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

trees,layered

THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU (DEPTH-FIRST SEARCH - DFS)

Toán Tổ Hợp và Lý Thuyết Đồ Thị

1 Lý thuyết cơ bản về DFS

1.1 Định nghĩa và đặc điểm

Tìm kiếm theo chiều sâu (DFS - Depth-First Search) là một thuật toán duyệt đồ thị có những đặc điểm sau:

- Sử dụng cấu trúc dữ liệu **ngăn xếp (Stack)** - LIFO (Last In, First Out)
- Đi sâu vào một nhánh trước khi quay lại duyệt nhánh khác
- Có thể cài đặt bằng đệ quy hoặc sử dụng stack tường minh
- Phù hợp cho việc tìm kiếm trên đồ thị sâu và hẹp

1.2 So sánh DFS và BFS

Tiêu chí	**DFS**	**BFS**
Cấu trúc dữ liệu	Stack (LIFO)	Queue (FIFO)
Chiến lược duyệt	Theo chiều sâu	Theo từng lớp
Đường đi ngắn nhất	Không	Có
Phát hiện chu trình	Có	Có
Sắp xếp topo	Có	Không
Thành phần liên thông mạnh	Có	Không

1.3 Ứng dụng chính của DFS

- Phát hiện chu trình trong đồ thị
- Sắp xếp topo (Topological sorting)
- Tìm thành phần liên thông mạnh (Strongly Connected Components)
- Tìm cầu và khớp trong đồ thị
- Giải bài toán đường đi và kết nối
- Thuật toán Tarjan và Kosaraju

2 Bài toán 11: DFS trên đồ thị đơn

2.1 Mô tả bài toán

Đề bài: Cho đồ thị đơn $G = (V, E)$ với n đỉnh và m cạnh. Cài đặt thuật toán tìm kiếm theo chiều sâu (DFS) trên đồ thị G.

Input:

- Đồ thị đơn G (không có cạnh bội, không có khuyên)
- Đỉnh xuất phát $s \in V$

Output:

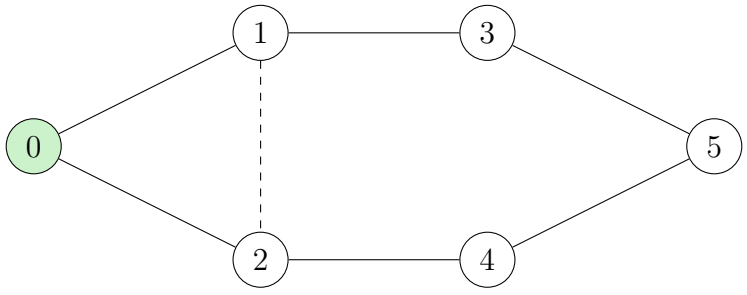
- Thứ tự duyệt các đỉnh theo DFS
- Thời gian bắt đầu và kết thúc thăm mỗi đỉnh
- Cây DFS (DFS tree) và các loại cạnh
- Phát hiện chu trình (nếu có)

2.2 Ý tưởng và giải pháp

Ý tưởng chính:

1. Bắt đầu từ đỉnh xuất phát s, đánh dấu s đã được thăm
2. Duyệt từng đỉnh v kề với s chưa được thăm
3. Đệ quy gọi DFS cho v
4. Sau khi duyệt xong tất cả đỉnh kề của s, kết thúc xử lý s

2.3 Minh họa thuật toán



Đồ thị mẫu - DFS bắt đầu từ đỉnh 0

Quá trình DFS từ đỉnh 0:

- **Bước 1:** Thăm 0, Stack = [0], Time: start[0] = 1
- **Bước 2:** Từ 0 → 1, Stack = [0,1], Time: start[1] = 2
- **Bước 3:** Từ 1 → 3, Stack = [0,1,3], Time: start[3] = 3
- **Bước 4:** Từ 3 → 5, Stack = [0,1,3,5], Time: start[5] = 4
- **Bước 5:** Kết thúc 5, Stack = [0,1,3], Time: finish[5] = 5
- **Bước 6:** Kết thúc 3, Stack = [0,1], Time: finish[3] = 6
- **Kết quả:** Thứ tự: 0 → 1 → 3 → 5 → 2 → 4

2.4 Thuật toán chi tiết

Algorithm 1 Depth-First Search

Require: Graph $G = (V, E)$, starting vertex s
Ensure: DFS traversal order, start/finish times, DFS tree

Initialize $visited[v] = false$ for all $v \in V$
Initialize $startTime[v] = 0$, $finishTime[v] = 0$ for all $v \in V$
Initialize $parent[v] = -1$ for all $v \in V$
$time = 0$
$DFS(s)$
DFSu
$visited[u] = true$
$time = time + 1$
$startTime[u] = time$
Process vertex u (e.g., print u)
for each vertex v adjacent to u **do**
  if $visited[v] = false$ **then**
    $parent[v] = u$
    $DFS(v)$
  end if
end for
$time = time + 1$
$finishTime[u] = time$

3 Cài đặt cho đồ thị đơn

```cpp
#include <iostream>
#include <vector>
#include <stack>
#include <set>
#include <iomanip>

using namespace std;
```

```
8
9  class SimpleDFS {
10 private:
11     int numVertices;
12     vector<vector<int>> adjList;
13     vector<bool> visited;
14     vector<int> startTime;
15     vector<int> finishTime;
16     vector<int> parent;
17     vector<int> dfsOrder;
18     int timeCounter;
19
20     // Phan loai canh
21     enum EdgeType { TREE, BACK, FORWARD, CROSS };
22     vector<tuple<int, int, EdgeType>> edges;
23
24 public:
25     SimpleDFS(int n) : numVertices(n), timeCounter(0) {
26         adjList.resize(n);
27         resetArrays();
28     }
29
30     void resetArrays() {
31         visited.assign(numVertices, false);
32         startTime.assign(numVertices, 0);
33         finishTime.assign(numVertices, 0);
34         parent.assign(numVertices, -1);
35         dfsOrder.clear();
36         edges.clear();
37         timeCounter = 0;
38     }
39
40     void addEdge(int u, int v) {
41         adjList[u].push_back(v);
42         adjList[v].push_back(u); // Do thi vo huong
43     }
44
45     // DFS de quy
46     void dfsRecursive(int u) {
47         visited[u] = true;
48         startTime[u] = ++timeCounter;
49         dfsOrder.push_back(u);
50
51         cout << "Tham dinh " << u << " (start: " << startTime[u] << ")"
   << endl;
52
53         // Duyet cac dinh ke
54         for (int v : adjList[u]) {
55             if (!visited[v]) {
56                 parent[v] = u;
57                 edges.push_back({u, v, TREE});
58                 cout << "  Canh cay: " << u << " -> " << v << endl;
59                 dfsRecursive(v);
60             } else {
61                 // Phan loai canh
62                 if (startTime[v] < startTime[u] && finishTime[v] == 0) {
```

```
                    edges.push_back({u, v, BACK});
                    cout << "  Canh nguoc: " << u << " -> " << v << " (
chu trinh!)" << endl;
                } else if (startTime[v] > startTime[u]) {
                    edges.push_back({u, v, FORWARD});
                    cout << "  Canh xuoi: " << u << " -> " << v << endl;
                } else {
                    edges.push_back({u, v, CROSS});
                    cout << "  Canh cheo: " << u << " -> " << v << endl;
                }
            }
        }

        finishTime[u] = ++timeCounter;
        cout << "Ket thuc dinh " << u << " (finish: " << finishTime[u]
<< ")" << endl;
    }

    // DFS khong de quy
    void dfsIterative(int start) {
        resetArrays();

        stack<int> stk;
        stack<int> path; // De theo doi duong di

        cout << "DFS khong de quy tu dinh " << start << ":" << endl;

        stk.push(start);

        while (!stk.empty()) {
            int u = stk.top();
            stk.pop();

            if (!visited[u]) {
                visited[u] = true;
                startTime[u] = ++timeCounter;
                dfsOrder.push_back(u);

                cout << "Tham dinh " << u << " (start: " << startTime[u]
 << ")" << endl;

                // Them cac dinh ke vao stack (thu tu nguoc lai)
                vector<int> neighbors = adjList[u];
                sort(neighbors.rbegin(), neighbors.rend());

                for (int v : neighbors) {
                    if (!visited[v]) {
                        stk.push(v);
                        if (parent[v] == -1) {
                            parent[v] = u;
                        }
                    }
                }
            }
        }
        cout << endl;
```

```
116     }
117
118     // DFS toan bo do thi
119     void dfsComplete() {
120         resetArrays();
121
122         cout << "DFS toan bo do thi:" << endl;
123         int componentCount = 0;
124
125         for (int i = 0; i < numVertices; i++) {
126             if (!visited[i]) {
127                 componentCount++;
128                 cout << "\nThanh phan lien thong " << componentCount <<
    ":" << endl;
129                 dfsRecursive(i);
130             }
131         }
132
133         cout << "\nTong so thanh phan lien thong: " << componentCount <<
     endl;
134         analyzeCycles();
135     }
136
137     // Phan tich chu trinh
138     void analyzeCycles() {
139         bool hasCycle = false;
140
141         cout << "\nPhan tich canh:" << endl;
142         for (auto& edge : edges) {
143             int u = get<0>(edge);
144             int v = get<1>(edge);
145             EdgeType type = get<2>(edge);
146
147             string typeStr;
148             switch (type) {
149                 case TREE: typeStr = "Cay"; break;
150                 case BACK: typeStr = "Nguoc"; hasCycle = true; break;
151                 case FORWARD: typeStr = "Xuoi"; break;
152                 case CROSS: typeStr = "Cheo"; break;
153             }
154
155             cout << "Canh (" << u << "," << v << "): " << typeStr <<
    endl;
156         }
157
158         cout << "\nDo thi co chu trinh: " << (hasCycle ? "Co" : "Khong")
     << endl;
159     }
160
161     // Tim tat ca duong di tu start den end
162     void findAllPaths(int start, int end) {
163         vector<vector<int>> allPaths;
164         vector<int> currentPath;
165         vector<bool> tempVisited(numVertices, false);
166
167         findAllPathsHelper(start, end, currentPath, allPaths,
```

```
    tempVisited);

        cout << "\nTat ca duong di tu " << start << " den " << end << ":
    " << endl;
        for (int i = 0; i < allPaths.size(); i++) {
            cout << "Duong " << i + 1 << ": ";
            for (int j = 0; j < allPaths[i].size(); j++) {
                cout << allPaths[i][j];
                if (j < allPaths[i].size() - 1) cout << " -> ";
            }
            cout << endl;
        }

        if (allPaths.empty()) {
            cout << "Khong co duong di tu " << start << " den " << end
    << endl;
        }
    }

private:
    void findAllPathsHelper(int current, int end, vector<int>& path,
                            vector<vector<int>>& allPaths, vector<bool>&
    tempVisited) {
        path.push_back(current);
        tempVisited[current] = true;

        if (current == end) {
            allPaths.push_back(path);
        } else {
            for (int neighbor : adjList[current]) {
                if (!tempVisited[neighbor]) {
                    findAllPathsHelper(neighbor, end, path, allPaths,
    tempVisited);
                }
            }
        }

        path.pop_back();
        tempVisited[current] = false;
    }

public:
    // Sap xep topo (chi cho do thi co huong)
    vector<int> topologicalSort() {
        resetArrays();
        stack<int> topoStack;

        // Goi DFS cho tat ca dinh
        for (int i = 0; i < numVertices; i++) {
            if (!visited[i]) {
                topoDFS(i, topoStack);
            }
        }

        vector<int> result;
        while (!topoStack.empty()) {
```

```
            result.push_back(topoStack.top());
            topoStack.pop();
        }

        return result;
    }

private:
    void topoDFS(int u, stack<int>& topoStack) {
        visited[u] = true;

        for (int v : adjList[u]) {
            if (!visited[v]) {
                topoDFS(v, topoStack);
            }
        }

        topoStack.push(u);
    }

public:
    // In thong tin thoi gian
    void printTimes() {
        cout << "\nThong tin thoi gian DFS:" << endl;
        cout << setw(6) << "Dinh" << setw(12) << "Start" << setw(12) <<
   "Finish"
             << setw(10) << "Cha" << endl;
        cout << string(40, '-') << endl;

        for (int i = 0; i < numVertices; i++) {
            cout << setw(6) << i
                 << setw(12) << startTime[i]
                 << setw(12) << finishTime[i]
                 << setw(10) << parent[i] << endl;
        }
        cout << endl;
    }

    void printDFSTree() {
        cout << "Cay DFS:" << endl;
        for (int i = 0; i < numVertices; i++) {
            if (parent[i] != -1) {
                cout << parent[i] << " -> " << i << endl;
            }
        }
        cout << endl;
    }

    void printGraph() {
        cout << "Bieu dien do thi don:" << endl;
        for (int i = 0; i < numVertices; i++) {
            cout << i << ": ";
            for (int neighbor : adjList[i]) {
                cout << neighbor << " ";
            }
            cout << endl;
```

```
        }
        cout << endl;
    }

    // Kiem tra tinh lien thong
    bool isConnected() {
        dfsRecursive(0);

        for (int i = 0; i < numVertices; i++) {
            if (!visited[i]) {
                return false;
            }
        }
        return true;
    }
};

// Demo cho do thi don
void demonstrateSimpleDFS() {
    cout << "=== DEMO DFS CHO DO THI DON ===" << endl << endl;

    SimpleDFS graph(6);

    // Tao do thi mau
    graph.addEdge(0, 1);
    graph.addEdge(0, 2);
    graph.addEdge(1, 3);
    graph.addEdge(2, 4);
    graph.addEdge(3, 5);
    graph.addEdge(4, 5);
    graph.addEdge(1, 2); // Tao chu trinh

    graph.printGraph();

    // DFS de quy
    cout << "1. DFS DE QUY:" << endl;
    graph.dfsComplete();
    graph.printTimes();
    graph.printDFSTree();

    // DFS khong de quy
    cout << "2. DFS KHONG DE QUY:" << endl;
    graph.dfsIterative(0);

    // Tim tat ca duong di
    graph.findAllPaths(0, 5);

    // Kiem tra lien thong
    cout << "\nDo thi co lien thong: "
         << (graph.isConnected() ? "Co" : "Khong") << endl;
}
```

Listing 1: DFS cho đồ thị đơn - C++

```
from collections import defaultdict
from typing import List, Tuple, Dict, Set
```

```
import sys

class SimpleDFS:
    """DFS cho        t h     n  """

    def __init__(self, num_vertices: int):
        self.num_vertices = num_vertices
        self.adj_list = defaultdict(list)
        self.reset_arrays()

    def reset_arrays(self):
        self.visited = [False] * self.num_vertices
        self.start_time = [0] * self.num_vertices
        self.finish_time = [0] * self.num_vertices
        self.parent = [-1] * self.num_vertices
        self.dfs_order = []
        self.time_counter = 0
        self.edges = []  # (u, v, type)

    def add_edge(self, u: int, v: int):
        """Th  m   c nh   v  o         t h    v    h    ng """
        self.adj_list[u].append(v)
        self.adj_list[v].append(u)

    def dfs_recursive(self, u: int):
        """DFS       quy"""
        self.visited[u] = True
        self.time_counter += 1
        self.start_time[u] = self.time_counter
        self.dfs_order.append(u)

        print(f"Th  m    nh    {u} (start: {self.start_time[u]})")

        #  D u y t   c  c    nh     k
        for v in sorted(self.adj_list[u]):
            if not self.visited[v]:
                self.parent[v] = u
                self.edges.append((u, v, "TREE"))
                print(f"   C nh   c  y: {u} -> {v}")
                self.dfs_recursive(v)
            else:
                # Ph  n   l o i   c nh
                if (self.start_time[v] < self.start_time[u] and
                    self.finish_time[v] == 0):
                    self.edges.append((u, v, "BACK"))
                    print(f"   C nh   n g   c : {u} -> {v} (chu  tr  nh!)"
    )
                elif self.start_time[v] > self.start_time[u]:
                    self.edges.append((u, v, "FORWARD"))
                    print(f"   C nh   xu  i: {u} -> {v}")
                else:
                    self.edges.append((u, v, "CROSS"))
                    print(f"   C nh   ch  o: {u} -> {v}")

        self.time_counter += 1
        self.finish_time[u] = self.time_counter
```

```
58          print(f" K t   th c    nh    {u} (finish: {self.finish_time[u]})"
    )
59
60      def dfs_iterative(self, start: int):
61          """DFS kh ng       quy  s    d ng  stack"""
62          self.reset_arrays()
63
64          stack = [start]
65          print(f"DFS kh ng       quy  t     nh    {start}:")
66
67          while stack:
68              u = stack.pop()
69
70              if not self.visited[u]:
71                  self.visited[u] = True
72                  self.time_counter += 1
73                  self.start_time[u] = self.time_counter
74                  self.dfs_order.append(u)
75
76                  print(f"Th  m    nh    {u} (start: {self.start_time[u]})"
    )
77
78                  # Th m  c  c    nh     k    v  o  stack  ( t h    t
     n g   c   l i )
79                  neighbors = sorted(self.adj_list[u], reverse=True)
80                  for v in neighbors:
81                      if not self.visited[v]:
82                          stack.append(v)
83                          if self.parent[v] == -1:
84                              self.parent[v] = u
85          print()
86
87      def dfs_complete(self):
88          """DFS to n  b          t h  """
89          self.reset_arrays()
90
91          print("DFS to n  b          t h  :")
92          component_count = 0
93
94          for i in range(self.num_vertices):
95              if not self.visited[i]:
96                  component_count += 1
97                  print(f"\nTh nh  p h n  li n  th ng {component_count}:"
    )
98                  self.dfs_recursive(i)
99
100         print(f"\ n T ng   s   th nh  p h n  li n  th ng: {
    component_count}")
101         self.analyze_cycles()
102
103     def analyze_cycles(self):
104         """Ph n  t ch  chu  tr nh"""
105         has_cycle = False
106
107         print("\nPh n  t ch  c nh :")
108         for u, v, edge_type in self.edges:
```

```
109             print(f" C nh  ({u},{v}): {edge_type}")
110             if edge_type == "BACK":
111                 has_cycle = True
112 
113         print(f"\ n      t h   c   chu tr nh: {'C  ' if has_cycle else '
    Kh ng'}")
114 
115     def find_all_paths(self, start: int, end: int) -> List[List[int]]:
116         """T m  t t   c       ng      i   t   start   n   end"""
117         all_paths = []
118         current_path = []
119         temp_visited = [False] * self.num_vertices
120 
121         self._find_all_paths_helper(start, end, current_path, all_paths,
     temp_visited)
122 
123         print(f"\ n T t   c       ng      i   t   {start}   n   {end}:")
124         for i, path in enumerate(all_paths, 1):
125             path_str = ' -> '.join(map(str, path))
126             print(f"    ng     {i}: {path_str}")
127 
128         if not all_paths:
129             print(f"Kh ng  c      ng      i   t   {start}   n   {end}")
130 
131         return all_paths
132 
133     def _find_all_paths_helper(self, current: int, end: int, path: List[
    int],
134                                all_paths: List[List[int]], temp_visited:
    List[bool]):
135         """Helper function for finding all paths"""
136         path.append(current)
137         temp_visited[current] = True
138 
139         if current == end:
140             all_paths.append(path.copy())
141         else:
142             for neighbor in self.adj_list[current]:
143                 if not temp_visited[neighbor]:
144                     self._find_all_paths_helper(neighbor, end, path,
    all_paths, temp_visited)
145 
146         path.pop()
147         temp_visited[current] = False
148 
149     def topological_sort(self) -> List[int]:
150         """ S p   x p  topo ( c h   cho       t h   c   h   ng )"""
151         self.reset_arrays()
152         topo_stack = []
153 
154         #  G i  DFS cho  t t   c      nh
155         for i in range(self.num_vertices):
156             if not self.visited[i]:
157                 self._topo_dfs(i, topo_stack)
158 
159         return topo_stack[::-1]
```

```
160
161     def _topo_dfs(self, u: int, topo_stack: List[int]):
162         """DFS helper for topological sort"""
163         self.visited[u] = True
164
165         for v in self.adj_list[u]:
166             if not self.visited[v]:
167                 self._topo_dfs(v, topo_stack)
168                 topo_stack.append(u)
169
170     def print_times(self):
171         """In th ng tin  t h i  gian DFS"""
172         print("\nTh ng tin  t h i  gian DFS:")
173         print(f"{'  nh  ':>6} {'Start':>12} {'Finish':>12} {'Cha':>10}"
    )
174         print("-" * 40)
175
176         for i in range(self.num_vertices):
177             print(f"{i:>6} {self.start_time[i]:>12} {self.finish_time[i
    ]:>12} {self.parent[i]:>10}")
178         print()
179
180     def print_dfs_tree(self):
181         """In  c  y  DFS"""
182         print("C  y  DFS:")
183         for i in range(self.num_vertices):
184             if self.parent[i] != -1:
185                 print(f"{self.parent[i]} -> {i}")
186         print()
187
188     def print_graph(self):
189         """In  b i u  d i n  t h  """
190         print(" B i u  d i n  t h  n  :")
191         for i in range(self.num_vertices):
192             neighbors = ' '.join(map(str, sorted(self.adj_list[i])))
193             print(f"{i}: {neighbors}")
194         print()
195
196     def is_connected(self) -> bool:
197         """ K i m  tra  t  nh  li  n  th ng"""
198         self.reset_arrays()
199         self.dfs_recursive(0)
200
201         return all(self.visited)
202
203 # Demo cho  t h  n
204 def demonstrate_simple_dfs():
205     print("=== DEMO DFS CHO  T H  N  ===\n")
206
207     graph = SimpleDFS(6)
208
209     #  T o  t h  m u
210     graph.add_edge(0, 1)
211     graph.add_edge(0, 2)
212     graph.add_edge(1, 3)
213     graph.add_edge(2, 4)
```

```
    graph.add_edge(3, 5)
    graph.add_edge(4, 5)
    graph.add_edge(1, 2)  #  T o  chu tr nh

    graph.print_graph()

    # DFS       quy
    print("1. DFS       QUY:")
    graph.dfs_complete()
    graph.print_times()
    graph.print_dfs_tree()

    # DFS kh ng       quy
    print("2. DFS KH NG       QUY:")
    graph.dfs_iterative(0)

    # T m  t t  c      ng     i
    graph.find_all_paths(0, 5)

    #  K i m  tra li n th ng
    print(f"\ n     t h  c  li n th ng: {'C ' if graph.is_connected
    () else 'Kh ng'}")

if __name__ == "__main__":
    demonstrate_simple_dfs()
```

Listing 2: DFS cho đồ thị đơn - Python

4 Bài toán 12: DFS trên đa đồ thị

4.1 Mô tả bài toán

Đề bài: Cho đa đồ thị $G = (V, E)$ với n đỉnh và m cạnh, trong đó có thể có cạnh bội (multiple edges) giữa hai đỉnh. Cài đặt thuật toán DFS trên đa đồ thị G.

Đặc điểm của đa đồ thị:

- Có thể có nhiều cạnh giữa cùng một cặp đỉnh
- Không có khuyên (self-loop)
- Cần xử lý cạnh bội một cách hiệu quả

Input:

- Đa đồ thị G với cạnh bội
- Đỉnh xuất phát $s \in V$

Output:

- Thứ tự duyệt DFS
- Số lần xuất hiện mỗi cạnh
- Cây DFS và phân loại cạnh
- Phát hiện cạnh bội

4.2 Ý tưởng và giải pháp

Thách thức chính:

1. Xử lý cạnh bội hiệu quả
2. Tránh duyệt trùng lặp không cần thiết
3. Phân biệt các cạnh bội khác nhau
4. Tính toán độ phức tạp chính xác

Giải pháp:

- Sử dụng multiset hoặc vector để lưu cạnh bội
- Đếm số lần xuất hiện mỗi cạnh
- Chỉ duyệt một lần qua mỗi đỉnh kề
- Ghi nhận tất cả cạnh bội trong cây DFS

```
#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <algorithm>

using namespace std;

class MultigraphDFS {
private:
    int numVertices;
    //  S    d ng  multiset       l  u   c nh   b i
    vector<multiset<int>> adjList;
    //    m     s    c nh   g i a   m i   c p    nh
    map<pair<int,int>, int> edgeCount;

    vector<bool> visited;
    vector<int> startTime;
    vector<int> finishTime;
    vector<int> parent;
    vector<int> dfsOrder;
    int timeCounter;

    // Th ng tin  v    c nh   b i
    vector<tuple<int, int, int, string>> edgeInfo; // (u, v, count, type
   )

public:
    MultigraphDFS(int n) : numVertices(n), timeCounter(0) {
        adjList.resize(n);
        resetArrays();
    }

```

```
33      void resetArrays() {
34          visited.assign(numVertices, false);
35          startTime.assign(numVertices, 0);
36          finishTime.assign(numVertices, 0);
37          parent.assign(numVertices, -1);
38          dfsOrder.clear();
39          edgeInfo.clear();
40          timeCounter = 0;
41      }
42
43      void addEdge(int u, int v) {
44          adjList[u].insert(v);
45          adjList[v].insert(u); //  a          t h    v    h    ng
46
47          //  C p    n h t    s     l     ng    c nh
48          pair<int,int> edge = {min(u,v), max(u,v)};
49          edgeCount[edge]++;
50      }
51
52      // DFS cho  a          t h
53      void dfsRecursive(int u) {
54          visited[u] = true;
55          startTime[u] = ++timeCounter;
56          dfsOrder.push_back(u);
57
58          cout << "Th  m     nh     " << u << " (start: " << startTime[u] <<
    ")" << endl;
59
60          //  D u y t   c  c     nh      k    ( c h    d u y t    m  t    l  n    m  i
      nh      k    duy   n h t )
61          set<int> uniqueNeighbors(adjList[u].begin(), adjList[u].end());
62
63          for (int v : uniqueNeighbors) {
64              //    m     s     c nh    g i a   u  v    v
65              pair<int,int> edge = {min(u,v), max(u,v)};
66              int count = edgeCount[edge];
67
68              if (!visited[v]) {
69                  parent[v] = u;
70                  edgeInfo.push_back({u, v, count, "TREE"});
71                  cout << "    C nh   c  y: " << u << " -> " << v
72                       << " ( s     c nh : " << count << ")" << endl;
73                  dfsRecursive(v);
74              } else {
75                  // Ph  n   l o i    c nh    b i
76                  if (startTime[v] < startTime[u] && finishTime[v] == 0) {
77                      edgeInfo.push_back({u, v, count, "BACK"});
78                      cout << "    C nh    n g    c : " << u << " -> " << v
79                           << " ( s     c nh : " << count << ", chu  tr  nh
    !)" << endl;
80                  } else if (startTime[v] > startTime[u]) {
81                      edgeInfo.push_back({u, v, count, "FORWARD"});
82                      cout << "    C nh    xu  i : " << u << " -> " << v
83                           << " ( s     c nh : " << count << ")" << endl;
84                  } else if (v != parent[u]) { // Tr  nh    c nh    cha-con
85                      edgeInfo.push_back({u, v, count, "CROSS"});
```

```
                    cout << "   C nh   ch o: " << u << " -> " << v
                         << " ( s    c nh : " << count << ")" << endl;
                }
            }
        }

        finishTime[u] = ++timeCounter;
        cout << " K t   th c    nh    " << u << " (finish: " <<
    finishTime[u] << ")" << endl;
    }

    void dfsComplete() {
        resetArrays();

        cout << "DFS to n   b    a         t h  :" << endl;
        int componentCount = 0;

        for (int i = 0; i < numVertices; i++) {
            if (!visited[i]) {
                componentCount++;
                cout << "\nTh nh   p h n   li  n   th ng " <<
    componentCount << ":" << endl;
                dfsRecursive(i);
            }
        }

        cout << "\ n T ng   s    th nh   p h n   li  n   th ng: " <<
    componentCount << endl;
        analyzeMultipleEdges();
    }

    void analyzeMultipleEdges() {
        cout << "\nPh n  t  ch   c nh    b i  :" << endl;

        bool hasCycle = false;
        int totalEdges = 0;
        int multipleEdges = 0;

        for (auto& info : edgeInfo) {
            int u = get<0>(info);
            int v = get<1>(info);
            int count = get<2>(info);
            string type = get<3>(info);

            cout << " C nh   (" << u << "," << v << "): "
                 << type << ",  S    l    ng : " << count << endl;

            totalEdges += count;
            if (count > 1) {
                multipleEdges++;
            }

            if (type == "BACK") {
                hasCycle = true;
            }
        }
```

```
139
140         cout << "\ n T h ng  k  :" << endl;
141         cout << " T ng  s  c nh : " << totalEdges/2 << endl; // Chia
    2 v  t h  v  h  ng
142         cout << " S  c p  nh  c  c nh  b i : " << multipleEdges
     << endl;
143         cout << "  t h  c  chu tr nh: " << (hasCycle ? "C  " : "
    Kh ng") << endl;
144     }
145
146     void printMultigraph() {
147         cout << " B i u  d i n  a  t h  :" << endl;
148         for (int i = 0; i < numVertices; i++) {
149             cout << i << ": ";
150             for (int neighbor : adjList[i]) {
151                 cout << neighbor << " ";
152             }
153             cout << endl;
154         }
155
156         cout << "\ n S  l  ng  c nh  g i a  c  c  p  nh  :" <<
    endl;
157         for (auto& edge : edgeCount) {
158             cout << "(" << edge.first.first << "," << edge.first.second
159                  << "): " << edge.second << "  c nh " << endl;
160         }
161         cout << endl;
162     }
163
164     // T m  ng  i  v i  t  c nh  b i  n h t
165     void findPathWithMinMultipleEdges(int start, int end) {
166         vector<int> dist(numVertices, INT_MAX);
167         vector<int> parent(numVertices, -1);
168         vector<bool> visited(numVertices, false);
169
170         dist[start] = 0;
171
172         for (int count = 0; count < numVertices - 1; count++) {
173             int u = -1;
174             for (int v = 0; v < numVertices; v++) {
175                 if (!visited[v] && (u == -1 || dist[v] < dist[u])) {
176                     u = v;
177                 }
178             }
179
180             if (dist[u] == INT_MAX) break;
181             visited[u] = true;
182
183             set<int> uniqueNeighbors(adjList[u].begin(), adjList[u].end
    ());
184             for (int v : uniqueNeighbors) {
185                 pair<int,int> edge = {min(u,v), max(u,v)};
186                 int edgeWeight = edgeCount[edge]; //  T r ng  s  =  s
      c nh  b i
187
188                 if (dist[u] + edgeWeight < dist[v]) {
```

```
                    dist[v] = dist[u] + edgeWeight;
                    parent[v] = u;
                }
            }
        }

        if (dist[end] == INT_MAX) {
            cout << "Kh ng  c      ng      i   t   " << start << "
    n    " << end << endl;
        } else {
            cout << "    ng      i   v i   t   c nh   b i   n h t   t   "
   << start << "    n    " << end << ":" << endl;
            vector<int> path;
            int current = end;
            while (current != -1) {
                path.push_back(current);
                current = parent[current];
            }

            reverse(path.begin(), path.end());

            cout << "    ng      i : ";
            for (int i = 0; i < path.size(); i++) {
                cout << path[i];
                if (i < path.size() - 1) {
                    cout << " -> ";
                }
            }
            cout << "\ n T ng   t r ng   s    c nh   b i : " << dist[end]
   << endl;
        }
    }
};

// Demo cho  a         t h
void demonstrateMultigraphDFS() {
    cout << "=== DEMO DFS CHO  A         T H   ===" << endl << endl;

    MultigraphDFS graph(5);

    //  T o   a         t h    v i   c nh   b i
    graph.addEdge(0, 1);
    graph.addEdge(0, 1); //  C nh   b i
    graph.addEdge(0, 1); //  C nh   b i   t h   3
    graph.addEdge(0, 2);
    graph.addEdge(1, 3);
    graph.addEdge(1, 3); //  C nh   b i
    graph.addEdge(2, 4);
    graph.addEdge(3, 4);
    graph.addEdge(3, 4); //  C nh   b i
    graph.addEdge(1, 2); //  T o  chu  tr nh

    graph.printMultigraph();

    // DFS  tr  n   a         t h
    graph.dfsComplete();
```

```
242
243     // T m     ng      i   v i   t   c nh   b i   n h t
244     graph.findPathWithMinMultipleEdges(0, 4);
245 }
```

Listing 3: DFS cho đa đồ thị - C++

```
1  from collections import defaultdict, Counter
2  from typing import List, Dict, Set, Tuple
3  import sys
4
5  class MultigraphDFS:
6      """DFS cho  a         t h  """
7
8      def __init__(self, num_vertices: int):
9          self.num_vertices = num_vertices
10         self.adj_list = [[] for _ in range(num_vertices)]  # Danh  s  ch
       k    v i   c nh   b i
11         self.edge_count = defaultdict(int)  #   m    s    c nh   g i a
       m i   c p    nh
12         self.reset_arrays()
13
14     def reset_arrays(self):
15         self.visited = [False] * self.num_vertices
16         self.start_time = [0] * self.num_vertices
17         self.finish_time = [0] * self.num_vertices
18         self.parent = [-1] * self.num_vertices
19         self.dfs_order = []
20         self.time_counter = 0
21         self.edge_info = []  # (u, v, count, type)
22
23     def add_edge(self, u: int, v: int):
24         """Th m  c nh  v o  a        t h  """
25         self.adj_list[u].append(v)
26         self.adj_list[v].append(u)
27
28         #  C p  n h t  s   l   ng  c nh
29         edge = tuple(sorted([u, v]))
30         self.edge_count[edge] += 1
31
32     def dfs_recursive(self, u: int):
33         """DFS       quy cho  a         t h  """
34         self.visited[u] = True
35         self.time_counter += 1
36         self.start_time[u] = self.time_counter
37         self.dfs_order.append(u)
38
39         print(f"Th  m    nh   {u} (start: {self.start_time[u]})")
40
41         #  D u y t  c  c   nh    k   duy  n h t
42         unique_neighbors = list(set(self.adj_list[u]))
43         unique_neighbors.sort()
44
45         for v in unique_neighbors:
46             #   m    s    c nh   g i a  u v  v
47             edge = tuple(sorted([u, v]))
```

```
            count = self.edge_count[edge]

            if not self.visited[v]:
                self.parent[v] = u
                self.edge_info.append((u, v, count, "TREE"))
                print(f"   C nh   c  y: {u} -> {v} ( s    c nh : {count})
")
                self.dfs_recursive(v)
            else:
                # Ph n  l o i   c nh   b i
                if (self.start_time[v] < self.start_time[u] and
                    self.finish_time[v] == 0):
                    self.edge_info.append((u, v, count, "BACK"))
                    print(f"   C nh   n g   c : {u} -> {v} ( s    c nh :
{count}, chu  tr nh!)")
                elif self.start_time[v] > self.start_time[u]:
                    self.edge_info.append((u, v, count, "FORWARD"))
                    print(f"   C nh   xu  i: {u} -> {v} ( s    c nh : {
count})")
                elif v != self.parent[u]:  # Tr nh   c nh   cha-con
                    self.edge_info.append((u, v, count, "CROSS"))
                    print(f"   C nh   ch  o: {u} -> {v} ( s    c nh : {
count})")

        self.time_counter += 1
        self.finish_time[u] = self.time_counter
        print(f" K t   th  c    nh    {u} (finish: {self.finish_time[u]})"
)

    def dfs_complete(self):
        """DFS  to  n   b    a         t h  """
        self.reset_arrays()

        print("DFS  to  n   b    a         t h  :")
        component_count = 0

        for i in range(self.num_vertices):
            if not self.visited[i]:
                component_count += 1
                print(f"\nTh  nh   p h n   li  n   th  ng {component_count}:"
)
                self.dfs_recursive(i)

        print(f"\ n T ng   s   th  nh   p h n   li  n   th  ng: {
component_count}")
        self.analyze_multiple_edges()

    def analyze_multiple_edges(self):
        """Ph  n  t  ch   c nh   b i """
        print("\nPh  n  t  ch   c nh   b i :")

        has_cycle = False
        total_edges = 0
        multiple_edges = 0

        for u, v, count, edge_type in self.edge_info:
```

```
97              print(f" C nh  ({u},{v}): {edge_type},  S    l    ng : {count
    }")
98
99              total_edges += count
100             if count > 1:
101                 multiple_edges += 1
102
103             if edge_type == "BACK":
104                 has_cycle = True
105
106         print(f"\ n T h ng  k  :")
107         print(f" T ng   s    c nh : {total_edges // 2}")  # Chia 2  v
              t h   v   h   ng
108         print(f" S    c p    nh   c   c nh   b i : {multiple_edges}")
109         print(f"       t h   c   chu  tr  nh: {' C  ' if has_cycle else '
    Kh  ng '}")
110
111     def print_multigraph(self):
112         """In  b i u   d i n   a         t h  """
113         print(" B i u   d i n   a         t h  :")
114         for i in range(self.num_vertices):
115             neighbors = ' '.join(map(str, self.adj_list[i]))
116             print(f"{i}: {neighbors}")
117
118         print("\ n S   l   ng   c nh   g i a  c  c   c p    nh   :")
119         for edge, count in sorted(self.edge_count.items()):
120             print(f"({edge[0]},{edge[1]}): {count}  c nh ")
121         print()
122
123     def find_path_with_min_multiple_edges(self, start: int, end: int):
124         """T  m    ng     i   v i   t   c nh   b i   n h t """
125         dist = [float('inf')] * self.num_vertices
126         parent = [-1] * self.num_vertices
127         visited = [False] * self.num_vertices
128
129         dist[start] = 0
130
131         for _ in range(self.num_vertices - 1):
132             u = -1
133             for v in range(self.num_vertices):
134                 if not visited[v] and (u == -1 or dist[v] < dist[u]):
135                     u = v
136
137             if dist[u] == float('inf'):
138                 break
139
140             visited[u] = True
141
142             unique_neighbors = list(set(self.adj_list[u]))
143             for v in unique_neighbors:
144                 edge = tuple(sorted([u, v]))
145                 edge_weight = self.edge_count[edge]  #  T r ng   s   =
    s    c nh   b i
146
147                 if dist[u] + edge_weight < dist[v]:
148                     dist[v] = dist[u] + edge_weight
```

```
                    parent[v] = u

        if dist[end] == float('inf'):
            print(f"Kh ng  c      ng      i   t   {start}   n   {end}")
        else:
            print(f"   ng      i   v i   t   c nh   b i   n h t   t   {
    start}   n   {end}:")
            path = []
            current = end
            while current != -1:
                path.append(current)
                current = parent[current]

            path.reverse()

            path_str = ' -> '.join(map(str, path))
            print(f"   ng      i : {path_str}")
            print(f" T ng   t r ng   s    c nh   b i : {int(dist[end])}")

# Demo cho  a         t h
def demonstrate_multigraph_dfs():
    print("=== DEMO DFS CHO  A         T H   ===\n")

    graph = MultigraphDFS(5)

    #  T o   a         t h   v i   c nh   b i
    graph.add_edge(0, 1)
    graph.add_edge(0, 1)  #  C nh   b i
    graph.add_edge(0, 1)  #  C nh   b i   t h   3
    graph.add_edge(0, 2)
    graph.add_edge(1, 3)
    graph.add_edge(1, 3)  #  C nh   b i
    graph.add_edge(2, 4)
    graph.add_edge(3, 4)
    graph.add_edge(3, 4)  #  C nh   b i
    graph.add_edge(1, 2)  #  T o  chu  tr  nh

    graph.print_multigraph()

    # DFS  tr  n  a         t h
    graph.dfs_complete()

    # T m     ng      i   v i   t   c nh   b i   n h t
    graph.find_path_with_min_multiple_edges(0, 4)

if __name__ == "__main__":
    demonstrate_multigraph_dfs()
```

Listing 4: DFS cho đa đồ thị - Python

5 Bài toán 13: DFS trên đồ thị tổng quát

5.1 Mô tả bài toán

Đề bài: Cho đồ thị tổng quát $G = (V, E)$ với n đỉnh và m cạnh, trong đó có thể có:

- Cạnh bội (multiple edges)
- Khuyên (self-loops)
- Cạnh có hướng và vô hướng (mixed graph)

Cài đặt thuật toán DFS trên đồ thị tổng quát G.
Đặc điểm đồ thị tổng quát:

- Bao gồm tất cả loại cạnh có thể
- Cần xử lý khuyên một cách đặc biệt
- Phân biệt cạnh có hướng và vô hướng
- Phức tạp nhất trong các loại đồ thị

Input:

- Đồ thị tổng quát G với đầy đủ loại cạnh
- Đỉnh xuất phát $s \in V$
- Thông tin về hướng của từng cạnh

Output:

- Thứ tự duyệt DFS
- Phân loại tất cả loại cạnh
- Xử lý khuyên và cạnh bội
- Cây DFS tổng quát
- Phát hiện chu trình mạnh

5.2 Ý tưởng và giải pháp

Thách thức chính:

1. Xử lý khuyên (self-loops)
2. Phân biệt cạnh có hướng và vô hướng
3. Cạnh bội với hướng khác nhau
4. Chu trình trong đồ thị hỗn hợp

5. Độ phức tạp tính toán cao

Giải pháp:

- Cấu trúc dữ liệu phức hợp để lưu thông tin cạnh
- Xử lý riêng biệt cho từng loại cạnh
- Thuật toán DFS mở rộng với nhiều trường hợp
- Phân loại cạnh chi tiết hơn

```
#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <unordered_map>
#include <algorithm>

using namespace std;

struct Edge {
    int to;
    bool isDirected;  // true = c    h    ng , false = v    h    ng
    int id;           // ID duy  n h t   c a   c nh

    Edge(int t, bool dir, int i) : to(t), isDirected(dir), id(i) {}
};

class GeneralGraphDFS {
private:
    int numVertices;
    vector<vector<Edge>> adjList;
    map<pair<int,int>, vector<Edge>> edgeMap;  // (u,v) -> list of edges

    vector<bool> visited;
    vector<int> startTime;
    vector<int> finishTime;
    vector<int> parent;
    vector<int> dfsOrder;
    int timeCounter;
    int edgeIdCounter;

    // Th ng tin chi  t i t   v    c nh
    struct EdgeInfo {
        int from, to, id;
        bool isDirected;
        string type;
        bool isSelfLoop;

        EdgeInfo(int f, int t, int i, bool dir, string typ, bool self)
            : from(f), to(t), id(i), isDirected(dir), type(typ),
   isSelfLoop(self) {}
    };

    % T i p   t c   t    p h n       c  ...
```

```
44
45      vector<EdgeInfo> edgeInfoList;
46
47      //    m     s    khuy n   t i    m i     nh
48      vector<int> selfLoopCount;
49
50  public:
51      GeneralGraphDFS(int n) : numVertices(n), timeCounter(0),
    edgeIdCounter(0) {
52          adjList.resize(n);
53          selfLoopCount.resize(n, 0);
54          resetArrays();
55      }
56
57      void resetArrays() {
58          visited.assign(numVertices, false);
59          startTime.assign(numVertices, 0);
60          finishTime.assign(numVertices, 0);
61          parent.assign(numVertices, -1);
62          dfsOrder.clear();
63          edgeInfoList.clear();
64          timeCounter = 0;
65      }
66
67      void addEdge(int u, int v, bool isDirected = false) {
68          Edge edge(v, isDirected, edgeIdCounter++);
69          adjList[u].push_back(edge);
70
71          //  X    l    khuy n
72          if (u == v) {
73              selfLoopCount[u]++;
74              cout << "Th  m  khuy n   t i     nh    " << u << " (ID: " <<
    edge.id << ")" << endl;
75          } else {
76              //  C nh    t h    ng  -  th  m  v  o   c   2  c h i u   n u   v
     h    ng
77              if (!isDirected) {
78                  Edge reverseEdge(u, false, edgeIdCounter++);
79                  adjList[v].push_back(reverseEdge);
80              }
81
82              cout << "Th  m   c nh   " << u << " -> " << v
83                   << (isDirected ? " ( c    h    ng )" : " ( v    h    ng )")
84                   << " (ID: " << edge.id << ")" << endl;
85          }
86
87          //  C p   n h t   b n          c nh
88          pair<int,int> edgeKey = {u, v};
89          edgeMap[edgeKey].push_back(edge);
90      }
91
92      // DFS cho         t h    t ng   qu  t
93      void dfsRecursive(int u) {
94          visited[u] = true;
95          startTime[u] = ++timeCounter;
96          dfsOrder.push_back(u);
```

```
97
98          cout << "Th m    nh    " << u << " (start: " << startTime[u] <<
    ")" << endl;
99
100         //  X    l    khuy n   t r    c
101         if (selfLoopCount[u] > 0) {
102             cout << "   Ph t   h i n   " << selfLoopCount[u] << " khuy n
     t i     nh     " << u << endl;
103             // Khuy n  lu n     c     coi  l     c nh      c     b i t
104             edgeInfoList.push_back(EdgeInfo(u, u, -1, true, "SELF_LOOP",
     true));
105         }
106
107         //  D u y t   c  c    c nh    t h    ng
108         for (const Edge& edge : adjList[u]) {
109             int v = edge.to;
110             bool isDirected = edge.isDirected;
111
112             //  B    qua  khuy n  (       x    l        tr  n)
113             if (u == v) continue;
114
115             if (!visited[v]) {
116                 parent[v] = u;
117                 edgeInfoList.push_back(EdgeInfo(u, v, edge.id,
    isDirected, "TREE", false));
118                 cout << "    C nh    c  y: " << u << " -> " << v
119                      << (isDirected ? " (c     h     ng )" : " (v
     h     ng )") << endl;
120                 dfsRecursive(v);
121             } else {
122                 // Ph n   l o i    c nh    p h c    t p
123                 string edgeType = classifyEdge(u, v, isDirected);
124                 edgeInfoList.push_back(EdgeInfo(u, v, edge.id,
    isDirected, edgeType, false));
125
126                 cout << "    C nh    " << edgeType << ": " << u << " -> "
    << v
127                      << (isDirected ? " (c     h     ng )" : " (v
     h     ng )");
128
129                 if (edgeType == "BACK") {
130                     cout << " (chu  tr  nh!)";
131                 }
132                 cout << endl;
133             }
134         }
135
136         finishTime[u] = ++timeCounter;
137         cout << " K t    th  c     nh     " << u << " (finish: " <<
    finishTime[u] << ")" << endl;
138     }
139
140 private:
141     string classifyEdge(int u, int v, bool isDirected) {
142         // Ph n   l o i    c nh    d a    tr  n    t h i   gian  v     h     ng
143         if (startTime[v] < startTime[u] && finishTime[v] == 0) {
```

```
144             return "BACK";  //  C nh   n g   c  -  t o  chu tr nh
145         }
146
147         if (startTime[v] > startTime[u]) {
148             return "FORWARD";  //  C nh  xu i
149         }
150
151         if (finishTime[v] < startTime[u]) {
152             return "CROSS";  //  C nh  ch o
153         }
154
155         //  T r   ng   h p    c    b i t  cho  c nh  v   h   ng
156         if (!isDirected && parent[u] == v) {
157             return "PARENT";  //  C nh   v   cha (kh ng t nh chu
    tr nh)
158         }
159
160         return "UNKNOWN";
161     }
162
163 public:
164     void dfsComplete() {
165         resetArrays();
166
167         cout << "DFS to n  b         t h   t ng  qu t:" << endl;
168         int componentCount = 0;
169
170         for (int i = 0; i < numVertices; i++) {
171             if (!visited[i]) {
172                 componentCount++;
173                 cout << "\nTh nh  p h n  li n  th ng " <<
    componentCount << ":" << endl;
174                 dfsRecursive(i);
175             }
176         }
177
178         cout << "\ n T ng  s   th nh  p h n  li n  th ng: " <<
    componentCount << endl;
179         analyzeComplexGraph();
180     }
181
182     void analyzeComplexGraph() {
183         cout << "\nPh n  t ch       t h   t ng  qu t:" << endl;
184
185         int treeEdges = 0, backEdges = 0, forwardEdges = 0, crossEdges =
    0;
186         int selfLoops = 0, directedEdges = 0, undirectedEdges = 0;
187         bool hasCycle = false;
188
189         for (const EdgeInfo& info : edgeInfoList) {
190             if (info.isSelfLoop) {
191                 selfLoops++;
192                 continue;
193             }
194
195             if (info.isDirected) {
```

```
196                 directedEdges++;
197             } else {
198                 undirectedEdges++;
199             }
200 
201             if (info.type == "TREE") treeEdges++;
202             else if (info.type == "BACK") { backEdges++; hasCycle = true
    ; }
203             else if (info.type == "FORWARD") forwardEdges++;
204             else if (info.type == "CROSS") crossEdges++;
205         }
206 
207         cout << " T h ng  k    c nh :" << endl;
208         cout << "-  C nh  c  y: " << treeEdges << endl;
209         cout << "-  C nh  n g  c : " << backEdges << endl;
210         cout << "-  C nh  xu i: " << forwardEdges << endl;
211         cout << "-  C nh  ch o: " << crossEdges << endl;
212         cout << "- Khuy n: " << selfLoops << endl;
213         cout << "-  C nh  c   h   ng : " << directedEdges << endl;
214         cout << "-  C nh  v   h   ng : " << undirectedEdges << endl;
215 
216         cout << "\ n      t h   c   chu tr nh: " << (hasCycle ? "C  " :
    "Kh ng") << endl;
217 
218         // Ph n  t ch  t nh  c h t    c    b i t
219         analyzeSpecialProperties();
220     }
221 
222     void analyzeSpecialProperties() {
223         cout << "\nPh n  t ch  t nh  c h t    c    b i t :" << endl;
224 
225         //  K i m  tra       t h  Euler
226         bool isEulerian = checkEulerian();
227         cout << "       t h  Euler: " << (isEulerian ? "C  " : "Kh ng")
     << endl;
228 
229         //  K i m  tra  t nh  li n  th ng  m nh  (cho  p h n  c   h   ng
    )
230         bool isStronglyConnected = checkStrongConnectivity();
231         cout << "Li n  th ng  m nh : " << (isStronglyConnected ? "C  "
    : "Kh ng") << endl;
232 
233         //   m    s    b c   c a   m i    nh
234         printDegreeInformation();
235     }
236 
237     bool checkEulerian() {
238         //   n   g i n  h a:  k i m  tra  b c   c h n  cho  t t   c
      nh
239         for (int i = 0; i < numVertices; i++) {
240             int degree = adjList[i].size() + selfLoopCount[i];
241             if (degree % 2 != 0) {
242                 return false;
243             }
244         }
245         return true;
```

```
246     }
247
248     bool checkStrongConnectivity() {
249         // K i m  tra   n    g i n :  t t   c      nh    c    t h     n
        c     t     nh    0
250         resetArrays();
251         dfsRecursive(0);
252
253         for (int i = 0; i < numVertices; i++) {
254             if (!visited[i]) {
255                 return false;
256             }
257         }
258         return true;
259     }
260
261     void printDegreeInformation() {
262         cout << "\nTh ng tin  b c     nh   :" << endl;
263         cout << "   nh   \ t B c  v  o\ t B c  ra\tKhuy n" << endl;
264         cout << "----\t-------\t------\t------" << endl;
265
266         for (int i = 0; i < numVertices; i++) {
267             int inDegree = 0, outDegree = 0;
268
269             // T nh  b c  ra
270             outDegree = adjList[i].size();
271
272             // T nh  b c  v  o
273             for (int j = 0; j < numVertices; j++) {
274                 for (const Edge& edge : adjList[j]) {
275                     if (edge.to == i && j != i) {
276                         inDegree++;
277                     }
278                 }
279             }
280
281             cout << i << "\t" << inDegree << "\t" << outDegree
282                  << "\t" << selfLoopCount[i] << endl;
283         }
284     }
285
286     void printGeneralGraph() {
287         cout << " B i u  d i n        t h   t ng  qu  t:" << endl;
288
289         for (int i = 0; i < numVertices; i++) {
290             cout << "  nh   " << i << ":" << endl;
291
292             if (selfLoopCount[i] > 0) {
293                 cout << "  Khuy n: " << selfLoopCount[i] << endl;
294             }
295
296             cout << "  K  : ";
297             for (const Edge& edge : adjList[i]) {
298                 if (edge.to != i) {  // Kh ng in khuy n       y
299                     cout << edge.to << (edge.isDirected ? "(   )" : "(
    )") << " ";
```

```
300                 }
301             }
302             cout << endl;
303         }
304         cout << endl;
305     }
306
307     // T m  t t  c  chu tr nh trong  t h  t ng qu t
308     void findAllCycles() {
309         cout << "T m  t t  c  chu tr nh trong  t h :" << endl;
310
311         vector<vector<int>> cycles;
312         vector<int> currentPath;
313         vector<bool> inPath(numVertices, false);
314
315         for (int i = 0; i < numVertices; i++) {
316             findCyclesFromVertex(i, currentPath, inPath, cycles);
317         }
318
319         if (cycles.empty()) {
320             cout << "Kh ng  c  chu tr nh" << endl;
321         } else {
322             cout << "T m  t h y  " << cycles.size() << " chu tr nh:" <<
    endl;
323             for (int i = 0; i < cycles.size(); i++) {
324                 cout << "Chu tr nh " << i + 1 << ": ";
325                 for (int j = 0; j < cycles[i].size(); j++) {
326                     cout << cycles[i][j];
327                     if (j < cycles[i].size() - 1) cout << " -> ";
328                 }
329                 cout << " -> " << cycles[i][0] << endl;
330             }
331         }
332     }
333
334 private:
335     void findCyclesFromVertex(int start, vector<int>& path,
336                               vector<bool>& inPath, vector<vector<int>>&
    cycles) {
337         path.push_back(start);
338         inPath[start] = true;
339
340         for (const Edge& edge : adjList[start]) {
341             int next = edge.to;
342
343             if (next == start) continue;  //  B  qua khuy n
344
345             if (inPath[next]) {
346                 // T m  t h y  chu tr nh
347                 auto it = find(path.begin(), path.end(), next);
348                 if (it != path.end()) {
349                     vector<int> cycle(it, path.end());
350                     cycles.push_back(cycle);
351                 }
352             } else if (find(path.begin(), path.end(), next) == path.end
    ()) {
```

```
                findCyclesFromVertex(next, path, inPath, cycles);
            }
        }

        path.pop_back();
        inPath[start] = false;
    }
};

// Demo cho        t h    t ng  qu  t
void demonstrateGeneralGraphDFS() {
    cout << "=== DEMO DFS CHO        T H    T NG  QU  T ===" << endl <<
    endl;

    GeneralGraphDFS graph(6);

    //  T o         t h    t ng  qu  t  p h c  t p
    cout << " T o         t h    t ng  qu  t:" << endl;

    //  C nh  v    h    ng
    graph.addEdge(0, 1, false);
    graph.addEdge(1, 2, false);

    //  C nh  c    h    ng
    graph.addEdge(2, 3, true);
    graph.addEdge(3, 4, true);

    //  C nh   b i
    graph.addEdge(0, 1, false);  //  C nh   b i  v    h    ng
    graph.addEdge(2, 3, true);   //  C nh   b i  c    h    ng

    // Khuy n
    graph.addEdge(4, 4, true);   // Khuy n  c    h    ng
    graph.addEdge(5, 5, false);  // Khuy n  v    h    ng

    //  C nh   t o  chu  tr  nh
    graph.addEdge(4, 1, true);   // Chu  tr  nh  c    h    ng
    graph.addEdge(3, 0, false);  // Chu  tr  nh  h n   h p

    cout << endl;
    graph.printGeneralGraph();

    // DFS  tr  n        t h    t ng  qu  t
    graph.dfsComplete();

    // T  m  chu  tr  nh
    graph.findAllCycles();
}
```

Listing 5: DFS cho đồ thị tổng quát - C++

```
from collections import defaultdict
from typing import List, Dict, Set, Tuple, Optional
import copy

class Edge:
```

```
    """ L p   b i u   d i n   c nh  trong       t h    t ng  qu t"""

    def __init__(self, to: int, is_directed: bool, edge_id: int):
        self.to = to
        self.is_directed = is_directed
        self.id = edge_id

class EdgeInfo:
    """Th ng tin chi  t i t   v    c nh """

    def __init__(self, from_vertex: int, to_vertex: int, edge_id: int,
                 is_directed: bool, edge_type: str, is_self_loop: bool):
        self.from_vertex = from_vertex
        self.to_vertex = to_vertex
        self.id = edge_id
        self.is_directed = is_directed
        self.type = edge_type
        self.is_self_loop = is_self_loop

class GeneralGraphDFS:
    """DFS cho       t h    t ng  qu t"""

    def __init__(self, num_vertices: int):
        self.num_vertices = num_vertices
        self.adj_list = [[] for _ in range(num_vertices)]
        self.edge_map = defaultdict(list)
        self.self_loop_count = [0] * num_vertices
        self.edge_id_counter = 0
        self.reset_arrays()

    def reset_arrays(self):
        self.visited = [False] * self.num_vertices
        self.start_time = [0] * self.num_vertices
        self.finish_time = [0] * self.num_vertices
        self.parent = [-1] * self.num_vertices
        self.dfs_order = []
        self.edge_info_list = []
        self.time_counter = 0

    def add_edge(self, u: int, v: int, is_directed: bool = False):
        """Th m  c nh  v o       t h    t ng  qu t"""
        edge = Edge(v, is_directed, self.edge_id_counter)
        self.edge_id_counter += 1
        self.adj_list[u].append(edge)

        #  X   l  khuy n
        if u == v:
            self.self_loop_count[u] += 1
            print(f"Th m khuy n  t i    nh   {u} (ID: {edge.id})")
        else:
            #  C nh  t h  ng - th m v o  c  2 c h i u  n u  v
    h   ng
            if not is_directed:
                reverse_edge = Edge(u, False, self.edge_id_counter)
                self.edge_id_counter += 1
                self.adj_list[v].append(reverse_edge)
```

```
61
62              direction_str = "c    h    ng " if is_directed else "v
    h    ng "
63              print(f"Th  m   c nh  {u} -> {v} ({direction_str}) (ID: {edge
    .id})")
64
65          #  C p   n h t   b n         c nh
66          edge_key = (u, v)
67          self.edge_map[edge_key].append(edge)
68
69      def dfs_recursive(self, u: int):
70          """DFS       quy cho       t h    t ng   qu t"""
71          self.visited[u] = True
72          self.time_counter += 1
73          self.start_time[u] = self.time_counter
74          self.dfs_order.append(u)
75
76          print(f"Th  m    nh    {u} (start: {self.start_time[u]})")
77
78          #  X    l   khuy n  t r   c
79          if self.self_loop_count[u] > 0:
80              print(f"  Ph  t   h i n  {self.self_loop_count[u]} khuy n
    t i     nh    {u}")
81              self.edge_info_list.append(
82                  EdgeInfo(u, u, -1, True, "SELF_LOOP", True)
83              )
84
85          #  D u y t   c  c   c nh   t h   ng
86          for edge in self.adj_list[u]:
87              v = edge.to
88              is_directed = edge.is_directed
89
90              #  B   qua khuy n (      x   l       tr n)
91              if u == v:
92                  continue
93
94              if not self.visited[v]:
95                  self.parent[v] = u
96                  self.edge_info_list.append(
97                      EdgeInfo(u, v, edge.id, is_directed, "TREE", False)
98                  )
99                  direction_str = "c    h    ng " if is_directed else "v
    h    ng "
100                 print(f"   C nh   c  y: {u} -> {v} ({direction_str})")
101                 self.dfs_recursive(v)
102             else:
103                 # Ph n   l o i   c nh   p h c   t p
104                 edge_type = self._classify_edge(u, v, is_directed)
105                 self.edge_info_list.append(
106                     EdgeInfo(u, v, edge.id, is_directed, edge_type,
    False)
107                 )
108
109                 direction_str = "c    h    ng " if is_directed else "v
    h    ng "
110                 print(f"   C nh  {edge_type}: {u} -> {v} ({direction_str
```

```
    })", end="")
111
112                 if edge_type == "BACK":
113                     print(" (chu tr nh!)", end="")
114                 print()
115
116         self.time_counter += 1
117         self.finish_time[u] = self.time_counter
118         print(f" K t  th c  nh  {u} (finish: {self.finish_time[u]})"
    )
119
120     def _classify_edge(self, u: int, v: int, is_directed: bool) -> str:
121         """Ph n l o i  c nh  d a tr n  t h i gian v  h  ng """
122         if (self.start_time[v] < self.start_time[u] and
123             self.finish_time[v] == 0):
124             return "BACK"  #  C nh  n g  c - t o  chu tr nh
125
126         if self.start_time[v] > self.start_time[u]:
127             return "FORWARD"  #  C nh  xu i
128
129         if self.finish_time[v] < self.start_time[u]:
130             return "CROSS"  #  C nh  ch o
131
132         #  T r  ng  h p  c  b i t cho  c nh  v  h  ng
133         if not is_directed and self.parent[u] == v:
134             return "PARENT"  #  C nh  v  cha (kh ng t nh chu tr nh)
135
136         return "UNKNOWN"
137
138     def dfs_complete(self):
139         """DFS to n  b  t h  t ng  qu t"""
140         self.reset_arrays()
141
142         print("DFS to n  b  t h  t ng  qu t:")
143         component_count = 0
144
145         for i in range(self.num_vertices):
146             if not self.visited[i]:
147                 component_count += 1
148                 print(f"\nTh nh  p h n  li n  th ng {component_count}:"
    )
149                 self.dfs_recursive(i)
150
151         print(f"\ n T ng  s  th nh  p h n  li n  th ng: {
    component_count}")
152         self.analyze_complex_graph()
153
154     def analyze_complex_graph(self):
155         """Ph n t ch  t h  t ng  qu t"""
156         print("\nPh n t ch  t h  t ng  qu t:")
157
158         tree_edges = back_edges = forward_edges = cross_edges = 0
159         self_loops = directed_edges = undirected_edges = 0
160         has_cycle = False
161
162         for info in self.edge_info_list:
```

```
163             if info.is_self_loop:
164                 self_loops += 1
165                 continue
166 
167             if info.is_directed:
168                 directed_edges += 1
169             else:
170                 undirected_edges += 1
171 
172             if info.type == "TREE":
173                 tree_edges += 1
174             elif info.type == "BACK":
175                 back_edges += 1
176                 has_cycle = True
177             elif info.type == "FORWARD":
178                 forward_edges += 1
179             elif info.type == "CROSS":
180                 cross_edges += 1
181 
182         print(" T h ng  k    c nh :")
183         print(f"-  C nh  c  y: {tree_edges}")
184         print(f"-  C nh  n g  c : {back_edges}")
185         print(f"-  C nh  xu i: {forward_edges}")
186         print(f"-  C nh  ch o: {cross_edges}")
187         print(f"- Khuy n: {self_loops}")
188         print(f"-  C nh  c  h  ng : {directed_edges}")
189         print(f"-  C nh  v  h  ng : {undirected_edges}")
190 
191         print(f"\ n  t h  c  chu tr nh: {'C  ' if has_cycle else '
    Kh ng'}")
192 
193         # Ph n t ch t nh  c h t  c  b i t
194         self.analyze_special_properties()
195 
196     def analyze_special_properties(self):
197         """Ph n t ch t nh  c h t  c  b i t """
198         print("\nPh n t ch t nh  c h t  c  b i t :")
199 
200         #  K i m tra  t h  Euler
201         is_eulerian = self.check_eulerian()
202         print(f"  t h  Euler: {'C  ' if is_eulerian else 'Kh ng'}"
    )
203 
204         #  K i m  tra t nh li n th ng  m nh  (cho  p h n  c  h  ng )
205         is_strongly_connected = self.check_strong_connectivity()
206         print(f"Li n th ng  m nh : {'C  ' if is_strongly_connected
    else 'Kh ng'}")
207 
208         #  m  s  b c  c a  m i  nh
209         self.print_degree_information()
210 
211     def check_eulerian(self) -> bool:
212         """ K i m  tra t nh Euler  n  g i n """
213         for i in range(self.num_vertices):
214             degree = len(self.adj_list[i]) + self.self_loop_count[i]
215             if degree % 2 != 0:
```

```
216                 return False
217         return True
218 
219     def check_strong_connectivity(self) -> bool:
220         """ K i m  tra t nh  li n  th ng  m nh    n    g i n """
221         self.reset_arrays()
222         self.dfs_recursive(0)
223 
224         for i in range(self.num_vertices):
225             if not self.visited[i]:
226                 return False
227         return True
228 
229     def print_degree_information(self):
230         """In  th ng tin  b c    nh  """
231         print("\nTh ng tin  b c    nh  :")
232         print("  nh  \ t B c  v  o\ t B c  ra\tKhuy n")
233         print("----\t-------\t------\t------")
234 
235         for i in range(self.num_vertices):
236             in_degree = out_degree = 0
237 
238             # T nh  b c  ra
239             out_degree = len(self.adj_list[i])
240 
241             # T nh  b c  v  o
242             for j in range(self.num_vertices):
243                 for edge in self.adj_list[j]:
244                     if edge.to == i and j != i:
245                         in_degree += 1
246 
247             print(f"{i}\t{in_degree}\t{out_degree}\t{self.
    self_loop_count[i]}")
248 
249     def print_general_graph(self):
250         """In  b i u  d i n  t h  t ng  qu  t"""
251         print(" B i u  d i n  t h  t ng  qu  t:")
252 
253         for i in range(self.num_vertices):
254             print(f"  nh  {i}:")
255 
256             if self.self_loop_count[i] > 0:
257                 print(f"  Khuy n: {self.self_loop_count[i]}")
258 
259             print("  K  : ", end="")
260             for edge in self.adj_list[i]:
261                 if edge.to != i:  # Kh ng in khuy n  y
262                     direction = "(  )" if edge.is_directed else "(  )"
263                     print(f"{edge.to}{direction} ", end="")
264             print()
265         print()
266 
267     def find_all_cycles(self):
268         """T m  t t  c  chu tr nh trong  t h  t ng  qu  t"""
269         print("T m  t t  c  chu tr nh trong  t h  :")
270 
```

```
271         cycles = []
272         current_path = []
273         in_path = [False] * self.num_vertices
274 
275         for i in range(self.num_vertices):
276             self._find_cycles_from_vertex(i, current_path, in_path,
    cycles)
277 
278         if not cycles:
279             print("Kh ng  c   chu tr nh")
280         else:
281             print(f"T m  t h y  {len(cycles)} chu tr nh:")
282             for i, cycle in enumerate(cycles, 1):
283                 cycle_str = ' -> '.join(map(str, cycle))
284                 print(f"Chu tr nh {i}: {cycle_str} -> {cycle[0]}")
285 
286     def _find_cycles_from_vertex(self, start: int, path: List[int],
287                                  in_path: List[bool], cycles: List[List[
    int]]):
288         """Helper function       t m chu tr nh  t    m t     nh   """
289         path.append(start)
290         in_path[start] = True
291 
292         for edge in self.adj_list[start]:
293             next_vertex = edge.to
294 
295             if next_vertex == start:  #  B   qua khuy n
296                 continue
297 
298             if in_path[next_vertex]:
299                 # T m  t h y  chu tr nh
300                 try:
301                     cycle_start = path.index(next_vertex)
302                     cycle = path[cycle_start:]
303                     cycles.append(cycle.copy())
304                 except ValueError:
305                     pass
306             elif next_vertex not in path:
307                 self._find_cycles_from_vertex(next_vertex, path, in_path
    , cycles)
308 
309         path.pop()
310         in_path[start] = False
311 
312 # Demo cho        t h    t ng  qu t
313 def demonstrate_general_graph_dfs():
314     print("=== DEMO DFS CHO        T H    T NG  QU T ===\n")
315 
316     graph = GeneralGraphDFS(6)
317 
318     #  T o         t h    t ng  qu t  p h c   t p
319     print(" T o         t h    t ng  qu t:")
320 
321     #  C nh  v   h   ng
322     graph.add_edge(0, 1, False)
323     graph.add_edge(1, 2, False)
```

```
    # C nh c h ng
    graph.add_edge(2, 3, True)
    graph.add_edge(3, 4, True)

    # C nh b i
    graph.add_edge(0, 1, False)  # C nh b i v h ng
    graph.add_edge(2, 3, True)   # C nh b i c h ng

    # Khuy n
    graph.add_edge(4, 4, True)   # Khuy n c h ng
    graph.add_edge(5, 5, False)  # Khuy n v h ng

    # C nh t o chu tr nh
    graph.add_edge(4, 1, True)   # Chu tr nh c h ng
    graph.add_edge(3, 0, False)  # Chu tr nh h n h p

    print()
    graph.print_general_graph()

    # DFS tr n t h t ng qu t
    graph.dfs_complete()

    # T m chu tr nh
    graph.find_all_cycles()

if __name__ == "__main__":
    demonstrate_general_graph_dfs()
```

Listing 6: DFS cho đồ thị tổng quát - Python

6 So sánh và đánh giá các thuật toán

6.1 Bảng so sánh độ phức tạp

Loại đồ thị	Thời gian	Không gian	Cài đặt	Ứng dụng
Đồ thị đơn	$O(V+E)$	$O(V)$	Đơn giản	Cơ bản
Đa đồ thị	$O(V+E)$	$O(V+E)$	Trung bình	Cạnh bội
Đồ thị tổng quát	$O(V+E+L)$	$O(V+E+L)$	Phức tạp	Đầy đủ

Trong đó: V = số đỉnh, E = số cạnh, L = số khuyên

6.2 Ưu điểm và nhược điểm

DFS trên đồ thị đơn:

- **Ưu điểm:** Đơn giản, hiệu quả, dễ cài đặt
- **Nhược điểm:** Hạn chế với loại đồ thị cơ bản
- **Phù hợp:** Bài toán cơ bản, học thuật toán

DFS trên đa đồ thị:

- **Ưu điểm:** Xử lý cạnh bội, thực tế hơn
- **Nhược điểm:** Phức tạp hơn, tốn bộ nhớ
- **Phù hợp:** Mạng giao thông, mạng máy tính

DFS trên đồ thị tổng quát:

- **Ưu điểm:** Đầy đủ tính năng, tổng quát
- **Nhược điểm:** Phức tạp cao, khó debug
- **Phù hợp:** Nghiên cứu, ứng dụng chuyên sâu

7 Ứng dụng thực tế

7.1 Trong khoa học máy tính

1. Phát hiện chu trình:

- Kiểm tra deadlock trong hệ điều hành
- Phát hiện circular dependency trong software
- Validation trong compiler design

2. Sắp xếp topo:

- Build systems (Make, Maven, Gradle)
- Task scheduling
- Course prerequisite ordering

3. Thành phần liên thông:

- Social network analysis
- Image processing (connected components)
- Network reliability analysis

7.2 Trong các lĩnh vực khác

1. Mạng giao thông:

- Tìm đường trong GPS navigation
- Phân tích connectivity của road networks
- Traffic flow optimization

2. Sinh học:

- Protein interaction networks
- Gene regulatory networks
- Phylogenetic tree construction

3. Mạng xã hội:

- Community detection
- Influence propagation
- Friend recommendation systems

8 Kết luận

8.1 Tổng kết

Thuật toán tìm kiếm theo chiều sâu (DFS) là một trong những thuật toán cơ fundamental nhất trong lý thuyết đồ thị. Qua ba bài toán đã trình bày:

- **DFS trên đồ thị đơn:** Cung cấp nền tảng vững chắc để hiểu thuật toán
- **DFS trên đa đồ thị:** Mở rộng khả năng xử lý các bài toán thực tế
- **DFS trên đồ thị tổng quát:** Đạt đến tính tổng quát cao nhất