CS F211

Data Structures and Algorithms

Practice Problems

Binary Trees, Euler Tour, and Priority Queue

Problem 1

You are given a binary tree rooted at 1. Each node in the tree has a value. You must organize the nodes of the tree into triangles, where each triangle is a parent and its two children. A node cannot belong to multiple triangles, and all nodes need not belong to a triangle. The score of a triangle is the product of the parent's value and the sum of the two children. Your task is to find the maximum total score obtainable.

The first line of input consists of $n-1$ integers, where the ith integer is the node which is the parent of the $i+1$th node. The second line consists of n integers which represent the values of the nodes.

```
#include <iostream>
#include <vector>
using namespace std;

struct Node {
    /* ??? */
};

void dfs(Node* node) {
    /* ??? */
}

int main() {
    int n;
    cin >> n;
    vector<Node*> nodes(n);
    for (int i = 0; i < n; i++) nodes[i] = new Node(0);
    for (int i = 1; i < n; i++) {
        int parent;
        cin >> parent;
        if (!nodes[parent - 1]->left) nodes[parent - 1]->left
            = nodes[i];
        else nodes[parent - 1]->right = nodes[i];
    }
    for (int i = 0; i < n; i++) cin >> nodes[i]->value;
    dfs(nodes[0]);
    /* ??? */
    return 0;
}
```

```
>> 6
>> 1 1 2 2 3
>> 3 2 5 1 4 6
21
```

Next task: Instead of organizing the nodes into triangles, your code must now organize them into parent-child pairs, where the score of each pair is the product of the values of the two nodes.

Problem 2

You have to implement a special BT-based hashing-based set data structure. Each leaf in this tree represents a set of some values, which are represented by a linked list. Initially, the tree only consists of a single root node where all new values are added. When the number of nodes in a leaf exceeds a predefined threshold value, two children are added to the node, and the values store by the leaf are split across the two new leaf children. When a leaf at depth d splits, the values are divided on the basis of their modulo with 2^{d+1}, or in other words, the values with 0 at the $d+1$th position in its binary representation go to the left child and the values with 1 go to the right child.

The data structure must provide methods to add values and to determine whether a particular value exists or not. As it is a set, each value is stored only once at maximum.

```
#include <iostream>
using namespace std;

struct ListNode {
    int value;
    ListNode* next;
    ListNode(int v) : value(v), next(nullptr) {}
};

struct Node {
    Node* left;
    Node* right;
    int depth;
    ListNode* head;

    Node(int d) : left(nullptr), right(nullptr), depth(d),
       head(nullptr) {}

    bool contains(int val) {
        /* ??? */
    }

    void insertValue(int val) {
        /* ??? */
    }
};

```

```
class BTHashingSet {
    Node* root;
    int threshold;

    void split(Node* node) {
        /* ??? */
    }

    void insert(Node*& node, int val) {
        /* ??? */
    }

    bool contains(Node* node, int val) {
        /* ??? */
    }

public:
    BTHashingSet(int t) : root(nullptr), threshold(t) {}
    void insert(int val) { insert(root, val); }
    bool contains(int val) { return contains(root, val); }
};

int main() {
    BTHashingSet tree(2);
    tree.insert(5);
    tree.insert(9);
    tree.insert(6);
    tree.insert(12);
    cout << tree.contains(5) << '\n';
    cout << tree.contains(7) << '\n';
    cout << tree.contains(12) << '\n';
    return 0;
}
```

```
1
0
1
```

Next task: Add a method to delete values.

Problem 3

You are given a tree rooted at 1. Each node in the tree has a value. Your task is to deal with the two given types of queries efficiently using an Euler Tour of the tree.

1. `1 s x` – change the value of node s to x.
2. `2 s` – print the sum of values of the nodes that lie on the shortest path from the root to node s.

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

class EnhancedVector {
private:
    int n, p;
    vector<int> arr;

public:
    EnhancedVector(int _n) {
        n = _n;
        p = 1;
        while (p < n) p *= 2;
        arr.resize(p * 2, 0);
    }

    void update(int i, int x) {
        int l = p + i, diff = x - arr[p + i];
        while (l) {
            arr[l] += diff;
            l /= 2;
        }
    }

    int rangeSum(int l, int r) {
        l += p, r += p;
        int sum = 0;
        while (l <= r) {
            if (l % 2) sum += arr[l++];
            if ((r % 2) == 0) sum += arr[r--];
```

```
            l /= 2;
            r /= 2;
        }
        return sum;
    }
};

void dfs(int x, int p, vector<vector<int>>& adj, vector<int>& vals, vector<int>& tour, vector<int>& first, vector<int>& second) {
    /* ??? */
}

int main() {
    int n, q;
    cin >> n >> q;
    vector<int> vals(n);
    for (int i = 0; i < n; i++) cin >> vals[i];

    vector<vector<int>> adj(n);
    for (int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        u--, v--;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    vector<int> tour, first(n), second(n);
    dfs(0, -1, adj, vals, tour, first, second);

    EnhancedVector* ev = new EnhancedVector(n * 2);
    for (int i = 0; i < (n * 2); i++) ev->update(i, tour[i]);

    while (q--) {
        int t;
        cin >> t;
        if (t == 1) {
            int s, x;
            cin >> s >> x;
            /* ??? */
        } else {
            int s;
            cin >> s;
            s--;
```

```
            /* ??? */
        }
    }

    return 0;
}
```

```
>> 5 3
>> 4 2 5 2 1
>> 1 2
>> 1 3
>> 3 4
>> 3 5
>> 2 4
>> 1 3 2
>> 2 4
11
8
```

Next task: Make use of the `EnhancedVector` data structure provided in the code to deal with each query in $O(\log n)$. The data structure can find the sum of any range of nodes in $O(\log n)$, but at the cost of also making updates in $O(\log n)$. As a hint, consider storing the Euler Tour using this data structure instead of a regular vector.

Problem 4

You are given k sorted vectors. Merge them into a single sorted vector efficiently using priority queues.

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

struct HeapNode {
    int value, row, index;
    HeapNode(int v, int r, int i) : value(v), row(r), index(i) {}
};

class MinHeap {
    HeapNode** heap;
    int capacity, size;

    void heapify(int i) {
        /* ??? */
    }

public:
    MinHeap(int cap) : capacity(cap), size(0) {
        heap = new HeapNode*[cap];
    }

    void insert(HeapNode* node) {
        int i = size++;
        heap[i] = node;
        while (i && heap[(i - 1) / 2]->value > heap[i]->value) {
            swap(heap[i], heap[(i - 1) / 2]);
            i = (i - 1) / 2;
        }
    }

    HeapNode* extractMin() {
        /* ??? */
    }

    bool isEmpty() { return size == 0; }
};
```

```
38
39 vector<int> mergeKSortedVectors(vector<vector<int>>& lists) {
40     /* ??? */
41 }
42
43 int main() {
44     int k;
45     cin >> k;
46     vector<vector<int>> lists(k);
47     for (int i = 0; i < k; i++) {
48         int size;
49         cin >> size;
50         lists[i].resize(size);
51         for (int j = 0; j < size; j++)
52             cin >> lists[i][j];
53     }
54     vector<int> result = mergeKSortedVectors(lists);
55     for (int i = 0; i < result.size(); i++) cout << result[i]
           << " \n"[i == result.size() - 1];
56     return 0;
57 }
```

```
1 >> 3
2 >> 3 1 4 7
3 >> 4 2 5 6 8
4 >> 2 3 9
5 1 2 3 4 5 6 7 8 9
```

Next task: Modify the code to merge k linked lists instead of vectors into a single linked list.

Problem 5

Write a function that constructs a binary tree using its preorder and inorder traversals and then prints its postorder. The function must be recursive.

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

struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int d) : data(d), left(nullptr), right(nullptr) {}
};

Node* buildTree(vector<int>& preorder, vector<int>& inorder,
   int inStart, int inEnd, int &preIndex) {
    /* ??? */
}

void postorder(Node* root) {
    /* ??? */
}

int main() {
    int n;
    cin >> n;
    vector<int> preorder(n), inorder(n);
    for (int i = 0; i < n; i++) cin >> preorder[i];
    for (int i = 0; i < n; i++) cin >> inorder[i];
    int preIndex = 0;
    Node* root = buildTree(preorder, inorder, 0, n - 1,
       preIndex);
    postorder(root);
    cout << '\n';
    return 0;
}
```

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

Next task: Modify the code to construct the BT using postorder and inorder traversals instead.