Notebook - Maratona de Programação

Tiago de Souza Fernandes

Sumário

1 Algoritmos

1.1 Mochila

```
int val[MAXN], peso[MAXN], dp[MAXN][MAXS]

int knapsack(int N, int M) // Objetos | Peso max
{
    for(i=0;i<=N;i++)
    {
        for(j=0;j<=M;j++)
        {
            if (i==0 || j==0)
                dp[i][j] = 0;
            else if (peso[i-1] <= j)
                dp[i][j] = max(val[i-1]+dp[i-1][j-
peso[i-1]],  dp[i-1][j]);
            else
                dp[i][j] = dp[i-1][j];
        }
    }
    return dp[N][M];
}
```

1.2 Kadane-DP

```
// Largest Sum Contiguous Subarray

int maxSubArraySum(vector<int> a)
{
    int size = a.size();
    int max_so_far = a[0];
    int curr_max = a[0];

    for (int i=1;i<size;i++)
    {
        curr_max = max(a[i], curr_max+a[i]);
        max_so_far = max(max_so_far, curr_max);
    }
    return max_so_far;
}
```

1.3 Iterative-BS

```
int main()
{
    int l=1, r=N;
    int res=-1;

    while(l <= r)
    {
        int m = (l + r)/2;
        if(!ver(m))
        {
            l = m+1;
        }
        else
        {
            res = m;
            r = m-1;
        }
    }
    cout << res << endl;

    return 0;
}
```

2 Grafos

2.1 BFS

```
//BFS (Breadth First Search) O(V+A)

vector<vector<int>> adj;  // adjacency list
    representation
int n; // number of nodes
int s; // source vertex

queue<int> q;
vector<int> d(n, INF);
d[s]=0;

q.push(s);
used[s] = true;
while (!q.empty()) {
    int v = q.front();
    q.pop();
    for (int u : adj[v]) {
        if (d[u] > d[v] + 1) {
            q.push(u);
            d[u] = d[v] + 1;
        }
    }
}
```

2.2 Find-bridges

```
#define vi vector<int>

vector< vector<int> > grafo;
vector<bool> visited;
vi t, low;
int timer=0;

void find_bridges(int v, int p=-1){
    visited[v] = true;
    t[v] = low[v] = timer++;
    for(int i=0;i<(int)grafo[v].size();i++){
        int vert = grafo[v][i];
        if(vert == p)
            continue;
        if(visited[vert])
            low[v] = min(low[v], t[vert]);
        else{
            find_bridges(vert, v);
            low[v] = min(low[v], low[vert]);
            if(low[to] > t[v])
                IS_BRIDGE(v, vert);
        }
    }
}

int main()
{
    timer = 0;
    visited.assign(N+1, false);
    t.assign(N+1, 0);
    low.assign(N+1, 0);

    for(int i=0;i<N;i++)
        if(!visited[i])
            find_bridges(1);

    return 0;
}
```

2.3 Dijkstra

```
// Dijkstra - Shortest Path

#define pii pair<int, int>
#define vi vector<int>
#define vii vector< pair<int,int> >
```

```
6  #define INF 0x3f3f3f3f
7
8  vector<vii> grafo(N+1, vii());
9  vi distancia(N+1, INF);
10 priority_queue< pii, vii, greater<pii> > fila;
11
12 void dijkstra(int k){
13     int dist, vert, aux;
14     distancia[k]=0;
15
16     fila.push(mp(k, 0));
17
18     while(!fila.empty()){
19         aux=fila.top().f;
20         fila.pop();
21
22         for(auto v: grafo[aux]){
23             vert=v.f;
24             dist=v.s;
25             if(distancia[vert]>distancia[aux]+dist){
26                 distancia[vert]=distancia[aux]+dist;
27                 fila.push(mp(vert, distancia[vert]));
28             }
29         }
30     }
31 }
32
33 int main()
34 {
35     for(int i=0; i<M; i++){
36         cin >> a >> b >> p;
37         grafo[a].pb(mp(b, p));
38         grafo[b].pb(mp(a, p));
39     }
40 }
```

2.4 LCA

```
1  const int K = 100;
2  int logv[MAX+1];
3  int st[MAX][K];
4  vector<vi> grafo(200010, vi());
5
6  void make(){
7      logv[1] = 0; // pre-computar tabela de log
8      for (int i = 2; i <= MAX; i++)
9          logv[i] = logv[i/2] + 1;
10 }
11
12 void precompute(int N, int array[]) { //
13   for (int i = 0; i < N; i++)
14       st[i][0] = array[i];
15
16   int k = logv[N];
17   for (int j = 1; j <= k; j++)
18       for (int i = 0; i + (1 << j) <= N; i++)
19           st[i][j] = min(st[i][j-1], st[i + (1 << (j
    - 1))][j - 1]);
20 }
21
22 int query(int L, int R) {
23     int j = logv[R - L + 1];
24     int minimum = min(st[L][j], st[R - (1 << j) + 1][
    j]);
25
26     return minimum;
27 }
28
29 int start[MAX+1], dfs_time;
30 int tour[2*MAX+1], id[2*MAX+1];
31
32 void dfs(int u, int pai=-1){
```

```
33     start[u] = dfs_time;
34     id[dfs_time] = u;
35     tour[dfs_time++] = start[u];
36     for(int v : grafo[u]){
37         if(v==pai)
38             continue;
39         dfs(v, u);
40         id[dfs_time] = u;
41         tour[dfs_time++] = start[u];
42     }
43 }
44
45 int LCA(int u, int v)
46 {
47     if(start[u] > start[v])
48         swap(u, v);
49     return id[query(start[u], start[v])];
50 }
51
52 int main()
53 {
54     int N, k, a, b;
55     cin >> N;
56
57     for(int i=0;i<N-1;i++)
58     {
59         cin >> a >> b;
60         grafo[a].pb(b);
61         grafo[b].pb(a);
62     }
63     dfs(1);
64
65     make();
66     precompute(2*N, tour);
67
68
69     cin >> k;
70     for(int i=0;i<k;i++)
71     {
72         cin >> a >> b;
73         cout << LCA(a, b) << endl;
74     }
75
76     return 0;
77 }
```

2.5 Floyd-Warshall

```
1  // Floyd Warshall
2
3  int dist[MAX][MAX];
4
5  void Floydwarshall()
6  {
7      for(int k = 1;k <= n;k++)
8          for(int i = 1;i <= n;i++)
9              for(int j = 1;j <= n;j++)
10                 dist[i][j] = min(dist[i][j], dist[i][
    k] + dist[k][j]);
11 }
```

2.6 Kruskal

```
1  // deve-se ter dsu codada com as funcoes make_set,
    find_set e union_sets
2  struct Edge {
3      int u, v, weight;
4      bool operator<(Edge const& other) {
5          return weight < other.weight;
6      }
7  };
8
```

```
9   int n;
10  vector<Edge> edges;
11
12  int cost = 0;
13  vector<Edge> result;
14  for (int i = 0; i < n; i++)
15      make_set(i);
16
17  sort(edges.begin(), edges.end());
18
19  for (Edge e : edges) {
20      if (find_set(e.u) != find_set(e.v)) {
21          cost += e.weight;
22          result.push_back(e); // vector com as arestas
          da MST
23          union_sets(e.u, e.v);
24      }
25  }
```

2.7 DFS

```
1   //DFS (Depth First Search) O(V+A)
2
3   void DFS(int x){
4       for(int i=0; i<(int)vizinhos[x].size(); i++){
5           int v = vizinhos[x][i];
6           if(componente[v] == -1){
7               componente[v] = componente[x];
8               DFS(v);
9           }
10      }
11  }
```

2.8 Kosaraju

```
1   // KOSARAJU - O(V+E) - encontra componentes
        fortemente conexos
2   // g -> grafo, gt -> grafo tempo
3   // vis -> visitado, cor -> componente fortemente
        conexo ordenado topologicamente
4   vector<int> g[N], gt[N], S; int vis[N], cor[N];
5   void dfs(int u){
6       vis[u] = 1; for(int v : g[u]) if(!vis[v]) dfs(v);
7       S.push_back(u);
8   }
9   void dfst(int u, int e){
10      cor[u] = e;
11      for(int v : gt[u]) if(!cor[v]) dfst(v, e);
12  }
13  void kosaraju(){
14      for(int i = 1; i <= n; i++) if(!vis[i]) dfs(i);
15      for(int i = 1; i <= n; i++) for(int j : g[i])
16          gt[j].push_back(i);
17      int e = 0; reverse(S.begin(), S.end());
18      for(int u : S) if(!cor[u]) dfst(u, ++e);
19  }
```

2.9 Centroid

```
1   vi g[MAX];
2   int size[MAX];
3   bool erased[MAX]; // vetor dos vertices apagados na
        decomp.
4
5   int sz(int u, int p) {
6     int s = 1;
7     for(auto prox : g[u]) {
8       if(prox != p and !erased[prox])
9         s += sz(prox, u);
10    }
11    return size[u] = s;
12  }
```

```
13
14  int centroid(int u, int p, int n) {
15    // chamar funcao sz antes, n = size[u]
16    for(auto prox : g[u]) {
17      if(prox != p and !erased[prox]) {
18        if(size[prox] > n/2) {
19          return centroid(prox, u, n);
20        }
21      }
22    }
23    return u;
24  }
```

2.10 Prim

```
1   // Prim Algorithm
2   #define MAXN 10100
3   #define INFINITO 999999999
4
5   int n, m;
6   int distancia[MAXN];
7   int processado[MAXN];
8   vector<pii> vizinhos[MAXN];
9
10  int Prim()
11  {
12      for(int i = 2;i <= n;i++) distancia[i] = INFINITO
        ;
13      distancia[1] = 0;
14
15      priority_queue< pii, vector<pii>, greater<pii> >
        fila;
16      fila.push( pii(distancia[1], 1) );
17
18      while(1){
19          int davez = -1;
20
21          while(!fila.empty()){
22              int atual = fila.top().second;
23              fila.pop();
24
25              if(!processado[atual]){
26                  davez = atual;
27                  break;
28              }
29          }
30
31          if(davez == -1)
32              break;
33
34          processado[davez] = true;
35
36          for(int i = 0;i < (int)vizinhos[davez].size()
        ;i++){
37              int dist  = vizinhos[davez][i].first;
38              int atual = vizinhos[davez][i].second;
39
40              if( distancia[atual] > dist && !
        processado[atual])
41              {
42                  distancia[atual] = dist;
43                  fila.push( pii(distancia[atual],
        atual) );
44              }
45          }
46      }
47
48      int custo_arvore = 0;
49      for(int i = 1;i <= n;i++)
50          custo_arvore += distancia[i];
51
52      return custo_arvore;
53  }
```

```cpp
54
55  int main(){
56
57      cin >> n >> m;
58
59      for(int i = 1;i <= m;i++){
60
61          int x, y, tempo;
62          cin >> x >> y >> tempo;
63
64          vizinhos[x].pb( pii(tempo, y) );
65          vizinhos[y].pb( pii(tempo, x) );
66      }
67
68      cout << Prim() << endl;
69
70      return 0;
71  }
```

3 Geometria

3.1 Convex-polygon-intersection

```cpp
1  cod cross(point a, point b){
2      return a.x*b.y - a.y*b.x;
3  }
4
5  int ccw(point a, point b, point e) //-1=esq; 0=
       collinear; 1=dir;
6  {
7      cod tmp = cross(b-a, e-a); // from a to b
8      return (tmp > EPS) - (tmp < -EPS);
9  }
10
11 int n=4;
12 vector<point> A, B;
13
14 bool intersect()
15 {
16     A.pb(A[0]);
17     B.pb(B[0]);
18     point centerA=point();
19     point centerB=point();
20
21     for(int i=0;i<n;i++)
22     {
23         centerA=centerA+A[i];
24         centerB=centerB+B[i];
25     }
26     centerA=centerA/n;
27     centerB=centerB/n;
28     A.pb(centerA);
29     B.pb(centerB);
30
31
32     bool d, e;
33
34     for(int j=1;j<n+2;j++)
35     {
36         d=false, e=false;
37         for(int i=0;i<n;i++)
38         {
39             int t = esq(A[i], A[i+1], B[j]);
40             if(t==1) e=true;
41             else if(t==-1) d=true;
42         }
43
44         if(!(e and d))
45             return 1;
46     }
47
48
```

```cpp
49      for(int j=1;j<n+2;j++)
50      {
51          d=false, e=false;
52          for(int i=0;i<n;i++)
53          {
54              int t = esq(B[i], B[i+1], A[j]);
55              if(t==1) e=true;
56              else if(t==-1) d=true;
57          }
58
59          if(!(e and d))
60              return 1;
61      }
62
63      return 0;
64  }
```

3.2 Angle-adjacent-vertices-regular-polygon

$a = 180/N$

3.3 Inter-Retas

```cpp
1  // Intersection between lines
2
3  typedef struct
4  {
5      int x, y;
6  } pnt;
7
8  bool collinear(pnt p, pnt q, pnt r)
9  {
10     if(q.x<=max(p.x,r.x) && q.x>=min(p.x,r.x) && q.y
        <=max(p.y,r.y) && q.y>=min(p.y,r.y))
11         return true;
12
13     return false;
14 }
15
16 int orientation(pnt p, pnt q, pnt r)
17 {
18     int val=(q.y-p.y)*(r.x-q.x)-(q.x-p.x)*(r.y-q.y);
19
20     if(val==0)
21         return 0;
22     else if(val>0)
23         return 1;
24     else
25         return 2;
26 }
27
28 bool intersect(pnt p1, pnt q1, pnt p2, pnt q2)
29 {
30     int o1 = orientation(p1, q1, p2);
31     int o2 = orientation(p1, q1, q2);
32     int o3 = orientation(p2, q2, p1);
33     int o4 = orientation(p2, q2, q1);
34
35     if(o1!=o2 and o3!=o4)
36         return true;
37
38     if(o1==0 && collinear(p1, p2, q1))
39         return true;
40
41     if(o2==0 && collinear(p1, q2, q1))
42         return true;
43
44     if(o3==0 && collinear(p2, p1, q2))
45         return true;
46
47     if(o4==0 && collinear(p2, q1, q2))
48         return true;
```

```
49
50      return false;
51
52 }
```

3.4 Pick's-theorem

- The area of a polygon with integer coordinates: $A = i + \frac{b}{2} - 1$

- i is the number os points inside the polygon;

- b is the number of points on the boundry;

- $2A$ is necessarily an integer value.

3.5 linesweep

```
1 typedef pair<double, double> dd;
2
3 bool compare(dd a, dd b){
4     return a.st < b.st;
5 }
6
7 double closest(dd v[], int n){
8     sort(v, v+n, compare);
9     double best = FLT_MAX;
10    set<dd> box;
11    box.insert(v[0]);
12    int left = 0;
13    rep2(i, 1, n){
14        while(left < i && v[i].st-v[left].st > best){
15            box.erase(v[left++]);
16        }
17        for(set<dd>::iterator it = box.lower_bound(mp
   (v[i].nd-best, v[i].st-best));it!=box.end() && v[
   i].nd + best >= it->nd;it++){
18            best = min(best, sqrt(pow(v[i].nd - it->
   nd, 2.0) + pow(v[i].st - it->st, 2.0)));
19        }
20        box.insert(v[i]);
21    }
22    return best;
23 }
```

3.6 Sort-by-Angle

```
1 int quarter(point a)
2 {
3     if(a.x>=0 and a.y>0) return 0;
4     if(a.x<0 and a.y>=0) return 1;
5     if(a.x<=0 and a.y<0) return 2;
6     return 3;
7 }
8
9 bool comp(point a, point b)
10 {
11    int qa = quarter(a);
12    int qb = quarter(b);
13    if(qa==qb)
14        return cross(a,b)>0;
15    else
16        return quarter(a)<quarter(b);
17 }
```

3.7 Cross-properties

- It equals zero if the vectors **a** and **b** are collinear (coplanar in triple product).

- It is negative if the rotation from the first to the second vector is clockwise and positive otherwise.

3.8 Inter-Retangulos

```
1 typedef struct
2 {
3     int x, y;
4 } Point;
5
6 bool doOverlap(Point l1, Point r1, Point l2, Point r2
   )
7 {
8     if (l1.x>r2.x or l2.x>r1.x or l1.y<r2.y or l2.y<
   r1.y)
9         return false;
10    return true;
11 }
```

3.9 Heron

$$A_{triangulo} = \sqrt{s(s-a)(s-b)(s-c)}$$
$$A_{quadrilatero} = \sqrt{(s-a)(s-b)(s-c)(s-d)}$$

3.10 3D

```
1 typedef ld cod;
2
3 bool eq(cod a, cod b){ return abs(a - b) <= EPS; }
4
5 struct point
6 {
7     cod x, y, z;
8     point(cod x=0, cod y=0, cod z=0): x(x), y(y), z(z
   ){}
9
10    point operator+(const point &o) const{
11        return {x+o.x, y+o.y, z+o.z};
12    }
13    point operator-(const point &o) const{
14        return {x-o.x, y-o.y, z-o.z};
15    }
16    point operator*(cod t) const{
17        return {x*t, y*t, z*t};
18    }
19    point operator/(cod t) const{
20        return {x/t, y/t, z/t};
21    }
22    bool operator==(const point &o) const{
23        return eq(x, o.x) and eq(y, o.y) and eq(z, o.
   z);
24    }
25 };
26
27 // Produto Escalar
28 cod dot(point a, point b){
29     return a.x*b.x + a.y*b.y + a.z*b.z;
30 }
31
32 // Produto Vetorial
33 point cross(point a, point b){
34     return point(a.y*b.z - a.z*b.y,
35                  a.z*b.x - a.x*b.z,
36                  a.x*b.y - a.y*b.x);
37 }
38
39 ld abs(point a){ // Modulo
40     return sqrt(dot(a, a));
41 }
42 ld proj(point a, point b){ // a sobre b
43     return dot(a, b)/abs(b);
44 }
```

```
45 ld angle(point a, point b){ // em radianos
46     return acos(dot(a, b) / abs(a) / abs(b));
47 }
48
49 cod triple(point a, point b, point c){
50     return dot(a, cross(b, c)); // Area do
       paralelepipedo
51 }
```

3.11 Dot-properties

- Length of $\mathbf{a}$: $|\mathbf{a}| = \sqrt{\mathbf{a} \cdot \mathbf{a}}$.

- Projection of $\mathbf{a}$ onto $\mathbf{b}$: $\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{b}|}$.

- Angle between vectors: $\arccos\left(\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| \cdot |\mathbf{b}|}\right)$.

- The dot product is positive if the angle between them is acute, negative if it is obtuse and it equals zero if they are orthogonal, i.e. they form a right angle.

3.12 Minkowski-Sum

```
1 vector<point> mk(const vector<point> &a,const vector<
    point> &b){
2     int i = 0, j = 0;
3     for(int k = 0; k < (int)a.size(); k++)if(a[k]<a[i
    ])
4         i = k;
5     for(int k = 0; k < (int)b.size(); k++)if(b[k]<b[j
    ])
6         j = k;
7
8     vector<point> c;
9     c.reserve(a.size() + b.size());
10    for(int k = 0; k < int(a.size()+b.size()); k++){
11        point pt{a[i] + b[j]};
12        if((int)c.size() >= 2 and !ccw(c[c.size()-2],
    c.back(), pt))
13            c.pop_back();
14        c.pb(pt);
15        int q = i+1, w = j+1;
16        if(q == int(a.size())) q = 0;
17        if(w == int(b.size())) w = 0;
18        if(ccw(c.back(), a[i]+b[w], a[q]+b[j]) < 0) i
     = q;
19        else j = w;
20    }
21
22    if(!ccw(c[0], c[(int)c.size()-1], c[(int)c.size()
    -2]))
23        c.pop_back();
24    if(!ccw(c.back(), c[0], c[1])){
25        c[0]=c.back();
26        c.pop_back();
27    }
28    c.shrink_to_fit();
29
30    return c;
31 }
```

3.13 2D

```
1 typedef ld cod;
2 bool eq(cod a, cod b){ return fabsl(a - b) <= EPS; }
3
4 // typedef int cod;
5 // bool eq(cod a, cod b){ return (a==b); }
6
7 struct point
8 {
9     cod x, y;
10    int id;
11    point(cod x=0, cod y=0): x(x), y(y){}
12
13
14    point operator+(const point &o) const{
15        return {x+o.x, y+o.y};
16    }
17    point operator-(const point &o) const{
18        return {x-o.x, y-o.y};
19    }
20    point operator*(cod t) const{
21        return {x*t, y*t};
22    }
23    point operator/(cod t) const{
24        return {x/t, y/t};
25    }
26    bool operator<(const point &o) const{
27        if(!eq(x, o.x)) return x < o.x;
28        return y < o.y;
29    }
30    bool operator==(const point &o) const{
31        return eq(x, o.x) and eq(y, o.y);
32    }
33
34 };
35
36 struct line
37 {
38     point fp, sp;
39     point(point fp=0, point sp=0): fp(fp), sp(sp){}
40
41     //a=y1-y2;
42     //b=x2-x1;
43     //c=x2*y1-y2*x1;
44
45 };
46
47
48 // Produto Escalar
49 cod dot(point a, point b){
50     return a.x*b.x + a.y*b.y;
51 }
52 // Produto Vetorial
53 cod cross(point a, point b){
54     return a.x*b.y - a.y*b.x;
55 }
56
57 ld norm(point a){ // Modulo
58     return sqrt(dot(a, a));
59 }
60 ld proj(point a, point b){ // a sobre b
61     return dot(a, b)/norm(b);
62 }
63 ld angle(point a, point b){ // em radianos
64     return acos(dot(a, b) / norm(a) / norm(b));
65 }
66 int ccw(point a, point b, point e) //-1=esq; 0=
       collinear; 1=dir;
67 {
68     cod tmp = cross(b-a, e-a); // from a to b
69     return (tmp > EPS) - (tmp < -EPS);
70 }
71
72 bool collinear(point a, point b, point c){
73     return eq(cross(a-c, b-c), 0);
74 }
75
76 point rotccw(point p, ld a) // em radianos
77 {
78     //a = a*acos(0.0)/90; // graus
79     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
```

```
        +p.x*sin(a)));
80 }
81
82 point rot90cw(point a) { return {a.y, -a.x} };
83 point rot90ccw(point a) { return {-a.y, a.x} };
84
85 // Area de um poligono (pontos ordenados por
       adjacencia)
86 ld area(vector <point> p){
87   ld ret = 0;
88   for(int i=2;i<(int)p.size();i++)
89     ret += cross(p[i] - p[0], p[i-1] - p[0]);
90   return fabsl(ret/2);
91   //return abs(ret/2);
92 }
93
94 // Dist entre ponto e reta
95 cod distr(point a, line b){
96     cod crs = cross(point(a - b.fp), point(b.sp - b.
       fp));
97     return norm(crs/dist(b.fp, b.sp));
98 }
```

4 ED

4.1 Trie

```
1 class Trie {
2 private:
3   struct Node {
4     map<char, Node*> children;
5     int qt = 0;
6     ll size = 0;
7   };
8
9   Node* root;
10
11  void dfs(Node* cur) {
12    ll sz = 1;
13
14    for(auto prox : cur->children) {
15      dfs(prox.second);
16      sz += (prox.second)->size;
17    }
18
19    cur->size = sz;
20  }
21
22  void del(Node* cur, int dep, string &s) {
23    if(dep >= 32)
24      return;
25
26    Node* prox = cur->children[s[dep]];
27    prox->qt--;
28    del(prox, dep+1, s);
29
30    if(prox->qt == 0)
31      cur->children.erase(s[dep]);
32  }
33
34 public:
35  Trie() {
36    root = new Node();
37    root->qt = 1;
38  }
39
40  void add(string s) {
41    Node* cur = root;
42
43    for(auto c : s) {
44      if(cur->children.count(c) == 0) {
45        cur->children[c] = new Node();
```

```
46      }
47      cur->children[c]->qt++;
48      cur = cur->children[c];
49    }
50  }
51
52  void del(string &s) {
53    Node* cur = root;
54    del(cur, 0, s);
55  }
56
57  void size() {
58    this->dfs(root);
59  }
60 };
```

4.2 Range-query-bigger-than-k-BIT

```
1 // C++ program to print the number of elements
2 // greater than k in a subarray of range L-R.
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 // Structure which will store both
7 // array elements and queries.
8 struct node{
9     int pos;
10    int l;
11    int r;
12    int val;
13 };
14
15 // Boolean comparator that will be used
16 // for sorting the structural array.
17 bool comp(node a, node b){
18    // If 2 values are equal the query will
19    // occur first then array element
20    if (a.val == b.val)
21      return a.l > b.l;
22
23    // Otherwise sorted in descending order.
24    return a.val > b.val;
25 }
26
27 // Updates the node of BIT array by adding
28 // 1 to it and its ancestors.
29 void update(int* BIT, int n, int idx){
30    while (idx <= n){
31      BIT[idx]++;
32      idx += idx & (-idx);
33    }
34 }
35 // Returns the count of numbers of elements
36 // present from starting till idx.
37 int query(int* BIT, int idx){
38    int ans = 0;
39    while (idx){
40      ans += BIT[idx];
41
42      idx -= idx & (-idx);
43    }
44    return ans;
45 }
46
47 // Function to solve the queries offline
48 void solveQuery(int arr[], int n, int QueryL[],
49                 int QueryR[], int QueryK[], int q){
50    // create node to store the elements
51    // and the queries
52    node a[n + q + 1];
53    // 1-based indexing.
54
55    // traverse for all array numbers
```

```
56      for(int i = 1; i <= n; ++i){
57          a[i].val = arr[i - 1];
58          a[i].pos = 0;
59          a[i].l = 0;
60          a[i].r = i;
61      }
62
63      // iterate for all queries
64      for(int i = n + 1; i <= n + q; ++i){
65          a[i].pos = i - n;
66          a[i].val = QueryK[i - n - 1];
67          a[i].l = QueryL[i - n - 1];
68          a[i].r = QueryR[i - n - 1];
69      }
70
71      // In-built sort function used to
72      // sort node array using comp function.
73      sort(a + 1, a + n + q + 1, comp);
74
75      // Binary Indexed tree with
76      // initially 0 at all places.
77      int BIT[n + 1];
78
79      // initially 0
80      memset(BIT, 0, sizeof(BIT));
81
82      // For storing answers for each query( 1-based
       indexing ).
83      int ans[q + 1];
84
85      // traverse for numbers and query
86      for (int i = 1; i <= n + q; ++i){
87          if (a[i].pos != 0) {
88
89              // call function to returns answer for
        each query
90              int cnt = query(BIT, a[i].r) - query(BIT,
         a[i].l - 1);
91
92              // This will ensure that answer of each
        query
93              // are stored in order it was initially
        asked.
94              ans[a[i].pos] = cnt;
95          }
96          else{
97              // a[i].r contains the position of the
98              // element in the original array.
99              update(BIT, n, a[i].r);
100         }
101     }
102     // Output the answer array
103     for (int i = 1; i <= q; ++i){
104         cout << ans[i] << endl;
105     }
106 }
107
108 // Driver Code
109 int main()
110 {
111     int arr[] = { 7, 3, 9, 13, 5, 4 };
112     int n = sizeof(arr) / sizeof(arr[0]);
113
114     // 1-based indexing
115     int QueryL[] = { 1, 2 };
116     int QueryR[] = { 4, 6 };
117
118     // k for each query
119     int QueryK[] = { 6, 8 };
120
121     // number of queries
122     int q = sizeof(QueryL) / sizeof(QueryL[0]);
123
```

```
124     // Function call to get
125     solveQuery(arr, n, QueryL, QueryR, QueryK, q);
126
127     return 0;
128 }
```

4.3   Iterative-SegTree

```
1  // Segment Tree Iterativa - Range maximum query
2
3  #define N 100010
4
5  struct Segtree{
6      int t[2*N]={0};
7
8      void build(){
9          for(int i=N-1; i>0; i--)
10             t[i]=max(t[i<<1], t[1<<1|1]);
11     }
12
13     int query(int l, int r){
14         int ans=0;
15         for(i+=N, r+=N; l<r; l>>=1, r>>=1)
16         {
17             if(l&1)
18                 ans=max(ans, t[l++]);
19             if(r&1)
20                 ans=max(ans, t[--r]);
21         }
22
23         return ans;
24     }
25
26     void update(int p, int value){
27         for(t[p+=n]=value; p>1; p>>=1)
28             t[p>>1]= max(t[p], t[p^1]);
29     }
30
31 };
32
33 int main()
34 {
35     Segtree st;
36
37     for(int i=0;i<n;i++){
38         cin >> aux;
39         st.t[N+i]=aux; //Leaves are stored in
       continuous nodes with indices starting with N
40     }
41
42     st.build();
43     x = st.query(inicio, fim);
44     st.update(ind, value);
45
46 }
```

4.4   Recursive-SegTree

```
1  // Segment Tree Recursiva - Range maximum query
2
3  vector<int> val(MAX, 0);
4  vector<int> vet(N);
5
6  void monta(int i, int j, int no){
7      if(i==j){
8          val[no]=vet[i];
9          return;
10     }
11
12     int esq = 2*no;
13     int dir = 2*no+1;
14     int meio = (i+j)/2;
```

```
15
16        monta(i, meio, esq);
17        monta(meio+1, j, dir);
18
19        val[no]=max(val[esq], val[dir]);
20 }
21
22 void atualiza(int no, int i, int j, int pos, int
       novo_valor){
23        if(i==j){
24            val[no]=novo_valor;
25        }else{
26            int esq = 2*no;
27            int dir = 2*no+1;
28            int meio = (i+j)/2;
29
30            if(pos<=meio)
31                atualiza(esq, i, meio, pos, novo_valor);
32            else
33                atualiza(dir, meio+1, j, pos, novo_valor)
       ;
34
35            if(val[esq]>val[dir])
36                val[no]=val[esq];
37            else
38                val[no]=val[dir];
39        }
40 }
41
42 int consulta(int no, int i, int j, int A, int B){
43        if(i>B || j<A)
44            return -1;
45        if(i>=A and j<=B)
46            return val[no];
47
48        int esq = 2*no;
49        int dir = 2*no+1;
50        int meio = (i+j)/2;
51
52        int resp_esq = consulta(esq, i, meio, A, B);
53        int resp_dir = consulta(dir, meio+1, j, A, B);
54
55        if(resp_dir==-1)
56            return resp_esq;
57        if(resp_esq==-1)
58            return resp_dir;
59
60        if(resp_esq>resp_dir)
61            return resp_esq;
62        else
63            return resp_dir;
64 }
65
66 int main()
67 {
68        monta(1, N, 1);
69        atualiza(1, 1, N, pos, valor);
70        x = consulta(1, 1, N, inicio, fim);
71 }
```

4.5   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 }
```

4.6   Seg-Tree-Farao

```
1 typedef struct
2 {
3        pii prefix, sufix, total, maximo;
4 } no;
5
6 int noleft[MAX], noright[MAX]; //Guarda os valores
       dos nos para que nao sejam calculados novamente
       nas querys
7 int v[MAX];
8 no arvore[MAX];
9
10 pii somar(pii a, pii b) // une pairs
11 {
12        return mp(a.f+b.f, a.s+b.s);
13 }
14
15 no une(no l, no r)
16 {
17        if(l.total.s==0)
18            return r;
19        if(r.total.s==0)
20            return l;
21
22        no m;
23
24        m.prefix = max(l.prefix, somar(l.total, r.prefix)
       ); //prefixo
25        m.sufix = max(r.sufix, somar(r.total, l.sufix));
       //sufixo
26        m.total = somar(l.total, r.total); //Soma de
       todos os elementos da subarvore
27        m.maximo = max(max(l.maximo, r.maximo), somar(l.
       sufix, r.prefix)); //Resultado para cada
       subarvore
28
29        return m;
30 }
31
32 no makenozero()
33 {
34        no m;
35        m.prefix=m.sufix=m.total=m.maximo=mp(0,0);
36        return m;
37 }
38
39 no makeno(int k)
40 {
41        no m;
42        m.prefix=m.sufix=m.total=m.maximo=mp(k,1);
43        return m;
44 }
45
46 void monta(int n)
47 {
48        if(noleft[n]==noright[n])
49        {
50            arvore[n]=makeno(v[noleft[n]]);
51            return;
52        }
53
54        int mid = (noleft[n]+noright[n])/2;
55        noleft[2*n]=noleft[n]; noright[2*n]=mid;
56        noleft[2*n+1]=mid+1; noright[2*n+1]=noright[n];
57
58        monta(2*n);
59        monta(2*n+1);
```

```
60      arvore[n]=une(arvore[2*n], arvore[2*n+1]);
61  }
62
63
64  no busca(int n, int esq, int dir)
65  {
66      if(noleft[n]>=esq and noright[n]<=dir)
67          return arvore[n];
68      if(noright[n]<esq or noleft[n]>dir)
69          return makenozero();
70
71      return une(busca(2*n, esq, dir),busca(2*n+1, esq,
         dir));
72  }
73
74  int main()
75  {
76      int T, N, Q, A, B;
77      no aux;
78
79      scanf("%d", &T);
80
81      while(T--)
82      {
83          scanf("%d", &N);
84          for(int i=1;i<=N;i++)
85              scanf("%d", &v[i]); //Elementos da arvore
86
87          noleft[1]=1;noright[1]=N;
88          monta(1);
89
90          cin >> Q;
91          while(Q--)
92          {
93              scanf("%d%d", &A, &B); //Intervalo da
         query
94              aux = busca(1, A, B);
95              printf("%d %d\n", aux.maximo.f, aux.
         maximo.s);
96          }
97      }
98
99
100     return 0;
101 }
```

4.7   BIT-2D

```
1  // BIT 2D
2
3  int bit[MAX][MAX];
4
5  int sum(int x, int y)
6  {
7      int resp=0;
8
9      for(int i=x;i>0;i-=i&-i)
10         for(int j=y;j>0;j-=j&-j)
11             resp+=bit[i][j];
12
13     return resp;
14 }
15
16 void update(int x, int y, int delta)
17 {
18     for(int i=x;i<MAX;i+=i&-i)
19         for(int j=y;j<MAX;j+=j&-j)
20             bit[i][j]+=delta;
21 }
22
23 int query(int x1, y1, int x2, y2)
24 {
```

```
25     return sum(x2,y2) - sum(x2,y1) - sum(x1,y2) + sum
       (x1,y1);
26 }
```

4.8   BIT

```
1  struct FT {
2      vector<int> bit;  // indexado em 1
3      int n;
4
5      FT(int n) {
6          this->n = n + 1;
7          bit.assign(n + 1, 0);
8      }
9
10     int sum(int idx) {
11         int ret = 0;
12         for (++idx; idx > 0; idx -= idx & -idx)
13             ret += bit[idx];
14         return ret;
15     }
16
17     int sum(int l, int r) {
18         return sum(r) - sum(l - 1);
19     }
20
21     void add(int idx, int delta) {
22         for (++idx; idx <= n; idx += idx & -idx)
23             bit[idx] += delta;
24     }
25 };
```

4.9   Sparse-Table

```
1  int logv[MAX+1];
2  void make_log() {
3      logv[1] = 0; // pre-computar tabela de log
4      for (int i = 2; i <= MAX; i++)
5          logv[i] = logv[i/2] + 1;
6  }
7
8  struct Sparse {
9      int n;
10     vector<vector<int>> st;
11
12     Sparse(int n, vi array) {
13         this->n = n;
14         int k = logv[n];
15         st.assign(n+1, vector<int>(k+1, 0));
16
17         for (int i = 0; i < n; i++)
18             st[i][0] = array[i];
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     int f(int a, int b) {
26         return min(a, b);
27     }
28
29     int query(int L, int R) {
30         int j = logv[R - L + 1];
31         int res = f(st[L][j], st[R - (1 << j) + 1][j
       ]);
32         return res;
33     }
34 };
```

4.10   Union-Find

```
1  struct DSU {
2      int n;
3      vector<int> parent, size;
4
5      DSU(int n) {
6          this->n = n;
7          parent.assign(n+1, 0);
8          size.assign(n+1, 1);
9
10         for(int i = 0; i <= n; i++)
11             parent[i] = i;
12     }
13
14     int find(int v) {
15         if(v == parent[v])
16             return v;
17         return find(parent[v]);
18     }
19
20     void join(int a, int b) {
21         a = find(a);
22         b = find(b);
23
24         if(a != b) {
25             if(size[a] < size[b])
26                 swap(a, b);
27
28             parent[b] = a;
29             size[a] += b;
30         }
31     }
32 };
```

4.11   Mo

```
1  const int BLK = 500; // tamanho do bloco, algo entre
       300 e 500 e nice
2
3  struct Query {
4      int l, r, idx;
5      Query(int l, int r, int idx) {
6          this->l = l;
7          this->r = r;
8          this->idx = idx;
9      }
10     bool operator<(Query other) const {
11         return make_pair(l / BLK, r) <
12         make_pair(other.l / BLK, other.r);
13     }
14 };
15
16 void add() void remove() // implementar operacoes de
       acordo com o problema, cuidado com TLE ao
       utilizar MAP
17
18 vector<int> mo(vector<Query> queries) {
19     vector<int> res(queries.size());
20     sort(queries.begin(), queries.end());
21     resposta = 0;
22
23     int l = 0, r = -1;
24     for(Query q : queries) {
25         while(l > q.l) {
26             l--;
27             add(l);
28         }
29         while(r < q.r) {
30             r++;
31             add(r);
32         }
33         while(l < q.l) {
34             remove(l);
35             l++;
```

```
36         }
37         while(r > q.r) {
38             remove(r);
39             r--;
40         }
41         res[q.idx] = resposta; // adicionar resposta
       de acordo com o problema
42     }
43     return res; // ordernar o vetor pelo indice e
       responder queries na ordem
44 }
```

5   Math

5.1   Totient

```
1  // phi(p^k) = (p^(k-1))*(p-1) com p primo
2  // O(sqrt(m))
3  ll phi(ll m) {
4      ll res = m;
5      for(ll d = 2; d*d <= m; d++) {
6          if(m % d == 0) {
7              res = (res/d) * (d-1);
8              while(m % d == 0) {
9                  m /= d;
10             }
11         }
12     }
13     if(m > 1) {
14         res /= m;
15         res *= (m-1);
16     }
17
18     return res;
19 }
20
21 // modificacao do crivo, O(n*log(log(n)))
22 vector<ll> phi_to_n(ll n) {
23     vector<bool> isprime(n+1, true);
24     vector<ll> tot(n+1);
25     tot[0] = 0; tot[1] = 1;
26     for(ll i = 1; i <= n; i++) {
27         tot[i] = i;
28     }
29
30 for(ll p = 2; p <= n; p++) {
31     if(isprime[p]) {
32         tot[p] = p-1;
33         for(ll i = p+p; i <= n; i += p) {
34             isprime[i] = false;
35             tot[i] = (tot[i]/p)*(p-1);
36         }
37     }
38 }
39
40     return tot;
41 }
```

5.2   Sqrt-BigInt

```
1  public static BigInteger isqrtNewton(BigInteger n) {
2      BigInteger a = BigInteger.ONE.shiftLeft(n.
       bitLength() / 2);
3      boolean p_dec = false;
4      for (;;) {
5          BigInteger b = n.divide(a).add(a).shiftRight
       (1);
6          if (a.compareTo(b) == 0 || a.compareTo(b) < 0
       && p_dec)
7              break;
8          p_dec = a.compareTo(b) > 0;
```

```
 9            a = b;
10        }
11        return a;
12 }
```

5.3   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
```

5.4   Sum-n2

Soma dos n primeiros números ao quadrado $= \frac{(2N^3+3N^2+N)}{6}$

5.5   Factorization-sqrt

```
 1 // Factorization of a number in sqrt(n)
 2
 3 int main()
 4 {
 5     ll N;
 6     vector<int> div;
 7
 8     cin >> N;
 9
10     for(ll i=2;i*i<=N;i++)
11     {
12         if(N%i==0)
13         {
14             vet.pb(i);
15             while(N%i==0)
16                 N/=i;
17         }
18     }
19     if(N!=1)
20         vet.pb(N);
21
22     return 0;
23 }
```

5.6   Modular-Exponentiation

```
 1 ll fexp(ll b, ll e, ll mod) {
 2     if(e == 0) return 1LL;
 3     ll res = fexp(b, e/2LL, mod);
 4     res = (res*res)%mod;
 5     if(e%2LL)
 6         res = (res*b)%mod;
 7
 8     return res%mod;
 9 }
```

5.7   Miller-Habin

```
 1 ll llrand()
 2 {
 3     ll tmp = rand();
 4     return (tmp << 31) | rand();
 5 }
 6
 7 ll add(ll a, ll b, ll c)
 8 {
 9     return (a + b)%c;
10 }
11
12 ll mul(ll a, ll b, ll c)
13 {
14     ll ans = 0;
15     while(b)
16     {
17         if(b & 1)
18             ans = add(ans, a, c);
19         a = add(a, a, c);
20         b /= 2;
21     }
22     return ans;
23 }
24
25 ll fexp(ll a, ll b, ll c)
26 {
27     ll ans = 1;
28     while(b)
29     {
30         if(b & 1)
31             ans = mul(ans, a, c);
32         a = mul(a, a, c);
33         b /= 2;
34     }
35     return ans;
36 }
37
38 bool rabin(ll n)
39 {
40     if(n <= 1)
41         return 1;
42     if(n <= 3)
43         return 1;
44
45     ll s=0, d=n-1;
46     while(d%2==0)
47     {
48         d/=2;
49         s++;
50     }
51
52     for(int k = 0; k < 64*4; k++)
53     {
54         ll a = (llrand()%(n - 3)) + 2;
55         ll x = fexp(a, d, n);
56         if(x != 1 and x != n-1)
57         {
58             for(int r = 1; r < s; r++)
59             {
60                 x = mul(x, x, n);
61                 if(x == 1)
```

```
62                    return 0;
63                if(x == n-1)
64                    break;
65            }
66            if(x != n-1)
67                return 0;
68        }
69    }
70
71    return 1;
72 }
73
74
75 int main()
76 {
77
78    ll N;
79    cin >> N;
80
81    cout << rabin(N) << endl;
82
83    return 0;
84
85 }
```

5.8 Inverso-Mult

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

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

5.9 Pollard-Rho

```cpp
// Pollard Rho Algorithm

#include <bits/stdc++.h>
#define ll long long

using namespace std;

ll llrand()
{
    ll tmp = rand();
    return (tmp << 31) | rand();
}

ll add(ll a, ll b, ll c)
{
    return (a + b)%c;
}

ll mul(ll a, ll b, ll c)
{
    ll ans = 0;
    while(b)
    {
        if(b & 1)
            ans = add(ans, a, c);
        a = add(a, a, c);
        b /= 2;
    }
    return ans;
}
```

```cpp
}

ll rho(ll n)
{
    ll x, c, y, d, k;
    int i;
    do{
        i = 1;
        x = llrand()%n;
        c = llrand()%n;
        y = x, k = 4;
        do{
            if(++i == k)
            {
                y = x;
                k *= 2;
            }
            x = add(mul(x, x, n), c, n);
            d = __gcd(abs(x - y), n);
        }
        while(d == 1);
    }
    while(d == n);

    return d;
}

int main()
{
    srand(time(0));

    ll N;
    cin >> N;

    ll div = rho(N);
    cout << div << " " << N/div << endl;


    // Finding all divisors

    vector<ll> div;

    while(N>1 and !rabin(N))
    {
        ll d = rho(N);
        div.pb(d);
        while(N%d==0)
            N/=d;
    }
    if(N!=1)
        div.pb(N);

    return 0;

}
```

5.10 Verif-primo

```cpp
// prime verification sqrt(N)

bool eh_primo(long long N)
{
    if(N==2)
        return true;
    else if(N==1 or N%2==0)
        return false;
    for(long long i=3;i*i<=N;i+=2)
        if(N%i==0)
            return false;
    return true;
}
```

5.11 Crivo

```
1  // Sieve of Eratosthenes
2
3  int N;
4  vector<bool> primos(100010, true);
5  cin >> N;
6
7  primos[0]=false;
8  primos[1]=false;
9
10 for(int i=2;i<=N;i++)
11     if(primos[i])
12         for(int j=i+i; j<=N; j+=i)
13             primos[j]=false;
```

5.12 Simpson's-formula

```
1  inline ld simpson(ld fl, ld fr, ld fmid, ld l, ld r){
2      return (fl+fr+4*fmid)*(r-l)/6;
3  }
4
5  ld rsimpson(ld slr, ld fl, ld fr, ld fmid, ld l, ld r)
6  {
7      ld mid = (l+r)/2;
8      ld fml = f((l+mid)/2), fmr = f((mid+r)/2);
9      ld slm = simpson(fl,fmid,fml,l,mid);
10     ld smr = simpson(fmid,fr,fmr,mid,r);
11     if(fabsl(slr-slm-smr) < EPS) return slm+smr; //
       aprox. good enough
12     return rsimpson(slm,fl,fmid,fml,l,mid)+rsimpson(
       smr,fmid,fr,fmr,mid,r);
13 }
14
15 ld integrate(ld l, ld r)
16 {
17     ld mid = (l+r)/2;
18     ld fl = f(l), fr = f(r);
19     ld fmid = f(mid);
20     return rsimpson(simpson(fl,fr,fmid,l,r),fl,fr,
       fmid,l,r);
21 }
```

5.13 FFT

```
1  struct num{
2      double x, y;
3      num() { x = y = 0; }
4      num(double x, double y) : x(x), y(y) {}
5  };
6
7  inline num operator+(num a, num b) { return num(a.x +
       b.x, a.y + b.y); }
8  inline num operator-(num a, num b) { return num(a.x -
       b.x, a.y - b.y); }
9  inline num operator*(num a, num b) { return num(a.x *
       b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
10 inline num conj(num a) { return num(a.x, -a.y); }
11
12 int base = 1;
13 vector<num> roots = {{0, 0}, {1, 0}};
14 vector<int> rev = {0, 1};
15
16 const double PI = acosl(-1.0);
17
18 void ensure_base(int nbase){
19     if(nbase <= base)
20         return;
21
22     rev.resize(1 << nbase);
23     for(int i = 0; i < (1 << nbase); i++)
24         rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (
       nbase - 1));
25
26     roots.resize(1 << nbase);
27
28     while(base < nbase){
29         double angle = 2*PI / (1 << (base + 1));
30         for(int i = 1 << (base - 1); i < (1 << base);
       i++){
31             roots[i << 1] = roots[i];
32             double angle_i = angle * (2 * i + 1 - (1
       << base));
33             roots[(i << 1) + 1] = num(cos(angle_i),
       sin(angle_i));
34         }
35         base++;
36     }
37 }
38
39 void fft(vector<num> &a, int n = -1){
40     if(n == -1)
41         n = a.size();
42
43     assert((n & (n-1)) == 0);
44     int zeros = __builtin_ctz(n);
45     ensure_base(zeros);
46     int shift = base - zeros;
47     for(int i = 0; i < n; i++)
48         if(i < (rev[i] >> shift))
49             swap(a[i], a[rev[i] >> shift]);
50
51     for(int k = 1; k < n; k <<= 1)
52         for(int i = 0; i < n; i += 2 * k)
53             for(int j = 0; j < k; j++){
54                 num z = a[i+j+k] * roots[j+k];
55                 a[i+j+k] = a[i+j] - z;
56                 a[i+j] = a[i+j] + z;
57             }
58 }
59
60 vector<num> fa, fb;
61 vector<int> multiply(vector<int> &a, vector<int> &b){
62     int need = a.size() + b.size() - 1;
63     int nbase = 0;
64     while((1 << nbase) < need) nbase++;
65     ensure_base(nbase);
66     int sz = 1 << nbase;
67     if(sz > (int) fa.size())
68         fa.resize(sz);
69
70     for(int i = 0; i < sz; i++){
71         int x = (i < (int) a.size() ? a[i] : 0);
72         int y = (i < (int) b.size() ? b[i] : 0);
73         fa[i] = num(x, y);
74     }
75     fft(fa, sz);
76     num r(0, -0.25 / sz);
77     for(int i = 0; i <= (sz >> 1); i++){
78         int j = (sz - i) & (sz - 1);
79         num z = (fa[j] * fa[j] - conj(fa[i] * fa[i]))
        * r;
80         if(i != j) {
81             fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[
       j])) * r;
82         }
83         fa[i] = z;
84     }
85     fft(fa, sz);
86     vector<int> res(need);
87     for(int i = 0; i < need; i++)
88         res[i] = fa[i].x + 0.5;
89
90     return res;
```

```
91  }
92
93  int main()
94  {sws;
95
96      //FFT
97      vector<int> fx{1, 2, 3}; // 1+2x+3x^2
98      vector<int> gx{4, 5}; // 4+5x
99      vector<int> res;
100
101     res = multiply(fx,gx); //4 + 13x + 22x^2 + 15x^3
102
103     return 0;
104
105 }
```

5.14 Next-Permutation

```
1  vector<int> a = {1, 2, 3};
2  int n = a.size();
3  do{
4      display(a, n);// 1,2,3; 1,3,2; 2,1,3; 3,1,2;
       2,3,1; 3,2,1;
5  }while(next_permutation(a.begin(), a.begin() + n));
```

5.15 Fast-Exponentiation

```
1  // Modular exponentiaion - (x^y)%mod in O(log y)
2  ll power(ll x, ll y, ll mod)
3  {
4      ll res = 1;
5      x%=mod;
6
7      while(y)
8      {
9          if(y&1)
10             res=(res*x)%mod;
11
12         y=y>>1;
13         x=(x*x)%mod;
14     }
15     return res;
16 }
```

5.16 Recursao-linear

```
1  vector<vl> id(int n) {
2      vector<vl> res(n, vl(n, 0));
3      for(int i = 0; i < n; i++) res[i][i] = 1;
4      return res;
5  }
6
7  vector<vl> mult(vector<vl> a, vector<vl> b, int n) {
8      vector<vl> res(n, vl(n, 0));
9
10     for(int row = 0; row < n; row++) {
11         for(int col = 0; col < n; col++) {
12             ll val = 0;
13             for(int k = 0; k < n; k++) {
14                 ll delta = (a[row][k] * b[k][col]) %
       MOD;
15                 val = (val + delta) % MOD;
16             }
17             res[row][col] = val;
18         }
19     }
20
21     return res;
22 }
23
24 vector<vl> fexp(vector<vl> b, ll e, int n) {
25     if(e == 0) {
```

```
26             return id(n);
27     }
28
29     vector<vl> res = fexp(b, e/2, n);
30     res = mult(res, res, n);
31
32     if(e%2)
33         res = mult(res, b, n);
34
35     return res;
36 }
37
38 // k = tamanho da recorrencia/matriz, n = n-esimo
       termo
39 // f(n) = c1*f(n-1) + c2*f(n-2) + ... + ck*f(n-k)
40 // base -> [f(k-1), f(k-2), ..., f(0)]
41 // coeficientes -> [c1, c2, ..., ck]
42 vl solve(int k, int n, vl base, vl coef) {
43     vector<vl> inicial;
44     inicial.pb(coef);
45     for(int row = 0; row < k-1; row++) {
46         vl tmp;
47         for(int col = 0; col < k; col++) {
48             if(col == row)
49                 tmp.pb(1);
50             else
51                 tmp.pb(0);
52         }
53         inicial.pb(tmp);
54     }
55
56     vector<vl> matexp = fexp(inicial, max(0, n-k+1),
       k);
57     vl res(k);
58
59     for(int row = 0; row < k; row++) {
60         ll val = 0;
61         for(int aux = 0; aux < k; aux++) {
62             val += matexp[row][aux]*base[aux];
63         }
64         res[row] = val; // res = (f(n), f(n-1), ...,
       f(n-k+1))
65     }
66
67     return res;
68 }
```

5.17 Raiz-primitiva

```
1  ll fexp(ll b, ll e, ll mod) {
2      if(e == 0) return 1LL;
3      ll res = fexp(b, e/2LL, mod);
4      res = (res*res)%mod;
5      if(e%2LL)
6          res = (res*b)%mod;
7
8      return res%mod;
9  }
10
11 vl fatorar(ll n) { // fatora em primos
12     vl fat;
13     for(int i = 2; i*i <= n; i++) {
14         if(n%i == 0) {
15             fat.pb(i);
16             while(n%i == 0)
17                 n /= i;
18         }
19     }
20     return fat;
21 }
22
23 // O(log(n) ^ 2)
24 bool raiz_prim(ll a, ll mod, ll phi, vl fat) {
```

```
25      if(__gcd(a, mod) != 1 or fexp(a, phi/2, mod) ==
        1) // phi de euler sempre eh PAR
26          return false;
27
28      for(auto f : fat) {
29          if(fexp(a, phi/f, mod) == 1)
30              return false;
31      }
32
33      return true;
34 }
35
36 // mods com raizes primitivas: 2, 4, p^k, 2*p^k, p eh
       primo impar, k inteiro --- O(n log^2(n))
37 ll achar_raiz(ll mod, ll phi) {
38      if(mod == 2) return 1;
39      vl fat, elementos;
40      fat = fatorar(phi);
41
42      for(ll i = 2; i <= mod-1; i++) {
43          if(raiz_prim(i, mod, phi, fat))
44              return i;
45      }
46
47      return -1; // retorna -1 se nao existe
48 }
49
50 vl todas_raizes(ll mod, ll phi, ll raiz) {
51      vl raizes;
52      if(raiz == -1) return raizes;
53      ll r = raiz;
54      for(ll i = 1; i <= phi-1; i++) {
55          if(__gcd(i, phi) == 1) {
56              raizes.pb(r);
57          }
58          r = (r * raiz) % mod;
59      }
60
61      return raizes;
62 }
```

5.18 Kamenetsky

```
1 // Number of digits in n! O(1)
2
3 #define Pi 3.14159265358979311599796346854
4 #define Eul 2.71828182845904509079559829842
5
6 long long findDigits(int n)
7 {
8      double x;
9
10      if (n < 0)
11          return 0;
12      if (n == 1)
13          return 1;
14
15      x = ((n * log10(n / euler) + log10(2 * Pi * n)
        /2.0));
16
17      return floor(x) + 1;
18 }
```

6 Misc

6.1 2SAT

```
1 vector<int> g[MAX], gt[MAX], S; int vis[MAX], cor[MAX
    ];
2
3 int val(int n, bool tvalue) {
4      if(tvalue) return 2*n;
5      return 2*n +1;
6 }
7
8 void dfs(int u) {
9      vis[u] = 1; for(int v : g[u]) if(!vis[v]) dfs(v);
10      S.push_back(u);
11 }
12
13 void dfst(int u, int e) {
14      cor[u] = e;
15      for(int v : gt[u]) if(!cor[v]) dfst(v, e);
16 }
17
18 void kosaraju(int n) {
19      for(int i = 0; i <= n; i++) if(!vis[i]) dfs(i);
20      for(int i = 0; i <= n; i++) for(int j : g[i])
21          gt[j].push_back(i);
22      int e = 0; reverse(S.begin(), S.end());
23      for(int u : S) if(!cor[u]) dfst(u, ++e);
24 }
25
26 // antes de chamar essa funcao, colocar as arestas do
        grafo
27 bool solve(int n, vi &res) {
28      kosaraju(2*n); // MAX > 2*N
29      vi r;
30
31      forn(i, n) {
32          int t = val(i, true), f = val(i, false);
33          if(cor[t] == cor[f]) {
34              return false;
35          }
36          else {
37              if(cor[t] > cor[f])
38                  r.pb(1);
39              else
40                  r.pb(0);
41          }
42      }
43      swap(r, res);
44      return true;
45 }
```

6.2 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();
```

6.3 Bitwise

```
1 // Bitwise
2      #pragma GCC target("popcnt")
3      unsigned char a = 5, b = 9; // a = (00000101), b
     = (00001001)
4
5      AND -          a&b   // The result is 00000001
     (1)
6      OR -           a|b   // The result is 00001101
     (13)
7      XOR -          a^b   // The result is 00001100
     (12)
8      NOT -          ~a    // The result is 11111010
     (250)
9      Left shift -   b<<1  // The result is 00010010
     (18)
```

```
Right shift -  b>>1  // The result is 00000100
(4)

// Exchange two int variables

    a^=b;
    b^=a;
    a^=b;

// Even or Odd

    (x & 1)? printf("Odd"): printf("Even");

// Turn on the j-th bit

    int S = 34; //(100010)
    int j = 3;

    S = S | (1<<j);

// Turn off the j-th bit

    int S = 42; //(101010)
    int j = 1;

    S &= ~(1<<j)

    S == 40 //(101000)

// Check the j-th element

    int S = 42; //(101010)
    int j = 3;

    T = S & (1<<j); // T = 0

// Least significant bit (lsb)

    int lsb(int x){ return x&-x; }

// Exchange o j-th element

    S ^= (1<<j)

// Position of the first bit on

    T = (S & (-S))
    T -> 4 bit ligado //(1000)

// Most significant digit of N

    double K = log10(N);
    K = K - floor(K);
    int X = pow(10, K);

// Number of digits in N

    X =floor(log10(N)) + 1;

// Power of two

    bool isPowerOfTwo(int x){ return x && (!(x&(x
-1))); }

// Turn off the first bit 1
    m = m & (m-1);

// Built-in functions

    // Number of bits 1
    __builtin_popcount()
    __builtin_popcountll()
```

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

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

// floor(log2(x))

    int flog2(int x){ return 32-1-__builtin_clz(x
); }

    int flog2ll(ll x){ return 64-1-
__builtin_clzll(x); }
```

6.4   Template

```
#include <bits/stdc++.h>
#define ff first
#define ss second
#define ll long long
#define ld long double
#define pb push_back
#define eb emplace_back
#define mp make_pair
#define mt make_tuple
#define pii pair<int, int>
#define vi vector<int>
#define sws ios_base::sync_with_stdio(false);cin.tie(
    NULL)
#define endl '\n'
#define teto(a, b) (a+b-1)/(b)

const int MAX = 400010;
const int MOD = 1e9+7;
const int INF = 0x3f3f3f3f;
const ll LLINF = 0x3f3f3f3f3f3f3f3f;
const ld EPS = 1e-7;

using namespace std;
```

7   Strings

7.1   KMP

```
vector<int> preffix_function(const string &s){
    int n = s.size(); vector<int> b(n+1);
    b[0] = -1; int i = 0, j = -1;
    while(i < n){
        while(j >= 0 && s[i] != s[j]) j = b[j];
        b[++i] = ++j;
    }
    return b;
}

void kmp(const string &t, const string &p){
    vector<int> b = preffix_function(p);
    int n = t.size(), m = p.size();
    int j = 0;
    for(int i = 0; i < n; i++){
        while(j >= 0 && t[i] != p[j]) j = b[j];
    j++;
    if(j == m) {
        j = b[j];
    }
    }
}
```

7.2   LCS

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

    int result = 0, end;
    int len[2][n];
    int currRow = 0;

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

        currRow = 1 - currRow;
    }

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

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

7.3 Pal-int

```cpp
bool ehpalindromo(ll n) {
    if(n<0)
        return false;

    int divisor = 1;
    while(n/divisor >= 10)
        divisor *= 10;

    while(n != 0) {
        int leading = n / divisor;
        int trailing = n % 10;

        if(leading != trailing)
            return false;

        n = (n % divisor)/10;

        divisor = divisor/100;
    }

    return true;
}
```

7.4 Z-Func

```cpp
vector<int> z_algo(const string &s)
{
    int n = s.size();
    int L = 0, R = 0;
    vector<int> z(n, 0);
    for(int i = 1; i < n; i++)
```

```cpp
    {
        if(i <= R)
            z[i] = min(z[i-L], R - i + 1);
        while(z[i]+i < n && s[ z[i]+i ] == s[ z[i] ])
            z[i]++;
        if(i+z[i]-1 > R)
        {
            L = i;
            R = i + z[i] - 1;
        }
    }
    return z;
}
```

7.5 Hash

```cpp
ll compute_hash(string const& s) {
    const ll p = 31; // primo, melhor = perto da
    quantidade de caracteres
    const ll m = 1e9 + 9; // maior mod = menor
    probabilidade de colisao
    ll hash_value = 0;
    ll p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' + 1) *
    p_pow) % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
```

7.6 Manacher

```cpp
// O(n), d1 -> palindromo impar, d2 -> palindromo par
    (centro da direita)
void manacher(string &s, vi &d1, vi &d2) {
    int n = s.size();
    for(int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 1 : min(d1[l + r - i], r -
    i + 1);
        while(0 <= i - k && i + k < n && s[i - k] ==
    s[i + k]) {
            k++;
        }
        d1[i] = k--;
        if(i + k > r) {
            l = i - k;
            r = i + k;
        }
    }

    for(int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 0 : min(d2[l + r - i + 1],
    r - i + 1);
        while(0 <= i - k - 1 && i + k < n && s[i - k
    - 1] == s[i + k]) {
            k++;
        }
        d2[i] = k--;
        if(i + k > r) {
            l = i - k - 1;
            r = i + k ;
        }
    }
}
```