C++ contest library

Miska Kananen

April 9, 2018

Contents

1 Environment and workflow

1.1 Compilation script

```
#!/bin/bash

g++ $1 -o ${1%.*} -std=c++11 -Wall -Wextra -Wshadow -
    ftrapv -Wfloat-equal -Wconversion -Wlogical-op -
    Wshift-overflow=2 -fsanitize=address -fsanitize=
    undefined -fno-sanitize-recover
```

2 General techniques

2.1 Bit tricks

g++ builtin functions:

- `__builtin_clz(x)`: number of zeros in the beginning
- `__builtin_ctz(x)`: number of zeros in the end
- `__builtin_popcount(x)`: number of set bits
- `__builtin_parity(x)`: parity of number of ones

There are separate functions of form `__builtin_clzll(x)` for 64-bit integers.

Iterate subsets of set s:

```
int cs = 0;
do {
        // process subset cs
} while(cs=(cs-s)&s);
```

3 Data structures

3.1 Lazy segment tree

Indexed $0..N-1$, defaults to range add+sum query tree which supports 2 queries:

1. Add x to all elements between $l..r$
2. Find the sum of all elements between $l..r$

```
#include <iostream>

using namespace std;
typedef long long ll;

const int N = (1<<18); // segtree max size

ll st[2*N]; // segtree values
ll lz[2*N]; // lazy updates
bool haslz[2*N]; // does a node have a lazy update
    pending

void push(int s, int l, int r) {
        if (haslz[s]) {
                st[s] += (r-l+1)*lz[s]; // change
                    operator+logic

                if (l != r) {
                        lz[2*s] += lz[s]; // change
                            operator
                        lz[2*s+1] += lz[s]; // change
                            operator
                        haslz[2*s] = true;
                        haslz[2*s+1] = true;
                }

                lz[s] = 0; // set to identity
                haslz[s] = false;
        }
}

ll kysy(int ql, int qr, int s = 1, int l = 0, int r = N
    -1) {
        push(s, l, r);
        if (l > qr || r < ql) {
                return 0; // set to identity
        }
        if (ql <= l && r <= qr) {
                return st[s];
        }

        int mid = (l+r)/2;
        ll res = 0; // set to identity
        res += kysy(ql, qr, 2*s, l, mid); // change
            operator
        res += kysy(ql, qr, 2*s+1, mid+1, r); // change
            operator
        return res;
}

void muuta(int ql, int qr, ll x, int s = 1, int l = 0,
    int r = N-1) {
        push(s, l, r);
        if (l > qr || r < ql) {
                return;
        }
        if (ql <= l && r <= qr) {
                lz[s] += x; // change operator
                haslz[s] = true;
                return;
        }

        int mid = (l+r)/2;
        muuta(ql, qr, x, 2*s, l, mid);
        muuta(ql, qr, x, 2*s+1, mid+1, r);

        st[s] = st[2*s] + st[2*s+1]; // change operator
        if (haslz[2*s]) {
                st[s] += (mid-l+1)*lz[2*s]; // change
                    operator+logic
        }
        if (haslz[2*s+1]) {
                st[s] += (r-(mid+1)+1)*lz[2*s+1]; //
                    change operator+logic
        }
}

void build(int s = 1, int l = 0, int r = N-1) {
```

```
        if (r-l > 1) {
                int mid = (l+r)/2;
                build(2*s, l, mid);
                build(2*s+1, mid+1, r);
        }
        st[s] = st[2*s]+st[2*s+1]; // change operator
}

// test code below
int n, q;

/*
        TESTED, correct
        Allowed indices 0..N-1
        2 types of queries: range add and range sum
*/
int main() {
        ios_base::sync_with_stdio(false);
        cin.tie(0);
        cin >> n >> q;
        for (int i = 1; i <= n; ++i) {
                cin >> st[i+N];
        }
        build();
        for (int cq = 0; cq < q; ++cq) {
                int tp;
                cin >> tp;
                if (tp == 1) {
                        int l, r;
                        ll x;
                        cin >> l >> r >> x;
                        muuta(l, r, x);
                }
                else {
                        int l, r;
                        cin >> l >> r;
                        cout << kysy(l, r) << "\n";
                }
        }
        return 0;
}
```

3.2 Sparse segment tree

3.3 Treap

Implements split, merge, kth element, range update and range reverse in $O(log(n))$. Range update adds a value to every element in a subarray. Treap is 1-indexed.

Note: Memory management tools warn of about 30 MB memory leak for 500 000 elements. This is because nodes are not deleted when exiting program and is irrelevant in a competition. Deleting nodes would slow treap down by a factor of 3.

```
#include <iostream>
#include <cstdlib>
#include <algorithm>

using namespace std;
typedef long long ll;

struct node {
        ll val; // change data type (char, integer...)
        int prio, size;
        bool lzinv;
        ll lzupd;
        bool haslz;
        node *left, *right;

        node(ll v) {
                val = v;
                prio = rand();
                size = 1;
                lzinv = false;
                lzupd = 0;
                haslz = false;
                left = nullptr;
                right = nullptr;
        }
};

int gsize(node *s) {
        if (s == nullptr) return 0;
        return s->size;
}
```

```
void upd(node *s) {
        if (s == nullptr) return;
        s->size = gsize(s->left) + 1 + gsize(s->right);
}

void push(node *s) {
        if (s == nullptr) return;

        if (s->haslz) {
                s->val += s->lzupd; // operator
        }
        if (s->lzinv) {
                swap(s->left, s->right);
        }

        if (s->left != nullptr) {
                if (s->haslz) {
                        s->left->lzupd += s->lzupd; //
                            operator
                        s->left->haslz = true;
                }
                if (s->lzinv) {
                        s->left->lzinv = !s->left->lzinv
                            ;
                }
        }
        if (s->right != nullptr) {
                if (s->haslz) {
                        s->right->lzupd += s->lzupd; //
                            operator
                        s->right->haslz = true;
                }
                if (s->lzinv)  {
                        s->right->lzinv = !s->right->
                            lzinv;
                }
        }

        s->lzupd = 0; // operator identity value
        s->lzinv = false;
        s->haslz = false;
}

// split a treap into two treaps, size of left treap = k
void split(node *t, node *&l, node *&r, int k) {
        push(t);
        if (t == nullptr) {
                l = nullptr;
                r = nullptr;
                return;
        }
        if (k >= gsize(t->left)+1) {
                split(t->right, t->right, r, k-(gsize(t
                    ->left)+1));
                l = t;
        }
        else {
                split(t->left, l, t->left, k);
                r = t;
        }
        upd(t);
}

// merge two treaps
void merge(node *&t, node *l, node *r) {
        push(l);
        push(r);
        if (l == nullptr) t = r;
        else if (r == nullptr) t = l;
        else {
                if (l->prio >= r->prio) {
                        merge(l->right, l->right, r);
                        t = l;
                }
                else {
                        merge(r->left, l, r->left);
                        t = r;
                }
        }
        upd(t);
}

// get k:th element in array (1-indexed)
ll kthElem(node *t, int k) {
        push(t);
        int cval = gsize(t->left)+1;
        if (k == cval) return t->val;
        if (k < cval) return kthElem(t->left, k);
        return kthElem(t->right, k-cval);
}
```

```
// do a lazy update on subarray [a..b]
void rangeUpd(node *&t, int a, int b, ll x) {
        node *cl, *cur, *cr;
        int tsz = gsize(t);
        bool lsplit = false;
        bool rsplit = false;
        cur = t;
        if (a > 1) {
                split(cur, cl, cur, a-1);
                lsplit = true;
        }
        if (b < tsz) {
                split(cur, cur, cr, b-a+1);
                rsplit = true;
        }
        cur->lzupd += x; // operator
        cur->haslz = true;
        if (lsplit) {
                merge(cur, cl, cur);
        }
        if (rsplit) {
                merge(cur, cur, cr);
        }
        t = cur;
}

// reverse subarray [a..b]
void rangeInv(node *&t, int a, int b) {
        node *cl, *cur, *cr;
        int tsz = gsize(t);
        bool lsplit = false;
        bool rsplit = false;
        cur = t;
        if (a > 1) {
                split(cur, cl, cur, a-1);
                lsplit = true;
        }
        if (b < tsz) {
                split(cur, cur, cr, b-a+1);
                rsplit = true;
        }
        cur->lzinv = !cur->lzinv;
        if (lsplit) {
                merge(cur, cl, cur);
        }
        if (rsplit) {
                merge(cur, cur, cr);
        }
        t = cur;
}


// test code below

int n, q;

/*
        TESTED, correct.

        Treap, allows split, merge, kth element, range
            update and range reverse in O(log n)
        It's also possible to implement range sum query
            (ioi16-treap IV)

        Implemented range update adds a value to every
            element in a subarray.

        NOTE: Memory management tools warn of a ~ 30MB
            memory leak for 500 000 nodes. This is
            because nodes are not deleted on program
            exit. Deleting would severely harm
            performance (over 3 times slower) and is
            unnecessary in a contest setting since the
            program is terminated anyway. Leak can be
            fixed by deleting nodes recursively on exit
            starting from leaf nodes and progressing
            towards root (post-order dfs).
*/
int main() {
        ios_base::sync_with_stdio(false);
        cin.tie(0);

        cin >> n >> q;
        node *tree = nullptr;
        for (int i = 1; i <= n; ++i) {
                node *nw = new node(0);
                merge(tree, tree, nw); // treap
                    construction
        }
```

```
        for (int cq = 0; cq < q; ++cq) {
                char tp;
                cin >> tp;
                if (tp == 'G') {
                        int cind;
                        cin >> cind;
                        cout << kthElem(tree, cind) << "
                            \n";
                }
                else if (tp == 'R') {
                        int a, b;
                        cin >> a >> b;
                        rangeInv(tree, a, b);
                }
                else {
                        int a, b;
                        ll d;
                        cin >> a >> b >> d;
                        rangeUpd(tree, a, b, d);
                }
        }
        return 0;
}
```

3.4 Indexed set (policy-based data structures)

Works like `std::set` but adds support for indices. Set is 0-indexed. Requires g++. Has two additional functions:

1. `find_by_order(x)`: return an iterator to element at index x
2. `order_of_key(x)`: return the index that element x has or would have in the set, depending on if it exists

Both functions work in $O(log(n))$.

Changing `less` to `less_equal` makes the set work like multiset. However, elements can't be removed.

```
#include <iostream>
#include <ext/pb_ds/assoc_container.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> indexed_set;

indexed_set s;

int main() {
        s.insert(2);
        s.insert(4);
        s.insert(5);

        auto x = s.find_by_order(1);
        cout << *x << "\n"; // prints 4

        cout << s.order_of_key(5) << "\n"; // prints 2
        cout << s.order_of_key(3) << "\n"; // prints 1
        return 0;
}
```

3.5 Union-find

Uses path compression, `id(x)` has amortized time complexity $O(a^{-1}(n))$ where a^{-1} is inverse Ackermann function.

```
#include <iostream>
#include <algorithm>

using namespace std;

int k[100005];
int s[100005];

int id(int x) {
        int tx = x;
        while (k[x] != x) x = k[x];
        return k[tx] = x;
}

bool equal(int a, int b) {
        return id(a) == id(b);
}
```

```
void join(int a, int b) {
        a = id(a);
        b = id(b);
        if (s[b] > s[a]) swap(a, b);
        s[a] += s[b];
        k[b] = a;
}

int n;

int main() {
        for (int i = 0; i < n; ++i) {
                k[i] = i;
                s[i] = 1;
        }
        return 0;
}
```

4 Mathematics

4.1 Number theory

- Prime factorization of n: $p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$
- Number of factors: $\tau(n) = \prod_{i=1}^{k} (\alpha_i + 1)$
- Sum of factors: $\sigma(n) = \prod_{i=1}^{k} \frac{p_i^{\alpha_i+1}-1}{p_i-1}$
- Product of factors: $\mu(n) = n^{\tau(n)/2}$

Euler's totient function $\varphi(n)$ $(1, 1, 2, 2, 4, 2, 6, 4, 6, 4, \dots)$: counts numbers coprime with n in range $1 \dots n$

$$\varphi(n) = \begin{cases} n-1 & \text{if } n \text{ is prime} \\ \prod_{i=1}^{k} p_i^{a_i-1}(p_i-1) & \text{otherwise} \end{cases}$$

Fermat's theorem: $x^{m-1} \mod m = 1$ when m is prime and x and m are coprime. It follows that $x^k \mod m = x^{k \mod (m-1)} \mod m$.

Modular inverse $x^{-1} = x^{\varphi(m)-1}$. If m is prime, $x^{-1} = x^{m-2}$. Inverse exists if and only if x and m are coprime.

4.2 Combinatorics

Binomial coefficients:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

Catalan numbers $(1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796 \dots)$:

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Classic examples of Catalan numbers: number of balanced pairs of parentheses, number of mountain ranges (n upstrokes and n downstrokes all staying above the original line), number of paths from upper left corner to lower right corner staying above the main diagonal in a $n \times n$ square, ways to trianguate a $n+2$ sided regular polygon, ways to shake hands between $2n$ people in a circle such that no arms cross, number of rooted binary trees with n nodes that have 2 children, number of rooted trees with n edges, number of permutations of $1 \dots n$ that don't have an increasing subsequence of length 3.

Number of derangements (no element stays in original place) of $1, 2, \dots, n$ $(1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961, \dots)$:

$$f(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ (n-1)(f(n-2) + f(n-1)) & n > 2 \end{cases}$$

4.3 Pollard-Rho

Finds a prime factor of x in $O(\sqrt[4]{x})$. Requires __int128 support for factoring 64-bit integers.

If x is prime, algorithm might not terminate or it might return 1. Primality must be checked separately.

```
#include <iostream>
#include <cstdlib>
#include <algorithm>

using namespace std;

typedef long long ll;
typedef __int128 lll;

ll n;

ll f(lll x) {
    return (x*x+1)%n;
}

ll gcd(ll a, ll b) {
    if (b == 0) return a;
    return gcd(b, a%b);
}

// return a prime factor of a
// st is a starting seed for pseudorandom numbers, start
     with 2, if algorithm fails (returns -1), increment
    seed
ll pollardrho(ll a, ll st) {
    if (n%2 == 0) return 2;

    ll x = st, y = st, d = 1;
    while (d == 1) {
        x = f(x);
        y = f(f(y));
        d = gcd(abs(x-y), a);
        if (d == a) return -1;
    }
    return d;
}

/*
    TESTED, correct.
    Finds a prime factor of n in O(root_4(n))
    If n is prime, alg might not terminate or it might
        return 1. Check for primality.
*/
int main() {
    cin >> n;
    ll fa = -1;
    ll st = 2;
    while (fa == -1) {
        fa = pollardrho(n, st++);
    }
    cout << min(fa, n/fa) << " " << max(fa, n/fa) << "\n
        ";
    return 0;
}
```

4.4 Matrices

Matrix $A = a \times n$, matrix $B = n \times b$. Matrix multiplication:

$$AB[i,j] = \sum_{k=1}^{n} A[i,k] \cdot B[k,j]$$

Let linear recurrence $f(n) = c_1 f(n-1) + c_2 f(n-2) + \cdots + c_k f(n-k)$ with initial values $f(0), f(1), \ldots, f(k-1)$. $c_1, c_2, \ldots, c_n$ are constants.

Transition matrix X:

$$X = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ c_k & c_{k-1} & c_{k-2} & \ldots & c_1 \end{pmatrix}$$

Now $f(n)$ can be calculated in $O(k^3 log(n))$:

$$\begin{pmatrix} f(n) \\ f(n+1) \\ \vdots \\ f(n+k-1) \end{pmatrix} = X^n \cdot \begin{pmatrix} f(0) \\ f(1) \\ \vdots \\ f(k-1) \end{pmatrix}$$

```
#include <iostream>
#include <cstring>

using namespace std;
typedef long long ll;

const int N = 2; // matrix size
const ll M = 1000000007; // modulo

struct matrix {
    ll m[N][N];
    matrix() {
        memset(m, 0, sizeof m);
    }
    matrix operator * (matrix b) {
        matrix c = matrix();
        for (int i = 0; i < N; ++i)
            for (int j = 0; j < N; ++j)
                for (int k = 0; k < N; ++k) {
                    c.m[i][j] = (c.m[i][j] + m[i][k] * b
                        .m[k][j])%M;
                }
        return c;
    }
    matrix unit() {
        matrix a = matrix();
        for (int i = 0; i < N; ++i) a.m[i][i] = 1;
        return a;
    }
};

matrix p(matrix a, ll e) {
    if (e == 0) return a.unit();
    if (e%2 == 0) {
        matrix h = p(a, e/2);
        return h*h;
    }
    return (p(a, e-1)*a);
}

ll n;

// prints nth Fibonacci number mod M
int main() {
    cin >> n;
    matrix x = matrix();
    x.m[0][1] = 1;
    x.m[1][0] = 1;
    x.m[1][1] = 1;
    x = p(x, n);
    cout << x.m[0][1] << "\n";
    return 0;
}
```

5 Graph algorithms

5.1 Kosaraju's algorithm

Finds strongly connected components in a directed graph in $O(n+m)$.

1. Create an inverse graph where all edges are reversed.
2. Do a DFS traversal on original graph and add all nodes in post-order to a vector.
3. Reverse the previous vector.
4. Iterate the vector. If a node doesn't belong to a component, create new component and assign current node to it, and do a DFS search **in inverse graph** from current node and add all reachable nodes to the component that was just created.

5.2 Bridges

An edge $u - v$ is a bridge if there is no edge from the subtree of v to any node with lower depth than u in DFS tree. $O(n + m)$.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int n, m;
vector<int> g[200010];

int v[200010];
int d[200010];

// found bridges
vector<pair<int, int>> res;

// find bridges
int bdfs(int s, int cd, int p) {
    if (v[s]) return d[s];
    v[s] = 1;
    d[s] = cd;

    int minh = cd;

    for (int a : g[s]) {
        if (a == p) continue;
        minh = min(minh, bdfs(a, cd+1, s));
    }

    if (p != -1) {
        if (minh == cd) {
            res.push_back({s, p});
        }
    }
    return minh;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int a, b;
        cin >> a >> b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    for (int i = 1; i <= n; ++i) {
        if (!v[i]) bdfs(i, 1, -1);
    }
    cout << res.size() << "\n";
    for (auto a : res) {
        cout << a.first << " " << a.second << "\n";
    }

    return 0;
}
```

5.3 Articulation points

A vertex u is an articulation point if there is no edge from the subtree of u to any parent of u in DFS tree, of if u is the root of DFS tree and has at least 2 children. $O(n+m)$ if removing duplicates doesn't count.

Set `res` can be replaced with a vector if duplicates are removed afterwards.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <set>

using namespace std;

int n, m;
vector<int> g[200010];
int v[200010];
int dt[200010];
int low[200010];

// found articulation points
// can be replaced with vector, but duplicates must be
    removed
set<int> res;

int curt = 1;

void adfs(int s, int p) {
```

```
    if (v[s]) return;
    v[s] = 1;
    dt[s] = curt++;
    low[s] = dt[s];

    int ccount = 0;

    for (int a : g[s]) {
        if (!v[a]) {
            ++ccount;
            adfs(a, s);
            low[s] = min(low[s], low[a]);

            if (low[a] >= dt[s] && p != -1) res.insert(s
                );
        }
        else if (a != p) {
            low[s] = min(low[s], dt[a]);
        }

        if (p == -1 && ccount > 1) {
            res.insert(s);
        }
    }
}


int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int a, b;
        cin >> a >> b;
        g[a].push_back(b);
        g[b].push_back(a);
    }

    for (int i = 1; i <= n; ++i) {
        if (!v[i]) adfs(i, -1);
    }
    cout << res.size() << "\n";
    for (int a : res) cout << a << "\n";
    return 0;
}
```

5.4 Maximum flow (scaling algorithm)

5.5 Heavy-light decomposition

Supports updates and queries on path between two vertices a and b in $O(log^2(n))$.

Doesn't explicitly look for LCA, instead climbs upwards from the lower chain until both vertices are in the same chain.

Requires a normal segment tree implementation that corresponds to the queries.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;
typedef long long ll;

const int S = 100005; // vertex count
const int N = (1<<18); // segtree size, must be >= S

vector<int> g[S];

int sz[S], de[S], pa[S];
int cind[S], chead[S], cpos[S];
int cchain, cstind, stind[S];

// IMPLEMENT SEGMENT TREE HERE
// st_update() and st_query() should call segtree
    functions
ll st[2*N];

void hdfs(int s, int p, int cd) {
    de[s] = cd;
    pa[s] = p;
    sz[s] = 1;
    for (int a : g[s]) {
        if (a == p) continue;
        hdfs(a, s, cd+1);
        sz[s] += sz[a];
    }
}

void hld(int s) {
```

```
    if (chead[cchain] == 0) {
        chead[cchain] = s;
        cpos[s] = 0;
    }
    else {
        cpos[s] = cpos[pa[s]]+1;
    }
    cind[s] = cchain;

    stind[s] = cstind;
    cstind++;

    int cmx = 0, cmi = -1;
    for (int i = 0; i < g[s].size(); ++i) {
        if (g[s][i] == pa[s]) continue;
        if (sz[g[s][i]] > cmx) {
            sz[g[s][i]] = cmx;
            cmi = i;
        }
    }

    if (cmi != -1) {
        hld(g[s][cmi]);
    }

    for (int i = 0; i < g[s].size(); ++i) {
        if (i == cmi) continue;
        if (g[s][i] == pa[s]) continue;
        cchain++;
        cstind++;
        hld(g[s][i]);
    }
}

// do a range update on underlying segtree
// sa and sb are segtree indices
void st_update(int sa, int sb, ll x) {

}

// do a range query on underlying segtree
// sa and sb are segtree indices
ll st_query(int sa, int sb) {

}
```

```
// update all vertices on path from vertex a to b
// a and b are vertex numbers
void path_update(int a, int b, ll x) {
    while (cind[a] != cind[b]) {
        if (de[chead[cind[b]]] > de[chead[cind[a]]])
            swap(a, b);
        st_update(stind[chead[cind[a]]], stind[a], x);
        a = pa[chead[cind[a]]];
    }
    if (stind[b] < stind[a]) swap(a, b);
    st_update(stind[a], stind[b], x);
}

// query all vertices on path from vertex a to b
// a and b are vertex numbers
ll path_query(int a, int b) {
        ll cres = 0; // set to identity
        while (cind[a] != cind[b]) {
        if (de[chead[cind[b]]] > de[chead[cind[a]]])
            swap(a, b);
        cres += st_query(stind[chead[cind[a]]], stind[a
            ]); // change operator
        a = pa[chead[cind[a]]];
    }
    if (stind[b] < stind[a]) swap(a, b);
    cres += st_query(stind[a], stind[b]); // change
        operator
    return cres;
}


int n, m;

// TESTED, correct
// do updates and queries on paths between two nodes in
    a tree
// interface: path_update() and path_query()
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cin >> n >> m;
    for (int i = 0; i < n-1; ++i) {
        int a, b;
        cin >> a >> b;
```

```
        g[a].push_back(b);
        g[b].push_back(a);
    }

    // init hld
    hdfs(1, -1, 0);
    hld(1);

    // handle queries
    return 0;
}
```

6 String algorithms

6.1 Polynomial hashing

If hash collisions are likely, compute two hashes with two distinct pairs of constants of magnitude 10^9 and use their product as the actual hash.

```
#include <iostream>

using namespace std;

const ll A = 957262683;
const ll B = 998735246;

string s;
ll h[1000005];
ll p[1000005];

ll ghash(int a, int b) {
        if (a == 0) return h[b];
        ll cres = (h[b]-h[a-1]*p[b-a+1])%B;
        if (cres < 0) cres += B;
        return cres;
}

int main() {
        cin >> s;

        h[0] = s[0];
        p[0] = 1;

        for (int i = 1; i < s.length(); ++i) {
                h[i] = (h[i-1]*A+s[i])%B;
                p[i] = (p[i-1]*A)%B;
        }
        return 0;
}
```