ICPC notebook

ilyes GLAYS (INSAT)

1 Algorithms and Data Structures

1.1 Trie

```
struct node{
    struct node *next_zero , *next_one;
    int nb;
    node(){
        next_zero=NULL;
        next_one=NULL;
        nb=0;
    }
};
node* root = new node();

void add_number(int x){
    node* cur_node = root;
    ++cur_node->nb;

    for(int i = 29; i >= 0; --i){{
        if(x&(1 << i)){
            if(cur_node->next_one == NULL)
                cur_node->next_one = new node();
            cur_node=cur_node->next_one;
        }else{
            if(cur_node->next_zero == NULL)
                cur_node->next_zero = new node();
            cur_node=cur_node->next_zero;
        }
        ++cur_node->nb;
    }
}

void delete_number(int x){
    node *cur_node = root , *prv;
    --cur_node->nb;

 for(int i = 29; i >= 0; --i){{
        prv = cur_node;

        if(x&(1 << i)) cur_node = cur_node->
            next_one;
        else cur_node = cur_node->next_zero;

        --cur_node->nb;
        if(cur_node->nb == 0){
            if(x&(1 << i)) prv->next_one = NULL;
            else prv->next_zero = NULL;
            return;
        }
    }
}

int query_answer(int x){
    node* cur_node = root;

 for(int i = 29; i >= 0; --i){{
        if(x&(1 << i)){
            if(cur_node->next_zero != NULL)
                cur_node = cur_node->next_zero;
            else x -= (1 << i) , cur_node =
                cur_node->next_one;
        }else{
            if(cur_node->next_one != NULL) x += (1
                 << i) , cur_node = cur_node->
                next_one;
            else  cur_node = cur_node->next_zero;
        }
    }

    return x;
}
```

1.2 Treap

```
namespace Treap{
    mt19937 rng(chrono::steady_clock::now().
        time_since_epoch().count());
    struct node {
        int sz, prior;
        int val, sum, lazy;
        node* l,* r;
```

```
    node() { }
    node(int _val) {
        sz = 1; prior =
            uniform_int_distribution<int>(1,1
            e9)(rng);
        val = sum = _val; lazy = 0;
        l = r = NULL;
    }
};
typedef node* treap;

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

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

void propagate(treap t) {
    if(t && t->lazy!=0){
        t->val += t->lazy;
        if(t->l) (t->l)->lazy += t->lazy;
        if(t->r) (t->r)->lazy += t->lazy;
        t->lazy = 0;
    }
}

void update(treap t) {
    if(t){
        propagate(t);
        t->sz = 1 + sz(t->l) + sz(t->r);
        t->sum = t->val + (t->lazy)*(t->sz) +
            sum(t->l) + sum(t->r);
    }
}

void split(treap t, treap& l, treap& r, int
    key, int cum = 0) {
    update(t);
    int cur_key = t ? cum+sz(t->l)+1 : -1;
    if(!t) l = r = NULL;
    else if(cur_key < key) split(t->r, t->r,
        r, key, cur_key), l = t;
    else split(t->l, l, t->l, key, cum), r =
        t;
    update(l);
    update(r);
}

void merge(treap& t, treap l, treap r) {
    update(l);
    update(r);
    if(!l || !r) t = l ? l : r;
    else if(l->prior > r->prior) merge(l->r,
        l->r, r), t = l;
    else merge(r->l, l, r->l), t = r;
    update(t);
}

void insert(treap& t, treap newt, int key) {
    if(!t){ t = newt; return; }
    treap t1, t2;
    split(t, t1, t2, key);
    merge(t1, t1, newt);
    merge(t, t1, t2);
}

void erase(treap& t, int key, int cum = 0) {
    int cur_key = t ? cum+sz(t->l)+1 : -1;
    if(cur_key == key) {
        treap tmp = t;
        merge(t, t->l, t->r);
        delete tmp;
    }
    else if(key < cur_key) erase(t->l, key,
        cum);
    else erase(t->r, key, cur_key);
    update(t);
}

int sumRange(treap& t, int qL, int qR) {
        treap al, ar; split(t, al, ar, qL);
        treap bl, br; split(ar, bl, br, qR-qL+2);
        int ans = sum(bl);
        treap tmp; merge(tmp, al, bl);
        merge(t, tmp, br);
        return ans;
    }

    void addRange(treap& t, int qL, int qR, int
        qVal) {
        treap al, ar; split(t, al, ar, qL);
        treap bl, br; split(ar, bl, br, qR-qL+2);
        bl->lazy += qVal;
        treap tmp; merge(tmp, al, bl);
        merge(t, tmp, br);
    }

} using namespace Treap;
```

1.3 Fenwick Tree

```
struct Fenwick {
 int n; int arr[SIZE+2];
 Fenwick(){}
 Fenwick(int _n){ n=_n; memset(arr, 0, sizeof(arr
    )); }
 void add(int qI, int qVal) { while(qI <= n){ arr
    [qI] += qVal; qI += (qI & -qI); } }
 int sum(int qI) { int ans = 0; while(qI > 0){
    ans += arr[qI]; qI -= (qI & -qI); } return
    ans; }
 int sum(int qL, int qR){ return (qL <= qR) ?
    querySum(qR) - querySum(qL-1) : 0; }
};
```

1.4 Ordered Set

```
// K-th LARGEST       (0-indexed)   *s.
    find_by_order(K)
// NUMBER OF ELEMENTS < X          s.order_of_key
    (X)
// NUMBER OF ELEMENTS IN RANGE [L;R] s.
    order_of_key(r+1) - s.order_of_key(l)


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

template <typename T> using OrderedSet = tree<T,
    null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

template<typename T> struct OrderedMultiset{
 tree<pair<T,int>, null_type, less<pair<T,int>>,
     rb_tree_tag,
     tree_order_statistics_node_update> _s;
 int id = 1;
 void insert(const T &x){ _s.insert({x, id}); ++
     id; }
 void erase(const T &x){ auto it=_s.upper_bound({
     x, -1}); if(it!=_s.end() && it->first==x) _s
     .erase(it); }
 int count(const T &x){ return _s.order_of_key({x
     , 1e9}) - _s.order_of_key({x, -1}); }
 int count(const T &l, const T &r){ return _s.
     order_of_key({r, 1e9}) - _s.order_of_key({l,
      -1}); } // [l;r] inclusive
 T operator[](const int& ind){ assert(1 <= ind &&
      ind <= size(_s)); return _s.find_by_order(
     ind-1)->first; } // 1-indexed
};
```

1.5 Segment Tree

```
struct SegmentTree{
 int n, sum[4*N], lazy[4*N];
    SegmentTree(int nn){ n = 1<<(__lg(nn-1)+1); }

    void pushDown(int i, int st, int en){
        sum[i] += lazy[i] * (en-st+1);
        if(st != en){
            lazy[2*i] += lazy[i];
            lazy[2*i+1] += lazy[i];
        }
        lazy[i] = 0;
    }

    void pushUp(int i, int st, int en){
        sum[i] = sum[2*i] + sum[2*i+1];
    }

    int Sum(int qL, int qR){ return Sum(qL, qR,
        1, 1, n); }
 int Sum(int qL, int qR, int i, int st, int en) {
        if(lazy[i]) pushDown(i, st, en);
  if (en < qL || qR < st) return 0;
        if (qL <= st && en <= qR) return sum[i];

  int mid = (st + en)/2;
  int l = Sum(qL, qR, 2*i, st, mid);
  int r = Sum(qL, qR, 2*i+1, mid+1, en);
  return l + r;
 }

    void Add(int qL, int qR, int qVal){ Add(qL,
        qR, qVal, 1, 1, n); }
 void Add(int qL, int qR, int qVal, int i, int st
     , int en) {
        if(lazy[i]) pushDown(i, st, en);
        if (en < qL || qR < st) return;
        if (qL <= st && en <= qR){
            lazy[i] += qVal;
            pushDown(i, st, en);
            return;
        }

  int mid = (st + en)/2;
  Add(qL, qR, qVal, 2*i, st, mid);
  Add(qL, qR, qVal, 2*i+1, mid+1, en);

        pushUp(i, st, en);
 }
};
```

1.6 Sparse Table

```
template<typename T, int n> struct MiniTable{
 T arr[__lg(n)+1][n+1];
 T& base(int x){ return arr[0][x]; }
 void computeAll(){ for(int i=1; i<=lg[n]; ++i)
     for(int x=1; x+(1<<i)-1<=n; ++x) arr[i][x] =
      min(arr[i-1][x], arr[i-1][x+(1<<(i-1))]); }
 T get(int x, int y){ int lgLen=lg[y-x+1]; return
      min(arr[lgLen][x], arr[lgLen][y-(1<<lgLen)
     +1]); }
};

template<typename T, int n> struct SumTable{
 T arr[__lg(n)+1][n+1];
 T& base(int x){ return arr[0][x]; }
 void computeAll(){ for(int i=1; i<=lg; ++i) for(
     int x=1; x+(1<<i)-1<=n; ++x) arr[i][x] = arr
     [i-1][x]+arr[i-1][x+(1<<(i-1))]; }
 T get(int x, int y){ int len=y-x+1; T ans=0;
     while(len>0){ int i=lg[len]; len-=(1<<i);
     ans+=arr[i][x]; x+=(1<<i); } return ans; }
};


struct ParentTable{
 int arr[LG+1][N+1];
 int& base(int x){ return arr[0][x]; }
```

```cpp
void computeAll(int n){ for(int i=1; i<=lg[n]; ++i) for(int x=1; x<=n; ++x) arr[i][x] = arr[i-1][arr[i-1][x]]; }
int get(int x, int len){ while(len > 0){ int i= lg[len]; len -= (1<<i); x = arr[i][x]; } return x; }
};
```

1.7 Point / Vector class

```cpp
namespace Geo{
 const LD EPS = 1e-16L, PI = acos(-1.0L);
 bool EQ(LD a, LD b){return fabs(a-b) < EPS;}
 bool LT(LD a, LD b){return fabs(a-b) > EPS && a < b;}
 bool GT(LD a, LD b){return fabs(a-b) > EPS && a > b;}
 #define LTE(a,b) (!GT(a,b))
 #define GTE(a,b) (!LT(a,b))
 #define Vector Point
 struct Point {
  LD X, Y;
  Point(){}
  Point(LD _X, LD _Y): X(_X), Y(_Y){}
  bool operator==(const Point& oth) { return EQ(X, oth.X) && EQ(Y, oth.Y); }
  bool operator!=(const Point& oth) { return !EQ(X, oth.X) || !EQ(Y, oth.Y); }
  Point operator+(const Point& oth) { return Point( X + oth.X, Y + oth.Y ); }
  Point operator-(const Point& oth) { return Point( X - oth.X, Y - oth.Y ); }
  LD operator*(const Point& oth) { return X * oth.X + Y * oth.Y; }
  LD operator%(const Point& oth) { return X * oth.Y - Y * oth.X; }
  Point operator*(LD f) { return Point( X * f, Y * f ); }
  Point operator/(LD f) { return Point( X / f, Y / f ); }
 };
 struct Line{
  LD a, b, c; Point p1, p2;
  Line(Point _p1, Point _p2) { p1 = _p1; p2 = _p2; a = p1.Y - p2.Y; b = p2.X - p1.X; c = p1.X*p2.Y - p2.X*p1.Y; }
 };
 LD norm(Vector v) { return sqrtl(v.X*v.X + v.Y*v.Y); }
 LD norm2(Vector v) { return v.X*v.X + v.Y*v.Y; }
 Vector perp(Vector v) { return Vector(v.Y, -v.X); }
 bool ccw(Point a, Point b, Point c) { return GTE((a-b)%(a-c), 0); }
 LD angle(Vector v) { LD ang = atan2(v.Y, v.X); return LT(ang, 0) ? ang+2*PI : ang; }
 Vector rotate(Vector v, LD ang) { return Point(v.X * cos(ang) - v.Y * sin(ang), v.X * sin(ang) + v.Y * cos(ang)); } // returns vector v rotated 'ang' radians ccw around the origin
 bool insideLine(Point p, Point a, Point b){ return EQ((p-a)%(p-b), 0); }
 bool insideSegment(Point p, Point a, Point b) { return insideLine(p,a,b) && LTE((p-a)*(p-b), 0); }
 LD distLine(Point p, Point a, Point b){ return fabs((p-a) % (b-a)) / norm(b-a); }
 LD distSegment(Point p, Point a, Point b){
  if(LTE((p-a) * (b-a), 0)) return norm(p-a);
  if(LTE((p-b) * (a-b), 0)) return norm(p-b);
  return distLine(p, a, b);
 }
 int windingNumber(Point p, const vector<Point> &polygon) {
  int wn = 0;
  for(int i = 0; i < size(polygon); ++i) {
   Point a = polygon[i], b = polygon[(i+1)%size(polygon)];
   if (LTE(a.Y, p.Y) && LT(p.Y, b.Y) && ccw(a, b, p)) ++wn;
   if (LTE(b.Y, p.Y) && LT(p.Y, a.Y) && !ccw(a, b, p)) --wn;
  }
  return wn;
 }
 bool insidePolygon(Point p, const vector<Point> &polygon) { return windingNumber(p, polygon) != 0; }
 LD minSegmentDist(Point a, Vector va, Point b, Vector vb){ // at instant t (in [0;1]) point a will be a + va*t and b will be b + vb*t
  if(va == vb) return norm(a - b);
  LD t = -(Vector(a - b) * Vector(va - vb)) / norm2(va - vb);
  if(LT(t, 0.0L)) t = 0.0L;
  if(GT(t, 1.0L)) t = 1.0L;
  return norm((a + va*t) - (b + vb*t));
 }
 bool intersectLines(Point a, Point b, Point c, Point d, Point&ans) {
  LD A1 = b.Y - a.Y, B1 = a.X - b.X, C1 = A1 * a.X + B1 * a.Y;
  LD A2 = d.Y - c.Y, B2 = c.X - d.X, C2 = A2 * c.X + B2 * c.Y;
  LD determinant = A1 * B2 - A2 * B1;
  if(EQ(determinant, 0)) return false; // parallel
  ans = Point((B2 * C1 - B1 * C2) /determinant, (A1 * C2 - A2 * C1) /determinant);
  return true;
 }
 Point getCircleCenter(Point& a, Point& b, Point& c){ // 3 NON COLINEAR POINTS
  Point ab = (a + b) * 0.5L;
  Point ab2 = ab + perp(b - a);
  Point bc = (b + c) * 0.5L;
```

```
 Point bc2 = bc + perp(c - b);
 Point ans;
 if(!intersectLines(ab, ab2, bc, bc2, ans))
     return Point(INFLL, INFLL);
 return ans;
}
pair<Point,LD> minEnclosingCircle(vector<Point>&
     pnts){     // O(n^4)
 Point c; LD r = INFLL;
 for(int i=0; i<size(pnts); ++i) for(int j=i+1;
     j<size(pnts); ++j){
  Point tmpC = (pnts[i] + pnts[j]) * 0.5L;
  LD tmpR = 0.0L;
  for(int ind=0; ind<size(pnts); ++ind) tmpR =
      max(tmpR, norm(tmpC - pnts[ind]));
  if(LT(tmpR,r)) r = tmpR, c = tmpC;
 }
 for(int i=0; i<size(pnts); ++i) for(int j=i+1;
     j<size(pnts); ++j) for(int k=j+1; k<size(
     pnts); ++k){
  Point tmpC = getCircleCenter(pnts[i], pnts[j],
      pnts[k]);
  LD tmpR = 0.0L;
  for(int ind=0; ind<size(pnts); ++ind) tmpR =
      max(tmpR, norm(tmpC - pnts[ind]));
  if(LT(tmpR,r)) r = tmpR, c = tmpC;
 }
 return {c, r};
}
} using namespace Geo;
```

1.8 Convex Hull

```
vector<Point> convexHull(vector<Point>&pnts) {
    if(size(pnts) == 1) return pnts;

    sort(begin(pnts), end(pnts), [](Point a,
        Point b) {
        return a.X < b.X || (a.X == b.X && a.Y <
            b.Y);
    });

    Point p1 = pnts[0];
    Point p2 = pnts.back();

    vector<Point> up, down;
    up.push_back(p1);
    down.push_back(p1);

    for(int i = 1; i < size(pnts); i++) {
        if (i == size(pnts) - 1 || cw(p1, pnts[i
            ], p2)) {
            while (size(up) >= 2 && !cw(up[size(up
                )-2], up[size(up)-1], pnts[i])) up
                .pop_back();
            up.push_back(pnts[i]);
        }

        if (i == a.size() - 1 || ccw(p1, a[i], p2
            )) {
            while(size(down) >= 2 && !ccw(down[
                size(down)-2], down[size(down)-1],
                 pnts[i])) down.pop_back();
            down.push_back([pnts[i]);
        }
    }

    vector<Point> hull;
    for(int i = 0; i < (int)up.size(); i++) hull.
        push_back(up[i]);
    for(int i = down.size() - 2; i > 0; i--) hull
        .push_back(down[i]);

    return hull;
}
```

1.9 Tarjan

```
// DIRECTED - FIND SCCs
struct Tarjan{
 int n; vector<int> *adj;
 int nbDfs, nbCmp;
 vector<int> dfsIndex, lowest, cmpID;
 vector<bool> inStck; vector<int> stck;
 Tarjan(int _n, vector<int> _adj[]) : n(_n), adj(
     _adj) {}
 void dfs(int i) {
  dfsIndex[i] = lowest[i] = ++nbDfs;
  stck.emplace_back(i);
  inStck[i] = true;
  for (int j: adj[i]) {
   if (dfsIndex[j] == -1) { dfs(j); lowest[i] =
       min(lowest[i], lowest[j]); }
   else if (inStck[j]) lowest[i] = min(lowest[i],
        lowest[j]);
  }
  if (dfsIndex[i] == lowest[i]) {
   ++nbCmp;
   for(;;) { int cur = stck.back(); cmpID[cur] =
       nbCmp; inStck[cur] = false; stck.pop_back
       (); if (cur == i) break; }
  }
 }
 void findSCCs() {
  nbDfs = 0; nbCmp = 0;
  dfsIndex.assign(n+1, -1); lowest.assign(n+1, 0)
      ; cmpID.assign(n+1, 0); inStck.assign(n+1,
      false); stck.clear();
  for (int i=1; i<=n; ++i) if (dfsIndex[i] == -1)
       dfs(i);
 }
};


// UNDIRECTED - FIND BRIDGEs
struct Tarjan{
```

```cpp
int n; vector<int> *adj;
int nbDfs;
vector<int> dfsIndex, lowest;
set<pii> bridges;
Tarjan(int _n, vector<int> _adj[]) : n(_n), adj(
    _adj) {}
void dfs(int u, int prv) {
 dfsIndex[u] = lowest[u] = ++nbDfs;
 for(int v: adj[u]) {
  if (dfsIndex[v] == -1) { dfs(v, u); lowest[u]
      = min(lowest[u], lowest[v]); if(lowest[v]
      == dfsIndex[v]) bridges.insert(minmax(u,v)
      ); }
  else if(v != prv) lowest[u] = min(lowest[u],
      lowest[v]);
 }
}
void findBridges() {
 nbDfs = 0;
 dfsIndex.assign(n+1, -1); lowest.assign(n+1, 0)
     ; bridges.clear();
 for (int i=1; i<=n; ++i) if (dfsIndex[i] == -1)
      dfs(i, i);
}
};
```

1.10 Centroid Decomposition

```cpp
struct CentroidTree {
      int n; vector<int> nbChild; vector<bool>
          done;
   vector<int> *adj;
      CentroidTree(int _n, vector<int> _adj[]) :
          adj(_adj){
            n = _n; nbChild.assign(n + 1, 0); done
                .assign(n + 1, false);
      }
      void dfs(int u, int prv) {
            nbChild[u] = 1;
            for (int v: adj[u]) if (v != prv && !
                done[v]){
    dfs(v, u);
    nbChild[u]+=nbChild[v];
   }
      }
      int getCentroid(int u, int prv, int nb) {
            int heavy = -1;
            for (int v: adj[u]) if (v != prv && !
                done[v]) {
                  if (heavy == -1 || nbChild[v] >
                      nbChild[heavy]) heavy = v;
            }
            if (heavy != -1 && nbChild[heavy] > nb
                / 2) return getCentroid(heavy, u,
                nb);
            else return u;
      }
      int decompose(int u) {
            dfs(u, u);
            int centroid = getCentroid(u, u,
                nbChild[u]);
            done[centroid] = true;

            int ans = 0;
   // compute ans of current centroid tree
            for (int nxt: adj[centroid]) if (!done
                [nxt]) ans += decompose(nxt);
            return ans;
      }
};
```

1.11 Dijkstra

```cpp
void dijkst(int _src, LL _dist[]){
 for(int i=1; i<=n; ++i) _dist[i] = INFLL;
 _dist[_src] = 0;
 set<pair<LL,int>> _sss;
 for(int i=1; i<=n; ++i) _sss.insert({_dist[i], i
     });
 while(!empty(_sss)){
  LL d; int u;
  tie(d, u) = *begin(_sss);
  _sss.erase(begin(_sss));
  for(auto [v, w]: adj[u]) if(_dist[v] > _dist[u]
       + w){
   _sss.erase({_dist[v], v});
   _dist[v] = _dist[u] + w;
   _sss.insert({_dist[v], v});
  }
 }
}
```

1.12 Disjoint Set Union

```cpp
struct Dsu {
    vector<int> _leader, _size;
    Dsu(int _n) { _leader.assign(_n+1, 0); _size.
        assign(_n+1, 0); for(int i=1; i<=_n; ++i)
         _leader[i]=i, _size[i]=1; }
    int leader(int x) {return (_leader[x]==x) ? x
         : _leader[x]=leader(_leader[x]);}
    void unite(int x, int y) { x = leader(x); y =
         leader(y); if(x != y) _leader[y] = x,
        _size[x] += _size[y]; }
};
```

1.13 Heavy Light Decomposition

```cpp
vector<int> parent, depth, size, heavy, head, pos
    ;
void dfs(int u, int p) {
    parent[u]=p, depth[u]=depth[p]+1, size[u]=1;
    int mxSize = 0;
    for (int v : g[u]) if (v != parent[v]) {
```

```cpp
            dfs(v,u);
            size[u] += size[v];
            if (size[v] > mxSize) mxSize = size[v],
                heavy[u] = v;
        }
}
int decompose(int u, int curHead, int&curPos) {
    head[u] = curHead, pos[u] = ++curPos;
    if (sz(g[u]) != 0) decompose(heavy[u],
        curHead, curPos);
    for (int v : g[u]) if (v != parent[u] && v !=
         heavy[u]) decompose(v, v, curPos);
}
void init() {
    int n = sz(g);
    parent.resize(n+1,0); depth.resize(n+1,0);
        size.resize(n+1,0);
    heavy.resize(n+1,0); head.resize(n+1,0); pos.
        resize(n+1,0);
    dfs(1,1);
    int curPos=0;
    decompose(1,1,curPos);
}
```

1.14 Lowest Common Ancestor

```cpp
int lca(int x, int y) {
    if (dep[y] > dep[x]) swap(x, y);
    int diff = dep[x] - dep[y];
    for(int jump=parent.lg; jump>=0; --jump) if(
        diff&(1<<jump)) x = parent.arr[jump][x];

 if(x == y) return x;

 for(int jump=parent.lg; jump>=0; --jump) if(
     parent.arr[jump][x] != parent.arr[jump][y])
     {
        x = parent.arr[jump][x];
     y = parent.arr[jump][y];
    }
    return parent.arr[0][x];
}
```

1.15 Sieve of Eratosthenes

```cpp
int primeCnt = 0;
for(int i=2; i<=X; ++i) spf[i] = 0;
for(int i=2; i<=X; ++i) {
    if(!spf[i]) primes[++primeCnt] = i, spf[i] =
        i;
    for(int j=1; j<=primeCnt && 1LL*i*primes[j]<X
         && primes[j]<=spf[i]; j++) spf[i*primes[
        j]] = primes[j];
}

phi[1] = 1;
for(int i=2; i<=X; i++) phi[i] = (spf[i] == spf[i
    /spf[i]]) ? phi[i/spf[i]]*spf[i] : phi[i/spf[
    i]]*(spf[i]-1);
```

1.16 Big Integer

```cpp
struct BigInt {
    int sign;
    string s;

    BigInt(): s(""){}

    BigInt(string x){ *this = x; }

    BigInt(int x){ *this = to_string(x);}

    BigInt negative(){ BigInt x = *this; x.sign
        *= -1; return x; }

    BigInt normalize(int newSign){
        for(int a = size(s) - 1; a > 0 && s[a] ==
            '0'; a--) s.erase(s.begin() + a);
        sign = (size(s) == 1 && s[0] == '0' ? 1 :
            newSign);
        return *this;
    }

    bool isZero(){ return s == "" || s == "0"; }

    void operator=(string x){
        int newSign = (x[0] == '-' ? -1 : 1);
        s = (newSign == -1 ? x.substr(1) : x);
        reverse(s.begin(), s.end());
        this->normalize(newSign);
    }

    bool operator==(const BigInt& x) const{
        return (s == x.s && sign == x.sign);
    }

    bool operator<(const BigInt& x) const{
        if (sign != x.sign) return sign < x.sign;
        if (size(s) != size(x.s)) return (sign ==
            1 ? size(s) < size(x.s) : size(s) >
           size(x.s));

        for (int a = size(s) - 1; a >= 0; a--) if
            (s[a] != x.s[a]) return (sign == 1 ?
           s[a] < x.s[a] : s[a] > x.s[a]);
        return false;
    }

    bool operator<=(const BigInt& x) const{
        return (*this < x || *this == x); }

    bool operator>(const BigInt& x) const{ return
        (!(*this < x) && !(*this == x)); }

    bool operator>=(const BigInt& x) const{
        return (*this > x || *this == x); }
```

```
BigInt operator+(BigInt x){
    BigInt curr = *this;
    if (curr.sign != x.sign) return curr - x.
        negative();
    BigInt res;
    for (int a = 0, carry = 0; a < size(s) ||
         a < size(x.s) || carry; a++) {
        carry += (a < size(curr.s) ? curr.s[a]
            - '0' : 0) + (a < size(x.s) ? x.s
            [a] - '0' : 0);
        res.s += (carry % 10 + '0');
        carry /= 10;
    }
    return res.normalize(sign);
}

BigInt operator-(BigInt x){
    BigInt curr = *this;
    if (curr.sign != x.sign) return curr + x.
        negative();
    int realSign = curr.sign;
    curr.sign = x.sign = 1;
    if (curr < x) return ((x - curr).negative
        ()).normalize(-realSign);
    BigInt res;
    for (int a = 0, borrow = 0; a < size(s);
        a++) {
        borrow = (curr.s[a] - borrow - (a <
            size(x.s) ? x.s[a] : '0'));
        res.s += (borrow >= 0 ? borrow + '0' :
             borrow + '0' + 10);
        borrow = (borrow >= 0 ? 0 : 1);
    }
    return res.normalize(realSign);
}

BigInt operator*(BigInt x){
    BigInt res("0");
    for (int a = 0, b = s[a] - '0'; a < size(
        s); a++, b = s[a] - '0') {
        while (b--) res = (res + x);
        x.s.insert(x.s.begin(), '0');
    }
    return res.normalize(sign * x.sign);
}

BigInt operator/(BigInt x){
    if (size(x.s) == 1 && x.s[0] == '0') x.s
        [0] /= (x.s[0] - '0');
    BigInt temp("0"), res;
    for (int a = 0; a < size(s); a++) res.s
        += "0";
    int newSign = sign * x.sign;
    x.sign = 1;
    for (int a = size(s) - 1; a >= 0; a--) {
        temp.s.insert(temp.s.begin(), '0');
        temp = temp + s.substr(a, 1);
        while (!(temp < x)) {
            temp = temp - x;
            res.s[a]++;
        }
    }
    return res.normalize(newSign);
}

BigInt operator%(BigInt x){
    if (size(x.s) == 1 && x.s[0] == '0') x.s
        [0] /= (x.s[0] - '0');
    BigInt res("0");
    x.sign = 1;
    for (int a = size(s) - 1; a >= 0; a--) {
        res.s.insert(res.s.begin(), '0');
        res = res + s.substr(a, 1);
        while (!(res < x)) res = res - x;
    }
    return res.normalize(sign);
}

string toString() const{
    string ret = s;
    reverse(ret.begin(), ret.end());
    return (sign == -1 ? "-" : "") + ret;
}

BigInt toBase10(int base){
    BigInt exp(1), res("0"), BASE(base);
    for (int a = 0; a < size(s); a++) {
        int curr = (s[a] < '0' || s[a] > '9' ?
             (toupper(s[a]) - 'A' + 10) : (s[a
            ] - '0'));
        res = res + (exp * BigInt(curr));
        exp = exp * BASE;
    }
    return res.normalize(sign);
}

BigInt toBase10(int base, BigInt mod){
    BigInt exp(1), res("0"), BASE(base);
    for (int a = 0; a < size(s); a++) {
        int curr = (s[a] < '0' || s[a] > '9' ?
             (toupper(s[a]) - 'A' + 10) : (s[a
            ] - '0'));
        res = (res + ((exp * BigInt(curr) %
            mod)) % mod);
        exp = ((exp * BASE) % mod);
    }
    return res.normalize(sign);
}

string convertToBase(int base){
    BigInt ZERO(0), BASE(base), x = *this;
    string modes = "0123456789
        ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    if (x == ZERO) return "0";
    string res = "";
    while (x > ZERO) {
        BigInt mod = x % BASE;
        x = x - mod;
```

```
        if (x > ZERO) x = x / BASE;
        res = modes[stoi(mod.toString())] +
            res;
    }
    return res;
}

BigInt toBase(int base){ return BigInt(this->
    convertToBase(base)); }

friend ostream& operator<<(ostream& os, const
     BigInt& x){ os << x.toString(); return
    os; }
};
BigInt _gcd(BigInt a, BigInt b) {return b.isZero
    () ? a : _gcd(b, a % b);}
BigInt _lcm(BigInt a, BigInt b) {return a / _gcd(
    a, b) * b;}
```

1.17 Chinese Remainder Theorem

```
LL _dioph(LL a, LL b, LL& x, LL& y) { if(b > 0){
    LL g=_dioph(b,a%b,y,x); y-=(a/b)*x; return g;
    } x=1, y=0; return a; }
bool dioph(LL a, LL b, LL c, LL& x, LL& y){ LL g=
    gcd(a,b); if(c%g!=0) return false; _dioph(a,
    b, x, y); x*=c/g; y*=c/g; return true; }

LL CRT(LL a, LL n, LL b, LL m){
 LL x, y;
 assert(dioph(n, m, b-a, x, y));
 LL mod = lcm(n, m);
 LL res = ((LL)x*n + a) % mod;
 if(res < 0) res = (res + (LL)mod*(abs(res)/mod +
     1)) % mod;
 return res;
}
```

```
LL CRT(const vector<LL>& rems, const vector<LL>&
    mods){
 LL rem = rems.front(), mod = mods.front();
 for(int i=1; i<=size(rems)-1; ++i){
  rem = CRT(rem, mod, rems[i], mods[i]);
  mod = lcm(mod, mods[i]);
 }
 return rem;
}
```

1.18 FFT

```
namespace FFT{
 void fft(vector<complex<float>>& a, bool invert)
     {
  int n = size(a);
  if(n == 1) return;
  vector<complex<float>> y0(n/2), y1(n/2);
  for (int i = 0, j = 0; i < n; i += 2, ++j) {
   y0[j] = a[i];
   y1[j] = a[i + 1];
  }

  fft(y0, invert);
  fft(y1, invert);

  float ang = ((2.0 * PI) / n) * (invert ? -1 :
      1);
  complex<float> w(1) , wn(cos(ang), sin(ang));

  for (int k = 0; k < n / 2; ++k) {
   a[k] = y0[k] + w * y1[k];
   a[k + n / 2] = y0[k] - w * y1[k];

   if (invert) a[k] /= 2 , a[k + n / 2] /= 2;
   w *= wn;
  }
 }
 void multiply(vector<int>& a , vector<int>& b,
    vector<int>& res) {
  int n = 1;
  while (n < max(size(a) , size(b))) n <<= 1;
  n <<= 1;

  vector<complex<float>> fx(all(a)) , fy(all(b));
  fx.resize (n) , fy.resize (n);
  fft(fx, false) , fft(fy, false);

  vector<complex<float>> hx(n);
  for (int i = 0; i < n; ++i) hx[i] = fx[i] * fy[
      i];
  fft(hx, true);

  res.resize(n);
  for (int i = 0; i < n; ++i)
   res[i] = int (hx[i].real() + 0.5);
 }
};
```

1.19 Binary exponentiation

```
LL power(LL x, LL n){
    if(n == 0) return 1;
    LL y = power(x, n>>1);
    y = mul(y, y);
    return (n & 1) ? mul(y, x) : y;
}
```

1.20 Longest Increasing Subsequence

```
set<int> lis; // multiset if there are duplicates
for(int i=1; i<=n; ++i) {
 auto it = lis.upper_bound(X); // lower_bound for
     strictly increasing
```

```
 if (it != end(lis)) lis.erase(it);
 lis.insert(X);
}
```

1.21 Unordered Map

```
struct custom_hash {
      static uint64_t splitmix64(uint64_t x) {
            x += 0x9e3779b97f4a7c15;
            x = (x ^ (x >> 30)) * 0
                xbf58476d1ce4e5b9;
            x = (x ^ (x >> 27)) * 0
                x94d049bb133111eb;
            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);
      }
};

unordered_map<int, int, custom_hash> safe_map;
```

1.22 Matrix Exponentiation

```
namespace MatrixExp{
 #define matrix vector<vector<long>>
 matrix mul(const matrix& a, const matrix& b){
  int n1 = size(a); int m1 = size(a[0]);
  int n2 = size(b); int m2 = size(b[0]);
  assert(m1 == n2);
  int shared = m1;
  matrix res(n1, vector<long>(m2, 0));
  for(int i=1; i<=n1; ++i) for(int j=1; j<=m2; ++
      j){
   for(int mid=1; mid<=shared; ++mid) res[i-1][j
       -1] = add(res[i-1][j-1], mul(a[i-1][mid
       -1],b[mid-1][j-1]));
  }
  return res;
 }
 matrix power(const matrix& x, const long& n){
  matrix y(size(x), vector<long>(size(x[0]), 0));
  for(int i=1; i<=size(y); ++i) y[i-1][i-1]=1;
  if(n == 0) return y;
  y=power(x, n>>1); y=mul(y,y);
  return (n&1) ? mul(y,x) : y;
 }
} using namespace MatrixExp;
```

1.23 Max Flow

```
struct Dinic {
 struct Edge { int from, to; int cap, flow; };
 vector<vector<int>> adj;
 vector<Edge> e;
 vector<int> level, edgeCount;
 int n, src, sink;
 Dinic(int _n) {
  n = _n+2; src = _n+1; sink = _n+2;
  adj.assign(n+1, vector<int>());
  level.assign(n+1, 0);
  edgeCount.assign(n+1, 0);
  e.clear();
 }
 void addEdge(int a, int b, int cap) {
  adj[a].emplace_back(size(e));
  e.push_back(Edge{ a, b, cap, 0 });

  adj[b].emplace_back(size(e));
  e.push_back(Edge{ b, a, 0, 0 });
 }
bool bfs() {
 fill(all(level), -1);
 queue<int> q;
 q.push(src);
 level[src] = 0;
 while (!q.empty()) {
  int cur = q.front(); q.pop();
  for (int i = 0; i < size(adj[cur]); i++) {
   int curID = adj[cur][i];
   int nxt = e[curID].to;
   if (level[nxt] == -1 && e[curID].flow < e[
       curID].cap) {
    q.push(nxt);
    level[nxt] = level[cur] + 1;
   }
  }
 }
 return (level[sink] != -1);
}
int sendFlow(int cur, int curFlow) {
 if (cur == sink || curFlow == 0) return curFlow
     ;
 for (; edgeCount[cur] < size(adj[cur]); ++
     edgeCount[cur]) {
   int curID = adj[cur][edgeCount[cur]];
   int nxt = e[curID].to;
   if (level[nxt] == level[cur] + 1) {
    int tmpFlow = sendFlow(nxt, min(curFlow, e[
        curID].cap - e[curID].flow));
    if (tmpFlow > 0) {
     e[curID].flow += tmpFlow;
     e[curID ^ 1].flow -= tmpFlow;
     return tmpFlow;
    }
   }
 }
 return 0;
}
int maxFlow() {
 int f = 0;
```

```
  while(bfs()) {
   fill(all(edgeCount), 0);
   while (int tmpFlow=sendFlow(src, INFLL)) f +=
       tmpFlow;
  }
  return f;
 }
};
```

1.24 Z Algorithm

```
vector<int> computeZ(const string& _s) {
    vector<int> _z(size(_s), 0);
    int l = 0, r = 0;
 for(int i = 1; i < size(_s); i++) {
  _z[i] = clamp(r - i + 1, 0, _z[i - l]);
  while (i + _z[i] < size(_s) && _s[_z[i]] == _s[
      i + _z[i]]) { l = i, r = i + _z[i], ++_z[i
      ]; }
 }
 return _z;
}
```

1.25 Hash

```
struct Hash{
    vector<int> h;
    const int p = some_prime_number;
    Hash(int val[], int n){
        h.clear();
        int curH = 0;
        int curP = 1;
        for(int i = 0; i < n; ++i){
            curH = add(curH, mul(val[i],curP));
            curP = mul(curP, p);
            h.emplace_back(curH);
        }
    }
    int rangeHash(int l, int r){
        int b = h[r];
        int a = (l == 0) ? 0 : h[l-1];
        return mul(add(b,-a),inv(power(p,l)));
    }
};
```

1.26 KMP

```
void KMPprefixfunction(string s, int pf[]) {
    for(int i=1; i<size(s); i++) {
        int j = pf[i-1];
        while(j > 0 && s[i] != s[j]) j = pf[j-1];
        if(s[i] == s[j]) j++;
        pf[i] = j;
    }
}

void computeNext(string s, int pf[], int nxt
    [][26]) {
    s += '#';
    KMPprefixfunction(s, pf);
    for(int i=0; i<size(s); i++) {
        for(int c=0; c<26; c++) {
            if(i > 0 && 'a' + c != s[i]) nxt[i][c]
                = nxt[pf[i-1]][c];
            else nxt[i][c] = i + ('a' + c == s[i])
                ;
        }
    }
}
```

1.27 2D Fenwick Tree

```
int bit[N][N];
void add(int i, int j, int v) {
  for (; i < N; i+=i&-i)
    for (int jj = j; jj < N; jj+=jj&-jj)
      bit[i][jj] += v;
}

int query(int i, int j) {
  int res = 0;
  for (; i; i-=i&-i)
    for (int jj = j; jj; jj-=jj&-jj)
      res += bit[i][jj];
  return res;
}

// Whole BIT 2D set to 1
void init() {
  cl(bit,0);
  for (int i = 1; i <= r; ++i)
    for (int j = 1; j <= c; ++j)
      add(i, j, 1);
}

// Return number of positions set
int query(int imin, int jmin, int imax, int jmax)
     {
  return query(imax, jmax) - query(imax, jmin-1)
      - query(imin-1, jmax) + query(imin-1, jmin
      -1);
}

// Find all positions inside rect (imin, jmin), (
    imax, jmax) where position is set
void proc(int imin, int jmin, int imax, int jmax,
     int v, int tot) {
  if (tot < 0) tot = query(imin, jmin, imax, jmax
      );
  if (!tot) return;

  int imid = (imin+imax)/2, jmid = (jmin+jmax)/2;
  if (imin != imax) {
```

```
    int qnt = query(imin, jmin, imid, jmax);
    if (qnt) proc(imin, jmin, imid, jmax, v, qnt)
        ;
    if (tot-qnt) proc(imid+1, jmin, imax, jmax, v
        , tot-qnt);
  } else if (jmin != jmax) {
    int qnt = query(imin, jmin, imax, jmid);
    if (qnt) proc(imin, jmin, imax, jmid, v, qnt)
        ;
    if (tot-qnt) proc(imin, jmid+1, imax, jmax, v
        , tot-qnt);
  } else {
    // single position set!
    // now process position!!!
    add(imin, jmin, -1);
  }
}
```

1.28 DP Convex Hull trick

```
// ATTENTION: This is the maximum convex hull. If
    you need the minimum
// CHT use {-b, -m} and modify the query function
    .

// In case of floating point parameters swap LL
    with long double
typedef LL type;
struct line { type b, m; };

line v[N]; // lines from input
int n; // number of lines
// Sort slopes in ascending order (in main):
sort(v, v+n, [](line s, line t){
     return (s.m == t.m) ? (s.b < t.b) : (s.m < t
        .m); });

// nh: number of lines on convex hull
// pos: position for linear time search
// hull: lines in the convex hull
int nh, pos;
line hull[N];

bool check(line s, line t, line u) {
  // verify if it can overflow. If it can just
      divide using long double
  return (s.b - t.b)*(u.m - s.m) < (s.b - u.b)*(t
      .m - s.m);
}

// Add new line to convex hull, if possible
// Must receive lines in the correct order,
    otherwise it won't work
void update(line s) {
  // 1. if first lines have the same b, get the
      one with bigger m
  // 2. if line is parallel to the one at the top
      , ignore
  // 3. pop lines that are worse
  // 3.1 if you can do a linear time search, use
  // 4. add new line

  if (nh == 1 and hull[nh-1].b == s.b) nh--;
  if (nh > 0 and hull[nh-1].m >= s.m) return;
  while (nh >= 2 and !check(hull[nh-2], hull[nh
      -1], s)) nh--;
  pos = min(pos, nh);
  hull[nh++] = s;
}

type eval(int id, type x) { return hull[id].b +
    hull[id].m * x; }

// Linear search query - O(n) for all queries
// Only possible if the queries always move to
    the right
type query(type x) {
  while (pos+1 < nh and eval(pos, x) < eval(pos
      +1, x)) pos++;
  return eval(pos, x);
  // return -eval(pos, x); ATTENTION: Uncomment
      for minimum CHT
}

// Ternary search query - O(logn) for each query
/*
type query(type x) {
  int lo = 0, hi = nh-1;
  while (lo < hi) {
    int mid = (lo+hi)/2;
    if (eval(mid, x) > eval(mid+1, x)) hi = mid;
    else lo = mid+1;
  }
  return eval(lo, x);
  // return -eval(lo, x);  ATTENTION: Uncomment
      for minimum CHT
}
// better use geometry line_intersect (this
    assumes s and t are not parallel)
ld intersect_x(line s, line t) { return (t.b - s.
    b)/(ld)(s.m - t.m); }
ld intersect_y(line s, line t) { return s.b + s.m
     * intersect_x(s, t); }
*/
```

1.29 DP Divide and Conquer optimization

```
// dp[i][j] = min k<i { dp[k][j-1] + C[k][i] }
//
// Condition: A[i][j] <= A[i+1][j]
// A[i][j] is the smallest k that gives an
    optimal answer to dp[i][j]
//
// reference (pt-br): https://algorithmmarch.
    wordpress.com/2016/08/12/a-otimizacao-de-pds-
    e-o-garcom-da-maratona/
```

```
int n, maxj;
int dp[N][J], a[N][J];

// declare the cost function
int cost(int i, int j) {
  // ...
}

void calc(int l, int r, int j, int kmin, int kmax
    ) {
  int m = (l+r)/2;
  dp[m][j] = LINF;

  for (int k = kmin; k <= kmax; ++k) {
    ll v = dp[k][j-1] + cost(k, m);

    // store the minimum answer for d[m][j]
    // in case of maximum, use v > dp[m][j]
    if (v < dp[m][j]) a[m][j] = k, dp[m][j] = v;
  }

  if (l < r) {
    calc(l,  m, j, kmin,  a[m][k]);
    calc(m+1, r, j, a[m][k], kmax );
  }
}

// run for every j
for (int j = 2; j <= maxj; ++j)
  calc(1, n, j, 1, n);
```

1.30 DP Knuth optimization

```
// 1) dp[i][j] = min i<k<j { dp[i][k] + dp[k][j]
    } + C[i][j]
// 2) dp[i][j] = min k<i { dp[k][j-1] + C[k][i] }
//
// Condition: A[i][j-1] <= A[i][j] <= A[i+1][j]
// A[i][j] is the smallest k that gives an
    optimal answer to dp[i][j]
//
// reference (pt-br): https://algorithmmarch.
    wordpress.com/2016/08/12/a-otimizacao-de-pds-
    e-o-garcom-da-maratona/
//

// 1) dp[i][j] = min i<k<j { dp[i][k] + dp[k][j]
    } + C[i][j]
int n;
int dp[N][N], a[N][N];

// declare the cost function
int cost(int i, int j) {
  // ...
}

void knuth() {
  // calculate base cases
  memset(dp, 63, sizeof(dp));
  for (int i = 1; i <= n; i++) dp[i][i] = 0;

  // set initial a[i][j]
  for (int i = 1; i <= n; i++) a[i][i] = i;

  for (int j = 2; j <= n; ++j)
    for (int i = j; i >= 1; --i)
      for (int k = a[i][j-1]; k <= a[i+1][j]; ++k
          ) {
        ll v = dp[i][k] + dp[k][j] + cost(i, j);

        // store the minimum answer for d[i][k]
        // in case of maximum, use v > dp[i][k]
        if (v < dp[i][j])
          a[i][j] = k, dp[i][j] = v;
      }
}
// 2) dp[i][j] = min k<i { dp[k][j-1] + C[k][i] }
int n, maxj;
int dp[N][J], a[N][J];

// declare the cost function
int cost(int i, int j) {
  // ...
}

void knuth() {
  // calculate base cases
  memset(dp, 63, sizeof(dp));
  for (int i = 1; i <= n; i++) dp[i][1] = // ...

  // set initial a[i][j]
  for (int i = 1; i <= n; i++) a[i][0] = 0, a[n
      +1][i] = n;

  for (int j = 2; j <= maxj; j++)
    for (int i = n; i >= 1; i--)
      for (int k = a[i][j-1]; k <= a[i+1][j]; k
          ++) {
        ll v = dp[k][j-1] + cost(k, i);

        // store the minimum answer for d[i][k]
        // in case of maximum, use v > dp[i][k]
        if (v < dp[i][j])
          a[i][j] = k, dp[i][j] = v;
      }
}
```

1.31 Closest Pair problem

```
struct pnt{
    LL x, y;
    pnt operator-(pnt p){ return {x - p.x, y - p.
        y}; }
    LL operator!(){ return x*x+y*y; }
};
```

```
const int N = 1e5 + 5;
pnt pnts[N];
pnt tmp[N];
pnt p1, p2;
unsigned LL d = 9e18;

void closest(int l, int r){
    if(l == r) return;
    int mid = (l + r)/2;

    int midx = pnts[mid].x;
    closest(l, mid), closest(mid + 1, r);

    merge(pnts + l, pnts + mid + 1, pnts + mid +
        1, pnts + r + 1, tmp + l,
            [](pnt a, pnt b){ return a.y < b.y; })
                ;

    for (int i = l; i <= r; i++) pnts[i] = tmp[i
        ];

    vector<pnt> margin;
    for(int i = l; i <= r; i++)
        if((pnts[i].x - midx)*(pnts[i].x - midx)
            < d)
            margin.push_back(pnts[i]);

    for(int i = 0; i < margin.size(); i++)
        for(int j = i + 1;
            j < margin.size() and
            (margin[j].y - margin[i].y)*(margin[j
                ].y - margin[i].y) < d;
            j++) {
            if(!(margin[i] - margin[j]) < d)
                p1 = margin[i], p2 = margin[j], d
                    = !(p1 - p2);
        }
}
```

1.32 Geometry basics 2

```
const int INF = 0x3f3f3f3f;

typedef long double ld;
const double EPS = 1e-9, PI = acos(-1.);

// Change long double to LL if using integers
typedef long double type;

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(
    x, y); }

struct point {
  type x, y;

  point() : x(0), y(0) {}
  point(type x, type y) : x(x), y(y) {}

  point operator -() { return point(-x, -y); }
  point operator +(point p) { return point(x+p.x,
       y+p.y); }
  point operator -(point p) { return point(x-p.x,
       y-p.y); }

  point operator *(type k) { return point(k*x, k*
      y); }
  point operator /(type k) { return point(x/k, y/
      k); }

  type operator *(point p) { return x*p.x + y*p.y
      ; }
  type operator %(point p) { return x*p.y - y*p.x
      ; }

  // o is the origin, p is another point
  // dir == +1 => p is clockwise from this
  // dir == 0 => p is colinear with this
  // dir == -1 => p is counterclockwise from this
  int dir(point o, point p) {
    type x = (*this - o) % (p - o);
    return ge(x,0) - le(x,0);
  }

  bool on_seg(point p, point q) {
    if (this->dir(p, q)) return 0;
    return ge(x, min(p.x, q.x)) and le(x, max(p.x
        , q.x)) and
           ge(y, min(p.y, q.y)) and le(y, max(p.y,
               q.y));
  }

  ld abs() { return sqrt(x*x + y*y); }
  type abs2() { return x*x + y*y; }
  ld dist(point x) { return (*this - x).abs(); }
  type dist2(point x) { return (*this - x).abs2()
      ; }

  ld arg() { return atan2l(y, x); }

  // Project point on vector y
  point project(point y) { return y * ((*this * y
      ) / (y * y)); }

  // Project point on line generated by points x
      and y
  point project(point x, point y) { return x + (*
      this - x).project(y-x); }

  ld dist_line(point x, point y) { return dist(
      project(x, y)); }

  ld dist_seg(point x, point y) {
    return project(x, y).on_seg(x, y) ? dist_line
        (x, y) : min(dist(x), dist(y));
  }
```

```
  point rotate(ld sin, ld cos) { return point(cos
      *x-sin*y, sin*x+cos*y); }
  point rotate(ld a) { return rotate(sin(a), cos(
      a)); }
  // rotate around the argument of vector p
  point rotate(point p) { return rotate(p.x / p.
      abs(), p.y / p.abs()); }
};


int direction(point o, point p, point q) { return
     p.dir(o, q); }

bool segments_intersect(point p, point q, point a
    , point b) {
  int d1, d2, d3, d4;
  d1 = direction(p, q, a);
  d2 = direction(p, q, b);
  d3 = direction(a, b, p);
  d4 = direction(a, b, q);
  if (d1*d2 < 0 and d3*d4 < 0) return 1;
  return p.on_seg(a, b) or q.on_seg(a, b) or
        a.on_seg(p, q) or b.on_seg(p, q);
}

point lines_intersect(point p, point q, point a,
    point b) {
  point r = q-p, s = b-a, c(p%q, a%b);
  if (eq(r%s,0)) return point(INF, INF);
  return point(point(r.x, s.x) % c, point(r.y, s.
      y) % c) / (r%s);
}

// Sorting points in counterclockwise order.
// If the angle is the same, closer points to the
     origin come first.
point origin;
bool radial(point p, point q) {
  int dir = p.dir(origin, q);
  return dir > 0 or (!dir and p.on_seg(origin, q)
      );
}

// Graham Scan
vector<point> convex_hull(vector<point> pts) {
  vector<point> ch(pts.size());
  point mn = pts[0];

  for(point p : pts) if (p.y < mn.y or (p.y == mn
      .y and p.x < p.y)) mn = p;

  origin = mn;
  sort(pts.begin(), pts.end(), radial);

  int n = 0;

  // IF: Convex hull without collinear points
  for(point p : pts) {
    while (n > 1 and ch[n-1].dir(ch[n-2], p) < 1)
        n--;
    ch[n++] = p;
  }

  /* ELSE IF: Convex hull with collinear points
  for(point p : pts) {
    while (n > 1 and ch[n-1].dir(ch[n-2], p) < 0)
        n--;
    ch[n++] = p;
  }
  for(int i=pts.size()-1; i >=1; --i)
    if (pts[i] != ch[n-1] and !pts[i].dir(pts[0],
        ch[n-1]))
      ch[n++] = pts[i];
  // END IF */

  ch.resize(n);
  return ch;
}

// Double of the triangle area
ld double_of_triangle_area(point p1, point p2,
    point p3) {
  return abs((p2-p1) % (p3-p1));
}

// TODO: test this code. This code has not been
    tested, please do it before proper use.
// http://codeforces.com/problemset/problem/975/E
     is a good problem for testing.
point centroid(vector<point> &v) {
  int n = v.size();
  type da = 0;
  point m, c;

  for(point p : v) m = m + p;
  m = m / n;

  for(int i=0; i<n; ++i) {
    point p = v[i] - m, q = v[(i+1)%n] - m;
    type x = p % q;
    c = c + (p + q) * x;
    da += x;
  }

  return c / (3 * da);
}

bool point_inside_triangle(point p, point p1,
    point p2, point p3) {
  ld a1, a2, a3, a;
  a = double_of_triangle_area(p1, p2, p3);
  a1 = double_of_triangle_area(p, p2, p3);
  a2 = double_of_triangle_area(p, p1, p3);
  a3 = double_of_triangle_area(p, p1, p2);
  return eq(a, a1 + a2 + a3);
}

bool point_inside_convex_poly(int l, int r,
    vector<point> v, point p) {
```

```
  while(l+1 != r) {
    int m = (l+r)/2;
    if (p.dir(v[0], v[m])) r = m;
    else l = m;
  }
  return point_inside_triangle(p, v[0], v[l], v[r
      ]);
}

vector<point> circle_circle_intersection(point p1
    , ld r1, point p2, ld r2) {
  vector<point> ret;

  ld d = p1.dist(p2);
  if (d > r1 + r2 or d + min(r1, r2) < max(r1, r2
      )) return ret;

  ld x = (r1*r1 - r2*r2 + d*d) / (2*d);
  ld y = sqrt(r1*r1 - x*x);

  point v = (p2 - p1)/d;

  ret.push_back(p1 + v * x + v.rotate(PI/2) * y);
  if (y > 0)
    ret.push_back(p1 + v * x - v.rotate(PI/2) * y
        );

  return ret;
}
```

1.33 SQRT Decomposition (MO's algorithm)

```
const int N = 1e5+1, SQ = 500;
int n, m, v[N];

void add(int p) { /* add value to aggregated data
     structure */ }
void rem(int p) { /* remove value from aggregated
     data structure */ }

struct query { int i, l, r, ans; } qs[N];

bool c1(query a, query b) {
  if(a.l/SQ != b.l/SQ) return a.l < b.l;
  return a.l/SQ&1 ? a.r > b.r : a.r < b.r;
}

bool c2(query a, query b) { return a.i < b.i; }

/* inside main */
int l = 0, r = -1;
sort(qs, qs+m, c1);
for (int i = 0; i < m; ++i) {
  query &q = qs[i];
  while (r < q.r) add(v[++r]);
  while (r > q.r) rem(v[r--]);
  while (l < q.l) rem(v[l++]);
  while (l > q.l) add(v[--l]);

  q.ans = /* calculate answer */;
}

sort(qs, qs+m, c2); // sort to original order
```

1.34 Manacher's algorithm

```
// Longest Palindromic String - O(n)
int lps[2*N+5];
char s[N];

int manacher() {
  int n = strlen(s);

  string p (2*n+3, '#');
  p[0] = '^';
  for (int i = 0; i < n; i++) p[2*(i+1)] = s[i];
  p[2*n+2] = '$';

  int k = 0, r = 0, m = 0;
  int l = p.length();
  for (int i = 1; i < l; i++) {
    int o = 2*k - i;
    lps[i] = (r > i) ? min(r-i, lps[o]) : 0;
    while (p[i + 1 + lps[i]] == p[i - 1 - lps[i
        ]]) lps[i]++;
    if (i + lps[i] > r) k = i, r = i + lps[i];
    m = max(m, lps[i]);
  }
  return m;
}
```

1.35 Miller Rabin

```
// Randomized Primality Test (Miller-Rabin):
//   Error rate: 2^(-TRIAL)
//   Almost constant time. srand is needed
#define EPS 1e-7
LL ModularMultiplication(LL a, LL b, LL m){
 LL ret=0, c=a;
 while(b){
  if(b&1) ret=(ret+c)%m;
  b>>=1; c=(c+c)%m;
 }
 return ret;
}
LL ModularExponentiation(LL a, LL n, LL m){
 LL ret=1, c=a;
 while(n){
  if(n&1) ret=ModularMultiplication(ret, c, m);
  n>>=1; c=ModularMultiplication(c, c, m);
 }
 return ret;
}
bool Witness(LL a, LL n){
 LL u=n-1;
```

```
    int t=0;
 while(!(u&1)){u>>=1; t++;}
 LL x0=ModularExponentiation(a, u, n), x1;
 for(int i=1;i<=t;i++){
  x1=ModularMultiplication(x0, x0, n);
  if(x1==1 && x0!=1 && x0!=n-1) return true;
  x0=x1;
 }
 if(x0!=1) return true;
 return false;
}
LL Random(LL n){
    LL ret=rand(); ret*=32768;
 ret+=rand(); ret*=32768;
 ret+=rand(); ret*=32768;
 ret+=rand();
  return ret%n;
}
bool IsPrimeFast(LL n, int TRIAL){
  while(TRIAL--){
    LL a=Random(n-2)+1;
    if(Witness(a, n)) return false;
  }
  return true;
}
```

1.36 Linear Diophantine Equations

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

bool find_any_solution(int a, int b, int c, int &
    x0, int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) return false;
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
void shift_solution(int & x, int & y, int a, int
    b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int
    minx, int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g; b /= g;
    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;
    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx) shift_solution(x, y, a, b,
        sign_b);
    if (x > maxx) return 0;
    int lx1 = x;
    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx) shift_solution(x, y, a, b, -
        sign_b);
    int rx1 = x;
    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny) shift_solution(x, y, a, b, -
        sign_a);
    if (y > maxy) return 0;
    int lx2 = x;
    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy) shift_solution(x, y, a, b,
        sign_a);
    int rx2 = x;
    if (lx2 > rx2) swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);
    if (lx > rx) return 0;
    return (rx - lx) / abs(b) + 1;
}
```

2 Error inspection

```
Made by https://github.com/mostafa-saad/
    ArabicCompetitiveProgramming

---- General Inspection

*) Do you Terminate before reading whole input?
    Do you break while reading, and test case is
    not fully read?
*) Correct input file / Correct input copy
*) Correctly initialize between test cases
*) Correct reading: E.g. scanf("%d ", &cases);
    Space is correct in this Input file?
*) Tested corner cases?
*) If you take something from libaray that does
    not mean it is 100% right
*) If you take something from libaray, Revise its
     comments & prerequisites
*) Validate:
    *) Input stoping conditions
    *) Functions base case / Problem Logic Lines.
    *) No TYPO
    *) Used data types enough to avoid overlfow
    *) operators *, ^, /, %
        *) /0, %0, /EPS, (n)%x, (-n)%x, +/- EPS
        *) *, ^, counting problems = Overflow
```

```
    *) Results fit in 32bit operations, what
       about intermediate values?
    *) Correct OO Value. Intial value when u
       maximize & minimize
    *) x%m is X negative? Given Matrix A, find (A
       ^b)%M, Make sure A is intialized
       correctly in case negatives
    *) if(x%2 == 1) .. What if x negative value?
    *) Set or Multiset?
    *) Wrong pair comparisons: pair1 > pair2,
       does not check both elements for larger
    *) truncate or approximate, double issues,
       watch out from -0.0, Floor(-2.3) = -3 but
        Floor(2.3)= 2
    *) not a number(NAN) which comes from sqrt(-
       ve), (0/0) ,or cos(1.00000000001) or cos
       (-1.000000000001)
    *) lp(i, n) lp(j, i+1, n): is there at least
       2 elements? Do you need special handling?
*) Percision problems
    *) You should calculate the worst percicion.
       E.g. 1 / 10^9 / 10^5.
    *) Avoid doubke operations if possible: e.g.
       integer floor & ceil - even with indepth
       replacement.
    *) try to do double operations as local as
       possilbe
       *) e.g. sum all vs sum part + call(nxt)
    *) couting the doubles changes little thier
       value
    *) Do binary search to 9 precision, and
       display x*100. output is only 7 precision
*) BFS with more than one start state
    *) Make sure they are all of same depth
    *) Make sure, In case lexi answer, that you
       use priority queue
    *) Do we need to validate the intial states?


---- WA

*) Pick a moderated examples, and do problem
   semantic tracing.

*) Sometimes your added tricky code to make
   programming easier is just a KILLER BUG
    *) TRY to validate your fancy added code to
       avoid debugging for silly mistakes
    *) E.g appending a 2d strings array to make
       it a complete 2d.
        *) Take care, is the appended character
           part of input? Does it matter?

*) If u have direction array, Does order matters?

*) OVERFLOW
    *) Read numbers N < 2^M where M = 60
    *) Manpulating bit masks with N >= 32, E.g.
       1<<40
    *) Multiplications( cross product ) & powers
       & Base conversions.
    *) Is whole code handeled for OVERFLOW or it
       is a mix of int and LL ?!
       *) 1<<x or 1LL<<x
    *) Correct overflow handling
       *) E.g use if( a*b > OO) or if( a > OO/b)
       *) E.g use if( a+b > OO) or if( a > OO-b)
    *) Input is a 32 integer bit
       *) yes, using int x; will be sufficient,
          but take care from Integer range
       *) int x; cin>>x; x = -x;   code(x);
       *) What if x value = -2^31 --> -x = 2^31
          which is OVERFLOW
    *) Exhaustive adding
    *) Final answer fit in 32bit but intermediate
        results don't (e.g. polynomial
       evaluation)

*) Wrong stoping conditions.

    *) Test ends with TestEnd and input with
       InputFinish. What if such words inside
       the main input also.
    *) You may need to read 5 numbers if any is
       valid, u alert
        *) lp(i, 5) { cin>>x; if(!valid(x)) { ok
           = 0; break;} --> What about output
           REMINDER?
    *) Read until number is less than 0, if(n
       ==-1) break --> if(n<0) break;
    *) Read until x & y & z be zero
    *) Read until one of x or or z is zero
    *) Read untill Input L, U is L = U = -1
        *) Stop if(L == -1 && U == -1) break;
        *) Check if input like L = -2, U = 3 is
           valid or not
        *) E.g Number of primes in range [-2,3] =
            2
    *) Each block will be terminated by a line
       starting with e.
        *) e
        *) egg
    *) Each block will be terminated by a line
       containing #.
        *) #
        *) Is this tricky #?

*) Tricky text description
    *) word is "sequence of upper/lower case
       letters.
        *) This means ali is 1 word, X-Ray is 2
           words, ali's book is 3 words
    *) Given 2 integers i, j, find number of
       primes between them, or in RANGE
        *) Input can be 4 200 OR 200 4
    *) Given N*M grid, Read N lines each start
       with M chars. E.g. 3*2
        *)1st line -> ab
        *)2nd line -> cdEXTRA  // use to depend
           on read N, M, as RE may happen
```

```
        *)3rd line -> ef
    *) Do not accept leading zeros numbers?
        *) Do not accept 004, but accept 0 (
            special case)

*) Geometry
    *) Is there duplicate points? Does it matter?
        Co-linearity?

*) Graph
    *) Connected or disconnected?
    *) Directed or Undirected?
    *) SelfLoops?
    *) Multiple edges & their effect (MaxFlow sum
        , SP min)

*) Percision
    *) Watchout -0.0
    *) int x = (int)(a +/- EPS) depends on a > 0
        | a < 0.

---- TLE

*) May be bug and just infinite loop
*) Can results precomputed in table?!!!!
*) Function calls, may need refrence variables.
*) % is used extensivly? memset is used
    extensivly?
*) What is blocks of code that reprsent order? Do
    we just need to optimize it?
```

```
*) Big Input file
    *) Need scanf & printf
    *) Optimize code operations
    *) Switch to arrays and char[]

*) DP Problems
    *) Do you clear each time while it is not
        needed?
    *) Clear only part of memory u need, not all
        of memo or use boolean array
    *) The base case order is not O(1)
        *) make sure if(memo != -1) before base
            case
    *) Use effective base conditions
        *) E.g If you are sure dp(0, M) is X, do
            not wait untill Dp(0,0)
    *) DP state did not change, so infinite loop
        *) DP(i) call DP(i+s) where s [0-4]
    *) Return result % 10^7, So each time you do
        operation, you apply %
        *) if DP is huge, change to while(ans >=
            10^7) ans -= 10^7
        *) If mod is 2^p-1, use bitwise

*) BackTracking
    *) If you have diffrent ways to do it, try to
        do what minimize stack depth

*) Graph problems
    *) Generate dynamic sub-states (edges) only
        when necessary
```

```
---- RTE

*) Correct input file?
*) Array index out of bondry

*) Make sure to have correct array size. E.g. If
    indexing N 1 based, arr[N+1].
*) Make sure no wrong indexing < 0 || x >= n
    *) Find Primes in range[-2, 3]
    *) Find factorial -5!
*) In DP, memo[X][Y], check you access dimensions
     correctly
*) In DP, if u have invald states, make sure to
    filter them before checking the memo
*) Stack overflow from infinite recursion
    *) Visited array not marked correctly
    *) DP with cyclic \ wrong recurrence
*) You have data structures that requires huge
    data
*) /0, %0
*) Extensive memory allocating until RTE
*) Using incorrect compare function (e.g. return
    that return (A, B) same answer as (B, A) )
*) Use unintialized data: int x; v.resize(x); cin
    >>x;
*) Watchout, if multiset contains (3 3 3 3 6 9)
    and u delete 3 -->will be (6, 9)
    *) To delete one item, use iterator to find &
        delete it
*) struct T { int A[]; };
```

Theoretical Computer Science Cheat Sheet

Definitions

Definitions	
$f(n) = O(g(n))$	iff $\exists$ positive c, n_0 such that $0 \le f(n) \le cg(n)$ $\forall n \ge n_0$.
$f(n) = \Omega(g(n))$	iff $\exists$ positive c, n_0 such that $f(n) \ge cg(n) \ge 0$ $\forall n \ge n_0$.
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
$f(n) = o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.
$\liminf_{n\to\infty} a_n$	$\lim_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\limsup_{n\to\infty} a_n$	$\lim_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.
$\left[{n \atop k}\right]$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.
$\left\{{n \atop k}\right\}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.
$\left\langle{n \atop k}\right\rangle$	1st order Eulerian numbers: Permutations $\pi_1\pi_2\dots\pi_n$ on $\{1,2,\dots,n\}$ with k ascents.
$\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle$	2nd order Eulerian numbers.
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.

Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

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

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \qquad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n}\frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \qquad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

1. $\displaystyle\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\displaystyle\sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\displaystyle\binom{n}{k} = \binom{n}{n-k}$,

4. $\displaystyle\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\displaystyle\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\displaystyle\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\displaystyle\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\displaystyle\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\displaystyle\sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\displaystyle\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\displaystyle\left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1$,

12. $\displaystyle\left\{{n \atop 2}\right\} = 2^{n-1} - 1$, **13.** $\displaystyle\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$,

14. $\displaystyle\left[{n \atop 1}\right] = (n-1)!$, **15.** $\displaystyle\left[{n \atop 2}\right] = (n-1)!H_{n-1}$, **16.** $\displaystyle\left[{n \atop n}\right] = 1$, **17.** $\displaystyle\left[{n \atop k}\right] \ge \left\{{n \atop k}\right\}$,

18. $\displaystyle\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$, **19.** $\displaystyle\left\{{n \atop n-1}\right\} = \left[{n \atop n-1}\right] = \binom{n}{2}$, **20.** $\displaystyle\sum_{k=0}^{n}\left[{n \atop k}\right] = n!$, **21.** $\displaystyle C_n = \frac{1}{n+1}\binom{2n}{n}$,

22. $\displaystyle\left\langle{n \atop 0}\right\rangle = \left\langle{n \atop n-1}\right\rangle = 1$, **23.** $\displaystyle\left\langle{n \atop k}\right\rangle = \left\langle{n \atop n-1-k}\right\rangle$, **24.** $\displaystyle\left\langle{n \atop k}\right\rangle = (k+1)\left\langle{n-1 \atop k}\right\rangle + (n-k)\left\langle{n-1 \atop k-1}\right\rangle$,

25. $\displaystyle\left\langle{0 \atop k}\right\rangle = \begin{cases}1 & \text{if } k = 0,\\ 0 & \text{otherwise}\end{cases}$ **26.** $\displaystyle\left\langle{n \atop 1}\right\rangle = 2^n - n - 1$, **27.** $\displaystyle\left\langle{n \atop 2}\right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2}$,

28. $\displaystyle x^n = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{x+k}{n}$, **29.** $\displaystyle\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $\displaystyle m!\left\{{n \atop m}\right\} = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{k}{n-m}$,

31. $\displaystyle\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{n}\left\{{n \atop k}\right\}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\displaystyle\left\langle\!\!\left\langle{n \atop 0}\right\rangle\!\!\right\rangle = 1$, **33.** $\displaystyle\left\langle\!\!\left\langle{n \atop n}\right\rangle\!\!\right\rangle = 0 \quad \text{for } n \ne 0$,

34. $\displaystyle\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle = (k+1)\left\langle\!\!\left\langle{n-1 \atop k}\right\rangle\!\!\right\rangle + (2n-1-k)\left\langle\!\!\left\langle{n-1 \atop k-1}\right\rangle\!\!\right\rangle$, **35.** $\displaystyle\sum_{k=0}^{n}\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

36. $\displaystyle\left\{{x \atop x-n}\right\} = \sum_{k=0}^{n}\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\displaystyle\left\{{n+1 \atop m+1}\right\} = \sum_{k}\binom{n}{k}\left\{{k \atop m}\right\} = \sum_{k=0}^{n}\left\{{k \atop m}\right\}(m+1)^{n-k}$,

Theoretical Computer Science Cheat Sheet

Identities Cont.

38. $\left[{n+1 \atop m+1}\right] = \sum_k \left[{n \atop k}\right]\binom{k}{m} = \sum_{k=0}^{n} \left[{k \atop m}\right] n^{\underline{n-k}} = n! \sum_{k=0}^{n} \frac{1}{k!}\left[{k \atop m}\right]$,

39. $\left[{x \atop x-n}\right] = \sum_{k=0}^{n} \left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle \binom{x+k}{2n}$,

40. $\left\{{n \atop m}\right\} = \sum_k \binom{n}{k}\left\{{k+1 \atop m+1}\right\}(-1)^{n-k}$,

41. $\left[{n \atop m}\right] = \sum_k \left[{n+1 \atop k+1}\right]\binom{k}{m}(-1)^{m-k}$,

42. $\left\{{m+n+1 \atop m}\right\} = \sum_{k=0}^{m} k\left\{{n+k \atop k}\right\}$,

43. $\left[{m+n+1 \atop m}\right] = \sum_{k=0}^{m} k(n+k)\left[{n+k \atop k}\right]$,

44. $\binom{n}{m} = \sum_k \left\{{n+1 \atop k+1}\right\}\left[{k \atop m}\right](-1)^{m-k}$,

45. $(n-m)!\binom{n}{m} = \sum_k \left[{n+1 \atop k+1}\right]\left\{{k \atop m}\right\}(-1)^{m-k}$, for $n \ge m$,

46. $\left\{{n \atop n-m}\right\} = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left[{m+k \atop k}\right]$,

47. $\left[{n \atop n-m}\right] = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left\{{m+k \atop k}\right\}$,

48. $\left\{{n \atop \ell+m}\right\}\binom{\ell+m}{\ell} = \sum_k \left\{{k \atop \ell}\right\}\left\{{n-k \atop m}\right\}\binom{n}{k}$,

49. $\left[{n \atop \ell+m}\right]\binom{\ell+m}{\ell} = \sum_k \left[{k \atop \ell}\right]\left[{n-k \atop m}\right]\binom{n}{k}$.

Trees

Every tree with n vertices has $n-1$ edges.

Kraft inequality: If the depths of the leaves of a binary tree are $d_1, \ldots, d_n$:
$$\sum_{i=1}^{n} 2^{-d_i} \le 1,$$
and equality holds only if every internal node has 2 sons.

Recurrences

Master method:
$$T(n) = aT(n/b) + f(n), \quad a \ge 1, b > 1$$

If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then
$$T(n) = \Theta(n^{\log_b a}).$$

If $f(n) = \Theta(n^{\log_b a})$ then
$$T(n) = \Theta(n^{\log_b a} \log_2 n).$$

If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \le cf(n)$ for large n, then
$$T(n) = \Theta(f(n)).$$

Substitution (example): Consider the following recurrence
$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have
$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by 2^{i+1} we get
$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find
$$u_{i+1} = \tfrac{1}{2} + u_i, \qquad u_1 = \tfrac{1}{2},$$

which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$.
Summing factors (example): Consider the following recurrence
$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving T are on the left side
$$T(n) - 3T(n/2) = n.$$

Now expand the recurrence, and choose a factor which makes the left side "telescope"

$$\begin{gathered}
1\big(T(n) - 3T(n/2) = n\big) \\
3\big(T(n/2) - 3T(n/4) = n/2\big) \\
\vdots \quad \vdots \quad \vdots \\
3^{\log_2 n - 1}\big(T(2) - 3T(1) = 2\big)
\end{gathered}$$

Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get
$$\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\tfrac{3}{2}\right)^i.$$

Let $c = \frac{3}{2}$. Then we have
$$\begin{aligned}
n \sum_{i=0}^{m-1} c^i &= n\left(\frac{c^m - 1}{c - 1}\right) \\
&= 2n(c^{\log_2 n} - 1) \\
&= 2n(c^{(k-1)\log_c n} - 1) \\
&= 2n^k - 2n,
\end{aligned}$$

and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider
$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that
$$T_{i+1} = 1 + \sum_{j=0}^{i} T_j.$$

Subtracting we find
$$\begin{aligned}
T_{i+1} - T_i &= 1 + \sum_{j=0}^{i} T_j - 1 - \sum_{j=0}^{i-1} T_j \\
&= T_i.
\end{aligned}$$

And so $T_{i+1} = 2T_i = 2^{i+1}$.

Generating functions:

1. Multiply both sides of the equation by x^i.
2. Sum both sides over all i for which the equation is valid.
3. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$.
3. Rewrite the equation in terms of the generating function $G(x)$.
4. Solve for $G(x)$.
5. The coefficient of x^i in $G(x)$ is g_i.

Example:
$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$

Multiply and sum:
$$\sum_{i \ge 0} g_{i+1} x^i = \sum_{i \ge 0} 2g_i x^i + \sum_{i \ge 0} x^i.$$

We choose $G(x) = \sum_{i \ge 0} x^i g_i$. Rewrite in terms of $G(x)$:
$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \ge 0} x^i.$$

Simplify:
$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$

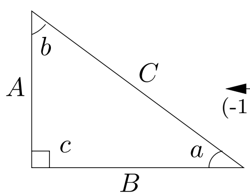
Solve for $G(x)$:
$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

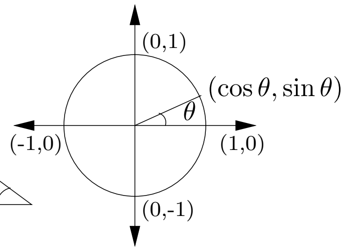
Expand this using partial fractions:
$$\begin{aligned}
G(x) &= x\left(\frac{2}{1-2x} - \frac{1}{1-x}\right) \\
&= x\left(2\sum_{i \ge 0} 2^i x^i - \sum_{i \ge 0} x^i\right) \\
&= \sum_{i \ge 0} (2^{i+1} - 1)x^{i+1}.
\end{aligned}$$

So $g_i = 2^i - 1$.

Theoretical Computer Science Cheat Sheet

Trigonometry





Pythagorean theorem:

$$C^2 = A^2 + B^2.$$

Definitions:

$$\sin a = A/C, \quad \cos a = B/C,$$
$$\csc a = C/A, \quad \sec a = C/B,$$
$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:

$$\tfrac{1}{2}AB, \quad \frac{AB}{A+B+C}.$$

Identities:

$$\sin x = \frac{1}{\csc x}, \qquad \cos x = \frac{1}{\sec x},$$
$$\tan x = \frac{1}{\cot x}, \qquad \sin^2 x + \cos^2 x = 1,$$
$$1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x,$$
$$\sin x = \cos\left(\tfrac{\pi}{2} - x\right), \qquad \sin x = \sin(\pi - x),$$
$$\cos x = -\cos(\pi - x), \qquad \tan x = \cot\left(\tfrac{\pi}{2} - x\right),$$
$$\cot x = -\cot(\pi - x), \qquad \csc x = \cot\tfrac{x}{2} - \cot x,$$
$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$
$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$
$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$
$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$
$$\sin 2x = 2 \sin x \cos x, \qquad \sin 2x = \frac{2\tan x}{1 + \tan^2 x},$$
$$\cos 2x = \cos^2 x - \sin^2 x, \qquad \cos 2x = 2\cos^2 x - 1,$$
$$\cos 2x = 1 - 2\sin^2 x, \qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$
$$\tan 2x = \frac{2\tan x}{1 - \tan^2 x}, \qquad \cot 2x = \frac{\cot^2 x - 1}{2\cot x},$$
$$\sin(x+y)\sin(x-y) = \sin^2 x - \sin^2 y,$$
$$\cos(x+y)\cos(x-y) = \cos^2 x - \sin^2 y.$$

Euler's equation:

$$e^{ix} = \cos x + i \sin x, \qquad e^{i\pi} = -1.$$

sseiden@acm.org
http://www.csc.lsu.edu/~seiden

Matrices

Multiplication:

$$C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Determinants: $\det A \neq 0$ iff A is non-singular.

$$\det A \cdot B = \det A \cdot \det B,$$
$$\det A = \sum_{\pi} \prod_{i=1}^{n} \operatorname{sign}(\pi) a_{i,\pi(i)}.$$

2×2 and 3×3 determinant:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$
$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g \begin{vmatrix} b & c \\ e & f \end{vmatrix} - h \begin{vmatrix} a & c \\ d & f \end{vmatrix} + i \begin{vmatrix} a & b \\ d & e \end{vmatrix}$$
$$= \begin{matrix} aei + bfg + cdh \\ - ceg - fha - ibd. \end{matrix}$$

Permanents:

$$\operatorname{perm} A = \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}.$$

Hyperbolic Functions

Definitions:

$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \operatorname{csch} x = \frac{1}{\sinh x},$$
$$\operatorname{sech} x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

Identities:

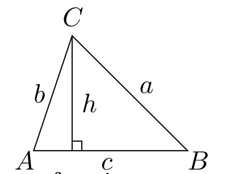
$$\cosh^2 x - \sinh^2 x = 1, \qquad \tanh^2 x + \operatorname{sech}^2 x = 1,$$
$$\coth^2 x - \operatorname{csch}^2 x = 1, \qquad \sinh(-x) = -\sinh x,$$
$$\cosh(-x) = \cosh x, \qquad \tanh(-x) = -\tanh x,$$
$$\sinh(x+y) = \sinh x \cosh y + \cosh x \sinh y,$$
$$\cosh(x+y) = \cosh x \cosh y + \sinh x \sinh y,$$
$$\sinh 2x = 2 \sinh x \cosh x,$$
$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$
$$\cosh x + \sinh x = e^x, \qquad \cosh x - \sinh x = e^{-x},$$
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$
$$2\sinh^2 \tfrac{x}{2} = \cosh x - 1, \qquad 2\cosh^2 \tfrac{x}{2} = \cosh x + 1.$$

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$
0	0	1	0
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$
$\frac{\pi}{2}$	1	0	∞

. . . in mathematics you don't understand things, you just get used to them.

– J. von Neumann

More Trig.



Law of cosines:

$$c^2 = a^2 + b^2 - 2ab\cos C.$$

Area:

$$A = \tfrac{1}{2}hc,$$
$$= \tfrac{1}{2}ab\sin C,$$
$$= \frac{c^2 \sin A \sin B}{2 \sin C}.$$

Heron's formula:

$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$
$$s = \tfrac{1}{2}(a + b + c),$$
$$s_a = s - a,$$
$$s_b = s - b,$$
$$s_c = s - c.$$

More identities:

$$\sin\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$
$$\cos\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$
$$\tan\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$
$$= \frac{1 - \cos x}{\sin x},$$
$$= \frac{\sin x}{1 + \cos x},$$
$$\cot\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$
$$= \frac{1 + \cos x}{\sin x},$$
$$= \frac{\sin x}{1 - \cos x},$$
$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$
$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$
$$\tan x = -i\frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$
$$= -i\frac{e^{2ix} - 1}{e^{2ix} + 1},$$
$$\sin x = \frac{\sinh ix}{i},$$
$$\cos x = \cosh ix,$$
$$\tan x = \frac{\tanh ix}{i}.$$