Reporte de Complejidades Proyecto Estructuras de Datos

Autor: Miguel Bejarano Perdomo

Codigo: 8959501

Fecha: 28/05/2023

Implementación del TAD BigInteger con Vectores

Operaciones y su complejidad temporal:

1. **Constructor por defecto**

- Complejidad: O(1)
- Descripción: Esta funcion crea un objeto de la clase BigInteger con el atributo de negativo como falso, su complejidad es lineal ya que solo crea el espacio en memoria

2. **Constructor por cadena de dígitos**

- Complejidad: O(N)
- Descripción: Esta función crea un objeto de la clase BigInteger cuya construcción pone las cifras más significativas al final del vector de dígitos. La complejidad es lineal ya que recorre la cadena de dígitos de entrada y va almacenándolos en un vector.

3. **Constructor de copia**

- Complejidad: O(1)
- Descripción: Esta función se encarga de copiar los atributos de un BigInteger a otro BigInteger. La complejidad es constante ya que solo almacena en memoria los mismos atributos del BigInteger que se pasa como parámetro.

4. **Sobrecarga del Operador +**

- Complejidad: O(N)
- Descripción: Esta sobrecarga se encarga de hacer el proceso para sumar dos objetos BigInteger y retornar un tercer objeto como resultado. La complejidad es lineal ya que debe recorrer todos los elementos del vector de dígitos (tamaño N) para realizar la suma dígito por dígito, manejar el acarreo y definir lo que se mete al BigInteger resultado. También realiza un previo analisis de los símbolos y determina si se debe realizar una suma o una resta, en cada caso se realiza la operación correspondiente, por lo que técnicamente su complejidad depende de la de la resta, pero como la complejidad de la resta también es lineal, no hay problema.

5. **Sobrecarga del Operador ***

- Complejidad: O(N*M)
- Descripción: Esta sobrecarga se encarga de hacer el proceso para multiplicar dos objetos BigInteger y retornar un tercer objeto como resultado. La complejidad es N*M ya que N y M son los tamaños de los vectores de dígitos de los BigIntegers. El algoritmo recorre por cada dígito del primer BigInteger todos los dígitos del segundo BigInteger y realiza las operaciones correspondientes.

6. **Sobrecarga del Operador -**

- Complejidad: O(N)
- Descripción: Esta sobrecarga se encarga de hacer el proceso para restar dos objetos BigInteger y retornar un tercer objeto como resultado. La complejidad es lineal ya que debe recorrer todos los elementos del vector de dígitos (tamaño N) para realizar la resta dígito por dígito, se encarga manejar el acarreo y definir lo que se mete al BigInteger resultado. Adicional a esto se predefine en un principio si la operación a realizar es una resta, o una suma, tomando en cuenta que se trata de un analisis de signos. Mas esto no afecta la complejidad significativamente.

7. **Sobrecarga del Operador /**

- Complejidad: O(N*M)
- Descripción: Esta sobrecarga se encarga de hacer el proceso para dividir dos objetos BigInteger por medio de un algoritmo de división larga y retornar un tercer objeto como resultado. La complejidad es O(N*M) (Donde N es el tamaño del vector de digitos del dividendo y M la cantidad de iteraciones para encontrar el cociente más grande posible) ya que el algoritmo recorre en sentido contrario los dígitos del dividendo y va metiéndolos en los dígitos del BigInteger llamado resto. Para cada dígito del dividendo, se busca el mayor cociente posible, lo cual implica recorrer los dígitos del divisor. Este algoritmo es más eficiente que uno totalmente implementado por restas ya que para los digitos va buscando su cociente, reduciendo significativamente el número de restas que debe realizar.

8. **Sobrecarga del Operador %**

- Complejidad: O(N*M)
- Descripción: Esta sobrecarga se encarga de hacer el proceso para obtener el modulo de dos objetos BigInteger por medio del algoritmo de división larga implementado en la sobrecarga de /, La complejidad por ende es básicamente la misma, solo variando ligeramente en la inserción de elementos a un vector. Lo cual realmente no afecta de manera significa la complejidad, destaco que N y M son las mismas variables de la sobrecarga de /.

9. **Sobrecarga del Operador ==**

- Complejidad: O(N)
- Descripción: Esta sobrecarga se encarga de verificar si dos objetos BigInteger son iguales, comparando cada uno de los dígitos de sus vectores. La complejidad es lineal ya que recorre los vectores de dígitos de ambos objetos y realiza una comparación dígito por dígito. La complejidad es O(N) ya que recorre por el vector de digitos del 1 de los dos BigInteger 's, podría hacerse ligeramente más eficiente si se conoce el menor y se recorre por el menor. Pero N representa el tamaño del vector de digitos del primer BigInteger.

10. Sobrecarga del Operador <

- Complejidad: O(N)
- Descripción: El algoritmo primero revisa si con los signos puede definir o no cual es de los dos BigInteger es menor, en caso de poder lo define. En caso de que tengan tamaños diferentes, se analizan sus signos igualmente, y se define cuál de los dos tiene mayor tamaño y esto en conjunto con el signo define cual es mayor o menor. Si ninguna de estas condiciones se dispara implica que tienen el mismo signo, y mismo tamaño, por lo que se analiza digito a digito entre ambos BigInteger, y a partir de ahí define cual es mayor, como se hace un recorrido del vector de digitos en reversa la complejidad es O(N) donde N representa el tamaño del vector de digitos del primer BigInteger.

11. Sobrecarga del Operador <=

- Complejidad: O(N)
- Descripción: El algoritmo primero revisa si con los signos puede definir o no cual es de los dos BigInteger es menor, en caso de poder lo define. En caso de que tengan tamaños diferentes, se analizan sus signos igualmente, y se define cuál de los dos tiene mayor tamaño y esto en conjunto con el signo define cual es mayor o menor. Si ninguna de estas condiciones se dispara implica que tienen el mismo tamaño, por lo que se analiza digito a digito entre ambos BigInteger, y a partir de ahí define cual es mayor, como se hace un recorrido del vector de digitos en reversa la complejidad es O(N) donde N representa el tamaño del vector de digitos del primer BigInteger.

12. Operación Add

- Complejidad: O(N)
- Descripción: Como solo empleo la sobrecarga del operador +, en cuanto a complejidad temporal no deberia haber diferencias respecto a la sobrecarga de +

13. Operación Product

- Complejidad: O(N*M)
- Descripción: Como solo empleo la sobrecarga del operador *, en cuanto a complejidad temporal no deberia haber diferencias respecto a la sobrecarga de *

14. Operación Substract

- Complejidad: O(N)

- Descripción: Como solo empleo la sobrecarga del operador -, en cuanto a complejidad temporal no deberia haber diferencias respecto a la sobrecarga de –

15. Operación Quotient

- Complejidad: O(N*M)
- Descripción: Como solo empleo la sobrecarga del operador /, en cuanto a complejidad temporal no deberia haber diferencias respecto a la sobrecarga de /

16. Operación Remainder

- Complejidad: O(N*M)
- Descripción: Como solo empleo la sobrecarga del operador %, en cuanto a complejidad temporal no deberia haber diferencias respecto a la sobrecarga de %

17. Operación Pow

- Complejidad O(N*LogM)
- Descripción: La complejidad es O(N*LogM) donde M representa el exponente al cual hay que elevar el BigInteger, y N representa el número de iteraciones que deben hacerse para poder reducir el exponente a 0 y haber calculado la potencia, es un procedimiento que radica su utilidad En utilizar las propiedades de la potencia y cada que se pueda calcular el cuadrado de lo que sea base en ese momento, y reducir el exponente a la mitad

18. Operación toString()

- Complejidad O(N)
- Descripción: Solo recorro por el tamaño del vector de digitos del BigInteger que estoy convirtiendo, luego uso el ASCII del 0 para poder convertirlo. Por el recorrido N representa el tamaño del vector de digitos del BigInteger a convertir.

19. Operación estática SumarListaValores

- Complejidad O(N*M)
- Descripción: Recorro la lista de BigIntegers y voy aplicando la suma a cada BigInteger, por lo que el recorrido es O(N) siendo N el tamaño de la lista de BigIntegers Luego el hacer las M sumas tiene una complejidad de O(M), dándonos una complejidad resultante de O(N*M).

20. Operación estática multiplicarListaValores

- Complejidad O(N*M*L)
- Descripción: Recorro la lista de BigIntegers y voy aplicando la multiplicación a cada BigInteger, por lo que el recorrido es O(N) siendo N el tamaño de la lista de BigIntegers. Luego el hacer las M multiplicaciones tiene una complejidad de O(M*L), donde M representa el tamaño de los digitos de cada primer BigInteger a multiplicar y L el tamaño de los digitos de cada siguiente BigInteger a multiplicar, obteniendo una complejidad resultante de O(N*M*L).

BiggerSmaller

La complejidad de este problema es de O(N) donde N representa el tamaño de la entrada de la cadenas de digitos a analizar, luego por cada pareja de digitos se extrae la parte

entera y la parte decimal de cada número, posterior a eso se realizan las comparaciones con los operadores < y == que tiene una complejidad de O(M) cada uno, en caso de que no se logre definir cual es mayor que el otro solo con la comparación directa, se pasa a los decimales y se analiza la resta de ambos, la resta tiene una complejidad O(L), por lo tanto la complejidad resultante es de O(N+2M), donde M representaría el tamaño de los digitos de cada BigInteger.