Báo cáo kết quả thực nghiệm các thuật toán sắp xếp

Trần Tuấn Khoa

Ngày 12 tháng 3 năm 2023

Mã Số Sinh Viên: 22025692
Lớp: KHTN2022
Tất cả những file chứa mã và tài liệu liên quan đều được chứa trên Github tại:

1 Giới thiệu

Sắp xếp là một trong những giải thuật cơ bản mà bất kỳ lập trình viên nào cũng cần phải biết. Bài báo cáo thực hành IT003 sẽ thực nghiệm một số thuật toán sắp xếp khác nhau và so sánh kết quả chúng để đưa ra cái nhìn tổng quan nhất về mỗi loại thuật toán.

2 Mô tả các thuật toán được đánh giá

2.1 Quick Sort

2.1.1 Khái niệm

Thuật toán nhanh sắp xếp dựa trên việc phân chia mảng dữ liệu thành các nhóm mảng nhỏ hơn, có phương pháp tiếp cận giống chia để trị. Giải thuật chia mảng ra thành 2 phần nhỏ hơn bằng cách so sánh các phần tử đó với một chốt (pivot) là một phần tử có sẵn trong dãy. Một dãy gồm các phần tử lớn hơn chốt và một dãy gồm các phần tử nhỏ hơn chốt.
Thuật toán này sắp xếp dữ liệu nhanh hơn so với nhiều thuật toán sắp xếp truyền thống khác nhưng lại không mấy ổn định vì tùy theo cách chọn khóa (pivot) của người cài đặt.

2.1.2 Ý tưởng

- Thuật toán Quick Sort được thực hiện theo các bước như sau:

1. Nếu độ dài mảng nhỏ hơn hoặc bằng 1 thì kết thúc hàm thực thi hiện tại.
2. Chọn một khóa *pivot* trong đoạn $[l, r]$ cần sắp xếp, gọi vị trí của phần tử này là id.
3. Khai báo 2 con trỏ *pleft* và *pright*, *pleft* chạy từ l về sau mảng, *pright* từ r về trước mảng, lưu ý điều kiện $pleft \leq pright$ luôn thỏa mãn.
4. Thực hiện vòng lặp sau: Tìm vị trí phần tử lớn hơn hoặc bằng vị trí *pleft* lớn hơn *pivot*, tìm vị trí phần tử nhỏ hơn hoặc bằng vị trí *pright* nhỏ hơn pivot, hoán đổi giá trị của 2 vị trí đó trong mảng.
5. Sau khi kết thúc vòng lặp ở bước 4, gọi đệ quy 2 dãy từ $[l, id-1]$ và $[id+1, r]$ và tiếp tục thực hiện đến hết chương trình.

- Đánh giá:

- Độ phức tạp: nếu ta chọn khóa (*pivot*) một cách ngẫu nhiên thì độ phức tạp trung bình của thuật toán là $O(n * log(n))$. Trường hợp tệ nhất của thuật toán là $O(n^2)$, tuy nhiên ta sẽ không bao giờ gặp trường hợp đó.

- Bộ nhớ tốn: $O(n)$.

Ưu điểm

- Chạy nhanh (nhanh nhất trong các thuật toán sắp xếp dựa trên việc so sánh phần tử).

Nhược điểm

- Phụ thuộc vào cách chọn khóa mà sẽ cho ra thời gian chạy khác nhau (Thông thường sẽ chọn khóa ngẫu nhiên).
- Không ổn định.

2.1.3 Cài đặt

```
void QuickSort(float *a, int l, int r) {
    int pleft = l, pright = r;
    //chọn khóa random bất kỳ trong đoạn từ [l, r - 1]
    float pivot = a[l + rand() % (r - l)];
    //thực hiện vòng lặp
    while (pleft <= pright) {
        //tìm vị trí phần tử lớn hơn hoặc bằng vị trí pleft lớn hơn pivot
        while (a[pleft] < pivot) ++pleft;
        //tìm vị trí phần tử nhỏ hơn hoặc bằng vị trí pright nhỏ hơn pivot
        while (a[pright] > pivot) --pright;
        if (pleft <= pright) {
            //hoán đổi giá trị tại hai vị trí đó
                swap(a[pleft], a[pright]);
                ++pleft;
                --pright;
        }
    }
    //gọi đệ quy thực hiện sắp xếp 2 dãy con [l, pright] và [pleft, r]
    if (l < pright) QuickSort(a, l, pright);
    if (pleft < r) QuickSort(a, pleft, r);
}

```

2.2 Merge Sort

2.2.1 Khái niệm

Merge Sort là thuật toán sắp xếp trộn theo phương pháp tiếp cận chia để trị. bằng cách chia dãy thành 2 dãy con nhỏ hơn có độ dài xấp xỉ nhau và liên tục chia nhỏ chúng ra. Thuật toán giải quyết từ những dãy con nhỏ và sau đó gộp chúng lại và sắp xếp cho dãy lớn hơn.
Merge Sort là một trong thuật toán phổ biết khi lập trình về các bài toán sắp xếp vì tính hiệu quả và dễ cài đặt. Ngoài ra, Merge Sort còn được dùng kết hợp các thuật toán khác để nâng tính hiệu suất trong quá trình sắp xếp.

2.2.2 Ý tưởng

- Thuật toán Merge Sort thực hiện theo các bước như sau:

1. Nếu dãy hiện tại có độ dài là 1 thì trở lại hàm trước.
2. Giả sử dãy hiện tại là đoạn $[l, r]$. Gọi đệ quy 2 dãy con của dãy hiện tại.
3. Sau khi sắp xếp 2 hai dãy con, ta gộp chúng lại vào dãy con hiện tại. Để gộp lại, ta làm như sau:
    - Tạo một dãy mới để lưu các phần sắp xếp vào.

- So sánh 2 phần tử đầu tiên của 2 dãy con, phần tử nào nhỏ hơn thì ta bỏ vào dãy mới đã được tạo.
- Tiếp tục thực hiện như vậy đến khi hết dãy

4. Quay về dãy trước và tiếp tục thực hiện đến khi chạy hết chương trình.

- Đánh giá:

• Độ phức tạp: Do thuật toán chia dãy làm 2 liên tục nên sẽ luôn có độ phức tạp ổn định là O(n ∗ log(n)) trong mọi trường hợp.

• Bộ nhớ: Do lưu trữ thêm một mảng để lưu lại các phần tử sắp xếp nên sẽ chiếm bộ nhớ là O(n)

Ưu điểm

• Thuật toán chạy nhanh.

• Ổn định với mọi trường hợp.

Nhược điểm

• Tốn thêm 1 mảng lưu trữ.

2.2.3 Cài đặt

```
float b[1000001]; //khai báo một mảng lưu trữ

void MergeSort(float *a, int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    //gọi đệ quy 2 mảng con [l, mid] và [mid + 1, r]
    MergeSort(a, l, mid);
    MergeSort(a, mid + 1, r);
    int pl = l, pr = mid + 1, cur = l - 1;
    while (pl <= mid || pr <= r) {
        ++cur;
        //nếu một trong 2 mảng rỗng thì bỏ dãy còn lại vào mảng
        if (pl > mid) b[cur] = a[pr++];
        else if (pr > r) b[cur] = a[pl++];
        else {
            //thực hiện so sánh 2 phần tử đầu tiên của 2 dãy
            if (a[pl] > a[pr]) b[cur] = a[pr++];
            else b[cur] = a[pl++];
        }
    }
    //gán lại vào mảng đang thực hiện sắp xếp
    for (int i = l; i <= r; i++) a[i] = b[i];
}
```

2.3 Heap Sort

2.3.1 Khái niệm

Heap Sort là thuật toán sắp xếp sử dụng cấu trúc dữ liệu Heap. Heap là cấu trúc dữ liệu dựa trên dạng cây nhị phân hoàn chỉnh thỏa mãn tính chất mỗi nút trên cây đều chứa một nhãn lớn hơn hoặc bằng các con của nó (nếu có) và nhỏ hơn hoặc bằng nút cha (trừ nút gốc là và nó là nút lớn nhất). Heap Sort thường được sử dụng nhiều vì có tốc độ nhanh và cài đặt dễ.

2.3.2 Ý tưởng

- Thuật toán Heap Sort hoạt động dựa trên các hoạt động sau đây:

1. Xây dựng một Heap chứa tất cả các phần tử có trong mảng cần sắp xếp.

2. Thực hiện một vòng lặp từ cuối chạy lên đầu và lần lượt gán phần tử tương ứng như sau:

• Lấy giá trị của nút gốc của Heap ra, đó là giá trị cần gán.
• Giảm kích thước cây khung đi 1.
• Nối 2 nhánh cây con (nếu có) sao cho phần tử có giá trị lớn làm gốc.
• Lặp lại quá trình này đến khi Heap không còn phần tử nào.

- Đánh giá:

• Độ phức tạp: Thuật toán sử dụng cấu trúc dữ liệu Heap nên có chi phí rất lớn và không ổn định. Theo lý thuyết, Heap Sort sẽ có độ phức tạp khi chạy là $O(n * log(n))$, tuy nhiên do có chi phí hằng rất lớn nên thời gian chạy có thể chậm hơn.
• Bộ nhớ: O(1) do cấu trúc dữ liệu Heap được sử dụng dựa trên mảng có sẵn nên không tốn thêm chi phí lưu trữ.

Ưu điểm

• Chạy nhanh.
• Cài đặt đơn giản nếu có sẵn thư viện Heap.

Nhược điểm

• Không ổn định.
• Có chi phí hằng rất lớn.

2.3.3 Cài đặt

```cpp
void heapify(float *a, int n, int i) {
    // khởi tạo gốc Heap và con trái, con phải của gốc
    int normalsizest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    // tìm kiếm con lớn hơn nhiều so với gốc
    if (l < n && a[l] > a[largest]) largest = l;
    if (r < n && a[r] > a[largest]) largest = r;
    // nếu giá trị lớn nhất không phải là gốc
    if (largest != i) {
        swap(a[i], a[largest]);
        // đệ quy xây dựng lại Heap
        heapify(a, n, largest);
    }
}

void HeapSort(float *a, int l, int r) {
    int n = r - l + 1;
    // xây dựng cây heap
    for (int i = n / 2 - 1; i >= 0; i--) heapify(a, n, i);
    // sắp xếp lại dãy
    for (int i= n - 1; i > 0; i--) {
        // gán phần tử lớn nhất theo vị trí giảm dần
        swap(a[0], a[i]);
```



```cpp
        heapify(a, i, 0);
    }
}
```

2.4 Hàm sort() của C++

2.4.1 Khái niệm

C++ có thư viện hỗ trợ hàm sort sẵn cho lập trình viên: std::sort(). Hàm sort() của C++ sử dụng thuật toán Intro Sort, là một thuật toán được kết hợp từ nhiều thuật toán sắp xếp khác để tăng hiệu suất, giảm thời gian chạy như Quick Sort, Heap Sort and Insertion Sort. Đây là thuật toán được sử dụng nhiều nhất vì tính nhanh gọn của hàm C++ được viết sẵn.

2.4.2 Đánh giá

• Độ phức tạp: Thời gian chạy trung bình là O(n ∗ log(n)).

• Bộ nhớ: O(1)

Ưu điểm

• Tốc độ chạy nhanh.

• Dễ dàng cài đặt.

Nhược điểm

• Không ổn định.

2.4.3 Cài đặt

sort (a,a+n);

3 Thực Nghiệm Đánh giá

3.1 Chuẩn bị dữ liệu

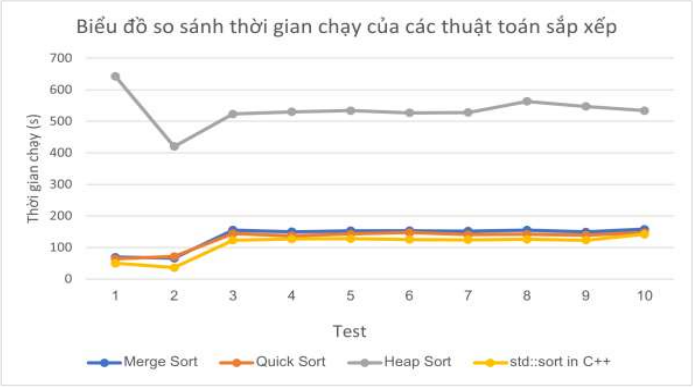
Dữ liệu chuẩn bị bao gồm 10 test case khác nhau và code chạy tính thời gian được viết bằng ngôn ngữ C++. Cụ thể hơn, việc chuẩn bị bao gồm:

1. 10 bộ test case khác nhau gồm 1 triệu số thực dạng float được sinh ngẫu nhiên, trong đó có:
   - Test case 1 là dãy gồm các số thực có giá trị tăng dần.
   - Test case 2 là dãy gồm các số thực có giá trị giảm dần.
   - 8 test case còn lại gồm các thực sinh ngẫu nhiên.
2. 1 đoạn code được viết bằng ngôn ngữ C++ bao gồm các thuật toán
   - Quick Sort
   - Heap Sort
   - Merge Sort
   - std::sort()

3.2 Kết quả thực ngiệm

Sau khi chạy thực nghiệm các thuật toán, đây là kết quả thống kê được:

Test	Merge Sort	Quick sort	Heap Sort	Sort
1	68	64	643	50
2	66	71	420	36
3	150	136	527	122
4	152	137	522	127
5	154	135	523	125
6	155	136	520	126
7	156	134	529	122
8	152	133	527	123
9	158	133	529	122
10	159	131	527	120



3.3 Nhận xét

- Thuật toán Heap Sort có thời gian chạy lâu hơn so với các thuật toán khác do bị tốn chi phí hằng số lớn. Khi mảng sắp xếp tăng dần thì việc xây dừng cấu trúc dữ liệu Heap trở nên tốn thời gian hơn do các phần tử trong Heap liên tục bị đổi gốc. Ngược lại khi dãy sắp xếp giảm dần thì thuật toán chạy hiêu quả và nhanh hơn.

- Các thuật toán khác đều chạy khá ổn định. Đặc biệt là hàm sort của thư viện chuẩn C++ có tốc độ chạy nhanh nhất bởi lẽ đã được tối ưu bằng việc kết hợp nhiều thuật toán khác nhau. Thế nhưng, thời gian chênh lệch giữa các test sinh ngẫu nhiên là khá nhiều vì không ổn định.

-Merge Sort chạy ổn định và không có chênh lệch thời gian nhiều sau mỗi test case được sinh ngẫu nhiên.

-Quick Sort chạy ổn định nhưng vẫn có vài chênh lệch. Bởi lẽ Quick Sort chọn khóa bằng phương pháp chọn ngẫu nhiên trong đoạn nên không ổn định so với Merge Sort.

3. 4 Kết Luận

Thông qua việc thực nghiệm các thuật toán sắp xếp, chúng ta đã hiểu rõ hơn về cách hoạt động, thời gian chạy của các thuật toán theo từng trường hợp khác nhau và có thể rút ra những nhận xét về chúng. Qua đó, tùy theo mục đích và nhu cần sử dụng mà ta sẽ cài thuật toán phù hợp với chương trình của mình