



# Notebook - Maratona de Programação

[UnB] HatsuneMiku não manda WA

## Contents

<b>1 Algoritmos</b>	<b>2</b>	5.4 Hungarian . . . . .	10
1.1 Ternary Search . . . . .	2	5.5 Kosaraju . . . . .	10
<b>2 DP</b>	<b>2</b>	<b>6 Math</b>	<b>10</b>
2.1 Dp Digitos . . . . .	2	6.1 Bigmod . . . . .	10
2.2 Knapsack . . . . .	2	6.2 Division Trick . . . . .	11
2.3 Lis . . . . .	2	6.3 Inverso Mult . . . . .	11
<b>3 ED</b>	<b>2</b>	6.4 Linear Diophantine Equation . . . . .	11
3.1 Minqueue . . . . .	2	6.5 Matrix Exponentiation . . . . .	11
3.2 Segtree Implicita Lazy . . . . .	3	6.6 Totient . . . . .	11
3.3 Sparse Table . . . . .	3	<b>7 Misc</b>	<b>12</b>
<b>4 Geometria</b>	<b>4</b>	7.1 Bitwise . . . . .	12
4.1 2d . . . . .	4	<b>8 Strings</b>	<b>12</b>
4.2 3d . . . . .	6	8.1 Aho Corasick . . . . .	12
4.3 Convex Hull . . . . .	7	8.2 Edit Distance . . . . .	12
4.4 Inside Polygon . . . . .	7	8.3 Hash . . . . .	12
4.5 Intersect Polygon . . . . .	7	8.4 Kmp . . . . .	13
4.6 Linear Transformation . . . . .	7	8.5 Lcs . . . . .	13
4.7 Mindistpair . . . . .	8	8.6 Lcsubseq . . . . .	13
4.8 Polygon Area . . . . .	8		
4.9 Sort By Angle . . . . .	8		
4.10 Voronoi . . . . .	8		
<b>5 Grafos</b>	<b>9</b>		
5.1 Dfs Tree . . . . .	9		
5.2 Dinic . . . . .	9		
5.3 Ford . . . . .	9		

# 1 Algoritmos

## 1.1 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 }
```

## 2 DP

### 2.1 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 }
```

### 2.2 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        ans = max(ans, temp);
16    }
```

### 2.3 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 ED

### 3.1 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            return minimum;
35        }
36
37        ll size() {
38            return in.size() + out.size();
39        }
40    }
```

```
37 };
```

### 3.2 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     prop(l, r, no);
54     if(a<=l and r<=b) return tree[no].val;
55     if(r<a or b<l) return {INF, 0};
56     int m = (l+r)/2;
57     int left = tree[no].l, right = tree[no].r;
58
59     return tree[no].val = merge(query(a, b, l, m,
        left),
60                                query(a, b, m+1, r,
61                                right));
62 }
```

### 3.3 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 <<
                    (j-1))][j-1]);
23             }
24         }
25     }
26
27     int f(int a, int b) {
28         return min(a, b);
29     }
30
31     int query(int l, int r) {
32         int k = logv[r-l+1];
33         return f(st[l][k], st[r - (1 << k) + 1][k]);
34     }
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,
            vector<int>(k+1)));
47
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][y1][j] = f(st[x1][y1][j],
65                     st[x1+delta][y1+delta][j-1]);
66                 }
67             }
68         }
69
70         // so funciona para quadrados
71     }
```

```

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 Geometria

### 4.1 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     T operator*(const point &o) const{ return x * o.x
+ y * o.y; }
19     T 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 int ccw(point a, point b, point e){ // -1=dir; 0=
collinear; 1=esq;
31     T tmp = (b-a) ^ (e-a); // vector from a to b
32     return (tmp > EPS) - (tmp < -EPS);
33 }
34
35 ld norm(point a){ // Modulo
36     return sqrt(a * a);
37 }
38 T norm2(point a){
39     return a * a;
40 }

```

```

41 bool nulo(point a){
42     return (eq(a.x, 0) and eq(a.y, 0));
43 }
44 point rotccw(point p, ld a){
45     // a = PI*a/180; // graus
46     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
+p.x*sin(a)));
47 }
48 point rot90cw(point a) { return point(a.y, -a.x); };
49 point rot90ccw(point a) { return point(-a.y, a.x); };
50
51 ld proj(point a, point b){ // a sobre b
52     return a*b/norm(b);
53 }
54 ld angle(point a, point b){ // em radianos
55     ld ang = a*b / norm(a) / norm(b);
56     return acos(max(min(ang, (ld)1), (ld)-1));
57 }
58 ld angle_vec(point v){
59     // return 180/PI*atan2(v.x, v.y); // graus
60     return atan2(v.x, v.y);
61 }
62 ld order_angle(point a, point b){ // from a to b ccw
(a in front of b)
63     ld aux = angle(a,b)*180/PI;
64     return ((a^b)<=0 ? aux:360-aux);
65 }
66 bool angle_less(point a1, point b1, point a2, point
b2){ // ang(a1,b1) <= ang(a2,b2)
67     point p1((a1^b1), abs((a1^b1)));
68     point p2((a2^b2), abs((a2^b2)));
69     return (p1^p2) <= 0;
70 }
71
72 ld area(vp &p){ // (points sorted)
73     ld ret = 0;
74     for(int i=2;i<(int)p.size();i++)
75         ret += (p[i]-p[0])^(p[i-1]-p[0]);
76     return abs(ret/2);
77 }
78 ld areaT(point &a, point &b, point &c){
79     return abs((b-a)^(c-a))/2.0;
80 }
81
82 point center(vp &A){
83     point c = point();
84     int len = A.size();
85     for(int i=0;i<len;i++)
86         c=c+A[i];
87     return c/len;
88 }
89
90 point forca_mod(point p, ld m){
91     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 point mirror(point m1, point m2, point p){
112     // mirror point p around segment m1m2
113     point seg = m2-m1;
114     ld t0 = ((p-m1)*seg) / (seg*seg);
115     point ort = m1 + seg*t0;
116     point pm = ort-(p-ort);
117     return pm;
118 }
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     bool inside_seg(point p){
157         return (
158             ((p1-p) ^ (p2-p)) == 0 and
159             ((p1-p) * (p2-p)) <= 0
160         );
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.inside_seg(ans[0]))
179         return {};
180     return ans;
181 }
182
183 bool seg_has_inter(line l1, line l2){
184     return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.p2, l2.p2) < 0 and
185            ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.p2, l1.p2) < 0;
186 }
187
188 ld dist_seg(point p, point a, point b){ // point - seg
189     if((p-a)*(b-a) < EPS) return norm(p-a);
190     if((p-b)*(a-b) < EPS) return norm(p-b);
191     return abs((p-a)^(b-a)) / norm(b-a);
192 }
193
194 ld dist_line(point p, line l){ // point - line
195     return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
196 }
197
198 line bisector(point a, point b){
199     point d = (b-a)*2;
200     return line(d.x, d.y, a*a - b*b);
201 }
202
203 line perpendicular(line l, point p){ // passes through p
204     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
205 }
206
207 ///////////////
208 // Circle //
209 ///////////////
210
211 struct circle{
212     point c; T r;
213     circle(): c(0, 0), r(0){}
214     circle(const point o): c(o), r(0){}
215     circle(const point a, const point b){
216         c = (a+b)/2;
217         r = norm(a-c);
218     }
219     circle(const point a, const point b, const point cc){
220         assert(ccw(a, b, cc) != 0);
221         c = inter_line(bisector(a, b), bisector(b, cc))[0];
222         r = norm(a-c);
223     }
224     bool inside(const point &a) const{
225         return norm(a - c) <= r + EPS;
226     }
227 };
228
229 pair<point, point> tangent_points(circle cr, point p)
230 {
231     ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
232     point p1 = rotccw(cr.c-p, -theta);
233     point p2 = rotccw(cr.c-p, theta);
234     assert(d1 >= cr.r);
235     p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
236     p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
237     return {p1, p2};
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         if(!ans.inside(v[i])){
295             ans = circle(v[i]);
296             for(int j=0; j<i; j++){
297                 if(!ans.inside(v[j])){
298                     ans = circle(v[i], v[j]);
299                     for(int k=0; k<j; k++){
300                         if(!ans.inside(v[k])){
301                             ans = circle(v[i], v[j], v[k]);
302                         }
303                     }
304                 }
305             }
306         }
307     }
308     return ans;
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

```

## 4.2 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 }

```

```

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

```

### 4.3 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 }

```

### 4.4 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;
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 }

```

### 4.5 Intersect Polygon

```

1 bool intersect(vector<point> A, vector<point> B) //
2     Ordered ccw
3 {
4     for(auto a: A)
5         if(inside(B, a))
6             return true;
7     for(auto b: B)
8         if(inside(A, b))
9             return true;
10    if(inside(B, center(A)))
11        return true;
12
13    return false;
14 }

```

### 4.6 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));
8 }

```

## 4.7 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 }
```

## 4.8 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;
9
10 area = abs(area);
```

## 4.9 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 }
```

## 4.10 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) {
```



```

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(
78             pts[j], pts[i]));
79         ans[i] = cutPolygon(ans[i], getBisector(
80             pts[i], pts[j]));
81     }
82     return ans;
83 }

```

## 5 Grafos

### 5.1 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 }

```

### 5.2 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.
23                 cap) continue;
24             // v.cap - v.flow
25             < lim

```

```

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

```

### 5.3 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 };

```

## 5.4 Hungarian

```

1 // Hungarian Algorithm
2 //
3 // Assignment problem
4 // Put the edges in the 'a' matrix (negative or
5 // positive)
6 // assignment() returns a pair with the min
7 // assignment,
8 // and the column choosen by each row
9 // assignment() - 0(m^3)
10
11 template<typename T>
12 struct hungarian {
13     int n, m;
14     vector<vector<T>> a;
15     vector<T> u, v;
16     vector<int> p, way;
17     T inf;
18
19     hungarian(int n_, int m_) : n(n_), m(m_), u(m+1),
20 v(m+1), p(m+1), way(m+1) {
21         a = vector<vector<T>>(n, vector<T>(m));
22         inf = numeric_limits<T>::max();

```

```

23     }
24     pair<T, vector<int>> assignment() {
25         for (int i = 1; i <= n; i++) {
26             p[0] = i;
27             int j0 = 0;
28             vector<T> minv(m+1, inf);
29             vector<int> used(m+1, 0);
30             do {
31                 used[j0] = true;
32                 int i0 = p[j0], j1 = -1;
33                 T delta = inf;
34                 for (int j = 1; j <= m; j++) if (!
35 used[j]) {
36                     T cur = a[i0-1][j-1] - u[i0] - v[
37 j];
38                     if (cur < minv[j]) minv[j] = cur,
39 way[j] = j0;
40                     if (minv[j] < delta) delta = minv
41 [j], j1 = j;
42                 }
43                 for (int j = 0; j <= m; j++)
44                     if (used[j]) u[p[j]] += delta, v[
45 j] -= delta;
46                 else minv[j] -= delta;
47                 j0 = j1;
48             } while (p[j0] != 0);
49             do {
50                 int j1 = way[j0];
51                 p[j0] = p[j1];
52                 j0 = j1;
53             } while (j0);
54             vector<int> ans(m);
55             for (int j = 1; j <= n; j++) ans[p[j]-1] = j
56 -1;
57             return make_pair(-v[0], ans);
58         }
59     };

```

## 5.5 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     }

```

## 6 Math

### 6.1 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 }

```

## 6.2 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 }

```

## 6.3 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 }

```

## 6.4 Linear Diophantine Equation

```

1 // Linear Diophantine Equation
2 int gcd(int a, int b, int &x, int &y)
3 {
4     if (a == 0)
5     {
6         x = 0; y = 1;
7         return b;
8     }
9     int x1, y1;
10    int d = gcd(b%a, a, x1, y1);
11    x = y1 - (b / a) * x1;
12    y = x1;
13    return d;
14 }
15
16 bool find_any_solution(int a, int b, int c, int &x0,
17 int &y0, int &g)
18 {
19     g = gcd(abs(a), abs(b), x0, y0);
20     if (c % g)
21         return false;
22
23     x0 *= c / g;
24     y0 *= c / g;
25     if (a < 0) x0 = -x0;
26     if (b < 0) y0 = -y0;
27     return true;
28 }
29 // All solutions
30 // x = x0 + k*b/g
31 // y = y0 - k*a/g

```

## 6.5 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
44        return res;
45    }

```

## 6.6 Totient

```

1 // phi(p^k) = (p^(k-1))*(p-1) com p primo
2 // 0(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 }

```

## 7 Misc

### 7.1 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()

```

## 8 Strings

### 8.1 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 }

```

### 8.2 Edit Distance

```

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

```

### 8.3 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 };

```

## 8.4 Kmp

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

## 8.5 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        currRow = 1 - currRow;
```

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

## 8.6 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 || !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 }
```