IT003.P21.CTTN - Assignment 04.02

Bảo Quý Định Tân - 24520028
Hà Xuân Thiện - 24520031

Đề bài

Cho một cây nhị phân, giả sử nếu đặt 01 camera vào một node thì sẽ quan sát được bản thân nó và node cha cùng 2 node con (nếu có - tức mỗi camera có thể nhìn thấy từ 1 đến 4 node).

Viết thuật toán và hàm xác định số camera tối thiểu cần dùng để có thể thấy toàn bộ các node trong cây.

Ý tưởng

Ta tiếp cận bài toán theo hướng tham lam.

Bởi vì khi đặt camera vào một đỉnh, ta sẽ nhìn thấy được cả các con trực tiếp và cha của mình.

Bắt đầu từ nút lá ta sẽ khoan vội đặt camera vào thì ta có thể đặt vào các đỉnh cha của nó, để có thể quan sát cả các đỉnh lá và lời thêm các đỉnh cha của cha nữa.

Bắt đầu với hướng đó, ta dần đi lên đặt camera chỉ khi cần thiết.

Ta nhận xét rằng chỉ cần đặt camera vào một đỉnh khi trong có đỉnh con trực tiếp chưa quan sát được.

Xây dựng bài toán

Vì đề không ghi rõ định dạng input nên em có nghĩ ra một định dạng input để thuận lợi cho việc tạo ra cây nhị phân code bằng con trỏ.

Ta sẽ in ra cây nhị phân theo thứ tự duyệt dfs và đi vào nhánh bên trái trước.

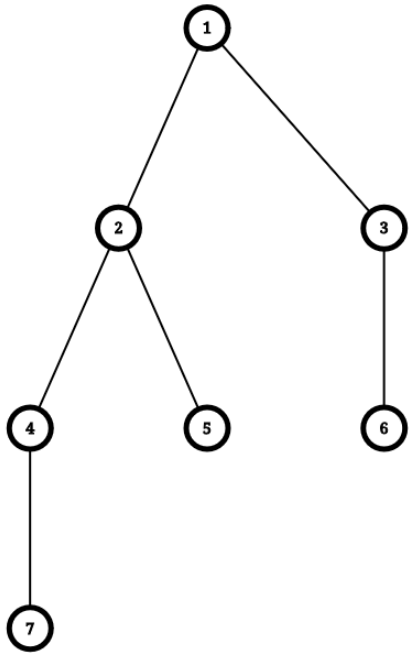
Khi tới mỗi đỉnh, ta sẽ xuất ra:

- Chỉ số của đỉnh hiện tại.
- Chỉ số của đỉnh con bên trái (nếu không có, in ra -1).
- Chỉ số của đỉnh con bên phải (nếu không có, in ra -1).

Ví dụ:

Một cây nhị phân gồm 7 đỉnh sẽ xuất ra input là:

```
7
1 2 3
2 4 5
4 7 -1
7 -1 -1
5 -1 -1
3 6 -1
6 -1 -1
```



Code

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

// Node structure
struct Node
{
    int id;
    Node *left;
    Node *right;
    Node(int _id) : id(_id), left(nullptr), right(nullptr) {}
};

int n;
int cam = 0;

// DFS returns:
// 0 - not monitored
// 1 - monitored (by child's camera)
// 2 - has camera
int dfs(Node *node)
{
    if (!node)
        return 1; // null nodes are considered monitored
```

```
24
25      int leftState = dfs(node->left);
26      int rightState = dfs(node->right);
27
28      if (leftState == 0 || rightState == 0)
29      {
30          cam++;
31          return 2; // place camera here
32      }
33
34      if (leftState == 2 || rightState == 2)
35          return 1; // this node is monitored by child
36
37      return 0; // this node is not monitored
38  }
39
40  void readTree(Node *&node)
41  {
42      int i, leftId, rightId;
43      cin >> i >> leftId >> rightId;
44
45      node = new Node(i);
46
47      if (~leftId)
48      {
49          readTree(node->left);
50      }
51      if (~rightId)
52      {
53          readTree(node->right);
54      }
55  }
56
57  signed main()
58  {
59      ios_base::sync_with_stdio(false);
60      cin.tie(nullptr);
61
62      cin >> n;
63
64      Node *root = nullptr;
65
66      readTree(root);
67
68      if (dfs(root) == 0)
69          cam++; // if root is not monitored, place a camera
70
71      cout << cam;
72
73      return 0;
74  }
```

Code để sinh ra input:

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <random>
#include <chrono>
using namespace std;

struct TreeNode
{
        int val;
        TreeNode *left;
        TreeNode *right;
        TreeNode(int _val) : val(_val), left(nullptr), right(
           nullptr) {}
};

// Global RNG
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count
   ());
uniform_int_distribution<int> childDist(1, 2); // for generating
   1 or 2 children

// DFS-based tree generator
TreeNode *generateTreeDFS(int &currIndex, int maxNodes)
{
        if (currIndex > maxNodes)
                return nullptr;

        TreeNode *node = new TreeNode(currIndex++);

        int childCount = childDist(rng);

        if (childCount >= 1 && currIndex <= maxNodes)
                node->left = generateTreeDFS(currIndex, maxNodes)
                   ;

        if (childCount == 2 && currIndex <= maxNodes)
                node->right = generateTreeDFS(currIndex, maxNodes
                   );

        return node;
}

// BFS-based tree generator
TreeNode *generateTreeBFS(int maxNodes)
{
        if (maxNodes < 1)
                return nullptr;

```

```
45          int currIndex = 1;
46          TreeNode *root = new TreeNode(currIndex++);
47          queue<TreeNode *> q;
48          q.push(root);
49
50          while (!q.empty() && currIndex <= maxNodes)
51          {
52                  TreeNode *node = q.front();
53                  q.pop();
54
55                  int childCount = childDist(rng);
56
57                  if (childCount >= 1 && currIndex <= maxNodes)
58                  {
59                          node->left = new TreeNode(currIndex++);
60                          q.push(node->left);
61                  }
62
63                  if (childCount == 2 && currIndex <= maxNodes)
64                  {
65                          node->right = new TreeNode(currIndex++);
66                          q.push(node->right);
67                  }
68          }
69
70          return root;
71  }
72
73  // Preorder tree traversal for display
74  void printTree(TreeNode *root)
75  {
76          if (!root)
77                  return;
78          cout << root->val << '␣';
79          if (root->left)
80                  cout << root->left->val;
81          else
82                  cout << -1;
83          cout << '␣';
84          if (root->right)
85                  cout << root->right->val;
86          else
87                  cout << -1;
88          cout << '\n';
89          printTree(root->left);
90          printTree(root->right);
91  }
92
93  int Rand(int l, int r)
94  {
95          uniform_int_distribution<int> dist(l, r);
```

```
        return dist(rng);
}

int main()
{

        //freopen("test.inp", "w", stdout);

        int maxNodes = Rand(1, 20);
        cout << maxNodes << '\n';

        // int currIndexDFS = 1;
        // TreeNode *rootDFS = generateTreeDFS(currIndexDFS,
           maxNodes);
        // printTree(rootDFS);

        TreeNode *rootBFS = generateTreeBFS(maxNodes);
        printTree(rootBFS);

        return 0;
}
```

Code trâu để kiểm tra kết quả:

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

using namespace std;

template <class A, class B>
bool maximize(A &x, B y)
{
    if (x < y)
        return x = y, true;
    else
        return false;
}
template <class A, class B>
bool minimize(A &x, B y)
{
    if (x > y)
        return x = y, true;
    else
        return false;
}

#define all(a) a.begin(), a.end()
#define pb push_back
#define fi first
#define se second
```

```cpp
#define int long long

typedef long long ll;
typedef unsigned long long ull;
typedef double db;
typedef long double ld;
typedef pair<int, int> pii;

const int MAX_N = 2e5 + 5;
const int mod = 1e9 + 7;
const ll inf = 1e18;

vector<int> adj[MAX_N];
int cam = 0;
bool mark[MAX_N];

bool dfs(int u, int parent)
{
    bool ok = true;

    for (int v : adj[u])
    {
        if (v == parent)
            continue;
        ok &= dfs(v, u);
    }

    bool watched = mark[u];

    if (mark[parent])
        watched = true;

    for (int v : adj[u])
    {
        if (v == parent)
            continue;
        if (mark[v])
            watched = true;
    }

    return ok && watched;
}

signed main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;

```

```
    for (int i = 1; i <= n; i++)
    {
        int u, v1, v2;
        cin >> u >> v1 >> v2;
        if (~v1)
            cerr << u << ' ' << v1 << '\n';
        if (~v2)
            cerr << u << ' ' << v2 << '\n';
        if (~v1)
        {
            adj[u].pb(v1);
            adj[v1].pb(u);
        }
        if (~v2)
        {
            adj[u].pb(v2);
            adj[v2].pb(u);
        }
    }

    int ans = inf;
    for (int mask = 1; mask <= (1 << n); mask++)
    {
        for (int i = 0; i < n; i++)
        {
            mark[i + 1] = (mask >> (i)) & 1;
        }
        if (dfs(1, 0))
        {
            minimize(ans, __builtin_popcount(mask));
        }
    }

    cout << ans;

    return 0;
}
```