       p, pt); }
  T dist_to_border(const P &pt) const {
    const auto [x,y] = pt;
    if(x1 <= x && x <= x2) return min((y-y1)*(y-y1), (y2-y)*(y2
         -y));
    if(y1 <= y && y <= y2) return min((x-x1)*(x-x1), (x2-x)*(x2
         -x));
    T t11 = GetDist(pt, {x1,y1}), t12 = GetDist(pt, {x1,y2});
    T t21 = GetDist(pt, {x2,y1}), t22 = GetDist(pt, {x2,y2});
    return min({t11, t12, t21, t22});
  }
};
template<bool IsFirst = 1> struct Cmp {
  bool operator() (const Node &a, const Node &b) const {
    return IsFirst ? a.p.x < b.p.x : a.p.y < b.p.y;
  }
};
struct KDTree { // Warning : no duplicate
  constexpr static size_t NAIVE_THRESHOLD = 16;
  vector<Node> tree;
  KDTree() = default;
  explicit KDTree(const vector<P> &v) {
    for(int i=0; i<v.size(); i++) tree.emplace_back(v[i], i);
         Build(0, v.size());
  }
  template<bool IsFirst = 1>
  void Build(int l, int r) {
    if(r - l <= NAIVE_THRESHOLD) return;
    const int m = (l + r) >> 1;
    nth_element(tree.begin()+l, tree.begin()+m, tree.begin()+r,
          Cmp<IsFirst>{});
    for(int i=l; i<r; i++){
      tree[m].x1 = min(tree[m].x1, tree[i].p.x); tree[m].y1 =
           min(tree[m].y1, tree[i].p.y);
      tree[m].x2 = max(tree[m].x2, tree[i].p.x); tree[m].y2 =
           max(tree[m].y2, tree[i].p.y);
    }
    Build<!IsFirst>(l, m); Build<!IsFirst>(m + 1, r);
  }
  template<bool IsFirst = 1>
  void Query(const P &p, int l, int r, Node &res) const {
    if(r - l <= NAIVE_THRESHOLD){
      for(int i=l; i<r; i++) if(p != tree[i].p && res.dist(p) >
           tree[i].dist(p)) res = tree[i];
    }
    else{
      const int m = (l + r) >> 1;
      const T t = IsFirst ? p.x - tree[m].p.x : p.y - tree[m].p
           .y;
      if(p != tree[m].p && res.dist(p) > tree[m].dist(p)) res =
           tree[m];
      if(!tree[m].contain(p) && tree[m].dist_to_border(p) >=
           res.dist(p)) return;
      if(t < 0){
        Query<!IsFirst>(p, l, m, res);
        if(t*t < res.dist(p)) Query<!IsFirst>(p, m+1, r, res);
      }
      else{
        Query<!IsFirst>(p, m+1, r, res);
        if(t*t < res.dist(p)) Query<!IsFirst>(p, l, m, res);
      }
    }
  }
  int Query(const P& p) const {
    Node ret(make_pair<T>(1e9, 1e9), -1); Query(p, 0, tree.size
         (), ret); return ret.idx;
  }
};
```

Various (7)

7.1 Structs

Fraction.h

d41d8c, 19 lines

```
struct Fraction {
  __int128 a, b;
  Fraction() {}
Fraction(__int128 _a, __int128 _b): a(_a), b(_b) {
  if (b < 0) a = -a, b = -b;
  __int128 d = gcd(a, b);
  a /= d, b /= d;
}
  bool operator==(const Fraction& r) const { return a * r.b ==
       b * r.a; }
  bool operator<(const Fraction& r) const { return a * r.b < b
       * r.a; }
  bool operator>(const Fraction& r) const { return a * r.b > b
       * r.a; }
  bool operator>=(const Fraction& r) const { return a * r.b >=
       b * r.a; }
  Fraction operator*(const Fraction& x) const { return Fraction
       (a*x.a, b*x.b); }
  Fraction operator-() const { return Fraction{-a, b}; }
  Fraction operator+(const Fraction& r) const { return Fraction
       (a*r.b+b*r.a, b*r.b); }
  Fraction operator-(const Fraction& r) const { return Fraction
       (a*r.b-b*r.a, b*r.b); }
};

ostream& operator<<(ostream& os, Fraction& x) { os << '(' << (
    ll)x.a << ' ' << (ll)x.b << ')'; return os; }
```

7.2 Skills

Random.h

d41d8c, 11 lines

```
mt19937 rng(1010101);
lint randInt(lint l, lint r) {
    return uniform_int_distribution<lint>(l, r)(rng);
}

import random
random.randrange(s, e)  # random integer from [s, e)
random.random()         # random float from [0, 1)
random.uniform(a, b)    # random float from [a, b]
random.shuffle(list)    # shuffle list
random.sample(list, n)  # sampling without replacement
```

Getline.h

d41d8c, 5 lines

```
int n;
string str;
cin >> n;
cin.ignore();
getline(cin, str);
```

Stress.h

d41d8c, 9 lines

```
import os
while True:
    with open("input.txt", "w") as f:
        # Generate Data
        pass

    os.system("source.exe < input.txt > output.txt")
    if os.system("checker.exe < input.txt") != 0:
        break
```

7.3 Some primes for NTT, hashing

$998244353 = 119 \times 2^{23} + 1$, Primitive root $= 3$
$985661441 = 235 \times 2^{22} + 1$, Primitive root $= 3$
$1012924417 = 483 \times 2^{21} + 1$, Primitive root $= 5$

7.4 Tricks

BitsOperation.h

d41d8c, 12 lines

```
int __builtin_clz(int x); // Number of leading zeros 0010 = 2
int __builtin_ctz(int x); // Number of trailing zeros 0010 = 1
```

```
int __builtin_popcount(int x); // Number of 1-bits in x 01011 =
      3

int lsb(int n) { return n & -n; } // Smallest bit
int remove_lsb(int n) { return n & (n - 1); } // n - lsb(n)

// Subset iteration, used in O(3^n) dp
for (int i = x; ; i = (i - 1) & x) {
    // i is a subset of x, decreasing in terms of integer value
  if (i == 0) break;
}
```

FloorLoop.h d41d8c, 7 lines

```
// floor(n / 1), floor(n / 2), ... has at most 2 * sqrt(n)
     different values
for (int l = 1; l <= n; ) {
  int q = n / l;
  int r = n / q;
  // floor(n / x) = q for x in [l, r]
  l = r + 1;
}
```

7.5 Optimization

FastIO.h d41d8c, 66 lines

```
namespace fio {
  const int BSIZE = 1<<18;
  char buffer[BSIZE];
  char wbuffer[BSIZE];
  char ss[30];
  int pos = BSIZE;
  int wpos = 0;
  int cnt = 0;

  inline char readChar() {
    if (pos == BSIZE) {
      fread(buffer, 1, BSIZE, stdin);
      pos = 0;
    }
    return buffer[pos++];
  }

  inline int readInt() {
    char c = readChar();
    while ((c < '0' || c > '9') && (c ^ '-')) c = readChar();

    int res = 0;
    bool neg = (c == '-');
    if (neg) c = readChar();
    while (c > 47 && c < 58) {
      res = res * 10 + c - '0';
      c = readChar();
    }

    if (neg) return -res;
    else return res;
  }

  inline void writeChar(char x){
    if (wpos == BSIZE) {
      fwrite(wbuffer, 1, BSIZE, stdout);
      wpos = 0;
    }
    wbuffer[wpos++] = x;
  }

  inline void writeInt(int x){
    if (x < 0) {
      writeChar('-');
      x = -x;
    }
    if (!x) {
      writeChar('0');
    } else {
      cnt = 0;
      while (x) {
        ss[cnt] = (x % 10) + '0';
        cnt++;
        x /= 10;
      }
      for (int j=cnt-1; j>=0; --j) writeChar(ss[j]);
    }
  }

  inline void my_flush() {
    if (wpos) {
      fwrite(wbuffer, 1, wpos, stdout);
      wpos = 0;
    }
  }
}
```

FastMod.h d41d8c, 18 lines

```
typedef unsigned long long ull;
typedef __uint128_t L;
struct FastMod {
  ull b, m;
FastMod(ull b) : b(b), m(ull((L(1) << 64) / b)) {}
  ull reduce(ull a) {
    ull q = (ull)((L(m) * a) >> 64);
    ull r = a - q * b; // can be proven that 0<= r < 2*b
    return r >= b ? r - b : r;
  }
};
FastMod F(2);

int main() {
  int M = 1000000007; F = FastMod(M);
  ull x = 10ULL*M+3;
  cout << x << " " << F.reduce(x) << "\n"; // 10000000073 3
}
```

7.6 Dynamic programming

KnuthDP.h

Description: When doing DP on intervals: $a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i,j)$, where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b,c) \le f(a,d)$ and $f(a,c) + f(b,d) \le f(a,d) + f(b,c)$ for all $a \le b \le c \le d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
Time: $\mathcal{O}(N^2)$

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i) \le k < hi(i)}(f(i,k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.
Time: $\mathcal{O}((N + (hi - lo)) \log N)$ d41d8c, 18 lines

```
struct DP { // Modify at will:
  int lo(int ind) { return 0; }
  int hi(int ind) { return ind; }
  ll f(int ind, int k) { return dp[ind][k]; }
  void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

  void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<ll, int> best(LLONG_MAX, LO);
    rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
      best = min(best, make_pair(f(mid, k), k));
    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second+1);
    rec(mid+1, R, best.second, HI);
  }
  void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

7.7 Graph Matching Application about cubic time

- **Game on a Graph**: There is a token on vertex s. Each player moves the token to an adjacent vertex on their turn and loses if they cannot move.
  There exists a maximum matching that does not include s $\leftrightarrow$ the second player to move wins.

- **Chinese Postman Problem**: The problem of finding the minimum weight circuit that visits every edge. Revisiting node and edges is allowed.
  Run Floyd's algorithm, then pair up odd-degree vertices to find the minimum weight matching (there are an even number of odd-degree vertices).

- **Unweighted Edge Cover**: The problem of finding the smallest set of edges (minimum cardinality/weight) that covers all vertices.
  $|V| - |M|$, no path of length 3, composed of several star graphs.

- **Weighted Edge Cover**: $sum_{v \in V}(w(v)) - sum_{(u,v) \in M}(w(u) + w(v) - d(u,v))$, where $w(x)$ is the minimum weight of an edge adjacent to x.

- **NEERC'18 B**: A problem where each machine must be operated by two workers, and each worker can operate at most one machine. Maximize the number of machines which works.
  Create two vertices for each machine and connect them with an edge, the solution is $|M| - |\text{machines}|$. It's good to consider that each contributes 1/2 to the solution.

- **Min Disjoint Cycle Cover**: The problem of finding a set of cycles of length 3 or more that covers all vertices without overlap.
  Each vertex must be matched with two distinct edges, and some edges must be matched with both endpoints, so we can think of flow, but flow is not possible since only one unit of flow can be sent through an edge with capacity 2.

Copy each vertex and edge into two ((v, v'), $(e_{i,u}, e_{i,v})$). For every edge $e = (u, v)$, create a weighted edge of weight w connecting e_u and e_v (like NEERC18), and create weight 0 edges connecting $(u, e_{i,u}), (u', e_{i,u}), (v, e_{i,v}), (v', e_{i,v})$. The existence of a perfect matching $\leftrightarrow$ existence of a Disjoint Cycle Cover. Find the maximum weight matching and then subtract the matching from the sum of all edge weights.

- **Two Matching**: The problem of finding the maximum weight matching where each vertex is adjacent to at most two edges.
Each component must become a single vertex/path/cycle. Create a weight 0 edge for every distinct pair of vertices, and a weight 0 edge (v, v') turns it into a Disjoint Cycle Cover problem. Components with only one vertex are self-loops, and it's convenient to think of path-shaped components as having their endpoints connected.

- **Parith Shortest Simple Path**: Without loss of generality, let's say we're looking for a path of even length (the same method applies for odd length). We create a copy G' of graph G, but instead of connecting opposite sides, for every edge $e = \{u, v\}$, we connect $G(u) \leftrightarrow G(v)$ and $G'(u) \leftrightarrow G'(v)$. Finally, for every vertex $v \in V$, we connect the vertices $G(v) \leftrightarrow G'(v)$. In this graph, if we find a perfect matching, the vertices of the original graph G will either be covered by the edges connecting $G(v) \leftrightarrow G'(v)$ or be part of even cycles that alternate between G and G'. Now, let's remove $G(s), G'(t)$ and find a perfect matching. Start from the matched vertex v of $G'(s)$, switch sides, and continue following the matched vertices in the same manner. During this process, we will never encounter edges connecting $G(v) \leftrightarrow G'(v)$. This process will not end until it reaches t, and it is guaranteed to terminate. Listing the vertices we encounter along the path gives us a path with an even number of edges. This way, we can confirm the existence of a Parity shortest path between s and t. To find the shortest path, set the weight of the edges connecting $G(v) \leftrightarrow G'(v)$ to 0, and use the original weights for the other edges.

- **Planar Graph Max Cut**: Given a graph, let's divide the set of vertices into two non-empty subsets (S, V§). The goal is to maximize the number of edges between the two subsets. (In contrast, Global Min Cut minimizes this number.)
The case where the answer to the Global Min Cut is 0 is when the graph is not connected, but the case where the answer to the Global Max Cut is the total number of edges is when the graph is bipartite. In other words, Max Cut is the problem of removing the minimum number of edges to make the graph bipartite.
What properties does a Planar Graph have if it's bipartite? Planar graphs have a very useful concept called the Dual. The Dual of a planar graph has the same set of edges, but the set of vertices is replaced with faces. Here, each face is a cycle, and since a bipartite graph has no odd cycles, each cycle is adjacent to an even number of edges. In other words, if a Planar Graph is bipartite, then every vertex in its Dual has even degree. Another way to say this is that an Euler circuit exists.
At this point, the problem looks almost identical to the Chinese postperson Problem, but there is a slight difference. In the Chinese postperson problem, existing edges are duplicated, while here, the operation is to compress the edge separating two faces, effectively merging them. Even if we think of it as an operation that merges two vertices, by finding a minimum weight matching for the Dual graph in the same way as the postperson problem and then combining the edges that entered the matching, we have a possible solution. If two vertices of degrees x and y are merged, the new vertex has degree $x + y - 2$, and this operation results in two odd-degree vertices matched together being combined into one vertex. Now we need to show that this is the optimal solution, and we can do so using the same proof technique used in problem 1.

7.8 Something Else

Suurballe's algorithm: finding two disjoint paths in a nonnegatively-weighted directed graph so that both paths connect the same pair of vertices and have minimum total length.

1. Find the shortest path tree T rooted at node s by running Dijkstra's algorithm. This tree contains for every vertex u, a shortest path from s to u. Let P_1 be the shortest cost path from s to t. The edges in T are called tree edges and the remaining edges (the edges missing from figure C) are called non-tree edges. 2. Modify the cost of each edge in the graph by replacing the cost $w(u, v)$ of every edge (u, v) by $w'(u, v) = w(u, v) - d(s, v) + d(s, u)$. According to the resulting modified cost function, all tree edges have a cost of 0, and non-tree edges have a non-negative cost. 3. Create a residual graph G_t formed from G by removing the edges of G on path P_1 that are directed into s and then reverse the direction of the zero length edges along path P_1. 4. Find the shortest path P_2 in the residual graph G_t by running Dijkstra's algorithm. 5. Discard the reversed edges of P_2 from both paths. The remaining edges of P_1 and P_2 form a subgraph with two outgoing edges at s, two incoming edges at t, and one incoming and one outgoing edge at each remaining vertex. Therefore, this subgraph consists of two edge-disjoint paths from s to t and possibly some additional (zero-length) cycles. Return the two disjoint paths from the subgraph.

7.9 Dominator Tree

Definition 1. 방향 그래프 $G = (V, E)$와, V의 모든 정점을 도달할 수 있는 정점 $s \in V$가 주어졌을 때, (G, s)에서 정점 u가 정점 v의 "Dominator"라는 것은, $u \neq v$이며 s에서 v로 가는 모든 경로가 u를 거침을 뜻한다.

"V의 모든 정점을 도달할 수 있는 정점 s" 에 대해서만 정의되는 게 껄끄러울 수는 있으나, 실제로는 문제 상황상 s에서 도달할 수 없는 정점에 신경 쓸 이유가 존재하지 않는다. 고로 어떠한 정점이 s에서 도달할 수 없다면, 그 정점을 그래프에서 지움으로써 정의에 맞게 G를 바꿔줄 수 있다.

Definition 2. (G, s)에서 정점 u가 정점 v의 "Immediate Dominator"이라는 것은, 정점 u가 정점 v의 Dominator이며, 정점 v의 모든 다른 Dominator들은 u의 Dominator임을 뜻한다.

Theorem 1. (G, s) 가 주어졌을 때, s를 제외한 모든 정점 v 에 대해서 Immediate Dominator $idom(v)$ 가 정확히 하나 존재한다.

Proof. 정의에 의해 s는 s를 제외한 모든 정점의 Dominator이기 때문에, s를 제외한 모든 정점에 대해서 최소 하나의 Dominator가 존재한다. 어떠한 정점 $x \neq s$의 Dominator의 집합을 $D(x)$ 라 하자. 정점 u가 정점 v의 Dominator이거나 $u = v$일 때 $u \leq v$라는 순서가 성립한다고 하면, 다음과 같은 방식으로 해당 순서가 $D(x)$ 에 대해 Total ordering임을 증명할 수 있다.

- u가 v의 Dominator이면서 v가 u의 Dominator일 수 없다. v를 통하는 어떠한 단순 경로는 u를 거치는데, 이 경로를 u에서 자르면 v를 거치지 않고 u를 도달하는 경로를 찾을 수 있기 때문이다. 대우 명제에 따라서, $u \leq v$이면서 $v \leq u$이기 위해서는 $u = v$여야 한다.

- u가 v의 Dominator이면서 v가 w의 Dominator이면, u는 w의 Dominator이다. 즉, $u \leq v$이면서 $v \leq w$ 이면 $u \leq w$이다.

- $u \neq v$일 때, u가 v의 Dominator이거나 v가 u의 Dominator이다. s에서 x를 지나는 모든 경로는 u와 v를 모두 포함한다. 둘이 Dominator 관계가 아니라면, u 다음에 v를 지나는 경로 P_1과 v 다음에 u를 지나는 경로 P_2가 존재한다. 이들을 적당히 잘라서 u를 지나지 않는 경로 $s - x$를 찾을 수 있다. 고로 가정에 모순이다.

Total ordering에서는 최댓값이 유일하게 존재한다. 이 최댓값은 v의 Immediate Dominator와 일대일 대응한다. □

Definition 3. s가 아닌 모든 정점 v에 대해서, $(idom(v), v)$를 이은 간선 집합으로 이루어진 그래프를 Dominator Tree라고 한다.

Theorem 2. Dominator Tree는 s를 루트로 하는 트리이다.

Proof. Dominator 관계는 Total ordering이기 때문에 사이클이 없다. 사이클이 없고 $|E| = |V| - 1$인 그래프는 트리이다. s를 루트로 할 시, $idom(v)$는 v의 부모가 된다. 고로 기분이 좋다. □

7.10 Continue

Theorem 3. 어떠한 정점 u가 v의 Dominator인 것과, Dominator Tree 상에서 u가 v의 조상인 것이 동치이다.

Proof. u가 v의 조상이면, v의 Dominator의 Dominator.. 가 u이기 때문에 자명히 u는 v의 Dominator이다. 어떠한 정점 u가 v의 Dominator이면, Immediate Dominator의 정의에 의해서 u는 $idom(v)$이거나, $idom(v)$의 Dominator이다. 수학적 귀납법을 사용하면, u는 $\{idom(v), idom(idom(v)), idom(idom(idom(v))), \cdots, s\}$ 중 하나임을 알 수 있다. 고로, u는 v의 조상이다. □

무향 그래프의 절선 / 절점 개념과 비교할 시, 조금 더 정의가 복잡하며 그 성질들도 자명하게 보이지 않는다. 하지만, 트리 관계를 통해서 그래프의 경로 존재 여부를 단순화할 수 있다는 것은 의미가 크다. 앞서 예로 든 Flow Graph 문제를 예시로 하면, "A 코드가 실행될 때 B 코드는 이미 실행되었을까?" 와 같은 질의는, Dominator Tree 상에서 어떠한 정점이 다른 정점의 조상인지를 판별하는 것과 동치이며, 트리의 깊이 우선 탐색을 통해서 초기 전처리 후 $O(1)$ 시간에 해결할 수 있다. 그 외에도 방향 그래프로 모델링 되는 다양한 문제들은, Dominator Tree를 통해서 더 쉬운 문제로 간소화 시킬 수 있다.

7.11 Tourist

7.12 Tourist