.

Descongelen a Victor Moreno

Contents

1 Estructuras de Datos

1.1 Unordered Map

```cpp
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

gp_hash_table<int, int,custom_hash> m1;

//Funcion count
m1.find(x)!=m1.end()
```

1.2 Segment tree Recursivo

```cpp
const int N=4e5+5;
int st[N], arr[N];
void build(int l, int r, int i){
    if(l==r){st[i]=arr[l]; return;}
    int m=l+r>>1;
    build(l,m,2*i+1); build(m+1,r,2*i+2);
    st[i]=st[2*i+1]+st[2*i+2];
}
void update(int l, int r, int idx, int x, int i){
    if(l==r) {st[i]+=x; return;}
    int m=l+r>>1;
    if(idx<=m) update(l,m,idx,x,i*2+1);
    else update(m+1,r,idx,x,i*2+2);
    st[i]=st[i*2+1]+st[i*2+2];
}
int query(int l, int r, int a, int b, int i){
    if(a>r||b<l) return 0;
    if(a<=l&&r<=b) return st[i];
    int m=l+r>>1;
    return query(l,m,a,b,2*i+1)+query(m+1,r,a,b,2*i+2);
}
```

1.3 Segment Tree Iterativo

```cpp
//Para procesar querys de tipo k-esimo es necesario crear un arbol binario perfector(llenar con 0's)
template<typename T>
struct SegmentTree{
  int N;
  vector<T> ST;

  //Creacion a partir de un arreglo O(n)
  SegmentTree(int N, vector<T> & arr): N(N){
    ST.resize(N << 1);
    for(int i = 0; i < N; ++i)
      ST[N + i] = arr[i];     //Dato normal
      ST[N + i] = creaNodo(); //Dato compuesto
    for(int i = N - 1; i > 0; --i)
      ST[i] = ST[i << 1] + ST[i << 1 | 1];        //Dato normal
      ST[i] = merge(ST[i << 1] , ST[i << 1 | 1]); //Dato compuesto
  }

  //Actualizacion de un elemento en la posicion i
  void update(int i, T value){
    ST[i += N] = value;      //Dato normal
    ST[i += N] = creaNodo();//Dato compuesto
    while(i >>= 1)
      ST[i] = ST[i << 1] + ST[i << 1 | 1];        //Dato normal
      ST[i] = merge(ST[i << 1] , ST[i << 1 | 1]); //Dato compuesto
  }

  //query en [l, r]
  T query(int l, int r){
    T res = 0;  //Dato normal
    nodo resl = creaNodo(), resr = creaNodo();//Dato compuesto
    for(l += N, r += N; l <= r; l >>= 1, r >>= 1){
      if(l & 1)       res += ST[l++]; //Dato normal
      if(!(r & 1))    res += ST[r--]; //Dato normal

      if(l & 1)       resl = merge(resl,ST[l++]); //Dato compuesto
      if(!(r & 1))    resr = merge(ST[r--],resr); //Dato compuesto
    }
    return res;                 //Dato normal
    return merge(resl,resr);    //Dato compuesto
  }

  //Para estas querys es necesario que el st tenga el tam de la siguiente potencia de 2
  //ll nT = 1;
  // while(nT<n) nT<<=1;
  //vector<int> a(nT,0);

  //Encontrar k-esimo 1 en un st de 1's
  int Kth_One(int k) {
    int i = 0, s = N >> 1;
    for(int p = 2; p < 2 * N; p <<= 1, s >>= 1) {
      if(k < ST[p]) continue;
      k -= ST[p++]; i += s;
    }
    return i;
  }

  //i del primer elemento >= k en todo el arr
  int atLeastX(int k){
    int i = 0, s = N >> 1;
    for(int p = 2; p < 2 * N; p <<= 1, s >>= 1) {
      if(ST[p] < k) p++, i += s;
    }
    if(ST[N + i] < k) i = -1;
    return i;
  }

  //i del primer elemento >= k en [l,fin]
  //Uso atLeastX(k,l,1,nT)
  int atLeastX(int x, int l, int p, int s) {
    if(ST[p] < x or s <= l) return -1;
    if((p << 1) >= 2 * N)
      return (ST[p] >= x) - 1;
    int i = atLeastX(x, l, p << 1, s >> 1);
    if(i != -1) return i;
    i = atLeastX(x, l - (s >> 1), p << 1 | 1, s >> 1);
    if(i == -1) return -1;
    return (s >> 1) + i;
  }
};
```

1.4 Segment Tree Lazy Recursivo

```
const int N=2e5+10;
ll st[4*N+10],lazy[4*N+10],arr[N];
void build(int l, int r, int i){
    lazy[i]=0;
    if(l==r){st[i]=arr[l];return;}
    int m=(l+r)>>1;
    build(l,m,2*i+1);
    build(m+1,r,2*i+2);
    st[i]=st[2*i+1]+st[2*i+2];
}
void push(int l, int r, int i){
    if(!lazy[i])return;
    st[i]+=(r-l+1)*lazy[i];
    if(l!=r){
        lazy[2*i+1]+=lazy[i];
        lazy[2*i+2]+=lazy[i];
    }
    lazy[i]=0;
}
void update(int l, int r, int a, int b, ll x, int i){
    push(l,r,i);
    if(a>r||b<l)return;
    if(a<=l&&r<=b){
        lazy[i]+=x;
        push(l,r,i);
        return;
    }
    int m=(l+r)>>1;
    update(l,m,a,b,x,2*i+1);update(m+1,r,a,b,x,2*i+2);
    st[i]=st[2*i+1]+st[2*i+2];
}
ll query(int l, int r, int a, int b, int i){
    if(a>r||b<l)return 0;
    push(l,r,i);
    if(a<=l&&r<=b) return st[i];
    int m=(l+r)>>1;
    return query(l,m,a,b,2*i+1)+query(m+1,r,a,b,2*i+2);
}
```

1.5 Segment Tree Lazy Iterativo

```
//Lazy propagacion con incremento de u en rango y minimo
//Hay varias modificaciones necesarias para suma en ambos
template<typename T>
struct SegmentTreeLazy{
  int N,h;
  vector<T> ST, d;

  //Creacion a partir de un arreglo
  SegmentTreeLazy(int n, vector<T> &a): N(n){
    //En caso de inicializar en cero o algo similar, revisar que la construccion tenga su respectivo neutro mult y 1
    ST.resize(N << 1);
    d.resize(N);
    h = 64 - __builtin_clzll(n);

    for(int i = 0; i < N; ++i)
      ST[N + i] = a[i];
    //Construir el st sobre la query que se necesita
    for(int i = N - 1; i > 0; --i)
      ST[i] = min(ST[i << 1] , ST[i << 1 | 1]);
  }

  //Modificar de acuerdo al tipo modificacion requerida, +,*,|,^,etc
  void apply(int p, T value) {
    ST[p] += value;
    if(p<N) d[p]+= value;
  }

  // Modifica valores de los padres de p
  //Modificar de acuerdo al tipo modificacion requerida, +,*,|,^,etc y a la respectiva query
  void build(int p){
    while(p>1){
      p >>= 1;
      ST[p] = min(ST[p << 1], ST[p << 1 | 1]) + d[p];
      //ST[p] = (ST[p << 1] & ST[p << 1 | 1]) | d[p]; Ejemplos con bitwise
    }
  }

  // Propagacion desde la raiz a p
  void push(int p){
    for (int s = h; s > 0; --s) {
      int i = p >> s;
      if (d[i] != 0) {
        apply(i << 1, d[i]);
        apply(i << 1 | 1, d[i]);
        d[i] = 0;   //Tener cuidado si estoy haciendo multiplicaciones
      }
    }
  }

  // Sumar v a cada elemento en el intervalo [l, r)
  void increment(int l, int r, T value) {
    l += N, r += N;
    int l0 = l, r0 = r;
    for (; l < r; l >>= 1, r >>= 1) {
      if(l & 1) apply(l++, value);
      if(r & 1) apply(--r, value);
    }
    build(l0);
    build(r0 - 1);
  }

  // min en el intervalo [l, r)
  T range_min(int l, int r) {
    l += N, r += N;
    push(l);
    push(r - 1);
    T res = LLONG_MAX;
    //T res = (1 << 30) - 1;    Requerir operacion and
    for (; l < r; l >>= 1, r >>= 1) {
      if(l & 1) res = min(res, ST[l++]);
      //if(res >= mod) res -= mod;
      if(r & 1) res = min(res, ST[--r]);
      //if(res >= mod) res -= mod;
    }
    return res;
  }

};
```

1.6 Rope

```
#include <ext/rope>
using namespace __gnu_cxx;
rope<int> s;
// Sequence with O(log(n)) random access, insert, erase at any position
// s.push_back(x);
// s.insert(i,r) // insert rope r at position i
// s.erase(i,k) // erase subsequence [i,i+k)
// s.substr(i,k) // return new rope corresponding to subsequence [i,i+k)
// s[i] // access ith element (cannot modify)
// s.mutable_reference_at(i) // acces ith element (allows modification)
// s.begin() and s.end() are const iterators (use mutable_begin(), mutable_end() to allow modification)
```

1.7 Ordered Set

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag, tree_order_statistics_node_update> ordered_set;
// find_by_order(i) -> iterator to ith element
// order_of_key(k) -> position (int) of lower_bound of k
```

1.8 Union Find

```
vector<pair<int,int>>ds(MAX,{-1,0});
// Solo siu requeires los elementos del union find, utiliza
// dsext en caso contrario borrarlo
list<int>dsext[MAX];
void init(int n){
    for(int i=0;i<n;i++)dsext[i].push_back(i);
}
int find(int x){
    if(-1==ds[x].first) return x;
    return ds[x].first=find(ds[x].first);
}
bool unionDs(int x, int y){
    int px=find(x),py=find(y);
    int &rx=ds[px].second,&ry=ds[py].second;
    if(px==py) return false;
    else{
        if(rx>ry){
            ds[py].first=px;
        }
        else{
            ds[px].first=py;
            if(rx==ry) ry+=1;
        }
    }
    return true;
}
```

1.9 Segment Tree Persistente

```
#define inf INT_MAX
const int MAX=5e5+2;
typedef pair<ll, ll> item;
struct node{
    item val;
    node *l, *r;
    node(): l(nullptr),r(nullptr),val({inf,inf}){};
    node(node *_l,node *_r):l(_l),r(_r){
        val=min(l->val,r->val);
    }
    node(ll value,ll pos):r(nullptr),l(nullptr){
        val=make_pair(value,pos);
    }
};
pair<ll,ll>all;
vector<node*>versions(MAX,nullptr);
node* build(int l,int r){
    if(l==r)return  new node(inf,l);
    int m=(l+r)/2;
    return new node(build(l,m),build(m+1,r));
}

node* update(node *root,int l,int r,int pos,int val){
    if(l==r){
        return new node(val,pos);}
    int m=(l+r)/2;
    if(pos<=m) return new node(update(root->l,l,m,pos,val),root->r);
    return new node(root->l,update(root->r,m+1,r,pos,val));
}
item query(node *root,int l,int r,int a,int b){
    if(a>r || b<l) return all;
    if(a<=l && r<=b) return root->val;
    int m=(l+r)/2;
    return min(query(root->l,l,m,a,b),query(root->r,m+1,r,a,b));
}
```

1.10 Sparce Table

```
//Se usa para RMQ porque se puede hacer en O(1), no acepta updates
vector<int>lg;
vector<vector<int>>st;
int *nums;
void init(int n){
    int logn=(int) log2(n)+1;
    lg.assign(n+1,0);
    st.assign(logn,vector<int>(n+1));
    for(int i=0;i<n;i++) st[0][i]=nums[i];
    lg[1]=0;
    for(int i=2;i<=n;i++) lg[i]=lg[i/2]+1;
    for(int i=1;i<logn;i++)
        for(int j=0;j+(1<<i)<n;j++)st[i][j]=min(st[i-1][j],st[i-1][j+(1<<(i-1))]);
}
int query(int a,int b){
    int logn=lg[(b-a+1)];
    cout<<st[logn][a]<<endl;
    return min(st[logn][a],st[logn][b-(1<<logn)+1]);
}
```

1.11 Walvet Tree

```
// indexed in 1
// from pointer to first element and to to end
// x and y The minimun element and y the max element
// If you need only one function or more erase the others
// If you need tu construct other function you only required to undertand the limit, this
// are the same
struct wavelet_tree{
  int lo, hi;
  wavelet_tree *l, *r;
  vector<int> b;
  wavelet_tree(int *from, int *to, int x, int y){
    lo = x, hi = y;
    if(lo == hi or from >= to) return;
    int mid = (lo+hi)/2;
    auto f = [mid](int x){ return x <= mid;};
    b.reserve(to-from+1);
    b.pb(0);
    for(auto it = from; it != to; it++)
      b.push_back(b.back() + f(*it));
    auto pivot = stable_partition(from, to, f);
    l = new wavelet_tree(from, pivot, lo, mid);
    r = new wavelet_tree(pivot, to, mid+1, hi);
  }
  //kth smallest element in [l, r]
  int kth(int l, int r, int k){
    if(l > r) return 0;
    if(lo == hi) return lo;
    int inLeft = b[r] - b[l-1];
    int lb = b[l-1];
    int rb = b[r];
    if(k <= inLeft) return this->l->kth(lb+1, rb , k);
    return this->r->kth(l-lb, r-rb, k-inLeft);
  }
  //count of nos in [l, r] Less than or equal to k
  int LTE(int l, int r, int k) {
    if(l > r or k < lo) return 0;
    if(hi <= k) return r - l + 1;
    int lb = b[l-1], rb = b[r];
    return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb, r-rb, k);
  }
  //count of nos in [l, r] equal to k
  int count(int l, int r, int k) {
    if(l > r or k < lo or k > hi) return 0;
    if(lo == hi) return r - l + 1;
    int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
    if(k <= mid) return this->l->count(lb+1, rb, k);
    return this->r->count(l-lb, r-rb, k);
  }
};
```

1.12 Trie

```
struct trie{
    int len,id;
    int children[26];
    trie(int _id){
        len=0,id=_id;
        for(int i=0;i<26;i++)children[i]=-1;
    }
};vector<trie>Trie;Trie.push_back(trie());
void inserString(string str,int root){
    int aux=root;
    for(int i=0;i<str.size();i++){
        int index=str[i]-'a';
        if(Trie[aux].children[index]==-1){
            Trie.push_back(trie(Trie.size()));
            Trie[aux].children[index]=Trie.size()-1;
        }
        aux=Trie[aux].children[index];
    }
    Trie[aux].len=str.size();
}
bool existInTrie(string str,int root){
    int aux=root;
    for(int i=0;i<str.size();i++){
        int index=str[i]-'a';
        if(Trie[aux].children[index]==-1) return false;
        aux=Trie[aux].children[index];
    }
    return Trie[aux].len;
}
```

1.13 Treap

```
struct Node {
  int val=0;
  ll weight, len=1,lazy=0,sum=0;
  Node *l, *r;
  Node(int c) : val(c) ,weight(rand()), l(NULL), r(NULL) {}
} *treap;
int size(Node *root) { return root ? root->len : 0; }
ll sum(Node *root){ return root? root->sum:0;}
void pushDown(Node *&root){
  if(!root || !root->lazy) return;
    if(root->l) root->l->lazy+=root->lazy;
    if(root->r) root->r->lazy+=root->lazy;
  ll num=root->lazy;num*=size(root);
  root->sum+=num;root->lazy=0;
}
void recal(Node *&root){
  if(!root) return;
  root->len=1+size(root->l)+size(root->r);
  root->sum=sum(root->l)+sum(root->r)+root->val;
  root->val+=root->lazy;
  pushDown(root);
}
void split(Node *root, Node *&l, Node *&r, int val) {
  recal(root);
  if (!root) l = r = NULL;
  else if (size(root->l) < val) {
    split(root->r, root->r, r, val - size(root->l) - 1); l = root; recal(l);
  } else {
    split(root->l, l, root->l, val); r = root; recal(r);
  }
  recal(root);
}
void merge(Node *&root, Node *l, Node *r) {
  recal(l);recal(r);
  if (!l || !r ){root = (!(l)?r:l);}
  else if (l->weight < r->weight) {
    merge(l->r, l->r, r); root = l;
  } else {
    merge(r->l, l, r->l); root = r;
  }
  root->len=1+size(root->l)+size(root->r);
}
// Not necesary functions indexed in 1
void insert(Node *&root,Node *nNode,int pos){
    Node *l=NULL,*r=NULL,*aux=NULL;
    split(root,l,r,pos-1);
    merge(aux,l,nNode);
    merge(root,aux,r);
}
void delateRange(Node *&root,int l, int r){
    Node *l1,*r1,*l2,*r2,*aux2;
    split(root,l1,r1,l-1);
    split(r1,r1,r2,r-l+1);
    merge(root,l1,r2);
}
// queries if you dont need this you can delete recal and push-down
// rembember change the size
ll query(Node *&root,int l,int r){
  Node *l1,*r1,*l2,*r2;
  split(root,l1,r1,l-1);
  split(r1,r1,l2,r-l+1);
  ll res=sum(r1);
  merge(root,l1,r1);merge(root,root,l2);
  return res;
}
void update(Node *&root,int l,int r,ll add){
  Node *l1,*r1,*l2,*r2,*aux;
  split(root,l1,r1,l-1);
  split(r1,r1,r2,r-l+1);
  r1->lazy+=add;
  merge(l1,l1,r1);merge(root,l1,r2);
}
// debugging
ostream &operator<<(ostream &os, Node *n) {
  if (!n) return os;
  os << n->l;
  os << n->val;
  os << n->r;
  return os;
}
```

2 Strings

2.1 Aho Corasick

```
int K, I = 1;
struct node {
    int fail, ch[26] = {};
    vector<int> lens;
} T[500005];

void add(string s) {
    int x = 1;
    for (int i = 0; i < s.size(); i++) {
        if (T[x].ch[s[i] - ’a’] == 0)
            T[x].ch[s[i] - ’a’] = ++I;
        x = T[x].ch[s[i] - ’a’];
    }
    T[x].lens.PB(s.size());
}

void build() {
    queue<int> Q;
    int x = 1;
    T[1].fail = 1;
    for (int i = 0; i < 26; i++) {
        if (T[x].ch[i])
            T[T[x].ch[i]].fail = x, Q.push(T[x].ch[i]);
        else
            T[x].ch[i] = 1;
    }
    while (!Q.empty()) {
        x = Q.front(); Q.pop();
        for (int i = 0; i < 26; i++) {
            if (T[x].ch[i])
                T[T[x].ch[i]].fail = T[T[x].fail].ch[i], Q.push(T[x].ch[i]);
            else
                T[x].ch[i] = T[T[x].fail].ch[i];
        }
    }
}
```

2.2 Hashing

```
struct Hash{
  const int mod=1e9+123;
  const int p=257;
  vector<int> prefix;
  static vector<int>pow;
  Hash(string str){
    int n=str.size();
    while(pow.size()<=n){
      pow.push_back(1LL*pow.back()*p%mod);
    }
    vector<int> aux(n+1);
    prefix=aux;
    for(int i=0;i<n;i++){
      prefix[i+1]=(prefix[i]+1LL*str[i]*pow[i])%mod;
    }
```

```
 }
 inline int getHashInInerval(int i,int len,int MxPow){
  int hashing=prefix[i+len]-prefix[i];
  if(hashing<0) hashing+=mod;
  hashing=1LL*hashing*pow[MxPow-(len+i-1)]%mod;
  return hashing;
 }
};
vector<int> Hash::pow{1};
```

2.3 KMP

```
vector<int> kmp(string s){
  int n=s.size();
  vector<int>pi(n);
  for(int i=1;i<n;i++){
    int j=pi[i-1];
    while(j>0 && s[i]!=s[j])j=pi[j-1];
    if(s[i]==s[j]) j++;
    pi[i]=j;
  }
  return pi;
}
```

2.4 Manacher

```
vector<int> manacher_odd(string s) {
  int n = s.size();
  s = "$" + s + "^";
  vector<int> p(n + 2);
  int l = 1, r = 1;
  for(int i = 1; i <= n; i++) {
    p[i] = max(0, min(r - i, p[l + (r - i)]));
    while(s[i - p[i]] == s[i + p[i]]) {
      p[i]++;
    }
    if(i + p[i] > r) {
      l = i - p[i], r = i + p[i];
    }
  }
  return vector<int>(begin(p) + 1, end(p) - 1);
}
vector<int> manacher_even(string s){
  string even;
  for(auto c:s){
    even+='#'+c;
  }
  even+='#';
  return manacher_odd(even);
}
```

2.5 Suffix Automata

```
struct node{
 map<char,int>edges;
 int link,length,terminal=0;
 node(int link,int length): link(link),length(length){};
};vector<node>sa;
// init in main with sa.push_back(node(-1,0));
int last=0;
// add one by one chars in order
void addChar(char s, int pos){
  sa.push_back(node(0,pos+1));
  int r=sa.size()-1;
  int p=last;
  while(p >= 0 && sa[p].edges.find(s) == sa[p].edges.end()) {
   sa[p].edges[s] = r;
   p = sa[p].link;
  }
  if(p != -1) {
   int q = sa[p].edges[s];
   if(sa[p].length + 1 == sa[q].length) {
    sa[r].link = q;
   } else {
    sa.push_back(node(sa[q].link,sa[p].length+1));
    sa[sa.size()-1].edges=sa[q].edges;
    int qq = sa.size()-1;
    sa[q].link = qq;
    sa[r].link= qq;
    while(p >= 0 && sa[p].edges[s] == q) {
     sa[p].edges[s] = qq;
     p = sa[p].link;
    }
   }
  }
  last = r;
}
```

```
// Not necesary functions
void findTerminals(){
    int p = last;
    while(p > 0) {
       sa[p].terminal=1;
       p = sa[p].link;
    }
}
```

3 Graph

```
struct disjointSet{
  int N;
  vector<short int> rank;
  vi parent, count;

  disjointSet(int N): N(N), parent(N), count(N), rank(N){}

  void makeSet(int v){
    count[v] = 1;
    parent[v] = v;
  }

  int findSet(int v){
    if(v == parent[v]) return v;
    return parent[v] = findSet(parent[v]);
  }

  void unionSet(int a, int b){
    a = findSet(a), b = findSet(b);
    if(a == b) return;
    if(rank[a] < rank[b]){
      parent[a] = b;
      count[b] += count[a];
    }else{
      parent[b] = a;
      count[a] += count[b];
      if(rank[a] == rank[b]) ++rank[a];
    }
  }
};

struct edge{
  int source, dest, cost;

  edge(): source(0), dest(0), cost(0){}

  edge(int dest, int cost): dest(dest), cost(cost){}

  edge(int source, int dest, int cost): source(source), dest(dest), cost(cost){}

  bool operator==(const edge & b) const{
    return source == b.source && dest == b.dest && cost == b.cost;
  }
  bool operator<(const edge & b) const{
    return cost < b.cost;
  }
  bool operator>(const edge & b) const{
    return cost > b.cost;
  }
};

struct path{
  int cost = inf;
  deque<int> vertices;
  int size = 1;
  int prev = -1;
};

struct graph{
  vector<vector<edge>> adjList;
  vector<vb> adjMatrix;
  vector<vi> costMatrix;
  vector<edge> edges;
  int V = 0;
  bool dir = false;

  graph(int n, bool dir): V(n), dir(dir), adjList(n), edges(n), adjMatrix(n, vb(n)), costMatrix(n, vi(n)){
    for(int i = 0; i < n; ++i)
      for(int j = 0; j < n; ++j)
        costMatrix[i][j] = (i == j ? 0 : inf);
  }

  void add(int source, int dest, int cost){
```

```
  adjList[source].emplace_back(source, dest, cost);
  edges.emplace_back(source, dest, cost);
  adjMatrix[source][dest] = true;
  costMatrix[source][dest] = cost;
  if(!dir){
    adjList[dest].emplace_back(dest, source, cost);
    adjMatrix[dest][source] = true;
    costMatrix[dest][source] = cost;
  }
}

void buildPaths(vector<path> & paths){
  for(int i = 0; i < V; i++){
    int u = i;
    for(int j = 0; j < paths[i].size; j++){
      paths[i].vertices.push_front(u);
      u = paths[u].prev;
    }
  }
}

vector<path> dijkstra(int start){
  priority_queue<edge, vector<edge>, greater<edge>> cola;
  vector<path> paths(V);
  cola.emplace(start, 0);
  paths[start].cost = 0;
  while(!cola.empty()){
    int u = cola.top().dest; cola.pop();
    for(edge & current : adjList[u]){
      int v = current.dest;
      int nuevo = paths[u].cost + current.cost;
      if(nuevo == paths[v].cost && paths[u].size + 1 < paths[v].size){
        paths[v].prev = u;
        paths[v].size = paths[u].size + 1;
      }else if(nuevo < paths[v].cost){
        paths[v].prev = u;
        paths[v].size = paths[u].size + 1;
        cola.emplace(v, nuevo);
        paths[v].cost = nuevo;
      }
    }
  }
  buildPaths(paths);
  return paths;
}

vector<path> bellmanFord(int start){
  vector<path> paths(V, path());
  vi processed(V);
  vb inQueue(V);
  queue<int> Q;
  paths[start].cost = 0;
  Q.push(start);
  while(!Q.empty()){
    int u = Q.front(); Q.pop(); inQueue[u] = false;
    if(paths[u].cost == inf) continue;
    ++processed[u];
    if(processed[u] == V){
      cout << "Negative␣cycle\n";
      return {};
    }
    for(edge & current : adjList[u]){
      int v = current.dest;
      int nuevo = paths[u].cost + current.cost;
      if(nuevo == paths[v].cost && paths[u].size + 1 < paths[v].size){
        paths[v].prev = u;
        paths[v].size = paths[u].size + 1;
      }else if(nuevo < paths[v].cost){
        if(!inQueue[v]){
          Q.push(v);
          inQueue[v] = true;
        }
        paths[v].prev = u;
        paths[v].size = paths[u].size + 1;
        paths[v].cost = nuevo;
      }
    }
  }
  buildPaths(paths);
  return paths;
}
```

```
vector<vi> floyd(){
  vector<vi> tmp = costMatrix;
  for(int k = 0; k < V; ++k)
    for(int i = 0; i < V; ++i)
      for(int j = 0; j < V; ++j)
        if(tmp[i][k] != inf && tmp[k][j] != inf)
          tmp[i][j] = min(tmp[i][j], tmp[i][k] + tmp[k][j]);
  return tmp;
}

vector<vb> transitiveClosure(){
  vector<vb> tmp = adjMatrix;
  for(int k = 0; k < V; ++k)
    for(int i = 0; i < V; ++i)
      for(int j = 0; j < V; ++j)
        tmp[i][j] = tmp[i][j] || (tmp[i][k] && tmp[k][j]);
  return tmp;
}

vector<vb> transitiveClosureDFS(){
  vector<vb> tmp(V, vb(V));
  function<void(int, int)> dfs = [&](int start, int u){
    for(edge & current : adjList[u]){
      int v = current.dest;
      if(!tmp[start][v]){
        tmp[start][v] = true;
        dfs(start, v);
      }
    }
  };
  for(int u = 0; u < V; u++)
    dfs(u, u);
  return tmp;
}

bool isBipartite(){
  vi side(V, -1);
  queue<int> q;
  for (int st = 0; st < V; ++st){
    if(side[st] != -1) continue;
    q.push(st);
    side[st] = 0;
    while(!q.empty()){
      int u = q.front();
      q.pop();
      for (edge & current : adjList[u]){
        int v = current.dest;
        if(side[v] == -1) {
          side[v] = side[u] ^ 1;
          q.push(v);
        }else{
          if(side[v] == side[u]) return false;
        }
      }
    }
  }
  return true;
}

vi topologicalSort(){
  int visited = 0;
  vi order, indegree(V);
  for(auto & node : adjList){
    for(edge & current : node){
      int v = current.dest;
      ++indegree[v];
    }
  }
  queue<int> Q;
  for(int i = 0; i < V; ++i){
    if(indegree[i] == 0) Q.push(i);
  }
  while(!Q.empty()){
    int source = Q.front();
    Q.pop();
    order.push_back(source);
    ++visited;
    for(edge & current : adjList[source]){
      int v = current.dest;
      --indegree[v];
      if(indegree[v] == 0) Q.push(v);
    }
  }
  if(visited == V) return order;
  else return {};
}
```

```

  bool hasCycle(){
    vi color(V);
    function<bool(int, int)> dfs = [&](int u, int parent){
      color[u] = 1;
      bool ans = false;
      int ret = 0;
      for(edge & current : adjList[u]){
        int v = current.dest;
        if(color[v] == 0)
          ans |= dfs(v, u);
        else if(color[v] == 1 && (dir || v != parent || ret++))
          ans = true;
      }
      color[u] = 2;
      return ans;
    };
    for(int u = 0; u < V; ++u)
      if(color[u] == 0 && dfs(u, -1))
        return true;
    return false;
  }

  pair<vb, vector<edge>> articulationBridges(){
    vi low(V), label(V);
    vb points(V);
    vector<edge> bridges;
    int time = 0;
    function<int(int, int)> dfs = [&](int u, int p){
      label[u] = low[u] = ++time;
      int hijos = 0, ret = 0;
      for(edge & current : adjList[u]){
        int v = current.dest;
        if(v == p && !ret++) continue;
        if(!label[v]){
          ++hijos;
          dfs(v, u);
          if(label[u] <= low[v])
            points[u] = true;
          if(label[u] < low[v])
            bridges.push_back(current);
          low[u] = min(low[u], low[v]);
        }
        low[u] = min(low[u], label[v]);
      }
      return hijos;
    };
    for(int u = 0; u < V; ++u)
      if(!label[u])
        points[u] = dfs(u, -1) > 1;
    return make_pair(points, bridges);
  }

  vector<vi> scc(){
    vi low(V), label(V);
    int time = 0;
    vector<vi> ans;
    stack<int> S;
    function<void(int)> dfs = [&](int u){
      label[u] = low[u] = ++time;
      S.push(u);
      for(edge & current : adjList[u]){
        int v = current.dest;
        if(!label[v]) dfs(v);
        low[u] = min(low[u], low[v]);
      }
      if(label[u] == low[u]){
        vi comp;
        while(S.top() != u){
          comp.push_back(S.top());
          low[S.top()] = V + 1;
          S.pop();
        }
        comp.push_back(S.top());
        S.pop();
        ans.push_back(comp);
        low[u] = V + 1;
      }
    };
    for(int u = 0; u < V; ++u)
      if(!label[u]) dfs(u);
    return ans;
  }

  vector<edge> kruskal(){
    sort(edges.begin(), edges.end());
  vector<edge> MST;
  disjointSet DS(V);
  for(int u = 0; u < V; ++u)
    DS.makeSet(u);
  int i = 0;
  while(i < edges.size() && MST.size() < V - 1){
    edge current = edges[i++];
    int u = current.source, v = current.dest;
    if(DS.findSet(u) != DS.findSet(v)){
      MST.push_back(current);
      DS.unionSet(u, v);
    }
  }
  return MST;
}

bool tryKuhn(int u, vb & used, vi & left, vi & right){
  if(used[u]) return false;
  used[u] = true;
  for(edge & current : adjList[u]){
    int v = current.dest;
    if(right[v] == -1 || tryKuhn(right[v], used, left, right)){
      right[v] = u;
      left[u] = v;
      return true;
    }
  }
  return false;
}

bool augmentingPath(int u, vb & used, vi & left, vi & right){
  used[u] = true;
  for(edge & current : adjList[u]){
    int v = current.dest;
    if(right[v] == -1){
      right[v] = u;
      left[u] = v;
      return true;
    }
  }
  for(edge & current : adjList[u]){
    int v = current.dest;
    if(!used[right[v]] && augmentingPath(right[v], used, left, right))
        {
      right[v] = u;
      left[u] = v;
      return true;
    }
  }
  return false;
}

//vertices from the left side numbered from 0 to l-1
//vertices from the right side numbered from 0 to r-1
//graph[u] represents the left side
//graph[u][v] represents the right side
//we can use tryKuhn() or augmentingPath()
vector<pair<int, int>> maxMatching(int l, int r){
  vi left(l, -1), right(r, -1);
  vb used(l);
  for(int u = 0; u < l; ++u){
    tryKuhn(u, used, left, right);
    fill(used.begin(), used.end(), false);
  }
  vector<pair<int, int>> ans;
  for(int u = 0; u < r; ++u){
    if(right[u] != -1){
      ans.emplace_back(right[u], u);
    }
  }
  return ans;
}

void dfs(int u, vi & status, vi & parent){
  status[u] = 1;
  for(edge & current : adjList[u]){
    int v = current.dest;
    if(status[v] == 0){ //not visited
      parent[v] = u;
      dfs(v, status, parent);
    }else if(status[v] == 1){ //explored
      if(v == parent[u]){
        //bidirectional node u<-->v
      }else{
        //back edge u-v
      }
      }else if(status[v] == 2){ //visited
        //forward edge u-v
      }
    }
    status[u] = 2;
  }
};

struct tree{
  vi parent, level, weight;
  vector<vi> dists, DP;
  int n, root;

  void dfs(int u, graph & G){
    for(edge & curr : G.adjList[u]){
      int v = curr.dest;
      int w = curr.cost;
      if(v != parent[u]){
        parent[v] = u;
        weight[v] = w;
        level[v] = level[u] + 1;
        dfs(v, G);
      }
    }
  }

  tree(int n, int root): n(n), root(root), parent(n), level(n), weight(n), dists(n, vi(20)), DP(n, vi(20)){
    parent[root] = root;
  }

  tree(graph & G, int root): n(G.V), root(root), parent(G.V), level(G.V), weight(G.V), dists(G.V, vi(20)), DP(G.V, vi(20)){
    parent[root] = root;
    dfs(root, G);
  }

  void pre(){
    for(int u = 0; u < n; u++){
      DP[u][0] = parent[u];
      dists[u][0] = weight[u];
    }
    for(int i = 1; (1 << i) <= n; ++i){
      for(int u = 0; u < n; ++u){
        DP[u][i] = DP[DP[u][i - 1]][i - 1];
        dists[u][i] = dists[u][i - 1] + dists[DP[u][i - 1]][i - 1];
      }
    }
  }

  int ancestor(int p, int k){
    int h = level[p] - k;
    if(h < 0) return -1;
    int lg;
    for(lg = 1; (1 << lg) <= level[p]; ++lg);
    lg--;
    for(int i = lg; i >= 0; --i){
      if(level[p] - (1 << i) >= h){
        p = DP[p][i];
      }
    }
    return p;
  }

  int lca(int p, int q){
    if(level[p] < level[q]) swap(p, q);
    int lg;
    for(lg = 1; (1 << lg) <= level[p]; ++lg);
    lg--;
    for(int i = lg; i >= 0; --i){
      if(level[p] - (1 << i) >= level[q]){
        p = DP[p][i];
      }
    }
    if(p == q) return p;

    for(int i = lg; i >= 0; --i){
      if(DP[p][i] != -1 && DP[p][i] != DP[q][i]){
        p = DP[p][i];
        q = DP[q][i];
      }
    }
    return parent[p];
  }

  int dist(int p, int q){
    if(level[p] < level[q]) swap(p, q);
    int lg;
    for(lg = 1; (1 << lg) <= level[p]; ++lg);
    lg--;
    int sum = 0;
    for(int i = lg; i >= 0; --i){
      if(level[p] - (1 << i) >= level[q]){
        sum += dists[p][i];
        p = DP[p][i];
      }
    }
    if(p == q) return sum;

    for(int i = lg; i >= 0; --i){
      if(DP[p][i] != -1 && DP[p][i] != DP[q][i]){
        sum += dists[p][i] + dists[q][i];
        p = DP[p][i];
        q = DP[q][i];
      }
    }
    sum += dists[p][0] + dists[q][0];
    return sum;
  }
};
```

4 Flow

4.1 Dinics

```
struct Dinic{
  int nodes,src,dst;
  vector<int> dist,q,work;
  struct edge {int to,rev;ll f,cap;};
  vector<vector<edge>> g;
  Dinic(int x):nodes(x),g(x),dist(x),q(x),work(x){}
  void add_edge(int s, int t, ll cap){
    g[s].pb((edge){t,SZ(g[t]),0,cap});
    g[t].pb((edge){s,SZ(g[s])-1,0,0});
  }
  bool dinic_bfs(){
    fill(ALL(dist),-1);dist[src]=0;
    int qt=0;q[qt++]=src;
    for(int qh=0;qh<qt;qh++){
      int u=q[qh];
      fore(i,0,SZ(g[u])){
        edge &e=g[u][i];int v=g[u][i].to;
        if(dist[v]<0&&e.f<e.cap)dist[v]=dist[u]+1,q[qt++]=v;
      }
    }
    return dist[dst]>=0;
  }
  ll dinic_dfs(int u, ll f){
    if(u==dst)return f;
    for(int &i=work[u];i<SZ(g[u]);i++){
      edge &e=g[u][i];
      if(e.cap<=e.f)continue;
      int v=e.to;
      if(dist[v]==dist[u]+1){
        ll df=dinic_dfs(v,min(f,e.cap-e.f));
        if(df>0){e.f+=df;g[v][e.rev].f-=df;return df;}
      }
    }
    return 0;
  }
  ll max_flow(int _src, int _dst){
    src=_src;dst=_dst;
    ll result=0;
    while(dinic_bfs()){
      fill(ALL(work),0);
      while(ll delta=dinic_dfs(src,INF))result+=delta;
    }
    return result;
  }
};
```

4.2 Edmon

```
struct Edmons{
    #define ll long long
    int n;
    vector<int>d;
    vector<tuple<int,ll,ll>>edges;
    vector<vector<int>> adj;
    vector<pair<int,int>>cam;
    Edmons(int _n):adj(_n+1),n(_n){}
    ll sentFlow(int s,int t,ll f){
        if(s==t)return f;
        auto &[u,idx]=cam[t];
        auto cap=get<1>(edges[idx]),&flow=get<2>(edges[idx]);
        ll push=sentFlow(s,u,min(cap-flow,f));
        flow+=push;
        auto &flowr=get<2>(edges[idx^1]);
        flowr-=push;
        return push;
    }
    bool bfs(int s,int t){
        d.assign(n+1,-1); d[s]=0;
        cam.assign(n+1,{-1,-1});
        queue<int> q({s});
        while(!q.empty()){
            int u=q.front();
            q.pop();
            for(auto idx:adj[u]){
                auto &v=get<0>(edges[idx]);auto &cap=get<1>(edges[idx]),&flow=get<2>(edges[idx]);
                if(cap-flow>0 && d[v]==-1) d[v]=d[u]+1,cam[v]={u,idx},q.push(v);
            }
        }
        return d[t]!=-1;
    }
    ll maxFlow(int s,int t){
        ll flow=0;
        while(bfs(s,t)){
            ll push=sentFlow(s,t,1e18);
            if(!push) return flow;
            flow+=push;
        }
        return flow;
    }
    void addEdge(int u,int v, ll c, bool dire=true){
        if(u==v) return;
        edges.emplace_back(v,c,0);
        adj[u].push_back(edges.size()-1);
        edges.emplace_back(u,(dire?0:c),0);
        adj[v].push_back(edges.size()-1);
    }
};
```

5 Geometria

```
#include <bits/stdc++.h>
using namespace std;
using ld = long double;
const ld eps = 1e-9, inf = numeric_limits<ld>::max(), pi = acos(-1);
// For use with integers, just set eps=0 and everything remains the same
bool geq(ld a, ld b){return a-b >= -eps;}     //a >= b
bool leq(ld a, ld b){return b-a >= -eps;}     //a <= b
bool ge(ld a, ld b){return a-b > eps;}        //a > b
bool le(ld a, ld b){return b-a > eps;}        //a < b
bool eq(ld a, ld b){return abs(a-b) <= eps;}  //a == b
bool neq(ld a, ld b){return abs(a-b) > eps;}  //a != b

struct point{
  ld x, y;
  point(): x(0), y(0){}
  point(ld x, ld y): x(x), y(y){}

  point operator+(const point & p) const{return point(x + p.x, y + p.y);}
  point operator-(const point & p) const{return point(x - p.x, y - p.y);}
  point operator*(const ld & k) const{return point(x * k, y * k);}
  point operator/(const ld & k) const{return point(x / k, y / k);}

  point operator+=(const point & p){*this = *this + p; return *this;}
  point operator-=(const point & p){*this = *this - p; return *this;}
  point operator*=(const ld & p){*this = *this * p; return *this;}
  point operator/=(const ld & p){*this = *this / p; return *this;}

  point rotate(const ld & a) const{return point(x*cos(a) - y*sin(a), x*sin(a) + y*cos(a));}
  point perp() const{return point(-y, x);}
  ld ang() const{
    ld a = atan2l(y, x); a += le(a, 0) ? 2*pi : 0; return a;
  }
  ld dot(const point & p) const{return x * p.x + y * p.y;}
  ld cross(const point & p) const{return x * p.y - y * p.x;}
  ld norm() const{return x * x + y * y;}
  ld length() const{return sqrtl(x * x + y * y);}
  point unit() const{return (*this) / length();}
```

```cpp

  bool operator==(const point & p) const{return eq(x, p.x) && eq(y, p.y);}
  bool operator!=(const point & p) const{return !(*this == p);}
  bool operator<(const point & p) const{return le(x, p.x) || (eq(x, p.x) && le(y, p.y));}
  bool operator>(const point & p) const{return ge(x, p.x) || (eq(x, p.x) && ge(y, p.y));}
  bool half(const point & p) const{return le(p.cross(*this), 0) || (eq(p.cross(*this), 0) && le(p.dot(*this), 0));}
};

istream &operator>>(istream &is, point & p){return is >> p.x >> p.y;}
ostream &operator<<(ostream &os, const point & p){return os << "(" << p.x << ",␣" << p.y << ")";}

int sgn(ld x){
  if(ge(x, 0)) return 1;
  if(le(x, 0)) return -1;
  return 0;
}

void polarSort(vector<point> & P, const point & o, const point & v){
  //sort points in P around o, taking the direction of v as first angle
  sort(P.begin(), P.end(), [&](const point & a, const point & b){
    return point((a - o).half(v), 0) < point((b - o).half(v), (a - o).cross(b - o));
  });
}

bool pointInLine(const point & a, const point & v, const point & p){
  //line a+tv, point p
  return eq((p - a).cross(v), 0);
}

bool pointInSegment(const point & a, const point & b, const point & p){
  //segment ab, point p
  return pointInLine(a, b - a, p) && leq((a - p).dot(b - p), 0);
}

int intersectLinesInfo(const point & a1, const point & v1, const point & a2, const point & v2){
  //lines a1+tv1 and a2+tv2
  ld det = v1.cross(v2);
  if(eq(det, 0)){
    if(eq((a2 - a1).cross(v1), 0)){
      return -1; //infinity points
    }else{
      return 0; //no points
    }
  }else{
    return 1; //single point
  }
}

point intersectLines(const point & a1, const point & v1, const point & a2, const point & v2){
  //lines a1+tv1, a2+tv2
  //assuming that they intersect
  ld det = v1.cross(v2);
  return a1 + v1 * ((a2 - a1).cross(v2) / det);
}

int intersectLineSegmentInfo(const point & a, const point & v, const point & c, const point & d){
  //line a+tv, segment cd
  point v2 = d - c;
  ld det = v.cross(v2);
  if(eq(det, 0)){
    if(eq((c - a).cross(v), 0)){
      return -1; //infinity points
    }else{
      return 0; //no point
    }
  }else{
    return sgn(v.cross(c - a)) != sgn(v.cross(d - a)); //1: single point, 0: no point
  }
}

int intersectSegmentsInfo(const point & a, const point & b, const point & c, const point & d){
  //segment ab, segment cd
  point v1 = b - a, v2 = d - c;
  int t = sgn(v1.cross(c - a)), u = sgn(v1.cross(d - a));
  if(t == u){
    if(t == 0){
      if(pointInSegment(a, b, c) || pointInSegment(a, b, d) || pointInSegment(c, d, a) || pointInSegment(c, d, b)){
        return -1; //infinity points
      }else{
        return 0; //no point
      }
    }else{
      return 0; //no point
    }
  }else{
    return sgn(v2.cross(a - c)) != sgn(v2.cross(b - c)); //1: single point, 0: no point
  }
}

ld distancePointLine(const point & a, const point & v, const point & p){
  //line: a + tv, point p
  return abs(v.cross(p - a)) / v.length();
}

ld perimeter(vector<point> & P){
  int n = P.size();
  ld ans = 0;
  for(int i = 0; i < n; i++){
    ans += (P[i] - P[(i + 1) % n]).length();
  }
  return ans;
}

ld area(vector<point> & P){
  int n = P.size();
  ld ans = 0;
  for(int i = 0; i < n; i++){
    ans += P[i].cross(P[(i + 1) % n]);
  }
  return abs(ans / 2);
}

vector<point> convexHull(vector<point> P){
  sort(P.begin(), P.end());
  vector<point> L, U;
  for(int i = 0; i < P.size(); i++){
    while(L.size() >= 2 && leq((L[L.size() - 2] - P[i]).cross(L[L.size() - 1] - P[i]), 0)){
      L.pop_back();
    }
    L.push_back(P[i]);
  }
  for(int i = P.size() - 1; i >= 0; i--){
    while(U.size() >= 2 && leq((U[U.size() - 2] - P[i]).cross(U[U.size() - 1] - P[i]), 0)){
      U.pop_back();
    }
    U.push_back(P[i]);
  }
  L.pop_back();
  U.pop_back();
  L.insert(L.end(), U.begin(), U.end());
  return L;
}

bool pointInPerimeter(const vector<point> & P, const point & p){
  int n = P.size();
  for(int i = 0; i < n; i++){
    if(pointInSegment(P[i], P[(i + 1) % n], p)){
      return true;
    }
  }
  return false;
}

bool crossesRay(const point & a, const point & b, const point & p){
  return (geq(b.y, p.y) - geq(a.y, p.y)) * sgn((a - p).cross(b - p)) > 0;
}

int pointInPolygon(const vector<point> & P, const point & p){
  if(pointInPerimeter(P, p)){
    return -1; //point in the perimeter
  }
  int n = P.size();
  int rays = 0;
  for(int i = 0; i < n; i++){
    rays += crossesRay(P[i], P[(i + 1) % n], p);
  }
```

```cpp
  return rays & 1; //0: point outside, 1: point inside
}

//point in convex polygon in O(log n)
//make sure that P is convex and in ccw
//before the queries, do the preprocess on P:
// rotate(P.begin(), min_element(P.begin(), P.end()), P.end());
// int right = max_element(P.begin(), P.end()) - P.begin();
//returns 0 if p is outside, 1 if p is inside, -1 if p is in the perimeter
int pointInConvexPolygon(const vector<point> & P, const point & p, int right){
  if(p < P[0] || P[right] < p) return 0;
  int orientation = sgn((P[right] - P[0]).cross(p - P[0]));
  if(orientation == 0){
    if(p == P[0] || p == P[right]) return -1;
    return (right == 1 || right + 1 == P.size()) ? -1 : 1;
  }else if(orientation < 0){
    auto r = lower_bound(P.begin() + 1, P.begin() + right, p);
    int det = sgn((p - r[-1]).cross(r[0] - r[-1])) - 1;
    if(det == -2) det = 1;
    return det;
  }else{
    auto l = upper_bound(P.rbegin(), P.rend() - right - 1, p);
    int det = sgn((p - l[0]).cross((l == P.rbegin() ? P[0] : l[-1]) - l[0])) - 1;
    if(det == -2) det = 1;
    return det;
  }
}





vector<point> cutPolygon(const vector<point> & P, const point & a, const point & v){
  //returns the part of the convex polygon P on the left side of line a+tv
  int n = P.size();
  vector<point> lhs;
  for(int i = 0; i < n; ++i){
    if(geq(v.cross(P[i] - a), 0)){
      lhs.push_back(P[i]);
    }
    if(intersectLineSegmentInfo(a, v, P[i], P[(i+1)%n]) == 1){
      point p = intersectLines(a, v, P[i], P[(i+1)%n] - P[i]);
      if(p != P[i] && p != P[(i+1)%n]){
        lhs.push_back(p);
      }
    }
  }
  return lhs;
}
















point centroid(vector<point> & P){
  point num;
  ld den = 0;
  int n = P.size();
  for(int i = 0; i < n; ++i){
    ld cross = P[i].cross(P[(i + 1) % n]);
    num += (P[i] + P[(i + 1) % n]) * cross;
    den += cross;
  }
  return num / (3 * den);
}

vector<pair<int, int>> antipodalPairs(vector<point> & P){
  vector<pair<int, int>> ans;
  int n = P.size(), k = 1;
  auto f = [&](int u, int v, int w){return abs((P[v%n]-P[u%n]).cross(P[w
%n]-P[u%n]));};
  while(ge(f(n-1, 0, k+1), f(n-1, 0, k))) ++k;
  for(int i = 0, j = k; i <= k && j < n; ++i){
    ans.emplace_back(i, j);
    while(j < n-1 && ge(f(i, i+1, j+1), f(i, i+1, j)))
      ans.emplace_back(i, ++j);
  }
  return ans;
}

pair<ld, ld> diameterAndWidth(vector<point> & P){
  int n = P.size(), k = 0;
  auto dot = [&](int a, int b){return (P[(a+1)%n]-P[a]).dot(P[(b+1)%n]-P[b]);};
  auto cross = [&](int a, int b){return (P[(a+1)%n]-P[a]).cross(P[(b+1)%n]-P[b]);};
  ld diameter = 0;
  ld width = inf;
  while(ge(dot(0, k), 0)) k = (k+1) % n;
  for(int i = 0; i < n; ++i){
    while(ge(cross(i, k), 0)) k = (k+1) % n;
    //pair: (i, k)
    diameter = max(diameter, (P[k] - P[i]).length());
    width = min(width, distancePointLine(P[i], P[(i+1)%n] - P[i], P[k]));
  }
  return {diameter, width};
}

pair<ld, ld> smallestEnclosingRectangle(vector<point> & P){
  int n = P.size();
  auto dot = [&](int a, int b){return (P[(a+1)%n]-P[a]).dot(P[(b+1)%n]-P[b]);};
  auto cross = [&](int a, int b){return (P[(a+1)%n]-P[a]).cross(P[(b+1)%n]-P[b]);};
  ld perimeter = inf, area = inf;
  for(int i = 0, j = 0, k = 0, m = 0; i < n; ++i){
    while(ge(dot(i, j), 0)) j = (j+1) % n;
    if(!i) k = j;
    while(ge(cross(i, k), 0)) k = (k+1) % n;
    if(!i) m = k;
    while(le(dot(i, m), 0)) m = (m+1) % n;
    //pairs: (i, k) , (j, m)
    point v = P[(i+1)%n] - P[i];
    ld h = distancePointLine(P[i], v, P[k]);
    ld w = distancePointLine(P[j], v.perp(), P[m]);
    perimeter = min(perimeter, 2 * (h + w));
    area = min(area, h * w);
  }
  return {area, perimeter};
}

ld distancePointCircle(const point & c, ld r, const point & p){
  //point p, circle with center c and radius r
  return max((ld)0, (p - c).length() - r);
}

point projectionPointCircle(const point & c, ld r, const point & p){
  //point p (outside the circle), circle with center c and radius r
  return c + (p - c).unit() * r;
}

pair<point, point> pointsOfTangency(const point & c, ld r, const point & p){
  //point p (outside the circle), circle with center c and radius r
  point v = (p - c).unit() * r;
  ld d2 = (p - c).norm(), d = sqrt(d2);
  point v1 = v * (r / d), v2 = v.perp() * (sqrt(d2 - r*r) / d);
  return {c + v1 - v2, c + v1 + v2};
}

vector<point> intersectLineCircle(const point & a, const point & v, const point & c, ld r){
  //line a+tv, circle with center c and radius r
  ld h2 = r*r - v.cross(c - a) * v.cross(c - a) / v.norm();
  point p = a + v * v.dot(c - a) / v.norm();
  if(eq(h2, 0)) return {p}; //line tangent to circle
  else if(le(h2, 0)) return {}; //no intersection
  else{
    point u = v.unit() * sqrt(h2);
    return {p - u, p + u}; //two points of intersection (chord)
  }
}

vector<point> intersectSegmentCircle(const point & a, const point & b, const point & c, ld r){
  //segment ab, circle with center c and radius r
  vector<point> P = intersectLineCircle(a, b - a, c, r), ans;
  for(const point & p : P){
    if(pointInSegment(a, b, p)) ans.push_back(p);
  }
  return ans;
}

pair<point, ld> getCircle(const point & m, const point & n, const point & p){
  //find circle that passes through points p, q, r
  point c = intersectLines((n + m) / 2, (n - m).perp(), (p + n) / 2, (p - n).perp());
  ld r = (c - m).length();
  return {c, r};
}

vector<point> intersectionCircles(const point & c1, ld r1, const point & c2, ld r2){
  //circle 1 with center c1 and radius r1
  //circle 2 with center c2 and radius r2
  point d = c2 - c1;
  ld d2 = d.norm();
  if(eq(d2, 0)) return {}; //concentric circles
  ld pd = (d2 + r1*r1 - r2*r2) / 2;
  ld h2 = r1*r1 - pd*pd/d2;
  point p = c1 + d*pd/d2;
  if(eq(h2, 0)) return {p}; //circles touch at one point
  else if(le(h2, 0)) return {}; //circles don't intersect
  else{
    point u = d.perp() * sqrt(h2/d2);
    return {p - u, p + u};
  }
}

int circleInsideCircle(const point & c1, ld r1, const point & c2, ld r2){
  //test if circle 2 is inside circle 1
  //returns "-1" if 2 touches internally 1, "1" if 2 is inside 1, "0" if they overlap
  ld l = r1 - r2 - (c1 - c2).length();
  return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}

int circleOutsideCircle(const point & c1, ld r1, const point & c2, ld r2){
  //test if circle 2 is outside circle 1
  //returns "-1" if they touch externally, "1" if 2 is outside 1, "0" if they overlap
  ld l = (c1 - c2).length() - (r1 + r2);
  return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}

int pointInCircle(const point & c, ld r, const point & p){
  //test if point p is inside the circle with center c and radius r
  //returns "0" if it's outside, "-1" if it's in the perimeter, "1" if it's inside
  ld l = (p - c).length() - r;
  return (le(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}

vector<vector<point>> tangents(const point & c1, ld r1, const point & c2, ld r2, bool inner){
  //returns a vector of segments or a single point
  if(inner) r2 = -r2;
  point d = c2 - c1;
  ld dr = r1 - r2, d2 = d.norm(), h2 = d2 - dr*dr;
  if(eq(d2, 0) || le(h2, 0)) return {};
  point v = d*dr/d2;
  if(eq(h2, 0)) return {{c1 + v*r1}};
  else{
    point u = d.perp()*sqrt(h2)/d2;
    return {{c1 + (v - u)*r1, c2 + (v - u)*r2}, {c1 + (v + u)*r1, c2 + (v + u)*r2}};
  }
}

ld signed_angle(const point & a, const point & b){
  return sgn(a.cross(b)) * acosl(a.dot(b) / (a.length() * b.length()));
}

ld intersectPolygonCircle(const vector<point> & P, const point & c, ld r){
  //Gets the area of the intersection of the polygon with the circle
  int n = P.size();
  ld ans = 0;
```

```cpp
  for(int i = 0; i < n; ++i){
    point p = P[i], q = P[(i+1)%n];
    bool p_inside = (pointInCircle(c, r, p) != 0);
    bool q_inside = (pointInCircle(c, r, q) != 0);
    if(p_inside && q_inside){
      ans += (p - c).cross(q - c);
    }else if(p_inside && !q_inside){
      point s1 = intersectSegmentCircle(p, q, c, r)[0];
      point s2 = intersectSegmentCircle(c, q, c, r)[0];
      ans += (p - c).cross(s1 - c) + r*r * signed_angle(s1 - c, s2 - c);
    }else if(!p_inside && q_inside){
      point s1 = intersectSegmentCircle(c, p, c, r)[0];
      point s2 = intersectSegmentCircle(p, q, c, r)[0];
      ans += (s2 - c).cross(q - c) + r*r * signed_angle(s1 - c, s2 - c);
    }else{
      auto info = intersectSegmentCircle(p, q, c, r);
      if(info.size() <= 1){
        ans += r*r * signed_angle(p - c, q - c);
      }else{
        point s2 = info[0], s3 = info[1];
        point s1 = intersectSegmentCircle(c, p, c, r)[0];
        point s4 = intersectSegmentCircle(c, q, c, r)[0];
        ans += (s2 - c).cross(s3 - c) + r*r * (signed_angle(s1 - c, s2 - c) + signed_angle(s3 - c, s4 - c));
      }
    }
  }
  return abs(ans)/2;
}

pair<point, ld> mec2(vector<point> & S, const point & a, const point & b, int n){
  ld hi = inf, lo = -hi;
  for(int i = 0; i < n; ++i){
    ld si = (b - a).cross(S[i] - a);
    if(eq(si, 0)) continue;
    point m = getCircle(a, b, S[i]).first;
    ld cr = (b - a).cross(m - a);
    if(le(si, 0)) hi = min(hi, cr);
    else lo = max(lo, cr);
  }
  ld v = (ge(lo, 0) ? lo : le(hi, 0) ? hi : 0);
  point c = (a + b) / 2 + (b - a).perp() * v / (b - a).norm();
  return {c, (a - c).norm()};
}

pair<point, ld> mec(vector<point> & S, const point & a, int n){
  random_shuffle(S.begin(), S.begin() + n);
  point b = S[0], c = (a + b) / 2;
  ld r = (a - c).norm();
  for(int i = 1; i < n; ++i){
    if(ge((S[i] - c).norm(), r)){
      tie(c, r) = (n == S.size() ? mec(S, S[i], i) : mec2(S, a, S[i], i));
    }
  }
  return {c, r};
}

pair<point, ld> smallestEnclosingCircle(vector<point> S){
  assert(!S.empty());
  auto r = mec(S, S[0], S.size());
  return {r.first, sqrt(r.second)};
}

bool comp1(const point & a, const point & b){
  return le(a.y, b.y);
}
pair<point, point> closestPairOfPoints(vector<point> P){
  sort(P.begin(), P.end(), comp1);
  set<point> S;
  ld ans = inf;
  point p, q;
  int pos = 0;
  for(int i = 0; i < P.size(); ++i){
    while(pos < i && geq(P[i].y - P[pos].y, ans)){
      S.erase(P[pos++]);
    }
    auto lower = S.lower_bound({P[i].x - ans - eps, -inf});
    auto upper = S.upper_bound({P[i].x + ans + eps, -inf});
    for(auto it = lower; it != upper; ++it){
      ld d = (P[i] - *it).length();
      if(le(d, ans)){
        ans = d;
        p = P[i];
        q = *it;
      }
    }
    S.insert(P[i]);
  }
  return {p, q};
}

struct vantage_point_tree{
  struct node
  {
    point p;
    ld th;
    node *l, *r;
  }*root;

  vector<pair<ld, point>> aux;

  vantage_point_tree(vector<point> &ps){
    for(int i = 0; i < ps.size(); ++i)
      aux.push_back({ 0, ps[i] });
    root = build(0, ps.size());
  }

  node *build(int l, int r){
    if(l == r)
      return 0;
    swap(aux[l], aux[l + rand() % (r - l)]);
    point p = aux[l++].second;
    if(l == r)
      return new node({ p });
    for(int i = l; i < r; ++i)
      aux[i].first = (p - aux[i].second).dot(p - aux[i].second);
    int m = (l + r) / 2;
    nth_element(aux.begin() + l, aux.begin() + m, aux.begin() + r);
    return new node({ p, sqrt(aux[m].first), build(l, m), build(m, r) })
        ;
  }

  priority_queue<pair<ld, node*>> que;

  void k_nn(node *t, point p, int k){
    if(!t)
      return;
    ld d = (p - t->p).length();
    if(que.size() < k)
      que.push({ d, t });
    else if(ge(que.top().first, d)){
      que.pop();
      que.push({ d, t });
    }
    if(!t->l && !t->r)
      return;
    if(le(d, t->th)){
      k_nn(t->l, p, k);
      if(leq(t->th - d, que.top().first))
        k_nn(t->r, p, k);
    }else{
      k_nn(t->r, p, k);
      if(leq(d - t->th, que.top().first))
        k_nn(t->l, p, k);
    }
  }

  vector<point> k_nn(point p, int k){
    k_nn(root, p, k);
    vector<point> ans;
    for(; !que.empty(); que.pop())
      ans.push_back(que.top().second->p);
    reverse(ans.begin(), ans.end());
    return ans;
  }
};

vector<point> minkowskiSum(vector<point> A, vector<point> B){
  int na = (int)A.size(), nb = (int)B.size();
  if(A.empty() || B.empty()) return {};

  rotate(A.begin(), min_element(A.begin(), A.end()), A.end());
  rotate(B.begin(), min_element(B.begin(), B.end()), B.end());

  int pa = 0, pb = 0;
  vector<point> M;

  while(pa < na && pb < nb){
    M.push_back(A[pa] + B[pb]);
    ld x = (A[(pa + 1) % na] - A[pa]).cross(B[(pb + 1) % nb] - B[pb]);
    if(leq(x, 0)) pb++;
    if(geq(x, 0)) pa++;
  }

  while(pa < na) M.push_back(A[pa++] + B[0]);
  while(pb < nb) M.push_back(B[pb++] + A[0]);

  return M;
}

//Delaunay triangulation in O(n log n)
const point inf_pt(inf, inf);

struct QuadEdge{
  point origin;
  QuadEdge* rot = nullptr;
  QuadEdge* onext = nullptr;
  bool used = false;
  QuadEdge* rev() const{return rot->rot;}
  QuadEdge* lnext() const{return rot->rev()->onext->rot;}
  QuadEdge* oprev() const{return rot->onext->rot;}
  point dest() const{return rev()->origin;}
};

QuadEdge* make_edge(const point & from, const point & to){
  QuadEdge* e1 = new QuadEdge;
  QuadEdge* e2 = new QuadEdge;
  QuadEdge* e3 = new QuadEdge;
  QuadEdge* e4 = new QuadEdge;
  e1->origin = from;
  e2->origin = to;
  e3->origin = e4->origin = inf_pt;
  e1->rot = e3;
  e2->rot = e4;
  e3->rot = e2;
  e4->rot = e1;
  e1->onext = e1;
  e2->onext = e2;
  e3->onext = e4;
  e4->onext = e3;
  return e1;
}

void splice(QuadEdge* a, QuadEdge* b){
  swap(a->onext->rot->onext, b->onext->rot->onext);
  swap(a->onext, b->onext);
}

void delete_edge(QuadEdge* e){
  splice(e, e->oprev());
  splice(e->rev(), e->rev()->oprev());
  delete e->rot;
  delete e->rev()->rot;
  delete e;
  delete e->rev();
}

QuadEdge* connect(QuadEdge* a, QuadEdge* b){
  QuadEdge* e = make_edge(a->dest(), b->origin);
  splice(e, a->lnext());
  splice(e->rev(), b);
  return e;
}

bool left_of(const point & p, QuadEdge* e){
  return ge((e->origin - p).cross(e->dest() - p), 0);
}

bool right_of(const point & p, QuadEdge* e){
  return le((e->origin - p).cross(e->dest() - p), 0);
}

ld det3(ld a1, ld a2, ld a3, ld b1, ld b2, ld b3, ld c1, ld c2, ld c3) {
  return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) + a3 * (b1
      * c2 - c1 * b2);
}

bool in_circle(const point & a, const point & b, const point & c, const
    point & d) {
  ld det = -det3(b.x, b.y, b.norm(), c.x, c.y, c.norm(), d.x, d.y, d.
      norm());
  det += det3(a.x, a.y, a.norm(), c.x, c.y, c.norm(), d.x, d.y, d.norm()
      );
  det -= det3(a.x, a.y, a.norm(), b.x, b.y, b.norm(), d.x, d.y, d.norm()
      );
  det += det3(a.x, a.y, a.norm(), b.x, b.y, b.norm(), c.x, c.y, c.norm()
      );
  return ge(det, 0);
}

pair<QuadEdge*, QuadEdge*> build_tr(int l, int r, vector<point> & P){
  if(r - l + 1 == 2){
    QuadEdge* res = make_edge(P[l], P[r]);
    return {res, res->rev()};
  }
  if(r - l + 1 == 3){
    QuadEdge *a = make_edge(P[l], P[l + 1]), *b = make_edge(P[l + 1], P[r]);
    splice(a->rev(), b);
    int sg = sgn((P[l + 1] - P[l]).cross(P[r] - P[l]));
    if(sg == 0)
      return {a, b->rev()};
    QuadEdge* c = connect(b, a);
    if(sg == 1)
      return {a, b->rev()};
    else
      return {c->rev(), c};
  }
  int mid = (l + r) / 2;
  QuadEdge *ldo, *ldi, *rdo, *rdi;
  tie(ldo, ldi) = build_tr(l, mid, P);
  tie(rdi, rdo) = build_tr(mid + 1, r, P);
  while(true){
    if(left_of(rdi->origin, ldi)){
      ldi = ldi->lnext();
      continue;
    }
    if(right_of(ldi->origin, rdi)){
      rdi = rdi->rev()->onext;
      continue;
    }
    break;
  }
  QuadEdge* basel = connect(rdi->rev(), ldi);
  auto valid = [&basel](QuadEdge* e){return right_of(e->dest(), basel);};
  if(ldi->origin == ldo->origin)
    ldo = basel->rev();
  if(rdi->origin == rdo->origin)
    rdo = basel;
  while(true){
    QuadEdge* lcand = basel->rev()->onext;
    if(valid(lcand)){
      while(in_circle(basel->dest(), basel->origin, lcand->dest(), lcand->onext->dest())){
        QuadEdge* t = lcand->onext;
        delete_edge(lcand);
        lcand = t;
      }
    }
    QuadEdge* rcand = basel->oprev();
    if(valid(rcand)){
      while(in_circle(basel->dest(), basel->origin, rcand->dest(), rcand->oprev()->dest())){
        QuadEdge* t = rcand->oprev();
        delete_edge(rcand);
        rcand = t;
      }
    }
    if(!valid(lcand) && !valid(rcand))
      break;
    if(!valid(lcand) || (valid(rcand) && in_circle(lcand->dest(), lcand->origin, rcand->origin, rcand->dest())))
      basel = connect(rcand, basel->rev());
    else
      basel = connect(basel->rev(), lcand->rev());
  }
  return {ldo, rdo};
}

vector<tuple<point, point, point>> delaunay(vector<point> & P){
  sort(P.begin(), P.end());
  auto res = build_tr(0, (int)P.size() - 1, P);
  QuadEdge* e = res.first;
  vector<QuadEdge*> edges = {e};
  while(le((e->dest() - e->onext->dest()).cross(e->origin - e->onext->dest()), 0))
    e = e->onext;
  auto add = [&P, &e, &edges](){
    QuadEdge* curr = e;
    do{
      curr->used = true;
      P.push_back(curr->origin);
      edges.push_back(curr->rev());
      curr = curr->lnext();
    }while(curr != e);
  };
  add();
  P.clear();
  int kek = 0;
  while(kek < (int)edges.size())
    if(!(e = edges[kek++])->used)
      add();
  vector<tuple<point, point, point>> ans;
  for(int i = 0; i < (int)P.size(); i += 3){
    ans.emplace_back(P[i], P[i + 1], P[i + 2]);
  }
  return ans;
}

struct circ{
  point c;
  ld r;
  circ() {}
  circ(const point & c, ld r): c(c), r(r) {}
  set<pair<ld, ld>> ranges;

  void disable(ld l, ld r){
    ranges.emplace(l, r);
  }

  auto getActive() const{
    vector<pair<ld, ld>> ans;
    ld maxi = 0;
    for(const auto & dis : ranges){
      ld l, r;
      tie(l, r) = dis;
      if(l > maxi){
        ans.emplace_back(maxi, l);
      }
      maxi = max(maxi, r);
    }
    if(!eq(maxi, 2*pi)){
      ans.emplace_back(maxi, 2*pi);
    }
    return ans;
  }
};

ld areaUnionCircles(const vector<circ> & circs){
  vector<circ> valid;
  for(const circ & curr : circs){
    if(eq(curr.r, 0)) continue;
    circ nuevo = curr;
    for(circ & prev : valid){
      if(circleInsideCircle(prev.c, prev.r, nuevo.c, nuevo.r)){
        nuevo.disable(0, 2*pi);
      }else if(circleInsideCircle(nuevo.c, nuevo.r, prev.c, prev.r)){
        prev.disable(0, 2*pi);
      }else{
        auto cruce = intersectionCircles(prev.c, prev.r, nuevo.c, nuevo.
            r);
        if(cruce.size() == 2){
          ld a1 = (cruce[0] - prev.c).ang();
          ld a2 = (cruce[1] - prev.c).ang();
          ld b1 = (cruce[1] - nuevo.c).ang();
          ld b2 = (cruce[0] - nuevo.c).ang();
          if(a1 < a2){
            prev.disable(a1, a2);
          }else{
            prev.disable(a1, 2*pi);
            prev.disable(0, a2);
          }
          if(b1 < b2){
            nuevo.disable(b1, b2);
          }else{
            nuevo.disable(b1, 2*pi);
            nuevo.disable(0, b2);
          }
        }
      }
    }
    valid.push_back(nuevo);
  }
  ld ans = 0;
  for(const circ & curr : valid){
    for(const auto & range : curr.getActive()){
      ld l, r;
      tie(l, r) = range;
      ans += curr.r*(curr.c.x * (sin(r) - sin(l)) - curr.c.y * (cos(r) - cos(l))) + curr.r*curr.r*(r-l);
    }
  }
  return ans/2;
};

struct plane{
  point a, v;
  plane(): a(), v(){}
  plane(const point& a, const point& v): a(a), v(v){}

  point intersect(const plane& p) const{
    ld t = (p.a - a).cross(p.v) / v.cross(p.v);
    return a + v*t;
  }

  bool outside(const point& p) const{ // test if point p is strictly outside
    return le(v.cross(p - a), 0);
  }

  bool inside(const point& p) const{ // test if point p is inside or in the boundary
    return geq(v.cross(p - a), 0);
  }

  bool operator<(const plane& p) const{ // sort by angle
    auto lhs = make_tuple(v.half({1, 0}), ld(0), v.cross(p.a - a));
    auto rhs = make_tuple(p.v.half({1, 0}), v.cross(p.v), ld(0));
    return lhs < rhs;
  }

  bool operator==(const plane& p) const{ // paralell and same directions, not really equal
    return eq(v.cross(p.v), 0) && ge(v.dot(p.v), 0);
  }
};

vector<point> halfPlaneIntersection(vector<plane> planes){
  planes.push_back({{0, -inf}, {1, 0}});
  planes.push_back({{inf, 0}, {0, 1}});
  planes.push_back({{0, inf}, {-1, 0}});
  planes.push_back({{-inf, 0}, {0, -1}});
  sort(planes.begin(), planes.end());
  planes.erase(unique(planes.begin(), planes.end()), planes.end());
  deque<plane> ch;
  deque<point> poly;
  for(const plane& p : planes){
    while(ch.size() >= 2 && p.outside(poly.back())) ch.pop_back(), poly.pop_back();
    while(ch.size() >= 2 && p.outside(poly.front())) ch.pop_front(), poly.pop_front();
    if(p.v.half({1, 0}) && poly.empty()) return {};
    ch.push_back(p);
    if(ch.size() >= 2) poly.push_back(ch[ch.size()-2].intersect(ch[ch.size()-1]));
  }
  while(ch.size() >= 3 && ch.front().outside(poly.back())) ch.pop_back(), poly.pop_back();
  while(ch.size() >= 3 && ch.back().outside(poly.front())) ch.pop_front(), poly.pop_front();
  poly.push_back(ch.back().intersect(ch.front()));
  return vector<point>(poly.begin(), poly.end());
}

vector<point> halfPlaneIntersectionRandomized(vector<plane> planes){
  point p = planes[0].a;
  int n = planes.size();
  random_shuffle(planes.begin(), planes.end());
  for(int i = 0; i < n; ++i){
    if(planes[i].inside(p)) continue;
    ld lo = -inf, hi = inf;
    for(int j = 0; j < i; ++j){
      ld A = planes[j].v.cross(planes[i].v);
      ld B = planes[j].v.cross(planes[j].a - planes[i].a);
      if(ge(A, 0)){
        lo = max(lo, B/A);
      }else if(le(A, 0)){
        hi = min(hi, B/A);
      }else{
        if(ge(B, 0)) return {};
      }
      if(ge(lo, hi)) return {};
    }
    p = planes[i].a + planes[i].v*lo;
  }
  return {p};
}

int main(){
  /*vector<pair<point, point>> centers = {{point(-2, 5), point(-8, -7)}, {point(14, 4), point(18, 6)}, {point(9, 20), point(9, 28)},
                      {point(21, 20), point(21, 29)}, {point(8, -10), point(14, -10)}, {point(24, -6), point(34, -6)},
                      {point(34, 8), point(36, 9)}, {point(50, 20), point(56, 24.5)}};
  vector<pair<ld, ld>> radii = {{7, 4}, {3, 5}, {4, 4}, {4, 5}, {3, 3}, {4, 6}, {5, 1}, {10, 2.5}};
  int n = centers.size();
  for(int i = 0; i < n; ++i){
    cout << "\n" << centers[i].first << " " << radii[i].first << " " << centers[i].second << " " << radii[i].second << "\n";
    auto extLines = tangents(centers[i].first, radii[i].first, centers[i].second, radii[i].second, false);
    cout << "Exterior tangents:\n";
    for(auto par : extLines){
      for(auto p : par){
        cout << p << " ";
      }
      cout << "\n";
    }
    auto intLines = tangents(centers[i].first, radii[i].first, centers[i].second, radii[i].second, true);
    cout << "Interior tangents:\n";
    for(auto par : intLines){
      for(auto p : par){
        cout << p << " ";
      }
      cout << "\n";
    }
  }*/

  /*int n;
  cin >> n;
  vector<point> P(n);
  for(auto & p : P) cin >> p;
  auto triangulation = delaunay(P);
  for(auto triangle : triangulation){
    cout << get<0>(triangle) << " " << get<1>(triangle) << " " << get<2>(triangle) << "\n";
  }*/

  /*int n;
  cin >> n;
  vector<point> P(n);
  for(auto & p : P) cin >> p;
  auto ans = smallestEnclosingCircle(P);
  cout << ans.first << " " << ans.second << "\n";*/

  /*vector<point> P;
  srand(time(0));
  for(int i = 0; i < 1000; ++i){
    P.emplace_back(rand() % 1000000000, rand() % 1000000000);
  }
  point o(rand() % 1000000000, rand() % 1000000000), v(rand() % 1000000000, rand() % 1000000000);
  polarSort(P, o, v);
  auto ang = [&](point p){
    ld th = atan2(p.y, p.x);
    if(th < 0) th += acosl(-1)*2;
    ld t = atan2(v.y, v.x);
    if(t < 0) t += acosl(-1)*2;
    if(th < t) th += acosl(-1)*2;
    return th;
  };
  for(int i = 0; i < P.size()-1; ++i){
    assert(leq(ang(P[i] - o), ang(P[i+1] - o)));
  }*/
  return 0;
}
```

6 Varios

6.1 Template

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

#define forn(i,n)       for(int i=0; i<n; i++)
```

```
#define forr(i,a,n)     for(int i=a; i<n; i++)
#define fore(i,a,n)     for(int i=a; i<=n; i++)
#define each(a,b)       for(auto a: b)
#define all(v)          v.begin(),v.end()
#define sz(a)           (int)a.size()
#define debln(a)        cout << a << "\n"
#define deb(a)          cout << a << " "
#define pb              push_back

typedef long long ll;
typedef vector<int> vi;
typedef pair<int,int> ii;

void sol(){

}

int main(){
    ios::sync_with_stdio(false);cin.tie(0);

    int t=1;
    cin>>t;
    while(t--){
        sol();
    }

    return 0;
}
```

6.2 String a vector¡int¿

```
//Convertir una cadena de numeros separados por " " en vector de enteros
//Leer varias de esas querys
cin.ignore();
while(q--){
  string s;
  getline(cin, s);
  vector<int> qr;
  stringstream ss(s);
  int num;
  while (ss >> num)   qr.push_back(num);
}
```

6.3 Generar permutaciones

```
//Generar todas las permutaciones de un arreglo
sort(all(a));
do{
  //hacer lo que quieras con la perm generada
}while(next_permutation(all(a)));
```

6.4 2 Sat

```
struct twoSat{
    int s;
    vector<vector<int>> g,gr;
    vector<int> visited,ids,topologic_sort,val;
    twoSat(int n){
        s=n;
        g.assign(n*2+1,vector<int>());
        gr.assign(n*2+1,vector<int>());
        visited.assign(n*2+1,0);
        ids.assign(n*2+1,0);
        val.assign(n+1,0);
    }
    void addEdge(int a,int b){
        g[a].push_back(b);
        gr[b].push_back(a);
    }
    void addOr(int a,bool ba,int b,bool bb){
        addEdge(a+(ba?s:0),b+(bb?0:s));
        addEdge(b+(bb?s:0),a+(ba?0:s));
    }
    void addXor(int a,bool ba,int b,bool bb){
        addOr(a,ba,b,bb);
        addOr(a,!ba,b,!bb);
    }
    void addAnd(int a,bool ba,int b,bool bb){
        addXor(a,!ba,b,bb);
    }
    void dfs(int u){
        if(visited[u]!=0) return;
        visited[u]=1;
        for(int node:g[u])dfs(node);
        topologic_sort.push_back(u);
    }
    void dfsr(int u,int id){
        if(visited[u]!=0) return;
        visited[u]=1;
        ids[u]=id;
        for(int node:gr[u])dfsr(node,id);
    }
    bool algo(){
        for(int i=0;i<s*2;i++) if(visited[i]==0) dfs(i);
        fill(visited.begin(),visited.end(),0);
        reverse(topologic_sort.begin(),topologic_sort.end());
        int id=0;
        for(int i=0;i<topologic_sort.size();i++){
            if(visited[topologic_sort[i]]==0)dfsr(topologic_sort[i],id++);
        }
        for(int i=0;i<s;i++){
            if(ids[i]==ids[i+s]) return false;
            val[i]=(ids[i]>ids[i+s]?0:1);
        }
        return true;
    }
};
```

6.5 Bits

```
__builtin_popcount(maks) // Count the numbers of on bits
```

6.6 Matrix

```
const int N=100, MOD=1e9+7;
struct Matrix {
  ll a[N][N];
  Matrix() {memset(a,0,sizeof(a));}
  Matrix operator *(Matrix other) {  // Product of a matrix
    Matrix product=Matrix();
        rep(i,0,N) rep(j,0,N) rep(k,0,N) {
            product.a[i][k]+=a[i][j]*other.a[j][k];
            product.a[i][k]%=MOD;
        }
    return product;
  }
};
Matrix expo_power(Matrix a, ll n) {  // Matrix exponentiation
  Matrix res=Matrix();
    rep(i,0,N) res.a[i][i]=1;  // Matriz identidad
  while(n){
        if(n&1) res=res*a;
        n>>=1;
        a=a*a;
  }
  return res;
} // Ej. Matrix M=Matrix();  M.a[0][0]=1;  M=M*M;   Matrix res=expo_power(M,k);
```

6.7 MO

```
void remove(idx);  // TODO: remove value at idx from data structure
void add(idx);     // TODO: add value at idx from data structure
int get_answer();  // TODO: extract the current answer of the data structure

int block_size;//Recomended sqrt(n)

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the range [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}
```

6.8 PBS

```

    1.Crear un arreglo con para procesar
    2.Para cada elemento inicialicar 1 l y en q+1 r;
    for(int i=1;i<=n;i++){
        m[i].x=1,m[i].y=q+1;
    }
    bool flag=true;
    while(flag){
        flag=false;
        // limpiar la estructura de datos
        for(int i=0;i<=4*n+5;i++)st[i]=0,lazy[i]=0;
        for(int i=1;i<=n;i++)
           //Si es diefente l!=r se procesa;
          if(m[i].x!=m[i].y){ flag=true;tocheck[(m[i].x+m[i].y)/2].push_back(i);}
        for(int i=1;i<=q;i++){
            if(!flag)break;
            // Se aplican las queries
            update(0,n-1,qs[i].x,qs[i].y,qs[i].z,0);
            update(0,n-1,qs[i].x,qs[i].x,qs[i].k,0);
            while(tocheck[i].size()){
                int id=tocheck[i].back();
                tocheck[i].pop_back();
                // Se obserba si se cumblio la caondicion para el elemeto
                if(ai[id]<=query(0,n-1,S[id],S[id],0)) m[id].y=i;
                else m[id].x=i+1;
            }
        }
    }
    // Solo se imprime
    for(int i=1;i<=n;i++){
        if(m[i].x<=q) cout<<m[i].x<<endl;
        else cout<<-1<<endl;
    }
```