Contents

0 Basic

0.1 Compiling

```
g++ std=c++17 -O2 -Wall -Wextra -Wshadow -Wconversion -
    fsanitize=address -fsanitize=undefined "%" -o "%:r"
```

0.2 .vimrc

```
sy on
set ru nu cin cul sc so=3 ts=4 sw=4 bs=2 ls=2
inoremap {<CR> {<CR>}<C-o>O
map <F7> :w<CR>:!g++ "%" -Wall -Wextra -Wshadow -Wconversion
     -fsanitize=address -fsanitize=undefined -
    D_GLIBCXX_DEBUG -o /owo/run<CR>:!/owo/run<CR>
```

0.3 spiltmix64

```
struct custom_hash
{
    static uint64_t splitmix64(uint64_t x)
    {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const
    {
        static const uint64_t FIXED_RANDOM = chrono::
            steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```

0.4 mt19937

```
mt19937 rd(chrono::steady_clock::now().time_since_epoch().
    count());
```

0.5 PBDS

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
tree<int, null_type, less<int>, rb_tree_tag,
     tree_order_statistics_node_update> bst;
// order_of_key(n): # of elements <= n
// find_by_order(n): 0-indexed

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
__gnu_pbds::priority_queue<int, greater<int>,
     pairing_heap_tag> pq;
```

1 Graph

1.1 Dinic

```
template <int V>
struct dinic
{
  const ll INF = 1e18;
  ll adjptr[V + 1], dis[V + 1], vis[V + 1], s, t, ans;
  struct edge
  {
    ll u, v, cap;
  };
  vector<edge> e;
  vector<int> adj[V + 1];
  bool bfs()
  {
    for (int i = 1; i <= V; i++)
      dis[i] = (i == s ? 0 : V + 1);
    queue<int> q;
    q.push(s);
    while (!q.empty())
    {
      int u = q.front();
      q.pop();
      for (auto id : adj[u])
      {
        auto [_, v, cap] = e[id];
        if (dis[v] > V && cap > 0)
        {
          dis[v] = dis[u] + 1;
          q.push(v);
        }
      }
    }
    return dis[t] <= V;
  }

  vector<pii> find_cut()
  {
    vector<pii> cut;
    bfs();
    for (int i = 0; i < e.size(); i += 2)
      if (dis[e[i].u] <= V && dis[e[i].v] > V)
        cut.emplace_back(pii(e[i].u, e[i].v));
    return cut;
  }
  ll dfs(int u, ll flow)
  {
    if (u == t || !flow)
      return flow;
    for (; adjptr[u] < (int)adj[u].size(); adjptr[u]++)
    {
      int d = adj[u][adjptr[u]];
      auto [_, v, cap] = e[d];
      if (cap <= 0 || vis[v] || dis[v] != dis[u] + 1)
        continue;
      vis[v] = 1;
      ll f = dfs(v, min(flow, cap));
      if (f)
      {
        e[d].cap -= f, e[d ^ 1].cap += f;
        return f;
      }
    }
    return 0;
  }
  void add_edge(int u, int v, ll c)
  {
    adj[u].emplace_back(e.size());
    e.emplace_back(edge{u, v, c});
    adj[v].emplace_back(e.size());
    e.emplace_back(edge{v, u, 0});
  }
  ll flow(int S, int T)
  {
    s = S, t = T;
    for (int i = 0; i < (int)e.size(); i += 2)
      e[i].cap = e[i].cap + e[i ^ 1].cap, e[i ^ 1].cap = 0;
    while (bfs())
    {
      for (int i = 1; i <= V; i++)
        adjptr[i] = 0, vis[i] = 0;
      ll f = 1;
      while (f)
        ans += (f = dfs(s, INF));
    }
    return ans;
  }
};
```

2 Data Structure

2.1 Treap

```
struct treap
{
    int pri, size;
    // val
    treap *l, *r;
    treap(/*val*/) : pri(rd()), size(1), l(nullptr), r(
         nullptr){};
    treap() : pri(rd()), size(1), l(nullptr), r(nullptr){};
    void pull()
    {
        size = (l == nullptr ? 0 : l->size) + (r == nullptr
             ? 0 : r->size) + 1;
    }
    void push();
};

void spilt(int cnt, treap *root, treap *&left, treap *&right
     )
{
    if (root == nullptr)
    {
        left = right = nullptr;
        return;
    }
    root->push();
    int lsize = (root->l != nullptr ? root->l->size : 0);
    if (cnt >= lsize + 1)
    {
        left = root;
        spilt(cnt - lsize - 1, left->r, left->r, right);
        left->pull();
    }
    else
    {
        right = root;
        spilt(cnt, right->l, left, right->l);
        right->pull();
    }
}

treap *merge(treap *left, treap *right)
{
    if (left == nullptr)
        return right;
    if (right == nullptr)
        return left;
    if (left->pri < right->pri)
    {
        left->push();
        left->r = merge(left->r, right);
        left->pull();
        return left;
    }
    else
    {
        right->push();
        right->l = merge(left, right->l);
        right->pull();
        return right;
    }
}
```

3 Math

3.1 Eratosthenes Sieve

```
struct Eseive // O(N log log N / 6)
{
    int N;
    bitset<100000005> notprime;
    void calc(int k);
    void seive()
    {
        auto id = [&](int k)
        { return k / 6 * 2 - (k % 6 == 1); };

        calc(2), calc(3);
        for (int i = 5; i <= N; i += (i % 6 == 1 ? 4 : 2))
        {
            if (!notprime[id(i)])
                calc(i);
            for (int j = i; i * j <= N; j += (j % 6 == 1 ? 4
                  : 2))
                notprime[id(i * j)] = 1;
        }
    }
};
```

3.2 Linear Sieve

```
bool np[maxN + 1];
int mu[maxN + 1], phi[maxN + 1];
vector<int> prime;
inline void build()
{
    np[0] = np[1] = 1;
    mu[1] = 1;
    phi[1] = 1;
    for (int i = 2; i <= maxN; i++)
    {
        if (!np[i])
        {
            prime.push_back(i);
            mu[i] = -1;
            phi[i] = i - 1;
        }
        for (auto j : prime)
        {
            int t = i * j;
            if (t > maxN)
                break;
            np[t] = 1;
            if (i % j == 0)
            {
                mu[t] = 0;
                phi[t] = phi[i] * j;
                break;
            }
            else
            {
                mu[t] = -mu[i];
                phi[t] = phi[i] * (j - 1);
            }
        }
    }
}
```

3.3 Extended GCD

```
// beware of negative numbers!
void extgcd(ll a, ll b, ll c, ll &x, ll &y)
{
    if (b == 0)
        x = c / a, y = 0;
    else
    {
        extgcd(b, a % b, c, y, x);
        y -= x * (a / b);
    }
}
```

3.4 FFT

```
#define base complex<double>
const double PI = acosl(-1);
base omega[4 * N], omega_[4 * N];
int rev[4 * N];

void calcW(int n)
{
    double arg = 2.0 * PI / n;
    for (int i = 0; i < n; i++)
    {
        omega[i] = base(cos(i * arg), sin(i * arg));
        omega_[i] = base(cos(-i * arg), sin(-i * arg));
    }
}

/* NTT:
void calcW(int n)
{
    ll r = fpow(g, (mod - 1) / n), invr = inverse(r);
    omega[0] = omega_[0] = 1;
    for (int i = 1; i < n; i++)
    {
        omega[i] = omega[i - 1] * r % mod;
        omega_[i] = omega_[i - 1] * invr % mod;
    }
}
*/
```

```
void calcrev(int n)
{
    int k = __lg(n);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < k; j++)
            if (i & (1 << j))
                rev[i] ^= 1 << (k - j - 1);
}

vector<base> FFT(vector<base> &poly, bool inv)
{
    base *w = (inv ? omega_ : omega);
    int n = poly.size(), k = __lg(n);
    for (int i = 0; i < n; i++)
        if (rev[i] > i)
            swap(poly[i], poly[rev[i]]);

    for (int len = 1; len < n; len <<= 1)
    {
        int arg = n / len / 2;
        for (int i = 0; i < n; i += 2 * len)
            for (int j = 0; j < len; j++)
            {
                base odd = w[j * arg] * poly[i + j + len];
                poly[i + j + len] = poly[i + j] - odd;
                poly[i + j] += odd;
            }
    }
    if (inv)
        for (auto &a : poly)
            a /= n;
    return poly;
}

vector<base> mul(vector<base> f, vector<base> g)
{
    int sz = 1 << (__lg(f.size() + g.size() - 1) + 1);
    f.resize(sz), g.resize(sz);
    calcrev(sz);
    calcW(sz);
    f = FFT(f, 0), g = FFT(g, 0);
    for (int i = 0; i < sz; i++)
        f[i] *= g[i];
    return FFT(f, 1);
}
```

3.5 NTT

$p = 1004535809 = 479 \times 2^{21} + 1, g = 3$

$p = 998244353 = 119 \times 2^{23} + 1, g = 3$

$p = 9223372036737335297 = 54975513881 \cdot 2^{24} + 1, g = 3$

$\omega = g^{\frac{p-1}{n}}$

4 Geometry

4.1 Basic

```
pll operator+(pll p1, pll p2)
{ return pll(p1.F + p2.F, p1.S + p2.S); }
pll operator-(pll p1, pll p2)
{ return pll(p1.F - p2.F, p1.S - p2.S); }
pdd operator*(pdd p, double d)
{ return pdd(p.F * d, p.S * d); }
pdd operator/(pdd p, double d)
{ return pdd(p.F / d, p.S / d); }
ll det(ll a, ll b, ll c, ll d)
{ return a * d - b * c; }
ll det(pll p1, pll p2)
{ return det(p1.F, p1.S, p2.F, p2.S); }
ll dot(pll a, pll b)
{ return a.F * b.F + a.S * b.S; }
ll cross(pll p, pll a, pll b)
{ return det(a - p, b - p); }
int ori(pll p, pll a, pll b)
{
    ll d = cross(p, a, b);
    return d < 0 ? -1 : (d == 0 ? 0 : 1);
}
bool btw(pll p, pll a, pll b)
{
    return ori(p, a, b) == 0 && dot(a - p, b - p) <= 0;
}
bool intersect(pll p1, pll p2, pll p3, pll p4)
{
    ll s123 = ori(p1, p2, p3), s124 = ori(p1, p2, p4),
       s341 = ori(p3, p4, p1), s342 = ori(p3, p4, p2);
    if (s123 == 0 && s124 == 0)
        return btw(p3, p1, p2) || btw(p4, p1, p2) ||
               btw(p1, p3, p4) || btw(p2, p3, p4);
    return s123 * s124 <= 0 && s341 * s342 <= 0;
}
pdd intersection(pll p1, pll p2, pll p3, pll p4)
{
    double a123 = cross(p1, p2, p3),
           a124 = -cross(p1, p2, p4);
    return (p4 * a123 + p3 * a124) / (a123 + a124);
}
ll dis2(pll p, pll q)
{ return dot(p - q, p - q); }

signed main()
{
    pdd p = intersection(pll(0, 0), pll(3, 3), pll(-1, 2),
        pll(4, -1));
    cout << p.F << ' ' << p.S;
}
```

4.2 Minimum Euclidean Distance

```
pll p[200005];
ll d = 8e18;
void dc(int l, int r)
{
    if (l == r)
        return;
    int mid = (l + r) / 2, midx = p[mid].F;
    dc(l, mid);
    dc(mid + 1, r);
    vector<pll> v;
    for (int i = l; i <= r; i++)
        if (abs(p[i].F - midx) * abs(p[i].F - midx) <= d)
            v.emplace_back(p[i]);
    sort(v.begin(), v.end(), [](pll p1, pll p2)
         { return p1.S < p2.S; });
    for (int i = 0; i < v.size(); i++)
        for (int j = i + 1; j <= i + 5 && j < v.size(); j++)
            d = min(dis2(v[i], v[j]), d);
}
```

4.3 Convex Hull

```
vector<pll> p;

vector<pll> convex(vector<pll> v)
{
    vector<pll> h;
    sort(v.begin(), v.end());
    for (int t = 0; t < 2; t++)
    {
        for (pll p : v)
        {
            while (h.size() >= 2 && (ori(h.back(), h[h.size
                () - 2], p) < 0 ||
                                     btw(h.back(), h[h.size
                                         () - 2], p)))
                h.pop_back();
            // if (h.empty() || p != h.back()) // keep
                colinear
            h.emplace_back(p);
        }
        reverse(v.begin(), v.end());
    }
    h.pop_back();
    return h;
}
```

4.4 Angle Sort

```
int quad(pll p)
{
    // alternative equal
    if (p.F > 0 && p.S >= 0)
        return 1;
    if (p.F <= 0 && p.S > 0)
        return 2;
    if (p.F < 0 && p.S <= 0)
        return 3;
    if (p.F >= 0 && p.S < 0)
        return 4;
    return 0;
}

auto angle_sort = [](pll p1, pll p2)
{
    int q1 = quad(p1), q2 = quad(p2);
    if (q1 != q2)
        return q1 < q2;
    return p1.S * p2.F < p2.S * p1.F; };
```

4.5 Pick's Theorem

$A = I + \frac{B}{2} - 1$

Area A, lattice points on Boundary B, in interior I.

5 String

5.1 Primes

1000000021, 1066636547, 1111111121, 1234567891, 1800399103 1820303297, 2000000011, 5681288813, 21182087419

5.2 KMP

```
int KMP(string s, string t)
{
    cin >> s >> t;
    int n = t.size(), ans = 0;
    vector<int> f(t.size(), 0);
    f[0] = -1;
    for (int i = 1, j = -1; i < t.size(); i++)
    {
        while (j >= 0)
            if (t[j + 1] == t[i]) break;
            else j = f[j];
        f[i] = ++j;
    }
    for (int i = 0, j = 0; i < s.size(); i++)
    {
        while (j >= 0)
            if (t[j + 1] == s[i]) break;
            else j = f[j];
        if (++j + 1 == t.size())
            ans++, j = f[j];
    }
    return ans;
}
```

5.3 Z

```
void z(string s)
{
    for (int i = 1, mx = 0; i < s.size(); i++)
    {
        if (i < Z[mx] + mx)
            Z[i] = min(Z[mx] - i + mx, Z[i - mx]);
        while (Z[i] + i < s.size() && s[i + Z[i]] == s[Z[i
             ]])
            Z[i]++;
        if (Z[i] + i > Z[mx] + mx)
            mx = i;
    }
}
```

5.4 Manacher

```
int man[2000006];
int manacher(string s)
{
    string t;
    for (int i = 0; i < s.size(); i++)
    {
        if (i)
            t.push_back('$');
        t.push_back(s[i]);
    }
    int mx = 0, ans = 0;
    for (int i = 0; i < t.size(); i++)
    {
        man[i] = 1;
        man[i] = min(man[mx] + mx - i, man[2 * mx - i]);
        while (man[i] + i < t.size() && i - man[i] >= 0 && t
             [i + man[i]] == t[i - man[i]])
            man[i]++;
        if (i + man[i] > mx + man[mx])
            mx = i;
    }
    for (int i = 0; i < t.size(); i++)
        ans = max(ans, man[i] - 1);
    return ans;
}
```

5.5 Minimum Rotation

```
int mincyc(string s)
{
    int n = s.size();
    s = s + s;
    int i = 0, ans = 0;
    while (i < n)
    {
        ans = i;
        int j = i + 1, k = i;
        while (j < s.size() && s[j] >= s[k])
        {
            k = (s[j] > s[k] ? i : k + 1);
            ++j;
        }
        while (i <= k)
            i += j - k;
    }
    return ans;
}
```

5.6 Aho Corasick

```
#define sigma 26
struct AhoCorasick
{
    int ch[500005][sigma] = {{}}, f[500005] = {-1}, cnt
         [500005];
    int idx = 0;
    int insert(string &s)
    {
        int j = 0;
        for (int i = 0; i < s.size(); i++)
        {
            if (!ch[j][s[i] - 'a'])
                ch[j][s[i] - 'a'] = ++idx;
            j = ch[j][s[i] - 'a'];
        }
        return j;
    }
    void build()
    {
        queue<int> q;
        q.push(0);
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            for (int v = 0; v < sigma; v++)
                if (ch[u][v])
                {
                    int cur = f[u];
                    while (cur >= 0)
                        if (ch[cur][v])
                            break;
                        else
                            cur = f[cur];
                    f[ch[u][v]] = (cur < 0 ? 0 : ch[cur][v])
                        ;
                    q.push(ch[u][v]);
                }
        }
    }
    void match(string &s)
    {
        for (int i = 0; i <= idx; i++)
            cnt[i] = 0;
        for (int i = 0, j = 0; i < s.size(); i++)
        {
            while (j >= 0)
                if (ch[j][s[i] - 'a'])
                    break;
                else
                    j = f[j];
            j = (j < 0 ? 0 : ch[j][s[i] - 'a']);
            cnt[j]++;
        }
        vector<int> v;
        v.emplace_back(0);
        for (int i = 0; i < v.size(); i++)
            for (int j = 0; j < sigma; j++)
                if (ch[v[i]][j])
                    v.emplace_back(ch[v[i]][j]);
        reverse(v.begin(), v.end());
        for (int i : v)
            if (f[i] > 0)
                cnt[f[i]] += cnt[i];
    }
};
```

5.7 Suffix Array

```
void SA(string s)
{
    int n = s.size();
    sa.resize(n), cnt.resize(n), rk.resize(n), tmp.resize(n)
        ;
    iota(sa.begin(), sa.end(), 0);
    sort(sa.begin(), sa.end(), [&](int i, int j)
        { return s[i] < s[j]; });
    for (int i = 1; i < n; i++)
        rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i - 1]] != s[sa[i
             ]]);
    for (int k = 1; k <= n; k <<= 1)
    {
        fill(cnt.begin(), cnt.end(), 0);
        for (int i = 0; i < n; i++)
            cnt[rk[(sa[i] - k + n) % n]]++;
        for (int i = 1; i < n; i++)
```

```
            cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; i--)
            tmp[--cnt[rk[(sa[i] - k + n) % n]]] = (sa[i] - k
                  + n) % n;
        sa.swap(tmp);
        tmp[sa[0]] = 0;
        for (int i = 1; i < n; i++)
            tmp[sa[i]] = tmp[sa[i - 1]] + (rk[sa[i - 1]] !=
                 rk[sa[i]] || rk[(sa[i - 1] + k) % n] != rk
                 [(sa[i] + k) % n]);
        rk.swap(tmp);
    }
}

void LCP(string s)
{
    int n = s.size(), k = 0;
    lcp.resize(n);
    for (int i = 0; i < n; i++)
        if (rk[i] == 0)
            lcp[rk[i]] = 0;
        else
        {
            if (k)
                k--;
            int j = sa[rk[i] - 1];
            while (i + k < n && j + k < n && s[i + k] == s[j
                  + k])
                k++;
            lcp[rk[i]] = k;
        }
}
```