Ana Gabriela Argüello Cedeño Mayo-30-2023

Proyecto Estructura de Datos

Análisis de complejidad de los algoritmos del TAD bigInteger

```
BigInteger::BigInteger(consta string& bignum)
```

Esta operación tiene una complejidad de O(n), donde n es tamaño de la string que se da como parámetro, esta operación posee un for que itera desde 0 hasta n-1, y es la encargada de convertir la string de números a un bigInteger. Dentro del for se hace uso de la función push_back, pero al bigInteger ser manejado como una lista tipo deque, la complejidad de esta función de agregar será constante, por tal motivo no hace ningún cambio a la complejidad ya antes dicha.

```
BigInteger BigInteger::operator/(BigInteger &bigTwo)
```

La operación de sobrecarga del operador / , en su peor caso, tendrá una complejidad será de O((n/2)*n), donde n es el número por dividir, este caso es cuando el divisor es 2 ya que 2 es el mínimo número por el que se puede dividir un número sin contar el 1, en el caso en el que el divisor fuese 3 su complejidad disminuir, ya que no tendrá que iterar tantas veces, pero en 2 está a su máximo nivel. Esa parte tendría una complejidad de O(n/2), pero al hacer llamado a la función substract, la complejidad de este algoritmo cambia puesto que la operación de resta tiene una complejidad O(n).

```
BigInteger BigInteger::operator%(BigInteger &bigTwo)
```

Esta operación al igual que la anterior tendrá una complejidad O((n/2)*n), por los motivos anteriormente mencionados, a diferencia de la división, la operación del residuo da el número que quedo después de hacer todas las restas necesarias para obtener el cociente.

```
BigInteger BigInteger::operator*(BigInteger &bigTwo)
```

Esta operación de sobrecargar el operador *, tiene una complejidad O(n*m), que en el caso promedio será O(n^2),donde n es el tamaño del bigInteger, en este algoritmo se usa un ciclo anidado que recorre primero el divisor y después el dividendo para ir multiplicándolos y guardándolos en una lista aux, para posteriormente se puesto en un bigInteger.

Acá el caso promedio es cuando ambos bigIntegers tienen el mismo tamaño

```
BigInteger BigInteger::operator-(BigInteger &bigTwo)
```

Esta operación de sobrecarga del operador - tiene una complejidad de O(n), donde n es el tamaño del bigInteger principal, esto ya que en la operación se usa un ciclo que itera las n veces desde n-1 hasta 0,aunque en este algoritmo se hace uso de la función suma, esta no se encuentra dentro del for por lo que no genera modificación alguna en la complejidad.

(en esta operación si los números no tienen igual tamaño, se agregan ceros al principio hasta que igualen tamaño, esto con el motivo de poder facilitar un poco el proceso)

```
BigInteger BigInteger::operator+(BigInteger &bigTwo)
```

Esta operación de sobrecarga del operador + tiene una complejidad de O(n), donde n es el tamaño del bigInteger principal, en esta operación al igual que en la resta si los números no tienen el mismo tamaño entonces se agregan ceros al inicio con la función front, en este algoritmo se usa un ciclo que recorre los bigInteger y la suma de estos se va a una lista auxiliar. Aunque en el algoritmo se llama a la función substract, esta al no encontrarse dentro del ciclo no hace modificación a la complejidad ya mencionada.

```
bool BigInteger::operator==(BigInteger &bigTwo)
```

Esta operación de sobrecarga del operador == tiene una complejidad de O(n), donde n es el tamaño del bigInteger principal, la complejidad es así por el motivo de que se usa un ciclo que itera desde o hasta n-1, y dentro de dicho ciclo se hace comparan los bigInteger hasta que se encuentra un numero distinto, o hasta que el ciclo termine sus respectivas iteraciones es decir que i=n-1.

```
bool BigInteger::operator<(BigInteger &bigTwo)
```

Esta operación de sobrecarga del operador < tiene una complejidad de O(n), donde n es el tamaño del bigInteger principal, la complejidad es así por el motivo de que se usa un ciclo que itera desde 0 hasta n-1 o(hasta que flag==0), y dentro de dicho ciclo se hace comparan los bigInteger hasta que se encuentre algún número mayor, o hasta que el ciclo termine sus respectivas iteraciones es decir que i=n-1.

```
bool BigInteger::operator<=(BigInteger &bigTwo)
```

Esta operación de sobrecarga del operador <= tiene una complejidad de O(n), ya que, aunque a simple vista parezca constante se hace uso de la sobrecarga de los operadores < y == las cuales tienen una complejidad de O(n) respectivamente.

```
string BigInteger::toString()
```
Esta operación toString tiene una complejidad lineal de O(n), donde n es el tamaño del bigInteger principal, dentro de este for se transforman los números del bigInteger a datos tipo char y se concatenan una cadena

```
BigInteger BigInteger::sumarListaValores(deque<BigInteger>&lista)
```
Esta función tiene una complejidad de O(n^2) donde n es el tamaño de la lista, a simple vista se podría inferir que es O(n), ya que no hay ningún ciclo anidado, sin embargo, se hace uso de la sobrecarga del operador + , esta sobrecarga al tener una complejidad lineal hace que la se modifique la complejidad inicial de esta operación pasando de lineal a cuadrática.

```
BigInteger
BigInteger::multiplicarListaValores(deque<BigInteger>&lista)
```
Esta función tiene una complejidad de O(n^2) donde n es el tamaño de la lista, al igual que en la operación sumarListasValores, se puede pensar que la complejidad es O(n), ya que no hay ningún ciclo anidado, sin embargo, se hace uso de la sobrecarga del operador * ,la cual tiene una complejidad lineal O(n) ,esto hace que se modifique la complejidad inicial de esta operación pasando de lineal a cuadrática.

```
void BigInteger::pow(int num3)
```
La complejidad de esta operación O(n*(n*m)), la complejidad es esa ya que se llama a la función product , la cual posee una complejidad de O(n*m), por los motivos anteriormente dichos, al hacer uso de la dicha operación la complejidad pasa de O(n) a O(n*(n*m)).

La complejidad de las operaciones del TAD bigInteger son las mismas que las de sus operaciones de sobrecarga sin embargo hare una pequeña mención de la complejidad que posee cada operación

```
void BigInteger::dad(BigInteger &bigTwo)
```
la operación suma posee una complejidad lineal de O(n), donde n es el tamaño del bigInteger principal

```
void BigInteger::substract(BigInteger &bigTwo)
```
la operación resta posee una complejidad lineal de O(n), donde n es el tamaño del bigInteger principal

```
void BigInteger::product(BigInteger &bigTwo)
```
la operación product tiene una complejidad O(n*m), que en el caso promedio será O(n^2),donde n es el tamaño del bigInteger

```
void BigInteger::quotient(BigInteger &bigTwo)
```
la operación división posee una complejidad de O((n/2)*n), donde n es el número por dividir

```
void BigInteger::remainder(BigInteger &bigTwo)
```
la operación residuo posee una complejidad de O((n/2)*n), donde n es el número por dividir