Apostila Puc-Rio

Thomaz Miranda, Miguel Batista, João Arthur Marques

Contents

Informações Importantes

n	**Worst AC Algorithm**
$\leq [10..11]$	$O(n!), O(n^6)$
$\leq [17..19]$	$O(2^n \times n^2)$
$\leq [18..22]$	$O(2^n \times n)$
$\leq [24..26]$	$O(2^n)$
≤ 100	$O(n^4)$
≤ 450	$O(n^3)$
$\leq 1.5K$	$O(n^{\{2.5\}})$
$\leq 2.5K$	$O(n^2 \log n)$
$\leq 10K$	$O(n^2)$
$\leq 200K$	$O(n^{\{1.5\}})$
$\leq 4.5M$	$O(n \log n)$
$\leq 10M$	$O(n \log \log n)$
$\leq 100M$	$O(n), O(\log n), O(1)$

Ideias Gerais

- Busca Binária
- Pareamento máximo
- Árvore geradora
- All points shortest path
- Fluxos
- E se a gente refizer removendo o item especial?
- E se a gente fizer o problema de tras pra frente
- O problema e uma quantidade entre [a,b],podemos reduzir pra <=b

Ideias de DP

- Olha pro tamanho dos números
- Cria o vetor de tamanho target
- Consegue escrever como uma equação linear?
- Olhar apostila do ITA

Template

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<ll, ll> p64;
typedef vector<ll> v64;

#define forn(i, s, e) for(ll i = (s); i < (e); i++)
#define ln "\n"

#if defined(DEBUG) || defined(debug)
    #define _ (void)0
    #define debug(x) cout << __LINE__ << ": " << #x << " = " <<
x << ln
#else
    #define _ ios_base::sync_with_stdio(false), cin.tie(NULL)
    #define debug(x) (void)0
#endif

const ll INF = 0x3f3f3f3f3f3f3f3fll;

int main(){
    _;
    return 0;
}
```

Math

Sieve

```cpp
v64 primes;
vector<bool> is_comp(MAXN,false);
ll phi[MAXN];
ll cum_sum[MAXN];

void sieve(ll n){
  phi[1] = 1;
  forn(i,2,n){
    if(!is_comp[i]){
      phi[i] = i-1;
      primes.push_back(i);
    }

    forn(j,0,primes.size()){
      if(i*primes[j] > n) break;
      is_comp[i*primes[j]] = true;

      if(i % primes[j] == 0){
        phi[i*primes[j]] = phi[i]*primes[j];
        break;
      }
      phi[i*primes[j]] = phi[i]*phi[primes[j]];
    }
  }
}
```

Data Structures

Segtree

```cpp
#define LC nd * 2 + 1
#define RC nd * 2 + 2

const T NEUTRAL = node_neutro;

struct SegTree {
  vector < T > tree;
  ll n;
  T op(const T & a,
    const T & b) {
    return a + b;
  }
```

```
  SegTree(const vector < T > & v): n(v.size()) {
    tree.resize(4 * n);
    _build(v, 0, 0, n - 1);
  }
  void update(ll idx,
    const T & val) {
    _update(idx, val, 0, 0, n - 1);
  }

  T query(ll l, ll r) {
    return _query(l, r, 0, 0, n - 1);
  }
  void _build(const vector < T > & v, ll nd, ll st, ll ed) {
    if (st == ed) {
      tree[nd] = v[st];
    } else {
      ll mid = (st + ed) / 2;
      _build(v, LC, st, mid);
      _build(v, RC, mid + 1, ed);
      tree[nd] = op(tree[LC], tree[RC]);
    }
  }
  void _update(ll idx,
    const T & val, ll nd, ll st, ll ed) {
    if (st == ed) tree[nd] = val;
    else {
      ll mid = (st + ed) / 2;
      if (idx <= mid) _update(idx, val, LC, st, mid);
      else _update(idx, val, RC, mid + 1, ed);
      tree[nd] = op(tree[LC], tree[RC]);
    }
  }
  T _query(ll l, ll r, ll nd, ll st, ll ed) {
    if (r < st || ed < l) return NEUTRAL;
    if (l <= st && ed <= r) return tree[nd];
    ll mid = (st + ed) / 2;
    T left = _query(l, r, LC, st, mid);
    T right = _query(l, r, RC, mid + 1, ed);
    return op(left, right);
  }
};
```

Treap

```
struct Treap{
    ll val;
    ll prio, size;
    vector<Treap*> kids;
    Treap(ll c): val(c), prio(rand()), size(1),
        kids({NULL,NULL}){};
};

ll size(Treap *me){return me ? me->size : 0;}
void rsz(Treap* me){me -> size =
    1 + size(me->kids[0]) + size(me->kids[1]);}

vector<Treap*> split(Treap *me, ll idx){
    if(!me) return {NULL,NULL};
    vector<Treap*> out;

    if(size(me->kids[0]) < idx){
        auto aux = split(me->kids[1],
            idx - size(me->kids[0]) -1);
        me->kids[1] = aux[0];
        rsz(me);
        out = {me, aux[1]};
    }else{
        auto aux = split(me->kids[0], idx);
        me->kids[0] = aux[1];
        rsz(me);
        out = {aux[0], me};
    }
    return out;
}

Treap* merge(Treap *left, Treap *right){
    if(left == NULL) return right;
```

```
    if(right == NULL) return left;

    Treap* out;

    if(left->prio < right->prio){
        left->kids[1] = merge(left->kids[1], right);
        rsz(left);
        out = left;
    }else{
        right->kids[0] = merge(left, right->kids[0]);
        rsz(right);
        out = right;
    }
    return out;
}
```

Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

#define ordered_set tree<p64, null_type,less<p64>,
rb_tree_tag,tree_order_statistics_node_update>

int main() {
    ordered_set s;
    s.find_by_order(position);
    s.order_of_key(value);
}
```

Sparse Table

```
ll m[MAXN][MAXLOGN];

void build(vector<long long>& v) {
    ll sz = v.size();

    forn(i, 0, sz) {
        m[i][0] = v[i];
    }

    for (ll j = 1; (1 << j) <= sz; j++) {
        for (ll i = 0; i + (1 << j) <= sz; i++) {
            m[i][j] = max(m[i][j-1], m[i + (1 << (j-1))][j-1]);
        }
    }
}

ll query(ll a, ll b) {
    ll j = __builtin_clzll(1) - __builtin_clzll(b - a + 1);
    return max(m[a][j], m[b - (1 << j) + 1][j]);
}
```

Union Find

```
ll parent[MAXN]; //Inicializar parent[i] = i
ll sz[MAXN];

ll find(ll x){
    ll pai = x;
    ll filho = x;
    ll aux;
    while( pai != parent[pai] ) pai = parent[pai];
    while( filho != parent[filho] ) {
        aux = parent[filho];
        parent[filho] = pai;
        filho = aux;
    }
    return pai;
}

void uni(ll x, ll y){
```

```
    ll rx = find(x);
    ll ry = find(y);
    if(rx == ry) return;
    if( sz[rx] < sz[ry]) swap(rx,ry);
    parent[ry] = rx;
    sz[rx] += sz[ry];
    return;
}
```

Fenwick Tree

```
//1-indexed
struct BIT {
    v64 ft;
    BIT(ll n) {
        ft.assign(n+1, 0);
    }

    ll rsq(ll i) {
        ll sum = neutral;
        for (;i; i-=(i&-i)) sum = comp(sum, ft[i]);
        return sum;
    }

    ll rsq(ll i, ll j) {
        return rsq(j) - rsq(i-1);
    }

    void add(ll i, ll v) {
        for(;i<(ll)ft.size(); i+= (i&-i)) ft[i] = comp(ft[i],
v);
    }
};
```

Misc

Binary Search

```
ll find_last_valid(ll val) {
    ll left = 0;
    ll right = n - 1;
    ll result = -1;

    while (left <= right) {
        ll mid = left + (right - left) / 2;
        if (condition) {
            result = mid;
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return result;
}

ll find_first_valid(ll val) {
    ll left = 0;
    ll right = n - 1;
    ll result = n;

    while (left <= right) {
        ll mid = left + (right - left) / 2;
        if (condition) {
            result = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
    return result;
}
```

Graphs

Dinitz

```
struct dinitz {
  const bool scaling = true;
  ll lim;
  struct edge {
    ll to, cap, rev, flow;
    bool res;
    edge(ll to_, ll cap_, ll rev_, bool res_)
      : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
  };

  vector<vector<edge>> g;
  vector<ll> lev, beg;
  ll F;
  dinitz(ll n) : g(n), F(0) {}

  void add(ll a, ll b, ll c) {
    g[a].emplace_back(b, c, g[b].size(), false);
    g[b].emplace_back(a, 0, g[a].size()-1, true);
  }

  bool bfs(ll s, ll t) {
    lev = vector<ll>(g.size(), -1); lev[s] = 0;
    beg = vector<ll>(g.size(), 0);
    queue<ll> q; q.push(s);
    while (q.size()) {
      ll u = q.front(); q.pop();
      for (auto& i : g[u]) {
        if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
        if (scaling and i.cap - i.flow < lim) continue;
        lev[i.to] = lev[u] + 1;
        q.push(i.to);
      }
    }
    return lev[t] != -1;
  }

  ll dfs(ll v, ll s, ll f = INF) {
    if (!f or v == s) return f;
    for (ll& i = beg[v]; i < g[v].size(); i++) {
      auto& e = g[v][i];
      if (lev[e.to] != lev[v] + 1) continue;
      ll foi = dfs(e.to, s, min(f, e.cap - e.flow));
      if (!foi) continue;
      e.flow += foi, g[e.to][e.rev].flow -= foi;
      return foi;
    }
    return 0;
  }

  ll max_flow(ll s, ll t) {
    for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
      while (bfs(s, t)) while (ll ff = dfs(s, t)) F += ff;
    return F;
  }

  void reset() {
    F = 0;
    for (auto& edges : g) for (auto& e : edges) e.flow = 0;
  }
};
```

Sparse Dijkstra

```
vector<vp64> adj; // (v, w)

// d = distance | p = path
void dijkstra(ll s, v64 &d, v64& p) {
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);

    d[s] = 0;
    priority_queue<p64, vp64, greater<p64>> pq;
```

```
    pq.push({0, s});
    while (!pq.empty()) {
        ll u = pq.top().second;
        ll d_u = pq.top().first;
        pq.pop();

        if (d_u != d[u]) continue;

        for (auto edge : adj[u]) {
            ll v = edge.first;
            ll w_v = edge.second;

            if (d[u] + w_v < d[v]) {
                d[v] = d[u] + w_v;
                p[v] = u;
                pq.push({d[v], v});
            }
        }
    }
}
```

Dense Dijkstra

```
vector<vp64> adj; // (v, w)

// d = distance | p = path
void dijkstra(int s, v64& d, v64& p) {
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);
    vector<bool> visited(n, false);

    d[s] = 0;
    forn(i, 0, n) {
        ll u = -1;
        forn(j, 0, n) {
            if (!visited[j] && (u == -1 || d[j] < d[u])) u = j;
        }
        if (d[u] == INF) break;

        visited[u] = true;
        for (auto edge : adj[u]) {
            ll v = edge.first;
            ll w_v = edge.second;

            if (d[u] + w_v < d[v]) {
                d[v] = d[u] + w_v;
                p[v] = u;
            }
        }
    }
}
```

Strings

Trie

```
struct trie {
  vector<v64> to;
  v64 end, pref;
  ll sigma; char norm;

  trie(ll sigma_=26, char norm_='a') : sigma(sigma_),
norm(norm_) {
    to = {v64(sigma)};
    end = {0}, pref = {0};
  }

  void insert(string s) {
    ll x = 0;
    for (auto c : s) {
      ll &nxt = to[x][c-norm];
      if (!nxt) {
        nxt = to.size();
```

```
        to.push_back(v64(sigma));
        end.push_back(0), pref.push_back(0);
      }
      x = nxt, pref[x]++;
    }
    end[x]++, pref[0]++;
  }

  void erase(string s) {
    ll x = 0;
    for (char c : s) {
      ll &nxt = to[x][c-norm];
      x = nxt, pref[x]--;
      if (!pref[x]) nxt = 0;
    }
    end[x]--, pref[0]--;
  }

  ll find(string s) {
    ll x = 0;
    for (auto c : s) {
      x = to[x][c-norm];
      if (!x) return -1;
    }
    return x;
  }

  ll count_pref(string s) {
    ll id = find(s);
    return id >= 0 ? pref[id] : 0;
  }
};
```

Kmp

```
v64 pi(string& s) {
  v64 p(s.size());
  for (ll i = 1, j = 0; i < (ll) s.size(); i++) {
    while (j and s[j] != s[i]) j = p[j-1];
    if (s[j] == s[i]) j++;
    p[i] = j;
  }
  return p;
}

v64 match(string& pat, string& s) {
  v64 p = pi(pat), match;
  for (ll i = 0, j = 0; i < (ll) s.size(); i++) {
    while (j and pat[j] != s[i]) j = p[j-1];
    if (pat[j] == s[i]) j++;
    if (j == pat.size()) match.push_back(i-j+1), j = p[j-1];
  }
  return match;
}

struct KMPaut : vector<v64> {
  KMPaut(){}
  KMPaut (string& s) : vector<v64>(26, v64(s.size()+1)) {
    v64 p = pi(s);
    auto& aut = *this;
    aut[s[0]-'a'][0] = 1;
    for (char c = 0; c < 26; c++)
      for (int i = 1; i <= s.size(); i++)
        aut[c][i] = s[i]-'a' == c ? i+1 : aut[c][p[i-1]];
  }
};
```

Extra

Random

```
mt19937_64 rng((ll)
chrono::steady_clock::now().time_since_epoch().count());

ll uniform(ll l, ll r){
```

```
  uniform_int_distribution<ll> uid(l, r);
  return uid(rng);
}
```

Stresstest

```
P=a
make ${P} ${P}2 gen || exit 1
for ((i = 1; ; i++)) do
  ./gen $i > in
  ./${P} < in > out
  ./${P}2 < in > out2
  if (! cmp -s out out2) then
    echo "--> entrada:"
    cat in
    echo "--> saida1:"
    cat out
    echo "--> saida2:"
    cat out2
    break;
  fi
  echo $i
done
```