



# Notebook - Maratona de Programação

[UnB] HatsuneMiku não manda WA

## Contents

<b>1</b>	<b>Informações</b>	<b>2</b>			
1.1	Compilação e Execução	2		5.3	Edit Distance
1.2	Ferramentas para Testes	2		5.4	Lcsubseq
<b>2</b>	<b>Misc</b>	<b>3</b>		5.5	Kmp
2.1	Submask	3		5.6	Hash
2.2	Safe Map	3		5.7	Aho Corasick
2.3	Ordered Set	3		5.8	Lcs
2.4	Bitwise	3	<b>6</b>	<b>Geometria</b>	<b>8</b>
2.5	Template	3	6.1	Polygon Diameter	8
<b>3</b>	<b>DP</b>	<b>3</b>	6.2	Mindistpair	9
3.1	Knapsack	3	6.3	Inside Polygon	9
3.2	Lis	4	6.4	Polygon Cut Length	9
3.3	Dp Digitos	4	6.5	3d	9
<b>4</b>	<b>ED</b>	<b>4</b>	6.6	Convex Hull	10
4.1	Prefixsum2d	4	6.7	Linear Transformation	10
4.2	Sparse Table	4	6.8	Voronoi	11
4.3	Dsu	5	6.9	Intersect Polygon	11
4.4	Minqueue	5	6.10	Sort By Angle	11
4.5	Segtree Implicita	5	6.11	2d	11
4.6	Segtree Implicita Lazy	6	<b>7</b>	<b>Grafos</b>	<b>14</b>
4.7	Segtree	6	7.1	Dfs Tree	14
4.8	Delta Encoding	7	7.2	Kosaraju	14
<b>5</b>	<b>Strings</b>	<b>7</b>	7.3	Topological Sort	14
5.1	Suffix Array	7	7.4	Dijkstra	14
5.2	Z Func	7	7.5	Dinic	14
			7.6	Centroid Decomp	15
			7.7	Hungarian	16
			7.8	Floyd Warshall	16
			7.9	2sat	16

7.10	Lca . . . . .	17	10.1.4	Geometria Espacial . . . . .	23
7.11	Kruskal . . . . .	17	10.1.5	Trigonometria . . . . .	24
7.12	Mcmf . . . . .	18	10.2	Teoria dos Grafos . . . . .	24
7.13	Ford . . . . .	18	10.3	Análise Combinatória . . . . .	24
<b>8</b>	<b>Algoritmos</b>	<b>19</b>	10.3.1	Permutação e Arranjo . . . . .	24
8.1	Ternary Search . . . . .	19	10.3.2	Combinação . . . . .	25
<b>9</b>	<b>Math</b>	<b>19</b>	10.3.3	Números de Catalan . . . . .	25
9.1	Totient . . . . .	19	10.3.4	Princípio da Inclusão-Exclusão . . . . .	26
9.2	Pollard Rho . . . . .	19	10.4	Álgebra . . . . .	26
9.3	Inverso Mult . . . . .	20	10.4.1	Fundamentos . . . . .	26
9.4	Miller Habin . . . . .	20	10.4.2	Funções . . . . .	27
9.5	Matrix Exponentiation . . . . .	20	10.4.3	Aritmética Modular . . . . .	28
9.6	Division Trick . . . . .	20	10.5	Matrizes . . . . .	30
9.7	Crivo . . . . .	21	10.6	Teoria da Probabilidade . . . . .	31
9.8	Bigmod . . . . .	21	10.6.1	Introdução à Probabilidade . . . . .	31
9.9	Linear Diophantine Equation . . . . .	21	10.6.2	Variáveis Aleatórias . . . . .	31
<b>10</b>	<b>Teoria</b>	<b>21</b>	10.6.3	Distribuições Discretas . . . . .	32
10.1	Geometria . . . . .	21	10.6.4	Distribuições Contínuas . . . . .	32
10.1.1	Geometria Básica . . . . .	21	10.7	Progressões . . . . .	32
10.1.2	Geometria Analítica . . . . .	22	10.8	Álgebra Booleana . . . . .	33
10.1.3	Geometria Plana . . . . .	22	10.8.1	Operações básicas . . . . .	33
			10.8.2	Operações secundárias . . . . .	33
			10.8.3	Leis . . . . .	33

# 1 Informações

## 1.1 Compilação e Execução

### Comandos de compilação

- C++:

```
g++ -std=c++17 -g3 -fsanitize=address -O3 -Wall  
-Wextra -Wconversion -Wshadow -o <nomeDoExecutável> <nomeDoArquivo>.cpp
```

- Java: `javac <nomeDoArquivo>.java`.
- Haskell: `ghc -o <nomeDoExecutável> <nomeDoArquivo>.hs`.

### Comandos de execução

- C++: `./<nomeDoExecutável>`.
- Java: `java -Xms1024m -Xmx1024m -Xss20m <nomeDoArquivo>`.
- Python: `python3 <nomeDoArquivo>.py`.
- Haskell: `./<nomeDoExecutável>`.

## 1.2 Ferramentas para Testes

### Python

```
1 import random  
2 import itertools  
3  
4 #randint: retorna um numero aleatorio x tq. a  
5 <= x <= b  
6 lista = [random.randint(1,100) for i in range  
7 (101)]  
8  
9 #shuffle: embaralha uma sequencia  
10 random.shuffle(lista)  
11  
12 #sample: retorna uma lista de k elementos  
13 unicos escolhidos de uma sequencia  
14 amostra = random.sample(lista, k = 10)  
15  
16 lista2 = [1,2,3,4,5]  
17 #permutations: iterable que retorna  
18 permutacoes de tamanho r  
19 permutacoes = [perm for perm in itertools.  
20 permutations(lista2, 2)]  
21  
22 #combinations: iterable que retorna  
23 combinacoes de tamanho r (ordenado)  
24 #combinations_with_replacement: combinations  
25 () com elementos repetidos  
26 combinacoes = [comb for comb in itertools.  
27 combinations(lista2, 2)]  
28  
29  
30
```

### C++

```
1 mt19937 rng(chrono::steady_clock::now().  
2 time_since_epoch().count()); // mt19937_64  
3 uniform_int_distribution<int> distribution(1,  
4 n);  
5  
6 num = distribution(rng); // num no range [1,  
7 n]  
8 shuffle(vec.begin(), vec.end(), rng); //  
9 shuffle  
10  
11 // permutacoes  
12 do {  
13 // codigo  
14 } while(next_permutation(vec.begin(), vec.end  
15 ()))  
16  
17 using ull = unsigned long long;  
18 ull mix(ull o){  
19 o+=0x9e3779b97f4a7c15;  
20 o=(o^(o>>30))*0xbf58476d1ce4e5b9;  
21 o=(o^(o>>27))*0x94d049bb133111eb;  
22 return o^(o>>31);  
23 }  
24  
25 ull hash(pii a) {return mix(a.first ^ mix(a.  
26 second));}
```

## 2 Misc

### 2.1 Submask

```
1 // 0(3~n)
2 for (int m = 0; m < (1<<n); m++) {
3     for (int s = m; s; s = (s-1) & m) {
4         // s is every submask of m
5     }
6 }
7
8 // 0(2~n * n) SOS dp like
9 for (int b = n-1; b >= 0; b--) {
10     for (int m = 0; m < (1 << n); m++) {
11         if (j & (1 << b)) {
12             // propagate info through submasks
13             amount[j ^ (1 << b)] += amount[j];
14         }
15     }
16 }
```

### 2.2 Safe Map

```
1 struct custom_hash {
2     static uint64_t splitmix64(uint64_t x) {
3         // http://xorshift.di.unimi.it/splitmix64.c
4         x += 0x9e3779b97f4a7c15;
5         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
6         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
7         return x ^ (x >> 31);
8     }
9
10     size_t operator()(uint64_t x) const {
11         static const uint64_t FIXED_RANDOM = chrono::
steady_clock::now().time_since_epoch().count();
12         return splitmix64(x + FIXED_RANDOM);
13     }
14 };
15
16 unordered_map<long long, int, custom_hash> safe_map;
17
18 // when using pairs
19 struct custom_hash {
20     inline size_t operator()(const pii & a) const {
21         return (a.first << 6) ^ (a.first >> 2) ^
2038074743 ^ a.second;
22     }
23 };
```

### 2.3 Ordered Set

```
1 #include <bits/extc++.h>
2 // or
3 #include <ext/pb_ds/assoc_container.hpp>
4 #include <ext/pb_ds/tree_policy.hpp>
5
6 using namespace __gnu_pbds; // or pb_ds;
7 template<typename T, typename B = null_type>
8 using ordered_set = tree<T, B, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
9
10 // order_of_key(k) : Number of items strictly
smaller than k
11 // find_by_order(k) : K-th element in a set (counting
from zero)
12
13 // to erase an element -> order_of_key(k) +
find_by_order(k) + erase(itr)
14
15 // to swap two sets, use a.swap(b);
```

## 2.4 Bitwise

```
1 // Least significant bit (lsb)
2 int lsb(int x) { return x&-x; }
3 int lsb(int x) { return __builtin_ctz(x); } //
bit position
4 // Most significant bit (msb)
5 int msb(int x) { return 32-1-__builtin_clz(x); }
// bit position
6
7 // Power of two
8 bool isPowerOfTwo(int x){ return x && !(x&(x-1))
}; }
9
10 // floor(log2(x))
11 int flog2(int x) { return 32-1-__builtin_clz(x); }
12 int flog2ll(ll x) { return 64-1-__builtin_clzll(x); }
13
14 // Built-in functions
15 // Number of bits 1
16 __builtin_popcount()
17 __builtin_popcountll()
18
19 // Number of leading zeros
20 __builtin_clz()
21 __builtin_clzll()
22
23 // Number of trailing zeros
24 __builtin_ctz()
25 __builtin_ctzll()
```

## 2.5 Template

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define ll long long
5 #define ff first
6 #define ss second
7 #define ld long double
8 #define pb push_back
9 #define sws cin.tie(0)->sync_with_stdio(false);
10 #define endl '\n'
11 #ifdef LOCAL
12 #define debug(var) cout << (var) << " = " << var <<
endl;
13 #endif
14 #ifndef LOCAL
15 #define debug(...)
16 #endif
17
18 const ll MOD = 998244353;
19 const int INF = 0x3f3f3f3f;
20 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
21
22 signed main() {
23     #ifndef LOCAL
24         sws;
25     #endif
26
27     return 0;
28 }
```

## 3 DP

### 3.1 Knapsack

```
1 // Caso base, como i == n
2 dp[0][0] = 0;
3
4 // Itera por todos os estados
```

```

5 for(int i = 1; i <= n; ++i)
6   for(int P = 0; P <= w; ++P){
7     int &temp = dp[i][P];
8     // Primeira possibilidade, ão pega i
9     temp = dp[i - 1][P];
10
11    // Segunda possibilidade, se puder, pega o
    item
12    if(P - p[i] >= 0)
13      temp = max(temp, dp[i - 1][P - p[i]] + v[
14      i]);
15
16    ans = max(ans, temp);
  }

```

## 3.2 Lis

```

1 multiset<int> S;
2 for(int i=0; i<n; i++){
3   auto it = S.upper_bound(vet[i]); // low for inc
4   if(it != S.end())
5     S.erase(it);
6   S.insert(vet[i]);
7 }
8 // size of the lis
9 int ans = S.size();
10
11 vi LIS(const vi &elements){
12   auto compare = [&](int x, int y) {
13     return elements[x] < elements[y];
14   };
15   set< int, decltype(compare) > S(compare);
16
17   vi previous( elements.size(), -1 );
18   for(int i=0; i<int( elements.size() ); ++i){
19     auto it = S.insert(i).first;
20     if(it != S.begin())
21       previous[i] = *prev(it);
22     if(*it == i and next(it) != S.end())
23       S.erase(next(it));
24   }
25
26   vi answer;
27   answer.push_back( *S.rbegin() );
28   while ( previous[answer.back()] != -1 )
29     answer.push_back( previous[answer.back()] );
30   reverse( answer.begin(), answer.end() );
31   return answer;
32 }

```

## 3.3 Dp Digitos

```

1 // dp de quantidade de numeros <= r com ate qt
  digitos diferentes de 0
2 ll dp(int idx, string& r, bool menor, int qt, vector<
  vector<vi>>& tab) {
3   if(qt > 3) return 0;
4   if(idx >= r.size()) {
5     return 1;
6   }
7   if(tab[idx][menor][qt] != -1)
8     return tab[idx][menor][qt];
9
10  ll res = 0;
11  for(int i = 0; i <= 9; i++) {
12    if(menor or i <= r[idx] - '0') {
13      res += dp(idx+1, r, menor or i < (r[idx] -
14      '0'), qt+(i>0), tab);
15    }
16  }
17  return tab[idx][menor][qt] = res;

```

```
18 }
```

## 4 ED

### 4.1 Prefixsum2d

```

1 ll find_sum(vector<vi> &mat, int x1, int y1, int x2,
  int y2){
2   // superior-esq(x1,y1) (x2,y2)inferior-dir
3   return mat[x2][y2]-mat[x2][y1-1]-mat[x1-1][y2]+
  mat[x1-1][y1-1];
4 }
5
6 int main(){
7
8   for(int i=1; i<=n; i++){
9     for(int j=1; j<=n; j++){
10      mat[i][j] += mat[i-1][j] + mat[i][j-1] - mat[i
11      -1][j-1];
12    }

```

### 4.2 Sparse Table

```

1 int logv[N+1];
2 void make_log() {
3   logv[1] = 0; // pre-comutar tabela de log
4   for (int i = 2; i <= N; i++)
5     logv[i] = logv[i/2] + 1;
6 }
7 struct Sparse {
8   int n;
9   vector<vector<int>> st;
10
11   Sparse(vector<int>& v) {
12     n = v.size();
13     int k = logv[n];
14     st.assign(n+1, vector<int>(k+1, 0));
15
16     for (int i=0; i<n; i++) {
17       st[i][0] = v[i];
18     }
19
20     for(int j = 1; j <= k; j++) {
21       for(int i = 0; i + (1 << j) <= n; i++) {
22         st[i][j] = f(st[i][j-1], st[i + (1 <<
23         (j-1))][j-1]);
24       }
25     }
26   }
27
28   int f(int a, int b) {
29     return min(a, b);
30   }
31
32   int query(int l, int r) {
33     int k = logv[r-l+1];
34     return f(st[l][k], st[r - (1 << k) + 1][k]);
35   };
36
37 struct Sparse2d {
38   int n, m;
39   vector<vector<vector<int>>> st;
40
41   Sparse2d(vector<vector<int>> mat) {
42     n = mat.size();
43     m = mat[0].size();
44     int k = logv[min(n, m)];
45   }
46 }

```

```

47     st.assign(n+1, vector<vector<int>>(m+1,
vector<int>(k+1)));
48     for(int i = 0; i < n; i++)
49         for(int j = 0; j < m; j++)
50             st[i][j][0] = mat[i][j];
51
52     for(int j = 1; j <= k; j++) {
53         for(int x1 = 0; x1 < n; x1++) {
54             for(int y1 = 0; y1 < m; y1++) {
55                 int delta = (1 << (j-1));
56                 if(x1+delta >= n or y1+delta >= m
) continue;
57
58                 st[x1][y1][j] = st[x1][y1][j-1];
59                 st[x1][y1][j] = f(st[x1][y1][j],
st[x1+delta][y1][j-1]);
60                 st[x1][y1][j] = f(st[x1][y1][j],
st[x1][y1+delta][j-1]);
61                 st[x1][y1][j] = f(st[x1][y1][j],
st[x1+delta][y1+delta][j-1]);
62             }
63         }
64     }
65 }
66
67 // so funciona para quadrados
68 int query(int x1, int y1, int x2, int y2) {
69     assert(x2-x1+1 == y2-y1+1);
70     int k = logv[x2-x1+1];
71     int delta = (1 << k);
72
73     int res = st[x1][y1][k];
74     res = f(res, st[x2 - delta+1][y1][k]);
75     res = f(res, st[x1][y2 - delta+1][k]);
76     res = f(res, st[x2 - delta+1][y2 - delta+1][k
]);
77     return res;
78 }
79
80 int f(int a, int b) {
81     return a | b;
82 }
83
84 };

```

### 4.3 Dsu

```

1 struct DSU {
2     int n;
3     vector<int> parent, size;
4
5     DSU(int n): n(n) {
6         parent.resize(n, 0);
7         size.assign(n, 1);
8
9         for(int i=0;i<n;i++)
10             parent[i] = i;
11     }
12
13     int find(int a) {
14         if(a == parent[a]) return a;
15         return parent[a] = find(parent[a]);
16     }
17
18     void join(int a, int b) {
19         a = find(a); b = find(b);
20         if(a != b) {
21             if(size[a] < size[b]) swap(a, b);
22             parent[b] = a;
23             size[a] += size[b];
24         }
25     }
26 };

```

### 4.4 Minqueue

```

1 struct MinQ {
2     stack<pair<ll,ll>> in;
3     stack<pair<ll,ll>> out;
4
5     void add(ll val) {
6         ll minimum = in.empty() ? val : min(val, in.
top().ss);
7         in.push({val, minimum});
8     }
9
10    ll pop() {
11        if(out.empty()) {
12            while(!in.empty()) {
13                ll val = in.top().ff;
14                in.pop();
15                ll minimum = out.empty() ? val : min(
val, out.top().ss);
16                out.push({val, minimum});
17            }
18        }
19        ll res = out.top().ff;
20        out.pop();
21        return res;
22    }
23
24    ll minn() {
25        ll minimum = LLINF;
26        if(in.empty() || out.empty())
27            minimum = in.empty() ? (ll)out.top().ss :
((ll)in.top().ss);
28        else
29            minimum = min((ll)in.top().ss, (ll)out.
top().ss);
30
31        return minimum;
32    }
33
34    ll size() {
35        return in.size() + out.size();
36    }
37 };

```

### 4.5 Segtree Implicita

```

1 // SegTree Implicita O(nlogMAX)
2
3 struct node{
4     int val;
5     int l, r;
6     node(int a=0, int b=0, int c=0){
7         l=a;r=b;val=c;
8     }
9 };
10
11 int idx=2; // 1-> root / 0-> zero element
12 node t[8600010];
13 int N;
14
15 int merge(int a, int b){
16     return a + b;
17 }
18
19 void update(int pos, int x, int i=1, int j=N, int no
=1){
20     if(i==j){
21         t[no].val+=x;
22         return;
23     }
24     int meio = (i+j)/2;
25
26     if(pos<=meio){

```

```

27         if(t[no].l==0) t[no].l=idx++;
28         update(pos, x, i, meio, t[no].l);
29     }
30     else{
31         if(t[no].r==0) t[no].r=idx++;
32         update(pos, x, meio+1, j, t[no].r);
33     }
34     t[no].val=merge(t[t[no].l].val, t[t[no].r].val);
35 }
36
37
38 int query(int A, int B, int i=1, int j=N, int no=1){
39     if(B<i or j<A)
40         return 0;
41     if(A<=i and j<=B)
42         return t[no].val;
43
44     int mid = (i+j)/2;
45
46     int ans1 = 0, ansr = 0;
47
48     if(t[no].l!=0) ans1 = query(A, B, i, mid, t[no].l);
49     if(t[no].r!=0) ansr = query(A, B, mid+1, j, t[no].r);
50
51     return merge(ans1, ansr);
52 }

```

## 4.6 Segtree Implicita Lazy

```

1 struct node{
2     pll val;
3     ll lazy;
4     ll l, r;
5     node(){
6         l=-1;r=-1;val={0,0};lazy=0;
7     }
8 };
9
10 node tree[40*MAX];
11 int id = 2;
12 ll N=1e9+10;
13
14 pll merge(pll A, pll B){
15     if(A.ff==B.ff) return {A.ff, A.ss+B.ss};
16     return (A.ff<B.ff ? A:B);
17 }
18
19 void prop(ll l, ll r, int no){
20     ll mid = (l+r)/2;
21     if(l!=r){
22         if(tree[no].l==-1){
23             tree[no].l = id++;
24             tree[tree[no].l].val = {0, mid-l+1};
25         }
26         if(tree[no].r==-1){
27             tree[no].r = id++;
28             tree[tree[no].r].val = {0, r-(mid+1)+1};
29         }
30         tree[tree[no].l].lazy += tree[no].lazy;
31         tree[tree[no].r].lazy += tree[no].lazy;
32     }
33     tree[no].val.ff += tree[no].lazy;
34     tree[no].lazy=0;
35 }
36
37 void update(int a, int b, int x, ll l=0, ll r=2*N, ll no=1){
38     prop(l, r, no);
39     if(a<=l and r<=b){
40         tree[no].lazy += x;
41         prop(l, r, no);

```

```

42         return;
43     }
44     if(r<a or b<l) return;
45     int m = (l+r)/2;
46     update(a, b, x, l, m, tree[no].l);
47     update(a, b, x, m+1, r, tree[no].r);
48
49     tree[no].val = merge(tree[tree[no].l].val, tree[tree[no].r].val);
50 }
51
52 pll query(int a, int b, int l=0, int r=2*N, int no=1)
53 {
54     prop(l, r, no);
55     if(a<=l and r<=b) return tree[no].val;
56     if(r<a or b<l) return {INF, 0};
57     int m = (l+r)/2;
58     int left = tree[no].l, right = tree[no].r;
59     return tree[no].val = merge(query(a, b, l, m, left),
60                                query(a, b, m+1, r, right));

```

## 4.7 Segtree

```

1 template <typename T> struct SegTree{
2     vector<T> st;
3     int n;
4     T zero;
5
6     SegTree(int n, T zero): n(n), zero(zero) {
7         st.resize(4*n);
8     }
9
10    T merge(const T & a, const T & b){
11        return a + b;
12    }
13
14    void update(int i, int x, int l, int r, int no){
15        if(l == r){
16            st[no] = x;
17        } else {
18            int mid = (l + r) >> 1;
19            if(i <= mid) update(i, x, l, mid, 2*no);
20            else update(i, x, mid + 1, r, 2*no + 1);
21            st[no] = merge(st[2*no], st[2*no + 1]);
22        }
23    }
24
25    T query(int gl, int gr, int l, int r, int no){
26        if(l >= gl && r <= gr) return st[no];
27        else if(l > gr || r < gl) return zero;
28        else {
29            int mid = (l + r) >> 1;
30            return merge(query(gl, gr, l, mid, 2*no),
31                          query(gl, gr, mid + 1, 2*no + 1));
32        }
33    }
34
35    void update(int i, int x){
36        update(i, x, 0, n - 1, 1);
37    }
38
39    T query(int l, int r){
40        return query(l, r, 0, n - 1, 1);
41    }
42 }
43
44 };

```

## 4.8 Delta Encoding

```
1 // Delta encoding
2
3 for(int i=0;i<q;i++){
4     int l,r,x;
5     cin >> l >> r >> x;
6     delta[l] += x;
7     delta[r+1] -= x;
8 }
9
10 int atual = 0;
11
12 for(int i=0;i<n;i++){
13     atual += delta[i];
14     v[i] += atual;
15 }
```

## 5 Strings

### 5.1 Suffix Array

```
1 vector<int> suffix_array(string s) {
2     s += "!";
3     int n = s.size(), N = max(n, 260);
4     vector<int> sa(n), ra(n);
5     for (int i = 0; i < n; i++) sa[i] = i, ra[i] = s[
6         i];
7
8     for (int k = 0; k < n; k ? k *= 2 : k++) {
9         vector<int> nsa(sa), nra(n), cnt(N);
10
11         for (int i = 0; i < n; i++) nsa[i] = (nsa[i]-
12             k+n)%n, cnt[ra[i]]++;
13         for (int i = 1; i < N; i++) cnt[i] += cnt[i
14             -1];
15         for (int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]
16             ]]] = nsa[i];
17
18         for (int i = 1, r = 0; i < n; i++) nra[sa[i]]
19             = r += ra[sa[i]] !=
20             ra[sa[i-1]] or ra[(sa[i]+k)%n] != ra[(sa[
21                 i-1]+k)%n];
22         ra = nra;
23         if (ra[sa[n-1]] == n-1) break;
24     }
25     return vector<int>(sa.begin()+1, sa.end());
26 }
27
28 vector<int> kasai(string s, vector<int> sa) {
29     int n = s.size(), k = 0;
30     vector<int> ra(n), lcp(n);
31     for (int i = 0; i < n; i++) ra[sa[i]] = i;
32
33     for (int i = 0; i < n; i++, k -= !!k) {
34         if (ra[i] == n-1) { k = 0; continue; }
35         int j = sa[ra[i]+1];
36         while (i+k < n and j+k < n and s[i+k] == s[j+
37             k]) k++;
38         lcp[ra[i]] = k;
39     }
40     return lcp;
41 }
```

### 5.2 Z Func

```
1 vector<int> Z(string s) {
2     int n = s.size();
3     vector<int> z(n);
4     int l = 0, r = 0;
5     for (int i = 1; i < n; i++) {
```

```
6         z[i] = max(0, min(z[i - 1], r - i + 1));
7         while (i + z[i] < n and s[z[i]] == s[i + z[i]
8             ]]) {
9             l = i; r = i + z[i]; z[i]++;
10        }
11    }
12    return z;
13 }
```

### 5.3 Edit Distance

```
1 int edit_distance(int a, int b, string& s, string& t)
2 {
3     // indexado em 0, transforma s em t
4     if(a == -1) return b+1;
5     if(b == -1) return a+1;
6     if(tab[a][b] != -1) return tab[a][b];
7
8     int ins = INF, del = INF, mod = INF;
9     ins = edit_distance(a-1, b, s, t) + 1;
10    del = edit_distance(a, b-1, s, t) + 1;
11    mod = edit_distance(a-1, b-1, s, t) + (s[a] != t[
12        b]);
13
14    return tab[a][b] = min(ins, min(del, mod));
15 }
```

### 5.4 Lcsubseq

```
1 // Longest Common Subsequence
2 string lcs(string x, string y){
3     int n = x.size(), m = y.size();
4     vector<vi> dp(n+1, vi(m+1, 0));
5
6     for(int i=0;i<=n;i++){
7         for(int j=0;j<=m;j++){
8             if(!i or !j)
9                 dp[i][j]=0;
10            else if(x[i-1] == y[j-1])
11                dp[i][j]=dp[i-1][j-1]+1;
12            else
13                dp[i][j]=max(dp[i-1][j], dp[i][j-1]);
14        }
15    }
16
17    // int len = dp[n][m];
18    string ans="";
19
20    // recover string
21    int i = n-1, j = m-1;
22    while(i>=0 and j>=0){
23        if(x[i] == y[j]){
24            ans.pb(x[i]);
25            i--; j--;
26        }else if(dp[i][j+1]>dp[i+1][j])
27            i--;
28        else
29            j--;
30    }
31
32    reverse(ans.begin(), ans.end());
33
34    return ans;
35 }
```

### 5.5 Kmp

```
1 string p;
2 int neighbor[N];
3 int walk(int u, char c) { // leader after inputting '
4     c'
```



```

4   while (u != -1 && (u+1 >= (int)p.size() || p[u + 24   fail[v] = to[ fail[u] ][i];
   1] != c)) // leader doesn't match 25   term[v] += term[ fail[v] ];
5   u = neighbor[u]; 26   }
6   return p[u + 1] == c ? u+1 : u; 27   }
7 } 28   else if(u != 1) to[u][i] = to[ fail[u] ][
8 void build() { 29   i];
9   neighbor[0] = -1; // -1 is the leftmost state 30   else to[u][i] = 1;
10  for (int i = 1; i < (int)p.size(); i++) 31   }
11      neighbor[i] = walk(neighbor[i-1], p[i]); 32 }
12 }

```

## 5.6 Hash

```

1 // String Hash template
2 // constructor(s) - O(|s|)
3 // query(l, r) - returns the hash of the range [l,r]
  from left to right - O(1)
4 // query_inv(l, r) from right to left - O(1)
5
6 struct Hash {
7     const ll P = 31;
8     int n; string s;
9     vector<ll> h, hi, p;
10    Hash() {}
11    Hash(string s): s(s), n(s.size()), h(n), hi(n), p
(n) {
12        for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1)
% MOD;
13        for (int i=0;i<n;i++)
14            h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
15        for (int i=n-1;i>=0;i--)
16            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P)
% MOD;
17    }
18    int query(int l, int r) {
19        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
0));
20        return hash < 0 ? hash + MOD : hash;
21    }
22    int query_inv(int l, int r) {
23        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
+1] % MOD : 0));
24        return hash < 0 ? hash + MOD : hash;
25    }
26 };

```

## 5.7 Aho Corasick

```

1 // https://github.com/joseleite19/icpc-notebook/blob/
  master/code/string/aho_corasick.cpp
2 const int A = 26;
3 int to[N][A];
4 int ne = 2, fail[N], term[N];
5 void add_string(string str, int id){
6     int p = 1;
7     for(auto c: str){
8         int ch = c - 'a'; // !
9         if(!to[p][ch]) to[p][ch] = ne++;
10        p = to[p][ch];
11    }
12    term[p]++;
13 }
14 void init(){
15     for(int i = 0; i < ne; i++) fail[i] = 1;
16     queue<int> q; q.push(1);
17     int u, v;
18     while(!q.empty()){
19         u = q.front(); q.pop();
20         for(int i = 0; i < A; i++){
21             if(to[u][i]){
22                 v = to[u][i]; q.push(v);
23                 if(u != 1){

```

## 5.8 Lcs

```

1 string LCSubStr(string X, string Y)
2 {
3     int m = X.size();
4     int n = Y.size();
5
6     int result = 0, end;
7     int len[2][n];
8     int currRow = 0;
9
10    for(int i=0;i<=m;i++){
11        for(int j=0;j<=n;j++){
12            if(i==0 || j==0)
13                len[currRow][j] = 0;
14            else if(X[i-1] == Y[j-1]){
15                len[currRow][j] = len[1-currRow][j-1]
+ 1;
16                if(len[currRow][j] > result){
17                    result = len[currRow][j];
18                    end = i - 1;
19                }
20            }
21            else
22                len[currRow][j] = 0;
23        }
24        currRow = 1 - currRow;
25    }
26
27    if(result==0)
28        return string();
29
30    return X.substr(end - result + 1, result);
31 }
32 }

```

## 6 Geometria

### 6.1 Polygon Diameter

```

1 pair<point, point> polygon_diameter(vp p) {
2     p = convex_hull(p);
3     int n = p.size(), j = n<2 ? 0:1;
4     pair<ll, vp> res({0, {p[0], p[0]}});
5     for (int i=0;i<j;i++){
6         for (; j = (j+1) % n) {
7             res = max(res, {norm2(p[i] - p[j]), {p[i]
], p[j]}});
8             if ((p[(j + 1) % n] - p[j]) ^ (p[i + 1] -
p[i]) >= 0)
9                 break;
10        }
11    }
12    return res.second;
13 }
14
15 double diameter(const vector<point> &p) {
16     vector<point> h = convexHull(p);
17     int m = h.size();
18     if (m == 1)

```

```

19     return 0;
20 if (m == 2)
21     return dist(h[0], h[1]);
22 int k = 1;
23 while (area(h[m - 1], h[0], h[(k + 1) % m]) >
24         area(h[m - 1], h[0], h[k]))
25     ++k;
26 double res = 0;
27 for (int i = 0, j = k; i <= k && j < m; i++) {
28     res = max(res, dist(h[i], h[j]));
29     while (j < m && area(h[i], h[(i + 1) % m], h
30         [(j + 1) % m]) > area(h[i], h[(i + 1) % m], h[j]))
31     {
32         res = max(res, dist(h[i], h[(j + 1) % m]));
33         ++j;
34     }
35 }
36 return res;
37 }

```

## 6.2 Mindistpair

```

1 ll MinDistPair(vp &vet){
2     int n = vet.size();
3     sort(vet.begin(), vet.end());
4     set<point> s;
5
6     ll best_dist = LLINF;
7     int j=0;
8     for(int i=0;i<n;i++){
9         ll d = ceil(sqrt(best_dist));
10        while(j<n and vet[i].x-vet[j].x >= d){
11            s.erase(point(vet[j].y, vet[j].x));
12            j++;
13        }
14
15        auto it1 = s.lower_bound({vet[i].y - d, vet[i]
16            ].x});
17        auto it2 = s.upper_bound({vet[i].y + d, vet[i]
18            ].x});
19
20        for(auto it=it1; it!=it2; it++){
21            ll dx = vet[i].x - it->x;
22            ll dy = vet[i].y - it->y;
23            if(best_dist > dx*dx + dy*dy){
24                best_dist = dx*dx + dy*dy;
25                // vet[i] e inv(it)
26            }
27        }
28        s.insert(point(vet[i].y, vet[i].x));
29    }
30    return best_dist;
31 }

```

## 6.3 Inside Polygon

```

1 // Convex O(logn)
2
3 bool insideT(point a, point b, point c, point e){
4     int x = ccw(a, b, e);
5     int y = ccw(b, c, e);
6     int z = ccw(c, a, e);
7     return !((x==1 or y==1 or z==1) and (x==-1 or y
8         ==-1 or z==-1));
9 }
10
11 bool inside(vp &p, point e){ // ccw
12     int l=2, r=(int)p.size()-1;
13     while(l<r){
14         int mid = (l+r)/2;

```

```

14         if(ccw(p[0], p[mid], e) == 1)
15             l=mid+1;
16         else{
17             r=mid;
18         }
19     }
20     // bordo
21     // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)
22     ==0) return false;
23     // if(r==2 and ccw(p[0], p[1], e)==0) return
24     false;
25     // if(ccw(p[r], p[r-1], e)==0) return false;
26     return insideT(p[0], p[r-1], p[r], e);
27
28 // Any O(n)
29
30 int inside(vp &p, point pp){
31     // 1 - inside / 0 - boundary / -1 - outside
32     int n = p.size();
33     for(int i=0;i<n;i++){
34         int j = (i+1)%n;
35         if(line({p[i], p[j]}).inside_seg(pp))
36             return 0;
37     }
38     int inter = 0;
39     for(int i=0;i<n;i++){
40         int j = (i+1)%n;
41         if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
42             [i], p[j], pp)==1)
43             inter++; // up
44         else if(p[j].x <= pp.x and pp.x < p[i].x and
45             ccw(p[i], p[j], pp)==-1)
46             inter++; // down
47     }
48     if(inter%2==0) return -1; // outside
49     else return 1; // inside
50 }

```

## 6.4 Polygon Cut Length

```

1 // Polygon Cut length
2 ld solve(vp &p, point a, point b){ // ccw
3     int n = p.size();
4     ld ans = 0;
5
6     for(int i=0;i<n;i++){
7         int j = (i+1) % n;
8
9         int signi = ccw(a, b, p[i]);
10        int signj = ccw(a, b, p[j]);
11
12        if(signi == 0 and signj == 0){
13            if((b-a) * (p[j]-p[i]) > 0){
14                ans += param(a, b, p[j]);
15                ans -= param(a, b, p[i]);
16            }
17        }else if(signi <= 0 and signj > 0){
18            ans -= param(a, b, inter_line({a, b}, {p[
19                i], p[j]}[0]));
20        }else if(signi > 0 and signj <= 0){
21            ans += param(a, b, inter_line({a, b}, {p[
22                i], p[j]}[0]));
23        }
24    }
25    return abs(ans * norm(b-a));
26 }

```

## 6.5 3d

```

1 // typedef ll cod;
2 // bool eq(cod a, cod b){ return (a==b); }
3
4 const ld EPS = 1e-6;
5 #define vp vector<point>
6 typedef ld cod;
7 bool eq(cod a, cod b){ return fabs(a - b) <= EPS; }
8
9 struct point
10 {
11     cod x, y, z;
12     point(cod x=0, cod y=0, cod z=0): x(x), y(y), z(z) {}
13
14     point operator+(const point &o) const {
15         return {x+o.x, y+o.y, z+o.z};
16     }
17     point operator-(const point &o) const {
18         return {x-o.x, y-o.y, z-o.z};
19     }
20     point operator*(cod t) const {
21         return {x*t, y*t, z*t};
22     }
23     point operator/(cod t) const {
24         return {x/t, y/t, z/t};
25     }
26     bool operator==(const point &o) const {
27         return eq(x, o.x) and eq(y, o.y) and eq(z, o.
28     z);
29     }
30     cod operator*(const point &o) const { // dot
31         return x*o.x + y*o.y + z*o.z;
32     }
33     point operator^(const point &o) const { // cross
34         return point(y*o.z - z*o.y,
35             z*o.x - x*o.z,
36             x*o.y - y*o.x);
37     }
38
39     ld norm(point a) { // Modulo
40         return sqrt(a * a);
41     }
42     cod norm2(point a) {
43         return a * a;
44     }
45     bool nulo(point a) {
46         return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0))
47     ;
48
49     ld proj(point a, point b) { // a sobre b
50         return (a*b)/norm(b);
51     }
52     ld angle(point a, point b) { // em radianos
53         return acos((a*b) / norm(a) / norm(b));
54     }
55     cod triple(point a, point b, point c) {
56         return (a * (b^c)); // Area do paralelepipedo
57     }
58
59     point normilize(point a) {
60         return a/norm(a);
61     }
62
63     struct plane {
64         cod a, b, c, d;
65         point p1, p2, p3;
66         plane(point p1=0, point p2=0, point p3=0): p1(p1)
67         , p2(p2), p3(p3) {
68             point aux = (p1-p3)^(p2-p3);
69             a = aux.x; b = aux.y; c = aux.z;
70             d = -a*p1.x - b*p1.y - c*p1.z;
71         }
72         plane(point p, point normal) {
73             normal = normilize(normal);
74             a = normal.x; b = normal.y; c = normal.z;
75             d = -(p*normal);
76         }
77         // ax+by+cz+d = 0;
78         cod eval(point &p) {
79             return a*p.x + b*p.y + c*p.z + d;
80         }
81
82         cod dist(plane pl, point p) {
83             return fabs(pl.a*p.x + pl.b*p.y + pl.c*p.z + pl.d
84             ) / sqrt(pl.a*pl.a + pl.b*pl.b + pl.c*pl.c);
85         }
86
87         point rotate(point v, point k, ld theta) {
88             // Rotaciona o vetor v theta graus em torno do
89             eixo k
90             // theta *= PI/180; // graus
91             return (
92                 v*cos(theta)) +
93                 ((k^v)*sin(theta)) +
94                 (k*(k^v))*(1-cos(theta)
95             );
96         }
97         // 3d line inter / mindistance
98         cod d(point p1, point p2, point p3, point p4) {
99             return (p2-p1) * (p4-p3);
100         }
101         vector<point> inter3d(point p1, point p2, point p3,
102             point p4) {
103             cod mua = ( d(p1, p3, p4, p3) * d(p4, p3, p2, p1)
104             - d(p1, p3, p2, p1) * d(p4, p3, p4, p3) )
105             / ( d(p2, p1, p2, p1) * d(p4, p3, p4, p3)
106             - d(p4, p3, p2, p1) * d(p4, p3, p2, p1) );
107             cod mub = ( d(p1, p3, p4, p3) + mua * d(p4, p3,
108             p2, p1) ) / d(p4, p3, p4, p3);
109             point pa = p1 + (p2-p1) * mua;
110             point pb = p3 + (p4-p3) * mub;
111             if (pa == pb) return {pa};
112             return {};
113         }
114     }
115 }

```

## 6.6 Convex Hull

```

1 vp convex_hull(vp P)
2 {
3     sort(P.begin(), P.end());
4     vp L, U;
5     for(auto p: P){
6         while(L.size()>=2 and ccw(L.end()[-2], L.back
7             (), p)!=1)
8             L.pop_back();
9         L.push_back(p);
10    }
11    reverse(P.begin(), P.end());
12    for(auto p: P){
13        while(U.size()>=2 and ccw(U.end()[-2], U.back
14            (), p)!=1)
15            U.pop_back();
16        U.push_back(p);
17    }
18    L.pop_back();
19    L.insert(L.end(), U.begin(), U.end()-1);
20    return L;
21 }

```

## 6.7 Linear Transformation

```

1 // Apply linear transformation (p -> q) to r.
2 point linear_transformation(point p0, point p1, point
   q0, point q1, point r) {
3     point dp = p1-p0, dq = q1-q0, num((dp~dq), (dp~dq
   ));
4     return q0 + point((r-p0)^(num), (r-p0)*(num))/(dp
   *dp);
5 }

```

## 6.8 Voronoi

```

1 bool polygonIntersection(line &seg, vp &p) {
2     long double l = -1e18, r = 1e18;
3     for(auto ps : p) {
4         long double z = seg.eval(ps);
5         l = max(l, z);
6         r = min(r, z);
7     }
8     return l - r > EPS;
9 }
10
11 int w, h;
12
13 line getBisector(point a, point b) {
14     line ans(a, b);
15     swap(ans.a, ans.b);
16     ans.b *= -1;
17     ans.c = ans.a * (a.x + b.x) * 0.5 + ans.b * (a.y
   + b.y) * 0.5;
18     return ans;
19 }
20
21 vp cutPolygon(vp poly, line seg) {
22     int n = (int) poly.size();
23     vp ans;
24     for(int i = 0; i < n; i++) {
25         double z = seg.eval(poly[i]);
26         if(z > -EPS) {
27             ans.push_back(poly[i]);
28         }
29         double z2 = seg.eval(poly[(i + 1) % n]);
30         if((z > EPS && z2 < -EPS) || (z < -EPS && z2
   > EPS)) {
31             ans.push_back(inter_line(seg, line(poly[i
   ], poly[(i + 1) % n]))[0]);
32         }
33     }
34     return ans;
35 }
36
37 // BE CAREFUL!
38 // the first point may be any point
39 // O(N^3)
40 vp getCell(vp pts, int i) {
41     vp ans;
42     ans.emplace_back(0, 0);
43     ans.emplace_back(1e6, 0);
44     ans.emplace_back(1e6, 1e6);
45     ans.emplace_back(0, 1e6);
46     for(int j = 0; j < (int) pts.size(); j++) {
47         if(j != i) {
48             ans = cutPolygon(ans, getBisector(pts[i],
   pts[j]));
49         }
50     }
51     return ans;
52 }
53
54 // O(N^2) expected time
55 vector<vp> getVoronoi(vp pts) {
56     // assert(pts.size() > 0);
57     int n = (int) pts.size();
58     vector<int> p(n, 0);

```

```

59     for(int i = 0; i < n; i++) {
60         p[i] = i;
61     }
62     shuffle(p.begin(), p.end(), rng);
63     vector<vp> ans(n);
64     ans[0].emplace_back(0, 0);
65     ans[0].emplace_back(w, 0);
66     ans[0].emplace_back(w, h);
67     ans[0].emplace_back(0, h);
68     for(int i = 1; i < n; i++) {
69         ans[i] = ans[0];
70     }
71     for(auto i : p) {
72         for(auto j : p) {
73             if(j == i) break;
74             auto bi = getBisector(pts[j], pts[i]);
75             if(!polygonIntersection(bi, ans[j]))
76                 continue;
77             ans[j] = cutPolygon(ans[j], getBisector(
   pts[j], pts[i]));
78             ans[i] = cutPolygon(ans[i], getBisector(
   pts[i], pts[j]));
79         }
80     }
81     return ans;

```

## 6.9 Intersect Polygon

```

1 bool intersect(vector<point> A, vector<point> B) //
   Ordered ccw
2 {
3     for(auto a: A)
4         if(inside(B, a))
5             return true;
6     for(auto b: B)
7         if(inside(A, b))
8             return true;
9
10    if(inside(B, center(A)))
11        return true;
12
13    return false;

```

## 6.10 Sort By Angle

```

1 // Comparator function for sorting points by angle
2
3 int ret[2][2] = {{3, 2},{4, 1}};
4 inline int quad(point p) {
5     return ret[p.x >= 0][p.y >= 0];
6 }
7
8 bool comp(point a, point b) { // ccw
9     int qa = quad(a), qb = quad(b);
10    return (qa == qb ? (a ^ b) > 0 : qa < qb);
11 }
12
13 // only vectors in range [x+0, x+180)
14 bool comp(point a, point b){
15     return (a ^ b) > 0; // ccw
16     // return (a ^ b) < 0; // cw
17 }

```

## 6.11 2d

```

1 #define vp vector<point>
2 #define ld long double
3 const ld EPS = 1e-6;
4 const ld PI = acos(-1);
5

```

```

6 typedef ld T;
7 bool eq(T a, T b){ return abs(a - b) <= EPS; }
8
9 struct point{
10     T x, y;
11     int id;
12     point(T x=0, T y=0): x(x), y(y){}
13
14     point operator+(const point &o) const{ return {x
15 + o.x, y + o.y}; }
16     point operator-(const point &o) const{ return {x
17 - o.x, y - o.y}; }
18     point operator*(T t) const{ return {x * t, y * t
19 }; }
20     point operator/(T t) const{ return {x / t, y / t
21 }; }
22     T operator*(const point &o) const{ return x * o.x
23 + y * o.y; }
24     T operator^(const point &o) const{ return x * o.y
25 - y * o.x; }
26     bool operator<(const point &o) const{
27         return (eq(x, o.x) ? y < o.y : x < o.x);
28     }
29     bool operator==(const point &o) const{
30         return eq(x, o.x) and eq(y, o.y);
31     }
32     friend ostream& operator<<(ostream& os, point p)
33     {
34         return os << "(" << p.x << "," << p.y << ")";
35     }
36 };
37
38 int ccw(point a, point b, point e){ // -1=dir; 0=
39 collinear; 1=esq;
40     T tmp = (b-a) ^ (e-a); // vector from a to b
41     return (tmp > EPS) - (tmp < -EPS);
42 }
43
44 ld norm(point a){ // Modulo
45     return sqrt(a * a);
46 }
47
48 T norm2(point a){
49     return a * a;
50 }
51
52 bool nulo(point a){
53     return (eq(a.x, 0) and eq(a.y, 0));
54 }
55
56 point rotccw(point p, ld a){
57     // a = PI*a/180; // graus
58     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
59 +p.x*sin(a)));
60 }
61
62 point rot90cw(point a) { return point(a.y, -a.x); };
63 point rot90ccw(point a) { return point(-a.y, a.x); };
64
65 ld proj(point a, point b){ // a sobre b
66     return a*b/norm(b);
67 }
68
69 ld angle(point a, point b){ // em radianos
70     ld ang = a*b / norm(a) / norm(b);
71     return acos(max(min(ang, (ld)1), (ld)-1));
72 }
73
74 ld angle_vec(point v){
75     // return 180/PI*atan2(v.x, v.y); // graus
76     return atan2(v.x, v.y);
77 }
78
79 ld order_angle(point a, point b){ // from a to b ccw
80 (a in front of b)
81     ld aux = angle(a,b)*180/PI;
82     return ((a^b)<=0 ? aux:360-aux);
83 }
84
85 bool angle_less(point a1, point b1, point a2, point
86 b2){ // ang(a1,b1) <= ang(a2,b2)
87
88     point p1((a1*b1), abs((a1^b1)));
89     point p2((a2*b2), abs((a2^b2)));
90     return (p1^p2) <= 0;
91 }
92
93 ld area(vp &p){ // (points sorted)
94     ld ret = 0;
95     for(int i=2;i<(int)p.size();i++)
96         ret += (p[i]-p[0])^(p[i-1]-p[0]);
97     return abs(ret/2);
98 }
99
100 ld areaT(point &a, point &b, point &c){
101     return abs((b-a)^(c-a))/2.0;
102 }
103
104 point center(vp &A){
105     point c = point();
106     int len = A.size();
107     for(int i=0;i<len;i++)
108         c=c+A[i];
109     return c/len;
110 }
111
112 point forca_mod(point p, ld m){
113     ld cm = norm(p);
114     if(cm<EPS) return point();
115     return point(p.x*m/cm,p.y*m/cm);
116 }
117
118 ld param(point a, point b, point v){
119     // v = t*(b-a) + a // return t;
120     // assert(line(a, b).inside_seg(v));
121     return ((v-a) * (b-a)) / ((b-a) * (b-a));
122 }
123
124 bool simetric(vp &a){ //ordered
125     int n = a.size();
126     point c = center(a);
127     if(n&1) return false;
128     for(int i=0;i<n/2;i++)
129         if(ccw(a[i], a[i+n/2], c) != 0)
130             return false;
131     return true;
132 }
133
134 point mirror(point m1, point m2, point p){
135     // mirror point p around segment m1m2
136     point seg = m2-m1;
137     ld t0 = ((p-m1)*seg) / (seg*seg);
138     point ort = m1 + seg*t0;
139     point pm = ort-(p-ort);
140     return pm;
141 }
142
143 ///////////////
144 // Line //
145 ///////////////
146
147 struct line{
148     point p1, p2;
149     T a, b, c; // ax+by+c = 0;
150     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
151     line(point p1=0, point p2=0): p1(p1), p2(p2){
152         a = p1.y - p2.y;
153         b = p2.x - p1.x;
154         c = p1 ^ p2;
155     }
156
157     T eval(point p){
158         return a*p.x+b*p.y+c;
159     }
160
161     bool inside(point p){

```

```

140     return eq(eval(p), 0);
141 }
142 point normal(){
143     return point(a, b);
144 }
145
146 bool inside_seg(point p){
147     return (
148         ((p1-p) ^ (p2-p)) == 0 and
149         ((p1-p) * (p2-p)) <= 0
150     );
151 }
152
153 };
154
155 // be careful with precision error
156 vp inter_line(line l1, line l2){
157     ld det = l1.a*l2.b - l1.b*l2.a;
158     if(det==0) return {};
159     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
160     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
161     return {point(x, y)};
162 }
163
164 // segments not collinear
165 vp inter_seg(line l1, line l2){
166     vp ans = inter_line(l1, l2);
167     if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.
inside_seg(ans[0]))
168         return {};
169     return ans;
170 }
171 bool seg_has_inter(line l1, line l2){
172     return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.
p2, l2.p2) < 0 and
173         ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.
p2, l1.p2) < 0;
174 }
175
176 ld dist_seg(point p, point a, point b){ // point -
seg
177     if((p-a)*(b-a) < EPS) return norm(p-a);
178     if((p-b)*(a-b) < EPS) return norm(p-b);
179     return abs((p-a)^(b-a)) / norm(b-a);
180 }
181
182 ld dist_line(point p, line l){ // point - line
183     return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
184 }
185
186 line bisector(point a, point b){
187     point d = (b-a)*2;
188     return line(d.x, d.y, a*a - b*b);
189 }
190
191 line perpendicular(line l, point p){ // passes
through p
192     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
193 }
194
195
196 ///////////////
197 // Circle //
198 ///////////////
199
200 struct circle{
201     point c; T r;
202     circle() : c(0, 0), r(0){}
203     circle(const point o) : c(o), r(0){}
204     circle(const point a, const point b){
205         c = (a+b)/2;
206         r = norm(a-c);
207     }
208     circle(const point a, const point b, const point
cc){
209         assert(ccw(a, b, cc) != 0);
210         c = inter_line(bisector(a, b), bisector(b, cc
))[0];
211         r = norm(a-c);
212     }
213     bool inside(const point &a) const{
214         return norm(a - c) <= r + EPS;
215     }
216 };
217
218 pair<point, point> tangent_points(circle cr, point p)
{
219     ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
220     point p1 = rotccw(cr.c-p, -theta);
221     point p2 = rotccw(cr.c-p, theta);
222     assert(d1 >= cr.r);
223     p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
224     p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
225     return {p1, p2};
226 }
227
228
229 circle incircle(point p1, point p2, point p3){
230     ld m1 = norm(p2-p3);
231     ld m2 = norm(p1-p3);
232     ld m3 = norm(p1-p2);
233     point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
234     ld s = 0.5*(m1+m2+m3);
235     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
236     return circle(c, r);
237 }
238
239 circle circumcircle(point a, point b, point c) {
240     circle ans;
241     point u = point((b-a).y, -(b-a).x);
242     point v = point((c-a).y, -(c-a).x);
243     point n = (c-b)*0.5;
244     ld t = (u^n)/(v^u);
245     ans.c = ((a+c)*0.5) + (v*t);
246     ans.r = norm(ans.c-a);
247     return ans;
248 }
249
250 vp inter_circle_line(circle C, line L){
251     point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.
p1)*(ab) / (ab*ab));
252     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
/ (ab*ab);
253     if (h2 < -EPS) return {};
254     if (eq(h2, 0)) return {p};
255     point h = (ab/norm(ab)) * sqrt(h2);
256     return {p - h, p + h};
257 }
258
259 vp inter_circle(circle c1, circle c2){
260     if (c1.c == c2.c) { assert(c1.r != c2.r); return
{}; }
261     point vec = c2.c - c1.c;
262     ld d2 = vec * vec, sum = c1.r + c2.r, dif = c1.r
- c2.r;
263     ld p = (d2 + c1.r * c1.r - c2.r * c2.r) / (2 * d2
);
264     ld h2 = c1.r * c1.r - p * p * d2;
265     if (sum * sum < d2 or dif * dif > d2) return {};
266     point mid = c1.c + vec * p, per = point(-vec.y,
vec.x) * sqrt(fmax(0, h2) / d2);
267     if (eq(per.x, 0) and eq(per.y, 0)) return {mid};
268     return {mid + per, mid - per};
269 }
270
271 // minimum circle cover O(n) amortizado

```

```

272 circle min_circle_cover(vp v){
273     random_shuffle(v.begin(), v.end());
274     circle ans;
275     int n = v.size();
276     for(int i=0;i<n;i++){ if(!ans.inside(v[i])){
277         ans = circle(v[i]);
278         for(int j=0;j<i;j++) if(!ans.inside(v[j])){
279             ans = circle(v[i], v[j]);
280             for(int k=0;k<j;k++) if(!ans.inside(v[k])){
281                 ans = circle(v[i], v[j], v[k]);
282             }
283         }
284     }
285     return ans;
286 }

```

## 7 Grafos

### 7.1 Dfs Tree

```

1 int desce[N], sobe[N], vis[N], h[N];
2 int backedges[N], pai[N];
3
4 // backedges[u] = backedges que comecam embaixo de (
   ou =) u e sobem pra cima de u; backedges[u] == 0
   => u eh ponte
5 void dfs(int u, int p) {
6     if(vis[u]) return;
7     pai[u] = p;
8     h[u] = h[p]+1;
9     vis[u] = 1;
10
11     for(auto v : g[u]) {
12         if(p == v or vis[v]) continue;
13         dfs(v, u);
14         backedges[u] += backedges[v];
15     }
16     for(auto v : g[u]) {
17         if(h[v] > h[u]+1)
18             desce[u]++;
19         else if(h[v] < h[u]-1)
20             sobe[u]++;
21     }
22     backedges[u] += sobe[u] - desce[u];
23 }

```

### 7.2 Kosaraju

```

1 vector<int> g[N], gi[N]; // grafo invertido
2 int vis[N], comp[N]; // componente conexo de cada
   vertice
3 stack<int> S;
4
5 void dfs(int u){
6     vis[u] = 1;
7     for(auto v: g[u]) if(!vis[v]) dfs(v);
8     S.push(u);
9 }
10
11 void scc(int u, int c){
12     vis[u] = 1; comp[u] = c;
13     for(auto v: gi[u]) if(!vis[v]) scc(v, c);
14 }
15
16 void kosaraju(int n){
17     for(int i=0;i<n;i++) vis[i] = 0;
18     for(int i=0;i<n;i++) if(!vis[i]) dfs(i);
19     for(int i=0;i<n;i++) vis[i] = 0;
20     while(S.size()){
21         int u = S.top();

```

```

22         S.pop();
23         if(!vis[u]) scc(u, u);
24     }
25 }

```

### 7.3 Topological Sort

```

1 int n; // number of vertices
2 vector<vector<int>> adj; // adjacency list of graph
3 vector<bool> visited;
4 vector<int> ans;
5
6 void dfs(int v) {
7     visited[v] = true;
8     for (int u : adj[v]) {
9         if (!visited[u])
10             dfs(u);
11     }
12     ans.push_back(v);
13 }
14
15 void topological_sort() {
16     visited.assign(n, false);
17     ans.clear();
18     for (int i = 0; i < n; ++i) {
19         if (!visited[i]) {
20             dfs(i);
21         }
22     }
23     reverse(ans.begin(), ans.end());
24 }

```

### 7.4 Dijkstra

```

1 #define pii pair<int, int>
2 vector<vector<pii>> g(N);
3 vector<bool> used(N);
4 vector<ll> d(N, LLINF);
5 priority_queue< pii, vector<pii>, greater<pii> > fila
   ;
6
7 void dijkstra(int k) {
8     d[k] = 0;
9     fila.push({0, k});
10
11     while (!fila.empty()) {
12         auto [w, u] = fila.top();
13         fila.pop();
14         if (used[u]) continue;
15         used[u] = true;
16
17         for (auto [v, w]: g[u]) {
18             if (d[v] > d[u] + w) {
19                 d[v] = d[u] + w;
20                 fila.push({d[v], v});
21             }
22         }
23     }
24 }

```

### 7.5 Dinic

```

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

```



```

8         T c, f;
9         Edge(int from, int to, T c, T f): from(from),
10         to(to), c(c), f(f) {}
11     };
12     vector<Edge> edges;
13     int tempo = 0, id = 0;
14     int lvl[N], vis[N], qu[N], nxt[N];
15     vector<int> adj[N];
16     T INF = (1ll)1e14;
17     #warning botar INF certo no dinic
18
19     void addEdge(int a, int b, T c, T rc=0) {
20         edges.pb({a, b, c, 0});
21         adj[a].pb(id++);
22         edges.pb({b, a, rc, 0});
23         adj[b].pb(id++);
24     }
25
26     bool bfs(int s, int t) {
27         tempo++;
28         vis[s] = tempo;
29         int qt = 0;
30         qu[qt++] = s;
31         lvl[s] = 0;
32
33         for(int i = 0; i < qt; i++) {
34             int u = qu[i];
35             nxt[u] = 0;
36
37             for(auto idx : adj[u]) {
38                 auto& e = edges[idx];
39                 if(e.f >= e.c or vis[e.to] == tempo)
40                     continue;
41                 // from[e.to] = idx; pra usar a outra bfs
42                 vis[e.to] = tempo;
43                 lvl[e.to] = lvl[u]+1;
44                 qu[qt++] = e.to;
45             }
46             return (vis[t] == tempo);
47         }
48
49     T dfs(int s, int t, T f) {
50         if(s == t) return f;
51
52         T res = 0;
53         for(; nxt[s] < (int)adj[s].size(); nxt[s]++)
54             {
55                 int idx = adj[s][nxt[s]];
56                 auto& e = edges[idx];
57                 auto& rev = edges[idx^1];
58
59                 if(e.f >= e.c or lvl[e.to] != lvl[s]+1)
60                     continue;
61                 T flow = dfs(e.to, t, min(f, e.c-e.f));
62                 res += flow;
63                 e.f += flow;
64                 rev.f -= flow;
65                 f -= flow;
66                 if(!f) break;
67             }
68         return res;
69     }
70
71     // dfs boa para grafos pequenos (n <= 500?), ruim
72     // para fluxos grandes?
73     // tem que criar o vetor from pra usar e marcar o
74     // from na bfs
75     T dfs(int s, int t) {
76         T res = INF;
77         for(int u = t; u != s; u = edges[from[u]].
78             from) {
79             res = min(res, edges[from[u]].c-edges[
80                 from[u]].f);
81         }
82         for(int u = t; u != s; u = edges[from[u]].
83             from) {
84             edges[from[u]].f += res;
85             edges[from[u]^1].f -= res;
86         }
87         return res;
88     }
89
90     T flow(int s, int t) {
91         T flow = 0;
92         while(bfs(s, t)) {
93             flow += dfs(s, t, INF);
94         }
95         return flow;
96     }
97
98     // NAO TESTADO DAQUI PRA BAIXO, MAS DEVE
99     // FUNCIONAR
100     void reset_flow() {
101         for(int i = 0; i < id; i++) // aqui eh id
102             mesmo ne?
103             edges[i].flow = 0;
104         memset(lvl, 0, sizeof(lvl));
105         memset(vis, 0, sizeof(vis));
106         memset(qu, 0, sizeof(qu));
107         memset(nxt, 0, sizeof(nxt));
108         tempo = 0;
109     }
110
111     vector<pair<int, int>> cut() {
112         vector<pair<int, int>> cuts;
113         for (auto [from, to, flow, cap]: edges) {
114             if (flow == cap and vis[from] == tempo
115                 and vis[to] < tempo and cap>0) {
116                 cuts.pb({from, to});
117             }
118         }
119         return cuts;
120     }
121 }
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

## 7.6 Centroid Decomp

```

1 vector<int> g[N];
2 int sz[N], rem[N];
3
4 void dfs(vector<int>& path, int u, int d=0, int p=-1)
5 {
6     path.push_back(d);
7     for (int v : g[u]) if (v != p and !rem[v]) dfs(
8         path, v, d+1, u);
9 }
10
11 int dfs_sz(int u, int p=-1) {
12     sz[u] = 1;
13     for (int v : g[u]) if (v != p and !rem[v]) sz[u]
14         += dfs_sz(v, u);
15     return sz[u];
16 }
17
18 int centroid(int u, int p, int size) {
19     for (int v : g[u]) if (v != p and !rem[v] and 2 *
20         sz[v] > size)
21         return centroid(v, u, size);
22     return u;
23 }

```



```

20
21 ll decomp(int u, int k) {
22     int c = centroid(u, u, dfs_sz(u));
23     rem[c] = true;
24
25     ll ans = 0;
26     vector<int> cnt(sz[u]);
27     cnt[0] = 1;
28     for (int v : g[c]) if (!rem[v]) {
29         vector<int> path;
30         dfs(path, v);
31         // d1 + d2 + 1 == k
32         for (int d : path) if (0 <= k-d-1 and k-d-1 <
sz[u])
33             ans += cnt[k-d-1];
34         for (int d : path) cnt[d+1]++;
35     }
36
37     for (int v : g[c]) if (!rem[v]) ans += decomp(v,
k);
38     return ans;
39 }

```

## 7.7 Hungarian

```

1 // Hungaro
2 //
3 // Resolve o problema de assignment (matriz n x n)
4 // Colocar os valores da matriz em 'a' (pode < 0)
5 // assignment() retorna um par com o valor do
6 // assignment minimo, e a coluna escolhida por cada
   linha
7 //
8 // O(n^3)
9
10 template<typename T> struct hungarian {
11     int n;
12     vector<vector<T>> a;
13     vector<T> u, v;
14     vector<int> p, way;
15     T inf;
16
17     hungarian(int n_) : n(n_), u(n+1), v(n+1), p(n+1)
, way(n+1) {
18         a = vector<vector<T>>(n, vector<T>(n));
19         inf = numeric_limits<T>::max();
20     }
21     pair<T, vector<int>> assignment() {
22         for (int i = 1; i <= n; i++) {
23             p[0] = i;
24             int j0 = 0;
25             vector<T> minv(n+1, inf);
26             vector<int> used(n+1, 0);
27             do {
28                 used[j0] = true;
29                 int i0 = p[j0], j1 = -1;
30                 T delta = inf;
31                 for (int j = 1; j <= n; j++) if (!
used[j]) {
32                     T cur = a[i0-1][j-1] - u[i0] - v[
j];
33                     if (cur < minv[j]) minv[j] = cur,
34                     if (minv[j] < delta) delta = minv
[j], j1 = j;
35                 }
36                 for (int j = 0; j <= n; j++)
37                     if (used[j]) u[p[j]] += delta, v[
j] -= delta;
38                 else minv[j] -= delta;
39                 j0 = j1;
40             } while (p[j0] != 0);
41             do {

```

```

42                 int j1 = way[j0];
43                 p[j0] = p[j1];
44                 j0 = j1;
45             } while (j0);
46         }
47         vector<int> ans(n);
48         for (int j = 1; j <= n; j++) ans[p[j]-1] = j
-1;
49         return make_pair(-v[0], ans);
50     }
51 };

```

## 7.8 Floyd Warshall

```

1 // Floyd Warshall
2
3 int dist[N][N];
4
5 for(int k = 1; k <= n; k++)
6     for(int i = 1; i <= n; i++)
7         for(int j = 1; j <= n; j++)
8             dist[i][j] = min(dist[i][j], dist[i][k] +
dist[k][j]);

```

## 7.9 2sat

```

1 #define rep(i,l,r) for (int i = (l); i < (r); i++)
2 struct TwoSat { // copied from kth-competitive-
programming/kactl
3     int N;
4     vector<vi> gr;
5     vi values; // 0 = false, 1 = true
6     TwoSat(int n = 0) : N(n), gr(2*n) {}
7     int addVar() { // (optional)
8         gr.emplace_back();
9         gr.emplace_back();
10        return N++;
11    }
12    void either(int f, int j) {
13        f = max(2*f, -1-2*f);
14        j = max(2*j, -1-2*j);
15        gr[f].push_back(j^1);
16        gr[j].push_back(f^1);
17    }
18    void atMostOne(const vi& li) { // (optional)
19        if ((int)li.size() <= 1) return;
20        int cur = ~li[0];
21        rep(i,2,(int)li.size()) {
22            int next = addVar();
23            either(cur, ~li[i]);
24            either(cur, next);
25            either(~li[i], next);
26            cur = ~next;
27        }
28        either(cur, ~li[1]);
29    }
30    vi _val, comp, z; int time = 0;
31    int dfs(int i) {
32        int low = _val[i] = ++time, x; z.push_back(i)
;
33        for(int e : gr[i]) if (!comp[e])
34            low = min(low, _val[e] ? dfs(e));
35        if (low == _val[i]) do {
36            x = z.back(); z.pop_back();
37            comp[x] = low;
38            if (values[x]>>1 == -1)
39                values[x]>>1 = x&1;
40        } while (x != i);
41        return _val[i] = low;
42    }
43    bool solve() {
44        values.assign(N, -1);

```

```

45     _val.assign(2*N, 0); comp = _val;
46     rep(i,0,2*N) if (!comp[i]) dfs(i);
47     rep(i,0,N) if (comp[2*i] == comp[2*i+1])
48         return 0;
49     return 1;
50 };

```

## 7.10 Lca

```

1  const int LOG = 22;
2  vector<vector<int>> g(N);
3  int t, n;
4  vector<int> in(N), height(N);
5  vector<vector<int>> up(LOG, vector<int>(N));
6  void dfs(int u, int h=0, int p=-1) {
7      up[0][u] = p;
8      in[u] = t++;
9      height[u] = h;
10     for (auto v: g[u]) if (v != p) dfs(v, h+1, u);
11 }
12
13 void blift() {
14     up[0][0] = 0;
15     for (int j=1; j<LOG; j++) {
16         for (int i=0; i<n; i++) {
17             up[j][i] = up[j-1][up[j-1][i]];
18         }
19     }
20 }
21
22 int lca(int u, int v) {
23     if (u == v) return u;
24     if (in[u] < in[v]) swap(u, v);
25     for (int i=LOG-1; i>=0; i--) {
26         int u2 = up[i][u];
27         if (in[u2] > in[v])
28             u = u2;
29     }
30     return up[0][u];
31 }
32
33 t = 0;
34 dfs(0);
35 blift();
36
37 // lca 0(1)
38
39 template<typename T> struct rmq {
40     vector<T> v;
41     int n; static const int b = 30;
42     vector<int> mask, t;
43
44     int op(int x, int y) { return v[x] < v[y] ? x : y; }
45     int msb(int x) { return __builtin_clz(1)-
46         __builtin_clz(x); }
47     rmq() {}
48     rmq(const vector<T>& v_) : v(v_), n(v.size()),
49         mask(n), t(n) {
50         for (int i = 0, at = 0; i < n; mask[i++] = at
51             |= 1) {
52             at = (at<<1)&((1<<b)-1);
53             while (at and op(i, i-msb(at&-at)) == i)
54                 at ^= at&-at;
55             for (int i = 0; i < n/b; i++) t[i] = b*i+b-1-
56                 msb(mask[b*i+b-1]);
57             for (int j = 1; (1<<j) <= n/b; j++) for (int
58                 i = 0; i+(1<<j) <= n/b; i++)
59                 t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j
60                 -1)+i+(1<<(j-1))]);
61         }
62     }
63 };

```

```

56 int small(int r, int sz = b) { return r-msb(mask[
57     r]&((1<<sz)-1)); }
58 T query(int l, int r) {
59     if (r-l+1 <= b) return small(r, r-l+1);
60     int ans = op(small(l+b-1), small(r));
61     int x = l/b+1, y = r/b-1;
62     if (x <= y) {
63         int j = msb(y-x+1);
64         ans = op(ans, op(t[n/b*j+x], t[n/b*j+y
65             -(1<<j)+1]));
66     }
67     return ans;
68 }
69
70 namespace lca {
71     vector<int> g[N];
72     int v[2*N], pos[N], dep[2*N];
73     int t;
74     rmq<int> RMQ;
75
76     void dfs(int i, int d = 0, int p = -1) {
77         v[t] = i, pos[i] = t, dep[t++] = d;
78         for (int j: g[i]) if (j != p) {
79             dfs(j, d+1, i);
80             v[t] = i, dep[t++] = d;
81         }
82     }
83     void build(int n, int root) {
84         t = 0;
85         dfs(root);
86         RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
87     }
88     int lca(int a, int b) {
89         a = pos[a], b = pos[b];
90         return v[RMQ.query(min(a, b), max(a, b))];
91     }
92     int dist(int a, int b) {
93         return dep[pos[a]] + dep[pos[b]] - 2*dep[pos[
94             lca(a, b)]];
95     }
96 }

```

## 7.11 Kruskal

```

1  struct DSU {
2      int n;
3      vector<int> parent, size;
4
5      DSU(int n): n(n) {
6          parent.resize(n, 0);
7          size.assign(n, 1);
8
9          for(int i=0; i<n; i++)
10             parent[i] = i;
11     }
12
13     int find(int a) {
14         if(a == parent[a]) return a;
15         return parent[a] = find(parent[a]);
16     }
17
18     void join(int a, int b) {
19         a = find(a); b = find(b);
20         if(a != b) {
21             if(size[a] < size[b]) swap(a, b);
22             parent[b] = a;
23             size[a] += size[b];
24         }
25     }
26
27     struct Edge {
28

```

```

29     int u, v, weight;
30     bool operator<(Edge const& other) {
31         return weight < other.weight;
32     }
33 };
34
35 vector<Edge> kruskal(int n, vector<Edge> edges) {
36     vector<Edge> mst;
37     DSU dsu = DSU(n+1);
38
39     sort(edges.begin(), edges.end());
40
41     for(Edge e : edges) {
42         if(dsu.find(e.u) != dsu.find(e.v)) {
43             mst.push_back(e);
44             dsu.join(e.u, e.v);
45         }
46     }
47     return mst;
48 }
49 }

```

## 7.12 Mcmf

```

1 // Time: O(F E log(V)) where F is max flow. (
  reference needed)
2 const int N = 502;
3 template<typename T> struct MCMF {
4     struct Edge {
5         int from, to;
6         T c, f, cost;
7         Edge(int from, int to, T c, T cost): from(
from), to(to), c(c), cost(cost) {}
8     };
9
10    vector<Edge> edges;
11    int tempo = 0, id = 0;
12    int nxt[N], vis[N];
13    vector<int> adj[N];
14    T lvl[N];
15    const T INF = 1e15;
16
17    void addEdge(int a, int b, int c, int cost) {
18        edges.pb({a, b, c, cost});
19        adj[a].pb(id++);
20        edges.pb({b, a, 0, -cost});
21        adj[b].pb(id++);
22    }
23
24    pair<T, T> dfs(int s, int t, T f) {
25        if(s == t or f == 0) return {f, 0};
26
27        pair<T, T> res = {0, 0};
28        for(; nxt[s] < (int)adj[s].size(); nxt[s]++)
29        {
30            int idx = adj[s][nxt[s]];
31            auto& e = edges[idx];
32            auto& rev = edges[idx^1];
33
34            if(e.f >= e.c or lvl[e.to] != lvl[s]+e
cost) continue;
35            auto [flow, cost] = dfs(e.to, t, min(f, e
.c-e.f));
36
37            if(!flow) continue;
38
39            res.ff += flow;
40            res.ss += cost + flow*e.cost;
41            e.f += flow;
42            rev.f -= flow;
43            f -= flow;
44
45            if(!f) break;

```

```

45        }
46        return res;
47    }
48
49    // funciona v
50    // pair<T, T> dfs(int s, int t) {
51    //     pair<T, T> res = {INF, 0};
52
53    //     for(int u = t; u != s; u = edges[from[u]].
from) {
54    //         res.ff = min(res.ff, edges[from[u]].c)
55    //     }
56
57    //     for(int u = t; u != s; u = edges[from[u]].
from) {
58    //         edges[from[u]].c -= res.ff;
59    //         edges[from[u]^1].c += res.ff;
60    //         res.ss += edges[from[u]].cost * res.ff
61    //     }
62
63    //     return res;
64    // }
65
66    bool spfa(int s, int t) {
67        for(int i = 0; i < N; i++) {
68            lvl[i] = INF;
69            vis[i] = 0;
70        }
71        lvl[s] = 0;
72        vis[s] = 1;
73        queue<int> q; q.push(s);
74
75        while(q.size()) {
76            int u = q.front(); q.pop();
77            vis[u] = 0;
78            nxt[u] = 0;
79
80            for(auto idx : adj[u]) {
81                auto& e = edges[idx];
82
83                if(e.f >= e.c) continue;
84                if(lvl[e.to] > lvl[u]+e.cost) {
85                    lvl[e.to] = lvl[u]+e.cost;
86                    if(!vis[e.to]) {
87                        q.push(e.to);
88                        vis[e.to] = 1;
89                    }
90                }
91            }
92        }
93
94        return (lvl[t] < INF);
95    }
96
97    pair<T, T> flow(int s, int t) {
98        pair<T, T> res = {0, 0};
99        while(spfa(s, t)) {
100            auto [flow, cost] = dfs(s, t, INF);
101            res.ff += flow;
102            res.ss += cost;
103        }
104        return res;
105    }
106 };

```

## 7.13 Ford

```

1 const int N = 2000010;
2
3 struct Ford {
4     struct Edge {

```

```

5     int to, f, c;
6 };
7
8 int vis[N];
9 vector<int> adj[N];
10 vector<Edge> edges;
11 int cur = 0;
12
13 void addEdge(int a, int b, int cap, int rcap) {
14     Edge e;
15     e.to = b; e.c = cap; e.f = 0;
16     edges.pb(e);
17     adj[a].pb(cur++);
18
19     e = Edge();
20     e.to = a; e.c = rcap; e.f = 0;
21     edges.pb(e);
22     adj[b].pb(cur++);
23 }
24
25 int dfs(int s, int t, int f, int tempo) {
26     if(s == t)
27         return f;
28     vis[s] = tempo;
29
30     for(int e : adj[s]) {
31         if(vis[edges[e].to] < tempo and (edges[e]
32 ].c - edges[e].f) > 0) {
33             if(int a = dfs(edges[e].to, t, min(f,
34 edges[e].c-edges[e].f), tempo)) {
35                 edges[e].f += a;
36                 edges[e^1].f -= a;
37                 return a;
38             }
39         }
40     }
41     return 0;
42 }
43
44 int flow(int s, int t) {
45     int mflow = 0, tempo = 1;
46     while(int a = dfs(s, t, INF, tempo)) {
47         mflow += a;
48         tempo++;
49     }
50     return mflow;
51 }

```

## 8 Algoritmos

### 8.1 Ternary Search

```

1 // Ternary
2 ld l = -1e4, r = 1e4;
3 int iter = 100;
4 while(iter--){
5     ld m1 = (2*l + r) / 3;
6     ld m2 = (l + 2*r) / 3;
7     if(check(m1) > check(m2))
8         l = m1;
9     else
10         r = m2;
11 }

```

## 9 Math

### 9.1 Totient

```

1 // phi(p^k) = (p^(k-1))*(p-1) com p primo

```

```

2 // O(sqrt(m))
3 ll phi(ll m){
4     ll res = m;
5     for(ll d=2;d*d<=m;d++){
6         if(m % d == 0){
7             res = (res/d)*(d-1);
8             while(m%d == 0)
9                 m /= d;
10        }
11    }
12    if(m > 1) {
13        res /= m;
14        res *= (m-1);
15    }
16    return res;
17 }
18
19 // modificacao do crivo, O(n*log(log(n)))
20 vector<ll> phi_to_n(ll n){
21     vector<bool> isprime(n+1, true);
22     vector<ll> tot(n+1);
23     tot[0] = 0; tot[1] = 1;
24     for(ll i=1;i<=n; i++){
25         tot[i] = i;
26     }
27
28     for(ll p=2;p<=n;p++){
29         if(isprime[p]){
30             tot[p] = p-1;
31             for(ll i=p+1;i<=n;i+=p){
32                 isprime[i] = false;
33                 tot[i] = (tot[i]/p)*(p-1);
34             }
35         }
36     }
37     return tot;
38 }

```

### 9.2 Pollard Rho

```

1 ll mul(ll a, ll b, ll m) {
2     ll ret = a*b - (ll)((ld)1/m*a*b+0.5)*m;
3     return ret < 0 ? ret+m : ret;
4 }
5
6 ll pow(ll a, ll b, ll m) {
7     ll ans = 1;
8     for (; b > 0; b /= 2ll, a = mul(a, a, m)) {
9         if (b % 2ll == 1)
10             ans = mul(ans, a, m);
11     }
12     return ans;
13 }
14
15 bool prime(ll n) {
16     if (n < 2) return 0;
17     if (n <= 3) return 1;
18     if (n % 2 == 0) return 0;
19
20     ll r = __builtin_ctzll(n - 1), d = n >> r;
21     for (int a : {2, 325, 9375, 28178, 450775,
22 9780504, 795265022}) {
23         ll x = pow(a, d, n);
24         if (x == 1 or x == n - 1 or a % n == 0)
25             continue;
26
27         for (int j = 0; j < r - 1; j++) {
28             x = mul(x, x, n);
29             if (x == n - 1) break;
30         }
31         if (x != n - 1) return 0;
32     }
33     return 1;
34 }

```

```

32 }
33
34 ll rho(ll n) {
35     if (n == 1 or prime(n)) return n;
36     auto f = [n](ll x) {return mul(x, x, n) + 1;};
37
38     ll x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
39     while (t % 40 != 0 or gcd(prd, n) == 1) {
40         if (x==y) x = ++x0, y = f(x);
41         q = mul(prd, abs(x-y), n);
42         if (q != 0) prd = q;
43         x = f(x), y = f(f(y)), t++;
44     }
45     return gcd(prd, n);
46 }
47
48 vector<ll> fact(ll n) { // retorna fatoracao em
    primos
49     if (n == 1) return {};
50     if (prime(n)) return {n};
51     ll d = rho(n);
52     vector<ll> l = fact(d), r = fact(n / d);
53     l.insert(l.end(), r.begin(), r.end());
54     return l;
55 }

```

### 9.3 Inverso Mult

```

1 // gcd(a, m) = 1 para existir solucao
2 // ax + my = 1, ou a*x = 1 (mod m)
3 ll inv(ll a, ll m) { // com gcd
4     ll x, y;
5     gcd(a, m, x, y);
6     return ((x % m) + m) % m;
7 }
8
9 ll inv(ll a, ll phim) { // com phi(m), se m for primo
10     entao phi(m) = p-1
11     ll e = phim-1;
12     return fexp(a, e);
13 }

```

### 9.4 Miller Habin

```

1 ll mul(ll a, ll b, ll m) {
2     return (a*b-ll(a*(long double)b/m+0.5)*m+m)%m;
3 }
4
5 ll expo(ll a, ll b, ll m) {
6     if (!b) return 1;
7     ll ans = expo(mul(a, a, m), b/2, m);
8     return b%2 ? mul(a, ans, m) : ans;
9 }
10
11 bool prime(ll n) {
12     if (n < 2) return 0;
13     if (n <= 3) return 1;
14     if (n % 2 == 0) return 0;
15
16     ll d = n - 1;
17     int r = 0;
18     while (d % 2 == 0) {
19         r++;
20         d /= 2;
21     }
22
23     // com esses primos, o teste funciona garantido
24     // para n <= 2^64
25     // funciona para n <= 3*10^24 com os primos ate
26     41
27     for (int i : {2, 325, 9375, 28178, 450775,
28         9780504, 795265022}) {

```

```

26         if (i >= n) break;
27         ll x = expo(i, d, n);
28         if (x == 1 or x == n - 1) continue;
29
30         bool composto = 1;
31         for (int j = 0; j < r - 1; j++) {
32             x = mul(x, x, n);
33             if (x == n - 1) {
34                 composto = 0;
35                 break;
36             }
37         }
38         if (composto) return 0;
39     }
40     return 1;
41 }

```

## 9.5 Matrix Exponentiation

```

1 struct Matrix {
2     vector<vl> m;
3     int r, c;
4
5     Matrix(vector<vl> mat) {
6         m = mat;
7         r = mat.size();
8         c = mat[0].size();
9     }
10
11     Matrix(int row, int col, bool ident=false) {
12         r = row; c = col;
13         m = vector<vl>(r, vl(c, 0));
14         if (ident) {
15             for (int i = 0; i < min(r, c); i++) {
16                 m[i][i] = 1;
17             }
18         }
19     }
20
21     Matrix operator*(const Matrix &o) const {
22         assert(c == o.r); // garantir que da pra
23         multiplicar
24         vector<vl> res(r, vl(o.c, 0));
25
26         for (int i = 0; i < r; i++) {
27             for (int k = 0; k < c; k++) {
28                 for (int j = 0; j < o.c; j++) {
29                     res[i][j] = (res[i][j] + m[i][k]*
30                         o.m[k][j]) % MOD;
31                 }
32             }
33         }
34         return Matrix(res);
35     };
36
37     Matrix fexp(Matrix b, int e, int n) {
38         if (e == 0) return Matrix(n, n, true); //
39         identidade
40         Matrix res = fexp(b, e/2, n);
41         res = (res * res);
42         if (e%2) res = (res * b);
43         return res;
44     }

```

## 9.6 Division Trick

```

1 for (int l = 1, r; l <= n; l = r + 1) {
2     r = n / (n / l);
3     // n / i has the same value for l <= i <= r

```

```
4 }
```

## 9.7 Crivo

```
1 vi p(N, 0);
2 p[0] = p[1] = 1;
3 for(11 i=4; i<N; i+=2) p[i] = 2;
4 for(11 i=3; i<N; i+=2)
5     if(!p[i])
6         for(11 j=i*i; j<N; j+=2*i)
7             p[j] = i;
```

## 9.8 Bigmod

```
1 11 mod(string a, 11 p) {
2     11 res = 0, b = 1;
3     reverse(all(a));
4
5     for(auto c : a) {
6         11 tmp = (((11)c-'0')*b) % p;
7         res = (res + tmp) % p;
8
9         b = (b * 10) % p;
10    }
```

# 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.  $\|a \times b\| = \|a\| \|b\| \sin \theta$ .

```
11
12     return res;
13 }
```

## 9.9 Linear Diophantine Equation

```
1 // Linear Diophantine Equation
2 array<11, 3> exgcd(int a, int b) {
3     if (a == 0) return {0, 1, b};
4     auto [x, y, g] = exgcd(b % a, a);
5     return {y - b / a * x, x, g};
6 }
7
8 array<11, 4> find_any_solution(11 a, 11 b, 11 c) {
9     auto [x, y, g] = exgcd(a, b);
10    if (c % g) return {false, 0, 0, 0};
11    x *= c / g;
12    y *= c / g;
13    return {true, x, y, g};
14 }
15
16 // All solutions
17 // x' = x + k*b/g
18 // y' = y - k*a/g
```

### 10.1.2 Geometria Analítica

**Distância entre dois pontos.** Dados dois pontos  $a = (x_1, y_1)$  e  $b = (x_2, y_2)$ , a distância entre  $a$  e  $b$  é dada por:

$$d_{a,b} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

**Condição de alinhamento de três pontos.** Dados três pontos  $a = (x_1, y_1)$ ,  $b = (x_2, y_2)$  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 Circunferência (forma reduzida).** Os pontos  $(x, y)$  que pertencem a uma circunferência  $c$  devem satisfazer:

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

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

**Equação da Circunferência (forma geral).** A partir da equação reduzida da circunferência, encontramos a equação geral:

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

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

1. Reta exterior à circunferência: nenhuma solução. A reta não possui nenhum ponto de comum com a circunferência.
2. Reta tangente à circunferência: solução única. A reta possui apenas 1 ponto em comum com a circunferência.
3. Reta secante à circunferência: duas soluções. A reta cruza a circunferência 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:  $p = \frac{a+b+c+d}{2}$
- 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:  $A_b + \frac{1}{2}P_b \cdot g$ ,  $g$  = 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 Trigonômicas.

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

### Ângulos notáveis.

$\theta$	$0^\circ$	$30^\circ$	$45^\circ$	$60^\circ$	$90^\circ$
$\sin \theta$	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	$\sqrt{3}$	$\infty$

### Propriedades.

$$1. \sin(a+b) = \sin a \cos b + \cos a \sin b$$

$$2. \cos(a+b) = \cos a \cos b - \sin a \sin b$$

$$3. \tan(a+b) = \frac{\tan a + \tan b}{1 - \tan a \tan b}$$

$$4. a \sin x + b \cos x = r \sin(x + \phi), \text{ onde } r = \sqrt{a^2 + b^2} \text{ e } \phi = \tan^{-1} \frac{b}{a}$$

$$5. a \cos x + b \sin x = r \cos(x - \phi), \text{ onde } r = \sqrt{a^2 + b^2} \text{ e } \phi = \tan^{-1} \frac{b}{a}$$

### 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

## 10.3 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  $\sum k_i = n$ :

$$P(n; k_1, k_2, \dots, k_m) = \frac{n!}{k_1! \cdot k_2! \cdot \dots \cdot k_m!}$$

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 uma 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 $n$ ).

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

#### 4. Soma com peso.

$$\sum_{k=0}^n k \cdot \binom{n}{k} = n2^{n-1}$$

#### 2. Soma sobre $k$ .

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

#### 5. $(n+1)$ -ésimo termo da sequência de Fibonacci.

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

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

#### 3. Soma sobre $n$ e $k$ .

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

#### 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:

$$C_0 = C_1 = 1$$

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}, \quad \text{para } n \geq 2.$$

#### 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.
- Número 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  $n \times n$ , que não passam acima da diagonal principal.
- Número de partições não cruzadas de um conjunto de  $n$  elementos.
- Número de maneiras de se cobrir uma escada  $1 \dots n$  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.

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

## 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.

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

**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 primeiros 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,
- Fórmula do produto de Euler:

$$\phi(p) = p - 1$$

- 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:

$$\phi(ab) = \phi(a) \cdot \phi(b) \cdot \frac{d}{\phi(d)}, \quad d = \gcd(a, b)$$

$$\phi(n) = n \prod_{p|n} (1 - \frac{1}{p})$$

- 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  $a \pmod{m}$  ( $\text{ord}_m(a)$ ), 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.**

- $(a \pm b) \pmod{m} = (a \pmod{m} \pm b \pmod{m}) \pmod{m}$ .
- $(a \cdot b) \pmod{m} = (a \pmod{m}) \cdot (b \pmod{m}) \pmod{m}$ .
- $a^b \pmod{m} = (a \pmod{m})^b \pmod{m}$ .
- $a \pm k \equiv b \pm k \pmod{m}$ , 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  $a'^{-1}$ , 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  $M_i = \prod_{j \neq i} m_j$ , o produto de todos os módulos menos  $m_i$ , e  $N_i$  os inversos modulares  $N_i = M_i^{-1} \pmod{m_i}$ . Então, a solução do sistema de congruências é:

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

Para módulos não coprimos, o sistema de congruências tem exatas uma solução módulo  $\text{lcm}(m_1, m_2, \dots, m_k)$ , 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  $p_1^{n_1} p_2^{n_2} \dots p_k^{n_k}$  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.

**Raiz primitiva.** Um número  $g$  é raiz primitiva módulo  $m$  se e somente se para qualquer inteiro  $a$  tal que  $\gcd(a, m) = 1$ , 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 gerador 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 uma 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 os valores são diferentes de 1, então  $g$  é uma raiz primitiva.

## 10.5 Matrizes

Uma matriz é uma estrutura matemática organizada em formato retangular composta por números, símbolos ou expressões dispostas em linhas e colunas.

$$A = [a_{ij}]_{n \times m} = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{vmatrix}$$

### Operações

**Soma.** A soma  $A + B$  de duas matrizes  $n \times m$   $A$  e  $B$  é calculada por:

$$[A + B]_{i,j} = A_{i,j} + B_{i,j}, \quad 1 \leq i \leq n \quad \text{e} \quad 1 \leq j \leq m.$$

**Multiplicação Escalar.** O produto  $cA$  de um escalar  $c$  e uma matriz  $A$  é calculado por:

$$[cA]_{i,j} = cA_{i,j}.$$

**Transposta.** A matriz transposta  $A^T$  da matriz  $A$  é obtida quando as linhas e colunas de  $A$  são trocadas:

$$[A^T]_{i,j} = A_{j,i}.$$

**Produto.** O produto  $AB$  das matrizes  $A$  e  $B$  é definido se  $A$  é de tamanho  $a \times n$  e  $B$  é de tamanho  $n \times b$ . O resultado é uma matriz de tamanho  $a \times b$  nos quais os elementos são calculados usando a fórmula:

$$[AB]_{i,j} = \sum_{k=1}^n A_{i,k} B_{k,j}.$$

Essa operação é associativa, porém não é comutativa.

Uma **matriz identidade** é uma matriz quadrada onde cada elemento na diagonal principal é 1 e os outros elementos são 0. Multiplicar uma matriz por uma matriz identidade não a muda.

**Potência.** A potência  $A^k$  de uma matriz  $A$  é definida se  $A$  é uma matriz quadrada. A definição é baseada na multiplicação de matrizes:

$$A^k = \prod_{i=1}^k A$$

Além disso,  $A^0$  é a matriz identidade.

**Determinante.** A determinante  $\det(A)$  de uma matriz  $A$  é definida se  $A$  é uma matriz quadrada. Se  $A$  é de tamanho  $1 \times 1$ , 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:

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

## 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 resultados 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  $\bar{A}$ , *i.e.* o evento  $A$  não ocorrer, é dado por:

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

**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.*  $A \cap B = \emptyset$ , 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

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

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

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$
$$\mu = np, \quad \sigma^2 = np(1-p)$$

$\text{Bin}(n, p)$  é aproximadamente  $\text{Pois}(np)$  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  $\text{Geo}(p)$ ,  $0 \leq p \leq 1$ .

$$P(X = k) = (1-p)^{k-1} p, \quad k \in \mathbb{N}$$
$$\mu = \frac{1}{p}, \quad \sigma^2 = \frac{1-p}{p^2}$$

**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  $\text{Pois}(\lambda)$ ,  $\lambda = tr$ .

$$P(X = k) = e^{-\lambda} \frac{\lambda^k}{k!}, \quad k \in \mathbb{N}_0$$
$$\mu = \lambda, \quad \sigma^2 = \lambda.$$

### 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  $\text{Uni}(a, b)$ ,  $a < b$ .

$$f(x) = \begin{cases} \frac{1}{b-a}, & a < x < b \\ 0, & \text{caso contrário} \end{cases}$$
$$\mu = \frac{a+b}{2}, \quad \sigma^2 = \frac{(b-a)^2}{12}.$$

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

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$
$$F(x) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$
$$\mu = \frac{1}{\lambda}, \quad \sigma^2 = \frac{1}{\lambda^2}.$$

**Distribuição Normal.** Maioria das variáveis aleatórias reais com média  $\mu$  e variância  $\sigma^2$  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.
2. Soma dos  $n$  primeiros quadrados.

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

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

3. Soma dos  $n$  primeiros cubos.

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

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.

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

(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 $x = y = 0$ , $x \vee y = 1$ caso contrário
Negação	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:

$$x \vee (y \vee z) = (x \vee y) \vee z$$

$$x \wedge (y \wedge z) = (x \wedge y) \wedge z$$

- 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:

$$\neg x \wedge \neg y = \neg(x \vee y)$$

$$\neg x \vee \neg y = \neg(x \wedge y)$$