/tikz/,/tikz/graphs/
conversions/canvas coordinate/.code=1 , conversions/coordinate/.code=1

trees,layered

BÀI TOÁN DUYỆT CÂY (TREE TRAVERSAL)

Toán Tổ Hợp và Lý Thuyết Đồ Thị

1 Lý thuyết cơ bản về cây

1.1 Định nghĩa cây

Cây là một đồ thị liên thông không có chu trình. Cây có những tính chất đặc biệt:

- Với n đỉnh, cây có đúng $n - 1$ cạnh
- Giữa hai đỉnh bất kỳ có đúng một đường đi duy nhất
- Bỏ đi một cạnh bất kỳ sẽ làm cây trở thành đồ thị không liên thông
- Thêm một cạnh bất kỳ sẽ tạo ra chu trình

1.2 Cây có gốc (Rooted Tree)

Cây có gốc là cây được chọn một đỉnh làm gốc. Từ đó, các khái niệm được định nghĩa:

- **Gốc (Root)**: Đỉnh được chọn làm gốc
- **Cha (Parent)**: Đỉnh liền kề với đỉnh hiện tại và gần gốc hơn
- **Con (Child)**: Đỉnh liền kề với đỉnh hiện tại và xa gốc hơn
- **Lá (Leaf)**: Đỉnh không có con
- **Độ sâu (Depth)**: Khoảng cách từ gốc đến đỉnh
- **Chiều cao (Height)**: Độ sâu lớn nhất của cây

1.3 Cây nhị phân (Binary Tree)

Cây nhị phân là cây có gốc mà mỗi đỉnh có tối đa 2 con:

- **Con trái (Left child)**: Con bên trái
- **Con phải (Right child)**: Con bên phải
- **Cây nhị phân đầy đủ**: Mọi đỉnh không phải lá đều có 2 con
- **Cây nhị phân hoàn chỉnh**: Tất cả mức đều đầy, trừ mức cuối được điền từ trái sang phải

2 Mô tả bài toán

Đề bài: Viết chương trình C/C++, Python để duyệt cây theo 4 phương pháp:

1. **Preorder Traversal**: Duyệt trước - Gốc → Trái → Phải
2. **Postorder Traversal**: Duyệt sau - Trái → Phải → Gốc
3. **Top-down Traversal**: Duyệt từ trên xuống theo mức
4. **Bottom-up Traversal**: Duyệt từ dưới lên theo mức

Yêu cầu:

- Cài đặt cả phiên bản đệ quy và không đệ quy
- Xử lý được cây tổng quát và cây nhị phân
- Phân tích độ phức tạp thời gian và không gian
- Ứng dụng vào các bài toán thực tế

3 Ý tưởng và giải pháp

3.1 Phân tích bài toán

Duyệt cây là quá trình thăm tất cả các đỉnh của cây theo một thứ tự nhất định. Mỗi phương pháp duyệt có:

- **Mục đích khác nhau**: Tùy thuộc vào bài toán cụ thể
- **Thứ tự thăm khác nhau**: Ảnh hưởng đến kết quả
- **Cài đặt khác nhau**: Đệ quy hoặc sử dụng stack/queue
- **Ứng dụng khác nhau**: Parser, tính toán, tìm kiếm

3.2 Chiến lược giải quyết

1. **Thiết kế cấu trúc dữ liệu**: Node cho cây nhị phân và cây tổng quát
2. **Cài đặt đệ quy**: Đơn giản, dễ hiểu
3. **Cài đặt không đệ quy**: Sử dụng stack/queue, tránh stack overflow
4. **Tối ưu hóa**: Giảm thiểu bộ nhớ và thời gian thực thi

4 Thuật toán chi tiết

4.1 1. Preorder Traversal (Duyệt trước)

Thứ tự: **Gốc → Cây con trái → Cây con phải**

```
Algorithm: PreorderTraversal(root)
Input: root - goc cua cay
Output: Danh sach cac dinh theo thu tu preorder

// Phien ban de quy
Procedure PreorderRecursive(node):
1. If node == NULL: return
2. Visit(node)                      // Tham goc
3. PreorderRecursive(node.left)     // Duyet cay con trai
4. PreorderRecursive(node.right)    // Duyet cay con phai

// Phien ban khong de quy
Procedure PreorderIterative(root):
1. Initialize stack S
2. Push root to S
3. While S is not empty:
4.      node = S.pop()
5.      Visit(node)
6.      If node.right != NULL: S.push(node.right)  // Push phai truoc
7.      If node.left != NULL: S.push(node.left)    // Push trai sau

Time Complexity: O(n)
Space Complexity: O(h) - h la chieu cao cay
```

Listing 1: Thuật toán Preorder Traversal

4.2 2. Postorder Traversal (Duyệt sau)

Thứ tự: **Cây con trái → Cây con phải → Gốc**

```
Algorithm: PostorderTraversal(root)

// Phien ban de quy
Procedure PostorderRecursive(node):
1. If node == NULL: return
2. PostorderRecursive(node.left)    // Duyet cay con trai
3. PostorderRecursive(node.right)   // Duyet cay con phai
4. Visit(node)                      // Tham goc

// Phien ban khong de quy (su dung 2 stack)
Procedure PostorderIterative(root):
1. Initialize stack S1, S2
2. Push root to S1
3. While S1 is not empty:
4.      node = S1.pop()
5.      S2.push(node)
6.      If node.left != NULL: S1.push(node.left)
7.      If node.right != NULL: S1.push(node.right)
8. While S2 is not empty:
```

```
9.      node = S2.pop()
10.     Visit(node)

Time Complexity: O(n)
Space Complexity: O(h)
```

Listing 2: Thuật toán Postorder Traversal

4.3 3. Top-down Traversal (Duyệt từ trên xuống)

Duyệt theo mức từ gốc xuống lá (Level-order traversal):

```
Algorithm: TopDownTraversal(root)

Procedure TopDown(root):
1. Initialize queue Q
2. Enqueue root to Q
3. While Q is not empty:
4.      levelSize = Q.size()
5.      For i = 1 to levelSize:
6.          node = Q.dequeue()
7.          Visit(node)
8.          If node.left != NULL: Q.enqueue(node.left)
9.          If node.right != NULL: Q.enqueue(node.right)

// Bien the: Duyet theo tung muc rieng biet
Procedure TopDownByLevel(root):
1. Initialize queue Q
2. Enqueue root to Q
3. level = 0
4. While Q is not empty:
5.      levelSize = Q.size()
6.      Print("Level", level, ":")
7.      For i = 1 to levelSize:
8.          node = Q.dequeue()
9.          Visit(node)
10.         If node.left != NULL: Q.enqueue(node.left)
11.         If node.right != NULL: Q.enqueue(node.right)
12.     level++

Time Complexity: O(n)
Space Complexity: O(w) - w la chieu rong lon nhat cua cay
```

Listing 3: Thuật toán Top-down Traversal

4.4 4. Bottom-up Traversal (Duyệt từ dưới lên)

Duyệt theo mức từ lá lên gốc:

```
Algorithm: BottomUpTraversal(root)

// Cach 1: Su dung stack de dao nguoc top-down
Procedure BottomUpWithStack(root):
1. Initialize queue Q, stack S
2. Enqueue root to Q
```

```
3. While Q is not empty:
4.      levelSize = Q.size()
5.      levelNodes = []
6.      For i = 1 to levelSize:
7.          node = Q.dequeue()
8.          levelNodes.append(node)
9.          If node.left != NULL: Q.enqueue(node.left)
10.         If node.right != NULL: Q.enqueue(node.right)
11.     S.push(levelNodes)
12. While S is not empty:
13.     levelNodes = S.pop()
14.     For each node in levelNodes:
15.         Visit(node)

// Cach 2: Su dung vector 2 chieu
Procedure BottomUpWithVector(root):
1. levels = []
2. TopDownByLevelToVector(root, 0, levels)
3. For i = levels.size()-1 down to 0:
4.      For each node in levels[i]:
5.          Visit(node)

Time Complexity: O(n)
Space Complexity: O(n)
```

Listing 4: Thuật toán Bottom-up Traversal

5 Code C++

```cpp
#include <iostream>
#include <vector>
#include <stack>
#include <queue>
#include <algorithm>

using namespace std;

// Cau truc node cho cay nhi phan
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
};

// Cau truc node cho cay tong quat
struct GeneralTreeNode {
    int data;
    vector<GeneralTreeNode*> children;

    GeneralTreeNode(int val) : data(val) {}
};

```

```
26 class TreeTraversal {
27 public:
28     // ===== PREORDER TRAVERSAL =====
29 
30     // De quy
31     void preorderRecursive(TreeNode* root, vector<int>& result) {
32         if (root == nullptr) return;
33 
34         result.push_back(root->data);           // Tham goc
35         preorderRecursive(root->left, result);  // Duyet trai
36         preorderRecursive(root->right, result); // Duyet phai
37     }
38 
39     // Khong de quy
40     vector<int> preorderIterative(TreeNode* root) {
41         vector<int> result;
42         if (root == nullptr) return result;
43 
44         stack<TreeNode*> stk;
45         stk.push(root);
46 
47         while (!stk.empty()) {
48             TreeNode* node = stk.top();
49             stk.pop();
50 
51             result.push_back(node->data);
52 
53             // Push phai truoc, trai sau (vi stack la LIFO)
54             if (node->right) stk.push(node->right);
55             if (node->left) stk.push(node->left);
56         }
57 
58         return result;
59     }
60 
61     // ===== POSTORDER TRAVERSAL =====
62 
63     // De quy
64     void postorderRecursive(TreeNode* root, vector<int>& result) {
65         if (root == nullptr) return;
66 
67         postorderRecursive(root->left, result);  // Duyet trai
68         postorderRecursive(root->right, result); // Duyet phai
69         result.push_back(root->data);            // Tham goc
70     }
71 
72     // Khong de quy - Phuong phap 2 stack
73     vector<int> postorderIterative(TreeNode* root) {
74         vector<int> result;
75         if (root == nullptr) return result;
76 
77         stack<TreeNode*> stk1, stk2;
78         stk1.push(root);
79 
80         // Stack 1: duyet va day vao stack 2
81         while (!stk1.empty()) {
```

```
            TreeNode* node = stk1.top();
            stk1.pop();
            stk2.push(node);

            if (node->left) stk1.push(node->left);
            if (node->right) stk1.push(node->right);
        }

        // Stack 2: lay ra theo thu tu postorder
        while (!stk2.empty()) {
            result.push_back(stk2.top()->data);
            stk2.pop();
        }

        return result;
    }

    // Khong de quy - Phuong phap 1 stack voi flag
    vector<int> postorderIterativeSingleStack(TreeNode* root) {
        vector<int> result;
        if (root == nullptr) return result;

        stack<pair<TreeNode*, bool>> stk; // bool: da xu ly con hay chua
        stk.push({root, false});

        while (!stk.empty()) {
            auto [node, processed] = stk.top();
            stk.pop();

            if (processed) {
                result.push_back(node->data);
            } else {
                // Danh dau da xu ly va push lai
                stk.push({node, true});
                // Push con phai truoc, trai sau
                if (node->right) stk.push({node->right, false});
                if (node->left) stk.push({node->left, false});
            }
        }

        return result;
    }

    // ===== TOP-DOWN TRAVERSAL =====

    vector<vector<int>> topDownTraversal(TreeNode* root) {
        vector<vector<int>> result;
        if (root == nullptr) return result;

        queue<TreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            int levelSize = q.size();
            vector<int> currentLevel;

```

```
            for (int i = 0; i < levelSize; i++) {
                TreeNode* node = q.front();
                q.pop();

                currentLevel.push_back(node->data);

                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }

            result.push_back(currentLevel);
        }

        return result;
    }

    // Phien ban chi tra ve 1 vector
    vector<int> topDownFlat(TreeNode* root) {
        vector<int> result;
        if (root == nullptr) return result;

        queue<TreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            TreeNode* node = q.front();
            q.pop();

            result.push_back(node->data);

            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }

        return result;
    }

    // ===== BOTTOM-UP TRAVERSAL =====

    vector<vector<int>> bottomUpTraversal(TreeNode* root) {
        vector<vector<int>> levels = topDownTraversal(root);
        reverse(levels.begin(), levels.end());
        return levels;
    }

    vector<int> bottomUpFlat(TreeNode* root) {
        vector<int> result = topDownFlat(root);

        // Chuyen doi thanh bottom-up bang cach group theo level roi
    reverse
        vector<vector<int>> levels = topDownTraversal(root);
        result.clear();

        for (int i = levels.size() - 1; i >= 0; i--) {
            for (int val : levels[i]) {
                result.push_back(val);
```

```
            }
        }

        return result;
    }

    // ===== CAY TONG QUAT =====

    // Preorder cho cay tong quat
    void preorderGeneral(GeneralTreeNode* root, vector<int>& result) {
        if (root == nullptr) return;

        result.push_back(root->data);
        for (GeneralTreeNode* child : root->children) {
            preorderGeneral(child, result);
        }
    }

    // Postorder cho cay tong quat
    void postorderGeneral(GeneralTreeNode* root, vector<int>& result) {
        if (root == nullptr) return;

        for (GeneralTreeNode* child : root->children) {
            postorderGeneral(child, result);
        }
        result.push_back(root->data);
    }

    // Level order cho cay tong quat
    vector<vector<int>> levelOrderGeneral(GeneralTreeNode* root) {
        vector<vector<int>> result;
        if (root == nullptr) return result;

        queue<GeneralTreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            int levelSize = q.size();
            vector<int> currentLevel;

            for (int i = 0; i < levelSize; i++) {
                GeneralTreeNode* node = q.front();
                q.pop();

                currentLevel.push_back(node->data);

                for (GeneralTreeNode* child : node->children) {
                    q.push(child);
                }
            }

            result.push_back(currentLevel);
        }

        return result;
    }
```

```
249
250     // ===== UTILITY FUNCTIONS =====
251
252     // Tao cay mau
253     TreeNode* createSampleTree() {
254         /*
255                 1
256                / \
257               2   3
258              / \   \
259             4   5   6
260         */
261         TreeNode* root = new TreeNode(1);
262         root->left = new TreeNode(2);
263         root->right = new TreeNode(3);
264         root->left->left = new TreeNode(4);
265         root->left->right = new TreeNode(5);
266         root->right->right = new TreeNode(6);
267         return root;
268     }
269
270     GeneralTreeNode* createSampleGeneralTree() {
271         /*
272                 1
273               / | \
274              2  3  4
275             /|     |\
276            5 6     7 8
277         */
278         GeneralTreeNode* root = new GeneralTreeNode(1);
279         root->children.push_back(new GeneralTreeNode(2));
280         root->children.push_back(new GeneralTreeNode(3));
281         root->children.push_back(new GeneralTreeNode(4));
282
283         root->children[0]->children.push_back(new GeneralTreeNode(5));
284         root->children[0]->children.push_back(new GeneralTreeNode(6));
285
286         root->children[2]->children.push_back(new GeneralTreeNode(7));
287         root->children[2]->children.push_back(new GeneralTreeNode(8));
288
289         return root;
290     }
291
292     // In ket qua
293     void printVector(const vector<int>& vec, const string& name) {
294         cout << name << ": ";
295         for (int val : vec) {
296             cout << val << " ";
297         }
298         cout << endl;
299     }
300
301     void printLevels(const vector<vector<int>>& levels, const string&
    name) {
302         cout << name << ":" << endl;
303         for (int i = 0; i < levels.size(); i++) {
```

```
            cout << "Level " << i << ": ";
            for (int val : levels[i]) {
                cout << val << " ";
            }
            cout << endl;
        }
    }

    // Tinh chieu cao cay
    int getHeight(TreeNode* root) {
        if (root == nullptr) return 0;
        return 1 + max(getHeight(root->left), getHeight(root->right));
    }

    // Dem so node
    int countNodes(TreeNode* root) {
        if (root == nullptr) return 0;
        return 1 + countNodes(root->left) + countNodes(root->right);
    }

    // Kiem tra cay co can bang hay khong
    bool isBalanced(TreeNode* root) {
        return checkBalance(root) != -1;
    }

private:
    int checkBalance(TreeNode* root) {
        if (root == nullptr) return 0;

        int leftHeight = checkBalance(root->left);
        if (leftHeight == -1) return -1;

        int rightHeight = checkBalance(root->right);
        if (rightHeight == -1) return -1;

        if (abs(leftHeight - rightHeight) > 1) return -1;

        return 1 + max(leftHeight, rightHeight);
    }
};

// Ham test toan bo
int main() {
    TreeTraversal traverser;

    cout << "=== DEMO DUYET CAY NHI PHAN ===" << endl;

    // Tao cay mau
    TreeNode* root = traverser.createSampleTree();

    cout << "Cay mau:" << endl;
    cout << "        1" << endl;
    cout << "       / \\" << endl;
    cout << "      2   3" << endl;
    cout << "     / \\   \\" << endl;
    cout << "    4   5   6" << endl;
```

```
    cout << endl;

    // Test preorder
    vector<int> preResult;
    traverser.preorderRecursive(root, preResult);
    traverser.printVector(preResult, "Preorder (De quy)");

    vector<int> preIter = traverser.preorderIterative(root);
    traverser.printVector(preIter, "Preorder (Khong de quy)");

    // Test postorder
    vector<int> postResult;
    traverser.postorderRecursive(root, postResult);
    traverser.printVector(postResult, "Postorder (De quy)");

    vector<int> postIter = traverser.postorderIterative(root);
    traverser.printVector(postIter, "Postorder (2 stack)");

    vector<int> postSingle = traverser.postorderIterativeSingleStack(
root);
    traverser.printVector(postSingle, "Postorder (1 stack)");

    // Test level order
    vector<vector<int>> topDown = traverser.topDownTraversal(root);
    traverser.printLevels(topDown, "Top-down traversal");

    vector<vector<int>> bottomUp = traverser.bottomUpTraversal(root);
    traverser.printLevels(bottomUp, "Bottom-up traversal");

    cout << endl;

    // Thong tin cay
    cout << "=== THONG TIN CAY ===" << endl;
    cout << "Chieu cao: " << traverser.getHeight(root) << endl;
    cout << "So node: " << traverser.countNodes(root) << endl;
    cout << "Can bang: " << (traverser.isBalanced(root) ? "Co" : "Khong"
) << endl;

    cout << endl;

    // Test cay tong quat
    cout << "=== DEMO DUYET CAY TONG QUAT ===" << endl;

    GeneralTreeNode* genRoot = traverser.createSampleGeneralTree();

    cout << "Cay tong quat mau:" << endl;
    cout << "         1" << endl;
    cout << "       / | \\" << endl;
    cout << "      2  3  4" << endl;
    cout << "     /|     |\\" << endl;
    cout << "    5 6     7 8" << endl;
    cout << endl;

    vector<int> genPre, genPost;
    traverser.preorderGeneral(genRoot, genPre);
    traverser.postorderGeneral(genRoot, genPost);
```

```cpp

    traverser.printVector(genPre, "Preorder cay tong quat");
    traverser.printVector(genPost, "Postorder cay tong quat");

    vector<vector<int>> genLevels = traverser.levelOrderGeneral(genRoot);
    traverser.printLevels(genLevels, "Level order cay tong quat");

    return 0;
}
```

Listing 5: Cài đặt đầy đủ bằng C++

6 Code Python

```python
from collections import deque
from typing import List, Optional

class TreeNode:
    """Node cho cay nhi phan"""
    def __init__(self, val: int = 0):
        self.val = val
        self.left: Optional['TreeNode'] = None
        self.right: Optional['TreeNode'] = None

class GeneralTreeNode:
    """Node cho cay tong quat"""
    def __init__(self, val: int = 0):
        self.val = val
        self.children: List['GeneralTreeNode'] = []

class TreeTraversal:
    """Lop xu ly cac phuong phap duyet cay"""

    # ===== PREORDER TRAVERSAL =====

    def preorder_recursive(self, root: Optional[TreeNode]) -> List[int]:
        """
        Duyet preorder bang de quy
        Thu tu: Goc -> Trai -> Phai
        """
        result = []

        def dfs(node):
            if not node:
                return
            result.append(node.val)      # Tham goc
            dfs(node.left)             # Duyet trai
            dfs(node.right)            # Duyet phai

        dfs(root)
        return result

    def preorder_iterative(self, root: Optional[TreeNode]) -> List[int]:
```

```
        """
        Duyet preorder khong de quy bang stack
        """
        if not root:
            return []

        result = []
        stack = [root]

        while stack:
            node = stack.pop()
            result.append(node.val)

            # Push phai truoc, trai sau (vi stack la LIFO)
            if node.right:
                stack.append(node.right)
            if node.left:
                stack.append(node.left)

        return result

    # ===== POSTORDER TRAVERSAL =====

    def postorder_recursive(self, root: Optional[TreeNode]) -> List[int
]:
        """
        Duyet postorder bang de quy
        Thu tu: Trai -> Phai -> Goc
        """
        result = []

        def dfs(node):
            if not node:
                return
            dfs(node.left)              # Duyet trai
            dfs(node.right)             # Duyet phai
            result.append(node.val)     # Tham goc

        dfs(root)
        return result

    def postorder_iterative_two_stacks(self, root: Optional[TreeNode])
-> List[int]:
        """
        Duyet postorder khong de quy bang 2 stack
        """
        if not root:
            return []

        stack1 = [root]
        stack2 = []
        result = []

        # Stack 1: duyet va day vao stack 2
        while stack1:
            node = stack1.pop()
```

```
            stack2.append(node)

            if node.left:
                stack1.append(node.left)
            if node.right:
                stack1.append(node.right)

        # Stack 2: lay ra theo thu tu postorder
        while stack2:
            result.append(stack2.pop().val)

        return result

    def postorder_iterative_one_stack(self, root: Optional[TreeNode]) ->
    List[int]:
        """
        Duyet postorder khong de quy bang 1 stack voi flag
        """
        if not root:
            return []

        result = []
        stack = [(root, False)]  # (node, processed)

        while stack:
            node, processed = stack.pop()

            if processed:
                result.append(node.val)
            else:
                # Danh dau da xu ly va push lai
                stack.append((node, True))
                # Push con phai truoc, trai sau
                if node.right:
                    stack.append((node.right, False))
                if node.left:
                    stack.append((node.left, False))

        return result

    # ===== TOP-DOWN TRAVERSAL =====

    def level_order_traversal(self, root: Optional[TreeNode]) -> List[
    List[int]]:
        """
        Duyet theo muc tu tren xuong (Level order)
        """
        if not root:
            return []

        result = []
        queue = deque([root])

        while queue:
            level_size = len(queue)
            current_level = []
```

```
148
149             for _ in range(level_size):
150                 node = queue.popleft()
151                 current_level.append(node.val)
152
153                 if node.left:
154                     queue.append(node.left)
155                 if node.right:
156                     queue.append(node.right)
157
158             result.append(current_level)
159
160         return result
161
162     def top_down_flat(self, root: Optional[TreeNode]) -> List[int]:
163         """
164         Duyet top-down tra ve 1 list phang
165         """
166         if not root:
167             return []
168
169         result = []
170         queue = deque([root])
171
172         while queue:
173             node = queue.popleft()
174             result.append(node.val)
175
176             if node.left:
177                 queue.append(node.left)
178             if node.right:
179                 queue.append(node.right)
180
181         return result
182
183     # ===== BOTTOM-UP TRAVERSAL =====
184
185     def bottom_up_traversal(self, root: Optional[TreeNode]) -> List[List
    [int]]:
186         """
187         Duyet theo muc tu duoi len
188         """
189         levels = self.level_order_traversal(root)
190         return levels[::-1]  # Dao nguoc
191
192     def bottom_up_flat(self, root: Optional[TreeNode]) -> List[int]:
193         """
194         Duyet bottom-up tra ve 1 list phang
195         """
196         levels = self.level_order_traversal(root)
197         result = []
198
199         for level in reversed(levels):
200             result.extend(level)
201
202         return result
```

```
203
204     # ===== CAY TONG QUAT =====
205
206     def preorder_general(self, root: Optional[GeneralTreeNode]) -> List[
    int]:
207         """
208         Duyet preorder cho cay tong quat
209         """
210         result = []
211
212         def dfs(node):
213             if not node:
214                 return
215             result.append(node.val)
216             for child in node.children:
217                 dfs(child)
218
219         dfs(root)
220         return result
221
222     def postorder_general(self, root: Optional[GeneralTreeNode]) -> List
    [int]:
223         """
224         Duyet postorder cho cay tong quat
225         """
226         result = []
227
228         def dfs(node):
229             if not node:
230                 return
231             for child in node.children:
232                 dfs(child)
233             result.append(node.val)
234
235         dfs(root)
236         return result
237
238     def level_order_general(self, root: Optional[GeneralTreeNode]) ->
    List[List[int]]:
239         """
240         Duyet level order cho cay tong quat
241         """
242         if not root:
243             return []
244
245         result = []
246         queue = deque([root])
247
248         while queue:
249             level_size = len(queue)
250             current_level = []
251
252             for _ in range(level_size):
253                 node = queue.popleft()
254                 current_level.append(node.val)
255
```

```
256                 for child in node.children:
257                     queue.append(child)
258 
259             result.append(current_level)
260 
261         return result
262 
263     # ===== UTILITY FUNCTIONS =====
264 
265     def create_sample_tree(self) -> TreeNode:
266         """
267         Tao cay mau:
268                 1
269               / \
270             2   3
271            / \   \
272           4   5   6
273         """
274         root = TreeNode(1)
275         root.left = TreeNode(2)
276         root.right = TreeNode(3)
277         root.left.left = TreeNode(4)
278         root.left.right = TreeNode(5)
279         root.right.right = TreeNode(6)
280         return root
281 
282     def create_sample_general_tree(self) -> GeneralTreeNode:
283         """
284         Tao cay tong quat mau:
285                 1
286               / | \
287              2  3  4
288             /|     |\
289            5 6     7 8
290         """
291         root = GeneralTreeNode(1)
292         root.children = [GeneralTreeNode(2), GeneralTreeNode(3),
    GeneralTreeNode(4)]
293 
294         root.children[0].children = [GeneralTreeNode(5), GeneralTreeNode
    (6)]
295         root.children[2].children = [GeneralTreeNode(7), GeneralTreeNode
    (8)]
296 
297         return root
298 
299     def get_height(self, root: Optional[TreeNode]) -> int:
300         """Tinh chieu cao cay"""
301         if not root:
302             return 0
303         return 1 + max(self.get_height(root.left), self.get_height(root.
    right))
304 
305     def count_nodes(self, root: Optional[TreeNode]) -> int:
306         """Dem so node trong cay"""
307         if not root:
```

```
            return 0
        return 1 + self.count_nodes(root.left) + self.count_nodes(root.
    right)

    def is_balanced(self, root: Optional[TreeNode]) -> bool:
        """Kiem tra cay co can bang hay khong"""
        def check_balance(node):
            if not node:
                return 0

            left_height = check_balance(node.left)
            if left_height == -1:
                return -1

            right_height = check_balance(node.right)
            if right_height == -1:
                return -1

            if abs(left_height - right_height) > 1:
                return -1

            return 1 + max(left_height, right_height)

        return check_balance(root) != -1

def print_list(lst, name):
    """In danh sach"""
    print(f"{name}: {lst}")

def print_levels(levels, name):
    """In cac muc"""
    print(f"{name}:")
    for i, level in enumerate(levels):
        print(f"  Level {i}: {level}")

# Ham test chinh
def main():
    traverser = TreeTraversal()

    print("=== DEMO DUYET CAY NHI PHAN ===")

    # Tao cay mau
    root = traverser.create_sample_tree()

    print("Cay mau:")
    print("        1")
    print("       / \\")
    print("      2   3")
    print("     / \\   \\")
    print("    4   5   6")
    print()

    # Test preorder
    pre_rec = traverser.preorder_recursive(root)
    pre_iter = traverser.preorder_iterative(root)
    print_list(pre_rec, "Preorder (De quy)")
```

```
363     print_list(pre_iter, "Preorder (Khong de quy)")
364 
365     # Test postorder
366     post_rec = traverser.postorder_recursive(root)
367     post_two = traverser.postorder_iterative_two_stacks(root)
368     post_one = traverser.postorder_iterative_one_stack(root)
369     print_list(post_rec, "Postorder (De quy)")
370     print_list(post_two, "Postorder (2 stack)")
371     print_list(post_one, "Postorder (1 stack)")
372 
373     # Test level order
374     top_down = traverser.level_order_traversal(root)
375     bottom_up = traverser.bottom_up_traversal(root)
376     print_levels(top_down, "Top-down traversal")
377     print_levels(bottom_up, "Bottom-up traversal")
378 
379     print()
380 
381     # Thong tin cay
382     print("=== THONG TIN CAY ===")
383     print(f"Chieu cao: {traverser.get_height(root)}")
384     print(f"So node: {traverser.count_nodes(root)}")
385     print(f"Can bang: {'Co' if traverser.is_balanced(root) else 'Khong'}
    ")
386 
387     print()
388 
389     # Test cay tong quat
390     print("=== DEMO DUYET CAY TONG QUAT ===")
391 
392     gen_root = traverser.create_sample_general_tree()
393 
394     print("Cay tong quat mau:")
395     print("         1")
396     print("       / | \\")
397     print("      2  3  4")
398     print("     /|     |\\")
399     print("    5 6     7 8")
400     print()
401 
402     gen_pre = traverser.preorder_general(gen_root)
403     gen_post = traverser.postorder_general(gen_root)
404     gen_levels = traverser.level_order_general(gen_root)
405 
406     print_list(gen_pre, "Preorder cay tong quat")
407     print_list(gen_post, "Postorder cay tong quat")
408     print_levels(gen_levels, "Level order cay tong quat")
409 
410 if __name__ == "__main__":
411     main()
```

Listing 6: Cài đặt đầy đủ bằng Python

7 Phân tích độ phức tạp

7.1 Độ phức tạp thời gian

Tất cả các phương pháp duyệt cây đều có độ phức tạp thời gian **O(n)**, trong đó n là số đỉnh của cây, vì:

- Mỗi đỉnh được thăm đúng một lần
- Thao tác xử lý tại mỗi đỉnh mất O(1)
- Tổng thời gian là $O(n \times 1) = O(n)$

7.2 Độ phức tạp không gian

- **Phiên bản đệ quy**: O(h) - với h là chiều cao cây
  - Trường hợp tốt nhất (cây cân bằng): O(log n)
  - Trường hợp xấu nhất (cây suy biến): O(n)
- **Phiên bản không đệ quy**:
  - Preorder, Postorder: O(h) cho stack
  - Level order: O(w) với w là độ rộng lớn nhất của cây
  - Trường hợp xấu nhất: $O(n/2) = O(n)$ cho cây hoàn chỉnh

8 Ứng dụng thực tế

8.1 1. Preorder Traversal

- **Sao chép cây**: Tạo bản sao của cây
- **Tiền tố biểu thức**: Chuyển đổi biểu thức sang dạng prefix
- **Serialize cây**: Chuyển cây thành chuỗi để lưu trữ
- **Directory listing**: Duyệt thư mục từ gốc

8.2 2. Postorder Traversal

- **Xóa cây**: Giải phóng bộ nhớ từ lá lên gốc
- **Tính toán biểu thức**: Evaluate expression tree
- **Tính kích thước thư mục**: Tính dung lượng từ file lên thư mục cha
- **Hậu tố biểu thức**: Chuyển đổi sang dạng postfix

8.3 3. Level Order Traversal

- **In cây theo mức**: Hiển thị cấu trúc cây rõ ràng
- **Tìm kiếm theo chiều rộng**: BFS trên cây
- **Tìm mức có tổng lớn nhất**: Xử lý từng mức riêng biệt
- **Xây dựng cây hoàn chỉnh**: Chèn node theo thứ tự level

8.4 4. Bottom-up Traversal

- **Tính toán từ lá lên gốc**: Dynamic programming trên cây
- **Aggregation**: Tổng hợp thông tin từ con lên cha
- **Phân tích cây phân tách**: Bottom-up parsing
- **Tính toán độ sâu**: Xác định mức sâu của mỗi node

9 Kết luận

Duyệt cây là một kỹ thuật cơ bản và quan trọng trong khoa học máy tính. Việc hiểu rõ các phương pháp duyệt khác nhau giúp:

- **Chọn thuật toán phù hợp**: Tùy theo bài toán cụ thể
- **Tối ưu hóa hiệu suất**: Sử dụng đúng cách để tiết kiệm tài nguyên
- **Giải quyết bài toán phức tạp**: Là nền tảng cho nhiều thuật toán khác
- **Hiểu sâu cấu trúc dữ liệu**: Nắm vững bản chất của cây

Trong thực tế, việc lựa chọn phương pháp duyệt phụ thuộc vào:

- Yêu cầu về thứ tự xử lý
- Giới hạn về bộ nhớ
- Cấu trúc của cây (cân bằng hay không)
- Tần suất sử dụng