BÀI TẬP TUẦN 5

Môn học: Cấu Trúc Dữ Liệu và Giải Thuật.
GV hướng dẫn thực hành: Nguyễn Khánh Toàn (ktoan271199@gmail.com).
Nội dung chính:

- Các thuật toán sắp xếp với thời gian chạy trung bình là $O(nlogn)$ và $O(n)$ với n là kích thước của mảng.

Thời gian thực hiện: **1** tuần.

Bài tập được biên soạn lại và tham khảo từ tài liệu thực hành môn Cấu Trúc Dữ Liệu và Giải Thuật của quý Thầy Cô Trợ Giảng khóa trước của bộ môn Khoa Học Máy Tính. Trân trọng cảm ơn quý Thầy Cô của bộ môn.

1 Cài đặt và so sánh các thuật toán sắp xếp

1.1 Sắp xếp vun đống (heap sort)

Ý tưởng của **sắp xếp vun đống (heap sort)** dựa trên thuật toán **sắp xếp chọn (selection sort**), ý tưởng là ở mỗi bước chọn được phần tử nhỏ nhất rồi còn có thông tin gì bổ sung nữa không, chẳng hạn thông tin này có thể giúp mình chọn phần tử nhỏ nhất tiếp theo nhanh hơn? Bước đầu thuật toán sẽ xây dựng min-heap hoặc max-heap. Sau đó hoán đổi phần tử min hoặc max với phần tử đầu hoặc cuối và thực hiện xây dựng lại heap trên mảng còn lại. Với ý tưởng này các bạn hãy cài đặt hoàn chỉnh thuật toán sắp xếp vun đống (heap sort).

1.2 Sắp xếp trộn (merge sort)

Thuật toán **sắp xếp trộn (merge sort)** lấy ý tưởng từ thuật toán *chia để trị*, chia mảng làm đôi, sắp xếp 2 phần con đó rồi trộn chúng lại với nhau. Khi trộn các phần mảng đã được sắp xếp với nhau, các bạn sử dụng kĩ thuật **two-pointer** đã được giới thiệu ở bài tập tuần 1 để tối ưu hóa quá trình trộn.

Dưới đây là hàm trộn hai mảng đã sắp xếp với nhau:

```
/**
Hàm trộn 2 mảng con arr[l..mid] và arr[mid+1..r] thành mảng arr[l..r] tăng dần
**/
void Merge(int arr[], int l, int mid, int r) {
    int i = l;              // vị trí bắt đầu của mảng con trái
    int j = mid + 1;        // vị trí bắt đầu của mảng con phải
    int size = r - l + 1;   // số phần tử trong mảng kết quả
    int temp[size];         // mảng tạm, dùng để lưu kết quả trộn của 2 mảng con
    // với mỗi vị trí trong mảng kết quả, chọn phần tử nhỏ hơn trong 2 phần tử đầu của 2
mảng con để đặt vô vị trí đó
    for (int k = 0; k < size; k++) {
        if (i <= mid && (j ... r || arr[i] ... arr[j])) {
            temp[k] = arr[i];
            ...
        }
        else {
            ...
            j++;
        }
    }
    copy(temp, temp + size, arr + l); // copy mảng tạm sang mảng chính
}
```

Hình 1: Trộn hai mảng đã được sắp xếp

Hàm sắp xếp trộn sẽ chia mảng gốc thành hai mảng con và thực hiện hàm trộn ở trên như sau:

```
/**
Hàm sắp xếp sử dụng MergeSort
Sắp xếp các phần tử của mảng arr từ vị trí l cho tới vị trí r
**/
void MergeSort(int arr[], int l, int r) {
    if (...) // không cần sắp xếp mảng có ít hơn 2 phần tử
        return;
    int mid = (l + r) / 2;
    MergeSort(arr, l, ...);       // mảng con trái là phần từ l -> mid
    MergeSort(arr, ..., r);       // mảng con phải là phần từ mid+1 -> r
    Merge(arr, l, mid, r);
}
```

Hình 2: Cài đặt hàm sắp xếp trộn

Với những gợi ý trên, các bạn hoàn thành mã nguồn của mình cho thuật toán sắp xếp trộn.

1.3 Sắp xếp nhanh (quick sort)

Thuật toán Quick Sort dựa trên ý tưởng lặp đi lặp lại việc phân hoạch mảng thành 2 phần bé hơn và lớn hơn một giá trị mốc (**pivot**) rồi lại tiếp tục phân

hoạch 2 phần đó. Trong bài này bạn hãy cài đặt lại thuật toán Quick Sort bằng cách điền phần còn thiếu vào đoạn code mẫu sau:

```
// Hàm sắp xếp mảng arr từ vị trí l tới vị trí r
void QuickSort(int arr[], int l, int r) {
    if (l >= r)
        return;
    int x = arr[...]; // chọn phần tử chính giữa làm phần tử mốc
    int i = l, j = r;
    while (i <= j) {
        while (...) i++;    // tìm phần tử >= x nhưng nằm bên trái
        while (...) j--;    // tìm phần tử <= x nhưng nằm bên phải
        if (i <= j) {       // hoán đổi 2 phần tử vừa tìm được với nhau
            swap(arr[i], arr[j]);
            i++;
            j--;
        }
    }
    QuickSort(...);
    QuickSort(...);
}
```

Hình 3: Cài đặt hàm sắp xếp nhanh

Lưu ý rằng trong trường hợp tệ nhất, thuật toán Quick sort sẽ có độ phức tạp là $O(n^2)$. Một ví dụ là khi các bạn chọn pivot là phần tử cuối cùng và mảng đang được sắp xếp theo thứ tự ngược lại.

1.4 Sắp xếp đếm (counting sort)

Thuật toán **sắp xếp đếm (counting sort)** không thuộc nhánh phương pháp sắp xếp có sử dụng so sánh như các phương pháp đã học. Thay vào đó người ta sẽ sử dụng một mảng phụ để đếm toàn bộ số lần xuất hiện của các giá trị trong mảng rồi từ đó sắp xếp lại mảng cho đúng thứ tự. Nhược điểm của nó là sẽ cần sử dụng một số lượng mảng phụ rất lớn khi giá trị lớn nhất của mảng lớn, ví dụ với mảng sau 1 2 3 10000000000. Các bạn hãy cài đặt thuật toán sắp xếp đếm.

1.5 Sắp xếp theo cơ số (radix sort)

Khắc phục nhược điểm trên của sắp xếp đếm, thuật toán **sắp xếp theo cơ số (radix sort)** sẽ sắp xếp các phần tử theo thứ tự tương ứng với từng hàng chữ số theo thứ tự từ bé đến lớn trong biểu diễn của nó theo một cơ số nào đó (ví dụ cơ số 2 hệ nhị phân và cơ số 10 hệ thập phân). Các bạn lựa chọn cơ số phù hợp (thường là 2 hoặc 10) rồi thực hiện cài đặt thuật toán sắp xếp theo cơ số.

Lưu ý: Đối với mảng trong các bài tập trên các bạn có thể sử dụng *vector*, tuy nhiên ở các bài tập không được sử dụng các thư viện sắp xếp có sẵn.

Sau bài tập này, các bạn nên làm bảng thống kê về các thuật toán sắp xếp đã học với các thông tin như: độ phức tạp tệ nhất, trung bình và tốt nhất (tự đưa ra được ví dụ cho mỗi trường hợp); có thỏa mãn tính ổn định và tính in-place không? Tính in-place tạm dịch là tính tại chỗ, tức là không dùng thêm bộ nhớ quá nhiều để thực hiện sắp xếp (thường là quy ước nhỏ hơn $O(N)$ về mặt bộ nhớ với N là kích thước mảng).

Hết