Ton Duc Thang University
Faculty of Information Technology

Assignment

MÔN: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

MÃ MÔN: 504008

Đề bài: Quản lý sinh viên bằng cây AVL

(Sinh viên đọc kỹ tất cả hướng dẫn trước khi làm bài)

I. Giới thiệu bài toán

Một trường học muốn quản lý sinh viên theo mã số sinh viên và thực hiện các thao tác quản lý thêm, xóa, tìm kiếm theo mã số sinh viên một cách nhanh chóng. Vì thao tác thêm, xóa có khả năng sai sót nên trường học này muốn có thêm chức năng hoàn tác (undo) và khôi phục (redo). Bên cạnh đó, trường học còn muốn xây dựng thêm chức năng từ danh sách sinh viên đang quản lý chuyển thành một danh sách quản lý theo điểm số của sinh viên.

Các chức năng mà sinh viên phải thực hiện cho hệ thống trên là:

- Xây dựng cây nhị phân tìm kiếm cân bằng (cây AVL) để lưu trữ sinh viên theo mã số sinh viên.
- Hiện thực các thao tác thêm, xóa, tìm kiếm sinh viên trên cây AVL.
- Hiện thực các thao tác undo, redo.
- Duyệt cây sinh viên theo level-order và xây dựng cây AVL theo điểm số.

II. Tài nguyên cung cấp

Source code được cung cấp sẵn bao gồm các file:

- File dữ liệu, file đầu vào và file kết quả mong đợi:
  - Thư mục *testcase* chứa 07 testcase từ testcase1.txt đến testcase7.txt.
  - Thư mục *expected_output* chứa 07 output từ output1.txt đến output7.txt là kết quả mong đợi ứng với 07 testcase trong thư mục testcase.
  - File list_student.txt: chứa thông tin của 100 sinh viên để làm testcase.
  - File information.txt: chứa thông tin giải thích các thao tác trong file testcase.
- File mã nguồn:
  - *Main.java*: tạo đối tượng, đọc file testcase và file list_student.txt, gọi đến các phương thức sinh viên định nghĩa và ghi file kết quả.

- *Student.java* và *Node.java*: lần lượt chứa lớp **Student** và **Node** được định nghĩa sẵn. **Sinh viên không chỉnh sửa các file này.**
- *AVL.java*: chứa lớp **AVL**, có sẵn một số phương thức phụ trợ và các phương thức để xây dựng cây AVL. Sinh viên sẽ hiện thực các phương thức còn thiếu trong file này.
- *StudentManagement.java*: chứa lớp **StudentManagement** được định nghĩa sẵn thuộc tính *cây AVL* để chứa danh sách sinh viên (Student). Sinh viên sẽ hiện thực các phương thức trong lớp **AVL** trước sau đó hiện thực tiếp các phương thức còn thiếu trong file này.
- *ScoreAVL.java*: chứa lớp **ScoreAVL** dùng cho Yêu cầu 6. Sinh viên sẽ hoàn thiện file này trong quá trình thực hiện Yêu cầu 6.

III. Trình tự thực hiện bài

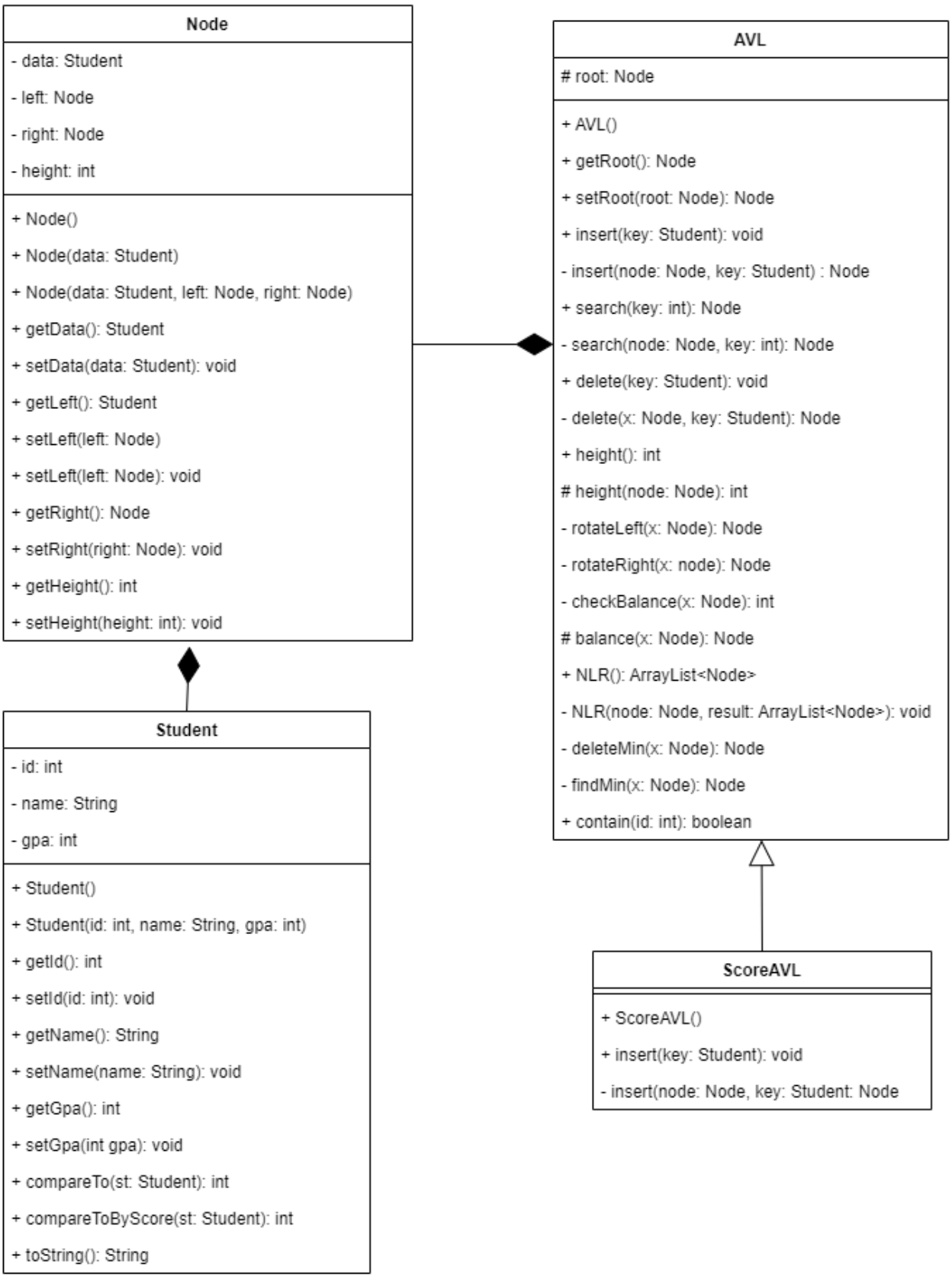
- Sinh viên tải file tài nguyên được cung cấp sẵn và giải nén.
- Đọc kĩ đề và mã nguồn được cung cấp cấp sẵn để hiểu cấu trúc file và các thao tác trong file testcase.
- File testcase1.txt đến testcase6.txt ứng với 6 yêu cầu trong bài, file testcase7.txt là tổng hợp 6 yêu cầu thực hiện cùng một lúc.
- Sinh viên hiện thực được hết 6 yêu cầu thì mới thực hiện trên testcase7.txt. Testcase chấm bài sẽ xuất hiện nhiều ở dạng của file testcase7.txt.
- Quy ước danh mục viết tắt các yêu cầu:
  - Yêu cầu 1: YC1
  - Yêu cầu 2: YC2
  - Yêu cầu 3: YC3
  - Yêu cầu 4: YC4
  - Yêu cầu 5: YC5
  - Yêu cầu 6: YC6
- Ở YC1, YC2 và YC3, sinh viên hiện thực cây **AVL** bằng cách hoàn thiện các phương thức còn thiếu trong file *AVL.java* và hiện thực các phương thức tương ứng của từng yêu cầu trong file *StudentManagement.java*.
- Ở YC4, YC5 và YC6, sinh viên hiện thực trong lớp **StudentManagement** theo yêu cầu. Tại YC4 và YC5 sinh viên phải làm cẩn thận vì quá trình làm sẽ ảnh hưởng đến các yêu cầu trước đó. YC6 sinh viên phải hiện thực trước phương thức còn thiếu trong file *AVLScore.java*.
- Sau khi hiện thực các phương thức trong lớp **StudentManagement**, sinh viên biên dịch và chạy với phương thức **main** trong file *Main.java* đã gọi sẵn các phương thức. Sinh viên

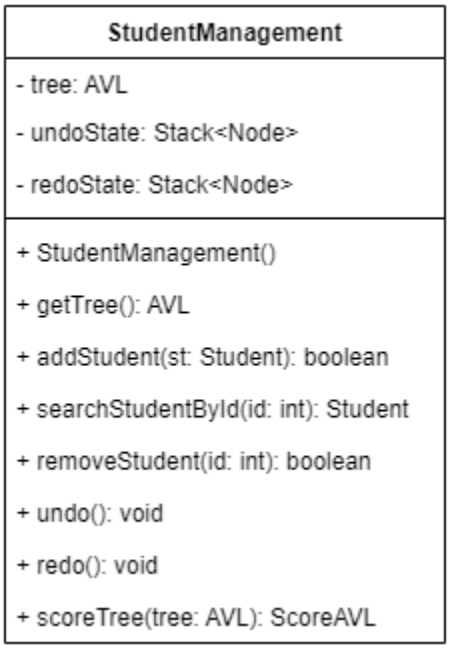
thay đổi tham số theo các file testcase tương ứng và so sánh kết quả output của mình với kết quả trong thư mục expected_output đã được cung cấp sẵn.

- Đối với các yêu cầu sinh viên không làm được vui lòng không xóa và phải đảm bảo chương trình chạy được với phương thức main trong Main.java được cung cấp sẵn ban đầu.
- Đường link Google Drive gửi đề này cho sinh viên sẽ chứa kèm một file *version.txt*, sinh viên nên thường xuyên vào xem file này, nếu thấy có nội dung mới thì nên đọc kĩ mô tả và tải lại đề mới nhất. File này được dùng để thông báo nội dung chỉnh sửa và ngày chỉnh sửa trong trường hợp đề bị sai hoặc lỗi.

IV. Mô tả lớp Student, Node, ScoreAVL, StudentManagement

- Giải thích một số thuộc tính và phương thức như sau:
  - Lớp **Student** (sinh viên chỉ đọc hiểu và không chỉnh sửa lớp này):
    - *id*: mã số sinh viên (MSSV)
    - *name*: họ tên sinh viên
    - *gpa*: điểm trung bình tích lũy GPA
    - **compareTo(Student st):** so sánh 2 sinh viên theo MSSV.
    - **compareToByScore(Student st):** so sánh 2 sinh viên theo điểm GPA.
    - **toString():** in thông tin sinh viên.
  - Lớp **Node** (sinh viên chỉ đọc hiểu và không chỉnh sửa lớp này):
    - *data:* đối tượng sinh viên Student
    - *left, right:* liên kết trái và liên kết phải
    - *height:* độ cao của nút
  - Lớp **AVL**:
    - Dùng để tạo cây AVL theo mã số sinh viên.
    - *root:* nút gốc
    - Một số phương thức như **height**, **rotateRight**, **rotateLeft**, **checkBalance**, **balance**, **NLR**, … đã được định nghĩa sẵn**.**
  - Lớp **ScoreAVL**:
    - Kế thừa từ cây AVL, dùng để tạo cây AVL theo điểm số.


Node
- data: Student
- left: Node
- right: Node
- height: int
+ Node()
+ Node(data: Student)
+ Node(data: Student, left: Node, right: Node)
+ getData(): Student
+ setData(data: Student): void
+ getLeft(): Student
+ setLeft(left: Node)
+ setLeft(left: Node): void
+ getRight(): Node
+ setRight(right: Node): void
+ getHeight(): int
+ setHeight(height: int): void
AVL
root: Node
+ AVL()
+ getRoot(): Node
+ setRoot(root: Node): Node
+ insert(key: Student): void
- insert(node: Node, key: Student) : Node
+ search(key: int): Node
- search(node: Node, key: int): Node
+ delete(key: Student): void
- delete(x: Node, key: Student): Node
+ height(): int
# height(node: Node): int
- rotateLeft(x: Node): Node
- rotateRight(x: node): Node
- checkBalance(x: Node): int
balance(x: Node): Node
+ NLR(): ArrayList<Node>
- NLR(node: Node, result: ArrayList<Node>): void
- deleteMin(x: Node): Node
- findMin(x: Node): Node
+ contain(id: int): boolean
Student
- id: int
- name: String
- gpa: int
+ Student()
+ Student(id: int, name: String, gpa: int)
+ getId(): int
+ setId(id: int): void
+ getName(): String
+ setName(name: String): void
+ getGpa(): int
+ setGpa(int gpa): void
+ compareTo(st: Student): int
+ compareToByScore(st: Student): int
+ toString(): String
ScoreAVL
+ ScoreAVL()
+ insert(key: Student): void
- insert(node: Node, key: Student: Node



- Giải thích một số thuộc tính và phương thức như sau:
  - Lớp **StudentManagement**:
    - *tree*: cây AVL theo mã số sinh viên.
    - *undoState*: trạng thái để undo
    - *redoState*: trạng thái để redo
    - **addStudent(Student st):** thêm sinh viên vào cây *tree*. Ứng với Yêu cầu 1.
    - **searchStudentById(int id):** tìm kiếm sinh viên trên *tree* theo **id**. Ứng với Yêu cầu 2.
    - **removeStudent(int id):** xóa sinh viên trên tree theo **id**. Ứng với Yêu cầu 3.
    - **undo():** thực hiện thao tác undo. Ứng với Yêu cầu 4.
    - **redo():** thực hiện thao tác redo. Ứng với Yêu cầu 5.
    - **scoreTree(AVL tree):** xây dựng cây AVL theo điểm GPA từ cây AVL **tree** theo MSSV truyền vào. Ứng với Yêu cầu 6.

V. Mô tả các file dữ liệu được cung cấp sẵn

- File information.txt là file mô tả số thứ tự các thao tác:
  - 0: thêm sinh viên vào cây AVL
  - 1: ghi kết quả NLR cây hiện tại
  - 2: tìm kiếm sinh viên qua mã số
  - 3: xóa sinh viên theo mã số
  - 4: thực hiện thao tác undo
  - 5: thực hiện thao tác redo

- 6: trả về cây AVL điểm GPA theo cây AVL hiện tại.

- File input trong một lần chạy chương trình gồm file list_student.txt và file testcaseX.txt với X là số thứ tự testcase. Trong thư mục **testcase** gồm 07 file testcase.
- File list_student.txt có định dạng:

Mã số sinh viên,Họ tên sinh viên,Điểm GPA

list_student.txt

```
0,Hugo Clark,4
1,Anita Mansell,62
2,Teddy Neal,30
3,Anaiya Cobb,7
4,Maximilian Mcclain,18
5,Prisha Clemons,71
```

- File testcastX.txt chứa các thao tác theo định dạng:

Số thứ tự thao tác <khoảng cách> [tham số id nếu có]

testcase > testcase7.txt

```
0 68
0 6
0 12
1
2 68
3 12
4
1
2 12
3 98
1
0 93
0 33
0 32
4
4
4
4
1
5
5
1
0 48
```

Ví dụ diễn giải hình trên từ dòng 8 đến dòng 15 là:

- Thêm sinh viên mã số 68
- Thêm sinh viên mã số 6
- Thêm sinh viên mã số 12
- Ghi kết quả NLR cây hiện tại
- Tìm kiếm để in thông tin sinh viên mã số 68
- Xóa sinh viên mã số 12
- Undo
- In ra màn hình NLR cây hiện tại

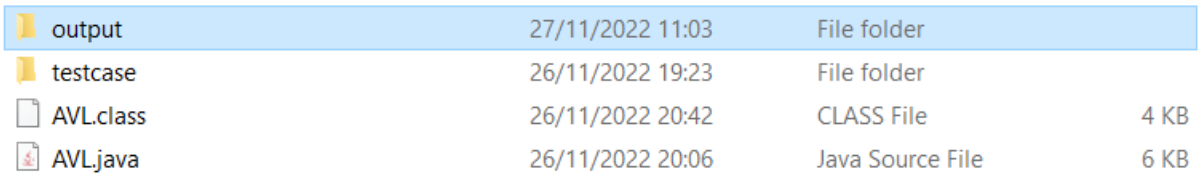
- Một file testcase sẽ có thể có nhiều thao tác, để hỗ trợ sinh viên dễ kiểm tra lại bài làm, các testcase từ 1 đến 6 chỉ chứa đúng các thao tác của yêu cầu tương ứng.
  - testcase1.txt: chỉ chứa thao tác thêm và thao tác in kết quả NLR.
  - testcase2.txt: chỉ chứa thao tác thêm và thao tác tìm kiếm sinh viên.
  - testcase3.txt: chỉ chứa thao tác thêm, xóa và in kết quả NLR.
  - testcase4.txt: chỉ chứa thao tác thêm, xóa, in kết quả NLR và undo.
  - testcase5.txt: chỉ chứa thao tác thêm, xóa, in kết quả NLR, undo và redo.
  - testcase6.txt: chỉ chứa thao tác xây dựng cây AVL điểm GPA.
  - testcase7.txt: chứa nhiều thao tác khác nhau (các testcase điểm cao sẽ nằm ở dạng này)
- Thư mục expected_output gồm có 7 file ứng với 7 testcase.

Lưu ý:

- Sinh viên có thể tự thêm dữ liệu hoặc tự tạo thêm testcase mới để thử nhiều trường hợp khác nhau nhưng lưu ý file dữ liệu phải đúng định dạng đã được nên bên trên.
- Sinh viên nên đọc kỹ phương thức **main** để xác định hướng hiện thực các lớp và các phương thức theo thứ tự.
- Sinh viên có thể thêm một số thao tác vào phương thức **main** để kiểm các phương thức đã định nghĩa tuy nhiên đảm bảo bài làm của sinh viên **phải chạy được trên phương thức main ban đầu đã cung cấp sẵn**.
- Sinh viên có thể thêm phương thức mới để phục vụ bài làm tuy nhiên phương thức phải thêm vào các file có nộp lại và bài làm của sinh viên phải chạy được với file *Main.java* đã được cung cấp sẵn.
- Tuyệt đối không chỉnh sửa **tên lớp hoặc tên của các phương thức** đã có sẵn (tuân theo đúng sơ đồ lớp phía trên).
- Sinh viên không thực hiện chỉnh sửa trên những file không yêu cầu nộp lại.

VI. Hướng dẫn chạy phương thức main

- Phương thức ***main*** được cung cấp sẵn trong file *Main.java* được định nghĩa sẵn các phương thức để đọc file và xét thao tác. Sinh viên hiện thực đúng các phương thức theo yêu cầu đề thì phương thức trong lớp **Main** sẽ tự đọc file và gọi đến các phương thức tương ứng mà sinh viên đã hiện thực.
- Để kiểm thử các yêu cầu, sinh viên phải định nghĩa các lớp đủ và đúng theo các yêu cầu, sau khi biên dịch thành công, tạo một thư mục tên **output** trong cùng cấp thư mục với source code.



- Phương thức ***main*** đã khai báo sẵn đường dẫn để đọc file *testcaseX.txt* từ thư mục **testcase** và đường dẫn ghi file *outputX.txt* vào thư mục **output**.
- Sinh viên gọi theo cú pháp để ghi kết quả của testcase ra file output tương ứng:

  java Main X

  Với X là testcase mà testcaseX.txt mà sinh viên muốn chạy.
  Ví dụ, sinh viên muốn gọi để kiểm thử testcase 1, sinh viên nhập *java Main 1*, kết quả sẽ được ghi ra file *output.txt* tại thư mục **output**.
- Sau khi đã kiểm thử thành công với các testcase có sẵn, sinh viên có thể tự tạo testcase mới và kiểm thử tương ứng.

VII. Yêu cầu

Sinh viên **không được phép thêm thư viện khác** ngoài các thư viện đã có sẵn trong file nhận được.

Sinh viên thực hiện bài tập lớn trên Java 11 hoặc Java 8. Sinh viên không được dùng kiểu dữ liệu *var*. Bài làm của sinh viên sẽ được chấm trên Java 11, sinh viên tự chịu trách nhiệm tất cả các lỗi phát sinh nếu dùng các phiên bản Java khác.

Phần mã nguồn cung cấp cho sinh viên đã định nghĩa sẵn đọc file và ghi file, sinh viên không cần hiện thực thêm cũng như cũng không cần chỉnh sửa đọc ghi file để tránh xảy ra lỗi.

1. YÊU CẦU 1 (2 điểm)

Sinh viên hiện thực phương thức **private Node insert(Node node, Student key)** để thực hiện đệ quy thêm một sinh viên mới vào cây AVL. Lưu ý thêm sinh viên phải cân bằng cây mỗi khi thêm một nút mới vào. (Sinh viên có thể dùng các phương thức phụ trợ có sẵn trong lớp **AVL** để hiện thực phương thức này)

Sau đó, trong lớp **StudentManagement,** hiện thực phương thức:

public boolean addStudent(Student st)

để thêm một sinh viên mới vào cây *tree* trong lớp **StudentManagement**. Nếu đối tượng sinh viên được thêm vào có mã số sinh viên chưa tồn tại trong cây thì thêm và trả về *true*, ngược lại nếu đối tượng có mã số trùng với mã số đã có trong cây thì không thêm và trả về *false*.

Lớp **Main** đã định nghĩa sẵn việc đọc thao tác cùng dữ liệu từ file, gọi đến phương thức trên và ghi file là họ tên sinh viên theo thứ tự NLR của cây hiện tại. Sinh viên kiểm thử bằng cách biên dịch và gọi câu lệnh *java Main 1* thì chương trình sẽ chạy *testcase1.txt* và cho ra file *output1.txt* trong thư mục **output** mà sinh viên đã tạo trước đó. Sinh viên xem kết quả file này và file *output1.txt* trong thư mục **expected_output** được cung cấp sẵn để so sánh kết quả.

Lưu ý: Đây là phương thức sinh viên phải định nghĩa được thì các Yêu cầu bên dưới mới được tính điểm.

2. YÊU CẦU 2 (1 điểm)

Trong lớp **AVL**, sinh viên hiện thực phương thức **private Node search(Node x, int key)** để tìm nút theo **key**. Trong lớp **StudentManagement**, sinh viên hiện thực phương thức:

public Student searchStudentById(int id)

để thực hiện tìm sinh viên có mã số sinh viên **id** truyền vào. Nếu tồn tại sinh viên cần tìm thì trả về đối tượng sinh viên, ngược lại không tồn tại thì trả về *null*.

Lớp **Main** đã định nghĩa sẵn việc đọc thao tác cùng dữ liệu từ file, gọi đến phương thức trên và ghi file là thông tin đối tượng sinh viên tìm được. Sinh viên kiểm thử bằng cách biên dịch và gọi câu lệnh *java Main 2* thì chương trình sẽ chạy *testcase2.txt* và cho ra file *output2.txt* trong thư mục **output** mà sinh viên đã tạo trước đó. Sinh viên xem kết quả file này và file *output2.txt* trong thư mục **expected_output** được cung cấp sẵn để so sánh kết quả.

3. YÊU CẦU 3 (2 điểm)

Trong lớp **AVL**, sinh viên hiện thực phương thức **private Node delete(Node x, Student key)** để xóa nút có mã số trùng với mã số của đối tượng **key**. Lưu ý thêm sinh viên phải cân bằng cây mỗi khi xóa một nút. (Sinh viên có thể dùng các phương thức phụ trợ có sẵn trong lớp **AVL** để hiện thực phương thức này)

Trong lớp **StudentManagement**, sinh viên hiện thực phương thức:

public boolean removeStudent(int id)

để thực hiện xóa sinh viên có mã số sinh viên **id** truyền vào. Nếu tồn tại sinh viên xóa thì xóa và trả về *true*, ngược lại không tồn tại thì không xóa và trả về *false*.

Lớp **Main** đã định nghĩa sẵn việc đọc thao tác cùng dữ liệu từ file, gọi đến phương thức trên và ghi file. Sinh viên kiểm thử bằng cách biên dịch và gọi câu lệnh *java Main 3* thì chương trình sẽ chạy *testcase3.txt* và cho ra file *output3.txt* trong thư mục **output** mà sinh viên đã tạo trước đó. Sinh viên xem kết quả file này và file *output3.txt* trong thư mục **expected_output** được cung cấp sẵn để so sánh kết quả

4. YÊU CẦU 4 (2 điểm)

Hiện thực phương thức

public void undo()

để hoàn tác (undo) thao tác vừa thực hiện. Việc hoàn tác chỉ có tác dụng với thao tác **thêm** sinh viên và thao tác **xóa** sinh viên, các thao tác còn lại không bị ảnh hưởng bởi thao tác hoàn tác.

Ví dụ cho dãy 8 thao tác sau:

1. Thêm sinh viên mã số 68
2. Thêm sinh viên mã số 72
3. Xóa sinh viên mã số 68
4. Ghi kết quả NLR cây hiện tại
5. Thêm sinh viên mã số 12
6. Undo
7. Undo
8. Ghi kết quả NLR cây hiện tại

Tại lần ghi kết quả NLR đầu tiên thì cây hiện tại sẽ chỉ có nút chứa sinh viên mã số 72. Sau 2 lệnh undo thì ghi kết quả NLR sẽ cho cây hiện tại ở trạng thái vừa thực thi xong dòng lệnh số 2. *Giải thích*: dòng lệnh undo số 6 sẽ hoàn tác kết quả Thêm sinh viên mã số 12 và dòng lệnh undo số 7 sẽ hoàn tác kết quả Xóa sinh viên mã số 68.

Gợi ý: Để thực hiện yêu cầu này, trong quá trình thực hiện thao tác thêm và xóa, sinh viên sao chép (clone) ra một cây mới giống cây hiện tại và sử dụng *Stack<Node> undoState* đã cung cấp sẵn để lưu nút gốc (root) của cây vừa sao chép ra. Khi cần undo thì sinh viên chỉ cần gán lại nút gốc của cây hiện tại thành nút gốc gần nhất trong Stack để khôi phục lại trạng thái của cây trước đó.

Lớp **Main** đã định nghĩa sẵn việc đọc thao tác cùng dữ liệu từ file, gọi đến phương thức trên và ghi file. Sinh viên kiểm thử bằng cách biên dịch và gọi câu lệnh *java Main 4* thì chương trình sẽ chạy *testcase4.txt* và cho ra file *output4.txt* trong thư mục **output** mà sinh viên đã tạo trước đó.

Sinh viên xem kết quả file này và file *output4.txt* trong thư mục **expected_output** được cung cấp sẵn để so sánh kết quả.

5. YÊU CẦU 5 (1 điểm)

Hiện thực phương thức

public void redo()

để trả lại (redo) thao tác undo vừa thực hiện. Việc trả lại chỉ có tác dụng với thao tác **undo**, nếu không **undo** thì lệnh **redo** không có tác dụng. Các thao tác undo và redo có thể xen kẽ nhau nhưng số lần có thể redo phụ thuộc vào số lệnh undo trước đó và khi thực hiện một thao tác mới không phải undo thì lệnh redo cũng không còn tác dụng.

Gợi ý: Để thực hiện yêu cầu này, trong quá trình thực hiện thao tác undo, sinh viên sao chép (clone) ra một cây mới giống cây hiện tại và sử dụng *Stack<Node> redoState* đã cung cấp sẵn để lưu nút gốc (root) của cây vừa sao chép ra. Khi cần redo thì sinh viên chỉ cần gán lại nút gốc của cây hiện tại thành nút gốc gần nhất trong Stack để khôi phục lại trạng thái của cây trước đó.

Lớp **Main** đã định nghĩa sẵn việc đọc thao tác cùng dữ liệu từ file, gọi đến phương thức trên và ghi file. Sinh viên kiểm thử bằng cách biên dịch và gọi câu lệnh *java Main 5* thì chương trình sẽ chạy *testcase5.txt* và cho ra file *output5.txt* trong thư mục **output** mà sinh viên đã tạo trước đó. Sinh viên xem kết quả file này và file *output5.txt* trong thư mục **expected_output** được cung cấp sẵn để so sánh kết quả.

6. YÊU CẦU 6 (1 điểm)

Lớp **ScoreAVL** là lớp kế thừa từ lớp **AVL** dùng để lưu trữ cây **AVL** dựa theo điểm GPA của sinh viên. Các testcase để chấm cho yêu cầu này sẽ không phát sinh dữ liệu trùng điểm GPA, do đó sinh viên không cần xử lý trường hợp trùng điểm GPA.

Trong lớp **ScoreAVL**, sinh viên hiện thực phương thức **private Node insert(Node x, Student key)** để thêm một nút mới vào cây.

Sau đó, trong lớp **StudentManagement,** hiện thực phương thức:

public ScoreAVL scoreTree(AVL tree)

để xây dựng và trả về cây AVL theo điểm GPA từ cây AVL theo mã số sinh viên *tree* truyền vào. Thứ tự của đối tượng sinh viên khi thêm vào cây AVL điểm GPA là thứ tự duyệt level-order của cây AVL *tree* tham số.

Lớp **Main** đã định nghĩa sẵn việc đọc thao tác cùng dữ liệu từ file, gọi đến phương thức trên và ghi file. Sinh viên kiểm thử bằng cách biên dịch và gọi câu lệnh *java Main 6* thì chương trình sẽ

chạy *testcase6.txt* và cho ra file *output6.txt* trong thư mục **output** mà sinh viên đã tạo trước đó. Sinh viên xem kết quả file này và file *output6.txt* trong thư mục **expected_output** được cung cấp sẵn để so sánh kết quả.

7. YÊU CẦU 7 (1 điểm)

Trong quá trình hiện thực 6 yêu cầu bên trên, sinh viên phải đảm bảo bài làm chạy được với file testcase có nhiều thao tác khác nhau. File *testcase7.txt* là ví dụ mẫu cho nhiều thao tác trong cùng một file.

Lớp **Main** đã định nghĩa sẵn việc đọc thao tác cùng dữ liệu từ file, gọi đến phương thức trên và ghi file. Sinh viên kiểm thử bằng cách biên dịch và gọi câu lệnh *java Main 7* thì chương trình sẽ chạy *testcase7.txt* và cho ra file *output7.txt* trong thư mục **output** mà sinh viên đã tạo trước đó. Sinh viên xem kết quả file này và file *output7.txt* trong thư mục **expected_output** được cung cấp sẵn để so sánh kết quả.

Sinh viên có thể tự tạo thêm nhiều file testcase với các thao tác khác nhau để kiểm thử lại bài làm.

VIII. Lưu ý kiểm tra trước khi nộp bài

- Nếu sinh viên không thực hiện được yêu cầu nào thì để nguyên phương thức của yêu cầu đó, TUYỆT ĐỐI KHÔNG XÓA PHƯƠNG THỨC CỦA YÊU CẦU sẽ dẫn đến lỗi khi chạy phương thức **main**. Trước khi nộp phải kiểm tra chạy được với phương thức **main** được cho sẵn.
- Tuyệt đối không dùng đường dẫn tuyệt đối (absolute path) trong bài làm.
- Tất cả các file outputX.txt (X = {1,2,3,4,5,6,7}) được ghi ra trong thư mục **output**, thư mục **output** nằm cùng cấp với thư mục chứa các file source code. Đối với sinh viên sử dụng IDE (Eclipse, Netbeans, …) phải đảm bảo file chạy được bằng command prompt, đảm bảo **bài làm không nằm trong package**, vị trí ghi file kết quả outputX.txt phải nằm trong thư mục **output**. Đường dẫn khai báo trong file **Main.java** cung cấp sẵn đã đảm bảo vấn đề này nếu sinh viên dùng command prompt.
- File kết quả ghi đúng thư mục khi chạy chương trình sẽ có dạng như sau:

output	26/11/2022 19:45	File folder	
testcase	26/11/2022 19:23	File folder	
AVL.class	26/11/2022 20:42	CLASS File	4 KB
AVL.java	26/11/2022 20:06	Java Source File	6 KB
information.txt	26/11/2022 19:48	Text Document	1 KB
list_student.txt	09/11/2022 18:36	Text Document	3 KB
Main.class	26/11/2022 20:42	CLASS File	5 KB
Main.java	26/11/2022 20:41	Java Source File	5 KB
Node.class	26/11/2022 20:42	CLASS File	2 KB
Node.java	21/11/2022 00:25	Java Source File	2 KB
ScoreAVL.class	26/11/2022 20:42	CLASS File	1 KB
ScoreAVL.java	26/11/2022 19:43	Java Source File	1 KB
Student.class	26/11/2022 20:42	CLASS File	2 KB
Student.java	26/11/2022 20:25	Java Source File	2 KB
StudentManagement.class	26/11/2022 20:42	CLASS File	3 KB
StudentManagement.java	26/11/2022 20:42	Java Source File	4 KB

- Trong thư mục output:

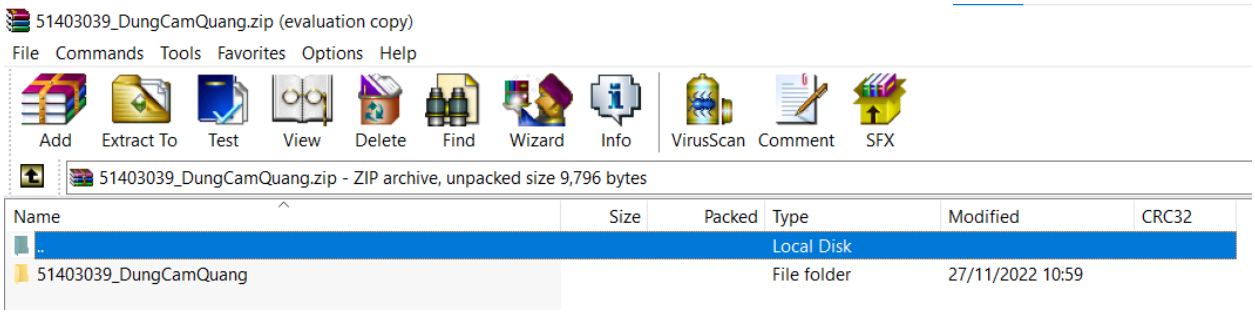
Name	Date modified	Type	Size
output1.txt	26/11/2022 20:49	Text Document	1 KB
output2.txt	26/11/2022 20:49	Text Document	1 KB
output3.txt	26/11/2022 20:49	Text Document	1 KB
output4.txt	26/11/2022 20:49	Text Document	1 KB
output5.txt	26/11/2022 20:49	Text Document	1 KB
output6.txt	26/11/2022 20:49	Text Document	1 KB
output7.txt	26/11/2022 20:49	Text Document	2 KB

IX. Hướng dẫn nộp bài

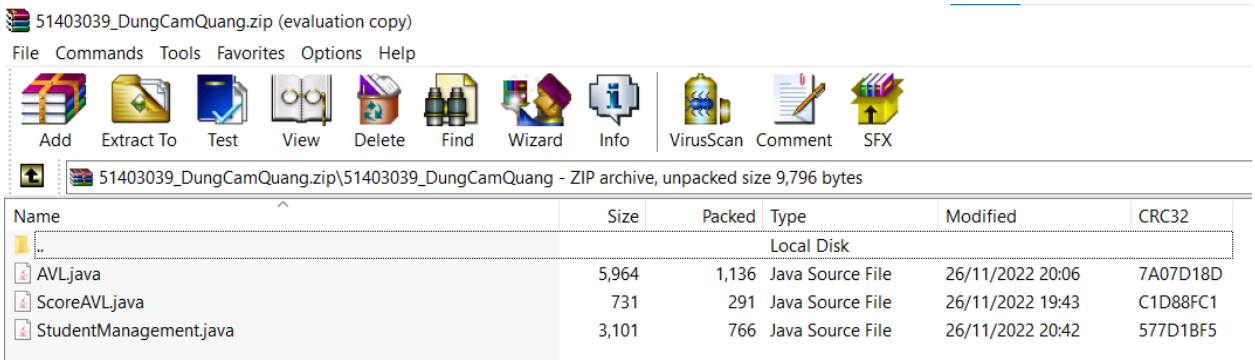
- Khi nộp bài sinh viên nộp lại file *AVL.java*, *StudentManagement.java* và file *ScoreAVL.java*, **không nộp kèm bất cứ file nào khác và tuyệt đối không được sửa tên 3 file này.**
- **Sinh viên đặt 3 file bài làm vào thư mục MSSV_HoTen** (HoTen viết liền, không dấu) và nén lại với định dạng **.zip** nộp theo sự hướng dẫn của giảng viên thực hành.
- Trường hợp làm sai yêu cầu nộp bài (đặt tên thư mục sai, không để bài làm vào thư mục khi nộp, nộp dư file, …) thì bài làm của sinh viên sẽ bị **0 điểm**.
- File nộp đúng sẽ như sau:
  - File nén nộp bài:

51403039_DungCamQuang.zip	27/11/2022 11:00	WinRAR ZIP archive	3 KB

- Bên trong file nén:



- Bên trong thư mục:



X. Đánh giá và quy định

- Bài làm sẽ được chấm tự động thông qua testcase (file input và output có định dạng như mẫu đã gửi kèm) do đó sinh viên tự chịu trách nhiệm nếu không thực hiện đúng theo Hướng dẫn nộp bài hoặc tự ý sửa tên các phương thức đã có sẵn dẫn đến bài làm không biên dịch được khi chấm.
- Testcase sử dụng để chấm bài là các file đầu vào có cùng định dạng nhưng khác nội dung với file sinh viên đã nhận, sinh viên chỉ được điểm mỗi YÊU CẦU khi chạy ra đúng hoàn toàn kết quả của yêu cầu đó.
- Nếu bài làm của sinh viên biên dịch bị lỗi thì **0 điểm cả bài**.
- **Tất cả code sẽ được kiểm tra đạo văn. Mọi hành vi sao chép code trên mạng, chép bài bạn hoặc cho bạn chép bài nếu bị phát hiện đều sẽ bị điểm 0 vào điểm Quá trình 2 hoặc cấm thi cuối kì.**
- Nếu bài làm của sinh viên có dấu hiệu sao chép trên mạng hoặc sao chép nhau, sinh viên sẽ được gọi lên phỏng vấn code để chứng minh bài làm là của mình.
- **Hạn chót nộp bài: 23h00 ngày 13/12/2022.**

-- HẾT --