BÀI TẬP 3 – CÁC KỸ THUẬT HIỆU CHỈNH MÃ NGUỒN VÀ LẬP TÀI LIỆU

Nhóm 8

March 2019

Bài tập nhóm môn học

IT 3040 - Kỹ thuật lập trình 2018 - 2019

Giảng viên hướng dẫn: PGS.TS Huỳnh Quyết Thắng

STT	Họ và tên	MSSV	Phân Chia công việc
1	Trương Ngọc Giang	20170067	Tìm mẫu tài liệu hướng dẫn sử dụng (User guide)
2	Nguyễn Mai Phương	20170106	Xây dựng nguyên tắc bổ sung chú thích (Comments)
3	Trương Quang Khánh	20170083	Tìm hiểu về Code Convention
4	Trần Minh Hiếu	20170075	Hiệu chỉnh Code
Lớp : KSTN - CNTT K62			

Mục lục

Chương 1

Code Tuning

Trong bài tập 2, nhóm đã viết chương trình giải mô phỏng thuật toán Merge Sort trên hai ngôn ngữ Java và Python, cũng như thực hiện các phương pháp kiểm thử tương ứng đối đối với hai mã nguồn này. Trong bài tập lần này, nhóm tiếp tục sử dụng các mã nguồn trên để thực hiện tối ưu hóa hiệu năng của chương trình.

1.1 Một số thay đổi trước khi bắt đầu bài tập

Chương trình mô phỏng thuật toán Merge Sort bao gồm hai module là hàm merge() và hàm sort(). Cả hai hàm này đều được đặt quyền truy cập public, nhằm cho phép các chương trình kiểm thử truy cập từ bên ngoài.

Tuy nhiên trong quá trình sử dụng, hàm merge() chỉ cần được gọi như là một bước con của thuật toán Merge Sort. Do đó trong bài tập lần này, quyền truy cập của merge() được đặt lại làm private. Điều này giúp giảm số lượng trường hợp cần phải xử lý, và làm cho ví dụ trở nên thực tế hơn.

1.2 Thực hiện các kỹ thuật hiệu chỉnh mã nguồn

1.2.1 Hiệu chỉnh các biểu thức logic

Trong quá trình kiểm thử, hàm merge() và hàm sort() đều có các điều kiểm tra mảng được truyền vào. Trong trường hợp mảng truyền vào là null, hoặc có độ dài không quá 1, chương trình sẽ lập tức thoát ra, không phải xử lý gì cả.

```
1     if (arr == null || arr.length <= 1)
2         return arr;
3
```

Tuy nhiên đoạn code kiểm tra này chỉ nên được gọi một lần, khi mảng được truyền vào lần đầu tiên. Trong các lần gọi tiếp theo của vòng lặp đệ quy, mảng được truyền vào được tính toán trực tiếp từ bên trong code, và không thể xảy ra trường hợp bằng null.

Ảnh hưởng về hiệu năng lên chương trình càng lớn hơn trong chương trình Python: do Python không định kiểu chặt lên biến, các thủ tục phải thực hiện kiểm tra dữ liệu đầu vào có đúng kiểu hỗ trợ hay không, trước khi tiếp tục xử lý. Việc kiểm tra này mất thời gian tuyến tính với độ dài của mảng, và làm giảm hiệu suất chương trình một cách đáng kể:

```
1     def __validate_(arr):
2         if (type(arr) is not list):
3             return False
4         for x in arr:
5             if (type(x) is not int):
6                 return False
7         return True
```

Phương án cải thiện: Tạo thêm một thủ tục sortRecursive(), đảm nhiệm việc tính toán một lần đệ quy của thuật toán sắp xếp. Việc kiểm tra null và kiểm tra kiểu dữ liệu chỉ được thực hiện một lần duy nhất tại hàm sort(). Sau khi kiểm tra xong, hàm sort() gọi sortRecursive(), thực hiện nội dung chính của thuật toán. Điều này vừa giúp cải thiện hiệu năng mà không làm thay đổi interface của chương trình.

```
    public static void sort(int[] arr) {
        if (arr == null)
            return;
        cache = new int[arr.length];
        sortRecursive(...);
    }
```

1.2.2 Hiệu chỉnh vòng lặp

Trong hàm merge(), việc ghép hai nửa đã sắp xếp của mảng được thực hiện bằng cách duyệt qua tất cả các vị trí của mảng kết quả, và gán giá trị tại vị trí đó bằng giá trị nhỏ hơn trong hai giá trị đứng đầu mỗi nửa. Nếu như một trong hai nữa đã hết giá trị thì mặc định là gán bằng giá trị của nửa còn lại.

```
    for(int i = 0; i < result.length; i ++) {
        if (idLeft == left.length)
            result[i] = right[idRight ++];
        else if (idRight == right.length)
            result[i] = left[idLeft ++];
        else {
            if (left[idLeft] < right[idRight])
                result[i] = left[idLeft ++];
            else
                result[i] = right[idRight ++];
        }
    }
```

Trong trường hợp nửa bên phải hết trước, điều kiện kiểm tra vẫn phải được thực hiện đối với nửa bên trái trước (và chắc chắn trả về đúng) khi thực hiện với nửa bên phải. Điều này làm ảnh hưởng tới hiệu năng của chương trình.

Phương án cải thiện: Thay một vòng lặp for bằng 3 vòng lặp while: một vòng lặp chạy khi cả hai nửa còn phần tử, và 2 vòng lặp chạy để đưa các phần tử còn lại vào cuối kết quả:

```
    while(idLeft < middle && idRight < to) {
        if (cache[idLeft] < cache[idRight])
            arr[iterPos ++] = cache[idLeft ++];
        else
            arr[iterPos ++] = cache[idRight ++];
    }
    while(idLeft < middle)
        arr[iterPos ++] = cache[idLeft ++];
    while(idRight < to)
        arr[iterPos ++] = cache[idRight ++];
```

Điều này giúp loại bỏ công việc thừa thãi trong vòng lặp, làm tăng tốc độ chương trình.

1.2.3 Hiệu chỉnh chuyển đổi dữ liệu

Do chương trình chỉ thao tác so sánh giữa các giá trị thuộc một kiểu dữ liệu duy nhất (số nguyên) nên không phát sinh việc chuyển đổi dữ liệu.

1.2.4 Hiệu chỉnh các biểu thức

Thuật toán Merge Sort cần thực hiện tính toán vị trí chính giữa của các đoạn để chia đoạn cần sắp xếp làm hai.

```
    int middle = arr.length / 2;

```

Phương án cải thiện: Phép toán chia một số nguyên cho hai được thực hiện rất nhiều lần, và có thể được thay thế bằng phép dịch phải sang 1 để tăng cường hiệu năng.

```
    int middle = from + (length >> 1);

```

1.2.5 Hiệu chỉnh hàm/thủ tục

Trong chương trình ban đầu, các thao tác tính toán trả về các object thuộc kiểu mảng số nguyên: việc chia mảng làm đôi được thực hiện bằng cách copy giá trị của mỗi nửa để tạo thành hai object mới; việc ghép hai mảng đã sắp xếp cũng trả về một object mảng, vân vân... Cách cài đặt này rất tiện lợi cho việc đọc hiểu, vì chỉ sử dụng các hàm được cung cấp sẵn bởi ngôn ngữ, và phải thực hiện rất ít thao tác toán.

```
    int[] left = sort(
        Arrays.copyOfRange(arr, 0, middle)
    );
    int[] right = sort(
        Arrays.copyOfRange(arr, middle, arr.length)
    );
    int[] result = merge(left, right);

```

Tuy nhiên, việc tạo thêm object mới liên tục khiến cho mức tiêu thụ bộ nhớ của thuật toán trở thành $O(nlog(n))$, cao hơn so với mức chuẩn là $O(n)$. Điều này vừa khiến chương trình trở nên lãng phí tài nguyên hơn, lại vừa làm giảm hiệu năng do phải liên tục kêu gọi cấp phát bộ nhớ cho các object mới.

Phương án cải thiện: chỉnh sửa hàm sortRecursive() và merge() thành thao tác theo vị trí trên mảng được truyền vào, thay vì thao tác trên object. Ngoài ra trong hàm sort(), ta cũng khởi tạo một mảng số nguyên cache[] có cùng kích cỡ với mảng truyền vào để lưu trữ giá trị trong khi chạy thuật toán. Nhờ đó, ta loại bỏ đi được việc cần phải yêu cầu bộ nhớ động.

```
    public static void sort(int[] arr) {
        if (arr == null)
            return;
        cache = new int[arr.length];
        sortRecursive(arr, 0, arr.length);
    }

```

1.3 Kiểm tra cải thiện hiệu năng

1.3.1 Môi trường kiểm thử

- Hệ điều hành: Antergos Linux
- Phiên bản Kernel: 5.0.5-arch1-1-ARCH

- Kiến trúc: 64-bit
- Vi xử lý: Intel® Core™ i3-3217U CPU, 4 nhân, 1.80GHz
- Bộ nhớ: 7.7 GiB of RAM
- Phiên bản Java: OpenJDK Runtime Environment 11.0.3
- Phieen bản Python: Python 3.7.3

Các chương trình đọc dữ liệu từ file test.txt, chứa một dãy các số nguyên được sinh ngẫu nhiên theo phân phối chuẩn. Chương trình Java sẽ thực hiện sắp xếp dãy số 10^7 phần tử, trong khi chương trình Python sẽ thực hiện trên dãy số 10^6 phần tử. Điều này là do Python là ngôn ngữ thông dịch, và do đó có tốc độ chậm hơn so với ngôn ngữ biên dịch là Java.

Thời gian hoạt động của thuật toán được tính duy nhất trên hàm sort() - bỏ qua các giai đoạn nhập và xuất dữ liệu. Đối với Java, việc tính toán được thực hiện như sau:

```
    long startTime = Instant.now().toEpochMilli();
    sort(arr);
    long endTime = Instant.now().toEpochMilli();
    long runTime = endTime - startTime;
    System.out.println(runTime);

```

Đối với Python:

```
    start_time = datetime.datetime.now()
    sort(array)
    end_time = datetime.datetime.now()
    print(end_time - start_time)

```

Nhóm cũng sử dụng timeout, một chương trình Perl nhỏ cho phép đo đạc lại thời gian hoạt động và lượng tài nguyên sử dụng của chương trình command line trên Linux.

1.3.2 Số liệu đo đạc

Ngôn ngữ	Thời gian chạy trước tối ưu	Thời gian chạy sau tối ưu	Mức độ cải tiến
Java	3444 ms	2907 ms	39.11%
Python	2340 ms	1442 ms	38.38%

Bảng 1.1: Số liệu đo đạc thời gian chạy của chương trình

Ngôn ngữ	Bộ nhớ sử dụng trước tối ưu	Bộ nhớ sử dụng sau tối ưu	Mức độ cải tiến
Java	302227 byte	265681 byte	12.09%
Python	99893 byte	100211 byte	0.32%

Bảng 1.2: Số liệu đo đạc thời gian chạy của chương trình

1.3.3 Nhận xét

Phần lớn các cải tiến được đề xuất trong báo cáo đã cho thấy hiệu quả rõ rệt: Hiệu năng của chương trình được cải thiện tới gần 40%. Mức độ yêu cầu bộ nhớ Java giảm hơn 12%. Tuy nhiên, vẫn xảy ra ngoại lệ trên số liệu về bộ nhớ của Python.

Chương 2

Quy ước code trong Python/Java

2.1 Giới thiệu chung

Tại sao lại cần đến các quy ước chuẩn chung khi viết code?
Thực tế, 80% thời gian sử dụng của các phần mền cần được bảo trì. Nhưng không phải lần bảo trì nào cũng được thực hiện bởi chính người đã viết ra phần mền đó, hay có nghĩa là các lập trình viên sẽ phải thao tác trên mã nguồn của người khác. Do đó, cần có một quy chuẩn viết code chung để có thể tăng khả năng dễ đọc của code, cho phép người lập trình hiểu code nhanh và kĩ càng hơn. Ngoài ra, khi viết code theo quy ước chung cũng giống như mặc đồng phục cho sản phẩm của mình, thể hiện sự "gọn gàng, ngăn lắp", tăng độ tin cậy cho sản phẩm.

2.2 Các quy ước chuẩn khi code trong ngôn ngữ Java

2.2.1 Cấu trúc của file mã nguồn Java

Một file mã nguồn Java sẽ cấu trúc theo thứ tự như sau:

- Dòng comment giới thiệu ngắn gọn về phiên bản, bản quyền và chức năng của file mã nguồn.
- Khai báo package rồi import statement.
- Viết class/interface implementation comment (/* ... */)
- Khai báo các class và interface.
- Khai báo biến toàn cục, theo thứ tự từ public, protected đến private.
- Khai báo hàm khởi tạo.
- Khai báo các phương thức.

2.2.2 Khoảng cách thụt đầu dòng và bố cục dòng code

- Khoảng cách thụt đầu dòng
  Một lần thụt vào để biểu thị quan hệ giữa các lệnh có độ dài là 4 space, thường thì điều này đã được mặc định sẵn trong editor.
- Đọ dài dòng code
  Không nên viết dòng code chứa quá 80 kí tự, do một số terminal không hỗ trợ các trường hợp đó.
- Xuống dòng khi đoạn code không phù hợp với một dòng code hiện tại
  - Vị trí ngắt sau dấu phẩy và trước toán tử.
  - Nên ngắt ở các toán hạng có thứ tự thực hiện thấp hơn.

– Khi ngắt dòng nên sắp xếp các tham số có cùng chức năng thẳng hàng nhau.

Dưới đây là một ví dụ:

```
function(longExpression1, longExpression2, longExpression3,
        longExpression4, longExpression5);
```

2.2.3 Comment

2.2.3.1 Block Comment

- Thường được dùng để mô tả file, thuật toán, cấu trúc dữ liệu, hàm.
- Thường được đặt trước hàm, file, hoặc nếu trong hàm thì được đặt cùng số dòng thụt vào với đoạn code nó mô tả.
- Đây là một ví dụ của block comment

```
/*
 * Đây là một ví dụ về block comment
 * /
```

2.2.3.2 Single-Line Comment

- Nếu comment chỉ trên một dòng thì sẽ dùng Single-Line Comment, có cùng số dòng thụt vào với đoạn code.
- Sau đây là ví dụ của một Single-Line Comment

```
if (condition} {
     /* Write Comment in here */
}
```

2.2.3.3 Trailing Comment

- Là một đoạn comment ngắn cùng dòng với code
- Trong một khối code thì các Trailing Comment sẽ được thụt ra sao cho thẳng hàng với nhau.

2.2.3.4 End-Of-Line Comment

- Trên cùng một dòng code, sau // sẽ là comment, chương trình dịch sẽ bỏ qua nó
- Thường được dùng để comment từng phần của dòng code.

2.2.3.5 Documentation Comment

- Doc-Comment được dùng để mô tả về class, interface, công cụ khởi tạo, phương thức và các trường. Mỗi giao diện chương trình sẽ có một doc-comment.
- Doc-comment thường được được đặt trước khai báo, ví dụ:

```
/**
 * Ví dụ về doc-comment.
 */
class Example{
  ...
}
```

- Doc Comment có thể phân tách thành file HTML bằng các công cụ của Java.

2.2.4 Declarations

- Nên sử dụng một khai báo trên một dòng code.
- Không khai báo khác kiểu trên cùng một dòng.
- Các biến địa phương nên được khai báo trên đầu khối code sử dụng chúng.

```
void MyMethod() {
   int int1;         // beginning of method block

   if (condition) {
    int int2;        // beginning of "if" block
    ...
   }
```

2.2.5 Các quy ước đặt tên

- Tên của class sẽ là danh từ, với các chữ cái đầu của từng từ được viết hoa.
- Cách đặt tên của interface giống với class.
- Tên phương thức là động từ, với chữ cái của từ đầu tiên được viết thường, còn chữ cái đầu tiên của các từ tiếp theo được viết hoa.
- Cách đặt tên biến cũng giống như phương thức, tránh các trường hợp tên biến quá dài hoặc chỉ có một kí tự, trừ các trường hợp là biến đếm tạm thời.
- Cách đặt tên các hằng số là toàn bộ bởi các chữ cái in hoa, các từ được nối với nhau bởi "_".

2.3 Các quy ước chuẩn khi code trong ngôn ngữ Python

Về các quy tắc cơ bản thì trong Python cũng quy ước giống trong Java. Do Python 2 và Python 3 sẽ có các định dạng chuẩn khác nhau nên dưới đây chỉ nói những phần chung nhất của cả hai.

2.3.1 Code Lay-Out

2.3.1.1 Indentation

- Sử dụng 4 khoảng trắng mỗi mức thụt vào đầu dòng.
- Các dòng nằm trong cùng khối code thì được xếp thẳng hàng.
- Khi ngắt dòng các dòng code thì các đoạn code hay các thành phần code có cùng chức năng sẽ được xếp thẳng hàng.
- Trong if-else statement, nếu điều kiện viết trong một dòng không đủ thì có thể ngắt dòng, thêm dấu ngoặc đơn vào điều kiện, dữ các tham chiếu thẳng hàng với nhau.
- Khi có nhiều dòng trong ngoặc đơn, ngoặc móc, thì dấu đóng ngoặc sẽ nằm ở dòng khác sau dòng cuối cùng của dãy code trong ngoặc. Ví dụ:

```
my_list = [
    1, 2, 3,
    4, 5, 6,
]
```

2.3.1.2 Độ dài dòng code

Một dòng code nói chung sẽ không có đến 80 kí tự trên một dòng, trong các block code thì độ dài không quá 72 kí tự.

2.3.1.3 Import

- Một dòng chỉ có một Import
- Import đứng ở đầu của file, chỉ sau comment hoặc docstring.

2.3.1.4 Naming Conventions

- Python không có cú pháp chặt chẽ cho các đối tượng private như trong Java. Nhưng quy ước chung là, bắt đầu tên với kí tự "_" nhằm chỉ thị rằng đây là thành phần private, không nên truy cập đến các thành phần này từ bên ngoài.
- Dùng kí tự "_" ở đuôi của tên để tránh sự trùng lặp với các tên, từ khoá,... đã sẵn có.
- Tên của Class sẽ được viết hoa chữ cái đầu tiên của các từ.
  VD: ClassName
- Tên của các Exception giống như cách đặt với tên của class, thường thêm hậu tố Error để phân biệt.
- Tên của hàm sẽ là các chữ cái viết thường, các từ được cách nhau bởi "_", mục đích tăng khả năng đọc.
- Quy tắc đặt tên biến cũng giống như tên hàm.
- Tên các hằng số được viết hoa tất cả chữ cái,các từ được cách nhau bởi kí tự "_"

Chương 3

Các quy tắc chú thích

3.1 Giới thiệu

Tại sao phải thêm chú thích cho các đoạn code?

Nhiều lập trình viên nghĩ rằng việc thêm chú thích chỉ gây lãng phí thời gian và công sức và nó chẳng có ý nghĩa gì.

Thật vậy, những chú thích tồi có thể lặp lại code và khiến chương trình trở nên dài dòng. Đôi khi nó còn làm cho chương trình trở nên khó hiểu hơn. Các lập trình viên thường cải thiện code mà không thay đổi chú thích cho phù hợp. Thật tồi tệ nếu mộtchú thích tóm tắt nội dung, mục đích mà không đúng với nội dung của đoạn code.

Tuy nhiên, các chú thích hiệu quả cũng giống như tiêu đề sách hoặc mục lục, chúng giúp ta dễ dàng tìm kiếm các đoạn code mà mình mong muốn. Đôi khi, các chú thích hiệu quả cũng giúp ta biết nên thay đổi hoặc sửa chỗ nào. Nếu chương trình của bạn không có chú thích, quản lí của bạn có thể không hiểu bạn viết gì, và họ có thể chỉ trích bạn về điều này.

Vậy nên, bạn cần viết chú thích cho các đoạn code. Và hãy chắc chắn rằng các chú thích đó đúng và hiệu quả.

3.2 Các loại chú thích trong lập trình

Thường ta phân ra làm 5 dạng của chú thích:
Loại 1: Lặp lại mã code

- Thường lặp lại những gì code thể hiện bằng một ngôn ngữ khác
- Chỉ đơn thuần cung cấp cho người đọc nhiều thứ để đọc hơn hơn là cung cấp thêm thông tin

⇒ Loại chú thích này thường có cũng như không.
Loại 2: Giúp giải thích code

- Giúp giải thích những chỗ phức tạp, khó khăn, hoặc những đoạn code nhạy cảm.
- Trong một vài tình huống nó hữu dụng, thường là khi đoạn code dễ gây nhầm lẫn.

⇒ Nếu code thật sự khó hiểu, nên chỉnh lại code thay vì viết chú thích.
Loại 3: Bình luận của người lập trình

- Nó thường dùng để chú ý cho các nhà phát triển biết rằng công việc chưa được hoàn thành hoặc một vài thứ tương tự.
- Thường đánh dấu nỗi bật bằng một vài cách như ****** ở trước hoặc !!!!!! ở sau...

⇒ Loại chú thích cần được sử dụng.
Loại 4: Tóm tắt code

- Thường trong một hoặc 2 câu dùng để tóm tắt một đoạn code mà các câu lệnh có liên quan tới nhau và cùng để thực hiện một mục đích nào đó.

⇒ Loại chú thích cần được sử dụng.
Loại 5: Mô tả mục đích của code
⇒ Loại chú thích cần được sử dụng.

3.3 Làm sao để viết chú thích hiệu quả?

3.3.1 Chỉ sử dụng một phong cách viết chú thích

3.3.2 Sử dụng mã giả trước khi code để làm rõ bạn muốn viết gì và tiết kiệm thời gian viết chú thích

3.3.3 Tích hợp việc viết chú thích vào quá trình code của bạn

Việc này có là nhiều bất tiện. Vì khi bạn sửa code, bạn phải sửa cả chú thích. Đó là lí do vì sao bạn nên phác họa mã giả trước khi bắt tay vào code. Phác thảo mã giả sẽ giúp bạn làm rõ những vì bạn nghĩ và bạn cần làm.

3.3.4 Đừng lấy năng suất làm một lí do để tránh viết chú thích

3.4 Các nguyên tắc bổ sung chú thích trong phần mềm

3.4.1 Nguyên tắc khi bình luận một câu lệnh code đơn lẻ

Trong các code hiệu quả, rất hiếm khi phải chú thích từng dòng ngoại trừ 2 trường hợp:

- Câu lệnh đó vô cùng phức tạp để hiểu và cần phải giải thích.
- Câu lệnh đó có một lỗi và bạn muốn ghi lại lỗi đó.

Nguyên tắc 1: Tránh viết những chú thích mà chỉ mình hiểu.

Nguyên tắc 2: Tránh viết những chú thích ở cuối dòng code đơn lẻ.
Những chú thích này rất khó để viết hiệu quả. Thường nó chỉ lặp lại code hơn là giúp ích gì đó. Thay vào đó, nó thường ở bên phải đoạn code và nó thường không nổi bật trong đó code và thường ta phải căn chỉnh để đọc.

Nguyên tắc 3: Tránh viết chú thích trên cuối một dòng và tự mình hiểu rằng nó dùng để áp dụng luôn cho các dòng khác.
Người đọc không thể nào biết được là nó sẽ áp dụng cho các dòng nào thậm chí dòng chú thích này rất hiêu quả.

Nguyên tắc 4: Tránh dùng chú thích ở cuối một dòng lệnh để ghi chú bảo trì.

Nguyên tắc 5: Khi nào nên dùng dùng chú thích ở cuối một dòng:

- *Để đánh dấu kết của một khối*
- *Để giải thích ý nghĩa các biến mà ta khai báo.*

3.4.2 Nguyên tắc khi bình luận một đoạn code

Nguyên tắc 1: Viết chú thích nhắm vào mục đích đoạn code.

- // find the command word terminator ($)

Ví dụ với một vòng lặp, chú thích này có thể giúp người đọc biết dòng chứa mục tiêu của vòng lặp đó, người đọc nên tập trung vào đó.

Nguyên tắc 2: Tập trung vào đoạn code, làm rõ mục tiêu mà nó hướng tới bằng các biến điều khiển

Nguyên tắc 3: Tập trung vào giải thích TẠI SAO thay vì LÀM SAO

Chú thích giải thích LÀM SAO

```
1                 // if account flag is zero
2                  if( accountFlag == 0)
3
```

Chú thích giải thích TẠI SAO

```
1                 // if establishing a new account
2                 if( accountFlag == 0)
3
```

Nguyên tắc 4: Sử dụng chú thích để chuẩn bị cho người đọc những gì cần theo dõi.

Nguyên tắc 5: Hạn chế viết các chú thích tối nghĩa.

Nguyên tắc 6: Nếu bạn đọc một tài liệu có những đoạn không rõ ràng hoặc cung cấp một kiến thức, hãy chú thích cho nó.

Nguyên tắc 7: Tránh viết vắn tắt.

```
// copy the string portion of the table, along the way omitting
// strings that are to be deleted

// ... determine number of strings in the table


...


// ... mark the strings to be deleted


...
```

3.4.3 Chú thích cho các dữ liệu được khai báo

Nguyên tắc 1: Chú thích về đơn vị của các dữ liệu số.

Nguyên tắc 2: Chú thích về phạm vi cho phép của các dữ liệu số.

Nguyên tắc 3: Chú thích về giới hạn của dữ liệu đầu vào.

Nguyên tắc 4: Nếu biến sử dụng để đánh dấu bằng các bit 0,1,... Cần chú thích để làm rõ ý nghĩa của các bit.

Nguyên tắc 5: Đảm bảo rằng khi tên biến thay đổi, các chú thích liên quan tới tên biến cũng thay đổi

Nguyên tắc 6: Chú thích cho các dữ liệu toàn cầu.

3.4.4 Chú thích cho các cấu trúc điều khiển

Nguyên tắc 1: Đặt chú thích trước các mệnh đề "if", "case", hoặc vòng lặp.

Nguyên tắc 2: Chú thích ở cuối mỗi cấu trúc điều khiển, sử dụng khi các mệnh đề lồng nhau.

Nguyên tắc 3: Coi các bình luận cuối vòng lặp như một cảnh báo chỉ ra mã phức tạp.

3.4.5 Chú thích cho các thủ tục

Nguyên tắc 1: Đặt chú thích gần đoạn code mà nó mô tả.

Nguyên tắc 2: Mô tả mỗi thủ tục bằng một hoặc hai dòng ở đầu của nó.

Nguyên tắc 3: Chỉ dẫn về nơi mà các biến hiển thị.

```
public void InsertionSort(
   int[] dataToSort, // elements to sort in locations firstElement..lastElement
   int firstElement, // index of first element to sort (>=0)
   int lastElement // index of last element to sort (<= MAX_ELEMENTS)
)
```

Nguyên tắc 4: Làm rõ sự khác biệt giữa dữ liệu vào và dữ liệu ra.

```
void StringCopy(
   char *target,     // out: string to copy to
   char *source      // in: string to copy from
)
...
```

Nguyên tắc 5: Chú thích về giới hạn của thủ tục.
Nếu một hàm có trả về một giá trị là số, nó có thể có giới hạn hoặc đơn vị cần chú thích. Nếu thủ tục có thể có một vài biểu hiện khác khi gặp một vài rắc rối, ta nên chú thích nó...

Nguyên tắc 6: Dẫn chứng về những hiệu ứng toàn cầu mà thủ tục gặp phải.
Nếu thủ tục có chứa những dữ liệu toàn cầu, mô tả chính xác ta làm gì với nó. Nếu chỉ dẫn trở nên khó khăn, viết lại code để giảm các dữ liệu toàn cầu.

Nguyên tắc 7: Chỉ dẫn nguồn của thuật toán mà ta sử dụng.

Nguyên tắc 8: Sử dụng chú thích để đánh dấu các phần của chương trình.

3.4.6 Chú thích cho các classes, files, programs

3.4.6.1 Chú thích cho các classes, files, programs

Nguyên tắc 1: Mô tả chung về cấu trúc của class.

Nguyên tắc 2: Mô tả các giới hạn, . . .

Tương tự như với các thủ tục, ta cần đảm bảo về giới hạn của class. Thêm nữa, ta cũng cần ước lượng đầu vào và đầu ra, các lỗi, ảnh hưởng toàn cầu, nguồn của thuật toán,. . .

Nguyên tắc 3: Chú thích về giao diện của class.

Nguyên tắc 4: Không chỉ dẫn quá chi tiết về giao diện của class.

Thông thường, các tập tin chỉ dẫn về giao diện của class thường được đính kèm với nó.

3.4.6.2 Hướng dẫn chung về chú thích cho files

Nguyên tắc 1: Mô tả mục đích và nội dung chính của một file.

Nguyên tắc 2: Đặt tên của bạn, email và số điện thoại trong một khối chú thích.

Nguyên tắc 3: Đặt cả nội dung về bản quyền trong chú thích.

Nguyên tắc 4: Đặt tên file liên quan tới nội dung của nó.

Chương 4

Hướng dẫn sử dụng (User guide)

4.1 Mẫu hướng dẫn sử dụng và lý do chọn

Nhóm quyết định sử dụng User guide của timeout làm mẫu hướng dẫn tài liệu.

Lý do chọn mẫu này bởi vì trong quá trình tìm hiểu thì User guide của timeout trình bày rất dễ hiểu, liền mạch logic từ dễ đến khó. Giúp người sử lập trình dễ sử dụng.

https://github.com/pshved/timeout

4.2 Hướng dẫn sử dụng MergeSort

4.2.1 Giới thiệu

MergeSort là một thuật toán sắp xếp đệ quy bằng cách chia nhỏ dãy cần sắp xếp thành 2 dãy con rồi lại thực hiện sắp xếp trên 2 dãy con này.

Độ phức tạp thuật toán là $O(nlog_2(n))$.

Bộ nhớ tính toán là $O(n)$.

4.2.2 Hướng dẫn cài đặt

Trong Python để sử dụng được ta cần

```
import merge_sort as ms
```

Còn trong Java ta để copy file MergeSort vào cùng thư mục rồi thêm MergeSort. vào trước tên hàm cần gọi là được (VD: MergeSort.sort(arr))).

4.2.3 Thông tin về các hàm

Python:

```
ms.sort(arr)
```

Truyền vào một biến kiểu List arr. Sau khi chạy các phần tử trong arr sẽ được sắp xếp không giảm.

Java:

```
public static void sort(int[] arr)
```

Truyền vào một mảng int arr. Sau khi chạy các phần tử trong arr sẽ được sắp xếp không giảm.

4.2.4 Ví dụ

*Trong **Python**:*

```
import merge_sort as ms

A = [5, 3, 1, 2, 4]
ms.sort(A)


print(A)

```

*Trong **Java**:*

```

class testSort {
public static void main(String args[]) {

    int[] arr = {5, 1, 3, 4, 2};
    MergeSort.sort(arr);

    for(int i = 0; i < arr.length; ++i)
        System.out.println(arr[i]);
    }
}


```

4.2.5 Thông tin về Code

Triển khai thuật toán đã ẩn đi 2 hàm:
def sort_recursive(arr, fr, to): Thực hiện đệ quy việc sắp xếp.
def merge(arr, fr, middle, to): Hợp nhất 2 dãy đã sắp xếp để thu được 1 dãy được sắp xếp.
Thông tin chi tiết về 2 hàm này mọi người có thể đọc trong file merge_sort.py hoặc MergeSort.java.

4.2.6 Bản quyền

Thuật toán được triển khai bởi Nhóm 8 sử dụng vào môn Kỹ Thuật Lập Trình (IT3040). Viết vào tháng 4 năm 2019.