

Содержание

1	Strategy.txt	1
2	algo/flows/globalcut.cpp	2
3	algo/flows/hungary.cpp	2
4	algo/flows/mincost.cpp	3
5	algo/geometry/primitives.cpp	4
6	algo/graphs/2sat.cpp	5
7	algo/math/fft_recursive.cpp	6
8	algo/math/golden_search.cpp	7
9	algo/math/numbers.txt	7
10	algo/strings/automaton.cpp	8
11	algo/strings/suffix_array.cpp	8
12	algo/strings/ukkonen.cpp	9
13	algo/structures/convex_hull_trick.cpp	10
14	algo/structures/ordered_set.cpp	11
15	algo/structures/splay.cpp	11
16	algo/structures/treap.cpp	12

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 `main`, 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

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i,n) for (int i = 0; i < int(n); ++i)
4const int inf = 1e9 + 1e5;
5
6const int maxn = 505;
7namespace StoerWagner {
8    int g[maxn][maxn];
9    int dist[maxn];
10    bool used[maxn];
11    int n;
12
13    void addEdge(int u, int v, int c) {
14        g[u][v] += c;
15        g[v][u] += c;
16    }
17
18    int run() {
19        vector<int> vertices;
20        forn (i, n)
21            vertices.push_back(i);
22        int mincut = inf;
23        while (vertices.size() > 1) {
24            int u = vertices[0];
25            for (auto v: vertices) {
26                used[v] = false;
27                dist[v] = g[u][v];
28            }
29            used[u] = true;
30            forn (ii, vertices.size() - 2) {
31                for (auto v: vertices)
32                    if (!used[v])
33                        if (used[u] || dist[v] > dist[u])
34                            u = v;
35                used[u] = true;
36                for (auto v: vertices)
37                    if (!used[v])
38                        dist[v] += g[u][v];
39            }
40            int t = -1;
41            for (auto v: vertices)
42                if (!used[v])
43                    t = v;
44            assert(t != -1);
45            mincut = min(mincut, dist[t]);
46            vertices.erase(find(vertices.begin(), vertices.end(), t));
47            for (auto v: vertices)
48                addEdge(u, v, g[v][t]);
49        }
50        return mincut;
51    }
52};
53
54int main() {
55    StoerWagner::n = 4;
56    StoerWagner::addEdge(0, 1, 5);
57    StoerWagner::addEdge(2, 3, 5);
58    StoerWagner::addEdge(1, 2, 4);
59    cerr << StoerWagner::run() << '\n';
60}

```

3 algo/flows/hungary.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i,n) for (int i = 0; i < int(n); ++i)
4const int inf = 1e9 + 1e5;
5
6// left half is the smaller one
7namespace Hungary {
8    const int maxn = 505;
9    int a[maxn][maxn];
10    int p[2][maxn];
11    int match[maxn];
12    bool used[maxn];
13    int from[maxn];
14    int mind[maxn];
15    int n, m;
16
17    int hungary(int v) {
18        used[v] = true;
19        int u = match[v];
20        int best = -1;
21        forn (i, m + 1) {
22            if (used[i])
23                continue;
24            int nw = a[u][i] - p[0][u] - p[1][i];
25            if (nw <= mind[i]) {
26                mind[i] = nw;
27                from[i] = v;
28            }
29            if (best == -1 || mind[best] > mind[i])
30                best = i;
31        }
32        v = best;
33        int delta = mind[best];
34        forn (i, m + 1) {
35            if (used[i]) {
36                p[1][i] -= delta;
37                p[0][match[i]] += delta;
38            } else
39                mind[i] -= delta;
40        }
41        if (match[v] == -1)
42            return v;
43        return hungary(v);
44    }
45
46    void check() {
47        int edges = 0, res = 0;
48        forn (i, m)
49            if (match[i] != -1) {
50                ++edges;
51                assert(p[0][match[i]] + p[1][i] == a[match[i]][i]);
52                res += a[match[i]][i];
53            } else
54                assert(p[1][i] == 0);
55        assert(res == -p[1][m]);
56        forn (i, n) forn (j, m)
57            assert(p[0][i] + p[1][j] <= a[i][j]);
58    }
59
60    int run() {
61        forn (i, n)
62            p[0][i] = 0;
63        forn (i, m + 1) {
64            p[1][i] = 0;
65            match[i] = -1;
66        }
67        forn (i, n) {
68            match[m] = i;
69            fill(used, used + m + 1, false);
70            fill(mind, mind + m + 1, inf);
71            fill(from, from + m + 1, -1);
72            int v = hungary(m);
73            while (v != m) {
74                int w = from[v];
75                match[v] = match[w];
76                v = w;
77            }
78        }
79        check();
80        return -p[1][m];
81    }
82};
83
84int main() {
85    int n = 300, m = 500;
86    Hungary::n = n, Hungary::m = m;
87    forn (i, n) forn (j, m) Hungary::a[i][j] = rand() % 200001 - 100000;
88    cerr << Hungary::run() << "\n";
89}
90

```

4 algo/flows/mincost.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3typedef long long ll;
4#define forn(i,n) for (int i = 0; i < int(n); ++i)
5
6namespace MinCost {
7    const ll infc = 1e12;
8
9    struct Edge {
10        int to;
11        ll c, f, cost;
12
13        Edge(int to, ll c, ll cost): to(to), c(c), f(0), cost(cost) {}
14    };
15
16    int N, S, T;
17    int totalFlow;
18    ll totalCost;
19    const int maxn = 505;
20    vector<Edge> edge;
21    vector<int> g[maxn];
22
23    void addEdge(int u, int v, ll c, ll cost) {
24        g[u].push_back(edge.size());
25        edge.emplace_back(v, c, cost);
26        g[v].push_back(edge.size());
27        edge.emplace_back(u, 0, -cost);
28    }
29
30    ll dist[maxn];
31    int fromEdge[maxn];
32
33    bool inQueue[maxn];
34    bool fordBellman() {
35        forn (i, N)
36            dist[i] = infc;
37        dist[S] = 0;
38        inQueue[S] = true;
39        vector<int> q;
40        q.push_back(S);
41        for (int ii = 0; ii < int(q.size()); ++ii) {
42            int u = q[ii];
43            inQueue[u] = false;
44            for (int e: g[u]) {
45                if (edge[e].f == edge[e].c)
46                    continue;
47                int v = edge[e].to;
48                ll nw = edge[e].cost + dist[u];
49                if (nw >= dist[v])
50                    continue;
51                dist[v] = nw;
52                fromEdge[v] = e;
53                if (!inQueue[v]) {
54                    inQueue[v] = true;
55                    q.push_back(v);
56                }
57            }
58        }
59        return dist[T] != infc;
60    }
61
62    ll pot[maxn];
63    bool dikstra() {
64        priority_queue<pair<ll, int>, vector<pair<ll, int>>,
65        ↪ greater<pair<ll, int>>> q;
66        forn (i, N)
67            dist[i] = infc;
68        dist[S] = 0;
69        q.emplace(dist[S], S);
70        while (!q.empty()) {
71            int u = q.top().second;
72            ll cdist = q.top().first;
73            q.pop();
74            if (cdist != dist[u])
75                continue;
76            for (int e: g[u]) {
77                int v = edge[e].to;
78                if (edge[e].c == edge[e].f)
79                    continue;
80                ll w = edge[e].cost + pot[u] - pot[v];
81                assert(w >= 0);
82                ll ndist = w + dist[u];
83                if (ndist >= dist[v])
84                    continue;
85                dist[v] = ndist;
86                fromEdge[v] = e;
87                q.emplace(dist[v], v);
88            }
89        }
90    }
91
92    int flow() {
93        int f = 0;
94        while (fordBellman()) {
95            ll c = 0;
96            for (int i: fromEdge)
97                c += dist[i];
98            f++;
99            totalCost += c;
100            totalFlow++;
101        }
102        return f;
103    }
104
105    int cost() {
106        int c = 0;
107        while (dikstra()) {
108            ll w = 0;
109            for (int i: fromEdge)
110                w += dist[i];
111            c += w;
112            pot[fromEdge[S]] = 0;
113            for (int i: fromEdge)
114                pot[i] = pot[fromEdge[i]] + dist[i];
115        }
116        return c;
117    }
118
119    int solve() {
120        int f = flow();
121        if (f < T)
122            return -1;
123        return cost();
124    }
125
126    int main() {
127        int n, m;
128        cin >> n >> m;
129        S = 0, T = n - 1;
130        forn (i, m) {
131            int u, v, c, f;
132            cin >> u >> v >> c >> f;
133            addEdge(u, v, c, f);
134        }
135        int f = solve();
136        if (f < T)
137            cout << -1 << endl;
138        else
139            cout << totalCost << endl;
140    }
141}

```

```

89     }
90     if (dist[T] == infc)
91         return false;
92     forn (i, N) {
93         if (dist[i] == infc)
94             continue;
95         pot[i] += dist[i];
96     }
97     return true;
98 }
99
100 bool push() {
101     //2 variants
102     //if (!fordBellman())
103     if (!dikstra())
104         return false;
105     ++totalFlow;
106     int u = T;
107     while (u != S) {
108         int e = fromEdge[u];
109         totalCost += edge[e].cost;
110         edge[e].f++;
111         edge[e ^ 1].f--;
112         u = edge[e ^ 1].to;
113     }
114     return true;
115 }
116};
117
118int main() {
119     MinCost::N = 3, MinCost::S = 1, MinCost::T = 2;
120     MinCost::addEdge(1, 0, 3, 5);
121     MinCost::addEdge(0, 2, 4, 6);
122     while (MinCost::push());
123     cout << MinCost::totalFlow << ' ' << MinCost::totalCost << '\n'; //3
124 }

```

5 algo/geometry/primitives.cpp

```

1#include <bits/stdc++.h>
2#define forn(i, n) for (int i = 0; i < int(n); ++i)
3using namespace std;
4typedef long double ld;
5
6const ld eps = 1e-9;
7
8bool eq(ld a, ld b) { return fabs1(a - b) < eps; }
9bool le(ld a, ld b) { return b - a > -eps; }
10bool ge(ld a, ld b) { return a - b > -eps; }
11bool lt(ld a, ld b) { return b - a > eps; }
12bool gt(ld a, ld b) { return a - b > eps; }
13ld sqr(ld x) { return x * x; }
14
15inline void gassert(bool expr) {
16    #ifdef LOCAL
17        assert(expr);
18    #endif
19}
20
21struct pt {
22    ld x, y;
23
24    pt operator+(const pt &p) const { return pt{x + p.x, y + p.y}; }
25    pt operator-(const pt &p) const { return pt{x - p.x, y - p.y}; }
26    ld operator*(const pt &p) const { return x * p.x + y * p.y; }
27    ld operator%(const pt &p) const { return x * p.y - y * p.x; }
28
29    pt operator*(const ld &a) const { return pt{x * a, y * a}; }
30    pt operator/(const ld &a) const { gassert(!eq(a, 0)); return pt{x /
    ↪ a, y / a}; }
31    void operator==(const ld &a) { x == a, y == a; }
32    void operator/=(const ld &a) { gassert(!eq(a, 0)); x /= a, y /= a; }
33
34    bool operator<(const pt &p) const {
35        if (eq(x, p.x)) return lt(y, p.y);
36        return x < p.x;
37    }
38
39    bool operator==(const pt &p) const { return eq(x, p.x) && eq(y, p.y); }
40    ↪ bool operator!=(const pt &p) const { return !(this == p); }
41
42    pt rot() { return pt{-y, x}; }
43    ld abs() const { return hypot1(x, y); }
44    ld abs2() const { return x * x + y * y; }
45};
46
47istream &operator>>(istream &in, pt &p) { return in >> p.x >> p.y; }
48ostream &operator<<(ostream &out, const pt &p) { return out << p.x << ' '
    ↪ << p.y; }
49
50//WARNING! do not forget to normalize vector (a,b)
51struct line {
52    ld a, b, c;
53
54    line(pt p1, pt p2) {
55        gassert(p1 != p2);
56        pt n = (p2 - p1).rot();
57        n /= n.abs();
58        a = n.x, b = n.y;
59        c = -(n * p1);
60    }
61
62    line(ld _a, ld _b, ld _c): a(_a), b(_b), c(_c) {
63        ld d = pt{a, b}.abs();
64        gassert(!eq(d, 0));
65        a /= d, b /= d, c /= d;
66    }
67
68    ld signedDist(pt p) {
69        return p * pt{a, b} + c;
70    }
71};
72
73ld pointSegmentDist(pt p, pt a, pt b) {
74    ld res = min((p - a).abs(), (p - b).abs());
75    if (a != b && ge((p - a) * (b - a), 0) && ge((p - b) * (a - b), 0))
76        res = min(res, fabs1((p - a) % (b - a)) / (b - a).abs());
77    return res;
78}
79
80pt linesIntersection(line l1, line l2) {
81    ld D = l1.a * l2.b - l1.b * l2.a;
82    if (eq(D, 0)) {
83        if (eq(l1.c, l2.c)) {
84            //equal lines
85        } else {
86            //no intersection

```

```

87    }
88    }
89    ld dx = -l1.c * l2.b + l1.b * l2.c;
90    ld dy = -l1.a * l2.c + l1.c * l2.a;
91    pt res{dx / D, dy / D};
92    gassert(eq(l1.signedDist(res), 0));
93    gassert(eq(l2.signedDist(res), 0));
94    return res;
95}
96
97bool pointInsideSegment(pt p, pt a, pt b) {
98    if (!eq((p - a) % (b - a), 0))
99        return false;
100    return le((a - p) * (b - p), 0);
101}
102
103bool checkSegmentIntersection(pt a, pt b, pt c, pt d) {
104    if (eq((a - b) % (c - d), 0)) {
105        if (pointInsideSegment(a, c, d) || pointInsideSegment(b, c, d) ||
106            ↪ pointInsideSegment(c, a, b) || pointInsideSegment(d, a,
107                ↪ b)) {
108            //intersection of parallel segments
109            return true;
110        }
111        return false;
112    }
113    ld s1, s2;
114
115    s1 = (c - a) % (b - a);
116    s2 = (d - a) % (b - a);
117    if (gt(s1, 0) && gt(s2, 0))
118        return false;
119    if (lt(s1, 0) && lt(s2, 0))
120        return false;
121
122    swap(a, c), swap(b, d);
123
124    s1 = (c - a) % (b - a);
125    s2 = (d - a) % (b - a);
126    if (gt(s1, 0) && gt(s2, 0))
127        return false;
128    if (lt(s1, 0) && lt(s2, 0))
129        return false;
130
131    return true;
132}
133
134//WARNING! run checkSegmentIntersection before and process parallel case
135↪ manually
136pt segmentsIntersection(pt a, pt b, pt c, pt d) {
137    ld S = (b - a) % (d - c);
138    ld s1 = (c - a) % (d - a);
139    return a + (b - a) / S * s1;
140}
141
142vector<pt> circlesIntersection(pt a, ld r1, pt b, ld r2) {
143    ld d2 = (a - b).abs2();
144    ld d = (a - b).abs();
145
146    if (a == b && eq(r1, r2)) {
147        //equal circles
148    }
149    if (lt(sqr(r1 + r2), d2) || gt(sqr(r1 - r2), d2)) {
150        //empty intersection
151        return {};
152    }
153    int num = 2;
154    if (eq(sqr(r1 + r2), d2) || eq(sqr(r1 - r2), d2))
155        num = 1;
156    ld cosa = (sqr(r1) + d2 - sqr(r2)) / ld(2 * r1 * d);
157    ld oh = cosa * r1;
158    pt h = a + ((b - a) / d * oh);
159    if (num == 1)
160        return {h};
161    ld hp = sqrt1(max(0.L, 1 - cosa * cosa)) * r1;
162
163    pt w = ((b - a) / d * hp).rot();
164    return {h + w, h - w};
165}
166//a is circle center, p is point
167vector<pt> circleTangent(pt a, ld r, pt p) {
168    ld d2 = (a - p).abs2();
169    ld d = (a - p).abs();
170
171    if (gt(sqr(r), d2)) {
172        //no tangents
173        return {};
174    }
175    if (eq(sqr(r), d2)) {

```

```

176 //point lies on circle - one tangent
177 return {p};
178 }
179
180 pt B = p - a;
181 pt H = B * sqr(r) / d2;
182 ld h = sqrtl(d2 - sqr(r)) * ld(r) / d;
183 pt w = (B / d * h).rot();
184 H = H + a;
185 return {H + w, H - w};
186 }
187
188 vector<pt> lineCircleIntersection(line l, pt a, ld r) {
189     ld d = l.signedDist(a);
190     if (gt(fabs1(d), r))
191         return {};
192     pt h = a - pt{l.a, l.b} * d;
193     if (eq(fabs1(d), r))
194         return {h};
195     pt w = pt{l.a, l.b}.rot() * sqrtl(max<ld>(0, sqr(r) - sqr(d)));
196     return {h + w, h - w};
197 }
198
199 int main() {
200     cout.precision(10);
201     cout << fixed;
202 }

```

6 algo/graphs/2sat.cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forn(i,n) for (int i = 0; i < int(n); ++i)
4 const int maxn = 200100; //2 * number of variables
5
6 namespace TwoSAT {
7     int n; //number of variables
8     bool used[maxn];
9     vector<int> g[maxn];
10    vector<int> gr[maxn];
11    int comp[maxn];
12    int res[maxn];
13
14    void addEdge(int u, int v) { //u or v
15        g[u].push_back(v ^ 1);
16        g[v].push_back(u ^ 1);
17        gr[u ^ 1].push_back(v);
18        gr[v ^ 1].push_back(u);
19    }
20
21    vector<int> ord;
22    void dfs1(int u) {
23        used[u] = true;
24        for (int v: g[u]) {
25            if (used[v])
26                continue;
27            dfs1(v);
28        }
29        ord.push_back(u);
30    }
31
32    int COL = 0;
33    void dfs2(int u) {
34        used[u] = true;
35        comp[u] = COL;
36        for (int v: gr[u]) {
37            if (used[v])
38                continue;
39            dfs2(v);
40        }
41    }
42
43    void mark(int u) {
44        res[u / 2] = u % 2;
45        used[u] = true;
46        for (int v: g[u]) {
47            if (used[v])
48                continue;
49            mark(v);
50        }
51    }
52
53    bool run() {
54        fill(res, res + 2 * n, -1);
55        fill(used, used + 2 * n, false);
56        forn(i, 2 * n)
57            if (!used[i])
58                dfs1(i);
59        reverse(ord.begin(), ord.end());
60        assert((int) ord.size() == (2 * n));
61        fill(used, used + 2 * n, false);
62        for (int u: ord) if (!used[u]) {
63            dfs2(u);
64            ++COL;
65        }
66        forn(i, n)
67            if (comp[i * 2] == comp[i * 2 + 1])
68                return false;
69
70        reverse(ord.begin(), ord.end());
71        fill(used, used + 2 * n, false);
72        for (int u: ord) {
73            if (res[u / 2] != -1) {
74                continue;
75            }
76            mark(u);
77        }
78        return true;
79    }
80 }
81
82 int main() {
83     TwoSAT::n = 2;
84     TwoSAT::addEdge(0, 2); //x or y
85     TwoSAT::addEdge(0, 3); //x or !y
86     TwoSAT::addEdge(3, 3); //!y or !y
87     assert(TwoSAT::run());
88     cout << TwoSAT::res[0] << ' ' << TwoSAT::res[1] << '\n'; //1 0
89 }
90

```

7 algo/math/fft_recursive.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
4typedef long long i64;
5
6typedef double ld;
7
8struct base {
9    ld re, im;
10    base(){}
11    base(ld re) : re(re), im(0) {}
12    base(ld re, ld im) : re(re), im(im) {}
13
14    base operator+(const base& o) const { return {re+o.re, im+o.im}; }
15    base operator-(const base& o) const { return {re-o.re, im-o.im}; }
16    base operator*(const base& o) const {
17        return {
18            re*o.re - im*o.im,
19            re*o.im + im*o.re
20        };
21    }
22};
23
24const int sz = 1<<20;
25
26int revb[sz];
27vector<base> ang[21];
28
29void init(int n) {
30    int lg = 0;
31    while ((1<<lg) != n) {
32        ++lg;
33    }
34    forn(i, n) {
35        revb[i] = (revb[i>>1]>>1)^((i&1)<<(lg-1));
36    }
37
38    ld e = M_PI * 2 / n;
39    ang[lg].resize(n);
40    forn(i, n) {
41        ang[lg][i] = { cos(e * i), sin(e * i) };
42    }
43
44    for (int k = lg - 1; k >= 0; --k) {
45        ang[k].resize(1 << k);
46        forn(i, 1<<k) {
47            ang[k][i] = ang[k+1][i*2];
48        }
49    }
50}
51
52void fft_rec(base *a, int lg, bool rev) {
53    if (lg == 0) {
54        return;
55    }
56    int len = 1 << (lg - 1);
57    fft_rec(a, lg-1, rev);
58    fft_rec(a+len, lg-1, rev);
59
60    forn(i, len) {
61        base w = ang[lg][i];
62        if (rev) w.im *= -1;
63        base u = a[i];
64        base v = a[i+len] * w;
65        a[i] = u + v;
66        a[i+len] = u - v;
67    }
68}
69
70void fft(base *a, int n, bool rev) {
71    forn(i, n) {
72        int j = revb[i];
73        if (i < j) swap(a[i], a[j]);
74    }
75    int lg = 0;
76    while ((1<<lg) != n) {
77        ++lg;
78    }
79    fft_rec(a, lg, rev);
80    if (rev) forn(i, n) {
81        a[i] = a[i] * (1.0 / n);
82    }
83}
84
85const int maxn = 1050000;
86
87int n;
88base a[maxn];
89base b[maxn];

```

```

90
91void test() {
92    int n = 1<<19;
93    mt19937 rr(55);
94    forn(i, n) a[i] = rr() % 10000;
95    forn(j, n) b[j] = rr() % 10000;
96
97    int N = 1;
98    while (N < 2*n) N *= 2;
99
100    clock_t start = clock();
101    init(N);
102    cerr << "init time: " << (clock()-start) / 1000 << " ms" << endl;
103    fft(a, N, 0);
104    fft(b, N, 0);
105    forn(i, N) a[i] = a[i] * b[i];
106    fft(a, N, 1);
107    clock_t end = clock();
108
109    ld err = 0;
110    forn(i, N) {
111        err = max(err, (ld)fabs1(a[i].im));
112        err = max(err, (ld)fabs1(a[i].re - (i64(a[i].re + 0.5))));
113    }
114
115    cerr << "Time: " << (end - start) / 1000 << " ms, err = " << err <<
    ↵ endl;
116}
117
118int main() {
119    test();
120}

```

8 algo/math/golden_search.cpp

```
1#include <bits/stdc++.h>
2typedef long double ld;
3#define forn(i, n) for (int i = 0; i < int(n); ++i)
4
5ld f(ld x) {
6    return 5 * x * x + 100 * x + 1; // -10 is minimum
7}
8
9ld goldenSearch(ld l, ld r) {
10    ld phi = (1 + sqrt(5)) / 2;
11    ld resphi = 2 - phi;
12    ld x1 = l + resphi * (r - l);
13    ld x2 = r - resphi * (r - l);
14    ld f1 = f(x1);
15    ld f2 = f(x2);
16    forn(iter, 60) {
17        if (f1 < f2) {
18            r = x2;
19            x2 = x1;
20            f2 = f1;
21            x1 = l + resphi * (r - l);
22            f1 = f(x1);
23        } else {
24            l = x1;
25            x1 = x2;
26            f1 = f2;
27            x2 = r - resphi * (r - l);
28            f2 = f(x2);
29        }
30    }
31    return (x1 + x2) / 2;
32}
33
34int main() {
35    std::cout << goldenSearch(-100, 100) << '\n';
36}
```

9 algo/math/numbers.txt

Simpson's numerical integration: integral from a to

$$\hookrightarrow \int_a^b f(x) dx = (b - a) / 6 * (f(a) + 4 * f((a + b) / 2) + f(b))$$

\hookrightarrow

10 algo/strings/automaton.cpp

```

1//real 4m27.689s
2#include <bits/stdc++.h>
3using namespace std;
4#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
5
6const int maxn = 100500;
7
8int t[maxn][26], lnk[maxn], len[maxn];
9int sz;
10int last;
11
12void init() {
13    sz = 3;
14    last = 1;
15    forn(i, 26) t[2][i] = 1;
16    len[2] = -1;
17    lnk[1] = 2;
18}
19
20void addchar(int c) {
21    int nlast = sz++;
22    len[nlast] = len[last] + 1;
23    int p = last;
24    for (; !t[p][c]; p = lnk[p]) {
25        t[p][c] = nlast;
26    }
27    int q = t[p][c];
28    if (len[p] + 1 == len[q]) {
29        lnk[nlast] = q;
30    } else {
31        int clone = sz++;
32        len[clone] = len[p] + 1;
33        lnk[clone] = lnk[q];
34        lnk[q] = lnk[nlast] = clone;
35        forn(i, 26) t[clone][i] = t[q][i];
36        for (; t[p][c] == q; p = lnk[p]) {
37            t[p][c] = clone;
38        }
39    }
40    last = nlast;
41}
42
43bool check(const string& s) {
44    int v = 1;
45    for (int c: s) {
46        c -= 'a';
47        if (!t[v][c]) return false;
48        v = t[v][c];
49    }
50    return true;
51}
52
53int main() {
54    string s;
55    cin >> s;
56    init();
57    for (int i: s) {
58        addchar(i - 'a');
59    }
60    forn(i, s.length()) {
61        assert(check(s.substr(i)));
62    }
63    cout << sz << endl;
64    return 0;
65}

```

11 algo/strings/suffix_array.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
4
5const int maxn = 100500;
6
7string s;
8int n;
9int sa[maxn], new_sa[maxn], cls[maxn], new_cls[maxn], cnt[maxn],
    ↪ lcp[maxn];
10int n_cls;
11
12void build() {
13    n_cls = 256;
14    forn(i, n) {
15        sa[i] = i;
16        cls[i] = s[i];
17    }
18    for (int d = 0; d < n; d = d ? d*2 : 1) {
19
20        forn(i, n) new_sa[i] = (sa[i] - d + n) % n;
21        forn(i, n_cls) cnt[i] = 0;
22        forn(i, n) ++cnt[cls[i]];
23        forn(i, n_cls) cnt[i+1] += cnt[i];
24        for (int i = n-1; i >= 0; --i) sa[--cnt[cls[new_sa[i]]]] =
    ↪ new_sa[i];
25
26        n_cls = 0;
27        forn(i, n) {
28            if (i && (cls[sa[i]] != cls[sa[i-1]] ||
29                ↪ cls[(sa[i] + d) % n] != cls[(sa[i-1] + d) % n]))
    ↪ {
30                ++n_cls;
31            }
32            new_cls[sa[i]] = n_cls;
33        }
34        ++n_cls;
35        forn(i, n) cls[i] = new_cls[i];
36    }
37
38    // cls is also a reverse permutation of sa if a string is not cyclic
39    // (i.e. a position of i-th lexicographical suffix)
40    int val = 0;
41    forn(i, n) {
42        if (val) --val;
43        if (cls[i] == n-1) continue;
44        int j = sa[cls[i] + 1];
45        while (i + val != n && j + val != n && s[i+val] == s[j+val])
    ↪ ++val;
46        lcp[cls[i]] = val;
47    }
48}
49
50int main() {
51    cin >> s;
52    s += '$';
53    n = s.length();
54    build();
55    forn(i, n) {
56        cout << s.substr(sa[i]) << endl;
57        cout << lcp[i] << endl;
58    }
59}

```


12 algo/strings/ukkonen.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define sz(x) ((int) (x).size())
4#define forn(i,n) for (int i = 0; i < int(n); ++i)
5const int inf = int(1e9) + int(1e5);
6
7string s;
8const int alpha = 26;
9
10namespace SuffixTree {
11    struct Node {
12        Node *to[alpha];
13        Node *lnk, *par;
14        int l, r;
15
16        Node(int l, int r): l(l), r(r) {
17            memset(to, 0, sizeof(to));
18            lnk = par = 0;
19        }
20    };
21
22    Node *root, *blank, *cur;
23    int pos;
24
25    void init() {
26        root = new Node(0, 0);
27        blank = new Node(0, 0);
28        forn (i, alpha)
29            blank->to[i] = root;
30        root->lnk = root->par = blank->lnk = blank->par = blank;
31        cur = root;
32        pos = 0;
33    }
34
35    int at(int id) {
36        return s[id];
37    }
38
39    void goDown(int l, int r) {
40        if (l >= r)
41            return;
42        if (pos == cur->r) {
43            int c = at(l);
44            assert(cur->to[c]);
45            cur = cur->to[c];
46            pos = min(cur->r, cur->l + 1);
47            ++l;
48        } else {
49            int delta = min(r - l, cur->r - pos);
50            l += delta;
51            pos += delta;
52        }
53        goDown(l, r);
54    }
55
56    void goUp() {
57        if (pos == cur->r && cur->lnk) {
58            cur = cur->lnk;
59            pos = cur->r;
60            return;
61        }
62        int l = cur->l, r = pos;
63        cur = cur->par->lnk;
64        pos = cur->r;
65        goDown(l, r);
66    }
67
68    void setParent(Node *a, Node *b) {
69        assert(a);
70        a->par = b;
71        if (b)
72            b->to[at(a->l)] = a;
73    }
74
75    void addLeaf(int id) {
76        Node *x = new Node(id, inf);
77        setParent(x, cur);
78    }
79
80    void splitNode() {
81        assert(pos != cur->r);
82        Node *mid = new Node(cur->l, pos);
83        setParent(mid, cur->par);
84        cur->l = pos;
85        setParent(cur, mid);
86        cur = mid;
87    }
88
89    bool canGo(int c) {
90        if (pos == cur->r)
91            return cur->to[c];
92        return at(pos) == c;
93    }
94
95    void fixLink(Node *&bad, Node *newBad) {
96        if (bad)
97            bad->lnk = cur;
98            bad = newBad;
99    }
100
101    void addCharOnPos(int id) {
102        Node *bad = 0;
103        while (!canGo(at(id))) {
104            if (cur->r != pos) {
105                splitNode();
106                fixLink(bad, cur);
107                bad = cur;
108            } else {
109                fixLink(bad, 0);
110            }
111            addLeaf(id);
112            goUp();
113        }
114        fixLink(bad, 0);
115        goDown(id, id + 1);
116    }
117
118    int cnt(Node *u, int ml) {
119        if (!u)
120            return 0;
121        int res = min(ml, u->r) - u->l;
122        forn (i, alpha)
123            res += cnt(u->to[i], ml);
124        return res;
125    }
126
127    void build(int l) {
128        init();
129        forn (i, l)
130            addCharOnPos(i);
131    }
132};
133
134int main() {
135    cin >> s;
136    SuffixTree::build(s.size());
137}

```

13 algo/structures/convex_hull_trick.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i,n) for (int i = 0; i < int(n); ++i)
4typedef long long ll;
5const ll inf1 = 1l(2e18) + 1l(2e15);
6
7const int maxn = 4004;
8
9/*
10 WARNING!!!
11 - finds maximum of A*x+B
12 - double check max coords for int/long long overflow
13 - set min x query in put function
14 - add lines with non-descending A coefficient
15*/
16struct FastHull {
17    int a[maxn];
18    ll b[maxn];
19    ll p[maxn];
20    int c;
21
22    FastHull(): c(0) {}
23
24    ll get(int x) {
25        if (c == 0)
26            return -inf1;
27        int pos = upper_bound(p, p + c, x) - p - 1;
28        assert(pos >= 0);
29        return (ll) a[pos] * x + b[pos];
30    }
31
32    ll divideCeil(ll p, ll q) {
33        assert(q > 0);
34        if (p >= 0)
35            return (p + q - 1) / q;
36        return -((-p) / q);
37    }
38
39    void put(int A, ll B) {
40        while (c > 0) {
41            if (a[c - 1] == A && b[c - 1] >= B)
42                return;
43            ll pt = p[c - 1];
44            if (a[c - 1] * pt + b[c - 1] < A * pt + B) {
45                --c;
46                continue;
47            }
48            ll q = A - a[c - 1];
49            ll np = divideCeil(b[c - 1] - B, q);
50            p[c] = np;
51            a[c] = A;
52            b[c] = B;
53            ++c;
54            return;
55        }
56        if (c == 0) {
57            a[c] = A, b[c] = B;
58            p[c] = -1e9; //min x query
59            ++c;
60            return;
61        }
62    }
63};
64};
65
66struct SlowHull {
67    vector<pair<int, ll>> v;
68
69    void put(int a, ll b) {
70        v.emplace_back(a, b);
71    }
72
73    ll get(ll x) {
74        ll best = -inf1;
75        for (auto p: v)
76            best = max(best, p.first * x + p.second);
77        return best;
78    }
79};
80
81int main() {
82    FastHull hull1;
83    SlowHull hull2;
84    vector<int> as;
85    forn (ii, 10000)
86        as.push_back(rand() % int(1e8));
87    sort(as.begin(), as.end());
88    forn (ii, 10000) {
89        int b = rand() % int(1e8);
90        hull1.put(as[ii], b);
91        hull2.put(as[ii], b);
92        int x = rand() % int(2e8 + 1) - int(1e8);
93        assert(hull1.get(x) == hull2.get(x));
94    }
95}

```

14 algo/structures/ordered_set.cpp

```

1#include <ext/pb_ds/assoc_container.hpp>
2#include <ext/pb_ds/tree_policy.hpp>
3
4typedef __gnu_pbds::tree<int, __gnu_pbds::null_type, std::less<int>,
5    __gnu_pbds::rb_tree_tag,
6    __gnu_pbds::tree_order_statistics_node_update> oset;
7#include <iostream>
8
9int main() {
10    oset X;
11    X.insert(1);
12    X.insert(2);
13    X.insert(4);
14    X.insert(8);
15    X.insert(16);
16
17    std::cout << *X.find_by_order(1) << std::endl; // 2
18    std::cout << *X.find_by_order(2) << std::endl; // 4
19    std::cout << *X.find_by_order(4) << std::endl; // 16
20    std::cout << std::boolalpha << (end(X)==X.find_by_order(6)) <<
    ↪    std::endl; // true
21
22    std::cout << X.order_of_key(-5) << std::endl; // 0
23    std::cout << X.order_of_key(1) << std::endl; // 0
24    std::cout << X.order_of_key(3) << std::endl; // 2
25    std::cout << X.order_of_key(4) << std::endl; // 2
26    std::cout << X.order_of_key(400) << std::endl; // 5
27}

```

15 algo/structures/splay.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
4
5const int maxn = 100500;
6
7struct node;
8void updson(node* p, node* v, node* was);
9
10struct node {
11    int val;
12    node *l, *r, *p;
13    node() {}
14    node(int val) : val(val), l(r=p=NULL) {}
15
16    bool isRoot() const { return !p; }
17    bool isRight() const { return p && p->r == this; }
18    bool isLeft() const { return p && p->l == this; }
19    void setLeft(node* t) {
20        if (t) t->p = this;
21        l = t;
22    }
23    void setRight(node* t) {
24        if (t) t->p = this;
25        r = t;
26    }
27};
28
29void updson(node *p, node *v, node *was) {
30    if (p) {
31        if (p->l == was) p->l = v;
32        else p->r = v;
33    }
34    if (v) v->p = p;
35}
36
37void rightRotate(node *v) {
38    assert(v && v->l);
39    node *u = v->l;
40    node *p = v->p;
41    v->setLeft(u->r);
42    u->setRight(v);
43    updson(p, u, v);
44}
45
46void leftRotate(node *v) {
47    assert(v && v->r);
48    node *u = v->r;
49    node *p = v->p;
50    v->setRight(u->l);
51    u->setLeft(v);
52    updson(p, u, v);
53}
54
55void splay(node *v) {
56    while (v->p) {
57        if (!v->p->p) {
58            if (v->isLeft()) rightRotate(v->p);
59            else leftRotate(v->p);
60        } else if (v->isLeft() && v->p->isLeft()) {
61            rightRotate(v->p->p);
62            rightRotate(v->p);
63        } else if (v->isRight() && v->p->isRight()) {
64            leftRotate(v->p->p);
65            leftRotate(v->p);
66        } else if (v->isLeft()) {
67            rightRotate(v->p);
68            leftRotate(v->p);
69        } else {
70            leftRotate(v->p);
71            rightRotate(v->p);
72        }
73    }
74    v->p = NULL;
75}
76
77node *insert(node *t, node *n) {
78    if (!t) return n;
79    int x = n->val;
80    while (true) {
81        if (x < t->val) {
82            if (t->l) {
83                t = t->l;
84            } else {
85                t->setLeft(n);
86                t = t->l;
87                break;
88            }
89        } else {
90            if (t->r) {
91                t = t->r;
92            } else {
93                t->setRight(n);
94                t = t->r;
95                break;
96            }
97        }
98    }
99}

```

```

90         if (t->r) {
91             t = t->r;
92         } else {
93             t->setRight(n);
94             t = t->r;
95             break;
96         }
97     }
98 }
99 splay(t);
100 return t;
101}
102
103node *insert(node *t, int x) {
104     return insert(t, new node(x));
105}
106
107int main() {
108     node *t = NULL;
109     forn(i, 1000000) {
110         int x = rand();
111         t = insert(t, x);
112     }
113     return 0;
114}

```

16 algo/structures/treap.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
4const int maxn = 100500;
5
6struct node {
7     int x, y;
8     node *l, *r;
9     node(int x) : x(x), y(rand()), l(r=NULL) {}
10};
11
12void split(node *t, node *&l, node *&r, int x) {
13     if (!t) return (void)(l=r=NULL);
14     if (x <= t->x) {
15         split(t->l, l, t->l, x), r = t;
16     } else {
17         split(t->r, t->r, r, x), l = t;
18     }
19}
20
21node *merge(node *l, node *r) {
22     if (!l) return r;
23     if (!r) return l;
24     if (l->y > r->y) {
25         l->r = merge(l->r, r);
26         return l;
27     } else {
28         r->l = merge(l, r->l);
29         return r;
30     }
31}
32
33node *insert(node *t, node *n) {
34     node *l, *r;
35     split(t, l, r, n->x);
36     return merge(l, merge(n, r));
37}
38
39node *insert(node *t, int x) {
40     return insert(t, new node(x));
41}
42
43node *fast_insert(node *t, node *n) {
44     if (!t) return n;
45     node *root = t;
46     while (true) {
47         if (n->x < t->x) {
48             if (!t->l || t->l->y < n->y) {
49                 split(t->l, n->l, n->r, n->x), t->l = n;
50                 break;
51             } else {
52                 t = t->l;
53             }
54         } else {
55             if (!t->r || t->r->y < n->y) {
56                 split(t->r, n->l, n->r, n->x), t->r = n;
57                 break;
58             } else {
59                 t = t->r;
60             }
61         }
62     }
63     return root;
64}
65
66node *fast_insert(node *t, int x) {
67     return fast_insert(t, new node(x));
68}
69
70int main() {
71     node *t = NULL;
72     forn(i, 1000000) {
73         int x = rand();
74         t = fast_insert(t, x);
75     }
76}

```