Handbook

October 5, 2022

Contents

1 Grafos **2**
1.1 Dinic 2

2 Strings **4**
2.1 Función Z 4
2.2 KMP 4
2.3 Suffix array 4
2.4 Longest Common Prefix on Suffixs 6

3 Búsqueda **8**
3.1 Ternary Search 8

4 Geometría **9**
4.1 Convex Hull 9

5 Matemáticas **11**
5.1 Inverso Modular 11
5.2 Miller-Rabin 11
5.3 Pollard Rho 12

1 Grafos

1.1 Dinic

Insertar utilidad del algoritmo:

C++:

```
struct edge{
  int x, y, flow;
};

int ans;
vector<edge> edges;
vector<vector<int> > grafo;
vector<int> sn;

void addEdge(int x, int y, int flow){
  grafo[x].PB(edges.size());
  edges.PB({x, y, flow});

  grafo[y].PB(edges.size());
  edges.PB({y, x, 0});
}

int bfs(int &ori, int &target){
  int x = ori, y, flow;

  FOR(i, 0, target + 1) sn[i] = INF;

  sn[x] = 0;
  deque<int> q(1, x);

  while(!q.empty()){
    x = q.F(); q.P_F();

    for(auto &e: grafo[x]){
      auto &edge = edges[e];
      y = edge.y;
      flow = edge.flow;

      if(flow <= 0) continue;
      if(sn[y] != INF) continue;
      sn[y] = sn[x] + 1;
      q.PB(y);
    }
  }

  return sn[target];
}

int dfs(int ori, int &target, int min_flow){
  int flow = INF, y;

  for(auto &e_id: grafo[ori]){
    auto &e = edges[e_id];
    y = e.y;
```

```

    if(sn[y] != 1 + sn[ori]) continue;
    if(e.flow <= 0) continue;

    if(y == target){
      flow = min(min_flow, e.flow);
      ans += flow;
      edges[e_id].flow -= flow;
      edges[e_id^1].flow += flow;
      return flow;
    }

    flow = dfs(y, target, min(min_flow, e.flow));

    if(flow != INF){
      edges[e_id].flow -= flow;
      edges[e_id^1].flow += flow;
      return flow;
    }
  }

  return flow;
}
```

2 Strings

2.1 Función Z

Insertar utilidad del algoritmo:

C++:

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; 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;
    }
    return z;
}
```

2.2 KMP

Insertar utilidad del algoritmo:

C++:

```
vector<int> z;

void kmp(string &s){
  int j, n = s.size();
  z.resize(n);

  FOR(i, 1, n){
    j = z[i - 1];
    while(j > 0 and s[i] != s[j]) j = z[j - 1];
    if(s[i] == s[j]) j++;
    z[i] = j;
  }
}
```

2.3 Suffix array

Devuelve un arreglo con el orden lexicográfico de los sufijos de un string S

C++:

```
vector<int> p, c;
void count_sort(vector<int> &p, vector<int> &c){
    int n = p.size();
    vector<int> cnt(n), p_new(n), pos(n);
    pos[0] = 0;
    for(auto x : c) cnt[x]++;
```

```
    for(int i = 1 ; i < n ; ++i) pos[i] = pos[i - 1] + cnt[i - 1];
    for(auto x : p){
        int i = c[x];
        p_new[pos[i]] = x;
        pos[i]++;
    }
    p = p_new;
}
vector<int> suffix_array(string &s){
    s+=" ";
    int n = s.size();
    p.resize(n);
    c.resize(n);
    vector<pair<char, int>> a(n);
    for(int i = 0 ; i < n ; ++i) a[i] = {s[i], i};
    sort(a.begin(), a.end());
    for(int i = 0 ; i < n ; ++i) p[i] = a[i].second;
    c[p[0]] = 0;
    for(int i = 1 ; i < n ; ++i) c[p[i]] = a[i].first == a[i - 1].first ? c[p[i - 1]] : c[p[i
    int k = 0, shift;
    while( (1<<k) < n ){
        shift = 1<<k;
        for(int i = 0 ; i<n ; ++i)
            p[i] = (p[i] - (1<<k) +  n) % n;
        count_sort(p,c);
        vector<int> c_new(n);
        c_new[p[0]] = 0;
        for(int i = 1 ; i < n ; ++i){
            pair<int, int> prev = {c[p[i - 1]], c[ (p[i - 1] + shift) % n]};
            pair<int, int> now = {c[p[i]], c[(p[i] + shift) % n]};
            if(prev == now) c_new[p[i]] = c_new[p[i - 1]];
            else c_new[p[i]] = c_new[p[i - 1]] + 1;
        }
        c = c_new;
        k++;
    }
    return p;
}
```

Java:

```
static int[]p, c;
public static class Suffix implements Comparable<Suffix> {
    int index, r, next;
    public Suffix(int index, int rank, int next){
        this.index = index; this.r = rank; this.next = next;
        }
    public int compareTo(Suffix s){
        return r != s.r ? r - s.r : (next != s.next ? next - s.next : index - s.index);
    }
}
public static int[] sort(int[] p, int[]c){
    int N = p.length;
    int[]cnt = new int[N], pos = new int[N], p_new = new int[N];;
    for(int e : c)  cnt[e]++;
        for(int i = 1 ; i < N ; ++i) pos[i] = pos[i - 1] + cnt[i - 1];
        for(int x : p){
            p_new[pos[c[x]]] = x; pos[c[x]]++;
```

```java
        }
        p = p_new;
        return p;
    }
public static int[] suffixArray(String s) {
    s+="$";
    int n = s.length();
    c = new int[n];
    p = new int[n];
    Suffix[] su = new Suffix[n];
    for (int i = 0; i < n; ++i) su[i] = new Suffix(i, s.charAt(i), 0);
    Arrays.sort(su);
    for(int i = 0 ; i < n ; ++i) p[i] = su[i].index;
    c[p[0]] = 0;
    for(int i = 1 ; i < n ; ++i) c[p[i]] = su[i].r == su[i - 1].r ?c[ p[i-1]] :c[p[i-1]] + 1;
    int k = 0, shift;
    while((1<<k) < n){
        shift = (1<<k);
        for(int i = 0 ; i < n ; ++i) p[i] = (p[i] - shift + n ) % n;
        p = sort(p, c);
        int[] c_new = new int[n];
        c_new[p[0]] = 0;
        for(int i = 1 ; i < n ; ++i)
            c_new[p[i]] = (c[p[i]] == c[p[i-1]] && c[(p[i]+shift) % n ] == c[(p[i - 1] + shift
                        ? c_new[p[i - 1]] : c_new[p[i - 1]] + 1;
        c = c_new;
        ++k;
    }
    return p;
}
```

2.4 Longest Common Prefix on Suffixs

Devuelve un arreglo que contiene el largo del prefijo común máximo entre 2 sufijos i e i+1

C++:

```cpp
vector<int> lcp(vector<int> &p, vector<int> &c, string &s){
    int n = p.size();
    vector<int> lcp(n);
    int k = 0;
    for(int i = 0 ; i < n - 1 ; ++i){
        int pi = c[i];
        int j = p[pi - 1];
        while(s[i + k] == s[j + k]) k++;
        lcp[pi] = k;
        k = max(k - 1, 0);
    }
    return lcp;
}
```

Java:

```java
static int[]p, c, LCP;

```

```java
static int[] lcp(int[] p, int[]c, String s){
    int n = p.length;
    LCP = new int[n];
    int k = 0;
    for(int i = 0 ; i < n - 1 ; ++i){
        int pi = c[i];
        int j = p[pi - 1];
        while(s.charAt(i + k) == s.charAt(j + k)) k++;
        LCP[pi] = k;
        k = Math.max(k - 1, 0);
    }
    return LCP;
}
```

3 Búsqueda

3.1 Ternary Search

Insertar utilidad del algoritmo:

C++:

```
#define ld long double

ld ternary_search(ld l, ld r) {
    ld eps = 1e-9;
    ld m1, m2, f1, f2;
    while (r - l > eps) {
        m1 = l + (r - l) / 3;
        m2 = r - (r - l) / 3;
        f1 = f(m1);       //evaluates the function at m1
        f2 = f(m2);       //evaluates the function at m2
        if (f1 < f2) l = m1;
        else r = m2;
    }

    //return the maximum of f(x) in [l, r]
    return f(l);
}
```

4 Geometría

4.1 Convex Hull

Insertar utilidad del algoritmo:

C++:

```cpp
struct pt {
  double x, y;
};

int orientation(pt a, pt b, pt c) {
  double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
  if (v < 0) return -1; // clockwise
  if (v > 0) return +1; // counter-clockwise
  return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
  int o = orientation(a, b, c);
  return o < 0 || (include_collinear && o == 0);
}
bool ccw(pt a, pt b, pt c, bool include_collinear) {
  int o = orientation(a, b, c);
  return o > 0 || (include_collinear && o == 0);
}

void convex_hull(vector<pt>& a, bool include_collinear = false) {
  if (a.size() == 1)
    return;

  sort(a.begin(), a.end(), [](pt a, pt b) {
    return make_pair(a.x, a.y) < make_pair(b.x, b.y);
  });
  pt p1 = a[0], p2 = a.back();
  vector<pt> up, down;
  up.push_back(p1);
  down.push_back(p1);
  for (int i = 1; i < (int)a.size(); i++) {
    if (i == a.size() - 1 || cw(p1, a[i], p2, include_collinear)) {
      while (up.size() >= 2){
        if(cw(up[up.size()-2], up[up.size()-1], a[i], include_collinear)) break;
        up.pop_back();
      }
      up.push_back(a[i]);
    }
    if (i == a.size() - 1 || ccw(p1, a[i], p2, include_collinear)) {
      while (down.size() >= 2){
        if(ccw(down[down.size()-2], down[down.size()-1], a[i], include_collinear)) break;
        down.pop_back();
      }
      down.push_back(a[i]);
    }
  }

  if (include_collinear && up.size() == a.size()) {
```

```
    reverse(a.begin(), a.end());
    return;
  }
  a.clear();
  for (int i = 0; i < (int)up.size(); i++)
    a.push_back(up[i]);
  for (int i = down.size() - 2; i > 0; i--)
    a.push_back(down[i]);
}
```

5 Matemáticas

5.1 Inverso Modular

Insertar utilidad del algoritmo:

C++:

```cpp
ll inv(int a){
  int n = mod - 2;
  ll dp[32], ans = 1;
  dp[0] = a;
  FOR(i, 1, 32) dp[i] = (dp[i - 1]*dp[i - 1])%mod;

  FOR(i, 0, 32){
    if(n & (1 << i)) ans = (ans*dp[i])%mod;
  }

  return ans;
}
```

5.2 Miller-Rabin

Insertar utilidad del algoritmo:

C++:

```cpp
vector<int> a{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

ll mult(ll a, ll b, ll mod) {
    return ((__int128)a * b) % mod;
}

/*
  a es la base.
  d es la potencia.
  n es el modulo.
*/
ll pw(ll a, ll d, ll n){ // pow in log(n)
  vector<ll> dp(63);
  dp[0] = a;
  ll res;

  FOR(i, 1, 63) dp[i] = mult(dp[i - 1], dp[i - 1], n);

  deque<int> bits;

  FOR(i, 0, 63) if(d & (ll)1 << i) bits.PB(i);

  res = dp[bits.F()]%n;
  bits.P_F();

  while(!bits.empty()){
    res = (mult(res, dp[bits.F()], n))%n;
    bits.P_F();
```

```
  }

  return res;
}

bool prime(ll n){ // test de primalidad
  ll r, x, m, d;
  bool out;
  r = 0;
  m = n - 1;

  while(m%2 == 0){
    m /= 2;
    r++;
  }
  d = m;

  FOR(i, 0, a.size()){
    x = pw(a[i], d, n);
    out = false;
    if(x == 1 or x == n - 1) continue;
    else{
      FOR(j, 0, r - 1){
        x = mult(x, x, n);
        if(x == n - 1){
          out = true;
          break;
        }
      }
    }

    if(out) continue;
    return false;
  }
  return true;
}
```

5.3 Pollard Rho

Insertar utilidad del algoritmo:

C++:

```
ll mult(ll a, ll b, ll mod) {
    return ((__int128)a * b) % mod;
}

ll f(ll x, ll c, ll mod) {
    return (mult(x, x, mod) + c) % mod;
}

ll rho(ll n) {
  ll c = 1, x, y, g;
  y = x = 2;
  g = c;
  while(g == 1){
```

```
    x = f(x, c, n);
    y = f(y, c, n);
    y = f(y, c, n);
    g = __gcd(abs(x - y), n);
  }
  return g;
}
```