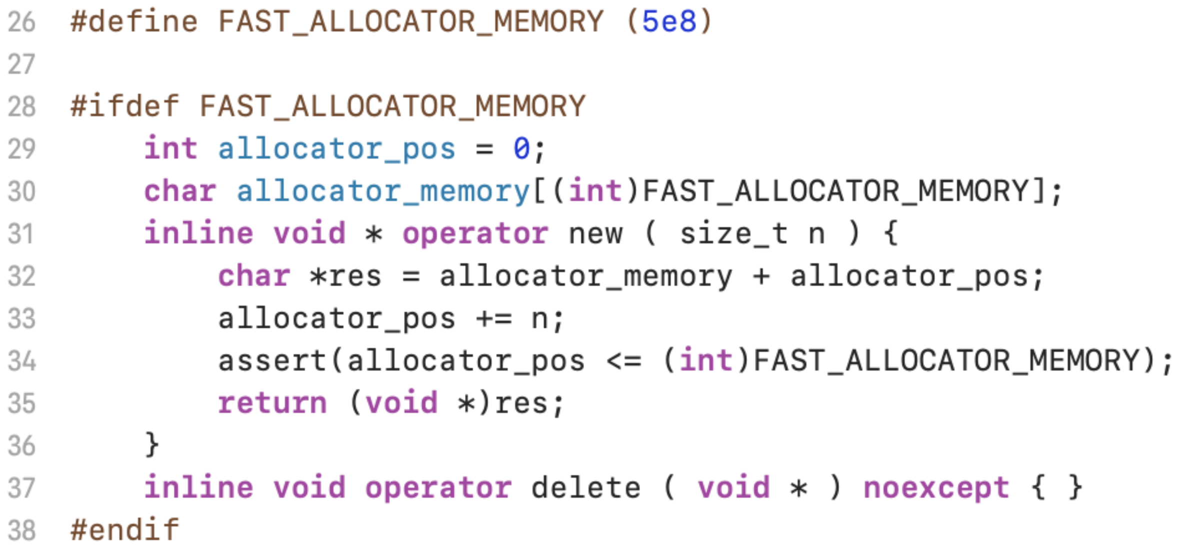
Team Notebook
CodeWinter Genius

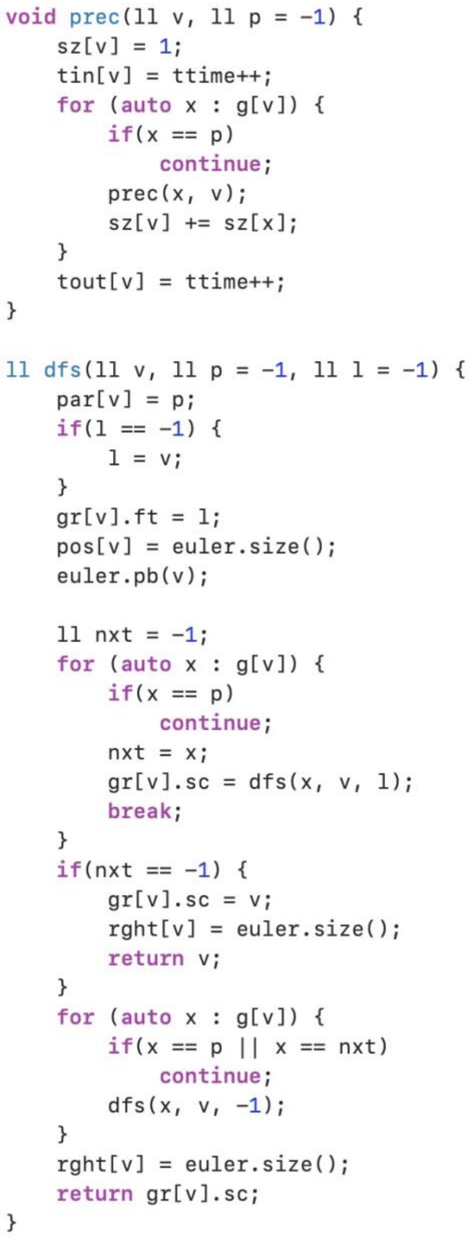
1) Source



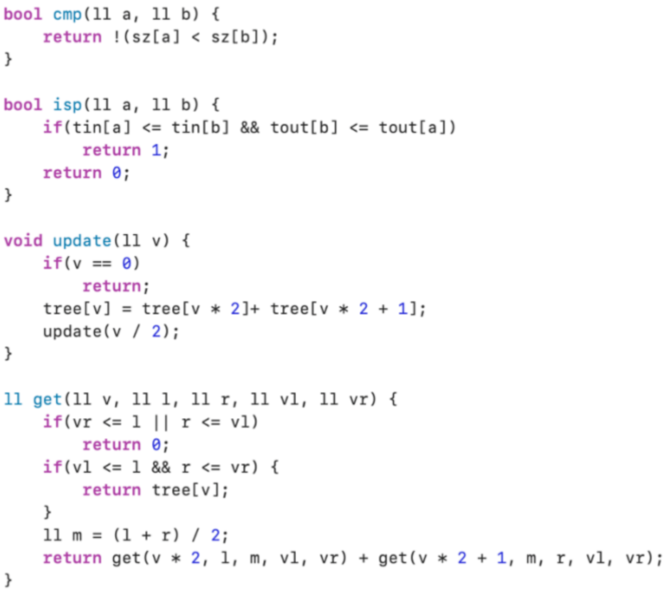
```
#define FAST_ALLOCATOR_MEMORY (5e8)

#ifdef FAST_ALLOCATOR_MEMORY
    int allocator_pos = 0;
    char allocator_memory[(int)FAST_ALLOCATOR_MEMORY];
    inline void * operator new ( size_t n ) {
        char *res = allocator_memory + allocator_pos;
        allocator_pos += n;
        assert(allocator_pos <= (int)FAST_ALLOCATOR_MEMORY);
        return (void *)res;
    }
    inline void operator delete ( void * ) noexcept { }
#endif
```

2) Деревья

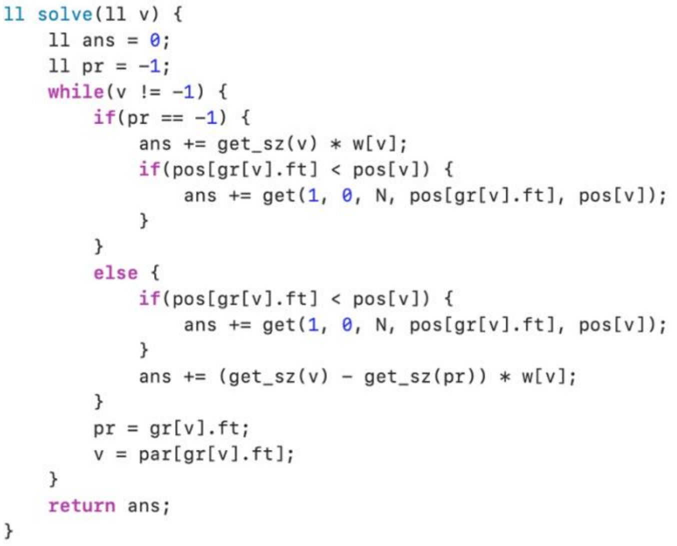
2.1) HLD



```
void prec(ll v, ll p = -1) {
    sz[v] = 1;
    tin[v] = ttime++;
    for (auto x : g[v]) {
        if(x == p)
            continue;
        prec(x, v);
        sz[v] += sz[x];
    }
    tout[v] = ttime++;
}

ll dfs(ll v, ll p = -1, ll l = -1) {
    par[v] = p;
    if(l == -1) {
        l = v;
    }
    gr[v].ft = l;
    pos[v] = euler.size();
    euler.pb(v);

    ll nxt = -1;
    for (auto x : g[v]) {
        if(x == p)
            continue;
        nxt = x;
        gr[v].sc = dfs(x, v, l);
        break;
    }
    if(nxt == -1) {
        gr[v].sc = v;
        rght[v] = euler.size();
        return v;
    }
    for (auto x : g[v]) {
        if(x == p || x == nxt)
            continue;
        dfs(x, v, -1);
    }
    rght[v] = euler.size();
    return gr[v].sc;
}
```



```
bool cmp(ll a, ll b) {
    return !(sz[a] < sz[b]);
}

bool isp(ll a, ll b) {
    if(tin[a] <= tin[b] && tout[b] <= tout[a])
        return 1;
    return 0;
}

void update(ll v) {
    if(v == 0)
        return;
    tree[v] = tree[v * 2]+ tree[v * 2 + 1];
    update(v / 2);
}

ll get(ll v, ll l, ll r, ll vl, ll vr) {
    if(vr <= l || r <= vl)
        return 0;
    if(vl <= l && r <= vr) {
        return tree[v];
    }
    ll m = (l + r) / 2;
    return get(v * 2, l, m, vl, vr) + get(v * 2 + 1, m, r, vl, vr);
}
```

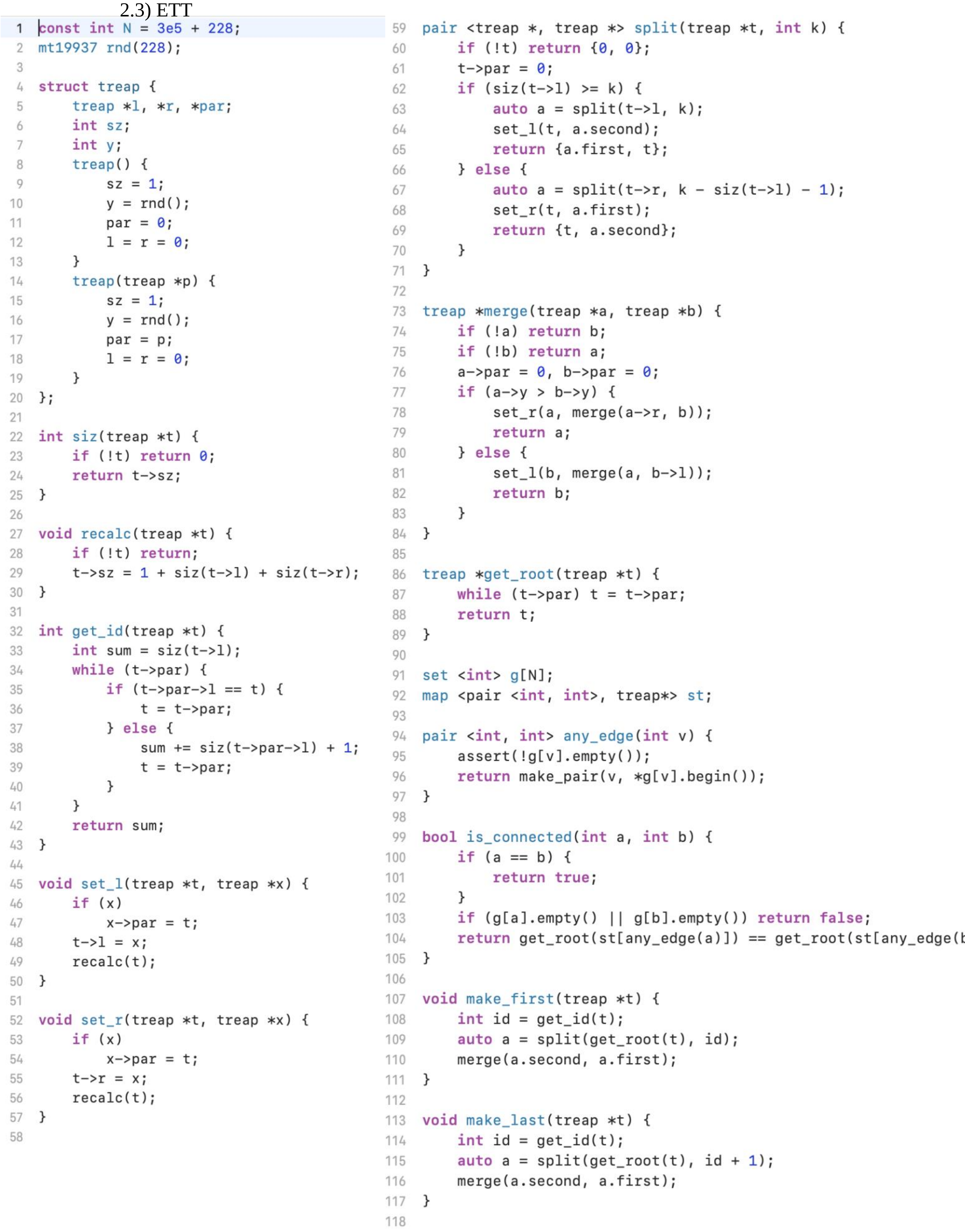


```
ll solve(ll v) {
    ll ans = 0;
    ll pr = -1;
    while(v != -1) {
        if(pr == -1) {
            ans += get_sz(v) * w[v];
            if(pos[gr[v].ft] < pos[v]) {
                ans += get(1, 0, N, pos[gr[v].ft], pos[v]);
            }
        }
        else {
            if(pos[gr[v].ft] < pos[v]) {
                ans += get(1, 0, N, pos[gr[v].ft], pos[v]);
            }
            ans += (get_sz(v) - get_sz(pr)) * w[v];
        }
        pr = gr[v].ft;
        v = par[gr[v].ft];
    }
    return ans;
}
```

2.2) Link-cut

```cpp
#include <iostream>
#include <cstdio>
#include <cassert>

using namespace std;

// BEGIN ALGO

const int MAXN = 110000;

typedef struct _node{
 _node *l, *r, *p, *pp;
 int size; bool rev;
 _node();
 explicit _node(nullptr_t){
  l = r = p = pp = this;
  size = rev = 0;
 }
 void push(){
  if (rev){
   l->rev ^= 1; r->rev ^= 1;
   rev = 0; swap(l,r);
  }
 }
 void update();
}* node;
node None = new _node(nullptr);
node v2n[MAXN];
_node::_node(){
 l = r = p = pp = None;
 size = 1; rev = false;
}
void _node::update(){
 size = (this != None) + l->size + r->size;
 l->p = r->p = this;
}
void rotate(node v){
 assert(v != None && v->p != None);
 assert(!v->rev); assert(!v->p->rev);
 node u = v->p;
 if (v == u->l)
  u->l = v->r, v->r = u;
 else
  u->r = v->l, v->l = u;
 swap(u->p,v->p); swap(v->pp,u->pp);
 if (v->p != None){
  assert(v->p->l == u || v->p->r == u);
  if (v->p->r == u) v->p->r = v;
  else v->p->l = v;
 }
 u->update(); v->update();
}
void bigRotate(node v){
 assert(v->p != None);
 v->p->p->push();
 v->p->push();
 v->push();
 if (v->p->p != None){
  if ((v->p->l == v) ^ (v->p->p->r == v->p))
   rotate(v->p);
  else
   rotate(v);
 }
 rotate(v);
}
inline void Splay(node v){
 while (v->p != None) bigRotate(v);
}
inline void splitAfter(node v){
 v->push();
 Splay(v);
 v->r->p = None;
 v->r->pp = v;
 v->r = None;
 v->update();
}
void expose(int x){
 node v = v2n[x];
 splitAfter(v);
 while (v->pp != None){
  assert(v->p == None);
  splitAfter(v->pp);
  assert(v->pp->r == None);
  assert(v->pp->p == None);
  assert(!v->pp->rev);
  v->pp->r = v;
  v->pp->update();
  v = v->pp;
  v->r->pp = None;
 }
 assert(v->p == None);
 Splay(v2n[x]);
}
inline void makeRoot(int x){
 expose(x);
 assert(v2n[x]->p == None);
 assert(v2n[x]->pp == None);
 assert(v2n[x]->r == None);
 v2n[x]->rev ^= 1;
}
inline void link(int x,int y){
 makeRoot(x); v2n[x]->pp = v2n[y];
}
inline void cut(int x,int y){
 expose(x);
 Splay(v2n[y]);
 if (v2n[y]->pp != v2n[x]){
  swap(x,y);
  expose(x);
  Splay(v2n[y]);
  assert(v2n[y]->pp == v2n[x]);
 }
 v2n[y]->pp = None;
}
inline int get(int x,int y){
 if (x == y) return 0;
 makeRoot(x);
 expose(y); expose(x);
 Splay(v2n[y]);
 if (v2n[y]->pp != v2n[x]) return -1;
 return v2n[y]->size;
}
// END ALGO

_node mem[MAXN];


int main(){
 freopen("linkcut.in","r",stdin);
 freopen("linkcut.out","w",stdout);

 int n,m;
 scanf("%d %d",&n,&m);

 for (int i = 0; i < n; i++)
  v2n[i] = &mem[i];

 for (int i = 0; i < m; i++){
  int a,b;
  if (scanf(" link %d %d",&a,&b) == 2)
   link(a-1,b-1);
  else if (scanf(" cut %d %d",&a,&b) == 2)
   cut(a-1,b-1);
  else if (scanf(" get %d %d",&a,&b) == 2)
   printf("%d\n",get(a-1,b-1));
  else
   assert(false);
```

2.3) ETT



```
const int N = 3e5 + 228;
mt19937 rnd(228);

struct treap {
    treap *l, *r, *par;
    int sz;
    int y;
    treap() {
        sz = 1;
        y = rnd();
        par = 0;
        l = r = 0;
    }
    treap(treap *p) {
        sz = 1;
        y = rnd();
        par = p;
        l = r = 0;
    }
};

int siz(treap *t) {
    if (!t) return 0;
    return t->sz;
}

void recalc(treap *t) {
    if (!t) return;
    t->sz = 1 + siz(t->l) + siz(t->r);
}

int get_id(treap *t) {
    int sum = siz(t->l);
    while (t->par) {
        if (t->par->l == t) {
            t = t->par;
        } else {
            sum += siz(t->par->l) + 1;
            t = t->par;
        }
    }
    return sum;
}

void set_l(treap *t, treap *x) {
    if (x)
        x->par = t;
    t->l = x;
    recalc(t);
}

void set_r(treap *t, treap *x) {
    if (x)
        x->par = t;
    t->r = x;
    recalc(t);
}

pair <treap *, treap *> split(treap *t, int k) {
    if (!t) return {0, 0};
    t->par = 0;
    if (siz(t->l) >= k) {
        auto a = split(t->l, k);
        set_l(t, a.second);
        return {a.first, t};
    } else {
        auto a = split(t->r, k - siz(t->l) - 1);
        set_r(t, a.first);
        return {t, a.second};
    }
}

treap *merge(treap *a, treap *b) {
    if (!a) return b;
    if (!b) return a;
    a->par = 0, b->par = 0;
    if (a->y > b->y) {
        set_r(a, merge(a->r, b));
        return a;
    } else {
        set_l(b, merge(a, b->l));
        return b;
    }
}

treap *get_root(treap *t) {
    while (t->par) t = t->par;
    return t;
}

set <int> g[N];
map <pair <int, int>, treap*> st;

pair <int, int> any_edge(int v) {
    assert(!g[v].empty());
    return make_pair(v, *g[v].begin());
}

bool is_connected(int a, int b) {
    if (a == b) {
        return true;
    }
    if (g[a].empty() || g[b].empty()) return false;
    return get_root(st[any_edge(a)]) == get_root(st[any_edge(b
}

void make_first(treap *t) {
    int id = get_id(t);
    auto a = split(get_root(t), id);
    merge(a.second, a.first);
}

void make_last(treap *t) {
    int id = get_id(t);
    auto a = split(get_root(t), id + 1);
    merge(a.second, a.first);
}

```

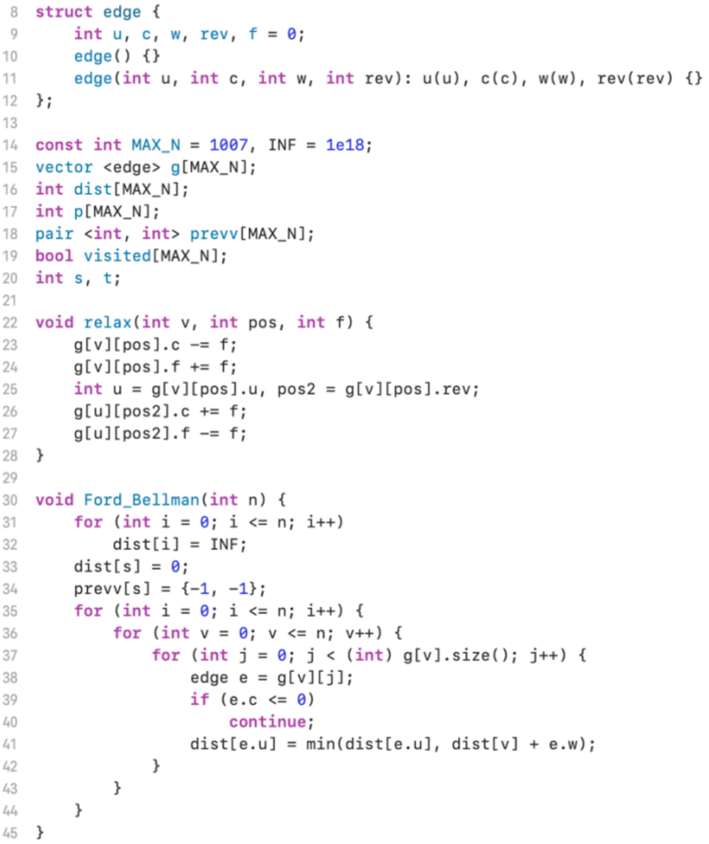
```
void link(int u, int v) {

    st[{u, v}] = new treap();
    st[{v, u}] = new treap();
    if (g[v].empty()) swap(u, v);
    if (g[u].empty()) {
        if (g[v].empty()) {
            merge(st[{u, v}], st[{v, u}]);
        } else {
            auto x = any_edge(v);
            make_first(st[x]);
            merge(st[{u, v}], get_root(st[x]));
            merge(get_root(st[x]), st[{v, u}]);
        }
    } else {
        pair <int, int> x, y;
        {
            x = any_edge(u);
            swap(x.first, x.second);
            make_last(st[x]);
        }
        {
            y = any_edge(v);
            make_first(st[y]);
        }
        merge(get_root(st[x]), st[{u, v}]);
        merge(get_root(st[x]), get_root(st[y]));
        merge(get_root(st[x]), st[{v, u}]);
    }
    g[u].insert(v);
    g[v].insert(u);
}

void cut(int u, int v) {
    g[u].erase(v);
    g[v].erase(u);
    auto x = make_pair(u, v);
    make_first(st[x]);
    auto a = split(get_root(st[x]), 1);
    auto y = make_pair(v, u);
    int ret = get_id(st[y]);
    auto b = split(a.second, ret);
    split(b.second, 1);
    delete st[x];
    delete st[y];
    st[x] = st[y] = 0;
}
```

3) Графы

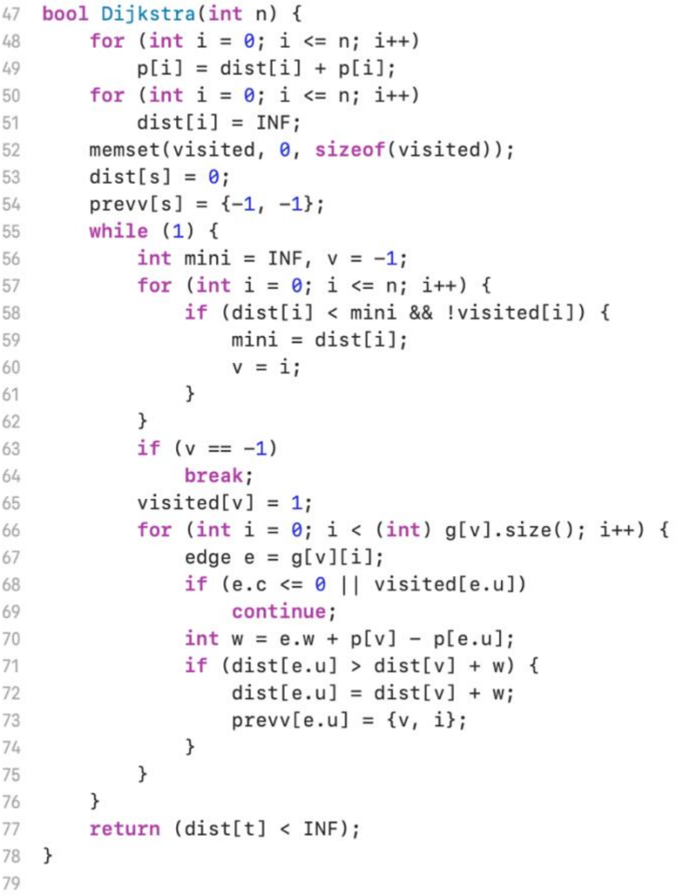
3.1) Нахождение отрицательного цикла

```
struct edge {
        int a, b, cost;
};

int n, m;
vector<edge> e;
const int INF = 1000000000;

void solve() {
        vector<int> d (n);
        vector<int> p (n, -1);
        int x;
        for (int i=0; i<n; ++i) {
                x = -1;
                for (int j=0; j<m; ++j)
                        if (d[e[j].b] > d[e[j].a] + e[j].cost) {
                                d[e[j].b] = max (-INF, d[e[j].a] +
e[j].cost);
                                p[e[j].b] = e[j].a;
                                x = e[j].b;
                        }
        }

        if (x == -1)
                cout << "No negative cycle found.";
        else {
                int y = x;
                for (int i=0; i<n; ++i)
                        y = p[y];

                vector<int> path;
                for (int cur=y; ; cur=p[cur]) {
                        path.push_back (cur);
                        if (cur == y && path.size() > 1)  break;
                }
                reverse (path.begin(), path.end());

                cout << "Negative cycle: ";
                for (size_t i=0; i<path.size(); ++i)
                        cout << path[i] << ' ';
        }
}
```

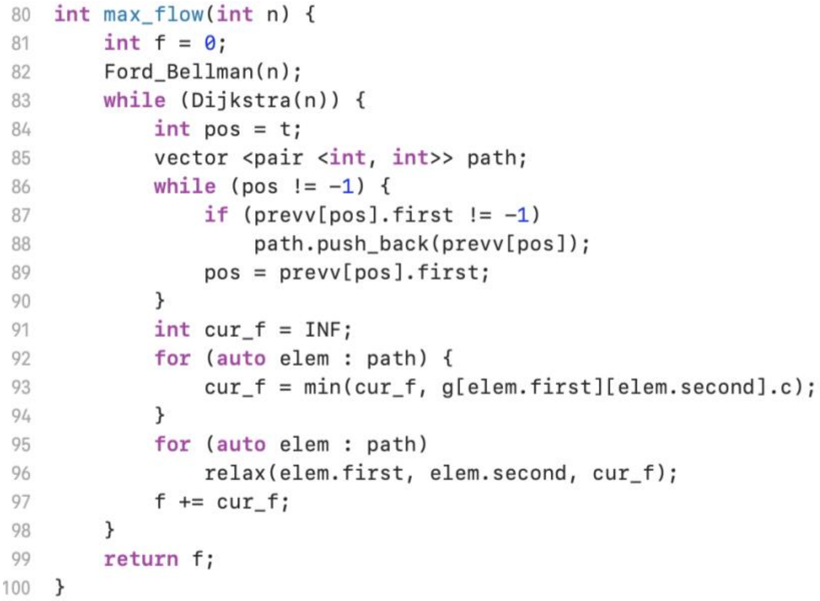
3.2) mincost maxflow



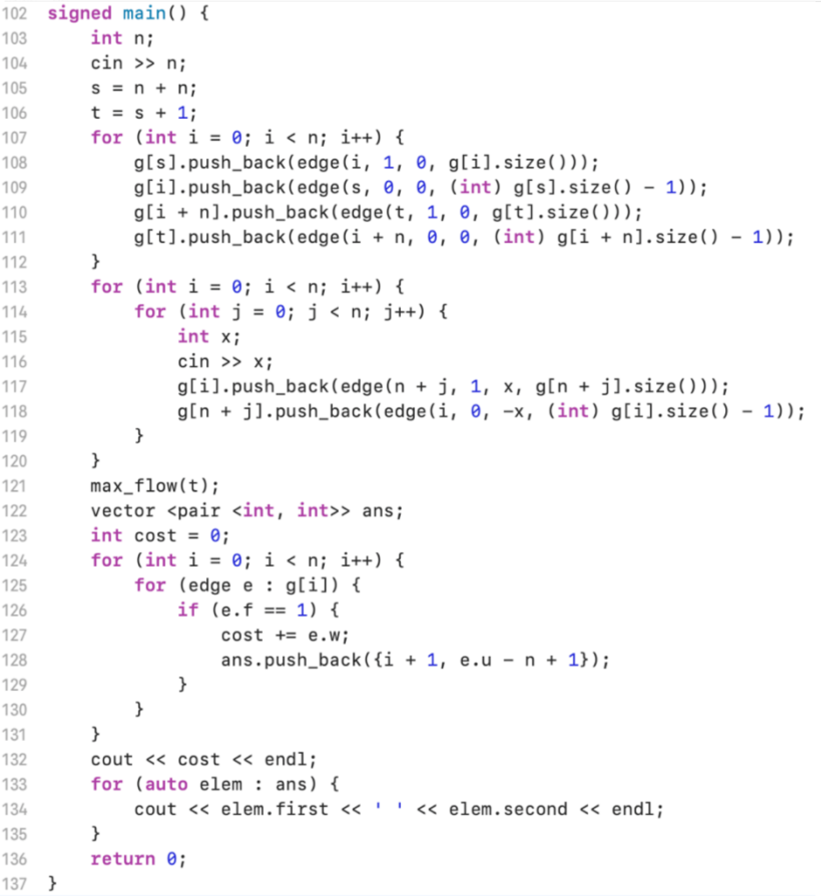
```
struct edge {
    int u, c, w, rev, f = 0;
    edge() {}
    edge(int u, int c, int w, int rev): u(u), c(c), w(w), rev(rev) {}
};

const int MAX_N = 1007, INF = 1e18;
vector <edge> g[MAX_N];
int dist[MAX_N];
int p[MAX_N];
pair <int, int> prevv[MAX_N];
bool visited[MAX_N];
int s, t;

void relax(int v, int pos, int f) {
    g[v][pos].c -= f;
    g[v][pos].f += f;
    int u = g[v][pos].u, pos2 = g[v][pos].rev;
    g[u][pos2].c += f;
    g[u][pos2].f -= f;
}

void Ford_Bellman(int n) {
    for (int i = 0; i <= n; i++)
        dist[i] = INF;
    dist[s] = 0;
    prevv[s] = {-1, -1};
    for (int i = 0; i <= n; i++) {
        for (int v = 0; v <= n; v++) {
            for (int j = 0; j < (int) g[v].size(); j++) {
                edge e = g[v][j];
                if (e.c <= 0)
                    continue;
                dist[e.u] = min(dist[e.u], dist[v] + e.w);
            }
        }
    }
}
```



```
bool Dijkstra(int n) {
    for (int i = 0; i <= n; i++)
        p[i] = dist[i] + p[i];
    for (int i = 0; i <= n; i++)
        dist[i] = INF;
    memset(visited, 0, sizeof(visited));
    dist[s] = 0;
    prevv[s] = {-1, -1};
    while (1) {
        int mini = INF, v = -1;
        for (int i = 0; i <= n; i++) {
            if (dist[i] < mini && !visited[i]) {
                mini = dist[i];
                v = i;
            }
        }
        if (v == -1)
            break;
        visited[v] = 1;
        for (int i = 0; i < (int) g[v].size(); i++) {
            edge e = g[v][i];
            if (e.c <= 0 || visited[e.u])
                continue;
            int w = e.w + p[v] - p[e.u];
            if (dist[e.u] > dist[v] + w) {
                dist[e.u] = dist[v] + w;
                prevv[e.u] = {v, i};
            }
        }
    }
    return (dist[t] < INF);
}

```



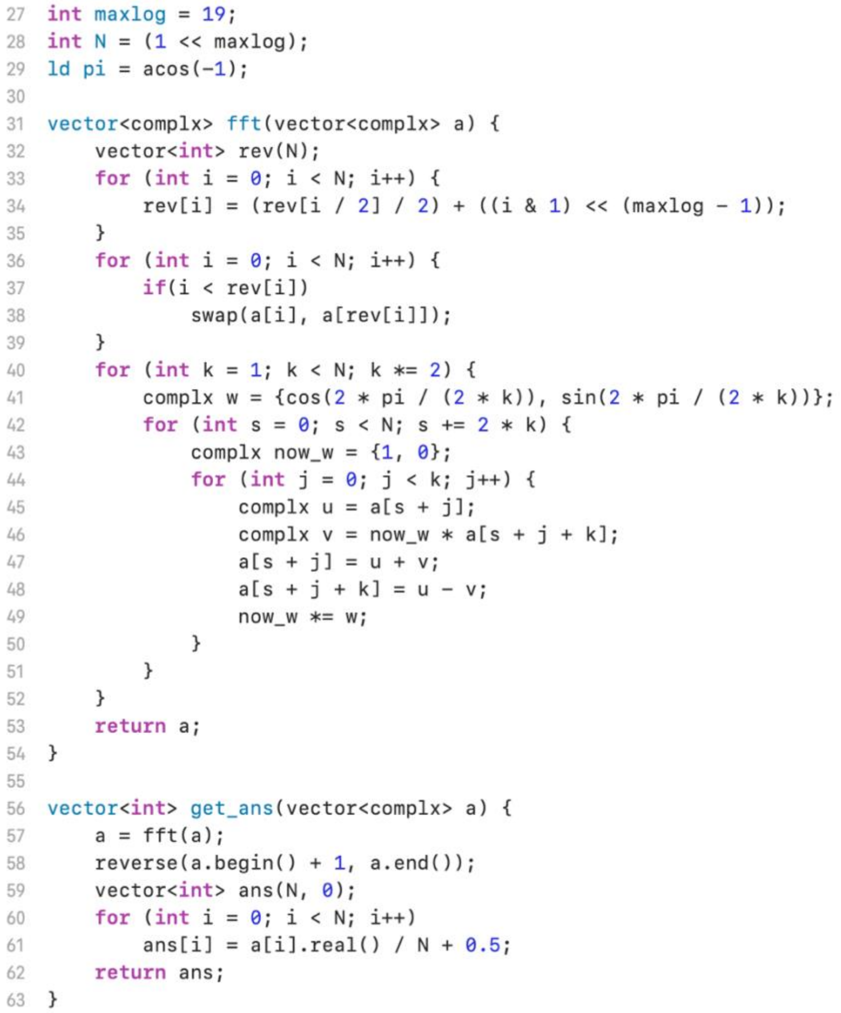
```
int max_flow(int n) {
    int f = 0;
    Ford_Bellman(n);
    while (Dijkstra(n)) {
        int pos = t;
        vector <pair <int, int>> path;
        while (pos != -1) {
            if (prevv[pos].first != -1)
                path.push_back(prevv[pos]);
            pos = prevv[pos].first;
        }
        int cur_f = INF;
        for (auto elem : path) {
            cur_f = min(cur_f, g[elem.first][elem.second].c);
        }
        for (auto elem : path)
            relax(elem.first, elem.second, cur_f);
        f += cur_f;
    }
    return f;
}
```



```
signed main() {
    int n;
    cin >> n;
    s = n + n;
    t = s + 1;
    for (int i = 0; i < n; i++) {
        g[s].push_back(edge(i, 1, 0, g[i].size()));
        g[i].push_back(edge(s, 0, 0, (int) g[s].size() - 1));
        g[i + n].push_back(edge(t, 1, 0, g[t].size()));
        g[t].push_back(edge(i + n, 0, 0, (int) g[i + n].size() - 1));
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            int x;
            cin >> x;
            g[i].push_back(edge(n + j, 1, x, g[n + j].size()));
            g[n + j].push_back(edge(i, 0, -x, (int) g[i].size() - 1));
        }
    }
    max_flow(t);
    vector <pair <int, int>> ans;
    int cost = 0;
    for (int i = 0; i < n; i++) {
        for (edge e : g[i]) {
            if (e.f == 1) {
                cost += e.w;
                ans.push_back({i + 1, e.u - n + 1});
            }
        }
    }
    cout << cost << endl;
    for (auto elem : ans) {
        cout << elem.first << ' ' << elem.second << endl;
    }
    return 0;
}
```

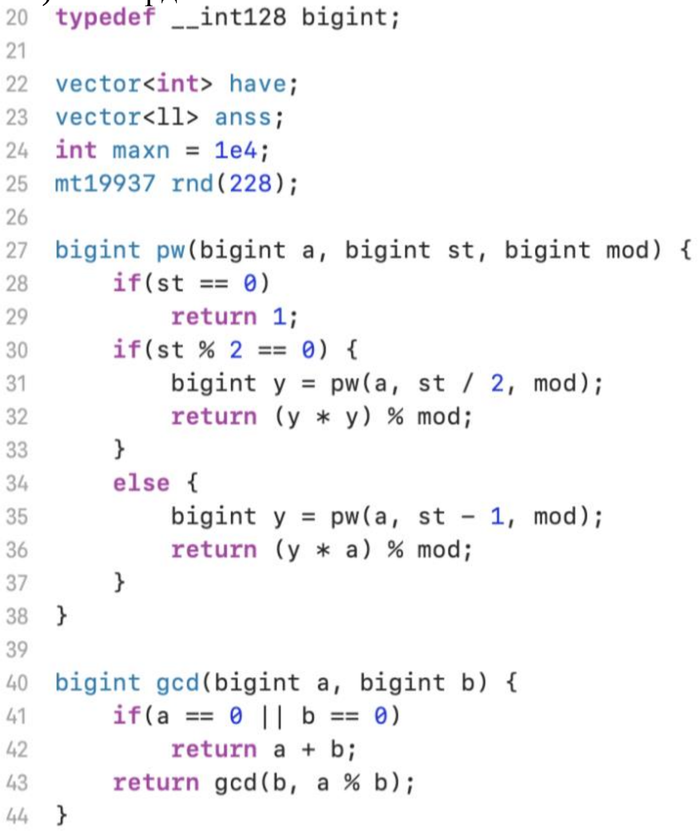
4) ТЧ

4.1) FFT

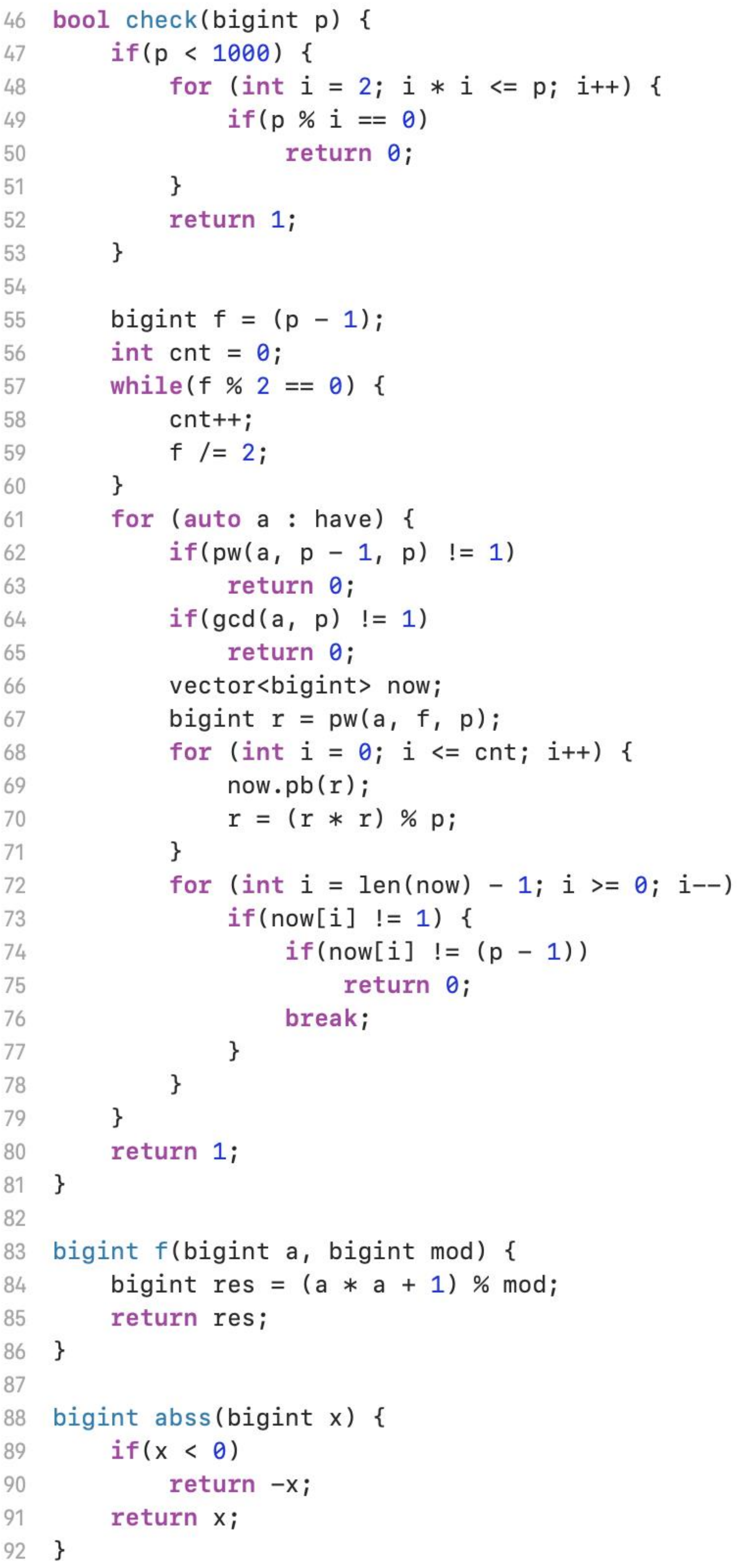


```
int maxlog = 19;
int N = (1 << maxlog);
ld pi = acos(-1);

vector<complx> fft(vector<complx> a) {
    vector<int> rev(N);
    for (int i = 0; i < N; i++) {
        rev[i] = (rev[i / 2] / 2) + ((i & 1) << (maxlog - 1));
    }
    for (int i = 0; i < N; i++) {
        if(i < rev[i])
            swap(a[i], a[rev[i]]);
    }
    for (int k = 1; k < N; k *= 2) {
        complx w = {cos(2 * pi / (2 * k)), sin(2 * pi / (2 * k))};
        for (int s = 0; s < N; s += 2 * k) {
            complx now_w = {1, 0};
            for (int j = 0; j < k; j++) {
                complx u = a[s + j];
                complx v = now_w * a[s + j + k];
                a[s + j] = u + v;
                a[s + j + k] = u - v;
                now_w *= w;
            }
        }
    }
    return a;
}

vector<int> get_ans(vector<complx> a) {
    a = fft(a);
    reverse(a.begin() + 1, a.end());
    vector<int> ans(N, 0);
    for (int i = 0; i < N; i++)
        ans[i] = a[i].real() / N + 0.5;
    return ans;
}
```

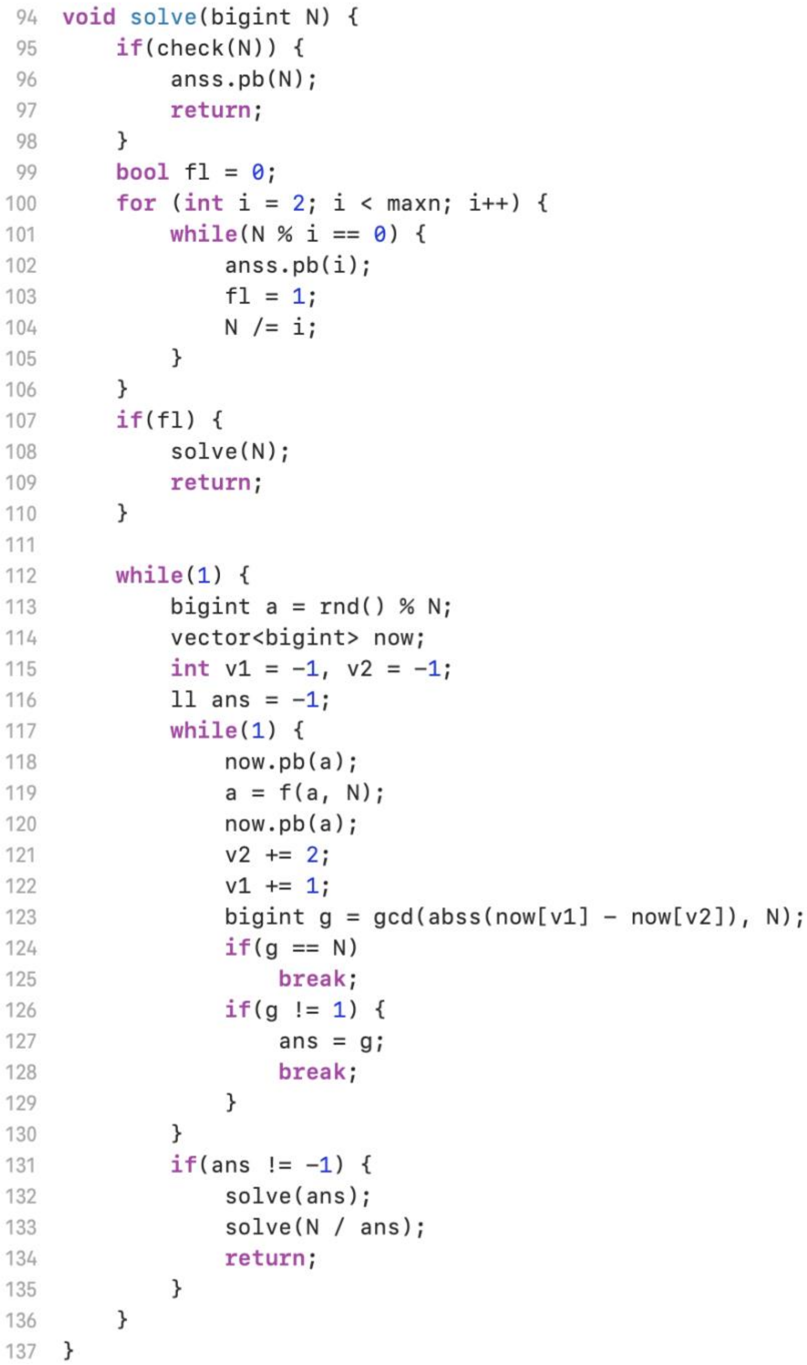
4.2) Поллард



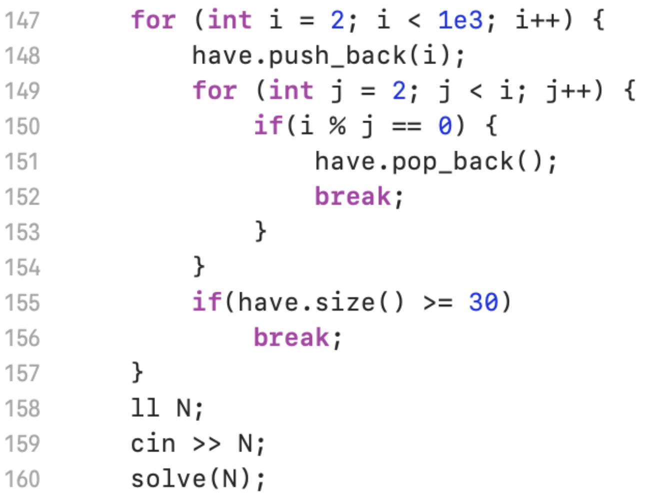
```
typedef __int128 bigint;

vector<int> have;
vector<ll> anss;
int maxn = 1e4;
mt19937 rnd(228);

bigint pw(bigint a, bigint st, bigint mod) {
    if(st == 0)
        return 1;
    if(st % 2 == 0) {
        bigint y = pw(a, st / 2, mod);
        return (y * y) % mod;
    }
    else {
        bigint y = pw(a, st - 1, mod);
        return (y * a) % mod;
    }
}

bigint gcd(bigint a, bigint b) {
    if(a == 0 || b == 0)
        return a + b;
    return gcd(b, a % b);
}
```



```
bool check(bigint p) {
    if(p < 1000) {
        for (int i = 2; i * i <= p; i++) {
            if(p % i == 0)
                return 0;
        }
        return 1;
    }

    bigint f = (p - 1);
    int cnt = 0;
    while(f % 2 == 0) {
        cnt++;
        f /= 2;
    }
    for (auto a : have) {
        if(pw(a, p - 1, p) != 1)
            return 0;
        if(gcd(a, p) != 1)
            return 0;
        vector<bigint> now;
        bigint r = pw(a, f, p);
        for (int i = 0; i <= cnt; i++) {
            now.pb(r);
            r = (r * r) % p;
        }
        for (int i = len(now) - 1; i >= 0; i--)
            if(now[i] != 1) {
                if(now[i] != (p - 1))
                    return 0;
                break;
            }
        }
    }
    return 1;
}

bigint f(bigint a, bigint mod) {
    bigint res = (a * a + 1) % mod;
    return res;
}

bigint abss(bigint x) {
    if(x < 0)
        return -x;
    return x;
}
```

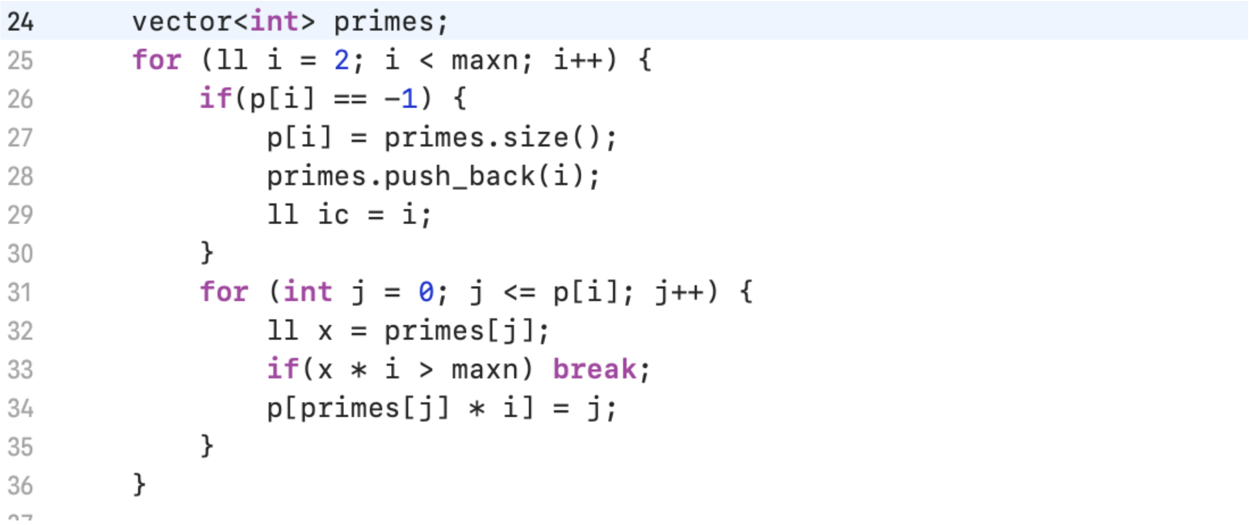


```
void solve(bigint N) {
    if(check(N)) {
        anss.pb(N);
        return;
    }
    bool fl = 0;
    for (int i = 2; i < maxn; i++) {
        while(N % i == 0) {
            anss.pb(i);
            fl = 1;
            N /= i;
        }
    }
    if(fl) {
        solve(N);
        return;
    }

    while(1) {
        bigint a = rnd() % N;
        vector<bigint> now;
        int v1 = -1, v2 = -1;
        ll ans = -1;
        while(1) {
            now.pb(a);
            a = f(a, N);
            now.pb(a);
            v2 += 2;
            v1 += 1;
            bigint g = gcd(abss(now[v1] - now[v2]), N);
            if(g == N)
                break;
            if(g != 1) {
                ans = g;
                break;
            }
        }
        if(ans != -1) {
            solve(ans);
            solve(N / ans);
            return;
        }
    }
}
```



```
    for (int i = 2; i < 1e3; i++) {
        have.push_back(i);
        for (int j = 2; j < i; j++) {
            if(i % j == 0) {
                have.pop_back();
                break;
            }
        }
        if(have.size() >= 30)
            break;
    }
    ll N;
    cin >> N;
    solve(N);
```

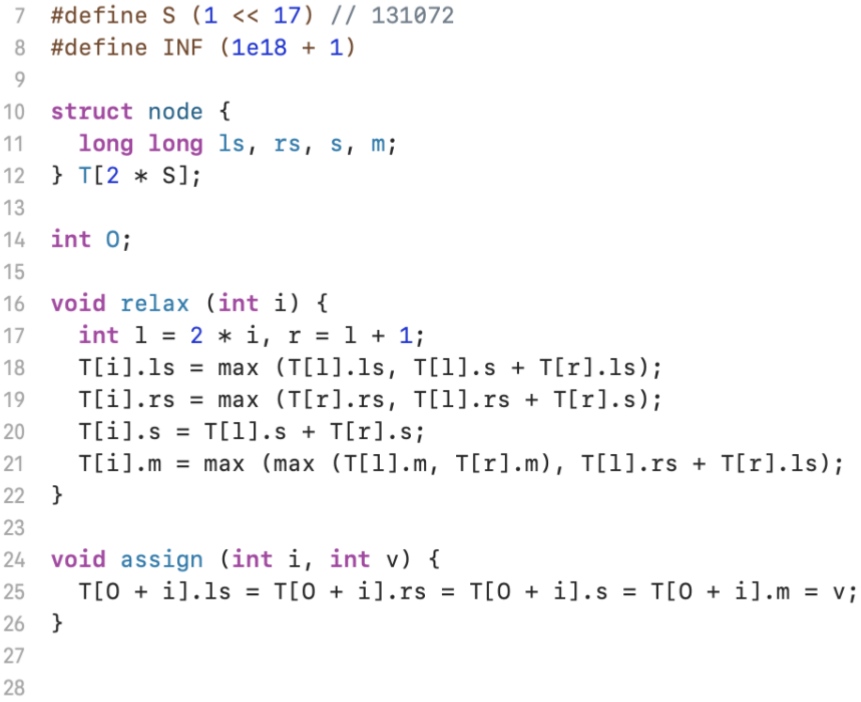
4.3) Решето Эратосфена за O(n)



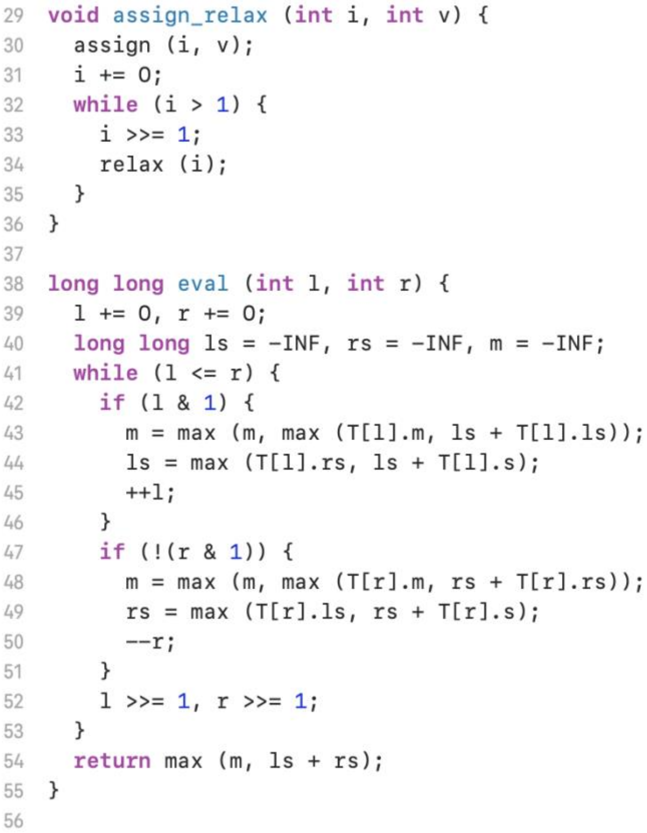
```
    vector<int> primes;
    for (ll i = 2; i < maxn; i++) {
        if(p[i] == -1) {
            p[i] = primes.size();
            primes.push_back(i);
            ll ic = i;
        }
        for (int j = 0; j <= p[i]; j++) {
            ll x = primes[j];
            if(x * i > maxn) break;
            p[primes[j] * i] = j;
        }
    }
```

5) Структуры данных

5.1) ДО снизу



```
#define S (1 << 17) // 131072
#define INF (1e18 + 1)

struct node {
  long long ls, rs, s, m;
} T[2 * S];

int O;

void relax (int i) {
  int l = 2 * i, r = l + 1;
  T[i].ls = max (T[l].ls, T[l].s + T[r].ls);
  T[i].rs = max (T[r].rs, T[l].rs + T[r].s);
  T[i].s = T[l].s + T[r].s;
  T[i].m = max (max (T[l].m, T[r].m), T[l].rs + T[r].ls);
}

void assign (int i, int v) {
  T[O + i].ls = T[O + i].rs = T[O + i].s = T[O + i].m = v;
}


```



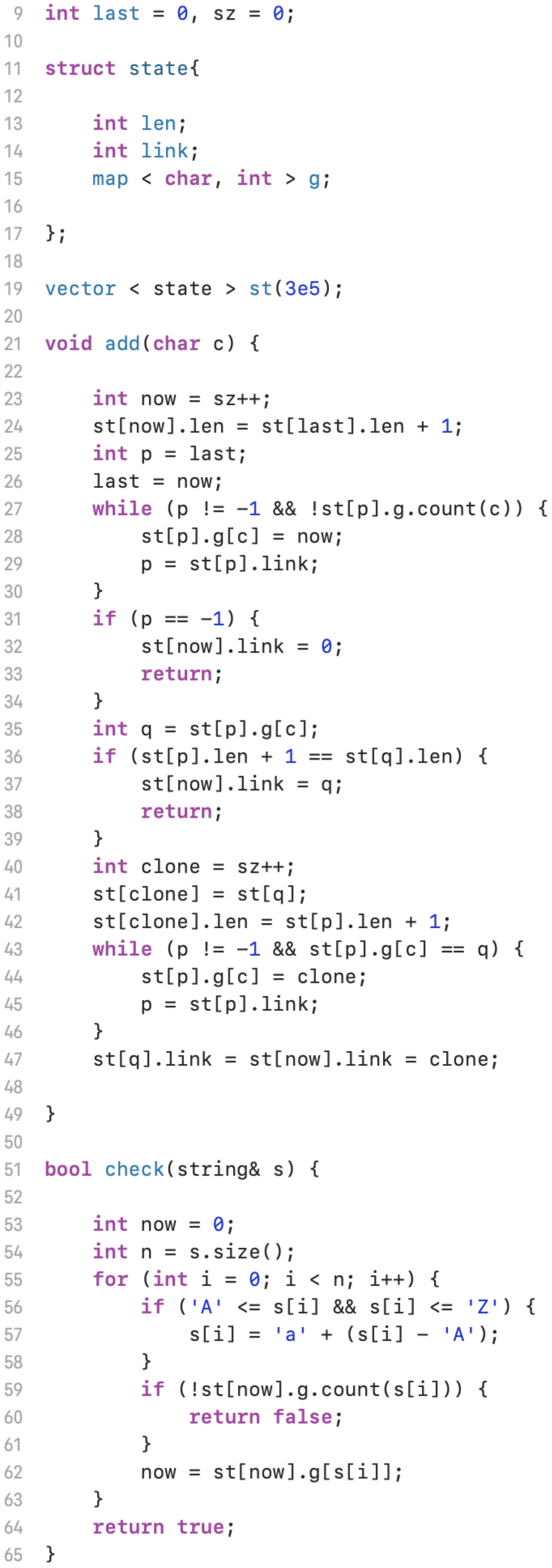
```
void assign_relax (int i, int v) {
  assign (i, v);
  i += O;
  while (i > 1) {
    i >>= 1;
    relax (i);
  }
}

long long eval (int l, int r) {
  l += O, r += O;
  long long ls = -INF, rs = -INF, m = -INF;
  while (l <= r) {
    if (l & 1) {
      m = max (m, max (T[l].m, ls + T[l].ls));
      ls = max (T[l].rs, ls + T[l].s);
      ++l;
    }
    if (!(r & 1)) {
      m = max (m, max (T[r].m, rs + T[r].rs));
      rs = max (T[r].ls, rs + T[r].s);
      --r;
    }
    l >>= 1, r >>= 1;
  }
  return max (m, ls + rs);
}

```

5.2) Персистентное ДД

```
struct treap
{
  int val, sz;
  treap *l, *r;
  treap(int val, treap *l, treap *r): val(val), sz(1), l(l), r(r)
  {
  }
  treap()
  {
  }
};

const int N = 10 * 10000 * 100 + 7;
const int MAX_SZ = N - 1e5;

treap node[N];
int ptr = 0;

treap *new_treap(int val, treap *l, treap *r)
{
  node[ptr].val = val;
  node[ptr].l = l;
  node[ptr].r = r;
  node[ptr].sz = 1;
  assert(ptr < N);
  return &node[ptr++];
}

int sz(treap *t)
{
  if (!t)
  {
    return 0;
  }
  else
  {
    return t->sz;
  }
}

void recalc(treap *t)
{
  if (!t)
  {
    return;
  }
  else
  {
    t->sz = sz(t->l) + 1 + sz(t->r);
  }
}

void zhfs(treap *t, vector <int> &a)
{
  if (!t)
  {
    return;
  }
  zhfs(t->l, a);
  a.push_back(t->val);
  zhfs(t->r, a);
}

pair <treap*, treap*> split(treap *t, int x)
{
  if (!t)
  {
    return {0, 0};
  }
  if (sz(t->l) >= x)
  {
    auto a = split(t->l, x);
    treap *l = a.first;
    treap *r = new_treap(t->val, a.second, t->r);
    recalc(r);
    return {l, r};
  }
  else
  {
    auto a = split(t->r, x - sz(t->l) - 1);
    treap *l = new_treap(t->val, t->l, a.first);
    treap *r = a.second;
    recalc(l);
    return {l, r};
  }
}

treap *merge(treap *a, treap *b)
{
  if (!a)
  {
    return b;
  }
  if (!b)
  {
    return a;
  }
  if (rnd() % (a->sz + b->sz) < a->sz)
  {
    treap *ret = new_treap(a->val, a->l, merge(a->r, b));
    recalc(ret);
    return ret;
  }
  else
  {
    treap *ret = new_treap(b->val, merge(a, b->l), b->r);
    recalc(ret);
    return ret;
  }
}

struct q
{
  int len, from, to;
  q(int len, int from, int to): len(len), from(from), to(to)
  {
  }
  q()
  {
  }
};

void print(treap *t)
{
  if (!t)
  {
    return;
  }
  print(t->l);
  printf("%d ", t->val);
  print(t->r);
}

treap *build(const vector <int> &a)
{
  if (a.empty())
  {
    return 0;
  }
  int mid = (int) a.size() / 2;
  vector <int> l, r;
  for (int i = 0; i < mid; i++) l.push_back(a[i]);
  for (int i = mid + 1; i < (int) a.size(); i++) r.push_back(a[i]);
  treap *go = new_treap(a[mid], build(l), build(r));
  recalc(go);
  return go;
}

treap *cur = 0, *rev = 0;

treap *get_segm(treap *t, int l, int r)
{
  l++, r++;
  auto a = split(t, r);
  auto b = split(a.first, l - 1);
  return b.second;
}

treap *to_copy(treap *t, int l, int r, treap *go)
{
  l++, r++;
  auto a = split(t, r);
  auto b = split(a.first, l - 1);
  return merge(b.first, merge(go, a.second));
}
```

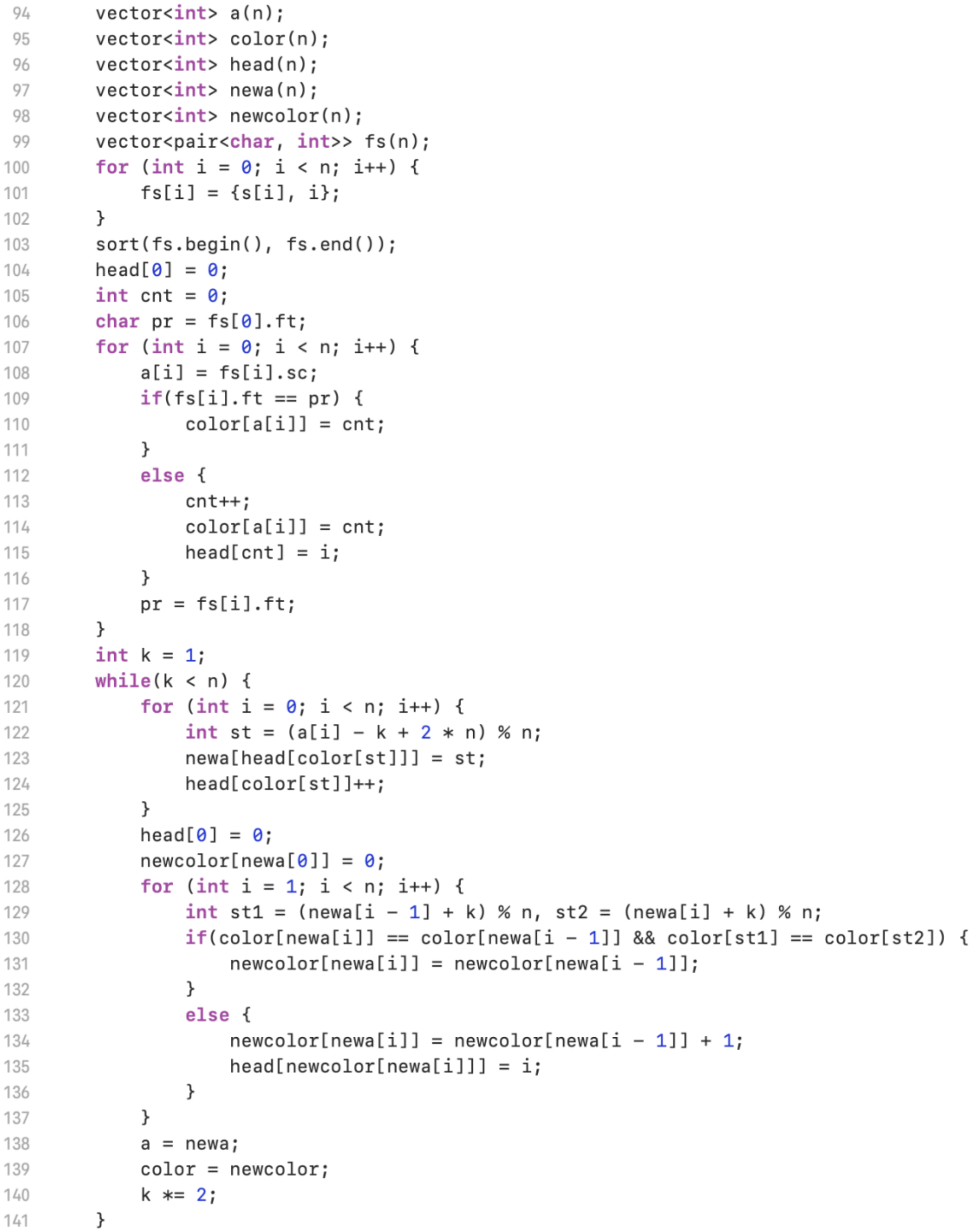
6) Строки

6.1) Суфф. автомат

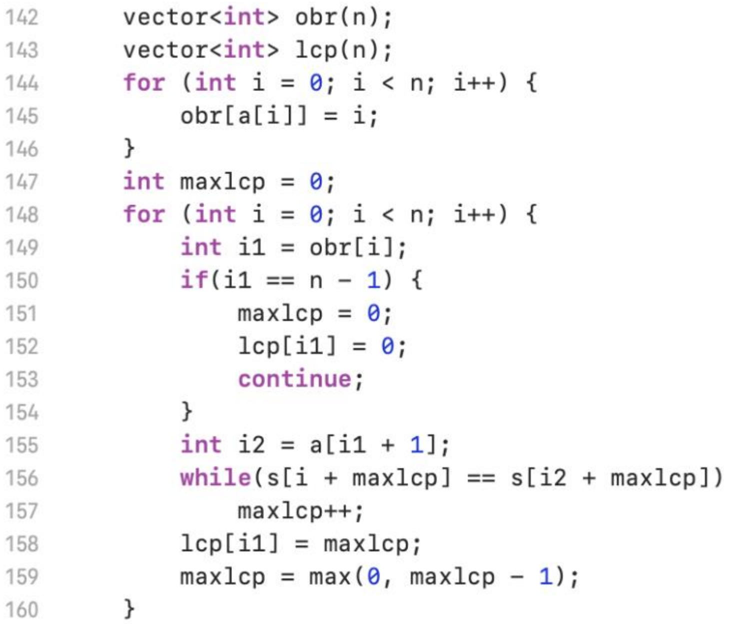


```cpp
int last = 0, sz = 0;

struct state{

    int len;
    int link;
    map < char, int > g;

};

vector < state > st(3e5);

void add(char c) {

    int now = sz++;
    st[now].len = st[last].len + 1;
    int p = last;
    last = now;
    while (p != -1 && !st[p].g.count(c)) {
        st[p].g[c] = now;
        p = st[p].link;
    }
    if (p == -1) {
        st[now].link = 0;
        return;
    }
    int q = st[p].g[c];
    if (st[p].len + 1 == st[q].len) {
        st[now].link = q;
        return;
    }
    int clone = sz++;
    st[clone] = st[q];
    st[clone].len = st[p].len + 1;
    while (p != -1 && st[p].g[c] == q) {
        st[p].g[c] = clone;
        p = st[p].link;
    }
    st[q].link = st[now].link = clone;

}

bool check(string& s) {

    int now = 0;
    int n = s.size();
    for (int i = 0; i < n; i++) {
        if ('A' <= s[i] && s[i] <= 'Z') {
            s[i] = 'a' + (s[i] - 'A');
        }
        if (!st[now].g.count(s[i])) {
            return false;
        }
        now = st[now].g[s[i]];
    }
    return true;
}
```

6.2) Суфф. масс + lcp



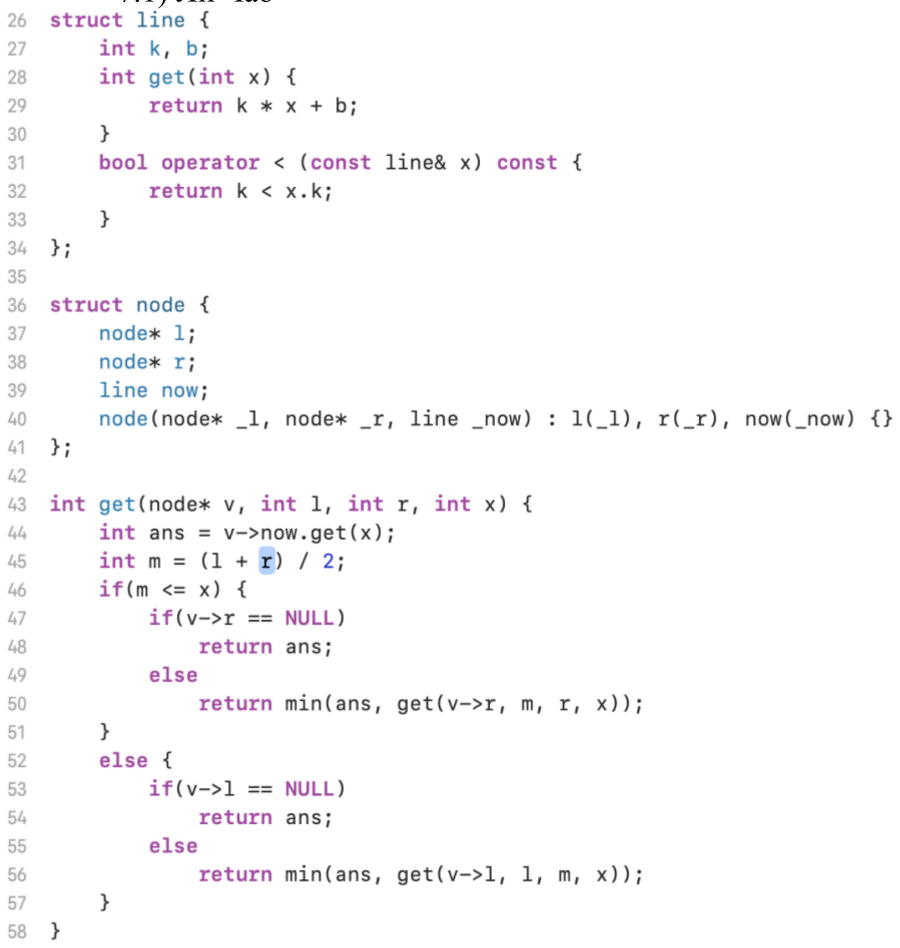
```cpp
    vector<int> a(n);
    vector<int> color(n);
    vector<int> head(n);
    vector<int> newa(n);
    vector<int> newcolor(n);
    vector<pair<char, int>> fs(n);
    for (int i = 0; i < n; i++) {
        fs[i] = {s[i], i};
    }
    sort(fs.begin(), fs.end());
    head[0] = 0;
    int cnt = 0;
    char pr = fs[0].ft;
    for (int i = 0; i < n; i++) {
        a[i] = fs[i].sc;
        if(fs[i].ft == pr) {
            color[a[i]] = cnt;
        }
        else {
            cnt++;
            color[a[i]] = cnt;
            head[cnt] = i;
        }
        pr = fs[i].ft;
    }
    int k = 1;
    while(k < n) {
        for (int i = 0; i < n; i++) {
            int st = (a[i] - k + 2 * n) % n;
            newa[head[color[st]]] = st;
            head[color[st]]++;
        }
        head[0] = 0;
        newcolor[newa[0]] = 0;
        for (int i = 1; i < n; i++) {
            int st1 = (newa[i - 1] + k) % n, st2 = (newa[i] + k) % n;
            if(color[newa[i]] == color[newa[i - 1]] && color[st1] == color[st2]) {
                newcolor[newa[i]] = newcolor[newa[i - 1]];
            }
            else {
                newcolor[newa[i]] = newcolor[newa[i - 1]] + 1;
                head[newcolor[newa[i]]] = i;
            }
        }
        a = newa;
        color = newcolor;
        k *= 2;
    }
```



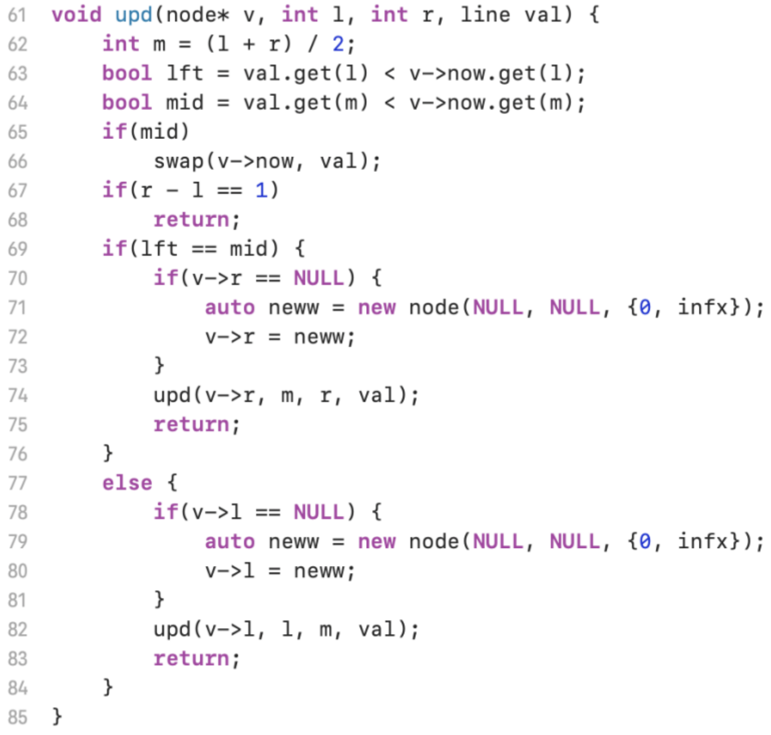
```cpp
    vector<int> obr(n);
    vector<int> lcp(n);
    for (int i = 0; i < n; i++) {
        obr[a[i]] = i;
    }
    int maxlcp = 0;
    for (int i = 0; i < n; i++) {
        int i1 = obr[i];
        if(i1 == n - 1) {
            maxlcp = 0;
            lcp[i1] = 0;
            continue;
        }
        int i2 = a[i1 + 1];
        while(s[i + maxlcp] == s[i2 + maxlcp])
            maxlcp++;
        lcp[i1] = maxlcp;
        maxlcp = max(0, maxlcp - 1);
    }
```

7) Оптимизация динамики

7.1) Ли Чао

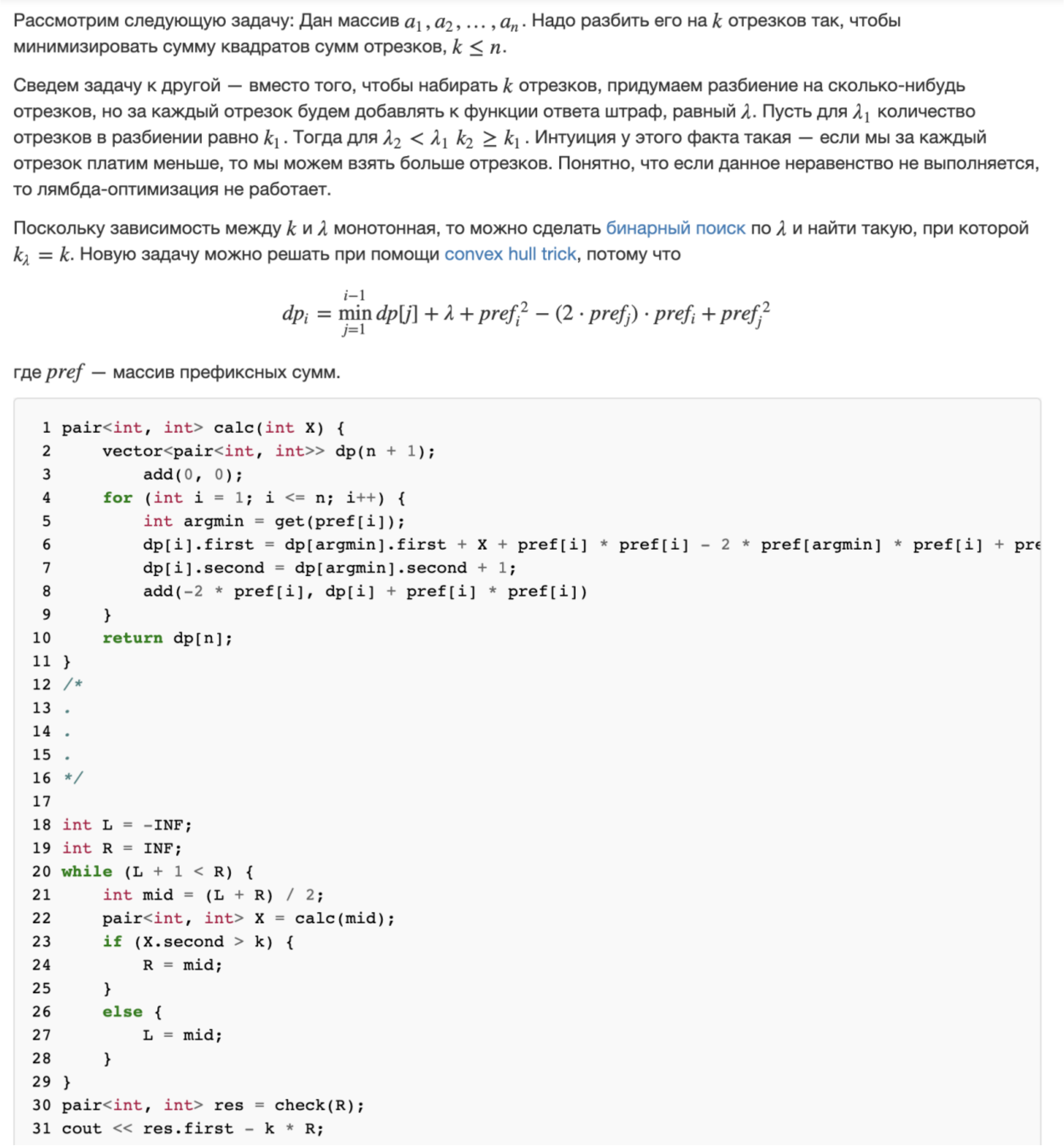


```cpp
struct line {
    int k, b;
    int get(int x) {
        return k * x + b;
    }
    bool operator < (const line& x) const {
        return k < x.k;
    }
};

struct node {
    node* l;
    node* r;
    line now;
    node(node* _l, node* _r, line _now) : l(_l), r(_r), now(_now) {}
};

int get(node* v, int l, int r, int x) {
    int ans = v->now.get(x);
    int m = (l + r) / 2;
    if(m <= x) {
        if(v->r == NULL)
            return ans;
        else
            return min(ans, get(v->r, m, r, x));
    }
    else {
        if(v->l == NULL)
            return ans;
        else
            return min(ans, get(v->l, l, m, x));
    }
}
```



```cpp
void upd(node* v, int l, int r, line val) {
    int m = (l + r) / 2;
    bool lft = val.get(l) < v->now.get(l);
    bool mid = val.get(m) < v->now.get(m);
    if(mid)
        swap(v->now, val);
    if(r - l == 1)
        return;
    if(lft == mid) {
        if(v->r == NULL) {
            auto neww = new node(NULL, NULL, {0, infx});
            v->r = neww;
        }
        upd(v->r, m, r, val);
        return;
    }
    else {
        if(v->l == NULL) {
            auto neww = new node(NULL, NULL, {0, infx});
            v->l = neww;
        }
        upd(v->l, l, m, val);
        return;
    }
}
```

7.2) Лямбда оптимизация



Рассмотрим следующую задачу: Дан массив $a_1, a_2, \dots, a_n$. Надо разбить его на k отрезков так, чтобы минимизировать сумму квадратов сумм отрезков, $k \le n$.

Сведем задачу к другой — вместо того, чтобы набирать k отрезков, придумаем разбиение на сколько-нибудь отрезков, но за каждый отрезок будем добавлять к функции ответа штраф, равный λ. Пусть для λ_1 количество отрезков в разбиении равно k_1. Тогда для $\lambda_2 < \lambda_1$ $k_2 \ge k_1$. Интуиция у этого факта такая — если мы за каждый отрезок платим меньше, то мы можем взять больше отрезков. Понятно, что если данное неравенство не выполняется, то лямбда-оптимизация не работает.

Поскольку зависимость между k и λ монотонная, то можно сделать бинарный поиск по λ и найти такую, при которой $k_\lambda = k$. Новую задачу можно решать при помощи convex hull trick, потому что

$$dp_i = \min_{j=1}^{i-1} dp[j] + \lambda + pref_i^2 - (2 \cdot pref_j) \cdot pref_i + pref_j^2$$

где $pref$ — массив префиксных сумм.

```
pair<int, int> calc(int X) {
    vector<pair<int, int>> dp(n + 1);
        add(0, 0);
    for (int i = 1; i <= n; i++) {
        int argmin = get(pref[i]);
        dp[i].first = dp[argmin].first + X + pref[i] * pref[i] - 2 * pref[argmin] * pref[i] + pre
        dp[i].second = dp[argmin].second + 1;
        add(-2 * pref[i], dp[i] + pref[i] * pref[i])
    }
    return dp[n];
}
/*
.
.
.
*/

int L = -INF;
int R = INF;
while (L + 1 < R) {
    int mid = (L + R) / 2;
    pair<int, int> X = calc(mid);
    if (X.second > k) {
        R = mid;
    }
    else {
        L = mid;
    }
}
pair<int, int> res = check(R);
cout << res.first - k * R;
```