



Notebook - Maratona de Programação

[UnB] HatsuneMiku não manda WA

Contents

1 Misc	2	5 Geometria	6
1.1 Submask	2	5.1 Mindistpair	6
1.2 Safe Map	2	5.2 Inside Polygon	6
1.3 Ordered Set	2	5.3 3d	7
1.4 Bitwise	2	5.4 Convex Hull	8
2 DP	2	5.5 Linear Transformation	8
2.1 Knapsack	2	5.6 Voronoi	8
2.2 Lis	2	5.7 Polygon Area	8
2.3 Dp Digitos	3	5.8 Intersect Polygon	9
3 ED	3	5.9 Sort By Angle	9
3.1 Prefixsum2d	3	5.10 2d	9
3.2 Sparse Table	3	6 Grafos	11
3.3 Dsu	4	6.1 Dfs Tree	11
3.4 Minqueue	4	6.2 Kosaraju	11
3.5 Segtree Implicita Lazy	4	6.3 Topological Sort	11
3.6 Delta Encoding	5	6.4 Dijkstra	12
4 Strings	5	6.5 Dinic	12
4.1 Z Func	5	6.6 Hungarian	12
4.2 Edit Distance	5	6.7 Floyd Warshall	13
4.3 Lcsubseq	5	6.8 Lca	13
4.4 Kmp	5	6.9 Kruskal	14
4.5 Hash	5	6.10 Ford	14
4.6 Aho Corasick	6	7 Algoritmos	14
4.7 Lcs	6	7.1 Ternary Search	14
		8 Math	15
		8.1 Totient	15
		8.2 Pollard Rho	15

8.3	Inverso Mult	15
8.4	Miller Habin	15
8.5	Matrix Exponentiation	16
8.6	Division Trick	16
8.7	Crivo	16
8.8	Bignod	16
8.9	Linear Diophantine Equation	16
9	Teoria	17
9.1	Geometria	17
9.1.1	Geometria Básica	17
9.1.2	Geometria Analítica	17
9.1.3	Geometria Plana	18
9.1.4	Trigonometria	19
9.2	Análise Combinatória	20
9.2.1	Permutação e Arranjo	20
9.2.2	Combinação	20
9.2.3	Números de Catalan	21
9.2.4	Princípio da Inclusão-Exclusão	21
9.3	Matrizes	22
9.4	Teoria da Probabilidade	23
9.4.1	Introdução à Probabilidade	23
9.4.2	Variáveis Aleatórias	23
9.4.3	Distribuições Discretas	24
9.4.4	Distribuições Contínuas	24
9.5	Progressões	24
9.6	Álgebra Booleana	25
9.6.1	Operações básicas	25
9.6.2	Operações secundárias	25
9.6.3	Leis	25

1 Misc

1.1 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 }
```

1.2 Safe Map

```
1 struct custom_hash {
2     static uint64_t splitmix64(uint64_t x) {
3         // http://xorshift.di.unimi.it/splitmix64.c
4         x += 0x9e3779b97ff4a7c15;
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::
12         steady_clock::now().time_since_epoch().count();
13         return splitmix64(x + FIXED_RANDOM);
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) ^
22             2038074743 ^ a.second;
23         };
24     };
25 }
```

1.3 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,
9     tree_order_statistics_node_update>;
10
11 // order_of_key(k) : Number of items strictly
12 // smaller than k
13 // find_by_order(k) : K-th element in a set (counting
14 // from zero)
15
16 // to swap two sets, use a.swap(b);
```

1.4 Bitwise

```
1 // Least significant bit (lsb)
```

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

2 DP

2.1 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, não pega i
9         temp = dp[i - 1][P];
10
11         // Segunda possibilidade, se puder, pega o
12         // item
13         if(P - p[i] >= 0)
14             temp = max(temp, dp[i - 1][P - p[i]] + v[i]);
15
16         ans = max(ans, temp);
17     }
18 }
```

2.2 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 }

```

2.3 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 }

```

3 ED

3.1 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]+
4     mat[x1-1][y1-1];
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 }

```

3.2 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             st.assign(n+1, vector<vector<int>>(m+1,
47                 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
57                             ) continue;
58
59                         st[x1][y1][j] = st[x1][y1][j-1];
60                         st[x1][y1][j] = f(st[x1][y1][j],
61                             st[x1+delta][y1][j-1]);
62                         st[x1][y1][j] = f(st[x1][y1][j],
63                             st[x1][y1+delta][j-1]);
64                         st[x1+delta][y1+delta][j-1]);
65                     }
66                 }
67             }
68
69             // so funciona para quadrados
70             int query(int x1, int y1, int x2, int y2) {
71                 assert(x2-x1+1 == y2-y1+1);
72                 int k = logv[x2-x1+1];
73                 int delta = (1 << k);
74
75                 int res = st[x1][y1][k];
76                 res = f(res, st[x2 - delta+1][y1][k]);
77                 res = f(res, st[x1][y2 - delta+1][k]);
78                 res = f(res, st[x2 - delta+1][y2 - delta+1][k
79                     ]);

```

```

77     return res;
78 }
79
80 int f(int a, int b) {
81     return a | b;
82 }
83
84 };

```

3.3 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 };

```

3.4 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.
7         top().ss);
8         in.push({val, minimum});
9     }
10
11     ll pop() {
12         if(out.empty()) {
13             while(!in.empty()) {
14                 ll val = in.top().ff;
15                 in.pop();
16                 ll minimum = out.empty() ? val : min(
17                 val, out.top().ss);
18                 out.push({val, minimum});
19             }
20             ll res = out.top().ff;
21             out.pop();
22             return res;
23         }
24
25         ll minn() {
26             ll minimum = LLINF;
27             if(in.empty() || out.empty())
28                 minimum = in.empty() ? (ll)out.top().ss :
29                 (ll)in.top().ss;
30             else
31                 minimum = min((ll)in.top().ss, (ll)out.
32                 top().ss);
33         }
34     }
35 };

```

```

30
31     return minimum;
32 }
33
34 ll size() {
35     return in.size() + out.size();
36 }
37 };

```

3.5 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
38 no=1){
39     prop(l, r, no);
40     if(a<=l and r<=b){
41         tree[no].lazy += x;
42         prop(l, r, no);
43         return;
44     }
45     if(r<a or b<l) return;
46     int m = (l+r)/2;
47     update(a, b, x, l, m, tree[no].l);
48     update(a, b, x, m+1, r, tree[no].r);
49
50     tree[no].val = merge(tree[tree[no].l].val, tree[
51     tree[no].r].val);
52 }
53
54 pll query(int a, int b, int l=0, int r=2*N, int no=1)
55 {
56     prop(l, r, no);
57     if(a<=l and r<=b) return tree[no].val;
58     if(r<a or b<l) return {INF, 0};
59     int m = (l+r)/2;
60     int left = tree[no].l, right = tree[no].r;
61 }

```

```

59     return tree[no].val = merge(query(a, b, l, m,
60                                query(a, b, m+1, r,
61                                right));

```

3.6 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 }

```

4 Strings

4.1 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 - l], 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    return z;
12 }

```

4.2 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 }

```

4.3 Lcs subseq

```

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 }

```

4.4 Kmp

```

1 string p;
2 int neighbor[N];
3 int walk(int u, char c) { // leader after inputting '
4     while (u != -1 && (u+1 >= (int)p.size() || p[u +
5     1] != c)) // leader doesn't match
6         u = neighbor[u];
7     return p[u + 1] == c ? u+1 : u;
8 }
9 void build() {
10    neighbor[0] = -1; // -1 is the leftmost state
11    for (int i = 1; i < (int)p.size(); i++)
12        neighbor[i] = walk(neighbor[i-1], p[i]);
13 }

```

4.5 Hash

```

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

```

```

21     }
22     int query_inv(int l, int r) {
23         ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-1
24         +1] % MOD : 0));
25         return hash < 0 ? hash + MOD : hash;
26     };

```

4.6 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){
24                     fail[v] = to[ fail[u] ][i];
25                     term[v] += term[ fail[v] ];
26                 }
27             }
28             else if(u != 1) to[u][i] = to[ fail[u] ][i];
29         }
30         else to[u][i] = 1;
31     }
32 }

```

4.7 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]
16                + 1;
17                if(len[currRow][j] > result){
18                    result = len[currRow][j];
19                    end = i - 1;
20                }
21            }
22            else
23                len[currRow][j] = 0;
24        }
25    }
26 }

```

```

25         currRow = 1 - currRow;
26     }
27
28     if(result==0)
29         return string();
30
31     return X.substr(end - result + 1, result);
32 }

```

5 Geometria

5.1 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->y;
22            ll dy = vet[i].y - it->x;
23            if(best_dist > dx*dx + dy*dy){
24                best_dist = dx*dx + dy*dy;
25                // vet[i] e inv(it)
26            }
27        }
28
29        s.insert(point(vet[i].y, vet[i].x));
30    }
31    return best_dist;
32 }

```

5.2 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 bool inside(vp &p, point e){ // ccw
11     int l=2, r=(int)p.size()-1;
12     while(l<r){
13         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;

```

```

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

```

5.3 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
.z);
28 }
29 cod operator*(const point &o) const { // dot
30 return x*o.x + y*o.y + z*o.z;
31 }
32 point operator^(const point &o) const { // cross
33 return point(y*o.z - z*o.y,
34 z*o.x - x*o.z,
35 x*o.y - y*o.x);
36 }
37 };

```

```

38 ld norm(point a) { // Modulo
39 return sqrt(a * a);
40 }
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 ld proj(point a, point b) { // a sobre b
49 return (a*b)/norm(b);
50 }
51 ld angle(point a, point b) { // em radianos
52 return acos((a*b) / norm(a) / norm(b));
53 }
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
), p2(p2), p3(p3) {
67 point aux = (p1-p3)^(p2-p3);
68 a = aux.x; b = aux.y; c = aux.z;
69 d = -a*p1.x - b*p1.y - c*p1.z;
70 }
71 plane(point p, point normal) {
72 normal = normilize(normal);
73 a = normal.x; b = normal.y; c = normal.z;
74 d = -(p*normal);
75 }
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
83 cod dist(plane pl, point p) {
84 return fabs(pl.a*p.x + pl.b*p.y + pl.c*p.z + pl.d
) / sqrt(pl.a*pl.a + pl.b*pl.b + pl.c*pl.c);
85 }
86
87 point rotate(point v, point k, ld theta) {
88 // Rotaciona o vetor v theta graus em torno do
eixo k
89 // theta *= PI/180; // graus
90 return (
91 v*cos(theta)) +
92 ((k^v)*sin(theta)) +
93 (k*(k^v))*(1-cos(theta)
94 );
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,
point p4) {
102 cod mua = ( d(p1, p3, p4, p3) * d(p4, p3, p2, p1)
- d(p1, p3, p2, p1) * d(p4, p3, p4, p3) )
103 / ( d(p2, p1, p2, p1) * d(p4, p3, p4, p3)
- d(p4, p3, p2, p1) * d(p4, p3, p2, p1) );

```



```

104     cod mub = ( d(p1, p3, p4, p3) + mua * d(p4, p3,
105     p2, p1) ) / d(p4, p3, p4, p3);
106     point pa = p1 + (p2-p1) * mua;
107     point pb = p3 + (p4-p3) * mub;
108     if (pa == pb) return {pa};
109     return {};

```

5.4 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;

```

5.5 Linear Transformation

```

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

```

5.6 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
18      + b.y) * 0.5;
19      return ans;
20  }
21
22  vp cutPolygon(vp poly, line seg) {
23      int n = (int) poly.size();
24      vp ans;
25      for(int i = 0; i < n; i++) {
26          double z = seg.eval(poly[i]);
27          if(z > -EPS) {

```

```

28              ans.push_back(poly[i]);
29          }
30          double z2 = seg.eval(poly[(i + 1) % n]);
31          if((z > EPS && z2 < -EPS) || (z < -EPS && z2
32          > EPS)) {
33              ans.push_back(inter_line(seg, line(poly[i
34              ], poly[(i + 1) % n]))[0]);
35          }
36      }
37      return ans;
38  }
39
40  // BE CAREFUL!
41  // the first point may be any point
42  // O(N^3)
43  vp getCell(vp pts, int i) {
44      vp ans;
45      ans.emplace_back(0, 0);
46      ans.emplace_back(1e6, 0);
47      ans.emplace_back(1e6, 1e6);
48      ans.emplace_back(0, 1e6);
49      for(int j = 0; j < (int) pts.size(); j++) {
50          if(j != i) {
51              ans = cutPolygon(ans, getBisector(pts[i],
52              pts[j]));
53          }
54      }
55      return ans;
56  }
57
58  // O(N^2) expected time
59  vector<vp> getVoronoi(vp pts) {
60      // assert(pts.size() > 0);
61      int n = (int) pts.size();
62      vector<int> p(n, 0);
63      for(int i = 0; i < n; i++) {
64          p[i] = i;
65      }
66      shuffle(p.begin(), p.end(), rng);
67      vector<vp> ans(n);
68      ans[0].emplace_back(0, 0);
69      ans[0].emplace_back(w, 0);
70      ans[0].emplace_back(w, h);
71      ans[0].emplace_back(0, h);
72      for(int i = 1; i < n; i++) {
73          ans[i] = ans[0];
74      }
75      for(auto i : p) {
76          for(auto j : p) {
77              if(j == i) break;
78              auto bi = getBisector(pts[j], pts[i]);
79              if(!polygonIntersection(bi, ans[j]))
80                  continue;
81              ans[j] = cutPolygon(ans[j], getBisector(
82              pts[j], pts[i]));
83              ans[i] = cutPolygon(ans[i], getBisector(
84              pts[i], pts[j]));
85          }
86      }
87      return ans;
88  }

```

5.7 Polygon Area

```

1  ll area = 0;
2
3  for(int i = 0; i < n - 1; ++i){
4      area += pontos[i].x*pontos[i+1].y - pontos[i+1].x
5      *pontos[i].y;
6  }
7  area += pontos[n-1].x*pontos[0].y - pontos[0].x*
8  pontos[n-1].y;

```

```
8 area = abs(area);
```

5.8 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;
14 }
```

5.9 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 }
```

5.10 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
+ o.x, y + o.y}; }
15     point operator-(const point &o) const{ return {x
- o.x, y - o.y}; }
16     point operator*(T t) const{ return {x * t, y * t
}; }
17     point operator/(T t) const{ return {x / t, y / t
}; }
18     operator*(const point &o) const{ return x * o.x
+ y * o.y; }
19     operator^(const point &o) const{ return x * o.y
- y * o.x; }
20     bool operator<(const point &o) const{
21         return (eq(x, o.x) ? y < o.y : x < o.x);
22     }
23     bool operator==(const point &o) const{
24         return eq(x, o.x) and eq(y, o.y);
```

```
25     }
26     friend ostream& operator<<(ostream& os, point p)
27     {
28         return os << "(" << p.x << "," << p.y << ")";
29     }
30 };
31
32 int ccw(point a, point b, point e){ // -1=dir; 0=
33     collinear; 1=esq;
34     T tmp = (b-a) ^ (e-a); // vector from a to b
35     return (tmp > EPS) - (tmp < -EPS);
36 }
37
38 ld norm(point a){ // Modulo
39     return sqrt(a * a);
40 }
41
42 T norm2(point a){
43     return a * a;
44 }
45
46 bool nulo(point a){
47     return (eq(a.x, 0) and eq(a.y, 0));
48 }
49
50 point rotccw(point p, ld a){
51     // a = PI*a/180; // graus
52     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
+ p.x*sin(a)));
53 }
54
55 point rot90cw(point a) { return point(a.y, -a.x); };
56 point rot90ccw(point a) { return point(-a.y, a.x); };
57
58 ld proj(point a, point b){ // a sobre b
59     return a*b/norm(b);
60 }
61
62 ld angle(point a, point b){ // em radianos
63     ld ang = a*b / norm(a) / norm(b);
64     return acos(max(min(ang, (ld)1), (ld)-1));
65 }
66
67 ld angle_vec(point v){
68     // return 180/PI*atan2(v.x, v.y); // graus
69     return atan2(v.x, v.y);
70 }
71
72 ld order_angle(point a, point b){ // from a to b ccw
73     (a in front of b)
74     ld aux = angle(a,b)*180/PI;
75     return ((a^b)<=0 ? aux:360-aux);
76 }
77
78 bool angle_less(point a1, point b1, point a2, point
79     b2){ // ang(a1,b1) <= ang(a2,b2)
80     point p1((a1*b1), abs((a1^b1)));
81     point p2((a2*b2), abs((a2^b2)));
82     return (p1^p2) <= 0;
83 }
84
85 ld area(vp &p){ // (points sorted)
86     ld ret = 0;
87     for(int i=2;i<(int)p.size();i++)
88         ret += (p[i]-p[0])^(p[i-1]-p[0]);
89     return abs(ret/2);
90 }
91
92 ld areaT(point &a, point &b, point &c){
93     return abs((b-a)^(c-a))/2.0;
94 }
95
96 point center(vp &A){
97     point c = point();
98     int len = A.size();
99     for(int i=0;i<len;i++)
100         c=c+A[i];
101     return c/len;
102 }
103
104 point forca_mod(point p, ld m){
105     ld cm = norm(p);
```

```

92     if(cm<EPS) return point();
93     return point(p.x*m/cm,p.y*m/cm);
94 }
95
96 ld param(point a, point b, point v){
97     // v = t*(b-a) + a // return t;
98     // assert(line(a, b).inside_seg(v));
99     return ((v-a) * (b-a)) / ((b-a) * (b-a));
100 }
101
102 bool simetric(vp &a){ //ordered
103     int n = a.size();
104     point c = center(a);
105     if(n&1) return false;
106     for(int i=0;i<n/2;i++)
107         if(ccw(a[i], a[i+n/2], c) != 0)
108             return false;
109     return true;
110 }
111
112 point mirror(point m1, point m2, point p){
113     // mirror point p around segment m1m2
114     point seg = m2-m1;
115     ld t0 = ((p-m1)*seg) / (seg*seg);
116     point ort = m1 + seg*t0;
117     point pm = ort-(p-ort);
118     return pm;
119 }
120
121
122 ///////////////
123 // Line //
124 ///////////////
125
126 struct line{
127     point p1, p2;
128     T a, b, c; // ax+by+c = 0;
129     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
130     line(point p1=0, point p2=0): p1(p1), p2(p2){
131         a = p1.y - p2.y;
132         b = p2.x - p1.x;
133         c = p1 ^ p2;
134     }
135     line(T a=0, T b=0, T c=0): a(a), b(b), c(c){
136         // Gera os pontos p1 p2 dados os coeficientes
137         // isso aqui eh um lixo mas quebra um galho
138         kkkkkk
139         if(b==0){
140             p1 = point(1, -c/a);
141             p2 = point(0, -c/a);
142         }else{
143             p1 = point(1, (-c-a*1)/b);
144             p2 = point(0, -c/b);
145         }
146     }
147     T eval(point p){
148         return a*p.x+b*p.y+c;
149     }
150     bool inside(point p){
151         return eq(eval(p), 0);
152     }
153     point normal(){
154         return point(a, b);
155     }
156
157     bool inside_seg(point p){
158         return (
159             ((p1-p) ^ (p2-p)) == 0 and
160             ((p1-p) * (p2-p)) <= 0
161         );
162     }
163
164 };
165
166 // be careful with precision error
167 vp inter_line(line l1, line l2){
168     ld det = l1.a*l2.b - l1.b*l2.a;
169     if(det==0) return {};
170     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
171     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
172     return {point(x, y)};
173 }
174
175 // segments not collinear
176 vp inter_seg(line l1, line l2){
177     vp ans = inter_line(l1, l2);
178     if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.
179         inside_seg(ans[0]))
180         return {};
181     return ans;
182 }
183
184 bool seg_has_inter(line l1, line l2){
185     return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.
186         p2, l2.p2) < 0 and
187         ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.
188         p2, l1.p2) < 0;
189 }
190
191 ld dist_seg(point p, point a, point b){ // point -
192     seg
193     if((p-a)*(b-a) < EPS) return norm(p-a);
194     if((p-b)*(a-b) < EPS) return norm(p-b);
195     return abs((p-a)^(b-a)) / norm(b-a);
196 }
197
198 ld dist_line(point p, line l){ // point - line
199     return abs(l.eval(p))/sqrt(1.a*1.a + 1.b*1.b);
200 }
201
202 line bisector(point a, point b){
203     point d = (b-a)*2;
204     return line(d.x, d.y, a*a - b*b);
205 }
206
207 line perpendicular(line l, point p){ // passes
208     through p
209     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
210 }
211
212 struct circle{
213     point c; T r;
214     circle(): c(0, 0), r(0){}
215     circle(const point o): c(o), r(0){}
216     circle(const point a, const point b){
217         c = (a+b)/2;
218         r = norm(a-c);
219     }
220     circle(const point a, const point b, const point
221         cc){
222         assert(ccw(a, b, cc) != 0);
223         c = inter_line(bisector(a, b), bisector(b, cc
224         ))[0];
225         r = norm(a-c);
226     }
227     bool inside(const point &a) const{
228         return norm(a - c) <= r + EPS;
229     }
230 };
231
232 pair<point, point> tangent_points(circle cr, point p)

```

```

    {
230     ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
231     point p1 = rotccw(cr.c-p, -theta);
232     point p2 = rotccw(cr.c-p, theta);
233     assert(d1 >= cr.r);
234     p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
235     p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
236     return {p1, p2};
237 }
238
239
240 circle incircle(point p1, point p2, point p3){
241     ld m1 = norm(p2-p3);
242     ld m2 = norm(p1-p3);
243     ld m3 = norm(p1-p2);
244     point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
245     ld s = 0.5*(m1+m2+m3);
246     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
247     return circle(c, r);
248 }
249
250 circle circumcircle(point a, point b, point c) {
251     circle ans;
252     point u = point((b-a).y, -(b-a).x);
253     point v = point((c-a).y, -(c-a).x);
254     point n = (c-b)*0.5;
255     ld t = (u^n)/(v^u);
256     ans.c = ((a+c)*0.5) + (v*t);
257     ans.r = norm(ans.c-a);
258     return ans;
259 }
260
261 vp inter_circle_line(circle C, line L){
262     point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.
263     p1)*(ab) / (ab*ab));
264     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
265     / (ab*ab);
266     if (h2 < -EPS) return {};
267     if (eq(h2, 0)) return {p};
268     point h = (ab/norm(ab)) * sqrt(h2);
269     return {p - h, p + h};
270 }
271
272 vp inter_circle(circle c1, circle c2){
273     if (c1.c == c2.c) { assert(c1.r != c2.r); return
274     {}; }
275     point vec = c2.c - c1.c;
276     ld d2 = vec * vec, sum = c1.r + c2.r, dif = c1.r
277     - c2.r;
278     ld p = (d2 + c1.r * c1.r - c2.r * c2.r) / (2 * d2
279     );
280     ld h2 = c1.r * c1.r - p * p * d2;
281     if (sum * sum < d2 or dif * dif > d2) return {};
282     point mid = c1.c + vec * p, per = point(-vec.y,
283     vec.x) * sqrt(fmax(0, h2) / d2);
284     if (eq(per.x, 0) and eq(per.y, 0)) return {mid};
285     return {mid + per, mid - per};
286 }
287
288 // minimum circle cover O(n) amortizado
289 circle min_circle_cover(vp v){
290     random_shuffle(v.begin(), v.end());
291     circle ans;
292     int n = v.size();
293     for(int i=0;i<n;i++){
294         ans = circle(v[i]);
295         for(int j=0;j<i;j++){
296             if(!ans.inside(v[j])){
297                 ans = circle(v[i], v[j]);
298                 for(int k=0;k<j;k++){
299                     if(!ans.inside(v[k])){
300                         ans = circle(v[i], v[j], v[k]);
301                     }
302                 }
303             }
304         }
305     }
306 }

```

```

    }
295     return ans;
296 }
297 }

```

6 Grafos

6.1 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 (
5 // ou =) u e sobem pra cima de u; backedges[u] == 0
6 // => u eh ponte
7 void dfs(int u, int p) {
8     if(vis[u]) return;
9     pai[u] = p;
10    h[u] = h[p]+1;
11    vis[u] = 1;
12
13    for(auto v : g[u]) {
14        if(p == v or vis[v]) continue;
15        dfs(v, u);
16        backedges[u] += backedges[v];
17    }
18    for(auto v : g[u]) {
19        if(h[v] > h[u]+1)
20            desce[u]++;
21        else if(h[v] < h[u]-1)
22            sobe[u]++;
23    }
24    backedges[u] += sobe[u] - desce[u];
25 }

```

6.2 Kosaraju

```

1 vector<int> g[N], gi[N]; // grafo invertido
2 int vis[N], comp[N]; // componente conexo de cada
3 // vertice
4 stack<int> S;
5
6 void dfs(int u){
7     vis[u] = 1;
8     for(auto v: g[u]) if(!vis[v]) dfs(v);
9     S.push(u);
10 }
11
12 void scc(int u, int c){
13     vis[u] = 1; comp[u] = c;
14     for(auto v: gi[u]) if(!vis[v]) scc(v, c);
15 }
16
17 void kosaraju(int n){
18     for(int i=0;i<n;i++) vis[i] = 0;
19     for(int i=0;i<n;i++) if(!vis[i]) dfs(i);
20     for(int i=0;i<n;i++) vis[i] = 0;
21     while(S.size()){
22         int u = S.top();
23         S.pop();
24         if(!vis[u]) scc(u, u);
25     }
26 }

```

6.3 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 }

```

6.4 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 }

```

6.5 Dinic

```

1 const int N = 300;
2
3 struct Dinic {
4     struct Edge{
5         int from, to; ll flow, cap;
6     };
7     vector<Edge> edge;
8
9     vector<int> g[N];
10    int ne = 0;
11    int lvl[N], vis[N], pass;
12    int qu[N], px[N], qt;
13
14    ll run(int s, int sink, ll minE) {
15        if(s == sink) return minE;
16
17        ll ans = 0;
18
19        for(; px[s] < (int)g[s].size(); px[s]++) {
20            int e = g[s][px[s]];
21            auto &v = edge[e], &rev = edge[e^1];
22            if(lvl[v.to] != lvl[s]+1 || v.flow >= v.
cap)

```

```

23                continue; // v.cap - v.flow
24            < lim
25                ll tmp = run(v.to, sink, min(minE, v.cap-v
26                .flow));
27                v.flow += tmp, rev.flow -= tmp;
28                ans += tmp, minE -= tmp;
29                if(minE == 0) break;
30            }
31            return ans;
32        }
33    }
34    bool bfs(int source, int sink) {
35        qt = 0;
36        qu[qt++] = source;
37        lvl[source] = 1;
38        vis[source] = ++pass;
39        for(int i = 0; i < qt; i++) {
40            int u = qu[i];
41            px[u] = 0;
42            if(u == sink) return true;
43            for(auto& ed : g[u]) {
44                auto v = edge[ed];
45                if(v.flow >= v.cap || vis[v.to] ==
pass)
46                    continue; // v.cap - v.flow < lim
47                vis[v.to] = pass;
48                lvl[v.to] = lvl[u]+1;
49                qu[qt++] = v.to;
50            }
51            return false;
52        }
53    }
54    ll flow(int source, int sink) {
55        reset_flow();
56        ll ans = 0;
57        //for(lim = (1LL << 62); lim >= 1; lim /= 2)
58        while(bfs(source, sink))
59            ans += run(source, sink, LLINF);
60        return ans;
61    }
62    void addEdge(int u, int v, ll c, ll rc) {
63        Edge e = {u, v, 0, c};
64        edge.pb(e);
65        g[u].push_back(ne++);
66
67        e = {v, u, 0, rc};
68        edge.pb(e);
69        g[v].push_back(ne++);
70    }
71    void reset_flow() {
72        for(int i = 0; i < ne; i++)
73            edge[i].flow = 0;
74        memset(lvl, 0, sizeof(lvl));
75        memset(vis, 0, sizeof(vis));
76        memset(qu, 0, sizeof(qu));
77        memset(px, 0, sizeof(px));
78        qt = 0; pass = 0;
79    }
80    vector<pair<int, int>> cut() {
81        vector<pair<int, int>> cuts;
82        for (auto [from, to, flow, cap]: edge) {
83            if (flow == cap and vis[from] == pass and
vis[to] < pass and cap>0) {
84                cuts.pb({from, to});
85            }
86        }
87        return cuts;
88    }
89 }
90 };

```

6.6 Hungarian

```

1 // Hungarian Algorithm
2 //

```

```

3 // Assignment problem
4 // Put the edges in the 'a' matrix (negative or
   positive)
5 // assignment() returns a pair with the min
   assignment,
6 // and the column chosen by each row
7 // assignment() - O(n^3)
8
9 template<typename T>
10 struct hungarian {
11     int n, m;
12     vector<vector<T>> a;
13     vector<T> u, v;
14     vector<int> p, way;
15     T inf;
16
17     hungarian(int n_, int m_) : n(n_), m(m_), u(m+1),
18         v(m+1), p(m+1), way(m+1) {
19         a = vector<vector<T>>(n, vector<T>(m));
20         inf = numeric_limits<T>::max();
21     }
22     pair<T, vector<int>> assignment() {
23         for (int i = 1; i <= n; i++) {
24             p[0] = i;
25             int j0 = 0;
26             vector<T> minv(m+1, inf);
27             vector<int> used(m+1, 0);
28             do {
29                 used[j0] = true;
30                 int i0 = p[j0], j1 = -1;
31                 T delta = inf;
32                 for (int j = 1; j <= m; j++) if (!
33                     used[j]) {
34                     T cur = a[i0-1][j-1] - u[i0] - v[
35                         j];
36                     if (cur < minv[j]) minv[j] = cur;
37                     way[j] = j0;
38                     if (minv[j] < delta) delta = minv[
39                         j];
40                     j1 = j;
41                 }
42                 for (int j = 0; j <= m; j++)
43                     if (used[j]) u[p[j]] += delta, v[
44                         j] -= delta;
45                 else minv[j] -= delta;
46                 j0 = j1;
47             } while (p[j0] != 0);
48             do {
49                 int j1 = way[j0];
50                 p[j0] = p[j1];
51                 j0 = j1;
52             } while (j0);
53             vector<int> ans(m);
54             for (int j = 1; j <= n; j++) ans[p[j]-1] = j
55             -1;
56             return make_pair(-v[0], ans);
57         }
58     };
59 };

```

6.7 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] +
9                 dist[k][j]);

```

6.8 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 // lca O(1)
37
38 template<typename T> struct rmq {
39     vector<T> v;
40     int n; static const int b = 30;
41     vector<int> mask, t;
42
43     int op(int x, int y) { return v[x] < v[y] ? x : y
44         ; }
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         }
56         for (int i = 0; i < n/b; i++) t[i] = b*i+b-1-
57             msb(mask[b*i+b-1]);
58         for (int j = 1; (1<<j) <= n/b; j++) for (int
59             i = 0; i+(1<<j) <= n/b; i++)
60             t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j
61                 -1)+i+(1<<(j-1))]);
62     }
63     int small(int r, int sz = b) { return r-msb(mask[
64         r]&((1<<sz)-1)); }
65     T query(int l, int r) {
66         if (r-l+1 <= b) return small(r, r-l+1);
67         int ans = op(small(l+b-1), small(r));
68         int x = l/b+1, y = r/b-1;
69         if (x <= y) {
70             int j = msb(y-x+1);
71             ans = op(ans, op(t[n/b*j+x], t[n/b*j+y
72                 -(1<<j)+1]));
73         }
74     }
75 };

```

```

64     }
65     return ans;
66 }
67 };
68
69 namespace lca {
70     vector<int> g[N];
71     int v[2*N], pos[N], dep[2*N];
72     int t;
73     rmq<int> RMQ;
74
75     void dfs(int i, int d = 0, int p = -1) {
76         v[t] = i, pos[i] = t, dep[t++] = d;
77         for (int j : g[i]) if (j != p) {
78             dfs(j, d+1, i);
79             v[t] = i, dep[t++] = d;
80         }
81     }
82     void build(int n, int root) {
83         t = 0;
84         dfs(root);
85         RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
86     }
87     int lca(int a, int b) {
88         a = pos[a], b = pos[b];
89         return v[RMQ.query(min(a, b), max(a, b))];
90     }
91     int dist(int a, int b) {
92         return dep[pos[a]] + dep[pos[b]] - 2*dep[pos[lca(a, b)]];
93     }
94 }

```

6.9 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
28 struct Edge {
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
48     return mst;
49 }

```

6.10 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].c - edges[e].f) > 0) {
32                if(int a = dfs(edges[e].to, t, min(f, edges[e].c - edges[e].f), tempo)) {
33                    edges[e].f += a;
34                    edges[e^1].f -= a;
35                    return a;
36                }
37            }
38        }
39        return 0;
40    }
41
42    int flow(int s, int t) {
43        int mflow = 0, tempo = 1;
44        while(int a = dfs(s, t, INF, tempo)) {
45            mflow += a;
46            tempo++;
47        }
48        return mflow;
49    }
50 };

```

7 Algoritmos

7.1 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 }

```

8 Math

8.1 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+p;i<=n;i+=p){
32                 isprime[i] = false;
33                 tot[i] = (tot[i]/p)*(p-1);
34             }
35         }
36     }
37     return tot;
38 }

```

8.2 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 /= 211, a = mul(a, a, m)) {
9         if (b % 211 == 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 }
35 ll rho(ll n) {
36     if (n == 1 or prime(n)) return n;
37     auto f = [n](ll x) {return mul(x, x, n) + 1;};
38
39     ll x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
40     while (t % 40 != 0 or gcd(prd, n) == 1) {
41         if (x==y) x = ++x0, y = f(x);
42         q = mul(prd, abs(x-y), n);
43         if (q != 0) prd = q;
44         x = f(x), y = f(f(y)), t++;
45     }
46     return gcd(prd, n);
47 }
48 vector<ll> fact(ll n) { // retorna fatoracao em
49     primos
50     if (n == 1) return {};
51     if (prime(n)) return {n};
52     ll d = rho(n);
53     vector<ll> l = fact(d), r = fact(n / d);
54     l.insert(l.end(), r.begin(), r.end());
55     return l;
56 }

```

8.3 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 }

```

8.4 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}) {
29         if (i >= n) break;
30         ll x = expo(i, d, n);
31         if (x == 1 or x == n - 1) continue;
32
33         bool composto = 1;
34         for (int j = 0; j < r - 1; j++) {
35             x = mul(x, x, n);
36             if (x == n - 1) {
37                 composto = 0;
38                 break;
39             }
40         }
41         if (composto) return 0;
42     }
43     return 1;
44 }

```

8.5 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     }
35 }

```

```

32
33         return Matrix(res);
34     }
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
44     return res;
45 }

```

8.6 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 }

```

8.7 Crivo

```

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

```

8.8 Bigmod

```

1 ll mod(string a, ll p) {
2     ll res = 0, b = 1;
3     reverse(all(a));
4
5     for(auto c : a) {
6         ll tmp = (((ll)c-'0')*b) % p;
7         res = (res + tmp) % p;
8
9         b = (b * 10) % p;
10    }
11
12    return res;
13 }

```

8.9 Linear Diophantine Equation

```

1 // Linear Diophantine Equation
2 array<ll, 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<ll, 4> find_any_solution(ll a, ll b, ll 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

```

9 Teoria

9.1 Geometria

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

9.1.2 Geometria Analítica

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

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

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

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

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

$$ax + by + c = 0$$

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

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

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

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

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

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

Equação da 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.

9.1.3 Geometria Plana

Triângulos.

- 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.

Figura	Área (A)	Perímetro (P)	Diagonal (d)
Quadrado	l^2	$4l$	$l\sqrt{2}$
Retângulo	bh	$2(b + h)$	$\sqrt{b^2 + h^2}$
Losango	$\frac{1}{2}Dd$	$4l$	$l\sqrt{2}$
Trapézio	$\frac{1}{2}h(B + b)$	$B + b + l_1 + l_2$	$\sqrt{h^2 + \frac{(B-b)^2}{4h}}$

Círculos.

- 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:

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

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

9.1.4 Trigonometria

Funções Trigonométricas.

$$\sin \theta = \frac{\text{cateto oposto a } \theta}{\text{hipotenusa}}$$

$$\cos \theta = \frac{\text{cateto adjacente a } \theta}{\text{hipotenusa}}$$

$$\tan \theta = \frac{\text{cateto oposto a } \theta}{\text{cateto adjacente a } \theta}$$

Ângulos notáveis.

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

Propriedades.

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

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

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

$$4. \sin a + \sin b = 2 \sin \frac{a+b}{2} \cos \frac{a-b}{2}$$

$$5. \sin a - \sin b = 2 \cos \frac{a+b}{2} \sin \frac{a-b}{2}$$

$$6. \cos a + \cos b = 2 \cos \frac{a+b}{2} \cos \frac{a-b}{2}$$

$$7. \cos a - \cos b = 2 \sin \frac{a+b}{2} \sin \frac{a-b}{2}$$

$$8. a \sin x + b \cos x = r \sin(x + \phi), \text{ onde } r = \sqrt{a^2 + b^2} \text{ e } \phi = \tan^{-1} \frac{b}{a}$$

$$9. a \cos x + b \sin x = r \cos(x - \phi), \text{ onde } r = \sqrt{a^2 + b^2} \text{ e } \phi = \tan^{-1} \frac{b}{a}$$

10. Lei dos Senos:

$$\frac{a}{\sin \hat{A}} = \frac{b}{\sin \hat{B}} = \frac{c}{\sin \hat{C}} = 2r.$$

11. Lei dos Cossenos:

$$a^2 = b^2 + c^2 + 2 \cdot b \cdot c \cdot \cos \hat{A}$$

$$b^2 = a^2 + c^2 + 2 \cdot a \cdot c \cdot \cos \hat{B}$$

$$c^2 = b^2 + a^2 + 2 \cdot b \cdot a \cdot \cos \hat{C}$$

12. Teorema de Tales: A interseção de um feixe de retas paralelas por duas retas transversais forma segmentos proporcionais:

$$\frac{\overline{AB}}{\overline{BC}} = \frac{\overline{DE}}{\overline{EF}}$$

13. 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).

9.2 Análise Combinatória

9.2.1 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

9.2.2 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).**

2. **Soma sobre k .**

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

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

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

3. Soma sobre n e k .

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

4. Soma com peso.

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

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

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

6. Soma dos quadrados.

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

9.2.3 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.
- Números de árvores binárias enraizadas cheias com $n+1$ folhas (vértices não são numerados), ou, equivalentemente, com um total de n nós internos. Uma árvore binária enraizada é cheia se cada vértice tem dois filhos ou nenhum.
- Número de maneiras de colocar parênteses completamente em $n+1$ fatores.
- Número de triangularizações de um polígono convexo com $n+2$ lados.
- Número de maneiras de conectar $2n$ pontos em um círculo para formar n cordas disjuntas.
- Número de árvores binárias completas não isomórficas com $n+1$ nós.
- Número de caminhos monotônicos na grade de pontos do ponto $(0,0)$ ao ponto (n,n) em uma grade quadrada de tamanho $n \times n$, que não passam acima da diagonal principal.
- Número de partições não cruzadas de um conjunto de n elementos.
- Números 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*.

9.2.4 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.

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

9.3 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 1×1 , 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:

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

9.4 Teoria da Probabilidade

9.4.1 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:

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

9.4.2 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 σ é o desvio-padrão.

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

Linearidade do Valor Esperado.

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

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

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

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

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

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

9.4.4 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 μ e variância σ^2 são bem descritas por $N(\mu, \sigma^2)$, $\sigma > 0$.

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

9.5 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.

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

(d) Produto dos termos.

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

9.6 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.

9.6.1 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 $x = 0$

9.6.2 Operações secundárias

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

Operador lógico	Operador	Notação	Definição	Equivalência
Condição material	\rightarrow	$x \rightarrow y$	$x \rightarrow y = 0$ se $x = 1$ e $y = 0$, $x \rightarrow y = 1$ caso contrário	$\neg x \vee y$
Bicondição 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	$(x \vee y) \wedge (\neg x \vee \neg y)$

9.6.3 Leis

• Associatividade:

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

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

• Comutatividade:

$$x \wedge y = y \wedge x$$

$$x \vee y = y \vee x$$

• Distributividade:

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

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

• Identidade: $x \vee 0 = x$ e $x \wedge 1 = x$

• 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:

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

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