

TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHIỆP

KHOA ĐIỆN TỬ

Bộ môn: Công nghệ thông tin



BÀI TẬP LỚN

MÔN HỌC

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ĐỀ TÀI: CÁC GIẢI THUẬT CỦA CÁC CHƯƠNG

Sinh viên:ĐẶNG ĐÌNH ĐẠT.....

Mã số sinh viên: K225480106003

Lớp:K58KMT.K01.....

Giáo viên hướng dẫn:Th.S Nguyễn Thị Hương.....

Thái Nguyên – 2024

PHIẾU GIAO BÀI TẬP TIỂU LUẬN

MÔN HỌC: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

BỘ MÔN : TIN HỌC CÔNG NGHIỆP

Sinh viên: Đặng Đình Đạt

MSV :K225480106003

Lớp: K58KMT.K01

Ngành: Kỹ thuật máy tính

Giáo viên hướng dẫn: Th.S Nguyễn Thị Hương

Ngày giao đề: 10/08/2024 Ngày hoàn thành: 05/11/2024

.....

I. nội dung bài tập tiểu luận:

- Đề tài môn học đã giao cho sinh viên thực hiện
- Yêu cầu khảo sát, phân tích, thiết kế, cài đặt chương trình

II. Hình thức nộp

1. Báo cáo bản word
2. File source code chương trình

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

(Ký ghi rõ họ tên)

MỤC LỤC

LỜI NÓI ĐẦU	6
CHƯƠNG 3. ĐỆ QUY	7
1. Khái niệm đệ quy	7
2. Giải thuật đệ quy	8
CHƯƠNG 4. MẢNG VÀ DANH SÁCH	10
1. Mảng	10
1.1. Khái niệm mảng	10
1.2. Giải thuật của mảng	11
2. Danh sách	14
2.1. Khái niệm danh sách	14
2.3. Hàng đợi (Queue)	15
2.4. Giải thuật của danh sách	15
CHƯƠNG 5. DANH SÁCH MÓC NỐI (LINKLIST)	23
1. Khái niệm danh sách móc nối	23
1.1. Ưu điểm và nhược điểm của danh sách móc nối	23
2. Các loại danh sách móc nối	23
2.1. Danh sách móc nối đơn	23
2.2. Danh sách móc nối kép	24
3. Giải thuật danh sách móc nối	25
3.1. Giải thuật của danh sách móc nối đơn:	25
3.2. Giải thuật của danh sách móc nối kép:	30
CHƯƠNG 6. CÂY NHỊ PHÂN	36
1. Khái niệm cây	36
2. Các loại cây	36
2.1. Cây nhị phân	36
2.2. Cây nhị phân tìm kiếm	36
3. Giải thuật cây nhị phân	36
3.1. Giải thuật của cây nhị phân	36
3.2. Giải thuật của cây nhị phân tìm kiếm	38
CHƯƠNG 7. ĐỒ THỊ	43
1. Khái niệm đồ thị	43

2. Giải thuật đồ thị	45
CHƯƠNG 8. SẮP XẾP	53
1. Khái Niệm sắp xếp.....	53
2. Giải thuật sắp xếp.....	54
CHƯƠNG 9. TÌM KIẾM	59
1. Khái niệm tìm kiếm.....	59
2. Giải thuật tìm kiếm.....	62
KẾT LUẬN	66
TÀI LIỆU THAM KHẢO	67

LỜI NÓI ĐẦU

Cấu trúc dữ liệu và giải thuật là một trong những nền tảng cốt lõi của khoa học máy tính, đóng vai trò quan trọng trong việc giải quyết các bài toán phức tạp và tối ưu hóa hiệu suất của các chương trình máy tính. Việc hiểu rõ và vận dụng các cấu trúc dữ liệu và giải thuật một cách hiệu quả không chỉ giúp tối ưu hóa tài nguyên, mà còn là tiền đề cho việc phát triển các hệ thống phần mềm lớn và đáp ứng các yêu cầu ngày càng cao trong thực tiễn.

Môn học "Cấu trúc dữ liệu và giải thuật" trang bị cho sinh viên những kiến thức cơ bản và nâng cao về các loại cấu trúc dữ liệu như mảng, danh sách liên kết, cây, đồ thị, và các Giải thuật cơ bản như tìm kiếm, sắp xếp, đệ quy, mảng, danh sách, chia để trị, quy hoạch động, và tham lam. Bên cạnh đó, môn học cũng giúp sinh viên rèn luyện tư duy Giải thuật, khả năng phân tích và thiết kế giải thuật để giải quyết các bài toán một cách hiệu quả nhất.

CHƯƠNG 3. ĐỆ QUY

1. Khái niệm đệ quy

Đệ quy là một phương pháp giải bài toán bằng cách chia bài toán lớn thành các bài toán con nhỏ hơn có cùng bản chất và tiếp tục chia nhỏ cho đến khi đạt đến một trường hợp cơ bản đơn giản có thể giải được ngay mà không cần chia nhỏ thêm nữa.

1.2. Khái niệm đệ quy trong lập trình

Đệ quy là khi một hàm tự gọi lại chính nó để giải quyết một vấn đề. Mỗi lần gọi hàm, chương trình tạo ra một phiên bản mới của hàm đó với các tham số đầu vào thay đổi (theo logic của bài toán), tạo thành một chuỗi các lời gọi hàm lồng vào nhau. Đệ quy thường được sử dụng để giải các bài toán lặp lại hoặc khi bài toán có thể chia thành nhiều bài toán con giống nhau.

1.3. Cấu trúc của đệ quy

Điều kiện cơ sở (Base case): Đây là điểm dừng của đệ quy, nơi hàm không tự gọi lại nữa. Điều kiện cơ sở rất quan trọng vì nếu không có, hàm sẽ lặp mãi mãi và dẫn đến lỗi tràn bộ nhớ (stack overflow).

Gọi đệ quy (Recursive call): Đây là nơi hàm tự gọi lại chính nó với đầu vào đơn giản hơn hoặc nhỏ hơn để tiến dần về điều kiện cơ sở.

1.4. Lợi ích của đệ quy

Đệ quy giúp các chương trình giải quyết bài toán phức tạp một cách ngắn gọn và dễ hiểu hơn, đặc biệt là khi cần xử lý các cấu trúc dữ liệu phân cấp hoặc bài toán chia để trị. Một số bài toán đệ quy phổ biến là:

Duyệt cây: Cây là một cấu trúc dữ liệu phân cấp, và duyệt cây (như duyệt theo thứ tự trước, thứ tự giữa, hay thứ tự sau) thường được thực hiện dễ dàng bằng đệ quy.

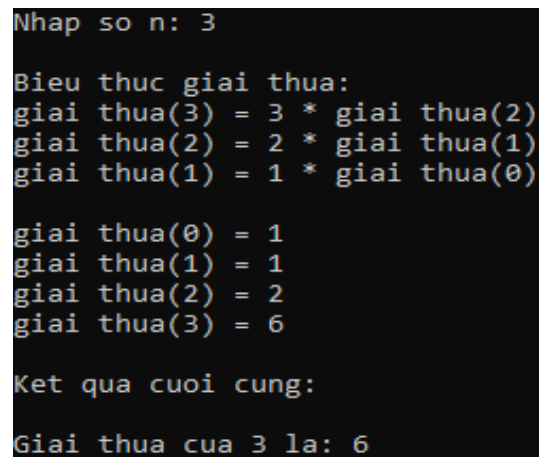
Giải bài toán Fibonacci: Hàm Fibonacci là một bài toán đệ quy tiêu biểu khi số Fibonacci thứ n bằng tổng hai số Fibonacci trước đó.

2. Giải thuật đệ quy

2.1. Tính giai thừa của một số tự nhiên n

```
int tinhGiaiThua(int n) {  
    if (n == 0) return 1;  
    return n * tinhGiaiThua(n - 1);  
}
```

- Bài toán demo:



```
Nhap so n: 3  
  
Bieu thuc giai thua:  
giai thua(3) = 3 * giai thua(2)  
giai thua(2) = 2 * giai thua(1)  
giai thua(1) = 1 * giai thua(0)  
  
giai thua(0) = 1  
giai thua(1) = 1  
giai thua(2) = 2  
giai thua(3) = 6  
  
Ket qua cuoi cung:  
Giai thua cua 3 la: 6
```

Hình 3.1: demo đệ quy

2.2. Tính số Fibonacci bằng giải thuật đệ quy

```
int tinhFibonacci(int n) {  
    if (n <= 1) {  
        return n;  
    }  
    return tinhFibonacci(n - 1) + tinhFibonacci(n - 2);  
}
```


- Kết quả sau khi chạy:

```
Nhap so n de tinh Fibonacci: 4

Tinh Fibonacci:
Fibonacci(2) = Fibonacci(1) + Fibonacci(0) = 1 + 0 = 1
Fibonacci(3) = Fibonacci(2) + Fibonacci(1) = 1 + 1 = 2
Fibonacci(4) = Fibonacci(3) + Fibonacci(2) = 2 + 1 = 3

Ket qua cuoi cung:
So Fibonacci thu 4 la: 3
```

Hình 3.2: Đề mô Fibonacci

CHƯƠNG 4. MẢNG VÀ DANH SÁCH

1. Mảng

1.1. Khái niệm mảng

Mảng là một cấu trúc dữ liệu chứa các phần tử có cùng kiểu dữ liệu, được lưu trữ liên tiếp trong bộ nhớ. Mảng cho phép truy cập các phần tử của nó thông qua chỉ số (index), giúp cho việc truy xuất phần tử diễn ra nhanh chóng.

1.1.1. Đặc điểm của mảng:

- **Đồng nhất kiểu dữ liệu:** Tất cả các phần tử trong mảng đều có cùng kiểu dữ liệu (ví dụ: tất cả là số nguyên, tất cả là chuỗi, v.v.).
- **Chỉ số (index):** Các phần tử trong mảng được đánh số thứ tự, bắt đầu từ 0 đối với hầu hết các ngôn ngữ lập trình. Ví dụ: phần tử đầu tiên có chỉ số 0, phần tử thứ hai có chỉ số 1, v.v.
- **Kích thước cố định:** Trong một số ngôn ngữ lập trình, mảng có kích thước cố định, tức là bạn phải xác định trước số lượng phần tử trong mảng khi khai báo. Tuy nhiên, trong các ngôn ngữ hỗ trợ mảng động, mảng có thể thay đổi kích thước trong quá trình chạy chương trình.
- **Lưu trữ liên tiếp:** Các phần tử trong mảng thường được lưu trữ trong bộ nhớ liên tiếp, giúp truy cập nhanh và dễ dàng.

1.1.2. Các loại mảng:

- **Mảng một chiều (1D array):** Là mảng có một chiều, trong đó các phần tử được tổ chức theo một danh sách.
- **Mảng hai chiều (2D array):** Là mảng có hai chiều, giống như một bảng với các dòng và cột.
- **Mảng đa chiều (Multi-dimensional array):** Là mảng có nhiều hơn hai chiều, ví dụ mảng 3D, 4D, v.v.

1.2. Giải thuật của mảng

Mảng hiện tại:

```
===== MENU =====
Chon giai thuat can thuc hien:
1. Nhap mang
2. Tim kiem tuyen tinh
3. Sap xep noi bong
4. Tim phan tu lon nhat
5. Tim phan tu nho nhat
6. Thoat
=====
Nhap lua chon (1-6): 1

Nhap so phan tu cho mang: 5
Nhap phan tu thu 1: 3
Nhap phan tu thu 2: 5
Nhap phan tu thu 3: 2
Nhap phan tu thu 4: 6
Nhap phan tu thu 5: 7
Mang da duoc nhap thanh cong.

Mang hien tai:
-----
Vi tri | Gia tri
-----
0      | 3
1      | 5
2      | 2
3      | 6
4      | 7
-----
```

Hình 4.1: Mảng ban đầu được nhập

1.2.1. Giải thuật tìm kiếm tuyến tính

```
int timKiemTuyenTinh(int mang[], int n, int x) {
    for (int i = 0; i < n; i++) {
        if (mang[i] == x) {
            return i;
        }
    }
    return -1;
}
```

- Bài toán demo:

```
Nhap gia tri can tim: 4
Khong tim thay phan tu 4

Mang hien tai:
-----
Vi tri | Gia tri
-----
0      | 3
1      | 5
2      | 2
3      | 6
4      | 7
-----
```

Hình 4.2: Demo giải thuật tìm kiếm tuyến tính

1.2.2. Giải thuật sắp xếp bọt (Bubble Sort)

```
void sapXepNoiBong(int mang[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1 - i; j++) {
            if (mang[j] > mang[j + 1]) {
                swap(mang[j], mang[j + 1]);
            }
        }
    }
}
```

- Bài toán demo:

```
Mang truoc khi sap xep:
Mang hien tai:
-----
Vi tri | Gia tri
-----
0      | 3
1      | 5
2      | 2
3      | 6
4      | 7
-----

Mang sau khi sap xep:
Mang hien tai:
-----
Vi tri | Gia tri
-----
0      | 2
1      | 3
2      | 5
3      | 6
4      | 7
-----
```

Hình 4.3: Demo giải thuật sắp xếp bọt (Bubble Sort)

1.2.3. Giải thuật tìm phần tử lớn nhất/nhỏ nhất trong mảng

```
int timMax(int mang[], int n) {  
    int max = mang[0];  
    for (int i = 1; i < n; i++) {  
        if (mang[i] > max) {  
            max = mang[i];  
        }  
    }  
    return max;  
}  
  
int timMin(int mang[], int n) {  
    int min = mang[0];  
    for (int i = 1; i < n; i++) {  
        if (mang[i] < min) {  
            min = mang[i];  
        }  
    }  
    return min;  
}
```

- Bài toán demo:

Phan tu lon nhat trong mang: 7

Mang hien tai:

Vi tri	Gia tri
0	2
1	3
2	5
3	6
4	7

Hình 4.4: Demo Giải thuật tìm phần tử lớn nhất trong mảng

```

Phan tu nho nhat trong mang: 2
Mang hien tai:
-----
Vi tri | Gia tri
-----
  0    |    2
  1    |    3
  2    |    5
  3    |    6
  4    |    7
-----

```

Hình 4.5: Demo Giải thuật tìm phần tử nhỏ nhất trong mảng

2. Danh sách

2.1. Khái niệm danh sách

Lưu trữ danh sách là một vấn đề quan trọng trong việc tối ưu hóa các thao tác như tìm kiếm, thêm, xóa và duyệt qua các phần tử. Các giải thuật và cấu trúc dữ liệu để lưu trữ và thao tác với danh sách có thể được chia thành các loại cơ bản như danh sách mảng, danh sách liên kết, và một số loại danh sách nâng cao hơn. Mỗi loại có ưu và nhược điểm riêng, và việc chọn lựa giữa chúng phụ thuộc vào bài toán cụ thể và yêu cầu hiệu suất.

2.2. Ngăn xếp (Stack)

Định nghĩa:

Là một cấu trúc dữ liệu hoạt động theo nguyên tắc **LIFO** (Last In, First Out), nghĩa là phần tử được thêm vào sau cùng sẽ được lấy ra đầu tiên.

Các thao tác chính:

- **Push(x)**: Thêm một phần tử x vào đỉnh ngăn xếp.
- **Pop()**: Lấy phần tử ở đỉnh ngăn xếp và xóa nó.
- **Peek() hoặc Top()**: Trả về phần tử ở đỉnh ngăn xếp nhưng không xóa nó.
- **isEmpty()**: Kiểm tra ngăn xếp có rỗng hay không.

2.3. Hàng đợi (Queue)

Định nghĩa:

Là một cấu trúc dữ liệu hoạt động theo nguyên tắc **FIFO** (First In, First Out), nghĩa là phần tử được thêm vào trước sẽ được lấy ra trước.

Các thao tác chính:

- **Enqueue(x)**: Thêm một phần tử x vào cuối hàng đợi.
- **Dequeue()**: Lấy phần tử ở đầu hàng đợi và xóa nó.
- **Front()**: Trả về phần tử ở đầu hàng đợi nhưng không xóa nó.
- **isEmpty()**: Kiểm tra hàng đợi có rỗng hay không.

2.4. Giải thuật của danh sách

2.4.1. Giải thuật Ngăn xếp (Stack)

- kích cỡ ngăn xếp (Stack) là 4

```
===== MENU =====
1. Nhap phan tu vao ngan xep
2. Them phan tu vao ngan xep
3. Lay phan tu khoi ngan xep
4. Xem phan tu o dinh
5. Hien thi ngan xep
6. Kiem tra ngan xep co rong hay khong
7. Kiem tra ngan xep co day hay khong
8. Xoa tat ca phan tu trong ngan xep
9. Thoat

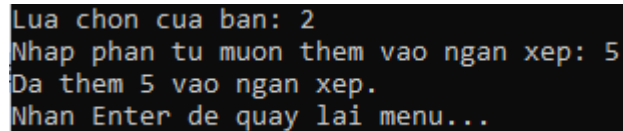
=====
Lua chon cua ban: 1
Nhap so luong phan tu ban muon them: 3
Nhap phan tu thu 1: 2
Da them 2 vao ngan xep.
Nhap phan tu thu 2: 5
Da them 5 vao ngan xep.
Nhap phan tu thu 3: 3
Da them 3 vao ngan xep.
Nhan Enter de quay lai menu...
```

Hình 4.6: Hàng đợi ban đầu được nhập

2.4.1.1. Thêm phần tử vào Ngăn xếp (Stack)

```
void push(int value) {  
    if (isFull()) {  
        cout << "Ngan xep day!" << endl;  
        return;  
    }  
    arr[++top] = value;  
    cout << "Da them " << value << " vao ngan xep." << endl;  
}
```

- bài toán demo



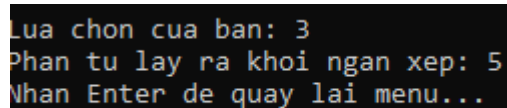
```
Lua chon cua ban: 2  
Nhap phan tu muon them vao ngan xep: 5  
Da them 5 vao ngan xep.  
Nhan Enter de quay lai menu...
```

Hình 4.7: Demo Giải thuật thêm phần tử vào ngăn xếp

2.4.1.2. Lấy phần tử khỏi Ngăn xếp (Stack)

```
int pop() {  
    if (isEmpty()) {  
        cout << "Ngan xep rong!" << endl;  
        return -1;  
    }  
    return arr[top--];  
}
```

- bài toán demo:



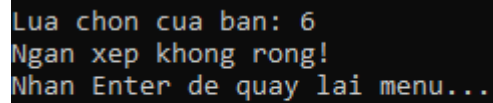
```
Lua chon cua ban: 3  
Phan tu lay ra khoi ngan xep: 5  
Nhan Enter de quay lai menu...
```

Hình 4.8: Demo Giải thuật lấy phần tử ra khỏi ngăn xếp

2.4.1.3. Kiểm tra Ngăn xếp (Stack) có rỗng hay không

```
bool isEmpty() {  
    return top == -1;  
}
```

- bài toán demo



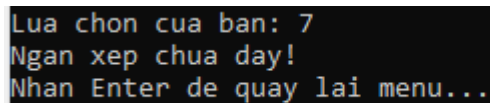
```
Lua chon cua ban: 6  
Ngan xep khong rong!  
Nhan Enter de quay lai menu...
```

Hình 4.9: Demo Giải thuật Kiểm tra Ngăn xếp (Stack) có rỗng hay không

2.4.1.4. Kiểm tra Ngăn xếp (Stack) có đầy hay không

```
bool isFull() {  
    return top == capacity - 1;  
}
```

- bài toán demo



```
Lua chon cua ban: 7  
Ngan xep chua day!  
Nhan Enter de quay lai menu...
```

Hình 4.10: Demo Giải thuật Kiểm tra Ngăn xếp (Stack) có đầy hay không

2.4.1.5. Xóa tất cả phần tử khỏi Ngăn xếp (Stack)

```
void clearAll() {  
    top = -1;  
    cout << "Da xoa tat ca phan tu trong ngan xep!" << endl;  
}
```

- bài toán demo

```
Lua chon cua ban: 8
Da xoa tat ca phan tu trong ngan xep!
Nhan Enter de quay lai menu...
_
```

```
Lua chon cua ban: 5
Ngan xep rong!
Nhan Enter de quay lai menu...
```

Hình 4.11: Demo giải thuật xoá tất cả phần tử khỏi Ngăn xếp (Stack)

2.4.2. giải thuật hàng đợi (Queue)

- Hàng đợi ban đầu

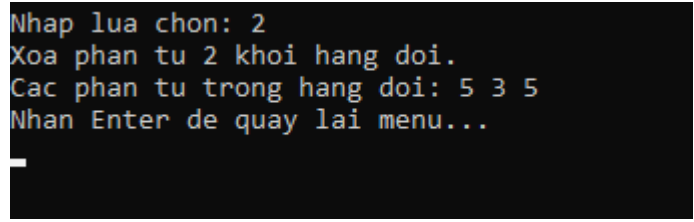
```
===== MENU =====
1. Nhap phan tu vao hang doi
2. Xoa phan tu khoi hang doi
3. Xem phan tu dau hang doi
4. Tinh kích thước hang doi
5. Xoa tat ca phan tu trong hang doi
6. Hien thi hang doi tu cuoi toi dau
7. Them phan tu vao vi tri bat ky
8. Thoat
=====
Nhap lua chon: 1
Nhap so luong phan tu can them: 4
Nhap phan tu thu 1: 2
Them phan tu 2 vao hang doi.
Cac phan tu trong hang doi: 2
Nhap phan tu thu 2: 5
Them phan tu 5 vao hang doi.
Cac phan tu trong hang doi: 2 5
Nhap phan tu thu 3: 3
Them phan tu 3 vao hang doi.
Cac phan tu trong hang doi: 2 5 3
Nhap phan tu thu 4: 5
Them phan tu 5 vao hang doi.
Cac phan tu trong hang doi: 2 5 3 5
Nhan Enter de quay lai menu...
```

Hình 4.12: Hàng đợi ban đầu được nhập

2.4.2.1. Xoá phần tử khỏi hàng đợi

```
int dequeue() {  
    if (isEmpty()) {  
        cout << "Hàng đợi trống! Không thể xóa." << endl;  
        return -1;  
    }  
    int dequeuedValue = arr[front];  
    front = (front + 1) % capacity;  
    size--;  
    cout << "Xóa phần tử " << dequeuedValue << " khỏi hàng đợi." << endl;  
    return dequeuedValue;  
}
```

- Bài toán demo



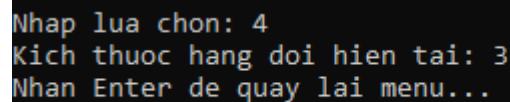
```
Nhap lua chon: 2  
Xóa phần tử 2 khỏi hàng đợi.  
Các phần tử trong hàng đợi: 5 3 5  
Nhấn Enter để quay lại menu...  
_
```

Hình 4.13: Demo giải thuật Xoá phần tử khỏi hàng đợi

2.4.2.2. Tính kích thước hàng đợi

```
int getSize() {  
    return size;  
}
```

- Bài toán demo



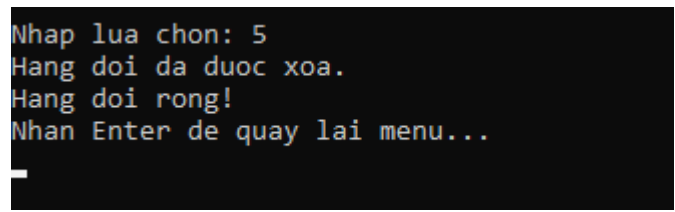
```
Nhap lua chon: 4  
Kích thước hàng đợi hiện tại: 3  
Nhấn Enter để quay lại menu...  
_
```

Hình 4.14: Demo giải thuật tính kích thước hàng đợi

2.4.2.3. Xoá tất cả phần tử ra khỏi hàng đợi

```
void clearQueue() {  
    front = rear = -1;  
    size = 0;  
    cout << "Hang doi da duoc xoa." << endl;  
}
```

- Bài toán demo



```
Nhap lua chon: 5  
Hang doi da duoc xoa.  
Hang doi rong!  
Nhan Enter de quay lai menu...
```

Hình 4.15: Demo giải thuật Xoá tất cả phần tử khỏi hàng đợi

2.4.2.4. Hiển thị hàng đợi từ cuối tới đầu

```
void displayReverse() {  
    if (isEmpty()) {  
        cout << "Hang doi rong!" << endl;  
        return;  
    }  
    cout << "Cac phan tu trong hang doi (tu cuoi toi dau): ";  
    for (int i = size - 1; i >= 0; i--) {  
        cout << arr[(front + i) % capacity] << " ";  
    }  
    cout << endl;
```

```
}
```

- Bài toán demo

```
Nhap lua chon: 6
Cac phan tu trong hang doi (tu cuoi toi dau): 5 7 3 5
Nhan Enter de quay lai menu...
```

Hình 4.16: Demo giải thuật đảo ngược hàng đợi

2.4.2.5. Thêm phần tử vào vị trí bất kì

```
void insertAtPosition(int value, int position) {
    if (position < 0 || position >= capacity || size == capacity) {
        cout << "Vi tri khong hop le!" << endl;
        return;
    }
    if (position == 0) {
        enqueue(value);
        return;
    }

    int *newArr = new int[capacity];
    int newSize = 0;
    for (int i = 0; i < position; i++) {
        newArr[i] = arr[(front + i) % capacity];
        newSize++;
    }
    newArr[position] = value;
    for (int i = position; i < size; i++) {
```

```

        newArr[i + 1] = arr[(front + i) % capacity];
        newSize++;
    }
    delete[] arr;
    arr = newArr;
    size++;
    front = 0;
    rear = size - 1;

    cout << "Them phan tu " << value << " vao vi tri " << position << " trong hang
    doi." << endl;
}

```

- Bài toán demo

```

Nhap lua chon: 7
Nhap phan tu can them: 7
Nhap vi tri can them: 2
Them phan tu 7 vao vi tri 2 trong hang doi.
Cac phan tu trong hang doi: 5 3 7 5
Nhan Enter de quay lai menu...
_

```

Hình 4.17: Demo giải thuật thêm phần vào vị trí bất kì

CHƯƠNG 5. DANH SÁCH MÓC NỘI (LINKLIST)

1. Khái niệm danh sách móc nối

Danh sách móc nối (link list) là một cấu trúc dữ liệu bao gồm một nhóm các nút tạo thành một chuỗi. Thông thường mỗi node cùng chứa dữ liệu ở nút đó và tham chiếu đến nút kế tiếp trong chuỗi

Danh sách móc nối là một trong những cấu trúc dữ liệu đơn giản và phổ biến nhất

1.1. Ưu điểm và nhược điểm của danh sách móc nối

Ưu điểm:

- Cung cấp giải pháp để chứa cấu trúc dữ liệu tuyến tính
- Dễ dàng thêm hoặc xóa các phần tử trong danh sách
- Cấp phát bộ nhớ động

Nhược điểm:

- Danh sách liên kết đơn giản không cho phép truy cập ngẫu nhiên dữ liệu

1. Các loại danh sách móc nối

2.1. Danh sách móc nối đơn

- Danh sách móc nối đơn (Singly Linked List) là một cấu trúc dữ liệu trong đó các phần tử (hay còn gọi là node) được liên kết với nhau theo dạng chuỗi. Mỗi node chứa dữ liệu và một con trỏ (hoặc tham chiếu) chỉ đến phần tử tiếp theo trong danh sách.

Các thao tác cơ bản:

- **Thêm phần tử vào đầu danh sách:** Được thực hiện bằng cách thay đổi con trỏ của phần tử đầu tiên sao cho nó trỏ đến phần tử mới.
- **Thêm phần tử vào cuối danh sách:** Thực hiện duyệt qua các phần tử của danh sách và thay đổi con trỏ của phần tử cuối cùng để nó trỏ đến phần tử mới.
- **Xóa phần tử:** Duyệt qua danh sách để tìm phần tử cần xóa và thay đổi con trỏ của phần tử trước đó sao cho nó không trỏ đến phần tử bị xóa nữa.

- **Duyệt danh sách:** Duyệt qua danh sách từ đầu đến cuối bằng cách sử dụng con trỏ Next

2.2. Danh sách móc nối kép

Danh sách móc nối kép (Doubly Linked List) là một cấu trúc dữ liệu gồm một chuỗi các nút (node), trong đó mỗi nút chứa ba phần: dữ liệu (giá trị), con trỏ đến nút trước và con trỏ đến nút sau. Nhờ có hai con trỏ này, mỗi nút trong danh sách biết được vị trí của nút đứng trước và nút đứng sau nó, cho phép việc di chuyển và thao tác trên danh sách linh hoạt hơn so với danh sách móc nối đơn. Nút đầu tiên trong danh sách gọi là "head" có con trỏ đến nút trước là nullptr (vì không có nút nào đứng trước nó), và nút cuối cùng gọi là "tail" có con trỏ đến nút sau là nullptr (vì không có nút nào đứng sau nó)

Ưu điểm và nhược điểm của danh sách móc nối kép

Ưu điểm của danh sách móc nối kép:

- **Điều hướng linh hoạt:** Cho phép duyệt danh sách theo cả hai chiều, từ đầu đến cuối và ngược lại từ cuối về đầu.
- **Chèn và xóa nhanh chóng:** Thêm hoặc xóa một nút tại bất kỳ vị trí nào trong danh sách chỉ yêu cầu điều chỉnh một số con trỏ lân cận, không cần duyệt lại toàn bộ danh sách như trong danh sách móc nối đơn.
- **Thực hiện các thao tác ở cuối hiệu quả:** Các thao tác thêm hoặc xóa ở cuối danh sách có thể thực hiện nhanh hơn mà không cần duyệt từ đầu.

Nhược điểm của danh sách móc nối kép:

- **Tốn bộ nhớ hơn:** So với danh sách móc nối đơn, mỗi nút trong danh sách móc nối kép cần thêm một con trỏ, làm tăng yêu cầu về bộ nhớ.
- **Cấu trúc phức tạp hơn:** Việc quản lý hai con trỏ next và prev ở mỗi nút khiến danh sách phức tạp hơn; khi thêm hoặc xóa các nút, cần cẩn thận điều chỉnh các con trỏ để tránh lỗi và không làm mất kết nối trong danh sách.

3. Giải thuật danh sách móc nối

3.1. Giải thuật của danh sách móc nối đơn:

- Danh sách ban đầu

```
=====
          MENU CHƯƠNG TRÌNH
=====
1. Nhập danh sách
2. Thêm phần tử
3. Xóa phần tử
4. Tìm phần tử
5. Đảo ngược danh sách
6. Tính tổng các phần tử
7. Hợp nhất hai danh sách
8. Xóa phần tử cuối
0. Thoát
=====
Nhập lựa chọn: 1
Nhập số lượng phần tử trong danh sách: 4
Nhập phần tử 1: 2
Nhập phần tử 2: 3
Nhập phần tử 3: 5
Nhập phần tử 4: 4
Danh sách: 2 -> 3 -> 5 -> 4
Nhấn Enter để quay lại menu...
```

Hình 5.1: Danh sách móc nối đơn được nhập

3.1.1. Thêm phần tử

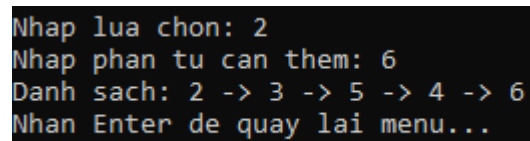
```
void themPhanTu(int giaTri) {
    Node* newNode = new Node();
    newNode->data = giaTri;
    newNode->next = nullptr;
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
```

```

while (temp->next != nullptr) {
    temp = temp->next;
}
temp->next = newNode;
}
}

```

- bài toán demo



```

Nhap lua chon: 2
Nhap phan tu can them: 6
Danh sach: 2 -> 3 -> 5 -> 4 -> 6
Nhap Enter de quay lai menu...

```

Hình 5.2: Demo giải thuật thêm phần tử vào danh sách móc nối đơn

3.1.2. Xoá phần tử

```

void xoaPhanTu(int giaTri) {
    if (head == nullptr) {
        cout << "Danh sach rong! Khong the xoa." << endl;
        return;
    }
    if (head->data == giaTri) {
        Node* temp = head;
        head = head->next;
        delete temp;
        cout << "Phan tu " << giaTri << " da duoc xoa." << endl;
        return;
    }
    Node* current = head;

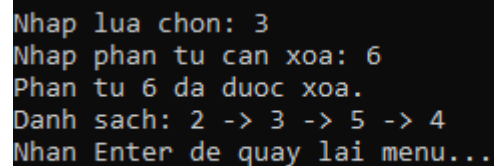
```

```

Node* previous = nullptr;
while (current != nullptr && current->data != giaTri) {
    previous = current;
    current = current->next;
}
if (current == nullptr) {
    cout << "Khong tim thay phan tu " << giaTri << " de xoa." << endl;
    return;
}
previous->next = current->next;
delete current;
cout << "Phan tu " << giaTri << " da duoc xoa." << endl;
}

```

- bài toán demo



```

Nhap lua chon: 3
Nhap phan tu can xoa: 6
Phan tu 6 da duoc xoa.
Danh sach: 2 -> 3 -> 5 -> 4
Nhan Enter de quay lai menu...

```

Hình 5.3: Demo giải thuật xóa phần tử khỏi danh sách móc nối đơn

3.1.3. Tìm phần tử

```

void timKiem(int giaTri) {
    Node* current = head;
    while (current != nullptr) {
        if (current->data == giaTri) {

```

```

        cout << "Phan tu " << giaTri << " duoc tim thay trong danh sach." <<
endl;

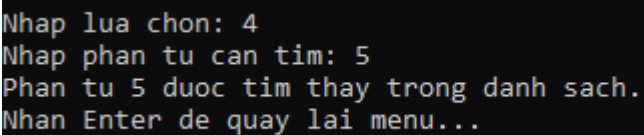
        return;
    }

    current = current->next;
}

cout << "Khong tim thay phan tu " << giaTri << " trong danh sach." <<
endl;
}

```

- bài toán demo



```

Nhap lua chon: 4
Nhap phan tu can tim: 5
Phan tu 5 duoc tim thay trong danh sach.
Nhan Enter de quay lai menu...

```

Hình 5.4: Demo giải thuật tìm phần tử trong danh sách móc nối đơn

3.1.4. Hợp nhất hai danh sách

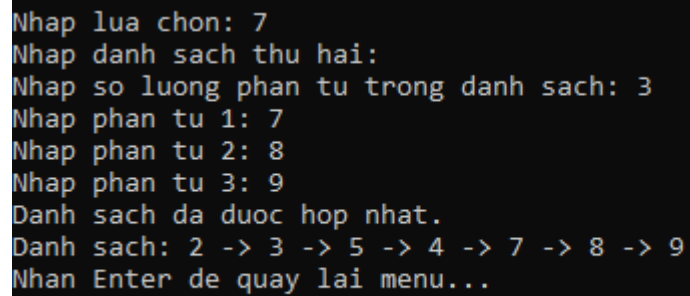
```

void hopNhatDanhSach(LinkedList& otherList) {
    if (otherList.head == nullptr) {
        cout << "Danh sach thu hai rong." << endl;
        return;
    }
    if (head == nullptr) {
        head = otherList.head;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {

```

```
        temp = temp->next;
    }
    temp->next = otherList.head;
}
otherList.head = nullptr;
cout << "Danh sach da duoc hop nhat." << endl;
}
```

- bài toán demo



```
Nhap lua chon: 7
Nhap danh sach thu hai:
Nhap so luong phan tu trong danh sach: 3
Nhap phan tu 1: 7
Nhap phan tu 2: 8
Nhap phan tu 3: 9
Danh sach da duoc hop nhat.
Danh sach: 2 -> 3 -> 5 -> 4 -> 7 -> 8 -> 9
Nhan Enter de quay lai menu...
```

Hình 5.5: Demo giải thuật hợp nhất hai danh sách móc nối đơn

3.2. Giải thuật của danh sách móc nối kép:

- Danh sách ban đầu

```
=====
MENU - DANH SACH LIEN KET KEP
=====
1. Nhap danh sach
2. Them phan tu vao cuoi danh sach
3. Xoa phan tu o dau danh sach
4. Xoa phan tu o cuoi danh sach
5. Xoa phan tu theo gia tri
6. Tim kiem phan tu
0. Thoat chuong trinh
=====
Nhap lua chon cua ban: 1

Nhap so luong phan tu trong danh sach: 5

Nhap cac phan tu:
Gia tri 1: 1
Gia tri 2: 2
Gia tri 3: 3
Gia tri 4: 4
Gia tri 5: 5

Danh sach hien tai: 1 -> 2 -> 3 -> 4 -> 5

Nhan Enter de quay lai menu..._
```

Hình 5.6: Danh sách ban đầu của danh sách móc nối kép

3.2.1. Thêm phần tử vào cuối danh sách

```
void themVaoCuoi(int giaTri) {
    Node* newNode = new Node{giaTri, nullptr, tail};
    if (tail == nullptr) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        tail = newNode;
    }
}
```

```
}
```

- Bài toán demo

```
Nhap lua chon cua ban: 2
Nhap gia tri can them vao cuoi: 6
Danh sach hien tai: 1 -> 2 -> 3 -> 4 -> 5 -> 6
Nhan Enter de quay lai menu...
```

Hình 5.7: Demo giải thuật thêm phần tử vào danh sách móc nối kép

3.2.2. Xóa phần tử ở đầu và cuối danh sách

```
void xoaDau() {
    if (head == nullptr) {
        cout << "\n=> Danh sach rong.\n";
        return;
    }
    Node* temp = head;
    head = head->next;
    if (head != nullptr) {
        head->prev = nullptr;
    } else {
        tail = nullptr;
    }
    delete temp;
    hienThi();
}

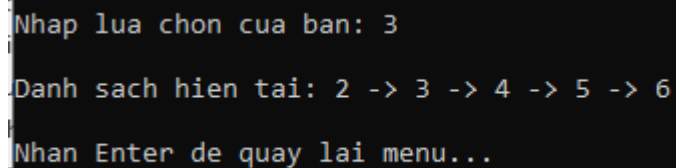
void xoaCuoi() {
```

```

    if (tail == nullptr) {
        cout << "\n=> Danh sach rong.\n";
        return;
    }
    Node* temp = tail;
    tail = tail->prev;
    if (tail != nullptr) {
        tail->next = nullptr;
    } else {
        head = nullptr;
    }
    delete temp;
    hienThi();
}

```

- Kết quả xoá phần tử đầu



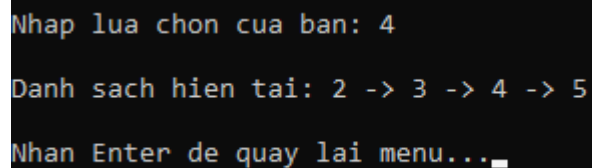
```

Nhap lua chon cua ban: 3
Danh sach hien tai: 2 -> 3 -> 4 -> 5 -> 6
Nhan Enter de quay lai menu...

```

Hình 5.8: Demo giải thuật xoá phần tử đầu khỏi danh sách móc nối kép

- kết quả xoá phần tử cuối



```

Nhap lua chon cua ban: 4
Danh sach hien tai: 2 -> 3 -> 4 -> 5
Nhan Enter de quay lai menu...

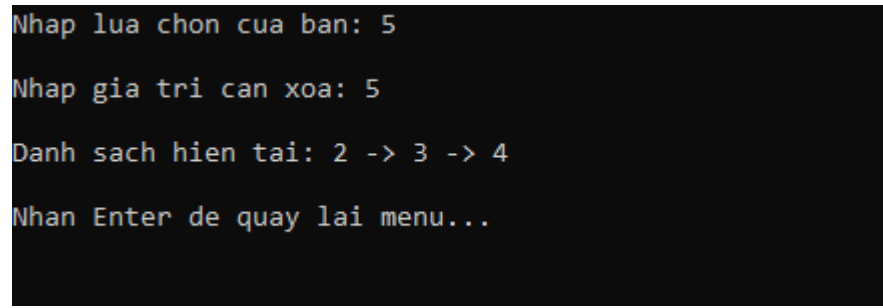
```


Hình 5.9: Demo giải thuật xoá phần tử cuối khỏi danh sách móc nối kép

3.2.3. Xoá phần tử theo giá trị

```
void xoaGiaTri(int giaTri) {
    Node* current = head;
    while (current != nullptr && current->data != giaTri) {
        current = current->next;
    }
    if (current == nullptr) {
        cout << "\n=> Khong tim thay gia tri " << giaTri << " trong danh
sach.\n";
        return;
    }
    if (current == head) {
        xoaDau();
    } else if (current == tail) {
        xoaCuoi();
    } else {
        current->prev->next = current->next;
        current->next->prev = current->prev;
        delete current;
        cout << "\n=> Da xoa gia tri " << giaTri << " trong danh sach.\n";
        hienThi();
    }
}
```

- Bài toán demo



```
Nhap lua chon cua ban: 5
Nhap gia tri can xoa: 5
Danh sach hien tai: 2 -> 3 -> 4
Nhan Enter de quay lai menu...
```

Hình 5.10: Demo giải thuật xóa phần tử theo giá trị khỏi danh sách móc nối kép

3.2.4. Tìm kiếm phần tử

```
void timKiem(int giaTri) {
    Node* current = head;
    int index = 0;
    while (current != nullptr) {
        if (current->data == giaTri) {
            cout << "\n=> Gia tri " << giaTri << " duoc tim thay o vi tri " <<
index << ".\n";
            return;
        }
        current = current->next;
        index++;
    }
    cout << "\n=> Gia tri " << giaTri << " khong co trong danh sach.\n";
}
```

- Bài toán demo

```
Nhap lua chon cua ban: 6
Nhap gia tri can tim: 2
=> Gia tri 2 duoc tim thay o vi tri 0.
Nhan Enter de quay lai menu...
```

Hình 5.11: Demo giải thuật tìm kiếm phân tử trong danh sách móc nối kép

CHƯƠNG 6. CÂY NHỊ PHÂN

1. Khái niệm cây

Trong cấu trúc dữ liệu, cây là một cấu trúc dữ liệu phi tuyến tính, trong đó các phần tử được tổ chức theo một cấu trúc phân cấp. Mỗi phần tử trong cây được gọi là một nút (node), và các nút được liên kết với nhau bằng các cạnh (edge). Cây có một nút đặc biệt gọi là gốc (root), từ đó có thể duyệt và truy cập tất cả các nút còn lại trong cây.

Cây thường được dùng để biểu diễn mối quan hệ phân cấp trong các hệ thống như cấu trúc thư mục trong hệ điều hành, hệ thống phân loại trong cơ sở dữ liệu, hay các Giải thuật tìm kiếm.

2. Các loại cây

2.1. Cây nhị phân

Mỗi một nút trên cây chỉ có tối đa là hai con, cây con trái và cây con phải. Cây nhị phân là một trong những cấu trúc dữ liệu cơ bản và được sử dụng rộng rãi trong các Giải thuật và ứng dụng máy tính

2.2. Cây nhị phân tìm kiếm

Cây nhị phân tìm kiếm là một dạng cây nhị phân đặc biệt, tất cả các nút thuộc cây con trái đều có giá trị nhỏ hơn gốc và tất cả các nút thuộc cây con phải đều có giá trị khóa lớn hơn giá trị nút.

3. Giải thuật cây nhị phân

3.1. Giải thuật của cây nhị phân

3.1.1. Duyệt cây theo thứ tự trước

```
void preOrder(Node* root) {  
    if (root == nullptr) return;  
    cout << root->data << " ";  
    preOrder(root->left);  
    preOrder(root->right);  
}
```

```
}
```

3.1.2. Duyệt cây theo thứ tự giữa

```
void inOrder(Node* root) {  
    if (root == nullptr) return;  
    inOrder(root->left);  
    cout << root->data << " ";  
    inOrder(root->right);  
}
```

3.1.3. Duyệt cây theo thứ tự sau

```
void postOrder(Node* root) {  
    if (root == nullptr) return;  
    postOrder(root->left);  
    postOrder(root->right);  
    cout << root->data << " ";  
}
```

Bài toán demo 3 dạng duyệt cây

```
Nhap so luong phan tu trong cay: 5  
Nhap cac gia tri cho cay (theo thu tu bat ky):  
Nhap gia tri 1: 6  
Nhap gia tri 2: 7  
Nhap gia tri 3: 4  
Nhap gia tri 4: 3  
Nhap gia tri 5: 2  
  
      7  
     /  
    6  
   /  \  
  4    3  
 /  \  /  \  
2    3 4    7  
  
Duyet truoc (Pre-order): 6 4 3 2 7  
Duyet giua (In-order): 2 3 4 6 7  
Duyet sau (Post-order): 2 3 4 7 6
```

Hình 6.1: Demo ba giải thuật duyệt của cây nhị phân

3.2. Giải thuật của cây nhị phân tìm kiếm

- Cây ban đầu

```
===== MENU CAY NHI PHAN TINH KIEM =====
1. Nhap so luong phan tu trong cay
2. Them phan tu vao cay
3. Tim phan tu trong cay
4. Xoa phan tu trong cay
5. Thoat
=====
Nhap lua chon cua ban: 1
Nhap so luong phan tu can them vao cay: 5
Nhap cac gia tri cho cay (theo thu tu bat ky):
Nhap gia tri 1: 5
Nhap gia tri 2: 9
Nhap gia tri 3: 8
Nhap gia tri 4: 3
Nhap gia tri 5: 2
Da them 5 phan tu vao cay.

Cay nhi phan hien tai:

      9
     /
    8
   /
  5
 /
3
 /
2
```

Hình 6.2: Nhập phần tử cho cây nhị phân

3.2.1. Thêm phần tử vào cây nhị phân

```
Node* themNode(Node* root, int val) {
    if (root == nullptr) {
        return new Node(val);
    }
    if (val < root->data) {
        root->left = themNode(root->left, val);
    } else {
```

```

    root->right = themNode(root->right, val);
}
return root;
}

```

- Bài toán demo

```

Nhap lua chon cua ban: 2
Nhap gia tri can them: 2
Da them phan tu 2 vao cay.

Cay nhi phan hien tai:
      9
     / \
    5   8
   / \
  3   2
 / \
2   2

```

Hình 6.3: Demo giải thuật thêm phần tử vào cây nhị phân

3.2.2. Tìm phần tử trong cây nhị phân

```

Node* timKiem(Node* root, int val) {
    if (root == nullptr || root->data == val) {
        return root;
    }
    if (val < root->data) {
        return timKiem(root->left, val);
    } else {
        return timKiem(root->right, val);
    }
}

```

}

- Bài toán demo

```
Nhap lua chon cua ban: 3
Nhap gia tri can tim: 3
Phan tu 3 tim thay trong cay.
```

```
Cay nhi phan hien tai:
```

```
      9
     / \
    5   8
   / \
  3   2
 / \
2   
```

Hình 6.4: Demo giải thuật tìm phần tử vào cây nhị phân

3.2.3. Xóa phần tử trong cây

```
Node* xoaNode(Node* root, int val) {
    if (root == nullptr) return root;
    if (val < root->data) {
        root->left = xoaNode(root->left, val);
    } else if (val > root->data) {
        root->right = xoaNode(root->right, val);
    } else {
        if (root->left == nullptr) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (root->right == nullptr) {
            Node* temp = root->left;
            delete root;
            return temp;
        }
    }
}
```

```

        delete root;
        return temp;
    } else {
        Node* temp = timMin(root->right);
        root->data = temp->data;
        root->right = xoaNode(root->right, temp->data);
    }
}
return root;
}

```

- Bài toán demo

```

Nhap lua chon cua ban: 4
Nhap gia tri can xoa: 2
Da xoa phan tu 2 trong cay.

```

Cay nhị phân hiện tại:

```

      9
     /
    5  8
   / \
  3   2

```

Hình 6.3: Demo giải thuật xoá phần tử vào cây nhị phân

CHƯƠNG 7. ĐỒ THỊ

1. Khái niệm đồ thị

Đồ thị là một cấu trúc dữ liệu trong khoa học máy tính và toán học, được sử dụng để biểu diễn các mối quan hệ hoặc kết nối giữa các đối tượng. Đồ thị bao gồm các đỉnh (vertices hoặc nodes) và cạnh (edges) nối các đỉnh lại với nhau. Các đồ thị thường được dùng để giải quyết nhiều bài toán trong thực tế, như tìm đường đi ngắn nhất, lập lịch công việc, mạng máy tính, và nhiều ứng dụng khác.

1.1. Cấu trúc của Đồ thị

- Đỉnh (Vertex): Là một điểm trong đồ thị. Mỗi đỉnh thường được ký hiệu bằng một chữ cái hoặc số (ví dụ: A, B, C, ...).
- Cạnh (Edge): Là một liên kết giữa hai đỉnh. Cạnh có thể là vô hướng (không có chiều) hoặc có hướng (chỉ có một chiều từ đỉnh này đến đỉnh kia).

1.2. Phân loại đồ thị

- Đồ thị vô hướng (Undirected Graph): Là đồ thị mà các cạnh không có hướng. Nếu có cạnh giữa hai đỉnh A và B, ta có thể di chuyển từ A đến B và ngược lại.
- Đồ thị có hướng (Directed Graph hoặc Digraph): Là đồ thị mà các cạnh có hướng. Nếu có cạnh từ A đến B, ta chỉ có thể di chuyển từ A đến B mà không thể ngược lại (trừ khi có thêm cạnh từ B đến A).
- Đồ thị trọng số (Weighted Graph): Là đồ thị mà mỗi cạnh đều được gán một giá trị (trọng số). Trọng số này có thể biểu thị khoảng cách, chi phí, hoặc bất kỳ giá trị nào liên quan đến cạnh đó.
- Đồ thị không trọng số (Unweighted Graph): Là đồ thị mà các cạnh không có trọng số.

1.3. Cách biểu diễn đồ thị

Có hai cách phổ biến để biểu diễn đồ thị trong máy tính:

- Ma trận kề (Adjacency Matrix): Là một ma trận vuông, trong đó mỗi hàng và cột biểu thị một đỉnh. Phần tử $[i][j]$ có giá trị 1 nếu có cạnh nối từ đỉnh i đến đỉnh j , và 0 nếu không có cạnh.
- Danh sách kề (Adjacency List): Là một mảng hoặc danh sách chứa các danh sách con. Mỗi danh sách con chứa các đỉnh kề với đỉnh tương ứng. Cách biểu diễn này thường tiết kiệm bộ nhớ hơn ma trận kề, đặc biệt với đồ thị thưa (có ít cạnh so với số đỉnh).

1.4. Các Giải thuật liên quan đến đồ thị

Một số Giải thuật phổ biến cho đồ thị bao gồm:

- Giải thuật tìm kiếm theo chiều sâu (DFS - Depth First Search): Duyệt đồ thị bằng cách đi sâu vào các nhánh trước khi quay lại các điểm trước đó.
- Giải thuật tìm kiếm theo chiều rộng (BFS - Breadth First Search): Duyệt đồ thị theo từng lớp, thăm tất cả các đỉnh lân cận của đỉnh hiện tại trước khi chuyển sang lớp tiếp theo.
- Giải thuật Dijkstra: Tìm đường đi ngắn nhất từ một đỉnh nguồn đến các đỉnh khác trong đồ thị có trọng số dương.
- Giải thuật Floyd-Warshall: Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị.
- Giải thuật Bellman-Ford là một Giải thuật tìm đường đi ngắn nhất từ một đỉnh nguồn đến tất cả các đỉnh còn lại trong đồ thị, đặc biệt là có thể xử lý đồ thị có trọng số âm.

2. Giải thuật đồ thị

2.1. Giải thuật tìm kiếm theo chiều sâu (DFS - Depth First Search)

```
void DFS(int u, vector<bool>& visited, vector<vector<int>>& adj) {  
    stack<int> s;  
    s.push(u);  
    visited[u] = true;  
  
    while (!s.empty()) {  
        int v = s.top();  
        s.pop();  
        cout << v << " "; // In ra đỉnh đã thăm  
  
        for (int neighbor : adj[v]) {  
            if (!visited[neighbor]) {  
                s.push(neighbor);  
                visited[neighbor] = true;  
            }  
        }  
    }  
}
```

- Bài toán demo

```
Nhap so luong dinh: 5
Nhap so luong canh: 6
Nhap cac canh (u, v, weight):
Canh 1: 0 1 10
Canh 2: 0 4 20
Canh 3: 1 2 10
Canh 4: 2 3 10
Canh 5: 3 4 10
Canh 6: 4 2 10

===== MENU =====
1. Thuc hien DFS
2. Thuc hien BFS
3. Thuc hien Dijkstra
4. Thuc hien Bellman-Ford
5. Thuc hien Floyd-Warshall
6. Thoat
Nhap lua chon cua ban: 1
Nhap dinh bat dau cho DFS: 3
DFS: 3 4 0 1 2
```

Hình 7.1: Demo giải thuật tìm kiếm theo chiều sâu (DFS - Depth First Search)

2.2. Giải thuật tìm kiếm theo chiều rộng (BFS - Breadth First Search)

```
void BFS(int start, vector<bool>& visited, vector<vector<int>>& adj) {
    queue<int> q;
    visited[start] = true;
    q.push(start);
    while (!q.empty()) {
        int v = q.front();
        cout << v << " "; // In ra dinh da tham
        for (int neighbor : adj[v]) {
            if (!visited[neighbor]) {
                q.push(neighbor);
                visited[neighbor] = true;
            }
        }
    }
}
```

```
}
```

- bài toán demo

```
===== MENU =====  
1. Thuc hien DFS  
2. Thuc hien BFS  
3. Thuc hien Dijkstra  
4. Thuc hien Bellman-Ford  
5. Thuc hien Floyd-Warshall  
6. Thoat  
Nhap lua chon cua ban: 2  
Nhap dinh bat dau cho BFS: 2  
BFS: 2 1 3 4 0
```

Hình 7.2: Demo giải thuật tìm kiếm theo chiều rộng (BFS - Breadth First Search)

2.3. Giải thuật Dijkstra

```
void Dijkstra(int start, vector<vector<pair<int, int>>>& adj, int n) {  
    vector<int> dist(n, INT_MAX);  
    dist[start] = 0;  
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>  
    pq;  
    pq.push({0, start});  
    while (!pq.empty()) {  
        int u = pq.top().second;  
        int d = pq.top().first;  
        pq.pop();  
        if (d > dist[u]) continue;  
        for (auto& neighbor : adj[u]) {  
            int v = neighbor.first;  
            int weight = neighbor.second;
```

```

        if (dist[u] + weight < dist[v]) {
            dist[v] = dist[u] + weight;
            pq.push({dist[v], v});
        }
    }
}

cout << "Khoang cach ngan nhat tu dinh " << start << ":\n";
for (int i = 0; i < n; ++i) {
    if (dist[i] == INT_MAX) {
        cout << "Khong co duong di den dinh " << i << endl;
    } else {
        cout << "Khoang cach den dinh " << i << ": " << dist[i] << endl;
    }
}
}

```

- bài toán demo

```

===== MENU =====
1. Thuc hien DFS
2. Thuc hien BFS
3. Thuc hien Dijkstra
4. Thuc hien Bellman-Ford
5. Thuc hien Floyd-Warshall
6. Thoat
Nhap lua chon cua ban: 3
Nhap dinh bat dau cho Dijkstra: 4
Khoang cach ngan nhat tu dinh 4:
Khoang cach den dinh 0: 20
Khoang cach den dinh 1: 20
Khoang cach den dinh 2: 10
Khoang cach den dinh 3: 10
Khoang cach den dinh 4: 0

```

Hình 7.3: Demo giải thuật Dijkstra

2.4. Giải thuật Floyd-Warshall

```
void FloydWarshall(int n, vector<vector<int>>& dist) {
    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX &&
                    dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
    cout << "Khoang cach ngan nhat giua moi cap dinh:\n";
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (dist[i][j] == INT_MAX) {
                cout << "INF ";
            } else {
                cout << dist[i][j] << " ";
            }
        }
        cout << endl;
    }
}
```

- bài toán demo

```
===== MENU =====
1. Thuc hien DFS
2. Thuc hien BFS
3. Thuc hien Dijkstra
4. Thuc hien Bellman-Ford
5. Thuc hien Floyd-Warshall
6. Thoat
Nhap lua chon cua ban: 5
Nhap cac canh cho Floyd-Warshall:
Canh 1: 0 1 10
Canh 2: 0 4 10
Canh 3: 2 3 10
Canh 4: 3 4 10
Canh 5: 4 2 10
Canh 6: 2 3 10
Khoang cach ngan nhat giua moi cap dinh:
0 10 20 20 10
10 0 30 30 20
20 30 0 10 10
20 30 10 0 10
10 20 10 10 0
```

Hình 7.4: Demo giải thuật Floyd-Warshall

2.5. Giải thuật Bellman-Ford

```
void BellmanFord(int n, vector<pair<int, pair<int, int>>>& edges, int start) {
    vector<int> dist(n, INT_MAX);
    dist[start] = 0;
    for (int i = 0; i < n - 1; ++i) {
        for (auto& edge : edges) {
            int u = edge.first;
            int v = edge.second.first;
            int weight = edge.second.second;
            if (dist[u] != INT_MAX && dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
            }
        }
    }
}
```

```

    }
    for (auto& edge : edges) {
        int u = edge.first;
        int v = edge.second.first;
        int weight = edge.second.second;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v]) {
            cout << "Do thi chua vong lap co trong so am!\n";
            return;
        }
    }
    cout << "Khoang cach ngan nhat tu dinh " << start << ":\n";
    for (int i = 0; i < n; ++i) {
        if (dist[i] == INT_MAX) {
            cout << "Khong co duong di den dinh " << i << endl;
        } else {
            cout << "Khoang cach den dinh " << i << ": " << dist[i] << endl;
        }
    }
}

```

- bài toán demo

```

===== MENU =====
1. Thuc hien DFS
2. Thuc hien BFS
3. Thuc hien Dijkstra
4. Thuc hien Bellman-Ford
5. Thuc hien Floyd-Warshall
6. Thoat
Nhap lua chon cua ban: 4
Nhap dinh bat dau cho Bellman-Ford: 3
Khoang cach ngan nhat tu dinh 3:
Khong co duong di den dinh 0
Khong co duong di den dinh 1
Khoang cach den dinh 2: 20
Khoang cach den dinh 3: 0
Khoang cach den dinh 4: 10

```

CHƯƠNG 8. SẮP XẾP

1. Khái Niệm sắp xếp

Sắp xếp là quá trình tổ chức lại các phần tử trong một dãy dữ liệu (mảng, danh sách, v.v.) theo một thứ tự nhất định, thường là thứ tự tăng dần hoặc giảm dần. Việc sắp xếp giúp dễ dàng thực hiện các thao tác khác như tìm kiếm, so sánh và xử lý dữ liệu.

1.1. Ứng Dụng Của Sắp Xếp

Tìm kiếm nhanh hơn: Khi dữ liệu đã được sắp xếp, các Giải thuật tìm kiếm như tìm kiếm nhị phân có thể hoạt động hiệu quả hơn.

Quản lý dữ liệu: Sắp xếp giúp tổ chức dữ liệu sao cho dễ dàng truy xuất, phân tích hoặc xử lý.

Giải quyết các bài toán tối ưu: Một số bài toán tối ưu có thể được giải quyết nhanh chóng khi dữ liệu đã được sắp xếp.

1.2. Các Đặc Tính Quan Trọng Của Giải thuật Sắp Xếp

Độ phức tạp thời gian: Đo lường số thao tác cần thiết để hoàn thành việc sắp xếp, thường được biểu thị bằng Big-O. Các Giải thuật có thể có độ phức tạp khác nhau trong trường hợp xấu nhất, trung bình và tốt nhất.

Độ phức tạp không gian: Đo lường lượng bộ nhớ cần thiết để thực hiện Giải thuật sắp xếp. Một số Giải thuật sắp xếp yêu cầu bộ nhớ phụ, trong khi những Giải thuật khác có thể thực hiện việc sắp xếp trực tiếp trên mảng ban đầu (sắp xếp in-place).

Stability (Tính ổn định): Giải thuật sắp xếp được gọi là ổn định nếu, khi có hai phần tử có cùng giá trị, thứ tự của chúng trong mảng không thay đổi sau khi sắp xếp.

In-place (Dữ liệu không thay đổi bộ nhớ): Giải thuật sắp xếp in-place chỉ sử dụng một lượng bộ nhớ cố định ngoài mảng dữ liệu đầu vào. Điều này có nghĩa là không cần bộ nhớ phụ quá lớn trong quá trình sắp xếp.

1.3. Các Phương Pháp Sắp Xếp Phổ Biến

- Sắp xếp nổi bọt (Bubble Sort): Lặp qua mảng, so sánh các phần tử liền kề và hoán đổi chúng nếu cần, thực hiện nhiều lần cho đến khi mảng đã được sắp xếp.
- Sắp xếp chọn (Selection Sort): Chọn phần tử nhỏ nhất (hoặc lớn nhất) trong mảng và đưa nó về vị trí đầu tiên. Sau đó, tiếp tục tìm phần tử nhỏ nhất trong phần còn lại và đưa nó vào vị trí thứ hai, v.v.
- Sắp xếp chèn (Insertion Sort): Giống như việc sắp xếp các lá bài, Giải thuật này lấy từng phần tử trong mảng và chèn vào đúng vị trí trong phần đã sắp xếp.
- Sắp xếp hợp nhất (Merge Sort): Sử dụng phương pháp chia để trị. Mảng được chia nhỏ ra, sắp xếp các phần nhỏ và sau đó hợp nhất chúng lại thành mảng đã sắp xếp.
- Sắp xếp nhanh (Quick Sort): Chọn một phần tử làm pivot, sau đó phân chia mảng thành hai phần sao cho phần bên trái chứa các phần tử nhỏ hơn pivot và phần bên phải chứa các phần tử lớn hơn pivot. Tiếp tục đệ quy với các phần này

2. Giải thuật sắp xếp

- **Mảng ban đầu**

```
===== MENU =====
1. Nhap cac phan tu trong mang
2. Sap xep Bubble Sort
3. Sap xep Selection Sort
4. Sap xep Insertion Sort
5. Sap xep Merge Sort
6. Sap xep Quick Sort
7. Thoat
Nhap lua chon cua ban: 1
Nhap so luong phan tu trong mang (phai lon hon 0): 5
Nhap cac phan tu trong mang:
Nhap phan tu thu 1: 2
Nhap phan tu thu 2: 3
Nhap phan tu thu 3: 4
Nhap phan tu thu 4: 5
Nhap phan tu thu 5: 6

Mang da nhap: [ 2 3 4 5 6 ]
```

Hình 8.1: Mảng ban đầu được nhập

2.1 Giải thuật Sắp xếp nổi bọt (Bubble Sort)

```
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        bool swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }
        if (!swapped) break;
    }
}
```

- Bài toán demo

```
Nhap lua chon cua ban: 2  
Mang sau khi duoc sap xep voi Bubble Sort: 2 3 4 5 6
```

Hình 8.2: Demo giải thuật Sắp xếp nổi bọt (Bubble Sort)

2.2. Giải thuật Sắp xếp chọn (Selection Sort)

```
void selectionSort(int arr[], int n) {  
    for (int i = 0; i < n - 1; ++i) {  
        int minIndex = i;  
        for (int j = i + 1; j < n; ++j) {  
            if (arr[j] < arr[minIndex]) {  
                minIndex = j;  
            }  
        }  
        swap(arr[i], arr[minIndex]);  
    }  
}
```

- bài toán demo

```
Nhap lua chon cua ban: 3  
Mang sau khi duoc sap xep voi Selection Sort: 2 3 4 5 6
```

Hình 8.3: Demo giải thuật Sắp xếp chọn (Selection Sort)

2.3. giải thuật Sắp xếp chèn (Insertion Sort)

```
void insertionSort(int arr[], int n) {  
    for (int i = 1; i < n; ++i) {
```

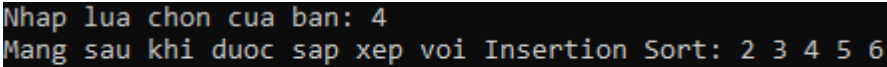


```

int key = arr[i];
int j = i - 1;
while (j >= 0 && arr[j] > key) {
    arr[j + 1] = arr[j];
    --j;
}
arr[j + 1] = key;
}
}

```

- Bài toán demo



```

Nhap lua chon cua ban: 4
Mang sau khi duoc sap xep voi Insertion Sort: 2 3 4 5 6

```

Hình 8.4: Demo giải thuật Sắp xếp chèn (Insertion Sort)

2.4. Giải thuật sắp xếp hợp nhất (Merge Sort)

```

void merge(int arr[], int left, int right) {
    if (left >= right) return;
    int mid = left + (right - left) / 2;
    merge(arr, left, mid);
    merge(arr, mid + 1, right);

    int n1 = mid - left + 1;
    int n2 = right - mid;
    int L[n1], R[n2];
    for (int i = 0; i < n1; ++i) L[i] = arr[left + i];
    for (int i = 0; i < n2; ++i) R[i] = arr[mid + 1 + i];
}

```

```

int i = 0, j = 0, k = left;
while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        ++i;
    } else {
        arr[k] = R[j];
        ++j;
    }
    ++k;
}
while (i < n1) {
    arr[k] = L[i];
    ++i;
    ++k;
}
while (j < n2) {
    arr[k] = R[j];
    ++j;
    ++k;
}
}

void mergeSort(int arr[], int left, int right) {
    merge(arr, left, right);
}

```

- Bài toán demo

```
Nhap lua chon cua ban: 5
Mang sau khi duoc sap xep voi Merge Sort: 2 3 4 5 6
```

Hình 8.5: Demo giải thuật Sắp xếp hợp nhất (Merge Sort)

2.5. Giải thuật Sắp xếp nhanh (Quick Sort)

```
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1); // Sắp xếp phần bên trái
        quickSort(arr, pi + 1, high); // Sắp xếp phần bên phải
    }
}
```

- Bài toán demo

```
Nhap lua chon cua ban: 6
Mang sau khi duoc sap xep voi Quick Sort: 3 3 4 5 6
```

Hình 8.5: Demo giải thuật Sắp xếp nhanh (Quick Sort)

CHƯƠNG 9. TÌM KIẾM

1. Khái niệm tìm kiếm

Tìm kiếm là một trong những vấn đề quan trọng trong cấu trúc dữ liệu và giải thuật. Mục tiêu của tìm kiếm là xác định xem một phần tử có tồn tại trong một cấu trúc dữ liệu hay không, hoặc tìm ra vị trí của phần tử đó. Có rất nhiều Giải thuật tìm kiếm, nhưng các Giải thuật phổ biến nhất bao gồm Tìm kiếm tuần tự và Tìm kiếm nhị phân.

1.1. Tìm kiếm tuần tự (Linear Search)

Tìm kiếm tuần tự là phương pháp đơn giản nhất, trong đó ta duyệt qua tất cả các phần tử trong dãy (hoặc danh sách) để tìm kiếm phần tử cần tìm.

Cách hoạt động:

- Bắt đầu từ phần tử đầu tiên, kiểm tra mỗi phần tử trong danh sách.
- Nếu tìm thấy phần tử cần tìm, trả về vị trí của nó.
- Nếu duyệt hết danh sách mà không tìm thấy, trả về giá trị "không tìm thấy".

1.2. Tìm kiếm nhị phân (Binary Search)

Tìm kiếm nhị phân chỉ áp dụng được trên các dãy dữ liệu đã được sắp xếp. Nó hoạt động bằng cách chia đôi dãy và tìm kiếm trong nửa phần tử còn lại dựa vào so sánh.

Cách hoạt động:

- Bắt đầu với phần tử ở giữa dãy.
- So sánh phần tử cần tìm với phần tử giữa:
 - Nếu tìm thấy, trả về vị trí của nó.
 - Nếu phần tử cần tìm nhỏ hơn phần tử giữa, tìm kiếm tiếp trong nửa bên trái.
 - Nếu phần tử cần tìm lớn hơn phần tử giữa, tìm kiếm tiếp trong nửa bên phải.
- Tiếp tục lặp lại quá trình này cho đến khi tìm thấy phần tử hoặc dãy không còn phần tử nào để kiểm tra.

1.3. Tìm kiếm theo bảng băm (Hash Search)

Bảng băm là một cấu trúc dữ liệu sử dụng hàm băm để ánh xạ phần tử vào một chỉ mục trong mảng. Việc tìm kiếm trong bảng băm có thể thực hiện rất nhanh.

Cách hoạt động:

- Hàm băm sẽ chuyển phần tử cần tìm thành một chỉ mục trong bảng.
- Kiểm tra phần tử ở chỉ mục này.
- Nếu trùng khớp, trả về kết quả tìm thấy.
- Nếu không trùng, tìm trong các phần tử khác tại chỉ mục (trường hợp xung đột).

1.4. Tìm kiếm nhảy (Jump Search)

Tìm kiếm nhảy là một cải tiến của tìm kiếm tuyến tính, trong đó thay vì kiểm tra từng phần tử, ta nhảy qua các phần tử theo bước cố định (ví dụ: bước nhảy là căn bậc hai của độ dài mảng) và kiểm tra mỗi phần tử nhảy.

Cách hoạt động:

- Chia mảng thành các khối có độ dài bằng căn bậc hai của tổng số phần tử.
- Kiểm tra các phần tử tại các vị trí nhảy.
- Nếu tìm thấy, tiếp tục tìm kiếm trong khối tương ứng.

2. Giải thuật tìm kiếm

- mảng ban đầu

```
===== MENU =====
1. Nhập phần tử cho mảng
2. Tìm kiếm tuần tự
3. Tìm kiếm nhị phân
4. Tìm kiếm theo bảng băm
5. Tìm kiếm nhảy
6. Thoát
=====
Chọn phương thức tìm kiếm (1-6): 1
Nhập số lượng phần tử mảng: 4
Nhập các phần tử mảng:
Nhập phần tử thứ 1 : 5
Nhập phần tử thứ 2 : 6
Nhập phần tử thứ 3 : 7
Nhập phần tử thứ 4 : 8
Nhấn Enter để quay lại menu...
```

Hình 9.1: Mảng ban đầu được nhập

2.1. Giải thuật tìm kiếm tuần tự

```
int timKiemTuanTu(int arr[], int n, int target) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
            return i;
        }
    }
    return -1;
}
```

- Bài toán demo

```
Chọn phương thức tìm kiếm (1-6): 2
Nhập phần tử cần tìm: 7

===== Chi tiết tìm kiếm tuần tự =====
Kiểm tra phần tử tại chỉ số [0] với giá trị: 5
Kiểm tra phần tử tại chỉ số [1] với giá trị: 6
Kiểm tra phần tử tại chỉ số [2] với giá trị: 7
=> Tìm thấy phần tử tại chỉ số [2]
Nhấn Enter để quay lại menu...
```

Hình 9.2: Demo giải thuật tìm kiếm tuần tự

2.2. Giải thuật tìm kiếm nhị phân

```
int timKiemNhiPhan(int arr[], int n, int target) {  
    int left = 0, right = n - 1;  
    while (left <= right) {  
        int mid = left + (right - left) / 2;  
        if (arr[mid] == target)  
            return mid;  
        else if (arr[mid] < target)  
            left = mid + 1;  
        else  
            right = mid - 1;  
    }  
    return -1;  
}
```

- Bài toán demo

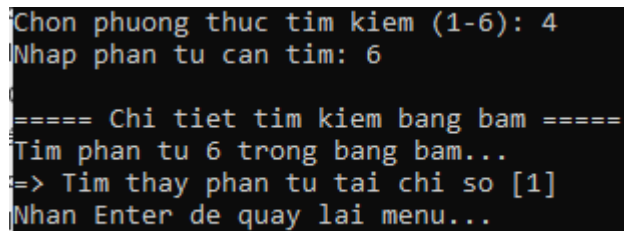
```
Chon phuong thuc tim kiem (1-6): 3  
Nhap phan tu can tim: 8  
  
===== Chi tiet tim kiem nhi phan =====  
Kiem tra mid = 1 (gia tri: 6)  
Phan tu can tim lon hon. Di chuyen left sang [2]  
Kiem tra mid = 2 (gia tri: 7)  
Phan tu can tim lon hon. Di chuyen left sang [3]  
Kiem tra mid = 3 (gia tri: 8)  
=> Tim thay phan tu tai chi so [3]  
Nhan Enter de quay lai menu...
```

Hình 9.3: Demo giải thuật tìm kiếm nhị phân

2.3. Giải thuật tìm kiếm bảng hàm băm

```
int timKiemBangHam(unordered_map<int, int>& hashTable, int target) {  
    if (hashTable.find(target) != hashTable.end()) {  
        return hashTable[target];  
    }  
    return -1;  
}
```

- Bài toán demo



```
Chon phuong thuc tim kiem (1-6): 4  
Nhap phan tu can tim: 6  
  
===== Chi tiet tim kiem bang bam =====  
Tim phan tu 6 trong bang bam...  
=> Tim thay phan tu tai chi so [1]  
Nhan Enter de quay lai menu...
```

Hình 9.4: Demo giải thuật tìm kiếm hàm băm

2.3. Giải thuật tìm kiếm nhảy

```
int timKiemNhay(int arr[], int n, int target) {  
    int step = sqrt(n);  
    int prev = 0;  
    while (arr[min(step, n) - 1] < target) {  
        prev = step;  
        step += sqrt(n);  
        if (prev >= n) {  
            return -1;  
        }  
    }  
    for (int i = prev; i < min(step, n); i++) {  
        if (arr[i] == target)
```



```
        return i;  
    }  
    return -1;  
}
```

- Bài toán demo

```
Chon phuong thuc tim kiem (1-6): 5  
Nhap phan tu can tim: 7  
  
===== Chi tiet tim kiem nhay =====  
Kiem tra khoi [0 -> 1] voi gia tri cuoi: 6  
Tim trong khoi [2 -> 3]  
Kiem tra phan tu tai chi so [2] voi gia tri: 7  
=> Tim thay phan tu tai chi so [2]  
Nhan Enter de quay lai menu...  
_
```

Hình 9.5: Demo giải thuật tìm kiếm nhảy

KẾT LUẬN

Bài tập lớn về "Cấu trúc dữ liệu và giải thuật" đã cung cấp những hiểu biết sâu rộng và thực tiễn về các loại cấu trúc dữ liệu cùng giải thuật. Nội dung đã được triển khai rõ ràng từ các khái niệm cơ bản đến các Giải thuật nâng cao, bao gồm mảng, danh sách liên kết, cây, đồ thị, các Giải thuật sắp xếp và tìm kiếm.

Những nội dung này không chỉ trang bị kiến thức lý thuyết mà còn giúp sinh viên thực hành tư duy lập trình, phân tích, và áp dụng Giải thuật trong việc giải quyết các vấn đề thực tiễn. Điều này góp phần xây dựng nền tảng vững chắc cho việc học tập các môn học chuyên sâu hơn trong lĩnh vực khoa học máy tính.

TÀI LIỆU THAM KHẢO

1. Tài liệu môn học "Cấu trúc dữ liệu và giải thuật" do Th.S Nguyễn Thị Hương hướng dẫn.
2. Giáo trình "Cấu trúc dữ liệu và Giải thuật" – Tài liệu nội bộ Trường Đại học Kỹ thuật Công nghiệp.
3. Các Giải thuật phổ biến trong lập trình được trích dẫn từ tài liệu học thuật và nguồn mở trực tuyến.