Báo cáo Bài tập lớn
môn Cấu trúc dữ liệu và Giải thuật

Hoàng Nguyên Bảo - 20224404

Mã lớp: 152455

1 Đề tài

Lựa chọn đề tài: **Sắp xếp đếm phân phối (Counting sort).** Sắp xếp đếm phân phối là một phương pháp sắp xếp có độ phức tạp tuyến tính. Nó dựa trên giả thiết rằng, các khoá cần sắp xếp là các số tự nhiên giới hạn trong một khoảng nào đó, chẳng hạn từ 1 đến N.

Khác với các thuật toán sắp xếp so sánh thông thường như Quick Sort hay Merge Sort, Counting Sort dựa trên việc đếm số lượng phần tử và sử dụng thông tin này để xác định vị trí chính xác của từng phần tử trong mảng kết quả. Điều này giúp thuật toán đạt được độ phức tạp thời gian O(n+k), trong đó nn là số lượng phần tử và kk là giá trị lớn nhất trong mảng, khiến nó cực kỳ nhanh trong các trường hợp dữ liệu có giá trị nhỏ và phân bố trong một phạm vi hẹp.

Ngoài ra, Counting Sort là một ví dụ điển hình về thuật toán không dựa trên so sánh, giúp mở rộng cách nhìn nhận về các phương pháp sắp xếp dữ liệu. Thuật toán này còn là nền tảng cho nhiều thuật toán phức tạp hơn, như Radix Sort, và được ứng dụng rộng rãi trong các bài toán yêu cầu hiệu suất cao, như xử lý dữ liệu lớn, phân loại trong thời gian thực, hoặc các bài toán liên quan đến sắp xếp số nguyên.

2 Phân tích bài toán

2.1 Đầu vào (Input):

Dữ liệu đầu vào được tổng quát hóa như sau:

- Một mảng dữ liệu ban đầu gồm các số tự nhiên
- Số lượng phần tử là n
- Giá trị lớn nhất trong mảng là 8 (được truyền vào làm tham số **maxVal**)

2.2 Đầu ra (Output):

- Mảng kết quả sau khi thực hiện sắp xếp sử dụng thuật toán Counting Sort với các tiêu chí mặc định như sau:
  - **Đúng thứ tự tăng dần:** Các phần tử trong mảng được sắp xếp theo thứ tự từ nhỏ đến lớn.
  - **Bảo toàn số lượng phần tử:** Tổng số phần tử trong mảng đầu ra bằng tổng số phần tử trong mảng đầu vào (n phần tử).

3 Xây dựng thuật toán

3.1 Ý tưởng chính:

Ý tưởng sơ bộ của thuật toán được trình bày như sau:

- Đếm số lần xuất hiện của mỗi giá trị trong mảng đầu vào.
- Sử dụng mảng đếm để xác định vị trí chính xác của từng phần tử trong mảng kết quả.
- Tạo mảng kết quả đã sắp xếp dựa trên mảng đếm.

3.2 Giải thích thuật toán:

Các bước thực hiện của thuật toán Counting Sort được trình bày như sau:

- **Bước 1: Khởi tạo mảng đếm**
  - Tạo một mảng đếm count[] có kích thước k+1 (tương ứng với tất cả các giá trị từ 0đến k).
  - Ban đầu, tất cả các phần tử trong mảng count[] được khởi tạo bằng 0.
- **Bước 2: Đếm số lần xuất hiện của từng giá trị**
  - Duyệt qua từng phần tử trong mảng đầu vào arr[]
  - Với mỗi phần tử arr[i], tăng giá trị tại vị trí count[arr[i]] lên 1
  - Sau bước này, count[x] sẽ chứa số lần xuất hiện của giá trị x trong mảng đầu vào
- **Bước 3: Cộng dồn mảng đếm**
  - Biến đổi mảng count[] để chứa thông tin về vị trí cuối cùng của mỗi giá trị trong mảng đã sắp xếp.
  - Cụ thể, tính tổng cộng dồn cho mảng count[] theo phép toán như sau: count[i]=count[i]+count[i-1].

- Sau bước này: count[x] sẽ cho biết vị trí cuối cùng của giá trị x trong mảng đã sắp xếp.

- **Bước 4: Xây dựng mảng kết quả**
  - Tạo một mảng kết quả output[] có cùng kích thước với mảng đầu vào.
  - Duyệt ngược qua mảng đầu vào arr[] (để đảm bảo tính ổn định):
    * Với mỗi phần tử arr[i], sử dụng giá trị trong count[arr[i]] để xác định vị trí của nó trong output[].
    * Sau khi đặt phần tử vào mảng output[], giảm giá trị count[arr[i]] đi 1.

- **Bước 5: Sao chép mảng kết quả**
  - Sao chép tất cả các phần tử từ output[] trở lại mảng đầu vào arr[] để hoàn thành quá trình sắp xếp.

3.3 Lưu đồ thuật toán

Chương trình hoạt động theo lưu đồ thuật toán như sau:

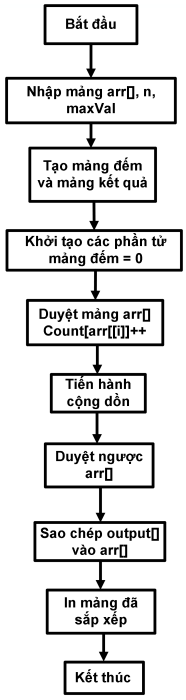

Figure 1: Lưu đồ thuật toán

Các bước trong lưu đồ thuật toán được trình bày như sau:

- Bắt đầu chương trình
- Nhập mảng và xác định số phần tử (n) cùng giá trị lớn nhất (maxVal).
- Khởi tạo mảng đếm (count[]) và mảng kết quả (output[]).
- Đếm số lần xuất hiện của từng phần tử trong arr[] và lưu vào count[].
- Cộng dồn count[] để xác định vị trí cuối của mỗi giá trị trong mảng sắp xếp.
- Duyệt ngược arr[], đặt phần tử vào vị trí đúng trong output[], đồng thời giảm count[].
- Sao chép output[] vào arr[] để hoàn thành sắp xếp.
- In mảng đã sắp xếp.
- Kết thúc chương trình.

3.4 Ví dụ minh họa

Cho mảng đầu vào như sau: arr[] = 4, 2, 2, 8, 3, 3, 1 Làm theo từng bước như phân tích ở trên, ta được:

- **Bước 1: Khởi tạo mảng đếm**
  - Giá trị lớn nhất k=8, tạo mảng count[] với kích thước 9 (từ 0 đến 8): count[] = 0, 0, 0, 0, 0, 0, 0, 0, 0
- **Bước 2: Đếm số lần xuất hiện**
  - Đếm số lần xuất hiện của mỗi giá trị trong arr[]: count[] = 0, 1, 2, 2, 1, 0, 0, 0, 1
- **Bước 3: Cộng dồn mảng đếm**
  - Tính tổng cộng dồn cho count[] để chứa thông tin về vị trí cuối cùng của mỗi giá trị trong mảng đã sắp xếp.: count[] = 0, 1, 3, 5, 6, 6, 6, 6, 7
- **Bước 4: Xây dựng mảng kết quả**
  - Duyệt ngược mảng arr[] và sử dụng count[] để sắp xếp: output[] = 1, 2, 2, 3, 3, 4, 8
- **Bước 5: Sao chép mảng kết quả**
  - Sao chép output[] trở lại arr[] ta được mảng kết quả: arr[] = 1, 2, 2, 3, 3, 4, 8

3.5 Mã giả

Chương trình được trình bày theo mã giả như sau:

```
CountingSort(arr, n, maxVal)
    count[maxVal + 1], count[i] = 0
    outpu[n]

    // Đếm số lần xuất hiện của từng phần tử trong arr
    i từ 0 đến n - 1
        count[arr[i]] = count[arr[i]] + 1
    // Cộng dồn mảng count để xác định vị trí của phần tử
    i từ 1 đến maxVal
        count[i] = count[i] + count[i - 1]
    // Sắp xếp các phần tử vào output theo thứ tự đã tính toán
    i từ n - 1 đến 0
        output[count[arr[i]] - 1] = arr[i]
        count[arr[i]] = count[arr[i]] - 1
    // Sao chép output vào arr để trả về kết quả đã sắp xếp
    i từ 0 đến n - 1
        arr[i] = output[i]

    PrintArray(arr, n)
    i từ 0 đến n - 1
        print arr[i]
    Main()
    Khởi tạo mảng arr (theo ví dụ) = [4, 2, 2, 8, 3, 3, 1]
    n = số lượng phần tử trong arr
    maxVal = 8 // Giá trị lớn nhất trong mảng
    PrintArray(arr, n)
    CountingSort(arr, n, maxVal)
    PrintArray(arr, n)
    Kết thúc chương trình
```

Figure 2: Mã giả

4 Cài đặt chương trình

Chương trình được cài đặt theo mã như sau:

```c
#include <stdio.h>
#include <string.h>

// Hàm Counting Sort
void countingSort(int arr[], int n, int maxVal) {
    // Tạo mảng đếm (count) có kích thước maxVal + 1 và mảng kết quả (output) có kích thước n
    int count[maxVal + 1];
    int output[n];
    // Khởi tạo tất cả phần tử trong mảng count[] về 0
    memset(count, 0, sizeof(count));
    // Đếm số lần xuất hiện của từng phần tử trong mảng arr[]
    for (int i = 0; i < n; i++) {
        count[arr[i]]++;  // Tăng giá trị count[arr[i]] lên 1
    }
    // Cộng dồn mảng count để xác định vị trí cuối cùng của mỗi giá trị trong mảng sắp xếp
    for (int i = 1; i <= maxVal; i++) {
        count[i] += count[i - 1];  // count[i] chứa vị trí cuối cùng của phần tử i trong output[]
    }
    // Duyệt ngược mảng arr[] để đặt giá trị vào vị trí đúng trong output[], đảm bảo tính ổn định
    for (int i = n - 1; i >= 0; i--) {
        output[count[arr[i]] - 1] = arr[i];  // Đặt phần tử vào vị trí đúng
        count[arr[i]]--;  // Giảm giá trị count[arr[i]] để xử lý phần tử trùng nhau
    }
    // Sao chép mảng output[] đã sắp xếp vào arr[] để trả kết quả về
    for (int i = 0; i < n; i++) {
        arr[i] = output[i];
    }
}
// Hàm hiển thị mảng
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int main() {
    // Khởi tạo mảng đầu vào
    int arr[] = {4, 2, 2, 8, 3, 3, 1};
    int n = sizeof(arr) / sizeof(arr[0]);  // Tính số lượng phần tử trong mảng
    int maxVal = 8; // Giá trị lớn nhất trong mảng (cần biết trước)
    printf("Mảng ban đầu:\n");
    printArray(arr, n);
    // Gọi hàm Counting Sort để sắp xếp mảng
    countingSort(arr, n, maxVal);
    printf("Mảng sau khi sắp xếp:\n");
    printArray(arr, n);
}
```

Figure 3: Cài đặt chương trình

4.1 Hàm countingSort()

- **Chức năng:** Sắp xếp mảng arr có n phần tử theo thuật toán Counting Sort.
- Các bước thực hiện:
  - Tạo mảng đếm count[] có kích thước maxVal + 1, khởi tạo toàn bộ giá trị bằng 0.
  - Đếm số lần xuất hiện của từng phần tử trong arr và lưu vào count[].

- Cộng dồn count[] để xác định vị trí thực tế của từng phần tử trong mảng đã sắp xếp.
- Duyệt ngược arr để đặt các phần tử vào mảng kết quả output[].
- Sao chép output[] vào arr để cập nhật kết quả sắp xếp.

4.1.1 Hàm printArray()

- **Chức năng:** In nội dung mảng arr gồm n phần tử ra màn hình.
- Các bước thực hiện:
  - Duyệt qua từng phần tử của arr.
  - In từng phần tử, cách nhau bởi dấu cách.
  - Xuống dòng sau khi in hết mảng.

4.1.2 Hàm main()

- **Chức năng:** Điều khiển chương trình, gọi các hàm để thực hiện sắp xếp và hiển thị kết quả.
- Các bước thực hiện:
  - Khởi tạo mảng arr cần sắp xếp.
  - Xác định số lượng phần tử n và giá trị lớn nhất maxVal.
  - In mảng ban đầu.
  - Gọi countingSort() để sắp xếp mảng.
  - In mảng sau khi sắp xếp.
  - Kết thúc chương trình.

5 Độ phức tạp và thời gian

Thuật toán **Counting Sort** có các bước chính như sau:

- Tạo mảng đếm $\rightarrow$ O(k)
- Duyệt qua mảng đầu vào để đếm số lần xuất hiện của từng phần tử $\rightarrow$ O(n)
- Cộng dồn mảng đếm để xác định vị trí cuối cùng của từng phần tử $\rightarrow$ O(k)
- Sắp xếp lại các phần tử vào mảng kết quả $\rightarrow$ O(n)
- Sao chép mảng kết quả vào mảng gốc $\rightarrow$ O(n)

Với:

- n là số lượng phần tử trong mảng
- k là giá trị lớn nhất trong mảng (maxVal)

5.1 Độ phức tạp thời gian

- Trường hợp tốt nhất (Best case): O(n+k)
- Trường hợp trung bình (Average case): O(n+k)
- Trường hợp xấu nhất (Worst case): O(n+k)

5.2 Độ phức tạp không gian

- Mảng đếm có kích thước k+1 $\rightarrow$ O(k)
- Mảng đầu ra (output) có kích thước n $\rightarrow$ O(n)
- **Tổng cộng: O(n+k)**

5.3 Nhận xét

- **Ưu điểm:** Rất nhanh khi kk không quá lớn so với nn, ổn định, không so sánh trực tiếp các phần tử.
- **Nhược điểm:** Không hiệu quả khi phạm vi giá trị kk quá lớn so với n, vì cần bộ nhớ O(k).
- Counting Sort có độ phức tạp tuyến tính O(n+k), nhanh hơn các thuật toán sắp xếp so sánh O(n log(n)) khi k nhỏ hơn n log(n).
- **Ứng dụng:** Phù hợp cho dữ liệu có giá trị nguyên trong một phạm vi nhỏ, như xử lý màu sắc, phân loại, và sắp xếp trong các hệ thống nhúng.