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

trees,layered

THUẬT TOÁN TÌM KIẾM THEO CHIỀU RỘNG (BREADTH-FIRST SEARCH - BFS)

Toán Tổ Hợp và Lý Thuyết Đồ Thị

1 Lý thuyết cơ bản về tìm kiếm trong đồ thị

1.1 Định nghĩa tìm kiếm trong đồ thị

Tìm kiếm trong đồ thị là quá trình duyệt qua các đỉnh của đồ thị một cách có hệ thống để:

- Tìm một đỉnh cụ thể
- Khám phá tất cả các đỉnh có thể đến được từ một đỉnh xuất phát
- Xây dựng cây khung của đồ thị
- Giải quyết các bài toán ứng dụng khác

1.2 Hai phương pháp tìm kiếm chính

- **Tìm kiếm theo chiều sâu (DFS - Depth-First Search)**: Đi sâu vào một nhánh trước khi quay lại
- **Tìm kiếm theo chiều rộng (BFS - Breadth-First Search)**: Khám phá tất cả đỉnh ở mức hiện tại trước khi chuyển sang mức tiếp theo

1.3 Đặc điểm của BFS

BFS có những đặc điểm quan trọng sau:

- Sử dụng cấu trúc dữ liệu **hàng đợi (Queue)** - FIFO (First In, First Out)
- Đảm bảo tìm được đường đi ngắn nhất (theo số cạnh) trong đồ thị không trọng số
- Duyệt đồ thị theo từng lớp (level-by-level)
- Phù hợp cho việc tìm kiếm trên đồ thị rộng và nông

2 Mô tả bài toán 8

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

Input:

- Đồ thị G được biểu diễn bằng danh sách kề hoặc ma trận kề
- Đỉnh xuất phát $s \in V$

Output:

- Thứ tự duyệt các đỉnh theo BFS
- Khoảng cách ngắn nhất từ s đến các đỉnh khác
- Cây BFS (BFS tree) được tạo ra trong quá trình duyệt

Ràng buộc:

- $1 \leq n \leq 10^5$ (số đỉnh)
- $0 \leq m \leq 10^6$ (số cạnh)
- Đồ thị có thể liên thông hoặc không liên thông

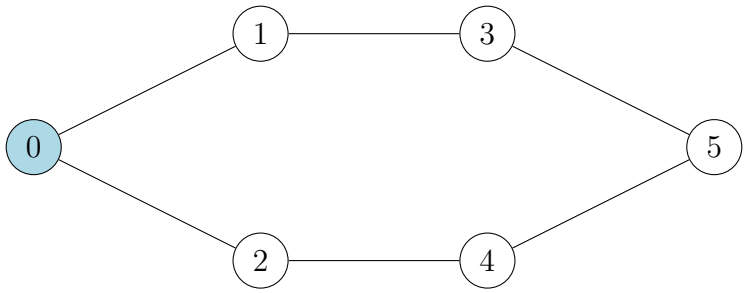
3 Ý tưởng và giải pháp

3.1 Ý tưởng chính

BFS hoạt động dựa trên nguyên lý *"khám phá từng lớp một"*:

1. Bắt đầu từ đỉnh xuất phát s, đánh dấu s đã được thăm
2. Đưa s vào hàng đợi
3. Lặp lại cho đến khi hàng đợi rỗng:
    - Lấy đỉnh u ở đầu hàng đợi
    - Duyệt tất cả đỉnh v kề với u
    - Nếu v chưa được thăm: đánh dấu v đã thăm, đưa v vào hàng đợi

3.2 Minh họa thuật toán



Đồ thị mẫu - BFS bắt đầu từ đỉnh 0

Quá trình BFS từ đỉnh 0:

- **Bước 1:** Queue = [0], Visited = {0}, Level 0: [0]
- **Bước 2:** Queue = [1, 2], Visited = {0, 1, 2}, Level 1: [1, 2]
- **Bước 3:** Queue = [2, 3], Visited = {0, 1, 2, 3}, Level 2: [3]
- **Bước 4:** Queue = [3, 4], Visited = {0, 1, 2, 3, 4}, Level 2: [3, 4]
- **Bước 5:** Queue = [4, 5], Level 3: [5]
- **Kết thúc:** Thứ tự duyệt: $0 \to 1 \to 2 \to 3 \to 4 \to 5$

3.3 Tính chất quan trọng của BFS

1. **Đường đi ngắn nhất:** BFS đảm bảo tìm được đường đi có số cạnh ít nhất từ đỉnh xuất phát đến mọi đỉnh khác
2. **Cây BFS:** Các cạnh được sử dụng để đến đỉnh mới lần đầu tạo thành một cây khung
3. **Phân lớp đỉnh:** Các đỉnh được chia thành các lớp dựa trên khoảng cách từ đỉnh xuất phát

4 Thuật toán chi tiết

4.1 Pseudo-code

Algorithm 1 Breadth-First Search

Require: Graph $G = (V, E)$, starting vertex s
Ensure: BFS traversal order, distances, BFS tree

```
Initialize queue Q and set visited[v] = false for all v ∈ V
Initialize distance[v] = ∞ and parent[v] = −1 for all v ∈ V
visited[s] = true, distance[s] = 0
Q.enqueue(s)
while Q is not empty do
    u = Q.dequeue()
    Process vertex u (e.g., print u)
    for each vertex v adjacent to u do
        if visited[v] = false then
            visited[v] = true
            distance[v] = distance[u] + 1
            parent[v] = u
            Q.enqueue(v)
        end if
    end for
end while
```

4.2 Phân tích thuật toán

Độ phức tạp thời gian:

- Mỗi đỉnh được đưa vào hàng đợi đúng một lần: $O(V)$
- Mỗi cạnh được kiểm tra đúng một lần (với danh sách kề): $O(E)$
- **Tổng cộng:** $O(V + E)$

Độ phức tạp không gian:

- Mảng visited, distance, parent: $O(V)$
- Hàng đợi (worst case): $O(V)$
- **Tổng cộng:** $O(V)$

5 Cài đặt bằng C++

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
#include <iomanip>

using namespace std;

class BFS {
private:
    int numVertices;
    vector<vector<int>> adjList;
    vector<bool> visited;
    vector<int> distance;
    vector<int> parent;
    vector<int> bfsOrder;

public:
    // Constructor
    BFS(int n) : numVertices(n) {
        adjList.resize(n);
        resetArrays();
    }

    // Reset cac mang trang thai
    void resetArrays() {
        visited.assign(numVertices, false);
        distance.assign(numVertices, INT_MAX);
        parent.assign(numVertices, -1);
        bfsOrder.clear();
    }

    // Them canh vao do thi
    void addEdge(int u, int v) {
        adjList[u].push_back(v);
        adjList[v].push_back(u); // Do thi vo huong
    }

    // Thuat toan BFS chinh
    void bfs(int start) {
        resetArrays();

        queue<int> q;
        visited[start] = true;
        distance[start] = 0;
        q.push(start);

        cout << "Bat dau BFS tu dinh " << start << ":" << endl;
        cout << "Thu tu duyet: ";

        while (!q.empty()) {
            int u = q.front();
            q.pop();

            // Xu ly dinh u
            bfsOrder.push_back(u);
```

```
            cout << u << " ";

            // Duyet tat ca dinh ke voi u
            for (int v : adjList[u]) {
                if (!visited[v]) {
                    visited[v] = true;
                    distance[v] = distance[u] + 1;
                    parent[v] = u;
                    q.push(v);
                }
            }
        }
        cout << endl << endl;
    }

    // BFS cho toan bo do thi (xu ly thanh phan lien thong)
    void bfsComplete() {
        resetArrays();

        cout << "BFS toan bo do thi:" << endl;
        int componentCount = 0;

        for (int i = 0; i < numVertices; i++) {
            if (!visited[i]) {
                componentCount++;
                cout << "Thanh phan lien thong " << componentCount << ":
    ";
                bfsComponent(i);
                cout << endl;
            }
        }

        cout << "Tong so thanh phan lien thong: " << componentCount <<
    endl << endl;
    }

    // BFS cho mot thanh phan lien thong
    void bfsComponent(int start) {
        queue<int> q;
        visited[start] = true;
        distance[start] = 0;
        q.push(start);

        while (!q.empty()) {
            int u = q.front();
            q.pop();
            cout << u << " ";

            for (int v : adjList[u]) {
                if (!visited[v]) {
                    visited[v] = true;
                    distance[v] = distance[u] + 1;
                    parent[v] = u;
                    q.push(v);
                }
            }
```

```
        }
    }

    // Tim duong di ngan nhat tu start den target
    vector<int> getShortestPath(int start, int target) {
        bfs(start);

        vector<int> path;
        if (distance[target] == INT_MAX) {
            return path; // Khong co duong di
        }

        // Tai tao duong di tu mang parent
        int current = target;
        while (current != -1) {
            path.push_back(current);
            current = parent[current];
        }

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

    // In thong tin khoang cach
    void printDistances(int start) {
        cout << "Khoang cach tu dinh " << start << ":" << endl;
        cout << setw(6) << "Dinh" << setw(12) << "Khoang cach" << setw
    (10) << "Cha" << endl;
        cout << string(28, '-') << endl;

        for (int i = 0; i < numVertices; i++) {
            cout << setw(6) << i;
            if (distance[i] == INT_MAX) {
                cout << setw(12) << "INF";
            } else {
                cout << setw(12) << distance[i];
            }
            cout << setw(10) << parent[i] << endl;
        }
        cout << endl;
    }

    // In cay BFS
    void printBFSTree(int start) {
        cout << "Cay BFS tu dinh " << start << ":" << endl;

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

    // Kiem tra lien thong
    bool isConnected() {
```

```
        bfs(0);

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

    // Tim dinh xa nhat tu start
    pair<int, int> findFarthestVertex(int start) {
        bfs(start);

        int maxDist = -1;
        int farthestVertex = -1;

        for (int i = 0; i < numVertices; i++) {
            if (distance[i] != INT_MAX && distance[i] > maxDist) {
                maxDist = distance[i];
                farthestVertex = i;
            }
        }

        return {farthestVertex, maxDist};
    }

    // In do thi
    void printGraph() {
        cout << "Bieu dien do thi (danh sach ke):" << endl;
        for (int i = 0; i < numVertices; i++) {
            cout << i << ": ";
            for (int neighbor : adjList[i]) {
                cout << neighbor << " ";
            }
            cout << endl;
        }
        cout << endl;
    }
};

// Cac ham tien ich
class BFSUtils {
public:
    // Tao do thi mau
    static BFS createSampleGraph1() {
        BFS graph(6);

        // Them cac canh
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(1, 3);
        graph.addEdge(2, 4);
        graph.addEdge(3, 5);
        graph.addEdge(4, 5);

```

```
        return graph;
    }

    static BFS createSampleGraph2() {
        BFS graph(8);

        // Tao do thi co nhieu thanh phan lien thong
        graph.addEdge(0, 1);
        graph.addEdge(1, 2);
        graph.addEdge(3, 4);
        graph.addEdge(5, 6);
        graph.addEdge(5, 7);

        return graph;
    }

    // Tim duong kinh cua cay (duong di dai nhat)
    static pair<int, int> findTreeDiameter(BFS& tree) {
        // Buoc 1: Tim dinh xa nhat tu dinh 0
        auto [farthest1, dist1] = tree.findFarthestVertex(0);

        // Buoc 2: Tim dinh xa nhat tu dinh vua tim duoc
        auto [farthest2, diameter] = tree.findFarthestVertex(farthest1);

        return {diameter, farthest2};
    }
};

// Ham demo day du
void demonstrateBFS() {
    cout << "=== DEMO THUAT TOAN BFS ===" << endl << endl;

    // Demo 1: Do thi lien thong
    cout << "1. DO THI LIEN THONG:" << endl;
    auto graph1 = BFSUtils::createSampleGraph1();
    graph1.printGraph();

    // BFS tu dinh 0
    graph1.bfs(0);
    graph1.printDistances(0);
    graph1.printBFSTree(0);

    // Tim duong di ngan nhat
    cout << "Duong di ngan nhat tu 0 den 5:" << endl;
    auto path = graph1.getShortestPath(0, 5);
    for (int i = 0; i < path.size(); i++) {
        cout << path[i];
        if (i < path.size() - 1) cout << " -> ";
    }
    cout << " (Do dai: " << path.size() - 1 << ")" << endl << endl;

    // Kiem tra lien thong
    cout << "Do thi co lien thong khong? "
         << (graph1.isConnected() ? "Co" : "Khong") << endl << endl;

    // Demo 2: Do thi khong lien thong
```

```
    cout << "2. DO THI KHONG LIEN THONG:" << endl;
    auto graph2 = BFSUtils::createSampleGraph2();
    graph2.printGraph();
    graph2.bfsComplete();

    cout << "Do thi co lien thong khong? "
         << (graph2.isConnected() ? "Co" : "Khong") << endl << endl;

    // Demo 3: Tim duong kinh cua cay
    cout << "3. TIM DUONG KINH CAY:" << endl;
    auto tree = BFSUtils::createSampleGraph1(); // Su dung graph1 nhu la cay
    auto [diameter, endpoint] = BFSUtils::findTreeDiameter(tree);
    cout << "Duong kinh cua cay: " << diameter << endl;
    cout << "Mot dau mut cua duong kinh: " << endpoint << endl;
}

int main() {
    demonstrateBFS();

    // Interactive demo
    cout << "=== DEMO TUONG TAC ===" << endl;
    int n, m;
    cout << "Nhap so dinh va so canh: ";
    cin >> n >> m;

    BFS userGraph(n);

    cout << "Nhap " << m << " canh (u v):" << endl;
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        userGraph.addEdge(u, v);
    }

    int start;
    cout << "Nhap dinh bat dau BFS: ";
    cin >> start;

    userGraph.printGraph();
    userGraph.bfs(start);
    userGraph.printDistances(start);
    userGraph.printBFSTree(start);

    return 0;
}
```

Listing 1: Cài đặt BFS đầy đủ bằng C++

6 Cài đặt bằng Python

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

```

class BFS:
    """
    Lop cai dat thuat toan BFS cho do thi
    """

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

    def reset_arrays(self):
        """Reset cac mang trang thai"""
        self.visited = [False] * self.num_vertices
        self.distance = [float('inf')] * self.num_vertices
        self.parent = [-1] * self.num_vertices
        self.bfs_order = []

    def add_edge(self, u: int, v: int):
        """Them canh vao do thi vo huong"""
        self.adj_list[u].append(v)
        self.adj_list[v].append(u)

    def add_directed_edge(self, u: int, v: int):
        """Them canh co huong"""
        self.adj_list[u].append(v)

    def bfs(self, start: int) -> List[int]:
        """
        Thuat toan BFS chinh

        Args:
            start: Dinh bat dau

        Returns:
            Thu tu duyet BFS
        """
        self.reset_arrays()

        queue = deque([start])
        self.visited[start] = True
        self.distance[start] = 0

        print(f"Bat dau BFS tu dinh {start}:")
        print("Thu tu duyet: ", end="")

        while queue:
            u = queue.popleft()

            # Xu ly dinh u
            self.bfs_order.append(u)
            print(u, end=" ")

            # Duyet tat ca dinh ke voi u
            for v in sorted(self.adj_list[u]):  # Sort de dam bao thu tu
    nhat quan
```

```
                if not self.visited[v]:
                    self.visited[v] = True
                    self.distance[v] = self.distance[u] + 1
                    self.parent[v] = u
                    queue.append(v)

        print("\n")
        return self.bfs_order.copy()

    def bfs_complete(self) -> Dict[int, List[int]]:
        """
        BFS toan bo do thi (xu ly thanh phan lien thong)

        Returns:
            Dictionary chua cac thanh phan lien thong
        """
        self.reset_arrays()
        components = {}
        component_count = 0

        print("BFS toan bo do thi:")

        for i in range(self.num_vertices):
            if not self.visited[i]:
                component_count += 1
                component = self._bfs_component(i)
                components[component_count] = component
                print(f"Thanh phan lien thong {component_count}: {' '.
join(map(str, component))}")

        print(f"Tong so thanh phan lien thong: {component_count}\n")
        return components

    def _bfs_component(self, start: int) -> List[int]:
        """BFS cho mot thanh phan lien thong"""
        component = []
        queue = deque([start])
        self.visited[start] = True
        self.distance[start] = 0

        while queue:
            u = queue.popleft()
            component.append(u)

            for v in sorted(self.adj_list[u]):
                if not self.visited[v]:
                    self.visited[v] = True
                    self.distance[v] = self.distance[u] + 1
                    self.parent[v] = u
                    queue.append(v)

        return component

    def get_shortest_path(self, start: int, target: int) -> List[int]:
        """
        Tim duong di ngan nhat tu start den target
```

```
114
115         Args:
116             start: Dinh bat dau
117             target: Dinh dich
118
119         Returns:
120             Duong di ngan nhat (rong neu khong co duong di)
121         """
122         self.bfs(start)
123
124         if self.distance[target] == float('inf'):
125             return []  # Khong co duong di
126
127         # Tai tao duong di tu mang parent
128         path = []
129         current = target
130         while current != -1:
131             path.append(current)
132             current = self.parent[current]
133
134         path.reverse()
135         return path
136
137     def print_distances(self, start: int):
138         """In thong tin khoang cach"""
139         print(f"Khoang cach tu dinh {start}:")
140         print(f"{'Dinh':<6} {'Khoang cach':<12} {'Cha':<10}")
141         print("-" * 28)
142
143         for i in range(self.num_vertices):
144             dist_str = "INF" if self.distance[i] == float('inf') else
    str(self.distance[i])
145             print(f"{i:<6} {dist_str:<12} {self.parent[i]:<10}")
146         print()
147
148     def print_bfs_tree(self, start: int):
149         """In cay BFS"""
150         print(f"Cay BFS tu dinh {start}:")
151
152         for i in range(self.num_vertices):
153             if self.parent[i] != -1:
154                 print(f"{self.parent[i]} -> {i}")
155         print()
156
157     def is_connected(self) -> bool:
158         """Kiem tra do thi co lien thong khong"""
159         self.bfs(0)
160         return all(self.visited)
161
162     def find_farthest_vertex(self, start: int) -> Tuple[int, int]:
163         """
164         Tim dinh xa nhat tu start
165
166         Returns:
167             Tuple (dinh xa nhat, khoang cach)
168         """
```

```
169         self.bfs(start)
170 
171         max_dist = -1
172         farthest_vertex = -1
173 
174         for i in range(self.num_vertices):
175             if self.distance[i] != float('inf') and self.distance[i] >
    max_dist:
176                 max_dist = self.distance[i]
177                 farthest_vertex = i
178 
179         return farthest_vertex, max_dist
180 
181     def print_graph(self):
182         """In bieu dien do thi"""
183         print("Bieu dien do thi (danh sach ke):")
184         for i in range(self.num_vertices):
185             neighbors = ' '.join(map(str, sorted(self.adj_list[i])))
186             print(f"{i}: {neighbors}")
187         print()
188 
189     def get_levels(self, start: int) -> Dict[int, List[int]]:
190         """
191         Lay cac muc trong BFS tree
192 
193         Returns:
194             Dictionary: {level: [list of vertices]}
195         """
196         self.bfs(start)
197         levels = defaultdict(list)
198 
199         for i in range(self.num_vertices):
200             if self.distance[i] != float('inf'):
201                 levels[self.distance[i]].append(i)
202 
203         return dict(levels)
204 
205     def has_path(self, start: int, end: int) -> bool:
206         """Kiem tra co duong di tu start den end khong"""
207         path = self.get_shortest_path(start, end)
208         return len(path) > 0
209 
210     def get_connected_components(self) -> List[List[int]]:
211         """Lay tat ca cac thanh phan lien thong"""
212         components_dict = self.bfs_complete()
213         return list(components_dict.values())
214 
215 
216 class BFSUtils:
217     """Cac ham tien ich cho BFS"""
218 
219     @staticmethod
220     def create_sample_graph1() -> BFS:
221         """Tao do thi mau 1 - lien thong"""
222         graph = BFS(6)
223 
```

```
        edges = [(0, 1), (0, 2), (1, 3), (2, 4), (3, 5), (4, 5)]
        for u, v in edges:
            graph.add_edge(u, v)

        return graph

    @staticmethod
    def create_sample_graph2() -> BFS:
        """Tao do thi mau 2 - khong lien thong"""
        graph = BFS(8)

        edges = [(0, 1), (1, 2), (3, 4), (5, 6), (5, 7)]
        for u, v in edges:
            graph.add_edge(u, v)

        return graph

    @staticmethod
    def create_grid_graph(rows: int, cols: int) -> BFS:
        """Tao do thi luoi"""
        graph = BFS(rows * cols)

        for i in range(rows):
            for j in range(cols):
                current = i * cols + j

                # Ket noi voi dinh ben phai
                if j < cols - 1:
                    right = i * cols + (j + 1)
                    graph.add_edge(current, right)

                # Ket noi voi dinh ben duoi
                if i < rows - 1:
                    down = (i + 1) * cols + j
                    graph.add_edge(current, down)

        return graph

    @staticmethod
    def find_tree_diameter(tree: BFS) -> Tuple[int, Tuple[int, int]]:
        """
        Tim duong kinh cua cay (duong di dai nhat)

        Returns:
            Tuple (do dai duong kinh, (dau mut 1, dau mut 2))
        """
        # Buoc 1: Tim dinh xa nhat tu dinh 0
        farthest1, dist1 = tree.find_farthest_vertex(0)

        # Buoc 2: Tim dinh xa nhat tu dinh vua tim duoc
        farthest2, diameter = tree.find_farthest_vertex(farthest1)

        return diameter, (farthest1, farthest2)

    @staticmethod
    def find_center_vertices(tree: BFS) -> List[int]:
```

```
        """Tim cac dinh trung tam cua cay"""
        n = tree.num_vertices
        if n == 1:
            return [0]

        # Dem bac cua moi dinh
        degree = [len(tree.adj_list[i]) for i in range(n)]

        # Tim cac la (dinh co bac 1)
        leaves = deque()
        for i in range(n):
            if degree[i] == 1:
                leaves.append(i)

        remaining = n

        # Loai bo cac la cho den khi con lai 1 hoac 2 dinh
        while remaining > 2:
            leaf_count = len(leaves)
            remaining -= leaf_count

            for _ in range(leaf_count):
                leaf = leaves.popleft()

                # Giam bac cua cac dinh ke
                for neighbor in tree.adj_list[leaf]:
                    degree[neighbor] -= 1
                    if degree[neighbor] == 1:
                        leaves.append(neighbor)

        # Cac dinh con lai la trung tam
        centers = []
        for i in range(n):
            if degree[i] > 0:
                centers.append(i)

        return centers


def demonstrate_bfs():
    """Ham demo day du cac tinh nang BFS"""
    print("=== DEMO THUAT TOAN BFS ===\n")

    # Demo 1: Do thi lien thong
    print("1. DO THI LIEN THONG:")
    graph1 = BFSUtils.create_sample_graph1()
    graph1.print_graph()

    # BFS tu dinh 0
    graph1.bfs(0)
    graph1.print_distances(0)
    graph1.print_bfs_tree(0)

    # Tim duong di ngan nhat
    print("Duong di ngan nhat tu 0 den 5:")
    path = graph1.get_shortest_path(0, 5)
```

```
    if path:
        path_str = ' -> '.join(map(str, path))
        print(f"{path_str} (Do dai: {len(path) - 1})")
    else:
        print("Khong co duong di")
    print()

    # Hien thi cac muc
    levels = graph1.get_levels(0)
    print("Cac muc trong BFS tree:")
    for level, vertices in sorted(levels.items()):
        print(f"Muc {level}: {vertices}")
    print()

    # Kiem tra lien thong
    print(f"Do thi co lien thong khong? {'Co' if graph1.is_connected()
else 'Khong'}\n")

    # Demo 2: Do thi khong lien thong
    print("2. DO THI KHONG LIEN THONG:")
    graph2 = BFSUtils.create_sample_graph2()
    graph2.print_graph()
    graph2.bfs_complete()

    print(f"Do thi co lien thong khong? {'Co' if graph2.is_connected()
else 'Khong'}\n")

    # Demo 3: Do thi luoi
    print("3. DO THI LUOI 3x3:")
    grid = BFSUtils.create_grid_graph(3, 3)
    print("Cau truc do thi luoi:")
    print("0 - 1 - 2")
    print("|   |   |")
    print("3 - 4 - 5")
    print("|   |   |")
    print("6 - 7 - 8")
    print()

    grid.bfs(0)
    grid.print_distances(0)

    # Demo 4: Tim duong kinh cay
    print("4. TIM DUONG KINH CAY:")
    tree = BFSUtils.create_sample_graph1()
    diameter, endpoints = BFSUtils.find_tree_diameter(tree)
    print(f"Duong kinh cua cay: {diameter}")
    print(f"Hai dau mut: {endpoints[0]} va {endpoints[1]}")

    # Hien thi duong di dai nhat
    path = tree.get_shortest_path(endpoints[0], endpoints[1])
    if path:
        path_str = ' -> '.join(map(str, path))
        print(f"Duong di dai nhat: {path_str}")
    print()

    # Demo 5: Tim trung tam cay
```

```
    print("5. TIM TRUNG TAM CAY:")
    centers = BFSUtils.find_center_vertices(tree)
    print(f"Cac dinh trung tam: {centers}")


def interactive_demo():
    """Demo tuong tac voi nguoi dung"""
    print("=== DEMO TUONG TAC ===")

    try:
        n, m = map(int, input("Nhap so dinh va so canh: ").split())

        if n <= 0 or m < 0:
            print("So dinh va canh phai la so duong!")
            return

        user_graph = BFS(n)

        print(f"Nhap {m} canh (u v):")
        for i in range(m):
            try:
                u, v = map(int, input(f"Canh {i+1}: ").split())
                if 0 <= u < n and 0 <= v < n:
                    user_graph.add_edge(u, v)
                else:
                    print(f"Canh ({u}, {v}) khong hop le! Dinh phai tu 0
    den {n-1}")
                    return
            except ValueError:
                print("Vui long nhap hai so nguyen!")
                return

        start = int(input("Nhap dinh bat dau BFS: "))
        if not (0 <= start < n):
            print(f"Dinh bat dau phai tu 0 den {n-1}!")
            return

        print("\nKet qua:")
        user_graph.print_graph()
        user_graph.bfs(start)
        user_graph.print_distances(start)
        user_graph.print_bfs_tree(start)

        # Tuy chon bo sung
        while True:
            print("\nCac tuy chon bo sung:")
            print("1. Tim duong di ngan nhat")
            print("2. Kiem tra lien thong")
            print("3. Hien thi cac thanh phan lien thong")
            print("4. Tim dinh xa nhat")
            print("0. Thoat")

            choice = input("Chon tuy chon (0-4): ").strip()

            if choice == '0':
                break
```

```
            elif choice == '1':
                target = int(input("Nhap dinh dich: "))
                if 0 <= target < n:
                    path = user_graph.get_shortest_path(start, target)
                    if path:
                        path_str = ' -> '.join(map(str, path))
                        print(f"Duong di ngan nhat tu {start} den {target}: {path_str}")
                        print(f"Do dai: {len(path) - 1}")
                    else:
                        print(f"Khong co duong di tu {start} den {target}")
                else:
                    print("Dinh dich khong hop le!")

            elif choice == '2':
                print(f"Do thi {'lien thong' if user_graph.is_connected() else 'khong lien thong'}")

            elif choice == '3':
                components = user_graph.get_connected_components()
                print(f"So thanh phan lien thong: {len(components)}")
                for i, component in enumerate(components, 1):
                    print(f"Thanh phan {i}: {component}")

            elif choice == '4':
                farthest, dist = user_graph.find_farthest_vertex(start)
                if farthest != -1:
                    print(f"Dinh xa nhat tu {start}: {farthest} (khoang cach: {dist})")
                else:
                    print("Khong tim thay dinh xa nhat")

            else:
                print("Tuy chon khong hop le!")

        except ValueError:
            print("Vui long nhap so nguyen hop le!")
        except KeyboardInterrupt:
            print("\nChuong trinh bi ngat boi nguoi dung.")


if __name__ == "__main__":
    # Chay demo tu dong
    demonstrate_bfs()

    # Chay demo tuong tac
    print("\n" + "="*50)
    interactive_demo()
```

Listing 2: Cài đặt BFS đầy đủ bằng Python

7 Mở rộng cho các loại đồ thị khác

7.1 Bài toán 9: BFS trên đa đồ thị (Multigraph)

7.1.1 Đặc điểm của multigraph

- Cho phép nhiều cạnh giữa hai đỉnh
- Không có khuyên (self-loop)
- Cần xử lý đặc biệt để tránh duyệt trùng lặp

7.1.2 Cài đặt BFS cho Multigraph

```
#include <iostream>
#include <vector>
#include <queue>
#include <set>
#include <map>
#include <climits>

using namespace std;

class MultigraphBFS {
private:
    int numVertices;
    // Su dung multiset de luu nhieu canh giua hai dinh
    vector<multiset<int>> adjList;
    vector<bool> visited;
    vector<int> distance;
    vector<int> parent;

    // dem so canh giua hai dinh
    map<pair<int,int>, int> edgeCount;

public:
    MultigraphBFS(int n) : numVertices(n) {
        adjList.resize(n);
        resetArrays();
    }

    void resetArrays() {
        visited.assign(numVertices, false);
        distance.assign(numVertices, INT_MAX);
        parent.assign(numVertices, -1);
    }

    // Th m   c nh   v o multigraph
    void addEdge(int u, int v) {
        adjList[u].insert(v);
        adjList[v].insert(u);

        //    m     s     c nh
        pair<int,int> edge = {min(u,v), max(u,v)};
```

```
41          edgeCount[edge]++;
42
43          cout << "Them canh (" << u << "," << v << ") - So canh giua "
44               << u << " va " << v << ": " << edgeCount[edge] << endl;
45      }
46
47      // BFS cho multigraph
48      void bfs(int start) {
49          resetArrays();
50
51          queue<int> q;
52          visited[start] = true;
53          distance[start] = 0;
54          q.push(start);
55
56          cout << "BFS tren Multigraph tu dinh " << start << ":" << endl;
57          cout << "Thu tu duyet: ";
58
59          while (!q.empty()) {
60              int u = q.front();
61              q.pop();
62              cout << u << " ";
63
64              //  S    d ng  set       tr nh  d u y t  tr ng    nh     k
65              set<int> uniqueNeighbors(adjList[u].begin(), adjList[u].end
    ());
66
67              for (int v : uniqueNeighbors) {
68                  if (!visited[v]) {
69                      visited[v] = true;
70                      distance[v] = distance[u] + 1;
71                      parent[v] = u;
72                      q.push(v);
73                  }
74              }
75          }
76          cout << endl << endl;
77      }
78
79      // In  th ng tin  v    c  c   c nh    b i
80      void printMultipleEdges() {
81          cout << "Cac canh boi trong multigraph:" << endl;
82          for (auto& edge : edgeCount) {
83              if (edge.second > 1) {
84                  cout << "Canh (" << edge.first.first << ","
85                       << edge.first.second << "): " << edge.second << "
    canh" << endl;
86              }
87          }
88          cout << endl;
89      }
90
91      // T m  t t   c       ng      i   t   u   n   v (x t   c nh   b i )
92      void findAllPaths(int start, int end, vector<int>& path, vector<
    vector<int>>& allPaths) {
93          path.push_back(start);
```

```
94
95          if (start == end) {
96              allPaths.push_back(path);
97          } else {
98              for (int neighbor : adjList[start]) {
99                  //  K i m  tra kh ng  t o  chu tr nh (  n    g i n )
100                 if (find(path.begin(), path.end(), neighbor) == path.end
    ()) {
101                     findAllPaths(neighbor, end, path, allPaths);
102                 }
103             }
104         }
105
106         path.pop_back();
107     }
108
109     void printGraph() {
110         cout << "Bieu dien Multigraph:" << endl;
111         for (int i = 0; i < numVertices; i++) {
112             cout << i << ": ";
113             for (int neighbor : adjList[i]) {
114                 cout << neighbor << " ";
115             }
116             cout << endl;
117         }
118         cout << endl;
119     }
120 };
```

Listing 3: BFS cho Multigraph - C++

```
1  from collections import deque, defaultdict, Counter
2  from typing import List, Dict, Set, Tuple
3
4  class MultigraphBFS:
5      """BFS cho multigraph ( a         t h  )"""
6
7      def __init__(self, num_vertices: int):
8          self.num_vertices = num_vertices
9          #  S    d ng  list       l u  n h i u   c nh
10         self.adj_list = [[] for _ in range(num_vertices)]
11         self.edge_count = defaultdict(int)
12         self.reset_arrays()
13
14     def reset_arrays(self):
15         self.visited = [False] * self.num_vertices
16         self.distance = [float('inf')] * self.num_vertices
17         self.parent = [-1] * self.num_vertices
18
19     def add_edge(self, u: int, v: int):
20         """Th m  c nh  v  o multigraph"""
21         self.adj_list[u].append(v)
22         self.adj_list[v].append(u)
23
24         #    m    s    c nh   g i a  u  v   v
25         edge = tuple(sorted([u, v]))
```

```
        self.edge_count[edge] += 1

        print(f"Th m  c nh  ({u},{v}) -  S    c nh   g i a  {u} v   {v}:
    {self.edge_count[edge]}")

    def bfs(self, start: int):
        """BFS  tr n  multigraph"""
        self.reset_arrays()

        queue = deque([start])
        self.visited[start] = True
        self.distance[start] = 0

        print(f"BFS  tr n  Multigraph  t      nh    {start}:")
        print(" T h    t    d u y t : ", end="")

        while queue:
            u = queue.popleft()
            print(u, end=" ")

            #  L y  danh  s ch    nh     k   duy  n h t         tr nh
    d u y t  tr ng
            unique_neighbors = set(self.adj_list[u])

            for v in sorted(unique_neighbors):
                if not self.visited[v]:
                    self.visited[v] = True
                    self.distance[v] = self.distance[u] + 1
                    self.parent[v] = u
                    queue.append(v)

        print("\n")

    def print_multiple_edges(self):
        """In  th ng tin  v   c  c   c nh   b i """
        print("C  c   c nh   b i  trong multigraph:")
        for edge, count in self.edge_count.items():
            if count > 1:
                print(f" C nh  {edge}: {count}  c nh ")
        print()

    def get_edge_multiplicity(self, u: int, v: int) -> int:
        """ L y   s    l   ng   c nh   g i a  u  v   v"""
        edge = tuple(sorted([u, v]))
        return self.edge_count.get(edge, 0)

    def print_graph(self):
        """In  b i u   d i n  multigraph"""
        print(" B i u   d i n  Multigraph:")
        for i in range(self.num_vertices):
            neighbors = ' '.join(map(str, self.adj_list[i]))
            print(f"{i}: {neighbors}")
        print()
```

Listing 4: BFS cho Multigraph - Python

7.2 Bài toán 10: BFS trên đồ thị tổng quát (General Graph)

7.2.1 Đặc điểm của general graph

- Cho phép cạnh lặp (multiple edges)
- Cho phép khuyên (self-loops)
- Cần xử lý đặc biệt cho khuyên trong BFS

7.2.2 Cài đặt BFS cho General Graph

```
#include <iostream>
#include <vector>
#include <queue>
#include <set>
#include <map>
#include <climits>

using namespace std;

class GeneralGraphBFS {
private:
    int numVertices;
    vector<multiset<int>> adjList;
    vector<bool> visited;
    vector<int> distance;
    vector<int> parent;

    map<pair<int,int>, int> edgeCount;
    set<int> selfLoops; // L u  c  c    nh    c   khuy n

public:
    GeneralGraphBFS(int n) : numVertices(n) {
        adjList.resize(n);
        resetArrays();
    }

    void resetArrays() {
        visited.assign(numVertices, false);
        distance.assign(numVertices, INT_MAX);
        parent.assign(numVertices, -1);
    }

    // Th m  c nh  v o general graph
    void addEdge(int u, int v) {
        adjList[u].insert(v);
        if (u != v) { // Kh ng  p h i  khuy n
            adjList[v].insert(u);
        } else { // L  khuy n
            selfLoops.insert(u);
            cout << "Them khuyen tai dinh " << u << endl;
        }

        //    m    s    c nh
```

```
44          pair<int,int> edge = {min(u,v), max(u,v)};
45          edgeCount[edge]++;
46
47          if (u != v) {
48              cout << "Them canh (" << u << "," << v << ") - So canh: "
49                   << edgeCount[edge] << endl;
50          }
51      }
52
53      // BFS cho general graph
54      void bfs(int start) {
55          resetArrays();
56
57          queue<int> q;
58          visited[start] = true;
59          distance[start] = 0;
60          q.push(start);
61
62          cout << "BFS tren General Graph tu dinh " << start << ":" <<
    endl;
63
64          //  K i m  tra khuy n  t i    nh    b t    u
65          if (selfLoops.count(start)) {
66              cout << "Dinh bat dau " << start << " co khuyen!" << endl;
67          }
68
69          cout << "Thu tu duyet: ";
70
71          while (!q.empty()) {
72              int u = q.front();
73              q.pop();
74              cout << u << " ";
75
76              //  T o  set  c  c   nh    k   duy  n h t  ( l o i  b
    tr ng  l p )
77              set<int> uniqueNeighbors;
78              for (int v : adjList[u]) {
79                  if (v != u) { //  B  qua khuy n trong BFS
80                      uniqueNeighbors.insert(v);
81                  }
82              }
83
84              for (int v : uniqueNeighbors) {
85                  if (!visited[v]) {
86                      visited[v] = true;
87                      distance[v] = distance[u] + 1;
88                      parent[v] = u;
89                      q.push(v);
90                  }
91              }
92          }
93          cout << endl << endl;
94      }
95
96      // In th ng tin  v        t h
97      void printGraphInfo() {
```

```
98          cout << "Thong tin General Graph:" << endl;
99          cout << "So dinh: " << numVertices << endl;
100         cout << "So dinh co khuyen: " << selfLoops.size() << endl;
101
102         if (!selfLoops.empty()) {
103             cout << "Cac dinh co khuyen: ";
104             for (int v : selfLoops) {
105                 cout << v << " ";
106             }
107             cout << endl;
108         }
109
110         cout << "Cac canh boi:" << endl;
111         for (auto& edge : edgeCount) {
112             if (edge.second > 1) {
113                 cout << "Canh (" << edge.first.first << ","
114                      << edge.first.second << "): " << edge.second << "
    canh" << endl;
115             }
116         }
117         cout << endl;
118     }
119
120     // K i m  tra t nh  c h t    c    b i t
121     void checkSpecialProperties() {
122         cout << "Kiem tra tinh chat dac biet:" << endl;
123
124         // K i m  tra c   p h i  simple graph kh ng
125         bool isSimple = selfLoops.empty();
126         for (auto& edge : edgeCount) {
127             if (edge.second > 1) {
128                 isSimple = false;
129                 break;
130             }
131         }
132
133         cout << "La simple graph: " << (isSimple ? "Co" : "Khong") <<
    endl;
134         cout << "La multigraph: " << (selfLoops.empty() ? "Co" : "Khong"
    ) << endl;
135         cout << "La general graph: Co" << endl << endl;
136     }
137
138     void printGraph() {
139         cout << "Bieu dien General Graph:" << endl;
140         for (int i = 0; i < numVertices; i++) {
141             cout << i << ": ";
142             for (int neighbor : adjList[i]) {
143                 cout << neighbor << " ";
144             }
145             cout << endl;
146         }
147         cout << endl;
148     }
```

```
};
```

Listing 5: BFS cho General Graph - C++

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

class GeneralGraphBFS:
    """BFS cho general graph (       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_count = defaultdict(int)
        self.self_loops = set()  # L u  c  c    nh    c   khuy n
        self.reset_arrays()

    def reset_arrays(self):
        self.visited = [False] * self.num_vertices
        self.distance = [float('inf')] * self.num_vertices
        self.parent = [-1] * self.num_vertices

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

        if u != v:  # Kh ng  p h i  khuy n
            self.adj_list[v].append(u)
        else:  # L   khuy n
            self.self_loops.add(u)
            print(f"Th  m  khuy n  t i     nh    {u}")

        #    m     s     c nh
        edge = tuple(sorted([u, v]))
        self.edge_count[edge] += 1

        if u != v:
            print(f"Th  m   c nh  ({u},{v}) -  S    c nh : {self.
    edge_count[edge]}")

    def bfs(self, start: int):
        """BFS tr n general graph"""
        self.reset_arrays()

        queue = deque([start])
        self.visited[start] = True
        self.distance[start] = 0

        print(f"BFS  tr n  General  Graph  t      nh    {start}:")

        #  K i m  tra  khuy n  t i     nh     b t     u
        if start in self.self_loops:
            print(f"   nh     b t     u    {start}  c   khuy n!")

        print(" T h    t    d u y t : ", end="")

```

```
52          while queue:
53              u = queue.popleft()
54              print(u, end=" ")
55
56              #  L c  ra c c    nh     k   duy  n h t  ( l o i   b   khuy n
    v   tr ng  l p )
57              unique_neighbors = set()
58              for v in self.adj_list[u]:
59                  if v != u:  #  B   qua khuy n trong BFS
60                      unique_neighbors.add(v)
61
62              for v in sorted(unique_neighbors):
63                  if not self.visited[v]:
64                      self.visited[v] = True
65                      self.distance[v] = self.distance[u] + 1
66                      self.parent[v] = u
67                      queue.append(v)
68
69          print("\n")
70
71      def print_graph_info(self):
72          """In  th ng tin  v          t h  """
73          print("Th ng tin General Graph:")
74          print(f" S      nh   : {self.num_vertices}")
75          print(f" S      nh    c   khuy n: {len(self.self_loops)}")
76
77          if self.self_loops:
78              print(f"C  c    nh    c   khuy n: {sorted(self.self_loops)}"
    )
79
80          print("C  c   c nh   b i :")
81          for edge, count in self.edge_count.items():
82              if count > 1:
83                  print(f" C nh  {edge}: {count}  c nh ")
84          print()
85
86      def check_special_properties(self):
87          """ K i m  tra t nh  c h t    c    b i t """
88          print(" K i m  tra t nh  c h t    c    b i t :")
89
90          #  K i m  tra c   p h i  simple graph kh ng
91          is_simple = len(self.self_loops) == 0
92          for count in self.edge_count.values():
93              if count > 1:
94                  is_simple = False
95                  break
96
97          print(f"L   simple graph: {'C  ' if is_simple else 'Kh ng'}")
98          print(f"L   multigraph: {'C  ' if len(self.self_loops) == 0 else
    'Kh ng'}")
99          print("L   general graph:  C  ")
100         print()
101
102     def get_self_loops(self) -> Set[int]:
103         """ L y  danh  s ch  c  c    nh    c   khuy n"""
104         return self.self_loops.copy()
```

```python

    def has_multiple_edges(self, u: int, v: int) -> bool:
        """ K i m  tra  c   n h i u  c nh  g i a  u  v  v kh ng"""
        edge = tuple(sorted([u, v]))
        return self.edge_count.get(edge, 0) > 1

    def print_graph(self):
        """In  b i u  d i n  general graph"""
        print(" B i u  d i n  General Graph:")
        for i in range(self.num_vertices):
            neighbors = ' '.join(map(str, self.adj_list[i]))
            print(f"{i}: {neighbors}")
        print()
```

Listing 6: BFS cho General Graph - Python

7.3 Demo tích hợp cho cả 3 loại đồ thị

```cpp
// Th m  v o  c u i  file main
void demonstrateAllGraphTypes() {
    cout << "=== DEMO TICH HOP CAC LOAI DO THI ===" << endl << endl;

    // Demo Simple Graph (B i 8)
    cout << "1. SIMPLE GRAPH (Bai 8):" << endl;
    auto simpleGraph = BFSUtils::createSampleGraph1();
    simpleGraph.printGraph();
    simpleGraph.bfs(0);
    cout << endl;

    // Demo Multigraph (B i 9)
    cout << "2. MULTIGRAPH (Bai 9):" << endl;
    MultigraphBFS multiGraph(5);

    // Th m  c nh  b nh  t h  ng
    multiGraph.addEdge(0, 1);
    multiGraph.addEdge(1, 2);

    // Th m  c nh  b i
    multiGraph.addEdge(0, 1); //  C nh  t h  2  g i a  0  v  1
    multiGraph.addEdge(0, 1); //  C nh  t h  3  g i a  0  v  1
    multiGraph.addEdge(2, 3);
    multiGraph.addEdge(3, 4);

    multiGraph.printGraph();
    multiGraph.printMultipleEdges();
    multiGraph.bfs(0);
    cout << endl;

    // Demo General Graph (B i 10)
    cout << "3. GENERAL GRAPH (Bai 10):" << endl;
    GeneralGraphBFS generalGraph(5);

    // Th m  c nh  b nh  t h  ng
    generalGraph.addEdge(0, 1);
    generalGraph.addEdge(1, 2);
```

```
38
39      // Th m   c nh    b i
40      generalGraph.addEdge(0, 1);
41      generalGraph.addEdge(0, 1);
42
43      // Th m  khuy n
44      generalGraph.addEdge(2, 2); // Khuy n   t i     nh    2
45      generalGraph.addEdge(3, 3); // Khuy n   t i     nh    3
46
47      // Th m   c nh   b nh   t h    ng
48      generalGraph.addEdge(2, 3);
49      generalGraph.addEdge(3, 4);
50
51      generalGraph.printGraph();
52      generalGraph.printGraphInfo();
53      generalGraph.checkSpecialProperties();
54      generalGraph.bfs(0);
55  }
56
57  // Th m  v  o  h  m  main
58  int main() {
59      demonstrateBFS();           // Demo   g c
60      demonstrateAllGraphTypes(); // Demo   m i
61
62      // Interactive demo
63      // ... code  c
64
65      return 0;
66  }
```

Listing 7: Demo tích hợp C++

```
1   def demonstrate_all_graph_types():
2       """Demo  t  ch   h p   cho   c    3   l o i          t h   """
3       print("=== DEMO  T  CH   H P   C  C   L O I          T H    ===\n")
4
5       # Demo Simple Graph (B  i  8)
6       print("1. SIMPLE GRAPH (B  i  8):")
7       simple_graph = BFSUtils.create_sample_graph1()
8       simple_graph.print_graph()
9       simple_graph.bfs(0)
10      print()
11
12      # Demo Multigraph (B  i  9)
13      print("2. MULTIGRAPH (B  i  9):")
14      multi_graph = MultigraphBFS(5)
15
16      # Th m   c nh   b nh   t h    ng
17      multi_graph.add_edge(0, 1)
18      multi_graph.add_edge(1, 2)
19
20      # Th m   c nh    b i
21      multi_graph.add_edge(0, 1)  #  C nh    t h   2
22      multi_graph.add_edge(0, 1)  #  C nh    t h   3
23      multi_graph.add_edge(2, 3)
24      multi_graph.add_edge(3, 4)
```

```
25
26      multi_graph.print_graph()
27      multi_graph.print_multiple_edges()
28      multi_graph.bfs(0)
29      print()
30
31      # Demo General Graph (B i 10)
32      print("3. GENERAL GRAPH (B i 10):")
33      general_graph = GeneralGraphBFS(5)
34
35      # Th m c nh b nh t h ng
36      general_graph.add_edge(0, 1)
37      general_graph.add_edge(1, 2)
38
39      # Th m c nh b i
40      general_graph.add_edge(0, 1)
41      general_graph.add_edge(0, 1)
42
43      # Th m khuy n
44      general_graph.add_edge(2, 2)  # Khuy n t i nh 2
45      general_graph.add_edge(3, 3)  # Khuy n t i nh 3
46
47      # Th m c nh b nh t h ng
48      general_graph.add_edge(2, 3)
49      general_graph.add_edge(3, 4)
50
51      general_graph.print_graph()
52      general_graph.print_graph_info()
53      general_graph.check_special_properties()
54      general_graph.bfs(0)
55
56  if __name__ == "__main__":
57      demonstrate_bfs()              # Demo g c
58      demonstrate_all_graph_types()  # Demo m i
59
60      print("\n" + "="*50)
61      interactive_demo()
```

Listing 8: Demo tích hợp Python

8 So sánh các loại đồ thị

Đặc điểm	**Simple Graph**	**Multigraph**	**General Graph**
Cạnh bội	Không	Có	Có
Khuyên	Không	Không	Có
Cài đặt BFS	Đơn giản	Cần lọc trùng	Cần xử lý khuyên
Độ phức tạp	$O(V+E)$	$O(V+E)$	$O(V+E)$
Ứng dụng	Cơ bản	Mạng giao thông	Mô hình tổng quát

9 Ứng dụng của BFS

9.1 Các bài toán ứng dụng trực tiếp

1. **Tìm đường đi ngắn nhất:** BFS đảm bảo tìm được đường đi có ít cạnh nhất trong đồ thị không trọng số
2. **Kiểm tra tính liên thông:** Sử dụng BFS để kiểm tra xem đồ thị có liên thông hay không
3. **Tìm các thành phần liên thông:** Phân chia đồ thị thành các thành phần liên thông riêng biệt
4. **Tìm chu trình ngắn nhất:** Trong đồ thị không trọng số, BFS có thể tìm chu trình có độ dài nhỏ nhất
5. **Bipartite Graph Detection:** Kiểm tra xem đồ thị có phải là đồ thị hai phần hay không

10 So sánh BFS và DFS

Tiêu chí	**BFS**	**DFS**
Cấu trúc dữ liệu	Queue (FIFO)	Stack (LIFO)
Độ phức tạp thời gian	$O(V + E)$	$O(V + E)$
Độ phức tạp không gian	$O(V)$	$O(V)$
Đường đi ngắn nhất	Có	Không
Phát hiện chu trình	Có	Có
Thứ tự duyệt	Theo lớp	Theo chiều sâu
Ứng dụng chính	Tìm đường ngắn	Topological sort

11 Kết luận

Thuật toán BFS là một trong những thuật toán cơ bản và quan trọng nhất trong lý thuyết đồ thị. Với độ phức tạp thời gian $O(V + E)$ và khả năng tìm đường đi ngắn nhất trong đồ thị không trọng số, BFS được ứng dụng rộng rãi trong nhiều bài toán thực tế.

Ưu điểm chính:

- Đảm bảo tìm được đường đi ngắn nhất
- Dễ hiểu và cài đặt
- Độ phức tạp tối ưu
- Ứng dụng đa dạng

Nhược điểm:

- Tốn nhiều bộ nhớ cho đồ thị rộng

- Không hiệu quả cho đồ thị có trọng số
- Không phù hợp cho đồ thị vô hạn

Việc nắm vững BFS là nền tảng để học các thuật toán đồ thị nâng cao hơn như Dijkstra, A*, và các thuật toán tìm kiếm heuristic khác.