Pancakes II - Editorial

Timothy Gao

We are given a tree with N ($1 \leq N \leq 2 \cdot 10^5$) nodes, each with a bit value of 0 or 1. Answer Q ($1 \leq N \leq 2 \cdot 10^5$) queries, each of which asks for the minimum required bit flips to make bits of all nodes on the shortest path from u to v zero. In one bit flip, we can choose any two nodes and flip the bits of all nodes on the shortest path between them.

1 Initial Observations

Consider each query independently. As with many path query problems, let's starting by reducing the shortest path from u to v to an array, arbitrarily starting from u or v. Any bit flip will affect a subsegment of the array, and all subsegments can be flipped through selecting some pair of nodes (elements in the array). A subsegment is defined a contiguous connected series of nodes that lie on the path from u to v. Define a "chunk" of 1s as a subsegment of the array that contains only 1s, and is not adjacent to any 1s. The lower bound for the number of flips is the number of chunks in the array, because any flip can remove at most one chunk in the array (note that a flip might create new chunks by flipping 0s to 1s).

2 Arriving at the Solution

With this observation, the problem is reduced to finding the number of chunks of 1s on the path from u to v for each query.

One way to tackle this is through Centroid Decomposition. For each node i in the CD-Tree, we compute $dp[i][j]$ for each node j below it (j is in the component of i as we're running the DFS), which is the number of chunks of 1s on the path from i to j. This runs in $\mathcal{O}(size)$, where $size$ is the number of nodes below i, through a DFS with careful bookkeeping. Due to the structure of the CD-tree, this results in a total time complexity of $\mathcal{O}(Nlog(N))$. The proof is similar to that of the divide and conquer algorithm (CD is essentially DnQ on a tree after all); Each node has at most $log(N)$ nodes above it. To find the answer for each query with nodes a and b, we find their ancestor c in the CD-tree and do case work on $dp[c][a]$ and $dp[c][b]$, depending on the value of c. In my code, I computed an additional dp_{half} which is the same as dp but excluding the ancestor node in order simplify the casework. Finding the ancestor takes $log(N)$ for each query, and calculating the DP values take $Nlog(N)$ time, resulting to a total time complexity of $\mathcal{O}(Nlog(N) + Qlog(N))$.

Heavy-Light Decomposition is another approach we could use to tackle this problem. In the segment tree, we would store (l, r, ans) for each node i, where l and r are the border values and ans is the number of chunks of the segment represented by node i. Some participants chose this approach, although it is more implementation heavy.

A third approach is to use binary lifting, which works the same as the CD solution but we need to do additional casework when we combine the two 2^{i-1} segments.

All three approaches yield $\mathcal{O}(Nlog(N) + Qlog(N))$, which is fast enough. My code implements the Centroid Decomposition approach. Originally, we intended this problem to be a subroutine of a different problem, finding the minimum bit flips to create a non-decreasing path from u to v, but decided against it so the problem wouldn't be too implementation-heavy.

3 Code

```
//@timothyg
```

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

using namespace std;

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

#define pb push_back

#define f first
#define s second

const int maxn = 2e5 + 3;

vector<int> adj[maxn];
vector<int> full[maxn], half[maxn];
//dp_full and dp_half

int csz[maxn], blocked[maxn], par[maxn], dep[maxn], bit[maxn];
//csz, blocked are subroutines of CD
//dep and par are used to find CD tree ancestor between two nodes a and b
//bit stores the given bit values of each node

//DFS computes the DP Values in O(size)
//mode 0 - full, mode 1 - half
void calc(int u, int mode, int last = 0, int lastval = 0, int p = 0){
    int val = lastval;
    if(last == 0 && bit[u]) val++;

    if(mode) {
        half[u].pb(val);
    }else{
        full[u].pb(val);
    }

    for(int n : adj[u]){
        if(!blocked[n] && n != p) {
            calc(n, mode, bit[u], val, u);
        }
    }
}

//finds the size of current group/component, needed for CD
int initsz(int u, int p = 0){
    csz[u] = 1;
    for(int n : adj[u]){
        if(!blocked[n] && n != p) csz[u] += initsz(n, u);
    }
    return csz[u];
}

//intialize the CD tree
int init(int u, int compsz, int p = 0, int d = 0){
    for(int n : adj[u]){
        if(n != p && !blocked[n] && csz[n] > compsz/2){
            return init(n, compsz, u, d);
        }
    }

    blocked[u] = 1;
    dep[u] = d;
    calc(u, 0);
    for(int n : adj[u]){
        if(!blocked[n]){
            calc(n, 1, bit[u]);
        }
    }

    for(int n : adj[u]){
        if(!blocked[n]){
            initsz(n);
            int temp = init(n, csz[n], u, d+1);
            par[temp] = u;
        }
    }
    blocked[u] = 0;
    return u;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(0);
    int N; cin >> N;
    string str; cin >> str;
    for(int i = 1; i<=N; i++) bit[i] = str[i-1] == '1';
    for(int i = 0; i<N-1; i++){
        int u, v; cin >> u >> v;
        adj[u].pb(v), adj[v].pb(u);
    }

    //Intialize CD tree and compute DP values
    initsz(1);
    init(1, csz[1]);

    int Q; cin >> Q;
    while(Q--){
        int u, v; cin >> u >> v;
        //Find ancestor in CD Tree
        int pu = u, pv = v;
        while(dep[pu] > dep[pv]) pu = par[pu];
        while(dep[pv] > dep[pu]) pv = par[pv];
        while(pu != pv) pu = par[pu], pv = par[pv];

        //Casework (not too bad thanks to dp_half)
        int ret = full[u][dep[pu]];
        if(dep[pv] < half[v].size()){
            ret += half[v][dep[pv]];
        }
        cout << ret << '\n';
    }
    return 0;
}
```