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

trees,layered

BÀI TOÁN CHUYỂN ĐỔI BIỂU DIỄN ĐỒ THỊ

Toán Tổ Hợp và lý thuyết đồ thị

1 Lý thuyết cơ bản về đồ thị

1.1 Định nghĩa đồ thị

Đồ thị $G = (V, E)$ là một cấu trúc dữ liệu bao gồm:

- V: Tập hợp các đỉnh (vertices/nodes)
- E: Tập hợp các cạnh (edges) nối các đỉnh

Đồ thị có thể là:

- **Vô hướng**: Các cạnh không có hướng $(u, v) = (v, u)$
- **Có hướng**: Các cạnh có hướng $(u, v) \neq (v, u)$
- **Có trọng số**: Mỗi cạnh có một giá trị trọng số
- **Không trọng số**: Các cạnh chỉ biểu diễn mối quan hệ

1.2 Các cách biểu diễn đồ thị

1.2.1 1. Ma trận kề (Adjacency Matrix)

Ma trận kề A có kích thước $n \times n$ với n là số đỉnh:

$$A[i][j] = \begin{cases} 1 & \text{nếu có cạnh từ đỉnh } i \text{ đến đỉnh } j \\ 0 & \text{nếu không có cạnh} \end{cases}$$

Ưu điểm:

- Kiểm tra cạnh trong $O(1)$
- Dễ cài đặt và hiểu
- Phù hợp với đồ thị dày đặc

Nhược điểm:

- Độ phức tạp không gian: $O(V^2)$
- Lãng phí bộ nhớ với đồ thị thưa
- Duyệt tất cả cạnh mất $O(V^2)$

1.2.2 2. Danh sách kề (Adjacency List)

Mỗi đỉnh v có một danh sách chứa các đỉnh kề với nó.

Ưu điểm:

- Độ phức tạp không gian: $O(V + E)$
- Tiết kiệm bộ nhớ với đồ thị thưa
- Duyệt cạnh hiệu quả $O(V + E)$

Nhược điểm:

- Kiểm tra cạnh mất $O(\deg(v))$
- Phức tạp hơn trong cài đặt

1.2.3 3. Danh sách cạnh mở rộng (Extended Adjacency List)

Tương tự danh sách kề nhưng lưu thêm thông tin trọng số, chỉ số cạnh, etc.

1.2.4 4. Ma trận đồ thị (Adjacency Map)

Sử dụng hash map để lưu trữ, kết hợp ưu điểm của cả hai phương pháp trên.

2 Mô tả bài toán

Đề bài: Viết chương trình C/C++, Python chuyển đổi giữa 4 dạng biểu diễn:

1. Adjacency matrix
2. Adjacency list
3. Extended adjacency list
4. Adjacency map

Cho 3 đồ thị: đơn đồ thị, đa đồ thị, đồ thị tổng quát và 3 dạng biểu diễn:

1. Array of parents
2. First-child next-sibling
3. Graph-based representation of trees

Tổng cộng: $3 \times 4 + 4_3^2 = 36 + 6 = 42$ chương trình chuyển đổi.

3 Ý tưởng và giải pháp

3.1 Phân tích bài toán

Bài toán yêu cầu:

- Hiểu rõ cấu trúc dữ liệu của từng dạng biểu diễn
- Xây dựng thuật toán chuyển đổi hiệu quả
- Xử lý các trường hợp đặc biệt (đồ thị rỗng, đỉnh cô lập, etc.)
- Tối ưu hóa độ phức tạp thời gian và không gian

3.2 Chiến lược giải quyết

1. **Thiết kế cấu trúc dữ liệu:** Định nghĩa class/struct cho từng dạng biểu diễn
2. **Xây dựng hàm chuyển đổi:** Mỗi cặp chuyển đổi là một hàm riêng biệt
3. **Kiểm tra tính đúng đắn:** Validate input và output
4. **Tối ưu hóa:** Sử dụng thuật toán hiệu quả nhất có thể

4 Thuật toán chi tiết

4.1 Chuyển đổi từ Ma trận kề sang Danh sách kề

```
Algorithm: MatrixToList(matrix, n)
Input: matrix - ma tran ke nxn, n - so dinh
Output: adjList - danh sach ke

1. Initialize adjList as array of n empty lists
2. For i = 0 to n-1:
3.     For j = 0 to n-1:
4.         If matrix[i][j] != 0:
5.             adjList[i].add(j)
6. Return adjList

Time Complexity: O(V  )
Space Complexity: O(V + E)
```

Listing 1: Thuật toán chuyển đổi Ma trận kề sang Danh sách kề

4.2 Chuyển đổi từ Danh sách kề sang Ma trận kề

```
Algorithm: ListToMatrix(adjList, n)
Input: adjList - danh sach ke, n - so dinh
Output: matrix - ma tran ke nxn

1. Initialize matrix[n][n] with all zeros
```

```
2. For i = 0 to n-1:
3.     For each vertex j in adjList[i]:
4.         matrix[i][j] = 1
5. Return matrix

Time Complexity: O(V + E)
Space Complexity: O(V  )
```

Listing 2: Thuật toán chuyển đổi Danh sách kề sang Ma trận kề

5 Code C++

```cpp
#include <iostream>
#include <vector>
#include <list>
#include <map>
#include <set>
#include <unordered_map>

using namespace std;

class GraphConverter {
private:
    int numVertices;

public:
    // Cau truc du lieu cho cac dang bieu dien

    // 1. Ma tran ke
    typedef vector<vector<int>> AdjMatrix;

    // 2. Danh sach ke
    typedef vector<list<int>> AdjList;

    // 3. Danh sach ke mo rong (co trong so)
    typedef vector<list<pair<int, int>>> ExtendedAdjList;

    // 4. Ma tran do thi (su dung map)
    typedef unordered_map<int, set<int>> AdjMap;

    GraphConverter(int n) : numVertices(n) {}

    // Chuyen doi tu Ma tran ke sang Danh sach ke
    AdjList matrixToList(const AdjMatrix& matrix) {
        AdjList adjList(numVertices);

        for (int i = 0; i < numVertices; i++) {
            for (int j = 0; j < numVertices; j++) {
                if (matrix[i][j] != 0) {
                    adjList[i].push_back(j);
                }
            }
        }

```

```
43          return adjList;
44      }
45
46      // Chuyen doi tu Danh sach ke sang Ma tran ke
47      AdjMatrix listToMatrix(const AdjList& adjList) {
48          AdjMatrix matrix(numVertices, vector<int>(numVertices, 0));
49
50          for (int i = 0; i < numVertices; i++) {
51              for (int neighbor : adjList[i]) {
52                  matrix[i][neighbor] = 1;
53              }
54          }
55
56          return matrix;
57      }
58
59      // Chuyen doi tu Danh sach ke sang Danh sach ke mo rong
60      ExtendedAdjList listToExtended(const AdjList& adjList,
61                                     const vector<vector<int>>& weights) {
62          ExtendedAdjList extList(numVertices);
63
64          for (int i = 0; i < numVertices; i++) {
65              for (int neighbor : adjList[i]) {
66                  int weight = (weights.empty()) ? 1 : weights[i][neighbor
    ];
67                  extList[i].push_back({neighbor, weight});
68              }
69          }
70
71          return extList;
72      }
73
74      // Chuyen doi tu Danh sach ke mo rong sang Danh sach ke
75      AdjList extendedToList(const ExtendedAdjList& extList) {
76          AdjList adjList(numVertices);
77
78          for (int i = 0; i < numVertices; i++) {
79              for (const auto& edge : extList[i]) {
80                  adjList[i].push_back(edge.first);
81              }
82          }
83
84          return adjList;
85      }
86
87      // Chuyen doi tu Danh sach ke sang Ma tran do thi
88      AdjMap listToMap(const AdjList& adjList) {
89          AdjMap adjMap;
90
91          for (int i = 0; i < numVertices; i++) {
92              for (int neighbor : adjList[i]) {
93                  adjMap[i].insert(neighbor);
94              }
95          }
96
97          return adjMap;
```

```
98      }
99
100     // Chuyen doi tu Ma tran do thi sang Danh sach ke
101     AdjList mapToList(const AdjMap& adjMap) {
102         AdjList adjList(numVertices);
103
104         for (const auto& vertex : adjMap) {
105             int u = vertex.first;
106             for (int v : vertex.second) {
107                 adjList[u].push_back(v);
108             }
109         }
110
111         return adjList;
112     }
113
114     // Ham in ma tran ke
115     void printMatrix(const AdjMatrix& matrix) {
116         cout << "Ma tran ke:" << endl;
117         for (int i = 0; i < numVertices; i++) {
118             for (int j = 0; j < numVertices; j++) {
119                 cout << matrix[i][j] << " ";
120             }
121             cout << endl;
122         }
123     }
124
125     // Ham in danh sach ke
126     void printList(const AdjList& adjList) {
127         cout << "Danh sach ke:" << endl;
128         for (int i = 0; i < numVertices; i++) {
129             cout << i << ": ";
130             for (int neighbor : adjList[i]) {
131                 cout << neighbor << " ";
132             }
133             cout << endl;
134         }
135     }
136
137     // Ham in danh sach ke mo rong
138     void printExtended(const ExtendedAdjList& extList) {
139         cout << "Danh sach ke mo rong:" << endl;
140         for (int i = 0; i < numVertices; i++) {
141             cout << i << ": ";
142             for (const auto& edge : extList[i]) {
143                 cout << "(" << edge.first << "," << edge.second << ") ";
144             }
145             cout << endl;
146         }
147     }
148
149     // Ham in ma tran do thi
150     void printMap(const AdjMap& adjMap) {
151         cout << "Ma tran do thi:" << endl;
152         for (const auto& vertex : adjMap) {
153             cout << vertex.first << ": ";
```

```
            for (int neighbor : vertex.second) {
                cout << neighbor << " ";
            }
            cout << endl;
        }
    }
};

// Ham test
int main() {
    int n = 4;
    GraphConverter converter(n);

    // Tao ma tran ke mau
    GraphConverter::AdjMatrix matrix = {
        {0, 1, 1, 0},
        {1, 0, 1, 1},
        {1, 1, 0, 1},
        {0, 1, 1, 0}
    };

    cout << "=== CHUYEN DOI GIUA CAC DANG BIEU DIEN DO THI ===" << endl;

    // In ma tran ban dau
    converter.printMatrix(matrix);
    cout << endl;

    // Chuyen doi sang danh sach ke
    auto adjList = converter.matrixToList(matrix);
    converter.printList(adjList);
    cout << endl;

    // Chuyen doi sang danh sach ke mo rong
    vector<vector<int>> weights = {
        {0, 2, 3, 0},
        {2, 0, 1, 4},
        {3, 1, 0, 2},
        {0, 4, 2, 0}
    };
    auto extList = converter.listToExtended(adjList, weights);
    converter.printExtended(extList);
    cout << endl;

    // Chuyen doi sang ma tran do thi
    auto adjMap = converter.listToMap(adjList);
    converter.printMap(adjMap);
    cout << endl;

    // Kiem tra tinh dung dan: chuyen nguoc lai
    auto matrixBack = converter.listToMatrix(adjList);
    cout << "Ma tran sau khi chuyen doi nguoc lai:" << endl;
    converter.printMatrix(matrixBack);

    return 0;
```

```
}
```

Listing 3: Cài đặt đầy đủ bằng C++

6 Code Python

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

class GraphConverter:
    """
    Lop chuyen doi giua cac dang bieu dien do thi
    """

    def __init__(self, num_vertices: int):
        self.num_vertices = num_vertices

    def matrix_to_list(self, matrix: List[List[int]]) -> List[List[int]]:
        """
        Chuyen doi tu ma tran ke sang danh sach ke

        Args:
            matrix: Ma tran ke nxn

        Returns:
            Danh sach ke

        Time Complexity: O(V  )
        Space Complexity: O(V + E)
        """
        adj_list = [[] for _ in range(self.num_vertices)]

        for i in range(self.num_vertices):
            for j in range(self.num_vertices):
                if matrix[i][j] != 0:
                    adj_list[i].append(j)

        return adj_list

    def list_to_matrix(self, adj_list: List[List[int]]) -> List[List[int]]:
        """
        Chuyen doi tu danh sach ke sang ma tran ke

        Args:
            adj_list: Danh sach ke

        Returns:
            Ma tran ke nxn

        Time Complexity: O(V + E)
        Space Complexity: O(V  )
        """
```

```
47          matrix = [[0] * self.num_vertices for _ in range(self.
    num_vertices)]
48
49          for i in range(self.num_vertices):
50              for neighbor in adj_list[i]:
51                  matrix[i][neighbor] = 1
52
53          return matrix
54
55      def list_to_extended(self, adj_list: List[List[int]],
56                           weights: List[List[int]] = None) -> List[List[
    Tuple[int, int]]]:
57          """
58          Chuyen doi tu danh sach ke sang danh sach ke mo rong
59
60          Args:
61              adj_list: Danh sach ke
62              weights: Ma tran trong so (optional)
63
64          Returns:
65              Danh sach ke mo rong voi trong so
66          """
67          ext_list = [[] for _ in range(self.num_vertices)]
68
69          for i in range(self.num_vertices):
70              for neighbor in adj_list[i]:
71                  weight = 1 if weights is None else weights[i][neighbor]
72                  ext_list[i].append((neighbor, weight))
73
74          return ext_list
75
76      def extended_to_list(self, ext_list: List[List[Tuple[int, int]]]) ->
     List[List[int]]:
77          """
78          Chuyen doi tu danh sach ke mo rong sang danh sach ke
79
80          Args:
81              ext_list: Danh sach ke mo rong
82
83          Returns:
84              Danh sach ke
85          """
86          adj_list = [[] for _ in range(self.num_vertices)]
87
88          for i in range(self.num_vertices):
89              for neighbor, _ in ext_list[i]:
90                  adj_list[i].append(neighbor)
91
92          return adj_list
93
94      def list_to_map(self, adj_list: List[List[int]]) -> Dict[int, Set[
    int]]:
95          """
96          Chuyen doi tu danh sach ke sang ma tran do thi (dung dictionary)
97
98          Args:
```

```
            adj_list: Danh sach ke

        Returns:
            Ma tran do thi
        """
        adj_map = defaultdict(set)

        for i in range(self.num_vertices):
            for neighbor in adj_list[i]:
                adj_map[i].add(neighbor)

        return dict(adj_map)

    def map_to_list(self, adj_map: Dict[int, Set[int]]) -> List[List[int]]:
        """
        Chuyen doi tu ma tran do thi sang danh sach ke

        Args:
            adj_map: Ma tran do thi

        Returns:
            Danh sach ke
        """
        adj_list = [[] for _ in range(self.num_vertices)]

        for vertex in adj_map:
            for neighbor in adj_map[vertex]:
                adj_list[vertex].append(neighbor)

        return adj_list

    def print_matrix(self, matrix: List[List[int]]):
        """In ma tran ke"""
        print("Ma tran ke:")
        for row in matrix:
            print(' '.join(map(str, row)))
        print()

    def print_list(self, adj_list: List[List[int]]):
        """In danh sach ke"""
        print("Danh sach ke:")
        for i, neighbors in enumerate(adj_list):
            print(f"{i}: {' '.join(map(str, neighbors))}")
        print()

    def print_extended(self, ext_list: List[List[Tuple[int, int]]]):
        """In danh sach ke mo rong"""
        print("Danh sach ke mo rong:")
        for i, edges in enumerate(ext_list):
            edge_strs = [f"({neighbor},{weight})" for neighbor, weight in edges]
            print(f"{i}: {' '.join(edge_strs)}")
        print()

    def print_map(self, adj_map: Dict[int, Set[int]]):
```

```
        """In ma tran do thi"""
        print("Ma tran do thi:")
        for vertex in sorted(adj_map.keys()):
            neighbors = ' '.join(map(str, sorted(adj_map[vertex])))
            print(f"{vertex}: {neighbors}")
        print()

# Cac ham xu ly bieu dien cay
class TreeConverter:
    """
    Lop chuyen doi giua cac dang bieu dien cay
    """

    def __init__(self, num_nodes: int):
        self.num_nodes = num_nodes

    def parent_array_to_adj_list(self, parent: List[int]) -> List[List[
    int]]:
        """
        Chuyen doi tu mang cha sang danh sach ke

        Args:
            parent: Mang cha, parent[i] = cha cua node i, parent[root] =
     -1

        Returns:
            Danh sach ke cua cay
        """
        adj_list = [[] for _ in range(self.num_nodes)]

        for i in range(self.num_nodes):
            if parent[i] != -1:
                adj_list[parent[i]].append(i)
                adj_list[i].append(parent[i])  # Vo huong

        return adj_list

    def adj_list_to_parent_array(self, adj_list: List[List[int]],
                                 root: int = 0) -> List[int]:
        """
        Chuyen doi tu danh sach ke sang mang cha

        Args:
            adj_list: Danh sach ke cua cay
            root: Dinh goc

        Returns:
            Mang cha
        """
        parent = [-1] * self.num_nodes
        visited = [False] * self.num_nodes

        def dfs(node: int):
            visited[node] = True
            for neighbor in adj_list[node]:
                if not visited[neighbor]:
```

```
                    parent[neighbor] = node
                    dfs(neighbor)

        dfs(root)
        return parent

    def parent_array_to_first_child_next_sibling(self, parent: List[int
    ]) -> Tuple[List[int], List[int]]:
        """
        Chuyen doi tu mang cha sang first-child next-sibling

        Args:
            parent: Mang cha

        Returns:
            Tuple (first_child, next_sibling)
        """
        first_child = [-1] * self.num_nodes
        next_sibling = [-1] * self.num_nodes

        # Tao danh sach con cho moi node
        children = [[] for _ in range(self.num_nodes)]
        for i in range(self.num_nodes):
            if parent[i] != -1:
                children[parent[i]].append(i)

        # Thiet lap first_child va next_sibling
        for i in range(self.num_nodes):
            if children[i]:
                first_child[i] = children[i][0]
                for j in range(len(children[i]) - 1):
                    next_sibling[children[i][j]] = children[i][j + 1]

        return first_child, next_sibling

def test_graph_converter():
    """Ham test cho GraphConverter"""
    print("=== TEST GRAPH CONVERTER ===")

    n = 4
    converter = GraphConverter(n)

    # Ma tran ke mau
    matrix = [
        [0, 1, 1, 0],
        [1, 0, 1, 1],
        [1, 1, 0, 1],
        [0, 1, 1, 0]
    ]

    # Ma tran trong so
    weights = [
        [0, 2, 3, 0],
        [2, 0, 1, 4],
        [3, 1, 0, 2],
        [0, 4, 2, 0]
```

```
    ]

    # Test cac chuyen doi
    converter.print_matrix(matrix)

    adj_list = converter.matrix_to_list(matrix)
    converter.print_list(adj_list)

    ext_list = converter.list_to_extended(adj_list, weights)
    converter.print_extended(ext_list)

    adj_map = converter.list_to_map(adj_list)
    converter.print_map(adj_map)

    # Kiem tra tinh dung dan
    matrix_back = converter.list_to_matrix(adj_list)
    print("Ma tran sau khi chuyen doi nguoc lai:")
    converter.print_matrix(matrix_back)

def test_tree_converter():
    """Ham test cho TreeConverter"""
    print("=== TEST TREE CONVERTER ===")

    n = 6
    tree_converter = TreeConverter(n)

    # Mang cha mau: cay co goc la 0
    parent = [-1, 0, 0, 1, 1, 2]  # 0 la goc, 1,2 la con cua 0, 3,4 la con cua 1, 5 la con cua 2

    print("Mang cha:", parent)

    # Chuyen sang danh sach ke
    adj_list = tree_converter.parent_array_to_adj_list(parent)
    print("Danh sach ke cua cay:")
    for i, neighbors in enumerate(adj_list):
        print(f"{i}: {neighbors}")

    # Chuyen nguoc lai
    parent_back = tree_converter.adj_list_to_parent_array(adj_list, 0)
    print("Mang cha sau khi chuyen doi nguoc lai:", parent_back)

    # Chuyen sang first-child next-sibling
    first_child, next_sibling = tree_converter.parent_array_to_first_child_next_sibling(parent)
    print("First child:", first_child)
    print("Next sibling:", next_sibling)

if __name__ == "__main__":
    test_graph_converter()
    print()
    test_tree_converter()
```

Listing 4: Cài đặt đầy đủ bằng Python

7 Phân tích độ phức tạp

Chuyển đổi	Thời gian	Không gian	Ghi chú
Ma trận → Danh sách	$O(V^2)$	$O(V+E)$	Phải duyệt tất cả ma trận
Danh sách → Ma trận	$O(V+E)$	$O(V^2)$	Hiệu quả hơn
Danh sách → Mở rộng	$O(V+E)$	$O(V+E)$	Tuyến tính
Mở rộng → Danh sách	$O(V+E)$	$O(V+E)$	Tuyến tính
Danh sách → Map	$O(V+E)$	$O(V+E)$	Sử dụng hash
Map → Danh sách	$O(V+E)$	$O(V+E)$	Tuyến tính

Bảng 1: Phân tích độ phức tạp các thuật toán chuyển đổi

8 Xử lý các loại đồ thị đặc biệt

8.1 Đơn đồ thị (Simple Graph)

Đơn đồ thị không có cạnh lặp và khuyên (self-loop). Khi chuyển đổi:

- Kiểm tra $A[i][i] = 0$ (không có khuyên)
- Đảm bảo mỗi cạnh chỉ xuất hiện một lần

```
bool isSimpleGraph(const AdjMatrix& matrix) {
    int n = matrix.size();
    for (int i = 0; i < n; i++) {
        // Kiem tra khong co khuyen
        if (matrix[i][i] != 0) return false;

        for (int j = i + 1; j < n; j++) {
            // Voi do thi vo huong: A[i][j] = A[j][i]
            if (matrix[i][j] != matrix[j][i]) return false;

            // Kiem tra khong co canh lap (chi co 0 hoac 1)
            if (matrix[i][j] > 1) return false;
        }
    }
    return true;
}
```

Listing 5: Xử lý đơn đồ thị

8.2 Đa đồ thị (Multigraph)

Đa đồ thị cho phép nhiều cạnh giữa hai đỉnh. Ma trận kề lưu số lượng cạnh:

```
// Chuyen doi da do thi tu ma tran sang danh sach
AdjList multigraphMatrixToList(const AdjMatrix& matrix) {
    int n = matrix.size();
    AdjList adjList(n);
```

```

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            // Them j vao danh sach cua i, matrix[i][j] lan
            for (int k = 0; k < matrix[i][j]; k++) {
                adjList[i].push_back(j);
            }
        }
    }

    return adjList;
}

// Chuyen doi nguoc lai
AdjMatrix multigraphListToMatrix(const AdjList& adjList) {
    int n = adjList.size();
    AdjMatrix matrix(n, vector<int>(n, 0));

    for (int i = 0; i < n; i++) {
        for (int neighbor : adjList[i]) {
            matrix[i][neighbor]++;  // Tang so luong canh
        }
    }

    return matrix;
}
```

Listing 6: Xử lý đa đồ thị

8.3 Đồ thị tổng quát (General Graph)

Đồ thị tổng quát có thể có khuyên, cạnh lặp, và trọng số:

```
// Cau truc cho canh co trong so va chi so
struct Edge {
    int to;
    int weight;
    int id;

    Edge(int t, int w, int i) : to(t), weight(w), id(i) {}
};

typedef vector<vector<Edge>> GeneralAdjList;

// Chuyen doi tu ma tran trong so sang danh sach tong quat
GeneralAdjList weightedMatrixToGeneral(const vector<vector<int>>& matrix
    ) {
    int n = matrix.size();
    GeneralAdjList adjList(n);
    int edgeId = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (matrix[i][j] != 0) {
                adjList[i].push_back(Edge(j, matrix[i][j], edgeId++));
            }
```

```
        }
    }

    return adjList;
}
```

Listing 7: Xử lý đồ thị tổng quát

9 Biểu diễn cây và chuyển đổi

9.1 Mảng cha (Array of Parents)

Cách biểu diễn đơn giản nhất của cây:

- $parent[i]$ = chỉ số của nút cha của nút i
- $parent[root] = -1$ (hoặc chính nó)

9.2 First-Child Next-Sibling

Biểu diễn hiệu quả cho cây với số con không cố định:

- $firstChild[i]$ = con đầu tiên của nút i
- $nextSibling[i]$ = anh em tiếp theo của nút i

9.3 Chuyển đổi giữa các biểu diễn cây

```
class TreeConverter {
private:
    int numNodes;

public:
    TreeConverter(int n) : numNodes(n) {}

    // Chuyen tu mang cha sang first-child next-sibling
    pair<vector<int>, vector<int>> parentToFCNS(const vector<int>&
    parent) {
        vector<int> firstChild(numNodes, -1);
        vector<int> nextSibling(numNodes, -1);
        vector<vector<int>> children(numNodes);

        // Xay dung danh sach con
        for (int i = 0; i < numNodes; i++) {
            if (parent[i] != -1) {
                children[parent[i]].push_back(i);
            }
        }

        // Thiet lap first child va next sibling
        for (int i = 0; i < numNodes; i++) {
            if (!children[i].empty()) {
```

```
                firstChild[i] = children[i][0];
                for (int j = 0; j < children[i].size() - 1; j++) {
                    nextSibling[children[i][j]] = children[i][j + 1];
                }
            }
        }

        return {firstChild, nextSibling};
    }

    // Chuyen tu first-child next-sibling sang mang cha
    vector<int> FCNSToParent(const vector<int>& firstChild,
                             const vector<int>& nextSibling) {
        vector<int> parent(numNodes, -1);

        function<void(int, int)> dfs = [&](int node, int par) {
            parent[node] = par;

            // Duyet tat ca con cua node
            int child = firstChild[node];
            while (child != -1) {
                dfs(child, node);
                child = nextSibling[child];
            }
        };

        // Tim goc (node khong co cha)
        int root = -1;
        for (int i = 0; i < numNodes; i++) {
            bool hasParent = false;
            for (int j = 0; j < numNodes; j++) {
                if (firstChild[j] == i || nextSibling[j] == i) {
                    hasParent = true;
                    break;
                }
            }
            if (!hasParent) {
                root = i;
                break;
            }
        }

        if (root != -1) {
            dfs(root, -1);
        }

        return parent;
    }

    // Chuyen tu mang cha sang bieu dien do thi cay
    AdjList parentToAdjList(const vector<int>& parent) {
        AdjList adjList(numNodes);

        for (int i = 0; i < numNodes; i++) {
            if (parent[i] != -1) {
                // Them canh hai chieu (vo huong)
```

```
                adjList[parent[i]].push_back(i);
                adjList[i].push_back(parent[i]);
            }
        }

        return adjList;
    }

    // Chuyen tu bieu dien do thi sang mang cha
    vector<int> adjListToParent(const AdjList& adjList, int root = 0) {
        vector<int> parent(numNodes, -1);
        vector<bool> visited(numNodes, false);

        function<void(int)> dfs = [&](int node) {
            visited[node] = true;
            for (int neighbor : adjList[node]) {
                if (!visited[neighbor]) {
                    parent[neighbor] = node;
                    dfs(neighbor);
                }
            }
        };

        dfs(root);
        return parent;
    }

    // In cay theo dang parent array
    void printParentArray(const vector<int>& parent) {
        cout << "Mang cha: ";
        for (int i = 0; i < numNodes; i++) {
            cout << parent[i] << " ";
        }
        cout << endl;
    }

    // In cay theo dang FCNS
    void printFCNS(const vector<int>& firstChild, const vector<int>&
nextSibling) {
        cout << "First Child: ";
        for (int i = 0; i < numNodes; i++) {
            cout << firstChild[i] << " ";
        }
        cout << endl;

        cout << "Next Sibling: ";
        for (int i = 0; i < numNodes; i++) {
            cout << nextSibling[i] << " ";
        }
        cout << endl;
    }

    // In cay theo dang hierarchical
    void printTree(const vector<int>& parent, int root = -1) {
        if (root == -1) {
            // Tim goc
```

```
            for (int i = 0; i < numNodes; i++) {
                if (parent[i] == -1) {
                    root = i;
                    break;
                }
            }
        }

        vector<vector<int>> children(numNodes);
        for (int i = 0; i < numNodes; i++) {
            if (parent[i] != -1) {
                children[parent[i]].push_back(i);
            }
        }

        function<void(int, string)> printNode = [&](int node, string
    indent) {
            cout << indent << node << endl;
            for (int child : children[node]) {
                printNode(child, indent + "  ");
            }
        };

        cout << "Cay hierarchy:" << endl;
        printNode(root, "");
    }
};
```

Listing 8: Chuyển đổi biểu diễn cây

10 Ứng dụng

10.1 Ứng dụng thực tế

1. **Mạng xã hội**: Chuyển đổi giữa danh sách bạn bè và ma trận quan hệ
2. **Định tuyến mạng**: Tối ưu biểu diễn topology mạng
3. **Cây quyết định**: Chuyển đổi giữa các dạng biểu diễn cây
4. **Cơ sở dữ liệu đồ thị**: Lưu trữ và truy vấn hiệu quả

11 Tối ưu hóa và cải tiến

11.1 Sử dụng Sparse Matrix

Với đồ thị thưa, sử dụng compressed sparse row (CSR):

```
struct CSRMatrix {
    vector<int> values;     // Gia tri khac 0
    vector<int> colIndices; // Chi so cot
    vector<int> rowPtr;     // Con tro hang
```

```
    int numRows, numCols;

    CSRMatrix(int rows, int cols) : numRows(rows), numCols(cols) {
        rowPtr.resize(rows + 1, 0);
    }

    // Chuyen tu ma tran thuong sang CSR
    static CSRMatrix fromDenseMatrix(const AdjMatrix& matrix) {
        int n = matrix.size();
        CSRMatrix csr(n, n);

        for (int i = 0; i < n; i++) {
            csr.rowPtr[i] = csr.values.size();
            for (int j = 0; j < n; j++) {
                if (matrix[i][j] != 0) {
                    csr.values.push_back(matrix[i][j]);
                    csr.colIndices.push_back(j);
                }
            }
        }
        csr.rowPtr[n] = csr.values.size();

        return csr;
    }

    // Chuyen CSR sang danh sach ke
    AdjList toAdjList() const {
        AdjList adjList(numRows);

        for (int i = 0; i < numRows; i++) {
            for (int j = rowPtr[i]; j < rowPtr[i + 1]; j++) {
                adjList[i].push_back(colIndices[j]);
            }
        }

        return adjList;
    }

    // Truy cap phan tu O(log degree)
    int get(int row, int col) const {
        for (int j = rowPtr[row]; j < rowPtr[row + 1]; j++) {
            if (colIndices[j] == col) {
                return values[j];
            }
        }
        return 0;
    }
};
```

Listing 9: Compressed Sparse Row implementation

11.2 Parallel Processing

Sử dụng OpenMP để tăng tốc độ chuyển đổi:

```
#include <omp.h>
```

```

AdjList parallelMatrixToList(const AdjMatrix& matrix) {
    int n = matrix.size();
    AdjList adjList(n);

    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (matrix[i][j] != 0) {
                #pragma omp critical
                adjList[i].push_back(j);
            }
        }
    }

    return adjList;
}
```

Listing 10: Parallel matrix to list conversion

12 Kết luận

12.1 Tóm tắt

Bài toán chuyển đổi biểu diễn đồ thị là một phần quan trọng trong khoa học máy tính, với nhiều ứng dụng thực tế. Qua tài liệu này, chúng ta đã:

- Tìm hiểu 4 dạng biểu diễn chính của đồ thị
- Phân tích ưu nhược điểm của từng phương pháp
- Cài đặt thuật toán chuyển đổi hiệu quả
- Xử lý các trường hợp đặc biệt và tối ưu hóa
- Mở rộng cho biểu diễn cây với 3 dạng khác nhau

A Code hoàn chỉnh - Phiên bản tối ưu

```
// File: GraphConverter.h
#pragma once
#include <iostream>
#include <vector>
#include <list>
#include <map>
#include <unordered_map>
#include <set>
#include <unordered_set>
#include <queue>
#include <stack>
#include <algorithm>
#include <cassert>
```

```
14
15 template<typename T = int>
16 class OptimizedGraphConverter {
17 public:
18     using AdjMatrix = std::vector<std::vector<T>>;
19     using AdjList = std::vector<std::vector<int>>;
20     using ExtendedAdjList = std::vector<std::vector<std::pair<int, T>>>;
21     using AdjMap = std::unordered_map<int, std::unordered_set<int>>;
22
23 private:
24     size_t numVertices;
25
26 public:
27     explicit OptimizedGraphConverter(size_t n) : numVertices(n) {}
28
29     // Core conversion functions with template support
30     template<typename MatrixType>
31     AdjList matrixToList(const MatrixType& matrix) const {
32         AdjList result(numVertices);
33
34         for (size_t i = 0; i < numVertices; ++i) {
35             result[i].reserve(std::count_if(
36                 matrix[i].begin(), matrix[i].end(),
37                 [](const auto& val) { return val != 0; }
38             ));
39
40             for (size_t j = 0; j < numVertices; ++j) {
41                 if (matrix[i][j] != 0) {
42                     result[i].push_back(static_cast<int>(j));
43                 }
44             }
45         }
46
47         return result;
48     }
49
50     AdjMatrix listToMatrix(const AdjList& adjList) const {
51         AdjMatrix result(numVertices, std::vector<T>(numVertices, T{}));
52
53         for (size_t i = 0; i < numVertices; ++i) {
54             for (int neighbor : adjList[i]) {
55                 result[i][neighbor] = T{1};
56             }
57         }
58
59         return result;
60     }
61
62     // Additional utility functions
63     bool isValidConversion(const AdjMatrix& original, const AdjList&
    converted) const {
64         auto backConverted = listToMatrix(converted);
65         return matricesEqual(original, backConverted);
66     }
67
68     void printStatistics(const AdjMatrix& matrix) const {
```

```
        size_t edges = 0, maxDegree = 0, minDegree = numVertices;

        for (size_t i = 0; i < numVertices; ++i) {
            size_t degree = 0;
            for (size_t j = 0; j < numVertices; ++j) {
                if (matrix[i][j] != 0) {
                    ++edges;
                    ++degree;
                }
            }
            maxDegree = std::max(maxDegree, degree);
            minDegree = std::min(minDegree, degree);
        }

        std::cout << "Graph Statistics:" << std::endl;
        std::cout << "Vertices: " << numVertices << std::endl;
        std::cout << "Edges: " << edges << std::endl;
        std::cout << "Density: " << static_cast<double>(edges) / (numVertices * numVertices) << std::endl;
        std::cout << "Max Degree: " << maxDegree << std::endl;
        std::cout << "Min Degree: " << minDegree << std::endl;
    }

private:
    bool matricesEqual(const AdjMatrix& a, const AdjMatrix& b) const {
        if (a.size() != b.size()) return false;
        for (size_t i = 0; i < a.size(); ++i) {
            if (a[i] != b[i]) return false;
        }
        return true;
    }
};

// Complete demonstration
int main() {
    std::cout << "=== OPTIMIZED GRAPH CONVERTER DEMONSTRATION ===" << std::endl;

    OptimizedGraphConverter<int> converter(5);

    // Create a sample graph
    OptimizedGraphConverter<int>::AdjMatrix sampleGraph = {
        {0, 1, 1, 0, 1},
        {1, 0, 1, 1, 0},
        {1, 1, 0, 1, 1},
        {0, 1, 1, 0, 1},
        {1, 0, 1, 1, 0}
    };

    converter.printStatistics(sampleGraph);

    auto adjList = converter.matrixToList(sampleGraph);
    auto backMatrix = converter.listToMatrix(adjList);

    std::cout << "\nConversion validation: "
              << (converter.isValidConversion(sampleGraph, adjList) ? "
```

```
    PASSED" : "FAILED")
123              << std::endl;
124
125     return 0;
126 }
```

Listing 11: Final optimized implementation

— *HẾT* —