



Notebook - Maratona de Programação

[UnB] HatsuneMiku não manda WA

Contents

1 Misc	2	5.4 Convex Hull	7
1.1 Safe Map	2	5.5 Linear Transformation	7
1.2 Bitwise	2	5.6 Voronoi	7
2 DP	2	5.7 Polygon Area	8
2.1 Knapsack	2	5.8 Intersect Polygon	8
2.2 Lis	2	5.9 Sort By Angle	8
2.3 Dp Digitos	2	5.10 2d	8
3 ED	3	6 Grafos	11
3.1 Prefixsum2d	3	6.1 Dfs Tree	11
3.2 Sparse Table	3	6.2 Kosaraju	11
3.3 Minqueue	3	6.3 Dinic	11
3.4 Segtree Implicita Lazy	4	6.4 Hungarian	12
3.5 Delta Encoding	4	6.5 Lca	12
4 Strings	4	6.6 Ford	13
4.1 Z Func	4	7 Algoritmos	13
4.2 Edit Distance	4	7.1 Ternary Search	13
4.3 Lcsubseq	4	8 Math	13
4.4 Kmp	5	8.1 Totient	13
4.5 Hash	5	8.2 Inverso Mult	14
4.6 Aho Corasick	5	8.3 Matrix Exponentiation	14
4.7 Lcs	5	8.4 Division Trick	14
5 Geometria	6	8.5 Crivo	14
5.1 Mindistpair	6	8.6 Bigmod	14
5.2 Inside Polygon	6	8.7 Linear Diophantine Equation	14
5.3 3d	6		

1 Misc

1.1 Safe Map

```
1 struct custom_hash {
2     static uint64_t splitmix64(uint64_t x) {
3         // http://xorshift.di.unimi.it/splitmix64.c
4         x += 0x9e3779b97f4a7c15;
5         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
6         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
7         return x ^ (x >> 31);
8     }
9
10    size_t operator()(uint64_t x) const {
11        static const uint64_t FIXED_RANDOM = chrono::
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
```

1.2 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()
29
```

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[
15            i]);
16
17        ans = max(ans, temp);
18    }
19
```

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 }
33
```

2.3 Dp Digitos

```
1 // dp de quantidade de numeros <= r com ate qt
2 // digitos diferentes de 0
3 ll dp(int idx, string& r, bool menor, int qt, vector<
4 vector<vi>&& tab) {
5     if(qt > 3) return 0;
6     if(idx >= r.size()) {
7         return 1;
8     }
9
10    if(tab[idx][menor][qt] != -1)
11        return tab[idx][menor][qt];
12
13    ll res = 0;
14    for(int i = 0; i <= 9; i++) {
15        if(menor or i <= r[idx]-'0') {
16            res += dp(idx+1, r, menor or i < (r[idx]-
17            '0'), qt+(i>0), tab);
18        }
19    }
20
21    return tab[idx][menor][qt] = res;
22 }
23
```

3 ED

3.1 Prefixsum2d

```
1 ll find_sum(vector<vi> &mat, int x1, int y1, int x2,
2   int y2){
3   // superior-esq(x1,y1) (x2,y2)inferior-dir
4   return mat[x2][y2]-mat[x2][y1-1]-mat[x1-1][y2]+
5   mat[x1-1][y1-1];
6 }
7
8 int main(){
9   for(int i=1;i<=n;i++){
10    for(int j=1;j<=m;j++){
11      mat[i][j]=mat[i-1][j]+mat[i][j-1]-mat[i-1][j-1];
12    }
13  }
```

3.2 Sparse Table

```
1 int logv[N+1];
2 void make_log() {
3   logv[1] = 0; // pre-computar 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,
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+delta][y1][j] = f(st[x1][y1][j],
61           st[x1+delta][y1][j-1]);
62           st[x1][y1+delta][j] = f(st[x1][y1][j],
63           st[x1][y1+delta][j-1]);
64           st[x1+delta][y1+delta][j] = f(st[x1][y1][j],
65           st[x1+delta][y1+delta][j-1]);
66         }
67       }
68     }
69
70     // so funciona para quadrados
71     int query(int x1, int y1, int x2, int y2) {
72       assert(x2-x1+1 == y2-y1+1);
73       int k = logv[x2-x1+1];
74       int delta = (1 << k);
75
76       int res = st[x1][y1][k];
77       res = f(res, st[x2 - delta+1][y1][k]);
78       res = f(res, st[x1][y2 - delta+1][k]);
79       res = f(res, st[x2 - delta+1][y2 - delta+1][k]);
80     }
81     return res;
82 }
83
84 int f(int a, int b) {
85   return a | b;
86 }
87
88 int minn() {
89   ll minimum = LLINF;
90   if(in.empty() || out.empty())
```

3.3 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     }
21     ll res = out.top().ff;
22     out.pop();
23     return res;
24 }
25
26 ll minn() {
27   ll minimum = LLINF;
28   if(in.empty() || out.empty())
```

```

27         minimum = in.empty() ? (ll)out.top().ss :
28         (ll)in.top().ss;
29         else
30             minimum = min((ll)in.top().ss, (ll)out.
31             top().ss);
32     }
33
34     ll size() {
35         return in.size() + out.size();
36     }
37 };

```

3.4 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;

```

```

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,
60     left),
61     query(a, b, m+1, r,
62     right));
63 }

```

3.5 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 - 1], r - i + 1));
7         while (i + z[i] < n and s[z[i]] == s[i + z[i]
8         ])) {
9             l = i; r = i + z[i]; z[i]++;
10        }
11    }
12    return z;
13 }

```

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 Lcsubseq

```

1 // Longest Common Subsequence
2 string lcs(string x, string y){
3     int n = x.size(), m = y.size();
4     vector<vi> dp(n+1, vi(m+1, 0));
5
6     for(int i=0;i<=n;i++){

```

```

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

```

4.4 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+1] != c)) // leader doesn't match
5         u = neighbor[u];
6     return p[u+1] == c ? u+1 : u;
7 }
8 void build() {
9     neighbor[0] = -1; // -1 is the leftmost state
10    for (int i = 1; i < (int)p.size(); i++)
11        neighbor[i] = walk(neighbor[i-1], p[i]);
12 }

```

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(n) {
12        for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1)
13        % MOD;
14        for (int i=0;i<n;i++)
15            h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
16        for (int i=n-1;i>=0;i--)
17            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P)
18            % MOD;
19    }
20    int query(int l, int r) {
21        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]:0) % MOD :
22        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-1
24        +1] % MOD : 0));
25        return hash < 0 ? hash + MOD : hash;
26    }
27 };

```

4.6 Aho Corasick

```

1 // https://github.com/joseleite19/icpc-notebook/blob/
2 // master/code/string/aho_corasick.cpp
3 const int A = 26;
4 int to[N][A];
5 int ne = 2, fail[N], term[N];
6 void add_string(string str, int id){
7     int p = 1;
8     for(auto c: str){
9         int ch = c - 'a'; // !
10        if(!to[p][ch]) to[p][ch] = ne++;
11        p = to[p][ch];
12    }
13    term[p]++;
14 }
15 void init(){
16     for(int i = 0; i < ne; i++) fail[i] = 1;
17     queue<int> q; q.push(1);
18     int u, v;
19     while(!q.empty()){
20         u = q.front(); q.pop();
21         for(int i = 0; i < A; i++){
22             if(to[u][i]){
23                 v = to[u][i]; q.push(v);
24                 if(u != 1){
25                     fail[v] = to[ fail[u] ][i];
26                     term[v] += term[ fail[v] ];
27                 }
28             }
29             else if(u != 1) to[u][i] = to[ fail[u] ][i];
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    }

```

```

24         currRow = 1 - currRow;
25     }
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        s.insert(point(vet[i].y, vet[i].x));
29    }
30    return best_dist;
31 }

```

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
11 bool inside(vp &p, point e){ // ccw
12     int l=2, r=(int)p.size()-1;
13     while(l<r){
14         int mid = (l+r)/2;
15         if(ccw(p[0], p[mid], e) == 1)
16             l=mid+1;
17         else{
18             r=mid;
19         }
20     }
21     // bordo
22     // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)
23     ==0) return false;

```

```

22 // if(r==2 and ccw(p[0], p[1], e)==0) return
23 false;
24 // if(ccw(p[r], p[r-1], e)==0) return false;
25 return insideT(p[0], p[r-1], p[r], e);
26 }
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

```

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

```

```

38
39 ld norm(point a) { // Modulo
40     return sqrt(a * a);
41 }
42 cod norm2(point a) {
43     return a * a;
44 }
45 bool nulo(point a) {
46     return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0))
47 }
48 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)
67     , p2(p2), p3(p3) {
68         point aux = (p1-p3)^(p2-p3);
69         a = aux.x; b = aux.y; c = aux.z;
70         d = -a*p1.x - b*p1.y - c*p1.z;
71     }
72     plane(point p, point normal) {
73         normal = normilize(normal);
74         a = normal.x; b = normal.y; c = normal.z;
75         d = -(p*normal);
76     }
77     // ax+by+cz+d = 0;
78     cod eval(point &p) {
79         return a*p.x + b*p.y + c*p.z + d;
80     }
81 };
82
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
88 point rotate(point v, point k, ld theta) {
89     // Rotaciona o vetor v theta graus em torno do
90     eixo k
91     // theta *= PI/180; // graus
92     return (
93         v*cos(theta)) +
94         ((k^v)*sin(theta)) +
95         (k*(k^v))*(1-cos(theta)
96 );
97 }
98 // 3d line inter / mindistance
99 cod d(point p1, point p2, point p3, point p4) {
100     return (p2-p1) * (p4-p3);
101 }
102
103 vector<point> inter3d(point p1, point p2, point p3,
104     point p4) {
105     cod mua = ( d(p1, p3, p4, p3) * d(p4, p3, p2, p1)
106 - d(p1, p3, p2, p1) * d(p4, p3, p4, p3) )
107     / ( d(p2, p1, p2, p1) * d(p4, p3, p4, p3)
108 - 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) {

```



```

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

```

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*
*pontos[i].y;
5 }
6 area += pontos[n-1].x*pontos[0].y - pontos[0].x*
pontos[n-1].y;
7

```

```
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     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         return os << "(" << p.x << "," << p.y << ")";
28     };
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
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 // Line //
123 ///////////////
124
125 struct line{
126     point p1, p2;
127     T a, b, c; // ax+by+c = 0;
128     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
129     line(point p1=0, point p2=0): p1(p1), p2(p2){
130         a = p1.y - p2.y;
131         b = p2.x - p1.x;
132         c = p1 ^ p2;
133     }
134     line(T a=0, T b=0, T c=0): a(a), b(b), c(c){
135         // Gera os pontos p1 p2 dados os coeficientes
136         // isso aqui eh um lixo mas quebra um galho
137         kkkkkk
138         if(b==0){
139             p1 = point(1, -c/a);
140             p2 = point(0, -c/a);
141         }else{
142             p1 = point(1, (-c-a*1)/b);
143             p2 = point(0, -c/b);
144         }
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 };

```

```

165 // be careful with precision error
166 vp inter_line(line l1, line l2){
167     ld det = l1.a*l2.b - l1.b*l2.a;
168     if(det==0) return {};
169     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
170     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
171     return {point(x, y)};
172 }
173
174 // segments not collinear
175 vp inter_seg(line l1, line l2){
176     vp ans = inter_line(l1, l2);
177     if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.
178         inside_seg(ans[0]))
179         return {};
180     return ans;
181 }
182 bool seg_has_inter(line l1, line l2){
183     return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.
184         p2, l2.p2) < 0 and
185         ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.
186         p2, l1.p2) < 0;
187 }
188 ld dist_seg(point p, point a, point b){ // point -
189     seg
190     if((p-a)*(b-a) < EPS) return norm(p-a);
191     if((p-b)*(a-b) < EPS) return norm(p-b);
192     return abs((p-a)^(b-a)) / norm(b-a);
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 line bisector(point a, point b){
198     point d = (b-a)*2;
199     return line(d.x, d.y, a*a - b*b);
200 }
201
202 line perpendicular(line l, point p){ // passes
203     through p
204     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
205 }
206
207 // Circle //
208
209 struct circle{
210     point c; T r;
211     circle() : c(0, 0), r(0){}
212     circle(const point o) : c(o), r(0){}
213     circle(const point a, const point b){
214         c = (a+b)/2;
215         r = norm(a-c);
216     }
217     circle(const point a, const point b, const point
218         cc){
219         assert(ccw(a, b, cc) != 0);
220         c = inter_line(bisector(a, b), bisector(b, cc
221             ))[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++) if(!ans.inside(v[i])){
294         ans = circle(v[i]);
295         for(int j=0;j<i;j++) if(!ans.inside(v[j])){
296             ans = circle(v[i], v[j]);
297             for(int k=0;k<j;k++) if(!ans.inside(v[k]))
298                 ans = circle(v[i], v[j], v[k]);
299         }
300     }
301 }

```

```

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

```

6.2 Kosaraju

```

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

```

6.3 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; // v.cap - v.flow
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     }
52     ll flow(int source, int sink) {
53         reset_flow();
54         ll ans = 0;
55         //for(lim = (1LL << 62); lim >= 1; lim /= 2)
56         while(bfs(source, sink))
57             ans += run(source, sink, LLINF);
58         return ans;
59     }
60     void addEdge(int u, int v, ll c, ll rc) {
61         Edge e = {u, v, 0, c};
62         edge.pb(e);
63         g[u].push_back(ne++);
64
65         e = {v, u, 0, rc};
66         edge.pb(e);
67         g[v].push_back(ne++);
68     }
69     void reset_flow() {
70         for(int i = 0; i < ne; i++)
71             edge[i].flow = 0;
72         memset(lvl, 0, sizeof(lvl));
73         memset(vis, 0, sizeof(vis));
74         memset(qu, 0, sizeof(qu));
75         memset(px, 0, sizeof(px));
76         qt = 0; pass = 0;

```

```

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
            vis[to] < pass and cap>0) {
81             cuts.pb({from, to});
82         }
83     }
84     return cuts;
85 }
86 };

```

6.4 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 choosen 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),
        v(m+1), p(m+1), way(m+1) {
18         a = vector<vector<T>>(n, vector<T>(m));
19         inf = numeric_limits<T>::max();
20     }
21     pair<T, vector<int>> assignment() {
22         for (int i = 1; i <= n; i++) {
23             p[0] = i;
24             int j0 = 0;
25             vector<T> minv(m+1, inf);
26             vector<int> used(m+1, 0);
27             do {
28                 used[j0] = true;
29                 int i0 = p[j0], j1 = -1;
30                 T delta = inf;
31                 for (int j = 1; j <= m; j++) if (!
                    used[j]) {
32                     T cur = a[i0-1][j-1] - u[i0] - v[
                        j];
33                     if (cur < minv[j]) minv[j] = cur,
34                     if (minv[j] < delta) delta = minv
35                     [j], j1 = j;
36                 }
37                 for (int j = 0; j <= m; j++)
38                     if (used[j]) u[p[j]] += delta, v[
39                     j] -= delta;
40                 else minv[j] -= delta;
41                 j0 = j1;
42             } while (p[j0] != 0);
43             do {
44                 int j1 = way[j0];
45                 p[j0] = p[j1];
46                 j0 = j1;
47             } while (j0);
48             vector<int> ans(m);
49             for (int j = 1; j <= n; j++) ans[p[j]-1] = j
50             -1;
51             return make_pair(-v[0], ans);
52         }
53     }
54 }

```

```

51 };

```

6.5 Lca

```

1 const int LOG = 22;
2 vector<vector<int>> g(N);
3 int t, n;
4 vector<int> in(N), height(N);
5 vector<vector<int>> up(LOG, vector<int>(N));
6 void dfs(int u, int h=0, int p=-1) {
7     up[0][u] = p;
8     in[u] = t++;
9     height[u] = h;
10    for (auto v: g[u]) if (v != p) dfs(v, h+1, u);
11 }
12
13 void blift() {
14     up[0][0] = 0;
15     for (int j=1; j<LOG; j++) {
16         for (int i=0; i<n; i++) {
17             up[j][i] = up[j-1][up[j-1][i]];
18         }
19     }
20 }
21
22 int lca(int u, int v) {
23     if (u == v) return u;
24     if (in[u] < in[v]) swap(u, v);
25     for (int i=LOG-1; i>=0; i--) {
26         int u2 = up[i][u];
27         if (in[u2] > in[v])
28             u = u2;
29     }
30     return up[0][u];
31 }
32
33 t = 0;
34 dfs(0);
35 blift();
36
37 // lca O(1)
38
39 template<typename T> struct rmq {
40     vector<T> v;
41     int n; static const int b = 30;
42     vector<int> mask, t;
43
44     int op(int x, int y) { return v[x] < v[y] ? x : y
45     ; }
46     int msb(int x) { return __builtin_clz(1)-
47     __builtin_clz(x); }
48     rmq() {}
49     rmq(const vector<T>& v_) : v(v_), n(v.size()),
50     mask(n), t(n) {
51         for (int i = 0, at = 0; i < n; mask[i++] = at
52         |= 1) {
53             at = (at<<1)&((1<<b)-1);
54             while (at and op(i, i-msb(at&-at)) == i)
55                 at ^= at&-at;
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     }
64     int small(int r, int sz = b) { return r-msb(mask[
65     r]&((1<<sz)-1)); }
66     T query(int l, int r) {
67         if (r-l+1 <= b) return small(r, r-l+1);
68         int ans = op(small(l+b-1), small(r));
69         int x = l/b+1, y = r/b-1;
70     }
71 }

```

```

61     if (x <= y) {
62         int j = msb(y-x+1);
63         ans = op(ans, op(t[n/b*j+x], t[n/b*j+y
        -(1<<j)+1]));
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 }

```

6.6 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  // 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 }

```

8.2 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 }

```

8.3 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
multiplicar
23         vector<vl> res(r, vl(o.c, 0));
24
25         for(int i = 0; i < r; i++) {
26             for(int k = 0; k < c; k++) {
27                 for(int j = 0; j < o.c; j++) {
28                     res[i][j] = (res[i][j] + m[i][k]*
o.m[k][j]) % MOD;
29                 }
30             }
31         }
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); //
identidade
39     Matrix res = fexp(b, e/2, n);
40     res = (res * res);
41     if(e%2) res = (res * b);
42
43     return res;

```

```

44 }

```

8.4 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 }

```

8.5 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;

```

8.6 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 }

```

8.7 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,
int &y0, int &g)
17 {
18     g = gcd(abs(a), abs(b), x0, y0);
19     if (c % g)
20         return false;
21
22     x0 *= c / g;
23     y0 *= c / g;
24     if (a < 0) x0 = -x0;
25     if (b < 0) y0 = -y0;
26     return true;
27 }
28
29 // All solutions
30 // x = x0 + k*b/g
31 // y = y0 - k*a/g

```