JOBSHEET 14
Heap

14.1 Tujuan Praktikum

Setelah melakukan praktikum ini, mahasiswa mampu:

1. memahami struktur data heap;
2. membuat dan mendeklarasikan struktur algoritma heap;
3. menerapkan algoritma dasar heap dalam beberapa studi kasus.

14.2 Praktikum

Implementasi Max Heap menggunakan Array

14.2.1 Tahapan Percobaan

Waktu percobaan (60 menit)

Pada percobaan ini akan diimplementasikan Max Heap menggunakan Array. Silakan lakukan langkah-langkah praktikum sebagai berikut.

1. Buatlah class **Node** dan konsrtruktornya

```
class Node
{
private int iData;                  // data item (key)

public Node(int key)                // constructor
  { iData = key; }
public int getKey()
  { return iData; }
public void setKey(int id)
  { iData = id; }
}                                   // end class Node
```

2. Tambahkan class **Heap** yang akan menyimpan method-method pada heap dan juga method main().

```
class Heap
{
  private Node[] heapArray;
  private int maxSize;              // size of array
  private int currentSize;          // number of nodes in array
```

3. Tambahkan konstruktror class Heap

```
public Heap(int mx) // constructor
{
   maxSize = mx;
   currentSize = 0;
   heapArray = new Node[maxSize];      // create array
}
```

4. Tambahkan method pengecekan apakah Heap masih dalam kondisi kosong

```
public boolean isEmpty()
{
   return currentSize==0;
}
```

5. Tambahkan method **insert** untuk menambah node

```
public boolean insert(int key) {
  if(currentSize==maxSize)
     return false;
  Node newNode = new Node(key);
  heapArray[currentSize] = newNode;
  trickleUp(currentSize++);
  return true;
}                                          // end insert()
```

6. Tambahkan method **trickeUp** untuk melakukan swap ke atas node child ke node parent jika node child lebih besar dari node parent

```
public void trickleUp(int index) {
   int parent = (index-1) / 2;
   Node bottom = heapArray[index];
   while( index > 0 &&
         heapArray[parent].getKey() < bottom.getKey() )
   {
```

```
        heapArray[index] = heapArray[parent]; // move it down
        index = parent;
        parent = (parent-1) / 2;
      }                                   // end while
heapArray[index] = bottom;
}                                         // end trickleUp()
```

7. Tambahkan method **trickleDown** untuk melakukan swap ke bawah jika node parent lebih kecil dari node child. Method ini berfungsi mengganti nilai parent dengan nilai node child yang lebih besar dari node parent. Perhatikan fungsi untuk membandingkan leftChild dan rightChild untuk mengecek mana node child yang lebih besar.

```
public void trickleDown(int index)
{
   int largerChild;
   Node top = heapArray[index];          // save root
   while(index < currentSize/2)          // while node has at least one child,
   {
   int leftChild = 2*index+1;
   int rightChild = leftChild+1;
                                    //find larger child
   if(rightChild < currentSize &&        //rightChild exists?
        heapArray[leftChild].getKey() <
        heapArray[rightChild].getKey())
      largerChild = rightChild;
else
   largerChild = leftChild;
                                         // top >= largerChild?
  if( top.getKey() >= heapArray[largerChild].getKey() )
     break;
                                         // shift child up
  heapArray[index] = heapArray[largerChild];
   index = largerChild;                  // go down
   }                                     // end while
   heapArray[index] = top; // root to index
```

```
    }                                   // end trickleDown()
```

8. Tambahkan method **remove**.

```
public Node remove()                    // delete item with max key
{                                   // (assumes non-empty list)
   Node root = heapArray[0];
   heapArray[0] = heapArray[--currentSize];
   trickleDown(0);
   return root;
}                                       // end remove()
```

9. Tambahkan method **change** untuk mengganti nilai node pada index tertentu.

```
public boolean change(int index, int newValue)
{
   if(index<0 || index>=currentSize)
   return false;
   int oldValue = heapArray[index].getKey(); // remember old
   heapArray[index].setKey(newValue); // change to new
   if(oldValue < newValue)              // if raised,
      trickleUp(index);                 // trickle it up
   else                             // if lowered,
      trickleDown(index);               // trickle it down
   return true;
}
```

10. Tambahkan method **displayHeap()** untuk mencetak struktur Heap terbaru

```
public void displayHeap() {
   System.out.print("heapArray: ");      // array format
   for(int m=0; m<currentSize; m++)
   if(heapArray[m] != null)
   System.out.print( heapArray[m].getKey() + " ");// end change()
else
   System.out.print( "--");
   System.out.println();
```

```
    int nBlanks = 32;
    int itemsPerRow = 1;
    int column = 0;
    int j = 0;
    String dots = "................................" ;
    System.out.println(dots+dots);
    while(currentSize > 0)
    {
       if(column == 0)
          for(int k=0; k<nBlanks; k++)
             System.out.print(' ');
       System.out.print(heapArray[j].getKey());

       if(++j == currentSize)
          break;
       if(++column==itemsPerRow)
       {
          nBlanks /= 2;
          itemsPerRow *= 2;
          column = 0;
          System.out.println(); }
else
    for(int k=0; k<nBlanks*2-2; k++)
    System.out.print(' ');                    // interim blanks
}                                             // end for
    System.out.println("\n" +dots+dots); //dotted bottom line
}                                             // end displayHeap()
```

11. Buatlah class Main untuk menjalankan program

```
public class HeapMain {
  public static void main(String[] args) throws IOException
  {
    int value, value2;
    Heap theHeap = new Heap(100);          // make a Heap; max size 100
    boolean success;
    theHeap.insert(70);
    theHeap.insert(40);
    theHeap.insert(50);
    theHeap.insert(20);
    theHeap.insert(60);
    theHeap.insert(100);
    theHeap.insert(80);
    theHeap.insert(30);
    theHeap.insert(10);
    theHeap.insert(90);

    while(true)
    {
      System.out.print("Enter first letter of ");
      System.out.print("show, insert, remove, change: ");
      int choice = getChar();
      switch(choice)
      {
        case 's':                          // show heap structure
        theHeap.displayHeap();
        break;
        case 'i':                          // insert node
          System.out.print("Enter value to insert: ");
          value = getInt();
          success = theHeap.insert(value);
```

```
            if( !success )
               System.out.println("Can't insert; heap full");
            break;
         case 'r':                        // remove
         if( !theHeap.isEmpty() )
            theHeap.remove(); else
            System.out.println("Can't remove; heap empty");
         break;
         case 'c':                        // change
         System.out.print("Enter current index of item: ");
         value = getInt();
         System.out.print("Enter new key: ");
         value2 = getInt();
         success = theHeap.change(value, value2);
         if( !success )
         System.out.println("Invalid index");
         break;
         default:
         System.out.println("Invalid entry\n");
         }                                // end switch
      }                                   // end while
   }                                      //end main
```

12. Tambahkan getString untuk membaca teks yang di-*input*-kan

```
//--------------------------------------------------------------
public static String getString() throws IOException {
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
String s = br.readLine();
return s;
   }
```

13. Tambahkan getChar untuk mengambil/ membaca karakter pertama dari teks yang di-*input*-kan

```
public static char getChar() throws IOException {
String s = getString(); return s.charAt(0);
}
//--------------------------------------------------------------
```

14. Tambahkan getInt untuk mengambil/ membaca teks yang di-*input*-kan dan mengubahnya menjadi bilangan bulat.

```
public static int getInt() throws IOException
{
String s = getString();
return Integer.parseInt(s);
}
}
```

15. Amati hasil running tersebut.
16. Jika terjadi error, import package yang dibutuhkan untuk menangani error.

14.2.2 Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

Hasil running

```
run:
Enter first letter of show, insert, remove, change: s
heapArray: 100 90 80 30 60 50 70 20 10 40
.....................................................................
                              100
                90                              80
        30              60              50              70
    20      10      40
.....................................................................
Enter first letter of show, insert, remove, change:
```

14.2.3 Pertanyaan Percobaan (30 menit)

1. Apa perbedaan method trickleUp dan trickleDown?
2. Jika dijalankan fungsi menambah nilai node, apa yang terjadi jika nilai node baru lebih besar dibandingkan nilai node parent?
3. Jika dijalankan fungsi menambah node, apa yang terjadi jika nilai node lebih besar dibandingkan nilai node terakhir?
4. Jika dijalankan fungsi menambah node, apa yang terjadi jika nilai node baru lebih kecil dibandingkan nilai node terakhir?
5. Bagaimana proses penghapusan node pada max heap? Tunjukkan baris kodenya
6. Jika dijalankan fungsi **change,** apa yang terjadi dengan node parent dan node child dari node tersebut ?

14.3. Tugas Praktikum (120 menit)

1. Sesuaikan kode program untuk implementasi min heap.
2. Tambahkan pilihan pada program untuk menjalankan max heap atau min heap.

--- *** ---