Содержание

1 Strategy.txt

```
-  Проверить руками сэмплы
-  Подумать как дебагать после написания
-  Выписать сложные формулы и все +-1

-  Проверить имена файлов
-  Прогнать сэмплы
-  Переполнения int, переполнения long long
-  Выход за границу массива: _GLIBCXX_DEBUG
-  Переполнения по модулю: в
↪   псевдо-онлайн-генераторе, в функциях-обертках
-  Проверить мультитест на разных тестах
-  Прогнать минимальный по каждому параметру тест
-  Прогнать псевдо-максимальный тест(немного чисел,
↪   но очень большие или очень маленькие)
-  Представить что не зайдет и заранее написать
↪   assert'ы, прогнать слегка модифицированные тесты
-  cout.precision: в том числе в интерактивных
↪   задачах
-  Удалить debug-output, отсечения для тестов,
↪   вернуть оригинальный maxn, удалить
↪   _GLIBCXX_DEBUG

-  Вердикт может врать
-  Если много тестов(>3), дописать в конец каждого
↪   теста ответ, чтобы не забыть
-  (WA) Потестить не только ответ, но и содержимое
↪   значимых массивов, переменных
-  (WA) Изменить тест так, чтобы ответ не менялся:
↪   поменять координаты местами, сжать/растянуть
↪   координаты, поменять ROOT дерева
-  (WA) Подвигать размер блока в корневой или
↪   битсете
-  (WA) Поставить assert'ы, возможно написать чекер
↪   с assert'ом
-  (WA) Проверить, что программа не печатает
↪   что-либо неожиданное, что должно попадать под
↪   PE: inf - 2, не лекс. мин. решение, одинаковые
↪   числа вместо разных, неправильное количество
↪   чисел, пустой ответ, перечитать output format
-  (TL) cin -> scanf -> getchar
-  (TL) Упихать в кэш большие массивы, поменять
↪   местами for'ы или измерения массива
-  (RE) Проверить формулы на деление на 0, выход за
↪   область определения(sqrt(-eps), acos(1 + eps))
```

2 algo/flows/globalcut.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i,n) for (int i = 0; i < int(n); ++i)
const int inf = 1e9 + 1e5;

const int maxn = 505;
namespace StoerWagner {
    int g[maxn][maxn];
    int dist[maxn];
    bool used[maxn];
    int n;

    void addEdge(int u, int v, int c) {
        g[u][v] += c;
        g[v][u] += c;
    }

    int run() {
        vector<int> vertices;
        forn (i, n)
            vertices.push_back(i);
        int mincut = inf;
        while (vertices.size() > 1) {
            int u = vertices[0];
            for (auto v: vertices) {
                used[v] = false;
                dist[v] = g[u][v];
            }
            used[u] = true;
            forn (ii, vertices.size() - 2) {
                for (auto v: vertices)
                    if (!used[v])
                        if (used[u] || dist[v] > dist[u])
                            u = v;
                used[u] = true;
                for (auto v: vertices)
                    if (!used[v])
                        dist[v] += g[u][v];
            }
            int t = -1;
            for (auto v: vertices)
                if (!used[v])
                    t = v;
            assert(t != -1);
            mincut = min(mincut, dist[t]);
            vertices.erase(find(vertices.begin(), vertices.end(), t));
            for (auto v: vertices)
                addEdge(u, v, g[v][t]);
        }
        return mincut;
    }
};

int main() {
    StoerWagner::n = 4;
    StoerWagner::addEdge(0, 1, 5);
    StoerWagner::addEdge(2, 3, 5);
    StoerWagner::addEdge(1, 2, 4);
    cerr << StoerWagner::run() << '\n';
}
```

3 algo/flows/hungary.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i,n) for (int i = 0; i < int(n); ++i)
const int inf = 1e9 + 1e5;

// left half is the smaller one
namespace Hungary {
    const int maxn = 505;
    int a[maxn][maxn];
    int p[2][maxn];
    int match[maxn];
    bool used[maxn];
    int from[maxn];
    int mind[maxn];
    int n, m;

    int hungary(int v) {
        used[v] = true;
        int u = match[v];
        int best = -1;
        forn (i, m + 1) {
            if (used[i])
                continue;
            int nw = a[u][i] - p[0][u] - p[1][i];
            if (nw <= mind[i]) {
                mind[i] = nw;
                from[i] = v;
            }
            if (best == -1 || mind[best] > mind[i])
                best = i;
        }
        v = best;
        int delta = mind[best];
        forn (i, m + 1) {
            if (used[i]) {
                p[1][i] -= delta;
                p[0][match[i]] += delta;
            } else
                mind[i] -= delta;
        }
        if (match[v] == -1)
            return v;
        return hungary(v);
    }

    void check() {
        int edges = 0, res = 0;
        forn (i, m)
            if (match[i] != -1) {
                ++edges;
                assert(p[0][match[i]] + p[1][i] == a[match[i]][i]);
                res += a[match[i]][i];
            } else
                assert(p[1][i] == 0);
        assert(res == -p[1][m]);
        forn (i, n) forn (j, m)
            assert(p[0][i] + p[1][j] <= a[i][j]);
    }

    int run() {
        forn (i, n)
            p[0][i] = 0;
        forn (i, m + 1) {
            p[1][i] = 0;
            match[i] = -1;
        }
        forn (i, n) {
            match[m] = i;
            fill(used, used + m + 1, false);
            fill(mind, mind + m + 1, inf);
            fill(from, from + m + 1, -1);
            int v = hungary(m);
            while (v != m) {
                int w = from[v];
                match[v] = match[w];
                v = w;
            }
        }
        check();
        return -p[1][m];
    }
};

int main() {
    int n = 300, m = 500;
    Hungary::n = n, Hungary::m = m;
    forn (i, n) forn (j, m) Hungary::a[i][j] = rand() % 200001 - 100000;
    cerr << Hungary::run() << "\n";
}
```

4 algo/flows/mincost.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define forn(i,n) for (int i = 0; i < int(n); ++i)

namespace MinCost {
    const ll infc = 1e12;

    struct Edge {
        int to;
        ll c, f, cost;

        Edge(int to, ll c, ll cost): to(to), c(c), f(0), cost(cost) {
        }
    };

    int N, S, T;
    int totalFlow;
    ll totalCost;
    const int maxn = 505;
    vector<Edge> edge;
    vector<int> g[maxn];

    void addEdge(int u, int v, ll c, ll cost) {
        g[u].push_back(edge.size());
        edge.emplace_back(v, c, cost);
        g[v].push_back(edge.size());
        edge.emplace_back(u, 0, -cost);
    }

    ll dist[maxn];
    int fromEdge[maxn];

    bool inQueue[maxn];
    bool fordBellman() {
        forn (i, N)
            dist[i] = infc;
        dist[S] = 0;
        inQueue[S] = true;
        vector<int> q;
        q.push_back(S);
        for (int ii = 0; ii < int(q.size()); ++ii) {
            int u = q[ii];
            inQueue[u] = false;
            for (int e: g[u]) {
                if (edge[e].f == edge[e].c)
                    continue;
                int v = edge[e].to;
                ll nw = edge[e].cost + dist[u];
                if (nw >= dist[v])
                    continue;
                dist[v] = nw;
                fromEdge[v] = e;
                if (!inQueue[v]) {
                    inQueue[v] = true;
                    q.push_back(v);
                }
            }
        }
        return dist[T] != infc;
    }

    ll pot[maxn];
    bool dikstra() {
        priority_queue<pair<ll, int>, vector<pair<ll, int>>,
  greater<pair<ll, int>>> q;
        forn (i, N)
            dist[i] = infc;
        dist[S] = 0;
        q.emplace(dist[S], S);
        while (!q.empty()) {
            int u = q.top().second;
            ll cdist = q.top().first;
            q.pop();
            if (cdist != dist[u])
                continue;
            for (int e: g[u]) {
                int v = edge[e].to;
                if (edge[e].c == edge[e].f)
                    continue;
                ll w = edge[e].cost + pot[u] - pot[v];
                assert(w >= 0);
                ll ndist = w + dist[u];
                if (ndist >= dist[v])
                    continue;
                dist[v] = ndist;
                fromEdge[v] = e;
                q.emplace(dist[v], v);
            }
        }
        if (dist[T] == infc)
            return false;
        forn (i, N) {
            if (dist[i] == infc)
                continue;
            pot[i] += dist[i];
        }
        return true;
    }

    bool push() {
        //2 variants
        //if (!fordBellman())
        if (!dikstra())
            return false;
        ++totalFlow;
        int u = T;
        while (u != S) {
            int e = fromEdge[u];
            totalCost += edge[e].cost;
            edge[e].f++;
            edge[e ^ 1].f--;
            u = edge[e ^ 1].to;
        }
        return true;
    }
};

int main() {
    MinCost::N = 3, MinCost::S = 1, MinCost::T = 2;
    MinCost::addEdge(1, 0, 3, 5);
    MinCost::addEdge(0, 2, 4, 6);
    while (MinCost::push());
    cout << MinCost::totalFlow << ' ' << MinCost::totalCost << '\n'; //3
  33
}
```

5 algo/geometry/primitives.cpp

```cpp
#include <bits/stdc++.h>
#define forn(i, n) for (int i = 0; i < int(n); ++i)
using namespace std;
typedef long double ld;

const ld eps = 1e-9;

bool eq(ld a, ld b) { return fabsl(a - b) < eps; }
bool le(ld a, ld b) { return b - a > -eps; }
bool ge(ld a, ld b) { return a - b > -eps; }
bool lt(ld a, ld b) { return b - a > eps; }
bool gt(ld a, ld b) { return a - b > eps; }
ld sqr(ld x) { return x * x; }

inline void gassert(bool expr) {
#ifdef LOCAL
    assert(expr);
#endif
}

struct pt {
    ld x, y;

    pt operator+(const pt &p) const { return pt{x + p.x, y + p.y}; }
    pt operator-(const pt &p) const { return pt{x - p.x, y - p.y}; }
    ld operator*(const pt &p) const { return x * p.x + y * p.y; }
    ld operator%(const pt &p) const { return x * p.y - y * p.x; }

    pt operator*(const ld &a) const { return pt{x * a, y * a}; }
    pt operator/(const ld &a) const { gassert(!eq(a, 0)); return pt{x / a, y / a}; }
    void operator*=(const ld &a) { x *= a, y *= a; }
    void operator/=(const ld &a) { gassert(!eq(a, 0)); x /= a, y /= a; }

    bool operator<(const pt &p) const {
        if (eq(x, p.x)) return lt(y, p.y);
        return x < p.x;
    }

    bool operator==(const pt &p) const { return eq(x, p.x) && eq(y, p.y); }
    bool operator!=(const pt &p) const { return !(*this == p); }

    pt rot() { return pt{-y, x}; }
    ld abs() const { return hypotl(x, y); }
    ld abs2() const { return x * x + y * y; }
};

istream &operator>>(istream &in, pt &p) { return in >> p.x >> p.y; }
ostream &operator<<(ostream &out, const pt &p) { return out << p.x << ' ' << p.y; }

//WARNING! do not forget to normalize vector (a,b)
struct line {
    ld a, b, c;

    line(pt p1, pt p2) {
        gassert(p1 != p2);
        pt n = (p2 - p1).rot();
        n /= n.abs();
        a = n.x, b = n.y;
        c = -(n * p1);
    }

    line(ld _a, ld _b, ld _c): a(_a), b(_b), c(_c) {
        ld d = pt{a, b}.abs();
        gassert(!eq(d, 0));
        a /= d, b /= d, c /= d;
    }

    ld signedDist(pt p) {
        return p * pt{a, b} + c;
    }
};

ld pointSegmentDist(pt p, pt a, pt b) {
    ld res = min((p - a).abs(), (p - b).abs());
    if (a != b && ge((p - a) * (b - a), 0) && ge((p - b) * (a - b), 0))
        res = min(res, fabsl((p - a) % (b - a)) / (b - a).abs());
    return res;
}

pt linesIntersection(line l1, line l2) {
    ld D = l1.a * l2.b - l1.b * l2.a;
    if (eq(D, 0)) {
        if (eq(l1.c, l2.c)) {
            //equal lines
        } else {
            //no intersection
        }
    }
    ld dx = -l1.c * l2.b + l1.b * l2.c;
    ld dy = -l1.a * l2.c + l1.c * l2.a;
    pt res{dx / D, dy / D};
    gassert(eq(l1.signedDist(res), 0));
    gassert(eq(l2.signedDist(res), 0));
    return res;
}

bool pointInsideSegment(pt p, pt a, pt b) {
    if (!eq((p - a) % (p - b), 0))
        return false;
    return le((a - p) * (b - p), 0);
}

bool checkSegmentIntersection(pt a, pt b, pt c, pt d) {
    if (eq((a - b) % (c - d), 0)) {
        if (pointInsideSegment(a, c, d) || pointInsideSegment(b, c, d) ||
                pointInsideSegment(c, a, b) || pointInsideSegment(d, a, b)) {
            //intersection of parallel segments
            return true;
        }
        return false;
    }

    ld s1, s2;

    s1 = (c - a) % (b - a);
    s2 = (d - a) % (b - a);
    if (gt(s1, 0) && gt(s2, 0))
        return false;
    if (lt(s1, 0) && lt(s2, 0))
        return false;

    swap(a, c), swap(b, d);

    s1 = (c - a) % (b - a);
    s2 = (d - a) % (b - a);
    if (gt(s1, 0) && gt(s2, 0))
        return false;
    if (lt(s1, 0) && lt(s2, 0))
        return false;

    return true;
}

//WARNING! run checkSegmentIntersecion before and process parallel case manually
pt segmentsIntersection(pt a, pt b, pt c, pt d) {
    ld S = (b - a) % (d - c);
    ld s1 = (c - a) % (d - a);
    return a + (b - a) / S * s1;
}

vector<pt> circlesIntersction(pt a, ld r1, pt b, ld r2) {
    ld d2 = (a - b).abs2();
    ld d = (a - b).abs();

    if (a == b && eq(r1, r2)) {
        //equal circles
    }
    if (lt(sqr(r1 + r2), d2) || gt(sqr(r1 - r2), d2)) {
        //empty intersection
        return {};
    }
    int num = 2;
    if (eq(sqr(r1 + r2), d2) || eq(sqr(r1 - r2), d2))
        num = 1;
    ld cosa = (sqr(r1) + d2 - sqr(r2)) / ld(2 * r1 * d);
    ld oh = cosa * r1;
    pt h = a + ((b - a) / d * oh);
    if (num == 1)
        return {h};
    ld hp = sqrtl(max(0.L, 1 - cosa * cosa)) * r1;

    pt w = ((b - a) / d * hp).rot();
    return {h + w, h - w};
}

//a is circle center, p is point
vector<pt> circleTangent(pt a, ld r, pt p) {
    ld d2 = (a - p).abs2();
    ld d = (a - p).abs();

    if (gt(sqr(r), d2)) {
        //no tangents
        return {};
    }
    if (eq(sqr(r), d2)) {
        //point lies on circle - one tangent
        return {p};
    }

    pt B = p - a;
    pt H = B * sqr(r) / d2;
    ld h = sqrtl(d2 - sqr(r)) * ld(r) / d;
    pt w = (B / d * h).rot();
    H = H + a;
    return {H + w, H - w};
}

vector<pt> lineCircleIntersection(line l, pt a, ld r) {
    ld d = l.signedDist(a);
    if (gt(fabsl(d), r))
        return {};
    pt h = a - pt{l.a, l.b} * d;
    if (eq(fabsl(d), r))
        return {h};
    pt w = pt{l.a, l.b}.rot() * sqrtl(max<ld>(0, sqr(r) - sqr(d)));
    return {h + w, h - w};
}

int main() {
    cout.precision(10);
    cout << fixed;
}
```

6 algo/graphs/2sat.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i,n) for (int i = 0; i < int(n); ++i)
const int maxn = 200100; //2 x number of variables

namespace TwoSAT {
    int n; //number of variables
    bool used[maxn];
    vector<int> g[maxn];
    vector<int> gr[maxn];
    int comp[maxn];
    int res[maxn];

    void addEdge(int u, int v) { //u or v
        g[u].push_back(v ^ 1);
        g[v].push_back(u ^ 1);
        gr[u ^ 1].push_back(v);
        gr[v ^ 1].push_back(u);
    }

    vector<int> ord;
    void dfs1(int u) {
        used[u] = true;
        for (int v: g[u]) {
            if (used[v])
                continue;
            dfs1(v);
        }
        ord.push_back(u);
    }

    int COL = 0;
    void dfs2(int u) {
        used[u] = true;
        comp[u] = COL;
        for (int v: gr[u]) {
            if (used[v])
                continue;
            dfs2(v);
        }
    }

    void mark(int u) {
        res[u / 2] = u % 2;
        used[u] = true;
        for (int v: g[u]) {
            if (used[v])
                continue;
            mark(v);
        }
    }

    bool run() {
        fill(res, res + 2 * n, -1);
        fill(used, used + 2 * n, false);
        forn (i, 2 * n)
            if (!used[i])
                dfs1(i);
        reverse(ord.begin(), ord.end());
        assert((int) ord.size() == (2 * n));
        fill(used, used + 2 * n, false);
        for (int u: ord) if (!used[u]) {
            dfs2(u);
            ++COL;
        }
        forn (i, n)
            if (comp[i * 2] == comp[i * 2 + 1])
                return false;

        reverse(ord.begin(), ord.end());
        fill(used, used + 2 * n, false);
        for (int u: ord) {
            if (res[u / 2] != -1) {
                continue;
            }
            mark(u);
        }
        return true;
    }
};

int main() {
    TwoSAT::n = 2;
    TwoSAT::addEdge(0, 2); //x or y
    TwoSAT::addEdge(0, 3); //x or !y
    TwoSAT::addEdge(3, 3); //!y or !y
    assert(TwoSAT::run());
    cout << TwoSAT::res[0] << ' ' << TwoSAT::res[1] << '\n'; //1 0
}
```

7 algo/math/fft_recursive.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
typedef long long i64;

typedef double ld;

struct base {
    ld re, im;
    base(){}
    base(ld re) : re(re), im(0) {}
    base(ld re, ld im) : re(re), im(im) {}

    base operator+(const base& o) const { return {re+o.re, im+o.im}; }
    base operator-(const base& o) const { return {re-o.re, im-o.im}; }
    base operator*(const base& o) const {
        return {
            re*o.re - im*o.im,
            re*o.im + im*o.re
        };
    }
};

const int sz = 1<<20;

int revb[sz];
vector<base> ang[21];

void init(int n) {
    int lg = 0;
    while ((1<<lg) != n) {
        ++lg;
    }
    forn(i, n) {
        revb[i] = (revb[i>>1]>>1)^((i&1)<<(lg-1));
    }

    ld e = M_PI * 2 / n;
    ang[lg].resize(n);
    forn(i, n) {
        ang[lg][i] = { cos(e * i), sin(e * i) };
    }

    for (int k = lg - 1; k >= 0; --k) {
        ang[k].resize(1 << k);
        forn(i, 1<<k) {
            ang[k][i] = ang[k+1][i*2];
        }
    }
}

void fft_rec(base *a, int lg, bool rev) {
    if (lg == 0) {
        return;
    }
    int len = 1 << (lg - 1);
    fft_rec(a, lg-1, rev);
    fft_rec(a+len, lg-1, rev);

    forn(i, len) {
        base w = ang[lg][i];
        if (rev) w.im *= -1;
        base u = a[i];
        base v = a[i+len] * w;
        a[i] = u + v;
        a[i+len] = u - v;
    }
}

void fft(base *a, int n, bool rev) {
    forn(i, n) {
        int j = revb[i];
        if (i < j) swap(a[i], a[j]);
    }
    int lg = 0;
    while ((1<<lg) != n) {
        ++lg;
    }
    fft_rec(a, lg, rev);
    if (rev) forn(i, n) {
        a[i] = a[i] * (1.0 / n);
    }
}

const int maxn = 1050000;

int n;
base a[maxn];
base b[maxn];

void test() {
    int n = 1<<19;
    mt19937 rr(55);
    forn(i, n) a[i] = rr() % 10000;
    forn(j, n) b[j] = rr() % 10000;

    int N = 1;
    while (N < 2*n) N *= 2;

    clock_t start = clock();
    init(N);
    cerr << "init time: " << (clock()-start) / 1000 << " ms" << endl;
    fft(a, N, 0);
    fft(b, N, 0);
    forn(i, N) a[i] = a[i] * b[i];
    fft(a, N, 1);
    clock_t end = clock();

    ld err = 0;
    forn(i, N) {
        err = max(err, (ld)fabsl(a[i].im));
        err = max(err, (ld)fabsl(a[i].re - (i64(a[i].re + 0.5))));
    }

    cerr << "Time: " << (end - start) / 1000 << " ms, err = " << err <<
 ↪   endl;
}

int main() {
    test();
}
```

8 algo/math/golden_search.cpp

```cpp
#include <bits/stdc++.h>
typedef long double ld;
#define forn(i, n) for (int i = 0; i < int(n); ++i)

ld f(ld x) {
    return 5 * x * x + 100 * x + 1; //-10 is minimum
}

ld goldenSearch(ld l, ld r) {
    ld phi = (1 + sqrtl(5)) / 2;
    ld resphi = 2 - phi;
    ld x1 = l + resphi * (r - l);
    ld x2 = r - resphi * (r - l);
    ld f1 = f(x1);
    ld f2 = f(x2);
    forn (iter, 60) {
        if (f1 < f2) {
            r = x2;
            x2 = x1;
            f2 = f1;
            x1 = l + resphi * (r - l);
            f1 = f(x1);
        } else {
            l = x1;
            x1 = x2;
            f1 = f2;
            x2 = r - resphi * (r - l);
            f2 = f(x2);
        }
    }
    return (x1 + x2) / 2;
}

int main() {
    std::cout << goldenSearch(-100, 100) << '\n';
}
```

9 algo/math/numbers.txt

```
Simpson's numerical integration: integral from a to
↪  b f(x) dx = (b - a) / 6 * (f(a) + 4 * f((a + b)
↪  / 2) + f(b))
```

10 algo/strings/automaton.cpp

```cpp
//real    4m27.689s
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for (int i = 0; i < (int)(n); ++i)

const int maxn = 100500;

int t[maxn][26], lnk[maxn], len[maxn];
int sz;
int last;

void init() {
    sz = 3;
    last = 1;
    forn(i, 26) t[2][i] = 1;
    len[2] = -1;
    lnk[1] = 2;
}

void addchar(int c) {
    int nlast = sz++;
    len[nlast] = len[last] + 1;
    int p = last;
    for (; !t[p][c]; p = lnk[p]) {
        t[p][c] = nlast;
    }
    int q = t[p][c];
    if (len[p] + 1 == len[q]) {
        lnk[nlast] = q;
    } else {
        int clone = sz++;
        len[clone] = len[p] + 1;
        lnk[clone] = lnk[q];
        lnk[q] = lnk[nlast] = clone;
        forn(i, 26) t[clone][i] = t[q][i];
        for (; t[p][c] == q; p = lnk[p]) {
            t[p][c] = clone;
        }
    }
    last = nlast;
}

bool check(const string& s) {
    int v = 1;
    for (int c: s) {
        c -= 'a';
        if (!t[v][c]) return false;
        v = t[v][c];
    }
    return true;
}

int main() {
    string s;
    cin >> s;
    init();
    for (int i: s) {
        addchar(i-'a');
    }
    forn(i, s.length()) {
        assert(check(s.substr(i)));
    }
    cout << sz << endl;
    return 0;
}
```

11 algo/strings/suffix_array.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for (int i = 0; i < (int)(n); ++i)

const int maxn = 100500;

string s;
int n;
int sa[maxn], new_sa[maxn], cls[maxn], new_cls[maxn], cnt[maxn],
 ↪  lcp[maxn];
int n_cls;

void build() {
    n_cls = 256;
    forn(i, n) {
        sa[i] = i;
        cls[i] = s[i];
    }
    for (int d = 0; d < n; d = d ? d*2 : 1) {

        forn(i, n) new_sa[i] = (sa[i] - d + n) % n;
        forn(i, n_cls) cnt[i] = 0;
        forn(i, n) ++cnt[cls[i]];
        forn(i, n_cls) cnt[i+1] += cnt[i];
        for (int i = n-1; i >= 0; --i) sa[--cnt[cls[new_sa[i]]]] =
 ↪  new_sa[i];

        n_cls = 0;
        forn(i, n) {
            if (i && (cls[sa[i]] != cls[sa[i-1]] ||
                        cls[(sa[i] + d) % n] != cls[(sa[i-1] + d) % n]))
 ↪  {
                ++n_cls;
            }
            new_cls[sa[i]] = n_cls;
        }
        ++n_cls;
        forn(i, n) cls[i] = new_cls[i];
    }

    // cls is also a reverse permutation of sa if a string is not cyclic
    // (i.e. a position of i-th lexicographical suffix)
    int val = 0;
    forn(i, n) {
        if (val) --val;
        if (cls[i] == n-1) continue;
        int j = sa[cls[i] + 1];
        while (i + val != n && j + val != n && s[i+val] == s[j+val])
 ↪  ++val;
        lcp[cls[i]] = val;
    }
}

int main() {
    cin >> s;
    s += '$';
    n = s.length();
    build();
    forn(i, n) {
        cout << s.substr(sa[i]) << endl;
        cout << lcp[i] << endl;
    }
}
```

12 algo/strings/ukkonen.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sz(x) ((int) (x).size())
#define forn(i,n) for (int i = 0; i < int(n); ++i)
const int inf = int(1e9) + int(1e5);

string s;
const int alpha = 26;

namespace SuffixTree {
    struct Node {
        Node *to[alpha];
        Node *lnk, *par;
        int l, r;

        Node(int l, int r): l(l), r(r) {
            memset(to, 0, sizeof(to));
            lnk = par = 0;
        }
    };

    Node *root, *blank, *cur;
    int pos;

    void init() {
        root = new Node(0, 0);
        blank = new Node(0, 0);
        forn (i, alpha)
            blank->to[i] = root;
        root->lnk = root->par = blank->lnk = blank->par = blank;
        cur = root;
        pos = 0;
    }

    int at(int id) {
        return s[id];
    }

    void goDown(int l, int r) {
        if (l >= r)
            return;
        if (pos == cur->r) {
            int c = at(l);
            assert(cur->to[c]);
            cur = cur->to[c];
            pos = min(cur->r, cur->l + 1);
            ++l;
        } else {
            int delta = min(r - l, cur->r - pos);
            l += delta;
            pos += delta;
        }
        goDown(l, r);
    }

    void goUp() {
        if (pos == cur->r && cur->lnk) {
            cur = cur->lnk;
            pos = cur->r;
            return;
        }
        int l = cur->l, r = pos;
        cur = cur->par->lnk;
        pos = cur->r;
        goDown(l, r);
    }

    void setParent(Node *a, Node *b) {
        assert(a);
        a->par = b;
        if (b)
            b->to[at(a->l)] = a;
    }

    void addLeaf(int id) {
        Node *x = new Node(id, inf);
        setParent(x, cur);
    }

    void splitNode() {
        assert(pos != cur->r);
        Node *mid = new Node(cur->l, pos);
        setParent(mid, cur->par);
        cur->l = pos;
        setParent(cur, mid);
        cur = mid;
    }

    bool canGo(int c) {
        if (pos == cur->r)
            return cur->to[c];
        return at(pos) == c;
    }

    void fixLink(Node *&bad, Node *newBad) {
        if (bad)
            bad->lnk = cur;
        bad = newBad;
    }

    void addCharOnPos(int id) {
        Node *bad = 0;
        while (!canGo(at(id))) {
            if (cur->r != pos) {
                splitNode();
                fixLink(bad, cur);
                bad = cur;
            } else {
                fixLink(bad, 0);
            }
            addLeaf(id);
            goUp();
        }
        fixLink(bad, 0);
        goDown(id, id + 1);
    }

    int cnt(Node *u, int ml) {
        if (!u)
            return 0;
        int res = min(ml, u->r) - u->l;
        forn (i, alpha)
            res += cnt(u->to[i], ml);
        return res;
    }

    void build(int l) {
        init();
        forn (i, l)
            addCharOnPos(i);
    }
};

int main() {
    cin >> s;
    SuffixTree::build(s.size());
}
```

13 algo/structures/convex_hull_trick.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i,n) for (int i = 0; i < int(n); ++i)
typedef long long ll;
const ll infl = ll(2e18) + ll(2e15);

const int maxn = 4004;

/*
    WARNING!!!
    - finds maximum of A*x+B
    - double check max coords for int/long long overflow
    - set min x query in put function
    - add lines with non-descending A coefficient
*/
struct FastHull {
    int a[maxn];
    ll b[maxn];
    ll p[maxn];
    int c;

    FastHull(): c(0) {}

    ll get(int x) {
        if (c == 0)
            return -infl;
        int pos = upper_bound(p, p + c, x) - p - 1;
        assert(pos >= 0);
        return (ll) a[pos] * x + b[pos];
    }

    ll divideCeil(ll p, ll q) {
        assert(q > 0);
        if (p >= 0)
            return (p + q - 1) / q;
        return -((-p) / q);
    }

    void put(int A, ll B) {
        while (c > 0) {
            if (a[c - 1] == A && b[c - 1] >= B)
                return;
            ll pt = p[c - 1];
            if (a[c - 1] * pt + b[c - 1] < A * pt + B) {
                --c;
                continue;
            }
            ll q = A - a[c - 1];
            ll np = divideCeil(b[c - 1] - B, q);
            p[c] = np;
            a[c] = A;
            b[c] = B;
            ++c;
            return;
        }
        if (c == 0) {
            a[c] = A, b[c] = B;
            p[c] = -1e9; //min x query
            ++c;
            return;
        }
    }

};

struct SlowHull {
    vector<pair<int, ll>> v;

    void put(int a, ll b) {
        v.emplace_back(a, b);
    }

    ll get(ll x) {
        ll best = -infl;
        for (auto p: v)
            best = max(best, p.first * x + p.second);
        return best;
    }
};

int main() {
    FastHull hull1;
    SlowHull hull2;
    vector<int> as;
    forn (ii, 10000)
        as.push_back(rand() % int(1e8));
    sort(as.begin(), as.end());
    forn (ii, 10000) {
        int b = rand() % int(1e8);
        hull1.put(as[ii], b);
        hull2.put(as[ii], b);
        int x = rand() % int(2e8 + 1) - int(1e8);
        assert(hull1.get(x) == hull2.get(x));
    }
}
```

14 algo/structures/ordered_set.cpp

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

typedef __gnu_pbds::tree<int, __gnu_pbds::null_type, std::less<int>,
                __gnu_pbds::rb_tree_tag,
 ↪   __gnu_pbds::tree_order_statistics_node_update> oset;

#include <iostream>

int main() {
    oset X;
    X.insert(1);
    X.insert(2);
    X.insert(4);
    X.insert(8);
    X.insert(16);

    std::cout << *X.find_by_order(1) << std::endl; // 2
    std::cout << *X.find_by_order(2) << std::endl; // 4
    std::cout << *X.find_by_order(4) << std::endl; // 16
    std::cout << std::boolalpha << (end(X)==X.find_by_order(6)) <<
 ↪   std::endl; // true

    std::cout << X.order_of_key(-5) << std::endl;  // 0
    std::cout << X.order_of_key(1) << std::endl;   // 0
    std::cout << X.order_of_key(3) << std::endl;   // 2
    std::cout << X.order_of_key(4) << std::endl;   // 2
    std::cout << X.order_of_key(400) << std::endl; // 5
}
```

15 algo/structures/splay.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for (int i = 0; i < (int)(n); ++i)

const int maxn = 100500;

struct node;
void updson(node* p, node* v, node* was);

struct node {
    int val;
    node *l, *r, *p;
    node() {}
    node(int val) : val(val), l(r=p=NULL) {}

    bool isRoot() const { return !p; }
    bool isRight() const { return p && p->r == this; }
    bool isLeft() const { return p && p->l == this; }
    void setLeft(node* t) {
        if (t) t->p = this;
        l = t;
    }
    void setRight(node *t) {
        if (t) t->p = this;
        r = t;
    }
};

void updson(node *p, node *v, node *was) {
    if (p) {
        if (p->l == was) p->l = v;
        else p->r = v;
    }
    if (v) v->p = p;
}

void rightRotate(node *v) {
    assert(v && v->l);
    node *u = v->l;
    node *p = v->p;
    v->setLeft(u->r);
    u->setRight(v);
    updson(p, u, v);
}

void leftRotate(node *v) {
    assert(v && v->r);
    node *u = v->r;
    node *p = v->p;
    v->setRight(u->l);
    u->setLeft(v);
    updson(p, u, v);
}

void splay(node *v) {
    while (v->p) {
        if (!v->p->p) {
            if (v->isLeft()) rightRotate(v->p);
            else leftRotate(v->p);
        } else if (v->isLeft() && v->p->isLeft()) {
            rightRotate(v->p->p);
            rightRotate(v->p);
        } else if (v->isRight() && v->p->isRight()) {
            leftRotate(v->p->p);
            leftRotate(v->p);
        } else if (v->isLeft()) {
            rightRotate(v->p);
            leftRotate(v->p);
        } else {
            leftRotate(v->p);
            rightRotate(v->p);
        }
    }
    v->p = NULL;
}

node *insert(node *t, node *n) {
    if (!t) return n;
    int x = n->val;
    while (true) {
        if (x < t->val) {
            if (t->l) {
                t = t->l;
            } else {
                t->setLeft(n);
                t = t->l;
                break;
            }
        } else {
            if (t->r) {
                t = t->r;
            } else {
                t->setRight(n);
                t = t->r;
                break;
            }
        }
    }
    splay(t);
    return t;
}

node *insert(node *t, int x) {
    return insert(t, new node(x));
}

int main() {
    node *t = NULL;
    forn(i, 1000000) {
        int x = rand();
        t = insert(t, x);
    }
    return 0;
}
```

16 algo/structures/treap.cpp

```
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
const int maxn = 100500;

struct node {
    int x, y;
    node *l, *r;
    node(int x) : x(x), y(rand()), l(r=NULL) {}
};

void split(node *t, node *&l, node *&r, int x) {
    if (!t) return (void)(l=r=NULL);
    if (x <= t->x) {
        split(t->l, l, t->l, x), r = t;
    } else {
        split(t->r, t->r, r, x), l = t;
    }
}

node *merge(node *l, node *r) {
    if (!l) return r;
    if (!r) return l;
    if (l->y > r->y) {
        l->r = merge(l->r, r);
        return l;
    } else {
        r->l = merge(l, r->l);
        return r;
    }
}

node *insert(node *t, node *n) {
    node *l, *r;
    split(t, l, r, n->x);
    return merge(l, merge(n, r));
}

node *insert(node *t, int x) {
    return insert(t, new node(x));
}

node *fast_insert(node *t, node *n) {
    if (!t) return n;
    node *root = t;
    while (true) {
        if (n->x < t->x) {
            if (!t->l || t->l->y < n->y) {
                split(t->l, n->l, n->r, n->x), t->l = n;
                break;
            } else {
                t = t->l;
            }
        } else {
            if (!t->r || t->r->y < n->y) {
                split(t->r, n->l, n->r, n->x), t->r = n;
                break;
            } else {
                t = t->r;
            }
        }
    }
    return root;
}

node *fast_insert(node *t, int x) {
    return fast_insert(t, new node(x));
}

int main() {
    node *t = NULL;
    forn(i, 1000000) {
        int x = rand();
        t = fast_insert(t, x);
    }
}
```