Material Maratona 2017

Faculdade de Computação
Universidade Federal de Uberlândia

Sumário

Capítulo 1

DP - Otimizações

1.1 Convex Hull Trick Decrescente - Mínimo $O(nlogn)$

Listagem 1.1: Convex Hull Trick Decrescente - Mínimo (n log n)

```
1  typedef long long int ll;
2
3  struct pt{
4      ll x, y;
5      pt(){x=y=0;}
6      pt(ll a, ll b) : x(a), y(b) {}
7  };
8
9  struct line{
10     ll a, b;
11     line(){a=b=0;}
12     line(ll i, ll j) : a(i), b(j) {}
13
14     ll value_at(ll x){
15         return a*x + b;
16     }
17 };
18
19 struct cht{
20     int sz;
21     vector<line> ch;
22
23     cht(){ch.clear(); sz=0;}
24
25     bool can_pop(line ant, line top, line at){
26         return (top.b - ant.b)*(ant.a - at.a) >= (at.b - ant.b)*(ant.a -
               top.a);
27     }
28
29     void add_line(line L){//retas ordenadas decrescente
30         while(sz>1 && can_pop(ch[sz-2], ch[sz-1], L)){
31             ch.pop_back();
32             sz--;
33         }
34         ch.push_back(L);
```

```
35          sz++;
36      }
37
38      ll query(int x){//query de minimo
39          int ini=0, fim = sz-1, meio, ans;
40          ans = sz-1;
41
42          while(ini<=fim){
43              meio = (ini+fim)/2;
44              if(ch[meio].value_at(x) > ch[meio+1].value_at(x)){
45                  ini = meio+1;
46              }
47              else{
48                  fim = meio-1;
49                  ans = meio;
50              }
51          }
52          return ch[ans].value_at(x);
53      }
54  };
```

1.2 Convex Hull Trick Decrescente - Mínimo (linear)

Listagem 1.2: Convex Hull Trick Decrescente - Mínimo (linear)

```
1  typedef long long int ll;
2
3  struct pt{
4      ll x, y;
5      pt(){x=y=0;}
6      pt(ll a, ll b) : x(a), y(b) {}
7  };
8
9  struct line{
10     ll a, b;
11     line(){a=b=0;}
12     line(ll i, ll j) : a(i), b(j) {}
13
14     ll value_at(ll x){
15         return a*x + b;
16     }
17 };
18
19 struct cht{
20     int sz, pos;
21     vector<line> ch;
22
23     cht(){ch.clear(); sz=pos=0;}
24
25     bool can_pop(line ant, line top, line at){
26         return (top.b - ant.b)*(ant.a - at.a) >= (at.b - ant.b)*(ant.a -
               top.a);
27     }
28
29     void add_line(line L){
```

```
30          while(sz>1 && can_pop(ch[sz-2], ch[sz-1], L)){
31              ch.pop_back();
32              sz--;
33          }
34          ch.push_back(L);
35          sz++;
36      }
37
38      ll query(int x){
39          int ans = sz-1;
40          for(int i=pos; i<sz-1; i++){
41              if(ch[i].value_at(x) > ch[i+1].value_at(x)) pos++;
42              else{
43                  ans=i;
44                  break;
45              }
46          }
47          return ch[ans].value_at(x);
48      }
49  };
```

1.3 Convex Hull Trick Crescente - Máximo $O(nlogn)$

Listagem 1.3: Convex Hull Trick Crescente - Máximo (n log n)

```
1  typedef long long int ll;
2
3  struct line{
4      ll a, b;
5      line(){a=b=0;}
6      line(ll i, ll j) : a(i), b(j) {}
7
8      ll value_at(ll x){
9          return a*x + b;
10     }
11 };
12
13 struct cht{
14     int sz;
15     vector<line> ch;
16
17     cht(){ch.clear(); sz=0;}
18
19     bool can_pop(line ant, line top, line at){
20         return (top.b - ant.b)*(ant.a - at.a) >= (at.b - ant.b)*(ant.a -
               top.a);
21     }
22
23     void add_line(line L){
24         while(sz>1 && can_pop(ch[sz-2], ch[sz-1], L)){
25             ch.pop_back();
26             sz--;
27         }
28         ch.push_back(L);
29         sz++;
```

```
30      }
31
32      ll query(ll x){
33          int ini=0, fim = sz-1, meio, ans;
34          ans = sz-1;
35
36          while(ini<=fim){
37              meio = (ini+fim)/2;
38              if(ch[meio].value_at(x) < ch[meio+1].value_at(x)){
39                  ini = meio+1;
40              }
41              else{
42                  fim = meio-1;
43                  ans = meio;
44              }
45          }
46          return ch[ans].value_at(x);
47      }
48  };
```

1.4 Convex Hull Trick Crescente - Máximo (linear)

Listagem 1.4: Convex Hull Trick Crescente - Máximo (linear)

```
1  typedef long long int ll;
2
3  struct line{
4      ll a, b;
5      line(){a=b=0;}
6      line(ll i, ll j) : a(i), b(j) {}
7
8      ll value_at(ll x){
9          return a*x + b;
10     }
11 };
12
13 struct cht{
14     int sz, pos;
15     vector<line> ch;
16
17     cht(){ch.clear(); sz=pos=0;}
18
19     bool can_pop(line ant, line top, line at){
20         return (top.b - ant.b)*(ant.a - at.a) >= (at.b - ant.b)*(ant.a -
               top.a);
21     }
22
23     void add_line(line L){
24         while(sz>1 && can_pop(ch[sz-2], ch[sz-1], L)){
25             ch.pop_back();
26             sz--;
27         }
28         ch.push_back(L);
29         sz++;
30     }
```

```
31
32      ll query(int x){
33          int ans = sz-1;
34          for(int i=pos; i<sz-1; i++){
35              if(ch[i].value_at(x) < ch[i+1].value_at(x)) pos++;
36              else{
37                  ans=i;
38                  break;
39              }
40          }
41          return ch[ans].value_at(x);
42      }
43  };
```

1.5 Convex Hull Trick Crescente - Mínimo (variação)

Listagem 1.5: Convex Hull Trick Crescente - Mínimo(variação)

```
1  typedef long long int ll;
2
3  struct line{
4      ll a, b, id;
5      line(){ a=b=0;}
6      line(ll x, ll y, ll c) : a(x), b(y), id(c) {}
7
8      ll value_at(ll x){
9          return a*x + b;
10     }
11 };
12
13 struct cht{
14     int sz, pos;
15     vector<line> ch;
16
17     cht(){ch.clear(); sz=pos=0;}
18
19     bool can_pop(line ant, line top, line at){
20         ll p = (ant.b - at.b)*(top.a - ant.a);
21         ll q = (ant.b - top.b)*(at.a - ant.a);
22         return p>=q;
23     }
24
25     void add_line(line at){
26         ll sz = ch.size();
27         while(sz>1 && can_pop(ch[sz-2], ch[sz-1], at)) {
28             ch.pop_back();
29             sz--;
30         }
31         ch.push_back(at);
32     }
33
34     bool check(ll i, ll x){
35         ll at = ch[i].value_at(x), ant = ch[i-1].value_at(x);
36         if(ant > at) return true;
37         return false;
```

```
38      }
39
40      ll query(ll x){
41          ll ini, fim, meio, meio_, sz = ch.size();
42          ini = 1; fim = sz-1;
43
44          ll ans=0;
45          while(ini<=fim){
46              meio = (ini+fim)/2;
47              if(check(meio, x)) {
48                  ini = meio+1;
49                  ans = meio;
50              }
51              else
52                  fim = meio-1;
53          }
54          return ch[ans].value_at(x);
55      }
56  };
```

1.6 Divide and Conquer

Listagem 1.6: Divide and Conquer

```
1  typedef long long int ll;
2
3  ll n, K, dp[2][N];
4
5  void build_cost(){
6      //depende do problema
7  }
8
9  ll get_cost(ll i, ll j){
10     //depende do problema
11 }
12
13 void func(ll at, ll l, ll r, ll optL, ll optR){
14     if(l>r) return;
15     ll mid = (l+r)>>1;
16     ll opt = 1;
17     ll best = dp[at^1][mid];
18
19     for(ll i=optL; i<=min(mid-1, optR); i++){
20         ll c = get_cost(i+1, mid);
21         if(dp[at^1][i]+c < best){
22             best = dp[at^1][i] + c;
23             opt = i;
24         }
25     }
26
27     dp[at][mid] = best;
28
29     func(at, l, mid-1, optL, opt);
30     func(at, mid+1, r, opt, optR);
31 }
```

```
32
33 int main(){
34     while(scanf("%lld %lld", &n, &K)!=EOF){
35       //entrada
36         build_cost();
37         for(ll i=1; i<=n; i++){
38             dp[1][i] = get_cost(1, i);
39         }
40         ll at=0;
41         for(ll k=2; k<=K; k++){
42             func(at, 1, n, 1, n);
43             at^=1;
44         }
45         at^=1;
46         printf("%lld\n", dp[at][n]);
47     }
48 }
```

1.7 Knuth

Listagem 1.7: Knuth

```
1  typedef long long int ll;
2
3  int n, opt[N][N];
4  ll acc[N], dp[N][N];
5  string answer;
6
7  void knuth(){
8      for(int i=1; i<=n; i++) {
9          dp[i][i] = acc[i]-acc[i-1];
10         opt[i][i] = i;
11     }
12
13     for(int s = 2; s<=n; s++){
14         for(int l=1; l+s-1<=n; l++){
15             int r = l+s-1;
16
17             int optL = opt[l][r-1];
18             int optR = opt[l+1][r];
19             int opt_ = optL;
20             ll best = inf;
21
22             for(int i=optL; i<=min(optR, r-1); i++){
23                 if(dp[l][i] + dp[i+1][r] < best){
24                     best = dp[l][i]+dp[i+1][r];
25                     opt_ = i;
26                 }
27             }
28             if(best == inf) best = 0;
29             opt[l][r] = opt_;
30             dp[l][r] = best+acc[r]-acc[l-1];
31         }
32     }
33 }
```

```
34
35 void solve(int l ,int r){//recupera resposta
36     if(r<l) return;
37     if(l == r){
38         cout << answer << endl;
39         return;
40     }
41
42     answer.push_back('0');
43     solve(l, opt[l][r]);
44     answer.back()='1';
45     solve(opt[l][r]+1, r);
46     answer.pop_back();
47 }
48
49 int main(){
50     ios_base::sync_with_stdio(0); cin.tie(0);
51     while(cin >> n){
52         for(int i=1; i<=n; i++){
53             cin >> acc[i];
54             acc[i]+=acc[i-1];
55         }
56
57         knuth();
58         solve(1, n);
59     }
60 }
```

Capítulo 2

Estruturas de Dados

2.1 BIT

Listagem 2.1: BIT

```
1  struct BIT{
2      #define LOGMAX 22
3      #define N 101010
4
5      int bit[N];
6      BIT(){};
7
8      void clear(){
9          memset(bit, 0, sizeof bit);
10     }
11
12     void update(int pos, int v){
13         for(; pos<N; pos+=(pos&(-pos))) bit[pos]+=v;
14     }
15
16     int sum(int pos){
17         int s=0;
18         for(; pos; pos-=(pos&(-pos))) s+=bit[pos];
19         return s;
20     }
21
22     int kth(int k){
23
24         int ans=0;
25         for(int j=LOGMAX; j>=0; j--){
26             if(ans+(1<<j) >= N) continue;
27
28             if(bit[ans+(1<<j)]<k){
29                 ans+=(1<<j);
30                 k-=bit[ans];
31             }
32         }
33         return ans+1;
34     }
35
```

```
36      int query(int l, int r){
37          if(l > r) return 0;
38          return sum(r) - sum(l-1);
39      }
40  };
```

2.2 BIT2D

Listagem 2.2: BIT2D

```
1  struct BIT2D{
2      #define MAXN 1010
3
4      int bit[MAXN][MAXN];
5      BIT2D(){}
6
7      void reset(){
8          memset(bit, 0, sizeof bit);
9      }
10
11     void update(int a, int b, int val){
12         for(int x = a;  x < MAXN;  x+= (x & -x) ){
13
14             for(int y = b;  y < MAXN;  y+= (y & -y) ){
15                 bit[x][y] += val;
16             }
17
18         }
19     }
20
21     int sum(int a, int b){
22         int ans = 0;
23         for(int x=a;  x;  x-= (x & -x)){
24             for(int y=b;  y;  y-= (y & -y) ){
25                 ans += bit[x][y];
26             }
27
28         }
29         return ans;
30     }
31
32     int query(int i1, int j1, int i2, int j2){
33         return sum(i2, j2) + sum(i1-1, j1-1) - sum(i1-1,j2) - sum(i2,j1-1)
               ;
34     }
35 };
```

2.3 Exponenciação de Matriz

Listagem 2.3: Exponenciação de Matriz

```
struct mat{
    ll m[N][N];
    mat(){ memset(m, 0, sizeof m); }
};

//obs: considerar passagem de parametros por referencia
//multiplica duas matrizes (na x c)*(c x mb)
mat mult(mat a, mat b, ll na, ll mb, ll c){
    mat ans;
    for(ll i=0; i<na; i++)
        for(ll j=0; j<mb; j++)
            for(ll k=0; k<c; k++)
                ans.m[i][j] = (ans.m[i][j] + a.m[i][k]*b.m[k][j])%MOD;
    return ans;
}

mat identity(){
    mat ans;
    for(ll i=0; i<N; i++) ans.m[i][i] = 1;
    return ans;
}

//obs: considerar passagem de parametros por referencia
mat mat_pow(mat base, ll p){
    mat ans = identity();
    while(p>0){
        if(p&1) ans = mult(ans, base, N, N, N);
        base = mult(base, base, N, N, N);
        p>>=1;
    }
    return ans;
}

mat build(){
    //constroi a matriz de transição. Depende do problema
}

int main(){
    mat base, ans, T;
    T = build();//monta a matriz de transição

    //monta a matriz do caso base

    ans = mat_pow(T, expoente);//exponencia
    ans = mult(ans, base, _, _, _);//multiplica pelo caso base
}
```

2.4 Merge Sort Tree

Listagem 2.4: Merge Sort Tree

```
int n, k, q;
int v[MAXN];

struct MERGESORT_TREE{
```

```
5       vector<int> st[4*MAXN];
6
7       MERGESORT_TREE(){}
8       void reset(){
9           for (int i = 0; i < 4*MAXN; i++){
10              st[i].clear();
11          }
12      }
13
14      vector<int> merge(const vector<int> &a, const vector<int> &b){
15          vector<int> ans;
16          int i = 0, j = 0;
17          while (ans.size() < k){
18              if(i==a.size() && j==b.size()) break;
19              if(i==a.size()){
20                  ans.pb(b[j++]);
21              }else if(j==b.size()){
22                  ans.pb(a[i++]);
23              }else{
24                  if(a[i] > b[j]){
25                      ans.pb(a[i++]);
26                  }else{
27                      ans.pb(b[j++]);
28                  }
29              }
30          }
31          return ans;
32      }
33
34      void build(int no, int l, int r){
35          if(l==r){
36              st[no].pb(v[l]);
37              return;
38          }
39          int nxt = 2*no;
40          int mid = (l+r)/2;
41          build(nxt, l, mid);
42          build(nxt+1, mid+1, r);
43          st[no] = merge(st[nxt], st[nxt+1]);
44      }
45
46      vector<int> query(int no, int l, int r, int i, int j){
47          vector<int> ans;
48          if(r<i || l>j) return ans;
49          if(i<=l && r<=j) return st[no];
50          int nxt = 2*no;
51          int mid = (l+r)/2;
52
53          return merge(query(nxt, l, mid, i, j), query(nxt+1, mid+1, r, i, j
                ));
54      }
55
56  };
57
58  int main(){
59      ios_base::sync_with_stdio(0);
60      cin.tie(0);
61
62      cin >> n >> k >> q;
```

```
63      MERGESORT_TREE tr;
64
65      for (int i = 0; i < n; i++)
66      {
67          cin >> v[i];
68      }
69      tr.build(1, 0, n-1);
70
71      vector<int> res;
72      int l, r;
73      ll ans;
74      for (int i = 0; i < q; i++)
75      {
76          cin >> l >> r;
77          l--; r--;
78          res.clear();
79          res = tr.query(1, 0, n-1, l, r);
80
81          ans = res[0];
82          for (int j = 1; j < res.size(); j++)
83          {
84              if(res[j]!=0){
85                  ans = (ans * 1LL * res[j])%MOD;
86              }
87          }
88
89          cout << ans << "\n";
90      }
91      return 0;
92  }
```

2.5 Segment Tree

Listagem 2.5: Segment Tree

```
1  int v[MAXN];
2
3  struct SEGTREE{
4      int st[MAXN * 4];
5
6      SEGTREE(){}
7
8      void reset(){
9          memset(st, 0, sizeof st);
10     }
11
12     int merge(int a, int b){
13         return a+b;
14     }
15
16     void build(int no, int l, int r){
17         if(l==r){
18             st[no] = v[l];
19             return;
20         }
```

```
21          int mid = (l+r)>>1;
22          int nxt = no<<1;
23          build(nxt, l, mid);
24          build(nxt+1, mid+1, r);
25          st[no] = merge(st[nxt], st[nxt+1]);
26      }
27
28      int query(int no, int l, int r, int i, int j){
29          if(i<=l && r<=j) return st[no];
30          if(i>r || j<l) return 0;
31
32          int mid = (l+r)>>1;
33          int nxt = no<<1;
34          return merge(query(nxt, l, mid, i, j), query(nxt+1, mid+1, r, i, j
                ));
35      }
36
37      void update(int no, int l, int r, int pos, int val){
38          if(pos<l || pos>r) return;
39          if(l==r){
40              st[no] = val;
41              return;
42          }
43
44          int mid=(l+r)>>1;
45          int nxt = no<<1;
46          update(nxt, l, mid, pos, val);
47          update(nxt+1, mid+1, r, pos, val);
48          st[no] = merge(st[nxt], st[nxt+1]);
49      }
50  };
```

2.6 Segment Tree + Lazy Propagation

Listagem 2.6: Segment Tree + Lazy Propagation

```
1  int v[MAXN];
2
3  struct SEGTREE_LAZY{
4      int st[MAXN * 4];
5      int lazy[MAXN * 4];
6
7      SEGTREE_LAZY(){}
8
9      void reset(){
10         memset(st, 0, sizeof st);
11         memset(lazy, 0, sizeof lazy);
12     }
13
14     int merge(int a, int b){
15         return a+b;
16     }
17
18     void build(int no, int l, int r){
19         if(l==r){
```

```
20              st[no] = v[l];
21              lazy[no] = 0;
22              return;
23          }
24          int mid = (l+r)>>1;
25          int nxt = no<<1;
26
27          build(nxt, l, mid);
28          build(nxt+1, mid+1, r);
29
30          st[no] = merge(st[nxt], st[nxt+1]);
31          lazy[no] = 0;
32      }
33
34      void propagate(int no, int l, int r){
35          if(!lazy[no]) return;
36
37          int mid = (l+r)>>1;
38          int nxt = no<<1;
39
40          st[no] += (r-l+1)*lazy[no];
41
42          if(l!=r){
43              lazy[nxt] += lazy[no];
44              lazy[nxt+1] += lazy[no];
45          }
46          lazy[no] = 0;
47      }
48
49      int query(int no, int l, int r, int i, int j){
50          propagate(no, l, r);
51
52          if(i<=l && r<=j) return st[no];
53          if(i>r || j<l) return 0;
54
55          int mid = (l+r)>>1;
56          int nxt = no<<1;
57          return merge(query(nxt, l, mid, i, j), query(nxt+1, mid+1, r, i, j
                ));
58      }
59
60      void update(int no, int l, int r, int i, int j, int val){
61          propagate(no, l, r);
62
63          if(i>r || j<l) return;
64          if(i<=l && r<=j){
65              lazy[no] += val;
66              propagate(no, l, r);
67              return;
68          }
69
70          int mid = (l+r)>>1;
71          int nxt = no<<1;
72
73          update(nxt, l, mid, i, j, val);
74          update(nxt+1, mid+1, r, i, j, val);
75
76          st[no] = merge(st[nxt], st[nxt+1]);
77      }
```

```
78 };
```

2.7 Segment Tree Dinâmica

Listagem 2.7: Segment Tree Dinâmica

```
1  #define N 101010
2
3  typedef long long int ll;
4
5  struct no{
6      ll val, lazy;
7      no *left, *right;
8      no() : val(0), lazy(0), left(NULL), right(NULL) {}
9
10     void do_lazy(int l, int r){
11         if(lazy==0) return;
12         val+= ((r-l)+1)*lazy;
13         if(l<r){
14             if(!left) left = new no();
15             if(!right) right = new no();
16             left->lazy+=lazy;
17             right->lazy+=lazy;
18         }
19         lazy = 0;
20     }
21
22     void update(int l, int r, int a, int b, ll v){
23         do_lazy(l, r);
24
25         if(l>b || r<a) return;
26         if(a<=l && b>=r) {
27             lazy+=v;
28             do_lazy(l, r);
29             return;
30         }
31
32         int mid = (l+r)>>1;
33         if(left == NULL) left = new no();
34             left->update(l, mid, a, b, v);
35
36         if(right == NULL) right = new no();
37             right->update(mid+1, r, a, b, v);
38
39         val = left->val + right->val;
40     }
41
42     ll query(int l, int r, int a, int b){
43         do_lazy(l, r);
44
45         if(l>b || r<a) return 0;
46         if(a<=l && b>=r) return val;
47
48         int mid = (l+r)>>1;
49         ll x = (left) ? left->query(l, mid, a, b) : 0;
```

```
50          ll y = (right) ? right->query(mid+1, r, a, b) : 0;
51          return x+y;
52      }
53
54      void destroy(){//nem todo problema precisa, mas pode dar merda se nao
            destruir
55          if(left) {
56              left->destroy();
57              free(left);
58          }
59          if(right) {
60              right->destroy();
61              free(right);
62          }
63          return;
64      }
65  };
```

2.8 Sparse Table

Listagem 2.8: Sparse Table

```
1  struct SparseTable{
2      #define N 101010
3      #define M 20
4
5      int n, table[N][M];
6
7      SparseTable() : n(0) {}
8
9      SparseTable(int a) : n(a) {}
10
11     // pressupoe que table[i][0] ja esteja calculado pra todo i
12     void build(){
13         for(int j=1; j<M; j++){
14             // 0-indexado. Pra 1-indexado faça: for(int i=1;i+(1<<j)<=n+1;
                   i++)
15             for(int i=0; i+(1<<j)<=n; i++){
16                 // se for soma, eh so trocar min por soma
17                 table[i][j]= min(table[i][j-1],table[i+(1<<(j-1))][j-1]);
18             }
19         }
20     }
21
22     int query_min(int l, int r){// pressupoe que l<=r
23         // se as variaveis forem long long, faça 63 - __builtin_clz(r-l+1)
24         int k = 31 - __builtin_clz(r-l+1);
25
26         return min(table[l][k], table[r-(1<<k)+1][k]);
27     }
28
29     //pressupoe que a sparse table calculada seja de soma
30     int query_soma(int l, int r){
31         int ans=0;
32         for(int j=M-1; j>=0; j--){
```

```
33              if(l+(1<<j) > r+1) continue;
34              ans+=table[l][j];
35              l+=(1<<j);
36          }
37          return ans;
38      }
39  };
```

2.9 Persistent Segment Tree - Estática

Listagem 2.9: Persistent Segment Tree - Estática

```
1  /*
2   *  SPOJ - MKTHNUM
3   */
4
5  #include <bits/stdc++.h>
6
7  using namespace std;
8
9  #define N 101010
10
11 struct no{
12     int l, r, val;
13     no() : l(0), r(0), val(0) {}
14 }st[10101010];
15
16 int n, q, root[N], vet[N], inv[N], aux[N], nxt;
17
18 int update(int no1, int l, int r, int pos, int v){
19     int no2 = nxt++;
20     st[no2] = st[no1];
21     if(l == r){
22         st[no2].val+=v;
23         return no2;
24     }
25
26     int mid = (l+r)>>1;
27     if(pos<=mid) st[no2].l = update(st[no1].l, l, mid, pos, v);
28     if(pos>mid) st[no2].r = update(st[no1].r, mid+1, r, pos, v);
29
30     st[no2].val = st[st[no2].l].val + st[st[no2].r].val;
31     return no2;
32 }
33
34 int query_k(int no1, int no2, int l, int r, int k){
35     if(l == r) return l;
36     int x = st[st[no2].l].val - st[st[no1].l].val;
37     int mid = (l+r)>>1;
38
39     if(x >= k) return query_k(st[no1].l, st[no2].l, l, mid, k);
40     return query_k(st[no1].r, st[no2].r, mid+1, r, k-x);
41 }
42
43 int main(){
```

```
44      scanf("%d %d", &n, &q);
45      for(int i=1; i<=n; i++){
46          scanf("%d", &vet[i]);
47          aux[i] = vet[i];
48      }
49
50      sort(aux+1, aux+n+1);
51      root[0] = 0;
52      nxt = 1;
53      for(int i=1; i<=n; i++){
54          int a = lower_bound(aux+1, aux+n+1, vet[i]) - aux;
55          inv[a] = vet[i];
56          vet[i] = a;
57          root[i] = update(root[i-1], 1, n, a, 1);
58      }
59
60      int a, b, c;
61      for(int i=0; i<q; i++){
62          scanf("%d %d %d", &a, &b, &c);
63          printf("%d\n", inv[query_k(root[a-1], root[b], 1, n, c)]);
64      }
65  }
```

2.10 Persistent Segment Tree - Dinâmica

Listagem 2.10: Persistent Segment Tree - Dinâmica

```
1  /*
2   *  SPOJ - MKTHNUM
3   */
4
5  #include <bits/stdc++.h>
6
7  using namespace std;
8
9  #define N 101010
10 #define inf 1000000100
11
12 struct no{
13     no *left, *right;
14     int val;
15
16     no() : val(0), left(NULL), right(NULL) {}
17
18     int join(no *a, no *b){
19         int x = a ? a->val : 0;
20         int y = b ? b->val : 0;
21         return x+y;
22     }
23
24     no * update(int l, int r, int pos, int v){
25         no *at = new no();
26         *at = *this;
27
28         if(l == r){
```

```
29              at->val+=v;
30              return at;
31          }
32
33          int mid = (l+r)>>1;
34
35          if(pos<=mid){
36              if(!left) left = new no();
37              at->left = left->update(l, mid, pos, v);
38          }else{
39              if(!right) right = new no();
40              at->right = right->update(mid+1, r, pos, v);
41          }
42
43          at->val = join(at->left, at->right);
44          return at;
45      }
46  };
47
48  no *root[N];
49  int vet[N], aux[N], inv[N];
50
51  int query_k(no *no1, no *no2, int l, int r, int k){
52      if(l == r) return l;
53      int a = (no1 && no1->left) ? no1->left->val : 0;
54      int b = (no2 && no2->left) ? no2->left->val : 0;
55      int x = b-a;
56
57      int mid = (l+r)>>1;
58      if(x>=k) return query_k( no1 ? no1->left : NULL, no2 ? no2->left :
            NULL, l, mid, k);
59
60      return query_k( no1 ? no1->right : NULL, no2 ? no2->right : NULL, mid
            +1, r, k-x);
61  }
62
63
64  int main(){
65      int n, q;
66      scanf("%d %d", &n, &q);
67      root[0] = new no();
68
69      for(int i=1; i<=n; i++){
70          scanf("%d", &vet[i]);
71          aux[i] = vet[i];
72      }
73      int a, b, c;
74      sort(aux+1, aux+n+1);
75      for(int i=1; i<=n; i++){
76          a = lower_bound(aux+1, aux+n+1, vet[i]) - aux;
77          inv[a] = vet[i];
78          vet[i] = a;
79          root[i] = root[i-1]->update(1, n, vet[i], 1);
80      }
81
82      for(int i=0; i<q; i++){
83          scanf("%d %d %d", &a, &b, &c);
84          printf("%d\n", inv[query_k(root[a-1], root[b], 1, n, c)]);
85      }
```

```
86 }
```

2.11 Wavelet Tree

Listagem 2.11: Wavelet Tree

```
1 /*
2  *  E da final brasileira de 2016
3  */
4
5 #include <bits/stdc++.h>
6
7 using namespace std;
8
9 #define N 101010
10 #define inf 1e9
11
12 int n, vet[N], q;
13
14 struct wavelet{
15     int low, high;
16     vector<int> b;
17     wavelet *left, *right;
18
19     wavelet(int *from, int *to, int l, int h){//l e h sao o menor e o
           maior elemento do alfabeto
20         low = l, high = h;
21         if(from == to || l == h) return;
22
23         int mid = (l+h)>>1;
24
25         auto f = [mid](int i){ return i<=mid; };
26
27         b.push_back(0);
28         for(int *it = from; it!=to; it++){
29             b.push_back( b.back() + f(*it) );
30         }
31
32         int *pivo = stable_partition(from, to, f);
33         left = new wavelet(from, pivo, l, mid);
34         right = new wavelet(pivo, to, mid+1, h);
35     }
36
37     int kth(int l, int r, int k){
38         if(low == high) return low;
39         int lb = b[l-1];
40         int rb = b[r];
41         int c = rb-lb;
42         if(c>=k) return left->kth(lb+1, rb, k);
43         else return right->kth(l-lb, r-rb, k-c);
44     }
45
46     bool esq(int p){
47         return b[p] == b[p-1]+1;
48     }
```

```
49
50      void update(int p){//swap p e p+1
51          if(low == high) return;
52
53          if(esq(p) && !esq(p+1)){
54              swap(b[p], b[p+1]);
55              b[p]--;
56              return;
57          }
58
59          if(!esq(p) && esq(p+1)){
60              b[p]++;
61              return;
62          }
63          if(esq(p)) left->update(b[p]);
64          else right->update(p-b[p]);
65      }
66  };
67
68  int main(){
69      scanf("%d %d", &n, &q);
70      for(int i=1; i<=n; i++) scanf("%d", &vet[i]);
71      wavelet *root = new wavelet(vet+1, vet+n+1, 0, inf);
72      int a, b, c;
73      char op;
74      while(q--){
75          scanf(" %c", &op);
76          if(op == 'Q'){
77              scanf("%d %d %d", &a, &b, &c);
78              printf("%d\n", root->kth(a, b, c));
79          }else{
80              scanf("%d", &a);
81              root->update(a);
82          }
83      }
84  }
```

2.12 Wavelet Tree + Toggle

Listagem 2.12: Wavelet Tree + Toggle

```
1  /*
2   *  ILKQUERY 2 - toggle
3   */
4
5  #include <bits/stdc++.h>
6
7  using namespace std;
8
9  #define N 101010
10 #define inf 1000000001
11
12 int vet[N], n, q, state[N];
13
14 typedef long long int ll;
```

```
15
16 struct BIT{
17     vector<int> bit;
18     int sz;
19
20     BIT(){ bit.clear(); sz=0;}
21
22     BIT(int n){
23         sz=n;
24         bit.assign(n+1, 0);
25     }
26
27     void update(int pos, int v){
28         for(; pos<=sz; pos+= (pos&(-pos))) bit[pos]+=v;
29     }
30
31     int sum(int pos){
32         int ans=0;
33         for(; pos; pos-=(pos&(-pos))) ans+=bit[pos];
34         return ans;
35     }
36 };
37
38 struct wavelet{
39     int low, high;
40     vector<int> b;
41     BIT bit;//a bit guarda a quantidade de elementos inativos no intervalo
42     wavelet *left, *right;
43
44     wavelet(int *from, int *to, int l, int h){
45         low = l, high = h;
46         left = right = NULL;
47
48         bit = BIT(to-from+1);
49         if(from == to || l==h) return;
50
51
52         int mid = int( (ll(l) + ll(h) )>>1LL);
53
54         auto f = [mid](int i){ return i<=mid; };
55
56         b.push_back(0);
57         for(int *it = from;  it!=to;  it++){
58             b.push_back(b.back()+f(*it));
59         }
60
61         int *pivo = stable_partition(from, to, f);
62         left = new wavelet(from, pivo, l, mid);
63         right = new wavelet(pivo, to, mid+1, h);
64     }
65
66     int count_active(int l, int r){
67         int x= (r-l+1) - bit.sum(r) + bit.sum(l-1);//qtd de elementos
               ativos: |range| - qtd inativos no range
68         return x;
69     }
70
71     void toggle(int pos, int v){
72         bit.update(pos, v);
```

```
73          if(low == high) return;
74
75          int rb = b[pos];
76          int lb = b[pos-1];
77          int c = rb-lb;
78
79          if(c) left->toggle(lb+1, v);
80          else right->toggle(pos-rb, v);
81      }
82
83      int query(int l, int r, int k){//quantos elementos igual a k ativos
            existem no intervalo
84          if(l>r) return 0;
85          if(low == high) return (low == k) ? count_active(l, r) : 0;
86
87          int mid = int( (ll(low)+ ll(high))>>1LL );
88          int rb = b[r];
89          int lb = b[l-1];
90          if(k<=mid) return (left) ? left->query(lb+1, rb, k) : 0;
91          else return (right) ? right->query(l-lb, r-rb, k) : 0;
92      }
93  };
94
95  wavelet *WT;
96
97  int main(){
98      scanf("%d %d", &n, &q);
99      int menor=inf, maior=-inf;
100     for(int i=1; i<=n; i++){
101         scanf("%d", &vet[i]);
102         maior = max(maior, vet[i]);
103         menor = min(menor, vet[i]);
104         state[i] = 1;
105     }
106
107     WT = new wavelet(vet+1, vet+n+1, menor, maior);
108
109     int op, a, b, k;
110     while(q--){
111         scanf("%d", &op);
112         if(op){
113             scanf("%d", &a); a++;
114             if(state[a]) WT->toggle(a, 1);
115             else WT->toggle(a, -1);
116
117             state[a]^=1;
118         }else{
119             scanf("%d %d %d", &a, &b, &k); a++; b++;
120             printf("%d\n", WT->query(a, b, k));
121         }
122     }
123 }
```

2.13 Treap

Listagem 2.13: Treap

```
#include <cstdio>
#include <set>
#include <algorithm>
using namespace std;

//Treap para arvore binária de busca
struct node{
    int x, y, size;
    node *l, *r;
    node(int _x){
        x = _x;
        y = rand();
        size = 1;
        l = r = NULL;
    }
};

//10 vezes mais lento que Red-Black....
//Tome uma array de pontos (x,y) ordenados por x. u é ancestral de v se e somente se y(u) é maior que todos os elementos de u a v, v incluso!
//Split separa entre k-1 e k.
class Treap{
private:
    node* root;
    void refresh(node* t){
        if (t == NULL) return;
        t->size = 1;
        if (t->l != NULL)
            t->size += t->l->size;
        if (t->r != NULL)
            t->size += t->r->size;
    }
    void split(node* &t, int k, node* &a, node* &b){
        node * aux;
        if(t == NULL){
            a = b = NULL;
            return;
        }
        else if(t->x < k){
            split(t->r, k, aux, b);
            t->r = aux;
            refresh(t);
            a = t;
        }
        else{
            split(t->l, k, a, aux);
            t->l = aux;
            refresh(t);
            b = t;
        }
    }
    node* merge(node* &a, node* &b){
        node* aux;
        if(a == NULL) return b;
        else if(b == NULL) return a;
        if(a->y < b->y){
            aux = merge(a->r, b);
            a->r = aux;
```

```
58              refresh(a);
59              return a;
60          }
61          else{
62              aux = merge(a, b->l);
63              b->l = aux;
64              refresh(b);
65              return b;
66          }
67      }
68      node* count(node* t, int k){
69          if(t == NULL) return NULL;
70          else if(k < t->x) return count(t->l, k);
71          else if(k == t->x) return t;
72          else return count(t->r, k);
73      }
74      int size(node* t){
75          if (t == NULL) return 0;
76          else return t->size;
77      }
78      node* nth_element(node* t, int n){
79          if (t == NULL) return NULL;
80          if(n <= size(t->l)) return nth_element(t->l, n);
81          else if(n == size(t->l) + 1) return t;
82          else return nth_element(t->r, n-size(t->l)-1);
83      }
84      void del(node* &t){
85          if (t == NULL) return;
86          if (t->l != NULL) del(t->l);
87          if (t->r != NULL) del(t->r);
88          delete t;
89          t = NULL;
90      }
91  public:
92      Treap(){ root = NULL; }
93      ~Treap(){ clear(); }
94      void clear(){ del(root); }
95      int size(){ return size(root); }
96      bool count(int k){ return count(root, k) != NULL; }
97      bool insert(int k){
98          if(count(root, k) != NULL) return false;
99          node *a, *b, *c, *d;
100         split(root, k, a, b);
101         c = new node(k);
102         d = merge(a, c);
103         root = merge(d, b);
104         return true;
105     }
106     bool erase(int k){
107         node * f = count(root, k);
108         if(f == NULL) return false;
109         node *a, *b, *c, *d;
110         split(root, k, a, b);
111         split(b, k+1, c, d);
112         root = merge(a, d);
113         delete f;
114         return true;
115     }
116     int nth_element(int n){
```

```
117         node* ans = nth_element(root, n);
118         if (ans == NULL) return -1;
119         else return ans->x;
120     }
121
122 };
123
124 /*
125  * TEST MATRIX
126  */
127
128 int vet[10000009];
129
130 void test(){
131     set<int> s;
132     Treap t;
133     int N = 1000000;
134     for(int i=0; i<N; i++){
135         int n = rand()%1000;
136         if(!s.count(n)){
137             s.insert(n);
138             t.insert(n);
139             //if(!t.insert(n)) printf("error inserting %d in treap!\n", n)
                    ;
140             //printf("inserted %d\n", n);
141         }
142         else{
143             s.erase(n);
144             t.erase(n);
145             //if(!t.erase(n)) printf("error erasing %d in treap!\n", n);
146             //printf("erased %d\n", n);
147         }
148         n = rand()%1000;
149         if (s.count(n) != t.count(n)){
150             printf("failed test %d, s.count(%d) = %d, t.count(%d) = %d\n",
                        i, n, s.count(n), n, t.count(n));
151         }
152     }
153     s.clear();
154     t.clear();
155     for(int i=0; i<N; i++){
156         vet[i] = i+1;
157     }
158     random_shuffle(vet, vet+N);
159     for(int i=0; i<N; i++){
160         t.insert(vet[i]);
161     }
162     for(int i=1; i<=N; i++){
163         if (t.nth_element(i) != i){
164             printf("failed test %d\n", i);
165         }
166     }
167 }
168
169 int main(){
170     test();
171     return 0;
172 }
```

2.14 Implicit Treap

Listagem 2.14: Implicit Treap

```
#include <cstdio>
#include <vector>
#include <algorithm>
#include <ctime>
#define INF (1 << 30)
using namespace std;

const int neutral = 0; //comp(x, neutral) = x
int comp(int a, int b){
    return a + b;
}

//Treap para arvore binária de busca
struct node{
    int y, v, sum, size;
    bool swap;
    node *l, *r;
    node(int _v){
        v = sum = _v;
        y = rand();
        size = 1;
        l = r = NULL;
        swap = false;
    }
};

//10 vezes mais lento que Red-Black....
//Tome uma array de pontos (x,y) ordenados por x. u é ancestral de v se e somente se y(u) é maior que todos os elementos de u a v, v incluso!
//Split separa entre em uma árvore com k elementos e outra com size-k.
class ImplicitTreap{
private:
    node* root;
    void refresh(node* t){
        if (t == NULL) return;
        t->size = 1;
        t->sum = t->v;
        if (t->l != NULL){
            t->size += t->l->size;
            t->sum = comp(t->sum, t->l->sum);
            t->l->swap ^= t->swap;
        }
        if (t->r != NULL){
            t->size += t->r->size;
            t->sum = comp(t->sum, t->r->sum);
            t->r->swap ^= t->swap;
        }
        if (t->swap){
            swap(t->l, t->r);
            t->swap = false;
        }
    }
    void split(node* &t, int k, node* &a, node* &b){
        refresh(t);
```

```
54          node * aux;
55          if(t == NULL){
56              a = b = NULL;
57              return;
58          }
59          else if(size(t->l) < k){
60              split(t->r, k-size(t->l)-1, aux, b);
61              t->r = aux;
62              refresh(t);
63              a = t;
64          }
65          else{
66              split(t->l, k, a, aux);
67              t->l = aux;
68              refresh(t);
69              b = t;
70          }
71      }
72      node* merge(node* &a, node* &b){
73          refresh(a);
74          refresh(b);
75          node* aux;
76          if(a == NULL) return b;
77          else if(b == NULL) return a;
78          if(a->y < b->y){
79              aux = merge(a->r, b);
80              a->r = aux;
81              refresh(a);
82              return a;
83          }
84          else{
85              aux = merge(a, b->l);
86              b->l = aux;
87              refresh(b);
88              return b;
89          }
90      }
91      node* at(node* t, int n){
92          if (t == NULL) return NULL;
93          refresh(t);
94          if(n < size(t->l)) return at(t->l, n);
95          else if(n == size(t->l)) return t;
96          else return at(t->r, n-size(t->l)-1);
97      }
98      int size(node* t){
99          if (t == NULL) return 0;
100         else return t->size;
101     }
102     void del(node* &t){
103         if (t == NULL) return;
104         if (t->l != NULL) del(t->l);
105         if (t->r != NULL) del(t->r);
106         delete t;
107         t = NULL;
108     }
109 public:
110     ImplicitTreap(){ root = NULL; }
111     ~ImplicitTreap(){ clear(); }
112     void clear(){ del(root); }
```

```
113     int size(){ return size(root); }
114     bool insertAt(int n, int v){
115         node *a, *b, *c, *d;
116         split(root, n, a, b);
117         c = new node(v);
118         d = merge(a, c);
119         root = merge(d, b);
120         return true;
121     }
122     bool erase(int n){
123         node *a, *b, *c, *d;
124         split(root, n, a, b);
125         split(b, 1, c, d);
126         root = merge(a, d);
127         if (c == NULL) return false;
128         delete c;
129         return true;
130     }
131     int at(int n){
132         node* ans = at(root, n);
133         if (ans == NULL) return -1;
134         else return ans->v;
135     }
136     int query(int l, int r){
137         if (l>r) swap(l, r);
138         node *a, *b, *c, *d;
139         split(root, l, a, d);
140         split(d, r-l+1, b, c);
141         int ans = (b != NULL ? b->sum : neutral);
142         d = merge(b, c);
143         root = merge(a, d);
144         return ans;
145     }
146     void reverse(int l, int r){
147         if (l>r) swap(l, r);
148         node *a, *b, *c, *d;
149         split(root, l, a, d);
150         split(d, r-l+1, b, c);
151         if(b != NULL) b->swap ^= 1;
152         d = merge(b, c);
153         root = merge(a, d);
154     }
155 };
156
157 /*
158  * TEST MATRIX
159  */
160
161 bool test(){
162     srand(time(NULL));
163     vector<int> v;
164     ImplicitTreap t;
165     int N = 10000;
166     vector<int>::iterator it;
167     bool toprint = false;
168     for(int i=0, n, k, l, r; i<N; i++){
169         if (i%5 == 0 && i > 0){
170             n = rand()%((int)v.size());
171             if (toprint) printf("deleting v[%d] = %d\n", n, v[n]);
```

```
172             it = v.begin()+n;
173             v.erase(it);
174             t.erase(n);
175         }
176         else if (i%5 == 4){
177             l = rand()%((int)v.size());
178             r = rand()%((int)v.size());
179             if (l>r) swap(l, r);
180             if (toprint) printf("reversing %d to %d\n", l, r);
181             for(int j=l; j<=r && j<=r-j+l; j++){
182                 swap(v[j], v[r-j+l]);
183             }
184             t.reverse(l, r);
185         }
186         else{
187             n = rand()%((int)v.size()+1);
188             k = rand()%1000;
189             if (toprint) printf("inserting %d in pos %d\n", k, n);
190             it = v.begin()+n;
191             v.insert(it, k);
192             t.insertAt(n, k);
193         }
194         if (toprint) printf("array: ");
195         for(int j=0; j<(int)v.size(); j++){
196             if (toprint) printf("%d ", v[j]);
197             if (v[j] != t.at(j)){
198                 printf("test %d failed, v[%d] = %d, t.at(%d) = %d\n", i+1,
                        j, v[j], j, t.at(j));
199                 return false;
200             }
201         }
202         if (toprint) printf("\n");
203         l = rand()%((int)v.size());
204         r = rand()%((int)v.size());
205         if (l>r) swap(l, r);
206         int ans = neutral;
207         for(int j=l; j<=r; j++){
208             ans = comp(ans, v[j]);
209         }
210         if (toprint) printf("sum(%d, %d) = %d = %d\n", l, r, ans, t.query(
                l, r));
211         if (ans != t.query(l, r)){
212             printf("test %d failed, ans(%d, %d) = %d = %d\n", i, l, r, ans
                    , t.query(l, r));
213             return false;
214         }
215     }
216     return true;
217 }
218
219 int main(){
220     if(test()) printf("all tests passed\n");
221     return 0;
222 }
```

Capítulo 3

Max Flow

3.1 Dinic

Listagem 3.1: Dinic

```
#define N 50500//depende do problema
#define M 10100100//depende do problema
#define inf 10101010

typedef pair<int, int> ii;

struct ed{
    int to, c, f;
} edge[M];

int n, m, ptr[N], dist[N], curr, s, t;
vector<int> adj[N];
queue<int> q;


void add_edge(int a, int b, int c, int r){
    edge[curr].to = b;
    edge[curr].c = c;
    edge[curr].f = 0;
    adj[a].push_back(curr++);

    edge[curr].to = a;
    edge[curr].c = r;
    edge[curr].f = 0;
    adj[b].push_back(curr++);
}

void build_graph(){
    s = curr = 0;
    t = N-2;
    //modelagem do grafo
}

bool bfs(){
    q.push(s);
```

```
36      memset(dist, -1, sizeof dist);
37      dist[s] = 0;
38
39      while(q.size()){
40          int u =q.front(); q.pop();
41
42          for(int i=0; i<adj[u].size(); i++){
43              int e = adj[u][i];
44              int v = edge[e].to;
45              int w = edge[e].c - edge[e].f;
46
47              if(dist[v] != -1 || w<=0) continue;
48
49              q.push(v);
50              dist[v] = dist[u]+1;
51          }
52      }
53
54      return dist[t]!=-1;
55  }
56
57
58  int dfs(int u, int f){
59      if(u == t) return f;
60
61      for(; ptr[u]<adj[u].size(); ptr[u]++){
62
63          int e = adj[u][ptr[u]];
64          int v = edge[e].to;
65          int w = edge[e].c - edge[e].f;
66
67          if(dist[v]!=dist[u]+1) continue;
68
69          if(w>0){
70              if(int a = dfs(v, min(f, w))){
71                  edge[e].f+=a;
72                  edge[e^1].f-=a;
73                  return a;
74              }
75          }
76      }
77      return 0;
78  }
79
80
81  int dinic(){
82      int flow = 0;
83      while(1){
84          if(!bfs()) break;
85
86          memset(ptr, 0, sizeof ptr);
87
88          while(int a = dfs(s, inf)){
89              flow+=a;
90          }
91      }
92      return flow;
93  }
94
```

```
95 int main(){
96     //le grafo
97     build_graph();
98 
99     int mf = dinic();
100 }
```

3.2 Edmonds Karp

Listagem 3.2: Edmonds Karp

```
1  struct ed{
2      int to, c, f;
3  }edge[M];
4  
5  int n, m, seen[N], tempo, curr, p[N], nxt[N], dist[N], s, t;
6  vector<int> adj[N];
7  
8  void add_edge(int a, int b, int c, int rev){
9      edge[curr].to = b;
10     edge[curr].c = c;
11     edge[curr].f = 0;
12     adj[a].push_back(curr++);
13 
14     edge[curr].to = a;
15     edge[curr].c = rev;
16     edge[curr].f = 0;
17     adj[b].push_back(curr++);
18 }
19 
20 build_graph(){
21   //depende do problema
22 }
23 
24 int augment(){
25     int ans = inf;
26     for(int u=t, e = p[u];    u!=s;     u = edge[e^1].to, e = p[u]){
27         int w = edge[e].c - edge[e].f;
28         ans = min(ans, w);
29     }
30 
31     for(int u=t, e = p[u];    u!=s;     u = edge[e^1].to, e = p[u]){
32         edge[e].f+=ans;
33         edge[e^1].f-=ans;
34     }
35     return ans;
36 }
37 
38 int bfs(){
39     p[t] = -1;
40     queue<int> q;
41     q.push(s);
42 
43     while(q.size()){
44         int u = q.front(); q.pop();
```

```
45          if(u == t) break;
46          for(int i=0; i<adj[u].size(); i++){
47              int e = adj[u][i];
48              int v = edge[e].to;
49              if(seen[v] < tempo && edge[e].c - edge[e].f > 0){
50                  q.push(v);
51                  seen[v] = tempo;
52                  p[v] = e;
53              }
54          }
55      }
56      if(p[t] == -1) return 0;
57      return augment();
58  }
59
60  int edmonds_karp(){
61      int flow=0;
62      memset(seen, 0, sizeof seen);
63      tempo = 1;
64
65      while(int a = bfs()){
66          flow+=a;
67          tempo++;
68      }
69      return flow;
70  }
71
72  int main(){
73      cin >> n >> m;
74      build_graph();
75      cout << "Max flow = " << edmonds_karp() << endl;
76  }
```

3.3 Ford Fulkerson

Listagem 3.3: Ford Fulkerson

```
1  #define N 10040//depende do problema
2  #define M 1010101//depende do problema
3  #define inf 10101010//depende do problema
4
5  struct ed{
6      int to, c, f;
7  } edge[M];
8
9  int n, curr, seen[N], tempo, s, t;
10 vector<int> adj[N];
11
12 void add_edge(int a, int b, int c, int r){
13     edge[curr].to = b;
14     edge[curr].c = c;
15     edge[curr].f = 0;
16     adj[a].push_back(curr++);
17
18     edge[curr].to = a;
```

```
19      edge[curr].c = r;
20      edge[curr].f = 0;
21      adj[b].push_back(curr++);
22  }
23
24
25  void build_graph(){
26      s = curr = 0;
27      t = N-2;
28      //modelagem do grafo
29  }
30
31  int dfs(int u, int f){
32      if(u == t) return f;
33
34      seen[u] = tempo;
35
36      for(int i=0; i<adj[u].size(); i++){
37          int e = adj[u][i];
38          int v = edge[e].to;
39          int w = edge[e].c - edge[e].f;
40
41          if(seen[v]<tempo && w>0){
42              if(int a = dfs(v, min(f, w))){
43                  edge[e].f+=a;
44                  edge[e^1].f-=a;
45                  return a;
46              }
47          }
48      }
49      return 0;
50  }
51
52  int ford_fulk(){
53      memset(seen, 0, sizeof seen);
54      tempo = 1;
55      int flow = 0;
56
57      while(int a = dfs(s, inf)){
58          flow+=a;
59          tempo++;
60      }
61      return flow;
62  }
63
64  int main(){
65      //le grafo
66      //monta o grafo
67
68      build_graph();
69
70      int mf = ford_fulk();
71  }
```

3.4 Min Cost Max Flow

Listagem 3.4: Min Cost Max Flow

```
struct ed{
    ll to, c, f, cost;
} edge[M];

ll n, k, dist[N], p[N], seen[N], curr, s, t;
vector<int> adj[N];

// arestas indo com custo positivo, e voltando com custo negativovoid
    add_edge(ll a, ll b, ll c, ll cost){
    edge[curr] = {b, c, 0, cost};
    adj[a].push_back(curr++);

    edge[curr] = {a, 0, 0, -cost};
    adj[b].push_back(curr++);
}


void build_graph(){
    s = curr = 0;
    t = N-2;
    //modelagem do grafo
}

ll augment(){
    ll mf = inf;
    ll ans = 0;
    for(ll u = t, e = p[u]; u!=s; u = edge[e^1].to, e = p[u]){
        mf = min(mf, edge[e].c - edge[e].f);
    }
    for(ll u = t, e = p[u]; u!=s; u = edge[e^1].to, e = p[u]){
        ans += mf*edge[e].cost;
        edge[e].f+=mf;
        edge[e^1].f-=mf;
    }
    return ans;
}

ll SPF(){
    for(ll i=0; i<N; i++) dist[i] = inf;
    p[s] = p[t] = -1;

    dist[s] = 0;  seen[s] = 1;
    queue<int> q; q.push(s);

    while(q.size()){

        ll u = q.front(); q.pop();

        seen[u] = 0;
        for(ll i=0; i<adj[u].size(); i++){
            ll e = adj[u][i];
            ll v = edge[e].to;
            ll w = edge[e].c - edge[e].f;

```

```
54              if(w>0 && dist[v] > dist[u]+edge[e].cost){
55                  dist[v] = dist[u]+edge[e].cost;
56                  p[v] = e;
57                  if(!seen[v]){
58                      seen[v] = 1;
59                      q.push(v);
60                  }
61              }
62          }
63
64      }
65
66      if(p[t] == -1) return inf;
67      return augment();
68
69  }
70
71  ll MCMF(){
72      ll ans = 0;
73      while(1) {
74          ll a = SPF();
75          if(a == inf) break;
76          ans+=a;
77      }
78      return ans;
79  }
80
81  int main(){
82      //leitura do grafo
83
84      build_graph();
85
86      ll x = MCMF();
87  }
```

3.5 Resumão de Flow

3.5.1 Resumo dos algoritmos clássicos de flow

- Min-Path-Cover:

  Minimo numero de caminhos para visitar todos os vertices num DAG.

  Constroi o grafo bipartido Vout / Vin, add todas as arestas u-v: out(u) - in(v).

  add aresta s-out(u) pra todo u, e in(u)-t pra todo u.

  Todas as arestas com capacidade 1.

- Edge-disjoint/independent paths

  Encontre o maior numero de caminhos que nao compartilham nenhuma aresta(edge-disjoint) no caminho de s-t, num grafo qualquer.

  Encontre o maior numero de caminhos que nao compartilham nenhuma aresta e nenhum vertice(independent path) no caminho de s-t, num grafo qualquer.

Coloque o peso de cada aresta igual a 1, e pra independent paths coloque capacidade 1 em cada vertice tambem.

- Max Weighted Independent Set

  Grafo bipartido, cada vertice tem um peso, coloque peso[u] como capacidade da aresta s-u, e todas as outras arestas como infinito.

3.5.2 Complexidade dos algoritmos

Grafos genéricos:

- Ford fulkerson: $O(f * E)$
- Edmonds Karp: VE^2
- Dinic: V^2E

Grafos bipartidos:

- Ford fulkerson: geralmente $O(V^2)$, dependendo do problema
- Dinic: $O(sqrt(V) * E)$

Capítulo 4

Grafos

4.1 Bellman Ford

Listagem 4.1: Bellman Ford

```
vii Grafo[MAXN];
int dist[MAXN];
int parent[MAXN];
vi pathToDest;
int n;
bool hasNegativeCycle;

int BellmanFord(int source, int dest){
    int custo, v;
    hasNegativeCycle = false;
    for (int i = 0; i < n; i++){
        dist[i] = 1e8;
        parent[i] = -1;
    }
    dist[source]=0;
    parent[source]=source;

    for (int j = 0; j < n-1; j++)//roda n-1 vezes
    {
        for (int u = 0; u < n; u++)
        {
            for (int i = 0; i < Grafo[u].size(); i++)
            {
                v = Grafo[u][i].first;
                custo = Grafo[u][i].second;
                if(dist[v] > dist[u] + custo){
                    dist[v] = dist[u] + custo;
                    parent[v] = u;
                }
            }
        }
    }

    //se quiser saber quais vertices estao no ciclo é só adicionar outr
       for de 0 até 5, por exemplo, e ver qual distancia diminuiu. Se
```

```
        rodar só uma vez dependendo da configuração das aresta pode ser que
         não ache todos do ciclo, por isso é melhor rodar uma quantidade X
        de vezes, o ideal seria X = n
35
36      for (int u = 0; !hasNegativeCycle && u < n; u++)
37      {
38          for (int i = 0; !hasNegativeCycle && i < Grafo[u].size(); i++)
39          {
40              v = Grafo[u][i].first;
41              custo = Grafo[u][i].second;
42
43              if(dist[v] > dist[u] + custo)//se depois de n-1 iterações
                    ainda existe um caminho menor, existe um ciclo negativo
44                  hasNegativeCycle = true;
45          }
46      }
47
48      if(!hasNegativeCycle){
49          pathToDest.clear();
50          v = dest;
51          while(v!=source){
52              pathToDest.push_back(v);
53              v = parent[v];
54              //~ cout << v << endl;
55          }
56          pathToDest.push_back(source);
57      }
58      return dist[dest];
59  }
60
61  /*
62  limpa();
63  BellmanFord(origem, destino) retorna o menor caminho. Se tiver ciclo
        negativo a variável hasNegativeCycle vai ser true.
64  */
```

4.2 Centroid Decomposition

Listagem 4.2: Centroid Decomposition

```
1  /*
2   *  Cf 161D : quantos pares de vertices com distancia = k
3   */
4
5  //ATENCAO: Prestar atenção nos caminhos que começam no centroid, e na
       contribuição de cada centroid na resposta final
6
7  int n, k, dist[N], h[N], sz[N], block[N];
8  ll answer;
9  vector<int> adj[N];
10
11 void build_sz(int u, int p){
12      sz[u] = 1;
13      for(int v : adj[u]){
14          if(v == p || block[v]) continue;
```

```
15          build_sz(v, u);
16          sz[u]+=sz[v];
17      }
18  }
19
20  int find_centroid(int u, int p, int tam){
21      for(int v : adj[u]){
22          if(v == p || block[v]) continue;
23          if(sz[v]*2 > tam) return find_centroid(v, u, tam);
24      }
25      return u;
26  }
27
28  void dfs(int u, int p, int d){
29      dist[d]++;
30      for(int v : adj[u]){
31          if(v == p || block[v]) continue;
32          dfs(v, u, d+1);
33      }
34  }
35
36  void solve(int u, int p, int d){
37      if(d>=k) return;
38      answer+= (ll)dist[k-d];
39      for(int v : adj[u]){
40          if(v == p || block[v]) continue;
41          solve(v, u, d+1);
42      }
43  }
44
45  void decompose(int u){
46      build_sz(u, u);
47      u = find_centroid(u, u, sz[u]);
48      block[u] = 1;
49
50      for(int v : adj[u]){
51          if(block[v]) continue;
52          solve(v, u, 1);
53          dfs(v, u, 1);
54      }
55
56      answer+= (ll)dist[k];
57      for(int i=1; dist[i] > 0; i++) dist[i] = 0;
58
59      for(int v : adj[u]){
60          if(block[v]) continue;
61          decompose(v);
62      }
63  }
64
65
66  int main(){
67      int a, b;
68      scanf("%d %d", &n, &k);
69      for(int i=1; i<n; i++){
70          scanf("%d %d", &a, &b);
71          adj[a].push_back(b);
72          adj[b].push_back(a);
73      }
```

```
74
75      answer = 0;
76      decompose(1);
77      printf("%lld\n", answer);
78  }
```

4.3 Dijkstra

Listagem 4.3: Dijkstra

```
1  int n, m, dist[N], pai[N], s, t;
2  vector<ii> adj[N];
3
4  int dijkstra(){
5      memset(pai, -1, sizeof pai);
6      for(int i=0; i<n; i++) dist[i] = inf;
7
8      dist[s] = 0;
9      priority_queue< ii, vector<ii>, greater<ii> > pq;
10     pq.push(ii(0, s));
11
12     while(pq.size()){
13         ii foo = pq.top(); pq.pop();
14         int u = foo.S, d = foo.F;
15
16         if(dist[u] < d) continue;
17         for(ii f : adj[u]){
18             int v = f.F, w = f.S;
19             if(dist[v] > dist[u]+w){
20                 pai[v] = u;
21                 dist[v] = dist[u]+w;
22                 pq.push(ii(dist[v], v));
23             }
24         }
25     }
26     return (pai[t] == -1) ? -1 : dist[t];
27 }
```

4.4 Flood Fill

Listagem 4.4: Flood Fill

```
1  char vis[MAXN][MAXN];
2  char grid[MAXN][MAXN];
3  int n, m;
4  int dx[]={1,0,-1,0};
5  int dy[]={0,1,0,-1};
6
7  bool pode(int x, int y){
8      return x>=0 && x<n && y>=0 && y<m && !vis[x][y] && grid[x][y] == 'A';
```

```
}

void dfs(int x, int y){
    vis[x][y] = 1;
    grid[x][y] = 'T';

    for (int i = 0; i < 4; i++)
    {
        if(pode(x+dx[i], y+dy[i])){
            dfs(x+dx[i], y+dy[i]);
        }
    }
}
```

4.5 Floyd Warshall - All Pairs of Shortest Paths + Recuperação de caminho

Listagem 4.5: Floyd Warshall - All Pairs of Shortest Paths + Recuperação de caminho

```
int n;
int dist[MAXN][MAXN];
int pai[MAXN][MAXN];

void reset(){
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n; j++) {
            dist[i][j] = INF;
            if(i==j) dist[i][j]=0;

            pai[i][j] = i;
        }
    }
}

void printPath(int i, int j) {
    if (i != j) printPath(i, pai[i][j]);
    printf(" %d", j+1);
}

int main(){
    int m;
    cin >> n >> m;
    reset();

    int u, v, w;
    for (int i = 0; i < m; i++)
    {
        cin >> u >> v >> w;
        u--; v--;
        dist[u][v] = w;
        dist[v][u] = w;
    }

```

```
36      for(int k = 0; k < n; k++){
37          for(int i = 0; i < n; i++) {
38              for(int j = 0; j < n; j++) {
39                  if(dist[i][k] + dist[k][j] < dist[i][j]) {
40                      dist[i][j] = dist[i][k] + dist[k][j];
41                      pai[i][j] = pai[k][j];
42                  }
43              }
44          }
45      }
46
47      while (cin >> u >> v)
48      {
49          u--; v--;
50          cout << "dist = " << dist[u][v] << "\n";
51          cout << "path = "; printPath(u, v); cout << "\n";
52      }
53  }
```

4.6 Floyd Warshall - Fecho Transitivo

Listagem 4.6: Floyd Warshall - Fecho Transitivo

```
1  //inicializa com 1 onde tem aresta e 0 onde não tem
2
3  for (int k = 0; k < V; k++)
4      for (int i = 0; i < V; i++)
5          for (int j = 0; j < V; j++)
6              dist[i][j] |= (dist[i][k] & dist[k][j]);
```

4.7 Floyd Warshall - Minimax

Listagem 4.7: Floyd Warshall - Minimax

```
1  Minimax: arv. ger. min e maior aresta
2  Maximin: arv. ger. max e menor aresta
3  */
4  int N, E;
5  int main()
6  {
7      int i, u, v, w, q;
8      int g[200][200];
9      int caso=1;
10
11     while(scanf("%d %d %d", &N, &E, &q), N != 0) {
12         for (int i = 1; i <= N; i++)
13         {
14             for (int j = 1; j <= N; j++)
15             {
16                 g[i][j]=10000000;
```

```
17                  if(i==j) g[i][j]=0;
18              }
19          }
20
21          for(i = 0; i < E; i++) {
22              scanf("%d %d %d", &u, &v, &w);
23              g[u][v]=w;
24              g[v][u]=w;
25          }
26
27          for(int k = 1; k <= N; k++)
28             for(int i = 1; i <= N; i++)
29                for(int j = 1; j <= N; j++)
30                   g[i][j] = min(g[i][j], max(g[i][k], g[k][j]));//pega a
                        maior aresta do caminho (so existe um caminho, é uma
                        arvore)
31      }
32
33      return 0;
34  }
```

4.8 Kosaraju - Componentes Fortemente Conexas

Listagem 4.8: Kosaraju - Componentes Fortemente Conexas

```
1  int n, m;
2  vector<int> g[MAXN];
3  vector<int> t[MAXN];//grafo transposto
4  char vis[MAXN];
5  stack<int> p;
6
7  void dfs(int u, int op){
8      vis[u] = 1;
9
10     int v;
11     if(op == 1){
12         for (int i = 0; i < g[u].size(); i++)
13         {
14             v = g[u][i];
15             if(!vis[v]){
16                 dfs(v, op);
17             }
18         }
19         p.push(u);
20     }else{
21         for (int i = 0; i < t[u].size(); i++)
22         {
23             v = t[u][i];
24             if(!vis[v]){
25                 dfs(v, op);
26             }
27         }
28     }
29 }
30
```

```
31 int kosaraju(){//retorna quantas componentes fortemente conexas existe
32     memset(vis, 0, sizeof vis);
33 
34     while (!p.empty())
35         p.pop();
36 
37     for (int i = 0; i < n; i++)
38     {
39         if(!vis[i]) dfs(i, 1);
40     }
41 
42     int u;
43     int qtd = 0;
44     memset(vis, 0, sizeof vis);
45     while (!p.empty())
46     {
47         u = p.top();
48         p.pop();
49         if(!vis[u]){
50             qtd++;
51             dfs(u, 0);
52         }
53     }
54 
55     return qtd;
56 }
57 
58 void reset(){
59     for (int i = 0; i < n; i++)
60     {
61         g[i].clear();
62         t[i].clear();
63     }
64 }
65 
66 int main(){
67     reset();
68     //le o grafo normal e transposto
69     int ans = kosaraju();
70 
71     return 0;
72 }
```

4.9 LCA $O(logn)$ padrão

Listagem 4.9: LCA log n padrão

```
1 ll lca[N][LOGMAX], h[N];
2 ll minAresta[N][LOGMAX];
3 
4 void dfs(ll x, ll ult, ll peso_ult_x) {
5     lca[x][0] = ult;
6     minAresta[x][0] = peso_ult_x;
7 
8     for(ll i = 1; i < LOGMAX; ++i){
```

```
9           lca[x][i] = lca[lca[x][i - 1]][i - 1];
10          minAresta[x][i] = min(minAresta[x][i-1], minAresta[lca[x][i-1]][i
                -1]);
11      }
12      ll y;
13      for(ll i=0; i<g[x].size(); i++) {
14          y = g[x][i].first;
15          if(y == ult) continue;
16          h[y] = h[x] + 1;
17          dfs(y, x, g[x][i].second);
18      }
19  }
20
21  ll getLca(ll a, ll b) {
22      menorAresta = 10000000;
23      if(h[a] < h[b]) swap(a, b);
24      ll d = h[a] - h[b];
25      for(ll i = LOGMAX - 1; i >= 0; --i){
26          if((d >> i) & 1){
27              menorAresta = min(menorAresta, minAresta[a][i]);
28              a = lca[a][i];
29          }
30      }
31      if(a == b) return a;
32      for(ll i = LOGMAX - 1; i >= 0; --i){
33          if(lca[a][i] != lca[b][i]){
34              menorAresta = min(menorAresta, minAresta[a][i]);
35              menorAresta = min(menorAresta, minAresta[b][i]);
36              a = lca[a][i];
37              b = lca[b][i];
38          }
39      }
40      menorAresta = min(menorAresta, minAresta[a][0]);
41      menorAresta = min(menorAresta, minAresta[b][0]);
42      return lca[a][0];
43  }
```

4.10 LCA com RMQ Query O(1)

Listagem 4.10: LCA com RMQ Query O(1)

```
1  //SPOJ LCA
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6
7  #define N 101010
8  #define M 22
9
10 int n, vet[N<<1], in[N], h[N<<1], dist[N], table[N<<1][M], tempo;
11 vector<int> adj[N];
12
13 void dfs(int u, int d, int pai){
14
```

```
15      in[u] = tempo;
16      h[tempo] = dist[u] = d+1;
17      vet[tempo++] = u;
18
19      for(int v : adj[u]){
20          if(v == pai) continue;
21          dfs(v, d+1, u);
22          h[tempo] = d+1;
23          vet[tempo++] = u;
24      }
25  }
26
27  void build_table(){
28      int sz = tempo;
29      for(int i=0; i<sz; i++) table[i][0] = vet[i];
30      for(int j=1; j<M; j++){
31          for(int i=0; i+(1<<j)<=sz; i++){
32              int u = table[i][j-1];
33              int v = table[i+(1<<(j-1))][j-1];
34              table[i][j] = (dist[u] < dist[v]) ? u : v;
35          }
36      }
37  }
38
39  int query(int l, int r){
40      int k = 31 - __builtin_clz(r-l+1);
41      int u = table[l][k];
42      int v = table[r-(1<<k)+1][k];
43      return (dist[u] < dist[v]) ? u : v;
44  }
45
46  int get_lca(int u, int v){
47      if(in[u] > in[v]) swap(u, v);
48      return query(in[u], in[v]);
49  }
50
51  int main(){
52      //le a árvore
53      tempo = 0;
54      dfs(1, 0, -1);//supondo que a raiz da arvore seja o vertice 1
55          build_table();
56  }
```

4.11 MST - Árvore Geradora Mínima

Listagem 4.11: MST - Árvore Geradora Mínima

```
1  int n, comp[N], m;
2  vector<iii> edge;
3
4  void init(){
5      edge.clear();
6      for(int i=0; i<=n; i++) comp[i] = i;
7  }
8
```

```
int find(int i){
    return (comp[i] == i) ? i : comp[i] = find(comp[i]);
}

bool same(int i, int j) {
    return find(i) == find(j);
}

void join(int i, int j){
    comp[find(i)] = find(j);
}

int MST(){
    sort(edge.begin(), edge.end());
    int ans=0;
    for(int i=0; i<m; i++){
        int u = edge[i].S.F, v = edge[i].S.S, w = edge[i].F;
        if(!same(u, v)){
            join(u, v);
            ans+=w;
        }
    }
    return ans;
}

int main(){
    while(scanf("%d %d", &n, &m)){
        if(!n && !m) break;
        init();
        int a, b, c, tot=0;
        for(int i=0; i<m; i++){
            scanf("%d %d %d", &a,&b,&c);
            edge.push_back(iii(c, ii(a, b)));
            tot+=c;
        }

        printf("%d\n", tot-MST());
    }
}
```

4.12 Ordenação Topológica - DFS

Listagem 4.12: Ordenação Topológica - DFS

```
int n, m;
vector<int> g[MAXN];
char vis[MAXN];
vector<int> ts;

void dfs(int u){
    vis[u] = 1;

    int v;
    for (int i = 0; i < g[u].size(); i++)
    {
```

```
12          v = g[u][i];
13          if(!vis[v]){
14              dfs(v);
15          }
16      }
17      ts.pb(u);
18  }
19
20  int main(){
21      // le o grafo
22      // chama dfs
23      // ordenação topológica invertida vai estar em ts
24      return 0;
25  }
```

4.13 Ordenação Topológica - Kahn

Listagem 4.13: Ordenação Topológica - Kahn

```
1  int grauEntrada[MAXN], u, v;
2  vector<int> g[MAXN];
3  vector<int> topoSort;
4  /*
5      - Mantem na fila os vertices que nao tem aresta de entrada
6      - Remove todas as arestas que saem de u, e diminui o grau de entrada
           de cada vizinho v de u
7      - Se v passou a ter grau de entrada 0, adiciona ele na fila
8      - Repete o processo até a fila esvaziar
9  */
10
11 void Kahn(){
12     queue<int> q;
13     for (int i = 0; i < n; i++)
14     {
15         if(grauEntrada[i]==0) q.push(i);
16     }
17
18     while (!q.empty())
19     {
20         u = q.front(); q.pop();
21         topoSort.pb(u);
22
23         for (int i = 0; i < g[u].size(); i++)
24         {
25             v = g[u][i];
26             grauEntrada[v]--;
27             if(grauEntrada[v]==0){
28                 q.push(v);
29             }
30         }
31         g[u].clear();
32     }
33 }
34
35 void limpa(){
```

```
36      for (int i = 0; i < n; i++)
37      {
38          g[i].clear();
39          grauEntrada[i]=0;
40      }
41      nome.clear();
42      mapa.clear();
43      topoSort.clear();
44  }
45
46  int main()
47  {
48      limpa();
49      //monta grafo
50      Kahn();
51      //percorre topoSort e printa
52      return 0;
53  }
```

4.14 Tarjan - Pontos/Pontes de articulação

Listagem 4.14: Tarjan - Pontos/Pontes de articulação

```
1  #define N 101010
2  #define GRAY 1
3
4  int n, m, seen[N], in[N], low[N], tempo, root, bridges, AP;
5  vector<int> adj[N];
6
7  void tarjan(int u, int p){
8      seen[u] = GRAY;
9      in[u] = low[u] = tempo++;
10     int any, child=any=0;
11
12     for(int v : adj[u]){
13         if(v == p) continue;
14
15         if(!seen[v]){
16             child++;
17             tarjan(v, u);
18
19             low[u] = min(low[u], low[v]);
20
21             if(low[v] >= in[u]) any=1;
22             if(low[v] > in[u]) bridges++;
23
24         }else low[u] = min(low[u], in[v]);
25     }
26
27     if(child>1 && u == root) AP++;//caso especial: raiz é um vertice de
           articulacao
28     else if(any && u!=root) AP++;
29 }
30
31 int main(){
```

```
    int a, b;
    scanf("%d %d", &n, &m);
    for(int i=0;i<m; i++){
        scanf("%d %d", &a, &b);
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    root = 1;
    bridges = tempo = AP = 0;
    tarjan(1, 0);

    printf("Articulation points: %d\n", AP);
    printf("Bridges: %d\n", bridges);
}
```

4.15 Tarjan - Componentes Fortemente Conexas

Listagem 4.15: Tarjan - Componentes Fortemente Conexas

```
#define N 101010
#define GRAY 1
#define BLACK 2

int n, m, seen[N], low[N], in[N], comp[N], tempo, comp_cont;
vector<int> adj[N];
stack<int> pilha;

void tarjan_scc(int u){
    seen[u] = GRAY;
    in[u] = low[u] = tempo++;
    pilha.push(u);

    for(int v : adj[u]){
        if(seen[v] == BLACK) continue;

        if(!seen[v]){
            tarjan_scc(v);

            low[u] = min(low[v], low[u]);
        }else low[u] = min(low[u], in[v]);
    }

    if(low[u] == in[u]){
        comp_cont++;
        while(pilha.size()){
            int v = pilha.top(); pilha.pop();
            seen[v] = BLACK;
            comp[v] = comp_cont;
            if(u == v) break;
        }
    }
}

int main(){
```

```
36      int a, b, op;
37      scanf("%d %d", &n, &m);
38      for(int i=0; i<m; i++){//recebe um grafo direcionado
39          scanf("%d %d", &a, &b);
40          adj[a].push_back(b);
41      }
42
43      memset(seen, 0, sizeof seen);
44      comp_cont=tempo=0;
45
46      for(int i=1; i<=n; i++){
47          if(!seen[i]) tarjan_scc(i);
48      }
49      printf("%d\n", comp_cont == 1);//printa 1 se for fortemente conexo
50  }
```

4.16 Tarjan - Grafo das Componentes Biconectadas

Listagem 4.16: Tarjan - Grafo das Componentes Biconectadas

```
1  // O Grafo gerado eh uma árvore (ou uma floresta se for desconexo)
2
3  #define N 101010
4  #define GRAY 1
5
6  int n, seen[N], in[N], low[N], id[N], tempo, bridges, diametro;
7  vector<int> adj[N], tr[N];//tr: arvore das componentes biconectadas
8  stack<int> pilha;
9
10 //op == 0: calcula pra cada vertice qual componente que ele faz parte (id)
11 void tarjan(int u, int p, int op){
12      seen[u] = GRAY;
13      in[u] = low[u] = tempo++;
14
15      if(!op) pilha.push(u);
16
17      for(int v : adj[u]){
18          if(v == p) continue;
19
20          if(!seen[v]){
21              tarjan(v, u, op);
22
23              if(!op && low[v] > in[u]){
24                  while(pilha.size()){
25                      int x = pilha.top(); pilha.pop();
26                      id[x] = v;
27                      if(v == x) break;
28                  }
29              }
30              if(op && low[v]>in[u]){
31                  tr[id[u]].push_back(id[v]);
32                  tr[id[v]].push_back(id[u]);
33              }
34
35              low[u] = min(low[u], low[v]);
```

```
36          }else low[u] = min(low[u], in[v]);
37      }
38  }
39
40  void build_tarjan(int op){
41      tempo = bridges = 0;
42      memset(seen, 0, sizeof seen);
43
44      tarjan(1, 0, op);
45
46      if(op) return;
47      while(pilha.size()){
48          int x = pilha.top(); pilha.pop();
49          id[x] = 1;
50      }
51  }
52
53  int main(){
54      int a, b, tc, m;
55      scanf("%d %d", &n, &m);
56      for(int i=0; i<m; i++){
57          scanf("%d %d", &a, &b);
58          adj[a].push_back(b);
59          adj[b].push_back(a);
60      }
61
62      build_tarjan(0);
63      build_tarjan(1);
64
65      //processa a arvore
66  }
```

4.17 Tarjan - Grafo das Componentes Fortemente Conexas

Listagem 4.17: Tarjan - Grafo das Componentes Fortemente Conexas

```
1  //responde qual vertice alcanca a maior quantidade de vertices num grafo
       com N<=100000
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6
7  #define N 101010
8  #define GRAY 1
9  #define BLACK 2
10
11 int n, m, seen[N], low[N], dp[N], in[N], comp[N], sz[N], tempo, comp_cont;
12 vector<int> adj[N], g[N];//adj eh o grafo normal, g eh o grafo das
       componentes
13 stack<int> pilha;
14
15 //op == 0: calcula as scc de cada vertice, op == 1: monta o grafo das scc
```

```
16 void tarjan_scc(int u, int op){
17
18     seen[u] = GRAY;
19     in[u] = low[u] = tempo++;
20     pilha.push(u);
21
22     for(int v : adj[u]){
23         if(seen[v] == BLACK) {
24             if(op == 1)   g[comp[u]].push_back(comp[v]);
25             continue;
26         }
27
28         if(!seen[v]){
29             tarjan_scc(v, op);
30
31             if(op == 1 && seen[v] == BLACK){
32                 g[comp[u]].push_back(comp[v]);
33             }
34
35             low[u] = min(low[v], low[u]);
36         }else low[u] = min(low[u], in[v]);
37     }
38
39     if(low[u] == in[u]){
40         comp_cont++;
41
42         while(pilha.size()){
43             int v = pilha.top(); pilha.pop();
44             seen[v] = BLACK;
45
46             if(!op) comp[v] = comp_cont;
47             if(!op) sz[comp_cont]++;
48
49
50             if(u == v) break;
51         }
52     }
53 }
54
55 void build_tarjan(int op){
56     memset(seen, 0, sizeof seen);
57     comp_cont=tempo=0;
58
59     for(int i=1; i<=n; i++){
60         if(!seen[i]) tarjan_scc(i, op);
61     }
62
63     if(!op) return;
64
65     for(int i=1; i<=comp_cont; i++){//tira as arestas repetidas do grafo
           das scc
66         if(!g[i].size()) continue;
67
68         sort(g[i].begin(), g[i].end());
69
70         g[i].resize( distance( g[i].begin(),  unique(g[i].begin(), g[i].
               end())  ) );//tira repetições
71     }
72 }
```

```
73
74 void solve(int u){
75     if(dp[u]!=0) return;
76     dp[u] = sz[u];
77     for(int v : g[u]){
78         solve(v);
79         dp[u]+=dp[v];
80     }
81 }
82
83 int main(){
84     int a, b, op;
85     scanf("%d %d", &n, &m);
86     for(int i=0; i<m; i++){//recebe um grafo direcionado
87         scanf("%d %d %d", &a, &b, &op);
88         adj[a].push_back(b);
89         if(op == 2) adj[b].push_back(a);
90     }
91
92     build_tarjan(0);
93     build_tarjan(1);
94     memset(dp, 0, sizeof dp);
95
96     for(int i=1; i<=comp_cont; i++) solve(i);
97
98     int ans=1;
99     for(int i=1; i<=n; i++){
100        if(dp[comp[i]] > dp[comp[ans]]) ans=i;
101    }
102
103    printf("%d\n", ans);
104 }
```

4.18 Shortest Path Faster - Menor caminho chinês

Listagem 4.18: Shortest Path Faster - Menor caminho chinês

```
1 int n, m, dist[N], pai[N], in[N], s, t;
2 vector<ii> adj[N];
3
4 int SPF(){
5     memset(pai, -1, sizeof pai);
6     memset(in, 0, sizeof in);
7     for(int i=0; i<n; i++) dist[i] = inf;
8     dist[s] = 0;
9     queue<int> q;
10    q.push(s);
11
12    while(q.size()){
13        int u = q.front(); q.pop();
14
15        in[u] = 0;
16        for(ii f : adj[u]){
17            int v = f.F, w = f.S;
18            if(dist[v] > dist[u]+w){
```

```
                pai[v] = u;
                dist[v] = dist[u]+w;
                if(!in[v]){
                    q.push(v);
                    in[v] = 1;
                }
            }
        }
    }
    return (pai[t] == -1) ? -1 : dist[t];
}

int main(){
    int tc, a, b, c, x=1;
    scanf("%d", &tc);
    while(tc--){
        scanf("%d %d %d %d", &n, &m, &s, &t);
        for(int i=0; i<=n; i++) adj[i].clear();
        for(int i=0; i<m; i++){
            scanf("%d %d %d", &a, &b, &c);
            adj[a].push_back(ii(b, c));
            adj[b].push_back(ii(a, c));
        }
        a = SPF();

        if(a>=0) printf("Case #%d: %d\n", x++, a);
        else printf("Case #%d: unreachable\n", x++);
    }
}
```

4.19 Union Find

Listagem 4.19: Union Find

```
int n, sz[N], comp[N], cont_comp, maior;

void init(){
    cont_comp = n;
    maior = 1;
    for(int i=0; i<=n; i++) {
        comp[i] = i;
        sz[i] = 1;
    }
}

int find(int i){
    return (comp[i] == i) ? i : comp[i] = find(comp[i]);
}

bool same(int i, int j){
    return find(i) == find(j);
}

void join(int i, int j){
    int x = find(i), y = find(j);
```

```
22      if(x == y) return;
23
24      comp[y] = x;
25      sz[x]+=sz[y];
26      sz[y] = 0;
27      cont_comp--;
28      maior = max(maior, sz[x]);
29  }
30
31  int main(){
32      int q, a, b;
33      char op;
34      scanf("%d %d", &n, &q);
35
36      init();
37
38      while(q--){
39          scanf(" %c", &op);
40          if(op == 'T'){
41              printf("%d %d\n", cont_comp, maior);
42              continue;
43          }
44          scanf("%d %d", &a, &b);
45
46          if(op=='F') {
47              join(a, b);
48          }else{
49              printf(find(a) == find(b) ? "sim\n" : "nao\n");
50          }
51      }
52  }
```

4.20 Todos os menores caminhos com Dijkstra

Listagem 4.20: Todos os menores caminhos com Dijkstra

```
1  int n, m;
2  int g[600][600];
3  int origem, destino;
4  set<int> parent[600];
5  char vis[600];
6  int dist[600];
7
8  void solve(int atual, int nxt){
9      if(atual == origem){
10         cout << origem << "\n";
11         return;
12     }
13
14     cout << atual << " ";
15     for (auto i : parent[atual])
16     {
17         int v = i;
18         solve(v, atual);
19     }
```

```
}

int dij(){
    priority_queue<pair<int, int> >pq;
    pq.push(mp(0, origem));
    parent[origem].insert(origem);
    dist[origem] = 0;

    int u, w, v;
    while (!pq.empty())
    {
        u = pq.top().S;
        pq.pop();
        if(vis[u]) continue;
        vis[u] = 1;

        if(u==destino) return dist[destino];
        for (int i = 0; i < n; i++)
        {
            if(g[u][i]){
                v = i;
                w = g[u][i];
                if(dist[u] + w <= dist[v]){
                    if(dist[u] + w < dist[v]) parent[v].clear();//se achou
                        caminho menor: limpa vetor de parent
                    parent[v].insert(u);
                    dist[v] = dist[u] + w;
                    pq.push(mp(-dist[v], v));
                }
            }
        }

    }
    return -1;
}

int main()
{
    reset();
    cout << dij() << endl;
    solve(destino, destino);//printa os caminhos invertidos: destino ...
        origem
    return 0;
}
```

4.21 2-SAT

Listagem 4.21: 2-SAT

```
vector<int> Grafo[MAXN], Transposto[MAXN];
int n, m, cnt;
int vis[MAXN];
int componente[MAXN];
stack<int> pilha;
map<string, int> mapa;
```

```
bool sat;
int ans[MAXN];

void limpa(){
    for (int i = 0; i <= MAXN; i++)
    {
        Grafo[i].clear();
        Transposto[i].clear();
    }
}
//da pra acessar a negacao de um elemento fazendo o xor. Deve ser indexado como:
//true: x*2
//false: x*2 + 1
//CODIGO SENDO INDEXADO A PARTIR DE 0***************

void addEdge(int u, int v){
    Grafo[u^1].pb(v);// !u -> v
    Grafo[v^1].pb(u);// !v -> u

    Transposto[v].pb(u^1);//Grafo transposto pra rodar o kosaraju
    Transposto[u].pb(v^1);
}

void dfs1(int u){
    if (!vis[u])
    {
        vis[u]=1;
        for (int i = 0; i < Grafo[u].size(); i++)
        {
            int v = Grafo[u][i];
            if(!vis[v]) dfs1(v);
        }
        pilha.push(u);
    }
}

void dfs2(int u){
    if (!vis[u])
    {
        vis[u]=1;
        componente[u] = cnt;
        for (int i = 0; i < Transposto[u].size(); i++)
        {
            int v = Transposto[u][i];
            if(!vis[v]) dfs2(v);
        }
    }
}

void Kosaraju(){
    memset(vis, 0, sizeof vis);
    while(!pilha.empty()) pilha.pop();
    for (int i = 0; i < 2*n; i++)
        if(!vis[i]) dfs1(i);//visita todos os vertices

    memset(vis, 0, sizeof vis);
    memset(componente, 0, sizeof componente);
    cnt=0;
```

```
65      int u;
66
67      while(!pilha.empty()){
68          u = pilha.top(); pilha.pop();
69          if(!vis[u]){
70              dfs2(u);
71              cnt++;
72          }
73          ans[u/2] = 1-u%2;//atribui valores aos elementos. Se for
                satisfativel da pra usar esse vetor
74      }
75
76      sat=true;
77      for (int i = 0; i < n; i++)
78      {
79          if(componente[2*i] == componente[2*i + 1]) sat = false;//se estão
                na mesma componente a formula nao tem solucao
80      }
81  }
```

Capítulo 5

Strings

5.1 Aho-Corasick

Listagem 5.1: Aho-Corasick

```
1  string s, txt;
2  int cont; //contador global de nós
3
4  struct no{
5      #define ALF 130 //depende do problema
6
7      no *pai, *suffix_link, *nxt[ALF];
8      char c;
9      int fim, num;
10
11     no(char letra, int id){
12         c = letra;
13         for(int i=0; i<ALF; i++) nxt[i] = NULL;
14         pai = suffix_link = NULL;
15         fim = 0;
16         num = id;
17     }
18
19
20     void insert(int i){
21         if(i == s.size()){
22             fim++;
23             return;
24         }
25
26         int letra = s[i]-'A';
27         if(!nxt[letra]){
28             nxt[letra] = new no(s[i], cont++);
29             nxt[letra]->pai = this;
30         }
31         nxt[letra]->insert(i+1);
32     }
33
34     void build_sf(){
35
```

```
36          queue<no*> q;
37          for(int i=0; i<ALF; i++)
38              if(nxt[i]) q.push(nxt[i]);
39
40          while(q.size()){
41              no *u = q.front(); q.pop();
42
43              no *tmp = u->pai->suffix_link;
44
45              char letra = u->c - 'A';
46
47              while(tmp && !tmp->nxt[letra])   tmp = tmp->suffix_link;
48
49              u->suffix_link = (tmp) ? tmp->nxt[letra] : this;
50              u->fim += u->suffix_link->fim;
51
52              for(int i=0; i<ALF; i++)
53                  if(u->nxt[i]) q.push(u->nxt[i]);
54          }
55      }
56
57      void destroy(){
58          for(int i=0; i<ALF; i++){
59              if(nxt[i]){
60                  nxt[i]->destroy();
61                  delete nxt[i];
62              }
63          }
64      }
65
66  };
67
68  no *root;
69
70  no *climb(no *u, char letra){
71      no *tmp = u;
72      while(tmp && !tmp->nxt[letra]) tmp = tmp->suffix_link;
73      return tmp ? tmp->nxt[letra] : root;
74  }
75
76  int query(int pos, no *u){//exemplo de query, mas varia de problema pra
        problema
77      if(pos==txt.size()) return u->fim;
78      return u->fim + query(pos+1, climb(u, txt[pos]-'A'));
79  }
```

5.2 Hash

Listagem 5.2: Hash

```
1  #define MAXN 100100
2  const ll A = 1009;
3  const ll MOD = 9LL + 1e18;
4  ll pot[MAXN];
5
```

```
6  ll normalize(ll r){
7      while(r<0) r+=MOD;
8      while(r>=MOD) r-=MOD;
9      return r;
10 }
11
12 ll mul(ll a, ll b){//(a*b)%MOD
13     ll q = ll((long double)a*b/MOD);
14     ll r = a*b - MOD*q;
15     return normalize(r);
16 }
17
18 ll add(ll hash, ll c){
19     return (mul(hash, A) + c)%MOD;
20 }
21
22 void buildPot(){
23     for (int i = 0; i < MAXN; i++)
24     {
25         pot[i] = i ? mul(pot[i-1], A) : 1LL;
26     }
27 }
28
29 struct Hash{
30     string s;
31     ll hashNormal, hashInvertida;
32     ll accNormal[MAXN], accInvertida[MAXN];
33
34     Hash(){}
35     Hash(string _s){
36         s = _s;
37     }
38
39     void build(){
40         accNormal[0] = 0LL;
41         for (int i = 1; i <= (int)s.size(); i++){
42             accNormal[i] = add(accNormal[i-1], s[i-1]-'a'+1);
43         }
44         hashNormal = accNormal[(int)s.size()];
45
46         accInvertida[s.size()] = 0LL;
47         for (int i = s.size()-1; i >= 0; i--){
48             accInvertida[i] = add(accInvertida[i+1], s[i]-'a'+1);
49         }
50         hashInvertida = accInvertida[0];
51
52     }
53
54     ll getRangeNormal(int l, int r){//pega a hash da substring (l, r) na
           string normal (abcd - [0, 2] = abc)
55         if(l>r) return 0LL;
56         ll ans = (accNormal[r+1] - mul(accNormal[l], pot[r-l+1]))%MOD;
57         return normalize(ans);
58     }
59
60     ll getRangeInvertido(int l, int r){//pega a hash da substring (l, r)
           na string invertida (abcd - [0, 2] = cba)
61         if(l>r) return 0LL;
```

```
62          ll ans = (accInvertida[l] - mul(accInvertida[r+1], pot[r-l+1]))%
                MOD;
63          return normalize(ans);
64      }
65  };
66
67  int main () {
68      buildPot();//cuidado com o limite do MAXN
69      //resolve o problema
70      Hash H = Hash(str);
71      H.build();
72
73      return 0;
74  }
```

5.3 Hash - Maior Substring Palíndromo $O(nlogn)$

Listagem 5.3: Hash - Maior Substring Palíndromo (nlogn)

```
1  Hash H;
2
3  bool ok(int tam){
4      int l = 0;
5      int r = tam-1;
6
7      while (r < (int)H.s.size())
8      {
9          if(H.getRangeNormal(l, r) == H.getRangeInvertido(l, r)){
10             return true;
11         }
12
13         l++; r++;
14     }
15
16     return false;
17 }
18
19 int longestPalindromicSubstring(string s, string &res){
20 //retorna o tamanho da maior substring palindromo
21     H = Hash(s);
22     H.build();
23
24     int lo = 1;
25     int hi = (int)s.size();
26     int mid;
27     int ans = 0;
28
29     while (lo <= hi)
30     {
31         mid = (lo+hi)/2;
32         if(ok(mid) || ok(mid+1)){
33             lo = mid+1;
34             ans = max(ans, mid);
35         }else{
36             hi = mid-1;
```

```
37          }
38      }
39
40      //recupera a primeira string palindromo de tamanho ans
41      res.clear();
42      int l = 0, r = ans-1;
43      while (r < (int)H.s.size())
44      {
45          if(H.getRangeNormal(l, r) == H.getRangeInvertido(l, r)){
46              res = H.s.substr(l, ans);
47              break;
48          }
49      }
50
51      return ans;
52  }
53
54  int main(){
55      //le a string
56      // chama a função
57  }
```

5.4 KMP

Listagem 5.4: KMP

```
1  string s, txt;
2  int n, m, p[N];
3
4  void kmp(){
5      p[0] = 0;
6      for(int i=1; i<n; i++){
7          p[i] = p[i-1];
8          while(txt[p[i]] != txt[i] && p[i]) p[i] = p[p[i]-1];
9
10         if(txt[p[i]] == txt[i]) p[i]++;
11
12     }
13     for(int i=0; i<n; i++) printf("p[%d] = %d\n", i, p[i]);
14 }
15
16
17 int main(){
18     getline(cin, s);
19     txt = s+"$";//importante
20     getline(cin, s);
21     txt+=s;
22
23     n = txt.size();
24     cout << txt << endl;
25     kmp();
26 }
```

5.5 Rabin Karp

Listagem 5.5: Rabin Karp

```
int rabin_karp(string &text, string &pattern){
//retorna a posição da primeira ocorrência do padrão no texto, ou -1, se n
    ão existir

    Hash T = Hash(text);
    Hash P = Hash(pattern);
    T.build();
    P.build();

    int l = 0, r = pattern.size()-1;
    while (r < (int)text.size())
    {
        if(T.getRangeNormal(l, r) == P.hashNormal){
            return l;
        }
        l++; r++;
    }

    return -1;
}
```

5.6 Suffix Array nlogn + LCP Array

Listagem 5.6: Suffix Array nlogn + LCP Array

```
int n, sa[N], tmpsa[N], rk[N], tmprk[N], cont[N], lcp[N], inv[N];
string s;

void radix(int  k){
    memset(cont, 0, sizeof cont);
    int maxi = max(300, n);

    for(int  i=0; i<n; i++){
        cont[ (i+k)<n ?  rk[i+k] : 0 ]++;
    }

    for(int  i=1; i<maxi; i++) cont[i]+=cont[i-1];

    for(int  i=n-1; i>=0; i--){
        tmpsa[ --cont[ ( sa[i]+k ) < n   ?  rk[sa[i]+k] : 0 ] ] = sa[i];
    }
    for(int  i=0; i<n; i++) sa[i] = tmpsa[i];
}

void build_SA(){

    for(int  i=0; i<n; i++){
        rk[i] = s[i];
        sa[i] = i;
```

```
25      }
26
27      for(int  k=1; k<n; k<<=1){
28
29          radix(k);
30          radix(0);
31
32          tmprk[sa[0]] = 0;
33          int  r = 0;
34          for(int  i=1; i<n; i++){
35              tmprk[sa[i]] = (rk[sa[i]] == rk[sa[i-1]] && rk[sa[i]+k] == rk[
                    sa[i-1]+k]) ? r : ++r;
36          }
37
38          for(int  i=0; i<n; i++){
39              rk[sa[i]] = tmprk[sa[i]];
40          }
41
42          if(rk[sa[n-1]] == n-1) break;
43      }
44  }
45
46  void build_lcp(){
47      for(int  i=0; i<n; i++){
48          inv[sa[i]] = i;
49      }
50      int  L=0;
51      for(int  i=0; i<n; i++){
52          if(inv[i] == 0){
53              lcp[inv[i]] = 0;
54              continue;
55          }
56          int  prev = sa[inv[i]-1];
57          while(i+L<n && prev+L<n && s[i+L] == s[prev+L]) L++;
58
59          lcp[inv[i]] = L;
60          L = max(L-1, 0);
61      }
62  }
63
64  int  solve(){
65      //depende do problema
66  }
67
68  int main(){
69      ios_base::sync_with_stdio(0); cin.tie(0);
70      getline(cin, s);
71      s.push_back('$');
72
73      n = s.size();
74
75      build_SA();
76      build_lcp();
77
78      solve();
79  }
```

5.7 Suffix Array O($nlog^2n$) + LCP Array

Listagem 5.7: Suffix Array nlog$^2n + LCPArray$

```
1  //OBS: usa a struct Hash
2
3  int sa[MAXN], lcp[MAXN];
4
5  string s;
6  Hash H;
7
8  int getLCP(int a, int b){//pega o LCP entre o sufixo começando em a e o
       sufix começando em b
9
10     int lo = 0;
11     int hi = min((int)s.size() - a, (int)s.size() - b);
12     int mid;
13     int ans = 0;
14
15     while(lo <= hi){
16         mid = (lo+hi)/2;
17         if(H.getRangeNormal(a, a+mid-1) == H.getRangeNormal(b, b+mid-1)){
18             lo = mid+1;
19             ans = max(ans, mid);
20         }else{
21             hi = mid-1;
22         }
23     }
24
25     return ans;
26 }
27
28 bool compareSA(int a, int b){//pega o LCP e compara o próximo caractere
29     int len = getLCP(a, b);
30     if(a+len == (int)s.size()) return true;
31     if(b+len == (int)s.size()) return false;
32
33     return s[a+len] < s[b+len];
34 }
35
36 void build_SA(){
37     int tam = (int)s.size();
38     for (int i = 0; i < tam; i++)
39     {
40         sa[i] = i;
41     }
42     sort(sa, sa + tam, compareSA);
43 }
44
45 void build_lcp(){
46     lcp[0] = 0;
47     for (int i = 1; i < (int)s.size(); i++)
48     {
49         lcp[i] = getLCP(sa[i], sa[i-1]);
50     }
51 }
52
53 int main () {
```

```
54      buildPot();//cuidado com o limite do MAXN
55      cin >> s;
56      H = Hash(s);
57      H.build();
58
59      build_SA();
60      build_lcp();
61
62      return 0;
63  }
```

5.8 Trie Estática

Listagem 5.8: Trie Estática

```
1  int trie[MAXN][26];
2  char fim[MAXN];
3  int counter[MAXN];
4  string s;
5  int cnt = 2;
6
7  void add(){
8      int no = 1;//1 é a raiz
9      int c;
10
11     for (int i = 0; i < (int)s.size(); i++)
12     {
13         c = s[i]-'a';
14         if(!trie[no][c]){
15             trie[no][c] = cnt++;
16         }
17         no = trie[no][c];
18         counter[no]++;
19     }
20     fim[no] = 1;
21 }
22
23 int main(){
24     cin >> n;
25     for (int i = 0; i < n; i++)
26     {
27         cin >> s;
28         add();
29     }
30     return 0;
31 }
```

5.9 Trie Dinâmica

Listagem 5.9: Trie Dinâmica

```
string s;

struct no{
    #define ALF 30 //depende do problema

    no *nxt[ALF];
    int cont, fim;
    char c;

    no(char k){
        c = k;
        for(int i=0; i<ALF; i++) nxt[i] = NULL;
        cont = fim = 0;
    }

    void insert(int i){
        cont++;
        if(i == s.size()){
            fim=1;
            return;
        }
        if(!nxt[s[i]-'a'])  nxt[s[i]-'a'] = new no(s[i]);

        return nxt[s[i]-'a']->insert(i+1);
    }

    void destroy(){
        for(int i=0; i<ALF; i++){
            if(nxt[i]) {
                nxt[i]->destroy();
                delete nxt[i];
            }
        }
    }

};

no *root;

int main(){
    ios_base::sync_with_stdio(0); cin.tie(0);

    root = new no('$');
    int n;
    cin >> n;
    for(int i=0; i<n; i++){
        cin >> s;
        root->insert(0);
    }

    // resolve problema

    root->destroy();
    delete root;
}
```

5.10 Z-Algorithm

Listagem 5.10: Z-Algorithm

```
string s;
int z[N];

void calc_z(){

    memset(z, 0, sizeof z);

    int n = s.size();
    int l=0, r=0;

    for(int i=1; i<n; i++){
        if(i<=r)  z[i] = min(r-i+1, z[i-l]);

        while(i+z[i] < n && s[z[i]] == s[i+z[i]])
            z[i]++;

        if(i+z[i]-1 > r){
            l=i;
            r = i+z[i]-1;
        }
    }
}
```

Capítulo 6

SQRT

6.1 MO

Listagem 6.1: MO

```
struct query{
    int l, r, pos;
    query(){}
    query(int a, int b, int d){
        l = a;
        r = b;
        pos = d;
    }
};

//~ int block_size = sqrt(MAXN);

void add(int pos){
    //~ Faz alguma coisa: conta frequência, por exemplo
    //~ Adiciona o elemento v[pos] no intervalo
}

void del(int pos){
    //~ Faz alguma coisa: conta frequência, por exemplo
    //~ Remove o elemento v[pos] no intervalo
}

bool compare(query &a, query &b){
    if(a.l / block_size == b.l / block_size) return a.r < b.r;
    return a.l < b.l;
    //se o bloco do left for o mesmo, ordena pelo r, senão ordena por l
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++)  //leitura do vetor de entrada
        cin >> v[i];

    int L, R;
```

```
36      cin >> q;
37      for (int i = 0; i < q; i++)  //leitura de query
38      {
39          cin >> L >> R;
40          L--; R--;
41          queries[i] = query(L, R, i);
42      }
43      sort(queries, queries+q, compare);//ordena as queries
44      int currL = 0, currR = 0;
45
46      for (int i = 0; i < q; i++)
47      {
48          L = queries[i].l;
49          R = queries[i].r;
50          while (currL < L)
51              del(currL++);   //remove elemento da posição currL
52          while (currR <= R)
53              add(currR++);   //adiciona elemento da posição currR
54          while (currL > L)
55              add(--currL);   //adiciona elemento da posição currL-1
56          while (currR > R+1)
57              del(--currR);   //remove elemento da posição currR-1
58
59          saida[queries[i].pos] = resposta;   //reordena a saída
60      }
61      for (int i = 0; i < q; i++)
62          cout << saida[i] << "\n";
63      return 0;
64  }
```

6.2 MO em Árvore

Listagem 6.2: MO em Árvore

```
1  //~ CONTAR QUANTOS PESOS DISTINTOS TEM NO CAMINHO DE U PRA V
2
3  struct query{
4      int l, r, lca, pos;
5      query(){}
6      query(int a, int b, int c, int d){
7          l = a;
8          r = b;
9          lca = c;
10         pos = d;
11     }
12 };
13
14 query queries[MAXN];
15 int lca[MAXN][LOG];
16 int valor[MAXN];
17 unordered_map<string, int> mapa;
18 unordered_map<string, int>::iterator it;
19 vector<int> g[MAXN];
20 int ini[MAXN];
21 int fim[MAXN];
```

```
22 int h[MAXN];
23 int ans[MAXN];
24 int f[MAXN];
25 int n, q;
26 vector<int> euler;
27 int block_size = 450;
28 int counter = 0;
29 int total = 0;
30 char vis[MAXN];
31
32 inline bool compare(const query &a, const query &b){
33     if(a.l/block_size == b.l/block_size) return a.r < b.r;
34     return a.l < b.l;
35 }
36
37 inline void dfs(int u, int pai){
38     lca[u][0] = pai;
39     for(int i = 1; i < LOG; i++)
40         lca[u][i] = lca[lca[u][i-1]][i-1];
41
42     euler.pb(u);
43     int v;
44     ini[u] = counter++;
45     for (int i = 0; i < g[u].size(); i++)
46     {
47         v = g[u][i];
48         if(v==pai) continue;
49         h[v] = h[u]+1;
50         dfs(v, u);
51     }
52     fim[u] = counter++;
53     euler.pb(u);
54 }
55
56 inline int getLca(int u, int v){
57     if(h[u] < h[v]) swap(u, v);
58     int dist = abs(h[u]-h[v]);
59
60     for (int i = LOG-1; i >= 0; i--)
61     {
62         if(dist & (1<<i))
63             u = lca[u][i];
64     }
65     if(u==v) return u;
66
67     for (int i = LOG-1; i >= 0; i--)
68     {
69         if(lca[u][i] != lca[v][i]){
70             u = lca[u][i];
71             v = lca[v][i];
72         }
73     }
74     return lca[u][0];
75 }
76
77 inline void add(int pos){
78     int u = euler[pos];
79     int val = valor[u];
80     if(vis[u]){
```

```
81          f[val]--;
82          if(f[val]==0) total--;
83      }else{
84          f[val]++;
85          if(f[val]==1) total++;
86      }
87      vis[u] ^= 1;
88  }
89
90  inline void del(int pos){
91      add(pos);
92  }
93
94  int main(){
95      ios_base::sync_with_stdio (0);
96      cin.tie (0);
97
98      int nxtIdx=0, u, v;
99      string s;
100     cin >> n >> q;
101     for (int i = 0; i < n; i++)
102     {
103         cin >> s;
104         it = mapa.find(s);
105         if(it == mapa.end()){
106             mapa[s] = nxtIdx++;
107         }
108         valor[i] = mapa[s];
109     }
110
111     for (int i = 0; i < n-1; i++)
112     {
113         cin >> u >> v;
114         u--; v--;
115         g[u].pb(v);
116         g[v].pb(u);
117     }
118
119     h[0] = 0;
120     dfs(0, 0);
121     int p;
122     for (int i = 0; i < q; i++)
123     {
124         cin >> u >> v;
125         u--; v--;
126         if(ini[u] > ini[v]) swap(u, v);
127         p = getLca(u, v);
128         if(p==u){
129             queries[i] = query(ini[u], ini[v], -1, i);
130         }else{
131             queries[i] = query(fim[u], ini[v], p, i);
132         }
133     }
134
135     sort(queries, queries+q, compare);
136     int L, R;
137     int currL=0, currR=0;
138     for (int i = 0; i < q; i++)
139     {
```

```
140         L = queries[i].l;
141         R = queries[i].r;
142 
143         while (currR <= R)
144         {
145             add(currR);
146             currR++;
147         }
148         while (currL > L)
149         {
150             add(currL-1);
151             currL--;
152         }
153         while (currL < L)
154         {
155             del(currL);
156             currL++;
157         }
158         while (currR > R+1)
159         {
160             del(currR-1);
161             currR--;
162         }
163 
164         if(queries[i].lca!=-1){
165             add(ini[queries[i].lca]);
166         }
167 
168         ans[queries[i].pos] = total;
169 
170         if(queries[i].lca!=-1){
171             del(ini[queries[i].lca]);
172         }
173     }
174 
175     for (int i = 0; i < q; i++)
176     {
177         cout << ans[i] << "\n";
178     }
179 
180     return 0;
181 }
```

6.3 SQRT decomposition em blocos

Listagem 6.3: SQRT decomposition em blocos

```
1 int n, q;
2 vector<int> block[600];
3 int block_size = 600;
4 int v[100010];
5 
6 int ini(int blocoAtual){ return blocoAtual*block_size; }
7 int fim(int blocoAtual){ return min(ini(blocoAtual+1) - 1, n-1); }
8 
```

```
9  int func(int blocoAtual, int X){//calcula quantos elementos <= X tem no
       blocoAtual
10     int ans = upper_bound(block[blocoAtual].begin(), block[blocoAtual].end
           (), X) - block[blocoAtual].begin();
11     return ans;
12 }
13
14 void update(int pos, int val){//atualiza só o bloco afetado
15     int valAntigo = v[pos];
16     int blocoAtual = pos / block_size;
17     v[pos] = val;
18
19     for(int i = 0; i < block[blocoAtual].size(); i++){
20         if(block[blocoAtual][i] == valAntigo){
21             block[blocoAtual][i] = val;
22             break;
23         }
24     }
25     sort(block[blocoAtual].begin(), block[blocoAtual].end());//ordena o
           bloco de novo
26 }
27
28 /*
29 int query(int L, int R, int X){
30     int blocoL, blocoR;
31     blocoL = L / block_size;
32     blocoR = R / block_size;
33     int pos;
34     int ans = 0LL;
35
36     for(pos = L; pos <= min(R, fim(blocoL)); pos++)
37         if(v[pos] <= X) ans++;
38
39     for (int i = blocoL+1; i <= blocoR-1; i++)
40         ans += func(i, X);
41
42     for(pos = max(pos, ini(blocoR)); pos <= R; pos++)
43         if(v[pos] <= X) ans++;
44
45     return ans;
46 }
47 */
48
49 int query(int L, int R, int X){//retorna quantos elementos <= X tem em [L,
       R]
50     int blocoL, blocoR;
51     blocoL = L / block_size;
52     blocoR = R / block_size;
53     int pos;
54     int ans = 0LL;
55     //para blocos que não estão inteiros dentro do intervalo: percorre em
           O(n)
56     //para blocos que estão inteiros dentr do intervalo: faz uma busca bin
           ária pra saber quantos elementos <= X existe
57     for (int i = 0; i < block_size; i++)
58     {
59         if(ini(i) > R) break;
60         if(ini(i) >= L && fim(i) <= R) ans += func(i, X);
61         else{
```

```
62              for(int j=max(ini(i), L); j<=min(fim(i), R); j++) ans += (v[j]
                    <= X);
63          }
64      }
65      return ans;
66  }
67
68  int main(){
69      cin >> n >> q;
70
71      for (int i = 0; i < n; i++)
72      {
73          cin >> v[i];
74          block[i/block_size].pb(v[i]);//adiciona no bloco correspondente
75      }
76      for (int i = 0; i < block_size; i++)
77      {
78          if(block[i].size()==0) break;
79          sort(block[i].begin(), block[i].end());//ordena cada bloco
80      }
81
82      char op;
83      int L, R, X, pos, val;
84      for (int i = 0; i < q; i++)
85      {
86          cin >> op;
87          if(op=='C'){
88              cin >> L >> R >> X;
89              cout << query(L-1, R-1, X) << "\n";
90          }else{
91              cin >> pos >> val;
92              update(pos-1, val);
93          }
94      }
95      return 0;
96  }
```

Capítulo 7

Math

7.1 Floyd's Cycle Finding

Complexidade: $O(tamanhoMaximoCiclo)$

Listagem 7.1: Floyd's Cycle Finding Algorithm

```
int f(int x){
    //dado pelo problema
}
ii solve(int L){
    int cycle_len, cycle_begin;
    int x=f(L), y = f(f(L));

    while(x!=y) {  // faz os dois ponteiros se encontrarem
        x = f(x);
        y = f(f(y));
    }

    cycle_len = 1;
    y = f(x);
    while(x!=y){   // anda com um e descobre o tamanho do ciclo
        cycle_len++;
        y = f(y);
    }

    x = y = L;
    for(int i=0; i<cycle_len; i++) y = f(y);

    cycle_begin = 0;
    while(x!=y){   // acha o começo do ciclo
        x = f(x);
        y = f(y);
        cycle_begin++;
    }
    return ii(cycle_begin, cycle_len);
}
```

7.2 Crivo de Erastótenes

Complexidade: $O(nlogn)$

Listagem 7.2: Crivo de Erastótenes

```
int last[maxn];
vector<int> p;
void sieve(){
    for(ll i=2;i<maxn;i++){
        if(last[i])
            continue;
        p.push_back(i);
        last[i] = i;
        for(ll j=i*i;j<maxn;j+=i)
            last[j] = i;
    }
}
```

7.3 Fatoração

7.3.1 Fatoração de números até 10^7

Complexidade: $O(logn)$

Listagem 7.3: Fatoração usando crivo

```
vector<int> fatoracao(int n){
    vector<int> ans;
    while(n != 1){
        ans.push_back(last[n]);
        n /= last[n];
    }
    return ans;
}
```

7.3.2 Fatoração de números até 10^{14}

Complexidade: $O(numPrimos \leq sqrt(n))$

Listagem 7.4: Fatoração usando crivo para números grandes

```
vi primeFactors(ll N) {
      vi factors;
      ll PF_idx = 0, PF = primes[PF_idx];
      while (N != 1 && (PF * PF <= N)) {

            while (N % PF == 0) { N /= PF; factors.push_back(PF); } //
               remove this PF
            PF = primes[++PF_idx];                                  //
               only consider primes!
```

```
8       }
9       if (N != 1) factors.push_back(N);   // special case if N is actually
            a prime
10      return factors;                     // if pf exceeds 32-bit integer,
            you have to change vi
11  }
```

7.3.3 Número de fatores primos

Complexidade: $O(numPrimos \leq sqrt(N))$

Listagem 7.5: Número de fatores primos

```
1  ll numPF(ll N) {
2       ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
3       while (N != 1 && (PF * PF <= N)) {
4             while (N % PF == 0) { N /= PF; ans++; }
5             PF = primes[++PF_idx];
6       }
7       if (N != 1) ans++;
8       return ans;
9  }
```

7.3.4 Número de fatores primos distintos

Complexidade: $O(numPrimos \leq sqrt(N))$

Listagem 7.6: Número de fatores primos distintos

```
1  ll numDiffPF(ll N) {
2       ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
3       while (N != 1 && (PF * PF <= N)) {
4             if (N % PF == 0) ans++;              // count this pf only
                 once
5             while (N % PF == 0) N /= PF;
6             PF = primes[++PF_idx];
7       }
8       if (N != 1) ans++;
9       return ans;
10 }
```

7.3.5 Soma de fatores primos

Complexidade: $O(numPrimos \leq sqrt(N))$

Listagem 7.7: Soma de fatores primos

```
1  ll sumPF(ll N) {
2       ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
3       while (N != 1 && (PF * PF <= N)) {
4             while (N % PF == 0) { N /= PF; ans += PF; }
```

```
5               PF = primes[++PF_idx];
6       }
7       if (N != 1) ans += N;
8       return ans;
9 }
```

7.3.6 Número de divisores

Complexidade: $O(numPrimos \leq sqrt(N))$

Listagem 7.8: Número de divisores

```
1 ll numDiv(ll N) {
2       ll PF_idx = 0, PF = primes[PF_idx], ans = 1;      // start from ans
            = 1
3       while (N != 1 && (PF * PF <= N)) {
4             ll power = 0;                                // count the power
5             while (N % PF == 0) { N /= PF; power++; }
6             ans *= (power + 1);                          // according to
                  the formula
7             PF = primes[++PF_idx];
8       }
9       if (N != 1) ans *= 2;                              // (last factor
            has pow = 1, we add 1 to it)
10      return ans;
11 }
```

7.3.7 Soma dos divisores

Complexidade: $O(numPrimos \leq sqrt(N))$

Listagem 7.9: Soma dos divisores

```
1 ll sumDiv(ll N) {
2       ll PF_idx = 0, PF = primes[PF_idx], ans = 1;                    //
            start from ans = 1
3       while (N != 1 && (PF * PF <= N)) {
4             ll power = 0;
5             while (N % PF == 0) { N /= PF; power++; }
6             ans *= ((ll)pow((double)PF, power + 1.0) - 1) / (PF - 1);  //
                  formula
7             PF = primes[++PF_idx];
8       }
9       if (N != 1) ans *= ((ll)pow((double)N, 2.0) - 1) / (N - 1);     //
            last one
10      return ans;
11 }
```

7.4 Teste de primalidade

7.4.1 Números até 10^{14}

Complexidade: $O(primos \leq sqrt(N))$

Listagem 7.10: Teste de primalidade

```
bool isPrime(ll N) {
      if (N <= _sieve_size) return bs[N];
      for (int i = 0; i < (int)primes.size(); i++)
            if (N % primes[i] == 0) return false;
      return true;
}
```

7.4.2 Números até 10^{16}

Complexidade: $O(sqrt(n))$

Listagem 7.11: Teste de primalidade para números grandes

```
bool isPrime(ll n){
    for(int i=2;i<=sqrt(n);i++)
        if(n % i == 0)
            return false;
    return true;
}
```

7.5 Função Totient (Euler phi)

7.5.1 Números até 10^7

Complexidade: $O(logn)$

Listagem 7.12: Totient usando crivo

```
int totientSieve(int n){
    double ans = n;
    int lastp = n+1;
    while(n != 1){
        if(lastp != last[n])
            ans *= 1 - (1.0 / last[n]);
        lastp = last[n];
        n /= last[n];
    }
    return int(ans);
}
```

7.5.2 Números até 10^{14}

Complexidade: $O(numPrimos \leq sqrt(n))$

Listagem 7.13: Totient usando crivo para números grandes

```
ll EulerPhi(ll N) {
      ll PF_idx = 0, PF = primes[PF_idx], ans = N;  // start from ans = N
      while (N != 1 && (PF * PF <= N)) {
            if (N % PF == 0) ans -= ans / PF;       // only count unique
               factor
            while (N % PF == 0) N /= PF;
            PF = primes[++PF_idx];
      }
      if (N != 1) ans -= ans / N;                   // last factor
      return ans;
}
```

7.5.3 Números até 10^{16}

Complexidade: $O(sqrt(n))$

Listagem 7.14: Totient sem crivo para números grandes

```
ll totient(ll n){
    double ans = n;
    for(int i=2;i<=sqrt(n);i++){
        if(n % i)
            continue;
        if(isPrime(i))
            ans *= 1 - (1.0 / i);
        if(n/i != i && isPrime(n/i))
            ans *= 1 - (1.0 / (n/i));
    }
    if(isPrime(n))
        ans *= 1 - (1.0 / n);
    return ans;
}
```

7.6 Algoritmo de Euclides Extendido

Complexidade: $O(log(max(a, b)))$

Listagem 7.15: Algoritmo de Euclides Extendido

```
ll mdc, x, y; // ax + by = mdc(a, b);
void extendEuclid(ll a, ll b){
    if(b == 0){
        mdc = a;
        x = 1;
        y = 0;
    }
    else{
```

```
9           extendEuclid(b, a%b);
10          ll aux = x;
11          x = y;
12          y = aux - (a/b)*y;
13      }
14  }
```

7.7 Inverso modular

7.7.1 Quando *mod* é primo e $\leq 10^7$

Complexidade: $O(nlogn)$ ou $O(n)$

Listagem 7.16: Inverso modular para números primos

```
1  ll modPow(ll n, ll k, ll mod){
2      if(!k)
3          return 1LL;
4      ll aux = modPow(n, k/2, mod);
5      aux = (aux * aux) % mod;
6      return k % 2 ? (aux * n) % mod : aux;
7  }
8
9  ll inverso(ll n, ll mod){
10   inv[0] = inv[1] = 1;
11   for(int i=2;i<n;i++){
12     inv[i] = modPow(n, mod-2, mod);                // opcao 1
13     inv[i] = ((mod - mod/i) * inv[mod%i]) % mod; // opcao 2
14   }
15 }
```

7.7.2 Quando *mod* é composto ou muito grande

Complexidade: $O(log(max(n, mod)))$ ou $O(O(totient) + log(totient)$

Listagem 7.17: Inverso modular para números compostos ou grandes

```
1  ll modEuclid(ll n, ll mod){                  // opcao 1
2      extendEuclid(n, mod);
3      x = ((x % m) + m) % m;
4      return x;
5  }
6
7  ll modPow(ll n, ll k, ll mod){               // opcao 2
8      if(!k)
9          return 1LL;
10     ll aux = modPow(n, k/2, mod);
11     aux = (aux * aux) % mod;
12     return k % 2 ? (aux * n) % mod : aux;
13 }
14
15 ll inverso(ll n, ll mod){
```

```
16   return modEuclid(n, mod);                // opcao 1
17   return modPow(n, totient(mod)-1, mod); // opcao 2
18 }
```

7.7.3 Preprocessamento para problemas que usem fatorial

Complexidade: $O(n)$ ou $O(nlogn)$

Listagem 7.18: Preprocessamento

```
1 void pre(){
2   fat[0] = fat[1] = den[0] = den[1] = deni[0] = deni[1] = 1;
3   for(int i=2;i<1005;i++){
4     fat[i] = (fat[i-1] * i) % mod;
5
6     // opcao 1:
7     deni[i] = ((mod - mod/i) * deni[mod%i]) % mod;
8     den[i] = deni[i] * den[i-1] % mod;
9
10    // opcao 2:
11    den[i] = modPow(fat[i], mod-2, mod);
12  }
13 }
```

7.8 Teoria dos jogos

7.8.1 Misère Nim

Lembrar do caso especial pra nim $g(i) = 1$: se só tem 1's, FIRST ganha se for par; senão FIRST ganha se $XOR \neq 0$.

Listagem 7.19: Misère Nim

```
1 int main(){
2     int t;
3     scanf("%d", &t);
4
5     while(t--){
6         int n;
7         scanf("%d", &n);
8
9         int ans = 0;
10        int maior = 0;
11        for(int i=0;i<n;i++){
12            int a;
13            scanf("%d", &a);
14
15            ans ^= a;
16            maior = max(maior, a);
17        }
18        if(maior <= 1) // so tem pilha de uns
```

```
19              printf("%s\n", !ans ? "First" : "Second");
20          else
21              printf("%s\n", ans ? "First" : "Second");
22      }
23      return 0;
24  }
```

7.8.2 Nim padrão

Calcular grundy number: First ganha se, e somente se, $XOR! = 0$.

Listagem 7.20: Nim Padrão

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  typedef long long ll;
6
7  map<int, int> primos;
8  map<int, int> dp;
9
10 int f(int i){
11     if(i <= 1)
12         return i;
13     if(dp.count(i))
14         return dp[i];
15
16     set<int> out;
17     for(int j=0;(1<<j) <= i;j++){ // todo bit >= j shifta j pos pra
           direita, o resto fica igual
18         int ficanormal = i & ((1<<j)-1);
19         int vaishiftar = (i ^ ficanormal);
20         // printf("%d, %d -> %d, %d, %d\n", i, j, vaishiftar, ficanormal,
               (vaishiftar>>(j+1))|ficanormal);
21         out.insert(f((vaishiftar>>(j+1)) | ficanormal));
22     }
23
24     int ans = 0;
25     for(set<int>::iterator it = out.begin();it!=out.end();it++){
26         if(*it != ans)
27             break;
28         ans++;
29     }
30     // printf("%d %d\n", i, ans);
31     return dp[i] = ans;
32 }
33
34 main(){
35     int n;
36     scanf("%d", &n);
37
38     for(int i=0;i<n;i++){
39         int a;
40         scanf("%d", &a);
41
```

```
42          for(int j=2;j<=sqrt(a);j++){
43              int c = 0;
44              while(a % j == 0){
45                  a /= j;
46                  c++;
47              }
48              if(c){
49                  if(!primos.count(j))
50                      primos[j] = 0;
51                  primos[j] |= (1<<c-1);
52              }
53          }
54          if(a != 1)
55              primos[a] |= 1;
56      }
57      int ans = 0;
58      for(map<int, int>::iterator it = primos.begin(); it != primos.end();it
            ++){
59          // printf("p=%d => f(%d) = %d\n", it->first, it->second, f(it->
                second));
60          ans ^= f(it->second);
61      }
62
63      printf("%s\n", ans ? "Mojtaba" : "Arpa");
64  }
```

Capítulo 8

JAVA

8.1 Exemplo BigDecimal

Listagem 8.1: Código ERRADO retornando o Exception

```
1 import java.math.BigDecimal;
2
3 public class MyAppBigDecimal {
4     /**
5     * @param args
6     */
7     public static void main(String[] args) {
8         System.out.println("Divide");
9         System.out.println(new BigDecimal("1.00").divide(new BigDecimal("
              1.3")));
10    }
11 }
12
13 /*
14 Saída:
15 Divide
16 Exception in thread "main" java.lang.ArithmeticException: Non-terminating
       decimal expansion; no exact representable decimal result.
17        at java.math.BigDecimal.divide(BigDecimal.java:1603)
18        at MyAppBigDecimal.main(MyAppBigDecimal.java:11)
19 */
```

Listagem 8.2: Código CERTO retornando o valor arredondado

```
1 import java.math.BigDecimal;
2 import java.math.RoundingMode;
3     public class MyAppBigDecimal {
4         /**
5         * @param args
6         */
7         public static void main(String[] args) {
8
9         System.out.println("Divide");
```

```
10          System.out.println(new BigDecimal("1.00").divide(new BigDecimal("
                1.3"),3,RoundingMode.UP));
11      }
12  }
13
14  /*
15  Saída:
16  Divide
17  0.770
18  */
```

Arredondamentos:

- CEILING: Rounding mode to round towards positive infinity.
- DOWN: Rounding mode to round towards zero.
- FLOOR: Rounding mode to round towards negative infinity.
- HALF_DOWN: Rounding mode to round towards "nearest neighbor"unless both neighbors are equidistant, in which case round down.
- HALF_EVEN: Rounding mode to round towards the "nearest neighbor"unless both neighbors are equidistant, in which case, round towards the even neighbor.
- HALF_UP: Rounding mode to round towards "nearest neighbor"unless both neighbors are equidistant, in which case round up.
- UNNECESSARY: Rounding mode to assert that the requested operation has an exact result, hence no rounding is necessary.
- UP: Rounding mode to round away from zero.

8.2 Inverter String

Listagem 8.3: Inverter String

```
1      a = new StringBuilder(a).reverse().toString();
```

8.3 Ordenação

Listagem 8.4: Diferentes tipos de sort

```
1  // ORDENAR UM ARRAY: usar arrays.sort(...);
2
3  int[] array = new int[10];
4  Random rand = new Random();
5  for (int i = 0; i < array.length; i++)
```

```
6     array[i] = rand.nextInt(100) + 1;
7 Arrays.sort(array);
8 System.out.println(Arrays.toString(array));
9 // in reverse order
10 for (int i = array.length - 1; i >= 0; i--)
11     System.out.print(array[i] + " ");
12 System.out.println();
13
14
15 //ORDENAR UM ARRAYLIST: usar Collections.sort(...);
16
17 Collections.sort(mArrayList, new Comparator<CustomData>() {
18     @Override
19     public int compare(CustomData lhs, CustomData rhs) {
20         // -1 - less than, 1 - greater than, 0 - equal, all inversed for
               descending
21         return lhs.customInt > rhs.customInt ? -1 : (lhs.customInt < rhs.
               customInt) ? 1 : 0;
22     }
23 });
```

Capítulo 9

Outros

9.1 Função Random

Listagem 9.1: Rand long long

```
typedef unsigned long long int llu;
llu seed = 0;
llu my_rand() {
    seed ^= llu(102938711);
    seed *= llu(109293);
    seed ^= seed >> 13;
    seed += llu(1357900102873);
    return seed;
}

int main () {
    //rand c++
    srand(time(NULL));
    cout << rand() << endl;

    //rand Endagorion / FMota
    cout << my_rand() << endl;

    return 0;
}
```

9.2 Radix Sort

Listagem 9.2: Radix Sort

```
//esse codigo so funciona pra numeros positivos na base 10

#include <bits/stdc++.h>
using namespace std;
#define N 10101000

```

```
7  int n, vet[N], cnt[10], tmp[N];
8
9  void counting_sort(int k){
10     memset(cnt, 0, sizeof cnt);
11
12     for(int i=0; i<n; i++){
13         cnt[ (vet[i]/k)%10 ]++;
14     }
15     for(int i=1; i<10; i++){
16         cnt[i]+=cnt[i-1];
17     }
18
19     for(int i=n-1; i>=0; i--){
20         tmp[ --cnt[ (vet[i]/k)%10 ] ] = vet[i];
21     }
22     for(int i=0; i<n; i++) vet[i] = tmp[i];
23 }
24
25 void radix(){
26     int b = 0;
27     for(int i=0; i<n; i++) b = max(b, (int)floor(log10(vet[i])));
28
29     for(int i=0, exp=1; i<=b; i++, exp*=10){
30         counting_sort(exp);
31     }
32 }
33
34 int main(){
35     while(scanf("%d", &n), n){
36         for(int i=0; i<n; i++) scanf("%d", &vet[i]);
37
38         radix();
39
40         printf("VETOR:");
41         for(int i=0; i<n; i++) printf(" %d", vet[i]);
42         printf("\n");
43     }
44
45 }
```

9.3 SCANINT

Listagem 9.3: Scanint

```
1  #define gc getchar_unlocked // ou usar só getchar
2
3  void scanint(ll &x)
4  {
5      register ll c = gc();
6      x = 0;
7      for(;(c<48 || c>57);c = gc());
8      for(;c>47 && c<58;c = gc()) {x = (x<<1) + (x<<3) + c - 48;}
9  }
10
11 int read_int(){
```

```
12      char r;
13      bool start=false,neg=false;
14      int ret=0;
15      while(true){
16          r=getchar();
17          if((r-'0'<0 || r-'0'>9) && r!='-' && !start){
18              continue;
19          }
20          if((r-'0'<0 || r-'0'>9) && r!='-' && start){
21              break;
22          }
23          if(start)ret*=10;
24          start=true;
25          if(r=='-')neg=true;
26          else ret+=r-'0';
27      }
28      if(!neg)
29          return ret;
30      else
31          return -ret;
32  }
```

9.4 Functions

Listagem 9.4: Functions

```
1  bool a(int x){
2      if(x>10) {
3          return true;
4      }
5      return false;
6  }
7  bool b(int x){
8      if(x>100) {
9          return true;
10     }
11     return false;
12 }
13 bool c(int x){
14     if(x>1000) {
15         return true;
16     }
17     return false;
18 }
19
20 void printa(int x){
21     if(x==1) printf("<100\n");
22     else if(x==2) printf("<1000\n");
23     else if(x == 3) printf(">1000\n");
24     else printf("<10\n");
25 }
26
27 int main(){
28     vector<  function<bool(int)>  > vet;
29     vet.push_back(a);
```

```
30      vet.push_back(b);
31      vet.push_back(c);
32      int x;
33      function<int(int)> f = [x](int i){
34              return i<=x;
35      };
36
37      auto f_ = [x](int i){
38              return i<=x;
39      };
40
41      scanf("%d", &x);
42      for(int i=0; i<vet.size(); i++){
43          if( !vet[i](x) ){
44              printa(i);
45              return 0;
46          }
47      }
48      printa(vet.size());
49  }
```

9.5 Builtins

Listagem 9.5: Builtins

```
1  int __builtin_ffs (int x)
2  // Returns one plus the index of the least significant 1-bit of x,
3  // or if x is zero, returns zero.
4
5  int __builtin_clz (unsigned int x)
6  // Returns the number of leading 0-bits in x, starting at the most
7  // significant bit position. If x is 0, the result is undefined.
8
9  int __builtin_ctz (unsigned int x)
10 // Returns the number of trailing 0-bits in x, starting at the
11 // least significant bit position. If x is 0, the result is undefined.
12
13 int __builtin_clrsb (int x)
14 // Returns the number of leading redundant sign bits in x, i.e.
15 // the number of bits following the most significant bit that are
16 // identical to it. There are no special cases for 0 or other values.
17
18 int __builtin_popcount (unsigned int x)
19 // Returns the number of 1-bits in x.
20
21 int __builtin_parity (unsigned int x)
22 // Returns the parity of x, i.e. the number of 1-bits in x modulo 2.
23
24 int __builtin_ffsl (long)
25 // Similar to __builtin_ffs, except the argument type is long.
26
27 int __builtin_clzl (unsigned long)
28 // Similar to __builtin_clz, except the argument type is
29 // unsigned long.
30
```

```
int __builtin_ctzl (unsigned long)
// Similar to __builtin_ctz, except the argument type is
// unsigned long.

int __builtin_clrsbl (long)
// Similar to __builtin_clrsb, except the argument type is long.

int __builtin_popcountl (unsigned long)
// Similar to __builtin_popcount, except the argument type is
// unsigned long.

int __builtin_parityl (unsigned long)
// Similar to __builtin_parity, except the argument type is
// unsigned long.

int __builtin_ffsll (long long)
// Similar to __builtin_ffs, except the argument type is long long.

int __builtin_clzll (unsigned long long)
// Similar to __builtin_clz, except the argument type is
// unsigned long long.

int __builtin_ctzll (unsigned long long)
// Similar to __builtin_ctz, except the argument type is
// unsigned long long.

int __builtin_clrsbll (long long)
// Similar to __builtin_clrsb, except the argument type is long long.

int __builtin_popcountll (unsigned long long)
// Similar to __builtin_popcount, except the argument type is
// unsigned long long.

int __builtin_parityll (unsigned long long)
// Similar to __builtin_parity, except the argument type is
// unsigned long long.

double __builtin_powi (double, int)
// Returns the first argument raised to the power of the second. Unlike
// the pow function no guarantees about precision and rounding are made.

float __builtin_powif (float, int)
// Similar to __builtin_powi, except the argument and return types
// are float.

long double __builtin_powil (long double, int)
// Similar to __builtin_powi, except the argument and return types
// are long double.

uint16_t __builtin_bswap16 (uint16_t x)
// Returns x with the order of the bytes reversed; for example,
// 0xaabb becomes 0xbbaa. Byte here always means exactly 8 bits.

uint32_t __builtin_bswap32 (uint32_t x)
// Similar to __builtin_bswap16, except the argument and return
// types are 32 bit.

uint64_t __builtin_bswap64 (uint64_t x)
// Similar to __builtin_bswap32, except the argument and return
```

```
90 // types are 64 bit.
```