Universidad de Buenos Aires

Facultad De Ingenier´ıa

A˜no 2021 - 1.er Cuatrimestre

Algor´ıtmos y Programacion II (95.12) ´

Trabajo Pr´actico N*~~◦~~*0

Berard, Luc´ıa Magdalena 101213 lberard@fi.uba.ar

Guglieri, Mariano Federico 99573 mguglieri@fi.uba.ar

Rubin, Ivan Eric 100577 irubin@fi.uba.ar

Sandoval, Diego Ariel 101639 dsandoval@fi.uba.ar

95.12 - Algor´ıtmos y Programaci´on II

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

´Indice

1. Introducci´on 1 1.1. L´ınea de comando . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 1.2. Formato de los archivos de entrada y salida . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

2. Dise˜no e implementaci´on del programa 2 3. Compilaci´on 12 4. Ejecuci´on del programa 13 5. Verificaci´on con Valgrind y resultados de los casos de prueba 14 6. Conclusiones 15 7. Anexo - C´odigos 15

i

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

1. Introducci´on

El objetivo fundamental del trabajo es implementar n´umeros enteros de precisi´on fija (pero arbitrariamente grande). Se entiende como precisi´on a la cantidad de d´ıgitos decimales que conforman el n´umero.

En una primera etapa, se desea representar este concepto utilizando arreglos. Para ello, se desarroll´o una clase C++ bignum siguiendo el esquema general que se muestra a continuaci´on:

|  |
| --- |
| class bignum  *{*  private:  unsigned short \*digits;  // ...  public:  // ...  friend bignum operator+(const bignum&, const bignum&);  friend bignum operator-(const bignum&, const bignum&);  friend bignum operator\*(const bignum&, const bignum&);  friend std::ostream& operator<<(std::ostream&, const bignum&); friend std::istream& operator>>(std::istream&, bignum&);  *}*; |

Como se observa en este fragmento, es preciso adem´as sobrecargar los operadores aritm´eticos de suma, resta y multiplicaci´on y los de entrada y salida con formato. Las porciones marcadas con puntos suspensivos corresponden a los restantes m´etodos y/o variables necesarias para el correcto funcionamiento del programa.

1.1. L´ınea de comando

Las opciones -i y -o permiten seleccionar los streams de entrada y salida respectivamente. Por defecto, ´estos son cin y cout. Lo mismo ocurre al recibir \-" como argumento.

Por otro lado, la opci´on -p indica el valor de la precisi´on con la que se lleva a cabo el procesamiento. Puede asumirse que los n´umeros ingresados en el stream de entrada se ajustan a dicha precisi´on, aunque podr´ıa ocurrir que el resultado de alguna de las expresiones de entrada requiera mayor cantidad de d´ıgitos.

Al finalizar, todos los programas retornan un valor nulo en caso de no detectar ning´un problema; en caso contrario, devuelven un valor no nulo

En la secci´on 4, ejecuci´on del programa, se detallar´a m´as el uso de los comandos de l´ınea. 1.2. Formato de los archivos de entrada y salida

El formato a adoptar para el stream de entrada consiste en una secuencia de cero o m´as expresiones aritm´eti cas binarias, cada una ocupando una l´ınea distinta. A su vez, la separaci´on entre cada operando y el operador puede darse con cero o m´as espacios, entendiendo por tales a los caracteres SP, \f}, \r, \t, \v.

Por otro lado, el stream de salida deber´a listar en l´ıneas distintas los n´umeros resultantes de evaluar cada expresi´on, manteniendo el mismo orden de las operaciones de la entrada.

1

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

2. Dise˜no e implementaci´on del programa

Para la creaci´on de la clase bignum, al tratarse de un n´umero entero de precisi´on fija pero arbitrariamente grande, se la pens´o como un vector de unsigned short que contenga los d´ıgitos, un bool que contenga al signo del n´umero y un unsigned short para la longitud del vector.

En cuanto a las operaciones propias de la clase se desarrollaron las pedidas por el enunciado y se agregaron otras. Las otras funciones desarrolladas se crearon con el prop´osito de facilitar la creaci´on de las operaciones b´asicas de suma resta y multiplicaci´on. Un ejemplo de estas operaciones son la sobrecarga de los operadores mayor, menor y el operador igualdad. Adem´as, se crearon funciones para darle formato a los n´umeros, como la funci´on para setear el largo, la precisi´on y el signo. Se crearon tambi´en 3 diferentes constructores para facilitar la creaci´on de bignums auxiliares en las operaciones. A continuaci´on se muestran las funciones que se ped´ıan en el desarrollo del trabajo:

1 class bignum

2 {

3 private:

4 unsigned short \*digits;

5 unsigned short len;

6 bool signo; //True: positivo - False: negativo

7

8 public:

9 //Constructores

10 bignum(void);

11 bignum(const unsigned short);

12 bignum(const string&);

13 //Destructor

14 ~bignum();

15

16 //Funciones ´utiles

17 void set\_signo(bool);

18 bool get\_signo();

19 unsigned char get\_len();

20 void set\_precision(const unsigned short);

21 friend bignum operator-(const bignum&);

22

23 //Sobrecarga de los operadores de suma, resta y multiplicaci´on

24 friend bignum operator+(const bignum&, const bignum&);

25 friend bignum operator-(const bignum&, const bignum&);

26 friend bignum operator\*(const bignum&, const bignum&);

27

28 //Sobrecarga del operador de asignaci´on

29 const bignum& operator=(const bignum&);

30 const bignum& operator=(const string&);

31

32 //Sobrecarga de los operadores de comparaci´on

33 friend bool operator==(const bignum&, const bignum&);

34 friend bool operator<(const bignum&, const bignum&);

35 friend bool operator>(const bignum&, const bignum&);

36

37 //Sobrecarga de los operadores de flujo de stream

38 friend std::ostream& operator<<(std::ostream&, const bignum&);

39 friend std::istream& operator>>(std::istream&, bignum&);

40 };

Listing 1: Declaraci´on de la clase bignum

Como se puede observar en el c´odigo, se desarrollaron 3 constructores para poder crear un bignum vac´ıo, uno a partir de una determinada longitud y otro a partir de un string:

2

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

1 bignum::bignum(void)

2 {

3 signo = true;

4 len = 10;

5 digits = new unsigned short[len];

6

7 for(size\_t i=0;i<len;i++)

8 {digits[i]=0;}

9 }

Listing 2: Constructor de un bignum vac´ıo de longitud por default de 10

1 bignum::bignum(const unsigned short a)

2 {

3 signo = true;

4 len = a;

5 digits = new unsigned short[len];

6 for(size\_t i=0;i<len;i++)

7 {digits[i]=0;}

8 }

Listing 3: Constructor de un bignum a partir de una longitud determinada

3

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

1 bignum::bignum(const string &str1)

2 {

3

4 //Saco los espacios en blanco.

5 string str;

6 for(char c:str1) if(!isspace(c)) str += c ;

7

8 if(!(str.find\_first\_not\_of("0123456789") == string::npos) && (str[0]!='-' && str[0]!='+')) 9 {

10 cerr<<"Asignacion de numero invalida"<<endl;

11 exit(1);

12 }

13

14 //Defino el signo.

15 bool hay\_signo;

16 if(str[0]=='-')

17 {

18 signo=false;

19 hay\_signo=true;

20 len=str.length()-1;

21 } else if(str[0]=='+')

22 {

23 signo=true;

24 hay\_signo=true;

25 len=str.length()-1;

26 }else{

27 hay\_signo=true;

28 len=str.length();

29 }

30 //Creo el arreglo de shorts

31 digits=new unsigned short[len];

32

33

34 for(size\_t i=0;i<len;i++)

35 {

36 digits[len-1-i]=str[len+hay\_signo-i]-ASCII\_FIX;

37 }

38 }

Listing 4: Constructor de un bignum a partir de un string

y el correspondiente destructor que se debe utilizar para evitar fallas de memoria:

1 bignum::~bignum()

2 {

3 if(digits)

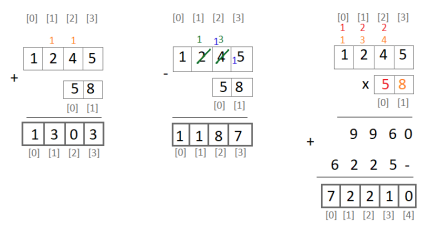
4 {delete[] digits;}

5 }

Listing 5: Constructor de un bignum a partir de un string

Para la resoluci´on de las operaciones, al tratarse de n´umeros en formato de arrays se solucion´o pens´andolo de la siguiente manera:

4

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa Figura 2.1: Operaciones

Se separaron las operaciones por d´ıgito, uni´endolas luego en el resultado final, teniendo en cuenta los casos espec´ıficos, tal como se muestra en la figura.

5

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

1 bignum operator+(const bignum& a, const bignum& b)

2 {

3

4 unsigned short n = a.len;

5 unsigned short m = b.len;

6 bignum result(max(n,m)+1);

7 if(!a.signo && b.signo){

8 bignum c = -a;

9 result=b-c;

10 return result;

11 }

12 if(a.signo && !b.signo){

13 bignum c = -b;

14 result=a-c;

15 return result;

16 }

17 if(!a.signo && !b.signo){

18 result.signo=false;

19 }

20 unsigned short carry = 0;

21 unsigned short aux = 0;

22

23 for(int i=0; i< max(n,m)+1;i++){

24

25 if((short)(m-i-1)<0 && (short)(n-i-1)<0){

26 aux=carry;

27 } else if((short)(n-i-1)<0){

28 aux=b.digits[m-i-1]+carry;

29 } else if((short)(m-i-1)<0){

30 aux=a.digits[n-i-1]+carry;

31 } else {

32 aux=a.digits[n-i-1]+b.digits[m-i-1]+carry;

33 }

34

35 if(aux>=10){

36 aux-=10;

37 carry=1;

38 } else {

39 carry=0;

40 }

41 result.digits[max(n,m)-i]=aux;

42 result.digits[0] = aux;

43 }

44

45 return result;

46 }

Listing 6: Sobrecarga del operador suma para la clase bignum

6

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

1 bignum operator-(const bignum& a, const bignum& b)

2 {

3

4 unsigned short n = a.len;

5 unsigned short m = b.len;

6 bignum result(max(n,m));

7

8

9 if(a==b){

10 return result;

11 }

12 if(a<b){

13 result = b-a;

14 result.signo=false;

15 return result;

16 }

17 if(!a.signo && b.signo){

18 bignum c=-a;

19 result = c+b;

20 result.signo=false;

21 return result;

22 }

23 if(a.signo && !b.signo){

24 bignum c = -b;

25 result=a+c;

26 return result;

27 }

28 if(!a.signo && !b.signo){

29

30 bignum c= -b;

31 bignum d= -a;

32 result = c-d;

33 return result;

34 }

35

36 short carry = 0;

37 short aux = 0;

38

39 for(size\_t i=0; i< max(n,m);i++){

40

41 if((short)(m-i-1)<0 && (short)(n-i-1)<0){

42 aux=carry;

43 } else if((short)(m-i-1)<0){

44 aux=a.digits[n-i-1]-carry;

45 } else {

46 aux=(short)a.digits[n-i-1]-(short)b.digits[m-i-1]-carry;

47 }

48

49

50 if(aux<0){

51 aux+=10;

52 carry=1;

53 } else {

54 carry=0;

55 }

56 result.digits[result.len-i-1]=aux;

57 }

58 return result;

59 }

Listing 7: Sobrecarga del operador resta para la clase bignum

7

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

1 bignum operator\*(const bignum& a, const bignum& b){

2

3 unsigned short n = a.len;

4 unsigned short m = b.len;

5 unsigned short carry = 0;

6 bignum aux(n+m);

7 bignum aux2(n+m);

8 bignum result(n+m);

9

10 for(size\_t j = 0; j < m; j++){

11 for(size\_t i = 0; i < n; i++){

12 aux.digits[n+m-1-i-j] = a.digits[n-1-i]\*b.digits[m-1-j] + carry;

13 if(aux.digits[n+m-1-i-j] >= 10){

14 carry = aux.digits[n+m-1-i-j]/10;

15 aux.digits[n+m-1-i-j] -= 10\*(aux.digits[n+m-1-i-j]/10);

16 }

17 else carry = 0;

18 }

19 aux.digits[m-1-j] = carry;

20

21 for(size\_t k = 0; k < j; k++){

22 aux.digits[n+m-1-k] = 0;

23 }

24 aux2 = aux2 + aux;

25 carry = 0;

26 }

27

28 result = aux2;

29 if((!a.signo && b.signo) ||(a.signo && !b.signo) ){

30 result.signo = false;

31 }

32 return result;

33 }

34 bool operator==(const bignum&a, const bignum&b)

35 {

36 if(a.signo==b.signo && a.len==b.len)

37 {

38 for(size\_t i = 0; i < a.len; i++)

39 {

40 if(a.digits[i]!=b.digits[i])

41 return false;

42 }

43 return true;

44 }

45 return false;

46 }

Listing 8: Sobrecarga del operador multiplicaci´on para la clase bignum

Luego, para poder observar los resultados, es necesario sobrecargar los operadores de entrada y salida: 8

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

1 ostream& operator<<(ostream& os, const bignum& num){

2

3 if(num.signo==false){

4 os << '-'; }

5 bool aux= false;

6

7 for(int i = 0; i< num.len;i++){

8

9 //saco los ceros de la izquierda

10 if(num.digits[i]==0 && aux==false){

11 if(i==num.len-1)

12 {

13 os<<'0';

14 return os;}

15 continue;

16 }

17 os << num.digits[i];

18 aux=true;

19 }

20 return os;

21 }

22

23 istream& operator>>(istream& is, bignum& num){

24

25 string s;

26 is >> s;

27 while(!(s.find\_first\_not\_of( "0123456789" ) == string::npos) && (s[0]!='-' && s[0]!='+')){ 28 cerr << "El valor ingresado no es correcto. Intente nuevamente." << endl;

29 is >> s;

30 }

31 num = s;

32 return is;

33 }

Listing 9: Sobrecarga de los operadores de flujo de stream para la clase bignum

Tambi´en se plantearon funciones extras ´utiles para el manejo de la clase, tal como:

9

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

1 //Setea el signo del bignum, true si es positivo y false si es negativo

2 void bignum::set\_signo(bool s){

3 signo=s;

4 }

5

6 //Devuelve el signo del bignum, true si es positivo y false si es negativo

7 bool bignum::get\_signo(){

8 return signo;

9 }

10

11 //Devuelve la longitud del bignum

12 unsigned char bignum::get\_len(){

13 return len;

14 }

15

16 //Setea la precision del bignum.

17 //Redondea para arriba las cifras significativas y deja el resto de los valores en cero. 18 void bignum::set\_precision(const unsigned short precision){

19 if(precision>0 && len>precision)

20 {

21 for(int i=0; i<len-precision; i++){

22 if(digits[precision]>=5)

23 digits[precision-1]++;

24 digits[precision+i]=0;

25 }

26 }

27 }

Listing 10: Funciones ´utiles

y la sobrecarga de los operadores booleanos de comparaci´on:

10

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

1 //Sobrecarga del operador == para la comparacion entre dos bignums

2 bool operator==(const bignum&a, const bignum&b)

3 {

4 if(a.signo==b.signo && a.len==b.len)

5 {

6 for(size\_t i = 0; i < a.len; i++)

7 {

8 if(a.digits[i]!=b.digits[i])

9 return false;

10 }

11 return true;

12 }

13 return false;

14 }

15

16 //Sobrecarga del operador < para la comparacion entre dos bignums

17 bool operator<(const bignum& a, const bignum& b)

18 {

19 if(a.len<b.len)

20 return true;

21 else if(b.len<a.len)

22 return false;

23 else{

24 for(int i=0; i<a.len; i++)

25 {

26 if(a.digits[i]<b.digits[i]){

27 return true;

28 }else if(b.digits[i]<a.digits[i])

29 return false;

30 }

31 }

32 return false;

33 }

34

35 //Sobrecarga del operador > para la comparacion entre dos bignums

36 bool operator>(const bignum& a, const bignum& b)

37 {

38 if(a.len>b.len)

39 return true;

40 else if(b.len>a.len)

41 return false;

42 else{

43 for(int i=0; i<a.len; i++)

44 {

45 if(a.digits[i]>b.digits[i]){

46 return true;

47 }else if(b.digits[i]>a.digits[i])

48 return false;

49 }

50 }

51 return false;

52 }

Listing 11: Sobrecarga de los operadores de comparaci´on

11

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

3. Compilaci´on

En cuanto al proceso de compilaci´on se cre´o un archivo Makefile para obtener el ejecutable a partir de los archivos fuente:

1 PROGRAM=tp0

2 CC := g++

3 FLAGS := -Wall -pedantic -g

4 LDLFLAGS=-lm

5 GREEN= \e[92m

6 NORMAL= \e[0m

7

8 all:

9 @echo "$(GREEN)Compilando ...$(NORMAL)"

10 $(MAKE) tp0

11 @echo "$(GREEN)Termin´o$(NORMAL)"

12

13 cmdline.o: cmdline.cpp cmdline.h

14 $(CC) $(FLAGS) cmdline.o

15

16 bignum.o: bignum.cpp bignum.h

17 $(CC) $(FLAGS) bignumm.0

18

19 tp0: cmdline.cpp bignum.cpp tp0.cpp

20 $(CC) $(FLAGS) $^ -o tp0

21

22 clean:

23 @echo "$(GREEN)Limpiando ...$(NORMAL)"

24 rm -vf \*.o $(PROGRAM)

25 @echo "$(GREEN)Listo!$(NORMAL)"

26

27 gdb: tp0

28 gdb ./tp0

29

30 valgrind: tp0

31 valgrind --leak-check=full --show-leak-kinds=all ./tp0

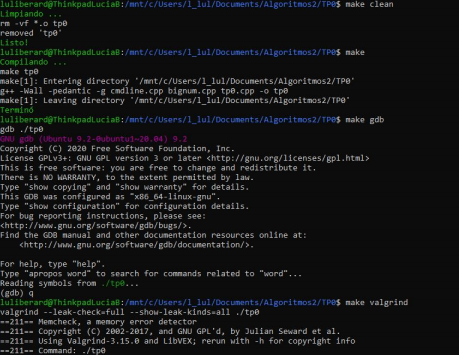
Listing 12: Archivo Makefile

De esta manera se simplifica el proceso de compilaci´on ya que solo se necesita escribir el comando make. Otros comandos ´utiles son:

make clean: borra todos los archivos .o generados.

make gdb: para debuggear con gdb.

make valgrind: ejecuta el programa con valgrind para verificar que no haya fallas de memoria. 12

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa Figura 3.1: Compilaci´on utilizando los comandos del Makefile

4. Ejecuci´on del programa

Luego de compilar el programa, se prosigue a ejecutar el mismo. Esto se realiza escribiendo echo, luego la operaci´on que se quiera realizar, un pipeline, el archivo ejecutable ./tp0 y opcionalmente el comando de la precisi´on (-p) con su respectivo valor. Por ejemplo:

echo "-1 - 5" | ./tp0 -p 2

echo -189\*50 | ./tp0 -p 6

La precisi´on ingresada afectar´a s´olo a los valores de entrada con los que se operar´a, es decir, que el resultado podr´a tener una precisi´on distinta de ser necesario, seg´un la operaci´on a realizar.

Otra de las opciones disponibles, es aquella que se ejecuta con el comando -i para realizar cuentas escritas en un archivo .txt aparte:

./tp0 -p 2 -i cuentas.txt

En el ejemplo, se realizar´an las operaciones le´ıdas del archivo *cuentas.txt*, las cuales deber´an ocupar una l´ınea cada una, de lo contrario se emitir´a un error de operaci´on.

Por otra parte, la opci´on correspondiente al comando -o puede utilizarse para abrir un archivo .txt donde se imprimir´an los resultados de las cuentas ingresadas, por ejemplo:

./tp0 -p 2 -o resultados.txt

13

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa

En el ejemplo, se imprimir´an los resultados en el archivo *resultados.txt*, borrando el contenido previo de dicho archivo.

Se destaca, que en caso de no recibir el comando -p, la precisi´on del procesamiento es la necesaria para realizar la operaci´on demandada, es decir, no se modificar´an los valores ingresados para ser procesados. En el caso en que un n´umero de entrada no pueda ser representado por la precisi´on indicada, se entender´a como precisi´on a la cantidad de cifras decimales significativas. Como ejemplo, si se ingresa el valor 12345 con una precisi´on de 4, la operaci´on tomar´a el valor 12350. Es decir, las primeras 4 cifras significativas del n´umero, redondeando la ´ultima cifra.

Por otra parte, si no se recibe el comando -o, se toma la salida est´andar por defecto (cout). An´alogamente con el comando -i (cin). Adem´as, se aclara que en caso de que el archivo ingresado como argumento de ´estas dos ´ultimas opciones no exista, se expresa un mensaje de error de apertura del archivo.

A continuaci´on, se observar´a la ´ultima opci´on disponible que se ingresa con el comando -h, el cual tiene como objetivo la impresi´on de un mensaje de ayuda que indica las opciones de ejecuci´on posibles.

./tp0 -h

cmdline -p precision [-i file] [-o file]

5. Verificaci´on con Valgrind y resultados de los casos de prueba

En primera instancia se ejecut´o el programa con Valgrind para verificar que no haya fallas de memoria en el c´odigo. El programa espera que se ingrese operaciones a realizar y corta cuando se ingresa una operaci´on inv´alida. Se obtuvo el siguiente mensaje:

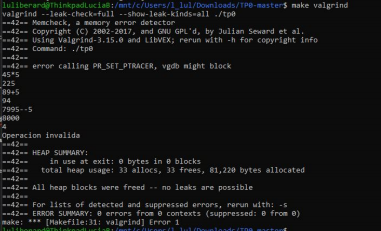
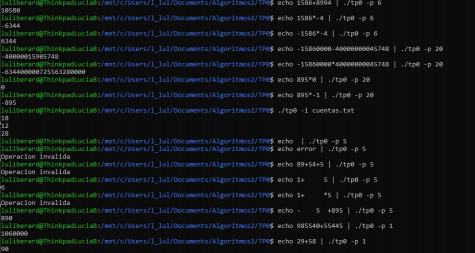
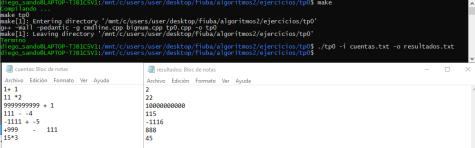


Figura 5.1: Verificaci´on con Valgrind

Lo cual verifica el correcto funcionamiento del manejo de memoria a lo largo del c´odigo. Luego se procedi´o a probar distintos casos para corroborar los resultados:

14

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa Figura 5.2: Casos de prueba

Figura 5.3: Casos de prueba con archivos .txt

6. Conclusiones

A lo largo del trabajo pr´actico se logr´o dise˜nar e implementar una clase bignum que representa n´umeros enteros de precisi´on fija pero arbitrariamente grande. Esto sirvi´o para comprender mejor el funcionamiento interno al hacer operaciones con distintos tipos de datos y la complejidad que eso conlleva. A diferencia del lenguaje de programaci´on C, donde para generar este tipo de operaciones se debe armar funciones especificas (por ejemplo ”sumar(a,b)”), en este trabajo se utiliz´o la ventaja de C++ de poder sobrecargar los operadores para que tambi´en puedan realizar operaciones con la clase bignum definida de una manera m´as intuitiva (a+b).

A la hora de dividir las tareas del trabajo, se tuvo que tener especial cuidado que todos los m´odulos y funciones dentro de la clase bignum sean compatibles con el desarrollo de los demas integrantes. Es por eso que se tuvo que definir de antemano c´omo se iba a encarar el proyecto. Se trato de dise˜nar funciones compactas y eficientes pero que a la vez mantuvieran un c´odigo legible, facilitando el trabajo en grupo. Se trabaj´o utilizando la plataforma Github para trabajar siempre bajo las ´ultimas modificaciones.

7. Anexo - C´odigos

Se adjunta los c´odigos utilizados en el siguiente link de Google Drive:

15

Algor´ıtmos y Programaci´on II 95.12 Primer Cuatrimestre 2021 UBA - Facultad de Ingenier´ıa https://drive.google.com/drive/folders/1rLj7F6r3\_dt0o9U1ecpbRClZs7u-khmJ?usp=sharing

16