

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



|  |  |
| --- | --- |
| Giảng viên hướng dẫn: Trần Thị Dung |  |
| Sinh viên 1: Vũ Quang Huy | MSSV: 6051071049 |
| Sinh viên 2: Hồ Ngọc Thống | MSSV: 6051071118 |
| Sinh viên 3: Nguyễn Văn Hoài | MSSV: 6051071044 |
| Mã học phần: CPM215.3 |  |

**BÁO CÁO BÀI TẬP LỚN**

**LẬP TRÌNH NÂNG CAO**

**CHỦ ĐỀ: XÂY DỰNG CẤU TRÚC VỀ CÁC QUỐC GIA   
VÀ CÁC THÀNH PHỐ TRÊN THẾ GIỚI**

# Lời cảm ơn

Trong quá trình làm bài tập lớn, em xin chân thành cảm ơn cô và các bạn trong lớp cũng như các nhóm khác đã nhiệt tình hỗ trợ, hướng dẫn và phân tích để giúp chúng em hiểu rõ hơn về đề tài này cũng như là các vấn đề trong viết code. Vì đây là lần đầu tiên làm bài tập lớn của cả nhóm nên nếu có vấn đề gì sai sót cũng như thiếu ý, chưa đáp ứng được nhu cầu thì mong cô và các bạn góp ý để em có thể dần hoàn thiện và phát triển hơn.

Nhóm xin chân thành cảm ơn.

**MỤC LỤC**

LỜI CẢM ƠN [1](#_Toc43384098)

[CHƯƠNG 1 : TỒNG QUAN VỀ ĐỀ TÀI 4](#_Toc43384099)

[1.1.Lý do chọn đề tài 4](#_Toc43384100)

[1.2.Mục đích đề tài 4](#_Toc43384101)

[1.3.Phạm vi nghiên cứu 4](#_Toc43384102)

[CHƯƠNG 2: GIẢI THÍCH NỘI DUNG LÍ THUYẾT 5](#_Toc43384103)

[2.1.Làm việc với tệp 5](#_Toc43384104)

[2.1.1.Định nghĩa 5](#_Toc43384105)

[2.1.2.Khai báo biến FILE 5](#_Toc43384108)

[2.1.3.Thao tác mở file 6](#_Toc43384109)

[2.1.4.Thao tác đóng file 6](#_Toc43384110)

[2.1.5. Đọc/ghi file 6](#_Toc43384111)

[2.2.Cấu trúc liên kết đơn 8](#_Toc43384114)

[2.3. Các thuật toán sắp xếp 12](#_Toc43384118)

[2.3.1.Sắp xếp nổi bọt (Bubble Sort) 12](#_Toc43384119)

[2.3.2.Sắp xếp chèn (Insertion Sort) 14](#_Toc43384121)

[2.3.4.Giải thuật sắp xếp chọn (Selection Sort) : 16](#_Toc43384123)

[2.3.5.Giải thuật sắp xếp trộn (Merge Sort) : 19](#_Toc43384125)

[2.3.6.Sắp xếp nhanh (Quick Sort) : 20](#_Toc43384127)

[2.4. Tìm kiếm: 21](#_Toc43384129)

[CHƯƠNG 3: GIẢI THÍCH THUẬT TOÁN VÀ PHÂN TÍCH YÊU CẦU 24](#_Toc43384133)

[3.1. Xác định yêu cầu 24](#_Toc43384134)

[3.2.Phân tích yêu cầu 24](#_Toc43384135)

[CHƯƠNG 4 : HƯỚNG DẪN SỬ DỤNG CHƯƠNG TRÌNH 31](#_Toc43384177)

[CHƯƠNG 5: KẾT LUẬN VÀ ĐỊNH HƯỚNG PHÁT TRIỂN. 32](#_Toc43384178)

# CHƯƠNG 1 : TỒNG QUAN VỀ ĐỀ TÀI

## 1.1.Lý do chọn đề tài

Trong thời kì cách mạng công nghiệp 4.0 hiện nay, những chương trình quản lí thành phố rất là quan trọng. Chúng ta có thể dùng nó để liệt kê những thành phố, tìm kiếm, hay dùng nó để áp dụng vào những vấn đề khác như : quản lí các thành phố nằm trong tâm dịch covid-19 , có bao nhiêu người bị nhiễm…. Vậy nên nhóm em chọn đề tài để thực hiện một phần nào trong chương trình, và hiểu hơn về thuật toán cũng như cách tạo một chương trình quản lí.

Đề tài có tính vận dụng cao trong đời sống, giúp mọi người có thêm hiểu biết về các quốc gia và các thành phố trên thế giới, nên nhóm quyết định chọn và thực hiện.

## 1.2.Mục đích đề tài

Chương trình sẽ là công cụ hỗ trợ đắc lực trong việc quản lí và tìm kiếm thành phố theo quốc gia. Giúp các bạn hiểu rõ hơn về các thuật toán đơn giản.

## 1.3.Phạm vi nghiên cứu

Ngôn ngữ lập trình C.

# 

# CHƯƠNG 2: GIẢI THÍCH NỘI DUNG LÍ THUYẾT

## 2.1.Làm việc với tệp

### 2.1.1.Định nghĩa

## Tệp (File) là kiểu đối tượng, nó xác định một stream và chứa các thông tin cần thiết để điều khiển, bao gồm một con trỏ trỏ đến buffer của nó, các chỉ mục và trạng thái của nó.

## Các bạn có thể hiểu file (trong ngôn ngữ lập trình C) là một kiểu đối tượng mà thông qua nó chúng ta có thể thao tác với dữ liệu được lưu trữ bên trong file (chứ không phải là một file trên máy tính).

### **2.1.2.Khai báo biến FILE**

**FILE\* <tên biến file>;**

* Kiểu dữ liệu của file là kiểu FILE\*.
* <tên biến file>: cách đặt tên biến file theo yêu cầu đặt tên của ngôn ngữ C.

**Gán biến file (đã khai báo) với một file vật lý**

**<tên biến file>=fopen(<duong dan file>, <cách mở file>);**

* Fopen là hàm mở file.
* <duong dan file>:là chuỗi chứa đường dẫn tới file bạn muốn mở.

VD: bạn muốn mở file **txt** trong ổ đĩa thư mục LapTrinhC của ổ đĩa E thì **<duong dan file>** chính là “**E:\\LapTrinhC\\dulieu.txt**”. Lưu ý ta phải dùng \\ thay vì \ trong chuỗi **<duong dan file>.**

* **<**cách mở file**>**: là một chuỗi qui định mục đích cho việc sử dụng file có thể sử dụng một trong các giá trị sau:
  + **“w”**: mở file để ghi, nếu file muốn mở chưa có thì chương trình sẽ tự động tạo ra file đó. Nếu file muốn mở đã có sẵn thì file sẽ được mở, tuy nhiên dữ liệu trong file nếu có sẽ bị xóa hết trở thành file trống.
  + **“r”**: mở file để đọc, nếu file chưa có thì chương trình sẽ báo lỗi. Nếu file đã tồn tại thì chương trình sẽ mở file. Dữ liệu trong file sẽ không thay đổi.
  + **“a”**: mở file để ghi thêm, cách hoạt động giống “w”, tuy nhiên nếu file đã có dữ liệu thì chương trình sẽ không xóa dữ liệu có trước mà sẽ ghi tiếp vào phía sau.
  + **“r+”, “w+”**: mở file, cho phép đọc và ghi file.

### 2.1.3.Thao tác mở file

Để đọc ghi file trong C cũng như trong mọi ngôn ngữ lập trình, việc đầu tiên bạn cần làm là mở file mà bạn muốn làm việc. Trong ngôn ngữ lập trình C, chúng ta có thể mở file bằng cách sử dụng hàm fopen() trong thư viện <stdio.h> như sau:

Tên biến con trỏ tệp = fopen("tên tệp", "kiểu");

VD: np=fopen("vb.txt", "w");

### 2.1.4.Thao tác đóng file

Sau khi thao tác với file xong, các bạn cần đóng file lại để tránh những lỗi phát sinh ngoài ý muốn. Để đóng file , chúng ta sử dụng hàm:

Fclose : int fclose(FILE \*file);

Trong đó, file là con trỏ được dùng để lưu trữ địa chỉ của đối tượng FILE đang mở. Nếu đóng file thành công thì trả về giá trị 0, ngược lại trả về EOF (End of file).

### 2.1.5. Đọc/ghi file

### Để làm việc với file văn bản, chúng ta sẽ sử dụng fprintf() và fscanf()

* Ghi file sử dụng fprintf()

#include <stdio.h>

#include <stdlib.h>

int main()

{

   int num;

   FILE \*fptr;

   fptr = fopen("C:\\program.txt","w");

   if(fptr == NULL)

   {

      printf("Error!");

      exit(1);

   }

   printf("Enter num: ");

   scanf("%d",&num);

   fprintf(fptr,"%d",num);

   fclose(fptr);

   return 0;

}

2.1.5 Hình 1 : ghi file

* Đọc file sử dụng fscanf()

#include <stdio.h>

#include <stdlib.h>

int main()

{

   int num;

   FILE \*fptr;

   if ((fptr = fopen("C:\\program.txt","r")) == NULL){

       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.

       exit(1);

2.1.5 Hình 2 : đọc file

   }

   fscanf(fptr,"%d", &num);

   printf("Value of n=%d", num);

   fclose(fptr);

   return 0;

}

## 2.2.Cấu trúc liên kết đơn

### 2.2.1.Khai báo liên kết đơn

Struct LinkedList{

    int data;

    struct LinkedList \*next;

};

Data sẽ lưu giữa giá trị và next sẽ là con trỏ để trỏ đến thằng kế tiếp nó.

**Định nghĩa** : Danh sách liên kết đơn là một tập hợp các Node được phân bố động, được sắp xếp theo cách sao cho mỗi Node chứa “một giá trị”(Data) và “một con trỏ”(Next). Con trỏ sẽ trỏ đến phần tử kế tiếp của danh sách liên kết đó. Nếu con trỏ mà trỏ tới NULL, nghĩa là đó là phần tử cuối cùng của linked list.

Hình ảnh mô tả cho một Node trong danh sách liên kết đơn:

[](https://nguyenvanhieu.vn/wp-content/uploads/2018/12/data-node-danh-sach-lien-ket.png)

2.2.1 hình 1 : định nghĩa liên kết đơn

Và đây là hình ảnh mô phỏng một danh sách liên đơn kết đầy đủ:

[](https://nguyenvanhieu.vn/wp-content/uploads/2018/12/danh-sach-lien-ket-la-gi.jpg) 2.2.1 hình 2 : mô phỏng danh sách liên kết đơn

## 2.2.2.Tạo mới 1 Node

Tạo một kiểu dữ liệu của struct LinkedList :

typedef struct LinkedList \*node; //Từ giờ dùng kiểu dữ liệu LinkedList có thể thay bằng node cho ngắn gọn

node CreateNode(int value){

    node temp; // declare a node

    temp = (node)malloc(sizeof(struct LinkedList)); // Cấp phát vùng nhớ dùng malloc()

    temp->next = NULL;// Cho next trỏ tới NULL

    temp->data = value; // Gán giá trị cho Node

    return temp;//Trả về node mới đã có giá trị

}

2.2.2 hình 1 : tạo mới một node

Mỗi một Node khi được khởi tạo, chúng ta cần cấp phát bộ nhớ cho nó, và mặc định cho con trỏ next trỏ tới NULL. Giá trị của Node sẽ được cung cấp khi thêm Node vào linked list.

* **typedef** được dùng để định nghĩa một kiểu dữ liệu trong C. VD: typeder long long LL;
* **malloc** là hàm cấp phát bộ nhớ của C. Với C++ chúng ta dùng new
* **sizeof** là hàm trả về kích thước của kiểu dữ liệu, dùng làm tham số cho hàm malloc

**Lưu ý:** Không giống với mảng, cần khai báo arr[size]. Trong linked list, vì mỗi Node sẽ có con trỏ liên kết đến Node tiếp theo. Do đó, với danh sách liên kết đơn, bạn chỉ cần lưu giữ Node đầu tiên(HEAD). Có head rồi bạn có thể đi tới bất cứ Node nào.

### 2.2.3.Xóa 1 Node ở danh sách liên kết :

* **Xóa đầu :**

Chúng ta chỉ cần cho thằng kế tiếp của head làm head là được thôi. Mà thằng kế tiếp của head chính là head->next

node DelHead(node head){

if(head == NULL){

printf("\nCha co gi de xoa het!");

}else{

head = head->next;

}

return head;

}

2.2.3 hình 1 : xóa node ở vị trí đầu

* **Xóa cuối :**

Không như xóa đầu, xóa cuối chúng ta phải duyệt đến thằng cuối – 1, cho next của cuối – 1 đó bằng NULL.

node DelTail(node head){

if (head == NULL || head->next == NULL){

return DelHead(head);

}

node p = head;

while(p->next->next != NULL){

p = p->next;

}

p->next = p->next->next; // Cho next bằng NULL

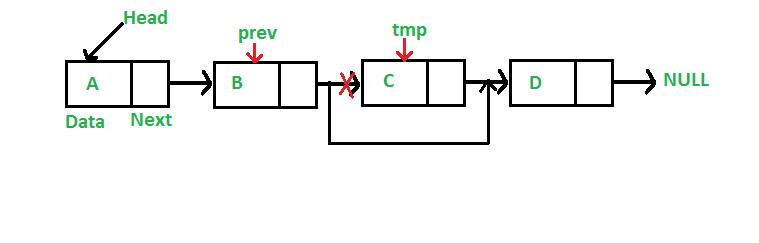
// Hoặc viết p->next = NULL cũng được

return head;

}

2.2.3 hình 2 : xóa node ở vị trí cuối

#### **Xóa ở vị trí bất kỳ**

[](https://nguyenvanhieu.vn/wp-content/uploads/2018/12/xoa-node-danh-sach-lien-ket.png)Việc xóa ở vị trí bất kỳ cũng khá giống xóa ở cuối kia. Đơn giản là chúng ta bỏ qua một phần tử, như ảnh sau:

node DelAt(node head, int position){

    if(position == 0 || head == NULL || head->next == NULL){

        head = DelHead(head); // Nếu vị trí chèn là 0, tức là thêm vào đầu

    }else{

        // Bắt đầu tìm vị trí cần chèn. Ta sẽ dùng k để đếm cho vị trí

        int k = 1;

        node p = head;

        while(p->next->next != NULL && k != position){

            p = p->next;

            ++k;

        }

        if(k != position){

            // Nếu duyệt hết danh sách lk rồi mà vẫn chưa đến vị trí cần chèn, ta sẽ mặc định xóa cuối

            // Nếu bạn không muốn xóa, hãy thông báo vị trí xóa không hợp lệ

            head = DelTail(head);

            // printf("Vi tri xoa vuot qua vi tri cuoi cung!\n");

        }else{

            p->next = p->next->next;

        }

    }

    return head;

}

2.2.3 hình 3 : xóa node ở vị trí bất kì

## 2.3. Các thuật toán sắp xếp

### 2.3.1.Sắp xếp nổi bọt (Bubble Sort)

Sắp xếp nổi bọt là một giải thuật sắp xếp đơn giản. Giải thuật sắp xếp này được tiến hành dựa trên việc so sánh cặp phần tử liền kề nhau và tráo đổi thứ tự nếu chúng không theo thứ tự.

Giải thuật này không thích hợp sử dụng với các tập dữ liệu lớn khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là Ο(n2) với n là số phần tử.

Giải thuật sắp xếp nổi bọt là giải thuật chậm nhất trong số các giải thuật sắp xếp cơ bản. Giải thuật này còn chậm hơn giải thuật đổi chỗ trực tiếp mặc dù số lần so sánh bằng nhau, nhưng do đổi chỗ hai phần tử kề nhau nên số lần đổi chỗ nhiều hơn.

## *Cách giải thuật sắp xếp nổi bọt làm việc?*

Giả sử chúng ta có một mảng không có thứ tự gồm các phần tử như dưới đây. Bây giờ chúng ta sử dụng giải thuật sắp xếp nổi bọt để sắp xếp mảng này.

Sắp xếp nổi bọt (Bubble Sort)

Giải thuật sắp xếp nổi bọt bắt đầu với hai phần tử đầu tiên, so sánh chúng để kiểm tra xem phần tử nào lớn hơn.

Sắp xếp nổi bọt (Bubble Sort)

Trong trường hợp này, 33 lớn hơn 14, do đó hai phần tử này đã theo thứ tự. Tiếp đó chúng ta so sánh 33 và 27.

Sắp xếp nổi bọt (Bubble Sort)

Chúng ta thấy rằng 33 lớn hơn 27, do đó hai giá trị này cần được tráo đổi thứ tự.

Sắp xếp nổi bọt (Bubble Sort)

Mảng mới thu được sẽ như sau:

Sắp xếp nổi bọt (Bubble Sort)

Tiếp đó chúng ta so sánh 33 và 35. Hai giá trị này đã theo thứ tự.

Sắp xếp nổi bọt (Bubble Sort)

Sau đó chúng ta so sánh hai giá trị kế tiếp là 35 và 10.

Sắp xếp nổi bọt (Bubble Sort)

Vì 10 nhỏ hơn 35 nên hai giá trị này chưa theo thứ tự.

Sắp xếp nổi bọt (Bubble Sort)

Tráo đổi thứ tự hai giá trị. Chúng ta đã tiến tới cuối mảng. Vậy là sau một vòng lặp, mảng sẽ trông như sau:

Sắp xếp nổi bọt (Bubble Sort)

Để đơn giản, tiếp theo mình sẽ hiển thị hình ảnh của mảng sau từng vòng lặp. Sau lần lặp thứ hai, mảng sẽ trông giống như:

Sắp xếp nổi bọt (Bubble Sort)

Sau mỗi vòng lặp, ít nhất một giá trị sẽ di chuyển tới vị trí cuối. Sau vòng lặp thứ 3, mảng sẽ trông giống như:

Sắp xếp nổi bọt (Bubble Sort)

Và khi không cần tráo đổi thứ tự phần tử nào nữa, giải thuật sắp xếp nổi bọt thấy rằng mảng đã được sắp xếp xong.

Sắp xếp nổi bọt (Bubble Sort)

VD: Thuật toán sắp xếp dãy số tăng dần

    int tg;

    for(int i = 0; i < n - 1; i++){

        for(int j = i + 1; j < n; j++){

            if(a[i] < a[j]){

                // Hoan vi 2 so a[i] va a[j]

                tg = a[i];

                a[i] = a[j];

                a[j] = tg;

            }

        }

    }

2.2.3 hình 1 : ví dụ sắp xếp nổi bọt

### 2.3.2.Sắp xếp chèn (Insertion Sort)

Sắp xếp chèn là một giải thuật sắp xếp dựa trên so sánh in-place. Ở đây, một danh sách con luôn luôn được duy trì dưới dạng đã qua sắp xếp. Sắp xếp chèn là chèn thêm một phần tử vào danh sách con đã qua sắp xếp. Phần tử được chèn vào vị trí thích hợp sao cho vẫn đảm bảo rằng danh sách con đó vẫn sắp theo thứ tự.

Với cấu trúc dữ liệu mảng, chúng ta tưởng tượng là: mảng gồm hai phần: một danh sách con đã được sắp xếp và phần khác là các phần tử không có thứ tự. Giải thuật sắp xếp chèn sẽ thực hiện việc tìm kiếm liên tiếp qua mảng đó, và các phần tử không có thứ tự sẽ được di chuyển và được chèn vào vị trí thích hợp trong danh sách con (của cùng mảng đó).

Giải thuật này không thích hợp sử dụng với các tập dữ liệu lớn khi độ phức tạp trường hợp xấu nhất và trường hợp trung bình là Ο(n2) với n là số phần tử.

## *Cách giải thuật sắp xếp chèn thực hiện?*

Ví dụ chúng ta có một mảng gồm các phần tử không có thứ tự:

Sắp xếp chèn (Insertion Sort)

Giải thuật sắp xếp chèn so sánh hai phần tử đầu tiên:

Sắp xếp chèn (Insertion Sort)

Giải thuật tìm ra rằng cả 14 và 33 đều đã trong thứ tự tăng dần. Bây giờ, 14 là trong danh sách con đã qua sắp xếp.

Sắp xếp chèn (Insertion Sort)

Giải thuật sắp xếp chèn tiếp tục di chuyển tới phần tử kế tiếp và so sánh 33 và 27.

Sắp xếp chèn (Insertion Sort)

Và thấy rằng 33 không nằm ở vị trí đúng.

Sắp xếp chèn (Insertion Sort)

Giải thuật sắp xếp chèn tráo đổi vị trí của 33 và 27. Đồng thời cũng kiểm tra tất cả phần tử trong danh sách con đã sắp xếp. Tại đây, chúng ta thấy rằng trong danh sách con này chỉ có một phần tử 14 và 27 là lớn hơn 14. Do vậy danh sách con vẫn giữ nguyên sau khi đã tráo đổi.

Sắp xếp chèn (Insertion Sort)

Bây giờ trong danh sách con chúng ta có hai giá trị 14 và 27. Tiếp tục so sánh 33 với 10.

Sắp xếp chèn (Insertion Sort)

Hai giá trị này không theo thứ tự.

Sắp xếp chèn (Insertion Sort)

Vì thế chúng ta tráo đổi chúng.

Sắp xếp chèn (Insertion Sort)

Việc tráo đổi dẫn đến 27 và 10 không theo thứ tự.

Sắp xếp chèn (Insertion Sort)

Vì thế chúng ta cũng tráo đổi chúng.

Sắp xếp chèn (Insertion Sort)

Chúng ta lại thấy rằng 14 và 10 không theo thứ tự.

Sắp xếp chèn (Insertion Sort)

Và chúng ta tiếp tục tráo đổi hai số này. Cuối cùng, sau vòng lặp thứ 3 chúng ta có 4 phần tử.

Sắp xếp chèn (Insertion Sort)

Tiến trình trên sẽ tiếp tục diễn ra cho tới khi tất cả giá trị chưa được sắp xếp được sắp xếp hết vào trong danh sách con đã qua sắp xếp.

### 2.3.4.Giải thuật sắp xếp chọn (Selection Sort) :

Giải thuật sắp xếp chọn (Selection Sort) là một giải thuật đơn giản. Giải thuật sắp xếp này là một giải thuật dựa trên việc so sánh **in-place**, trong đó danh sách được chia thành hai phần, phần được sắp xếp (sorted list) ở bên trái và phần chưa được sắp xếp (unsorted list) ở bên phải. Ban đầu, phần được sắp xếp là trống và phần chưa được sắp xếp là toàn bộ danh sách ban đầu.

Phần tử nhỏ nhất được lựa chọn từ mảng chưa được sắp xếp và được tráo đổi với phần bên trái nhất và phần tử đó trở thành phần tử của mảng được sắp xếp. Tiến trình này tiếp tục cho tới khi toàn bộ từng phần tử trong mảng chưa được sắp xếp đều được di chuyển sang mảng đã được sắp xếp.

Giải thuật này không phù hợp với tập dữ liệu lớn khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là O(n2) với n là số phần tử.

## *Cách giải thuật sắp xếp chọn (Selection Sort) làm việc :*

Dưới đây là các hình minh họa cho cách giải thuật sắp xếp chọn làm việc. Giả sử chúng ta có một mảng như sau:

Giải thuật sắp xếp chọn (Selection Sort)

Từ vị trí đầu tiên trong danh sách đã được sắp xếp, toàn bộ danh sách được duyệt một cách liên tục. Vị trí đầu tiên có giá trị 14, chúng ta tìm toàn bộ danh sách và thấy rằng 10 là giá trị nhỏ nhất.

Giải thuật sắp xếp chọn (Selection Sort)

Do đó, chúng ta thay thế 14 với 10. Sau một vòng lặp, giá trị 10 thay thế cho giá trị 14 tại vị trí đầu tiên trong danh sách đã được sắp xếp. Chúng ta tráo đổi hai giá trị này.

Giải thuật sắp xếp chọn (Selection Sort)

Tại vị trí thứ hai, giá trị 33, chúng ta tiếp tục quét phần còn lại của danh sách theo thứ tự từng phần tử.

Giải thuật sắp xếp chọn (Selection Sort)

Chúng ta thấy rằng 14 là giá trị nhỏ nhất thứ hai trong danh sách và nó nên xuất hiện ở vị trí thứ hai. Chúng ta tráo đổi hai giá trị này.

Giải thuật sắp xếp chọn (Selection Sort)

Giải thuật sắp xếp chọn (Selection Sort)Sau hai vòng lặp, hai giá trị nhỏ nhất đã được đặt tại phần đầu của danh sách đã được sắp xếp.

Tiến trình tương tự sẽ được áp dụng cho phần còn lại của danh sách. Các hình dưới minh họa cho các tiến trình này

2.3.4 hình 1 : minh họa sắp xếp chọn

Tiến trình tương tự sẽ được áp dụng cho phần còn lại của danh sách. Các hình dưới minh họa cho các tiến trình này.

### 2.3.5.Giải thuật sắp xếp trộn (Merge Sort) :

Sắp xếp trộn (Merge Sort) là một giải thuật sắp xếp dựa trên giải thuật **Chia để trị (Divide and Conquer)**. Với độ phức tạp thời gian trường hợp xấu nhất là Ο(n log n) thì đây là một trong các giải thuật đáng được quan tâm nhất.

Đầu tiên, giải thuật sắp xếp trộn chia mảng thành hai nửa và sau đó kết hợp chúng lại với nhau thành một mảng đã được sắp xếp.

## *Cách giải thuật sắp xếp trộn (Merge Sort) làm việc :*

Dưới đây là các hình minh họa cách giải thuật sắp xếp trộn làm việc. Giả sử chúng ta có mảng sau:

Giải thuật sắp xếp trộn (Merge Sort)

Đầu tiên, giải thuật sắp xếp trộn chia toàn bộ mảng thành hai nửa. Tiến trình chia này tiếp tục diễn ra cho đến khi không còn chia được nữa và chúng ta thu được các giá trị tương ứng biểu diễn các phần tử trong mảng. Trong hình dưới, đầu tiên chúng ta chia mảng kích cỡ 8 thành hai mảng kích cỡ 4.

Giải thuật sắp xếp trộn (Merge Sort)

Tiến trình chia này không làm thay đổi thứ tự các phần tử trong mảng ban đầu. Bây giờ chúng ta tiếp tục chia các mảng này thành 2 nửa.

Giải thuật sắp xếp trộn (Merge Sort)

Tiến hành chia tiếp cho tới khi không còn chia được nữa.

Giải thuật sắp xếp trộn (Merge Sort)

Bây giờ chúng ta tổ hợp chúng theo như đúng cách thức mà chúng được chia ra.

Đầu tiên chúng ta so sánh hai phần tử trong mỗi list và sau đó tổ hợp chúng vào trong một list khác theo cách thức đã được sắp xếp. Ví dụ, 14 và 33 là trong các vị trí đã được sắp xếp. Chúng ta so sánh 27 và 10 và trong list khác chúng ta đặt 10 ở đầu và sau đó là 27. Tương tự, chúng ta thay đổi vị trí của 19 và 35. 42 và 44 được đặt tương ứng.

Giải thuật sắp xếp trộn (Merge Sort)

Vòng lặp tiếp theo là để kết hợp từng cặp list một ở trên. Chúng ta so sánh các giá trị và sau đó hợp nhất chúng lại vào trong một list chứa 4 giá trị, và 4 giá trị này đều đã được sắp thứ tự.

Giải thuật sắp xếp trộn (Merge Sort)

Sau bước kết hợp cuối cùng, danh sách sẽ trông giống như sau:

Giải thuật sắp xếp trộn (Merge Sort)

### 2.3.6.Sắp xếp nhanh (Quick Sort) :

Giải thuật sắp xếp nhanh (Quick Sort) là một giải thuật hiệu quả cao và dựa trên việc chia mảng dữa liệu thành các mảng nhỏ hơn. Giải thuật sắp xếp nhanh chia mảng thành hai phần bằng cách so sánh từng phần tử của mảng với một phần tử được chọn gọi là **phần tử chốt (Pivot)**: một mảng bao gồm các phần tử nhỏ hơn hoặc bằng phần tử chốt và mảng còn lại bao gồm các phần tử lớn hơn hoặc bằng phần tử chốt.

Tiến trình chia này diễn ra tiếp tục cho tới khi độ dài của các mảng con đều bằng 1. Giải thuật sắp xếp nhanh tỏ ra khá hiệu quả với các tập dữ liệu lớn khi mà độ phức tạp trường hợp trung bình và trường hợp xấu nhất là O(nlogn) với n là số phần tử.

## Kỹ thuật chọn phần tử chốt trong giải thuật sắp xếp nhanh (Quick Sort)

Kỹ thuật chọn phần tử chốt ảnh hưởng khá nhiều đến khả năng rơi vào các vòng lặp vô hạn đối với các trường hợp đặc biệt. Tốt nhất là chọn phần tử chốt (pivot) nằm ở trung vị của danh sách. Khi đó, sau log2(n) lần chia chúng ta sẽ đạt tới kích thước các mảng con bằng 1.

Dưới đây là các cách chọn phần tử chốt:

* Chọn phần tử đứng đầu hoặc đứng cuối làm phần tử chốt.
* Chọn phần tử đứng giữa danh sách làm phần tử chốt.
* Chọn phần tử trung vị trong ba phần tử đứng đầu, đứng giữa và đứng cuối làm phần tử chốt.
* Chọn phần tử ngẫu nhiên làm phần tử chốt. Tuy nhiên cách này rất dễ dẫn đến khả năng rơi vào các trường hợp đặc biệt.

## 2.4. Tìm kiếm:

### 2.4.1.Thuật toán tìm kiếm nhị phân:

Cho một mảng đã sắp xếp mảng[] có n phần tử, viết một hàm tìm kiếm trả về chỉ số của phần tử có giá trị x trong mảng[].

Giải thuật đơn giản nhất cho bài toán này là sử dụng linear search(tìm kiếm tuyến tính). Tức là bạn sẽ phải đi qua từng phần tử của mảng để đối chiếu với x cần tìm. Thuật toán này trong trường hợp xấu nhất cho độ phức tạp là O(n). Đây là code C sử dụng linear search.

A screenshot of a map

Description automatically generated

### 2.4.1 hình 1 : sử dụng linear search

### 2.4.2.Minh họa code cho thuật toán tìm kiếm nhị phân

Code minh họa C sử dụng đệ quy

A screenshot of a cell phone

Description automatically generated  
 2.4.2 hình 2 : sử dụng đệ quy

### 2.4.3.Thuật toán tìm kiếm tuyến tính.

Giải thuật chính của tìm kiếm tuyến tính chính là: so sánh phần tử cần tìm với tất cả các phần tử có trong mảng hoặc danh sách cần tìm. Chạy từ phần tử đầu đến cuối và so sánh từng đôi một, nếu bằng thì thông báo có, ngược lại nếu đã đi hết dãy mà vẫn chưa có phần tử nào thõa mãn thì cho kết quả là không tìm thấy.

**Giải thuật:**

Bước 1: Khởi tạo biến i và gán biến i bằng 0;

Bước 2: so sánh a[i] với giá trị cầm tìm.

+ Nếu tìm được giá trị a[i] bằng giá trị cần tìm thì dừng lại và dừng. Ngược lại

+  Nếu  a[i] khác giá trị cần tìm thì sang bước 3

Bước 3: Tăng i lên một đơn vị, nếu i bằng số phần tử trừ 1 của mảng thì dừng lại và cho kết quả là không tìm thấy. Ngược lại quay lại bước 2.

**Ví dụ**: Cho dãy A gồm các phần tử: 11 4 3 9 8 0 2 45. Dùng giải thuật tìm kiếm tuyến tính để tìm xem có phần tử 8 nằm ở trong mảng hay không.

Bước 1: gán i=0.

Bước 2: so sánh a[0]= 11 != 8. Tăng i lên một đơn vị.

Bước 3: i=1 < n-1 (n = 8). Quay lại bước 2.

So sánh a[1] = 4 !=8. Tăng i lên một đơn vị .

Lặp lại Bước 3: i=2 < n-1 (n = 8). Quay lại bước 2.

So sánh a[2] = 3 !=8. Tăng i lên một đơn vị .

Lặp lại Bước 3: i=3 < n-1 (n = 8). Quay lại bước 2.

So sánh a[3] = 9 !=8. Tăng i lên một đơn vị .

Lặp lại Bước 3: i=4 < n-1 (n = 8). Quay lại bước 2.

So sánh a[4] = 8 = 8. Nên kết thúc và tìm được x ở vị trí số a[4].

Ngoài việc tìm kiếm trong mảng số chúng ta còn có thể áp dụng giải thuật tìm kiếm cho mảng, danh sách với cấu trúc dưới dạng struct.

# CHƯƠNG 3: GIẢI THÍCH THUẬT TOÁN VÀ PHÂN TÍCH YÊU CẦU

## 3.1. Xác định yêu cầu

- Chương trình phải chạy được.

- Đáp ứng đủ nhu cầu người dùng.

- Trình bày rõ ràng dễ hiểu.

## 3.2.Phân tích yêu cầu

***\* Định nghĩa 1 cấu trúc***

- struct city\_st: char city\_nm[30], char country [30], int year.

typedef struct{

char city\_nm[30];

char country[30];

int year;

} city\_st;

3.2 hình 1 : định nghĩa cấu trúc

- Trong hàm main() khai báo một mảng bao gồm n thành phố với cấu trúc ở trên có tên là "cityList".

int main(){

int n;

city\_st \*cityList;

cityList=(city\_st\*)malloc(n\*sizeof(city\_st));

}

3.2 hình 2

***\*Tạo ra Menu dưới đây:***

# Nhap du lieu cua tung thanh pho.

# Sap xep, thong kê va hien thi thong tin chi tiet cua tung thanh pho theo quoc gia (Z->A).

# Tim thanh pho theo quoc gia

# Ghi vao tap tin nhi phan city.dat.

# Thoat.

# Hàm menu được xây dựng như sau:

//Hàm menu

int menu(city\_st \*c, int n) {

int luachon;

//Hien ra bang menu cho nguoi dung chon

printf("\n");

printf("\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\n");

printf("\* \*\n");

printf("\* (MENU) Chon mot trong cac cong viec sau: \*\n");

printf("\* 1. Nhap du lieu cua tung thanh pho \*\n");

printf("\* 2. Sap xep va hien thi thong tin chi tiet \*\n\* cua tung thanh pho theo quoc gia (Z->A). \*\n");

printf("\* 3. Tim thanh pho theo quoc gia \*\n");

printf("\* 4. Ghi vao tap tin nhi phan city.dat \*\n");

printf("\* 5. Thoat \*\n");

printf("\* \*\n");

printf("\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\n\n");

printf("Nhap vao lua chon cua ban (1 den 5): ");

scanf("%d", &luachon);

switch(luachon) {

case 1: nhap(c, n); break;

case 2: xuat(c, n); sapXep(c, n); break;

case 3: timKiem(c, n); break;

case 4: ghi(c, n); break;

case 5: break;

default: printf("khong co lua chon %d, moi ban lua chon lai!\n", luachon);

menu(c,n);

}

}

# 3.2 hình 3 : menu

# \*Nhập dữ liệu của từng thành phố*.* Ví dụ: (chữ in đậm là dữ liệu được nhập vào từ người chạy chương trình):

# Nhap thanh pho 1:

# Ten: Ha noi

# Quoc gia: Viet nam

# Nam: 1010

# 3.2 hình 4 : nhập dữ liệu

# Sử dụng hàm để thực hiện chức năng này theo yêu cầu dưới đây:

# Sử dụng tham biến là con trỏ cấu trúc city\_st.

# Lưu các giá trị nhập từ người sử dụng vào mảng cityList trong hàm main().

# Ta xây dựng hàm nhập như sau:

//Hàm nhap

void nhap(city\_st \*c, int n){

printf("\nBan hay nhap du lieu cua %d thanh pho \n\n", n);

for(int i=0; i<n; i++){

printf("Nhap du lieu thanh pho %d\n", i+1);

fflush(stdin);

printf("Ten: ");

gets(c[i].city\_nm);

printf("Quoc gia: ");

gets(c[i].country);

printf("Nam: ");

scanf("%d", &c[i].year);

printf("\n");

}

}

# 3.2 hình 5 : hàm nhập

# Để thực hiện yêu cầu nhập dữ liệu, trong hàm main ta thực hiện như sau:

int main(){

int n;

city\_st \*cityList;

cityList=(city\_st\*)malloc(n\*sizeof(city\_st));

printf("Nhap so thanh pho ban muon nhap: ");

scanf("%d", &n);

nhap(cityList, n);

}

# *\*Sắp xếp và hiển thị thông tin chi tiết của từng thành phố theo quốc gia (Z->A).*

# 3.2 hình 6 : nhập dữ liệu

# *\*Sắp xếp và hiển thị thông tin chi tiết của từng thành phố theo quốc gia (Z->A).*

# Sắp xếp các thành phố theo tên quốc gia từ Z-> A sau đó hiển thị thông tin chi tiết của các thành phố như dưới đây:

# STT||Ten ||Quoc gia ||Nam

# 001||Ha noi ||Viet nam ||1010

# 002||Sai gon ||Viet nam ||1698

# 003||Kyoto ||Nhat Ban ||794

# 3.2 hình 7 : sắp xếp thành phố

# Sử dụng hàm để thực hiện chức năng này theo yêu cầu sử dụng tham biến là mảng cấu trúc city\_st.

# Để có thể hiển thị thông tin ra màn hình, đầu tiên chúng ta cần xây dựng một hàm xuất như sau:

//Hàm xuat

void xuat(city\_st \*c, int n){

printf("STT||Ten ||Quoc gia ||Nam\n");

for(int i=0; i<n; i++){

if(i<=8)

printf("00%d||%-20s||%-20s||%d\n", i+1, c[i].city\_nm, c[i].country, c[i].year);

else

printf("0%d||%-20s||%-20s||%d\n", i+1, c[i].city\_nm, c[i].country, c[i].year);

}

}

# 3.2 hình 8 : hàm xuất

# Hàm sắp xếp các thành phố theo tên quốc gia từ Z đến A:

//Hàm sap xep

void sapXep(city\_st \*c, int n){

for(int i=0; i<n-1; i++){

for(int j=i+1; j<n; j++){

if(strcmp(c[i].country, c[j].country) < 0){

city\_st t;

t=c[i];

c[i]=c[j];

c[j]=t;

}

}

}

}

# 3.2 hình 9 : hàm sắp xếp

# Trong hàm main, để thực hiện yêu cầu sắp xếp và hiển thị thông tin chi tiết của từng thành phố theo quốc gia (Z->A), ta thực hiện như sau:

int main(){

sapXep(cityList, n);

printf("\nCac thanh pho sau khi sap xep theo quoc gia (Z den A):\n");

xuat(cityList, n);

}

# 3.2 hình 10

# *\*Tìm thành phố theo quốc gia*

# Hiển thị “Nhap quoc gia: ”

# Tìm và hiển thị thông tin chi tiết của tất cả các thành phố thuộc quốc gia này theo thứ tự từ Z đến A. Trong trường hợp không tìm thấy thành phố nào, hiển thị thông báo “Khong co thanh pho thuoc quoc gia nay”.

# Sử dụng hàm để thực hiện chức năng này theo yêu cầu sử dụng tham biến là mảng cấu trúc city\_st

//Hàm tim kiem

void timKiem(city\_st \*c, int n){

char quocgia[30];

int dem=0;

city\_st city\_qg[50];

printf("\nNhap quoc gia ban muon tim so thanh pho: ");

fflush(stdin);

gets(quocgia);

for(int i=0; i<n; i++){

if(strcmp(c[i].country, quocgia) == 0){

dem+=1;

city\_qg[dem-1]=c[i];

}

}

if(dem>0){

printf("Quoc gia ban nhap vao: %s co %d thanh pho\n", quocgia, dem);

xuat(city\_qg, dem);

}

else

printf("Khong co thanh pho nao thuoc quoc gia nay\n");

}

# 3.2 hình 11 : hàm tìm kiếm

# Để thực hiện yêu cầu tìm kiếm, trong hàm main ta thực hiện:

int main(){

timKiem(cityList, n);

}

# *\*Ghi vào tập tin nhị phân city.dat*

# Ghi tất cả thông tin của từng thành phố vào file nhị phân “city.dat”.

# Sử dụng hàm để thực hiện chức năng này theo yêu cầu dưới đây:

# Sử dụng tham biến là con trỏ cấu trúc city\_st.

# Tạo file nhị phân “city.dat”.

# Lưu trữ tất cả thông tin của từng thành phố vào trong file này.

# Đóng file

# Ta xây dựng hàm như sau:

//Hàm ghi vào teo nhi phan city.dat

void ghi(city\_st \*c, int n){

FILE \*fnp;

fnp=fopen("city.dat", "w");

fprintf(fnp, "STT||%-20s||%-20s||%s\n", "Ten", "Quoc gia", "Nam");

for(int i=0; i<n; i++){

if(i<=8)

fprintf(fnp, "00%d||%-20s||%-20s||%d\n", i+1, c[i].city\_nm, c[i].country, c[i].year);

else

fprintf(fnp, "0%d||%-20s||%-20s||%d\n", i+1, c[i].city\_nm, c[i].country, c[i].year);

}

fclose(fnp);

}

# 3.2 hình 12 : hàm ghi vào tệp nhị phân

# Để thực hiện yêu cầu ghi vào tệp nhị phân, trong hàm main ta thực hiện:

int main(){

ghi(cityList, n);

}

# CHƯƠNG 4 : HƯỚNG DẪN SỬ DỤNG CHƯƠNG TRÌNH

1. Nhập dữ liệu của từng thành phố.

Chương trình khi khởi chạy sẽ xuất hiện sẽ cho phép chúng ta nhập dữ liệu của từng thành phố

1. Sắp xếp, thống kê và hiển thị thông tin chi tiết của từng thành phố theo quốc gia ( Z->A).

Sau khi chúng ta nhập thông tin các thành phố xong thì chương trình sẽ tự động sắp xếp thứ tự theo thứ tự từ ( Z->A).

1. Tìm thành phố theo quốc gia.

Sau khi khởi chạy và sắp xếp thì chương trình sẽ hiển thị 1 cái menu , nếu chúng ta muốn tìm kiếm thành phố theo tên quốc gia thì ấn số 3 từ bàn phím và nhập tên quốc gia muốn tìm.

1. Ghi vào tập tin nhị phân city.dat.

Chương trình sẽ lưu dữ liệu bạn vừa nhập vào file nhị phân city.dat

1. Thoát.

Chương trình sẽ kết thúc.

# CHƯƠNG 5: KẾT LUẬN VÀ ĐỊNH HƯỚNG PHÁT TRIỂN.

## 5.1.Kết luận:

* Đã làm được:
  + Phân tích được hệ thống quản lí thành phố và xây dựng được hệ thống quản lí.
  + Cải thiện được khả năng phân tích vấn đề, giải quyết vấn đề tốt hơn.
  + Cải thiện được khả năng độc lập tìm hiểu, chắt lọc và phân tích tài liệu
* Hạn chế:
* Cách giải quyết bài toán còn ở mức độ cơ bản, chưa bao quát được tất cả các trường hợp xảy ra.
* Còn nhiều thiếu sót tỏng quá trình làm bài

## 5.2.Hướng phát triển:

Từ chương trình trên, chúng ta có thể áp dụng vào cuộc sống để phát triển tạo thành các hệ thống quản lí, ví dụ như :

* Quản lí các thành phố....
* Quản lí sinh viên...
* Quản lí số sách trong thư viện....

**5.3.Tài liệu tham khảo :**

Về lí thuyết , bọn em đã tham khảo ở:

* Phạm Văn Ất, Lập trình nâng cao, Nhà xuất bản Giao thông Vận tải, 2015.
* <https://nguyenvanhieu.vn/doc-ghi-file-trong-c/> 16/6/2020 15h48p.
* <https://nguyenvanhieu.vn/danh-sach-lien-ket-don/> 16/6/2020 19h20p.
* [https://vietjack.com/bai-tap-c/cac-thuat-toan-sap-xep-trong-c.jsp 17/6/2020](https://vietjack.com/bai-tap-c/cac-thuat-toan-sap-xep-trong-c.jsp%2017/6/2020) 9h7p