# Báo Cáo Giải Quyết Các Bài Tập Thuật Toán

24120471-Nguyễn Trần Minh Trí

Ngày 1 tháng 4 năm 2025

## 1 Giới thiệu

Bài báo cáo này mô tả cách giải quyết các bài tập thuật toán từ bài tập tuần 2 của môn học. Mỗi bài tập đều được giải quyết bằng cách áp dụng các thuật toán cơ bản, như Tìm kiếm Tuyến tính, Tìm kiếm Nhị phân, và các bài toán tìm kiếm và tối ưu hóa. Các bài giải được trình bày theo các bước sau:

## 2 Bài Tập 1: Tìm kiếm Tuyến tính

### 2.1 Phương pháp

Để giải quyết bài toán tìm kiếm tuyến tính, chúng ta sử dụng thuật toán tìm kiếm tuần tự. Cụ thể, thuật toán sẽ duyệt qua từng phần tử của mảng và so sánh với giá trị cần tìm. Nếu tìm thấy, thuật toán trả về chỉ số của phần tử đó, nếu không sẽ trả về -1.

### 2.2 Mã nguồn

```
int linearSearch(int a[], int n, int k) {
    for (int i = 0; i < n; i++) {
        if (a[i] == k) {
            return i;  // Trả về chỉ số nếu tìm thấy
        }
    }
    return -1;  // Nếu không tìm thấy
}
```

# 3 Bài Tập 2: Tìm kiếm Tuyến tính với Sentinel

## 3.1 Phương pháp

Trong bài tập này, chúng ta sử dụng kỹ thuật *sentinel* để tối ưu hóa thuật toán tìm kiếm tuyến tính. Thay vì kiểm tra biên giới của mảng trong mỗi lần lặp, chúng ta gán giá trị cần tìm vào phần tử cuối cùng của mảng. Điều này giúp loại bỏ việc kiểm tra biên giới trong vòng lặp.

## 3.2 Mã nguồn

```
int sentinelLinearSearch(int a[], int n, int k) {
    int last = a[n - 1];  // Lưu giá trị cuối mảng
    a[n - 1] = k;         // Gán sentinel là k

    int i = 0;
    while (a[i] != k) {
        i++;
    }

    a[n - 1] = last;  // Khôi phục giá trị cuối mảng
    if (i < n - 1 || a[n - 1] == k)
        return i;

    return -1;
}
```

# 4 Bài Tập 3: Tìm kiếm Nhị phân trên Mảng Xoay

## 4.1 Phương pháp

Bài toán này yêu cầu tìm phần tử nhỏ nhất trong mảng đã được xoay. Chúng ta sử dụng thuật toán tìm kiếm nhị phân để giải quyết bài toán này, vì mảng đã được sắp xếp và xoay, chúng ta có thể áp dụng kỹ thuật chia đôi khoảng tìm kiếm để tìm ra phần tử nhỏ nhất.

## 4.2 Mã nguồn

```
int findMin(int nums[], int low, int high) {
    while (low < high) {
        int mid = low + (high - low) / 2;
        if (nums[mid] > nums[high]) {
            low = mid + 1;
        } else {
            high = mid;
        }
    }
    return nums[low];
}
```

# 5 Bài Tập 4: Tải Gói trong Giới Hạn Ngày

## 5.1 Phương pháp

Bài toán này yêu cầu tìm ra công suất tàu tối thiểu để vận chuyển tất cả các gói trong một số ngày nhất định. Chúng ta sử dụng phương pháp tìm kiếm nhị phân để xác định công suất tàu tối thiểu, thử nghiệm với các mức công suất khác nhau và kiểm tra xem có thể vận chuyển trong số ngày yêu cầu hay không.

## 5.2 Mã nguồn

```
int shipWithinDays(int weights[], int n, int days) {
    int left = 0, right = 0;
    for (int i = 0; i < n; i++) {
        if (weights[i] > left) left = weights[i];
        right += weights[i];
    }
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (canShipInDays(weights, n, days, mid)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }
    return left;
```

```
}
```

# 6 Bài Tập 5: Tìm Dãy Con Có Tổng Lớn Hơn Hoặc Bằng Target

## 6.1 Phương pháp

Trong bài này, chúng ta sử dụng kỹ thuật hai con trỏ để tìm dãy con có tổng lớn hơn hoặc bằng một giá trị cho trước. Kỹ thuật này giúp giảm thiểu việc phải duyệt lại các dãy con không cần thiết, tiết kiệm thời gian tính toán.

## 6.2 Mã nguồn

```
int minSubArrayLen(int target, int nums[], int n) {
    int left = 0, right = 0, sum = 0;
    int minLength = 1000000;  // Giá trị ban đầu lớn

    while (right < n) {
        sum += nums[right++];
        while (sum >= target) {
            minLength = minLength < (right - left) ? minLength : (right - left);
            sum -= nums[left++];
        }
    }
    return minLength == 1000000 ? 0 : minLength;
}
```

# 7 Bài Tập 6: Kiểm Tra Hai Số Có Tổng Bằng Target

## 7.1 Phương pháp

Ở bài này, chúng ta áp dụng phương pháp hai con trỏ. Do mảng đã được sắp xếp, chúng ta chỉ cần so sánh tổng của phần tử tại hai con trỏ. Nếu tổng nhỏ hơn target, di chuyển con trỏ bên trái, nếu lớn hơn, di chuyển con trỏ bên phải.

### 7.2 Mã nguồn

```
bool twoSum(int nums[], int n, int target) {
    int left = 0, right = n - 1;
    while (left < right) {
        int sum = nums[left] + nums[right];
        if (sum == target) {
            return true;
        } else if (sum < target) {
            left++;
        } else {
            right--;
        }
    }
    return false;
}
```

## 8 Bài Tập 7: Tìm Bộ Ba Có Tổng Bằng 0

### 8.1 Phương pháp

Bài toán này yêu cầu tìm tất cả bộ ba trong mảng có tổng bằng 0. Chúng ta sử dụng phương pháp duyệt qua mảng kết hợp với hai con trỏ để tìm bộ ba thỏa mãn điều kiện.

### 8.2 Mã nguồn

```
void threeSum(int nums[], int n) {
    for (int i = 0; i < n - 2; i++) {
        int left = i + 1, right = n - 1;
        while (left < right) {
            int sum = nums[i] + nums[left] + nums[right];
            if (sum == 0) {
                cout << "[" << nums[i] << ", " << nums[left] << ", " << nums[rig
                left++;
                right--;
            } else if (sum < 0) {
                left++;
            } else {
                right--;
            }
```

```
        }
    }
}
```

# 9 Kết luận

Các bài toán trên được giải quyết bằng các thuật toán tìm kiếm và tối ưu hóa cơ bản như Tìm kiếm Tuyến tính, Tìm kiếm Nhị phân, và phương pháp Hai Con Trỏ. Những thuật toán này không chỉ hiệu quả mà còn dễ dàng triển khai với thời gian tính toán hợp lý.