Team Notebook

UPSolving

May 12, 2023

Contents

1 BinaryLifting

```
namespace LCA{
    const int maxPow = 30;
    int lifting[maxn][maxPow];
    int in[maxn];
    int out[maxn];
    int timeTransversal=0;

    void clear(){ // CALL THIS
        timeTransversal=0;
    }

    void build(int x, int p, vector<vll>& vadj){
        lifting[x][0] = p;
        in[x]=timeTransversal++;
        for(int i=0; i<maxPow-1; i++){
            lifting[x][i+1] = lifting[lifting[x][i]][i];
        }
        for(int y: vadj[x]){
            if(y == p) continue;
            build(y,x,vadj);
        }
        out[x]=timeTransversal++;
    }

    int getKthParent(int x, int k){ // Only if you want to know this
        if(k == 0) return x;
        int nextBit = (k&(-k));
        return getKthParent(lifting[x][__builtin_ctz(k)],k-nextBit);
    }

    bool isAncestor(int x, int y){ // ancestor x of y
        return (in[x] <= in[y] && out[x] >= out[y]);
    }

    int lca(int x, int y){
        if(isAncestor(x,y)) return x;
        if(isAncestor(y,x)) return y;
        //Moving x;
        for(int pow2 = maxPow-1; pow2>=0; pow2--){
            if(!isAncestor(lifting[x][pow2],y)){
                x = lifting[x][pow2];
            }
        }
        return lifting[x][0];
    }
}
```

2 Congruencias

```
using rp = pair<ll,pll>;

rp mygcd(ll a, ll b){
    if(a==0) return {b,{0,1}};
    if(b==0) return {a,{1,0}};
    rp prev=mygcd(b%a,a);

    ll bd = b/prev.first;
    ll ax=prev.second.second-prev.second.first*(b/a);
    ax = (ax%bd+ bd)%bd;
    ll bx=(prev.first-a*ax)/b;

    rp ans={prev.first, {ax,
        bx}};
    return ans;
}

rp solve(ll c1, ll c2, ll b){
    rp ans = mygcd(c1,c2);
    if(b%ans.first != 0) return {-1,{-1,-1}};
    ll bd = b/ans.first;
    ll c2p = c2/ans.first;

    ans.second.first *= bd;
    ans.second.first = (ans.second.first%(c2p)+c2p)%c2p;

    ans.second.second = (b-ans.second.first*c1)/c2;
    return ans;
}

// Falta checar que el lcm no cause overflow
pll sistema(vll& c, vll& b){
    pll ans(b[0], c[0]);
    for(size_t i=1; i<c.size(); i++){
        ll pc = ans.second;
        ll pb = ans.first;
        rp sol = solve(pc, c[i],b[i]-pb);
        if(sol.first == -1) return {-1,-1};

        ans.second = pc*c[i]/sol.first;
        ans.first += pc*sol.second.first;
        ans.first %= ans.second;
    }
    return ans;
}
```

3 Dinics

```
const int maxn = 400;

namespace Dinics{
    map<ll,ll> vadj[maxn];
    vpll layered[maxn];
    vpll layeredSum[maxn];
    int level[maxn];
    int index[maxn];
    const long long maxFlow = 1e16;

    void clear(int N){
        for(int i=0; i<N; i++) vadj[i].clear();
    }

    void buildLayered(int N, int S){
        for(int i=0; i<N; i++) level[i] = -1;
        queue<int> q;
        queue<int> q2; //0-1 BFS
        int step = 1;
        level[S] = 0;
        q.push(S);
        while(1){
            while(!q.empty()){
                int x = q.front();
                q.pop();
                for(pll yy: vadj[x]){
                    int y = yy.first;
                    if(level[y] != -1) continue;
                    if(yy.second <= 0) continue;
                    level[y] = step;
                    q2.push(y);
                }
            }
            if(q2.empty()) break;
            step++;
            while(!q2.empty()){
                q.push(q2.front());
                q2.pop();
            }
        }

        for(int i=0; i<N; i++){
            layered[i].clear();
            layeredSum[i].clear();
            for(pll yy:vadj[i]){
                if(level[i]+1 != level[yy.first]) continue;
                layered[i].push_back(yy);
                layeredSum[i].push_back(yy);
        }
    }
}

ll blockFlow(int x, ll flow, int T){
    if(flow == 0) return flow;
    if(x == T) return flow;
    if(index[x] >= (ll)layered[x].size()) return 0;
    for(size_t i=index[x]; i<layered[x].size(); index[x
         ]++,i=index[x]){
        ll nextFlow = min(flow, layered[x][i].second);
        ll attempt = blockFlow(layered[x][i].first,
             nextFlow, T);
        if(attempt!=0){
            layered[x][i].second-=attempt;
            if(layered[x][i].second == 0) index[x]++;
            return attempt;
        }
    }
    return 0;
}

ll blockPaths(ll N,ll S,ll T){
    for(int i=0; i<N; i++) index[i]=0;
    ll ans = 0;
    while(1){
        ll flow = blockFlow(S,maxFlow, T);
        ans+=flow;
        if(flow == 0) return ans;
    }
}

ll dinics(ll N, ll S, ll T){
    ll ans = 0;
    while(1){
        buildLayered(N,S);
        ll push = blockPaths(N,S,T);
        ans+=push;
        if(push == 0) return ans;
        // actualizar cambios en residual
        for(int i=0; i<N; i++){
            for(size_t j=0; j<layered[i].size(); j++){
                vadj[i][layered[i][j].first] = layered[i][
                     j].second;
                vadj[layered[i][j].first][i] += layeredSum
                     [i][j].second-layered[i][j].second;
            }
        }
    }
}
```

```
}
```

4 GCD

```
using rp = pair<ll,pll>;

rp mygcd(ll a, ll b){
    if(a==0) return {b,{0,1}};
    if(b==0) return {a,{1,0}};
    rp prev=mygcd(b%a,a);

    ll bd = b/prev.first;
    ll ax=prev.second.second-prev.second.first*(b/a);
    ax = (ax%bd+ bd)%bd;
    ll bx=(prev.first-a*ax)/b;

    rp ans={prev.first, {ax,
        bx}};
    return ans;
}
// sea c = max(a,b), la solucion |x| <= c

// Soluciones de ax + by = 0.
// d = gcd(a,b); -> x=(b/d)*t, y=-1*(a/d)*t;
```

5 KMP

```
vector<int> kmp(string& s){
    int n = s.size();
    vector<int> vs(n);
    //vs[i] = kmp que acaba en la posicion i
    //en otras palabras, tiene tamanio i+1;
    for(int i=1; i<n; i++){
        int j = vs[i-1]; // j = aproximacion anterior
        while(j!=0 && s[i] != s[j]){
            j = vs[j-1];
        }
        if(s[i] == s[j]) j++;
        vs[i] = j;
    }
    return vs;
}
```

6 OrderStatisticsTree

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

using namespace __gnu_pbds;

using map_t = tree<ll,null_type,
      less<ll>, rb_tree_tag,
      tree_order_statistics_node_update>;

// No inserta duplicados.
// tiene mismas funciones que set o map (dependiendo de que
     uses);

int main(){
    map_t mp;
    mp.insert(3);
    mp.insert(5);
    mp.order_of_key(4); // devolveria 1
    mp.find_by_order(2); // devolveria el iterador al 5 (es
         un puntero);
}
```

7 SCC

```
const int maxn = // add maxn

namespace Tarjan{
    int scc=0; // Number of resulting scc;
    bool inStack[maxn];
    int lo[maxn];
    int hi[maxn];
    int vis[maxn];
    stack<int> st;
    int step=0;
    int sz[maxn]; //Size of each component. Components are 0
         indexed
    vector<vll> comps; // Each component
    int myComp[maxn]; // Maps each node to each component
    unordered_set<ll> compVadj[maxn]; // New graph.
         Components are 0 indexed

    void clear(int N){
        scc=0;
        step = 0;
        for(int i=0; i<N; i++){
            vis[i] =0;
            inStack[i] =0;
            sz[i] = 0;
            myComp[i] =0;
            compVadj[i].clear();
        }
        comps.clear();
    }

    int tarjan(int x, vector<vll>& vadj){
        if(vis[x]) return maxn+2;
        vis[x] = 1;
        inStack[x] = 1;
        st.push(x);
        lo[x] = hi[x] = step++;
        for(int y : vadj[x]){
            if(inStack[y]){
                lo[x] = min(lo[x], hi[y]);
                continue;
            }
            lo[x] = min(lo[x], tarjan(y, vadj));
        }

        if(lo[x] == hi[x]){
            int currSz = 0;
            vll thisComp;
            while(st.top() != x){
                thisComp.push_back(st.top());
                inStack[st.top()] = 0;
                myComp[st.top()] = scc;
                int y = st.top();
                for(int z : vadj[y]){
                    if(inStack[z]) continue;
                    z = myComp[z];
                    if(z == scc) continue;
                    compVadj[scc].insert(z);
                }
                currSz++;
                st.pop();
            }
            thisComp.push_back(x);
            inStack[st.top()] = 0;
            myComp[st.top()] = scc;
            int y = st.top();
            for(int z : vadj[y]){
                if(inStack[z]) continue;
                z = myComp[z];
                if(z == scc) continue;
                compVadj[scc].insert(z);
            }
            st.pop();
            currSz++;
            comps.push_back(thisComp);
            sz[scc] = currSz;
            scc++;
        }
        return lo[x];
    }

    void tarjanMain(int N, vector<vll>& vadj){ // Use this.
        clear(N); // Modify this if you use a range [1,N]
        for(int i=0; i<N; i++){ //Modify this if your nodes start from 1
            if(vis[i] == 0) tarjan(i,vadj);
        }
    }
}
```

8 SegmentTree

```
struct segmentTree{
    //Define maxn
    static const int maxn = 1e5+1;
    //Define Segment Tree container
    struct str{
        ll first;
        ll second;
        // important to define default initialization
        str(){
            first = 0;
             second = 0;
        }
        // useful to reduce code
        str(ll aa, ll bb){
             first= aa;
             second= bb;
        }
    };
    // Define Merge Function.
    // It can be associative or not.
    // aka it is fine that a + b != b + a
    str func(str a, str b){
        return (a.first < b.first ? a : b);
    }
    str st[maxn];
    ll n;
    // Recieves a 0-indexed array
    void build(vll& vl, int sizn){
        n = sizn;
        for(int i=0;i<n;i++){
            st[i+n].first = vl[i];
            st[i+n].second = i;
        }
        for(int i=n-1;i>0;i--){
            st[i]=func(st[i*2],st[i*2+1]);
        }
    }

    // Position is 0-indexed
    // Value is replaced
    void update(int pos, str x){
        for(pos+=n,st[pos]=x,pos/=2; pos; pos/=2){
            if(pos*2+1 >= 2*n) st[pos] = st[pos*2];
            else st[pos] = func(st[pos*2],st[pos*2+1]);
        }
    }

    // Query in the interval from [L,R)
    // Def is the default, neutral value;
    str query(int l, int r, str def){
        str ansl=def;
        str ansr=def;
        for(l+=n,r+=n;l<r;l/=2,r/=2){
            if(l%2) ansl=func(ansl,st[l++]);
            if(r%2) ansr=func(st[--r],ansr);
        }
        return func(ansl,ansr);
    }
};
```

9 SegmentTreeLazy

```
// Lazy Segment Tree.
// 1.- This segment tree doesn't know the range of the interval
// 2.- The order opperations must'nt matter (assigment doesn't work)
const int maxn;
struct segmentTree{
    struct st[2*maxn];
    ll n;
    ll h;
    struct d[2*maxn];
    struct merge(struct& a, struct& b){
        // define merge function
        // return struct c;
    }
    struct ifPropagated(ll idx){
       // return st[idx] + operationOf(d[idx]);
    }
    void build(ll sz){
        n = sz;
        // calcularel numero de bits de n;
        h = 64-__builtin_clzll(n);
        for(int i=0; i<n; i++){
            //st[i+n] = val; assignar los valores iniciales
        }
        for(int i=n-1; i>0; i--){
            st[i]=merge(st[i*2],st[i*2+1]);
        }
    }

    void apply(ll i, ll x){
        //apply update of value X, to segment i;
        //remember to update d[i] aka, values not propagated
             to children
    }

    void bi(ll x){
        //update all parents of x;
        for(x/=2; x; x/=2){
            st[x] = merge(st[x*2], st[x*2+1]);
            // IMPORTANTE. El flag d[x] puede no haberse
                 propagado;
            // Cual seria el valor de st[x], si d[x] se
                 hubiera propagado?
            // Ejemplo si nuestra operacion es la suma, st[x]
                  += range*d[x]
            st[x] = ifPropagated(x);
        }
    }

    // [l,r); 0 based indexed
    void update(ll l, ll r, ll x){
        ll l0 = l+n;
        ll r0 = r+n-1;
        for(l+=n,r+=n; l<r; l/=2, r/=2){
            if(l%2) apply(l++, x);
            if(r%2) apply(--r, x);
        }
        bi(l0);
        bi(r0);
    }

    // propagate lazy before doing queries
    void push(ll x){
        for(int s=h; s>0; s--){
            int i = x>>s;
            if(d[i] != 0){
                apply(i*2, d[i]);
                apply(i*2+1, d[i]);
            }
            d[i] = 0;
        }
    }

    // Query from [l,r), 0-indexed;
    struct query(ll l, ll r){
        push(l+n);
        push(r+n-1);
        bitset<maxb> ansl;
        bitset<maxb> ansr;
        for(l+=n,r+=n; l<r; l/=2,r/=2){
            if(l%2) ansl= merge(ansl, st[l++]);
            if(r%2) ansr = merge(st[--r], ansr);
        }
        return merge(ansl,ansr);
    }
};
```

10 SparseTableLCA

```
const int maxn=1e5+7;

namespace SparseLCA{
    const int maxPow = 30;
    ll f1[maxn];
    ll depth[2*maxn];
    ll depthAns[2*maxn];
    ll cc=0;
    ll dpVal[2*maxn][maxPow];
    ll dpAns[2*maxn][maxPow];

    void build(int x, int p, int d, vll vadj[maxn]){
        f1[x]=cc;
        depth[cc]=d;
        depthAns[cc]=x;
        cc++;
        for(ll y: vadj[x]){
            if(y==p) continue;
            build(y,x,d+1,vadj);
            depth[cc]=d;
            depthAns[cc]=x;
            cc++;
        }
    }

    void buildPow(ll N){
        for(int i=0; i<(2*N-1); i++){
            dpAns[i][0]=depthAns[i];
            dpVal[i][0]=depth[i];
        }
        for(int j=1; j<maxPow; j++){
            for(int i=0; i<(2*N-1); i++){
                dpAns[i][j]=dpAns[i][j-1];
                dpVal[i][j]=dpVal[i][j-1];
                int nex = i+(1<<(j-1));
                if(nex >= 2*N-1) continue;
                if(dpVal[nex][j-1] < dpVal[i][j]){
                    dpVal[i][j]=dpVal[nex][j-1];
                    dpAns[i][j]=dpAns[nex][j-1];
                }
            }
        }
    }

    ll query(ll a, ll b){
        ll pa = f1[a];
        ll pb = f1[b];
        ll fa = min(pa,pb); ll fb = max(pa,pb);
        ll dis = fb-fa+1;
        ll pp = 63-__builtin_clzll(dis);
        ll nex = fb-(1<<pp)+1;
        ll ans = dpAns[fa][pp];

        if(dpVal[nex][pp] < dpVal[fa][pp]) return dpAns[nex][
             pp];
        return ans;
    }

    void preprocess(ll x, ll N, vll vadj[maxn]){
        cc=0;
        build(x,x,0,vadj);
        buildPow(N);
    }
}
```

11 SuffixTree

```
const int maxn=100000;
```

```
namespace SuffixTree{
    // Nodes go from [1 to nodeCount]
    map<int,int> to[2*maxn];
    ll p[2*maxn];
    ll len[2*maxn];
    ll idx[2*maxn];
    char s[maxn];
    ll link[2*maxn];
    ll n=0; ll j=0; ll pos=0;
    ll node=1;
    const int root=1;
    ll nodeCount=1;

    void subo(){
        while(node != root && link[node] == 0){
            pos -= len[node];
            node = p[node];
        }
    }

    void bajo(){
        while(pos < n-1 && to[node][s[pos]] != 0 && len[to[
             node][s[pos]]] != -1 && len[to[node][s[pos]]] <=
              n-pos-1){
            ll tempPos = pos;
            pos+=len[to[node][s[pos]]];
            node = to[node][s[tempPos]];
        }
    }

    void add(char c){
        s[n++] = c;
        bool next = true;
        int prev = 0;

        while(next && j!=n){
            if(node == root) pos=j;
            bajo();
            if(to[node][s[pos]] == 0){
                nodeCount++;
                to[node][s[pos]] = nodeCount;
                p[nodeCount] = node;
                len[nodeCount] = -1;
                idx[nodeCount] = n-1;
                next=true;
                link[prev]=node;
                prev=0;
                j++;
            }else{
                char t = s[idx[to[node][s[pos]]] + n-1-pos];
                if(s[n-1] == t){
                    next=false;
                    link[prev]=node;
                }else{
                    nodeCount++;
                    ll child = to[node][s[pos]];
                    p[child] = nodeCount;
                    if(len[child] != -1) len[child] -= n-1-pos
                        ;
                    idx[child] += n-1-pos;
                    idx[nodeCount] = pos;
                    len[nodeCount] = n-1-pos;
                    p[nodeCount]=node;
                    to[node][s[pos]] = nodeCount;

                    nodeCount++;
                    len[nodeCount] = -1;
                    idx[nodeCount] = n-1;
                    p[nodeCount] = nodeCount-1;
                    to[nodeCount-1][s[n-1]] = nodeCount;
                    to[nodeCount-1][t] = child;
                    link[prev]=nodeCount-1;
                    prev = nodeCount-1;
                    next = true;
                    j++;
                }
            }

            if(next && j!=n){
                subo();
                if(node == root) pos = j;
                else node = link[node];
            }
        }
    }

    void setLen(){
        for(int i=1; i<=nodeCount; i++){
            if(len[i] == -1) len[i] = n-idx[i];
        }
    }
}
```