

# KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN MÔN HỌC THỰC HÀNH CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

BÀI TOÁN SẮP XẾP DỮ LIỆU LỚN

Thành viên:

Phạm Phước Bình – 20120436 Đào Duy Anh - 20120426

Dương Khánh An - 20120424

## **GIỚI THIỆU BÀI TOÁN:**

### Phát biết bài toán

* Bộ dữ liệu: <https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews.> Sinh viên vào đường link vào download file “Books\_rating.csv”.
* Nhiệm vụ là sắp xếp theo “The Id of Book” từ nhỏ đến lớn. Trường hợp hai rating có “book id” bằng nhau, thì cứ sắp xếp ngẫu nhiên.
* Kết quả sắp xếp được lưu vào file “sorted\_books\_rating.csv”.

### Gợi ý

* Sinh viên dùng ý tưởng thiết kế thuật toán Merge sort hoặc B Tree hoặc dùng bất kỳ các thiết kế nào sinh viên tìm hiểu.
* Dữ liệu khoảng 3GB, để đảm bảo sinh viên có thao tác sai thì vẫn không làm ảnh hưởng đến máy tính. Tuy nhiên, sinh viên cần xử lý dữ liệu như việc dữ liệu không fit trên RAM.
* Gợi ý cho việc chạy code:
* Tạo dữ liệu kiểm thử file nhỏ.
* Chạy thử trên file nhỏ.
* Nếu chạy trên file nhỏ thành công, thì chạy trên toàn bộ dữ liệu.

## **PHÂN CHIA CÔNG VIỆC:**

|  |  |
| --- | --- |
| **Họ tên** | **Công việc** |
| Phạm Phước Bình |  |
| Đào Duy Anh |  |
| Dương Khánh An |  |

## **ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH:**

|  |  |
| --- | --- |
| **Yêu cầu** | **Mức độ hoàn thành** |
| Làm việc nhóm (10%) | 100% |
| Thiết kết kiến trúc (30%) | 100% |
| Áp dụng vào bài toán sắp xếp (30%) | 100% |
| Báo cáo kết quả, độ phức tạp (30%) | 100% |
| Quản lý source code và làm việc trên Github (5%) | 100% |

## **MÔ TẢ KIẾN TRÚC VÀ THUẬT TOÁN ĐƯỢC ÁP DỤNG VÀO BÀI TOÁN:**

### Đọc file và ghi file

Chương trình sẽ đọc file Books\_rating.csv vào ghi dữ liệu vào 1 mảng lines: lines[n]

Với lines có cấu trúc: string ID, string line.

ID dùng để chứa các ID (cột đầu) để dùng cho việc sắp xếp.

line dùng để chứa các ký tự được đọc nguyên 1 dòng trong file csv. 1 dòng file file csv được tổ chức như sau:các dòng được ngăn cách bới dấu cách, các cột được ngăn cách bới dấu phấy( , ). Với các đoạn (1 cột trong 1 dòng) mà trong đó có dấu phẩy thì ở đầu và cuối đoạn ấy được cho thêm dấu ngoặc kép ( “” ), nếu trong đoạn đó vừa có dấu phẩy và dấu ngoặc kép thì ở đầu và cuối dấu ngoặc kép cho thêm dấu ngoặc kép ( “” ) nữa. Khi hết 1 dòng, file sẽ xuống hàng.

vd: trong file csv: 1 cột (trong 1 dòng) có ghi abc thì khi đọc sẽ là …,abc, …

1 cột có ghi zx,cv thì khi đọc sẽ là …,“zx,cv”, …

1 cột có ghi 2,3 “sd” thì khi đọc sẽ là …,“2,3 “”sd”””, …

Khi đọc, chương trình sẽ đọc nguyên 1 dòng bằng hàm getline(), tách các dòng bới xuống dòng (\n), dòng đó sẽ gán vào biến line. Vì ID ở cột đầu tiên và dữ liệu ID trong file csv không có các trường đặc biệt như có dấu phẩy… nên chỉ đọc đơn giản là đọc từng ký tự từ trái sang của line cho đến khi gặp dấu phẩy thì dừng, sau gán đoạn ấy vào biến ID.

Trong quá trình nạp dữ liệu vào mảng lines, khi hoàn thành đọc 1 dòng, chương trình sẽ chuyển đến dòng tiếp theo và tiếp tục cho đến khi gặp điểm dừng: đến giới hạn của mảng hoặc đến cuối file csv.

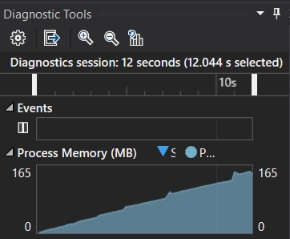
Do line chứa hết các ký tự trong 1 dòng nên không cần điều chỉnh gì cả chỉ cần xuất ra, mỗi dòng thì thêm dấu cách.

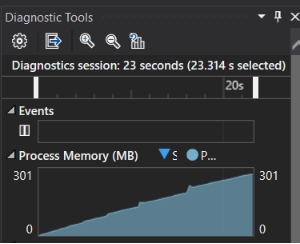
### Đặt giới hạn của mảng

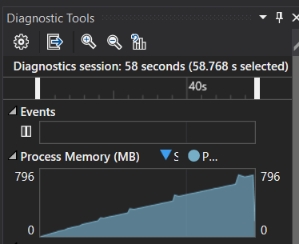
Kết quả chạy trên máy với cấu hình RAM 16GB, CPU Ryzen 5 4600H.

Khi nạp dữ liệu vào mảng lines có cấu trúc string ID, string line đã được định nghĩa ở trên:

Sau đây là 3 hình khi lần lượt 100 ngàn, 200 ngàn, 500 ngàn dòng đầu tiên đọc từ file Books\_rating.csv có 3 triệu dòng nạp vào mảng lines:







Ta có thể thấy 100 ngàn dòng cần 165 MB và chạy trong 12 giây

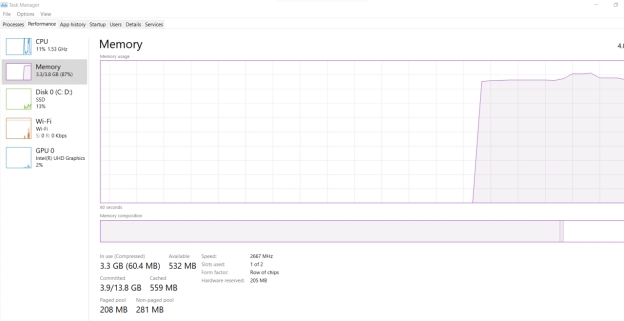
200 ngàn dòng cần 301 MB và chạy trong 23 giây

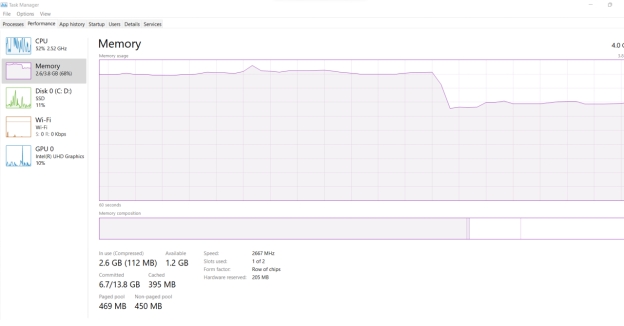
500 ngàn dòng cần 796 MB và chạy trong 58 giây

Tính ra trung bình 1 ngàn dòng cần cấp khoảng 1.5775 MB

1 GB chứa 1024 MB, suy ra 1 GB chứa được khoảng 649 ngàn dòng trong file Books\_rating.csv cho 1 lần nạp.

Sau đây là ảnh chụp từ máy của Đào Duy Anh với 4 GB RAM đang dùng Windows 11: ảnh đầu là khi vừa khởi động máy, ảnh sau là khi dùng các ứng dụng sau 1 thời gian và tắt các ứng dụng chiếm nhiều RAM đi.





Với máy 4 GB RAM chạy Windows 11 thì lý tưởng trống khoảng 1 GB nhưng thông thường là thấp hơn. Vậy nên 1 lần đọc file nạp vào mảng tối đa là 1 GB nhưng nên nạp tối đa 512 MB hoặc thấp hơn sẽ hợp lý vì còn có những chương trình khác đang chạy. 512 MB có thể chứa khoảng 324.5 ngàn dòng cho 1 lần nạp làm tròn xuống là 300 ngàn dòng.

Với máy 8 GB RAM chạy Windows 11 cho là trống 4 GB RAM 1 lần nạp là 2596 ngàn dòng thì nên giảm xuống 2500 ngàn dòng (chiếm 3943 MB) hoặc 2000 ngàn dòng (chiếm 3155 MB).

Với máy 16 GB RAM chạy Windows 11 cho là trống 8 GB RAM thì hoàn toàn có thể chứa hết 3 triệu dòng cho 1 lần nạp, chiếm 4732.5 MB RAM.

Mục đích của đặt giới hạn của mảng này nhằm tránh RAM bị đầy khi chạy; ngoài ra giảm số lần tạo ra nhiều file con (khi đặt giới hạn quá nhỏ) giúp giảm số lần merge các file lại với nhau từ đó cũng cải thiện thời gian chạy chương trình;...

### Tổ chức các file tạo ra

Mỗi lần dữ liệu được nạp vào mảng, sau khi nạp xong sẽ được sắp xếp. Khi sắp xếp xong sẽ xuất tạo ra 1 file mới. Quá trình lặp lại cho tới khi đọc hết file csv.

Các tên file tạo ra được lưu vào 1 mảng riêng có n phần tử cho tên là mảng sorted, sau đó tạo thêm n-1 file cũng có tên file được lưu vào trong 1 mảng khác phục vụ cho việc merge file cho tên là mảng merge, cách merge giống như merge sort. Cách đặt tên của file một cách tự động hoá bằng cách: name + count + “.csv” trong đó name là 1 string cho sẵn; count là 1 số int tăng lên 1 đơn vị mỗi lần tạo file, sau đó chuyển sang dạng string bằng hàm to\_string(); còn “.csv” để định dạng file tạo ra là csv.

\*tên file ám chỉ chính file đó trong mảng sorted và mảng merge, xoá tên file cũng có nghĩa là xoá file đó.

Cách chọn file của merge file: trước hết, phần tử đầu tiên và phần tử thứ hai của mảng sorted tiến hành merge lưu/copy dữ liệu vào phần tử đầu tiên của mảng merge, sau đó xoá 2 file đã được dùng để merge trong mảng sorted. Sau đó tiến hành vòng lặp: Phần tử tiếp theo của mảng sorted (lúc này là phần tử thứ ba) tiến hành merge với phần tử vừa được tạo (lúc này là phần tử đầu tiên) của mảng merge, tạo ra phần tử mới (lúc này là phần tử thứ hai) của mảng merge, sau đó xoá các file đã được dùng cho merge. Vòng lặp chạy cho đến khi mảng sorted chạy đến phần tử cuối cùng (file phần tử cùng đã được xoá), cũng có nghĩa là file cửa phần tử cuối cùng trong mảng merge là kết quả được sort từ file csv.

### Thuật toán sắp xếp

Có 2 thuật toán sắp xếp được sử dụng trong chương trình này: (quick sort?) và merge sort, trong chương trình này sắp xếp tăng dần.

(quick sort) được dùng trong việc sắp xếp dựa vào string ID của các phần tử lines trong mảng. Quá trình này nằm trong giai đoạn sau khi nạp vào mảng lines đọc từ file csv và trước khi ghi vào file trong mảng sorted.

Đối với merge sort là sử dụng cách thức merge vì merge sort có khả năng sắp xếp các mảng rời lại với nhau được, mảng lúc này tương đương với các file được tạo ra trong mảng sorted.

Điều kiện của việc merge này là các mảng ấy phải được sắp xếp vậy nên cần sắp xếp dữ liệu các file trong mảng sorted trước khi merge lại. Lí do không dùng luôn merge sort vì merge sort có dùng mảng phụ và dung lượng tạo ra có thể chiếm nhiều hơn 2 hoặc 3 lần so với việc nạp vào mảng lines (1024 MB hoặc 1536 MB so với 512 MB trong máy 4 GB RAM). Các thuật toán khác có dùng mảng phụ cũng không phù hợp, vì vậy mới dùng thuật toán chỉ sắp xếp trên chính mảng đó thì có các thuật toán như: selection sort, bubble sort,… trong đó (quick sort) phù hợp hơn vì có độ phức tạp là O(nlog(n)) so với O(n^2). (Để giảm thiểu gặp trường hợp xấu nhất, lấy 3 phẩn tử đầu, ở giữa, cuối để chọn pivot cho quick sort.)

### Các chi tiết nhỏ khác

* Dòng đầu của file csv chứa định nghĩa của các cột vậy nên khi đọc và nạp, bỏ qua nó; nhưng khi file kết quả cuối cùng được tạo thì chèn thêm dòng đó ở đầu file, có cách dòng ở cuối dòng.
* Khi chạy cần đảm bảo ổ cứng cần trống ít nhất 2 lần file cần sắp xếp (tổng dung lượng của 3 file khi chưa thực hiện việc xoá: phần tử cuối cùng của mảng sorted, phần tử kế cuối và phần tử cuối cùng của mảng merge)
* Cẩn thận khi dùng mảng lines được nạp từ file, đảm bảo khi dùng xong cần xoá/làm trống mảng đó trước khi nạp tiếp. Nếu không sẽ bị tràn bộ nhớ.
* Giữa các máy và giữa các lần chạy, kết quả dữ liệu:“1 ngàn dòng được nạp vào mảng lines chiếm 1.5775 MB RAM” và thời gian chạy có thể khác nhau. Vì vậy nên làm tròn xuống số dòng trong 1 lần nạp mảng lines phòng trường hợp 1 lần đọc 1 ngàn dòng chiếm nhiều RAM hơn.
* Chương trình chưa có tự động hoá trong việc xác định phần cứng, lượng RAM trống, cũng như trung bình 1 ngàn dòng chiếm bao nhiêu RAM để tính toán số dòng trong 1 lần nạp mảng lines. Ngoài ra khi chạy các dữ liệu có các dòng chênh lệch về ký tự càng nhiều thì khả RAM bị quá giới hạn càng cao.

### Các hàm và cấu trúc

Struct Book: string ID; string text;

Int partition(Book\*& arr, int low, int high): thuộc hàm phụ vụ cho quicksort

void quickSort(Book\*& arr, int low, int high): Hàm sắp xếp quicksort

void writeSmallFile(Book\*& Lines, int& end, vector<string>& Sorted, int& count): Hàm viết các file sau khi sắp xếp

void writeSmallMergeFile(string output, string input1, string input2, bool lastFile): Hàm viết các file sau khi merge lại với nhau

void ReadAndCreateFile(string fileName, vector<string>& Sorted, vector<string>& Merge): Hàm đọc và tạo các file nhỏ hơn nếu RAM không đủ, các trước khi ghi được sắp xếp bằng quicksort

void MergeFile(vector<string> Sorted, vector<string> Merge): Hàm tạo vòng lặp khi merge các file

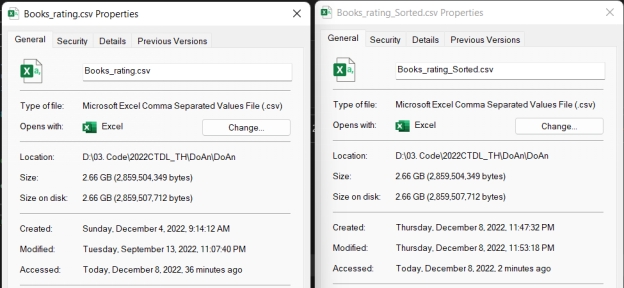
void Run(): Hàm chạy sắp xếp file

void runResultFile(): Hàm chạy xuất file Result.csv từ file đã sắp xếp với số dòng được giới hạn

int main(): Hàm chạy chính

Ngoài ra có các lênh cout và hàm phục vụ cho việc xác định giai đoạn chưởng trình và cho biết thời gian chạy.

### Kết quả



So sánh kích thước, đến kích cỡ bytes là như nhau cho thấy nội dung ít nhất giống nhau, chưa xác nhận được số dòng có thêm hay mất không. Nội dung thì trong file Result.csv là kết quả 1000 dòng đầu tiên của file Books\_rating\_Sorted.csv. Lần chạy 1500 ngàn dòng cho 1 lần nạp:

Quá trình được thực hiện trên máy 16 GB RAM, CPU Ryzen 5 4600H, có 1 số tác vụ chạy ngầm nhưng khi chạy CPU chưa dùng quá 25%, RAM trước khi chạy trống hơn 6GB. Thời gian chạy là 1097 giây tức 18 phút 17 giây.

Quan sát thấy: khi đặt giới hạn là 1500 ngàn dòng thì 1 lần nạp chiếm tới khoảng 2GB và cũng là giai đoạn tốn thời gian nhất khi qua quá trình sort và ghi file nhanh hơn nhiều. Trong quá trình sort, tuy hàm có đệ quy nhiều lần nhưng không chiếm RAM đáng kể so với quá trình nạp. Mỗi lần ghi file xong, trong hàm sẽ giải phóng mảng động đi và lượng RAM sử dụng cũng giảm theo. Merge thì vừa nạp vừa ghi file xen kẽ nhưng dựa vào suy diễn trước, việc đọc và nạp vào biến chiếm thời gian là chủ yếu. Khi chạy các file được tách ra có tổng kích thước gần bằng file ban đầu và khi merge tổng kích ngày càng tăng gấp 3 lần khi càng về giai đoạn merge cuối.

Vậy nên khi sort file csv này cần ít nhất 6 GB ổ đĩa trống, nếu không muốn tách file cần ít nhất 4-4.5 GB RAM trống nhưng nhờ đó bỏ qua giai đoạn merge giúp cải thiện thời gian chạy nhiều hơn.

Còn nhiều chỗ có thể cải tiến như quá trình merge file áp dụng buffer nạp 1 lần rồi dùng đến khi hết rồi lại nạp tiếp (nhanh hay chậm hơn thì không rõ), hay thay đổi cách nạp file nhanh hơn nhưng không gây ảnh hưởng nhiều các giai đoạn sau khiến thời gian chạy lâu hơn.

## **ĐỘ PHỨC TẠP THỰC TOÁN:**

+ Với thuật toán QuickSort ta có độ phức tạp trung bình là O(nLogn).

+ Hàm writeSmallFile vì chỉ có một vòng lặp while khi mở được file nên có độ phức tạp là O(n).

+ Hàm writeSmallFile có một vòng lặp while lớn và bên trong có nhiều trường hợp nhưng mỗi trường hợp chỉ có một vòng lặp for hoặc while không lồng nhau nên có độ phức tạp là O(n^2).

+ Hàm ReadAndCreateFile có một vòng lặp while lớn và bên trong có nhiều trường hợp nhưng mỗi trường hợp chỉ có một vòng lặp for nên độ phức tạp là O(n^2)

+ Hàm MergeFile Chỉ có một vòng lặp for nên độ phức tạp là 0(n)

+ runResultFile Chỉ có một vòng lặp for nên độ phức tạp là 0(n)

## **KẾT LUẬN:**

