Chuyên đề: Phát triển tư duy của học sinh thông qua việc tối ưu hóa các bài toán trên đồ thị

MỤC LỤC

A. Mở đầu 2

1. Lý do chọn đề tài 2

2. Mục đích chọn đề tài 2

B. Nội dung 2

I. Lý thuyết 2

II. Bài tập vận dụng 3

1. Chiều cao của một cây 3

Cách chưa tối ưu: 4

Cách tối ưu: 5

Đoạn code tham khảo: 6

Bộ test: 7

Bài 2. Con đường có chiều dài bằng k 8

Cách chưa tối ưu: 8

Cách tối ưu: 10

Đoạn code tham khảo: 11

Bộ test 14

Bài 3. Virus corona 14

Cách chưa tối ưu: 16

Cách tối ưu: 20

Code 20

Bộ test 22

Bài 4. Cây con trắng đen lớn nhất 22

Đoạn code tham khảo 25

Bộ test: 27

Bài 5. Tổ tiên chung gần nhất 27

Cách chưa tối ưu: 28
Cách tối ưu 31
Đoạn code tham khảo 31
C. Kết luận 34

A. Mở đầu

1. Lý do chọn đề tài

Lý thuyết đồ thị có rất nhiều thuật toán ứng dụng vào các bài toán thực tế của cuộc sống như tìm đường đi ngắn nhất, tìm cây khung có trọng số nhỏ nhất,…Việc tìm ra mối liên hệ giữa các bài toán thực tế trong cuộc sống với các thuật toán trên đồ thị, khiến cho học sinh cảm thấy hứng thú rất lớn với lý thuyết về đồ thị. Tuy nhiên, có một khó khăn khi dạy học về lý thuyết đồ thị, đó là số lượng thuật toán liên quan đến đồ thị không hề nhỏ. Do vậy, học sinh khi làm các bài tập liên quan đến đồ thị, các em thường có nhiều thuật toán khác nhau để giải quyết vấn đề. Đôi khi, các em cảm thấy chưa biết phải vận dụng thuật toán nào để giải quyết các bài toán trên đồ thị. Do đó, tôi muốn tiếp cận cách giải quyết những bài toán trên đồ thị, dựa trên việc phát triển từ những thuật toán, phương pháp cơ bản các em cảm thấy quen thuộc như quy hoạch động, số học,…

2. Mục đích chọn đề tài

Đề tài tôi xây dựng hướng tới mục đích sau:

- Xây dựng và phân tích được những bài tập đồ thị có nhiều cách giải, sử dụng những thuật toán cơ bản trên đồ thị như dfs, bfs, tìm đường đi ngắn nhất,…. sau đó tối ưu hóa chúng bằng các phương pháp cơ bản của lớp 10 như số học, quy hoạch động, tổ hợp,…
- Hệ thống hóa các bài tập cơ bản trên đồ thị, nhằm giúp cho việc phát triển tư duy của học sinh theo mạch logic.

B. Nội dung

I. Lý thuyết

1. Các lý thuyết về quy hoạch động, tổ hợp, số học,……

2. Các thuật toán cơ bản trên đồ thị như dfs, bfs,……

II. Bài tập vận dụng

1. Chiều cao của một cây

Cho một cây gồm có N đỉnh được đánh số từ 1 tới N và N-1 cạnh. Chiều cao của một cây là số cạnh trên đường đi dài nhất từ nút gốc tới nút lá của cây. Yêu cầu: Nếu mỗi nút được coi là một nút gốc, đưa ra chiều cao của cây khi đó.

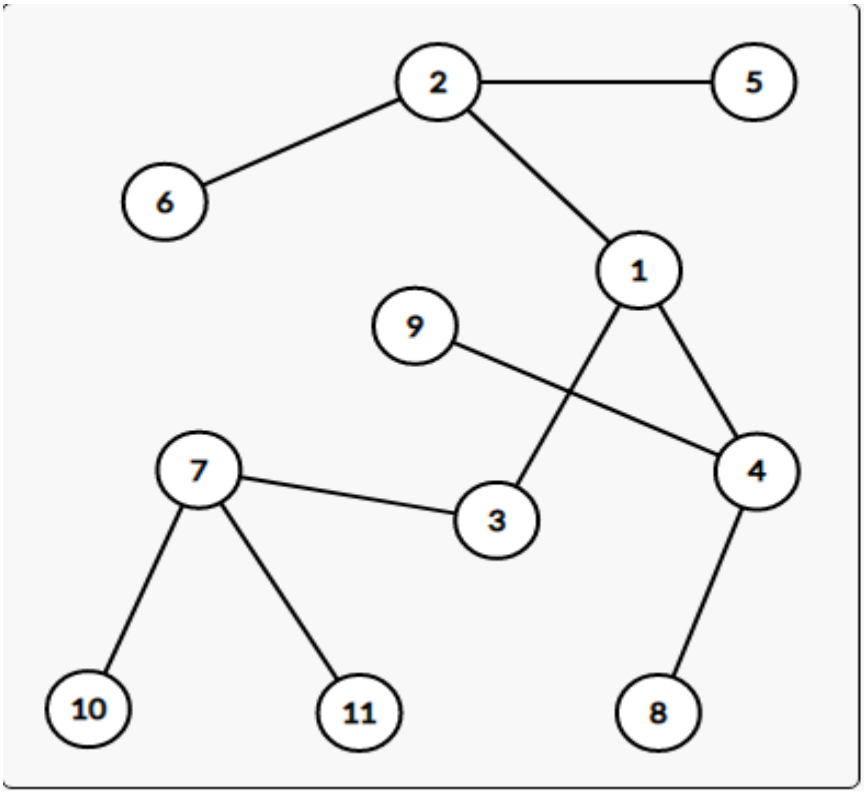
Input: Dòng đầu tiên gồm số nguyên N (2<=N<=100000). N-1 dòng tiếp theo mỗi dòng gồm hai số nguyên a và b (1<=a,b<=N) mô tả cạnh vô hướng nối giữa a và b

Giới hạn: 2<=N<=100000

Output: Đưa ra N dòng, dòng thứ i là chiều cao của cây nếu coi nút thứ i là nút gốc

Input	Output
11 2 1 3 1 4 1 5 2 6 2 7 3 10 7 11 7 8 4 9 4	3 4 3 4 5 5 4 5 5 5 5

Hình vẽ minh họa:



Chiều cao của cây là một trong những khái niệm cơ bản khi dạy học về cây. Sau khi dạy xong lý thuyết về cây, giáo viên có thể áp dụng ngay bài tập này để dạy học cho học sinh

Cách chưa tối ưu:

Theo đề bài, tìm chiều cao của cây nếu quan niệm nút thứ I là nút gốc, mà có N nút. Do đó, đa số học sinh sẽ nghĩ ngay đến việc sử dụng thuật toán dfs, bắt đầu từ nút gốc là nút thứ i, với i từ 1 tới N, ứng với mỗi nút liền kề với nút thứ i ta sẽ tăng chiều cao của cây lên một đơn vị, sau đó tiếp tục đệ quy bắt đầu từ nút liền kề với nút thứ i. Tuy nhiên, có thể có nhiều nút liền kề với nút thứ i, do đó ta phải tìm max trong số chiều cao xuất phát từ những nút liền kề với nút thứ i.

Khi đó, các em phải sử dụng công thức

H(i)=1 + max (h(j)) trong đó h(i), h(j) là chiều cao của cây xuất phát từ nút thứ i, j trong đó j là nút liền kề với nút thứ i

Sau đây là code của cách chưa tối ưu:

```
#include <bits/stdc++.h>
using namespace std;
int n,x,y,d,m;
vector<vector<int> >a;
int kt[100001];
void dfs(int t,int d)
```

```
{
        kt[t]=1;
        m=max(d,m);
        for (int i=0;i<a[t].size();i++)
            if(kt[a[t][i]]==0)
                dfs(a[t][i],d+1);

}
int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin>>n;
    a.resize(n+5);
    for(int i=1;i<n;i++)
        {
            cin>>x>>y;
            a[x].push_back(y);
            a[y].push_back(x);
        }
    for(int i=1;i<=n;i++)
    {
        dfs(i,0);
        memset(kt,0,sizeof(kt));
        cout<<m<<'\n';
        m=0;
    }
    return 0;
}
```

Phân tích độ phức tạp của thuật toán: Tuy nhiên, thuật toán dfs xuất phát từ nút thứ i có độ phức tạp là O(n+m) trong đó n là số đỉnh, m là số cạnh. Do sử dụng n nút, nên độ phức tạp của thuật toán là $O(n^2+m)$, trong khi $n=10^5$, số lượng phép tính lên tới 10^{10} phép tính khiến cho thuật toán chưa tối ưu.

Vậy nguyên nhân khiến thuật toán này chưa tối ưu?

Nguyên nhân chính khiến thuật toán này chưa tối ưu, đó là chiều cao xuất phát từ một nút của cây có thể bị tính lại nhiều lần. Do đó, ta sẽ tìm cách để chiều cao xuất phát từ một nút của cây chỉ tính một lần mà thôi. Khi đó, ta nghĩ đến phương pháp quy hoạch động.

Cách tối ưu:

Ta gọi in[i] là chiều cao lớn nhất của cây khi xuất phát từ nút i, đi xuống những cây con của chúng cho tới lá. Out[i] là chiều cao lớn nhất của cây khi xuất phát từ nút thứ I, đi lên thông qua cha của chúng. Chiều cao của cây khi xuất phát từ nút thứ i sẽ bằng max (in[i], out[i]). Trong đó In[i]=max(in[i],in[child]+1), trong đó child là nút con của nút thứ i. Để tính được out[i], xuất phát từ nút thứ I, ta sẽ di chuyển lên thông qua cha của nút thứ i gọi là cha1. Từ cha1 của nút thứ i, có 2 cách để đi, một là đi theo các cây con của nút cha1, hai là đi theo cha2 là cha của nút cha1. Chiều cao lớn nhất thông qua nút cha2 là out[cha1]. Tổng quát hóa: out[i]=1+max(out[cha của i]+ đường đi dài nhất của tất cả các cây con của cha của i)

Đoạn code tham khảo:

```
#include <bits/stdc++.h>
using namespace std;
const int MAX_NODES = 100;
int in[MAX_NODES];
int out[MAX_NODES];
void dfs1(vector<int> v[], int u, int parent)
{
    // initially every node has 0 height
    in[u] = 0;

    // traverse in the subtree of u
    for (int child : v[u]) {

        // if child is same as parent
        if (child == parent)
            continue;

        // dfs called
        dfs1(v, child, u);

        // recursively calculate the max height
        in[u] = max(in[u], 1 + in[child]);
    }
}

// function to pre-calculate the array ouut[]
// which stores the maximum height when traveled
// via parent
void dfs2(vector<int> v[], int u, int parent)
{
    // stores the longest and second
```

```
    // longest branches
    int mx1 = -1, mx2 = -1;

    // traverse in the subtress of u
    for (int child : v[u]) {
        if (child == parent)
            continue;

        // compare and store the longest
        // and second longest
        if (in[child] >= mx1) {
            mx2 = mx1;
            mx1 = in[child];
        }

        else if (in[child] > mx2)
            mx2 = in[child];
    }

    // traverse in the subtree of u
    for (int child : v[u]) {
        if (child == parent)
            continue;

        int longest = mx1;

        // if longest branch has the node, then
        // consider the second longest branch
        if (mx1 == in[child])
            longest = mx2;

        // recursively calculate out[i]
        out[child] = 1 + max(out[u], 1 + longest);

        // dfs function call
        dfs2(v, child, u);
    }
}
```

Bộ test:
https://drive.google.com/drive/folders/1PMPJRgyvIcNV5KlBo70kDzcLJIrnHQXs?usp=sharing

Bài 2. Con đường có chiều dài bằng k

Cho N địa điểm ($2 \le N \le 10^5$) kết nối với nhau bởi N-1 con đường hai chiều, sao cho từ một địa điểm có thể tới bất kỳ địa điểm nào khác. Nói cách khác, các địa điểm này hình thành nên một cây. Yêu cầu đặt ra là chia cây đã cho thành nhiều cây con, sao cho đường đi trong mỗi cây con này có chiều dài bằng nhau. Cụ thể, cho $1 \le K \le N-1$, liệu có tồn tại cách phân chia tập các con đường hai chiều, thành nhiều tập con các con đường hai chiều, sao cho chúng tạo thành đường đi có chiều dài bằng K, đưa ra 1 nếu tồn tại, đưa ra 0 nếu không tồn tại.

Input: Dòng đầu tiên gồm số nguyên N. N-1 dòng tiếp theo gồm hai số nguyên a và b mô tả một cạnh nối giữa hai đỉnh a và b, a và b nằm trong đoạn [1..N]

Output: Đưa ra dãy bit có chiều dài N-1. Với mỗi $1 \le K \le N-1$, tính từ trái qua phải, bit thứ K sẽ bằng 1 nếu có thể phân chia, bằng 0 nếu không thể phân chia

Input	Output
13 1 2 2 3 2 4 4 5 2 6 6 7 6 8 8 9 9 10 8 11 11 12 12 13	111000000000

Giải thích test: Ta có thể phân chia cây thành đường đi có chiều dài K, với K=1,2,3. Với K=3, ta có thể phân chia như sau: 13−12−11−8,10−9−8−6,7−6−2−3,5−4−2−1

Cách chưa tối ưu:

Để phân chia cây thành tập các đường đi có chiều dài K, rõ ràng số lượng con đường phải là bội số của K. Nếu số lượng các con đường không phải là bội số của K, thì câu trả lời rõ ràng là 0. Nếu số lượng các con đường là bội số của K, ta sẽ tìm cách tính số lượng các nút trong mỗi cây con xuất phát từ nút x, từ đó ta suy ra được số lượng các con đường trong từng cây con xuất phát từ nút x. Sau đó, ta tìm cách ghép những cây con có chiều dài a với cây con có chiều dài b, để tổng a+b chia hết cho K (a, b là phần dư trong phép

chia chiều dài cho K). Nếu việc ghép dẫn tới thừa ra những cây con có chiều dài không chia hết cho K, câu trả lời là 0, nếu ta có thể ghép theo cặp, câu trả lời là 1

Đoạn code tham khảo:

```
#include<bits/stdc++.h>
#define gcd(a,b) __gcd(a,b)
#define x first
#define y second
#define LL long long
#define ii pair<LL,LL>
#define MP make_pair
#define MT make_tuple
#define fort(i,a,b) for(LL i=a;i<=b;i++)
#define forn(i,a,b) for(LL i=a;i>=b;i--)
#define rep(i,a,b) for(LL i=a;i<b;i++)
#define MT make_tuple
#define pb push_back
using namespace std;
const LL oo=int(1e9);
typedef tuple<double,LL,LL,LL> tii;
typedef pair<LL,LL> pii;
typedef pair<LL,pii> iii;
const LL N=int(1e5)+5;
vector<vector<LL> > a;
LL n;
bool k[N];
LL dfs(LL x,LL c,LL val)
{
   vector<long> luu;
   for(LL i:a[x])
      if(i!=c)
   {
      LL tt=dfs(i,x,val);
      if (tt==-1) return -1;
      tt++;
      if (tt%val!=0) luu.pb(tt%val);
   }
   if (luu.size())
   {
      luu.pb(0);
      sort(luu.begin(),luu.end());
      long m=luu.size()-1;
```

```
        fort(i,1,m/2)
         if (luu[i]+luu[m-i+1]!=val) return -1;
        if (m%2==0) return 0;else return luu[m/2+1];
    } else return 0;
}
bool ok(LL x)
{
    LL tt=dfs(1,0,x);
    return (tt==0);
}
int main()
{
    #define task "codeforces."
    //freopen(task"inp","r",stdin);
    //freopen(task"out","w",stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n;
    a.resize(n+1);
    fort(i,1,n-1)
    {
        LL u,v;
        cin>>u>>v;
        a[u].pb(v);a[v].pb(u);
    }
    n--;
    vector<long> kq;
    fort(i,1,n)
     if (n%i==0) kq.pb(i);
    for(long i:kq)
        if (ok(i)) k[i]=1;
    fort(i,1,n) cout<<k[i];
}
```

Nếu đề bài chỉ cho một số nguyên K, thì cách giải quyết trên có thể coi là tối ưu, vì ta chỉ dùng một lần dfs. Tuy nhiên đề bài lại cho N số nguyên K, với K chạy từ 1 tới N, do đó, độ phức tạp của thuật toán lúc này là $O(N^2)$. Cách trên trở nên chậm vì ta phải tính chiều dài của đường đi xuất phát từ một nút x nhiều lần

Cách tối ưu:

Ta sẽ tìm cách lưu lại tổng các con đường xuất phát từ một nút x, ta gọi là sub[x], và ứng với mỗi nút x, ta tìm cách lưu lại chiều dài của các con đường là nhánh con của nút x, vào một vector gọi là num[a]. Khi đó ta chỉ cần gọi một lần dfs, sau đó các kết quả trên sẽ

được lưu lại. Ứng với mỗi k chạy từ 1 tới N, ta sẽ dùng một hàm kiểm tra để kiểm tra xem ta có khả năng ghép cặp các nhánh con bằng cách sau:

- Ta dùng một mảng cur[i] dùng để lưu số lượng các nhánh con có số dư trong phép chia chiều dài của nhánh con cho K bằng i. Ban đầu mảng cur[i]=0 với i chạy từ 0 tới K-1
- Tiếp đến ứng với mỗi đỉnh j từ 1 tới N, ta sẽ kiểm tra các chiều dài tất cả các nhánh con được lưu trong mảng num[i], lấy phần dư và lưu vào mảng cur[i], khi đó ta tăng giảm mảng cur[i] để tìm cách ghép cặp. Chỉ cần trong lúc kiểm tra j từ 1 tới N, nếu gặp một tình huống xuất phát từ đỉnh j mà ghép cặp không thành công, ta đưa ngay ra giá trị false

Đoạn code tham khảo:

```
#include "bits/stdc++.h"

using namespace std;

#define f first
#define s second

const int MOD = 1e9+7;
const int MX = 1e5+5;

int N,sub[MX];
vector<int> adj[MX], num[MX];
bool bad = 0;

void dfs(int a, int b) {
     sub[a] = 1;
     for(auto& t: adj[a]) if (t != b) {
          dfs(t,a);
          sub[a] += sub[t];
          num[a].push_back(sub[t]);
     }
     if (sub[a] != N) num[a].push_back(N-sub[a]);
}

int cur[MX]; bool ok(int K) {
     if ((N-1)%K != 0) return 0;
     for (int i = 0; i < K; ++i) cur[i] = 0;
     for (int i = 1; i <= N; ++i) {
          int cnt = 0;
          for (auto& t: num[i]) {
               int z = t%K; if (z == 0) continue;
```

```
                if (cur[K-z]) cur[K-z] --, cnt --;
                else cur[z] ++, cnt ++;
           }
           if (cnt) return 0; // paths don't pair up
     }
     return 1;
}

int main() {
     setIO("deleg");
     cin >> N;
     for (int i = 1; i < N; ++i) {
          int a,b; cin >> a >> b;
          adj[a].push_back(b), adj[b].push_back(a);
     }
     dfs(1,0);
     for (int i = 1; i < N; ++i) {
          if (ok(i)) cout << 1;
          else cout << 0;
     }
     cout << "\n";
}
```

Một cách cài đặt khác cho bài toán trên:

```
#include <bits/stdc++.h>
#define N 100002
using namespace std;
vector <int> a[N];
int n,i,u,v,kt,sl[N],d[N],ts[N],kq[N],j,dd[N],cnt;
void dfs(int u,int p)
{
    sl[u]=1;
    for(int i=0;i<a[u].size();i++)
    {
        int v=a[u][i];
        if(v==p) continue;
        dfs(v,u);
        sl[u]+=sl[v];
    }
}
void DFS(int u,int p,int k)
{
    cnt++;
    int luu=cnt;
```

```
    for(int i=0;i<a[u].size();i++)
    {
        int v=a[u][i];
         if(v==p) continue;
        DFS(v,u,k);
        if(kt==0) return ;
    }
    int dem=0;
    for(int i=0;i<a[u].size();i++)
    {
        int v=a[u][i];
        if(v==p) continue;
        int x=sl[v]%k;
        if(d[x]!=luu)
        {
            d[x]=luu; ts[x]=0;
        }
        if(x==0) continue;
        if(d[k-x]==luu && ts[k-x]>0)
        {
            ts[k-x]--; dem--;
        }
        else
        {
            dem++;
            ts[x]++;
        }
    }
    if(dem>1)
    {
        kt=0;
    }
}
void xuly1()
{
    dfs(1,0);
    for(i=n-1;i>=1;i--)
    {
        if((n-1)%i>0) continue;
        for(j=2*i;j<n;j+=i)
            if(kq[j]==1) { kq[i]=1; break; }
        if(kq[i]==1) continue;
        kt=1;
```

```
        DFS(1,0,i);
        kq[i]=kt;
    }
}

int main()
{
 //   freopen("ntu.inp","r",stdin);
 //   freopen("ntu.out","w",stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin>>n;
    for(i=1;i<n;i++)
    {
        cin>>u>>v;
        a[u].push_back(v); a[v].push_back(u);
    }
    xuly1();
    for(i=1;i<n;i++)
    {
        if(kq[i]==1) putchar('1');
        else putchar('0');
    }
}
```

Bộ test

https://drive.google.com/drive/folders/1PMPJRgyvIcNV5KlBo70kDzcLJIrnHQXs?usp=sharing

Bài 3. Virus corona

Số nguyên dương gọi là may mắn nếu biểu diễn ở hệ thập phân của chúng không chứa các chữ số nào khác ngoài 4 và 7. Ví dụ, số 47, 744, 4, 7 là số may mắn, còn 5,17,467 không là số may mắn. Phương có ý định du học Nhật Bản. Sau khi virut Covid19 xuất hiện ở Nhật Bản, Phương cảm thấy rất buồn vì mình rất thích văn hóa Nhật, những đức tính kỷ luật, tự giác, trung thực của con người nơi đây. Phương nằm mơ thấy mình may mắn trở thành thủ tướng của đất nước Nhật Bản, đẩy lùi đại dịch Covid19. Đất nước này bao gồm gồm nhiều khu vực, mỗi hòn đảo thuộc chính xác một khu vực, có một con đường đi giữa bất kỳ hai hòn đảo trong một khu vực, và không có con đường đi nào giữa hai hòn đảo thuộc hai khu vực khác nhau.

> Một khu vực gọi là may mắn nếu số lượng đảo trong khu vực đó là con số may mắn. Trong giấc mơ Phương làm Thủ tướng, lần đầu tiên Phương quyết định xây một bệnh viện dã chiến. Phương rất thích các con số may mắn, do đó, Phương muốn đặt bệnh viện dã chiến của mình ở một trong những khu vực may mắn. Tuy nhiên, ban đầu đất nước Nhật Bản không có những khu vực may mắn như vậy. Trong tình huống này, Phương quyết định xây dựng thêm đường đi để nối giữa các khu vực khác nhau, để chúng hợp thành một khu vực may mắn

Yêu cầu đặt ra là, để tiết kiệm về kinh tế, em hãy tư vấn cho Phương số lượng con đường ít nhất cần xây thêm để Phương có thể xây bệnh viện dã chiến ở một khu vực may mắn.

Input: Dòng đầu tiên gồm hai số nguyên dương n và m, biểu thị số lượng đảo và số lượng đường đi hai chiều tương ứng. m dòng tiếp theo mô tả một con đường kết nối giữa hai đảo u và v. Một số con đường có thể kết nối một hòn đảo với chính nó; có thể có nhiều hơn một con đường giữa một cặp đảo.

Output: Nếu không thể giúp Phương xây được con đường nào, em hãy đưa ra -1. Nếu giúp được Phương, em hãy đưa ra số lượng con đường ít nhất để các đảo có thể hình thành nên một khu vực may mắn.

Input

Input của test 1:

4 3

1 2

2 3

1 3

Input test 2:

5 4

1 2

3 4

4 5

3 5

Giới hạn:

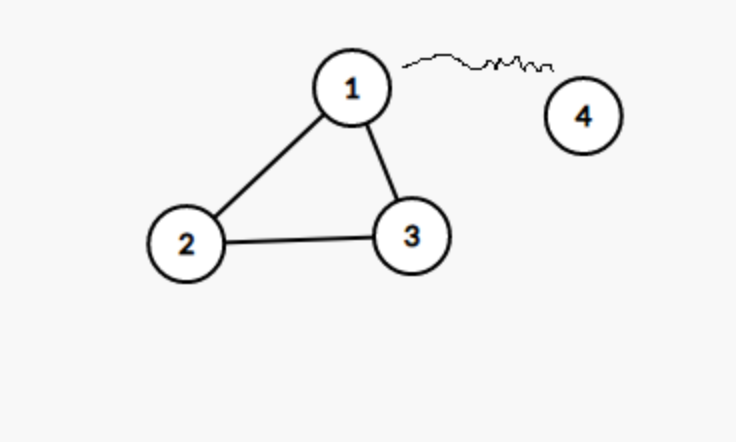
$1 \leq n, m \leq 10\ 5$
$1 \leq u, v \leq n$

Output của test 1:

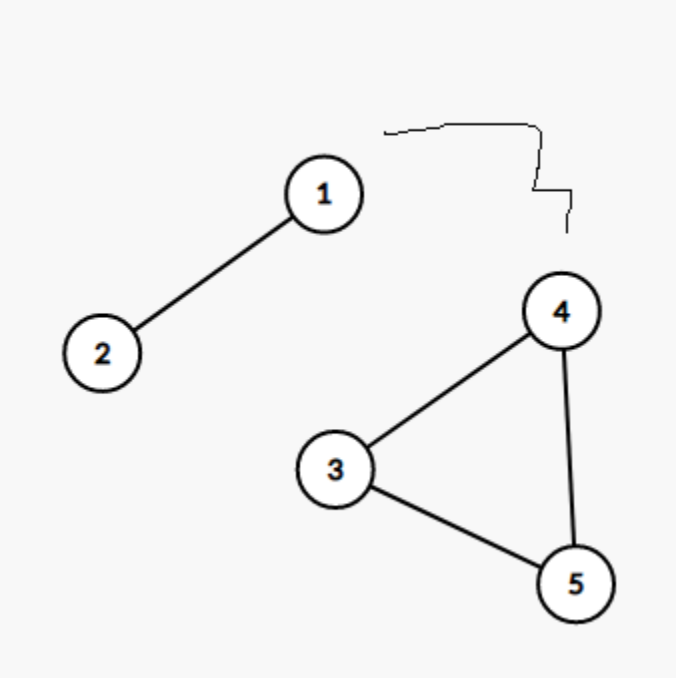
1

Output của test 2:

-1



Trong test số 1 ta chỉ cần nối thêm như hình vẽ là tạo ra một khu vực có số lượng đảo tạo thành số may mắn



Trong test số 2, dù có nối thêm hay không thì không có cách nào tạo ra một khu vực có số lượng đảo là số may mắn

Cách chưa tối ưu:

Ta có thể tóm tắt lại bài toán này như sau: " Cho đồ thị gồm nhiều thành phần liên thông với nhau, ta sẽ tìm cách nối các thành phần liên thông này lại với nhau, với mục đích, tạo thành một thành phần liên thông có số đỉnh là con số may mắn, đồng thời số cạnh nối cũng là ít nhất"

Đây là một bài tập rất khó nhưng cũng rất hay, bài toán này khi đọc đề học sinh sẽ cảm thấy khó ở những vấn đề sau:

Vấn đề 1: đó là học sinh sẽ cảm thấy hoang mang vì không biết nên nối các thành phần liên thông với nhau như thế nào. Việc đầu tiên mà học sinh nghĩ đến có lẽ là kiểm tra xem từng thành phần liên thông xem có phải là số may mắn hay không. Học sinh có thể nghĩ đến việc nối tập con các thành phần liên thông lại với nhau, sau đó cộng số lượng các nút trong các thành phần liên thông của tập con này lại, và kiểm tra xem chúng có phải là số may mắn hay không.

Vấn đề 2: Để giải quyết vấn đề 1 nói trên, học sinh có thể tư duy bằng bài toán quy hoạch động, đó là bài toán cái túi. Mỗi lần ta nối một thành phần liên thông, ta coi chi phí của của việc nối này là 1. Vấn đề đặt ra là ta phải nối các thành phần liên thông để tổng chi phí của việc nối là thấp nhất và trọng lượng của việc nối (hay số các nút sau khi nối) phải là số may mắn. Đây đúng là bài toán cái túi. Tuy nhiên, số may mắn thì có rất nhiều số may mắn, vậy ta có thể làm như thế nào?

Để giải quyết vấn đề 2 nói trên, ta nhận thấy rằng số may mắn tối đa chính là số lượng các nút trong đồ thị. Do đó, số may mắn chắc chắn sẽ phải nằm trong đoạn từ 1 tới n. Vậy để giải quyết vấn đề 2, ta sẽ cố định từng số may mắn một trong đoạn từ 1 tới n. Giả sử ta gọi số may mắn đó là i, vậy thì d[i] chính là chi phí thấp nhất để nối các thành phần liên thông, để chúng có kích thước là i

Vậy câu trả lời của bài toán chắc chắn là min(d[i]) với i là số may mắn trong đoạn từ 1 tới n

Vấn đề 3. Công thức tính d[i] như thế nào? Công thức này lại phải suy xét từ việc ta nên nối các thành phần liên thông như thế nào? Ở đây ta sẽ tìm cách đếm số lượng các thành phần liên thông có kích thước giống nhau lại với nhau, ta sẽ lưu vào một mảng tần số là c[i], với c[i] là số lượng các thành phần liên thông có kích thước là i. Vậy ban đầu ta sẽ tìm cách nối 1 thành phần liên thông giống nhau lại, 2 thành phần liên thông giống nhau lại, 3 thành phần liên thông giống nhau lại,….cho đến c[i] thành phần liên thông giống nhau lại với nhau. Khi đó: chi phí nhỏ nhất để nối thành một thành phần liên thông có kích thước i sẽ bằng chi phí để nối thành phần liên thông có kích thước [i trừ đi số lượng nối thành phần liên thông giống nhau lại với nhau] cộng với chi phí nối thành phần liên thông giống nhau lại, với chi phí nối thành phần liên thông giống nhau lại dao động từ 0 tới c[i]

Sau khi giải quyết được vấn đề 1, vấn đề 2, vấn đề 3, ta sẽ đi đến được cách cài đặt sau:

```
#include<bits/stdc++.h>
using namespace std;
#define maxn 100010
int n,m;
int par[maxn],sz[maxn],c[maxn],dp[maxn];
bool isLucky(int x){
     while(x){
          if(x%10!=4 && x%10!=7)   return false;
          x/=10;
     }
     return true;
}
int find(int x){
     return (par[x]==x) ? x:par[x]=find(par[x]);
}
void upd(int val,int wt){
     for(int i=n;i>=val;i--)
          dp[i]=min(dp[i],dp[i-val]+wt);
}
int main(){
     scanf("%d %d",&n,&m);
     for(int i=1;i<=n;i++) par[i]=i,sz[i]=1;
```

```
	for(int i=1;i<=m;i++){
		int u,v;
		scanf("%d %d",&u,&v);
		int U=find(u),V=find(v);
		if(U!=V){
			if(sz[U]<=sz[V]) sz[V]+=sz[U],sz[U]=0,par[U]=V;
			else sz[U]+=sz[V],sz[V]=0,par[V]=U;
		}
	}
	for(int i=1;i<=n;i++)	c[sz[i]]++,dp[i]=1e9;
	for(int i=1;i<=n;i++) if(c[i]){
		for(int j=1;j<=c[i];j++)
			upd(i,1);
	}
	int res=1e9;
	for(int i=1;i<=n;i++)
		if(isLucky(i)) res=min(res,dp[i]);
	printf("%d\n",res==1e9?-1:res-1);
	return 0;
}
```

Một cách cài đặt khác cho cách chưa tối ưu này:

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
const int SQ=320;
const int inf=1e6;

vector<int> adj[N];
int t[N];
bool mark[N];
int sz;
vector<int> can;

int dp[SQ][N];

void dfs(int a)
{
	sz++;
	mark[a]=1;
	for(int p:adj[a])
		if(!mark[p])
			dfs(p);
}
```

```
bool ok(int a)
{
     while(a)
     {
          if(a%10 !=4 && a%10 !=7)
               return 0;
          a/=10;
     }
     return 1;
}

int main()
{
     ios_base::sync_with_stdio(0);
     int n,m;
     cin >> n >> m;
     for(int i=1,u,v;i<=m;i++)
     {
          cin >> u >> v;
          adj[u].push_back(v);
          adj[v].push_back(u);
     }
     for(int i=1;i<=n;i++)
     {
          if(!mark[i])
          {
               sz=0;
               dfs(i);
               t[sz]++;
          }
     }
     for(int i=1;i<=n;i++)
          if(t[i])
               can.push_back(i);
     int s=can.size();
     dp[0][0]=0;
     for(int i=1;i<=n;i++)
          dp[0][i]=inf;
     for(int i=1;i<=s;i++)
     {
          int k=can[i-1];
          for(int j=0;j<=n;j++)
```

```
          {
               dp[i][j]=dp[i-1][j];
               for(int d=1;d<=t[k];d++)
               {
                    if(j-k*d>=0)
                         dp[i][j]=min(dp[i][j],d+dp[i-1][j-
k*d]);
                    else
                         break;
               }
          }
     }
     int ans=1e6;
     for(int i=1;i<=n;i++)
          if(ok(i))
               ans=min(ans,dp[s][i]);
     cout<<(ans<=n ? ans-1 : -1);
     return 0;
}
```

Cách tối ưu:

Cách suy nghĩ trên hoàn toàn hợp lý, tuy nhiên, việc thử với tất cả các số dao động từ 0 tới c[i] với c[i] là số lượng các thành phần liên thông có kích thước là I, khiến cho bài toán trở nên chậm. Do đó, ta sẽ tìm cách viết c[i] dưới dạng tổng các lũy thừa của 2, vậy thì mỗi lần thử, ta sẽ lấy c[i]-c[i]/2; bằng cách này, số lần thử với c[i] sẽ giảm xuống còn logarit cơ số 2 của c[i].

Code

Sau đây là đoạn code mô tả cách cài đặt như vậy:

```
#include<bits/stdc++.h>
using namespace std;
#define rep(i,a,b) for(int i=a;i<=b;++i)
const int maxn = 101000;
int fa[maxn],sz[maxn],c[maxn];
int fd(int f){
     return fa[f]==f?f:fa[f]=fd(fa[f]);
}
int n,m,f[maxn];
bool isok(int x){
     while(x){
          int d=x%10;
          if(d!=4&&d!=7)return false;
          x/=10;
```

```
	}
	return true;
}
void bp(int v,int w){
	for(int i=n;i>=v;--i)
		f[i]=min(f[i],f[i-v]+w);
}
int main(){
	scanf("%d%d",&n,&m);
	rep(i,1,n)fa[i]=i;
	rep(i,1,m){
		int u,v;
		scanf("%d%d",&u,&v);
		fa[fd(u)]=fd(v);
	}
	rep(i,1,n)sz[fd(i)]++;
	rep(i,1,n)c[sz[i]]++,f[i]=1e9;
	rep(i,1,n)if(c[i]){
		for(int s=1;s<=c[i];s<<=1)
			bp(s*i,s),c[i]-=s;
		bp(c[i]*i,c[i]);
	}
	int ans=1e9;
	rep(i,1,n)if(isok(i))ans=min(ans,f[i]);
	printf("%d\n",ans==1e9?-1:ans-1);
	return 0;
}
```

Như vậy, bài này độ khó ngang với cái tên, nhưng lại rất hay ở chỗ, dùng đồ thị nhưng lại phải nắm vững bài toán cái túi

Bộ test

https://drive.google.com/drive/folders/1PMPJRgyvIcNV5KlBo70kDzcLJIrnHQXs?usp=sharing

Bài 4. Cây con trắng đen lớn nhất

Cho một cây gồm n đỉnh. Một cây là một đồ thị vô hướng liên thông, gồm có n-1 cạnh. Mỗi đỉnh v của cây này được gán một màu (av=1 nếu đỉnh v màu trắng và bằng 0 nếu đỉnh v màu đen)

Yêu cầu đặt ra: Hiệu lớn nhất giữa số đỉnh trắng và số đỉnh đen nếu bạn chọn một cây con trong cây đã cho có chứa đỉnh v? Cây con của cây là đồ thị con liên thông của cây đã cho. Cụ thể hơn, nếu bạn chọn cây con có chứa ***cntw*** đỉnh màu trắng và *cntb* đỉnh màu đen, bạn phải tìm max của hiệu cntw-cntb

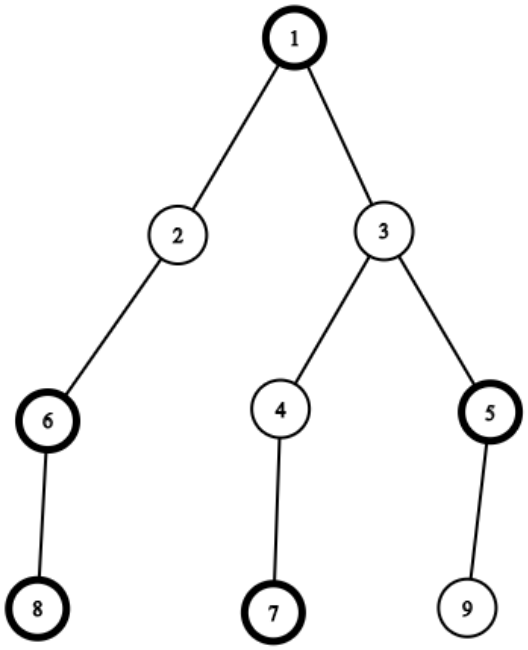
Input: Dòng đầu tiên gồm số nguyên n ($2<=n<=2*10^5$), số đỉnh trong cây

Dòng thứ hai gồm n số nguyên $a_1,a_2,\ldots,a_n$ ($0<=a_i<=1$), trong đó a_i là màu của đỉnh thứ i. Mỗi dòng trong số n-1 dòng tiếp theo mô tả một cạnh của cây. Cạnh i là biểu thị bởi hai số nguyên u_i và v_i, cạnh nối giữa u_i và v_i ($1<=u_i,v_i<=n$, u_i khác v_i). Đảm bảo rằng cạnh đã cho hình thành nên một cây

Output: Đưa ra n số nguyên res1, res 2,….resn, trong đó resi là chênh lệch lớn nhất giữa số đỉnh trắng và số đỉnh đen trong cây con chứa đỉnh i

Input	**Output**
9 0 1 1 1 0 0 0 0 1 1 2 1 3 3 4 3 5 2 6 4 7 6 8 5 9	2 2 2 2 2 1 1 0 2
4 0 0 1 0 1 2	0 -1 1 -1

1 3 1 4	



Test đầu tiên như minh họa ở trên

Cách chưa tối ưu: Ta sẽ sử dụng thuật toán dfs xuất phát từ mỗi đỉnh của đồ thị, sau đó xuất phát từ mỗi đỉnh, ta sẽ tìm kiếm tất cả các nhánh con của đồ thị, nếu nhánh con nào có độ chênh lệch giữa đỉnh trắng và đỉnh đen là âm, thì ta sẽ bỏ qua nhánh con đó, chỉ cộng vào nhánh con có độ chênh lệch giữa đỉnh trắng và đỉnh đen là dương.

Đoạn code tham khảo:

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long double ld;

const int INF = 1000000009, MAXN = 200005;

int n;
vector<int> a, adj[MAXN], s, ans, pa;
```

```
int dfs(int u, int p = -1){
    if(pa[u] == p)   return s[u];
    pa[u] = p;
    s[u] = 0;

    for(int v : adj[u]){
        if(v == p)  continue;
        if(dfs(v, u) > 0)
            s[u] += s[v];
    }
    if(a[u])    s[u] ++;
    else        s[u] --;
    return s[u];
}

void solve(){
    cin >> n;
    a.resize(n+1), ans.resize(n+1), s.resize(n+1),
pa.resize(n+1);
    for(int i = 1; i <= n; i++){
        cin >> a[i];
    }
    for(int i = 0; i < n-1; i++){
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    for(int i = 1; i <= n; i++)
        dfs(i), cout << s[i] << " ";
    cout << "\n";
}

int main(){
     ios_base::sync_with_stdio(0),cin.tie(0),cout.tie(0);
    // int t;cin >> t;while(t--)   solve();
    solve();
    return 0;
}
```

Cách trên sở dĩ chưa tối ưu vì mỗi lần ta cần đến đỉnh nào, ta gọi là gọi dfs xuất phát từ đỉnh đó, qua niệm đỉnh đó làm gốc, sau đó cộng vào tất cả các nhánh con có sự chênh lệch giữa số đỉnh trắng và số đỉnh đen là lớn hơn 0

Cách tối ưu: Vẫn giống cách chưa tối ưu, nhưng ta có sự cải tiến để việc dfs không diễn ra trên tất cả các gốc, mà chỉ diễn ra trên một gốc thôi. Đầu tiên ta tính toán cho một gốc cố định, dfs(1). Gọi dp[v] là hiệu số lớn nhất có thể giữa số đỉnh trắng và số đỉnh đen trên một số cây con của v (cây con của cây gốc v) chứa đỉnh v. Ta có thể tính được điều này bằng công thức quy hoạch động sau:

$$dp_v = a_v + \sum_{to \in con(v)} \max(0, dp_{to})$$

Giả sử **to** là một đỉnh bất kỳ trong lần gọi dfs đầu tiên này. Như vậy, ở đây, công thức tìm hiệu số giữa số đỉnh trắng và số đỉnh đen mới chỉ lưu lại kết quả là các con của đỉnh **to**, mà chưa lưu lại kết quả từ đỉnh **to** tới cha của nó (ta gọi là đỉnh v).

Do đó khi duyệt đến cạnh (v,to). Đầu tiên, ta sẽ xóa cây con với gốc ***to*** từ cây con của đỉnh v:

$$dp_v = dp_v - \max(0, \mathrm{dp}_{to})$$

Sau đó ta biến đỉnh v trở thành cây con của đỉnh to:

$$dp_{to} = dp_{to} + \max(0, \mathrm{dp}_v)$$

Sau lời gọi đệ quy xuất phát từ đỉnh to tới các đỉnh là cây con, ta khôi phục trở lại trạng thái của đỉnh v và đỉnh to

Độ phức tạp thuật toán là O(n)

Đoạn code tham khảo

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

using namespace std;

vector<int> a;
vector<int> dp;
vector<int> ans;
vector<vector<int>> g;

void dfs(int v, int p = -1) {
     dp[v] = a[v];
     for (auto to : g[v]) {
          if (to == p) continue;
          dfs(to, v);
          dp[v] += max(dp[to], 0);
     }
}
```

```
void dfs2(int v, int p = -1) {
	ans[v] = dp[v];
	for (auto to : g[v]) {
		if (to == p) continue;
		dp[v] -= max(0, dp[to]);
		dp[to] += max(0, dp[v]);
		dfs2(to, v);
		dp[to] -= max(0, dp[v]);
		dp[v] += max(0, dp[to]);
	}
}

int main() {
#ifdef _DEBUG
	freopen("input.txt", "r", stdin);
//	freopen("output.txt", "w", stdout);
#endif

	int n;
	cin >> n;
	a = dp = ans = vector<int>(n);
	g = vector<vector<int>>(n);
	for (int i = 0; i < n; ++i) {
		cin >> a[i];
		if (a[i] == 0) a[i] = -1;
	}
	for (int i = 0; i < n - 1; ++i) {
		int x, y;
		cin >> x >> y;
		--x, --y;
		g[x].push_back(y);
		g[y].push_back(x);
	}

	dfs(0);
	dfs2(0);
	for (auto it : ans) cout << it << " ";
	cout << endl;

	return 0;
}
```

Bộ test:
https://drive.google.com/drive/folders/1PMPJRgyvIcNV5KlBo70kDzcLJIrnHQXs?usp=sharing

Bài 5. Tổ tiên chung gần nhất

Cho một cây gồm có N đỉnh, bạn hãy trả lời câu hỏi có dạng "r u v", nghĩa là gốc của cây nếu đặt gốc của cây ở r thì khi đó tổ tiên chung gần nhất (LCA) của u và v là đỉnh nào?

Input

Dòng đầu tiên gồm số nguyên N. Mỗi dòng trong N-1 dòng tiếp theo gồm một cặp số nguyên u và v biểu thị một cạnh nối giữa hai đỉnh. Dòng tiếp theo gồm số nguyên Q mô tả số lượng câu hỏi. Mỗi dòng trong số Q dòng tiếp theo gồm 3 số nguyên r, u, v biểu thị một câu hỏi.

Output: Ứng với mỗi câu hỏi, đưa ra câu trả lời trên một dòng

Giới hạn:

Subtaks1:

- $1 \leq \mathbf{N}, \mathbf{Q} \leq 100$

Subtask 2:

- $1 \leq \mathbf{N}, \mathbf{Q} \leq 10^5$
- Có ít hơn 10 giá trị duy nhất của r trong các câu hỏi.

Subtaks 3:

- $1 \leq \mathbf{N}, \mathbf{Q} \leq 2 \times 10^5$

Input	Output
4 1 2 2 3 1 4 2 1 4 2 2 4 2	1 2

Giải thích: "1 4 2": nếu 1 là gốc, đỉnh này là cha của 2 và 4 vì vậy LCA của 2 và 4 là 1.

- "2 4 2": nếu 2 là gốc của cây, LCA của 4 và 2 chính là 2

Giải thích:

Cách chưa tối ưu:

Subtask 1: Đối với sub này, bạn tìm LCA theo bất kỳ cách nào bạn muốn miễn là độ phức tạp không chậm hơn O (N). Ví dụ:

Tổ tiên chung gần nhất là nút có thời gian vào (in-time) nhỏ hơn hoặc bằng hai nút đã cho, và thời gian ra (out-time) lớn hơn hoặc bằng (bằng nếu tổ tiên chung chính là một trong hai nút đã cho) tất cả các nút trong tập hợp đó. Do đó, để giải quyết bài toán này, ta cần duyệt qua toàn bộ cây bắt đầu từ nút gốc bằng cách sử dụng DFS và lưu trữ thời gian vào (in-time), thời gian ra (out-time) và mức (level) cho mỗi nút. Nút với thời gian vào nhỏ hơn hoặc bằng hai nút đã cho và thời gian ra lớn hơn hoặc bằng hai nút đã cho, cùng với mức lớn nhất, sẽ là câu trả lời cho bài toán

Đoạn code dưới đây mô phỏng thuật toán như vậy, tìm tổ tiên chung gần nhất cho một tập các nút, trường hợp có hai nút là một trường hợp riêng của trường hợp này:

```
/ C++ Program to find the LCA
// in a rooted tree for a given
// set of nodes

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

// Set time 1 initially
int T = 1;
void dfs(int node, int parent,
         vector<int> g[],
         int level[], int t_in[],
         int t_out[])
{

    // Case for root node
    if (parent == -1) {
        level[node] = 1;
    }
    else {
        level[node] = level[parent] + 1;
    }

    // In-time for node
    t_in[node] = T;
    for (auto i : g[node]) {
        if (i != parent) {
```

```
            T++;
            dfs(i, node, g,
                level, t_in, t_out);
        }
    }
    T++;

    // Out-time for the node
    t_out[node] = T;
}

int findLCA(int n, vector<int> g[],
            vector<int> v)
{

    // level[i]--> Level of node i
    int level[n + 1];

    // t_in[i]--> In-time of node i
    int t_in[n + 1];

    // t_out[i]--> Out-time of node i
    int t_out[n + 1];

    // Fill the level, in-time
    // and out-time of all nodes
    dfs(1, -1, g,
        level, t_in, t_out);

    int mint = INT_MAX, maxt = INT_MIN;
    int minv = -1, maxv = -1;
    for (auto i = v.begin(); i != v.end(); i++) {
        // To find minimum in-time among
        // all nodes in LCA set
        if (t_in[*i] < mint) {
            mint = t_in[*i];
            minv = *i;
        }
        // To find maximum in-time among
        // all nodes in LCA set
        if (t_out[*i] > maxt) {
            maxt = t_out[*i];
            maxv = *i;
```

```
        }
    }

    // Node with same minimum
    // and maximum out time
    // is LCA for the set
    if (minv == maxv) {
        return minv;
    }

    // Take the minimum level as level of LCA
    int lev = min(level[minv], level[maxv]);
    int node, l = INT_MIN;
    for (int i = 1; i <= n; i++) {
        // If i-th node is at a higher level
        // than that of the minimum among
        // the nodes of the given set
        if (level[i] > lev)
            continue;

        // Compare in-time, out-time
        // and level of i-th node
        // to the respective extremes
        // among all nodes of the given set
        if (t_in[i] <= mint
            && t_out[i] >= maxt
            && level[i] > l) {
            node = i;
            l = level[i];
        }
    }

    return node;
}

// Driver code
int main()
{
    int n = 10;
    vector<int> g[n + 1];
    g[1].push_back(2);
    g[2].push_back(1);
    g[1].push_back(3);
```

```
    g[3].push_back(1);
    g[1].push_back(4);
    g[4].push_back(1);
    g[2].push_back(5);
    g[5].push_back(2);
    g[2].push_back(6);
    g[6].push_back(2);
    g[3].push_back(7);
    g[7].push_back(3);
    g[4].push_back(10);
    g[10].push_back(4);
    g[8].push_back(7);
    g[7].push_back(8);
    g[9].push_back(7);
    g[7].push_back(9);

    vector<int> v = { 7, 3, 8 };

    cout << findLCA(n, g, v) << endl;
}
```

Subtask 2: Ở đây có không quá 10 gốc khác nhau nhưng số lượng câu hỏi rất lớn nên phải tìm LCA nhanh chóng. Cụ thể hơn O (Nlog (N)) là đủ. Do không quá 10 gốc khác nhau nên độ phức tạp sẽ là O (10 × Nlog (N)).

Cách tối ưu

Subtask 3: Sử dụng 2 nhận xét sau:

- Với câu hỏi "r u v", câu trả lời có thể là r, u, v, LCA (r,u), LCA (r, v), LCA (u,v), trong đó LCA (x,y) là LCA của x và y khi cây có gốc ở 1.
- LCA của u và v khi gốc tại r là đỉnh x sao cho tổng đường đi ngắn nhất từ x đến u, x đến v, và x đến r là nhỏ nhất

Với hai quan sát này, ta cần thực hiện hai hàm: tìm LCA và khoảng cách của hai đỉnh trong cây.

Đoạn code tham khảo

```
#include <bits/stdc++.h>
#define ios
ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NU
LL)
typedef long long ll;
```

```
typedef double dl;
#define pb push_back
#define ins insert
#define pll pair<ll,ll>
#define fi first
#define se second
#define mpr make_pair
#define vtr vector
#define itor iterator
#define N ll(1e9+7)
#define endl '\n'
#define cont continue
#define inf ll(1e10)
using namespace std;
#define val 20
vtr<ll> tin(ll(2e5+10)),tout(ll(2e5+10));
ll timer=0;
ll up[ll(2e5+10)][val];
vtr<ll> adj[ll(2e5+10)];
void dfs(ll src,ll par)
{
     tin[src]=++timer;
     ll i,j,v;
     up[src][0]=par;
     for(i=1;i<val;i++)
     {
          up[src][i]=up[up[src][i-1]][i-1];
     }
     for(i=0;i<adj[src].size();i++)
     {
          v=adj[src][i];
          if(v!=par)
          {
               dfs(v,src);
          }
     }
     tout[src]=++timer;
}
bool is_ancestor(ll u,ll v)
{
     return(tin[u]<=tin[v] and tout[u]>=tout[v]);
}
ll find_ancestor(ll u,ll v)
```

```
{
     if(is_ancestor(u,v))return u;
     if(is_ancestor(v,u))return v;
     ll i;
     for(i=val-1;i>=0;i--)
     {
          if(!is_ancestor(up[u][i],v))u=up[u][i];
     }
     return up[u][0];
}
int main()
{
          ios;
          ll n;
          cin>>n;
          timer=0;
          ll num,i,j,u,v;
          for(i=1;i<=n-1;i++)
          {
               cin>>u>>v;
               adj[u].pb(v);
               adj[v].pb(u);
          }
          dfs(1,1);
          ll q,r;
          cin>>q;
          while(q--)
          {
               cin>>r>>u>>v;
               if(u==v)cout<<u<<endl;
               else
               {
                    ll a1=find_ancestor(u,v);
                    ll a2=find_ancestor(u,r);
                    ll a3=find_ancestor(v,r);
                    if(a1==a2)cout<<a3;
                    else if(a2==a3)cout<<a1;
                    else if(a3==a1)cout<<a2;
                    cout<<endl;
               }
          }
}
```

Bộ test:
https://drive.google.com/drive/folders/1PMPJRgyvIcNV5KlBo70kDzcLJIrnHQXs?usp=sharing

C. Kết luận

Như vậy, đề tài của tôi đã đạt được những kết quả sau:

- Xây dựng và phân bậc được tư duy của học sinh thông qua các cách giải khác nhau, đồng thời qua từng bài đã phân tích được cách để phát triển tư duy từ cách chưa tối ưu sang cách tối ưu. Việc chỉ ra cho học sinh thấy những nhược điểm trong cách chưa tối ưu, sẽ giúp học sinh có thêm kinh nghiệm trong việc giải các bài tập tương tự sau này.
- Hệ thống bài tập tôi đưa ra không quá khó, nhằm khiến cho học sinh có thể vận dụng được thuật toán cơ bản trong đồ thị như dfs, bfs,…, sau đó tìm cách tối ưu nhờ những kiến thức được trang bị từ trước đó như quy hoạch động,…