Một trong những vấn đề quan trọng bậc nhất của khoa học máy tính là tìm kiếm thông tin. Có thể nói, hầu hết các hoạt động của người dùng hoặc các ứng dụng tin học đều liên quan đến tìm kiếm. Muốn tìm kiếm thông tinh nhanh, hiệu quả, chính xác ta cần có phương pháp tổ chức và sắp xếp dữ liệu tốt. Chính vì vậy, sắp xếp được xem như giai đoạn đầu chuẩn bị cho quá trình tìm kiếm. Nội dung chương này trình bày các thuật toán sắp xếp và tìm kiếm, bao gồm: các thuật toán sắp xếp đơn giản, các thuật toán sắp xếp nhanh, các thuật toán tìm kiếm tuyến tính, tìm kiếm nhị phân, tìm kiếm nội suy & tìm kiếm Jumping.

1. Giới thiệu vấn đề

Bài toán tìm kiếm có thể được phát biểu như sau: Cho dãy gồm n đối tượng r_1, r_2, .., r_n. Mỗi đối tượng r_i được tương ứng với một khóa k_i ($1 \le i \le n$). Nhiệm vụ của tìm kiếm là xây dựng thuật toán tìm đối tượng có giá trị khóa là X cho trước. X còn được gọi là khóa tìm kiếm hay tham biến tìm kiếm (argument). Bài toán tìm kiếm bao giờ cũng hoàn thành bởi một trong hai tình huống:

- Nếu tìm thấy đối tượng có khóa X trong tập các đối tượng thì ta nói phép tìm kiếm thành công (successful).
- Nếu không tìm thấy đối tượng có khóa X trong tập các đối tượng thì ta nói phép tìm kiếm không thành công (unsuccessful).

Sắp xếp là phương pháp bố trí lại các đối tượng theo một trật tự nào đó. Ví dụ bố trí theo thứ tự tăng dần hoặc giảm dần đối với dãy số, bố trị theo thứ tự từ điển đối với các xâu ký tự. Mục tiêu của sắp xếp là để lưu trữ và tìm kiếm đối tượng (thông tin) để đạt hiệu quả cao trong tìm kiếm. Có thể nói, sắp xếp là sân sau quả quá trình tìm kiếm. Muốn tìm kiếm và cung cấp thông tin nhanh thì ta cần phải sắp xếp thông tin sao cho hợp lý. Bài toán sắp xếp có thể được phát biểu như sau:

Bài toán sắp xếp: Cho dãy gồm n đối tượng r_1, r_2, .., r_n. Mỗi đối tượng r_i được tương ứng với một khóa k_i ($1 \le i \le n$). Nhiệm vụ của sắp xếp là xây dựng thuật toán bố trí các đối tượng theo một trật tự nào đó của các giá trị khóa. Trật tự của các giá trị khóa có thể là tăng dần hoặc giảm dần tùy thuộc vào mỗi thuật toán tìm kiếm cụ thể.

Trong các mục tiếp theo, chúng ta xem tập các đối tượng cần sắp xếp là tập các số. Việc mở rộng các số cho các bản ghi tổng quát cũng được thực hiện tương tự bằng cách thay đổi các kiểu dữ liệu tương ứng. Cũng giống như tìm kiếm, việc làm này không làm mất đi bản chất của thuật toán.

2. Các thuật toán sắp xếp đơn giản

Các thuật toán sắp xếp đơn giản được trình bày ở đây bao gồm:

- Thuật toán sắp xếp kiểu lựa chọn (Selection Sort).

- Thuật toán sắp xếp kiểu chèn trực tiếp (Insertion Sort).
- Thuật toán sắp xếp kiểu sủi bọt (Bubble Sort)

2.1 Thuật toán Selection-Sort

Thuật toán sắp xếp đơn giản nhất được đề cập đến là thuật toán sắp xếp kiểu chọn. Thuật toán thực hiện sắp xếp dãy các đối tượng bằng cách lặp lại việc tìm kiếm phần tử có giá trị nhỏ nhất từ thành phần chưa được sắp xếp trong mảng và đặt nó vào vị trí đầu tiên của dãy. Trên dãy các đối tượng ban đầu, thuật toán luôn duy trì hai dãy con: dãy con đã được sắp xếp là các phần tử bên trái của dãy và dãy con chưa được sắp xếp là các phần tử bên phải của dãy. Quá trình lặp sẽ kết thúc khi dãy con chưa được sắp xếp chỉ còn lại đúng một phần tử. Thuật toán được trình bày chi tiết trong Hình 3.1.

2.1.1 Biểu diễn thuật toán

Thuật toán Selection-Sort:
Input:
- Dãy các đối tượng (các số) : Arr[0], Arr[1],..,Arr[n-1].
- Số lượng các đối tượng cần sắp xếp: n.

Output:
- Dãy các đối tượng đã được sắp xếp (các số) : Arr[0], Arr[1],..,Arr[n-1].

Formats: Selection-Sort(Arr[] , n);
Actions:

```
for (i =0; i<n-1; i++) { //duyệt các phần tử i=0,1,.., n-1
    min_idx = i;  //gọi min_idx là vị trí của phần tử nhỏ nhất trong dãy con
    for ( j = i +1; j<n; j++ ) { //duyệt từ phần tử tiếp theo j=i+1,..,n.
        if (Arr[i] > Arr[j] ) // nếu Arr[i] không phải nhỏ nhất trong dãy con
            min_idx = j; //ghi nhận đây mới là vị trí phần tử nhỏ nhất.
    }
    //đặt phần tử nhỏ nhất vào vị trí đầu tiên của dãy con chưa được sắp
    Temp = Arr[i] ; Arr[i] = Arr[min_idx]; Arr[min_idx] = temp;
}
End.
```

Hình 3.1. Thuật toán Selection Sort.

2.1.2 Độ phức tạp thuật toán

Độ phức tạp thuật toán Selection Sort là $O(N^2)$, trong đó N là số lượng phần tử cần sắp xếp. Bạn đọc tự tìm hiểu phương pháp xác định độ phức tạp thuật toán Selection Sort trong các tài liệu tham khảo liên quan.

2.1.3 Kiểm nghiệm thuật toán

Kiểm nghiệm thuật toán: Arr[] = { 9, 7, 12, 8, 6, 5 }, n = 6.

Bước	min-idx	Dãy số Arr[] =?
i = 0	min-idx=5	Arr[] = {**5**, 7, 12, 8, 6, **9**}
i = 1	min-idx=4	Arr[] = {**5**, **6**, 12, 8, 7, **9**}
i = 2	min-idx=4	Arr[] = {**5**, **6**, **7**, 8, 12, **9**}
i = 3	min-idx=3	Arr[] = {**5**, **6**, **7**, **8**, 12, **9**}
i = 4	min-idx=5	Arr[] = {**5**, **6**, **7**, **8**, **9**, 12}

2.1.4 Cài đặt thuật toán

```
#include
<iostream>
#include
<iomanip> using
namespace std;
void swap(int *x, int *y){ //đổi giá trị của x và y
        int temp = *x; *x = *y;       *y = temp;
}
void SelectionSort(int arr[], int n){ //thuật toán selection sort
        int i, j, min_idx; //min_idx là vị trí để arr[min_index] nhỏ nhất
        for (i = 0; i < n-1; i++)    { //duyệt n-1 phần tử
                min_idx = i; //vị trí số bé nhất tạm thời là i
                for (j = i+1; j < n; j++){ //tìm vị trí số bé nhất arr[i+1],.., arr[n-1]
                        if (arr[j] < arr[min_idx])
                                min_idx = j;
                }
                swap(&arr[min_idx], &arr[i]);//tráo đổi arr[min_idx] và arr[i]
        }
}
void printArray(int arr[], int size){ //hiển thị kết quả
        for (int i=0; i < size; i++)
                cout<<arr[i]<<setw(5);
        cout<<endl;
}
int main(){ //chương trình chính
        int arr[] = {64, 25, 12, 22, 11};
```

```
        int n =
        sizeof(arr)/sizeof(arr[0]);
        SelectionSort(arr, n);
        cout<<"Dãy số được sắp: \n";
        printArray(arr, n);
}
```

2.2 Thuật toán Insertion Sort

Thuật toán sắp xếp kiểu chèn được thực hiện đơn giản theo cách của người chơi bài thông thường. Phương pháp được thực hiện như sau:

- Lấy phần tử đầu tiên Arr[0] (quân bài đầu tiên) như vậy ta có dãy một phần tử được sắp.
- Lấy phần tiếp theo (quân bài tiếp theo) Arr[1] và tìm vị trí thích hợp chèn Arr[1] vào dãy Arr[0] để có dãy hai phần tử đã được sắp.
- Tổng quát, tại bước thứ i ta lấy phần tử thứ i và chèn vào dãy Arr[0],..,Arr[i-1] đã được sắp trước đó để nhận được dãy i phần tử được sắp. Quá trình sắp xếp sẽ kết thúc khi quân bài cuối cùng (i = n) được chèn đúng vị trí. Thuật toán Insertion Sort được mô tả chi tiết trong Hình 3.2.

2.2.1 Biểu diễn thuật toán

Thuật toán Insertion-Sort:

Input:

- Dãy các đối tượng (các số) : Arr[0], Arr[1],..,Arr[n-1].
- Số lượng các đối tượng cần sắp xếp: n.

Output:

- Dãy các đối tượng đã được sắp xếp (các số) : Arr[0], Arr[1],..,Arr[n-1].

Formats: Insertion-Sort(Arr, n):

Actions:

```
for (i = 1; i < n; i++) { //lặp i=1, 2,..,n.
        key = Arr[i]; //key là phần tử cần chèn vào dãy Arr[0],.., Arr[i-1]
        j = i-1;
        while (j >= 0 && Arr[j] > key) { //Duyệt lùi từ vị trí j=i-1
                Arr[j+1] = Arr[j]; //dịch chuyển Arr[j] lên vị trí Arr[j+1]
                j = j-1;
        }
        Arr[j+1] = key; // vị trí thích hợp của key trong dãy là Arr[j+1]
}
```

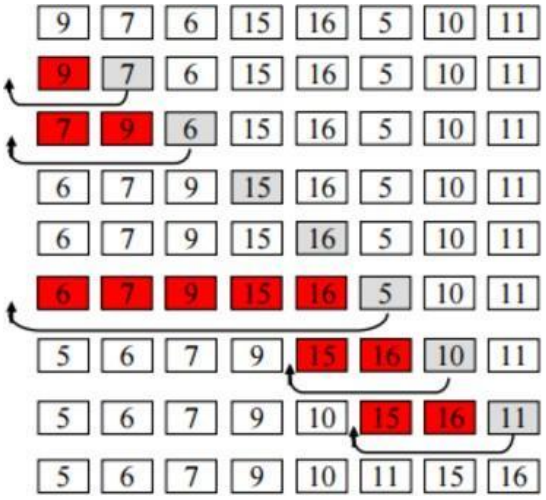
End.

Hình 3.2. Thuật toán Insertion Sort

2.2.2 Độ phức tạp thuật toán

Độ phức tạp thuật toán là $O(N^2)$, với N là số lượng phần tử. Thuật toán có thể cải tiến bằng cách sử dụng hàng đợi ưu tiên với độ phức tạp O(N.Log(N)).

2.2.3 Kiểm nghiệm thuật toán



2.2.4 Cài đặt thuật toán

```
#include <iostream>
#include <iomanip>
using namespace std;
    void insertionSort(int arr[], int
            n){ int i, key, j;
            for (i = 1; i < n; i++){
                    key =
                    arr[i]; j =
                    i-1;
                    while (j >= 0 && arr[j] > key){
                            arr[j+1] = arr[j];
                            j = j-1;
                    }
                    arr[j+1] = key;
            }
    }
    void printArray(int arr[], int n){
            int i;cout<<"\n Day so duoc sap:";
            for (i=0; i < n; i++)
```

```
            cout<<arr[i]<<setw(3);
}
int main(){
        int arr[] = {12, 11, 13, 5, 6};
        int n =
        sizeof(arr)/sizeof(arr[0]);
        insertionSort(arr, n);
        printArray(arr, n);
}
```