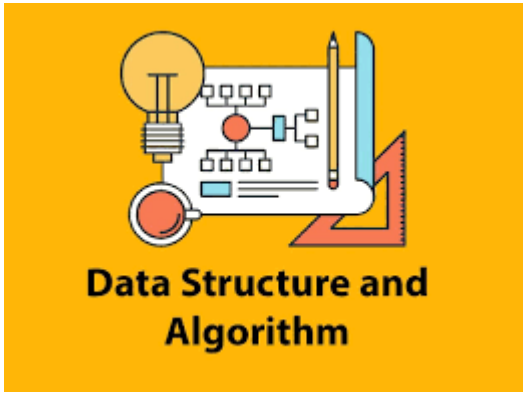
CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

NGÀY 2.2: SẮP XẾP VÀ TÌM KIẾM



Giảng viên: Th.S Bùi Văn Kiên

Nội dung

- Bài toán sắp xếp
  - Các thuật toán sắp xếp đơn giản
  - Thuật toán Radix-Sort
  - Thuật toán Quick-Sort (Học ở phần Chia để trị)
  - Thuật toán Merge-Sort (Học ở phần Chia để trị)
- Bài toán tìm kiếm:
  - Tìm kiếm tuần tự
  - Tìm kiếm nhị phân
  - Tìm kiếm với sự hỗ trợ của CTDL đặc biệt (*)

CÁC THUẬT TOÁN SẮP XẾP

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] ($1 \leq i \leq n$).
- 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.
- Để ví dụ, ta xét tập các đối tượng cần sắp xếp là tập các số.

Bài toán sắp xếp

- Các đặc trưng:
  - Ý tưởng dễ hiểu
  - Cài đặt đơn giản
  - Độ phức tạp cao

- Một số thuật toán sắp xếp đơn giản:
  - Thuật toán sắp xếp kiểu lựa chọn (Selection Sort).
  - Thuật toán sắp xếp kiểu chèn (Insertion Sort).
  - Thuật toán sắp xếp kiểu nổi bọt (Bubble Sort).

Sắp xếp chọn (Selection sort)

- Ý tưởng chính: 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.
  - 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.

Sắp xếp chọn (Selection sort)

- Input:
  - Số lượng phần tử của dãy số.
  - Dãy số chứa các phần tử: Arr[0], Arr[1],..,Arr[n-1].
- Output:
  - Dãy số sau khi được sắp xếp.

Formats: Selection-Sort(Arr, n);

Actions:

```
for (i =0; i<n-1; i++) {
    minIdx = i;
    for ( j = i +1; j<n; j++ ) {
        if (Arr[minIdx] > Arr[j] )
            minIdx = j;
     }
     temp = Arr[i] ; Arr[i] = Arr[minIdx]; Arr[minIdx] = temp;
}
```

End.

Sắp xếp chèn (Insertion sort)

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

- 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 để 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 xếp.
- Quá trình sắp xếp sẽ kết thúc khi i = n.

Sắp xếp chèn (Insertion sort)

- Input:
  - Số lượng phần tử của dãy số.
  - Dãy số chứa các phần tử:

    Arr[0], Arr[1],..,Arr[n-1].
- Output:
  - Dãy số sau khi được sắp xếp.

Formats: Insertion-Sort(Arr, n);
Actions:

```
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;
}
```

End.

Sắp xếp nổi bọt (Bubble sort)

- Thuật toán sắp xếp nổi bọt thực hiện đổi chỗ hai phần từ liền kề nhau nếu chúng chưa được sắp xếp. Thuật toán dừng khi không còn cặp nào sai thứ tự.

Formats: Bubble-Sort(Arr, n);
Actions:

```
for (i = 0; i < n; i++) {
        check = false;
        for (j=0; j<n-i-1; j++) {
                if (Arr[j] > Arr[j+1]) {
                        check = true;
                        temp = Arr[j];  Arr[j] = Arr[j+1]; Arr[j+1] = temp;
                }
        }
        if (!check) break;
}
```

End.

Học viện
Công nghệ Bưu chính Viễn thông

CÁC THUẬT TOÁN TÌM KIẾM

Bài toán tìm kiếm

- Cho dãy gồm n phần tử r[1], r[2], .., r[n]. Mỗi phần tử 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 phần tử có giá trị khóa là X cho trước.

- Kết quả:
  - Nếu tìm thấy phần tử có khóa X → phép tìm kiếm thành công.
  - Nếu không tìm thấy đối tượng có khóa X → không tồn tại

Tìm kiếm tuần tự

- Tìm kiếm phần tử có giá trị khóa X trong tập các khóa {A[1], A[2], …, A[N]}.
- Thuật toán trả lại vị trí của X trong dãy khóa A[], trả lại giá trị -1 nếu x không có mặt trong dãy khóa A[].
- Độ phức tạp của thuật toán Sequential-Search() là O(n).

```
Thuật toán Sequential-Search (int A[], int n, int x) {
        for (i =1; i <= n; i++) {
                if (x == A[i]) return (i);
        }
        return (-1);
}
```

Tìm kiếm nhị phân

- Thuật toán tìm kiếm nhị phân là phương pháp định vị phần tử X trong một danh sách A[] gồm n phần tử đã được sắp xếp.
- Chia danh sách thành hai phần. So sánh X với phần tử ở giữa.
- Có 3 trường hợp :
  - Trường hợp 1: nếu x bằng phần tử ở giữa A[mid], thì mid chính là vị trí của x trong danh sách A[].
  - Trường hợp 2: Nếu x lớn hơn phần tử ở giữa thì nếu x có mặt trọng dãy A[] thì ta chỉ cần tìm các phần tử từ mid+1 đến vị trí thứ n.
  - Trường hợp 3: Nếu x nhỏ hơn A[mid] thì x chỉ có thể ở dãy con bên trái của dãy A[].
- Lặp lại quá trình trên cho đến khi cận dưới vượt cận trên của dãy A[] mà vẫn chưa tìm thấy X → kết luận X không có mặt trong dãy A[].
- Độ phức tạp thuật toán là : O(log(n)).

Tìm kiếm nhị phân

Thuật toán thông thường

```
int Binary_Search(int A[], int n, int X) {
    int  low = 0;
    int  high = n-1, mid;
    while (low <= high) {
        mid = (low+high)/2;
        if (X > A[mid])
            low = mid + 1;
        else if (X < A[mid])
            high = mid -1;
        else
            return (mid);

        mid = (low + high)/2;
    }
    return (-1);
}
```

Ví dụ 1:: Dãy số có các phần tử riêng biệt
A = [1 2 3 4 5 6]
X = 3
Return 2

Ví dụ 2:
A = [1 3 3 3 5 6]
X = 3
Return 2

Tìm kiếm nhị phân

Lower_bound và upper_bound

```
int UpperBound_Search(int A[], int n, int X) {
    int  low = 0, high = n-1, mid;
    int  ans = -1;
    while (low <= high) {
        mid = (low+high)/2;
        if (X >= A[mid]) {
            low = mid + 1;
            ans = mid;
        }
        else if (X < A[mid]) {
            high = mid -1;
        }
    }
    return ans;
}
```

Ví dụ 2:
A = [1 3 3 3 5 6]
X = 3
Upper_bound = 3

Tìm kiếm nhị phân

Lower_bound và upper_bound

```
int LowerBound_Search(int A[], int n, int X) {
    int  low = 0,  high = n-1, mid;
    int  ans = -1;
    while (low <= high) {
        mid = (low+high)/2;
        if (X > A[mid]) {
            low = mid + 1;
        }
        else if (X <= A[mid]) {
            high = mid -1;
            ans = mid;
        }
    }
    return ans;
}
```

Ví dụ 2:
A = [1 3 3 3 5 6]
X = 3
Lower_bound = 1
Upper_bound = 3

Tìm kiếm sử dụng CTDL đặc biệt

- Cho dãy số A[] có N phần tử. Có M truy vấn, mỗi truy vấn yêu cầu kiểm tra có tồn tại phần tử có giá trị X hay không?
- Output: Yes/No với mỗi truy vấn
- Giới hạn:
  - Subtask 1: $N \le 10^6$, $M \le 100$, $A[i] \le 10^9$

    → Có thể sử dụng tìm kiếm tuần tự.
  - Subtask 2: $N \le 10^6$, $M \le 10^6$, $A[i] \le 10^6$

    → Sử dụng mảng hằng để đánh dấu
  - Subtask 3: $N \le 10^6$, $M \le 10^6$, $A[i] \le 10^9$

    → (1) Sắp xếp dãy số A[] rồi dùng tìm kiếm nhị phân,

    (2) sử dụng CTDL cây nhị phân tìm kiếm (tự xây dựng), hoặc cây tìm kiếm cân bằng (như std::set, std::map trong thư viện STL)

QUESTIONS & ANSWERS