BÁO CÁO ĐỒ ÁN MÔN HỌC

Tổ Hợp & Lý Thuyết Đồ Thị

BÀI TOÁN 4: CÁC BÀI TOÁN DUYỆT ĐỒ THỊ & CÂY

Chuyển Đổi Giữa Các Dạng Biểu Diễn Đồ Thị & Cây

(Converting Between Graph & Tree Representations)

Sinh viên: Huỳnh Nhật Quang

Ngày 23 tháng 7 năm 2025

Mục lục

1 Giới Thiệu Bài Toán **4**

1.1 Mô Tả Chi Tiết Bài Toán 4

1.2 Ý Nghĩa Thực Tiễn 4

2 Cơ Sở Lý Thuyết Toán Học **5**

2.1 Lý Thuyết Đồ Thị 5

2.1.1 Định Nghĩa Toán Học Cơ Bản 5

2.1.2 Các Dạng Biểu Diễn Đồ Thị - Phân Tích Toán Học 5

2.2 Lý Thuyết Cây 6

2.2.1 Định Nghĩa Toán Học của Cây 6

2.2.2 Các Dạng Biểu Diễn Cây - Phân Tích Toán Học 6

3 Phân Tích Thuật Toán Chi Tiết **7**

3.1 Derivation Các Công Thức Đệ Quy 7

3.1.1 Chuyển Đổi Ma Trận Kề → Danh Sách Kề 7

3.1.2 Chuyển Đổi Mảng Cha → First-Child Next-Sibling 8

3.2 Quy Hoạch Động trong Chuyển Đổi 9

3.2.1 Tối Ưu Hóa Tính Bậc trong Extended Adjacency List 9

4 Chi Tiết Cài Đặt và Giải Thích Code **9**

4.1 Cài Đặt C++ - Phân Tích Chi Tiết 9

4.1.1 Cấu Trúc Dữ Liệu Edge 9

4.1.2 Lớp GraphConverter - Biến Quan Trọng 10

4.1.3 Hàm Chuyển Đổi Quan Trọng 11

4.2 Cài Đặt Tree Converter 12

4.2.1 Cấu Trúc TreeNode 12

4.2.2 Thuật Toán Parent Array → FCNS 12

4.2.3 So Sánh Hiệu Suất Các Biểu Diễn 14

5 Kết Luận **14**

5.1 Tổng Kết Kết Quả 14

5.2 Đóng Góp Chính 14

5.3 Kiến Thức Thu Được 15

5.4 Ý Nghĩa và Ứng Dụng 15

6 Tài Liệu Tham Khảo **15**

1 Giới Thiệu Bài Toán

1.1 Mô Tả Chi Tiết Bài Toán

Bài toán 4 yêu cầu chúng ta thực hiện việc chuyển đổi giữa các dạng biểu diễn khác nhau của đồ thị và cây. Đây là một bài toán quan trọng trong lý thuyết đồ thị, có ứng dụng rộng rãi trong thiết kế thuật toán và tối ưu hóa bộ nhớ.

Yêu cầu cụ thể:

Đối với đồ thị (Graph):

- **Ba loại đồ thị cần xử lý:**
  1. **Đơn đồ thị (Simple Graph):** Không có cạnh lặp, không có khuyên (self-loop)
  2. **Đa đồ thị (Multigraph):** Cho phép nhiều cạnh giữa hai đỉnh
  3. **Đồ thị tổng quát (General Graph):** Có trọng số, có thể có trọng số âm
- **Bốn dạng biểu diễn:**
  1. **Adjacency Matrix** - Ma trận kề
  2. **Adjacency List** - Danh sách kề
  3. **Extended Adjacency List** - Danh sách kề mở rộng
  4. **Adjacency Map** - Bản đồ kề

Đối với cây (Tree):

- **Ba dạng biểu diễn:**
  1. **Array of Parents** - Mảng cha
  2. **First-Child Next-Sibling** - Con đầu anh em kế
  3. **Graph-based Representation** - Biểu diễn dựa trên đồ thị

Tổng số chương trình chuyển đổi: $3 \times A_4^3 + A_3^2 = 36 + 6 = 42$ chương trình.

1.2 Ý Nghĩa Thực Tiễn

Việc chuyển đổi giữa các dạng biểu diễn có ý nghĩa quan trọng trong:

- **Tối ưu hóa thuật toán:** Mỗi dạng biểu diễn có ưu thế riêng cho các thao tác khác nhau
- **Quản lý bộ nhớ hiệu quả:** Lựa chọn biểu diễn phù hợp với đặc điểm của đồ thị
- **Tương thích giữa các hệ thống:** Chuyển đổi dữ liệu giữa các ứng dụng khác nhau
- **Hiểu sâu về cấu trúc dữ liệu:** Nắm vững bản chất của từng biểu diễn

2 Cơ Sở Lý Thuyết Toán Học

2.1 Lý Thuyết Đồ Thị

2.1.1 Định Nghĩa Toán Học Cơ Bản

Một đồ thị G được định nghĩa là một cặp có thứ tự $G = (V, E)$ trong đó:

- V là tập hợp hữu hạn các đỉnh (vertices)
- E là tập hợp các cạnh (edges) nối các đỉnh

Công thức toán học cho các loại đồ thị:

1. **Đơn đồ thị:** $E \subseteq V \times V$ và $\forall v \in V : (v, v) \notin E$
2. **Đa đồ thị:** Cho phép $|(u, v)| > 1$ với $u, v \in V$
3. **Đồ thị có trọng số:** $w : E \to \mathbb{R}$ (hàm trọng số)

2.1.2 Các Dạng Biểu Diễn Đồ Thị - Phân Tích Toán Học

1. Ma Trận Kề (Adjacency Matrix)
Cho đồ thị $G = (V, E)$ với $|V| = n$, ma trận kề A là ma trận vuông cấp $n \times n$:

$$A[i][j] = \begin{cases} 1 & \text{nếu có cạnh từ đỉnh } i \text{ đến đỉnh } j \text{ (đơn đồ thị)} \\ k & \text{số cạnh từ đỉnh } i \text{ đến đỉnh } j \text{ (đa đồ thị)} \\ w_{ij} & \text{trọng số cạnh từ đỉnh } i \text{ đến đỉnh } j \text{ (đồ thị tổng quát)} \\ 0 & \text{không có cạnh} \end{cases} \tag{1}$$

Tính chất toán học quan trọng:

- Đối với đồ thị vô hướng: $A[i][j] = A[j][i]$ (ma trận đối xứng)
- Tổng số cạnh: $|E| = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} A[i][j]$ (đồ thị vô hướng)
- Bậc của đỉnh i: $deg(i) = \sum_{j=1}^{n} A[i][j]$

Độ phức tạp tính toán:

- Không gian lưu trữ: $\mathcal{O}(n^2)$
- Kiểm tra cạnh (i, j): $\mathcal{O}(1)$
- Duyệt tất cả láng giềng của đỉnh i: $\mathcal{O}(n)$
- Thêm/xóa cạnh: $\mathcal{O}(1)$

2. Danh Sách Kề (Adjacency List)

Cho đồ thị $G = (V, E)$, danh sách kề adj là một mảng gồm $|V|$ danh sách:

$$adj[i] = \{j : (i, j) \in E\} \tag{2}$$

Công thức tính toán:

- Tổng số phần tử trong tất cả danh sách: $\sum_{i=1}^{n} |adj[i]| = |E|$ (đồ thị có hướng)
- Tổng số phần tử: $\sum_{i=1}^{n} |adj[i]| = 2|E|$ (đồ thị vô hướng)
- Bậc của đỉnh i: $deg(i) = |adj[i]|$

Độ phức tạp tính toán:

- Không gian lưu trữ: $\mathcal{O}(n + m)$ với $m = |E|$
- Kiểm tra cạnh (i, j): $\mathcal{O}(deg(i))$
- Duyệt láng giềng của đỉnh i: $\mathcal{O}(deg(i))$
- Thêm cạnh: $\mathcal{O}(1)$
- Xóa cạnh: $\mathcal{O}(deg(i))$

2.2 Lý Thuyết Cây

2.2.1 Định Nghĩa Toán Học của Cây

Một cây T là một đồ thị liên thông không có chu trình với các tính chất đặc biệt:

Các định nghĩa tương đương:

1. T là đồ thị liên thông và $|E| = |V| - 1$
2. T là đồ thị không có chu trình và $|E| = |V| - 1$
3. Giữa hai đỉnh bất kỳ có đúng một đường đi đơn
4. T liên thông và việc loại bỏ bất kỳ cạnh nào cũng làm T không liên thông
5. T không có chu trình và việc thêm bất kỳ cạnh nào cũng tạo ra đúng một chu trình

2.2.2 Các Dạng Biểu Diễn Cây - Phân Tích Toán Học

1. Mảng Cha (Array of Parents)

Cho cây T có gốc r với n đỉnh được đánh số từ 0 đến $n - 1$:

$$parent[i] = \begin{cases} j & \text{nếu đỉnh } j \text{ là cha của đỉnh } i \\ -1 & \text{nếu đỉnh } i \text{ là gốc} \end{cases} \tag{3}$$

Tính chất toán học:

- Có đúng một đỉnh r thỏa mãn $parent[r] = -1$
- $\forall i \neq r : parent[i] \in \{0, 1, ..., n-1\}$
- Không tồn tại chu trình

Công thức đệ quy quan trọng:

Độ sâu của đỉnh v:

$$depth(v) = \begin{cases} 0 & \text{nếu } parent[v] = -1 \\ 1 + depth(parent[v]) & \text{ngược lại} \end{cases} \tag{4}$$

2. First-Child Next-Sibling (FCNS)

Đây là phép biến đổi toán học từ cây n-nhánh thành cây nhị phân:

$$first\_child[v] = \text{con đầu tiên của } v \text{ (nếu có)} \tag{5}$$

$$next\_sibling[v] = \text{anh em kế tiếp của } v \text{ (nếu có)} \tag{6}$$

Định lý biến đổi Knuth: Mọi cây n-nhánh đều có thể biểu diễn duy nhất dưới dạng cây nhị phân bằng phép biến đổi FCNS.

3 Phân Tích Thuật Toán Chi Tiết

3.1 Derivation Các Công Thức Đệ Quy

3.1.1 Chuyển Đổi Ma Trận Kề → Danh Sách Kề

Bài toán: Cho ma trận kề $A[n][n]$, xây dựng danh sách kề $adj[n]$.

Algorithm 1 Chuyển đổi từ Ma trận Kề sang Danh sách Kề

Require: Ma trận kề $A[n][n]$
Ensure: Danh sách kề $adj[n]$

```
Khởi tạo: adj[i] ← ∅ với ∀i ∈ {0, 1, ..., n − 1}
for i = 0 to n − 1 do
    for j = 0 to n − 1 do
        if A[i][j] ≠ 0 then
            Thêm cạnh (i, j) với trọng số A[i][j] vào adj[i]
        end if
    end for
end for
```

Phân tích độ phức tạp bằng công thức đệ quy:

Gọi $T(n)$ là thời gian thực hiện thuật toán với đồ thị n đỉnh:

$$T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \mathcal{O}(1) \tag{7}$$

$$= \sum_{i=0}^{n-1} \mathcal{O}(n) \tag{8}$$

$$= n \times \mathcal{O}(n) \tag{9}$$

$$= \mathcal{O}(n^2) \tag{10}$$

3.1.2 Chuyển Đổi Mảng Cha → First-Child Next-Sibling

Bài toán: Cho mảng cha $parent[n]$, xây dựng cấu trúc FCNS.

Algorithm 2 Chuyển đổi từ Mảng Cha sang FCNS

Require: Mảng cha $parent[n]$
Ensure: Cấu trúc FCNS với $first\_child[n]$ và $next\_sibling[n]$

Khởi tạo: $first\_child[i] \leftarrow null$, $next\_sibling[i] \leftarrow null$ $\forall i$
for $i = 0$ **to** $n - 1$ **do**
    Tạo node i
end for
for $i = 0$ **to** $n - 1$ **do**
    if $parent[i] \neq -1$ **then**
        $p \leftarrow parent[i]$
        if $first\_child[p] = null$ **then**
            $first\_child[p] \leftarrow i$
        else
            $sibling \leftarrow first\_child[p]$
            while $next\_sibling[sibling] \neq null$ **do**
                $sibling \leftarrow next\_sibling[sibling]$
            end while
            $next\_sibling[sibling] \leftarrow i$
        end if
    end if
end for

Phân tích độ phức tạp chi tiết:

Gọi d_v là số con của đỉnh v. Thuật toán có độ phức tạp:

$$T(n) = \mathcal{O}(n) + \sum_{v \in V} \mathcal{O}(d_v - 1) \tag{11}$$

$$= \mathcal{O}(n) + \mathcal{O}\left(\sum_{v \in V} d_v\right) - \mathcal{O}(|V|) \tag{12}$$

$$= \mathcal{O}(n) + \mathcal{O}(n-1) - \mathcal{O}(n) \tag{13}$$

$$= \mathcal{O}(n) \tag{14}$$

Vì $\sum_{v \in V} d_v = n - 1$ (tổng số cạnh trong cây).

3.2 Quy Hoạch Động trong Chuyển Đổi

3.2.1 Tối Ưu Hóa Tính Bậc trong Extended Adjacency List

Bài toán: Tính in_degree và out_degree cho mọi đỉnh một cách hiệu quả.

Công thức quy hoạch động:

Với danh sách kề $adj[n]$:

$$out\_degree[i] = |adj[i]| \tag{15}$$

$$in\_degree[j] = \sum_{i=0}^{n-1} \mathbf{1}_{j \in adj[i]} \tag{16}$$

trong đó $\mathbf{1}_{condition}$ là hàm indicator.

Algorithm 3 Tính bậc bằng Quy hoạch động

Require: Danh sách kề $adj[n]$
Ensure: Mảng $in\_degree[n]$ và $out\_degree[n]$

```
Khởi tạo: in_degree[i] ← 0, out_degree[i] ← 0 ∀i
for i = 0 to n − 1 do
    out_degree[i] ← |adj[i]|
    for each j ∈ adj[i] do
        in_degree[j] ← in_degree[j] + 1
    end for
end for
```

4 Chi Tiết Cài Đặt và Giải Thích Code

4.1 Cài Đặt C++ - Phân Tích Chi Tiết

4.1.1 Cấu Trúc Dữ Liệu Edge

```cpp
struct Edge {
    int to;         //    nh      ch
    int weight;     //  T r ng   s    c nh
    int id;         // ID  c nh  (cho multigraph)

    Edge(int t, int w = 1, int i = 0)
        : to(t), weight(w), id(i) {}
};
```

Listing 1: Cấu trúc Edge cho đồ thị

Giải thích ý nghĩa các biến:

- `int to`: Đại diện cho đỉnh đích của cạnh. Đây là biến quan trọng nhất vì nó xác định mối quan hệ kề giữa các đỉnh. Trong ngữ cảnh toán học, `to` biểu diễn đỉnh j trong cạnh (i, j).
- `int weight`: Lưu trữ trọng số của cạnh. Đối với đơn đồ thị, giá trị mặc định là 1. Đối với đồ thị tổng quát, có thể là số âm hoặc dương. Biến này quan trọng cho các thuật toán tìm đường đi ngắn nhất.
- `int id`: Định danh duy nhất cho cạnh, đặc biệt quan trọng trong multigraph nơi có thể có nhiều cạnh giữa hai đỉnh.

4.1.2 Lớp GraphConverter - Biến Quan Trọng

```cpp
class GraphConverter {
private:
    int n; //  S      nh          t h   -  b i n   c t   l  i

public:
    // Ma  t r n   k   cho 3  l o i        t h
    vector<vector<int>> adjacencyMatrix_simple;
    vector<vector<int>> adjacencyMatrix_multi;
    vector<vector<int>> adjacencyMatrix_general;

    // Danh  s  ch  k   cho 3  l o i        t h
    vector<vector<int>> adjacencyList_simple;
    vector<vector<Edge>> adjacencyList_multi;
    vector<vector<Edge>> adjacencyList_general;

    // Extended adjacency list
    struct ExtendedNode {
        vector<int> neighbors;  // Danh  s  ch  l  ng  g i ng
        int in_degree;          //  B c   v  o
        int out_degree;         //  B c   ra

        ExtendedNode() : in_degree(0), out_degree(0) {}
    };

    vector<ExtendedNode> extendedAdjList_simple;
    // ...
};
```

Listing 2: Lớp GraphConverter với các biến chính

Phân tích ý nghĩa các biến chính:

1. `int n`: Biến cốt lõi nhất, xác định kích thước của tất cả cấu trúc dữ liệu. Trong toán học, đây chính là $|V|$ - số lượng đỉnh.
2. `adjacencyMatrix_*`: Mảng 2 chiều biểu diễn ma trận kề. Mỗi phần tử `[i][j]` có ý nghĩa khác nhau tùy loại đồ thị.
3. `adjacencyList_*`: Mảng các danh sách, trong đó `adjacencyList[i]` chứa tất cả đỉnh kề với đỉnh i.
4. `ExtendedNode`: Cấu trúc mở rộng chứa thêm thông tin về bậc của đỉnh.

4.1.3 Hàm Chuyển Đổi Quan Trọng

```
1  void GraphConverter::matrixToList_Simple() {
2      // B c 1: K h i t o danh s ch k r ng
3      adjacencyList_simple.clear();
4      adjacencyList_simple.resize(n);
5
6      // B c 2: D u y t qua t ng p h n t c a ma t r n
7      for (int i = 0; i < n; i++) {
8          for (int j = 0; j < n; j++) {
9              if (adjacencyMatrix_simple[i][j] == 1) {
10                 // C c nh t i n j
11                 adjacencyList_simple[i].push_back(j);
12             }
13         }
14     }
15
16     cout << "Converted Simple Graph: Matrix to List" << endl;
17 }
```

Listing 3: Chuyển đổi Ma trận sang Danh sách - Simple Graph

Giải thích chi tiết từng dòng code:

1. **Dòng 3-4**: Khởi tạo cấu trúc dữ liệu đầu ra
   - `clear()`: Xóa toàn bộ dữ liệu cũ
   - `resize(n)`: Tạo đúng n danh sách rỗng
2. **Dòng 7-8**: Vòng lặp duyệt ma trận
   - Biến `i`: Đỉnh nguồn
   - Biến `j`: Đỉnh đích
3. **Dòng 9-12**: Kiểm tra và thêm cạnh
   - Điều kiện: `adjacencyMatrix_simple[i][j] == 1`
   - Thao tác: `push_back(j)` - thêm đỉnh j vào danh sách của đỉnh i

4.2 Cài Đặt Tree Converter

4.2.1 Cấu Trúc TreeNode

```
struct TreeNode {
    int data;                    // D    l i u   c a  node
    TreeNode* parent;           // Con  t r     n    node cha
    TreeNode* firstChild;       // Con  t r     n    con    u    ti n
    TreeNode* nextSibling;      // Con  t r     n    anh em  k    t i p

    TreeNode(int val) : data(val), parent(nullptr),
                        firstChild(nullptr), nextSibling(nullptr) {}
};
```

Listing 4: Cấu trúc TreeNode cho FCNS

Ý nghĩa các biến trong TreeNode:

1. `int data`: Lưu trữ giá trị/ID của node
2. `TreeNode* parent`: Con trỏ đến node cha
   - Bằng `nullptr` nếu là node gốc
   - Giúp di chuyển lên trên trong cây
3. `TreeNode* firstChild`: Con trỏ đến con đầu tiên
   - Đây là điểm bắt đầu để duyệt tất cả con của node
   - Bằng `nullptr` nếu node là lá
   - Trong biến đổi FCNS, đây trở thành "left child"của cây nhị phân
4. `TreeNode* nextSibling`: Con trỏ đến anh em kế tiếp
   - Tạo chuỗi liên kết giữa các con của cùng một cha
   - Bằng `nullptr` nếu là con cuối cùng
   - Trong biến đổi FCNS, đây trở thành "right child"của cây nhị phân

4.2.2 Thuật Toán Parent Array → FCNS

```
void TreeConverter::parentArrayToFCNS() {
    //  B    c  1:  T o   t t   c    c  c  node
    fcnsNodes.clear();
    fcnsNodes.resize(n);
    for (int i = 0; i < n; i++) {
        fcnsNodes[i] = new TreeNode(i);
    }

    //  B    c  2:  X y   d ng  quan  h   cha-con  v   anh-em
    for (int i = 0; i < n; i++) {
        if (parentArray[i] != -1) {
            int parentIdx = parentArray[i];
            TreeNode* parentNode = fcnsNodes[parentIdx];
```

```
14              TreeNode* childNode = fcnsNodes[i];
15
16              // Thiết lập quan hệ cha
17              childNode->parent = parentNode;
18
19              // Xử lý đúng cấu trúc first-child next-sibling
20              if (parentNode->firstChild == nullptr) {
21                  // Là con đầu tiên
22                  parentNode->firstChild = childNode;
23              } else {
24                  // Thêm vào cuối chuỗi anh em
25                  TreeNode* sibling = parentNode->firstChild;
26                  while (sibling->nextSibling != nullptr) {
27                      sibling = sibling->nextSibling;
28                  }
29                  sibling->nextSibling = childNode;
30              }
31          } else {
32              // Là node gốc
33              root = i;
34          }
35      }
36
37      cout << "Converted Tree: Parent Array to FCNS" << endl;
38  }
```

Listing 5: Chuyển đổi Parent Array sang FCNS

Phân tích chi tiết từng bước:

1. **Bước 1 - Tạo nodes (Dòng 3-7)**:
   - Tạo n nodes độc lập, chưa có liên kết
   - Mỗi node có `data = i` với i từ 0 đến $n-1$
   - Độ phức tạp: $\mathcal{O}(n)$
2. **Bước 2 - Xây dựng liên kết (Dòng 10-32)**:
   - **Xử lý quan hệ cha-con**: `childNode->parent = parentNode`
   - **Xây dựng FCNS**: Phân biệt con đầu tiên và các con tiếp theo

Tại sao phải duyệt chuỗi anh em?

Trong cấu trúc FCNS, các con của một node được tổ chức thành chuỗi liên kết:

`Parent → FirstChild → NextSibling → NextSibling → ... → null`

Để thêm con mới, phải tìm con cuối cùng trong chuỗi và gắn con mới vào cuối.

4.2.3 So Sánh Hiệu Suất Các Biểu Diễn

Bảng 1: So sánh độ phức tạp các biểu diễn

Biểu diễn	**Không gian**	**Thêm cạnh**	**Xóa cạnh**	**Kiểm tra cạnh**
Adjacency Matrix	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Adjacency List	$\mathcal{O}(n+m)$	$\mathcal{O}(1)$	$\mathcal{O}(deg(v))$	$\mathcal{O}(deg(v))$
Extended Adj. List	$\mathcal{O}(n+m)$	$\mathcal{O}(1)$	$\mathcal{O}(deg(v))$	$\mathcal{O}(deg(v))$
Adjacency Map	$\mathcal{O}(n+m)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Kết luận về lựa chọn biểu diễn:

- **Đồ thị dày đặc** ($m > \frac{n^2}{4}$): Nên sử dụng **Adjacency Matrix**
- **Đồ thị thưa** ($m < \frac{n^2}{10}$): Nên sử dụng **Adjacency List**
- **Nhiều truy vấn bậc**: Sử dụng **Extended Adjacency List**
- **Nhiều truy vấn tồn tại cạnh**: Sử dụng **Adjacency Map**

5 Kết Luận

5.1 Tổng Kết Kết Quả

Đồ án đã hoàn thành thành công việc chuyển đổi giữa các dạng biểu diễn đồ thị và cây với tổng cộng 42 chương trình chuyển đổi:

- **36 chuyển đổi đồ thị**: Bao gồm 3 loại đồ thị (Simple, Multigraph, General) với 4 dạng biểu diễn (Matrix, List, Extended List, Map)
- **6 chuyển đổi cây**: Giữa 3 dạng biểu diễn (Parent Array, FCNS, Graph-based)

5.2 Đóng Góp Chính

1. **Phân tích toán học sâu sắc**: Derivation đầy đủ các công thức đệ quy và độ phức tạp thuật toán
2. **Cài đặt hoàn chỉnh**: Source code C++ và Python với giải thích chi tiết từng thành phần
3. **Verification system**: Đảm bảo tính đúng đắn thông qua chuyển đổi ngược và kiểm tra tính chất toán học
4. **Phân tích so sánh**: Đánh giá trade-off giữa các dạng biểu diễn khác nhau

5.3 Kiến Thức Thu Được

Qua quá trình thực hiện đồ án, đã nắm vững:

- **Lý thuyết đồ thị cơ bản**: Hiểu sâu về các dạng biểu diễn và trade-off giữa chúng
- **Phân tích thuật toán**: Kỹ năng derivation công thức đệ quy và tính toán độ phức tạp
- **Lập trình hệ thống**: Quản lý bộ nhớ, tối ưu hóa và debugging kỹ thuật
- **Ứng dụng thực tiễn**: Khả năng lựa chọn biểu diễn phù hợp cho từng bài toán cụ thể

5.4 Ý Nghĩa và Ứng Dụng

Đồ án này không chỉ hoàn thành yêu cầu học thuật mà còn:

- **Cung cấp nền tảng vững chắc** cho các nghiên cứu tiếp theo trong lĩnh vực lý thuyết đồ thị
- **Phát triển kỹ năng phân tích** và thiết kế thuật toán hiệu quả
- **Hiểu rõ bản chất** của các cấu trúc dữ liệu và sự chuyển đổi giữa chúng
- **Áp dụng được** vào các bài toán thực tế như mạng xã hội, hệ thống định tuyến, quản lý dữ liệu

Kết quả nghiên cứu cho thấy việc lựa chọn đúng dạng biểu diễn có thể cải thiện đáng kể hiệu suất của thuật toán, đồng thời tiết kiệm tài nguyên hệ thống.

6 Tài Liệu Tham Khảo

Tài liệu

[1] Gabriel Valiente. *Algorithms on Trees and Graphs: With Python Code*. Springer, 2021.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms, Third Edition*. MIT Press, 2009.

[3] V. K. Balakrishnan. *Schaum's Outline of Graph Theory*. McGraw-Hill, 1997.

[4] Robert Sedgewick, Kevin Wayne. *Algorithms, 4th Edition*. Addison-Wesley, 2011.

[5] Robert Endre Tarjan. *Data Structures and Network Algorithms*. SIAM, 1983.

[6] Donald E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, 1997.

[7] Shahriar Shahriari. *An Invitation to Combinatorics*. Cambridge University Press, 2022.

[8] Boris Goldengorin (Editor). *Optimization Problems in Graph Theory*. Springer, 2018.