Notebook - Maratona de Programação

[UnB] HatsuneMiku não manda WA

Contents

1 Informações

1.1 Compilação e Execução

Comandos de compilação

- C++: g++ -std=c++17 -g3 -fsanitize=address -Wall -Wconversion -Wshadow -o <executável> <arquivo>.cpp
- Java: javac <arquivo>.java.
- Haskell: ghc -o <executável> <arquivo>.hs.

Comandos de execução

- C++:./<executável>.
- Java: java -Xms1024m -Xmx1024m -Xss20m <arquivo>.
- Python: python3 <arquivo>.py.
- Haskell: ./<executável>.

1.2 Ferramentas para Testes

Python

```python
import random
import itertools

#randint: retorna um numero aleatorio x tq. a <= x <= b
lista = [random.randint(1,100) for i in range(101)]

#shuffle: embaralha uma sequencia
random.shuffle(lista)

#sample: retorna uma lista de k elementos unicos escolhidos de uma sequencia
amostra = random.sample(lista, k = 10)

lista2 = [1,2,3,4,5]
#permutations: iterable que retorna permutacoes de tamanho r
permutacoes = [perm for perm in itertools.permutations(lista2, 2)]

#combinations: iterable que retorna combinacoes de tamanho r (ordenado)
#combinations_with_replacement: combinations() com elementos repetidos
combinacoes = [comb for comb in itertools.combinations(lista2, 2)]

```

C++

```cpp
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count()); // mt19937_64
uniform_int_distribution<int> distribution(1, n);

num = distribution(rng); // num no range [1, n]
shuffle(vec.begin(), vec.end(), rng); // shuffle

// permutacoes
do {
    // codigo
} while(next_permutation(vec.begin(), vec.end()))

using ull = unsigned long long;
ull mix(ull o){
    o+=0x9e3779b97f4a7c15;
    o=(o^(o>>30))*0xbf58476d1ce4e5b9;
    o=(o^(o>>27))*0x94d049bb133111eb;
    return o^(o>>31);
}

ull hash(pii a) {return mix(a.first ^ mix(a.second));}

```

2 Misc

2.1 Submask

```
// O(3^n)
for (int m = 0; m < (1<<n); m++) {
    for (int s = m; s; s = (s-1) & m) {
        // s is every submask of m
    }
}

// O(2^n * n) SOS dp like
for (int b = n-1; b >= 0; b--) {
    for (int m = 0; m < (1 << n); m++) {
        if (j & (1 << b)) {
            // propagate info through submasks
            amount[j ^ (1 << b)] += amount[j];
        }
    }
}
```

2.2 Safe Map

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        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);
    }
};

unordered_map<long long, int, custom_hash> safe_map;

// when using pairs
struct custom_hash {
    inline size_t operator ()(const pii & a) const {
        return (a.first << 6) ^ (a.first >> 2) ^ 2038074743 ^ a.second;
    }
};
```

2.3 Ordered Set

```
#include <bits/extc++.h>
// or
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds; // or pb_ds;
template<typename T, typename B = null_type>
using ordered_set = tree<T, B, less<T>, rb_tree_tag, tree_order_statistics_node_update>;

// order_of_key(k)  : Number of items strictly smaller than k
// find_by_order(k) : K-th element in a set (counting from zero)

// to erase an element -> order_of_key(k) + find_by_order(k) + erase(itr)

// to swap two sets, use a.swap(b);
```

2.4 Bitwise

```
// Least significant bit (lsb)
    int lsb(int x) { return x&-x; }
    int lsb(int x) { return __builtin_ctz(x); } // bit position
// Most significant bit (msb)
    int msb(int x) { return 32-1-__builtin_clz(x); } // bit position

// Power of two
    bool isPowerOfTwo(int x){ return x && (!(x&(x-1))); }

// floor(log2(x))
int flog2(int x)  { return 32-1-__builtin_clz(x); }
int flog2ll(ll x) { return 64-1-__builtin_clzll(x); }

// Built-in functions
// Number of bits 1
__builtin_popcount()
__builtin_popcountll()

// Number of leading zeros
__builtin_clz()
__builtin_clzll()

// Number of trailing zeros
__builtin_ctz()
__builtin_ctzll()
```

2.5 Template

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

#define ll long long
#define ff first
#define ss second
#define ld long double
#define pb push_back
#define sws cin.tie(0)->sync_with_stdio(false);
#define endl '\n'
#ifdef LOCAL
#define debug(var) cout << (#var) << " = " << var << endl;
#endif
#ifndef LOCAL
#define debug(...)
#endif

const ll MOD = 998244353;
const int INF = 0x3f3f3f3f;
const ll LLINF = 0x3f3f3f3f3f3f3f3f;

signed main() {
    #ifndef LOCAL
    sws;
    #endif

    return 0;
}
```

3 DP

3.1 Knapsack

```
// Caso base, como i == n
dp[0][0] = 0;

// Itera por todos os estados
```

```
for(int i = 1; i <= n; ++i)
    for(int P = 0; P <= w; ++P){
        int &temp = dp[i][P];
        // Primeira possibilidade, ãno pega i
        temp = dp[i - 1][P];

        // Segunda possibilidade, se puder, pega o item
        if(P - p[i] >= 0)
            temp = max(temp, dp[i - 1][P - p[i]] + v[i]);

        ans = max(ans, temp);
    }
}
```

3.2 Lis

```
multiset<int> S;
for(int i=0;i<n;i++){
    auto it = S.upper_bound(vet[i]); // low for inc
    if(it != S.end())
        S.erase(it);
    S.insert(vet[i]);
}
// size of the lis
int ans = S.size();

vi LIS(const vi &elements){
    auto compare = [&](int x, int y) {
        return elements[x] < elements[y];
    };
    set< int, decltype(compare) > S(compare);

    vi previous( elements.size(), -1 );
    for(int i=0; i<int( elements.size() ); ++i){
        auto it = S.insert(i).first;
        if(it != S.begin())
            previous[i] = *prev(it);
        if(*it == i and next(it) != S.end())
            S.erase(next(it));
    }

    vi answer;
    answer.push_back( *S.rbegin() );
    while ( previous[answer.back()] != -1 )
        answer.push_back( previous[answer.back()] );
    reverse( answer.begin(), answer.end() );
    return answer;
}
```

3.3 Dp Digitos

```
// dp de quantidade de numeros <= r com ate qt digitos diferentes de 0
ll dp(int idx, string& r, bool menor, int qt, vector<vector<vi>>& tab) {
    if(qt > 3) return 0;
    if(idx >= r.size()) {
        return 1;
    }
    if(tab[idx][menor][qt] != -1)
        return tab[idx][menor][qt];

    ll res = 0;
    for(int i = 0; i <= 9; i++) {
        if(menor or i <= r[idx]-'0') {
            res += dp(idx+1, r, menor or i < (r[idx]-'0') , qt+(i>0), tab);
        }
    }

    return tab[idx][menor][qt] = res;
}
```

4 ED

4.1 Prefixsum2d

```
ll find_sum(vector<vi> &mat, int x1, int y1, int x2, int y2){
    // superior-esq(x1,y1) (x2,y2)inferior-dir
    return mat[x2][y2]-mat[x2][y1-1]-mat[x1-1][y2]+mat[x1-1][y1-1];
}

int main(){

    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            mat[i][j]+=mat[i-1][j]+mat[i][j-1]-mat[i-1][j-1];

}
```

4.2 Sparse Table

```
int logv[N+1];
void make_log() {
    logv[1] = 0; // pre-computar tabela de log
    for (int i = 2; i <= N; i++)
        logv[i] = logv[i/2] + 1;
}
struct Sparse {
    int n;
    vector<vector<int>> st;

    Sparse(vector<int>& v) {
        n = v.size();
        int k = logv[n];
        st.assign(n+1, vector<int>(k+1, 0));

        for (int i=0;i<n;i++) {
            st[i][0] = v[i];
        }

        for(int j = 1; j <= k; j++) {
            for(int i = 0; i + (1 << j) <= n; i++) {
                st[i][j] = f(st[i][j-1], st[i + (1 << (j-1))][j-1]);
            }
        }
    }

    int f(int a, int b) {
        return min(a, b);
    }

    int query(int l, int r) {
        int k = logv[r-l+1];
        return f(st[l][k], st[r - (1 << k) + 1][k]);
    }
};


struct Sparse2d {
    int n, m;
    vector<vector<vector<int>>> st;

    Sparse2d(vector<vector<int>> mat) {
        n = mat.size();
        m = mat[0].size();
        int k = logv[min(n, m)];

```

```
        st.assign(n+1, vector<vector<int>>(m+1, vector<int>(k+1)));
        for(int i = 0; i < n; i++)
            for(int j = 0; j < m; j++)
                st[i][j][0] = mat[i][j];

        for(int j = 1; j <= k; j++) {
            for(int x1 = 0; x1 < n; x1++) {
                for(int y1 = 0; y1 < m; y1++) {
                    int delta = (1 << (j-1));
                    if(x1+delta >= n or y1+delta >= m) continue;

                    st[x1][y1][j] = st[x1][y1][j-1];
                    st[x1][y1][j] = f(st[x1][y1][j], st[x1+delta][y1][j-1]);
                    st[x1][y1][j] = f(st[x1][y1][j], st[x1][y1+delta][j-1]);
                    st[x1][y1][j] = f(st[x1][y1][j], st[x1+delta][y1+delta][j-1]);
                }
            }
        }
    }

    // so funciona para quadrados
    int query(int x1, int y1, int x2, int y2) {
        assert(x2-x1+1 == y2-y1+1);
        int k = logv[x2-x1+1];
        int delta = (1 << k);

        int res = st[x1][y1][k];
        res = f(res, st[x2 - delta+1][y1][k]);
        res = f(res, st[x1][y2 - delta+1][k]);
        res = f(res, st[x2 - delta+1][y2 - delta+1][k]);
        return res;
    }

    int f(int a, int b) {
        return a | b;
    }

};
```

4.3 Dsu

```
struct DSU {
    int n;
    vector<int> parent, size;

    DSU(int n): n(n) {
        parent.resize(n, 0);
        size.assign(n, 1);

        for(int i=0;i<n;i++)
            parent[i] = i;
    }

    int find(int a) {
        if(a == parent[a]) return a;
        return parent[a] = find(parent[a]);
    }

    void join(int a, int b) {
        a = find(a); b = find(b);
        if(a != b) {
            if(size[a] < size[b]) swap(a, b);
            parent[b] = a;
            size[a] += size[b];
        }
    }
};
```

4.4 Minqueue

```
struct MinQ {
    stack<pair<ll,ll>> in;
    stack<pair<ll,ll>> out;

    void add(ll val) {
        ll minimum = in.empty() ? val : min(val, in.top().ss);
        in.push({val, minimum});
    }

    ll pop() {
        if(out.empty()) {
            while(!in.empty()) {
                ll val = in.top().ff;
                in.pop();
                ll minimum = out.empty() ? val : min(val, out.top().ss);
                out.push({val, minimum});
            }
        }
        ll res = out.top().ff;
        out.pop();
        return res;
    }

    ll minn() {
        ll minimum = LLINF;
        if(in.empty() || out.empty())
            minimum = in.empty() ? (ll)out.top().ss : (ll)in.top().ss;
        else
            minimum = min((ll)in.top().ss, (ll)out.top().ss);

        return minimum;
    }

    ll size() {
        return in.size() + out.size();
    }
};
```

4.5 Segtree Lazy

```
template <typename T> class SegTreeLazy{
  private:
    vector<T> st, lz;
    T elemNeutro;
    int n;
    T merge(const T & a, const T & b){ // #!
      return a + b;
    }
    void prop(int l, int r, int no){ // #!
      st[no] += (r - l + 1)*lz[no];
      if(l != r){
        lz[2*no] += lz[no];
        lz[2*no + 1] += lz[no];
      }
      lz[no] = 0;
    }
    void update(int gl, int gr, T x, int l, int r, int no){
      prop(l, r, no);
      if(l >= gl && r <= gr){
        lz[no] += x; // #!
        prop(l, r, no);
      } else if(l > gr || r < gl) return;
      else {
        int mid = (l + r) >> 1;
        update(gl, gr, x, l, mid, 2*no);
        update(gl, gr, x, mid + 1, r, 2*no + 1);
```

```
        st[no] = merge(st[2*no], st[2*no + 1]);
      }
    }
    T query(int gl, int gr, int l, int r, int no){
      prop(l, r, no);
      if(l >= gl && r <= gr) return st[no];
      else if(l > gr || r < gl) return elemNeutro;
      else {
        int mid = (l + r) >> 1;
        return merge(query(gl, gr, l, mid, 2*no), query(gl, gr, mid + 1, r, 2*no + 1));
      }
    }
  public:
    SegTreeLazy(int _n, T _elemNeutro){
      n = _n;
      elemNeutro = _elemNeutro;
      st.resize(4*n);
      lz.resize(4*n);
    }
    void update(int l, int r, T x){
      update(l, r, x, 0, n - 1, 1);
    }
    T query(int l, int r){
      return query(l, r, 0, n - 1, 1);
    }
};
```

4.6 Segtree Implicita

```
// SegTree Implicita O(nlogMAX)

struct node{
    int val;
    int l, r;
    node(int a=0, int b=0, int c=0){
        l=a;r=b;val=c;
    }
};

int idx=2; // 1-> root / 0-> zero element
node t[8600010];
int N;

int merge(int a, int b){
    return a + b;
}

void update(int pos, int x, int i=1, int j=N, int no=1){
    if(i==j){
        t[no].val+=x;
        return;
    }
    int meio = (i+j)/2;

    if(pos<=meio){
        if(t[no].l==0) t[no].l=idx++;
        update(pos, x, i, meio, t[no].l);
    }
    else{
        if(t[no].r==0) t[no].r=idx++;
        update(pos, x, meio+1, j, t[no].r);
    }

    t[no].val=merge(t[t[no].l].val, t[t[no].r].val);
}

int query(int A, int B, int i=1, int j=N, int no=1){
    if(B<i or j<A)
        return 0;
    if(A<=i and j<=B)
        return t[no].val;

    int mid = (i+j)/2;

    int ansl = 0, ansr = 0;

    if(t[no].l!=0) ansl = query(A, B, i, mid, t[no].l);
    if(t[no].r!=0) ansr = query(A, B, mid+1, j, t[no].r);

    return merge(ansl, ansr);
}
```

4.7 Segtree Implicita Lazy

```
struct node{
    pll val;
    ll lazy;
    ll l, r;
    node(){
        l=-1;r=-1;val={0,0};lazy=0;
    }
};

node tree[40*MAX];
int id = 2;
ll N=1e9+10;

pll merge(pll A, pll B){
    if(A.ff==B.ff) return {A.ff, A.ss+B.ss};
    return (A.ff<B.ff ? A:B);
}

void prop(ll l, ll r, int no){
    ll mid = (l+r)/2;
    if(l!=r){
        if(tree[no].l==-1){
            tree[no].l = id++;
            tree[tree[no].l].val = {0, mid-l+1};
        }
        if(tree[no].r==-1){
            tree[no].r = id++;
            tree[tree[no].r].val = {0, r-(mid+1)+1};
        }
        tree[tree[no].l].lazy += tree[no].lazy;
        tree[tree[no].r].lazy += tree[no].lazy;
    }
    tree[no].val.ff += tree[no].lazy;
    tree[no].lazy=0;
}

void update(int a, int b, int x, ll l=0, ll r=2*N, ll no=1){
    prop(l, r, no);
    if(a<=l and r<=b){
        tree[no].lazy += x;
        prop(l, r, no);
        return;
    }
    if(r<a or b<l) return;
    int m = (l+r)/2;
    update(a, b, x, l, m, tree[no].l);
    update(a, b, x, m+1, r, tree[no].r);

    tree[no].val = merge(tree[tree[no].l].val, tree[tree[no].r].val);
}

pll query(int a, int b, int l=0, int r=2*N, int no=1){
    prop(l, r, no);
    if(a<=l and r<=b) return tree[no].val;
    if(r<a or b<l) return {INF, 0};
```

```cpp
    int m = (l+r)/2;
    int left = tree[no].l, right = tree[no].r;

    return tree[no].val = merge(query(a, b, l, m, left),
                                query(a, b, m+1, r, right));
}
```

4.8 Segtree

```cpp
template <typename T> class SegTree{
  private:
    vector<T> st;
    T elemNeutro;
    int n;
    T merge(const T & a, const T & b){
      return a + b; // #!
    }
    void update(int i, T x, int l, int r, int no){
      if(l == r) st[no] += x;
      else {
        int mid = (l + r) >> 1;
        if(i <= mid) update(i, x, l, mid, 2*no);
        else update(i, x, mid + 1, r, 2*no + 1);
        st[no] = merge(st[2*no], st[2*no + 1]);
      }
    }
    T query(int gl, int gr, int l, int r, int no){
      if(l >= gl && r <= gr) return st[no];
      else if(l > gr || r < gl) return elemNeutro;
      else {
        int mid = (l + r) >> 1;
        return merge(query(gl, gr, l, mid, 2*no), query(gl, gr, mid + 1, r, 2*no + 1));
      }
    }
  public:
    SegTree(int _n, T _elemNeutro){
      n = _n;
      elemNeutro = _elemNeutro;
      st.resize(4*n);
    }
    void update(int i, T x){
      update(i, x, 0, n - 1, 1);
    }
    T query(int l, int r){
      return query(l, r, 0, n - 1, 1);
    }
};
```

4.9 Delta Encoding

```cpp
// Delta encoding

for(int i=0;i<q;i++){
    int l,r,x;
    cin >> l >> r >> x;
    delta[l] += x;
    delta[r+1] -= x;
}

int atual = 0;

for(int i=0;i<n;i++){
    atual += delta[i];
    v[i] += atual;
}
```

5 Strings

5.1 Suffix Array

```cpp
vector<int> suffix_array(string s) {
    s += "!";
    int n = s.size(), N = max(n, 260);
    vector<int> sa(n), ra(n);
    for (int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];

    for (int k = 0; k < n; k ? k *= 2 : k++) {
        vector<int> nsa(sa), nra(n), cnt(N);

        for (int i = 0; i < n; i++) nsa[i] = (nsa[i]-k+n)%n, cnt[ra[i]]++;
        for (int i = 1; i < N; i++) cnt[i] += cnt[i-1];
        for (int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]]]] = nsa[i];

        for (int i = 1, r = 0; i < n; i++) nra[sa[i]] = r += ra[sa[i]] !=
            ra[sa[i-1]] or ra[(sa[i]+k)%n] != ra[(sa[i-1]+k)%n];
        ra = nra;
        if (ra[sa[n-1]] == n-1) break;
    }
    return vector<int>(sa.begin()+1, sa.end());
}

vector<int> kasai(string s, vector<int> sa) {
    int n = s.size(), k = 0;
    vector<int> ra(n), lcp(n);
    for (int i = 0; i < n; i++) ra[sa[i]] = i;

    for (int i = 0; i < n; i++, k -= !!k) {
        if (ra[i] == n-1) { k = 0; continue; }
        int j = sa[ra[i]+1];
        while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
        lcp[ra[i]] = k;
    }
    return lcp;
}
```

5.2 Z Func

```cpp
vector<int> Z(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        z[i] = max(0, min(z[i - l], r - i + 1));
        while (i + z[i] < n and s[z[i]] == s[i + z[i]]) {
            l = i; r = i + z[i]; z[i]++;
        }
    }
    return z;
}
```

5.3 Edit Distance

```cpp
int edit_distance(int a, int b, string& s, string& t) {
    // indexado em 0, transforma s em t
    if(a == -1) return b+1;
    if(b == -1) return a+1;
    if(tab[a][b] != -1) return tab[a][b];

```

```
    int ins = INF, del = INF, mod = INF;
    ins = edit_distance(a-1, b, s, t) + 1;
    del = edit_distance(a, b-1, s, t) + 1;
    mod = edit_distance(a-1, b-1, s, t) + (s[a] != t[b]);

    return tab[a][b] = min(ins, min(del, mod));
}
```

5.4 Lcsubseq

```
// Longest Common Subsequence
string lcs(string x, string y){
    int n = x.size(), m = y.size();
    vector<vi> dp(n+1, vi(m+1, 0));

    for(int i=0;i<=n;i++){
        for(int j=0;j<=m;j++){
            if(!i or !j)
                dp[i][j]=0;
            else if(x[i-1] == y[j-1])
                dp[i][j]=dp[i-1][j-1]+1;
            else
                dp[i][j]=max(dp[i-1][j], dp[i][j-1]);
        }
    }

    // int len = dp[n][m];
    string ans="";

    // recover string
    int i = n-1, j = m-1;
    while(i>=0 and j>=0){
        if(x[i] == y[j]){
            ans.pb(x[i]);
            i--; j--;
        }else if(dp[i][j+1]>dp[i+1][j])
            i--;
        else
            j--;
    }

    reverse(ans.begin(), ans.end());

    return ans;
}
```

5.5 Kmp

```
string p;
int neighbor[N];
int walk(int u, char c) { // leader after inputting `c`
    while (u != -1 && (u+1 >= (int)p.size() || p[u + 1] != c)) // leader doesn't match
        u = neighbor[u];
    return p[u + 1] == c ? u+1 : u;
}
void build() {
    neighbor[0] = -1; // -1 is the leftmost state
    for (int i = 1; i < (int)p.size(); i++)
        neighbor[i] = walk(neighbor[i-1], p[i]);
}
```

5.6 Hash

```
// String Hash template
// constructor(s) - O(|s|)
// query(l, r) - returns the hash of the range [l,r] from left to right - O(1)
// query_inv(l, r) from right to left - O(1)

struct Hash {
    const ll P = 31;
    int n; string s;
    vector<ll> h, hi, p;
    Hash() {}
    Hash(string s): s(s), n(s.size()), h(n), hi(n), p(n) {
        for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1) % MOD;
        for (int i=0;i<n;i++)
            h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
        for (int i=n-1;i>=0;i--)
            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P) % MOD;
    }
    int query(int l, int r) {
        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
    int query_inv(int l, int r) {
        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l+1] % MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
};
```

5.7 Aho Corasick

```
// https://github.com/joseleite19/icpc-notebook/blob/master/code/string/aho_corasick.cpp
const int A = 26;
int to[N][A];
int ne = 2, fail[N], term[N];
void add_string(string str, int id){
    int p = 1;
    for(auto c: str){
        int ch = c - 'a'; // !
        if(!to[p][ch]) to[p][ch] = ne++;
        p = to[p][ch];
    }
    term[p]++;
}
void init(){
    for(int i = 0; i < ne; i++) fail[i] = 1;
    queue<int> q; q.push(1);
    int u, v;
    while(!q.empty()){
        u = q.front(); q.pop();
        for(int i = 0; i < A; i++){
            if(to[u][i]){
                v = to[u][i]; q.push(v);
                if(u != 1){
                    fail[v] = to[ fail[u] ][i];
                    term[v] += term[ fail[v] ];
                }
            }
            else if(u != 1) to[u][i] = to[ fail[u] ][i];
            else to[u][i] = 1;
        }
    }
}
```

5.8 Lcs

```
string LCSubStr(string X, string Y)
{
    int m = X.size();
    int n = Y.size();

    int result = 0, end;
```

```
    int len[2][n];
    int currRow = 0;

    for(int i=0;i<=m;i++){
        for(int j=0;j<=n;j++){
            if(i==0 || j==0)
                len[currRow][j] = 0;
            else if(X[i-1] == Y[j-1]){
                len[currRow][j] = len[1-currRow][j-1] + 1;
                if(len[currRow][j] > result){
                    result = len[currRow][j];
                    end = i - 1;
                }
            }
            else
                len[currRow][j] = 0;
        }

        currRow = 1 - currRow;
    }

    if(result==0)
        return string();

    return X.substr(end - result + 1, result);
}
```

6 Geometria

6.1 Polygon Diameter

```
pair<point, point> polygon_diameter(vp p) {
    p = convex_hull(p);
    int n = p.size(), j = n<2 ? 0:1;
    pair<ll, vp> res({0, {p[0], p[0]}});
    for (int i=0;i<j;i++){
        for (;; j = (j+1) % n) {
            res = max(res, {norm2(p[i] - p[j]), {p[i], p[j]}});
            if ((p[(j + 1) % n] - p[j]) ^ (p[i + 1] - p[i]) >= 0)
                break;
        }
    }
    return res.second;
}

double diameter(const vector<point> &p) {
    vector<point> h = convexHull(p);
    int m = h.size();
    if (m == 1)
        return 0;
    if (m == 2)
        return dist(h[0], h[1]);
    int k = 1;
    while (area(h[m - 1], h[0], h[(k + 1) % m]) > area(h[m - 1], h[0], h[k]))
        ++k;
    double res = 0;
    for (int i = 0, j = k; i <= k && j < m; i++) {
        res = max(res, dist(h[i], h[j]));
        while (j < m && area(h[i], h[(i + 1) % m], h[(j + 1) % m]) > area(h[i], h[(i + 1) % m], h[j])) {
            res = max(res, dist(h[i], h[(j + 1) % m]));
            ++j;
        }
    }
    return res;
}
```

6.2 Mindistpair

```
ll MinDistPair(vp &vet){
    int n = vet.size();
    sort(vet.begin(), vet.end());
    set<point> s;

    ll best_dist = LLINF;
    int j=0;
    for(int i=0;i<n;i++){
        ll d = ceil(sqrt(best_dist));
        while(j<n and vet[i].x-vet[j].x >= d){
            s.erase(point(vet[j].y, vet[j].x));
            j++;
        }

        auto it1 = s.lower_bound({vet[i].y - d, vet[i].x});
        auto it2 = s.upper_bound({vet[i].y + d, vet[i].x});

        for(auto it=it1; it!=it2; it++){
            ll dx = vet[i].x - it->y;
            ll dy = vet[i].y - it->x;
            if(best_dist > dx*dx + dy*dy){
                best_dist = dx*dx + dy*dy;
                // vet[i] e inv(it)
            }
        }

        s.insert(point(vet[i].y, vet[i].x));
    }
    return best_dist;
}
```

6.3 Inside Polygon

```
// Convex O(logn)

bool insideT(point a, point b, point c, point e){
    int x = ccw(a, b, e);
    int y = ccw(b, c, e);
    int z = ccw(c, a, e);
    return !((x==1 or y==1 or z==1) and (x==-1 or y==-1 or z==-1));
}

bool inside(vp &p, point e){ // ccw
    int l=2, r=(int)p.size()-1;
    while(l<r){
        int mid = (l+r)/2;
        if(ccw(p[0], p[mid], e) == 1)
            l=mid+1;
        else{
            r=mid;
        }
    }
    // bordo
    // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)==0) return false;
    // if(r==2 and ccw(p[0], p[1], e)==0) return false;
    // if(ccw(p[r], p[r-1], e)==0) return false;
    return insideT(p[0], p[r-1], p[r], e);
}


// Any O(n)

int inside(vp &p, point pp){
    // 1 - inside / 0 - boundary / -1 - outside
    int n = p.size();
    for(int i=0;i<n;i++){
```

```
        int j = (i+1)%n;
        if(line({p[i], p[j]}).inside_seg(pp))
            return 0;
    }
    int inter = 0;
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p[i], p[j], pp)==1)
            inter++; // up
        else if(p[j].x <= pp.x and pp.x < p[i].x and ccw(p[i], p[j], pp)==-1)
            inter++; // down
    }

    if(inter%2==0) return -1; // outside
    else return 1; // inside
}
```

6.4 Polygon Cut Length

```
// Polygon Cut length
ld solve(vp &p, point a, point b){ // ccw
    int n = p.size();
    ld ans = 0;

    for(int i=0;i<n;i++){
        int j = (i+1) % n;

        int signi = ccw(a, b, p[i]);
        int signj = ccw(a, b, p[j]);

        if(signi == 0 and signj == 0){
            if((b-a) * (p[j]-p[i]) > 0){
                ans += param(a, b, p[j]);
                ans -= param(a, b, p[i]);
            }
        }else if(signi <= 0 and signj > 0){
            ans -= param(a, b, inter_line({a, b}, {p[i], p[j]})[0]);
        }else if(signi > 0 and signj <= 0){
            ans += param(a, b, inter_line({a, b}, {p[i], p[j]})[0]);
        }
    }

    return abs(ans * norm(b-a));
}
```

6.5 3d

```
// typedef ll cod;
// bool eq(cod a, cod b){ return (a==b); }

const ld EPS = 1e-6;
#define vp vector<point>
typedef ld cod;
bool eq(cod a, cod b){ return fabs(a - b) <= EPS; }

struct point
{
    cod x, y, z;
    point(cod x=0, cod y=0, cod z=0): x(x), y(y), z(z) {}

    point operator+(const point &o) const {
        return {x+o.x, y+o.y, z+o.z};
    }
    point operator-(const point &o) const {
        return {x-o.x, y-o.y, z-o.z};
    }
    point operator*(cod t) const {
        return {x*t, y*t, z*t};
    }
    point operator/(cod t) const {
        return {x/t, y/t, z/t};
    }
    bool operator==(const point &o) const {
        return eq(x, o.x) and eq(y, o.y) and eq(z, o.z);
    }
    cod operator*(const point &o) const { // dot
        return x*o.x + y*o.y + z*o.z;
    }
    point operator^(const point &o) const { // cross
        return point(y*o.z - z*o.y,
                     z*o.x - x*o.z,
                     x*o.y - y*o.x);
    }
};

ld norm(point a) { // Modulo
    return sqrt(a * a);
}
cod norm2(point a) {
    return a * a;
}
bool nulo(point a) {
    return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0));
}
ld proj(point a, point b) { // a sobre b
    return (a*b)/norm(b);
}
ld angle(point a, point b) { // em radianos
    return acos((a*b) / norm(a) / norm(b));
}

cod triple(point a, point b, point c) {
    return (a * (b^c)); // Area do paralelepipedo
}

point normilize(point a) {
    return a/norm(a);
}

struct plane {
    cod a, b, c, d;
    point p1, p2, p3;
    plane(point p1=0, point p2=0, point p3=0): p1(p1), p2(p2), p3(p3) {
        point aux = (p1-p3)^(p2-p3);
        a = aux.x; b = aux.y; c = aux.z;
        d = -a*p1.x - b*p1.y - c*p1.z;
    }
    plane(point p, point normal) {
        normal = normilize(normal);
        a = normal.x; b = normal.y; c = normal.z;
        d = -(p*normal);
    }

    // ax+by+cz+d = 0;
    cod eval(point &p) {
        return a*p.x + b*p.y + c*p.z + d;
    }
};

cod dist(plane pl, point p) {
    return fabs(pl.a*p.x + pl.b*p.y + pl.c*p.z + pl.d) / sqrt(pl.a*pl.a + pl.b*pl.b + pl.c*pl.c);
}

point rotate(point v, point k, ld theta) {
    // Rotaciona o vetor v theta graus em torno do eixo k
```

```
    // theta *= PI/180; // graus
    return (
        v*cos(theta)) +
        ((k^v)*sin(theta)) +
        (k*(k*v))*(1-cos(theta)
    );
}

// 3d line inter / mindistance
cod d(point p1, point p2, point p3, point p4) {
    return (p2-p1) * (p4-p3);
}
vector<point> inter3d(point p1, point p2, point p3, point p4) {
    cod mua = ( d(p1, p3, p4, p3) * d(p4, p3, p2, p1) - d(p1, p3, p2, p1) * d(p4, p3, p4, p3) )
            / ( d(p2, p1, p2, p1) * d(p4, p3, p4, p3) - d(p4, p3, p2, p1) * d(p4, p3, p2, p1) );
    cod mub = ( d(p1, p3, p4, p3) + mua * d(p4, p3, p2, p1) ) / d(p4, p3, p4, p3);
    point pa = p1 + (p2-p1) * mua;
    point pb = p3 + (p4-p3) * mub;
    if (pa == pb) return {pa};
    return {};
}
```

6.6 Convex Hull

```
vp convex_hull(vp P)
{
    sort(P.begin(), P.end());
    vp L, U;
    for(auto p: P){
        while(L.size()>=2 and ccw(L.end()[-2], L.back(), p)!=1)
            L.pop_back();
        L.push_back(p);
    }
    reverse(P.begin(), P.end());
    for(auto p: P){
        while(U.size()>=2 and ccw(U.end()[-2], U.back(), p)!=1)
            U.pop_back();
        U.push_back(p);
    }
    L.pop_back();
    L.insert(L.end(), U.begin(), U.end()-1);
    return L;
}
```

6.7 Linear Transformation

```
// Apply linear transformation (p -> q) to r.
point linear_transformation(point p0, point p1, point q0, point q1, point r) {
    point dp = p1-p0, dq = q1-q0, num((dp^dq), (dp*dq));
    return q0 + point((r-p0)^(num), (r-p0)*(num))/(dp*dp);
}
```

6.8 Voronoi

```
bool polygonIntersection(line &seg, vp &p) {
    long double l = -1e18, r = 1e18;
    for(auto ps : p) {
        long double z = seg.eval(ps);
        l = max(l, z);
        r = min(r, z);
    }
    return l - r > EPS;
}

int w, h;

line getBisector(point a, point b) {
    line ans(a, b);
    swap(ans.a, ans.b);
    ans.b *= -1;
    ans.c = ans.a * (a.x + b.x) * 0.5 + ans.b * (a.y + b.y) * 0.5;
    return ans;
}

vp cutPolygon(vp poly, line seg) {
    int n = (int) poly.size();
    vp ans;
    for(int i = 0; i < n; i++) {
        double z = seg.eval(poly[i]);
        if(z > -EPS) {
            ans.push_back(poly[i]);
        }
        double z2 = seg.eval(poly[(i + 1) % n]);
        if((z > EPS && z2 < -EPS) || (z < -EPS && z2 > EPS)) {
            ans.push_back(inter_line(seg, line(poly[i], poly[(i + 1) % n]))[0]);
        }
    }
    return ans;
}

// BE CAREFUL!
// the first point may be any point
// O(N^3)
vp getCell(vp pts, int i) {
    vp ans;
    ans.emplace_back(0, 0);
    ans.emplace_back(1e6, 0);
    ans.emplace_back(1e6, 1e6);
    ans.emplace_back(0, 1e6);
    for(int j = 0; j < (int) pts.size(); j++) {
        if(j != i) {
            ans = cutPolygon(ans, getBisector(pts[i], pts[j]));
        }
    }
    return ans;
}

// O(N^2) expected time
vector<vp> getVoronoi(vp pts) {
    // assert(pts.size() > 0);
    int n = (int) pts.size();
    vector<int> p(n, 0);
    for(int i = 0; i < n; i++) {
        p[i] = i;
    }
    shuffle(p.begin(), p.end(), rng);
    vector<vp> ans(n);
    ans[0].emplace_back(0, 0);
    ans[0].emplace_back(w, 0);
    ans[0].emplace_back(w, h);
    ans[0].emplace_back(0, h);
    for(int i = 1; i < n; i++) {
        ans[i] = ans[0];
    }
    for(auto i : p) {
        for(auto j : p) {
            if(j == i) break;
            auto bi = getBisector(pts[j], pts[i]);
            if(!polygonIntersection(bi, ans[j])) continue;
            ans[j] = cutPolygon(ans[j], getBisector(pts[j], pts[i]));
```

```
            ans[i] = cutPolygon(ans[i], getBisector(
    pts[i], pts[j]));
        }
    }
    return ans;
}
```

6.9 Intersect Polygon

```
bool intersect(vector<point> A, vector<point> B) //
    Ordered ccw
{
    for(auto a: A)
        if(inside(B, a))
            return true;
    for(auto b: B)
        if(inside(A, b))
            return true;

    if(inside(B, center(A)))
        return true;

    return false;
}
```

6.10 Sort By Angle

```
// Comparator funcion for sorting points by angle

int ret[2][2] = {{3, 2},{4, 1}};
inline int quad(point p) {
    return ret[p.x >= 0][p.y >= 0];
}

bool comp(point a, point b) { // ccw
    int qa = quad(a), qb = quad(b);
    return (qa == qb ? (a ^ b) > 0 : qa < qb);
}

// only vectors in range [x+0, x+180)
bool comp(point a, point b){
    return (a ^ b) > 0; // ccw
    // return (a ^ b) < 0; // cw
}
```

6.11 2d

```
#define vp vector<point>
#define ld long double
const ld EPS = 1e-6;
const ld PI = acos(-1);

typedef ld T;
bool eq(T a, T b){ return abs(a - b) <= EPS; }

struct point{
    T x, y;
    int id;
    point(T x=0, T y=0): x(x), y(y){}

    point operator+(const point &o) const{ return {x
    + o.x, y + o.y}; }
    point operator-(const point &o) const{ return {x
    - o.x, y - o.y}; }
    point operator*(T t) const{ return {x * t, y * t
    }; }
    point operator/(T t) const{ return {x / t, y / t
    }; }
    T operator*(const point &o) const{ return x * o.x
     + y * o.y; }
    T operator^(const point &o) const{ return x * o.y
     - y * o.x; }
    bool operator<(const point &o) const{
        return (eq(x, o.x) ? y < o.y : x < o.x);
    }
    bool operator==(const point &o) const{
        return eq(x, o.x) and eq(y, o.y);
    }
    friend ostream& operator<<(ostream& os, point p)
    {
        return os << "(" << p.x << "," << p.y << ")";
     }
};

int ccw(point a, point b, point e){ // -1=dir; 0=
    collinear; 1=esq;
    T tmp = (b-a) ^ (e-a); // vector from a to b
    return (tmp > EPS) - (tmp < -EPS);
}

ld norm(point a){ // Modulo
    return sqrt(a * a);
}
T norm2(point a){
    return a * a;
}
bool nulo(point a){
    return (eq(a.x, 0) and eq(a.y, 0));
}
point rotccw(point p, ld a){
    // a = PI*a/180; // graus
    return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
    +p.x*sin(a)));
}
point rot90cw(point a) { return point(a.y, -a.x); };
point rot90ccw(point a) { return point(-a.y, a.x); };

ld proj(point a, point b){ // a sobre b
    return a*b/norm(b);
}
ld angle(point a, point b){ // em radianos
    ld ang = a*b / norm(a) / norm(b);
    return acos(max(min(ang, (ld)1), (ld)-1));
}
ld angle_vec(point v){
    // return 180/PI*atan2(v.x, v.y); // graus
    return atan2(v.x, v.y);
}
ld order_angle(point a, point b){ // from a to b ccw
    (a in front of b)
    ld aux = angle(a,b)*180/PI;
    return ((a^b)<=0 ? aux:360-aux);
}
bool angle_less(point a1, point b1, point a2, point
    b2){ // ang(a1,b1) <= ang(a2,b2)
    point p1((a1*b1), abs((a1^b1)));
    point p2((a2*b2), abs((a2^b2)));
    return (p1^p2) <= 0;
}

ld area(vp &p){ // (points sorted)
    ld ret = 0;
    for(int i=2;i<(int)p.size();i++)
        ret += (p[i]-p[0])^(p[i-1]-p[0]);
    return abs(ret/2);
}
ld areaT(point &a, point &b, point &c){
    return abs((b-a)^(c-a))/2.0;
}

point center(vp &A){
    point c = point();
    int len = A.size();
    for(int i=0;i<len;i++)
        c=c+A[i];
```

```cpp
    return c/len;
}

point forca_mod(point p, ld m){
    ld cm = norm(p);
    if(cm<EPS) return point();
    return point(p.x*m/cm,p.y*m/cm);
}

ld param(point a, point b, point v){
    // v = t*(b-a) + a // return t;
    // assert(line(a, b).inside_seg(v));
    return ((v-a) * (b-a)) / ((b-a) * (b-a));
}

bool simetric(vp &a){ //ordered
    int n = a.size();
    point c = center(a);
    if(n&1) return false;
    for(int i=0;i<n/2;i++)
        if(ccw(a[i], a[i+n/2], c) != 0)
            return false;
    return true;
}

point mirror(point m1, point m2, point p){
    // mirror point p around segment m1m2
    point seg = m2-m1;
    ld t0 = ((p-m1)*seg) / (seg*seg);
    point ort = m1 + seg*t0;
    point pm = ort-(p-ort);
    return pm;
}


///////////
//  Line  //
///////////

struct line{
    point p1, p2;
    T a, b, c; // ax+by+c = 0;
    // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
    line(point p1=0, point p2=0): p1(p1), p2(p2){
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = p1 ^ p2;
    }

    T eval(point p){
        return a*p.x+b*p.y+c;
    }
    bool inside(point p){
        return eq(eval(p), 0);
    }
    point normal(){
        return point(a, b);
    }

    bool inside_seg(point p){
        return (
            ((p1-p) ^ (p2-p)) == 0 and
            ((p1-p) * (p2-p)) <= 0
        );
    }

};

// be careful with precision error
vp inter_line(line l1, line l2){
    ld det = l1.a*l2.b - l1.b*l2.a;
    if(det==0) return {};
    ld x = (l1.b*l2.c - l1.c*l2.b)/det;
    ld y = (l1.c*l2.a - l1.a*l2.c)/det;
    return {point(x, y)};
}

// segments not collinear
vp inter_seg(line l1, line l2){
    vp ans = inter_line(l1, l2);
    if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.inside_seg(ans[0]))
        return {};
    return ans;
}
bool seg_has_inter(line l1, line l2){
    return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.p2, l2.p2) < 0 and
           ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.p2, l1.p2) < 0;
}

ld dist_seg(point p, point a, point b){ // point - seg
    if((p-a)*(b-a) < EPS) return norm(p-a);
    if((p-b)*(a-b) < EPS) return norm(p-b);
    return abs((p-a)^(b-a)) / norm(b-a);
}

ld dist_line(point p, line l){ // point - line
    return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
}

line bisector(point a, point b){
    point d = (b-a)*2;
    return line(d.x, d.y, a*a - b*b);
}

line perpendicular(line l, point p){ // passes through p
    return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
}


///////////
// Circle //
///////////

struct circle{
    point c; T r;
    circle() : c(0, 0), r(0){}
    circle(const point o) : c(o), r(0){}
    circle(const point a, const point b){
        c = (a+b)/2;
        r = norm(a-c);
    }
    circle(const point a, const point b, const point cc){
        assert(ccw(a, b, cc) != 0);
        c = inter_line(bisector(a, b), bisector(b, cc))[0];
        r = norm(a-c);
    }
    bool inside(const point &a) const{
        return norm(a - c) <= r + EPS;
    }
};

pair<point, point> tangent_points(circle cr, point p){
    ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
    point p1 = rotccw(cr.c-p, -theta);
    point p2 = rotccw(cr.c-p, theta);
    assert(d1 >= cr.r);
    p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
    p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
```

```
    return {p1, p2};
}


circle incircle(point p1, point p2, point p3){
    ld m1 = norm(p2-p3);
    ld m2 = norm(p1-p3);
    ld m3 = norm(p1-p2);
    point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
    ld s = 0.5*(m1+m2+m3);
    ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
    return circle(c, r);
}

circle circumcircle(point a, point b, point c) {
    circle ans;
    point u = point((b-a).y, -(b-a).x);
    point v = point((c-a).y, -(c-a).x);
    point n = (c-b)*0.5;
    ld t = (u^n)/(v^u);
    ans.c = ((a+c)*0.5) + (v*t);
    ans.r = norm(ans.c-a);
    return ans;
}

vp inter_circle_line(circle C, line L){
    point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.p1)*(ab) / (ab*ab));
    ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s / (ab*ab);
    if (h2 < -EPS) return {};
    if (eq(h2, 0)) return {p};
    point h = (ab/norm(ab)) * sqrt(h2);
    return {p - h, p + h};
}

vp inter_circle(circle c1, circle c2){
    if (c1.c == c2.c) { assert(c1.r != c2.r); return {}; }
    point vec = c2.c - c1.c;
    ld d2 = vec * vec, sum = c1.r + c2.r, dif = c1.r - c2.r;
    ld p = (d2 + c1.r * c1.r - c2.r * c2.r) / (2 * d2);
    ld h2 = c1.r * c1.r - p * p * d2;
    if (sum * sum < d2 or dif * dif > d2) return {};
    point mid = c1.c + vec * p, per = point(-vec.y, vec.x) * sqrt(fmax(0, h2) / d2);
    if (eq(per.x, 0) and eq(per.y, 0)) return {mid};
    return {mid + per, mid - per};
}

// minimum circle cover O(n) amortizado
circle min_circle_cover(vp v){
    random_shuffle(v.begin(), v.end());
    circle ans;
    int n = v.size();
    for(int i=0;i<n;i++) if(!ans.inside(v[i])){
        ans = circle(v[i]);
        for(int j=0;j<i;j++) if(!ans.inside(v[j])){
            ans = circle(v[i], v[j]);
            for(int k=0;k<j;k++) if(!ans.inside(v[k])){
                ans = circle(v[i], v[j], v[k]);
            }
        }
    }
    return ans;
}
```

7 Grafos

7.1 Dfs Tree

```
int desce[N], sobe[N], vis[N], h[N];
int backedges[N], pai[N];

// backedges[u] = backedges que comecam embaixo de (ou =) u e sobem pra cima de u; backedges[u] == 0 => u eh ponte
void dfs(int u, int p) {
    if(vis[u]) return;
    pai[u] = p;
    h[u] = h[p]+1;
    vis[u] = 1;

    for(auto v : g[u]) {
        if(p == v or vis[v]) continue;
        dfs(v, u);
        backedges[u] += backedges[v];
    }
    for(auto v : g[u]) {
        if(h[v] > h[u]+1)
            desce[u]++;
        else if(h[v] < h[u]-1)
            sobe[u]++;
    }
    backedges[u] += sobe[u] - desce[u];
}
```

7.2 Kosaraju

```
vector<int> g[N], gi[N]; // grafo invertido
int vis[N], comp[N]; // componente conexo de cada vertice
stack<int> S;

void dfs(int u){
    vis[u] = 1;
    for(auto v: g[u]) if(!vis[v]) dfs(v);
    S.push(u);
}

void scc(int u, int c){
    vis[u] = 1; comp[u] = c;
    for(auto v: gi[u]) if(!vis[v]) scc(v, c);
}

void kosaraju(int n){
    for(int i=0;i<n;i++) vis[i] = 0;
    for(int i=0;i<n;i++) if(!vis[i]) dfs(i);
    for(int i=0;i<n;i++) vis[i] = 0;
    while(S.size()){
        int u = S.top();
        S.pop();
        if(!vis[u]) scc(u, u);
    }
}
```

7.3 Topological Sort

```
int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
```

```
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i]) {
            dfs(i);
        }
    }
    reverse(ans.begin(), ans.end());
}
```

7.4 Dijkstra

```
#define pii pair<int, int>
vector<vector<pii>> g(N);
vector<bool> used(N);
vector<ll> d(N, LLINF);
priority_queue< pii, vector<pii>, greater<pii> > fila;

void dijkstra(int k) {
    d[k] = 0;
    fila.push({0, k});

    while (!fila.empty()) {
        auto [w, u] = fila.top();
        fila.pop();
        if (used[u]) continue;
        used[u] = true;

        for (auto [v, w]: g[u]) {
            if (d[v] > d[u] + w) {
                d[v] = d[u] + w;
                fila.push({d[v], v});
            }
        }
    }
}
```

7.5 Dinic

```
// Description: Flow algorithm with complexity O(VE log U) where U = max |cap|.
// O(min(E^{1/2}, V^{2/3})E) if U = 1; O(sqrt(V)E)$ for bipartite matching.
// testado em https://www.spoj.com/problems/FASTFLOW/ 0.20s
const int N = 200003;
template<typename T> struct Dinic {
    struct Edge {
        int from, to;
        T c, f;
        Edge(int from, int to, T c, T f): from(from), to(to), c(c), f(f) {}
    };

    vector<Edge> edges;
    int tempo = 0, id = 0;
    int lvl[N], vis[N], qu[N], nxt[N];
    vector<int> adj[N];
    T INF = (ll)1e14;
    #warning botar INF certo no dinic

    void addEdge(int a, int b, T c, T rc=0) {
        edges.pb({a, b, c, 0});
        adj[a].pb(id++);
        edges.pb({b, a, rc, 0});
        adj[b].pb(id++);
    }

    bool bfs(int s, int t) {
        tempo++;
        vis[s] = tempo;
        int qt = 0;
        qu[qt++] = s;
        lvl[s] = 0;

        for(int i = 0; i < qt; i++) {
            int u = qu[i];
            nxt[u] = 0;

            for(auto idx : adj[u]) {
                auto& e = edges[idx];
                if(e.f >= e.c or vis[e.to] == tempo) continue;
                // from[e.to] = idx; pra usar a outra dfs
                vis[e.to] = tempo;
                lvl[e.to] = lvl[u]+1;
                qu[qt++] = e.to;
            }
        }
        return (vis[t] == tempo);
    }

    T dfs(int s, int t, T f) {
        if(s == t) return f;

        T res = 0;
        for(; nxt[s] < (int)adj[s].size(); nxt[s]++) {
            int idx = adj[s][nxt[s]];
            auto& e = edges[idx];
            auto& rev = edges[idx^1];

            if(e.f >= e.c or lvl[e.to] != lvl[s]+1) continue;
            T flow = dfs(e.to, t, min(f, e.c-e.f));
            res += flow;
            e.f += flow;
            rev.f -= flow;
            f -= flow;

            if(!f) break;
        }
        return res;
    }

//   dfs boa para grafos pequenos (n <= 500?), ruim para fluxos grandes?
//   tem que criar o vetor from pra usar e marcar o from na bfs
//   T dfs(int s, int t) {
//        T res = INF;

//        for(int u = t; u != s; u = edges[from[u]].from) {
//            res = min(res, edges[from[u]].c-edges[from[u]].f);
//        }

//        for(int u = t; u != s; u = edges[from[u]].from) {
//            edges[from[u]].f += res;
//            edges[from[u]^1].f -= res;
//        }

//        return res;
//    }

    T flow(int s, int t) {
```

```cpp
        T flow = 0;
        while(bfs(s, t)) {
            flow += dfs(s, t, INF);
        }
        return flow;
    }

    // NAO TESTADO DAQUI PRA BAIXO, MAS DEVE FUNCIONAR
    void reset_flow() {
        for(int i = 0; i < id; i++) // aqui eh id mesmo ne?
            edges[i].flow = 0;
        memset(lvl, 0, sizeof(lvl));
        memset(vis, 0, sizeof(vis));
        memset(qu, 0, sizeof(qu));
        memset(nxt, 0, sizeof(nxt));
        tempo = 0;
    }
    vector<pair<int, int>> cut() {
        vector<pair<int, int>> cuts;
        for (auto [from, to, flow, cap]: edges) {
            if (flow == cap and vis[from] == tempo and vis[to] < tempo and cap>0) {
                cuts.pb({from, to});
            }
        }
        return cuts;
    }
};
```

7.6 Centroid Decomp

```cpp
vector<int> g[N];
int sz[N], rem[N];

void dfs(vector<int>& path, int u, int d=0, int p=-1) {
    path.push_back(d);
    for (int v : g[u]) if (v != p and !rem[v]) dfs(path, v, d+1, u);
}

int dfs_sz(int u, int p=-1) {
    sz[u] = 1;
    for (int v : g[u]) if (v != p and !rem[v]) sz[u] += dfs_sz(v, u);
    return sz[u];
}

int centroid(int u, int p, int size) {
    for (int v : g[u]) if (v != p and !rem[v] and 2 * sz[v] > size)
        return centroid(v, u, size);
    return u;
}

ll decomp(int u, int k) {
    int c = centroid(u, u, dfs_sz(u));
    rem[c] = true;

    ll ans = 0;
    vector<int> cnt(sz[u]);
    cnt[0] = 1;
    for (int v : g[c]) if (!rem[v]) {
        vector<int> path;
        dfs(path, v);
        // d1 + d2 + 1 == k
        for (int d : path) if (0 <= k-d-1 and k-d-1 < sz[u])
            ans += cnt[k-d-1];
        for (int d : path) cnt[d+1]++;
    }

    for (int v : g[c]) if (!rem[v]) ans += decomp(v, k);
    return ans;
}
```

7.7 Hungarian

```cpp
// Hungaro
//
// Resolve o problema de assignment (matriz n x n)
// Colocar os valores da matriz em 'a' (pode < 0)
// assignment() retorna um par com o valor do
// assignment minimo, e a coluna escolhida por cada linha
//
// O(n^3)

template<typename T> struct hungarian {
    int n;
    vector<vector<T>> a;
    vector<T> u, v;
    vector<int> p, way;
    T inf;

    hungarian(int n_) : n(n_), u(n+1), v(n+1), p(n+1), way(n+1) {
        a = vector<vector<T>>(n, vector<T>(n));
        inf = numeric_limits<T>::max();
    }
    pair<T, vector<int>> assignment() {
        for (int i = 1; i <= n; i++) {
            p[0] = i;
            int j0 = 0;
            vector<T> minv(n+1, inf);
            vector<int> used(n+1, 0);
            do {
                used[j0] = true;
                int i0 = p[j0], j1 = -1;
                T delta = inf;
                for (int j = 1; j <= n; j++) if (!used[j]) {
                    T cur = a[i0-1][j-1] - u[i0] - v[j];
                    if (cur < minv[j]) minv[j] = cur, way[j] = j0;
                    if (minv[j] < delta) delta = minv[j], j1 = j;
                }
                for (int j = 0; j <= n; j++)
                    if (used[j]) u[p[j]] += delta, v[j] -= delta;
                    else minv[j] -= delta;
                j0 = j1;
            } while (p[j0] != 0);
            do {
                int j1 = way[j0];
                p[j0] = p[j1];
                j0 = j1;
            } while (j0);
        }
        vector<int> ans(n);
        for (int j = 1; j <= n; j++) ans[p[j]-1] = j-1;
        return make_pair(-v[0], ans);
    }
};
```

7.8 Floyd Warshall

```cpp
// Floyd Warshall

```

```cpp
int dist[N][N];

for(int k = 1; k <= n; k++)
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
```

7.9 2sat

```cpp
#define rep(i,l,r) for (int i = (l); i < (r); i++)
struct TwoSat { // copied from kth-competitive-programming/kactl
    int N;
    vector<vi> gr;
    vi values; // 0 = false, 1 = true
    TwoSat(int n = 0) : N(n), gr(2*n) {}
    int addVar() { // (optional)
        gr.emplace_back();
        gr.emplace_back();
        return N++;
    }
    void either(int f, int j) {
        f = max(2*f, -1-2*f);
        j = max(2*j, -1-2*j);
        gr[f].push_back(j^1);
        gr[j].push_back(f^1);
    }
    void atMostOne(const vi& li) { // (optional)
        if ((int)li.size() <= 1) return;
        int cur = ~li[0];
        rep(i,2,(int)li.size()) {
            int next = addVar();
            either(cur, ~li[i]);
            either(cur, next);
            either(~li[i], next);
            cur = ~next;
        }
        either(cur, ~li[1]);
    }
    vi _val, comp, z; int time = 0;
    int dfs(int i) {
        int low = _val[i] = ++time, x; z.push_back(i);
        for(int e : gr[i]) if (!comp[e])
            low = min(low, _val[e] ?: dfs(e));
        if (low == _val[i]) do {
            x = z.back(); z.pop_back();
            comp[x] = low;
            if (values[x>>1] == -1)
                values[x>>1] = x&1;
        } while (x != i);
        return _val[i] = low;
    }
    bool solve() {
        values.assign(N, -1);
        _val.assign(2*N, 0); comp = _val;
        rep(i,0,2*N) if (!comp[i]) dfs(i);
        rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
        return 1;
    }
};
```

7.10 Lca

```cpp
const int LOG = 22;
vector<vector<int>> g(N);
int t, n;
vector<int> in(N), height(N);
vector<vector<int>> up(LOG, vector<int>(N));
void dfs(int u, int h=0, int p=-1) {
    up[0][u] = p;
    in[u] = t++;
    height[u] = h;
    for (auto v: g[u]) if (v != p) dfs(v, h+1, u);
}

void blift() {
    up[0][0] = 0;
    for (int j=1;j<LOG;j++) {
        for (int i=0;i<n;i++) {
            up[j][i] = up[j-1][up[j-1][i]];
        }
    }
}

int lca(int u, int v) {
    if (u == v) return u;
    if (in[u] < in[v]) swap(u, v);
    for (int i=LOG-1;i>=0;i--) {
        int u2 = up[i][u];
        if (in[u2] > in[v])
            u = u2;
    }
    return up[0][u];
}

t = 0;
dfs(0);
blift();

// lca O(1)

template<typename T> struct rmq {
    vector<T> v;
    int n; static const int b = 30;
    vector<int> mask, t;

    int op(int x, int y) { return v[x] < v[y] ? x : y; }
    int msb(int x) { return __builtin_clz(1)-__builtin_clz(x); }
    rmq() {}
    rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(n) {
        for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
            at = (at<<1)&((1<<b)-1);
            while (at and op(i, i-msb(at&-at)) == i) at ^= at&-at;
        }
        for (int i = 0; i < n/b; i++) t[i] = b*i+b-1-msb(mask[b*i+b-1]);
        for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0; i+(1<<j) <= n/b; i++)
            t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j-1)+i+(1<<(j-1))]);
    }
    int small(int r, int sz = b) { return r-msb(mask[r]&((1<<sz)-1)); }
    T query(int l, int r) {
        if (r-l+1 <= b) return small(r, r-l+1);
        int ans = op(small(l+b-1), small(r));
        int x = l/b+1, y = r/b-1;
        if (x <= y) {
            int j = msb(y-x+1);
            ans = op(ans, op(t[n/b*j+x], t[n/b*j+y-(1<<j)+1]));
        }
        return ans;
    }
};

namespace lca {
```

```cpp
    vector<int> g[N];
    int v[2*N], pos[N], dep[2*N];
    int t;
    rmq<int> RMQ;

    void dfs(int i, int d = 0, int p = -1) {
        v[t] = i, pos[i] = t, dep[t++] = d;
        for (int j : g[i]) if (j != p) {
            dfs(j, d+1, i);
            v[t] = i, dep[t++] = d;
        }
    }
    void build(int n, int root) {
        t = 0;
        dfs(root);
        RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
    }
    int lca(int a, int b) {
        a = pos[a], b = pos[b];
        return v[RMQ.query(min(a, b), max(a, b))];
    }
    int dist(int a, int b) {
        return dep[pos[a]] + dep[pos[b]] - 2*dep[pos[lca(a, b)]];
    }
}
```

7.11 Kruskal

```cpp
struct DSU {
    int n;
    vector<int> parent, size;

    DSU(int n): n(n) {
        parent.resize(n, 0);
        size.assign(n, 1);

        for(int i=0;i<n;i++)
            parent[i] = i;
    }

    int find(int a) {
        if(a == parent[a]) return a;
        return parent[a] = find(parent[a]);
    }

    void join(int a, int b) {
        a = find(a); b = find(b);
        if(a != b) {
            if(size[a] < size[b]) swap(a, b);
            parent[b] = a;
            size[a] += size[b];
        }
    }
};

struct Edge {
    int u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

vector<Edge> kruskal(int n, vector<Edge> edges) {
    vector<Edge> mst;
    DSU dsu = DSU(n+1);

    sort(edges.begin(), edges.end());

    for(Edge e : edges) {
        if(dsu.find(e.u) != dsu.find(e.v)) {
            mst.push_back(e);
            dsu.join(e.u,e.v);
        }
    }

    return mst;
}
```

7.12 Mcmf

```cpp
// Time: O(F E log(V)) where F is max flow. (reference needed)
const int N = 502;
template<typename T> struct MCMF {
    struct Edge {
        int from, to;
        T c, f, cost;
        Edge(int from, int to, T c, T cost): from(from), to(to), c(c), cost(cost) {}
    };

    vector<Edge> edges;
    int tempo = 0, id = 0;
    int nxt[N], vis[N];
    vector<int> adj[N];
    T lvl[N];
    const T INF = 1e15;

    void addEdge(int a, int b, int c, int cost) {
        edges.pb({a, b, c, cost});
        adj[a].pb(id++);
        edges.pb({b, a, 0, -cost});
        adj[b].pb(id++);
    }

    pair<T,T> dfs(int s, int t, T f) {
        if(s == t or f == 0) return {f, 0};

        pair<T, T> res = {0, 0};
        for(; nxt[s] < (int)adj[s].size(); nxt[s]++) {
            int idx = adj[s][nxt[s]];
            auto& e = edges[idx];
            auto& rev = edges[idx^1];

            if(e.f >= e.c or lvl[e.to] != lvl[s]+e.cost) continue;
            auto [flow, cost] = dfs(e.to, t, min(f, e.c-e.f));

            if(!flow) continue;

            res.ff += flow;
            res.ss += cost + flow*e.cost;
            e.f += flow;
            rev.f -= flow;
            f -= flow;

            if(!f) break;
        }
        return res;
    }

    // funciona v
    // pair<T,T> dfs(int s, int t) {
    //     pair<T,T> res = {INF, 0};

    //     for(int u = t; u != s; u = edges[from[u]].from) {
    //         res.ff = min(res.ff, edges[from[u]].c);
    //     }

    //     for(int u = t; u != s; u = edges[from[u]].from) {
```

```
    //          edges[from[u]].c -= res.ff;
    //          edges[from[u]^1].c += res.ff;
    //          res.ss += edges[from[u]].cost * res.ff;
    //      }

    //      return res;
    // }

    bool spfa(int s, int t) {
        for(int i = 0; i < N; i++) {
            lvl[i] = INF;
            vis[i] = 0;
        }
        lvl[s] = 0;
        vis[s] = 1;
        queue<int> q; q.push(s);

        while(q.size()) {
            int u = q.front(); q.pop();
            vis[u] = 0;
            nxt[u] = 0;

            for(auto idx : adj[u]) {
                auto& e = edges[idx];

                if(e.f >= e.c) continue;
                if(lvl[e.to] > lvl[u]+e.cost) {
                    lvl[e.to] = lvl[u]+e.cost;
                    if(!vis[e.to]) {
                        q.push(e.to);
                        vis[e.to] = 1;
                    }
                }

            }
        }
        return (lvl[t] < INF);
    }

    pair<T,T> flow(int s, int t) {
        pair<T,T> res = {0, 0};
        while(spfa(s, t)) {
            auto [flow, cost] = dfs(s, t, INF);
            res.ff += flow;
            res.ss += cost;
        }
        return res;
    }
};
```

7.13 Ford

```
const int N = 2000010;

struct Ford {
    struct Edge {
        int to, f, c;
    };

    int vis[N];
    vector<int> adj[N];
    vector<Edge> edges;
    int cur = 0;

    void addEdge(int a, int b, int cap, int rcap) {
        Edge e;
        e.to = b; e.c = cap; e.f = 0;
        edges.pb(e);
        adj[a].pb(cur++);

        e = Edge();
        e.to = a; e.c = rcap; e.f = 0;
        edges.pb(e);
        adj[b].pb(cur++);
    }

    int dfs(int s, int t, int f, int tempo) {
        if(s == t)
            return f;
        vis[s] = tempo;

        for(int e : adj[s]) {
            if(vis[edges[e].to] < tempo and (edges[e].c - edges[e].f) > 0) {
                if(int a = dfs(edges[e].to, t, min(f, edges[e].c-edges[e].f) , tempo)) {
                    edges[e].f += a;
                    edges[e^1].f -= a;
                    return a;
                }
            }
        }
        return 0;
    }

    int flow(int s, int t) {
        int mflow = 0, tempo = 1;
        while(int a = dfs(s, t, INF, tempo)) {
            mflow += a;
            tempo++;
        }
        return mflow;
    }
};
```

8 Algoritmos

8.1 Ternary Search

```
// Ternary
ld l = -1e4, r = 1e4;
int iter = 100;
while(iter--){
    ld m1 = (2*l + r) / 3;
    ld m2 = (l + 2*r) / 3;
    if(check(m1) > check(m2))
        l = m1;
    else
        r = m2;
}
```

9 Math

9.1 Discrete Log

```
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m, int k = 1) {
    a %= m, b %= m;
    int add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;
```

```
    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

9.2 Totient

```
// phi(p^k) = (p^(k-1))*(p-1) com p primo
// O(sqrt(m))
ll phi(ll m){
    ll res = m;
    for(ll d=2;d*d<=m;d++){
        if(m % d == 0){
            res = (res/d)*(d-1);
            while(m%d == 0)
                m /= d;
        }
    }
    if(m > 1) {
        res /= m;
        res *= (m-1);
    }
    return res;
}

// modificacao do crivo, O(n*log(log(n)))
vector<ll> phi_to_n(ll n){
    vector<bool> isprime(n+1, true);
    vector<ll> tot(n+1);
    tot[0] = 0; tot[1] = 1;
    for(ll i=1;i<=n; i++){
        tot[i] = i;
    }

    for(ll p=2;p<=n;p++){
        if(isprime[p]){
            tot[p] = p-1;
            for(ll i=p+p;i<=n;i+=p){
                isprime[i] = false;
                tot[i] = (tot[i]/p)*(p-1);
            }
        }
    }
    return tot;
}
```

9.3 Pollard Rho

```
ll mul(ll a, ll b, ll m) {
    ll ret = a*b - (ll)((ld)1/m*a*b+0.5)*m;
    return ret < 0 ? ret+m : ret;
}

ll pow(ll a, ll b, ll m) {
    ll ans = 1;
    for (; b > 0; b /= 2ll, a = mul(a, a, m)) {
        if (b % 2ll == 1)
            ans = mul(ans, a, m);
    }
    return ans;
}

bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;

    ll r = __builtin_ctzll(n - 1), d = n >> r;
    for (int a : {2, 325, 9375, 28178, 450775, 9780504, 795265022}) {
        ll x = pow(a, d, n);
        if (x == 1 or x == n - 1 or a % n == 0) continue;

        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) break;
        }
        if (x != n - 1) return 0;
    }
    return 1;
}

ll rho(ll n) {
    if (n == 1 or prime(n)) return n;
    auto f = [n](ll x) {return mul(x, x, n) + 1;};

    ll x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
    while (t % 40 != 0 or gcd(prd, n) == 1) {
        if (x==y) x = ++x0, y = f(x);
        q = mul(prd, abs(x-y), n);
        if (q != 0) prd = q;
        x = f(x), y = f(f(y)), t++;
    }
    return gcd(prd, n);
}

vector<ll> fact(ll n) { // retorna fatoracao em primos
    if (n == 1) return {};
    if (prime(n)) return {n};
    ll d = rho(n);
    vector<ll> l = fact(d), r = fact(n / d);
    l.insert(l.end(), r.begin(), r.end());
    return l;
}
```

9.4 Inverso Mult

```
// gcd(a, m) = 1 para existir solucao
// ax + my = 1, ou a*x = 1 (mod m)
ll inv(ll a, ll m) { // com gcd
    ll x, y;
    gcd(a, m, x, y);
    return (((x % m) +m) %m);
}

ll inv(ll a, ll phim) { // com phi(m), se m for primo entao phi(m) = p-1
    ll e = phim-1;
    return fexp(a, e);
}
```

9.5 Miller Habin

```
ll mul(ll a, ll b, ll m) {
    return (a*b-ll(a*(long double)b/m+0.5)*m+m)%m;
}

ll expo(ll a, ll b, ll m) {
    if (!b) return 1;
    ll ans = expo(mul(a, a, m), b/2, m);
```

```
    return b%2 ? mul(a, ans, m) : ans;
}

bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;

    ll d = n - 1;
    int r = 0;
    while (d % 2 == 0) {
        r++;
        d /= 2;
    }

    // com esses primos, o teste funciona garantido para n <= 2^64
    // funciona para n <= 3*10^24 com os primos ate 41
    for (int i : {2, 325, 9375, 28178, 450775, 9780504, 795265022}) {
        if (i >= n) break;
        ll x = expo(i, d, n);
        if (x == 1 or x == n - 1) continue;

        bool composto = 1;
        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) {
                composto = 0;
                break;
            }
        }
        if (composto) return 0;
    }
    return 1;
}
```

9.6 Matrix

```
struct Matrix {
    vector<vl> m;
    int r, c;

    Matrix(vector<vl> mat) {
        m = mat;
        r = mat.size();
        c = mat[0].size();
    }

    Matrix(int row, int col, bool ident=false) {
        r = row; c = col;
        m = vector<vl>(r, vl(c, 0));
        if(ident) {
            assert(r == c);
            for(int i = 0; i < min(r, c); i++) {
                m[i][i] = 1;
            }
        }
    }

    Matrix operator+(const Matrix &o) const {
        assert(r == o.r and c == o.c);

        vector<vl> res(r, vl(c, 0));

        for(int i = 0; i < r; i++) {
            for(int j = 0; j < c; j++) {
                res[i][j] = (m[i][j] + o.m[i][j]) % MOD;
            }
        }

        return Matrix(res);
    }

    Matrix operator-(const Matrix &o) const {
        assert(r == o.r and c == o.c);

        vector<vl> res(r, vl(c, 0));

        for(int i = 0; i < r; i++) {
            for(int j = 0; j < c; j++) {
                res[i][j] = (m[i][j] - o.m[i][j] + MOD) % MOD;
            }
        }

        return Matrix(res);
    }

    Matrix operator*(const int k) const {
        vector<vl> res(r, vl(c, 0));

        for(int i = 0; i < r; i++) {
            for(int j = 0; j < c; j++) {
                res[i][j] = (k * m[i][j]) % MOD;
            }
        }

        return Matrix(res);
    }

    Matrix operator*(const Matrix &o) const {
        assert(c == o.r); // garantir que da pra multiplicar
        vector<vl> res(r, vl(o.c, 0));

        for(int i = 0; i < r; i++) {
            for(int k = 0; k < c; k++) {
                for(int j = 0; j < o.c; j++) {
                    res[i][j] = (res[i][j] + m[i][k]*o.m[k][j]) % MOD;
                }
            }
        }

        return Matrix(res);
    }

    Matrix transpose() const {
        vector<vl> res(c, vl(r,0));

        for(int i = 0; i < r; i++) {
            for(int j = 0; j < c; j++) {
                res[j][i] = m[i][j];
            }
        }

        return Matrix(res);
    }
};

Matrix fexp(Matrix b, int e, int n) {
    if(e == 0) return Matrix(n, n, true); // identidade
    Matrix res = fexp(b, e/2, n);
    res = (res * res);
    if(e%2) res = (res * b);

    return res;
}
```

9.7 Division Trick

```
for(int l = 1, r; l <= n; l = r + 1) {
```

```
    r = n / (n / l);
    // n / i has the same value for l <= i <= r
}
```

9.8 Crivo

```
vi p(N, 0);
p[0] = p[1] = 1;
for(ll i=4; i<N; i+=2) p[i] = 2;
for(ll i=3; i<N; i+=2)
    if(!p[i])
        for(ll j=i*i; j<N; j+=2*i)
            p[j] = i;
```

9.9 Bigmod

```
ll mod(string a, ll p) {
    ll res = 0, b = 1;
    reverse(all(a));

    for(auto c : a) {
        ll tmp = (((ll)c-'0')*b) % p;
        res = (res + tmp) % p;

        b = (b * 10) % p;
    }

    return res;
}
```

9.10 Linear Diophantine Equation

```
// Linear Diophantine Equation
array<ll, 3> exgcd(int a, int b) {
    if (a == 0) return {0, 1, b};
    auto [x, y, g] = exgcd(b % a, a);
    return {y - b / a * x , x, g};
}

array<ll, 4> find_any_solution(ll a, ll b, ll c) {
    auto[x, y, g] = exgcd(a, b);
    if (c % g) return {false, 0, 0, 0};
    x *= c / g;
    y *= c / g;
    return {true, x, y, g};
}

// All solutions
// x' = x + k*b/g
// y' = y - k*a/g
```

9.11 Primitiveroot

```
long long fexp(long long x, long long e, long long
    mod = MOD) {
    long long ans = 1;
    x %= mod;
    for(; e > 0; e /= 2, x = x * x % mod) {
        if(e & 1) ans = ans * x % mod;
    }
    return ans;
}
//is n primitive root of p ?
bool test(long long x, long long p) {
    long long m = p - 1;
    for(int i = 2; i * i <= m; ++i) if(!(m % i)) {
        if(fexp(x, i, p) == 1) return false;
        if(fexp(x, m / i, p) == 1) return false;
    }
    return true;
}
//find the smallest primitive root for p
int search(int p) {
    for(int i = 2; i < p; i++) if(test(i, p)) return
    i;
    return -1;
}
```

10 Teoria

10.1 Geometria

10.1.1 Geometria Básica

Produto Escalar. Geometricamente é o produto do comprimento do vetor a pelo comprimento da projeção do vetor b sobre a.

$$a \cdot b = \|a\|\|b\| \cos\theta.$$

Propriedades.

1. $a \cdot b = b \cdot a$.
2. $(\alpha \cdot a) \cdot b = \alpha \cdot (a \cdot b)$.
3. $(a + b) \cdot c = a \cdot c + b \cdot c$.
4. Norma de a (comprimento ao quadrado): $\|a\|^2 = a \cdot a$.
5. Projeção de a sobre o vetor b: $\frac{a \cdot b}{\|b\|}$.
6. Ângulo entre os vetores: $\cos^{-1} \frac{a \cdot b}{\|a\|\|b\|}$

Produto Vetorial. Dados dois vetores independentes linearmente a e b, o produto vetorial $a \times b$ é um vetor perpendicular ao vetor a e ao vetor b e é a normal do plano contendo os dois vetores.

$$a \times b = det \begin{vmatrix} e_x & e_y & e_z \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}$$

O sinal do coeficiente e_z do produto vetorial indica a orientação relativa dos vetores. Se positivo, o ângulo de a e b é anti-horário. Se negativo, o ângulo é horário e se for zero, os vetores são colineares.

Propriedades.

1. $a \times b = -b \times a$.
2. $(\alpha \cdot a) \times b = \alpha \cdot (a \times b)$.
3. $a \cdot (b \times c) = b \cdot (c \times a) = -a \cdot (c \times b)$.
4. $(a + b) \times c = a \times c + b \times c$.
5. $\|a \times b\| = \|a\|\|b\| \sin \theta$.

10.1.2 Geometria Analítica

Distância entre dois pontos. Dados dois pontos $a = (x_1, y_2)$ e $b = (x_2, y_2)$, a distância entre a e b é dada por:

$$d_{a,b} = \sqrt{(x_1 - x_2)^2 + (y_1 - y2)^2}$$

Condição de alinhamento de três pontos. Dados três pontos $a = (x_1, y_2)$, $b = (x_2, y_2)$ e $c = (x_3, y_3)$, os pontos a, b e c estão alinhados se:

$$det(A) = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0$$

Equação da Reta (forma geral). Os pontos (x, y) que pertencem a uma reta r devem satisfazer:

$$ax + by + c = 0$$

Equação da Reta (forma reduzida). A equação reduzida da reta, em que $m = \tan(a) = \frac{\Delta y}{\Delta x}$ é o coef. angular, e n é o coef. linear, isto é, o valor de y em que a reta intercepta o eixo y, é dada por:

$$y = mx + n = m(x - x_0) + y_0$$

Distância entre ponto e reta. Dados um pontos $p = (x_1, y_1)$ e uma reta r de equação $ax + by + c = 0$, a distância entre p e r é dada por:

$$d_{p,r} = \frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}}$$

Interseção de retas. Para determinar os pontos de interseção é necessário resolver um sistema de equações. Há três possibilidades para interseção de retas:

1. Retas concorrentes: solução única. Apenas 1 ponto em comum.
2. Retas paralelas coincidentes: infinitas soluções. As retas possuem todos os pontos em comum.
3. Retas paralelas distintas: nenhuma solução. As retas não possuem nenhum ponto em comum.

Equação da Circuferência (forma reduzida). Os pontos (x, y) que pertencem a uma circuferência c devem satisfazer:

$$(x - a)^2 + (y - b)^2 = r^2,$$

onde (a, b) é o centro da circuferência e r o seu raio.

Equação da Circuferência (forma geral). A partir da equação reduzida da circuferência, encontramos a equação geral:

$$x^2 + y^2 - 2ax - 2by + (a^2 + b^2 - r^2) = 0$$

Interseção entre reta e circuferência. Para determinar o tipo de interseção é necessário resolver um sistema não-linear. Há três possibilidades como solução do sistema:

1. Reta exterior à circuferência: nenhuma solução. A reta não possui nenhum ponto de comum com a circuferência.
2. Reta tangente à circuferência: solução única. A reta possui apenas 1 ponto em comum com a circuferência.
3. Reta secante à circuferência: duas soluções. A reta cruza a circuferência em 2 pontos distintos.

10.1.3 Geometria Plana

Triângulos. Polígono com três vértices e três arestas. Uma aresta arbitrária é escolhida como a base e, nesse caso, o vértice oposto é chamado de ápice. Um triângulo com vértices A, B e C é denotado $\triangle ABC$.

- Comprimento dos lados: a, b, c
- Semiperímetro: $p = \frac{a+b+c}{2}$
- Altura:
  - Equilátero: $h = \frac{\sqrt{3}}{2} l$
  - Isósceles: $h = \sqrt{l^2 - \frac{b^2}{4}}$
- Área:
  - Equilátero: $A = \frac{l^2\sqrt{3}}{4}$
  - Isósceles: $A = \frac{1}{2} bh$
  - Escaleno: $A = \sqrt{p(p-a)(p-b)(p-c)}$
- Raio circunscrito: $R = \frac{1}{4A} abc$
- Raio inscrito: $r = \frac{1}{p} A$
- Tamanho da mediana: $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Quadriláteros. Polígono de quatro lados, tendo quatro arestas e quatro vértices. Um quadrilátero com vértices A, B, C e D é denotado com $\square ABCD$.

- Comprimento dos lados: a, b, c, d
- Semiperímetro: $p = \frac{a+b+c+b}{2}$
- Área:
  - Quadrado: a^2
  - Retângulo: $b \cdot h$
  - Losango: $\frac{1}{2} D \cdot d$
  - Trapézio: $\frac{1}{2} h(B+b)$
- Perímetro:
  - Quadrado: $4a$
  - Retângulo: $2(b+h)$
  - Losango: $4a$
  - Trapézio: $B + b + l_1 + l_2$
- Diagonal:
  - Quadrado: $a\sqrt{2}$
  - Retângulo: $\sqrt{b^2 + h^2}$
  - Losango: $a\sqrt{2}$
  - Trapézio: $\sqrt{h^2 + \frac{(B-b)^2}{4h}}$

Círculos. Forma que consiste em todos os pontos de um plano que estão a uma determinada distância de um ponto dado, o centro. A distância entre qualquer ponto do círculo e o centro é chamada de raio.

- Área: $A = \pi r^2$
- Perímetro: $C = 2\pi r$
- Diâmetro: $d = 2r$
- Área do setor circular: $A = \frac{1}{2} r^2 \theta$
- Comprimento do arco: $L = r\theta$
- Perímetro do setor circular: $P = r(\theta + 2)$

Teorema de Pick. Suponha que um polígono tenha coordenadas inteiras para todos os seus vértices. Seja i o número de pontos inteiros no interior do polígono e b o número de pontos inteiros na sua fronteira (incluindo vértices e pontos ao longo dos lados). Então, a área A deste polígono é:

$$A = i + \frac{b}{2} - 1.$$

$$b = \gcd(|x_2 - x_1|, |y_2 - y_1|) + 1.$$

10.1.4 Geometria Espacial

- Área da Superfície:
  - Cubo: $6a^2$
  - Prisma: $A_l + 2A_b$
  - Esfera: $4\pi r^2$
  - Cilindro: $2\pi r(h + r)$
  - Cone: $\pi r(r + \sqrt{h^2 + r^2})$
  - Pirâmide: $A_b + \frac{1}{2}P_b \cdot g, \quad g = \text{geratriz}$
- Volume:
  - Cubo: a^3
  - Prisma: $A_b \cdot h$
  - Esfera: $\frac{4}{3}\pi r^3$
  - Cilindro: $\pi r^2 h$
  - Cone: $\frac{1}{3}\pi r^2 h$
  - Pirâmide: $\frac{1}{3}A_b \cdot h$

Fórmula de Euler para Poliedros. Os números de faces, vértices e arestas de um sólido não são independentes, mas estão relacionados de uma maneira simples.

$$F + V - A = 2.$$

10.1.5 Trigonometria

Funções Trigonométricas.

$$\sin\theta = \frac{\text{cateto oposto a }\theta}{\text{hipotenusa}} \quad \cos\theta = \frac{\text{cateto adjacente a }\theta}{\text{hipotenusa}} \quad \tan\theta = \frac{\text{cateto oposto a }\theta}{\text{cateto adjacente a}\theta}$$

Ângulos notáveis.

θ	$0°$	$30°$	$45°$	$60°$	$90°$
$\sin\theta$	0	$\frac{1}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$	1
$\cos\theta$	1	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{1}{2}$	0
$\tan\theta$	0	$\frac{\sqrt{3}}{3}$	1	$\sqrt{3}$	∞

Propriedades.

1. $\sin(a + b) = \sin a \cos b + \cos a \sin b$
2. $\cos(a + b) = \cos a \cos b - \sin a \sin b$
3. $\tan(a + b) = \frac{\tan a + \tan b}{1 - \tan a \tan b}$
4. $a \sin x + b \cos x = r \sin(x + \phi)$, onde $r = \sqrt{a^2 + b^2}$ e $\phi = \tan^{-1}\frac{b}{a}$
5. $a \cos x + b \sin x = r \cos(x - \phi)$, onde $r = \sqrt{a^2 + b^2}$ e $\phi = \tan^{-1}\frac{b}{a}$
6. **Lei dos Senos**:
$$\frac{a}{\sin\hat{A}} = \frac{b}{\sin\hat{B}} = \frac{c}{\sin\hat{C}} = 2r.$$
7. **Lei dos Cossenos**:
$$a^2 = b^2 + c^2 + 2 \cdot b \cdot c \cdot \cos\hat{A}$$
$$b^2 = a^2 + c^2 + 2 \cdot a \cdot c \cdot \cos\hat{B}$$
$$c^2 = b^2 + a^2 + 2 \cdot b \cdot a \cdot \cos\hat{C}$$
8. **Teorema de Tales**: A interseção de um feixe de retas paralelas por duas retas transversais forma segmentos proporcionais:
$$\frac{\overline{AB}}{\overline{BC}} = \frac{\overline{DE}}{\overline{EF}}$$
9. **Casos de semelhança**: dois triângulos são semelhantes se
   - dois ângulos de um são congruentes a dois do outro. Critério AA (Ângulo, Ângulo).
   - os três lados são proporcionais aos três lados do outro. Critério LLL (Lado, Lado, Lado).
   - possuem um ângulo congruente compreendido entre lados proporcionais. Critério LAL (Lado, Ângulo, Lado).

10.2 Teoria dos Grafos

10.2.1 Caminhos

Caminho de Euler

Um caminho de Euler em um grafo é o caminho que visita cada aresta exatamente uma vez. Um ciclo de Euler, ou Tour de Euler, em um grafo é um ciclo que usa cada aresta exatamente uma vez.

Teorema: Um grafo conectado tem um ciclo de Euler se, e somente se, cada vértice possui grau par.

Caminho Hamiltoniano

Um caminho Hamiltoniano em um grafo é o caminho que visita cada vértice exatamente uma vez. Um ciclo Hamiltoniano em um grafo é um ciclo que visita cada vértice exatamente uma vez.

Teoremas:

- Teorema de Dirac: Um grafo simples com n vértices ($n \geq 3$) é Hamiltoniano se cada vértice tem grau $\geq \frac{n}{2}$.
- Teorema de Ore: Um grafo simples com n vértices ($n \geq 3$) é Hamiltoniano se, para cada par de vértices não-adjacentes, a soma de seus graus é $\geq n$.
- Ghouila-Houiri: Um grafo direcionado simples fortemente conexo com n vértices é Hamiltoniano se cada vértice tem um grau $\geq n$.
- Meyniel: Um grafo direcionado simples fortemente conexo com n vértices é Hamiltoniano se a soma dos graus de cada par de vértices não-adjacentes é $\geq 2n - 1$.

10.3 Análise Combinatória

10.3.1 Permutação e Arranjo

Uma r-permutação de n objetos é uma seleção **ordenada** (ou arranjos) de r deles.

1. **Objetos distintos.**
$$P(n,r) = \frac{n!}{(n-r)!}$$

2. **Objetos com repetição.** Se temos n objetos com k_1 do tipo 1, k_2 do tipo 2,…,k_m do tipo m, e $\sum k_i = n$:
$$P(n; k_1, k_2, ..., k_m) = \frac{n!}{k_1! \cdot k_2! \cdot ... \cdot k_m!}$$

3. **Repetição ilimitada.** Se temos n objetos e uma quantidade ilimitada deles:
$$P(n,r) = n^r$$

Tabela de fatoriais.

n	0	1	2	3	4	5	6	7	8	9	10
$n!$	1	1	2	6	24	120	720	5040	40320	362880	3628800

10.3.2 Combinação

Uma r-combinação de n objetos é um seleção de r deles, sem diferenciação de ordem.

1. **Objetos distintos.**
$$C(n,r) = \frac{n!}{r!(n-r)!} = \binom{n}{r}.$$

   Definimos também:
$$C(n,r) = C(n,n-r)$$
$$C(n,0) = C(n,n) = 1$$
$$C(n,r) = 0, \quad \text{para } r < 0 \text{ ou } r > n.$$

2. **Objetos com repetição (Stars and Bars).** Número de maneiras de dividir n objetos idênticos em k grupos:
$$C(n,k) = \binom{n+k-1}{n}$$

3. **Teorema Binomial.** Sendo a e b números reais quaisquer e n um número inteiro positivo, temos que:
$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k$$

4. **Triângulo de Pascal.** Triângulo com o elemento na n-ésima linha e k-ésima coluna denotado por $\binom{n}{k}$, satisfazendo:
$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \quad \text{para } n > k \geq 1.$$

Propriedades.

1. **Hockey-stick (soma sobre n).**
$$\sum_{m=0}^{n} \binom{m}{k} = \binom{n+1}{k+1}$$

2. **Soma sobre k.**
$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

$$\sum_{k=0}^{n}\binom{n}{2k}=\sum_{k=0}^{n}\binom{n}{2k+1}=2^{n-1}$$

3. **Soma sobre n e k.**
$$\sum_{k=0}^{m}\binom{n+k}{k}=\binom{n+m+1}{m}$$

4. **Soma com peso.**
$$\sum_{k=0}^{n}k\cdot\binom{n}{k}=n2^{n-1}$$

5. **(n+1)-ésimo termo da sequência de Fibonacci.**
$$\sum_{k=0}^{n}\binom{n-k}{k}=F_{n+1}$$

6. **Soma dos quadrados.**
$$\sum_{k=0}^{n}\binom{n}{k}^2=\binom{2n}{n}$$

10.3.3 Números de Catalan

O n-ésimo número de Catalan, C_n, pode ser calculado de duas formas:

1. **Fórmula recursiva:**
$$C_0=C_1=1$$
$$C_n=\sum_{k=0}^{n-1}C_kC_{n-1-k},\quad \text{para } n\geq 2.$$

2. **Fórmula analítica:**
$$C_n=\frac{1}{n+1}\binom{2n}{n}=\prod_{k=2}^{n}\frac{n+k}{k},\quad \text{para } n\geq 0$$

Tabela dos 10 primeiros números de Catalan.

n	0	1	2	3	4	5	6	7	8	9	10
C_n	1	1	2	5	14	42	132	429	1430	4862	16796

Aplicações

O número de Catalan C_n é a solução para os seguintes problemas:

- Número de sequências de parênteses balanceados consistindo de n pares de parênteses.
- Números de árvores binárias enraizadas cheias com $n+1$ folhas (vértices não são numerados), ou, equivalentemente, com um total de n nós internos. Uma árvore binária enraizada é cheia se cada vértice tem dois filhos ou nenhum.
- Número de maneiras de colocar parênteses completamente em $n+1$ fatores.
- Número de triangularizações de um polígono convexo com $n+2$ lados.
- Número de maneiras de conectar $2n$ pontos em um círculo para formar n cordas disjuntas.
- Número de árvores binárias completas não isomórficas com $n+1$ nós.
- Número de caminhos monotônicos na grade de pontos do ponto $(0,0)$ ao ponto (n,n) em uma grade quadrada de tamanho nxn, que não passam acima da diagonal principal.
- Número de partições não cruzadas de um conjunto de n elementos.
- Números de manieras de se cobrir uma escada $1\ldots n$ usando n retângulos (a escada possui n colunas e a i-ésima coluna possui altura i).
- Número de permutações de tamanho n que podem ser *stack sorted*.

10.3.4 Princípio da Inclusão-Exclusão

Para calcular o tamanho da união de múltiplos conjuntos, é necessário somar os tamanhos desses conjuntos **separadamente**, e depois subtrair os tamanhos de todas as interseções **em pares** dos conjuntos, em seguida adicionar de volta o tamanho das interseções de **trios** dos conjuntos, subtrair o tamanho das interseções de **quartetos** dos conjuntos, e assim por diante, até a interseção de **todos** os conjuntos.

$$|\bigcup_{i=1}^{n}A_i|=\sum_{\emptyset\neq J\subseteq\{1,2,\ldots n\}}(-1)^{|J|-1}|\bigcap_{j\in J}A_j|$$

10.3.5 Lema de Burnside / Teorema da Enumeração de Pólya

Para contar o número de classes de equivalência de um conjunto G, baseando-se na simetria interna, utilizamos a seguinte fórmula:

$$|\text{Classes}| = \frac{1}{|G|} \sum_{\pi \in G} k^{C(\pi)},$$

onde denotamos $C(\pi)$ como o número de ciclos em uma permutação π e k o número de valores que cada elemento de representação pode assumir.

10.4 Álgebra

10.4.1 Fundamentos

Maior Divisor Comum (MDC). Dados dois inteiros não-negativos a e b, o maior número que é um divisor de tanto de a quanto de b é chamado de MDC.

$$\gcd(a, b) = max\{d > 0 : (d|a) \wedge (d|b)\}$$

Menor Múltiplo Comum (MMC). Dados dois inteiros não-negativos a e b, o menor número que é múltiplo de tanto de a quanto de b é chamado de MMC.

$$lcm(a, b) = \frac{ab}{\gcd(a, b)}$$

Equação Diofantina Linear. Um Equação Diofantina Linear é uma equação de forma geral:

$$ax + by = c,$$

onde a,b,c são inteiros dados, e x,y são inteiros desconhecidos.

Para achar uma solução de uma equação Diofantina com duas incógnitas, podemos utilizar o algoritmo de Euclides. Quando aplicamos o algoritmo em a e b, podemos encontrar seu MDC d e dois números x_d e y_d tal que:

$$a \cdot x_d + b \cdot y_d = d.$$

Se c é divisível por $d = \gcd(a, b)$, logo a equação Diofantina tem solução, caso contrário ela não tem nenhuma solução.

Supondo que c é divisível por g, obtemos:

$$a \cdot (x_d \cdot \frac{c}{d}) + b \cdot (y_d \cdot \frac{c}{d}) = c.$$

Logo uma das soluções da equação Diofantina é:

$$x_0 = x_d \cdot \frac{c}{d}$$

$$y_0 = y_d \cdot \frac{c}{d}.$$

A partir de uma solução (x_0, y_0), podemos obter todas as soluções. São soluções da equação Diofantina todos os números da forma:

$$x = x_0 + k \cdot \frac{b}{d}$$

$$y = y_0 - k \cdot \frac{a}{d}.$$

Números de Fibonacci. A sequência de Fibonacci é definida da seguinte forma:

$$F_n = \begin{cases} 0, \text{se } n = 0 \\ 1, \text{se } n = 1 \\ F_{n-1} + F_{n-2}, \text{caso contrário} \end{cases}$$

Os 11 primerios números da sequência são:

n	0	1	2	3	4	5	6	7	8	9	10
F_n	0	1	1	2	3	5	8	13	21	34	55

Propriedades.

- Identidade de Cassini: $F_{n-1}F_{n+1} - {F_n}^2 = (-1)^n$
- Regra da adição: $F_{n+k} = F_kF_{n+1} + F_{k-1}F_n$
- Identidade do MDC: $\gcd(F_n, F_m) = F_{\gcd(n,m)}$

Fórmulas para calcular o n-ésimo número de Fibonacci.

- Forma matricial:

$$\begin{vmatrix} 1 & 1 \\ 1 & 0 \end{vmatrix}^n = \begin{vmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{vmatrix}$$

10.4.2 Funções

Função Totiente de Euler. A função-phi $\phi(n)$ conta o número de inteiros entre 1 e n incluso, nos quais são coprimos com n. Dois números são coprimos se o MDC deles é igual a 1.

Propriedades.

- Se p é primo, logo o $\gcd(p,q) = 1$ para todo $1 \leq q < p$. Logo,
$$\phi(p) = p - 1$$
- Se p é primo e $k \geq 1$, então há exatos p^k/p números entre 1 e p^k que são divisíveis por p. Portanto,
$$\phi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1)$$
- Se a e b forem coprimos ou não, então:
$$\phi(ab) = \phi(a) \cdot \phi(b) \cdot \frac{d}{\phi(d)}, \quad d = \gcd(a,b)$$
- Fórmula do produto de Euler:
$$\phi(n) = n \prod_{p|n} (1 - \frac{1}{p})$$
- Soma dos divisores:
$$n = \sum_{d|n} \phi(d)$$

Aplicações:

- Teorema de Euler: Seja m um inteiro positivo e a um inteiro coprimo com m, então:
$$a^{\phi(m)} \equiv 1 \pmod m$$
$$a^n \equiv a^{n \pmod \phi)(m)} \pmod m$$
- Generalização do Teorema de Euler: Seja x,m inteiros positivos e $n \geq \log_2 m$,
$$x^n \equiv x^{\phi(m)+[n \pmod{\phi(m)})]} \pmod m$$
- Teoria dos Grupos: $\phi(n)$ é a ordem de um grupo multiplicativo mod n $(\mathbb{Z}/n\mathbb{Z})^\times$, que é o grupo dos elementos com inverso multiplicativo (aqueles coprimos com n). A ordem multiplicativa de um elemento a mod m ($\text{ord}_m(a)$), na qual também é o tamanho do subgrupo gerado por a, é o menor $k > 0$ tal que $a^k \equiv 1 \pmod m$. Se a ordem multiplicativa de a é $\phi(m)$, o maior possível, então a é **raiz primitiva** e o grupo é cíclico por definição.

Número de Divisores. Se a fatoração prima de n é $p_1^{e_1} \cdot p_2^{e_2} \dots p_k^{e_k}$, onde p_i são números primos distintos, então o número de divisores é dado por:

$$d(n) = (e_1 + 1) \cdot (e_2 + 1) \dots (e_k + 1)$$

Um número altamente composto (HCN) é um número inteiro que possui mais divisores do que qualquer número inteiro positivo menor.

n	6	60	360	5040	83160	720720	735134400	74801040398884800
$d(n)$	4	12	24	60	128	240	1344	64512

Soma dos Divisores. Para $n = p_1^{e_1} \cdot p_2^{e_2} \dots p_k^{e_k}$ temos a seguinte fórmula:

$$\sigma(n) = \frac{p_1^{e_1+1} - 1}{p_1 - 1} \cdot \frac{p_2^{e_2+1} - 1}{p_2 - 1} \dots \frac{p_k^{e_k+1} - 1}{p_k - 1}$$

Contagem de números primos. A função $\pi(n)$ conta a quantidade de números primos menores ou iguais à algum número real n. Pelo Teorema do Número Primo, a função tem crescimento aproximado à $\frac{x}{\ln(x)}$.

n	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8
$\pi(n)$	4	25	168	1229	9592	78489	664579	5761455

10.4.3 Aritmética Modular

Dado um inteiro $m \geq 1$, chamado módulo, dois inteiros a e b são ditos congruentes módulo m, se existe um inteiro k tal que

$$a - b = km,$$

Congruência módulo m é denotada: $a \equiv b \pmod m$

Propriedades.

- $(a \pm b) \pmod m = (a \mod m \pm b \mod m) \pmod m$.
- $(a \cdot b) \pmod m = (a \mod m) \cdot (b \mod m) \pmod m$.
- $a^b \pmod m = (a \mod m)^b \pmod m$.
- $a \pm k \equiv b \pm k \pmod m$, para qualquer inteiro k.
- $a \cdot k \equiv b \cdot k \pmod m$, para qualquer inteiro k.
- $a \cdot k \equiv b \cdot k \pmod{k \cdot m}$, para qualquer inteiro k.

Inverso Multiplicativo Modular. O inverso multiplicativo modular de um número a é um inteiro a^{-1} tal que

$$a \cdot a^{-1} \equiv 1 \pmod m.$$

O inverso modular existe se, e somente se, a e m são coprimos.

Um método para achar o inverso modular é usando o Teorema de Euler. Multiplicando ambos os lados da equação do teorema por a^{-1} obtemos:

$$a^{\phi(m)} \equiv 1 \pmod m \xrightarrow{\times(a^{-1})} a^{\phi(m)-1} \equiv a^{-1} \pmod m$$

Equação de Congruência Linear. Essa equação é da forma:

$$a \cdot x \equiv b \pmod m,$$

onde a,b e m são inteiros conhecidos e x uma incógnita.

Uma forma de achar uma solução é via achando o elemento inverso. Seja $g = \gcd(a, m)$, se b não é divisível por g, não há solução.

Se g divide b, então ao dividir ambos os lados da equação por g (a,b e m), recebemos uma nova equação:

$$a' \cdot x \equiv b' \pmod{m'}.$$

Como a' e m' são coprimo, podemos encontrar o inverso a', e multiplicar ambos os lados da equação pelo inverso, e então obtemos uma solução única.

$$x \equiv b' \cdot a'^{-1} \pmod{m'}$$

A equação original possui exatas g soluções, e elas possuem a forma:

$$x_i \equiv (x + i \cdot m') \pmod m, \quad 0 \leq i \leq g - 1.$$

Teorema do Resto Chinês. Seja $m = m_1 \cdot m_2 \cdot \dots \cdot m_k$, onde m_i são coprimos dois a dois. Além de m_i, recebemos também um sistema de congruências

$$\begin{cases} a \equiv a_1 \pmod{m_1} \\ a \equiv a_2 \pmod{m_2} \\ \vdots \\ a \equiv a_k \pmod{m_k} \end{cases}$$

onde a_i são constantes dadas. O teorema afirma que o sistema de congruências dado sempre tem uma e apenas uma solução módulo m.

Seja $M_i = \prod_{i \neq j} m_j$, o produto de todos os módulos menos m_i, e N_i os inversos modulares $N_i = M_i^{-1} \mod m_i$. Então, a solução do sistema de congruências é:

$$a \equiv \sum_{i=1}^{k} a_i M_i N_i \pmod{m_1 \cdot m_2 \cdot \ldots m_k}.$$

Para módulos não coprimos, o sistema de congruências tem exatas uma solução módulo $lcm(m_1, m_2, \ldots, m_k)$, ou tem nenhuma solução.

Uma única congruência $a \equiv a_i \pmod{m_i}$ é equivalente ao sistema de congruências $a \equiv a_i \pmod{p_j^{n_j}}$, onde $p_1^{n_1} p_2^{n_2} \ldots p_k^{n_k}$ é a fatoração prima de m_i. A congruência com o maior módulo de potência prima será a congruência mais forte dentre todas as congruências com a mesma base prima. Ou dará uma contradição com alguma outra congruência, ou implicará já todas as outras congruências.

Se não há contradições, então o sistema de equações tem uma solução. Podemos ignorar todas as congruências, exceto aquelas com os módulos de maior potência de primo. Esses módulos agora são coprimos e, portanto, podemos resolver com o algoritmo do caso geral.

Logaritmo discreto. Sejam a, b, k, m inteiros, queremos encontrar x tal que a equação seja válida:

$$ka^x \equiv b \pmod{m}.$$

Para encontrá-lo:

1. Reescrevemos $x = np - q$, onde obteremos a seguinte equação:
$$ka^{np-q} \equiv b \pmod{m}, \quad n = \sqrt{m} + 1.$$
2. No caso de $g = \gcd(a, m) = 1$, obtemos:
$$ka^{np} \equiv ba^q \pmod{m}.$$
3. Caso contrário:
   (a) Se $g \nmid b$, a equação não possui solução.
   (b) Se $g \mid b$, escrevemos $a = g\alpha$, $b = g\beta$, $m = g\mu$, e obtemos a seguinte equação:
$$k(g\alpha)a^{x-1} \equiv g\beta \pmod{g\mu}$$
$$(k\alpha)a^{x-1} \equiv \beta \pmod{\mu}$$
4. Para todo $q \in [0, n]$, calculamos todos os valores possíveis de $f_1(q) = ba^q \pmod{m}$.
5. Por fim, para todo $p \in [0, n]$, calculamos todos os valores possíveis de $f_2(p) = ka^{np} \pmod{m}$ até encontrarmos um valor p tal que
$$f_1(q) = f_2(p).$$

Seguindo esses passos, iremos encontrar o menor x que tornará a equação válida.

Raiz primitiva. Um número g é raiz primitiva módulo m se e somente se para qualquer inteiro a tal que $\gcd(a, n) = 1$, existe um inteiro k tal que:

$$g^k \equiv a \pmod{m}.$$

k é chamado de índice ou logaritmo discreto de a na base g módulo m. g é chamado de generador do grupo multiplicativo dos inteiros módulo m.

A raiz primitiva módulo m existe se e somente se:

- m é 1,2,4, ou
- m é um potência de um primo ímpar ($m = p^k$), ou
- m é o dobro de uma potência de um primo ímpar ($m = 2 \cdot p^k$).

Para encontrar a raiz primitiva:

1. Encontrar $\phi(m)$ (Função Totiente de Euler) e fatorizá-lo.
2. Iterar por todos os números $g \in [1, m]$, e para cada número, para verificar se é raiz primitiva, fazemos:
   (a) Calcular todos $g^{\frac{\phi(m)}{p_i}} \pmod{m}$.
   (b) Se todos o valores são diferentes de 1, então g é uma raiz primitiva.

Raiz discreta. Sejam k, a inteiros e m um primo, queremos encontrar todo x tal que:

$$x^k \equiv a \pmod{m}.$$

Seja g a raiz primitiva módulo m, podemos representar a raiz discreta como uma potência de g. Assim, podemos reescrever a equação como:

$$x^k \equiv (g^y)^k \equiv a \pmod{m}$$
$$(g^k)^y \equiv a \pmod{m}.$$

Por fim, basta resolver o logaritmo discreto para descobrir uma solução.

Ao achar uma solução $x_0 = g^{y_0} \pmod{m}$, as demais soluções possuem a forma:

$$x_i = g^{y_0 + i\frac{\phi(m)}{\gcd(\phi(m), k)}}, \quad i \in \mathbb{Z}.$$

10.5 Matrizes

10.5.1 Determinante

A determinante $det(A)$ de uma matriz A é definida se A é uma matriz quadrada. Se A é de tamanho 1×1, então $det(A) = A_{11}$. A determinante de matrizes maiores é calculada recursivamente usando a fórmula:

$$det(A) = \sum_{j=1}^{m} A_{1,j} C_{1,j},$$

onde $C_{i,j}$ é o **cofator** de A em i, j. O cofator é calculado usando a fórmula:

$$C_{i,j} = (-1)^{i+j} det(M_{i,j}),$$

onde $M_{i,j}$ é obtido ao remover a linha i e a coluna j de A.

A determinante de A indica se existe uma **matriz inversa** A^{-1} tal que $AA^{-1} = I$, onde I é uma matriz identidade. A^{-1} existe somente quando $det(A) \neq 0$, e pode ser calculada usando a fórmula:

$$A^{-1}_{i,j} = \frac{C_{i,j}}{det(A)}.$$

10.5.2 Aplicações

Contando caminhos. Quando V é a matriz de adjacência de um grafo sem peso, a matriz V^n contém o número de caminhos de n arestas entres os vértices do grafo.

Menores caminhos. Usando uma ideia similar para grafos com peso, podemos calcular para cada par de vértices a distância mínima para um caminho entre eles no qual contém exatos n arestas. Construímos a matriz de adjacência onde ∞ significa que uma aresta não existe, e outros valores correspondem ao peso da aresta. Utilizamos a fórmula

$$AB_{i,j} = \min_{k=1}^{n} A_{i,k} + B_{k,j}$$

na multiplicação de matriz. Após essa modificação, as potências da matriz correspondem aos menores caminhos no grafo.

Teorema de Kirchhoff. Para calcular o número de árvores geradoras de um grafo, construímos uma matrix laplaciana L, onde $L_{i,i}$ é o grau do vértice i e $L_{i,j} = -1$ se há uma aresta entre os vértices i e j, caso contrário $L_{i,j} = 0$. O número de árvores geradoras é igual à determinante da matriz que é obtida ao removermos qualquer linha e qualquer coluna de L.

10.6 Teoria da Probabilidade

10.6.1 Introdução à Probabilidade

Eventos. Um evento pode ser representado como um conjunto $A \subset X$ onde X contém todos os resultudos possíveis e A é um subconjunto de resultados.

Cada resultado x é designado uma probabilidade $p(x)$. Então, a probabilidade $P(A)$ de um evento A pode ser calculada como a soma das probabilidades dos resultados:

$$P(A) = \sum_{x \in A} p(x).$$

Complemento. A probabilidade do complemento $\overline{A}$, *i.e.* o evento A não ocorrer, é dado por:

$$P(\overline{A}) = 1 - P(A).$$

Eventos não mutualmente exclusivos. A probabilidade da união $A \cup B$ é dada por:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B).$$

Se A e B forem eventos mutualmente exclusivos, *i.e.* $A \cup B = \emptyset$, a probabilidade é dada por:

$$P(A \cup B) = P(A) + P(B).$$

Probabilidade condicional. A probabilidade de A assumindo que B ocorreu é dada por:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

Os eventos A e B são ditos **independentes** se, e somente se,

$$P(A|B) = P(A) \quad \text{e} \quad P(B|A) = P(B).$$

Teorema de Bayes. A probabilidade de um evento A ocorrer, antes e depois de condicionar em outro evento B é dada por:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad \text{ou} \quad P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{j \in A} P(B|A_j)P(A_j)}$$

10.6.2 Variáveis Aleatórias

Seja X uma variável aleatória discreta com probabilidade $P(X = x)$ de assumir o valor x. Ela vai então ter um valor esperado (média)

$$\mu = E[X] = \sum_{i=1}^{n} x_i P(X = x_i)$$

e variância

$$\sigma^2 = V[X] = E[X^2] - (E[X])^2 = \sum_{i=1}^{n} (x - E[X])^2 P(X = x_i)$$

onde σ é o desvio-padrão.

Se X for contínua ela terá uma função de densidade $f_X(x)$ e as somas acima serão em vez disso integrais com $P(X = x)$ substituído por $f_X(x)$.

Linearidade do Valor Esperado.

$$E[aX + bY + c] = aE[X] + bE[Y] + c.$$

No caso de X e Y serem independentes, temos que:

$$E[XY] = E[X]E[Y]$$

$$V[aX + bY + c] = a^2 E[X] + b^2 E[Y].$$

10.6.3 Distribuições Discretas

Distribuição Binomial. Número de sucessor k em n experimentos independentes de sucesso/fracasso, cada um dos quais produz sucesso com probabilidade p é Bin(n,p), $n \in \mathbb{N}$, $0 \leq p \leq 1$.

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np, \quad \sigma^2 = np(1-p)$$

Bin(n,p) é aproximadamente Pois(np) para p pequeno.

Distribuição Geométrica. Número de tentativas k necessárias para conseguir o primeiro sucesso em experimentos independentes de sucesso/fracasso, cada um dos quais produz sucesso com probabilidade p é Geo(p), $0 \leq p \leq 1$.

$$P(X = k) = (1-p)^{k-1} p, \quad k \in \mathbb{N}$$

$$\mu = \frac{1}{p}, \quad \sigma^2 = \frac{1-p}{p}$$

Distribuição de Poisson. Número de eventos k ocorrendo em um período de tempo fixo t se esses eventos ocorrerem com uma taxa média conhecida r e independente do tempo já que o último evento é Pois(λ), $\lambda = tr$.

$$P(X = k) = e^{-k} \frac{\lambda^k}{k!}, \quad k \in \mathbb{N}_0$$

$$\mu = \lambda, \quad \sigma^2 = \lambda.$$

10.6.4 Distribuições Contínuas

Distribuição Uniforme. Se a função de densidade é constante entre a e b e 0 em outro lugar ela é Uni(a,b), $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a}, & a < x < b \\ 0, & \text{caso contrário} \end{cases}$$

$$\mu = \frac{a+b}{2}, \quad \sigma^2 = \frac{(b-a)^2}{12}.$$

Distribuição Exponencial. Tempo entre eventos em um processo de Poisson é Exp(λ), $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

$$F(x) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \quad \sigma^2 = \frac{1}{\lambda^2}.$$

Distribuição Normal. Maioria das variáveis aleatórias reais com média μ e variância σ^2 são bem descritas por $N(\mu, \sigma^2)$, $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

10.7 Progressões

1. Soma dos n primeiros termos.

$$\sum_{k=1}^{n}(k) = \frac{n(n+1)}{2}$$

2. Soma dos n primeiros quadrados.

$$\sum_{k=1}^{n}(k^2) = \frac{n(n+1)(2n+1)}{6}$$

3. Soma dos n primeiros cubos.

$$\sum_{k=1}^{n}(k^3) = (\frac{n(n+1)}{2})^2$$

4. Soma dos n primeiros pares.

$$\sum_{k=1}^{n}(2k) = n^2 + n$$

5. Soma dos n primeiros ímpares.

$$\sum_{k=1}^{n}(2k-1) = n^2$$

6. Progressão Aritmética (PA)

   (a) Termo geral a partir do k-ésimo termo.

   $$a_n = a_k + r(n-k)$$

   (b) Soma dos termos.

   $$\sum_{i=1}^{n}(a_i) = \frac{n(a_1 + a_n)}{2}$$

7. Progressão Geométrica (PG)

   (a) Termo geral a partir do k-ésimo termo.

   $$a_n = a_k r^{n-k}$$

   (b) Soma dos termos.

   $$\sum_{k=1}^{n}(ar^{k-1}) = \frac{a_1(r^n - 1)}{r-1}, \quad \text{para } r \neq 1.$$

   (c) Soma dos termos de uma progressão infinita.

   $$\sum_{k=1}^{\infty}(ar^{k-1}) = \frac{a_1}{1-r}, \quad \text{para } |q| < 1.$$

   (d) Produto dos termos.

   $$\prod_{k=0}^{n}(ar^k) = a^{n+1} r^{\frac{n(n+1)}{2}}$$

8. Série Harmônica 1.

$$\sum_{i=1}^{n}\frac{1}{i} \approx \ln n$$

9. Série Harmônica 2.

$$\sum_{i=1}^{\infty}\frac{(-1)^{i+1}}{i} = \ln 2$$

10.8 Álgebra Booleana

Álgebra booleana é a categoria da álgebra em que os valores das variáveis são os valores de verdade, verdadeiro e falso, geralmente denotados por 1 e 0, respectivamente.

10.8.1 Operações básicas

A álgebra booleana possui apenas três operações básicas: conjunção, disjunção e negação, expressas pelos operadores binários correspondentes E ($\wedge$) e OU ($\vee$) e pelo operador unário NÃO ($\neg$), coletivamente chamados de operadores booleanos.

Operador lógico	Operador	Notação	Definição
Conjunção	AND	$x \wedge y$	$x \wedge y = 1$ se $x = y = 1, x \wedge y = 0$ caso contrário
Disjunção	OR	$x \vee y$	$x \vee y = 0$ se $x = y = 0, x \wedge y = 1$ caso contrário
Negeação	NOT	$\neg x$	$\neg x = 0$ se $x = 1, \neg x = 1$ se $x = 0$

10.8.2 Operações secundárias

Operações compostas a partir de operações básicas incluem, dentro outras, as seguintes:

Operador lógico	Operador	Notação	Definição	Equivalência
Condicional material	$\rightarrow$	$x \rightarrow y$	$x \rightarrow y = 0$ se $x = 1$ e $y = 0, x \rightarrow y = 1$ caso contrário	$\neg x \vee y$
Bicondicional material	$\Leftrightarrow$	$x \Leftrightarrow y$	$x \Leftrightarrow y = 1$ se $x = y, x \Leftrightarrow y = 0$ caso contrário	$(x \vee \neg y) \wedge (\neg x \vee y)$
OR Exclusivo	XOR	$x \oplus y$	$x \oplus y = 1$ se $x \neq y, x \oplus y = 0$ caso contrário	$(x \vee y) \wedge (\neg x \vee \neg y)$

10.8.3 Leis

- Associatividade:
$$x \wedge (y \wedge z) = (x \wedge y) \wedge z$$
$$x \vee (y \vee z) = (x \vee y) \vee z$$
- Comutatividade:
$$x \wedge y = y \wedge x$$
$$x \vee y = y \vee x$$
- Distributividade:
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$
$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$$
- Identidade: $x \vee 0 = x \wedge 1 = x$
- Aniquilador:
$$x \vee 1 = 1$$
$$x \wedge 0 = 0$$
- Idempotência: $x \wedge x = x \vee x = x$
- Absorção: $x \wedge (x \vee y) = x \vee (x \wedge y) = x$
- Complemento:
$$x \wedge \neg x = 0$$
$$x \vee \neg x = 1$$
- Negação dupla: $\neg(\neg x) = x$
- De Morgan:
$$\neg x \wedge \neg y = \neg(x \vee y)$$
$$\neg x \vee \neg y = \neg(x \wedge y)$$