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))
- (WA) Проверить, что ответ влезает в int

2   flows/dinic.cpp

```cpp
namespace Dinic {
const int maxn = 10010;

struct Edge {
    int to, c, f;
} es[maxn*2];
int ne = 0;

int n;
vector<int> e[maxn];
int q[maxn], d[maxn], pos[maxn];
int S, T;

void addEdge(int u, int v, int c) {
    assert(c <= 1000000000);
    es[ne] = {v, c, 0};
    e[u].push_back(ne++);
    es[ne] = {u, 0, 0};
    e[v].push_back(ne++);
}

bool bfs() {
    forn(i, n) d[i] = maxn;
    d[S] = 0, q[0] = S;
    int lq = 0, rq = 1;
    while (lq != rq) {
        int v = q[lq++];
        for (int id: e[v]) if (es[id].f < es[id].c) {
            int to = es[id].to;
            if (d[to] == maxn)
                d[to] = d[v] + 1, q[rq++] = to;
        }
    }
    return d[T] != maxn;
}

int dfs(int v, int curf) {
    if (v == T || curf == 0) return curf;
    for (int &i = pos[v]; i < (int)e[v].size(); ++i) {
        int id = e[v][i];
        int to = es[id].to;
        if (es[id].f < es[id].c && d[v] + 1 == d[to]) {
            if (int ret = dfs(to, min(curf, es[id].c -
  es[id].f))) {
                es[id].f += ret;
                es[id^1].f -= ret;
                return ret;
            }
        }
    }
    return 0;
}

i64 dinic(int S, int T) {
    Dinic::S = S, Dinic::T = T;
    i64 res = 0;
    while (bfs()) {
        forn(i, n) pos[i] = 0;
        while (int f = dfs(S, 1e9)) {
            assert(f <= 1000000000);
            res += f;
        }
    }
    return res;
}

} // namespace Dinic

void test() {
    Dinic::n = 4;
    Dinic::addEdge(0, 1, 1);
    Dinic::addEdge(0, 2, 2);
    Dinic::addEdge(2, 1, 1);
    Dinic::addEdge(1, 3, 2);
    Dinic::addEdge(2, 3, 1);
    cout << Dinic::dinic(0, 3) << endl; // 3

}

/*
 * LR-поток.
 * LR-поток находит не максимальный поток.
 * Добавим новый сток S' и исток T'. Заменим ребро (u, v, l,
  r) LR-сети
 * на ребра (u, T', l), (S', v, l), (u, v, r - l).
 * Добавим ребро (T, S, k). Ставим значение k=inf, пускаем
  поток.
 * Проверяем, что все ребра из S' насыщены(иначе ответ не
  существует).
 * Бинпоиском находим наименьшее k, что величина потока не
  изменится.
 * Это k - величина МИНИМАЛЬНОГО потока, удовлетворяющего
  ограничениям.
 */
```

3   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';
}
```

4   flows/hungary.cpp

```cpp
// 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];
    }
};
```

5   flows/mincost.cpp

```
1  namespace MinCost {
2      const ll infc = 1e12;
3
4      struct Edge {
5          int to;
6          ll c, f, cost;
7
8          Edge(int to, ll c, ll cost): to(to), c(c), f(0),
   ↪   cost(cost) {
9          }
10     };
11
12     int N, S, T;
13     int totalFlow;
14     ll totalCost;
15     const int maxn = 505;
16     vector<Edge> edge;
17     vector<int> g[maxn];
18
19     void addEdge(int u, int v, ll c, ll cost) {
20         g[u].push_back(edge.size());
21         edge.emplace_back(v, c, cost);
22         g[v].push_back(edge.size());
23         edge.emplace_back(u, 0, -cost);
24     }
25
26     ll dist[maxn];
27     int fromEdge[maxn];
28
29     bool inQueue[maxn];
30     bool fordBellman() {
31         forn (i, N)
32             dist[i] = infc;
33         dist[S] = 0;
34         inQueue[S] = true;
35         vector<int> q;
36         q.push_back(S);
37         for (int ii = 0; ii < int(q.size()); ++ii) {
38             int u = q[ii];
39             inQueue[u] = false;
40             for (int e: g[u]) {
41                 if (edge[e].f == edge[e].c)
42                     continue;
43                 int v = edge[e].to;
44                 ll nw = edge[e].cost + dist[u];
45                 if (nw >= dist[v])
46                     continue;
47                 dist[v] = nw;
48                 fromEdge[v] = e;
49                 if (!inQueue[v]) {
50                     inQueue[v] = true;
51                     q.push_back(v);
52                 }
53             }
54         }
55         return dist[T] != infc;
56     }
57
58     ll pot[maxn];
59     bool dikstra() {
60         priority_queue<pair<ll, int>, vector<pair<ll, int>>,
   ↪   greater<pair<ll, int>>> q;
61         forn (i, N)
62             dist[i] = infc;
63         dist[S] = 0;
64         q.emplace(dist[S], S);
65         while (!q.empty()) {
66             int u = q.top().second;
67             ll cdist = q.top().first;
68             q.pop();
69             if (cdist != dist[u])
70                 continue;
71             for (int e: g[u]) {
72                 int v = edge[e].to;
73                 if (edge[e].c == edge[e].f)
74                     continue;
75                 ll w = edge[e].cost + pot[u] - pot[v];
76                 assert(w >= 0);
77                 ll ndist = w + dist[u];
78                 if (ndist >= dist[v])
79                     continue;
80                 dist[v] = ndist;
81                 fromEdge[v] = e;
82                 q.emplace(dist[v], v);
83             }
84         }
85         if (dist[T] == infc)
86             return false;
87         forn (i, N) {
88             if (dist[i] == infc)
89                 continue;
90             pot[i] += dist[i];
91         }
92         return true;
93     }
94
95     bool push() {
96         //2 variants
97         //if (!fordBellman())
98         if (!dikstra())
99             return false;
100        ++totalFlow;
101        int u = T;
102        while (u != S) {
103            int e = fromEdge[u];
104            totalCost += edge[e].cost;
105            edge[e].f++;
106            edge[e ^ 1].f--;
107            u = edge[e ^ 1].to;
108        }
109        return true;
110    }
111 };
112
113 int main() {
114     MinCost::N = 3, MinCost::S = 1, MinCost::T = 2;
115     MinCost::addEdge(1, 0, 3, 5);
116     MinCost::addEdge(0, 2, 4, 6);
117     while (MinCost::push());
118     cout << MinCost::totalFlow << ' ' << MinCost::totalCost <<
   ↪   '\n'; //3 33
119 }
```

6   geometry/convex_hull.cpp

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

#include "primitives.cpp"

bool cmpAngle(const pt &a, const pt &b) {
    bool ar = a.right(), br = b.right();
    if (ar ^ br)
        return ar;
    return gt(a % b, 0);
}

struct Hull {
    vector<pt> top, bot;

    void append(pt p) {
        while (bot.size() > 1 && ge((p - bot.back()) %
  (bot.back() - *next(bot.rbegin())), 0))
            bot.pop_back();
        bot.push_back(p);
        while (top.size() > 1 && ge(0, (p - top.back()) %
  (top.back() - *next(top.rbegin()))))
            top.pop_back();
        top.push_back(p);
    }

    void build(vector<pt> h) {
        sort(h.begin(), h.end());
        h.erase(unique(h.begin(), h.end()), h.end());
        top.clear(), bot.clear();
        for (pt p: h)
            append(p);
    }

    pt kth(int k) {
        if (k < sz(bot))
            return bot[k];
        else
            return top[sz(top) - (k - sz(bot)) - 2];
    }

    pt mostDistant(pt dir) {
        if (bot.empty()) {
            //empty hull
            return pt{1e18, 1e18};
        }
        if (bot.size() == 1)
            return bot.back();
        dir = dir.rot();
        int n = sz(top) + sz(bot) - 2;
        int L = -1, R = n;
        while (L + 1 < R) {
            int C = (L + R) / 2;
            pt v = kth((C + 1) % n) - kth(C);
            if (cmpAngle(dir, v)) //finds upper bound
                R = C;
            else
                L = C;
        }
        return kth(R % n);
    }
};
```

7   geometry/halfplanes.cpp

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

ld det3x3(line &l1, line &l2, line &l3) {
    return l1.a * (l2.b * l3.c - l2.c * l3.b) +
           l1.b * (l2.c * l3.a - l2.a * l3.c) +
           l1.c * (l2.a * l3.b - l2.b * l3.a);
}

vector<pt> halfplanesIntersecion(vector<line> lines) {
    sort(lines.begin(), lines.end(), [](const line &a, const
  line &b) {
            bool ar = a.right(), br = b.right();
            if (ar ^ br)
                return ar;
            ld prod = (pt{a.a, a.b} % pt{b.a, b.b});
            if (!eq(prod, 0))
                return prod > 0;
            return a.c < b.c;
        });
    vector<line> lines2;
    pt pr;
    forn (i, lines.size()) {
        pt cur{lines[i].a, lines[i].b};
        if (i == 0 || cur != pr)
            lines2.push_back(lines[i]);
        pr = cur;
    }
    lines = lines2;
    int n = lines.size();
    forn (i, n)
        lines[i].id = i;
    vector<line> hull;
    forn (i, 2 * n) {
        line l = lines[i % n];
        while ((int) hull.size() >= 2) {
            ld D = det3x3(*prev(prev(hull.end())),
  hull.back(), l);
            if (ge(D, 0))
                break;
            hull.pop_back();
        }
        hull.push_back(l);
    }
    vector<int> firstTime(n, -1);
    vector<line> v;
    forn (i, hull.size()) {
        int cid = hull[i].id;
        if (firstTime[cid] == -1) {
            firstTime[cid] = i;
            continue;
        }
        forab(j, firstTime[cid], i)
            v.push_back(hull[j]);
        break;
    }
    n = v.size();
    if (v.empty()) {
        //empty intersection
        return {};
    }
    v.push_back(v[0]);
    vector<pt> res;
    pt center{0, 0};
    forn (i, n) {
        res.push_back(linesIntersection(v[i], v[i + 1]));
        center = center + res.back();
    }
    center = center / n;
    for (auto l: lines)
        if (gt(0, l.signedDist(center))) {
            //empty intersection
            return {};
        }
    return res;
}
```

8   geometry/polygon.cpp

```cpp
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forn(i,n) for (int i = 0; i < int(n); ++i)
4
5 #include "primitives.cpp"
6
7 bool pointInsidePolygon(pt a, pt *p, int n) {
8     double sumAng = 0;
9     forn (i, n) {
10         pt A = p[i], B = p[(i + 1) % n];
11         if (pointInsideSegment(a, A, B))
12             return true;
13         sumAng += atan2((A - a) % (B - a), (A - a) * (B - a));
14     }
15     return fabs(sumAng) > 1;
16 }
17
18 //p must be oriented counterclockwise
19 bool segmentInsidePolygon(pt a, pt b, pt *p, int n) {
20     if (!pointInsidePolygon((a + b) / 2, p, n))
21         return false;
22     if (a == b)
23         return true;
24     forn (i, n) {
25         pt c = p[i];
26         if (eq((a - c) % (b - c), 0) && gt(0, (a - c) * (b -
   c))) {
27             //point on segment
28             pt pr = p[(i + n - 1) % n];
29             pt nx = p[(i + 1) % n];
30             if (gt((c - pr) % (nx - c), 0))
31                 return false;
32             ld s1 = (pr - a) % (b - a);
33             ld s2 = (nx - a) % (b - a);
34             if ((gt(s1, 0) || gt(s2, 0)) && (gt(0, s1) ||
   gt(0, s2)))
35                 return false;
36         }
37         //interval intersection
38         pt d = p[(i + 1) % n];
39         ld s1 = (a - c) % (d - c);
40         ld s2 = (b - c) % (d - c);
41         if (ge(s1, 0) && ge(s2, 0))
42             continue;
43         if (ge(0, s1) && ge(0, s2))
44             continue;
45
46         s1 = (c - a) % (b - a);
47         s2 = (d - a) % (b - a);
48         if (ge(s1, 0) && ge(s2, 0))
49             continue;
50         if (ge(0, s1) && ge(0, s2))
51             continue;
52
53         return false;
54     }
55     return true;
56 }
```

9   geometry/primitives.cpp

```cpp
1 #pragma once
2 #include <bits/stdc++.h>
3 #define forn(i, n) for (int i = 0; i < int(n); ++i)
4 using namespace std;
5 typedef long double ld;
6
7 const ld eps = 1e-9;
8
9 bool eq(ld a, ld b) { return fabsl(a - b) < eps; }
10 bool ge(ld a, ld b) { return a - b > -eps; }
11 bool gt(ld a, ld b) { return a - b > eps; }
12 ld sqr(ld x) { return x * x; }
13
14 #ifdef LOCAL
15 #define gassert assert
16 #else
17 void gassert(bool) {}
18 #endif
19
20 struct pt {
21     ld x, y;
22
23     pt operator+(const pt &p) const { return pt{x + p.x, y +
   p.y}; }
24     pt operator-(const pt &p) const { return pt{x - p.x, y -
   p.y}; }
25     ld operator*(const pt &p) const { return x * p.x + y *
   p.y; }
26     ld operator%(const pt &p) const { return x * p.y - y *
   p.x; }
27
28     pt operator*(const ld &a) const { return pt{x * a, y * a};
   }
29     pt operator/(const ld &a) const { gassert(!eq(a, 0));
   return pt{x / a, y / a}; }
30     void operator*=(const ld &a) { x *= a, y *= a; }
31     void operator/=(const ld &a) { gassert(!eq(a, 0)); x /= a,
   y /= a; }
32
33     bool operator<(const pt &p) const {
34         if (eq(x, p.x)) return gt(p.y, y);
35         return x < p.x;
36     }
37
38     bool operator==(const pt &p) const { return eq(x, p.x) &&
   eq(y, p.y); }
39     bool operator!=(const pt &p) const { return !(*this == p);
   }
40
41     bool right() const { return pt{0, 0} < *this; }
42
43     pt rot() { return pt{-y, x}; }
44     ld abs() const { return hypotl(x, y); }
45     ld abs2() const { return x * x + y * y; }
46 };
47
48 istream &operator>>(istream &in, pt &p) { return in >> p.x >>
   p.y; }
49 ostream &operator<<(ostream &out, const pt &p) { return out <<
   p.x << ' ' << p.y; }
50
51 //WARNING! do not forget to normalize vector (a,b)
52 struct line {
53     ld a, b, c;
54     int id;
55
56     line(pt p1, pt p2) {
57         gassert(p1 != p2);
58         pt n = (p2 - p1).rot();
59         n /= n.abs();
60         a = n.x, b = n.y;
61         c = -(n * p1);
62     }
63
64     bool right() const {
65         return gt(a, 0) || (eq(a, 0) && gt(b, 0));
66     }
67
68     line(ld _a, ld _b, ld _c): a(_a), b(_b), c(_c) {
69         ld d = pt{a, b}.abs();
70         gassert(!eq(d, 0));
71         a /= d, b /= d, c /= d;
72     }
73
74     ld signedDist(pt p) {
75         return p * pt{a, b} + c;
76     }
77 };
78
79 ld pointSegmentDist(pt p, pt a, pt b) {
```

```
80      ld res = min((p - a).abs(), (p - b).abs());
81      if (a != b && ge((p - a) * (b - a), 0) && ge((p - b) * (a
    ↪    - b), 0))
82          res = min(res, fabsl((p - a) % (b - a)) / (b -
    ↪    a).abs());
83      return res;
84 }
85
86 pt linesIntersection(line l1, line l2) {
87      ld D = l1.a * l2.b - l1.b * l2.a;
88      if (eq(D, 0)) {
89          if (eq(l1.c, l2.c)) {
90              //equal lines
91          } else {
92              //no intersection
93          }
94      }
95      ld dx = -l1.c * l2.b + l1.b * l2.c;
96      ld dy = -l1.a * l2.c + l1.c * l2.a;
97      pt res{dx / D, dy / D};
98      //gassert(eq(l1.signedDist(res), 0));
99      //gassert(eq(l2.signedDist(res), 0));
100     return res;
101 }
102
103 bool pointInsideSegment(pt p, pt a, pt b) {
104     if (!eq((p - a) % (p - b), 0))
105         return false;
106     return ge(0, (a - p) * (b - p));
107 }
108
109 bool checkSegmentIntersection(pt a, pt b, pt c, pt d) {
110     if (eq((a - b) % (c - d), 0)) {
111         if (pointInsideSegment(a, c, d) ||
    ↪    pointInsideSegment(b, c, d) ||
112             pointInsideSegment(c, a, b) ||
    ↪    pointInsideSegment(d, a, b)) {
113             //intersection of parallel segments
114             return true;
115         }
116         return false;
117     }
118
119     ld s1, s2;
120
121     s1 = (c - a) % (b - a);
122     s2 = (d - a) % (b - a);
123     if (gt(s1, 0) && gt(s2, 0))
124         return false;
125     if (gt(0, s1) && gt(0, s2))
126         return false;
127
128     swap(a, c), swap(b, d);
129
130     s1 = (c - a) % (b - a);
131     s2 = (d - a) % (b - a);
132     if (gt(s1, 0) && gt(s2, 0))
133         return false;
134     if (gt(0, s1) && gt(0, s2))
135         return false;
136
137     return true;
138 }
139
140 //WARNING! run checkSegmentIntersecion before and process
    ↪    parallel case manually
141 pt segmentsIntersection(pt a, pt b, pt c, pt d) {
142     ld S = (b - a) % (d - c);
143     ld s1 = (c - a) % (d - a);
144     return a + (b - a) / S * s1;
145 }
146
147 vector<pt> circlesIntersction(pt a, ld r1, pt b, ld r2) {
148     ld d2 = (a - b).abs2();
149     ld d = (a - b).abs();
150
151     if (a == b && eq(r1, r2)) {
152         //equal circles
153     }
154     if (gt(d2, sqr(r1 + r2)) || gt(sqr(r1 - r2), d2)) {
155         //empty intersection
156         return {};
157     }
158     int num = 2;
159     if (eq(sqr(r1 + r2), d2) || eq(sqr(r1 - r2), d2))
160         num = 1;
161     ld cosa = (sqr(r1) + d2 - sqr(r2)) / ld(2 * r1 * d);
162     ld oh = cosa * r1;
163     pt h = a + ((b - a) / d * oh);
164     if (num == 1)
165         return {h};
166     ld hp = sqrtl(max(0.L, 1 - cosa * cosa)) * r1;
```

```
167
168     pt w = ((b - a) / d * hp).rot();
169     return {h + w, h - w};
170 }
171
172 //a is circle center, p is point
173 vector<pt> circleTangents(pt a, ld r, pt p) {
174     ld d2 = (a - p).abs2();
175     ld d = (a - p).abs();
176
177     if (gt(sqr(r), d2)) {
178         //no tangents
179         return {};
180     }
181     if (eq(sqr(r), d2)) {
182         //point lies on circle - one tangent
183         return {p};
184     }
185
186     pt B = p - a;
187     pt H = B * sqr(r) / d2;
188     ld h = sqrtl(d2 - sqr(r)) * ld(r) / d;
189     pt w = (B / d * h).rot();
190     H = H + a;
191     return {H + w, H - w};
192 }
193
194 vector<pt> lineCircleIntersection(line l, pt a, ld r) {
195     ld d = l.signedDist(a);
196     if (gt(fabsl(d), r))
197         return {};
198     pt h = a - pt{l.a, l.b} * d;
199     if (eq(fabsl(d), r))
200         return {h};
201     pt w = pt{l.a, l.b}.rot() * sqrtl(max<ld>(0, sqr(r) -
    ↪    sqr(d)));
202     return {h + w, h - w};
203 }
204
205 //modified magic from e-maxx
206 vector<line> commonTangents(pt a, ld r1, pt b, ld r2) {
207     if (a == b && eq(r1, r2)) {
208         //equal circles
209         return {};
210     }
211     vector<line> res;
212     pt c = b - a;
213     ld z = c.abs2();
214     for (int i = -1; i <= 1; i += 2)
215         for (int j = -1; j <= 1; j += 2) {
216             ld r = r2 * j - r1 * i;
217             ld d = z - sqr(r);
218             if (gt(0, d))
219                 continue;
220             d = sqrtl(max<ld>(0, d));
221             pt magic = pt{r, d} / z;
222             line l(magic * c, magic % c, r1 * i);
223             l.c -= pt{l.a, l.b} * a;
224             res.push_back(l);
225         }
226     return res;
227 }
```

10   geometry/svg.cpp

```
1 struct SVG {
2     FILE *out;
3     ld sc = 50;
4
5     void open() {
6         out = fopen("image.svg", "w");
7         fprintf(out, "<svg xmlns='http://www.w3.org/2000/svg'
↪   viewBox='-1000 -1000 2000 2000'>\n");
8     }
9
10    void line(pt a, pt b) {
11        a = a * sc, b = b * sc;
12        fprintf(out, "<line x1='%Lf' y1='%Lf' x2='%Lf'
↪   y2='%Lf' stroke='black'/>\n", a.x, -a.y, b.x, -b.y);
13    }
14
15    void circle(pt a, ld r = -1, string col = "red") {
16        r = (r == -1 ? 10 : sc * r);
17        a = a * sc;
18        fprintf(out, "<circle cx='%Lf' cy='%Lf' r='%Lf'
↪   fill='%s'/>\n", a.x, -a.y, r, col.c_str());
19    }
20
21    void text(pt a, string s) {
22        a = a * sc;
23        fprintf(out, "<text x='%Lf' y='%Lf'
↪   font-size='10px'>%s</text>\n", a.x, -a.y, s.c_str());
24    }
25
26    void close() {
27        fprintf(out, "</svg>\n");
28        fclose(out);
29        out = 0;
30    }
31
32    ~SVG() {
33        if (out)
34            close();
35    }
36 } svg;
```

11   graphs/2sat.cpp

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

12   graphs/directed_mst.cpp

```
1  // WARNING: this code wasn't submitted anywhere
2
3  namespace TwoChinese {
4
5  struct Edge {
6      int to, w, id;
7      bool operator<(const Edge& other) const {
8          return to < other.to || (to == other.to && w <
   ↪   other.w);
9      }
10 };
11 typedef vector<vector<Edge>> Graph;
12
13 const int maxn = 2050;
14
15 // global, for supplementary algorithms
16 int b[maxn];
17 int tin[maxn], tup[maxn];
18 int dtime; // counter for tin, tout
19 vector<int> st;
20 int nc; // number of strongly connected components
21 int q[maxn];
22
23 int answer;
24
25 void tarjan(int v, const Graph& e, vector<int>& comp) {
26     b[v] = 1;
27     st.push_back(v);
28     tin[v] = tup[v] = dtime++;
29
30     for (Edge t: e[v]) if (t.w == 0) {
31         int to = t.to;
32         if (b[to] == 0) {
33             tarjan(to, e, comp);
34             tup[v] = min(tup[v], tup[to]);
35         } else if (b[to] == 1) {
36             tup[v] = min(tup[v], tin[to]);
37         }
38     }
39
40     if (tin[v] == tup[v]) {
41         while (true) {
42             int t = st.back();
43             st.pop_back();
44             comp[t] = nc;
45             b[t] = 2;
46             if (t == v) break;
47         }
48         ++nc;
49     }
50 }
51
52 vector<Edge> bfs(
53     const Graph& e, const vector<int>& init, const
   ↪   vector<int>& comp)
54 {
55     int n = e.size();
56     forn(i, n) b[i] = 0;
57     int lq = 0, rq = 0;
58     for (int v: init) b[v] = 1, q[rq++] = v;
59
60     vector<Edge> result;
61
62     while (lq != rq) {
63         int v = q[lq++];
64         for (Edge t: e[v]) if (t.w == 0) {
65             int to = t.to;
66             if (b[to]) continue;
67             if (!comp.empty() && comp[v] != comp[to])
   ↪   continue;
68             b[to] = 1;
69             q[rq++] = to;
70             result.push_back(t);
71         }
72     }
73
74     return result;
75 }
76
77 // warning: check that each vertex is reachable from root
78 vector<Edge> run(Graph e, int root) {
79     int n = e.size();
80
81     // find minimum incoming weight for each vertex
82     vector<int> minw(n, inf);
83     forn(v, n) for (Edge t: e[v]) {
84         minw[t.to] = min(minw[t.to], t.w);
85     }
86     forn(v, n) for (Edge &t: e[v]) if (t.to != root) {
87         t.w -= minw[t.to];
88     }
89     forn(i, n) if (i != root) answer += minw[i];
90
91     // check if each vertex is reachable from root by zero
   ↪   edges
92     vector<Edge> firstResult = bfs(e, {root}, {});
93     if ((int)firstResult.size() + 1 == n) {
94         return firstResult;
95     }
96
97     // find stongly connected components and build compressed
   ↪   graph
98     vector<int> comp(n);
99     forn(i, n) b[i] = 0;
100    nc = 0;
101    dtime = 0;
102    forn(i, n) if (!b[i]) tarjan(i, e, comp);
103
104    // multiple edges may be removed here if needed
105    Graph ne(nc);
106    forn(v, n) for (Edge t: e[v]) {
107        if (comp[v] != comp[t.to]) {
108            ne[comp[v]].push_back({comp[t.to], t.w, t.id});
109        }
110    }
111
112    // run recursively on compressed graph
113    vector<Edge> subres = run(ne, comp[root]);
114
115    // find incoming edge id for each component, init queue
116    // if there is an edge (u, v) between different components
117    // than v is added to queue
118    vector<int> incomingId(nc);
119    for (Edge e: subres) {
120        incomingId[e.to] = e.id;
121    }
122
123    vector<Edge> result;
124    vector<int> init;
125    init.push_back(root);
126    forn(v, n) for (Edge t: e[v]) {
127        if (incomingId[comp[t.to]] == t.id) {
128            result.push_back(t);
129            init.push_back(t.to);
130        }
131    }
132
133    // run bfs to add edges inside components and return
   ↪   answer
134    vector<Edge> innerEdges = bfs(e, init, comp);
135    result.insert(result.end(), all(innerEdges));
136
137    assert((int)result.size() + 1 == n);
138    return result;
139 }
140
141 } // namespace TwoChinese
142
143 void test () {
144     auto res = TwoChinese::run({
145         {{1,5,0},{2,5,1}},
146         {{3,1,2}},
147         {{1,2,3},{4,1,4}},
148         {{1,1,5},{4,2,6}},
149         {{2,1,7}}},
150         0);
151     cout << TwoChinese::answer << endl;
152     for (auto e: res) cout << e.id << " ";
153     cout << endl;
154     // 9    0 6 2 7
155 }
```

13   graphs/euler_cycle.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

const int maxn = 100100;
const int maxm = 100100;

struct Edge {
    int to, id;
};

bool usedEdge[maxm];
vector<Edge> g[maxn];
int ptr[maxn];

vector<int> cycle;
void eulerCycle(int u) {
    while (ptr[u] < (int) g[u].size() &&
      usedEdge[g[u][ptr[u]].id])
        ++ptr[u];
    if (ptr[u] == (int) g[u].size())
        return;
    const Edge &e = g[u][ptr[u]];
    usedEdge[e.id] = true;
    eulerCycle(e.to);
    cycle.push_back(e.id);
    eulerCycle(u);
}

int edges = 0;
void addEdge(int u, int v) {
    g[u].push_back(Edge{v, edges});
    g[v].push_back(Edge{u, edges++});
}

int main() {
}
```

14   math/fft.cpp

```cpp
const int maxlg = 20;

vector<base> ang[maxlg + 5];

void init_fft() {
    int n = 1 << maxlg;
    ld e = acosl(-1) * 2 / n;
    ang[maxlg].resize(n);
    forn(i, n) {
        ang[maxlg][i] = { cos(e * i), sin(e * i) };
    }

    for (int k = maxlg - 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;
    }
}

//n must be power of 2
void fft(base *a, int n, bool rev) {
    int lg = 0;
    while ((1<<lg) != n) {
        ++lg;
    }
    int j = 0, bit;
    for (int i = 1; i < n; ++i) {
        for (bit = n >> 1; bit & j; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j) swap(a[i], a[j]);
    }
    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 = 8;
    init_fft();
    base a[8] = {1,3,5,2,4,6,7,1};
    base b[16];
    fft(b, 16, 0);
    fft(a, n, 0);
    forn(i, n) cout << a[i].re << " "; cout << endl;
    forn(i, n) cout << a[i].im << " "; cout << endl;
    // 29 -5.82843 -7 -0.171573 5 -0.171573 -7 -5.82843
    // 0 -3.41421 6 0.585786 0 -0.585786 -6 3.41421
}
```

15   math/golden_search.cpp

```
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';
}
```

16   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))

Gauss 5-th order numerical integration:
integral from -1 to 1
x1, x3 = +-sqrt(0.6), x2 = 0
a1, a3 = 5/9, a2 = 8/9

large primes: 10^18 +3, +31, +3111

fft modules for 2**20:
7340033 13631489 26214401 28311553 70254593
976224257 (largest less than 10**9)

fibonacci numbers:
1, 2: 1
45: 1134903170
46: 1836311903 (max int)
47: 2971215073 (max unsigned)
91: 4660046610375530309
92: 7540113804746346429 (max i64)
93: 12200160415121876738 (max unsigned i64)

2**31 = 2147483648 = 2.1e9
2**32 = 4294967296 = 4.2e9
2**63 = 9223372036854775808 = 9.2e18
2**64 = 18446744073709551616 = 1.8e19

highly composite: todo
```

17 strings/automaton.cpp

```cpp
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;
}
```

18 strings/duval.cpp

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

/*
    Строка простая, если строго меньше всех суффиксов <=>
    наименьший циклический сдвиг - первый.
    Декомпозиция Линдона - разбиение s на w1, w2, ... wk -
    простые строки такие, что w1 >= w2 >= ... wk.
*/
int duval(string s) {
    s += s; //remove this to find Lyndon decomposition of s
    int n = s.size();
    int i = 0;
    int ans = 0;
    //while (i < n) { //for Lyndon decomposition
    while (i < n / 2) {
        ans = i;
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j])
                k = i;
            else
                ++k;
            ++j;
        }
        while (i <= k) {
            //s.substr(i, j - k) -
            //next prime string of Lyndon decomposition
            i += j - k;
        }
    }
    return ans;
}

int main() {
    cout << duval("ababcabab") << '\n'; // 5
}
```

19   strings/eertree.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 5000100;
const int inf = 1e9 + 1e5;

char buf[maxn];
char *s = buf + 1;
int to[maxn][2];
int suff[maxn];
int len[maxn];
int sz;
int last;

const int odd = 1;
const int even = 2;
const int blank = 3;

inline void go(int &u, int pos) {
    while (u != blank && s[pos - len[u] - 1] != s[pos])
        u = suff[u];
}

void add_char(int pos) {
    go(last, pos);
    int u = suff[last];
    go(u, pos);
    int c = s[pos] - 'a';
    if (!to[last][c]) {
        to[last][c] = sz++;
        len[sz - 1] = len[last] + 2;
        assert(to[u][c]);
        suff[sz - 1] = to[u][c];
    }
    last = to[last][c];
}

void init() {
    sz = 4;
    to[blank][0] = to[blank][1] = even;
    len[blank] = suff[blank] = inf;
    len[even] = 0, suff[even] = odd;
    len[odd] = -1, suff[odd] = blank;
    last = 2;
}

void build() {
    init();
    scanf("%s", s);
    for (int i = 0; s[i]; ++i)
        add_char(i);
}
```

20   strings/manacher.cpp

```cpp
//actual odd length is (odd[i] * 2 - 1)
//actual even length is (even[i] * 2)
void manacher(const string &s, vector<int> &odd, vector<int>
    &even) {
    int n = s.size();
    odd.resize(n);
    int c = -1, r = -1;
    forn (i, n) {
        int k = (r <= i ? 0 : min(odd[2 * c - i], r - i));
        while (i + k < n && i - k >= 0 && s[i + k] == s[i -
    k])
            ++k;
        odd[i] = k;
        if (i + k > r)
            r = i + k, c = i;
    }
    c = -1, r = -1;
    even.resize(n - 1);
    forn (i, n - 1) {
        int k = (r <= i ? 0 : min(even[2 * c - i], r - i));
        while (i + k + 1 < n && i - k >= 0 && s[i + k + 1] ==
    s[i - k])
            ++k;
        even[i] = k;
        if (i + k > r)
            c = i, r = i + k;
    }
}

void test() {
    vector<int> odd, even;
    string s = "aaaabbaaaaa";
    manacher(s, odd, even);
    for (int x: even)
        cerr << x << ' ';
    cerr << '\n';
    for (int x: odd)
        cerr << x << ' ';
    cerr << '\n';
    // 1 2 1 0 5 0 1 2 2 1
    // 1 2 2 1 1 1 1 2 3 2 1
}
```

21   strings/suffix_array.cpp

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

22   strings/ukkonen.cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define sz(x) ((int) (x).size())
4 #define forn(i,n) for (int i = 0; i < int(n); ++i)
5 const int inf = int(1e9) + int(1e5);
6
7 string s;
8 const int alpha = 26;
9
10 namespace 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
134 int main() {
135     cin >> s;
136     SuffixTree::build(s.size());
137 }
```

23   structures/convex_hull_trick.cpp

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

24   structures/heavy_light.cpp

```
1  const int maxn = 100500;
2  const int maxd = 17;
3
4  vector<int> g[maxn];
5
6  struct Tree {
7      vector<int> t;
8      int base;
9
10     Tree(): base(0) {
11     }
12
13     Tree(int n) {
14         base = 1;
15         while (base < n)
16             base *= 2;
17         t = vector<int>(base * 2, 0);
18     }
19
20     void put(int v, int delta) {
21         assert(v < base);
22         v += base;
23         t[v] += delta;
24         while (v > 1) {
25             v /= 2;
26             t[v] = max(t[v * 2], t[v * 2 + 1]);
27         }
28     }
29
30     //Careful here: cr = 2 * maxn
31     int get(int l, int r, int v = 1, int cl = 0, int cr = 2 *
   ↪  maxn) {
32         cr = min(cr, base);
33         if (l <= cl && cr <= r)
34             return t[v];
35         if (r <= cl || cr <= l)
36             return 0;
37         int cc = (cl + cr) / 2;
38         return max(get(l, r, v * 2, cl, cc), get(l, r, v * 2 +
   ↪  1, cc, cr));
39     }
40 };
41
42 namespace HLD {
43     int h[maxn];
44     int timer;
45     int in[maxn], out[maxn], cnt[maxn];
46     int p[maxd][maxn];
47     int vroot[maxn];
48     int vpos[maxn];
49     int ROOT;
50     Tree tree[maxn];
51
52     void dfs1(int u, int prev) {
53         p[0][u] = prev;
54         in[u] = timer++;
55         cnt[u] = 1;
56         for (int v: g[u]) {
57             if (v == prev)
58                 continue;
59             h[v] = h[u] + 1;
60             dfs1(v, u);
61             cnt[u] += cnt[v];
62         }
63         out[u] = timer;
64     }
65
66     int dfs2(int u, int prev) {
67         int to = -1;
68         for (int v: g[u]) {
69             if (v == prev)
70                 continue;
71             if (to == -1 || cnt[v] > cnt[to])
72                 to = v;
73         }
74         int len = 1;
75         for (int v: g[u]) {
76             if (v == prev)
77                 continue;
78             if (to == v) {
79                 vpos[v] = vpos[u] + 1;
80                 vroot[v] = vroot[u];
81                 len += dfs2(v, u);
82             }
83             else {
84                 vroot[v] = v;
85                 vpos[v] = 0;
86                 dfs2(v, u);
87             }
88         }
89         if (vroot[u] == u)
90             tree[u] = Tree(len);
91         return len;
92     }
93
94     void init(int n) {
95         timer = 0;
96         h[ROOT] = 0;
97         dfs1(ROOT, ROOT);
98         forn (d, maxd - 1)
99             forn (i, n)
100                 p[d + 1][i] = p[d][p[d][i]];
101        vroot[ROOT] = ROOT;
102        vpos[ROOT] = 0;
103        dfs2(ROOT, ROOT);
104        //WARNING: init all trees
105     }
106
107     bool isPrev(int u, int v) {
108         return in[u] <= in[v] && out[v] <= out[u];
109     }
110
111     int lca(int u, int v) {
112         for (int d = maxd - 1; d >= 0; --d)
113             if (!isPrev(p[d][u], v))
114                 u = p[d][u];
115         if (!isPrev(u, v))
116             u = p[0][u];
117         return u;
118     }
119
120     //for each v: h[v] >= toh
121     int getv(int u, int toh) {
122         int res = 0;
123         while (h[u] >= toh) {
124             int rt = vroot[u];
125             int l = max(0, toh - h[rt]), r = vpos[u] + 1;
126             res = max(res, tree[rt].get(l, r));
127             if (rt == ROOT)
128                 break;
129             u = p[0][rt];
130         }
131         return res;
132     }
133
134     int get(int u, int v) {
135         int w = lca(u, v);
136         return max(getv(u, h[w]), getv(v, h[w] + 1));
137     }
138
139     void put(int u, int val) {
140         int rt = vroot[u];
141         int pos = vpos[u];
142         tree[rt].put(pos, val);
143     }
144 };
```

25 structures/linkcut.cpp

```
1 namespace LinkCut {
2
3 typedef struct _node {
4     _node *l, *r, *p, *pp;
5     int size; bool rev;
6     _node();
7
8     explicit _node(nullptr_t) {
9         l = r = p = pp = this;
10        size = rev = 0;
11    }
12
13    void push() {
14        if (rev) {
15            l->rev ^= 1; r->rev ^= 1;
16            rev = 0; swap(l,r);
17        }
18    }
19
20    void update();
21 }* node;
22
23 node None = new _node(nullptr);
24 node v2n[maxn];
25
26 _node::_node(){
27     l = r = p = pp = None;
28     size = 1; rev = false;
29 }
30
31 void _node::update() {
32     size = (this != None) + l->size + r->size;
33     l->p = r->p = this;
34 }
35
36 void rotate(node v) {
37     assert(v != None && v->p != None);
38     assert(!v->rev);
39     assert(!v->p->rev);
40     node u = v->p;
41     if (v == u->l)
42         u->l = v->r, v->r = u;
43     else
44         u->r = v->l, v->l = u;
45     swap(u->p,v->p);
46     swap(v->pp,u->pp);
47     if (v->p != None) {
48         assert(v->p->l == u || v->p->r == u);
49         if (v->p->r == u)
50             v->p->r = v;
51         else
52             v->p->l = v;
53     }
54     u->update();
55     v->update();
56 }
57
58 void bigRotate(node v) {
59     assert(v->p != None);
60     v->p->p->push();
61     v->p->push();
62     v->push();
63     if (v->p->p != None) {
64         if ((v->p->l == v) ^ (v->p->p->r == v->p))
65             rotate(v->p);
66         else
67             rotate(v);
68    }
69    rotate(v);
70 }
71
72 inline void splay(node v) {
73     while (v->p != None)
74         bigRotate(v);
75 }
76
77 inline void splitAfter(node v) {
78     v->push();
79     splay(v);
80     v->r->p = None;
81     v->r->pp = v;
82     v->r = None;
83     v->update();
84 }
85
86 void expose(int x) {
87     node v = v2n[x];
88     splitAfter(v);
89     while (v->pp != None) {
90         assert(v->p == None);
```

```
91         splitAfter(v->pp);
92         assert(v->pp->r == None);
93         assert(v->pp->p == None);
94         assert(!v->pp->rev);
95         v->pp->r = v;
96         v->pp->update();
97         v = v->pp;
98         v->r->pp = None;
99     }
100    assert(v->p == None);
101    splay(v2n[x]);
102 }
103
104 inline void makeRoot(int x) {
105     expose(x);
106     assert(v2n[x]->p == None);
107     assert(v2n[x]->pp == None);
108     assert(v2n[x]->r == None);
109     v2n[x]->rev ^= 1;
110 }
111
112 inline void link(int x, int y) {
113     makeRoot(x);
114     v2n[x]->pp = v2n[y];
115 }
116
117 inline void cut(int x, int y) {
118     expose(x);
119     splay(v2n[y]);
120     if (v2n[y]->pp != v2n[x]) {
121         swap(x,y);
122         expose(x);
123         splay(v2n[y]);
124         assert(v2n[y]->pp == v2n[x]);
125     }
126     v2n[y]->pp = None;
127 }
128
129 inline int get(int x, int y) {
130     if (x == y)
131         return 0;
132     makeRoot(x);
133     expose(y);
134     expose(x);
135     splay(v2n[y]);
136     if (v2n[y]->pp != v2n[x])
137         return -1;
138     return v2n[y]->size;
139 }
140
141 }
```

26　structures/ordered_set.cpp

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3
4 typedef __gnu_pbds::tree<int, __gnu_pbds::null_type,
5         std::less<int>,
6         t__gnu_pbds::rb_tree_tag,
7         __gnu_pbds::tree_order_statistics_node_update> oset;
8
9 #include <iostream>
10
11 int main() {
12     oset X;
13     X.insert(1);
14     X.insert(2);
15     X.insert(4);
16     X.insert(8);
17     X.insert(16);
18
19     std::cout << *X.find_by_order(1) << std::endl; // 2
20     std::cout << *X.find_by_order(2) << std::endl; // 4
21     std::cout << *X.find_by_order(4) << std::endl; // 16
22     std::cout << std::boolalpha <<
↪   (end(X)==X.find_by_order(6)) << std::endl; // true
23
24     std::cout << X.order_of_key(-5) << std::endl;  // 0
25     std::cout << X.order_of_key(1) << std::endl;   // 0
26     std::cout << X.order_of_key(3) << std::endl;   // 2
27     std::cout << X.order_of_key(4) << std::endl;   // 2
28     std::cout << X.order_of_key(400) << std::endl; // 5
29 }
```

27　structures/treap.cpp

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