LÝ THUYẾT ĐỒ THỊ VÀ GIẢI BÀI TẬP

Course Report

Ngày 23 tháng 7 năm 2025

Mục lục

1 Lý thuyết cơ bản về Đồ thị

1.1 Định nghĩa cơ bản

Đồ thị vô hướng $G = (V, E)$ bao gồm:

- V: tập hợp các đỉnh (vertices)
- E: tập hợp các cạnh (edges), mỗi cạnh nối hai đỉnh

Đồ thị đầy đủ K_n: đồ thị có n đỉnh và mọi cặp đỉnh đều được nối với nhau bởi một cạnh.
Đồ thị hai phần đầy đủ $K_{p,q}$: đồ thị có hai tập đỉnh rời nhau A (có p đỉnh) và B (có q đỉnh), mọi đỉnh trong A đều nối với mọi đỉnh trong B.
Đồ thị vòng C_n: đồ thị có n đỉnh được sắp xếp thành một vòng tròn, mỗi đỉnh nối với hai đỉnh kề cận.

1.2 Tính chất quan trọng

- Số cạnh của K_n: $\binom{n}{2} = \frac{n(n-1)}{2}$
- Số cạnh của $K_{p,q}$: $p \times q$
- Số cạnh của C_n: n
- Đồ thị hai phần: đồ thị có thể tô màu với 2 màu sao cho không có hai đỉnh kề cận nào cùng màu

2 Bài toán 1.1: Kích thước đồ thị đầy đủ và đồ thị hai phần đầy đủ

2.1 Mô tả bài toán

Xác định kích thước (số cạnh) của đồ thị đầy đủ K_n trên n đỉnh và đồ thị hai phần đầy đủ $K_{p,q}$ trên $p + q$ đỉnh.

2.2 Lý thuyết và giải pháp

Đồ thị đầy đủ K_n:

- Mỗi đỉnh nối với $(n - 1)$ đỉnh khác
- Tổng số cạnh: $\frac{n(n-1)}{2}$ (chia 2 vì mỗi cạnh được tính 2 lần)

Đồ thị hai phần đầy đủ $K_{p,q}$:

- Tập đỉnh được chia thành 2 phần: A (có p đỉnh) và B (có q đỉnh)
- Mỗi đỉnh trong A nối với tất cả q đỉnh trong B
- Tổng số cạnh: $p \times q$

2.3 Code minh họa

Listing 1: Tính số cạnh của đồ thị

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

// Tinh so canh cua do thi day du K_n
long long complete_graph_edges(int n) {
    return (long long)n * (n - 1) / 2;
}

// Tinh so canh cua do thi hai phan day du K_{p,q}
long long bipartite_complete_graph_edges(int p, int q) {
    return (long long)p * q;
}

int main() {
    int n, p, q;

    cout << "Nhap n cho do thi day du K_n: ";
    cin >> n;
    cout << "So canh cua K_" << n << " la: "
         << complete_graph_edges(n) << endl;

    cout << "Nhap p va q cho do thi hai phan K_{p,q}: ";
    cin >> p >> q;
    cout << "So canh cua K_" << p << "," << q << " la: "
         << bipartite_complete_graph_edges(p, q) << endl;

    return 0;
}
```

3 Bài toán 1.2: Tính hai phần của đồ thị vòng và đồ thị đầy đủ

3.1 Mô tả bài toán

Xác định giá trị của n để đồ thị vòng C_n và đồ thị đầy đủ K_n là đồ thị hai phần.

3.2 Lý thuyết

Định lý: Một đồ thị là hai phần khi và chỉ khi nó không chứa chu trình có độ dài lẻ.

Đối với đồ thị vòng C_n:

- C_n chỉ có một chu trình duy nhất có độ dài n
- C_n là hai phần khi và chỉ khi n chẵn

Đối với đồ thị đầy đủ K_n:

- Với $n \geq 3$: K_n chứa tam giác (chu trình độ dài 3)

- K_1: không có cạnh nào → hai phần
- K_2: chỉ có một cạnh → hai phần
- K_n với $n \geq 3$: không phải hai phần

3.3 Giải pháp

Kết luận:

- C_n là hai phần khi và chỉ khi n chẵn
- K_n là hai phần khi và chỉ khi $n \leq 2$

Listing 2: Kiểm tra tính hai phần

```python
def is_cycle_bipartite(n):
    """Kiem tra xem do thi vong C_n co phai la hai phan khong"""
    return n % 2 == 0

def is_complete_bipartite(n):
    """Kiem tra xem do thi day du K_n co phai la hai phan khong"""
    return n <= 2

def check_bipartite_using_coloring(adj_list):
    """Kiem tra tinh hai phan bang thuat toan to mau DFS"""
    n = len(adj_list)
    color = [-1] * n  # -1: chaa to, 0: mau do, 1: mau xanh

    def dfs(v, c):
        color[v] = c
        for u in adj_list[v]:
            if color[u] == -1:
                if not dfs(u, 1 - c):
                    return False
            elif color[u] == c:
                return False
        return True

    for i in range(n):
        if color[i] == -1:
            if not dfs(i, 0):
                return False
    return True

# Test
print("C_4 la hai phan:", is_cycle_bipartite(4))  # True
print("C_5 la hai phan:", is_cycle_bipartite(5))  # False
print("K_2 la hai phan:", is_complete_bipartite(2))  # True
print("K_3 la hai phan:", is_complete_bipartite(3))  # False
```

4 Bài toán 1.3: Cây khung (Spanning Tree)

4.1 Mô tả bài toán

Tìm tất cả các cây khung của đồ thị trong Hình 1.30 và số lượng cây khung của đồ thị vô hướng tương ứng.

4.2 Lý thuyết về cây khung

Cây khung của đồ thị liên thông $G = (V, E)$ là đồ thị con $T = (V, E')$ sao cho:

- T là một cây (liên thông và không có chu trình)
- T chứa tất cả các đỉnh của G
- $|E'| = |V| - 1$

Định lý Matrix-Tree: Số lượng cây khung của đồ thị G bằng bất kỳ cofactor nào của ma trận Laplacian $L = D - A$, trong đó:

- D: ma trận bậc (diagonal matrix)
- A: ma trận kề

4.3 Thuật toán tìm cây khung

Algorithm 1 Thuật toán Kruskal tìm cây khung tối tiểu

```
Sắp xếp các cạnh theo trọng số không giảm
Khởi tạo forest với mỗi đỉnh là một cây riêng biệt
for mỗi cạnh (u, v) trong danh sách đã sắp xếp do
    if u và v thuộc các cây khác nhau then
        Thêm cạnh (u, v) vào cây khung
        Hợp hai cây chứa u và v
    end if
end for
```

Listing 3: Thuật toán tìm tất cả cây khung

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

class SpanningTreeFinder {
private:
    int n;
    vector<vector<int>> adj;
    vector<vector<int>> spanning_trees;

public:
    SpanningTreeFinder(int vertices) : n(vertices) {
        adj.resize(n);
```

```
    }

    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    // Tim tat ca cay khung bang backtracking
    void find_all_spanning_trees() {
        vector<pair<int,int>> edges;

        // Thu thap tat ca cac canh
        for (int i = 0; i < n; i++) {
            for (int j : adj[i]) {
                if (i < j) {  // Tranh trung lap
                    edges.push_back({i, j});
                }
            }
        }

        vector<int> current_tree;
        vector<bool> used(edges.size(), false);

        backtrack(edges, current_tree, used, 0);
    }

private:
    void backtrack(const vector<pair<int,int>>& edges,
                   vector<int>& current_tree,
                   vector<bool>& used, int start) {

        if (current_tree.size() == n - 1) {
            // Kiem tra xem co tao thanh cay khong
            if (is_connected(current_tree, edges)) {
                spanning_trees.push_back(current_tree);
            }
            return;
        }

        for (int i = start; i < edges.size(); i++) {
            if (!used[i]) {
                current_tree.push_back(i);
                used[i] = true;

                // Kiem tra khong tao thanh chu trinh
                if (!has_cycle(current_tree, edges)) {
                    backtrack(edges, current_tree, used, i + 1);
                }

                current_tree.pop_back();
                used[i] = false;
            }
        }
```

```
68      }
69
70      bool is_connected(const vector<int>& tree_edges,
71                        const vector<pair<int,int>>& all_edges) {
72          // Su dung DFS de kiem tra tinh lien thong
73          vector<vector<int>> tree_adj(n);
74
75          for (int edge_idx : tree_edges) {
76              int u = all_edges[edge_idx].first;
77              int v = all_edges[edge_idx].second;
78              tree_adj[u].push_back(v);
79              tree_adj[v].push_back(u);
80          }
81
82          vector<bool> visited(n, false);
83          dfs(0, tree_adj, visited);
84
85          for (bool v : visited) {
86              if (!v) return false;
87          }
88          return true;
89      }
90
91      void dfs(int v, const vector<vector<int>>& graph, vector<bool>&
           visited) {
92          visited[v] = true;
93          for (int u : graph[v]) {
94              if (!visited[u]) {
95                  dfs(u, graph, visited);
96              }
97          }
98      }
99
100     bool has_cycle(const vector<int>& tree_edges,
101                    const vector<pair<int,int>>& all_edges) {
102         // Su dung Union-Find de kiem tra chu trinh
103         vector<int> parent(n);
104         for (int i = 0; i < n; i++) parent[i] = i;
105
106         function<int(int)> find = [&](int x) {
107             return parent[x] == x ? x : parent[x] = find(parent[x]);
108         };
109
110         for (int edge_idx : tree_edges) {
111             int u = all_edges[edge_idx].first;
112             int v = all_edges[edge_idx].second;
113
114             int pu = find(u), pv = find(v);
115             if (pu == pv) return true;
116             parent[pu] = pv;
117         }
118         return false;
119     }
```

```
120
121 public:
122     void print_spanning_trees() {
123         cout << "Tong so cay khung: " << spanning_trees.size() << endl
                ;
124         for (int i = 0; i < spanning_trees.size(); i++) {
125             cout << "Cay khung " << (i+1) << ": ";
126             for (int edge_idx : spanning_trees[i]) {
127                 cout << "(" << edge_idx << ") ";
128             }
129             cout << endl;
130         }
131     }
132 };
```

5 Bài toán 1.4: Mở rộng biểu diễn ma trận kề

5.1 Mô tả bài toán

Mở rộng biểu diễn ma trận kề bằng cách thay thế các thao tác có cạnh làm đối số hoặc trả về cạnh bằng các thao tác tương ứng với đỉnh nguồn và đỉnh đích.

5.2 Thiết kế cấu trúc dữ liệu

Listing 4: Lớp Graph với biểu diễn ma trận kề mở rộng

```
1  #include <iostream>
2  #include <vector>
3  #include <utility>
4  using namespace std;
5
6  class Graph {
7  private:
8      int n;  // So dinh
9      vector<vector<bool>> adj_matrix;  // Ma tran ke
10
11 public:
12     Graph(int vertices) : n(vertices) {
13         adj_matrix.resize(n, vector<bool>(n, false));
14     }
15
16     // Them canh bang cach chi dinh dinh nguon va dich
17     void add_edge(int source, int target) {
18         if (source >= 0 && source < n && target >= 0 && target < n) {
19             adj_matrix[source][target] = true;
20             adj_matrix[target][source] = true;  // do thi vo huong
21         }
22     }
23
24     // Xoa canh
25     void del_edge(int source, int target) {
26         if (source >= 0 && source < n && target >= 0 && target < n) {
```

```
            adj_matrix[source][target] = false;
            adj_matrix[target][source] = false;
        }
    }

    // Kiem tra co canh khong
    bool has_edge(int source, int target) const {
        if (source >= 0 && source < n && target >= 0 && target < n) {
            return adj_matrix[source][target];
        }
        return false;
    }

    // Lay danh sach cac dinh ke
    vector<int> get_neighbors(int vertex) const {
        vector<int> neighbors;
        if (vertex >= 0 && vertex < n) {
            for (int i = 0; i < n; i++) {
                if (adj_matrix[vertex][i]) {
                    neighbors.push_back(i);
                }
            }
        }
        return neighbors;
    }

    // Lay tat ca cac canh duoi dang cap (source, target)
    vector<pair<int, int>> get_all_edges() const {
        vector<pair<int, int>> edges;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (adj_matrix[i][j]) {
                    edges.push_back({i, j});
                }
            }
        }
        return edges;
    }

    // Lay bac cua dinh
    int get_degree(int vertex) const {
        if (vertex < 0 || vertex >= n) return -1;
        int degree = 0;
        for (int i = 0; i < n; i++) {
            if (adj_matrix[vertex][i]) degree++;
        }
        return degree;
    }

    // Lay dinh nguon cua canh thu i (trong danh sach cac canh)
    int get_source(int edge_index) const {
        vector<pair<int, int>> edges = get_all_edges();
        if (edge_index >= 0 && edge_index < edges.size()) {
```

```
80              return edges[edge_index].first;
81          }
82          return -1;
83      }
84
85      // Lay dinh dich cua canh thu i
86      int get_target(int edge_index) const {
87          vector<pair<int, int>> edges = get_all_edges();
88          if (edge_index >= 0 && edge_index < edges.size()) {
89              return edges[edge_index].second;
90          }
91          return -1;
92      }
93
94      // In ma tran ke
95      void print_adjacency_matrix() const {
96          cout << "Ma tran ke:" << endl;
97          cout << "   ";
98          for (int j = 0; j < n; j++) {
99              cout << j << " ";
100         }
101         cout << endl;
102
103         for (int i = 0; i < n; i++) {
104             cout << i << ": ";
105             for (int j = 0; j < n; j++) {
106                 cout << adj_matrix[i][j] << " ";
107             }
108             cout << endl;
109         }
110     }
111 };
112
113 // Test
114 int main() {
115     Graph g(5);
116
117     // Them cac canh
118     g.add_edge(0, 1);
119     g.add_edge(1, 2);
120     g.add_edge(2, 3);
121     g.add_edge(3, 4);
122     g.add_edge(4, 0);
123
124     g.print_adjacency_matrix();
125
126     cout << "\nCac canh trong do thi:" << endl;
127     auto edges = g.get_all_edges();
128     for (int i = 0; i < edges.size(); i++) {
129         cout << "Canh " << i << ": (" << edges[i].first
130              << ", " << edges[i].second << ")" << endl;
131     }
132
```

```
    return 0;
}
```

6 Bài toán 1.5: Biểu diễn cây first-child, next-sibling

6.1 Mô tả bài toán

Mở rộng biểu diễn cây first-child, next-sibling để hỗ trợ các thao tác cơ bản trong thời gian $O(1)$.

6.2 Lý thuyết

Biểu diễn first-child, next-sibling là một cách hiệu quả để lưu trữ cây với số con thay đổi. Mỗi nút chỉ cần 2 con trỏ:

- `first_child`: trỏ đến con đầu tiên
- `next_sibling`: trỏ đến anh em tiếp theo

6.3 Implementation

Listing 5: Cau truc cay first-child next-sibling

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

struct TreeNode {
    int data;
    TreeNode* first_child;
    TreeNode* next_sibling;
    TreeNode* parent;  // De ho tro mot so thao tac O(1)

    TreeNode(int val) : data(val), first_child(nullptr),
                        next_sibling(nullptr), parent(nullptr) {}
};

class Tree {
private:
    TreeNode* root;

public:
    Tree() : root(nullptr) {}

    // Tao nut goc
    TreeNode* create_root(int data) {
        root = new TreeNode(data);
        return root;
    }

    // Them con dau tien - O(1)
    TreeNode* add_first_child(TreeNode* parent, int data) {
```

```
30          if (!parent) return nullptr;
31
32          TreeNode* new_child = new TreeNode(data);
33          new_child->parent = parent;
34
35          if (parent->first_child) {
36              // Neu da co con, them vao dau danh sach anh em
37              new_child->next_sibling = parent->first_child;
38          }
39          parent->first_child = new_child;
40
41          return new_child;
42      }
43
44      // Lay con dau tien - O(1)
45      TreeNode* first_child(TreeNode* node) {
46          return node ? node->first_child : nullptr;
47      }
48
49      // Lay anh em tiep theo - O(1)
50      TreeNode* next_sibling(TreeNode* node) {
51          return node ? node->next_sibling : nullptr;
52      }
53
54      // Lay cha - O(1)
55      TreeNode* parent(TreeNode* node) {
56          return node ? node->parent : nullptr;
57      }
58
59      // Dem so con - O(k) voi k la so con
60      int number_of_children(TreeNode* node) {
61          if (!node || !node->first_child) return 0;
62
63          int count = 0;
64          TreeNode* child = node->first_child;
65          while (child) {
66              count++;
67              child = child->next_sibling;
68          }
69          return count;
70      }
71
72      // Lay con thu i - O(i)
73      TreeNode* get_child(TreeNode* node, int index) {
74          if (!node || index < 0) return nullptr;
75
76          TreeNode* child = node->first_child;
77          int current = 0;
78          while (child && current < index) {
79              child = child->next_sibling;
80              current++;
81          }
82          return child;
```

```
    }

    // Kiem tra nut co phai la la khong - O(1)
    bool is_leaf(TreeNode* node) {
        return node && !node->first_child;
    }

    // Them anh em - O(1)
    TreeNode* add_sibling(TreeNode* node, int data) {
        if (!node) return nullptr;

        TreeNode* new_sibling = new TreeNode(data);
        new_sibling->parent = node->parent;
        new_sibling->next_sibling = node->next_sibling;
        node->next_sibling = new_sibling;

        return new_sibling;
    }

    // In cay theo thu tu pre-order
    void print_preorder(TreeNode* node, int depth = 0) {
        if (!node) return;

        // In khoang trang theo do sau
        for (int i = 0; i < depth; i++) {
            cout << "  ";
        }
        cout << node->data << endl;

        // Duyet tat ca cac con
        TreeNode* child = node->first_child;
        while (child) {
            print_preorder(child, depth + 1);
            child = child->next_sibling;
        }
    }

    TreeNode* get_root() { return root; }
};

// Test
int main() {
    Tree tree;

    // Tao cay
    TreeNode* root = tree.create_root(1);
    TreeNode* child1 = tree.add_first_child(root, 2);
    TreeNode* child2 = tree.add_first_child(root, 3);
    TreeNode* child3 = tree.add_first_child(root, 4);

    // Them con cho child1
    tree.add_first_child(child1, 5);
    tree.add_first_child(child1, 6);
```

```
136
137     cout << "Cay theo thu tu pre-order:" << endl;
138     tree.print_preorder(tree.get_root());
139
140     cout << "\nSo con cua nut goc: " << tree.number_of_children(root)
           << endl;
141     cout << "Nut goc co phai la la: " << (tree.is_leaf(root) ? "Co" :
           "Khong") << endl;
142
143     return 0;
144 }
```

7 Bài toán 1.6: Kiểm tra biểu diễn cây hợp lệ

7.1 Mô tả bài toán

Kiểm tra xem biểu diễn dựa trên đồ thị của một cây có thực sự là cây hay không trong thời gian tuyến tính.

7.2 Lý thuyết

Một đồ thị $G = (V, E)$ là cây khi và chỉ khi thỏa mãn hai trong ba điều kiện sau:

1. G liên thông
2. G không có chu trình (acyclic)
3. $|E| = |V| - 1$

7.3 Thuật toán kiểm tra

Algorithm 2 Kiểm tra đồ thị có phải là cây

```
Kiểm tra |E| = |V| − 1
if |E| ≠ |V| − 1 then
    return false
end if
Kiểm tra tính liên thông bằng DFS/BFS
if Không liên thông then
    return false
end if
return true
```

7.4 Implementation

Listing 6: Kiem tra do thi co phai la cay

```
1 #include <iostream>
2 #include <vector>
3 #include <queue>
```

```cpp
using namespace std;

class TreeValidator {
private:
    int n;
    vector<vector<int>> adj;

public:
    TreeValidator(int vertices) : n(vertices) {
        adj.resize(n);
    }

    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    // Kiem tra do thi co phai la cay - O(V + E)
    bool is_tree() {
        // Dieu kien 1: Kiem tra so canh
        int edge_count = 0;
        for (int i = 0; i < n; i++) {
            edge_count += adj[i].size();
        }
        edge_count /= 2; // Vi do thi vo huong

        if (edge_count != n - 1) {
            cout << "So canh khong bang n-1: " << edge_count
                 << " != " << (n-1) << endl;
            return false;
        }

        // Dieu kien 2: Kiem tra tinh lien thong
        if (!is_connected()) {
            cout << "Do thi khong lien thong" << endl;
            return false;
        }

        return true;
    }

private:
    // Kiem tra tinh lien thong bang DFS - O(V + E)
    bool is_connected() {
        vector<bool> visited(n, false);

        // Tim dinh dau tien co canh
        int start = -1;
        for (int i = 0; i < n; i++) {
            if (!adj[i].empty()) {
                start = i;
                break;
            }
```

```
        }

        if (start == -1) {
            // Do thi khong co canh nao
            return n <= 1;
        }

        dfs(start, visited);

        // Kiem tra tat ca dinh co canh da duoc tham
        for (int i = 0; i < n; i++) {
            if (!adj[i].empty() && !visited[i]) {
                return false;
            }
        }

        return true;
    }

    void dfs(int v, vector<bool>& visited) {
        visited[v] = true;
        for (int u : adj[v]) {
            if (!visited[u]) {
                dfs(u, visited);
            }
        }
    }

public:
    // Phuong phap thay the: Kiem tra chu trinh bang Union-Find
    bool is_tree_union_find() {
        // Kiem tra so canh
        int edge_count = 0;
        vector<pair<int,int>> edges;

        for (int i = 0; i < n; i++) {
            for (int j : adj[i]) {
                if (i < j) {
                    edges.push_back({i, j});
                    edge_count++;
                }
            }
        }

        if (edge_count != n - 1) return false;

        // Kiem tra chu trinh bang Union-Find
        vector<int> parent(n);
        for (int i = 0; i < n; i++) parent[i] = i;

        function<int(int)> find = [&](int x) {
            return parent[x] == x ? x : parent[x] = find(parent[x]);
        };
```

```
110
111         for (auto& edge : edges) {
112             int u = edge.first, v = edge.second;
113             int pu = find(u), pv = find(v);
114
115             if (pu == pv) {
116                 cout << "Tim thay chu trinh tai canh (" << u << ", "
                        << v << ")" << endl;
117                 return false; // Co chu trinh
118             }
119             parent[pu] = pv;
120         }
121
122         // Kiem tra tinh lien thong
123         int components = 0;
124         for (int i = 0; i < n; i++) {
125             if (find(i) == i && !adj[i].empty()) {
126                 components++;
127             }
128         }
129
130         return components <= 1;
131     }
132
133     void print_graph_info() {
134         int edge_count = 0;
135         for (int i = 0; i < n; i++) {
136             edge_count += adj[i].size();
137         }
138         edge_count /= 2;
139
140         cout << "So dinh: " << n << endl;
141         cout << "So canh: " << edge_count << endl;
142         cout << "Dieu kien can: " << (edge_count == n - 1 ? "Thoa man"
            : "Khong thoa man") << endl;
143         cout << "Lien thong: " << (is_connected() ? "Co" : "Khong") <<
            endl;
144         cout << "La cay: " << (is_tree() ? "Co" : "Khong") << endl;
145     }
146 };
147
148 // Test
149 int main() {
150     // Test 1: Do thi la cay
151     cout << "=== Test 1: Do thi la cay ===" << endl;
152     TreeValidator tree1(5);
153     tree1.add_edge(0, 1);
154     tree1.add_edge(1, 2);
155     tree1.add_edge(1, 3);
156     tree1.add_edge(3, 4);
157     tree1.print_graph_info();
158
159     cout << "\n=== Test 2: Do thi co chu trinh ===" << endl;
```

```
    TreeValidator tree2(4);
    tree2.add_edge(0, 1);
    tree2.add_edge(1, 2);
    tree2.add_edge(2, 3);
    tree2.add_edge(3, 0);
    tree2.print_graph_info();

    cout << "\n=== Test 3: Do thi khong lien thong ===" << endl;
    TreeValidator tree3(4);
    tree3.add_edge(0, 1);
    tree3.add_edge(2, 3);
    tree3.print_graph_info();

    return 0;
}
```

Listing 7: Kiem tra cay bang Python

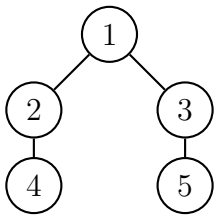
```
from collections import defaultdict, deque

class TreeValidator:
    def __init__(self, vertices):
        self.n = vertices
        self.adj = defaultdict(list)
        self.edge_count = 0

    def add_edge(self, u, v):
        self.adj[u].append(v)
        self.adj[v].append(u)
        self.edge_count += 1

    def is_tree(self):
        """Kiem tra do thi co phai la cay - O(V + E)"""
        # Dieu kien 1: So canh = n - 1
        if self.edge_count != self.n - 1:
            print(f"So canh khong bang n-1: {self.edge_count} != {self
                .n-1}")
            return False

        # Dieu kien 2: Tinh lien thong
        if not self.is_connected():
            print("Do thi khong lien thong")
            return False

        return True

    def is_connected(self):
        """Kiem tra tinh lien thong bang BFS"""
        if self.n == 0:
            return True

        # Tim dinh dau tien co canh
        start = None
        for v in range(self.n):
```

```
36              if self.adj[v]:
37                  start = v
38                  break
39
40          if start is None:
41              return self.n <= 1
42
43          visited = set()
44          queue = deque([start])
45          visited.add(start)
46
47          while queue:
48              v = queue.popleft()
49              for u in self.adj[v]:
50                  if u not in visited:
51                      visited.add(u)
52                      queue.append(u)
53
54          # Kiem tra tat ca dinh co canh da duoc tham
55          for v in range(self.n):
56              if self.adj[v] and v not in visited:
57                  return False
58
59          return True
60
61      def has_cycle_dfs(self):
62          """Kiem tra chu trinh bang DFS"""
63          visited = set()
64
65          def dfs(v, parent):
66              visited.add(v)
67              for u in self.adj[v]:
68                  if u not in visited:
69                      if dfs(u, v):
70                          return True
71                  elif u != parent:
72                      return True
73              return False
74
75          for v in range(self.n):
76              if v not in visited and self.adj[v]:
77                  if dfs(v, -1):
78                      return True
79          return False
80
81      def print_analysis(self):
82          """In phan tich do thi"""
83          print(f"So dinh: {self.n}")
84          print(f"So canh: {self.edge_count}")
85          print(f"Dieu kien can (|E| = |V|-1): {'Thoa man' if self.
                edge_count == self.n-1 else 'Khong thoa man'}")
86          print(f"Lien thong: {'Co' if self.is_connected() else 'Khong'}
                ")
```

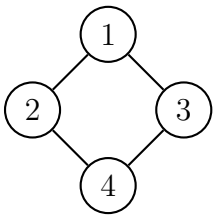
```
        print(f"Co chu trinh: {'Co' if self.has_cycle_dfs() else '
            Khong'}")
        print(f"La cay: {'Co' if self.is_tree() else 'Khong'}")

# Test cases
if __name__ == "__main__":
    # Test 1: Do thi la cay
    print("=== Test 1: Do thi la cay ===")
    tree1 = TreeValidator(5)
    tree1.add_edge(0, 1)
    tree1.add_edge(1, 2)
    tree1.add_edge(1, 3)
    tree1.add_edge(3, 4)
    tree1.print_analysis()

    print("\n=== Test 2: Do thi co chu trinh ===")
    tree2 = TreeValidator(4)
    tree2.add_edge(0, 1)
    tree2.add_edge(1, 2)
    tree2.add_edge(2, 3)
    tree2.add_edge(3, 0)
    tree2.print_analysis()

    print("\n=== Test 3: Do thi khong lien thong ===")
    tree3 = TreeValidator(4)
    tree3.add_edge(0, 1)
    tree3.add_edge(2, 3)
    tree3.print_analysis()
```

8 Minh họa bằng hình vẽ

8.1 Ví dụ về các loại đồ thị



Cây: 5 đỉnh, 4 cạnh



Có chu trình: 4 đỉnh, 4 cạnh

9 Tổng kết

9.1 Bảng tóm tắt các bài toán

Bài toán	**Mô tả**	**Độ phức tạp**	**Ý tưởng chính**
1.1 Kích thước đồ thị	Tính số cạnh K_n, $K_{p,q}$	$O(1)$	Công thức toán học
1.2 Tính hai phần	Xác định khi nào C_n, K_n là hai phần	$O(V+E)$	Kiểm tra chu trình lẻ
1.3 Cây khung	Tìm tất cả cây khung	$O(E^{V-1})$	Backtracking, Matrix-Tree
1.4 Ma trận kề	Mở rộng thao tác với đỉnh nguồn/đích	$O(1)$ - $O(V^2)$	Biểu diễn ma trận
1.5 First-child next-sibling	Biểu diễn cây hiệu quả	$O(1)$ - $O(k)$	2 con trỏ mỗi nút
1.6 Kiểm tra cây	Xác định đồ thị có phải là cây	$O(V+E)$	DFS + kiểm tra điều kiện