Assignment
Tính toán đơn giản trên số nguyên rất lớn

Quang D. C.
dcquang@it.tdt.edu.vn

October 21, 2020

Bài tập lớn này sẽ chiếm 50% điểm Quá trình 2, sinh viên hoàn tất assignment và nộp lại đúng hạn. Kết quả bài tập lớn sẽ được **chấm tự động**, yêu cầu sinh viên thực hiện chính xác tất cả hướng dẫn trong bài, **mọi trường hợp làm sai hướng dẫn** nộp, đặt tên, tự ý chỉnh sửa các phần không được yêu cầu **đều sẽ bị 0 điểm**.

1. Giới thiệu

Ngày xưa ở đất nước Ấn Độ có một ông vua rất ham chơi. Ông nói với các đại thần rằng, ước mong lớn nhất của ông là có được một trò chơi mà chơi mãi vẫn không thấy chán. Ai có thể nghĩ ra trò chơi đó, sẽ được ông trọng thưởng. Không lâu sau, một vị đại thần thông minh hiến tặng nhà vua một loại cờ. Bàn cờ có 64 ô, quân cờ có khắc chữ "Hậu", "Vua", "Xe", "Ngựa" v.v. Đánh loại cờ này - một loại trò chơi có sự biến hóa vô cùng - quả thật làm cho người ta không cảm thấy chán. Nhà vua vui mừng nói với vị đại thần đó: "Ta muốn trọng thưởng cho nhà ngươi. Nhà ngươi muốn gì, ta sẽ làm cho nhà ngươi được toại nguyện".

Vị đại thần đáp: "Thần chỉ muốn những hạt lúa mạch." "Lúa mạch ư ? Nhà ngươi muốn bao nhiêu?"

"Thưa bệ hạ, người hãy cho đặt ở ô thứ nhất của bàn cờ một hạt lúa mạch, ô thứ hai hai hạt, ô thứ ba bốn hạt, ô thứ tư tám hạt..., cứ như thế cho đến khi đặt hết 64 ô bàn cờ."

Nhà vua nghĩ bụng: Nếu như vậy thì có đáng là bao nhiêu chứ? Nhiều lắm thì cũng vài trăm đấu! Chuyện nhỏ! Vì thế, ông nói với đại thần quản lý kho lương: "Nhà ngươi đi lấy vài bao lúa để thưởng cho ông ấy."

Vị đại thần quản lý kho lương tính toán một lúc, bỗng nhiên mặt ông ta biến sắc. Ông ta tâu trình với nhà vua: "Kết quả tính toán của thần cho thấy, số lương thực của cả nước ta còn kém xa số lương thực mà ông ấy muốn."

Nói xong liền đưa kết quả tính toán cho nhà vua xem. $1 + 2 + 2^2 + 2^3 + ... + 2^{63} = 2^{64} - 1 = 18,446,774,073,709,551,615$ hạt lúa mạch.

Một mét khối lúa mạch có khoảng 15 triệu hạt, theo cách tính này thì cần phải đưa cho vị đại thần đó 1,200 tỉ mét khối lúa mạch. Số lúa mạch này còn nhiều hơn tổng số lúa mạch mà toàn thế giới gieo trồng trong 200 năm.

Sắc mặt nhà vua tái mét, ông vội hỏi đại thần quản lý lương thực: "Làm thế nào bây giờ? Nếu đồng ý cho ông ta số lúa mạch đó, trẫm sẽ vĩnh viễn trở thành kẻ thiếu nợ.

Nếu không đồng ý đưa cho ông ta, chẳng phải trẫm sẽ trở thành kẻ nuốt lời hứa sao? Nhà ngươi hãy nghĩ cách giúp ta đi!"

Vị đại thần quản lý kho lương nghĩ ngợi một lát rồi nói: "Chỉ có một cách duy nhất, bệ hạ phải thực hiện lời hứa của mình thì mới làm cho người dân tin tưởng bệ hạ là một ông vua tốt".

"Nhưng trẫm lấy đâu ra nhiều lúa mạch như thế?"

"Xin bệ hạ hãy ra lệnh mở kho lương thực, rồi mời ông ta tự đếm và nhận số lúa mạch đó".

"Như thế thì cần bao nhiêu thời gian?"

Vị đại thần quản lý kho lương tính toán một lát rồi trả lời: "Giả sử mỗi giây ông ta có thể đếm được hai hạt lúa mạch, mỗi ngày đếm 12 tiếng đồng hồ, tức là 43200 giây, thì trong vòng 10 năm ông ta mới đếm được 20 mét khối lúa mạch. Để đếm được hết số lúa mạch mà ông ta yêu cầu thì phải mất 290 tỉ năm. Ông ta có thể sống được bao nhiêu năm chứ? Hơn nữa cuộc sống đơn điệu, tẻ nhạt như thế sẽ giày vò con người, cứ như thế chẳng phải ông ta sẽ đoản thọ hay sao? Cho nên, thần trộm nghĩ, ông ta không có ý định muốn bệ hạ trọng thưởng cho ông ta số lúa mạch mà ông ta sẽ không thể nào có được, mà ông ta chỉ muốn thử xem đất nước chúng ta còn ai thông minh hơn ông ta hay không."

Bây giờ chúng ta thử viết một chương trình bằng Java để tính số lượng hạt lúa mạch.

```
public class Main {
    public static void main(String[] args) {
        long number = 1;
        for (int i = 0; i < 64; i++)
            number *= 2;
        number -= 1;
        System.out.println("2^64 - 1 = " + number);
    }
}
```

Tuy nhiên khi thực thi, kết quả không như mong muốn.

Điều này là vì kiểu dữ liệu long được cung cấp bởi Java chỉ chứa được giá trị trong khoảng từ -9223372036854775808 đến 9223372036854775807.

Trong bài tập lớn này, sinh viên sẽ hiện thực danh sách liên kết được dùng để biểu diễn các số nguyên rất lớn cùng với một số phép toán đơn giản trên cấu trúc dữ liệu này.

2. Các file được cung cấp

Sinh viên nhận file **Assignment_DAA_2020.zip** từ giảng viên dạy thực hành hoặc trang elit của môn học. Khi giải nén file này, sẽ có các file sau:

testcase	folder chứa các testcase input và output
Main.java	Chương trình chính
Utils.java	Chứa một số hàm hỗ trợ đọc ghi từ file
DigitList.java	Mã nguồn thực hiện bởi sinh viên
BigInteger.java	Mã nguồn thực hiện bởi sinh viên
input.txt	testcase chạy mã nguồn
Assignment_DSA_2020.pdf	Đề bài tập lớn

Lưu ý rằng ***sinh viên không được phép thay đổi file Main.java và Utils.java khi thực hiện chương trình***. Ngoài ra, các hàm do sinh viên viết không được xuất bất kì dữ liệu nào ra màn hình khi được thực thi.

Để dịch và thực thi chương trình, sinh viên chứa cả 5 files *Main.java*, *Utils.java*, *DigitList.java*, *BigInteger.java* và *input.txt* trong cùng một thư mục; sau đó chỉ cần dịch và thực thi duy nhất file Main.java. Mọi công việc cần phải làm sẽ được hiện thực trong file DigitList.java và BigInteger.java.

Ví dụ: Để dịch và thực thi chương trình, thực thi các lệnh sau:

```
$ javac Main.java
$ java Main
```

Sau khi thực thi, file output.txt sẽ được ghi ra là kết quả tính toán từ file input.txt

3. Hướng dẫn làm bài

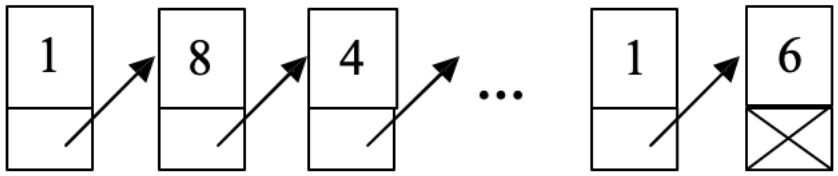
Sinh viên cần phải hiện thực 2 lớp là class DigitList và class BigInteger. Một số phương thức của 2 lớp này đã được cung cấp sẵn, một số phương thức yêu cầu sinh viên phải hoàn thiện.

Lưu ý:

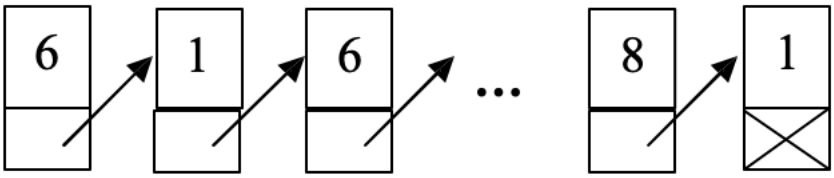
- **Sinh viên KHÔNG được import bất cứ thư viện nào khác để xử lý các phép toán trong bài.**
- **Sinh viên KHÔNG tự ý chỉnh sửa tên các phương thức hoặc nội dung mà đề không yêu cầu chỉnh sửa.**

3.1. Lớp DigitList

Để biểu diễn các số nguyên rất lớn, ta thường lưu chúng dưới dạng danh sách các chữ số. Ví dụ, có thể biểu diễn $2^{64} = 18446774073709551616$ bằng danh sách.



Mặc dù cách này hoàn toàn giống với cách viết một số thông thường nhưng để tính toán, ta có cách lưu trữ đảo ngược thuận tiện hơn rất nhiều.



Có thể thấy được điều này qua định nghĩa đệ quy của một số được biểu diễn bằng các chữ số của nó theo dạng thập phân:

$$n = (n\%10) + 10 \times (n/10)$$

Phần tử đầu tiên của danh sách là n%10, phần còn lại là n/10. Có thể thay 10 bằng một số khác và biểu diễn con số theo hệ cơ số bất kỳ nhưng hệ thập phân là quen thuộc nhất nên ta vẫn sử dụng hệ này.

Ta sử dụng lớp digitList để biểu diễn một số nguyên dương.

```java
public class DigitList {
    private int digit;
    private DigitList nextDigit;

    public DigitList() {
        this.digit = 0;
        this.nextDigit = null;
    }

    public DigitList(int d, DigitList next) {
        this.digit = d;
        this.nextDigit = next;
    }

    public int getDigit() {
        return this.digit;
    }

    public DigitList getNextDigit() {
        return this.nextDigit;
    }

    public void setDigit(int digit) {
        this.digit = digit;
    }

    public void setNextDigit(DigitList next) {
        this.nextDigit = next;
    }
    ...
}
```

Lớp DigitList chỉ dùng để biểu diễn các chữ số và lớp BigInteger phía sau sẽ biểu diễn số âm hay số dương.

Trong lớp **DigitList** có các hàm thành phần, như hàm tính độ dài danh sách, hàm copy danh sách, cũng như hàm **DigitList leftDigits (int n)** trả về danh sách chứa n phần tử tận cùng bên trái, hàm **DigitList rightDigits (int n)** trả về danh sách chứa n phần tử tận cùng bên phải. Ngoài ra, trong tập tin DigitList.java, có một số hàm cung cấp sẵn như:

- **DigitList copy()**: copy thành danh sách DigitList mới.
- **int length()**: trả về độ dài danh sách.
- **void print()**: in danh sách.
- **int compareDigitLists(DigitList L1, DigitList L2)**: so sánh hai số nguyên được biểu diễn bởi L1 và L2. Với L1, L2 ở dạng DigitList, hàm này sẽ trả về 0 nếu hai số bằng nhau, trả về 1 nếu L1 > L2, trả về -1 nếu L1 < L2.
- **DigitList addDigitLists(int c, DigitList L1, DigitList L2)**: cộng hai danh sách. Hàm này được viết dưới dạng đệ quy. Tuy nhiên, cần xử lý chữ số nhớ khi cộng. Để xử lý số nhớ một cách tổng quát, ta thêm tham số c để truyền giá trị nhớ. Lưu ý rằng, nếu không danh sách nào rỗng, ta cộng hai chữ số đơn vị là L1.getDigit() và L2.getDigit() với chữ số nhớ, sau đó cộng chữ số của tổng này với chữ số đơn vị

của kết quả. Phần kết quả còn lại được tính bằng cách cộng đệ quy chữ số hàng chục của tổng đó với phần đuôi của hai danh sách.

- **DigitList trimDigitList(DigitList L)**: có chức năng loại bỏ những chữ số 0 ở cuối danh sách. Dùng cho các phép toán cần loại bỏ số 0 ở cuối danh sách.
- **DigitList leftDigits(int n)**: trả về danh sách chứa n phần tử tận cùng bên trái, dùng cho phép nhân.
- **DigitList rightDigits(int n)**: trả về danh sách chứa n phần tử tận cùng bên phải, dùng cho phép nhân.
- **DigitList digitize(String str)**: chuyển một số dương thành một danh sách DigitList và cho phép đọc được chuỗi ký tự biểu diễn giai thừa và số mũ. Sinh viên sẽ hoàn thiện thêm code của phương thức này ở phần sau.

3.2. Lớp BigInteger

Ta sử dụng lớp **DigitList** vừa mô tả ở trên để định nghĩa lớp **BigInteger**. Một đối tượng của lớp này sẽ chứa một danh sách **DigitList** (digits) cùng với một số nguyên (sign) dùng để biểu diễn dấu, nhờ vậy lớp **BigInteger** có thể biểu diễn được cả số nguyên âm.

```
1  public class BigInteger {
2      private DigitList digits;
3      private int sign;
4
5      public BigInteger() {
6          this.digits = null;
7          this.sign = 1;
8      }
9
10     public BigInteger(DigitList L) {
11         this.digits = L;
12         this.sign = 1;
13     }
14
15     public BigInteger(int i, DigitList L) {
16         this.digits = L;
17         this.sign = sgn(i);
18     }
19
20     public BigInteger(int i) {
21         this.digits = DigitList.digitize(Math.abs(i));
22         this.sign = sgn(i);
23     }
24
25     public BigInteger(String str) {
26         if (str.charAt(0) == '-') {
27             str = str.substring(1);
28             this.digits = DigitList.digitize(str);
29             this.sign = -1;
30         }
31         else {
32             this.digits = DigitList.digitize(str);
33             this.sign = 1;
```

```
34          }
35      }
36
37      public DigitList getDigits() {
38          return this.digits;
39      }
40
41      public int getSign() {
42          return this.sign;
43      }
44
45      public void setDigits(DigitList digits) {
46          this.digits = digits;
47      }
48
49      public void setSign(int sign) {
50          this.sign = sign;
51      }
52      private int sgn(int i) {
53          if (i < 0)
54              return -1;
55          else
56              return 1;
57      }
58      ...
59  }
```

Lớp đã cung cấp sẵn một số phương thức như:

- **int length()**: tính chiều dài của số nguyên, tức là tính số lượng các chữ số
- **BigInteger copy()**: copy số nguyên
- **int sgn(int i)**: trả về dấu của i
- **BigInteger trimDigit()**: loại bỏ các chữ số 0 ở cuối danh sách digits (tức là các số không ở đầu số nguyên)
- 5 hàm khởi tạo, trong đó hàm khởi tạo **BigInteger (String str)** được dùng để tạo ra danh sách liên kết từ một chuỗi ký tự. Hàm khởi tạo này gọi hàm **DigitList digitize(String str)** của lớp DigitList.
- **BigInteger leftDigits(int n)**: trả về đối tượng **BigInteger** có n chữ số là n chữ số bên trái của đối tượng được gọi. Được dùng cho phép nhân sẽ được mô tả bên dưới.
- **BigInteger rightDigits(int n)**: trả về đối tượng **BigInteger** có n chữ số là n chữ số bên phải của đối tượng được gọi. Được dùng cho phép nhân sẽ được mô tả ở phần dưới.
- **BigInteger shift(int n)**: trả về đối tượng Integer là kết quả của phép dịch n chữ số của đối tượng được gọi. Chức năng của phương thức này là thực hiện phép tính nhân với 10^n. Được dùng cho phép nhân sẽ được mô tả ở phần dưới.

4. Định dạng của file số liệu nhập

Dữ liệu nhập của chương trình được chứa trong file mang tên *input.txt*. File này chứa các thông tin như sau:

- Hàng đầu tiên: cho biết tập tin có bao nhiêu toán hạng
- Các hàng từ thứ 2 đến $2+n$, mỗi hàng chứa một toán hạng
- $n-1$ hàng tiếp theo mỗi hàng chứa một toán tử

Phép toán được thực hiện không xét đến độ ưu tiên, tức là lấy toán hạng thứ nhất thao tác với toán hạng thứ hai sau đó kết quả sẽ thao tác với toán hạng thứ ba, v.v... Kết quả cuối cùng được ghi vào file *output.txt*.

Ví dụ 1: File số liệu này chứa 6 toán hạng và 5 toán tử, chương trình cần phải tính ra kết quả của phép toán sau:

$$((((123456789 + 12345678) * 1234567) – 123456) + 12345) * 1234$$

Ví dụ 2: Toán hạng ngoài việc chứa các chữ số từ 0 đến 9 cũng có thể chứa các ký hiệu (!) biểu diễn giai thừa, (^) biểu diễn số mũ:

Trong trường hợp này, mỗi khi đọc đến toán hạng chứa (!) hoặc (^), chương trình sẽ tính ra giá trị của toán hạng này. Chương trình cần phải tính ra kết quả của phép toán sau:

$$((((123456789 + 12345678) * (1234567\text{^}2)) - 123456) + (12345\text{^}8)) * (12!)$$

5. Các bước tiến hành

Bước 1: Đọc hiểu và chạy thử mã nguồn được cung cấp

- Trong tập tin Utils.java, có các hàm **ArrayList<Object> readFileInput(String filePath)** dùng để đọc file *input.txt*, hàm **void writeFileOutput(BigInteger number, String filePath)** dùng để xuất kết quả ra file *output.txt*.
- Trong hàm **main()**, các bước tiến hành như sau: đầu tiên đọc file *input.txt*, sau đó gọi hàm **computeValue(...)** của lớp *BigInteger* để tính toán, cuối cùng ghi kết quả ra file *output.txt*. Sinh viên sẽ dần hoàn thiện hàm **computeValue(...)** ở những bước sau.
- Chạy thử với file *input.txt*
- Kiểm tra kết quả trong file *output.txt*

Bước 2: Hoàn thiện phép tính cộng và hiện thực phép tính trừ (6 điểm)

Chương trình mẫu mới thực hiện được phép cộng các số cùng dấu. Ở bước này, sinh viên sẽ tự hiện thực phép trừ và hoàn tất phần cộng trái dấu.

Việc hiện thực phép trừ cũng gần giống như phép cộng nhưng có hai điểm khác biệt quan trọng. Thứ nhất, ý nghĩa của giá trị nhớ khác trước. Thứ hai, do phép trừ có thể tạo ra các số 0 đứng đầu (nghĩa là chúng ở đuôi danh sách) nên trong kết quả cần bỏ chúng đi. Ví dụ, nếu lấy 11 021 948 trừ cho 11 021 932, thì các chữ số 0 thừa ở kết quả sẽ phải bỏ đi.

Các công việc cụ thể cần làm:

- Hiện thực hàm **DigitList subDigitLists(int c, DigitList L1, DigitList L2)** (trong file *DigitList.java*). Hàm này có chức năng trừ hai danh sách cho nhau.
- Sử dụng hàm **DigitList trimDigitList(DigitList L)** để xử lý kết quả của phép trừ.
- Hoàn thiện hàm **BigInteger computeValue(...)** (trong file *BigInteger.java*).
- Hoàn thiện phương thức **BigInteger add(BigInteger other)** của lớp **BigInteger** (trong file *BigInteger.java*). Hiện tại phương thức này mới chỉ thực hiện được phép cộng các số cùng dấu.
- Hiện thực phương thức **BigInteger sub(BigInteger other)** của lớp **BigInteger** (trong file *BigInteger.java*).
- Tự tạo file input.txt để kiểm chứng công việc đã làm. Kết quả dưới là một ví dụ minh họa.

$$((12345678 - 123456789) + 123456) + (-12345) = -111000000$$

Bước 3: Hiện thực phép nhân (2 điểm)

Trong bước này, sinh viên hiện thực phép nhân cho lớp BigInteger. Để nhân x và y, giả sử x có $l \geq 2$ chữ số và y có nhiều nhất l chữ số. Gọi $x_0, x_1, x_2, ..., x_{l-1}$ là các chữ số của x và $y_0, y_1, y_2, ..., y_{l-1}$ là các chữ số của y (trong trường hợp số lượng chữ số của y nhỏ hơn của x, ta sẽ thêm các chữ số 0 vào cuối danh sách biểu diễn y, để cho x và y đều có cùng số chữ số). Ta có:

$$x = x_0 + 10x_1 + 10^2x_2 + ... + 10^{l-1}x_{l-1}$$

$$y = y_0 + 10y_1 + 10^2y_2 + ... + 10^{l-1}y_{l-1}$$

chia x và y thành hai phần: n chữ số bên trái và các chữ số còn lại (trong đó $n = l/2$):

$$x = x_{left} + 10^n x_{right}$$
$$y = y_{left} + 10^n y_{right}$$

Tích số $x \times y$ có thể được viết thành:

$$x \times y = (x_{left} + 10^n x_{right}) \times (y_{left} + 10^n y_{right})$$
$$= x_{left} \times y_{left} + 10^n(x_{right} \times y_{left} + x_{left} \times y_{right}) + 10^{2n} x_{right} \times y_{right}$$

Với phân tích ở trên có thể thấy việc sử dụng giải thuật đệ quy sẽ đơn giản và ngắn gọn. Các công việc cụ thể cần làm:

- Hiện thực phương thức **BigInteger mul(BigInteger other)** của lớp **BigInteger** (trong file *BigInteger.java*) dựa trên phân tích ở trên với sự hỗ trợ của 3 phương thức **leftDigits(...)**, **rightDigits(...)** và **shift(...)**.
- Hoàn thiện hàm **BigInteger computeValue(...)** (trong file *BigInteger.java*).
- Tự tạo file input.txt để kiểm chứng công việc đã làm. Kết quả dưới là một ví dụ minh họa.

$$((12345678 \times 123456789) + 123456) \times (-12345) = -18815727596967608310$$

Bước 4: Hiện thực phép giai thừa, số mũ và đọc được chuỗi số biểu diễn giai thừa, số mũ (2 điểm)

Sinh viên dựa vào phép nhân đã hiện thực ở bước 3 để hiện thực phương thức tính số mũ **BigInteger pow(BigInteger X, BigInteger Y)** và tính giai thừa **BigInteger factorial(BigInteger X)** (trong file BigInteger.java).

Phương thức **DigitList digitize(String str)** (trong file DigitList.java) cho phép đọc được chuỗi ký tự biểu diễn giai thừa và số mũ. Phương thức này đã được hiện thực phần chuyển một số dương thành một danh sách DigitList và đọc chuỗi có ký tự (^) để xử lý phép lũy thừa. Sinh hiện tự hiện thực thêm phần đọc chuỗi có ký tự (!) để xử lý chuỗi có phép giai thừa. Ví dụ:

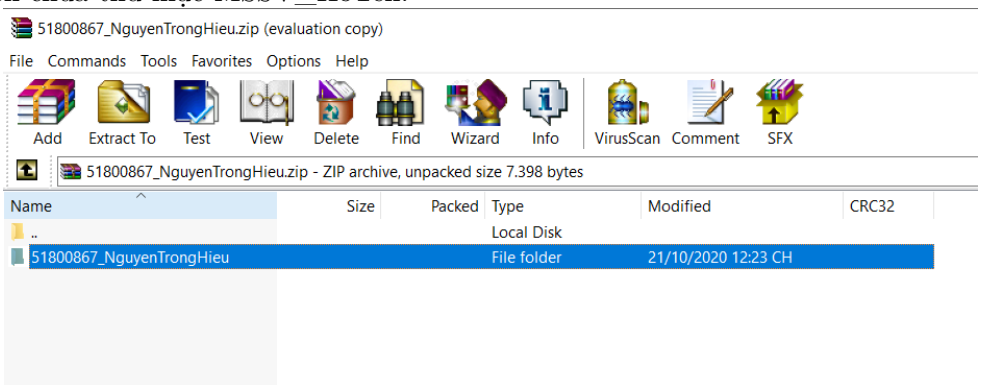
$$(((123\hat{}3) \times (12!)) + 123456) \times (12345\hat{}4) = 207022083718920907977141123600000$$

* Testcase sẽ không xảy ra trường hợp 0^0 và số $n^{-a}(a > 0)$.
* Giả sử $-12!$ sẽ được xem là $-(12!)$, tương tự -12^2 là $-(12^2)$.

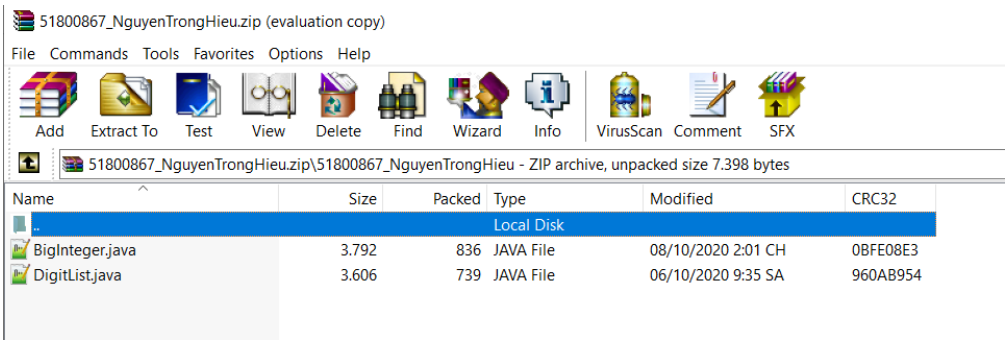
6. Hướng dẫn nộp bài

- Kiểm tra kỹ bài làm phải chạy được với các file được cung cấp sẵn, nếu bài làm trong package thì phải xóa package đi và kiểm tra lại chạy với **cmd** trước khi nộp.
- Sau khi hoàn tất và kiểm tra kỹ bài làm, sinh viên tạo thư mục đặt tên **MSSV_HoTen** (họ tên viết liền không dấu, ví dụ: **51800867_NguyenTrongHieu**), sau đó chép 2 file **DigitList.java** và **BigInteger.java** chứa bài làm vào thư mục trên.
- Sau đó nén thư mục lại với tên tương tự dưới dạng **.zip** (KHÔNG nén .rar) và nộp lại cho giảng viên thực hành.
- Hạn chót nộp bài: **đến hết 23h59 ngày 22/11/2020**

Ví dụ một bài nộp đúng theo hướng dẫn sẽ là file nén MSSV_HoTen.zip và trong file nén chứa thư mục MSSV_HoTen:



Trong thư mục MSSV_HoTen chứa 2 file bài làm:



7. Chấm bài

Bài tập lớn sẽ chấm tự động bằng máy:

- Nếu bài có lỗi biên dịch chương trình → 0 testcase → 0 điểm.
- Nếu bài không có lỗi biên dịch chương trình → Bắt đầu chấm.
- Sinh viên được cung cấp 10 testcase mẫu trong thư mục testcase. Bộ testcase chấm bài sẽ hoàn toàn khác bộ testcase mẫu nhưng có tính chất tương tự. Do đó sinh viên nên kiểm tra chương trình của mình chạy đúng tất cả testcase mẫu cũng như tự nghĩ ra các trường hợp khác.
- **Sinh viên tự làm bài, tất cả các trường hợp phát hiện đạo văn (kể cả trường hợp sinh viên không hiểu code của mình) đều sẽ đưa về khoa xử lý.**

—HẾT—