1 Data-Structure

1.1 Treap

```cpp
template<class S,
    S (*node_pull)(S, S),
    S (*node_init)(S),
    class T,
    S (*mapping)(S, T),
    T (*tag_pull)(T, T),
    T (*tag_init)()>
struct Treap{
  struct node{
    node *l = NULL,*r = NULL,*p = NULL;
    int pri = mt(),sz = 1;
    S info;
    T tag = tag_init();
    bool rev;
    node(S k) : info(k){}
    ~node(){
      for(auto &i:{l,r})
        delete i;
    }
    void all_apply(T t,bool is_rev){
      if(is_rev){
        swap(l,r);
        rev^=1;
      }
      info = mapping(info, t);
      tag = tag_pull(tag, t);
    }
    void push(){
      for(auto &i:{l,r})
        if(i)i->all_apply(tag, rev);
      tag = tag_init();
      rev = 0;
    }
    void pull(){
      sz = 1,info = node_init(info);
      for(auto &i:{l,r}){
        if(i){
          sz+=i->sz,i->p = this;
          info = node_pull(info,i->info);
        }
      }
    }
  };
  node *root = NULL;
  int size(node *a){
    return a?a->sz:0;
  }
  int size(){
    return size(root);
  }
  node *merge(node *a,node *b){
    if(!a or !b)return a?:b;
    if(a->pri>b->pri){
      a->push();
      a->r = merge(a->r,b);
      a->r->p = a;
      a->pull();
      return a;
    }
    else{
      b->push();
      b->l = merge(a,b->l);
      b->l->p = b;
      b->pull();
      return b;
    }
  }
  void split(node *t, long long k, node *&a, node *&b, bool bst){
    if(!t){a = b = NULL;return;}
    t->push();
    if((bst==0 and size(t->l)+1<=k) or (bst==1 and t->info.key<=k)){
      a = t;
      split(t->r, ( bst ? k : k - size(t->l) - 1 ), a->r, b, bst);
      if(b)b->p = NULL;
      a->pull();
    }
    else{
      b = t;
      split(t->l, k, a, b->l, bst);
      if(a)a->p = NULL;
      b->pull();
    }
  }
  node *insert(long long idx, S x,bool bst = 0){
    node *a,*b;
    split(root, idx, a, b, bst);
    node *tmp = new node(x);
    root = merge(a, merge(tmp, b));
    return tmp;
  }
  void erase(long long l,long long r,bool bst = 0){
    node *a,*b,*c;
    split(root, (bst? l-1 : l), a, b, bst);
    split(b, (bst? r : r - l + 1), b, c, bst);
    delete b;
    root = merge(a,c);
  }
  S operator [](int x){
    node *a, *b, *c;
    split(root, x, a, b, 0);
    split(b, 1, b, c, 0);
    assert(b!=NULL);
    S ans = b->info;
    root = merge(a, merge(b, c));
    return ans;
  }
  void update(long long l,long long r,T t,bool bst = 0){
    node *a, *b, *c;
    split(root, (bst? l - 1 : l), a, b, bst);
    split(b, (bst? r : r - l + 1), b, c, bst);
    if(b)b->all_apply(t, 0);
    root = merge(a, merge(b, c));
  }
  void reverse(long long l,long long r,bool bst = 0){
    node *a, *b, *c;
    split(root, (bst? l - 1 : l), a, b, bst);
    split(b, (bst? r : r - l + 1), b, c, bst);
    if(b)b->all_apply(tag_init(), 1);
    root = merge(a, merge(b, c));
  }
  int rank(long long k){
    node *a, *b;
    split(root, k - 1, a, b, 1);
    int ans = size(a);
    root = merge(a, b);
    return ans;
  }
  S* find_next(long long k){
    node *a, *b, *c;
    split(root, k - 1, a, b, 1);
    split(b, 1, b, c, 0);
    S* ans = NULL;
    if(b)ans = &b->info;
    root = merge(a, merge(b, c));
    return ans;
  }
  S* find_prev(long long k){
    node *a, *b, *c;
    split(root, k , a, b, 1);
    split(a, size(a) - 1, a, c, 0);
    S* ans = NULL;
    if(c)ans = &c->info;
    root = merge(merge(a, c), b);
    return ans;
  }
  S query(long long l,long long r,bool bst = 0){
    node *a, *b, *c;
    split(root, (bst? l - 1 : l), a, b, bst);
    split(b, (bst? r : r - l + 1), b, c, bst);
    assert(b!=NULL);
    S ans = b->info;
    root = merge(a, merge(b, c));
    return  ans;
  }
};
```

1.2 Segtree

```cpp
template<class S,
     S (*node_pull)(S, S),
     S (*node_init)(),
     class T,
     S (*mapping)(S, T),
     T (*tag_pull)(T, T),
     T (*tag_init)()>
struct segment_tree{
  struct node{
    S seg;
    T tag = tag_init();
    int l,r;
    node(S _seg = node_init(),int _l = -1,int _r = -1) : seg(_seg), l(_l), r(_r){}
    friend node operator +(const node &lhs,const node &rhs){
      if(lhs.l==-1)return rhs;
      if(rhs.l==-1)return lhs;
      return node(node_pull(lhs.seg,rhs.seg),lhs.l,rhs.r);
    };
  };
  vector<node>arr;
  void all_apply(int idx,T t){
    arr[idx].seg = mapping(arr[idx].seg, t);
    arr[idx].tag = tag_pull(arr[idx].tag, t);
  }
  void push(int idx){
    all_apply(idx<<1, arr[idx].tag);
    all_apply(idx<<1|1, arr[idx].tag);
    arr[idx].tag = tag_init();
  }
  inline void build(const vector<S> &v,const int &l,const int &r,int idx = 1){
    if(idx==1)arr.resize((r-l+1)<<2);
    if(l==r){
      arr[idx].seg = v[l];
      arr[idx].tag = tag_init();
      arr[idx].l = arr[idx].r = l;
      return;
    }
    int m = (l+r)>>1;
    build(v,l,m,idx<<1);
    build(v,m+1,r,idx<<1|1);
    arr[idx] = arr[idx<<1]+arr[idx<<1|1];
  }
  inline void update(const int &ql,const int &qr,T t,int idx = 1){
    assert(ql<=qr);
    if(ql<=arr[idx].l and arr[idx].r<=qr){
      all_apply(idx, t);
      return;
    }
    push(idx);
    int m = (arr[idx].l+arr[idx].r)>>1;
    if(ql<=m)update(ql,qr,t,idx<<1);
    if(qr>m)update(ql,qr,t,idx<<1|1);
    arr[idx] = arr[idx<<1]+arr[idx<<1|1];
  }
  inline S query(const int &ql,const int &qr,int idx = 1){
    assert(ql<=qr);
    if(ql<=arr[idx].l and arr[idx].r<=qr){
      return arr[idx].seg;
    }
    push(idx);
    int m = (arr[idx].l+arr[idx].r)>>1;
    S ans = node_init(),lhs = node_init(),rhs = node_init();
    if(ql<=m)lhs = query(ql,qr,idx<<1);
    if(qr>m)rhs = query(ql,qr,idx<<1|1);
    ans = node_pull(lhs,rhs);
    return ans;
  }
};
```

1.3 DsuUndo

```
struct dsu_undo{
  vector<int>sz,p;
  int comps;
  dsu_undo(int n){
    sz.assign(n+5,1);
    p.resize(n+5);
    for(int i = 1;i<=n;++i)p[i] = i;
    comps = n;
  }
  vector<pair<int,int>>opt;
  int Find(int x){
    return x==p[x]?x:Find(p[x]);
  }
  bool Union(int a,int b){
    int pa = Find(a),pb = Find(b);
    if(pa==pb)return 0;
    if(sz[pa]<sz[pb])swap(pa,pb);
    sz[pa]+=sz[pb];
    p[pb] = pa;
    opt.push_back({pa,pb});
    comps--;
    return 1;
  }
  void undo(){
        auto [pa,pb] = opt.back();
        opt.pop_back();
        p[pb] = pb;
        sz[pa]-=sz[pb];
        comps++;
  }
};
```

1.4 DSU

```
struct DSU{
  vector<int>sz;
  int n;
  DSU(int _n):n(_n){
    sz.assign(n+1,-1);
  }
  int Find(int x){
    return sz[x]<0?x:sz[x] = Find(sz[x]);
  }
  bool Union(int a,int b){
    int pa = Find(a),pb = Find(b);
    if(pa==pb)return 0;
    if((-sz[pa])<(-sz[pb]))swap(pa,pb);
    sz[pa]+=sz[pb];
    sz[pb] = pa;
    return 1;
  }
};
```

1.5 Fenwick

```
template<class T>struct fenwick_tree{
  int n;
  vector<T>arr;
  inline int lowbit(int x){
    return x&(-x);
  }
  fenwick_tree(int _n) : n(_n){
    arr.assign(n+5,0);
  }
  T query(int x){
    T ans = 0;
    for(int i = x;i>0;i-=lowbit(i)){
      ans+=arr[i];
    }
    return ans;
  }
  void update(int x,T y){
    for(int i = x;i<=n;i+=lowbit(i)){
      arr[i]+=y;
    }
  }
};
```

1.6 Persistent DSU

```
int rk[200001] = {};
struct Persistent_DSU{
  rope<int>*p;
  int n;
  Persistent_DSU(int _n = 0):n(_n){
    if(n==0)return;
    p = new rope<int>;
    int tmp[n+1] = {};
    for(int i = 1;i<=n;++i)tmp[i] = i;
    p->append(tmp,n+1);
  }
  Persistent_DSU(const Persistent_DSU &tmp){
    p = new rope<int>(*tmp.p);
    n = tmp.n;
  }
  int Find(int x){
    int px = p->at(x);
    return px==x?x:Find(px);
  }
  bool Union(int a,int b){
    int pa = Find(a),pb = Find(b);
    if(pa==pb)return 0;
    if(rk[pa]<rk[pb])swap(pa,pb);
    p->replace(pb,pa);
    if(rk[pa]==rk[pb])rk[pa]++;
    return 1;
  }
};
```

1.7 TimingSegtree

```
template<class T,class D>struct timing_segment_tree{
  struct node{
    int l,r;
    vector<T>opt;
  };
  vector<node>arr;
  void build(int l,int r,int idx = 1){
    if(idx==1)arr.resize((r-l+1)<<2);
    if(l==r){
      arr[idx].l = arr[idx].r = l;
      arr[idx].opt.clear();
      return;
    }
    int m = (l+r)>>1;
    build(l,m,idx<<1);
    build(m+1,r,idx<<1|1);
    arr[idx].l = l,arr[idx].r = r;
    arr[idx].opt.clear();
```

```cpp
  }
  void update(int ql,int qr,T k,int idx = 1){
    if(ql<=arr[idx].l and arr[idx].r<=qr){
      arr[idx].opt.push_back(k);
      return;
    }
    int m = (arr[idx].l+arr[idx].r)>>1;
    if(ql<=m)update(ql,qr,k,idx<<1);
    if(qr>m)update(ql,qr,k,idx<<1|1);
  }
  void dfs(D &d,vector<int>&ans,int idx = 1){
    int cnt = 0;
    for(auto [a,b]:arr[idx].opt){
      if(d.Union(a,b))cnt++;
    }
    if(arr[idx].l==arr[idx].r)ans[arr[idx].l] = d.comps;
    else{
      dfs(d,ans,idx<<1);
      dfs(d,ans,idx<<1|1);
    }
    while(cnt--)d.undo();
  }
};
```

1.8 AreaOfRectangles

```cpp
long long AreaOfRectangles(vector<tuple<int,int,int,int>>v){
  vector<tuple<int,int,int,int>>tmp;
  int L = INT_MAX,R = INT_MIN;
  for(auto [x1,y1,x2,y2]:v){
    tmp.push_back({x1,y1+1,y2,1});
    tmp.push_back({x2,y1+1,y2,-1});
    R = max(R,y2);
    L = min(L,y1);
  }
  vector<long long>seg((R-L+1)<<2),tag((R-L+1)<<2);
  sort(tmp.begin(),tmp.end());
  function<void(int,int,int,int,int,int)>update = [&](int ql, int qr,int val,int l,int r,int idx){
    if(ql<=l and r<=qr){
      tag[idx]+=val;
      if(tag[idx])seg[idx] = r-l+1;
      else if(l==r)seg[idx] = 0;
      else seg[idx] = seg[idx<<1]+seg[idx<<1|1];
      return;
    }
    int m = (l+r)>>1;
    if(ql<=m)update(ql,qr,val,l,m,idx<<1);
    if(qr>m)update(ql,qr,val,m+1,r,idx<<1|1);
    if(tag[idx])seg[idx] = r-l+1;
    else seg[idx] = seg[idx<<1]+seg[idx<<1|1];
  };
  long long last_pos = 0,ans = 0;
  for(auto [pos,l,r,val]:tmp){
    ans+=(pos-last_pos)*seg[1];
    update(l,r,val,L,R,1);
    last_pos = pos;
  }
  return ans;
}
```

1.9 SparseTable

```cpp
template<class T,T (*op)(T,T)>struct sparse_table{
  int n;
  vector<vector<T>>mat;
  sparse_table(): n(0){}
  sparse_table(const vector<T>&v){
    n = (int)(v.size());
    mat.resize(30);
    mat[0] = v;
    for(int i = 1;(1<<i)<=n;++i){
      mat[i].resize(n-(1<<i)+1);
      for(int j = 0;j<n-(1<<i)+1;++j){
        mat[i][j] = op(mat[i-1][j],mat[i-1][j+(1<<(i-1))]);
      }
    }
  }
  T query(int ql,int qr){
    int k = __lg(qr-ql+1);
    return op(mat[k][ql],mat[k][qr-(1<<k)+1]);
  }
};
```

1.10 VEBTree

```cpp
// Can correctly work with numbers in range [0; MAXN]
// Supports all std::set operations in O(1) on random queries / dense arrays, O(log_64(N)) in worst case (sparce array).
// Count operation works in O(1) always.
template<unsigned int MAXN>
class fast_set {
private:
  static const unsigned int PREF = (MAXN <= 64 ? 0 :
                MAXN <= 4096 ? 1 :
                MAXN <= 262144 ? 1 + 64 :
                MAXN <= 16777216 ? 1 + 64 + 4096 :
                MAXN <= 1073741824 ? 1 + 64 + 4096 + 262144 : 227) + 1;
  static constexpr unsigned long long lowest_bitsll[] = {0ULL, 1ULL, 3ULL, 7ULL, 15ULL, 31ULL, 63ULL, 127ULL, 255ULL, 511ULL, 1023ULL, 2047ULL, 4095ULL, 8191ULL, 16383ULL, 32767ULL, 65535ULL, 131071ULL, 262143ULL, 524287ULL, 1048575ULL, 2097151ULL, 4194303ULL, 8388607ULL, 16777215ULL, 33554431ULL, 67108863ULL, 134217727ULL, 268435455ULL, 536870911ULL, 1073741823ULL, 2147483647ULL, 4294967295ULL, 8589934591ULL, 17179869183ULL, 34359738367ULL, 68719476735ULL, 137438953471ULL, 274877906943ULL, 549755813887ULL, 1099511627775ULL, 2199023255551ULL, 4398046511103ULL, 8796093022207ULL, 17592186044415ULL, 35184372088831ULL, 70368744177663ULL, 140737488355327ULL, 281474976710655ULL, 562949953421311ULL, 1125899906842623ULL, 2251799813685247ULL, 4503599627370495ULL, 9007199254740991ULL, 18014398509481983ULL, 36028797018963967ULL, 72057594037927935ULL, 144115188075855871ULL, 288230376151711743ULL, 576460752303423487ULL, 1152921504606846975ULL, 2305843009213693951ULL, 4611686018427387903ULL, 9223372036854775807ULL, 18446744073709551615ULL};
  static const unsigned int SZ = PREF + (MAXN + 63) / 64 + 1;
  unsigned long long m[SZ] = {0};

  inline unsigned int left(unsigned int v) const {
    return (v - 62) * 64;
  }

  inline unsigned int parent(unsigned int v) const {
    return v / 64 + 62;
  }

  inline void setbit(unsigned int v) {
    m[v >> 6] |= 1ULL << (v & 63);
  }

  inline void resetbit(unsigned int v) {
    m[v >> 6] &= ~(1ULL << (v & 63));
  }

  inline unsigned int getbit(unsigned int v) const {
    return m[v >> 6] >> (v & 63) & 1;
  }

  inline unsigned long long childs_value(unsigned int v) const {
    return m[left(v) >> 6];
  }

  inline int left_go(unsigned int x, const unsigned int c) const {
    const unsigned long long rem = x & 63;
    unsigned int bt = PREF * 64 + x;
    unsigned long long num = m[bt >> 6] & lowest_bitsll[rem + c];
    if(num) {
      return (x ^ rem) | __lg(num);
    }
    for(bt = parent(bt); bt > 62; bt = parent(bt)) {
      const unsigned long long rem = bt & 63;
      num = m[bt >> 6] & lowest_bitsll[rem];
      if(num) {
        bt = (bt ^ rem) | __lg(num);
        break;
      }
    }
    if(bt == 62) {
      return -1;
    }
    while(bt < PREF * 64) {
      bt = left(bt) | __lg(m[bt - 62]);
    }
    return bt - PREF * 64;
  }

  inline int right_go(unsigned int x, const unsigned int c) const {
    const unsigned long long rem = x & 63;
    unsigned int bt = PREF * 64 + x;
    unsigned long long num = m[bt >> 6] & ~lowest_bitsll[rem + c];
    if(num) {
      return (x ^ rem) | __builtin_ctzll(num);
    }
    for(bt = parent(bt); bt > 62; bt = parent(bt)) {
      const unsigned long long rem = bt & 63;
      num = m[bt >> 6] & ~lowest_bitsll[rem + 1];
      if(num) {
        bt = (bt ^ rem) | __builtin_ctzll(num);
        break;
      }
    }
    if(bt == 62) {
      return -1;
    }
    while(bt < PREF * 64) {
      bt = left(bt) | __builtin_ctzll(m[bt - 62]);
    }
    return bt - PREF * 64;
  }

public:
  fast_set() {
    assert(PREF != 228);
    setbit(62);
  }

  bool empty() const {return getbit(63);}

  void clear() {
    fill(m, m + SZ, 0);
    setbit(62);
  }

  bool count(unsigned int x) const {
    return m[PREF + (x >> 6)] >> (x & 63) & 1;
  }

  void insert(unsigned int x) {
    for(unsigned int v = PREF * 64 + x; !getbit(v); v = parent(v)) {
      setbit(v);
    }
  }

  void erase(unsigned int x) {
    if(!getbit(PREF * 64 + x)) {
      return;
    }
    resetbit(PREF * 64 + x);
    for(unsigned int v = parent(PREF * 64 + x); v > 62 && !childs_value(v); v = parent(v)) {
      resetbit(v);
    }
  }

  int find_next(unsigned int x) const {
    return right_go(x, 0);
  }

  int find_prev(unsigned int x) const {
    return left_go(x, 1);
  }
};
```

1.11 DynamicSegtree

```

template<class T>struct dynamic_segment_tree{
  struct node{
    node *l = NULL,*r = NULL;
    T sum;
    node(T k = 0): sum(k){}
    node(node *p){if(p)*this = *p;}
    ~node(){
      for(auto &i:{l,r})
        if(i)delete i;
    }
    void pull(){
      sum = 0;
      for(auto i:{l,r})
        if(i)sum+=i->sum;
    }
  }*root = NULL;
  int n;
  dynamic_segment_tree(){}
  dynamic_segment_tree(const dynamic_segment_tree<T>&tmp){root = new node(tmp.root);}
  void update(node *&t,int pos,T k,int l,int r){
    if(!t)t = new node();
    if(l==r)return t = new node(k),void();
    int m = (l+r)>>1;
    t = new node(t);
    if(pos<=m)update(t->l,pos,k,l,m);
    else update(t->r,pos,k,m+1,r);
    t->pull();
  }void update(int pos,T k,int l = -1e9,int r = 1e9){update(root,pos,k,l,r);}
  T query(node *&t,int ql,int qr,int l,int r){
    if(!t)return 0;
    if(ql<=l and r<=qr)return t->sum;
    int m = (l+r)>>1;
    T ans = 0;
    if(ql<=m)ans+=query(t->l,ql,qr,l,m);
    if(qr>m)ans+=query(t->r,ql,qr,m+1,r);
    return ans;
  }T query(int ql,int qr,int l = -1e9,int r = 1e9){return query(root,ql,qr,l,r);}
};
```

1.12 ZkwSegtree

```
template<class S,
         S (*node_pull)(S, S),
         S (*node_init)(),
         class F,
         S (*mapping)(S, F),
         F (*tag_pull)(F, F),
         F (*tag_init)()>
class segment_tree {
public:
  segment_tree() : segment_tree(0) {}
  explicit segment_tree(int _n) : segment_tree(vector<S>(_n, node_init())) {}
  explicit segment_tree(const vector<S>& v) : n((int) v.size()) {
    log = std::__lg(2 * n - 1);
    size = 1 << log;
    d = vector<S>(size << 1, node_init());
    lz = vector<F>(size, tag_init());
    for(int i = 0; i < n; i++) {
      d[size + i] = v[i];
    }
    for(int i = size - 1; i; --i) {
      update(i);
    }
  }
  void set(int p, S x) {
    assert(0 <= p && p < n);
    p += size;
    for(int i = log; i; --i) {
      push(p >> i);
    }
    d[p] = x;
    for(int i = 1; i <= log; ++i) {
      update(p >> i);
    }
  }
  S get(int p) {
    assert(0 <= p && p < n);
    p += size;
    for(int i = log; i; i--) {
      push(p >> i);
    }
    return d[p];
  }
  S operator[](int p) {
    return get(p);
  }
  S query(int l, int r) {
    r++;
    assert(l<=r);
    l += size;
    r += size;
    for(int i = log; i; i--) {
      if(((l >> i) << i) != l) {
        push(l >> i);
      }
      if(((r >> i) << i) != r) {
        push(r >> i);
      }
    }
    S sml = node_init(), smr = node_init();
    while(l < r) {
      if(l & 1) {
        sml = node_pull(sml, d[l++]);
      }
      if(r & 1) {
        smr = node_pull(d[--r], smr);
      }
      l >>= 1;
      r >>= 1;
    }
    return node_pull(sml, smr);
  }
  void apply(int p, F f) {
    assert(0 <= p && p < n);
    p += size;
    for(int i = log; i; i--) {
      push(p >> i);
    }
    d[p] = mapping(f, d[p]);
    for(int i = 1; i <= log; i++) {
```

```cpp
      update(p >> i);
    }
  }
  void update(int l, int r, F f) {
    r++;
    assert(l<=r);
    l += size;
    r += size;
    for(int i = log; i; i--) {
      if(((l >> i) << i) != l) {
        push(l >> i);
      }
      if(((r >> i) << i) != r) {
        push((r - 1) >> i);
      }
    }
    {
      int l2 = l, r2 = r;
      while(l < r) {
        if(l & 1) {
          all_apply(l++, f);
        }
        if(r & 1) {
          all_apply(--r, f);
        }
        l >>= 1;
        r >>= 1;
      }
      l = l2;
      r = r2;
    }
    for(int i = 1; i <= log; i++) {
      if(((l >> i) << i) != l) {
        update(l >> i);
      }
      if(((r >> i) << i) != r) {
        update((r - 1) >> i);
      }
    }
  }
private:
  int n, size, log;
  vector<S> d;
  vector<F> lz;
  inline void update(int k) { d[k] = node_pull(d[k << 1], d[k << 1 | 1]); }
  void all_apply(int k, F f) {
    d[k] = mapping(d[k], f);
    if(k < size) {
      lz[k] = tag_pull(lz[k], f);
    }
  }
  void push(int k) {
    all_apply(k << 1, lz[k]);
    all_apply(k << 1 | 1, lz[k]);
    lz[k] = tag_init();
  }
};
```

1.13 MoAlgo

```cpp
struct qry{
  int ql,qr,id;
};
template<class T>struct Mo{
  int n,m;
  vector<pii>ans;
  Mo(int _n,int _m): n(_n),m(_m){
    ans.resize(m);
  }
  void solve(vector<T>&v,vector<qry>&q){
    int l = 0,r = -1;
    vector<int>cnt,cntcnt;
    cnt.resize(n+5);
    cntcnt.resize(n+5);
    int mx = 0;
    function<void(int)>add = [&](int pos){
      cntcnt[cnt[v[pos]]]--;
      cnt[v[pos]]++;
      cntcnt[cnt[v[pos]]]++;
      mx = max(mx,cnt[v[pos]]);
    };
    function<void(int)>sub = [&](int pos){
      if(!--cntcnt[cnt[v[pos]]] and cnt[v[pos]]==mx)mx--;
      cnt[v[pos]]--;
      cntcnt[cnt[v[pos]]]++;
      mx = max(mx,cnt[v[pos]]);
    };
    sort(all(q),[&](qry a,qry b){
      static int B = max((int)1,n/max((int)sqrt(m),(int)1));
      if(a.ql/B!=b.ql/B)return a.ql<b.ql;
      if((a.ql/B)&1)return a.qr>b.qr;
      return a.qr<b.qr;
    });
    for(auto [ql,qr,id]:q){
      while(l>ql)add(--l);
      while(r<qr)add(++r);
      while(l<ql)sub(l++);
      while(r>qr)sub(r--);
      ans[id] = {mx,cntcnt[mx]};
    }
  }
};
```

1.14 Hash

```cpp
struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    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);
  }
  size_t operator()(pair<uint64_t,uint64_t> x) const {
    static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
    return splitmix64(3*x.first + x.second + FIXED_RANDOM);
  }
};
template<class T,class U>using hash_map = gp_hash_table<T,U,custom_hash>;
```

1.15 RedBlackTree

```cpp
template<class T, typename cmp=less<>>struct _tree{//#include<bits/extc++.h>
  tree<pair<T,int>,null_type,cmp,rb_tree_tag,tree_order_statistics_node_update>st;
  int id = 0;
  void insert(T x){st.insert({x,id++});}
  void erase(T x){st.erase(st.lower_bound({x,0}));}
  int order_of_key(T x){return st.order_of_key(*st.lower_bound({x,0}));}
  T find_by_order(int x){return st.find_by_order(x)->first;}
  T lower_bound(T x){return st.lower_bound({x,0})->first;}
  T upper_bound(T x){return st.upper_bound({x,(int)1e9+7})->first;}
  T smaller_bound(T x){return (--st.lower_bound({x,0}))->first;}
};
```

2 Geometry

2.1 Theorem

- Pick′ s Theorem

  $A = I + \frac{B}{2} - 1$

  $A := Area$

  $i := PointsInside$

  $B := PointsBoundary$

2.2 PointInPolygon

```cpp
template<class T>
int PointInPolygon(const vector<Point<T>> &Poly, const Point<T> p){
  int ans = 0;
  for(auto a = --Poly.end(),b = Poly.begin();b!=Poly.end();a = b++){
    if(PointOnSegment(*a,*b,p)){
      return -1;
    }
    if(seg_intersect(p,p+Point<T>(2e9+7,1),*a,*b)){
      ans = !ans;
    }
  }
  return ans;
}
```

2.3 PointInConvex

```cpp
template<class T>
int PointInConvex(const vector<Point<T>>&C,const Point<T>&p){
  if(btw(C[0],C[1],p) || btw(C[0],C.back(),p))return -1;
  int l = 0,r = (int)C.size()-1;
  while(l<=r){
    int m = (l+r)>>1;
    auto a1 = (C[m]-C[0])^(p-C[0]);
    auto a2 = (C[(m+1)%C.size()]-C[0])^(p-C[0]);
    if(a1>=0 and a2<=0){
      auto res = (C[(m+1)%C.size()]-C[m])^(p-C[m]);
      return res > 0 ? 1 : (res >= 0 ? -1 : 0);
    }
    if(a1 < 0) r = m-1;
    else l = m+1;
  }
  return 0;
}
```

2.4 MaximumDistance

```cpp
template<class T>
T MaximumDistance(vector<Point<T>>&p){
  vector<Point<T>>C = ConvexHull(p,0);
  int n = C.size(),t = 2;
  T ans = 0;
  for(int i = 0;i<n;i++){
    while(((C[i] - C[t]) ^ (C[(i+1)%n] - C[t])) < ((C[i] - C[(t+1)%n]) ^ (C[(i+1)%n] - C[(t+1)%n]))) t = (t + 1)%n;
    ans = max({ans, abs2(C[i] - C[t]), abs2(C[(i+1)%n] - C[t])});
  }
  return ans;
}
```

2.5 PolarAngleSort

```cpp
template<class T>
bool cmp(const Point<T> &a,const Point<T> &b){
  int lhs = (a.y < 0 || a.y==0 && a.x > 0) ? 0 : (1 + (a.x != 0 || a.y != 0));
  int rhs = (b.y < 0 || b.y==0 && b.x > 0) ? 0 : (1 + (b.x != 0 || b.y != 0));
  if(lhs != rhs) {
    return lhs < rhs;
  }
  long long area = (a^b);
  return area ? area > 0 : abs(a.x) + abs(a.y) < abs(b.x) + abs(b.y);
}
```

2.6 MinimumDistance

```cpp
template<class T>
T MinimumDistance(vector<Point<T>>&p,int l = -1,int r = -1){
  if(l==-1 and r==-1){
    sort(p.begin(),p.end(),[](Point<T> a,Point<T> b){
      if(a.x!=b.x)return a.x<b.x;
      return a.y<b.y;
    });
    p.erase(unique(p.begin(),p.end()),p.end());
    return MinimumDistance(p,0,p.size()-1);
  }
  if(l==r)return numeric_limits<T>::max();
  int m = (l+r)>>1,mid_pos = p[m].x;
  T ans = min(MinimumDistance(p,l,m),MinimumDistance(p,m+1,r));
  vector<Point<T>>tmp((r-l+1),Point<T>(0,0));
  merge(p.begin()+l,p.begin()+m+1, p.begin()+m+1,p.begin()+r+1, tmp.begin(), [](Point<T> a,Point<T> b){return a.y<b.y;});
  for(int i = l;i<=r;++i)p[i] = tmp[i-l];
  tmp.clear();
  for(int i = l;i<=r;++i){
    if((p[i].x-mid_pos)*(p[i].x-mid_pos)<ans){
      tmp.push_back(p[i]);
    }
  }
  int n = tmp.size();
  for(int i = 0;i<n;++i){
    for(int j = i+1;j<n;++j){
      ans = min(ans,abs2(tmp[i]-tmp[j]));
      if(((tmp[i].y-tmp[j].y)*(tmp[i].y-tmp[j].y))>ans){
        break;
      }
    }
  }
  return ans;
}
```

2.7 ConvexHull

```cpp
template<class T>
vector<Point<T>> ConvexHull(vector<Point<T>> v,bool Boundary = 1){
  sort(begin(v),end(v),[&](Point<T> &a,Point<T> &b){
    if(a.x!=b.x)return a.x<b.x;
    return a.y<b.y;
  });
  vector<Point<T>>ans;
  int t = 1;
  auto add = [&](Point<T> &p){
    while(ans.size() > t and ((p - ans[ans.size() - 2])^(ans.back() - ans[ans.size() - 2])) > (Boundary ? 0 : 0-eps))
      ans.pop_back();
    ans.push_back(p);
  };
  for(int i = 0; i < v.size(); ++i) add(v[i]);
  t = ans.size();
  for(int i = (int)(v.size())-2; i >= 0; --i) add(v[i]);
  if(v.size() > 1) ans.pop_back();
  return ans;
}
```

2.8 Template

```cpp
template<class T>
struct Point{
  T x,y;
  Point(T x = 0,T y = 0) : x(x), y(y) {}
  Point operator + (const Point &b) const {
    return Point(x + b.x,y + b.y);
  }
  Point operator - (const Point &b) const {
    return Point(x - b.x,y - b.y);
  }
  Point operator * (T b) const {
    return Point(x*b,y*b);
  }
  Point operator / (T b) const {
    return Point(x/b,y/b);
  }
  T operator * (const Point &b) const {
    return x * b.x + y * b.y;
  }
  T operator ^ (const Point &b) const {
    return x * b.y - y * b.x;
  }
};
int sign(double a){
  return fabs(a) < eps ? 0 : a > 0 ? 1 : -1;
}
template<class T>
double abs(const Point<T>&p){
  return sqrtl(p*p);
}
template<class T>
T abs2(const Point<T>&p){
  return p*p;
}
template<class T>
int ori(Point<T> a,Point<T> b,Point<T> c){
  return sign((b-a)^(c-a));
}
template<class T>
bool collinearity(Point<T> p1,Point<T> p2,Point<T> p3){
  return sign((p1-p3)^(p2-p3)) == 0;
}
template<class T>
bool btw(Point<T> p1,Point<T> p2,Point<T> p3) {
  if(!collinearity(p1, p2, p3)) return 0;
  return sign((p1-p3)*(p2-p3)) <= 0;
}
template<class T>
bool PointOnSegment(const Point<T> &p1,const Point<T> &p2, const Point<T> &p3){
  return collinearity(p1,p2,p3) && btw(p1,p2,p3);
}
template<class T>
bool seg_intersect(Point<T> p1, Point<T> p2, Point<T> p3, Point<T> p4) {
  int a123 = ori(p1, p2, p3);
  int a124 = ori(p1, p2, p4);
  int a341 = ori(p3, p4, p1);
  int a342 = ori(p3, p4, p2);
  if(a123 == 0 && a124 == 0)
```

```cpp
    return btw(p1, p2, p3) || btw(p1, p2, p4) || btw(p3, p4, p1) || btw(p3, p4, p2);
  return a123 * a124 <= 0 && a341 * a342 <= 0;
}
template<class T>
double area(vector<Point<T>> v){
  if(v.size()<=2)return 0;
  double ans = 0;
  for(int i = 1;i<v.size()-1;++i){
    ans+=((v[i]-v[0])^(v[i+1]-v[0]));
  }
  return abs(ans)/2.;
}
```

3 Graph

3.1 HLD

```cpp
struct HLD{
  int n,root;
  vector<int>dep,father,sz,mxson,topf,id;
  HLD(int _n,int _root,vector<vector<int>>&g): n(_n),root(_root){
    dep.resize(n+5);
    father.resize(n+5);
    sz.resize(n+5);
    mxson.resize(n+5);
    topf.resize(n+5);
    id.resize(n+5);
    function<void(int,int)>dfs = [&](int u,int p){
      dep[u] = dep[p]+1;
      father[u] = p;
      sz[u] = 1;
      mxson[u] = 0;
      for(auto v:g[u]){
        if(v!=p){
          dfs(v,u);
          sz[u]+=sz[v];
          if(sz[v]>sz[mxson[u]])mxson[u] = v;
        }
      }
    };
    function<void(int,int)>dfs2 = [&](int u,int top){
      static int idn = 0;
      topf[u] = top;
      id[u] = ++idn;
      if(mxson[u])dfs2(mxson[u],top);
      for(auto v:g[u]){
        if(v!=father[u] and v!=mxson[u]){
          dfs2(v,v);
        }
      }
    };
    dfs(root,0);
    dfs2(root,root);
  }
  int query(int u,int v,const auto &qry,const auto &op){
    int ans = 0;
    while(topf[u]!=topf[v]){
      if(dep[topf[u]]<dep[topf[v]])swap(u,v);
      ans = op(ans,qry(id[topf[u]],id[u]));
      u = father[topf[u]];
    }
    if(id[u]>id[v])swap(u,v);
    ans = op(ans,qry(id[u],id[v]));
    return ans;
  }
  void update(int u,int v,int val,const auto &upd){
    while(topf[u]!=topf[v]){
      if(dep[topf[u]]<dep[topf[v]])swap(u,v);
      upd(id[topf[u]],id[u],val);
      u = father[topf[u]];
    }
    if(id[u]>id[v])swap(u,v);
    upd(id[u],id[v],val);
  }
};
```

3.2 Bridges

```cpp
vector<pii> findBridges(const vector<vector<int>>& g) {
  int n = (int) g.size();
  vector<int> id(n, -1), low(n);
  vector<pii> bridges;
  function<void(int, int)> dfs = [&](int u, int p) {
    static int cnt = 0;
    id[u] = low[u] = cnt++;
    for(auto v : g[u]) {
      if(v == p) continue;
      if(id[v] != -1) low[u] = min(low[u], id[v]);
      else {
        dfs(v, u);
        low[u] = min(low[u], low[v]);
        if(low[v] > id[u]) bridges.EB(u, v);
      }
    }
  };
  for(int i = 0; i < n; ++i) {
    if(id[i] == -1) dfs(i, -1);
  }
  return bridges;
}
```

3.3 TwoSat

```cpp
struct two_sat{
  SCC s;
  vector<bool>ans;
  int have_ans = 0;
  int n;
  two_sat(int _n) : n(_n) {
    ans.resize(n+1);
    s = SCC(2*n);
  }
  int inv(int x){
    if(x>n)return x-n;
    return x+n;
  }
  void add_or_clause(int u, bool x, int v, bool y){
    if(!x)u = inv(u);
    if(!y)v = inv(v);
    s.add_edge(inv(u), v);
    s.add_edge(inv(v), u);
  }
  void check(){
    if(have_ans!=0)return;
    s.build();
    for(int i = 0;i<=n;++i){
      if(s.scc[i]==s.scc[inv(i)]){
        have_ans = -1;
        return;
      }
      ans[i] = (s.scc[i]<s.scc[inv(i)]);
    }
    have_ans = 1;
  }
};
```

3.4 MCMF

```cpp
template<class Cap_t, class Cost_t>
class MCMF {
public:
  struct Edge {
    int from;
    int to;
    Cap_t cap;
    Cost_t cost;
    Edge(int u, int v, Cap_t _cap, Cost_t _cost) : from(u), to(v), cap(_cap), cost(_cost) {}
  };

  static constexpr Cap_t EPS = static_cast<Cap_t>(1e-9);

  int n;
  vector<Edge> edges;
  vector<vector<int>> g;
  vector<Cost_t> d;
  vector<bool> in_queue;
  vector<int> previous_edge;

  MCMF() {}
  MCMF(int _n) : n(_n+1), g(_n+1), d(_n+1), in_queue(_n+1), previous_edge(_n+1) {}

  void add_edge(int u, int v, Cap_t cap, Cost_t cost) {
    assert(0 <= u && u < n);
    assert(0 <= v && v < n);
    g[u].push_back(edges.size());
    edges.emplace_back(u, v, cap, cost);
    g[v].push_back(edges.size());
    edges.emplace_back(v, u, 0, -cost);
  }

  bool spfa(int s, int t) {
    bool found = false;
    fill(d.begin(), d.end(), numeric_limits<Cost_t>::max());
    d[s] = 0;
    in_queue[s] = true;
    queue<int> que;
    que.push(s);
    while(!que.empty()) {
      int u = que.front();
      que.pop();
      if(u == t) {
        found = true;
      }
      in_queue[u] = false;
      for(auto& id : g[u]) {
        const Edge& e = edges[id];
        if(e.cap > EPS && d[u] + e.cost < d[e.to]) {
          d[e.to] = d[u] + e.cost;
          previous_edge[e.to] = id;
          if(!in_queue[e.to]) {
            que.push(e.to);
            in_queue[e.to] = true;
          }
        }
      }
    }
    return found;
  }

  pair<Cap_t, Cost_t> flow(int s, int t, Cap_t f =
       numeric_limits<Cap_t>::max()) {
    assert(0 <= s && s < n);
    assert(0 <= t && t < n);
    Cap_t cap = 0;
    Cost_t cost = 0;
    while(f > 0 && spfa(s, t)) {
      Cap_t send = f;
      int u = t;
      while(u != s) {
        const Edge& e = edges[previous_edge[u]];
        send = min(send, e.cap);
        u = e.from;
      }
      u = t;
      while(u != s) {
        Edge& e = edges[previous_edge[u]];
        e.cap -= send;
        Edge& b = edges[previous_edge[u] ^ 1];
        b.cap += send;
        u = e.from;
      }
      cap += send;
      f -= send;
      cost += send * d[t];
    }
    return make_pair(cap, cost);
  }
};
```

3.5 LCA

```
vector<vector<int>>g,dp;
vector<int>deep;
void build(int root,int n){
  dp.assign(25,vector<int>(n+5));
  deep.assign(n+5,0);
  function<void(int,int,int)>dfs = [&](int u,int p,int dis){
    dp[0][u] = p;
    deep[u] = dis;
    for(auto v:g[u]){
      if(v==p)continue;
      dfs(v,u,dis+1);
    }
  };
  dfs(root,0,1);
  for(int i = 1;i<=20;++i){
    for(int j = 1;j<=n;++j){
      dp[i][j] = dp[i-1][dp[i-1][j]];
    }
  }
}
int LCA(int u,int v){
  if(deep[u]<deep[v])swap(u,v);
  for(int i = 20;i>=0;--i){
    if(deep[dp[i][u]]>=deep[v])
      u = dp[i][u];
  }
  if(u==v)return u;
  for(int i = 20;i>=0;--i){
    if(dp[i][u]!=dp[i][v])u = dp[i][u],v = dp[i][v];
  }
  return dp[0][u];
}
```

3.6 CentroidDecomposition

```
vector<vector<int>>g;
vector<int>sz,tmp;
vector<bool>vis;//visit_centroid
int tree_centroid(int u,int n){
  function<void(int,int)>dfs1 = [&](int u,int p){
    sz[u] = 1;
    for(auto v:g[u]){
      if(v==p)continue;
      if(vis[v])continue;
      dfs1(v,u);
      sz[u]+=sz[v];
    }
  };
  function<int(int,int)>dfs2 = [&](int u,int p){
    for(auto v:g[u]){
      if(v==p)continue;
      if(vis[v])continue;
      if(sz[v]*2<n)continue;
      return dfs2(v,u);
    }
    return u;
  };
  dfs1(u,-1);
  return dfs2(u,-1);
}
int cal(int u,int p = -1,int deep = 1){
  int ans = 0;
  tmp.pb(deep);
  sz[u] = 1;
  for(auto v:g[u]){
    if(v==p)continue;
    if(vis[v])continue;
    ans+=cal(v,u,deep+1);
    sz[u]+=sz[v];
  }
  //calcuate the answer
  return ans;
}
int centroid_decomposition(int u,int tree_size){
  int center = tree_centroid(u,tree_size);
  vis[center] = 1;
  int ans = 0;
  for(auto v:g[center]){
    if(vis[v])continue;
    ans+=cal(v);
    for(int i = sz(tmp)-sz[v];i<sz(tmp);++i){
      //update
    }
  }
  while(!tmp.empty()){
    //roll_back(tmp.back())
    tmp.pop_back();
  }
  for(auto v:g[center]){
    if(vis[v])continue;
    ans+=centroid_decomposition(v,sz[v]);
  }
  return ans;
}
```

3.7 BCC AP

```
struct BCC_AP{
  int dfn_cnt = 0,bcc_cnt = 0,n;
  vector<int>dfn,low,ap,bcc_id;
  stack<int>st;
  vector<bool>vis,is_ap;
  vector<vector<int>>bcc;
  BCC_AP(int _n):n(_n){
    dfn.resize(n+5),low.resize(n+5),bcc.resize(n+5),vis.resize(
         n+5),is_ap.resize(n+5),bcc_id.resize(n+5);
  }
  inline void build(const vector<vector<int>>&g,int u,int p =
       -1){
    int child = 0;
    dfn[u] = low[u] = ++dfn_cnt;
    st.push(u);
    vis[u] = 1;
    if(g[u].empty() and p==-1){
      bcc_id[u] = ++bcc_cnt;
      bcc[bcc_cnt].push_back(u);
      return;
    }
    for(auto v:g[u]){
      if(v==p)continue;
      if(!dfn[v]){
        build(g,v,u);
        child++;
        if(dfn[u]<=low[v]){
          is_ap[u] = 1;
          bcc_id[u] = ++bcc_cnt;
          bcc[bcc_cnt].push_back(u);
          while(vis[v]){
            bcc_id[st.top()] = bcc_cnt;
            bcc[bcc_cnt].push_back(st.top());
            vis[st.top()] = 0;
            st.pop();
          }
        }
        low[u] = min(low[u],low[v]);
      }
      low[u] = min(low[u],dfn[v]);
    }
    if(p==-1 and child<2)is_ap[u] = 0;
    if(is_ap[u])ap.push_back(u);
  }
};
```

3.8 SCC

```
struct SCC{
  int n,cnt = 0,dfn_cnt = 0;
  vector<vector<int>>g;
  vector<int>sz,scc,low,dfn;
  stack<int>st;
  vector<bool>vis;
  SCC(int _n = 0) : n(_n){
    sz.resize(n+5),scc.resize(n+5),low.resize(n+5),dfn.resize(n+5),vis.resize(n+5);
    g.resize(n+5);
  }
  inline void add_edge(int u, int v){
    g[u].push_back(v);
  }
  inline void build(){
    function<void(int, int)>dfs = [&](int u,int dis){
      low[u] = dfn[u] = ++dfn_cnt,vis[u] = 1;
      st.push(u);
      for(auto v:g[u]){
        if(!dfn[v]){
          dfs(v, dis+1);
          low[u] = min(low[u],low[v]);
        }
        else if(vis[v]){
          low[u] = min(low[u],dfn[v]);
        }
      }
      if(low[u]==dfn[u]){
        ++cnt;
        while(vis[u]){
          auto v = st.top();
          st.pop();
          vis[v] = 0;
          scc[v] = cnt;
          sz[cnt]++;
        }
      }
    };
    for(int i = 0;i<=n;++i){
      if(!scc[i]){
        dfs(i, 1);
      }
    }
  }
  vector<vector<int>> compress(){
    vector<vector<int>>ans(cnt+1);
    for(int u = 0;u<=n;++u){
      for(auto v:g[u]){
        if(scc[u] == scc[v]){
          continue;
        }
        ans[scc[u]].push_back(scc[v]);
      }
    }
    for(int i = 0;i<=cnt;++i){
      sort(ans[i].begin(), ans[i].end());
      ans[i].erase(unique(ans[i].begin(), ans[i].end()), ans[i].end());
    }
    return ans;
  }
};
```

3.9 LineContainer

```
template<class T>
T floor_div(T a, T b) {
  return a / b - ((a ^ b) < 0 && a % b != 0);
}

template<class T>
T ceil_div(T a, T b) {
  return a / b + ((a ^ b) > 0 && a % b != 0);
}

namespace line_container_internal {

struct line_t {
  mutable long long k, m, p;

  inline bool operator<(const line_t& o) const { return k < o.k; }
  inline bool operator<(long long x) const { return p < x; }
};

} // line_container_internal

template<bool MAX>
struct line_container : std::multiset<line_container_internal::line_t, std::less<>> {
  static const long long INF = std::numeric_limits<long long>::max();

  bool isect(iterator x, iterator y) {
    if(y == end()) {
      x->p = INF;
      return 0;
    }
    if(x->k == y->k) {
      x->p = (x->m > y->m ? INF : -INF);
    } else {
      x->p = floor_div(y->m - x->m, x->k - y->k);
    }
    return x->p >= y->p;
  }

  void add_line(long long k, long long m) {
    if(!MAX) {
      k = -k;
      m = -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));
    }
  }

  long long get(long long x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return (l.k * x + l.m) * (MAX ? +1 : -1);
  }
};
```

3.10 Dinic

```
template<class T>
struct Dinic{
  struct edge{
    int from, to;
    T cap;
    edge(int _from, int _to, T _cap) : from(_from), to(_to), cap(_cap) {}
  };
  int n;
  vector<edge> edges;
  vector<vector<int>> g;
  vector<int> cur, h;
  Dinic(int _n) : n(_n+1), g(_n+1) {}
  void add_edge(int u, int v, T cap){
    g[u].push_back(edges.size());
    edges.push_back(edge(u, v, cap));
    g[v].push_back(edges.size());
    edges.push_back(edge(v, u, 0));
  }
  bool bfs(int s,int t){
    h.assign(n, -1);
    h[s] = 0;
    queue<int> que;
    que.push(s);
    while(!que.empty()) {
      int u = que.front();
      que.pop();
      for(auto id : g[u]) {
        const edge& e = edges[id];
        int v = e.to;
        if(e.cap > 0 && h[v] == -1) {
          h[v] = h[u] + 1;
          if(v == t) {
            return 1;
          }
          que.push(v);
        }
```

```cpp
      }
    }
    return 0;
  }
  T dfs(int u, int t, T f) {
    if(u == t) {
      return f;
    }
    T r = f;
    for(int& i = cur[u]; i < (int) g[u].size(); ++i) {
      int id = g[u][i];
      const edge& e = edges[id];
      int v = e.to;
      if(e.cap > 0 && h[v] == h[u] + 1) {
        T send = dfs(v, t, min(r, e.cap));
        edges[id].cap -= send;
        edges[id ^ 1].cap += send;
        r -= send;
        if(r == 0) {
          return f;
        }
      }
    }
    return f - r;
  }
  T flow(int s, int t, T f = numeric_limits<T>::max()) {
    T ans = 0;
    while(f > 0 && bfs(s, t)) {
      cur.assign(n, 0);
      T send = dfs(s, t, f);
      ans += send;
      f -= send;
    }
    return ans;
  }
  vector<pair<int,int>> min_cut(int s) {
    vector<bool> vis(n);
    vis[s] = true;
    queue<int> que;
    que.push(s);
    while(!que.empty()) {
      int u = que.front();
      que.pop();
      for(auto id : g[u]) {
        const auto& e = edges[id];
        int v = e.to;
        if(e.cap > 0 && !vis[v]) {
          vis[v] = true;
          que.push(v);
        }
      }
    }
    vector<pair<int,int>> cut;
    for(int i = 0; i < (int) edges.size(); i += 2) {
      const auto& e = edges[i];
      if(vis[e.from] && !vis[e.to]) {
        cut.push_back(make_pair(e.from, e.to));
      }
    }
    return cut;
  }
};
```

4 Math

4.1 Numbers

- Bernoulli numbers

  $B_0 - 1, B_1^{\pm} = \pm\frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$

  $$\sum_{j=0}^{m}\binom{m+1}{j}B_j = 0, \text{ EGF is } B(x) = \frac{x}{e^x - 1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}.$$

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

- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups.

  $S(n,k) = S(n-1,k-1) + kS(n-1,k), S(n,1) = S(n,n) = 1$

  $S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i}i^n$

  $x^n = \sum_{i=0}^{n} S(n,i)(x)_i$

- Pentagonal number theorem

  $$\prod_{n=1}^{\infty}(1-x^n) = 1 + \sum_{k=1}^{\infty}(-1)^k\left(x^{k(3k+1)/2} + x^{k(3k-1)/2}\right)$$

- Catalan numbers

  $$C_n^{(k)} = \frac{1}{(k-1)n+1}\binom{kn}{n}$$

  $C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k$

- Eulerian numbers

  Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

  $E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$

  $E(n,0) = E(n,n-1) = 1$

  $E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$

4.2 ExtendGCD

```cpp
// @return $x, y$ s.t. $ax + by = \gcd(a, b)$
ll ext_gcd(ll a, ll b, ll& x, ll& y) {
  if(b == 0) {
    x = 1; y = 0;
    return a;
  }
  ll x2, y2;
  ll c = a % b;
  if(c < 0) c += b;
  ll g = ext_gcd(b, c, x2, y2);
  x = y2;
  y = x2 - (a / b) * y2;
  return g;
}
```

4.3 BerlekampMassey

```cpp
template <typename Tfield>
std::pair<int, std::vector<Tfield>> find_linear_recurrence(const std::vector<Tfield> &S) {
    int N = S.size();
    using poly = std::vector<Tfield>;
    poly C_reversed{1}, B{1};
    int L = 0, m = 1;
    Tfield b = 1;

    // adjust: C(x) <- C(x) - (d / b) x^m B(x)
    auto adjust = [](poly C, const poly &B, Tfield d, Tfield b, int m) -> poly {
        C.resize(std::max(C.size(), B.size() + m));
        Tfield a = d / b;
        for (unsigned i = 0; i < B.size(); i++) C[i + m] -= a * B[i];
        return C;
    };

    for (int n = 0; n < N; n++) {
        Tfield d = S[n];
        for (int i = 1; i <= L; i++) d += C_reversed[i] * S[n - i];

        if (d == 0)
            m++;
        else if (2 * L <= n) {
            poly T = C_reversed;
            C_reversed = adjust(C_reversed, B, d, b, m);
            L = n + 1 - L;
            B = T;
            b = d;
            m = 1;
        } else
            C_reversed = adjust(C_reversed, B, d, b, m++);
    }
    return std::make_pair(L, C_reversed);
}

// Calculate $x^N \bmod f(x)$
// Known as `Kitamasa method`
// Input: f_reversed: monic, reversed (f_reversed[0] = 1)
// Complexity: $O(K^2 \log N)$ ($K$: deg. of $f$)
// Example: (4, [1, -1, -1]) -> [2, 3]
//          ( x^4 = (x^2 + x + 2)(x^2 - x - 1) + 3x + 2 )
// Reference: http://misawa.github.io/others/fast_kitamasa_method.html
//            http://sugarknri.hatenablog.com/entry/2017/11/18/233936
template <typename Tfield>
std::vector<Tfield> monomial_mod_polynomial(long long N, const std::vector<Tfield> &f_reversed) {
    assert(!f_reversed.empty() and f_reversed[0] == 1);
    int K = f_reversed.size() - 1;
    if (!K) return {};
    int D = 64 - __builtin_clzll(N);
    std::vector<Tfield> ret(K, 0);
    ret[0] = 1;
    auto self_conv = [](std::vector<Tfield> x) -> std::vector<Tfield> {
        int d = x.size();
        std::vector<Tfield> ret(d * 2 - 1);
        for (int i = 0; i < d; i++) {
            ret[i * 2] += x[i] * x[i];
            for (int j = 0; j < i; j++) ret[i + j] += x[i] * x[
                 j] * 2;
        }
        return ret;
    };
    for (int d = D; d--;) {
        ret = self_conv(ret);
        for (int i = 2 * K - 2; i >= K; i--) {
            for (int j = 1; j <= K; j++) ret[i - j] -= ret[i] *
                  f_reversed[j];
        }
        ret.resize(K);
        if ((N >> d) & 1) {
            std::vector<Tfield> c(K);
            c[0] = -ret[K - 1] * f_reversed[K];
            for (int i = 1; i < K; i++) { c[i] = ret[i - 1] -
                 ret[K - 1] * f_reversed[K - i]; }
            ret = c;
        }
    }
    return ret;
}

// Guess k-th element of the sequence, assuming linear
     recurrence
// initial_elements: 0-ORIGIN
// Verify: abc198f https://atcoder.jp/contests/abc198/
     submissions/21837815
template <typename Tfield>
Tfield guess_kth_term(const std::vector<Tfield> &
     initial_elements, long long k) {
    assert(k >= 0);
    if (k < static_cast<long long>(initial_elements.size()))
         return initial_elements[k];
    const auto f = find_linear_recurrence<Tfield>(
         initial_elements).second;
    const auto g = monomial_mod_polynomial<Tfield>(k, f);
    Tfield ret = 0;
    for (unsigned i = 0; i < g.size(); i++) ret += g[i] *
         initial_elements[i];
    return ret;
}
```

4.4 InvGCD

```
pair<long long, long long> inv_gcd(long long a, long long b) {
  a %= b;
  if(a < 0) a += b;
  if(a == 0) return {b, 0};
  long long s = b, t = a;
  long long m0 = 0, m1 = 1;
  while(t) {
    long long u = s / t;
    s -= t * u;
    m0 -= m1 * u;
    swap(s, t);
    swap(m0, m1);
  }
  if(m0 < 0) m0 += b / s;
  return {s, m0};
}
```

4.5 GeneratingFunctions

- Ordinary Generating Function $A(x) = \sum_{i \geq 0} a_i x^i$
    - $A(rx) \Rightarrow r^n a_n$
    - $A(x) + B(x) \Rightarrow a_n + b_n$
    - $A(x)B(x) \Rightarrow \sum_{i=0}^{n} a_i b_{n-i}$
    - $A(x)^k \Rightarrow \sum_{i_1+i_2+\cdots+i_k=n} a_{i_1} a_{i_2} \ldots a_{i_k}$
    - $xA(x)' \Rightarrow na_n$
    - $\frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^{n} a_i$
- Exponential Generating Function $A(x) = \sum_{i \geq 0} \frac{a_i}{i!} x_i$
    - $A(x) + B(x) \Rightarrow a_n + b_n$
    - $A^{(k)}(x) \Rightarrow a_{n+k}$
    - $A(x)B(x) \Rightarrow \sum_{i=0}^{n} \binom{n}{i} a_i b_{n-i}$
    - $A(x)^k \Rightarrow \sum_{i_1+i_2+\cdots+i_k=n} \binom{n}{i_1, i_2, \ldots, i_k} a_{i_1} a_{i_2} \ldots a_{i_k}$
    - $xA(x) \Rightarrow na_n$
- Special Generating Function
    - $(1+x)^n = \sum_{i \geq 0} \binom{n}{i} x^i$
    - $\frac{1}{(1-x)^n} = \sum_{i \geq 0} \binom{i}{n-1} x^i$

4.6 Theorem

- Cramer's rule

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \Rightarrow \begin{aligned} x &= \frac{ed - bf}{ad - bc} \\ y &= \frac{af - ec}{ad - bc} \end{aligned}$$

- Kirchhoff's Theorem

  Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G, where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G.

    - The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
    - The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

- Tutte's Matrix

  Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{rank(D)}{2}$ is the maximum matching on G.

- Cayley's Formula
    - Given a degree sequence $d_1, d_2, \ldots, d_n$ for each labeled vertices, there are $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$ spanning trees.
    - Let $T_{n,k}$ be the number of labeled forests on n vertices with k components, such that vertex $1, 2, \ldots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

- Erd□s–Gallai theorem

  A sequence of nonnegative integers $d_1 \geq \cdots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + \cdots + d_n$ is even and $\sum_{i-1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$ holds for every $1 \leq k \leq n$.

- Gale–Ryser theorem

  A pair of sequences of nonnegative integers $a_1 \geq \cdots \geq a_n$ and $b_1, \ldots, b_n$ is bigraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{n} \min(b_i, k)$ holds for every $1 \leq k \leq n$.

- Fulkerson–Chen–Anstee theorem

  A sequence $(a_1, b_1), \ldots, (a_n, b_n)$ of nonnegative integer pairs with $a_1 \geq \cdots \geq a_n$ is digraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{k} \min(b_i, k-1) + \sum_{i=k+1}^{n} \min(b_i, k)$ holds for every $1 \leq k \leq n$.

- M□bius inversion formula
    - $f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f(\frac{n}{d})$
    - $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$

- Spherical cap
    - A portion of a sphere cut off by a plane.
    - r: sphere radius, a: radius of the base of the cap, h: height of the cap, θ: $\arcsin(a/r)$.
    - Volume $= \pi h^2(3r - h)/3 = \pi h(3a^2 + h^2)/6 = \pi r^3(2 + \cos\theta)(1 - \cos\theta)^2/3$.
    - Area $= 2\pi rh = \pi(a^2 + h^2) = 2\pi r^2(1 - \cos\theta)$.

4.7 FloorSum

```
// @param $n < 2^{32}$
// @param $1 \leq m < 2^{32}$
// @return $\sum_{i=0}^{n-1} \lfloor \frac{ai + b}{m} \rfloor \
     pmod{2^{64}}$
ull floor_sum_unsigned(ull n, ull m, ull a, ull b) {
  ull ans = 0;
  while(true) {
    if(a >= m) {
      ans += n * (n - 1) / 2 * (a / m);
      a %= m;
    }
    if(b >= m) {
      ans += n * (b / m);
      b %= m;
    }
    ull y_max = a * n + b;
    if(y_max < m) break;
    n = (ull)(y_max / m);
    b = (ull)(y_max % m);
    swap(m, a);
  }
  return ans;
}

ll floor_sum(ll n, ll m, ll a, ll b) {
  assert(0 <= n && n < (1LL << 32));
  assert(1 <= m && m < (1LL << 32));
  ull ans = 0;
  if(a < 0) {
    ull a2 = (a % m + m) % m;
    ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
    a = a2;
  }
  if(b < 0) {
    ull b2 = (b % m + m) % m;
    ans -= 1ULL * n * ((b2 - b) / m);
    b = b2;
  }
  return ans + floor_sum_unsigned(n, m, a, b);
}
```

4.8 Factorizer

```
template<class T>
vector<pair<T, int>> MergeFactors(const vector<pair<T, int>>& a, const vector<pair<T, int>>& b) {
  vector<pair<T, int>> c;
  int i = 0, j = 0;
  while(i < SZ(a) || j < SZ(b)) {
    if(i < SZ(a) && j < SZ(b) && a[i].F == b[j].F) {
      c.EB(a[i].F, a[i].S + b[j].S);
      ++i, ++j;
      continue;
    }
    if(j == SZ(b) || (i < SZ(a) && a[i].F < b[j].F)) c.PB(a[i++]);
    else c.PB(b[j++]);
  }
  return c;
}
template<class T>
vector<pair<T, int>> RhoC(const T& n, const T& c) {
  if(n <= 1) return {};
  if(n % 2 == 0) return MergeFactors({{2, 1}}, RhoC(n / 2, c));
  if(is_prime_constexpr(n)) return {{n, 1}};
  T x = 2, saved = 2, p = 1, lam = 1;
  while(true) {
    x = (x * x % n + c) % n;
    T g = __gcd(((x - saved) + n) % n, n);
    if(g != 1) return MergeFactors(RhoC(g, c + 1), RhoC(n / g, c + 1));
    if(p == lam) {
      saved = x;
      p <<= 1;
      lam = 0;
    }
    lam += 1;
  }
  return {};
}
template<class T>
vector<pair<T, int>> Factorize(T n) {
  if(n <= 1) return {};
  return RhoC(n, T(1));
}
template<class T>
vector<T> BuildDivisorsFromFactors(const vector<pair<T, int>>& factors) {
  int total = 1;
  for(int i = 0; i < SZ(factors); ++i) total *= factors[i].second + 1;
  vector<T> divisors;
  divisors.reserve(total);
  divisors.PB(1);
  for(auto [p, cnt] : factors) {
    int sz = SZ(divisors);
    for(int i = 0; i < sz; ++i) {
      T cur = divisors[i];
      for(int j = 0; j < cnt; ++j) {
        cur *= p;
        divisors.PB(cur);
      }
    }
  }
  // sort(ALL(divisors));
  return divisors;
}
```

4.9 PowMod

```
constexpr long long Pow(long long x, long long n, int m) {
  if(m == 1) return 0;
  unsigned int _m = (unsigned int)(m);
  unsigned long long r = 1;
  x %= m;
  if(x < 0) x += m;
  unsigned long long y = x;
  while(n) {
    if(n & 1) r = (r * y) % _m;
    y = (y * y) % _m;
    n >>= 1;
  }
  return r;
}
```

4.10 CRT

```
// @return
//  $\text{remainder, modulo}$
//      or
//  $0, 0$ if do not exist
pair<long long, long long> crt(const vector<long long>& r, const vector<long long>& m) {
  assert(r.size()==m.size());
  int n = r.size();
  // Contracts: 0 <= r0 < m0
  long long r0 = 0, m0 = 1;
  for(int i = 0; i < n; i++) {
    assert(1 <= m[i]);
    long long r1 = r[i] % m[i];
    if(r1 < 0) r1 += m[i];
    long long m1 = m[i];
    if(m0 < m1) {
      swap(r0, r1);
      swap(m0, m1);
    }
    if(m0 % m1 == 0) {
      if(r0 % m1 != r1) return {0, 0};
      continue;
    }
    long long g, im;
    tie(g, im) = inv_gcd(m0, m1);
    long long u1 = (m1 / g);
    if((r1 - r0) % g) return {0, 0};
    long long x = (r1 - r0) / g % u1 * im % u1;
    r0 += x * m0;
    m0 *= u1;
    if(r0 < 0) r0 += m0;
  }
  return {r0, m0};
}
```

4.11 DiscreteLog

```
int DiscreteLog(int s, int x, int y, int m) {
  constexpr int K = 0;
  hash_map<int, int> p;
  int b = 1;
  for(int i = 0; i < K; ++i) {
    p[y] = i;
    y = 1LL * y * x % m;
    b = 1LL * b * x % m;
  }
  for(int i = 0; i < m + 10; i += K) {
    s = 1LL * s * b % m;
    if(p.find(s) != p.end()) return i + K - p[s];
  }
  return -1;
}
int DiscreteLog(int x, int y, int m) {
  if(m == 1) return 0;
  int s = 1;
  for(int i = 0; i < 100; ++i) {
    if(s == y) return i;
    s = 1LL * s * x % m;
  }
  if(s == y) return 100;
  int p = 100 + DiscreteLog(s, x, y, m);
  return (pow_mod(x, p, m) != y ? -1 : p);
}
```

4.12 LinearSieve

```
vector<bool> is_prime;
vector<int> primes, phi, mobius, least;
void linear_sieve(int n) {
  n += 1;
  is_prime.resize(n);
  least.resize(n);
  fill(2 + begin(is_prime),end(is_prime), true);
  phi.resize(n); mobius.resize(n);
  phi[1] = mobius[1] = 1;
  least[0] = 0,least[1] = 1;
  for(int i = 2; i < n; ++i) {
    if(is_prime[i]) {
      primes.push_back(i);
      phi[i] = i - 1;
      mobius[i] = -1;
      least[i] = i;
    }
    for(auto j : primes) {
      if(i * j >= n) break;
      is_prime[i * j] = false;
      least[i * j] = j;
      if(i % j == 0) {
        mobius[i * j] = 0;
        phi[i * j] = phi[i] * j;
        break;
      } else {
        mobius[i * j] = mobius[i] * mobius[j];
        phi[i * j] = phi[i] * phi[j];
      }
    }
  }
}
```

5 Misc

5.1 FastIO

```
inline char gc() {
    static const int BUF_SIZE = 1 << 22;
    static int Counts = 1 << 23;
    static char Buffer[BUF_SIZE];
    static char *Pointer = Buffer, *End = Buffer;
    if(Pointer == End) {
        if(Counts < BUF_SIZE) {
            return EOF;
        }
        Counts = fread(Buffer, 1, BUF_SIZE, stdin);
        Pointer = Buffer;
        End = Buffer + Counts;
    }
    return *(Pointer++);
}

template<class T>
inline void read(T& x) {
    static char c;
    do {
        c = gc();
    } while(c < '0' && c != '-');
    bool neg = (c == '-');
    if(!neg) {
        x = c - '0';
    } else x = 0;
    while((c = gc()) >= '0') {
        x = (x << 3) + (x << 1) + (c & 15);
    }
    if(neg) {
        x = -x;
    }
}

template<class T, class... U>
inline void read(T& a, U&... b) {
    read(a);
    read(b...);
}

template<class T>
inline void write(T temp, char end = '\n') {
    static short digits[20], P;
    if(temp == 0) {
        putchar_unlocked('0');
        putchar_unlocked(end);
        return;
    }
    if(temp < 0) {
        putchar_unlocked('-');
        write(-temp,end);
        return;
    }
    P = -1;
    while(temp) {
        digits[++P] = temp % 10;
        temp /= 10;
    }
    while(P >= 0) {
        putchar_unlocked(digits[P--] + '0');
    }
    putchar_unlocked(end);
    return;
}
```

5.2 Debug

```
#ifdef LOCAL
  #define eprintf(...) { fprintf(stderr, __VA_ARGS__); fflush(stderr); }
#else
  #define eprintf(...) 42
#endif
```

5.3 Discrete

```
template<class T>
vector<int> Discrete(const vector<T>&v){
  vector<int>ans;
  vector<T>tmp(v);
  sort(begin(tmp),end(tmp));
  tmp.erase(unique(begin(tmp),end(tmp)),end(tmp));
  for(auto i:v)ans.push_back(lower_bound(begin(tmp),end(tmp),i)-tmp.begin()+1);
  return ans;
}
```

5.4 DuiPai

```
#include<bits/stdc++.h>
using namespace std;
int main(){
  string sol,bf,make;
  cout<<"Your solution file name :";
  cin>>sol;
  cout<<"Brute force file name :";
  cin>>bf;
  cout<<"Make data file name :";
  cin>>make;
  system(("g++ "+sol+" -o sol").c_str());
  system(("g++ "+bf+" -o bf").c_str());
  system(("g++ "+make+" -o make").c_str());
  for(int t = 0;t<10000;++t){
    system("./make > ./1.in");
    double st = clock();
        system("./sol < ./1.in > ./1.ans");
        double et = clock();
        system("./bf < ./1.in > ./1.out");
        if(system("diff ./1.out ./1.ans")) {
      printf("\033[0;31mWrong Answer\033[0m on test #%d",t);
            return 0;
        }
    else if(et-st>=2000){
      printf("\033[0;32mTime limit exceeded\033[0m on test #%d, Time %.0lfms\n",t,et-st);
      return 0;
    }
    else {
            printf("\033[0;32mAccepted\033[0m on test #%d, Time %.0lfms\n", t, et - st);
        }
  }
}
```

5.5 Timer

```
const clock_t startTime = clock();
inline double getCurrentTime() {
  return (double) (clock() - startTime) / CLOCKS_PER_SEC;
}
```

5.6 TenarySearch

```
// return the maximum of $f(x)$ in $[l, r]$
double ternary_search(double l, double r) {
  while(r - l > EPS) {
    double m1 = l + (r - l) / 3;
    double m2 = r - (r - l) / 3;
    double f1 = f(m1), f2 = f(m2);
    if(f1 < f2) l = m1;
    else r = m2;
  }
  return f(l);
}

// return the maximum of $f(x)$ in $(l, r]$
int ternary_search(int l, int r) {
  while(r - l > 1) {
    int mid = (l + r) / 2;
    if(f(m) > f(m + 1)) r = m;
    else l = m;
  }
  return r;
}
```

6 Other

6.1 TouristIO

```
static struct FastInput {
  static constexpr int BUF_SIZE = 1 << 20;
  char buf[BUF_SIZE];
  size_t chars_read = 0;
  size_t buf_pos = 0;
  FILE *in = stdin;
  char cur = 0;

  inline char get_char() {
    if(buf_pos >= chars_read) {
      chars_read = fread(buf, 1, BUF_SIZE, in);
      buf_pos = 0;
      buf[0] = (chars_read == 0 ? -1 : buf[0]);
    }
    return cur = buf[buf_pos++];
    // return cur = getchar_unlocked();
  }

  inline void tie(int) {}

  inline explicit operator bool() {
    return cur != -1;
  }

  inline static bool is_blank(char c) {
    return c <= ' ';
  }

  inline bool skip_blanks() {
    while(is_blank(cur) && cur != -1) {
      get_char();
    }
    return cur != -1;
  }

  inline FastInput& operator>>(char& c) {
    skip_blanks();
    c = cur;
    return *this;
  }

  inline FastInput& operator>>(string& s) {
    if(skip_blanks()) {
      s.clear();
      do {
        s += cur;
      } while(!is_blank(get_char()));
    }
    return *this;
  }

  template<class T>
  inline FastInput& read_integer(T& n) {
    // unsafe, doesn't check that characters are actually digits
    n = 0;
    if(skip_blanks()) {
      int sign = +1;
      if(cur == '-') {
        sign = -1;
        get_char();
      }
      do {
        n += n + (n << 3) + cur - '0';
      } while(!is_blank(get_char()));
      n *= sign;
    }
    return *this;
  }

  template<class T>
  inline typename enable_if<is_integral<T>::value, FastInput&>::type operator>>(T& n) {
    return read_integer(n);
  }

  #if!defined(_WIN32) || defined(_WIN64)
  inline FastInput& operator>>(__int128& n) {
    return read_integer(n);
  }
  #endif

  template<class T>
  inline typename enable_if<is_floating_point<T>::value, FastInput&>::type operator>>(T& n) {
    // not sure ifreally fast, for compatibility only
    n = 0;
    if(skip_blanks()) {
      string s;
      (*this) >> s;
      sscanf(s.c_str(), "%lf", &n);
    }
    return *this;
  }
} fast_input;

#define cin fast_input

static struct FastOutput {
  static constexpr int BUF_SIZE = 1 << 20;
  char buf[BUF_SIZE];
  size_t buf_pos = 0;
  static constexpr int TMP_SIZE = 1 << 20;
  char tmp[TMP_SIZE];
  FILE *out = stdout;

  inline void put_char(char c) {
    buf[buf_pos++] = c;
    if(buf_pos == BUF_SIZE) {
      fwrite(buf, 1, buf_pos, out);
      buf_pos = 0;
    }
    // putchar_unlocked(c);
  }

  ~FastOutput() {
    fwrite(buf, 1, buf_pos, out);
  }

  inline FastOutput& operator<<(char c) {
    put_char(c);
    return *this;
  }

  inline FastOutput& operator<<(const char* s) {
    while(*s) {
      put_char(*s++);
    }
    return *this;
  }

  inline FastOutput& operator<<(const string& s) {
    for(int i = 0; i < (int) s.size(); i++) {
      put_char(s[i]);
    }
    return *this;
  }

  template<class T>
  inline char* integer_to_string(T n) {
    // beware of TMP_SIZE
    char* p = tmp + TMP_SIZE - 1;
    if(n == 0) {
      *--p = '0';
    } else {
      bool is_negative = false;
      if(n < 0) {
        is_negative = true;
        n = -n;
      }
      while(n > 0) {
        *--p = (char) ('0' + n % 10);
        n /= 10;
      }
      if(is_negative) {
        *--p = '-';
      }
    }
    return p;
  }

  template<class T>
  inline typename enable_if<is_integral<T>::value, char*>::type stringify(T n) {
    return integer_to_string(n);
  }

  #if!defined(_WIN32) || defined(_WIN64)
  inline char* stringify(__int128 n) {
    return integer_to_string(n);
  }
  #endif

  template<class T>
  inline typename enable_if<is_floating_point<T>::value, char*>::type stringify(T n) {
    sprintf(tmp, "%.17f", n);
    return tmp;
  }

  template<class T>
  inline FastOutput& operator<<(const T& n) {
    auto p = stringify(n);
    for(; *p != 0; p++) {
      put_char(*p);
    }
    return *this;
  }
} fast_output;

#define cout fast_output
```

7 Setup

7.1 Template

```
#include <bits/extc++.h>
#include <bits/stdc++.h>
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
#define IOS ios::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#define int long long
#define double long double
#define pb push_back
#define sz(x) (int)(x).size()
#define all(v) begin(v),end(v)
#define debug(x) cerr<<#x<<" = "<<x<<'\n'
#define LINE cout<<"\n-----------------\n"
#define endl '\n'
#define VI vector<int>
#define F first
#define S second
#define MP(a,b) make_pair(a,b)
#define rep(i,m,n) for(int i = m;i<=n;++i)
#define res(i,m,n) for(int i = m;i>=n;--i)
#define gcd(a,b) __gcd(a,b)
#define lcm(a,b) a*b/gcd(a,b)
#define Case() int _;cin>>_;for(int Case = 1;Case<=_;++Case)
#define pii pair<int,int>
using namespace __gnu_cxx;
using namespace __gnu_pbds;
using namespace std;
template <typename K, typename cmp = less<K>, typename T = thin_heap_tag> using _heap = __gnu_pbds::priority_queue<K, cmp, T>;
template <typename K, typename M = null_type> using _hash = gp_hash_table<K, M>;
const int N = 1e6+5,L = 20,mod = 1e9+7;
const long long inf = 2e18+5;
const double eps = 1e-7,pi = acos(-1);
void solve(){
}
signed main(){
  IOS;
  solve();
}
```

8 String

8.1 DynamicKMP

```
template<int ALPHABET, int (*f)(char)>
class DynamicKMP {
public:
  DynamicKMP() {}

  DynamicKMP(const string& s) {
    reserve(s.size());
    for(const char& c : s) {
      push(c);
    }
  }

  void push(char c) {
    int v = f(c);
    dp.emplace_back();
    dp.back()[v] = (int) dp.size();
    if(p.empty()) {
      p.push_back(0);
      return;
    }
    int i = (int) p.size();
    for(int j = 0; j < ALPHABET; ++j) {
      if(j == v) {
        p.push_back(dp[p[i - 1]][j]);
      } else {
        dp.back()[j] = dp[p[i - 1]][j];
      }
    }
  }

  void pop() {
    p.pop_back();
    dp.pop_back();
  }

  int query() const {
    return p.back();
  }

  vector<int> query_all() const {
    return p;
  }

  void reserve(int sz) {
    p.reserve(sz);
    dp.reserve(sz);
  }

private:
  vector<int> p;
  vector<array<int, ALPHABET>> dp;
};
```

8.2 RollingHash

```
template<int HASH_COUNT, bool PRECOMPUTE_POWERS = false>
class Hash {
public:
  static constexpr int MAX_HASH_PAIRS = 10;

  // {mul, mod}
  static constexpr const pair<int, int> HASH_PAIRS[] =
      {{827167801, 999999937}, {998244353, 999999929},
      {146672737, 922722049}, {204924373, 952311013},
      {585761567, 955873937}, {484547929, 901981687},
      {856009481, 987877511}, {852853249, 996724213},
      {937381759, 994523539}, {116508269, 993179543}};

  Hash() : Hash("") {}

  Hash(const string& s) : n(s.size()) {
    static_assert(HASH_COUNT > 0 && HASH_COUNT <= MAX_HASH_PAIRS);
    for(int i = 0; i < HASH_COUNT; ++i) {
      const auto& p = HASH_PAIRS[i];
      pref[i].resize(n);
      pref[i][0] = s[0];
      for(int j = 1; j < n; ++j) {
        pref[i][j] = (1LL * pref[i][j - 1] * p.first + s[j]) % p.second;
      }
    }
    if(PRECOMPUTE_POWERS) {
      build_powers(n);
    }
  }

  void add_char(char c) {
    for(int i = 0; i < HASH_COUNT; ++i) {
      const auto& p = HASH_PAIRS[i];
      pref[i].push_back((1LL * (n == 0 ? 0 : pref[i].back()) * p.first + c) % p.second);
    }
    n += 1;
    if(PRECOMPUTE_POWERS) {
      build_powers(n);
    }
  }

  // Return hash values for [l, r)
  array<int, HASH_COUNT> substr(int l, int r) {
    array<int, HASH_COUNT> res{};
    for(int i = 0; i < HASH_COUNT; ++i) {
      res[i] = substr(i, l, r);
    }
    return res;
  }

  array<int, HASH_COUNT> merge(const vector<pair<int, int>>& seg) {
    array<int, HASH_COUNT> res{};
    for(int i = 0; i < HASH_COUNT; ++i) {
      const auto& p = HASH_PAIRS[i];
      for(auto [l, r] : seg) {
        res[i] = (1LL * res[i] * get_power(i, r - l) + substr(i, l, r)) % p.second;
      }
    }
    return res;
  }

  // build powers up to x^k
  void build_powers(int k) {
    for(int i = 0; i < HASH_COUNT; ++i) {
      const auto& p = HASH_PAIRS[i];
      int sz = (int) POW[i].size();
      if(sz > k) {
        continue;
      }
      if(sz == 0) {
        POW[i].push_back(1);
        sz = 1;
      }
      while(sz <= k) {
        POW[i].push_back(1LL * POW[i].back() * p.first % p.second);
        sz += 1;
      }
    }
  }

  inline int size() const {
    return n;
  }

private:
  int n;
  static vector<int> POW[MAX_HASH_PAIRS];
  array<vector<int>, HASH_COUNT> pref;

  int substr(int k, int l, int r) {
    assert(0 <= k && k < HASH_COUNT);
    assert(0 <= l && l <= r && r <= n);
    const auto& p = HASH_PAIRS[k];
    if(l == r) {
      return 0;
    }
    int res = pref[k][r - 1];
    if(l > 0) {
      res -= 1LL * pref[k][l - 1] * get_power(k, r - l) % p.second;
    }
    if(res < 0) {
      res += p.second;
    }
    return res;
  }

  int get_power(int a, int b) {
    if(PRECOMPUTE_POWERS) {
      build_powers(b);
      return POW[a][b];
    }
    const auto& p = HASH_PAIRS[a];
    return power(p.first, b, p.second);
  }
};
template<int A, bool B> vector<int> Hash<A, B>::POW[Hash::MAX_HASH_PAIRS];
```

8.3 SuffixArray

```
struct suffix_array{
  int n;
  vector<int>SA,Rank,LCP;
  void counting_sort(vector<int>&v,auto getkey){
    int n = 0;
    for(auto i:v)n = max(n,getkey(i)+1);
    vector<int>bucket(n),ans(v.size());
    for(auto i:v)++bucket[getkey(i)];
    partial_sum(begin(bucket),end(bucket),begin(bucket));
    for(auto ite = v.rbegin();ite!=v.rend();++ite)ans[--bucket[getkey(*ite)]] = move(*ite);
    v.swap(ans);
    return;
  }
  suffix_array(string s):n(s.size()){
    SA.resize(n),Rank.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 = 0;i<n;++i){
      Rank[SA[i]] = (i?Rank[SA[i-1]]+(s[SA[i]]!=s[SA[i-1]]):SA[0]);
    }
    for(int k = 0;(1<<k)<=n;++k){
      vector<int>idx;
      for(int i = n-(1<<k);i<n;++i)idx.push_back(i);
      for(auto i:SA)if(i>=(1<<k))idx.push_back(i-(1<<k));
      counting_sort(idx,[&](int a){return Rank[a];});
      SA.swap(idx);
      vector<int>new_rank(n);
      new_rank[SA[0]] = 0;
      for(int i = 1;i<n;++i){
        auto cmp = [&](int a,int b){
          return Rank[a]!=Rank[b] or a+(1<<k)>=n or Rank[a+(1<<k)]!=Rank[b+(1<<k)];
        };
        new_rank[SA[i]] = new_rank[SA[i-1]]+cmp(SA[i-1],SA[i]);
      }
      Rank.swap(new_rank);
    }
    for(int i = 0,k = 0;i<n;++i){
      if(Rank[i]==0)continue;
      if(k)--k;
      while(i+k<n and SA[Rank[i]-1]+k<n and s[i+k]==s[SA[Rank[i]-1]+k])++k;
      LCP[Rank[i]] = k;
    }
  }
};
```

8.4 Trie

```
template<int ALPHABET = 26, char MIN_CHAR = 'a'>
class trie {
public:
  struct Node {
    int go[ALPHABET];
    Node() {
      memset(go, -1, sizeof(go));
    }
  };

  trie() {
    newNode();
  }

  inline int next(int p, int v) {
    return nodes[p].go[v] != -1 ? nodes[p].go[v] : nodes[p].go[v] = newNode();
  }

  inline void insert(const vector<int>& a, int p = 0) {
    for(int v : a) {
      p = next(p, v);
    }
  }

  inline void clear() {
    nodes.clear();
    newNode();
  }

  inline int longest_common_prefix(const vector<int>& a, int p = 0) const {
    int ans = 0;
    for(int v : a) {
      if(nodes[p].go[v] != -1) {
        ans += 1;
        p = nodes[p].go[v];
      } else {
        break;
      }
    }
    return ans;
  }

private:
  vector<Node> nodes;

  inline int newNode() {
    nodes.emplace_back();
    return (int) nodes.size() - 1;
  }
};
```

ACM ICPC Team Reference - LeeJiaHuaPlayMinecraft

Contents

ACM ICPC Judge Test - LeeJiaHuaPlayMinecraft

C++ Resource Test

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

namespace system_test {

const size_t KB = 1024;
const size_t MB = KB * 1024;
const size_t GB = MB * 1024;

size_t block_size, bound;
void stack_size_dfs(size_t depth = 1) {
  if (depth >= bound)
    return;
  int8_t ptr[block_size]; // 若無法編譯將 block_size 改成常數
  memset(ptr, 'a', block_size);
  cout << depth << endl;
  stack_size_dfs(depth + 1);
}

void stack_size_and_runtime_error(size_t block_size, size_t bound = 1024) {
  system_test::block_size = block_size;
  system_test::bound = bound;
  stack_size_dfs();
}

double speed(int iter_num) {
  const int block_size = 1024;
  volatile int A[block_size];
  auto begin = chrono::high_resolution_clock::now();
  while (iter_num--)
    for (int j = 0; j < block_size; ++j)
      A[j] += j;
  auto end = chrono::high_resolution_clock::now();
  chrono::duration<double> diff = end - begin;
  return diff.count();
}

void runtime_error_1() {
  // Segmentation fault
  int *ptr = nullptr;
  *(ptr + 7122) = 7122;
}

void runtime_error_2() {
  // Segmentation fault
  int *ptr = (int *)memset;
  *ptr = 7122;
}

void runtime_error_3() {
  // munmap_chunk(): invalid pointer
  int *ptr = (int *)memset;
  delete ptr;
}

void runtime_error_4() {
  // free(): invalid pointer
  int *ptr = new int[7122];
  ptr += 1;
  delete[] ptr;
}

void runtime_error_5() {
  // maybe illegal instruction
  int a = 7122, b = 0;
  cout << (a / b) << endl;
}

void runtime_error_6() {
  // floating point exception
  volatile int a = 7122, b = 0;
  cout << (a / b) << endl;
}

void runtime_error_7() {
  // call to abort.
  assert(false);
}

} // namespace system_test

#include <sys/resource.h>
void print_stack_limit() { // only work in Linux
  struct rlimit l;
  getrlimit(RLIMIT_STACK, &l);
  cout << "stack_size = " << l.rlim_cur << " byte" << endl;
}
```