PENERAPAN TEORI BILANGAN DALAM IMPLEMENTASI TIPE DATA *BIG INTEGER*

Anna Maria J S– NIM : 13503075

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13075@students.if.itb.ac.id

Abstrak

Aplikasi kriptografi modern dengan berbagai macam algoritma saat ini sangat banyak dikerjakan oleh komputer. Hal ini berarti diperlukan implementasi berbagai algoritma kriptografi yang melibatkan berbagai jenis bilangan, khususnya bilangan bulat. Pada beberapa algoritma kriptografi modern, digunakan bilangan-bilangan bulat yang sangat besar untuk menangani pembangkitan kunci-kuncinya. Kekuatan algoritma-algoritma kriptografi tersebut terletak pada berapa besar bilangan bulat yang digunakan sebagai kunci. Sebagai contoh, pada algoritma RSA (Rivers, Shamir, Adleman), sudah terbukti bahwa semakin besar bilangan bulat yang digunakan sebagai kunci publik dan kunci privat, maka semakin sulit untuk memecahkan chiperteksnya.

Namun, tipe data bilangan bulat (*integer*) yang disediakan oleh bahasa pemrograman biasanya sangat terbatas pada ukuran, sehingga pemrogram kerap kali merasa kesulitan dalam menghasilkan tipe data bilangan bulat yang panjangnya tidak dibatasi. Tipe data yang disediakan biasanya hanya sampat 32 byte, sedangkan, kerapkali dibutuhkan pengolahan bilangan yang panjangnya bisa mencapai 100 byte.

Oleh karena itu, dibutuhkan suatu kelas yang merupakan manipulasi dari kelas bilangan bulat biasa yang mampu menangani bilangan bulat hingga 32 byte. Kelas ini sering disebut dengan kelas BigInt, Big_Integer, Big_Int, atau istilah-istilah sejenisnya. Kelas-kelas ini sebenarnya sudah banyak tersedia di internet dalam berbagai bahasa dan versi. Pada makalah ini pun, digunakan kode yang dapat diperoleh dari internet. Namun, seringkali pengguna kelas-kelas ini tidak mengerti bagaimana algoritma dibalik penggunaan fungsi-fungsi yang disediakan oleh kelas BigInt. Oleh karena itu, pada makalah ini akan dikaji algoritma di balik fungsi-fungsi yang disediakan oleh kelas BigInt, yang sebenarnya merupakan penerapan matematika diskrit, khususnya teori bilangan

Kata kunci*:* bilangan bulat, *integer*, *big integer,* matematika diskrit.

1. Pendahuluan

Makalah ini berisi penerapan prinsip-prinsip matematika diskrit di dalam membuat kelas BigInt. Dasar teori yang digunakan sebagian besar merupakan teori bilangan yang sudah sangat umum digunakan.

Tujuan pembuatan makalah ini adalah

- Mengetahui bagaimana penerapan prinsip-prinsip teori bilangan dalam membuat kelas BigInt.
- Mengetahui bagaimana algoritma algoritma BigInt dengan menggunakan bahasa Java.

Pada makalah ini, fungsi-fungsi yang akan dibahas implementasinya hanyalah fungsi-fungsi aritmatika biasa, seperti penjumlahan, pengurangan, perkalian dan pembagian Selain karena keterbatasan tempat dan waktu (akan sangat panjang untuk membahas seluruh fungsi-fungsi aritmatika kelas BigInt), penulis beranggapan, pengetahuan yang paling diperlukan dan mendasar sebenarnya terletak pada bagaimana mengkonstruksi kelas BigInt dan membuat fungsi-fungsi aritmatikanya.

Pada makalah ini penjelasan mengenai penerapan teori matematika diskrit dijelaskan pada bab 2, yaitu Kelas Int, karena prinsip penerapan untuk kelas Bigint akan sama saja dengan terhadap kelas Int. Sedangkan pada kelas BigInt, akan dicantumkan prosedur-prosedur dan fungsi-fungsi kelas BigInt. Baik kelas Int maupun kelas BigInt sebenarnya sudah dapat di-*download* dalam berbagai bahasa pemrograman dari internet. Namun pada makalah ini,

yang digunakan hanya kelas BigInt yang diimplementasikan dengan bahasa Java.

2. Kelas Int

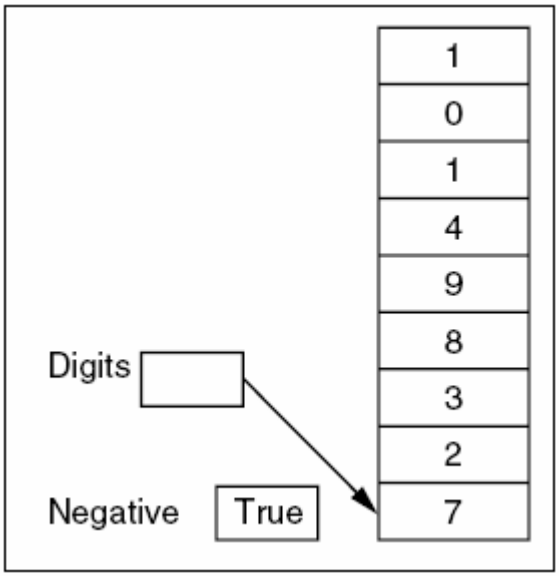
2.1 Deklarasi

Langkah awal di dalam mengembangkan kelas BigInteger adalah dengan mengembangkan kelas Int. Kelas ini dapat dibentuk dengan menggunakan metode-metode memanipulasi integer biasa yang bisa menangani bilangan yang ratusan dan ribuan. Integer seperti ini sangat umum digunakan di dalam kriptografi.

Kelas Int yang akan dikembangkan adalah kelas yang juga dapat merepresentasikan bilangan dengan panjang tertentu, dan juga dapat merepresentasikan beberapa konstruktor yang menerima parameter untuk menginisialisasikan field data. D
engan demikian integer dapat direpresentasikan sebagai array of integer, dimana setiap integer nya berupa 1 digit. Pemrogram juga dapat membuat sebuah data field untuk menandai apakan bilangan yang digunakan positif/negatif.

Kode yang dapat digunakan adalah sebagai berikut:

Sebagai contoh untuk menyimpan bilangan bulat sebesar -000723894101 dilakukan dengan representasi yang ditunjukkan pada gambar 1. Angka 0 yang ada di depan akan diabaikan,sehingga bilangan bulat -000723894101 hanya akan terdiri dari 1 buah angka 0.



Gambar 1 Representasi Lojik dari bilangan -000723894101

2.2 Konstruktor

Langkah selanjutnya dalam membuat kelas Int adalah dengan membuat konstruktor. Hal-hal yang diperlukan pada konstruktor adalah:

1. Konstruktor Int yang dapat mengkonversi tipe atomik int menjadi Int;
2. Konstruktor yang dapat mengkonversi string menjadi Int;
3. Konstruktor yang dapat mennyalin (*copy*) Int kepada new Int;
4. Konstruktor *default*, yang dapat mengabaikan nilai 0 yang ada di depan

Untuk membentuk konstruktor pertama, misal hendak dilakukan konversi -562 menjadi sebuah kelas Int, maka tidak perlu dilakukan alokasi sebesar 10 digit, karena bilangan integer terbesar tidak membutuhkan digit yang lebih besar dari 10 digit. Yang pertama sekali dilakukan adalah mengalokasikan tanda negatif dan menge-set nilainya menjadi true atau false. Kemudian dapat dilakukan pembagian terus menerus dengan angka 10, dan menyimpannya sisanya, ditempatkan pada array yang dimulai dari belakang. Jika terdapat slot yang belum terisi saat pembagian selesai dilakukan, maka semua

```
public class Int {
  //Records if Int negative/nonnegative
  boolean negative=false;
  //Digits are stored as decimal digits,
  //highest order digit first
  int[] digits;
  //Declare zero constant
  final public static Int ZERO=new
  Int();
  //Records position of 0 (zero) in the
  //character set
  //final private static int
  zeroPos='0';
```

Elemen dari belakang digeser ke depan. Algoritma untuk melakukan hal ini dijelaskan sebagai berikut:

```
public Int(int n) {
  //Produce the array-an int can not
  have //more than 10 decimal digits
  int[] temp=new int[10];

  //zero is a special case
  if (n==0) {
    negative=false;
    digits=new int[1];
    digits[0]=0;
    return;
  }

  //Negative int n-set negative to true,
  //take absolute value of n
  if (n<0) {
```

```
    negative=true;
    n=Math.abs(n);
  }
  int count=10;
  //Divide by 10 until nothing left
  while (n>0) {
    //Remainder is the count-th digit in
    //the array
    temp[—count]=n%10;
    n/=10;
  }

  //Remove any leading zeros-make new
  //array and copy
  digits=new int[temp.length-count];
  for (int i=0;i<digits.length;i++)
  digits[i]=temp[count+i];
}
```

Konstruktor yang menghasilkan hasil copy dari Int dari Int yang lain dan konstruktor Int() mudah untuk dibuat. Konstruktor Int() seharusnya menge-set Int mejadi 0.

```
//This one produces an array of one int
containing 0
public Int() {
  negative=false;
  digits=new int[1];
  digits[0]=0;
}

public Int(Int n) {
  negative=n.negative;
  digits=new int[n.digits.length];
  for (int i=0;i<digits.length;i++)
  digits[i]=n.digits[i];
}
```

Sekarang akan dilakukan pengembangan konstruktor yang menghasilkan objek Int dari *strings*. Hal ini dapat dilakukan dengan melakukan parse terhadap *strings* dan menetukan apakah *string* tersebut dapat diubah ke dalam Int, kemudian menempatkan karakter (yang telah dikonversi menjadi int) pada *array*. Pada kasus-kasus strung yang tidak bisa di-*parse* menjadi Int, dapat di-throw dengan IntException :

```
public class IntException extends
Exception {
  public IntException() {super();}
  public IntException(String s)
  {super(s);}
}

//This constructor converts a String to
//an Int. May throw an
//Exception if the String cannot be
//converted to an Int.
```

```
  //this gives the
  //corresponding int
  for (int i=0;i<c.length;i++)
  digits[i]=(int)c[i]-zeroPos;
}
```

Sebagai hasil output, harus dilakukan pengubahan objek Int ke string dan ditampulkan. Untuk melakukan hal ini digunakan fungsi toString() yang ada pada java.

```
//Returns the Int as a String, mainly
//for output purposes
public String toString() {
  //Use a StringBuffer for efficiency
  StringBuffer answer=new
  StringBuffer("");

  //Put a "-" symbol in front if
  //negative
  if (negative) answer.append("-");
    //Append each digit to the
    StringBuffer and return it as a
    String
    for (int i=0;i<digits.length;i++) {
    answer.append(new
    Integer(digits[i]).toString());
  }
  return new String(answer);
}
```

2.3 Fungsi-Fungsi Perbandingan

Untuk membandingkan objek Int, metode yang akan dibuat adalah metode yang bisa memeriksa apakah dua objek tersebut sama atau yang satu lebih kecil atau lebih besar dari yang lainnya. Sebagai contoh, untuk menentukan apakah x lebih kecil daripada y, digunakan cara sebagai berikut:

- Jika x dan y berbeda tanda, maka yang negatif akan lebih kecil, jika tidak lanjutkan ke langkah berikutnya
- Jika x dan y panjangnya berbeda dan keduanya negatif, maka bilangan yang panjangnya paling besar adalah bilangan terkecil. Jika tidak, maka bilangan dengan panjang paling kecil adalah bilangan terkecil dari keduanya. Jika kedua bilangan sama panjangnya, lanjutkan ke langkah ke-3
- Telusuri array of int sampai ditemukan digit yang berbeda. Jika x dan y negatif, array yang mengandung nilai paling kecil adalah yang terbesar. Jika tidak, maka array yang mengandung digit yang paling besar akan merepresentasikan integer terbesat. Jika tidak ditemukan digit yang berbeda, maka kedua integer tersebut sama besarnya.

Algoritma diatas dapat diubah kedalam fungsi lessThan(Int) sebagai berikut:

```
public boolean lessThan(Int other) {
  //Start by assuming this is less than
  //other
  boolean answer=false;

  //Both Ints are nonnegative here
  if (!negative&&!other.negative) {

  //If they are the same length, must
  //compare the digits
  if
  (digits.length==other.digits.length) {
  int i=0;
  while
  (i<this.digits.length&&digits[i]==othe
  r.digits[i]) i++;

  //Each digit of this was less than
  //each digit of other
  if (i<this.digits.length)
  if (digits[i]<other.digits[i])
  answer=true;

  //this has smaller length than other-
  //must be less than
  } else if
  (digits.length<other.digits.length)
  answer=true;
  //If both Ints negative, do the
  //reverse of the above comparisons
  } else if (negative&&other.negative) {
  if
  (digits.length==other.digits.length) {
  int i=0;
  while
  (i<this.digits.length&&digits[i]==othe
  r.digits[i]) i++;
  if (i<this.digits.length)
  if (digits[i]>other.digits[i])
  answer=true;
  } else if
  (other.digits.length<digits.length)
  answer=true;

  //If this is negative and other
  //nonnegative, must be less than
  } else if (negative&&!other.negative)
  answer=true;

  //Otherwise, this is nonnegative and
  //other negative
  //Return answer, which was initialized
  //to false
  return answer;
}
```

Fungsi untuk menentukan apakah dua buah objek Ints sama besar adalah sebagai berikut:

```
public boolean equals(Int other) {
  boolean answer=true;

  //Check if same sign
  if (negative!=other.negative)
  answer=false;

  //Check if different lengths
  else if
  (digits.length!=other.digits.length)
  answer=false;

  //If same length and sign, compare
  //each digit
  else for (int i=0;i<digits.length;i++)

  //Any nonmatching digit sets answer to
  //false
  if (digits[i]!=other.digits[i])
  answer=false;
  return answer;
}
```

2.4 Fungsi-Fungsi Aritmatik

Fungsi yang harus diimplementasikan selanjutnya adalah fungsi-fungsi aritmatika seperti penjumlahan, pengurangan, perkalian dan pembagian.

Pertama sekali diimplementasikan adalah fungsi untuk penjumlahan dua buah bilangan bulat a dan b. Langkah-langkahnya dijelaskan sebagai berikut:

- Jika salah satu bilangan adalah 0, maka bilangan yang lain adalah jawabannya.
- Jika a = -b, maka hasil penjumlahan adalah 0, jika tidak, lanjutkan ke langkah selanjutnya
- Jika kedua bilangan sama tandanya, jumlahkan, dengan aturan, bilangan terkecil yang dijumlahkan ke bilangan besar. Jika berbeda tanda, lanjutkan ke langkah selanjutnya.
- Jika kedua bilangan berbeda tanda, maka penjumlahan disini akan berubah menjadi fungsi pengurangan, yaitu ... Untuk melakukan pengurangan, pertama sekali harus ditentukan bilangan terbesar tanpa mempertimbangkan tanda kedua bilangan tersebut. Lalu kurangi bilangan terbesar dengan bilangan terkecil. Lalu, hasil tersebut diberikan tanda yang sama dengan bilangan terbesar.

Cara yang dipaparkan diatas adalah cara yang secara normal dilakukan manusia untuk menjumlahkan dua buah bilangan yang berbasis 10. Untuk menerapkan fungsi add(Int) dan subtract() diperlukan fungsi-fungsi kecil sebagai perantara, seperti equals(Int) dan lessThan(Int), Sebagai contoh, berikut adalah fungsi menegasikan sebuah objek kelas Int dan mengabsolkutkan sebuah nilai.

```
public Int absoluteValue() {
  //Make a new Int by copying this Int
  Int answer=new Int(this);
  //Set negative to false
  answer.negative=false;
  return answer;
}
```

```
public Int negative() {
  Int answer=new Int(this);
  //Flip the negative value
  answer.negative=!this.negative;
  return answer;
}
```

Fungsi yang paling banyak digunakan adalah fungsi add(Int). Pada langkah pertama, tentukan tanda kedua bilangan yang akan dijumlahkan, jika keduanya sama tanda, maka ini adalah kasus penjumlahan, sehingga fungsi addDigits(Int) dipanggil. Selanjutnya, jika kedua bilangan berbeda tanda, maka fungsi yang dipanggil ssadlaah SubtractDigits(Int). Pada fungsi subtractDigits, harus digunakan fungsi borrow(int), yang berfungsi untuk meminjamkan digit dari digit di sebelah kiri untuk pengurangan. Setelah penjumlahan maupun pengurangan selesai dilakukan, maka bisa saja terdapat leading zeros, yaitu nilai 0 di awal bilangan, yang seharusnya dibuang.

Contoh code yang digunakan adalah:

```
public Int add(Int other) {
  Int ans;
  //zero is a special case-nothing to do
  //but return a copy of the
  //nonzero Int
  if (this.equals(ZERO)) return new
    Int(other);
    else if (other.equals(ZERO)) return
    new Int(this);
    else if
    (this.equals(other.negative()))
    return new Int();

  //If they are the same sign, perform
  //the addition; add carries
    else if (negative==other.negative) {
    ans=addDigits(other);
    ans.negative=negative;
    }

  //If they are of different signs,
  //determine the larger
  //(magnitude-wise) and subtract the
  //smaller from it.
  //Result has same sign as first
  //(larger) operand.
    else if
    (this.absoluteValue().lessThan(other.
    absoluteValue())) {
```

```
      ans=other.subtractDigits(this);
      ans.negative=other.negative;
    } else {
      ans=this.subtractDigits(other);
      ans.negative=this.negative;
    }
  //Trim leading zeros and return
  return ans.trimZeros();
}
```

```
public Int subtract(Int other) {
  //To subtract, we add the negative
  return this.add(other.negative());
}
```

```
private Int addDigits(Int other) {
  int top1=this.digits.length-1;
  int top2=other.digits.length-1;
  int
  top3=Math.max(this.digits.length,other
  .digits.length)+1;
  Int answer=new Int();
  answer.digits=new int[top3];
  top3--;
  int carry=0; int sum=0;
  while (top1>=0&&top2>=0) {
    sum=this.digits[top1]+other.digits[to
    p2]+carry;
    if (sum>9) {sum%=10; carry=1;} else
    carry=0;
    answer.digits[top3]=sum;
    top1--;top2--;top3--;
  }
  if (top1<0&&top2<0) {
    answer.digits[0]=carry;
  } else if (top1<0) {

  while (top2>=0) {
    sum=other.digits[top2]+carry;
    if (sum>9) {sum%=10; carry=1;} else
    carry=0;
    answer.digits[top3]=sum;
    top2--;top3--;
  }
  answer.digits[top3]=carry;
  } else {
    while (top1>=0) {
    sum=this.digits[top1]+carry;
    if (sum>9) {sum%=10; carry=1;} else
    carry=0;
    answer.digits[top3]=sum;
    top1--;top3--;
    }
    answer.digits[top3]=carry;
  }
  return answer;
}
```

```
private Int subtractDigits(Int other) {
  Int answer=new Int();
  Int copy=new Int(this);
```

```
//Method to “borrow” for subtraction
private void borrow(Int n,int pos) {
  while (n.digits[pos]==0) {
  n.digits[pos]=9;
  pos--;
  }
  n.digits[pos]—;
}
```

```
//Method to chop off any leading zeros
private Int trimZeros() {
  int i;
  //Look for first nonzero in the array
  for (i=0;i<this.digits.length;i++)
  if (this.digits[i]!=0)
  break;
  Int answer=new Int();
  answer.negative=this.negative;

  //Make a (possibly) smaller array for
  //answer
  answer.digits=new
  int[this.digits.length-i];

  //Copy the nonzero digits over, and
  //return answer
  for (int
  j=0;j<answer.digits.length;j++)
  answer.digits[j]=this.digits[j+i];
  return answer;
}
```

Fungsi selanjutnya yang akan diimplementasikan adalah perkalian dan pembagian. Untuk fungsi perkalian dua buah bilangan bular, yang harus dilakukan hanya mengalikan salah satu bilangan bulat dengan salah satu digit dari bilangan lainnya setiap kali perkalian. Sebagai contoh untuk perkalian bilangan bulat berikut:

$$\begin{array}{r} 527 \\ \times\ 613 \\ \hline \end{array}$$

maka, perkalian tersebut dapat dipisahkan menjadi berikut:

527 x 6 = 3162
527 x 1 = 527
527 x 3 = 1581

Selanjutny,a hasil sub-perkalian tersebut dijumlahkan dengan menggeser sebagian hasil sub-perkalian ke kiri. Sebagai contoh, dapat dilakukan penambahan angka 0 pada hasil sub-perkalian 527 x 1 karena 1 terletak pada kolom puluhan pada bilangan 613, sedagkan untuk hasil sub-perkalian 526 x 6, tambahkan dua buah bilangan 0 di depan hasil sub-perkalian, karena digit 6 terletak pada kolom ratusan pada bilangan 613.
Bila dituliskan, hal ini menjadi:

```
   527
x  613
  1581
+ 5270
+316200
 323051
```

Metode seperti ini menjadikan permasalahan perkalian menjadi permasalah penjumlahan, asalkan sudah dilakukan perkalian bilangan pertama dengan 1 per satu digit bilangan kedua, dan telah dilakukan penambahan bilangan 0 di awal hasil-hasil sub-perkalian sesuai dengan posisi digit pada bilangan kedua. Untuk melakukan penambahan (*append*) bilangan nol dapat dilakukan dengan cara berikut:

```
private Int appendZeros(int places) {
  //Make a new Int object
  Int result=new Int();

  //If this equals 0, return 0; no need
  to append
  if (this.equals(ZERO)) return result;

  //Make the resulting array larger
  result.digits=new
  int[this.digits.length+places];

  //Shift the digits into the new array
    for (int
    i=0;i<this.digits.length;i++) {
    result.digits[i]=this.digits[i];
  }
  return result;
}
```

Selanjutnya adalah fungsi yang digunakan untuk mengalikan bilangan Int dengan 1 buah digit. Cara yang dilakukan adalah dengan mengalikan satu per satu digit dari bilangan pertama dengan 1 buah digit yang sudah dipilih tadi. Jika hasil perkalian lebih besar dari 9, maka bagi hasil perkalian dengan 10. Sisa pembagian, ditambahkan dengan carry yang mungkin muncul dari penjumlahan sebelumnya akan menjadi salah satu digit hasil. Hasil pembagian dengan 10 digunakan sebagai carry untuk perkalian selanjutnya.

Contoh perkalian yang dilakuakan adalah:

```
  527
x   3
  ???
```

Pertama sekali, kalikan 7 x 3 = 21; 1 menjadi salah satu digit di kolom hasil, tepatnya di sebelah kanan, dan 2 menjadi carry.

```
    2
  527
x   3
  ??1
```

Langkah kedua, lakukan perkalian 2 x 3= 6, tambahkan dengan carry sebelumnya, yaitu 2, maka hasil yang diperoleh 8;

```
    0
  527
x   3
  ?81
```

Langkah selanjunya, lakukan perkalian 5 x 3 dan tambahkan carry sebelumnya, yaitu 0. maka angka 5 menjadi digit ratusan di kolom hasil dan bilangan 1 menjadi *carry.*

```
    1
  527
x   3
  581
```

Pada kasus ini, sudah tidak ada lagi digit yang akan dikalikan. Namun, masih ada 1 buah carry yang tersisa pada kasus ini. *Carry* ini akan menjadi

bilangan ribuan pada hasil. Hasil yang diperoleh adalah sebagai berikut:

```
   527
x    3
------
  1581
```

Cara yang digunakan pada langkah-langkah yang telah dijelaskan sangat mudah jika dijelaskan secara matematis. Namun, pada kenyataanya, menuliskannya menjadi kode, tidak semudah proses matematisnya. Kode yang digunakan adalah sebagai berikut:

```
//Method to multiply an Int by a
single //decimal digit
//Called repeatedly by multiply(Int)
//method

private Int multiply(int otherDigit) {
  //Make a new Int for the answer
  Int result=new Int();

  //If digit to multiply by is 0,
  return //0
  if (otherDigit==0) return result;

  //Make the answer array one longer
  //than the first operand,
  //in case there is a carry

  result.digits=new
  int[this.digits.length+1];
  int carry=0;
  int tempInteger;
  int i;
  for (i=this.digits.length-1;i>=0;i--
  ) {

  //i+1th digit of result is the ith
  /digit of the first operand
  //times the digit in the second
  //operand. If this is more than
  //10, we must keep only the least
  //significant digit.
  //We also add any previous carries

  tempInteger=this.digits[i]*otherDigi
  t+carry;
  result.digits[i+1]=tempInteger%10;
  //If the product is more than 10, we
  //must set carry
  //for the next round
  carry=tempInteger/10;
  }

  //Possibility of one last carry; do
  //the final digit.
  result.digits[0]=carry;
  return result;
}
```

Bila, telah ada fungsi penambahan bilangan 0 dan perkalian sebuah bilangan Int dengan sebuah digit, maka pada proses perkalian dua buah kelas Int, yang harus dilakukan adalah menghitung dan menjumlahkan hasil sub-sub perkalian. Hal ini merupakan metode yang lebih mudah dilakukan dibandingkan kedua fungsi sebelumnya. Fungsi yang dilakukan adalah:

```
public Int multiply(Int other) {
  //Initialize the answer to 0
  Int result=new Int();
  //If either operand is 0, return 0
  // (this.equals(ZERO)||other.equals(ZERO))
  return result;
  //Now, multiply the first operand by //ch
  digit in the
  //second operand, shifting left each //swer
  by a power of ten
  //as we pass through the digits, //ding
  each time to result.
  for (int i=0; i<other.digits.length; i++) {
    result=result.add(this.multiply(other.digi
    ts[i])
    .appendZeros(other.digits.length-1-i));
  }

  //If operands are same sign, result is
  //sitive
  //otherwise, the result is negative; 0 //
  already taken care of
    if (this.negative==other.negative)
    result.negative=false;
    else result.negative=true;

  //Return the result
  return result.trimZeros();
}
```

Harus diingat bahwa pada akhir fungsi multipl(Int), harus dilakuan penghilangan bilangan 0 yang ada di depan bilangan hasil Hal lain yang harus diperhatikan adalah tanda bilangan kedua bilangan yang dikalikan (operand). Jika kedua bilangan sama tandanya, maka bilangan hasil pasti memiliki tanda positif, demikian sebaliknya. Bilangan 0 adalah kasus khusus, dimana setiap bilangan yang dikalikan dengan 0 akan memberikan hasil 0.

Pembagian adalah proses yang paling mahal untuk dilakukan komputer, atau kompleksitasnya paling besar. Saat ini, akan dibahas kasus pembagian bilangan bulat positif dimana pembagi dan bilangan yang akan dibagi bernilai positif.

Salah satu masalah yang dijumpai adalah bagaimana memperkirakan besar digit hadil pembagian. Sebagai contoh untuk perkalian bilangan berikut

6772190 ÷ 37658

Bila dilakukan pembagian bilangan 6 (digit pertama dari bilangan yang akan dibagi) dengan angka 3 (digit pertama dari pembagi), maka akan diperoleh bilangan 2. Sehingga estimasi yang diperoleh adalah 200. Padahal hasil pembagian ini adalah 179. Perkiraan yang diperoleh jelas terlalu besar terhadap hasil.

Persoalan serupa dijumpai pada kasus berikut:

19276 ÷ 273.

Dari percobaan ini, dapat disimpulkan bahwa untuk memperoleh estimasi hasil, tidak mungkin digunakan digit pertama dari bilangan yang akan dibagi dan digit pertama dari pembagi. Oleh karena itu, cara yang digunakan adalah dengan mencoba dua digit dari bilangan yang akan dibagi, dan membaginya dengan pembagi.

Salah satu metode yang digunakan agar tidak menghabiskan waktu memperkitakan hasil pembagian adalah dengan memungkinkan hasil pembagian bernilai positif maupun negatif. Misalnya pada pembagian berikut:

19276 ÷ 273.

Posisi dari setiap digit adalah sebagai berikut:

19276	**273**
54321	*321*

Langkah pertama, cobalah untuk membagi 1 (digit pertama dari 19276) dengan angka 2 (digit pertama dari 273). Jika gagal, maka ambil bilangan 19 dan bagikan dengan 2, sehingga diperoleh hasil 8. Hasil pembagi pertama yang diperoleh adalah :

90

Angka 0 ditambahkan 1 buah, karena posisi angka 9 pada 19276 adalah 4, sedangkan posisi angka 2 pada bilangan 273 adalah 3. Sehingga banyaknya angka 0 yang harus ditambahkan adalah 4 – 3 = 1 buah. Selanjutnya, lakukan pengurangan 90 x 273 terhadap angka 19276. Proses ini tuliskan sebagai berikut:

```
 19276
-24570
 -5294
```

Langkah kedua, lakukan pembagian angka -5 pada bilangan -5294 dengan angka 2 pada bilangan -273. Hasil yang diperoleh adalah -2. Karena -5 pada bilangan -5294 terletak pada posisi ke-4 dan angka 2 terletak pada posisi ke-3, maka lakukan penambahan bilangan 0 di akhir bilangan 2, sehingga hasil yang diperoleh adalah:

-20

Selanjutnya hasil pembagian yang diperoleh dijumlahkan menjadi persamaan sebagai berikut:

90 + -20 = 70.

Langkah berikut, lakukan pengurangan -20 x 273 dari -5294. Hasil yang diperoleh adalah:

```
 -5294
--5460
  166
```

Hasil sisa pembagian yang diperoleh lebih kecil daripada 273, sehingga diperoleh

Hasil pembagian *(quotient)* = 70
Sisa hasil pembagian *(remainder)* = 166

Perbedaan yang dilakukan pada proses ini dengan proses pembagian biasa yang dilakukan adalah pada pembagian ini, diijinkan hasil pembagian bernilai negatif. Contoh tadi dapat dituliskan sebagai berikut:

```
    70
   -20
    90
273)19276
  -24570
   -5294
  --5460
   166
```

Contoh lain yang mirip adalah sebagai berikut:

```
          80297
         -3
           300
        -10000
         90000
123)9876543
    -11070000
     -1193457
    --1230000
         36543
        -36900
          -357
         --369
            12
```

Proses ini memungkinkan adanya sisa hasil pembagian yang bernilai negatif, seperti contoh berikut:

```
        10
982)9278
   -9820
    -542
```

Pada contoh diatas nilai absolut dari 542 tetap lebih kecil dari 982. Bila hal ini terjadi, maka langkah yang harus dilakukan adalah menambahkan pembahgi dengan hasil pembagian berniali negatif tadi dan mengurani 1 dari hasil pembagian. Pada kasus ini, solusi menjadi:

Hasil pembagian *(quotient)* = 9
Sisa hasil pembagian *(remainder)* = 440

Sekarang analisis dilakukan bila salah satu dari bilangan yang akan dibagi atau bilangan pembahi bernilai negatif. Perhatikan jika x merupakan bilangan yang akan dibagi dan bernilai positif, y adalah bilangan pembagi positif, q adalah sisa pembagian dan r adalah sisa hasil pembagian, maka hubungan keempat variabel ini dapat dinyatakn sebagai berikut:

$x = yq + r\ b > r \geq 0.$

Jika salah satu dari x atau y bernilai negatif, maka kesamaan tadi dapat dijaga dengan mengganti beberapa tanda. Sebagai contoh, bila y bernilai negatid, maka tanda bilangan q harus diganti karena

x = yq + r iff
x = _y(_q) + r,

Jika x dan y bernilai negatif, maka ubah tanda r, karena

x = yq + r iff
_x = _(_y(_q) + r)
_x = _yq _ r.

Dengan prinsip-prinsip pembagian yang telah dijelaskan diatas, seharusnya pembagian pada kelas BigInt menjadi mudah dilakukan. Namun, harus diingat untuk tetap melakukan percobaan-percobaan untuk menghasilkan sebuah program yang bebas bugs.

3. Kelas BigInt

Pada sub-bab ini tidak akan dijelaskan lagi mengenai penerapan matematika diskrit dalam membuat kelas BigInt., karena algoritmantya sama saja dengan kelas Int di atas. Sub-bab ini hanya akan berisikan prosedur-prosedur dan fungsi-fungsi yang ada pada kelas BigInt.

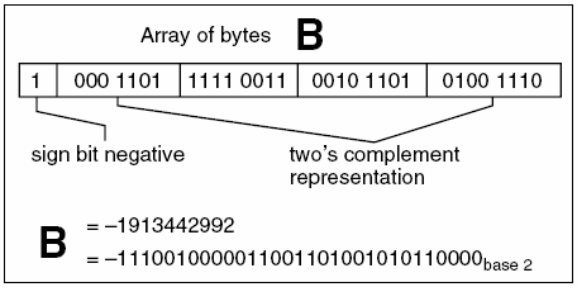
3.1 Konstruktor

Beberapa konstruktor yang digunakan adalah

```
public BigInteger(byte B[]) throws
NumberFormatException
```

Konstruktor ini menerima sebuah *array of byte* yang bersifat biner dan mempunyai *most significant bit,* yaitu bit yang menunjukkan tanda positif atau negatif pada elemen pertama dari byte pertama.

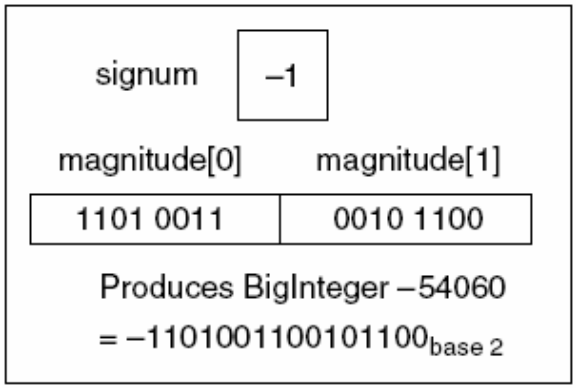
Representasi lojik dapat digambarkan sebagai berikut:



Gambar 2 Representasi Lojik dari Konstruktor pertama

```
public BigInteger(int signum, byte
magnitude[]) throws
NumberFormatException
```

Konstruktor di atas menerima array of byte dan sebuah integer di awal untuk menandakan tanda bilangan ini
Representasi lojik dapat digambarkan sebagai berikut:



Gambar 3 Representasi Lojik dari Konstruktor kedua.

Integer yang digunakan sebagai signum adalah:
-1 menandakan negatif,
0 menandakan 0
1 menandakan positif

```
public BigInteger(String val) throws
NumberFormatException
```

Konstruktor ini mengubah string menjadi BigInt. Contoh pemanggilan fungsi ini adalah

```
BigInteger m = new
BigInteger("923875698326534298745692868
98623498");
```

```
public BigInteger(String val, int
radix) throws NumberFormatException
```

Konstruktor ini mengubah string yang mungkin berisi nilai minus dan basis (radix) yang digunakan. Basis yang diijinkan berkisar 2-36. Contoh pemanggilan konstruktor ini adalah:

```
BigInteger m = new
BigInteger("101111100001010111010000001
111101010011111101",2);
```

```
public BigInteger(int bitLength, int
certainty, Random rnd)
```

Konstruktor ini memberikan BigInteger yang random dengan panjang yang spesifik dan mungkin bernilai prima. Konstruktor ini mungkin paling banyak digunakan dalam kriptografi, karena ia mampu men-generate BigInteger yang mungkin pula bernilai negatif. Contoh pemanggilan fungsi ini adalah:

```
SecureRandom sr=new SecureRandom();
BigInteger p=new BigInteger(1024,3,sr);
```

3.2 Fungsi-fungsi dan Prosedur-prosedur

Sebenarnya ada banyak sekali fungsi-fungsi maupun prosedur yang bisa dikembangkan pada kelas BigInt, namun pada makalah ini hanya akan dicantumkan fungsi-fungsi atau prosedur-prosedur yang umum digunakan, yaitu:

- `public BigInteger add(BigInteger val) throws ArithmeticException`

Fungsi ini mengembalikan nilai (this + val)

- `public BigInteger subtract(BigInteger val)`

Fungsi ini mengembalikan nilai (this - val)

- `public BigInteger multiply(BigInteger val)`

Fungsi ini mengembalikan nilai (this.val)

- `public BigInteger divide(BigInteger val) throws ArithmeticException`

Fungsi ini mengembalikan nilai (this/val) dan akan melakukan throw, jika val = 0.

- `public BigInteger remainder(BigInteger val) throws ArithmeticException`

Fungsi ini mengembalikan sisa hasil pembagian (this/val) dan akan melakukan throw, jika val = 0.

- `public BigInteger[] divideAndRemainder(BigInteger val) throws ArithmeticException`

Fungsi ini menggabungkan fungsi divide() dan remainder() karena kedua fungsi ini serng digunakan secara bersama-sama. Hasil fungsi ini adalah array of BigInt yang elemennya berjumlah 2.

- `public BigInteger pow(int exponent) throws ArithmeticException`

Fungsi ini melakukan perpangkatan this dengan exponent dan akan melakukan throw jika exponent bernilai lebih kecil dari 0.

- `public BigInteger gcd(BigInteger v)`

Fungsi ini akan memberikan hasil gcd (great common divisor) dari BigInt this dan v.

- `public BigInteger abs()`

Fungsi ini akan menghasilkan nilai absolut dari BigInt this

- `public BigInteger negate()`

Fungsi ini akan menhasilkan nilai -this

- `public int signum()`

Fungsi ini akan mengembalikan nilai signum dari objek BigInt

- `public BigInteger mod(BigInteger m)`

Fungsi ini akan mengembalikan nilai this mod m. Dan akan melakukan throw jika m <= 0.

- `public BigInteger modPow(BigInteger e, BigInteger m)`

Fungsi ini akan mengembalikan nilai (this) mod m.

DAFTAR REFERENSI

[BIS03] Bishop, Davis, *Introduction to Cryptography with Java Applets*, Jons and Bartlett Computer Science, 2003

http://www.cs.utk.edu/ tanggal akses 20 Desembar 2006, pukul 15.00 WIB