

Dự án Hệ thống quản lý thiết bị và mượn trả

Võ Khắc Huấn

Trương Đình Thanh

Giới thiệu bài toán

Input: Danh sách thiết bị (ID, tên, loại, mức sử dụng, giá trị), thông tin người mượn.

Output: Quản lý thiết bị (thêm, xóa, cập nhật, tìm kiếm, mượn/trả).

Động lực thực hiện

- Nâng cao hiệu quả quản lý thiết bị trong các tổ chức hoặc phòng thí nghiệm
- Giảm thiểu sai sót trong quản lý mượn trả thiết bị.

Ứng dụng thực tế

Quản lý phòng lab, trung tâm dịch vụ, thư viện thiết bị.

Thành Viên Nhóm và Nhiệm Vụ

Võ Khắc Huấn: Cài đặt thuật toán và các
chức năng chính.

Trương Đình Thanh: Phân tích bài toán,
thiết kế cấu trúc dữ liệu.

- Đánh giá mức độ
hoàn thành: 100%

Phương Pháp Giải Quyết - Dữ Liệu

- Hash Table:
- Dùng để lưu trữ và tìm kiếm thiết bị theo ID (tốc độ cao).

```
#define TABLE_SIZE 100

typedef struct Device {
    int id;
    char name[50];
    char category[50];
    int usageLevel;
    float value;
    struct Device* next; // Xử lý xung đột bảng danh sách liên kết
} Device;

Device* hashTable[TABLE_SIZE]; // Khai báo bảng băm

// Hàm băm
int hashFunction(int id) {
    return id % TABLE_SIZE;
}
```

- Danh sách liên kết:
- Quản lý xung đột trong bảng băm và danh sách mượn trả.
- Mảng:
- Lưu thiết bị để hỗ trợ sắp xếp.

```
typedef struct BorrowRecord {
    int deviceId;
    char borrowerName[50];
    int isReturned; // 0 = đang mượn, 1 = đã trả
    struct BorrowRecord* next;
} BorrowRecord;

BorrowRecord* borrowList = NULL; // Khởi tạo danh sách rỗng
```

Lý do lựa chọn

01

Hash Table: Tìm kiếm nhanh $O(1)$ (trung bình).

Heap Sort: Sắp xếp nhanh $O(n \log n)$.

02

Danh sách liên kết: Dễ dàng thêm và xóa phần tử.

03

Sử dụng bảng băm và danh sách liên kết để xử lý dữ liệu lớn mà không gặp vấn đề về bộ nhớ cố định.

04

Đáp ứng yêu cầu bài toán: Bảng băm giúp tìm kiếm nhanh, danh sách liên kết quản lý thông tin mượn trả hiệu quả, và mảng hỗ trợ sắp xếp giá trị thiết bị.

Phương Pháp Giải Quyết - Giải Thuật

Xóa thiết bị: Tìm trong bảng băm, xóa khỏi danh sách liên kết và mảng.

```
void addDevice(int id, const char* name, const char* category, int usageLevel, float value) {  
    int index = hashFunction(id);  
    Device* newDevice = (Device*)malloc(sizeof(Device));  
    newDevice->id = id;  
    strcpy(newDevice->name, name);  
    strcpy(newDevice->category, category);  
    newDevice->usageLevel = usageLevel;  
    newDevice->value = value;  
    newDevice->next = hashTable[index];  
    hashTable[index] = newDevice;  
  
    deviceArray[deviceCount++] = newDevice;  
}
```

Thêm thiết bị: Tính chỉ số băm, thêm vào danh sách liên kết.

```
void removeDevice(int id) {  
    int index = hashFunction(id);  
    Device* current = hashTable[index];  
    Device* prev = NULL;  
  
    while (current != NULL) {  
        if (current->id == id) {  
            if (prev == NULL) {  
                hashTable[index] = current->next;  
            } else {  
                prev->next = current->next;  
            }  
  
            // Xoa khoi mang  
            for (int i = 0; i < deviceCount; i++) {  
                if (deviceArray[i]->id == id) {  
                    deviceArray[i] = deviceArray[--deviceCount];  
                    break;  
                }  
            }  
  
            free(current);  
            return;  
        }  
        prev = current;  
        current = current->next;  
    }  
    printf("Khong tim thay thiet bi: %d\n", id);  
}
```

Mượn: Thêm bản ghi vào danh sách liên kết.

```
| void borrowDevice(int deviceId, const char* borrowerName) {  
|     BorrowRecord* record = (BorrowRecord*)malloc(sizeof(BorrowRecord));  
|     record->deviceId = deviceId;  
|     strcpy(record->borrowerName, borrowerName);  
|     record->isReturned = 0;  
|     record->next = borrowList;  
|     borrowList = record;  
|     printf("Thiet bi %d da duoc muon boi: %s\n", deviceId, borrowerName);  
| }
```

Trả: Tìm bản ghi theo ID và cập nhật trạng thái.

```
void returnDevice(int deviceId) {  
    BorrowRecord* current = borrowList;  
    while (current) {  
        if (current->deviceId == deviceId && current->isReturned == 0) {  
            current->isReturned = 1;  
            printf("Thiet bi %d da duoc tra\n", deviceId);  
            return;  
        }  
        current = current->next;  
    }  
    printf("Thiet bi %d khong co trong danh sach muon\n", deviceId);  
}
```

```
void heapify(int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && deviceArray[left]->value > deviceArray[largest]->value) {
        largest = left;
    }

    if (right < n && deviceArray[right]->value > deviceArray[largest]->value) {
        largest = right;
    }

    if (largest != i) {
        Device* temp = deviceArray[i];
        deviceArray[i] = deviceArray[largest];
        deviceArray[largest] = temp;

        heapify(n, largest);
    }
}

void heapSortDevices() {
    for (int i = deviceCount / 2 - 1; i >= 0; i--) {
        heapify(deviceCount, i);
    }

    for (int i = deviceCount - 1; i >= 0; i--) {
        Device* temp = deviceArray[0];
        deviceArray[0] = deviceArray[i];
        deviceArray[i] = temp;

        heapify(i, 0);
    }
}
```

Heap Sort: Xây dựng max-heap, sắp xếp phần tử trong mảng deviceArray.

Ưu điểm: Tối ưu hiệu suất cho danh sách lớn.

Tối Ưu Hóa Tìm Kiếm với Bảng Băm

Kết Hợp Nhiều Cấu Trúc Dữ Liệu

```
Device* findDevice(int id) {
    int index = hashFunction(id); // Tìm vị trí trong bảng băm
    Device* current = hashTable[index];
    while (current != NULL) {
        if (current->id == id) {
            return current; // Trả về thiết bị nếu tìm thấy
        }
        current = current->next; // Duyệt qua danh sách liên kết
    }
    return NULL; // Không tìm thấy
}

void addDevice(int id, const char* name, const char* category, int usageLevel, float value) {
    int index = hashFunction(id);
    Device* newDevice = (Device*)malloc(sizeof(Device));
    newDevice->id = id;
    strcpy(newDevice->name, name);
    strcpy(newDevice->category, category);
    newDevice->usageLevel = usageLevel;
    newDevice->value = value;
    newDevice->next = hashTable[index];
    hashTable[index] = newDevice;

    deviceArray[deviceCount++] = newDevice;
}
```

Môi Trường Lập Trình

Ngôn ngữ: C

Thư viện sử dụng :

stdlib.h: Quản lý bộ nhớ.

string.h: Xử lý chuỗi.

stdio.h: Nhập dữ liệu xuất.

IDE : Code::Blocks, Visual Studio Code.

Hệ điều hành : Windows hoặc Linux.

1. Mục Tiêu Thực Nghiệm

Kiểm tra tính đúng đắn: Xem chương trình có xử lý đúng các yêu cầu như thêm, tìm kiếm, xóa thiết bị, mượn trả thiết bị không.

Đánh giá hiệu năng: Đo lường tốc độ xử lý của các chức năng như tìm kiếm bằng bảng băm hoặc sắp xếp bằng Heap Sort.

Độ ổn định: Đánh giá khả năng xử lý dữ liệu lớn mà không xảy ra lỗi hoặc tràn bộ nhớ.

Test Case 1: Kiểm Tra Tính Năng Thêm, Tìm Kiếm, Xóa Thiết Bị

Input:

Thêm các thiết bị vào danh sách.

Tìm kiếm thiết bị theo ID.

Xóa thiết bị và kiểm tra lại danh sách.

```
addDevice(1, "May hien song", "Dien tu", 8, 15000000);
addDevice(2, "Dong ho van nang", "Dien tu", 5, 300000);
addDevice(3, "Nguon cap dien", "Dien tu", 7, 500000);
addDevice(4, "Ampe Ke", "Dien tu", 9, 200000);
displayDevices();|
```

Output Mong Đợi:

Danh sách thiết bị hiển thị đầy đủ sau khi thêm.

Kết quả tìm kiếm trả về đúng thiết bị cần tìm.

Thiết bị bị xóa không còn xuất hiện trong danh sách.

Danh sách thiết bị:

```
ID: 1, Ten: May hien song, Loai: Dien tu, Muc do su dung: 8, Gia tri: 15000000.00
ID: 2, Ten: Dong ho van nang, Loai: Dien tu, Muc do su dung: 5, Gia tri: 300000.00
ID: 3, Ten: Nguon cap dien, Loai: Dien tu, Muc do su dung: 7, Gia tri: 500000.00
ID: 4, Ten: Ampe Ke, Loai: Dien tu, Muc do su dung: 9, Gia tri: 200000.00
```

Test Case 2: Kiểm Tra Tính Năng Mượn/Trả Thiết Bị

Input:

Mượn một số thiết bị với thông tin người mượn.

Trả thiết bị đã mượn.

Hiển thị danh sách mượn trả.

```
borrowDevice(1, "Truong Dinh Thanh");
borrowDevice(2, "Vo Khac Huan");
borrowDevice(3, "Nguyen Van A");
returnDevice(1);
returnDevice(3);
displayBorrowRecords();
updateDevice(3, "Nguon DC", "Dien tu", 9, 550000.00);
sortDevicesByValue();
removeDevice(2);
```

Output Mong Đợi:

Thiết bị được ghi nhận là "đang mượn" sau khi mượn.

Thiết bị chuyển sang trạng thái "đã trả" sau khi hoàn trả.

```
Thiet bi 1 da duoc muon boi: Truong Dinh Thanh
Thiet bi 2 da duoc muon boi: Vo Khac Huan
Thiet bi 3 da duoc muon boi: Nguyen Van A
Thiet bi 1 da duoc tra
Thiet bi 3 da duoc tra

Danh sach muon/trai:
ID Thiet bi: 3, Nguoi muon: Nguyen Van A, Trang thai: Da tra
ID Thiet bi: 2, Nguoi muon: Vo Khac Huan, Trang thai: Dang muon
ID Thiet bi: 1, Nguoi muon: Truong Dinh Thanh, Trang thai: Da tra
Da cap nhat thiet bi: 3

Danh sach thiet bi:
ID: 4, Ten: Ampe Ke, Loai: Dien tu, Muc do su dung: 9, Gia tri: 200000.00
ID: 2, Ten: Dong ho van nang, Loai: Dien tu, Muc do su dung: 5, Gia tri: 300000.00
ID: 3, Ten: Nguon DC, Loai: Dien tu, Muc do su dung: 9, Gia tri: 550000.00
ID: 1, Ten: May hien song, Loai: Dien tu, Muc do su dung: 8, Gia tri: 15000000.00
```

Đưa ra các số liệu so sánh thời gian thực hiện các chức năng chính với dữ liệu lớn. Ví dụ:

Thời gian tìm kiếm (với bảng băm): O(1)

Thời gian sắp xếp (Heap Sort): O(nlogn)

Thời gian thêm thiết bị (vào bảng băm): O(1)

Đánh Giá

Tính đúng đắn: Chương trình chạy đúng với tất cả test case.

Hiệu suất: Tìm kiếm và sắp xếp hoạt động nhanh với dữ liệu lớn (10,000 thiết bị).

Độ ổn định: Không phát hiện lỗi tràn bộ nhớ hoặc sai sót dữ liệu.

Mức độ hoàn thành:

Đạt được mục tiêu quản lý thiết bị hiệu quả.

Hệ thống ổn định và có thể mở rộng.

Khó khăn:

Tối ưu hóa thuật toán băm.

Quản lý xung đột trong bảng băm.

Ưu điểm:

Tốc độ cao, dễ sử dụng.

Giải pháp:

Tăng kích thước bảng băm.

Sử dụng danh sách liên kết để xử lý xung đột.

Hạn chế:

Không có giao diện đồ họa.

Chưa hỗ trợ nhiều người dùng đồng thời.

Hướng phát triển:

Xây dựng giao diện web hoặc app.

Hỗ trợ truy cập từ xa qua mạng.