Team notebook

BubleGum UC

June 12, 2024

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Contents

1 bfs

```
vector<int> dist(n, 1e9);
queue<int> q;

int start = 0;
dist[start] = 0;
q.push(start);

while(!q.empty())
{
    int u = q.front(); q.pop();
    for(int v : graph[u])
    {
        if(dist[u] + 1 < dist[v])
        {
            dist[v] = dist[u] + 1;
            q.push(v);
        }
    }
}
```

2 busqueda binaria

```
// [l, r] = rango donde podria estar la respuesta, incluyendo bordes

// condicion: F F F F F V V V V V

// el ultimo que es falso
int l = 0, r = n-1;
while(l != r)
{
    int mid = (l + r + 1) / 2;
    if(condicion(mid))
        r = mid-1;
    else
        l = mid;
}

// el primero que es verdadero
int l = 0, r = n-1;
while(l != r)
{
    int mid = (l + r) / 2;
    if(condicion(mid))
        r = mid;
    else
        l = mid+1;
}

// condicion: V V V V V F F F F F

// el ultimo que es verdadero
int l = 0, r = n-1;
while(l != r)
{
    int mid = (l + r + 1) / 2;
    if(condicion(mid))
        l = mid;
    else
        r = mid-1;
}

// el primero que es falso
int l = 0, r = n-1;
while(l != r)
{
    int mid = (l + r) / 2;
    if(condicion(mid))
        l = mid+1;
    else
        r = mid;
}
```

3 comandos

```
// A.cpp archivo c++
// a.out archivo ejecutable
// in archivo de entrada
// out archivo de salida


// comilar
g++ -std=c++17 A.cpp -o a.out
// ejecutar
./a.out
// en una linea
g++ -std=c++17 A.cpp -o a.out && ./a.out


// compilar con informacion para valgrind
g++ -std=c++17 -g A.cpp -o a.out
// ejecutar con valgrind
valgrind -q ./a.out
// en una linea
g++ -std=c++17 -g A.cpp -o a.out && valgrind -q ./a.out
// ejecutar con valgrind y redireccionar la entrada y salida
valgrind -q ./a.out < in > out
```

4 criba numeros primos

```
int n = 10000;
vector<bool> prime(n + 1, true);
prime[0] = prime[1] = false;
```

```
for(int i = 2; i < n; i++)
    if(prime[i])
        for(int j = 2*i; j < n; j += i)
            prime[j] = false;

// Crea un vector prime donde prime[i] es true si i es primo
// O(N log(log N))

// n/2 + n/3 + n/4 + n/5 .....
// n (1/2 + 1/3 + 1/4 + 1/5 ....)
// n log n
```

5 dfs

```
vector<int> depth; // guarda la distancia desde un nodo a la raiz
vector<int> parent; // guarda el padre de un nodo
vector<int> subtree_size; // guarda el tamao del subarbol de un nodo
    (incluyendolo a el mismo)

void dfs(int u, int p, int d)
{
    depth[u] = d;
    parent[u] = p;
    int size = 1;
    for(int v : graph[u])
    {
        if(v != p){
            dfs(v, u, d+1);
            size += subtree_size[v];
        }
    }
    subtree_size[u] = size;
}

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

    int n;
    cin >> n;
    vector<vector<int>> graph(n);
    rep(i, n-1)
    {
        int u, v; cin >> u >> v;
        u--, v--;
        graph[u].pb(v);
        graph[v].pb(u);
    }

    depth.resize(n);
    parent.resize(n);
    subtree_size.resize(n);
    int root = 0;
    dfs(root, -1, 0);

    return 0;
}
```

6 dijkstra

```
vector<ll> dist(n, 1e9);
priority_queue<pii, vector<pii>, greater<pii> > q;

ll start = 0;
dist[start] = 0;
q.push(pii(0,start));

while(!q.empty())
{
    pii p = q.top(); q.pop();
    int d = p.first, u = p.second;

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

    for(pii a : graph[u])
    {
        int v = a.first, w = a.second;
        if(dist[u] + w < dist[v])
        {
            dist[v] = dist[u] + w;
            q.push(pii(dist[v],v));
```

```
        }
    }
}
```

7 exponenciacion binaria

```
#define MOD 1000000007

// O(log b)
// calcula a^b % MOD
// el modulo se puede sacar
ll bin_exp(int a, int b)
{
    ll res = 1;
    while(b)
    {
        if(b & 1) res = (res * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return res;
}
```

8 inverso modular

```
ll extended_euclidean(ll a, ll b, ll &x, ll &y)
{
    if (a == 0){
        x = 0, y = 1;
        return b;
    }
    ll x1, y1;
    ll gcd = extended_euclidean(b % a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return gcd;
}

#define MOD 1000000007

// O(log a)
ll modular_inverse(ll a)
{
    ll x, y;
    ll g = extended_euclidean(a, MOD, x, y);
    if (g != 1)
        throw; //No solution!
    else
        return (x % MOD + MOD) % MOD;
}

// O(n)
void modular_inverse(vector<ll> &inv)
{
    inv[1] = 1;
    for(ll i = 2; i < inv.size(); i++)
        inv[i] = MOD - (MOD/i) * inv[MOD%i] % MOD;
}
```

9 kmp prefix function

```
vector<int> prefix_function(string s) {
    int n = s.size();
    vector<int> pi(n);
    repx(i, 1, n) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
// retorna un vector pi donde pi[i] es el largo del prefijo mas largo de
    s que es sufijo de s[0..i]

// pi = prefix_function("abcdabc")
// abcdabc
// pi[n-1] = 3
```

```
// ejemplo
// s  = abc#fiigjewnaabfsfdabcfsdffdsabcfsdfaba
// pi = 000000000000112000012300000001230000121

// el patron "abc" aparece en las posiciones donde pi[i] == patron.size()
// en este caso en las posiciones 12, 24, 37

int main()
{
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    string texto = "fiigjewnaabfsfdabcfsdffdsabcfsdfaba";
    string patron = "abc";

    string s = patron + "#" + texto;
    vector<int> pi = prefix_function(s);

    cout << s << endl;
    rep(i, pi.size())
        cout << pi[i] << " ";
    cout << endl;
    return 0;
}
```

10 lca

```
#include<bits/stdc++.h>

using namespace std;

vector<vector<int>> graph;

// depth[u] = la profundidad de u en el arbol (la raiz tiene profundidad
    0)
vector<int> depth;

// anc[u][i] = el ancestro 2^i de u (por ejemplo, anc[u][3] = el ancestro
    8 de u)
vector<vector<int>> anc;

void dfs(int u, int p){
    depth[u] = depth[p] + 1;     // La profundidad de u es la profundidad
        de su padre + 1
    anc[u][0] = p;               // El ancestro 2^0 de u es su padre
    for(int i = 1; i < 30; i++){ // Calculo todos los ancestros de u
        anc[u][i] = anc[anc[u][i-1]][i-1];
    }

    for(int v : graph[u]){       // Visito todos los hijos de u (los
        vecinos excepto su padre)
        if(v != p){
            dfs(v, u);
        }
    }
}

// Sube dist niveles desde u en el arbol
int lift(int u, int dist){
    for(int i = 0; i < 30; i++){
        if(dist & (1 << i)){
            u = anc[u][i];
        }
    }
    return u;
}

// Ancestro comun mas bajo de a y b
// O(log n)
int lca(int a, int b){
    if(depth[a] < depth[b]) swap(a, b); // Los ordeno de tal manera que a
        sea el mas profundo
    a = lift(a, depth[a] - depth[b]); // Subo a a la misma profundidad
        que b
    if(a == b) return a;             // Si a y b son iguales, entonces a
        es el lca

    for(int i = 29; i >= 0; i--){    // Subo a y b lo mas que pueda sin
        pasarme del lca
        if(anc[a][i] != anc[b][i]){
            a = anc[a][i];
            b = anc[b][i];
        }
    }
    return anc[a][0];                // El lca es el padre de a
}

// Distancia entre a y b
```

```cpp
// O(log n)
int dist(int a, int b){
    return depth[a] + depth[b] - 2 * depth[lca(a, b)];
}

int main() {

    ios_base::sync_with_stdio(false); cin.tie(NULL);

    int n;
    cin >> n;

    graph.resize(n);
    depth.resize(n);
    anc.resize(n, vector<int>(30));

    for(int i = 0; i < n; i++){
        int m;
        cin >> m;
        for(int j = 0; j < m; j++){
            int a;
            cin >> a;
            a--;
            graph[i].push_back(a);
            graph[a].push_back(i);
        }
    }
    dfs(0, 0);
    // Obtener el lca de a y b
    cout << lca(a, b) << "\n";
    // Obtener la distancia entre a y b
    cout << dist(a, b) << "\n";
    return 0;
}
```

11 manacher

```cpp
// odd[i]: length of longest palindrome centered at i
// even[i]: ...longest palindrome centered between i and i+1
// O(n)
void manacher(string &s,vector<int> &odd,vector<int> &even){
    string t = "$#";
    for(char c: s) t += c + string("#");
    t += "^";
    int n = t.size();
    vector<int> p(n);
    int l = 1, r = 1;
    repx(i, 1, n-1) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while(t[i - p[i]] == t[i + p[i]]) p[i]++;
        if(i + p[i] > r) l = i - p[i], r = i + p[i];
    }
    repx(i, 2, n-2) {
        if(i%2) even.push_back(p[i]-1);
        else odd.push_back(p[i]-1);
    }
}

// ejemplo:
// s    = a a a c a b a
// odd = 1 3 1 3 1 3 1
// even = 2 2 0 0 0 0

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

    string s = "aaacaba";
    vector<int> odd, even;
    manacher(s, odd, even);

    cout << "a a a c a b a" << endl;
    for(int x: odd) cout << x << " ";
    cout << endl;
    cout << " ";
    for(int x: even) cout << x << " ";
    cout << endl;
    return 0;
}
```

12 segment tree lazy

```cpp
// Segment Tree with Lazy Propagation
// Queries en rango O(log n)
// Updates en rango O(log n)
```

```
struct SegmentTreeLazy {
    int n;
    vector<ll> a, b; // a -> valores actuales, b -> valores pendientes

    // Neutro de la operacion
    //  suma -> 0
    //  multiplicacion -> 1
    //  maximo -> -inf -> -1e9
    //  minimo -> inf -> 1e9
    ll qneut() { return -2e9; }             // **** MODIFICAR ****
    // Operacion
    ll merge(ll x, ll y) { return max(x, y); } // **** MODIFICAR ****

    // Neutro del update
    //  suma -> 0
    //  multiplicacion -> 1
    ll uneut() { return 0; }                // **** MODIFICAR ****
    // Operacion del update
    void upd(int v, ll x, int l, int r){    // **** MODIFICAR ****
        // queries maximo - minimo
        a[v] += x;
        b[v] += x;
        // queries suma
        // a[v] += (r - l) * x;
        // b[v] += x;
        // queries multiplicacion
        // a[v] *= pow(x, r - l);
        // b[v] *= x;

        // setear un valor - queries de maximo - minimo
        // a[v] = x;
        // b[v] = x;
        // setear un valor - queries de suma
        // a[v] = (r - l) * x;
        // b[v] = x;
    }

    SegmentTreeLazy(int n = 0) : n(n), a(4 * n, qneut()),
        b(4 * n, uneut()) {}

    void push(int v, int vl, int vm, int vr) {
        upd(2 * v, b[v], vl, vm);
        upd(2 * v + 1, b[v], vm, vr);
        b[v] = uneut();
    }

    ll query(int l, int r, int v=1, int vl=0, int vr=1e9) {
        vr = min(vr, n);
        if (l <= vl && r >= vr) return a[v];
        if (l >= vr || r <= vl) return qneut();
        int vm = (vl + vr) / 2;
        push(v, vl, vm, vr);
        return merge(query(l, r, 2 * v, vl, vm),
            query(l, r, 2 * v + 1, vm, vr));
    }

    void update(int l, int r, ll x, int v = 1, int vl = 0,
            int vr = 1e9) {
        vr = min(vr, n);
        if (l >= vr || r <= vl || r <= l) return;
        if (l <= vl && r >= vr) upd(v, x, vl, vr);
        else {
            int vm = (vl + vr) / 2;
            push(v, vl, vm, vr);
            update(l, r, x, 2 * v, vl, vm);
            update(l, r, x, 2 * v + 1, vm, vr);
            a[v] = merge(a[2 * v], a[2 * v + 1]);
        }
    }
};



int main()
{
    int n = 100;
    // Comienza el arreglo de largo n con todos los valores neutros
    SegmentTreeLazy st(n);

    // Actualiza el rango [l, r) con el valor x
    // inclusivo - exclusivo
    st.update(5, 10, 5);

    // Setear todos los valores del arreglo
    rep(i, n){
        int x;
        cin >> x;
        st.update(i, i+1, x);
    }
```

```
    // Consulta en rango [l, r)
    int l, r;
    cin >> l >> r;
    // inclusivo - exclusivo
    cout << st.query(l, r) << endl;
    // inclusivo - inclusivo
    cout << st.query(l, r+1) << endl;
}
```

13 segment tree

```
// Segment Tree Normal
// Queries en rango O(log n)
// Updates puntuales O(log n)
struct SegmentTree {

    // Neutro de la operacion
    // suma -> 0
    // multiplicacion -> 1
    // maximo -> -inf -> -1e9
    // minimo -> inf -> 1e9
    ll neut() { return 0; }           // **** MODIFICAR ****

    // Operacion
    ll merge(ll x, ll y) { return x + y; }// **** MODIFICAR ****

    int n; vector<ll> a;
    SegmentTree(int n = 0) : n(n), a(2 * n, neut()) {}

    // Consulta en rango [l, r), incluye l pero no incluye r
    ll query(int l, int r) {
        ll x = neut(), y = neut();
        for (l += n, r += n; l < r; l /= 2, r /= 2) {
            if (l & 1) x = merge(x, a[l++]);
            if (r & 1) y = merge(a[--r], y);
        }
        return merge(x, y);
    }

    // Actualiza el valor en la posicion i, a[i] = x
    void update(int i, ll x) {
        for (a[i += n] = x; i /= 2;)
            a[i] = merge(a[2 * i], a[2 * i + 1]);
    }
};

int main()
{
    int n = 100;
    // Comienza el arreglo de largo n con todos los valores neutros
    SegmentTree st(n);

    // Actualiza el valor en la posicion i
    // a[5] = 10
    st.update(5, 10);

    // Setear todos los valores del arreglo
    rep(i, n){
        int x;
        cin >> x;
        st.update(i, x);
    }

    // Consulta en rango [l, r)
    int l, r;
    cin >> l >> r;
    // inclusivo - exclusivo
    cout << st.query(l, r) << endl;
    // inclusivo - inclusivo
    cout << st.query(l, r+1) << endl;
}
```

14 template

```
#pragma GCC optimize("Ofast")
#include<bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<ll,ll> pii;

#define rep(i, n) for (int i = 0; i < (int)n; i++)
```

```
#define repx(i, a, b) for (int i = (int)a; i < (int)b; i++)
#define eb emplace_back
#define pb push_back
#define mp make_pair
#define ff first
#define ss second

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


    return 0;
}
```

15 union find

```
vector <int> parent;
vector <int> size_set;
int cantidad_componentes = 0;

void make_set(int v){
    parent.push_back(v);
    size_set.push_back(1);
    cantidad_componentes++;
}

int find_set(int v){
    if (v == parent[v]){
        return v;
    }
    return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b){
    a = find_set(a);
    b = find_set(b);
    if (a != b){
        if (size_set[a] < size_set[b]){
            swap(a, b);
        }
        cantidad_componentes--;
        parent[b] = a;
        size_set[a] += size_set[b];
    }
}

int main()
{
    // crea n nodos
    int n = 10;
    rep(i, n){
        make_set(i);
    }

    // une los nodos
    union_sets(0, 1);
    union_sets(1, 2);
    union_sets(3, 4);

    // encuentra el representante de un nodo
    cout << find_set(0) << endl;

    // revisa si dos nodos estan unidos
    cout << (find_set(0) == find_set(3)) << endl;

    // cantidad de componentes distintas
    cout << cantidad_componentes << endl;
}
```