ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA CÔNG NGHỆ THÔNG TIN
——————oOo——————

IT003 - Cấu Trúc Dữ liệu Và Giải Thuật

So sánh các phương pháp sắp xếp phổ biến Heap Sort - Merge Sort - Quick Sort

Sinh viên : *Trần Nhật Khoa - 22520591*

Thầy hướng dẫn : *Nguyễn Thanh Sơn*

THÀNH PHỐ HỒ CHÍ MINH

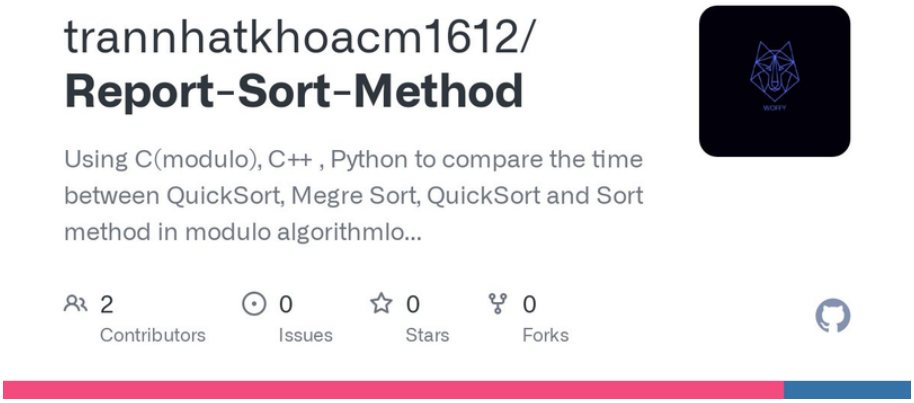
3 - 2023

Mục lục

1 Lời nói đầu

Trong bài báo cáo sau đây em xin phép bỏ qua ý tưởng của các thuật toán sắp xếp mà chỉ tập trung vào hiệu quả cũng như trường hợp ứng dụng thực tế của các phương pháp.

Xin người bạn đọc tài nguyên trên Github của toàn bộ bài viết:
`https://github.com/trannhatkhoacm1612/Report-Sort-Method`



Chú ý: Hãy đọc kĩ file ***Readme.md*** *để hiểu rõ cách bố trí sắp xếp tài nguyên.*

2 Báo cáo

2.1 Đề bài

1. Tạo bộ dữ liệu gồm 10 dãy, mỗi dãy khoảng 1 triệu số thực (ngẫu nhiên).

2. Hãy viết các chương trình sắp xếp dãy theo các thuật toán **QuickSort, HeapSort, MergeSort** và chương trình gọi hàm sort có sẵn trong C++(của thư viện *algorithm*)

3. Chạy thử nghiệm mỗi chương trình đã viết ở trên với bộ dữ liệu đã tạo, ghi nhận thời gian thực thi từng lần thử nghiệm, sau đó nhận xét về kết quả.

2.2 Cách thức thực hiện

Task 1: Sau khi đã tạo một file gồm 10 triệu số thực thì em cảm thấy việc xử lý rất lâu và rườm rà, thậm chí bộ nhớ không thể kham nổi với một lượng lớn dữ liệu như vậy. Vì lẽ đó, em sử dụng **Python** để sinh ra 10 file, mỗi file có 1 triệu số thực tượng trưng cho 1 dãy theo yêu cầu đề bài.

```python
import random
import pandas as pd
import numpy as np

for i in range(10):
    df = pd.DataFrame(np.random.rand(1,1000000)) * 1000
    df.to_csv("array_" + str(i + 1) + ".csv",index =
False)
```

Code 1. Hàm sinh data

Tiếp theo, modulo hóa một file **C** cho việc xử lý dữ liệu, đọc - ghi file và lưu vào một mảng động(**processing.h**).

```
void preprocessData(string path_file, double*& arr) {
    string line;
    fstream file(path_file);
    getline(file, line);
    getline(file,line);
        int col = 0;
        string substr;
        stringstream ss(line);

        while(col < si){

            getline(ss, substr, ',');
            arr[col] = stod(substr);
            col ++;


    }

}
  return go(f, seed, [])
}
```

Code 2. Hàm xử lý data

```
void writeTimeToFile(string path_file, double time){
    ofstream file;
    file.open(path_file,ios_base::app);
    file << time << " ";
}
```

Code 3. Hàm ghi nhận thời gian

Task 2: Tương tự modulo hóa, ta sẽ modulo hóa 3 hàm sort đã đề cập, ngoài ở file chính sử dụng hàm sort đã có sẵn, sau đó ghi ghi lại thowifgian ở các bước sử dụng hàm sắp xếp, đây là một phần của file main:

```
// Heapsort
    for(int i = 1; i <= 10; i++){
        clock_t start,end;
        stringstream s;
        s <<   "D:\\CTDL _ GT\\data\\array_" << i << ".csv";
        string path = s.str();
        double *arra = new double[si], time_use;
        preprocessData(path,arra);
        start = clock();
        heapSort(arra, si);
        end = clock();
        time_use = double(end - start) / CLOCKS_PER_SEC;
        writeTimeToFile("time_heapsort", time_use);
        delete[] arra;
    }
```

Code 4. Đoạn code sử dụng, đo thời gian hàm Heapsort và

*Chú ý: Xem folder **code** để đọc đầy đủ các modulo **sort** và hàm **main**.*

Task 3: Sau khi đo lại thời gian vào một file, ta sẽ xử lý file đó và trực quan hóa nó lên đồ thị bằng thư viện **matplotlib** của **Python**:

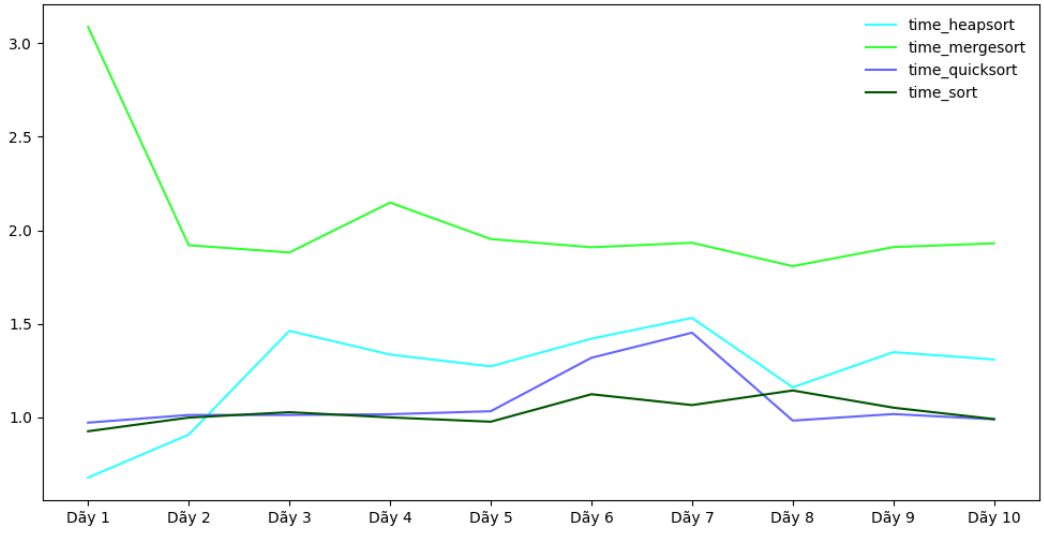
```
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
path_head = "D:\\CTDL _ GT\\output_time"
data = []
for item in os.listdir(path_head):
    file = open(path_head + "\\" + item,"r")
    li = file.readline().split()
    li_sub = []
    for item in li:
        li_sub.append(float(item))
    data.append(li_sub)

arr_li = ['Dãy 1', 'Dãy 2', 'Dãy 3', 'Dãy 4', 'Dãy 5', 'Dãy 6', 'Dãy 7', 'Dãy 8', 'Dãy 9', 'Dãy 10']
col_li = ['#33FFFF',"#33FF33","#6666FF","#005500"]

fig, ax = plt.subplots()
fig.set_size_inches(12,6)
for i in range(4):
    d = {"dãy":arr_li[:],"time":data[i][:]}
    df = pd.DataFrame(d)
    x = np.asarray(df["dãy"])
    y = np.asarray(df["time"])
    plt.plot(x, y, color=col_li[i], label= os.listdir(path_head)[i])

ax.legend(loc='upper right', frameon=False)
plt.show()
```

Code 5. Code xử lý file thời gian và trực quan hóa

3 Kết quả và đánh giá

3.1 kết quả

Qua hàm trực quan hóa, ta đã có kết quả như sau:



Kết quả dễ dàng cho t thấy được tương đướng thời gian các hàm như:

$$MergeSort \leq HeapSort \leq QuickSort \leq quick.$$

Tuy nhiên đây chỉ là kết quả của một số trường hợp, ta không thể tổng quát hóa và kết luận thuật tóa nào hiểu quả và nhanh hơn được.
Em đã thử một số trường hợp dữ liệu khác và kết quả mỗi lần lại khác nhau.

Sau khi tìm hiểu thì tùy vào bộ dữ liệu mà các thuật toán sẽ nhanh hay chậm, và độ hiểu quả sẽ khác nhau.

3.2 Đánh giá

Heapsort:

- Lợi ích chính của heapsort là tính ổn định và độ phức tạp thời gian trung bình O(nlogn).
- Nó thực hiện sắp xếp trực tiếp trên mảng ban đầu mà không cần tạo mảng phụ, do đó tiết kiệm không gian bộ nhớ.
- Heapsort thường được sử dụng trong các ứng dụng thời gian thực vì nó hoạt động tốt trong tình huống có dữ liệu đầu vào không đồng đều.

Quicksort:

- Có độ phức tạp trung bình là O(nlogn), và trong trường hợp tốt nhất là O(nlogn)..
- Nó thường nhanh hơn các thuật toán sắp xếp khác như merge sort hoặc heap-sort.
- Quicksort được sử dụng phổ biến trong các thư viện sắp xếp bởi tính hiệu quả và khả năng tối ưu hóa độ phức tạp theo thời gian.

Merge sort:

- Có độ phức tạp trung bình và trong trường hợp tốt nhất đều là O(nlogn).
- Nó ổn định và có thể sắp xếp các phần tử trong các danh sách liên kết, do đó có ứng dụng rộng rãi trong các ngôn ngữ lập trình như Java.
- Merge sort có thể được tối ưu hóa để sắp xếp song song các phần tử trên nhiều CPU hoặc máy tính.

Sort có sẵn:

- Tuy không tìm được nguồn về code của hàm sort này, nhưng trong nhiều trường hợp hàm này hoạt động rất tốt và ổn định(gần như lúc nào cũng đứng hạng nhất về thời gian).
- Có lẽ hàm này đã được kết hợp và cải tiến từ các hàm sort ở trên để tăng tốc độ và sự hiệu quả.
- Đây là hàm có độ phổ biến cao trong các chương trình cần sắp xếp ví dụ như C++, Python, Java,...

– Hết. –