UNC-CH COMP 410

Skip List Implementation

You will write code that implements and exercises a **skip list**. The implementation will be similar to the approach we discussed in class, where a node in the list is an object with a data field and then an array of links (node object pointers) to connect it other nodes in the various levels.

To be a bit more space efficient, we will make the array of links on each cell not contain the max (like, say, 20) but rather be allocated to some number between 1 and the max by calling the constructor with an integer parameter. When we add a new cell (on insert) we will roll the dice and be told that the new node will have a specific level... say, a level 4 node. We then call the node constructor and pass 4 as a parameter; the constructor will allocate inside the node an array of links that has 4 elements. See the interfaces following for more details.

What to Hand-in

Hand in your code as per the instructions under Assignment 4 in Sakai. There will be zipping and naming instructions there.

Interfaces

For uniformity, and to aid timely grading, you must write your program with these specific structures.

Interface SkipList_Interface

```
/**
 * COMP 410
 * Make your class and its methods public!
 * Your skiplist implementation will implement this interface.
 *
*/

package SKPLIST_A4;

/*
  Interface: The skiplist will provide this collection of operations:

  insert:
    in: a douoble (the element to be stored into the skiplist)
    return: boolean, return true if insert is successful, false otherwise
    effect: if the double is already in the skiplist, then there is no change to
            the skiplist state, and return false
```

```
            if the double is not already in the skiplist, then a new skiplist node
              is created, the double put into it as data, the new node is linked
              into the skiplist at the proper place; size is incremented by 1,
              and return a true
    remove:
      in: a double (the element to be taken out of the skiplist)
      return: boolean, return true if the remove is successful, false otherwise
              this means return false if the skiplist size is 0
      effect: if the element being looked for is in the skiplist, unlink the node containing
              it and return true (success); size decreases by one
              if the element being looked for is not in the skiplist, return false and
              make no change to the skiplist state
    contains:
      in: a double (the element to be seaarched for)
      return: boolean, return true if the double being looked for is in the skiplist;
              return false otherwise
              this means return false if skiplist size is 0
      effect: no change to the state of the skiplist

    findMin:
      in: none
      return: double, the element value from the skiplist that is smallest
      effect: no skiplist state change
      error: is skiplist size is 0, return Double.NaN

    findMax:
      in: none
      return: double, the element value from the skiplist that is largest
      effect: no skiplist state change
      error: is skiplist size is 0, return Double.NaN

    size:
      in: nothing
      return: number of elements stored in the skiplist
      effect: no change to skiplist state

    empty:
      in: nothing
      return: boolean, true if the skiplist has no elements in it, true if size is 0
              return false otherwise
      effect: no change to skiplist state

    clear:
      in: none
      return: void
      effect: all elements in the skip list are unhooked so that no elements are in the list
              size is set to 0
              sentinel node remains
              the effect is to create a skip list state that exists when it is first
              produced by the constructor

    level:
      in: none
      return: integer, the level of the highest node in the skiplist
      effect: no change in skiplist state
      error: return -1 if skiplist is empty (size is 0, only sentinel node is there)

    max:
      in: none
      return: integer, the value of MAXHEIGHT that is set in the list constructor
      effect: no change in skip list state

    getRoot:
      in: none
      return: a skiplist node, the one that is the starter of the entire skiplist
              the skiplist starts with a sentinel, made in the list constructor.
              so getRoot returns the sentinel always, even if the skiplist is empty.
              If the list is empty, then level 0 of the senetinel link array would be null.
      effect: no change to skiplist state

  */
```

```
// ADT operations

public interface SkipList_Interface {
  public void setSeed(long seed);
  public void setProbability(double probability);
  public SkipList_Node getRoot();
  public boolean insert(double value);
  public boolean remove(double value);
  public boolean contains(double value);
  public double findMin();
  public double findMax();
  public boolean empty();
  public void clear();
  public int size();
  public int level();
  public int max();
}
```

Class SkipList_Node

```
package SKPLIST_A4;

public class SkipList_Node {
  private double value;
  private int level;
  private SkipList_Node[] next;

  public SkipList_Node(double value, int height) {
    next = new SkipList_Node[height];
    this.value = value;
    this.level = height;
  }

  public void setNext(int height, SkipList_Node node) {
    next[height] = node;
  }
  public double getValue() { return value; }
  public SkipList_Node[] getNext() { return next; }
  public SkipList_Node getNext(int level) { return next[level]; }
  public String toString() { return "" + value; }

  // ---------------------------------------------------------------------
  // you may add any other methods you want to get the job done
  // ---------------------------------------------------------------------
}
```

Class SkipList

```
package SKPLIST_A4;

import java.util.Arrays;
import java.util.Random;

public class SkipList implements SkipList_Interface {
  private SkipList_Node root;
  private final Random rand;
  private double probability;
  private const int MAXHEIGHT = 30; // the most links that a data cell may contain

  public SkipList(int maxHeight) {
    root = new SkipList_Node(Double.NaN, maxHeight);
    rand = new Random();
    probability = 0.5;
  }

  @Override
  public void setSeed(long seed) { rand.setSeed(seed); }
```

```
  @Override
  public void setProbability(double probability) {
     this.probability = probability;
  }

  private boolean flip() {
    // use this where you "roll the dice"
    // call it repeatedly until you determine the level
    // for a new node
    return rand.nextDouble() < probability;
  }

  @Override
  public SkipList_Node getRoot() { return root; }

  @Override
  public String toString() {
    StringBuilder sb = new StringBuilder();

    int levels;
    for(levels = 0; levels < root.getNext().length && root.getNext(levels) != null; levels ++);

    StringBuilder[] sbs = new StringBuilder[levels];

    for(int i = 0; i < sbs.length; i ++) {
      sbs[i] = new StringBuilder();
      sbs[i].append("level ").append(i).append(":");
    }

    SkipList_Node cur = root;

    while (cur.getNext(0) != null) {
      cur = cur.getNext(0);
      for(int i = levels - 1; i >= cur.getNext().length; i --) {
        sbs[i].append("\t");
      }
      for(int i = cur.getNext().length - 1; i >= 0; i --) {
        if (cur.getNext(i) == null) {
          levels --;
        }
        sbs[i].append("\t").append(cur.getValue());
      }
    }

    for(int i = sbs.length - 1; i >= 0; i --) {
      sb.append(sbs[i]).append("\n");
    }

    return sb.toString();
  }

  //-------------------------------------------------------
  // student code follows
  // implement the methods of the interface
  //-------------------------------------------------------

}
```

Class SkipList_Playground

```
package SKPLIST_A4;

public class SkipList_Playground {
  public static void main(String[] args) {
    test1();
  }

  private static void test2() {
```

```
    SkipList_Interface list = new SkipList(5);
    System.out.println("=== INSERT ===");
    for(double i = 0; i < 5; i ++) {
      list.insert(i);
      System.out.println(list);
    }
    System.out.println("=== REMOVE ===");
    for(double i = 4; i >= 0; i --) {
      if (list.remove(i)) {
        System.out.println(list);
      }
    }
    System.out.println("=== INSERT ===");
    for(double i = 0; i < 5; i ++) {
      list.insert(i);
      System.out.println(list);
    }
  }

  private static void test1() {
    SkipList_Interface list = new SkipList(5);
    System.out.println("=== INSERT ===");
    for(double i = 0; i < 10; i ++) {
      list.insert(i);
      System.out.println(list);
    }
//    System.out.println(list);
//    System.out.println("=== CONTAINS ===");
//    for(double i = -5; i < 15; i ++) {
//      System.out.println(i + ": " + list.contains(i));
//    }
    System.out.println("=== REMOVE ===");
    for(double i = -5; i < 15; i +=2) {
//      System.out.println(i + ": " + list.remove(i));
      if (list.remove(i)) {
        System.out.println(list);
      }
    }
//    System.out.println(list);
//    System.out.println("=== CONTAINS ===");
//    for(double i = -5; i < 15; i ++) {
//      System.out.println(i + ": " + list.contains(i));
//    }
    System.out.println("=== INSERT ===");
    for(double i = 0; i < 10; i ++) {
      list.insert(i);
      System.out.println(list);
    }
//    System.out.println(list);
  }
}
```

Class RunTests

```
package SKPLIST_A4;

import gradingTools.comp410s20.assignment4.testcases.Assignment4Suite;

public class RunTests {
  public static void main(String[] args){ //runs Assignment 4 oracle tests
    Assignment4Suite.main(args);
  }
}
```

Grading Notes

In the beginning, there is one initial node called root. This is a sentinel, with all the forward pointers of the root node next array null. The double data is Double.NaN, meaning unimportant because it is not used... this sentinel root node is not a data node in the list. Its function is to be a place for all the levels of the list can begin and the pointers to the first data node in each level can be easily found and accessed.

When a new node is inserted, it needs to be placed after the sentinel, and hooked into the necessary levels for the added node. A right place for a newly-created node should be searched with forward pointers starting from the root node. When creating a node for insertion, it should get a level along with double data. The level should be determined writing a flipping loop using the flip() method. The maximum level of a node in skiplist is pre-defined as 30 in class SkipList. The sentinel has this many levels in its link array.

For removing, the target node to be deleted should be first searched with forward pointers starting from the root node, same as insertion. Forward pointers of relevant nodes need to be adjusted correctly and the size of skiplist is decreased by one if remove succeed.

Implementation Notes

You may wish to have two "temp node" allocated in your list so you can keep track of where you are at the various levels as you wander around in the skip list.
