VICTUS ACM-ICPC Notebook 2018 (C++)

Contents

1 dsubipart

1.1 Bipart Check

```
int n;
vector<vector<int>> adj;
vector<int> side(n, -1);
bool is_bipartite = true;
queue<int> q;
for (int st = 0; st < n; ++st) {
    if (side[st] == -1) {
        q.push(st);
        side[st] = 0;
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int u : adj[v])
                if (side[u] == -1) {
                    side[u] = side[v] ^ 1
                    q.push(u);
                } else
                    is_bipartite &= side[u] !=
                        side[v];
        }
    }
}
cout << (is_bipartite ? "YES" : "NO") << endl;
```

1.2 Bipart Match

```
#define MAX 100001
#define NIL 0
#define INF (1<<28)
vector< int > G[MAX];
int n, m, match[MAX], dist[MAX];
// n: number of nodes on left side, nodes are
   numbered 1 to n
// m: number of nodes on right side, nodes are
    numbered n+1 to n+m
// G = NIL[0]   ł  G1[G[1---n]]   ł  G2[G[n+1---
   n+m]]
bool bfs() {
    int i, u, v, len;
    queue< int > Q;
    for(i=1; i<=n; i++) {
        if(match[i]==NIL) {
            dist[i] = 0;
            Q.push(i);
        }
        else dist[i] = INF;
    }
    dist[NIL] = INF;
```

```
    while(!Q.empty()) {
        u = Q.front(); Q.pop();
        if(u!=NIL) {
            len = G[u].size();
            for(i=0; i<len; i++) {
                v = G[u][i];
                if(dist[match[v]]==INF) {
                    dist[match[v]] = dist[u] +
                        1;
                    Q.push(match[v]);
                }
            }
        }
    }
    return (dist[NIL]!=INF);
}
bool dfs(int u) {
    int i, v, len;
    if(u!=NIL) {
        len = G[u].size();
        for(i=0; i<len; i++) {
            v = G[u][i];
            if(dist[match[v]]==dist[u]+1) {
                if(dfs(match[v])) {
                    match[v] = u;
                    match[u] = v;
                    return true;
                }
            }
        }
        dist[u] = INF;
        return false;
    }
    return true;
}
int hopcroft_karp() {
    int matching = 0, i;
    // match[] is assumed NIL for all vertex
       in G
    while(bfs())
        for(i=1; i<=n; i++)
            if(match[i]==NIL && dfs(i))
                matching++;
    return matching;
} //calling function
```

1.3 DSU

```
void make_set(int v) {
    parent[v] = make_pair(v, 0);
    rank[v] = 0;
    bipartite[v] = true;
}
pair<int, int> find_set(int v) {
    if (v != parent[v].first) {
        int parity = parent[v].second;
        parent[v] = find_set(parent[v].first);
        parent[v].second ^= parity;
    }
    return parent[v];
}
void add_edge(int a, int b) {
    pair<int, int> pa = find_set(a);
    a = pa.first;
    int x = pa.second;

    pair<int, int> pb = find_set(b);
    b = pb.first;
    int y = pb.second;

    if (a == b) {
        if (x == y)
            bipartite[a] = false;
    } else {
        if (rank[a] < rank[b])
            swap (a, b);
        parent[b] = make_pair(a, x^y^1);
        bipartite[a] &= bipartite[b];
        if (rank[a] == rank[b])
            ++rank[a];
    }
}
bool is_bipartite(int v) {
    return bipartite[find_set(v).first];
}
```

2 Geometry

2.1 Convex Hull

```
typedef pair<ll,ll> point;
```

```
ll cross (point a, point b, point c) {
return (b.x - a.x) * (c.y - a.y) - (b.y - a.y)
    * (c.x - a.x); }

vector<point> ConvexHull(vector<point>&p, ll n
   ) {
        ll sz = 0;
        vector<point> hull(n + n);
        sort(p.begin(), p.end());
        for(ll i = 0; i < n; ++i) {
                while (sz > 1 and cross(hull[
                   sz-2], hull[sz-1], p[i]) <=
                   0) --sz;
                        hull[sz++] = p[i];
        }
        for(ll i = n - 2, j = sz + 1; i >= 0;
           --i) {
                while (sz >= j and cross(hull[
                   sz-2], hull[sz-1], p[i]) <=
                   0) --sz;
                        hull[sz++] = p[i];
        }
        hull.resize(sz - 1);
        return hull;
}
```

2.2 Point inside polygon

```
//points inside convex poLygon O(Logn)
const ll N = 100009;
struct point {
        ll x,y;
}a[N];
ll n;
doubLe cross(const point& p1,const point& p2,
   const point& org) {
        return ((p1.x-org.x)*1.0)*(p2.y-org.y)
           -((p2.x-org.x)*1.0)*(p1.y-org.y);}

inLine booL comp(const point& x,const point& y
   ) {
        return cross(x,y,a[0]) >= 0;
}

booL inside(point& p) {
        if (cross(a[0], a[n-1], p)>=0) return
           faLse;
        if (cross(a[0], a[1], p)<=0) return
           faLse;
        ll L =1;
        ll r =n-1;
        whiLe(L<r) {
                ll m = L + (r-L)/2;
                if(cross(a[m],p,a[0]) >=0)
                        L=m+1;
                eLse
                r=m;
        }
        if(L == 0)
                return faLse;
return cross(a[L-1],a[L],p) >0;
}
sort(a+1,a+n,comp);
```

2.3 Geometry

```
//CIRCLE LINE INTERSECTION
double r, a, b, c; // given as input
double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b
   );
if (c*c > r*r*(a*a+b*b)+EPS)
    puts ("no points");
else if (abs (c*c - r*r*(a*a+b*b)) < EPS) {
    puts ("1 point");
    cout << x0 << ' ' << y0 << '\n';
}
else {
    double d = r*r - c*c/(a*a+b*b);
    double mult = sqrt (d / (a*a+b*b));
    double ax, ay, bx, by;
    ax = x0 + b * mult;
    bx = x0 - b * mult;
    ay = y0 - a * mult;
    by = y0 + a * mult;
    puts ("2 points");
    cout << ax << ' ' << ay << '\n' << bx << '
        ' << by << '\n';
}
//LENGTH OF UNION OF SEGMENTS
int length_union(const vector<pair<int, int>>
   &a) {
    int n = a.size();
    vector<pair<int, bool>> x(n*2);
    for (int i = 0; i < n; i++) {
        x[i*2+1] = {a[i].second, true};
```

```
    }
        x[i*2] = {a[i].first, false};

    sort(x.begin(), x.end());

    int result = 0;
    int c = 0;
    for (int i = 0; i < n * 2; i++) {
        if (i > 0 && x[i].first > x[i-1].first
            && c > 0)
            result += x[i].first - x[i-1].
               first;
        if (x[i].second)
            c++;
        else
            --c;
    }
    return result;
}
//LATTICE POINTS
int count_lattices(Fraction k, Fraction b,
   long long n) {
    auto fk = k.floor();
    auto fb = b.floor();
    auto cnt = 0LL;
    if (k >= 1 || b >= 1) {
        cnt += (fk * (n - 1) + 2 * fb) * n /
           2;
        k -= fk;
        b -= fb;
    }
    auto t = k * n + b;
    auto ft = t.floor();
    if (ft >= 1) {
        cnt += count_lattices(1 / k, (t - t.
           floor()) / k, t.floor());
    }
    return cnt;
}
//AREA OF POLYGON
double area(const vector<point>& fig) {
    double res = 0;
    for (unsigned i = 0; i < fig.size(); i++)
       {
        point p = i ? fig[i - 1] : fig.back();
        point q = fig[i];
        res += (p.x - q.x) * (p.y + q.y);
    }
    return fabs(res) / 2;

//RECTANGLES OVERLAPING AREA
//l1,r1 are second diag's opposite points
int overlappingArea(Point l1, Point r1, Point
   l2, Point r2)
{
    // areai : Area of ith Rectangle
    int area1 = abs(l1.x - r1.x) * abs(l1.y -
       r1.y);
    // Length of intersecting part i.e start
       from max(l1.x, l2.x) of x-coordinate and
        end at min(r1.x,r2.x) x-coordinate by
       subtracting start from end we get
       required lengths
    int areaI = (min(r1.x, r2.x)-max(l1.x, l2.
       x))*(min(r1.y, r2.y)-max(l1.y, l2.y));
    return (area1 + area2 - areaI);
}

//FINDING INTERSECTION OF TWO SEGMENTS
const double EPS = 1E-9;
struct pt {
    double x, y;
    bool operator<(const pt& p) const{
        return x < p.x - EPS || (abs(x - p.x)
           < EPS && y < p.y - EPS);
    }
};
struct line {
    double a, b, c;
    line() {}
    line(pt p, pt q){
        a = p.y - q.y;
        b = q.x - p.x;
        c = -a * p.x - b * p.y;
        norm();
    }
    void norm(){
        double z = sqrt(a * a + b * b);
        if (abs(z) > EPS)
            a /= z, b /= z, c /= z;
    }
    double dist(pt p) const { return a * p.x +
        b * p.y + c; }
};
double det(double a, double b, double c,
   double d){
```

```
    return a * d - b * c;
}
inline bool betw(double l, double r, double x)
    {
    return min(l, r) <= x + EPS && x <= max(l,
        r) + EPS;
}
inline bool intersect_1d(double a, double b,
    double c, double d){
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    return max(a, c) <= min(b, d) + EPS;
}
bool intersect(pt a, pt b, pt c, pt d, pt&
    left, pt& right){
    if (!intersect_1d(a.x, b.x, c.x, d.x) || !
        intersect_1d(a.y, b.y, c.y, d.y))
        return false;
    line m(a, b);
    line n(c, d);
    double zn = det(m.a, m.b, n.a, n.b);
    if (abs(zn) < EPS) {
        if (abs(m.dist(c)) > EPS || abs(n.dist
            (a)) > EPS)         return false;
        if (b < a)          swap(a, b);
        if (d < c)          swap(c, d);
        left = max(a, c);
        right = min(b, d);
        return true;
    } else {
        left.x = right.x = -det(m.c, m.b, n.c,
            n.b) / zn;
        left.y = right.y = -det(m.a, m.c, n.a,
            n.c) / zn;
        return betw(a.x, b.x, left.x) && betw(
            a.y, b.y, left.y) &&
                betw(c.x, d.x, left.x) && betw(
                    c.y, d.y, left.y);
    }
}
```

3 Graphs

3.1 Bridge

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of
    graph
vector<bool> visited;
vector<int> tin, fup;
int timer;
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = fup[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            fup[v] = min(fup[v], tin[to]);
        } else {
            dfs(to, v);
            fup[v] = min(fup[v], fup[to]);
            if (fup[to] > tin[v])
                IS_BRIDGE(v, to); // bridge
                    found from v-to
        }
    }
}
void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    fup.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

3.2 Merge Sort Tree

```
vector<int> t[4*MAXN];

void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = vector<int>(1, a[tl]);
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        merge(t[v*2].begin(), t[v*2].end(), t[
            v*2+1].begin(), t[v*2+1].end(),
```

```
                back_inserter(t[v]));
    }
}


int query(int v, int tl, int tr, int l, int r,
    int x) {
    if (l > r)
        return INF;
    if (l == tl && r == tr) {
        vector<int>::iterator pos =
           lower_bound(t[v].begin(), t[v].end()
           , x);
        if (pos != t[v].end())
            return *pos;
        return INF;
    }
    int tm = (tl + tr) / 2;
    return min(query(v*2, tl, tm, l, min(r, tm
       ), x),
               query(v*2+1, tm+1, tr, max(l,
                  tm+1), r, x));
}

void update(int v, int tl, int tr, int pos,
   int new_val) {
    t[v].erase(t[v].find(a[pos]));
    t[v].insert(new_val);
    if (tl != tr) {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos,
               new_val);
    } else {
        a[pos] = new_val;
    }
}
```

3.3 LCA

```
struct LCA {
    vector<int> height, euler, first, segtree;
    vector<bool> visited;
    int n;
    LCA(vector<vector<int>> &adj, int root =
       0) {
        n = adj.size();
        height.resize(n);
        first.resize(n);
        euler.reserve(n * 2);
        visited.assign(n, false);
        dfs(adj, root);
        int m = euler.size();
        segtree.resize(m * 4);
        build(1, 0, m - 1);
    }
    void dfs(vector<vector<int>> &adj, int
       node, int h = 0) {
        visited[node] = true;
        height[node] = h;
        first[node] = euler.size();
        euler.push_back(node);
        for (auto to : adj[node]) {
            if (!visited[to]) {
                dfs(adj, to, h + 1);
                euler.push_back(node);
            }
        }
    }
    void build(int node, int b, int e) {
        if (b == e) {
            segtree[node] = euler[b];
        } else {
            int mid = (b + e) / 2;
            build(node << 1, b, mid);
            build(node << 1 | 1, mid + 1, e);
            int l = segtree[node << 1], r =
               segtree[node << 1 | 1];
            segtree[node] = (height[l] <
               height[r]) ? l : r;
        }
    }
    int query(int node, int b, int e, int L,
       int R) {
        if (b > R || e < L)
            return -1;
        if (b >= L && e <= R)
            return segtree[node];
        int mid = (b + e) >> 1;
        int left = query(node << 1, b, mid, L,
            R);
```

```
        int right = query(node << 1 | 1, mid +
            1, e, L, R);
        if (left == -1) return right;
        if (right == -1) return left;
        return height[left] < height[right] ?
           left : right;
    }
    int lca(int u, int v) {
        int left = first[u], right = first[v];
        if (left > right)
            swap(left, right);
        return query(1, 0, euler.size() - 1,
           left, right);
    }
};
```

3.4 01bfs

```
d.assign(n, INF);
d[s] = 0;
set<pair<int, int>> q;
q.insert({0, s});
while (!q.empty()) {
    int v = q.begin()->second;
    q.erase(q.begin());

    for (auto edge : adj[v]) {
        int u = edge.first;
        int w = edge.second;

        if (d[v] + w < d[u]) {
            q.erase({d[u], u});
            d[u] = d[v] + w;
            q.insert({d[u], u});
        }
    }
}
```

```
vector<int> d(n, INF);
d[s] = 0;
deque<int> q;
q.push_front(s);
while (!q.empty()) {
    int v = q.front();
    q.pop_front();
    for (auto edge : adj[v]) {
        int u = edge.first;
        int w = edge.second;
        if (d[v] + w < d[u]) {
            d[u] = d[v] + w;
            if (w == 1)
                q.push_back(u);
            else
                q.push_front(u);
        }
    }
}
```

3.5 2 SAT

```
int n;
vector<vector<int>> g, gt;
vector<bool> used;
vector<int> order, comp;
vector<bool> assignment;
void dfs1(int v) {
    used[v] = true;
    for (int u : g[v]) {
        if (!used[u])
            dfs1(u);
    }
    order.push_back(v);
}
void dfs2(int v, int cl) {
    comp[v] = cl;
    for (int u : gt[v]) {
        if (comp[u] == -1)
            dfs2(u, cl);
    }
}
bool solve_2SAT() {
    used.assign(n, false);
    for (int i = 0; i < n; ++i) {
        if (!used[i])
            dfs1(i);
    }
    comp.assign(n, -1);
    for (int i = 0, j = 0; i < n; ++i) {
        int v = order[n - i - 1];
        if (comp[v] == -1)
            dfs2(v, j++);
    }
    assignment.assign(n / 2, false);
```

```
    for (int i = 0; i < n; i += 2) {
        if (comp[i] == comp[i + 1])
            return false;
        assignment[i / 2] = comp[i] > comp[i +
            1];
    }
    return true;
}
```

3.6 Articulation

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of
   graph
vector<bool> visited;
vector<int> tin, fup;
int timer;
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = fup[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            fup[v] = min(fup[v], tin[to]);
        } else {
            dfs(to, v);
            fup[v] = min(fup[v], fup[to]);
            if (fup[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v); // v is cut-point
}
void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    fup.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

3.7 BIT

```
int BIT[1000], a[1000], n;
void update(int x, int delta){
      for(; x <= n; x += x&-x)
        BIT[x] += delta;
}
int query(int x){
     int sum = 0;
     for(; x > 0; x -= x&-x)
        sum += BIT[x];
     return sum;
}
```

3.8 PRIM

```
int n;
vector<vector<int>> adj; // adjacency matrix
   of graph
const int INF = 1000000000; // weight INF
   means there is no edge
struct Edge {
    int w = INF, to = -1;
};
void prim() {
    int total_weight = 0;
    vector<bool> selected(n);
    vector<Edge> min_e(n);
    min_e[0].w = 0;
    for (int i=0; i<n; ++i) {
        int v = -1;
        for (int j = 0; j < n; ++j) {
            if (!selected[j] && (v == -1 ||
                min_e[j].w < min_e[v].w))
                v = j;
        }
        if (min_e[v].w == INF) {
            cout << "No MST!" << endl;
            exit(0);
        }
        selected[v] = true;
        total_weight += min_e[v].w;
        if (min_e[v].to != -1)
            cout << v << " " << min_e[v].to <<
                endl;
```

```
            for (int to = 0; to < n; ++to) {
                if (adj[v][to] < min_e[to].w)
                    min_e[to] = {adj[v][to], v};
            }
        }
        cout << total_weight << endl;
    }
```

3.9 SCC

```
vector<ll>adj[400005],adjr[400005];
vector<ll>visited(400005,0),visitedr(400005,0)
   ;
vector<ll>order,component;
void dfs1(ll src){
        visited[src] = 1;
        for(auto e:adj[src])
                if(!visited[e])
                        dfs1(e);
        order.pb(src);
}
void dfs2(ll src){
        visitedr[src] = 1;
        component.pb(src);
        for(auto e:adjr[src])
                if(!visitedr[e])
                        dfs2(e);
}
for(int i=0;i<n;i++){
        cin >> a >> b;
        adj[a].push_back(b);
        adjr[b].push_back(a);
}
for(int i=0;i<n;++i)
        if(!visited[i])
                dfs1(i);
for(int i=0;i<n;++i){
        ll v = order[n-1-i];
        if(!visitedr[v]){
                dfs2 (v);component.clear();
        }
}
```

3.10 Topo

```
int n; // number of vertices
vector<vector<int>> adj; // adjacency list of
   graph
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());
}
```

3.11 TRIE

```
const int ALPHABET_SIZE = 26;
struct TrieNode {
        struct TrieNode *children[
           ALPHABET_SIZE]; // isEndOfWord is
           true if the node represents end of a
            word
        bool isEndOfWord;
};
struct TrieNode *getNode(void) {
        struct TrieNode *pNode = new TrieNode;
        pNode->isEndOfWord = false;
        for (int i = 0; i < ALPHABET_SIZE; i
           ++)
                pNode->children[i] = NULL;
        return pNode;
}
void insert(struct TrieNode *root, string key)
    {
        struct TrieNode *pCrawl = root;
        for (int i = 0; i < key.length(); i++)
            {
```

```
                int index = key[i] - 'a';
                if (!pCrawl->children[index])
                        pCrawl->children[index
                           ] = getNode();
                pCrawl = pCrawl->children[
                   index];
        } // mark last node as leaf
        pCrawl->isEndOfWord = true;
}
bool search(struct TrieNode *root, string key)
    {
        struct TrieNode *pCrawl = root;
        for (int i = 0; i < key.length(); i++)
            {
                int index = key[i] - 'a';
                if (!pCrawl->children[index])
                        return false;
                pCrawl = pCrawl->children[
                   index];
        }
        return (pCrawl != NULL && pCrawl->
           isEndOfWord);
}
struct TrieNode *root = getNode();
for (int i = 0; i < n; i++) insert(root, keys[
   i]);
search(root, "the");
```

4 Flows

4.1 Max Flow

```
int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity
               [cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow,
                   capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

5 Math

5.1 CRT

```
from functools import reduce
def chinese_remainder(n, a):
    sum = 0
    prod = reduce(lambda a, b: a*b, n)
    for n_i, a_i in zip(n, a):
        p = prod / n_i
        sum += a_i * mul_inv(p, n_i) * p
```

```
    return sum % prod

def mul_inv(a, b):
    b0 = b
    x0, x1 = 0, 1
    if b == 1: return 1
    while a > 1:
        q = a // b
        a, b = b, a%b
        x0, x1 = x1 - q * x0, x0
    if x1 < 0: x1 += b0
    return x1
if __name__ == '__main__':
    n = [3, 5, 7]        #xi=ai%ni
    a = [2, 3, 2]
    print(chinese_remainder(n, a))
```

5.2 DigitDP

```
/// How many numbers x are there in the range
   a to b, where the digit d occurs exactly k
   times in x?
vector<int> num;
int a, b, d, k;
int DP[12][12][2];
/// DP[p][c][f] = Number of valid numbers <= b
    from this state
/// p = current position from left side (zero
   based)
/// c = number of times we have placed the
   digit d so far
/// f = the number we are building has already
    become smaller than b? [0 = no, 1 = yes]
int call(int pos, int cnt, int f){
    if(cnt > k) return 0;

    if(pos == num.size()){
        if(cnt == k) return 1;
        return 0;}

    if(DP[pos][cnt][f] != -1) return DP[pos][
       cnt][f];
    int res = 0;
    int LMT;

    if(f == 0){
        /// Digits we placed so far matches
           with the prefix of b
        /// So if we place any digit > num[pos
           ] in the current position, then the
           number will become greater than b
        LMT = num[pos];
    } else {
        /// The number has already become
           smaller than b. We can place any
           digit now.
        LMT = 9;
    }

    /// Try to place all the valid digits such
        that the number doesn't exceed b
    for(int dgt = 0; dgt<=LMT; dgt++){
        int nf = f;
        int ncnt = cnt;
        if(f == 0 && dgt < LMT) nf = 1; ///
           The number is getting smaller at
           this position
        if(dgt == d) ncnt++;
        if(ncnt <= k) res += call(pos+1, ncnt,
            nf);
    }

    return DP[pos][cnt][f] = res;
}

int solve(int b){
    num.clear();
    while(b>0){
        num.push_back(b%10);
        b/=10;
    }
    reverse(num.begin(), num.end());
    /// Stored all the digits of b in num for
       simplicity
    memset(DP, -1, sizeof(DP));
    int res = call(0, 0, 0);
    return res;
}

int main () {
 cin >> a >> b >> d >> k;
    int res = solve(b) - solve(a-1);
    cout << res << endl;
```

```
}
```

5.3 Diophantine

```
bool find_any_solution(int a, int b, int c,
    int &x0, int &y0, int &g) { //check if
    solution exists
    g = gcd(abs(a), abs(b), x0, y0); //
        extended-gcd
    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) { //
    returns no. of solution
    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;
}
```

5.4 Euclidean

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

5.5 INCLU Exclu

```
int solve (int n, int r) {
    vector<int> p;
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0) {
            p.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        p.push_back (n);
```

```
    int sum = 0;
    for (int msk=1; msk<(1<<p.size()); ++msk)
       {
        int mult = 1,
            bits = 0;
        for (int i=0; i<(int)p.size(); ++i)
            if (msk & (1<<i)) {
                ++bits;
                mult *= p[i];
            }
        int cur = r / mult;
        if (bits % 2 == 1)
            sum += cur;
        else
            sum -= cur;
    }
    return r - sum;
}
```

5.6 Linear eq

```
int gauss (vector < vector<double> > a, vector
   <double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;
    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++
       col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][
               col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;
        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][
                   col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }
    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[
               i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }
    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

5.7 Matrix expo

```
//answer in F
void power(ll F[2][2],ll n){
        if(n<=1)
                return;
        ll M[2][2]={{(2*f)%MOD,-1},{1,0}};
        power(F,n/2);multi(F,F);
        if(n%2)
                multi(F,M);
}
void multi(ll F[2][2],ll M[2][2]){
        ll x=(F[0][0]%MOD*M[0][0]%MOD)%MOD+(F
           [0][1]%MOD*M[1][0]%MOD)%MOD;
        ll y=(F[0][0]%MOD*M[0][1]%MOD)%MOD+(F
           [0][1]%MOD*M[1][1]%MOD)%MOD;
        ll z=(F[1][0]%MOD*M[0][0]%MOD)%MOD+(F
           [1][1]%MOD*M[1][0]%MOD)%MOD;
        ll w=(F[1][0]%MOD*M[0][1]%MOD)%MOD+(F
           [1][1]%MOD*M[1][1]%MOD)%MOD;
        if(x<0)
                x=(x+MOD)%MOD;
        if(y<0)
                y=(y+MOD)%MOD;
        if(z<0)
                z=(z+MOD)%MOD;
        if(x<0)
                w=(w+MOD)%MOD;
        F[0][0]=x;F[0][1]=y;F[1][0]=z;F[1][1]=
```

```
        w;
}
```

5.8 Seg sieve

```
#define MAX 46656
#define LMT 216
#define LEN 4830
#define RNG 100032
#define sq(x) ((x)*(x))
#define mset(x,v) memset(x, v , sizeof(x))
#define chkC(x,n) (x[n >> 6] & (1 << ((n >> 1)
    & 31)))
#define setC(x,n) (x[n >> 6] |= (1 << ((n >>
   1) & 31)))
using namespace std;
unsigned base[MAX/64], segment[RNG/64], primes
   [LEN];

/*
 * Generates all the necessary prime numbers
    and marks them in base[]
 */
void sieve()
{
    unsigned i, j, k;
    for (i = 3; i < LMT; i += 2)
    {
        if (!chkC(base, i))
        {
            for (j = i * i, k = i << 1; j <
               MAX; j += k)
                setC(base, j);
        }
    }
    for (i = 3, j = 0; i < MAX; i += 2)
    {
        if (!chkC(base, i))
            primes[j++] = i;
    }
}

/*
 * Returns the prime-count within range [a,b]
    and marks them in segment[]
 */
int segmented_sieve(int a, int b)
{
    unsigned i, j, k, cnt = (a <= 2 && 2 <=b )
       ? 1 : 0;
    if (b < 2)
        return 0;
    if (a < 3)
        a = 3;
    if (a % 2 == 0)
        a++;
    mset (segment, 0);
    for (i = 0; sq(primes[i]) <= b; i++)
    {
        j = primes[i] * ((a + primes[i] - 1) /
            primes[i]);
        if (j % 2 == 0) j += primes[i];
        for (k = primes[i] << 1; j <= b; j +=
           k)
        {
            if (j != primes[i])
                setC(segment, (j - a));
        }
    }
    for (i = 0; i <= b - a; i += 2)
    {
        if (!chkC(segment, i))
            cnt++;
    }
    return cnt;
}
int main()
{
    sieve();
    int a, b;
    cout<<"Enter Lower Bound: ";
    cin>>a;
    cout<<"Enter Upper Bound: ";
    cin>>b;
    cout<<"Number of primes between "<<a<<"
       and "<<b<<":  ";
    cout<<segmented_sieve(a, b)<<endl;
}
```

5.9 Euler totient

```
//phi
int totient[100008];
```

```
void phi(){
        for(int i=1;i<=100000;i++){
                int ans=i;
                set<int> s;
                int temp=i;
                while(temp!=1){
                        s.insert(pr[temp]); //
                           pr is spf
                        temp/=pr[temp];
                }
                for(auto e:s){
                        ans-=ans/e;
                }
                totient[i]=ans;
        }
}
```

6 Strings

6.1 Knuth Morris Pratt Algorithm

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
// Counting the number of occurrences of each
   prefix
vector<int> ans(n + 1);
for (int i = 0; i < n; i++)
    ans[pi[i]]++;
for (int i = n-1; i > 0; i--)
    ans[pi[i-1]] += ans[i];
for (int i = 0; i <= n; i++)
    ans[i]++;
```

6.2 String hashing

```
long long compute_hash(string const& s) {
    const int p = 31;
    const int m = 1e9 + 9;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' +
           1) * p_pow) % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
```

6.3 z function

```
vector<ll>z;
void zfunc(string s){                    //
   caLcuLates z vaLue at index i such that
   maximum prefix Length for string p starting
   from index i
  ll sz = s.size();
  z.pb(-1);
  ll L=0,r=0;
  for(int i=1;i<sz;i++){
    if(i>r){
      L=i;r=i;
      whiLe(r<sz && s[r-L]==s[r])
        r++;
      z.pb(r-L);r--;
    }
    eLse{
      ll k = i-L;
      if(z[k]<r-i+1)
        z.pb(z[k]);
      eLse
      {
        L=i;
        whiLe(r<sz && s[r-L]==s[r])
          r++;
        z.pb(r-L);r--;
      }
    }
  }
}
```

7 EZPZ

7.1 dp recurences

```
// matrix chain multiplication
// Matrix Ai has dimension p[i-1] x p[i] for i
    = 1..n
int MatrixChainOrder(int p[], int n)
{ /* For simplicity of the program, one extra
   row and one
        extra column are allocated in m[][]. 0
           th row and 0th
        column of m[][] are not used */
        int m[n][n];
        int i, j, k, L, q;
/* m[i,j] = Minimum number of scalar
   multiplications needed
        to compute the matrix A[i]A[i+1]...A[j
           ] = A[i..j] where
        dimension of A[i] is p[i-1] x p[i] */
// cost is zero when multiplying one matrix.
        for (i=1; i<n; i++)
                m[i][i] = 0;
// L is chain length.
        for (L=2; L<n; L++)
        {
                for (i=1; i<n-L+1; i++) {
                        j = i+L-1;
                        m[i][j] = INT_MAX;
                        for (k=i; k<=j-1; k++)
                            {
                                // q = cost/
                                   scalar
                                   multiplications

                                q = m[i][k] +
                                   m[k+1][j] +
                                   p[i-1]*p[k]*
                                   p[j];
                                if (q < m[i][j
                                   ])
                                        m[i][j
                                           ] =
                                           q; }
                }
        }
    return m[1][n-1];
}
// bell no
//Let S(n, k) be total number of partitions of
    n elements into k sets. The value of n th
    Bell Number is sum of S(n, k) for k = 1 to
   n.
int bellNumber(int n)
{
   int bell[n+1][n+1];
   bell[0][0] = 1;
   for (int i=1; i<=n; i++)
   {
      // Explicitly fill for j = 0
      bell[i][0] = bell[i-1][i-1];

      // Fill for remaining values of j
      for (int j=1; j<=i; j++)
         bell[i][j] = bell[i-1][j-1] + bell[i
            ][j-1];
   }
   return bell[n][0];
}
// subset sum
isSubsetSum(set, n, sum) = isSubsetSum(set, n
   -1, sum) ||
                           isSubsetSum(set, n
                              -1, sum-set[n-1])
Base Cases:
isSubsetSum(set, n, sum) = false, if sum > 0
   and n == 0
isSubsetSum(set, n, sum) = true, if sum == 0
//rod cutting
Let cutRod(n) be the required (best possible
   price) value for a rod of length n. cutRod(n
   ) can be written as following.

cutRod(n) = max(price[i] + cutRod(n-i-1)) for
   all i in {0, 1 .. n-1}
//LCS
/* Returns length of LCS for X[0..m-1], Y[0..n
   -1] */
int lcs( char *X, char *Y, int m, int n )
{
   if (m == 0 || n == 0)
```

```
    return 0;
  if (X[m-1] == Y[n-1])
    return 1 + lcs(X, Y, m-1, n-1);
  else
    return max(lcs(X, Y, m, n-1), lcs(X, Y, m
       -1, n));
}
//
The longest common suffix has following
   optimal substructure property
   LCSuff(X, Y, m, n) = LCSuff(X, Y, m-1, n-1)
       + 1 if X[m-1] = Y[n-1]
                          0  Otherwise (if X[m
                             -1] != Y[n-1])

The maximum length Longest Common Suffix is
   the longest common substring.
   LCSubStr(X, Y, m, n)   = Max(LCSuff(X, Y, i,
       j)) where 1 <= i <= m
```

```
//Kadane
    max_so_far = 0
    max_ending_here = 0

Loop for each element of the array
  (a) max_ending_here = max_ending_here + a[i]
  (b) if(max_ending_here < 0)
            max_ending_here = 0
  (c) if(max_so_far < max_ending_here)
            max_so_far = max_ending_here
return max_so_far
//0-1 knapsack
int knapSack(int W, int wt[], int val[], int n
   )
{
   // Base Case
   if (n == 0 || W == 0)
       return 0;

   // If weight of the nth item is more than
      Knapsack capacity W, then
   // this item cannot be included in the
      optimal solution
   if (wt[n-1] > W)
       return knapSack(W, wt, val, n-1);

   // Return the maximum of two cases:
   // (1) nth item included
   // (2) not included
   else return max( val[n-1] + knapSack(W-wt[n
      -1], wt, val, n-1),
                    knapSack(W, wt, val, n-1)
                  );
}
//Egg-Droping
 k ==> Number of floors
 and n ==> Number of Eggs
 eggDrop(n, k) ==> Minimum number of trials
    needed to find the critical
                      floor in worst case.
 eggDrop(n, k) = 1 + min{max(eggDrop(n - 1, x
    - 1), eggDrop(n, k - x)):
                  x in {1, 2, ..., k}}
//Partition Problem
Let isSubsetSum(arr, n, sum/2) be the function
    that returns true if
there is a subset of arr[0..n-1] with sum
   equal to sum/2

The isSubsetSum problem can be divided into
   two subproblems
 a) isSubsetSum() without considering last
    element
    (reducing n to n-1)
 b) isSubsetSum considering the last element
    (reducing sum/2 by arr[n-1] and n to n-1)
If any of the above the above subproblems
   return true, then return true.
isSubsetSum (arr, n, sum/2) = isSubsetSum (arr
   , n-1, sum/2) ||
                                isSubsetSum (arr
                                   , n-1, sum/2 -
                                    arr[n-1])
//Longest Palindromic Subsequence
```

```
// Every single character is a palindrome of
   length 1
L(i, i) = 1 for all indexes i in given
   sequence

// IF first and last characters are not same
If (X[i] != X[j])  L(i, j) =  max{L(i + 1, j),
   L(i, j - 1)}

// If there are only 2 characters and both are
    same
Else if (j == i + 1) L(i, j) = 2

// If there are more than two characters, and
   first and last
// characters are same
Else L(i, j) =  L(i + 1, j - 1) + 2
//Coin Change
To count the total number of solutions, we can
    divide all set solutions into two sets.
1) Solutions that do not contain mth coin (or
   Sm).
2) Solutions that contain at least one Sm.
Let count(S[], m, n) be the function to count
   the number of solutions, then it can be
   written as sum of count(S[], m-1, n) and
   count(S[], m, n-Sm).
//LOngest repeating Subsequence
int findLongestRepeatingSubSeq(string X, int m
   , int n)
{
    if(dp[m][n]!=-1)
    return dp[m][n];

    // return if we have reached the end of
       either string
    if (m == 0 || n == 0)
        return dp[m][n] = 0;

    // if characters at index m and n matches
    // and index is different
    if (X[m - 1] == X[n - 1] && m != n)
        return dp[m][n] =
           findLongestRepeatingSubSeq(X,
                          m - 1, n - 1) + 1;

    // else if characters at index m and n don
       't match
    return dp[m][n] = max (
       findLongestRepeatingSubSeq(X, m, n - 1),
                                findLongestRepeatingSubSeq
                                   (X, m - 1, n));
}
// job-scheduling
1) First sort jobs according to finish time.
2) Now apply following recursive process.
   // Here arr[] is array of n jobs
   findMaximumProfit(arr[], n)
   {
     a) if (n == 1) return arr[0];
     b) Return the maximum of following two
        profits.
          (i) Maximum profit by excluding
             current job, i.e.,
              findMaximumProfit(arr, n-1)
          (ii) Maximum profit by including the
             current job
   }
//L[0] = {job[0]}
L[i] = {MaxSum(L[j])} + job[i] where j < i and
    job[j].finish <= job[i].start
     = job[i], if there is no such j
```

7.2 Lazy

```
Range updates (Lazy Propagation)

Addition on segments

void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = 0;
    }
}

void update(int v, int tl, int tr, int l, int
   r, int add) {
    if (l > r)
```

```
        return;
    if (l == tl && r == tr) {
        t[v] += add;
    } else {
        int tm = (tl + tr) / 2;
        update(v*2, tl, tm, l, min(r, tm), add
           );
        update(v*2+1, tm+1, tr, max(l, tm+1),
           r, add);
    }
}

int get(int v, int tl, int tr, int pos) {
    if (tl == tr)
        return t[v];
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return t[v] + get(v*2, tl, tm, pos);
    else
        return t[v] + get(v*2+1, tm+1, tr, pos
           );
}
```

Assignment on segments

Suppose now that the modification query asks to assign each element of a certain segment a[l r] to some value p. As a second query we will again consider reading the value of the array a[i]

```
void push(int v) {
    if (marked[v]) {
        t[v*2] = t[v*2+1] = t[v];
        marked[v*2] = marked[v*2+1] = true;
        marked[v] = false;
    }
}

void update(int v, int tl, int tr, int l, int
   r, int new_val) {
    if (l > r)
        return;
    if (l == tl && tr == r) {
        t[v] = new_val;
        marked[v] = true;
    } else {
        push(v);
        int tm = (tl + tr) / 2;
        update(v*2, tl, tm, l, min(r, tm),
           new_val);
        update(v*2+1, tm+1, tr, max(l, tm+1),
           r, new_val);
    }
}

int get(int v, int tl, int tr, int pos) {
    if (tl == tr) {
        return t[v];
    }
    push(v);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return get(v*2, tl, tm, pos);
    else
        return get(v*2+1, tm+1, tr, pos);
}
```

Adding on segments, querying for maximum

```
void push(int v) {
    t[v*2] += lazy[v];
    lazy[v*2] += lazy[v];
    t[v*2+1] += lazy[v];
    lazy[v*2+1] += lazy[v];
    lazy[v] = 0;
}

void update(int v, int tl, int tr, int l, int
   r, int addend) {
    if (l > r)
        return;
    if (l == tl && tr == r) {
        t[v] += addend;
        lazy[v] += addend;
    } else {
        push(v);
        int tm = (tl + tr) / 2;
        update(v*2, tl, tm, l, min(r, tm),
           addend);
        update(v*2+1, tm+1, tr, max(l, tm+1),
           r, addend);
        t[v] = max(t[v*2], t[v*2+1]);
    }
}

int query(int v, int tl, int tr, int l, int r)
```

```
    {
    if (l > r)
        return -INF;
    if (l <= tl && tr <= r)
        return t[v];
    push(v);
    int tm = (tl + tr) / 2;
    return max(query(v*2, tl, tm, l, min(r, tm
       )),
               query(v*2+1, tm+1, tr, max(l,
                  tm+1), r));
}
```

7.3 LIS nlogn

```
//lis NLOGN
int lis(int a[],int n){
        ll dp[n+3];
        //int lis[n+3];
        //ms(lis,0,sz lis);
        dp[0]=-LLONG_MAX;
        for(int i=1;i<=n;i++){
                dp[i]=LLONG_MAX;
        }
int anss=-1;
for(int i=1;i<=n;i++){
        int l=1,r=n,ans;
        while(l<=r){
                int mid=(l+r)/2;
                if(a[i]<=dp[mid]){
                        ans=mid;
                        r=mid-1;
                }
                else{
                        l=mid+1;
                }
        }
        dp[ans]=a[i];
        //lis[i]=max(lis[i],ans);
        anss=max(anss,ans);
        }
return anss;
```

7.4 MOs

```
ll block; //sqrt(N)
struct QUERY{
        ll L,R;
};
bool compare(QUERY a,QUERY b){
        if(a.L/block != b.L/block)
                return (a.L/block) < (b.L/
                   block);
        return a.R<b.R;
}
void mo(vector<ll>a,vector<QUERY>q){
        block = sqrt(a.size());
        sort(q.begin(),q.end(),compare);
        ll curL=0,curR=0,curSum=0;
        for(int i=0;i<q.size();i++){
                ll L = q[i].L,R = q[i].R;
                while(curL<L){
                        curSum-=a[curL];
                        curL++;
                }
                while(curL>L){
                        curSum+=a[curL];
                        curL--;
                }
                while(curR<=R){
                        curSum+=a[curR];
                        curR++;
                }
                while(curR>(R+1)){
                        curSum-=a[curR-1];
                        curR--;
                }
                cout << curSum << "\n";
        }
}
```

7.5 nge stack

```
void printNGE(int arr[], int n) {
  stack < int > s;

  /* push the first element to stack */
  s.push(arr[0]);

  // iterate for rest of the elements
  for (int i = 1; i < n; i++) {

    if (s.empty()) {
      s.push(arr[i]);
```

```
            continue;
        }

        /* if stack is not empty, then
           pop an element from stack.
           If the popped element is smaller
           than next, then
        a) print the pair
        b) keep popping while elements are
        smaller and stack is not empty */
        while (s.empty() == false && s.top() < arr
           [i])
        {
            cout << s.top() << " --> " << arr[i]
               << endl;
            s.pop();
        }

        /* push next to stack so that we can find
        next greater for it */
        s.push(arr[i]);
    }

    /* After iterating over the loop, the
       remaining
    elements in stack do not have the next
       greater
    element, so print -1 for them */
    while (s.empty() == false) {
        cout << s.top() << " --> " << -1 << endl;
        s.pop();
    }
}
```

7.6 Segment CP

```
// Normal Segment tree
int n, t[4*MAXN];

void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = t[v*2] + t[v*2+1];
    }
}

int sum(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && r == tr) {
        return t[v];
    }
    int tm = (tl + tr) / 2;
    return sum(v*2, tl, tm, l, min(r, tm))
           + sum(v*2+1, tm+1, tr, max(l, tm+1)
              , r);
}

void update(int v, int tl, int tr, int pos,
   int new_val) {
    if (tl == tr) {
        t[v] = new_val;
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos,
               new_val);
        t[v] = t[v*2] + t[v*2+1];
    }
}

// advance version of segment tree
pair<int, int> t[4*MAXN];

pair<int, int> combine(pair<int, int> a, pair<
   int, int> b) {
    if (a.first > b.first)
        return a;
    if (b.first > a.first)
        return b;
    return make_pair(a.first, a.second + b.
       second);
}

void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = make_pair(a[tl], 1);
    } else {
        int tm = (tl + tr) / 2;
```

```
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = combine(t[v*2], t[v*2+1]);
    }
}

pair<int, int> get_max(int v, int tl, int tr,
   int l, int r) {
    if (l > r)
        return make_pair(-INF, 0);
    if (l == tl && r == tr)
        return t[v];
    int tm = (tl + tr) / 2;
    return combine(get_max(v*2, tl, tm, l, min
       (r, tm)),
                   get_max(v*2+1, tm+1, tr,
                      max(l, tm+1), r));
}

void update(int v, int tl, int tr, int pos,
   int new_val) {
    if (tl == tr) {
        t[v] = make_pair(new_val, 1);
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos,
               new_val);
        t[v] = combine(t[v*2], t[v*2+1]);
    }
}

//int find_kth(int v, int tl, int tr, int k) {
    if (k > t[v])
        return -1;
    if (tl == tr)
        return tl;
    int tm = (tl + tr) / 2;
    if (t[v*2] >= k)
        return find_kth(v*2, tl, tm, k);
    else
        return find_kth(v*2+1, tm+1, tr, k - t
           [v*2]);
}
//
struct data {
    int sum, pref, suff, ans;
};

data combine(data l, data r) {
    data res;
    res.sum = l.sum + r.sum;
    res.pref = max(l.pref, l.sum + r.pref);
    res.suff = max(r.suff, r.sum + l.suff);
    res.ans = max(max(l.ans, r.ans), l.suff +
       r.pref);
    return res;
}
//
data make_data(int val) {
    data res;
    res.sum = val;
    res.pref = res.suff = res.ans = max(0, val
       );
    return res;
}

void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = make_data(a[tl]);
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = combine(t[v*2], t[v*2+1]);
    }
}

void update(int v, int tl, int tr, int pos,
   int new_val) {
    if (tl == tr) {
        t[v] = make_data(new_val);
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos,
               new_val);
        t[v] = combine(t[v*2], t[v*2+1]);
    }
data query(int v, int tl, int tr, int l, int r
   ) {
    if (l > r)
```

```
        return make_data(0);
    if (l == tl && r == tr)
        return t[v];
    int tm = (tl + tr) / 2;
    return combine(query(v*2, tl, tm, l, min(r
       , tm)),
                   query(v*2+1, tm+1, tr, max(
                      l, tm+1), r));
}
```

7.7 seg tree

```
vector<int>tree(400020),arr(100005);
int n,k;
void build(int node, int start, int end){//
   1,1,n
    if(start == end)
        tree[node] = A[start];
    else{
        int mid = (start + end) / 2;
        build(2*node, start, mid);
        build(2*node+1, mid+1, end);
        tree[node] = tree[2*node] + tree[2*
           node+1];
    }
}
void update(int node, int start, int end, int
   idx, int val){//1,1,n,i,val          i is 1
   based
    if(start == end){
        arr[idx] += val;
        tree[node] += val;
    }
    else{
        int mid = (start + end) / 2;
        if(start <= idx && idx <= mid)
            update(2*node, start, mid, idx,
               val);
        else
            update(2*node+1, mid+1, end, idx,
               val);
        tree[node] = tree[2*node] + tree[2*
           node+1];
    }
}
int query(int node, int start, int end, int l,
    int r){//1,1,n,l,r            l,r is 1
   based
    if(r < start || end < l)
        return 0;
    if(l <= start && end <= r)
        return tree[node];
    int mid = (start + end) / 2;
    int p1 = query(2*node, start, mid, l, r);
    int p2 = query(2*node+1, mid+1, end, l, r)
       ;
    return (p1 + p2);
}
```

7.8 sqrt decomp

```
// input data
int n;
vector<int> a (n);
// preprocessing
int len = (int) sqrt (n + .0) + 1; // size of
   the block and the number of blocks
vector<int> b (len);
for (int i=0; i<n; ++i)
    b[i / len] += a[i];
// answering the queries
for (;;) {
    int l, r;
  // read input data for the next query
    int sum = 0;
    for (int i=l; i<=r; )
        if (i % len == 0 && i + len - 1 <= r)
           {// if the whole block starting at i
            belongs to [l; r]
            sum += b[i / len];
            i += len;
        }
        else {
            sum += a[i];
            ++i;
        }
}
int sum = 0;
int c_l = l / len,   c_r = r / len;
if (c_l == c_r)
    for (int i=l; i<=r; ++i)
        sum += a[i];
else {
```

```
for (int i=l, end=(c_l+1)*len-1; i<=end;
   ++i)
    sum += a[i];
for (int i=c_l+1; i<=c_r-1; ++i)
    sum += b[i];
    for (int i=c_r*len; i<=r; ++i)
        sum += a[i];
}
```

Stirling Numbers of the second kind
S(k)[n] number of partitions of [n] into k non-empty parts
S(0)[n]=0 S(0)[n]=0 S(1)[n]=1 S(2)[n]=2^(n-1)-1 S(n)[n]=1
S(k)[n]=k*S(k)[n-1] + S(k-1)[n-1]

Partition function, denoted by p(k)[n].
• p(0)[n] = 0 (for n ≥ 1): No positive number can be partitioned into zero numbers.
• p(n)[n] = 1: To write n as the sum of n positive numbers, there is exactly one choice:
n = 1 + 1 + . . . 1. .ntimes
p(k)[n] = p(k)[n − k] + p(k−1)[n − 1]

n balls	k boxes	≤ 1 per box	≥ 1 per box	arbitrary
U	L	C(k,n)	C(n-1,k-1)	C(n+k-1,k-1)
L	U	1	S(k)[n]	sum(i=1 to k) of S(i)[n]
L	L	C(k,n)*n!	S(k)[n]*k!	k^n
U	U	1	p(k)[n]	sum(i=1 to k) of p(i)[n]

Arrangements	Correspond to
U → L	Integer solutions of x 1 + . . . + x k = n.
L → U	Partitions of the set [n] into k parts.
L → L	Functions from [n] to [k]
U → U	Partitions of the number n into k non-negative integers.

Legendre Symbol:
x*x=a mod p
Let p be an odd prime and let a be an integer.
The Legendre symbol of a with respect to p is defined by
a/p={ 1 if a is quadratic residue modulo p and a!=0(mod p)
-1 if a is not quadratic residue 0 if a=0(mod p) } =a^(p-1)/2

CRT:
x=ai (mod ni)
x=aj (mod nj)
check for validity ai=aj (mod gcd(ni,nj))
ans is x=b (mod lcm(n1,n2,..))

Totient:
- If p is a prime number, then gcd(p,q)=1 for all $1 \le q < p$. Therefore we have:
$$\phi(p) = p - 1.$$
- If p is a prime number and $k \ge 1$, then there are exactly pk/p numbers between 1 and pk that are divisible by p. Which gives us:

```
cheatsheet/p05.pdf
cheatsheet/p09.pdf
```