Notebook - Maratona de Programação

Tiago de Souza Fernandes

Sumário

1 Algoritmos

1.1 Mochila

```
1  int val[MAXN], peso[MAXN], dp[MAXN][MAXS]
2
3  int knapsack(int N, int M) // Objetos | Peso max
4  {
5      for(i=0;i<=N;i++)
6      {
7          for(j=0;j<=M;j++)
8          {
9              if (i==0 || j==0)
10                 dp[i][j] = 0;
11             else if (peso[i-1] <= j)
12                 dp[i][j] = max(val[i-1]+dp[i-1][j-
   peso[i-1]],  dp[i-1][j]);
13             else
14                 dp[i][j] = dp[i-1][j];
15         }
16     }
17     return dp[N][M];
18 }
```

1.2 Kadane-DP

```
1  // Largest Sum Contiguous Subarray
2
3  int maxSubArraySum(vector<int> a)
4  {
5      int size = a.size();
6      int max_so_far = a[0];
7      int curr_max = a[0];
8
9      for (int i=1;i<size;i++)
10     {
11         curr_max = max(a[i], curr_max+a[i]);
12         max_so_far = max(max_so_far, curr_max);
13     }
14     return max_so_far;
15 }
```

1.3 Iterative-BS

```
1  int main()
2  {
3      int l=1, r=N;
4      int res=-1;
5
6      while(l <= r)
7      {
8          int m = (l + r)/2;
9          if(!ver(m))
10         {
11             l = m+1;
12         }
13         else
14         {
15             res = m;
16             r = m-1;
17         }
18     }
19     cout << res << endl;
20
21     return 0;
22 }
```

2 Grafos

2.1 BFS

```
1  //BFS (Breadth First Search) O(V+A)
2
3  vector<vector<int>> adj;  // adjacency list
      representation
4  int n; // number of nodes
5  int s; // source vertex
6
7  queue<int> q;
8  vector<int> d(n, INF);
9  d[s]=0;
10
11 q.push(s);
12 used[s] = true;
13 while (!q.empty()) {
14     int v = q.front();
15     q.pop();
16     for (int u : adj[v]) {
17         if (d[u] > d[v] + 1) {
18             q.push(u);
19             d[u] = d[v] + 1;
20         }
21     }
22 }
```

2.2 Find-bridges

```
1  #define vi vector<int>
2
3  vector< vector<int> > grafo;
4  vector<bool> visited;
5  vi t, low;
6  int timer=0;
7
8  void find_bridges(int v, int p=-1)
9  {
10     visited[v] = true;
11     t[v] = low[v] = timer++;
12     for(int i=0;i<(int)grafo[v].size();i++)
13     {
14         int vert = grafo[v][i];
15         if(vert == p)
16             continue;
17         if(visited[vert])
18             low[v] = min(low[v], t[vert]);
19         else
20         {
21             find_bridges(vert, v);
22             low[v] = min(low[v], low[vert]);
23             if(low[to] > t[v])
24                 IS_BRIDGE(v, vert);
25         }
26     }
27 }
28
29 int main()
30 {
31     timer = 0;
32     visited.assign(N+1, false);
33     t.assign(N+1, 0);
34     low.assign(N+1, 0);
35
36     for(int i=0;i<N;i++)
37         if(!visited[i])
38             find_bridges(1);
39
40     return 0;
41 }
```

2.3 Dijkstra

```
1  // Dijkstra - Shortest Path
2
```

```
3  #define pii pair<int, int>
4  #define vi vector<int>
5  #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 {
14     int dist, vert, aux;
15     distancia[k]=0;
16
17     fila.push(mp(k, 0));
18
19     while(!fila.empty())
20     {
21         aux=fila.top().f;
22         fila.pop();
23
24         for(auto v: grafo[aux])
25         {
26             vert=v.f;
27             dist=v.s;
28             if(distancia[vert]>distancia[aux]+dist)
29             {
30                 distancia[vert]=distancia[aux]+dist;
31                 fila.push(mp(vert, distancia[vert]));
32             }
33         }
34     }
35 }
36
37 int main()
38 {
39     for(int i=0; i<M; i++)
40     {
41         cin >> a >> b >> p;
42         grafo[a].pb(mp(b, p));
43         grafo[b].pb(mp(a, p));
44     }
45 }
```

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  {
5      for(int i=0; i<(int)vizinhos[x].size(); i++)
6      {
7          int v = vizinhos[x][i];
8          if(componente[v] == -1)
9          {
10             componente[v] = componente[x];
11             DFS(v);
12         }
13     }
14 }
```

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 Represent

```
1  // Grafos
2
3  // List of edges
```

```
4
5      vector< pair<int, int> > arestas;
6      arestas.push_back(make_pair(1, 2));
7      arestas.push_back(make_pair(1, 3));
8
9  // Adjacency Matrix
10
11     int grafo[10][10];
12
13     grafo[1][2] = grafo[2][1] = 1;
14     grafo[1][3] = grafo[3][1] = 2;
15
16 // Adjacency List
17
18     vector<int> vizinhos[10];
19
20     vizinhos[1].push_back(2);
21     vizinhos[1].push_back(2);
```

2.10 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 }
```

2.11 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     {
20         int davez = -1;
21
```

```cpp
        while(!fila.empty())
        {
            int atual = fila.top().second;
            fila.pop();

            if(!processado[atual])
            {
                davez = atual;
                break;
            }
        }

        if(davez == -1)
            break;

        processado[davez] = true;

        for(int i = 0;i < (int)vizinhos[davez].size();i++)
        {

            int dist  = vizinhos[davez][i].first;
            int atual = vizinhos[davez][i].second;

            if( distancia[atual] > dist && !processado[atual])
            {
                distancia[atual] = dist;
                fila.push( pii(distancia[atual], atual) );
            }
        }
    }

    int custo_arvore = 0;
    for(int i = 1;i <= n;i++)
        custo_arvore += distancia[i];

    return custo_arvore;
}

int main(){

    cin >> n >> m;

    for(int i = 1;i <= m;i++){

        int x, y, tempo;
        cin >> x >> y >> tempo;

        vizinhos[x].pb( pii(tempo, y) );
        vizinhos[y].pb( pii(tempo, x) );
    }

    cout << Prim() << endl;

    return 0;
}
```

3 Geometria

3.1 Inter-Retas

```cpp
// Intersection between lines

typedef struct
{
    int x, y;
} pnt;

bool collinear(pnt p, pnt q, pnt r)
{
    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))
        return true;

    return false;
}

int orientation(pnt p, pnt q, pnt r)
{
    int val=(q.y-p.y)*(r.x-q.x)-(q.x-p.x)*(r.y-q.y);

    if(val==0)
        return 0;
    else if(val>0)
        return 1;
    else
        return 2;
}

bool intersect(pnt p1, pnt q1, pnt p2, pnt q2)
{
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    if(o1!=o2 and o3!=o4)
        return true;

    if(o1==0 && collinear(p1, p2, q1))
        return true;

    if(o2==0 && collinear(p1, q2, q1))
        return true;

    if(o3==0 && collinear(p2, p1, q2))
        return true;

    if(o4==0 && collinear(p2, q1, q2))
        return true;

    return false;

}
```

3.2 Rotation

```cpp
// Rotate clockwise 90 degree
(x, y) => (y, -x)

// Rotate counterclockwise 90 degree
(x, y) => (-y, x)
```

3.3 Inter-Retangulos

```cpp
typedef struct
{
    int x, y;
} Point;

bool doOverlap(Point l1, Point r1, Point l2, Point r2
    )
{
    if (l1.x>r2.x or l2.x>r1.x or l1.y<r2.y or l2.y<
    r1.y)
        return false;
    return true;
}
```

3.4 Analytic-Geometry

```
1  struct point
2  {
3      double x, y;
4      point(double _x=0, double _y=0){
5          x=_x;y=_y;
6      }
7
8      void show(){
9          cout << "x = " << x << endl;
10         cout << "y = " << y << endl;
11     }
12
13     point operator+(const point &o) const{
14         return {x + o.x, y + o.y};
15     }
16     point operator-(const point &o) const{
17         return {x - o.x, y - o.y};
18     }
19     bool operator==(const point &o) const{
20         return (x == o.x and y == o.y);
21     }
22
23 };
24
25 struct line
26 {
27     point fp, sp;
28     line(point _fp=0, point _sp=0){
29         fp=_fp;sp=_sp;
30     }
31
32     //a=y1-y2;
33     //b=x2-x1;
34     //c=x2*y1-y2*x1;
35
36 };
37
38 // Produto Escalar
39 double dot(point a, point b){
40     return a.x*b.x + a.y*b.y;
41 }
42
43 // Produto Vetorial
44 double cross(point a, point b){
45     return a.x*b.y - a.y*b.x;
46 }
47
48 // Dist entre dois pontos
49 double dist(point a, point b){
50     point c = a - b;
51     return sqrt(c.x*c.x + c.y*c.y);
52 }
53
54 // Colinearidade entre 3 pontos
55 bool collinear(point a, point b, point c){
56     return ((c.y-b.y)*(b.x-a.x)==(b.y-a.y)*(c.x-b.x))
         ;
57     // return (a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b
         .y)); // Triangle area
58     // No caso de pontos tridimensionais, usar
         produto vetorial.
59 }
60
61 // Dist entre ponto e reta
62 double distr(point a, line b){
63     double crs = cross(point(a - b.fp), point(b.sp -
         b.fp));
64     return abs(crs/dist(b.fp, b.sp));
65 }
66
67 void esq(point a, point b, point ext)
68 { // Esquerda = 1; Direita = -1; Collinear = 0;
69     ll v = a.x*b.y+b.x*ext.y+ext.x*a.y - (a.y*b.x+b.y
```

```
         *ext.x+ext.y*a.x);
70     if(v>0) return 1;
71     if(v==0) return 0;
72     return -1;
73 }
74
75 // Area de um poligono (pontos ordenados por
      adjacencia)
76 double area(vector <point> p){
77   double ret = 0;
78   for(int i=2;i<(int)p.size();i++)
79     ret += cross(p[i] - p[0], p[i-1] - p[0])/2;
80   return abs(ret);
81 }
82 // Concavo ou Convexo
83 double ccw(point a, point b, point c){
84   double ret = cross(b - a, c - b);
85   return ret < 0;
86 }
```

4   ED

4.1   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 {
19     // If 2 values are equal the query will
20     // occur first then array element
21     if (a.val == b.val)
22         return a.l > b.l;
23
24     // Otherwise sorted in descending order.
25     return a.val > b.val;
26 }
27
28 // Updates the node of BIT array by adding
29 // 1 to it and its ancestors.
30 void update(int* BIT, int n, int idx)
31 {
32     while (idx <= n) {
33         BIT[idx]++;
34         idx += idx & (-idx);
35     }
36 }
37 // Returns the count of numbers of elements
38 // present from starting till idx.
39 int query(int* BIT, int idx)
40 {
41     int ans = 0;
42     while (idx) {
43         ans += BIT[idx];
44
45         idx -= idx & (-idx);
46     }
47     return ans;
48 }
```

```
49
50  // Function to solve the queries offline
51  void solveQuery(int arr[], int n, int QueryL[],
52                  int QueryR[], int QueryK[], int q)
53  {
54      // create node to store the elements
55      // and the queries
56      node a[n + q + 1];
57      // 1-based indexing.
58
59      // traverse for all array numbers
60      for (int i = 1; i <= n; ++i) {
61          a[i].val = arr[i - 1];
62          a[i].pos = 0;
63          a[i].l = 0;
64          a[i].r = i;
65      }
66
67      // iterate for all queries
68      for (int i = n + 1; i <= n + q; ++i) {
69          a[i].pos = i - n;
70          a[i].val = QueryK[i - n - 1];
71          a[i].l = QueryL[i - n - 1];
72          a[i].r = QueryR[i - n - 1];
73      }
74
75      // In-built sort function used to
76      // sort node array using comp function.
77      sort(a + 1, a + n + q + 1, comp);
78
79      // Binary Indexed tree with
80      // initially 0 at all places.
81      int BIT[n + 1];
82
83      // initially 0
84      memset(BIT, 0, sizeof(BIT));
85
86      // For storing answers for each query( 1-based
        indexing ).
87      int ans[q + 1];
88
89      // traverse for numbers and query
90      for (int i = 1; i <= n + q; ++i) {
91          if (a[i].pos != 0) {
92
93              // call function to returns answer for
        each query
94              int cnt = query(BIT, a[i].r) - query(BIT,
        a[i].l - 1);
95
96              // This will ensure that answer of each
        query
97              // are stored in order it was initially
        asked.
98              ans[a[i].pos] = cnt;
99          }
100         else {
101             // a[i].r contains the position of the
102             // element in the original array.
103             update(BIT, n, a[i].r);
104         }
105     }
106     // Output the answer array
107     for (int i = 1; i <= q; ++i) {
108         cout << ans[i] << endl;
109     }
110 }
111
112 // Driver Code
113 int main()
114 {
115     int arr[] = { 7, 3, 9, 13, 5, 4 };
116     int n = sizeof(arr) / sizeof(arr[0]);
```

```
117
118     // 1-based indexing
119     int QueryL[] = { 1, 2 };
120     int QueryR[] = { 4, 6 };
121
122     // k for each query
123     int QueryK[] = { 6, 8 };
124
125     // number of queries
126     int q = sizeof(QueryL) / sizeof(QueryL[0]);
127
128     // Function call to get
129     solveQuery(arr, n, QueryL, QueryR, QueryK, q);
130
131     return 0;
132 }
```

4.2 Iterative-SegTree

```
1  // Segment Tree Iterativa - Range maximum query
2
3  #define N 100010
4
5  struct Segtree
6  {
7      int t[2*N]={0};
8
9      void build()
10     {
11         for(int i=N-1; i>0; i--)
12             t[i]=max(t[i<<1], t[1<<1|1]);
13     }
14
15     int query(int l, int r)
16     {
17         int ans=0;
18         for(i+=N, r+=N; l<r; l>>=1, r>>=1)
19         {
20             if(l&1)
21                 ans=max(ans, t[l++]);
22             if(r&1)
23                 ans=max(ans, t[--r]);
24         }
25
26         return ans;
27     }
28
29     void update(int p, int value)
30     {
31         for(t[p+=n]=value; p>1; p>>=1)
32             t[p>>1]= max(t[p], t[p^1]);
33     }
34
35 };
36
37 int main()
38 {
39     Segtree st;
40
41     for(int i=0;i<n;i++)
42     {
43         cin >> aux;
44         st.t[N+i]=aux; //Leaves are stored in
        continuous nodes with indices starting with N
45     }
46
47     st.build();
48     x = st.query(inicio, fim);
49     st.update(ind, value);
50
51 }
```

4.3 Recursive-SegTree

```cpp
// Segment Tree Recursiva - Range maximum query

vector<int> val(MAX, 0);
vector<int> vet(N);

void monta(int i, int j, int no)
{
    if(i==j)
    {
        val[no]=vet[i];
        return;
    }

    int esq = 2*no;
    int dir = 2*no+1;
    int meio = (i+j)/2;

    monta(i, meio, esq);
    monta(meio+1, j, dir);

    val[no]=max(val[esq], val[dir]);
}

void atualiza(int no, int i, int j, int pos, int
    novo_valor)
{
    if(i==j)
    {
        val[no]=novo_valor;
    }else
    {
        int esq = 2*no;
        int dir = 2*no+1;
        int meio = (i+j)/2;

        if(pos<=meio)
            atualiza(esq, i, meio, pos, novo_valor);
        else
            atualiza(dir, meio+1, j, pos, novo_valor)
    ;

        if(val[esq]>val[dir])
            val[no]=val[esq];
        else
            val[no]=val[dir];
    }
}

int consulta(int no, int i, int j, int A, int B)
{
    if(i>B || j<A)
        return -1;
    if(i>=A and j<=B)
        return val[no];

    int esq = 2*no;
    int dir = 2*no+1;
    int meio = (i+j)/2;

    int resp_esq = consulta(esq, i, meio, A, B);
    int resp_dir = consulta(dir, meio+1, j, A, B);

    if(resp_dir==-1)
        return resp_esq;
    if(resp_esq==-1)
        return resp_dir;

    if(resp_esq>resp_dir)
        return resp_esq;
    else
        return resp_dir;
}

int main()
{
    monta(1, N, 1);
    atualiza(1, 1, N, pos, valor);
    x = consulta(1, 1, N, inicio, fim);

}
```

4.4 Delta-Encoding

```cpp
// Delta encoding

for(int i=0;i<q;i++)
{
    int l,r,x;
    cin >> l >> r >> x;
    delta[l] += x;
    delta[r+1] -= x;
}

int atual = 0;

for(int i=0;i<n;i++)
{
    atual += delta[i];
    v[i] += atual;
}
```

4.5 Seg-Tree-Farao

```cpp
typedef struct
{
    pii prefix, sufix, total, maximo;
} no;

int noleft[MAX], noright[MAX]; //Guarda os valores
    dos nos para que nao sejam calculados novamente
    nas querys
int v[MAX];
no arvore[MAX];

pii somar(pii a, pii b) // une pairs
{
    return mp(a.f+b.f, a.s+b.s);
}

no une(no l, no r)
{
    if(l.total.s==0)
        return r;
    if(r.total.s==0)
        return l;

    no m;

    m.prefix = max(l.prefix, somar(l.total, r.prefix)
    ); //prefixo
    m.sufix = max(r.sufix, somar(r.total, l.sufix));
    //sufixo
    m.total = somar(l.total, r.total); //Soma de
    todos os elementos da subarvore
    m.maximo = max(max(l.maximo, r.maximo), somar(l.
    sufix, r.prefix)); //Resultado para cada
    subarvore

    return m;
}

no makenozero()
{
```

```
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
61        arvore[n]=une(arvore[2*n], arvore[2*n+1]);
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 }
```

4.6 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, x2, y2)
24 {
25        return sum(x2,y2) - sum(x2,y1) - sum(x1,y2) + sum
       (x1,y1);
26 }
```

4.7 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 };
```

4.8 Sparse-Table

```
1 logv[1] = 0; // pre-computar tabela de log
2 for (int i = 2; i <= MAXN; i++)
3        logv[i] = logv[i/2] + 1;
4
5 int logv[MAXN+1];
6 int st[MAXN][K];
7
8 // operacao da sparse table deve ser idempotente ->
       op(x, x) = x
9 void precompute(int N) { //
10   for (int i = 0; i < N; i++)
11        st[i][0] = array[i];
```

```
12     int k = logv[N];
13     for (int j = 1; j <= k; j++)
14         for (int i = 0; i + (1 << j) <= N; i++)
15             st[i][j] = max(st[i][j-1], st[i + (1 << (j
16     - 1))][j - 1]);
17 }
18
19 int query(int L, int R) {
20     int j = logv[R - L + 1];
21     int maximum = max(st[L][j], st[R - (1 << j) + 1][
       j]);
22
23     return maximum;
24 }
```

4.9 Union-Find

```
1  // Union-Find Functions
2
3  int pai[MAX], peso[MAX];
4
5  int find(int aux)
6  {
7      if(pai[aux]==aux)
8          return aux;
9      else
10         return pai[aux]=find(pai[aux], pai);
11 }
12
13 void join(int x, int y)
14 {
15     x = find(x);
16     y = find(y);
17
18     if(pesos[x]<pesos[y])
19         pai[x] = y;
20     else if(pesos[x]>pesos[y])
21         pai[y] = x;
22     else if(pesos[x]==pesos[y])
23     {
24         pai[x] = y;
25         pesos[y]++;
26     }
27 }
28
29 int main()
30 {
31     for(int i=1;i<=N;i++)
32         pai[i]=i;
33 }
```

4.10 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    bool operator<(Query other) const
6    {
7      return make_pair(l / BLK, r) <
8      make_pair(other.l / BLK, other.r);
9    }
10 };
11
12 void add(); void remove() // implementar operacoes de
        acordo com o problema, cuidado com TLE ao
       utilizar MAP
13
14 vector<pair<int,ll>> mo() {
15   vector<pair<int,ll>> res;
```

```
16     sort(queries.begin(), queries.end());
17
18     int l = 0, r = -1;
19     for(Query q : queries) {
20         while(l > q.l) {
21             l--;
22             add(l);
23         }
24         while(r < q.r) {
25             r++;
26             add(r);
27         }
28         while(l < q.l) {
29             remove(l);
30             l++;
31         }
32         while(r > q.r) {
33             remove(r);
34             r--;
35         }
36         res.pb(mp(q.idx, RESPOSTA)); // adicionar
       resposta de acordo com o problema
37     }
38     return res; // ordernar o vetor pelo indice e
       responder queries na ordem
39 }
```

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 Linear-Diophantine-Equation

```
// Linear Diophantine Equation
int gcd(int a, int b, int &x, int &y)
{
    if (a == 0)
    {
        x = 0; y = 1;
        return b;
    }
    int x1, y1;
    int d = gcd(b%a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}

bool find_any_solution(int a, int b, int c, int &x0,
    int &y0, int &g)
{
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g)
        return false;

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

// All solutions
// x = x0 + k*b/g
// y = y0 - k*a/g
```

5.3 Factorization-sqrt

```
// Factorization of a number in sqrt(n)

int main()
{
    ll N;
    vector<int> div;

    cin >> N;

    for(ll i=2;i*i<=N;i++)
    {
        if(N%i==0)
        {
            vet.pb(i);
            while(N%i==0)
                N/=i;
        }
    }
    if(N!=1)
        vet.pb(N);

    return 0;
}
```

5.4 Modular-Exponentiation

```
// Modular exponentiaion - (x^y)%mod in O(log y)
ll power(ll x, ll y, ll mod)
{
    ll res = 1;
    x%=mod;

    while(y)
    {
        if(y&1)
```

```
            res=(res*x)%mod;

        y=y>>1;
        x=(x*x)%mod;
    }
    return res;
}
```

5.5 Miller-Habin

```
#include <bits/stdc++.h>
#define mod 1000000007
#define Pi 3.1415926535897931159979634685
#define INF 0x3f3f3f3f
#define MAX 1000010
#define f first
#define s second
#define ll long long
#define pb push_back
#define mp make_pair
#define pii pair<int, int>
#define vi vector<int>
#define vii vector< pii >
#define sws ios_base::sync_with_stdio(false);cin.tie(
    NULL)
#define forn(i, n) for(int i=0; i<(int)(n); i++)
#define mdc(a, b) (__gcd((a), (b)))
#define mmc(a, b) (((a)/__gcd(a, b)) * b)
#define endl '\n'
#define teto(a, b) (a+b-1)/b

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;
}

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

bool rabin(ll n)
{
    if(n <= 1)
```

```
63          return 1;
64      if(n <= 3)
65          return 1;
66
67      ll s=0, d=n-1;
68      while(d%2==0)
69      {
70          d/=2;
71          s++;
72      }
73
74      for(int k = 0; k < 64*4; k++)
75      {
76          ll a = (llrand()%(n - 3)) + 2;
77          ll x = fexp(a, d, n);
78          if(x != 1 and x != n-1)
79          {
80              for(int r = 1; r < s; r++)
81              {
82                  x = mul(x, x, n);
83                  if(x == 1)
84                      return 0;
85                  if(x == n-1)
86                      break;
87              }
88              if(x != n-1)
89                  return 0;
90          }
91      }
92
93      return 1;
94 }
95
96
97 int main()
98 {
99      //sws;
100     //freopen("input.txt", "r", stdin);
101     //freopen("output.txt", "w", stdout);
102
103     ll N;
104     cin >> N;
105
106     cout << rabin(N) << endl;
107
108     return 0;
109
110 }
```

5.6   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
        entao phi(m) = p-1
10     ll e = phim-1;
11     return fexp(a, e);
12 }
```

5.7   Pollard-Rho

```
1 // Pollard Rho Algorithm
2
3 #include <bits/stdc++.h>
4 #define ll long long
5
6 using namespace std;
7
8 ll llrand()
9 {
10     ll tmp = rand();
11     return (tmp << 31) | rand();
12 }
13
14 ll add(ll a, ll b, ll c)
15 {
16     return (a + b)%c;
17 }
18
19 ll mul(ll a, ll b, ll c)
20 {
21     ll ans = 0;
22     while(b)
23     {
24         if(b & 1)
25             ans = add(ans, a, c);
26         a = add(a, a, c);
27         b /= 2;
28     }
29     return ans;
30 }
31
32 ll rho(ll n)
33 {
34     ll x, c, y, d, k;
35     int i;
36     do{
37         i = 1;
38         x = llrand()%n;
39         c = llrand()%n;
40         y = x, k = 4;
41         do{
42             if(++i == k)
43             {
44                 y = x;
45                 k *= 2;
46             }
47             x = add(mul(x, x, n), c, n);
48             d = __gcd(abs(x - y), n);
49         }
50         while(d == 1);
51     }
52     while(d == n);
53
54     return d;
55 }
56
57 int main()
58 {
59     srand(time(0));
60
61     ll N;
62     cin >> N;
63
64     ll div = rho(N);
65     cout << div << " " << N/div << endl;
66
67
68     // Finding all divisors
69
70     vector<ll> div;
71
72     while(N>1 and !rabin(N))
73     {
74         ll d = rho(N);
75         div.pb(d);
76         while(N%d==0)
77             N/=d;
78     }
```

```
79        if(N!=1)
80            div.pb(N);
81
82        return 0;
83
84 }
```

5.8   Verif-primo

```
1 // prime verification sqrt(N)
2
3 bool eh_primo(long long N)
4 {
5      if(N==2)
6          return true;
7      else if(N==1 or N%2==0)
8          return false;
9      for(long long i=3;i*i<=N;i+=2)
10         if(N%i==0)
11             return false;
12     return true;
13 }
```

5.9   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;
```

5.10   Formulas

```
1 int sum_x2(ll N)
2 {
3      return (2*N*N*N + 3*N*N + N)/6;
4 }
```

5.11   FFT-golfbot

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int N = (1<<19);
6 const double two_pi = 4 * acos(0);
7
8 struct cpx
9 {
10     cpx(){}
11     cpx(double aa): a(aa){}
12     cpx(double aa,double bb):a(aa),b(bb){}
13     double a;
14     double b;
15     double modsq(void) const
16     {
17         return a*a+b*b;
18     }
19     cpx bar(void) const
20     {
21         return cpx(a,-b);
22     }
23 };
```

```
24
25 cpx b[N+100];
26 cpx c[N+100];
27 cpx B[N+100];
28 cpx C[N+100];
29 int a[N+100];
30 int x[N+100];
31 double coss[N+100], sins[N+100];
32 int n,m,p;
33
34 cpx operator +(cpx a,cpx b)
35 {
36     return cpx(a.a+b.a,a.b+b.b);
37 }
38
39 cpx operator *(cpx a,cpx b)
40 {
41     return cpx(a.a*b.a-a.b*b.b,a.a*b.b+a.b*b.a);
42 }
43
44 cpx operator /(cpx a,cpx b)
45 {
46     cpx r = a*b.bar();
47     return cpx(r.a/b.modsq(),r.b/b.modsq());
48 }
49
50 cpx EXP(int i,int dir)
51 {
52     return cpx(coss[i],sins[i]*dir);
53 }
54
55 void FFT(cpx *in,cpx *out,int step,int size,int dir)
56 {
57     if(size<1) return;
58     if(size==1)
59     {
60         out[0]=in[0];
61         return;
62     }
63     FFT(in,out,step*2,size/2,dir);
64     FFT(in+step,out+size/2,step*2,size/2,dir);
65     for(int i=0;i<size/2;++i)
66     {
67         cpx even=out[i];
68         cpx odd=out[i+size/2];
69         out[i] = even+EXP(i*step,dir)*odd;
70         out[i+size/2]=even+EXP((i+size/2)*step,dir)*
    odd;
71     }
72 }
73
74 int main()
75 {
76     for(int i=0;i<=N;++i)
77     {
78         coss[i]=cos(two_pi*i/N);
79         sins[i]=sin(two_pi*i/N);
80     }
81     while(cin >> n) // Numero de tacadas possiveis
82     {
83         fill(x,x+N+100,0);
84         fill(a,a+N+100,0);
85         for(int i=0;i<n;++i)
86         {
87             cin >> p; // Distancia das tacadas
88             x[p]=1;
89         }
90         for(int i=0;i<N+100;++i)
91         {
92             b[i]=cpx(x[i],0);
93         }
94         cin >> m; // Querys
95         for(int i=0;i<m;++i)
```

```
96          {
97              cin >> a[i]; // Distancia da query
98          }
99          FFT(b,B,1,N,1);
100         for(int i=0;i<N;++i)
101             C[i]=B[i]*B[i];
102         FFT(C,c,1,N,-1);
103         for(int i=0;i<N;++i)
104             c[i]=c[i]/N;
105         int cnt=0;
106         for(int i=0;i<m;++i)
107             if(c[a[i]].a>0.5 || x[a[i]])
108                 cnt++;
109         cout << cnt << endl;
110     }
111     return 0;
112 }
```

5.12 Modular-Factorial

```
1 // C++ program to comput n! % p using Wilson's
      Theorem
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int power(int x, unsigned int y, int p)
6 {
7     int res = 1;
8     x = x % p;
9
10    while(y > 0)
11    {
12        if(y & 1)
13            res = (res * x) % p;
14
15        y = y >> 1;
16        x = (x * x) % p;
17    }
18    return res;
19 }
20
21 int modInverse(int a, int p)
22 {
23     return power(a, p-2, p);
24 }
25
26 int modFact(int n, int p)
27 {
28     if (p <= n)
29         return 0;
30
31     int res = (p - 1);
32
33     for(int i = n + 1; i < p; i++)
34         res = (res * modInverse(i, p)) % p;
35     return res;
36 }
37
38 int main()
39 {
40     int n = 25, p = 29;
41     cout << modFact(n, p);
42     return 0;
43 }
```

5.13 Recursao-linear

```
1 vector<vl> mult(vector<vl> a, vector<vl> b, int n) {
2   vector<vl> res;
3   for(int i = 0; i < n; i++) {
4     vl tmp;
5     for(int j = 0; j < n; j++) {
```

```
6       tmp.pb(0);
7     }
8     res.pb(tmp);
9   }
10
11  for(int row = 0; row < n; row++) {
12    for(int col = 0; col < n; col++) {
13      ll val = 0;
14      for(int k = 0; k < n; k++) {
15        val += (a[row][k]*b[k][col]);
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     vector<vl> id;
27     for(int i = 0; i < n; i++) {
28       vl tmp;
29       for(int j = 0; j < n; j++) {
30         if(i == j)
31           tmp.pb(1);
32         else
33           tmp.pb(0);
34       }
35       id.pb(tmp);
36     }
37
38     return id;
39   }
40
41   vector<vl> res = fexp(b, e/2, n);
42   res = mult(res, res, n);
43
44   if(e%2)
45     res = mult(res, b, n);
46
47   return res;
48 }
49
50 // k = tamanho da recorrencia/matriz, n = n-esimo
      termo
51 // f(n) = c1*f(n-1) + c2*f(n-2) + ... + ck*f(n-k)
52 // base -> [f(k-1), f(k-2), ..., f(0)]
53 // coeficientes -> [c1, c2, ..., ck]
54 vl solve(int k, int n, vl base, vl coef) {
55   vector<vl> inicial;
56   inicial.pb(coef);
57   for(int row = 0; row < k-1; row++) {
58     vl tmp;
59     for(int col = 0; col < k; col++) {
60       if(col == row)
61         tmp.pb(1);
62       else
63         tmp.pb(0);
64     }
65     inicial.pb(tmp);
66   }
67
68   vector<vl> matexp = fexp(inicial, max(0, n-k+1), k)
      ;
69   vl res(k);
70
71   for(int row = 0; row < k; row++) {
72     ll val = 0;
73     for(int aux = 0; aux < k; aux++) {
74       val += matexp[row][aux]*base[aux];
75     }
76     res[row] = val; // res = (f(n), f(n-1), ..., f(n-
```

```
        k+1))
77    }
78
79    return res;
80 }
```

5.14   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   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();
```

6.2   Bitwise

```
1  // Bitwise
2
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)
10     Right shift -  b>>1  // The result is 00000100
       (4)
11
12     // Exchange two int variables
13
14         a^=b;
15         b^=a;
16         a^=b;
17
18     // Even or Odd
19
20         (x & 1)? printf("Odd"): printf("Even");
```

```
21
22     // Turn on the j-th bit
23
24         int S = 34; //(100010)
25         int j = 3;
26
27         S = S | (1<<j);
28
29     // Turn off the j-th bit
30
31         int S = 42; //(101010)
32         int j = 1;
33
34         S &= ~(1<<j)
35
36         S == 40 //(101000)
37
38     // Check the j-th element
39
40         int S = 42; //(101010)
41         int j = 3;
42
43         T = S & (1<<j); // T = 0
44
45     // Least significant bit (lsb)
46
47         int lsb(int x){ return x&-x; }
48
49     // Exchange o j-th element
50
51         S ^= (1<<j)
52
53     // Position of the first bit on
54
55         T = (S & (-S))
56         T -> 4 bit ligado //(1000)
57
58     // Most significant digit of N
59
60         double K = log10(N);
61         K = K - floor(K);
62         int X = pow(10, K);
63
64     // Number of digits in N
65
66         X =floor(log10(N)) + 1;
67
68     // Power of two
69
70         bool isPowerOfTwo(int x){ return x && (!(x&(x
       -1))); }
71
72     // Turn off the first bit 1
73         m = m & (m-1);
74
75     // Built-in functions
76
77         // Number of bits 1
78         __builtin_popcount()
79         __builtin_popcountll()
80
81         // Number of leading zeros
82         __builtin_clz()
83         __builtin_clzll()
84
85         // Number of trailing zeros
86         __builtin_ctz()
87         __builtin_ctzll()
88
89  // floor(log2(x))
90
91     int flog2(int x){ return 32-1-__builtin_clz(x
       ); }
```

```
92
93         int flog2ll(ll x){ return 64-1-
       __builtin_clzll(x); }
```

7   Strings

7.1   KMP

```
1 vector<int> preffix_function(const string &s){
2     int n = s.size(); vector<int> b(n+1);
3     b[0] = -1; int i = 0, j = -1;
4     while(i < n){
5         while(j >= 0 && s[i] != s[j]) j = b[j];
6         b[++i] = ++j;
7     }
8     return b;
9 }
10 void kmp(const string &t, const string &p){
11     vector<int> b = preffix_function(p);
12     int n = t.size(), m = p.size();
13     int j = 0;
14     for(int i = 0; i < n; i++){
15         while(j >= 0 && t[i] != p[j]) j = b[j];
16         j++;
17         if(j == m){
18
19             j = b[j];
20         }
21     }
22 }
```

7.2   LCS

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

7.3   Pal-int

```
1 bool ehpalindromo(ll n)
2 {
3     if(n<0)
4         return false;
5
6     int divisor = 1;
7     while(n/divisor >= 10)
8         divisor *= 10;
9
10     while(n != 0)
11     {
12         int leading = n / divisor;
13         int trailing = n % 10;
14
15         if(leading != trailing)
16             return false;
17
18         n = (n % divisor)/10;
19
20         divisor = divisor/100;
21     }
22
23     return true;
24 }
```

7.4   Z-Func

```
1 vector<int> z_algo(const string &s)
2 {
3     int n = s.size();
4     int L = 0, R = 0;
5     vector<int> z(n, 0);
6     for(int i = 1; i < n; i++)
7     {
8         if(i <= R)
9             z[i] = min(z[i-L], R - i + 1);
10         while(z[i]+i < n && s[ z[i]+i ] == s[ z[i] ])
11             z[i]++;
12         if(i+z[i]-1 > R)
13         {
14             L = i;
15             R = i + z[i] - 1;
16         }
17     }
18     return z;
19 }
```

7.5   Hash

```
1 ll compute_hash(string const& s) {
2     const ll p = 31; // primo, melhor = perto da
    quantidade de caracteres
3     const ll m = 1e9 + 9; // maior mod = menor
    probabilidade de colisao
4     ll hash_value = 0;
5     ll p_pow = 1;
6     for (char c : s) {
7         hash_value = (hash_value + (c - 'a' + 1) *
    p_pow) % m;
8         p_pow = (p_pow * p) % m;
9     }
10     return hash_value;
11 }
```