Bài 1: max.cpp

Sub 1: Với K=1, duyệt 1 vòng lặp và duy trì giá trị **min, res.** ĐPT: O(N)

```
for(int _min=a[1],i=1; i<=n; i++) {
        res=max(res,a[i]-_min);
        _min=min(_min,a[i]);
    }
cout<<res;
```

Sub 2: Với K=2, cải tiến Sub 1, dùng 2 mảng đánh dấu lợi nhuận lớn nhất xuôi và ngược. ĐPT: O(N).

```
res=0;
for(int _min=INT_MAX,i=1; i<=n; i++) {
    res=max(res,a[i]-_min);
    xuoi[i]=res;
    _min=min(_min,a[i]);
}
res=0;
for(int _max=a[n],i=n; i>=1; i--) {
    res=max(res,_max-a[i]);
    nguoc[i]=res;
    _max=max(_max,a[i]);
}
res=0;
for(int i=1;i<=n;i++)
    res=max(res,xuoi[i]+nguoc[i]);
cout<<res;
```

Sub 3: Dùng Quy hoạch động $dp[k][n]$ là giá trị lợi nhuận lớn nhất khi xét đến bán tối đa k lần và xét đến ngày thứ n sẽ là ngày bán. ĐPT: $O(K.N^2)$. Để tối ưu thì cần kết hợp kĩ thuật làm của Sub1 để giảm độ phức tạp còn $O(K.N)$.

$$dp[i][j] = max(dp[i-1][j], dp[i-1][x] + a[j] - a[x])$$

Kết quả tính đến ngày thứ k sẽ tối ưu qua kết quả ngày thứ $k-1$.

Sub 4: Nhận xét thấy rằng chênh lệch lớn nhất khi lấy các cực đại địa phương trừ các cực tiểu địa phương bên trái gần nó nhất. Nếu số cực đại nhỏ hơn N thì dễ dàng, nhưng nếu số cực đại và cực tiểu nhiều hơn K thì cần dùng stack, priority_queue để có thể phải kết hợp một số đoạn với nhau. ĐPT: $O(N.\log(N))$

```
int ans = 0, buy = 0, sell = 0;
stack<pair<int, int> > trans;
priority_queue<int> profits;
while(sell < n) {
    buy = sell;
    while(buy < n - 1&&a[buy] >= a[buy + 1])
        buy++;
    sell = buy + 1;
    while(sell < n  && a[sell] >= a[sell - 1])
        sell++;
    while(!trans.empty()&& a[buy]<a[trans.top().first]) {
        pair<int, int> p = trans.top();
        profits.push(a[p.second - 1] - a[p.first]);
        trans.pop();
    }
    while(!trans.empty()&&a[sell-1]>a[trans.top().second-1]) {
        pair<int, int> p = trans.top();
        profits.push(a[p.second - 1] - a[buy]);
```

```
            buy = p.first;
            trans.pop();
        }
        trans.push({ buy, sell });
    }
    while(!trans.empty()) {
        profits.push(a[trans.top().second-1]-a[trans.top().first]);
        trans.pop();
    }
    while(k && !profits.empty()) {
        ans += profits.top();
        profits.pop();
        --k;
    }
    cout<<ans;
```

Bài 2: Office.cpp

Sub 1: QHĐ trên cây dạng cơ bản, không được chọn 2 thành phố liên tiếp để đặt văn phòng. ĐPT: $O(N)$

Sub 2: QHĐ trên cây với ĐPT: $O(N.K)$

Sub 3. QHĐ có sử dụng kết hợp ghi nhớ đỉnh đã được chọn đặt văn phòng gần đó nhất. ĐPT: $O(N)$

```
#include <bits/stdc++.h>
using namespace std;
int n, d;
vector<int> g[200005];
vector<int> level, h, s, ans;
void dfs(int u, int p, int depth)
{
    level[u]=depth; ///level
    h[u]=1000000000;
    s[u]=u; ///trace
    for(auto v:g[u])
        if(v!=p)
        {
            dfs(v, u, depth+1);
            if(s[v])
            {
                if((level[s[v]]-level[u])*2>=d)
                {
                    ans.push_back(s[v]);
                    h[v]=level[s[v]];
                }
                else if(level[s[v]]>level[s[u]])
                    s[u]=s[v];
            }
            h[u]=min(h[u], h[v]);
        }
    if(level[s[u]]+h[u]-2*level[u]<d)
        s[u]=0;
}
int main()
{
    freopen("Office.inp","r",stdin);
    cin >> n >> d;
    level.resize(n+1);
    h.resize(n+1);
```

```
        s.resize(n+1);
        for(int i=1; i<=n; i++)
        {
            int a, b;
            cin >> a >> b;
            g[a].push_back(b);
            g[b].push_back(a);
        }
        dfs(1, 0, 1);
        if(s[1])
            ans.push_back(s[1]);
        cout << ans.size();
    }
```

Bài 3: updtree

Sub 1: Xử lí mỗi truy vấn cập nhật bằng cách thay đổi lại toàn bộ nhánh cây bị thay đổi. ĐPT: $O(N.Q)$

Sub 2: Do các truy vấn loại 1 được xếp trước, nên tại mỗi đỉnh cần lưu lại giá trị mà nó sẽ truyền xuống con. Không nên cập nhật ngay thông tin kiểu Sub 1. Xử lí một truy vấn trong $O(1)$. Dùng DFS để cập nhật lại thông tin các đỉnh sau khi xong hết truy vấn loại 1.
Trả một truy vấn loại 2 trong O(1). ĐPT: $O(N+Q)$.

Sub 3: Ta sẽ xử lí từng loại truy vấn như sau:

Truy vấn loại 1: Với đỉnh x, có độ sâu L. Thì các đỉnh trong cây con của x sẽ cập nhật như sau: các đỉnh có độ sâu $L+2, L+4, \ldots$ cùng tính chẵn lẻ với x sẽ được cộng một lượng val, các đỉnh khác tính chẵn lẻ sẽ cộng một lượng $-val$. Thay vì đi quản lý tính chẵn lẽ ta sẽ chia tập các đỉnh trên cây thành 2 tập, tập các đỉnh có độ sâu chẵn, tập các đỉnh có độ sâu lẻ. Tùy thuộc vào L mà ta có cách cập nhật thông tin khác nhau. Sau đó cũng có thao tác lấy dữ liệu tương ứng.

Vấn đề là làm sao để xác định các đỉnh thuộc cây con của x nhanh nhất? Ta sẽ trải cây thành mảng:

```
    void dfs(int u) {
       in[u]=cnt++;
       visited[u]=true;
       for(auto v:adj[u]) {
           if(!visited[v]) {
               depth[v]=depth[u]+1;
               dfs(v);
           }
       }
       out[u]=cnt++;
   }
```

Để xác định phạm vi các đỉnh trong cây con của x bị ảnh hưởng ta cập nhật vị trí $in[x]$ lên val và giảm các vị trí $out[x]$ đi val.

Thao tác lấy thông tin tương tự như cập nhật, nếu độ sâu chẵn thì cộng, lẻ thì trừ.

ĐPT: $O(Q.\log(N))$.

Chương trình:

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

```
const int maxn = 412345;
int BIT[maxn],a[maxn],visited[maxn],depth[maxn];
int in[maxn],out[maxn];
int cnt=1;
int n,m,id,x,val;
vector<int> adj[maxn];
void dfs(int u) {
    in[u]=cnt++;
    visited[u]=true;
    for(auto v:adj[u]) {
        if(!visited[v]) {
            depth[v]=depth[u]+1;
            dfs(v);
        }
    }
    out[u]=cnt++;
}
void update(int pos,int val) {
    while(pos<maxn) {
        BIT[pos]+=val;
        pos+=pos&(-pos);
    }
}
int get(int pos) {
    int r=0;
    while(pos>0) {
        r+=BIT[pos];
        pos-=pos&(-pos);
    }
    return r;
}
int main() {
    freopen("updtree.inp","r",stdin);
    scanf("%d%d",&n,&m);
    for(int i=1; i<=n; i++)
        scanf("%d",&a[i]);
    for(int i=0; i<n-1; i++) {
        int u,v;
        scanf("%d%d",&u,&v);
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    dfs(1);
    for(int i=0; i<m; i++) { ///m = queries
        scanf("%d%d",&id,&x);
        if(id==1) { ///truy vấn 1
            scanf("%d",&val);
            if(depth[x]&1) { ///nếu độ sâu là lẻ
                update(in[x],-val);
                update(out[x],val);
            } else {
                update(in[x],val);
                update(out[x],-val);
            }
        } else {  ///truy vấn 2
            if(depth[x]&1) ///nếu độ sâu là lẻ
                printf("%d\n",a[x]-get(in[x]));
            else
                printf("%d\n",a[x]+get(in[x]));
```

```
            }
        }
        return 0;
    }
```

Ngoài cách này thì ta có thể dùng Segment tree với lazy propagation.