FACULTY OF ENGINEERING
CHULALONGKORN UNIVERSITY
2110211 Introduction to Data Structures

YEAR II, Second Semester, Final Examination, March 7, 2006, Time 8.30 – 11.30

ชื่อ-นามสกุล_____เลขประจำตัว | 4 | 7 | | | | | | | 2 | 1 | CR58_____

หมายเหตุ

1. ข้อสอบมีทั้งหมด 3 แผ่น 6 หน้า 9 ข้อ คะแนนเต็ม 55 คะแนน (คิดเป็น 40% ของทั้งวิชา)
2. ไม่อนุญาตให้นำตำราและเครื่องคำนวณต่างๆ ใดๆ เข้าห้องสอบ
3. เขียนตอบในกระดาษข้อสอบชุดนี้
4. **คะแนนที่ได้จะแปรตามประสิทธิภาพการทำงานของโปรแกรมที่เขียน**
5. **ควรเขียนคำอธิบายหรือยกตัวอย่าง ประกอบโปรแกรมที่เขียนด้วย**
6. ห้ามการหยิบยืมสิ่งใดๆ ทั้งสิ้น จากผู้สอบอื่นๆ เว้นแต่ผู้คุมสอบจะหยิบยืมให้
7. ห้ามนำส่วนใดส่วนหนึ่งของข้อสอบออกจากห้องสอบ ข้อสอบเป็นทรัพย์สินของราชการซึ่งผู้ลักพาอาจมีโทษทางคดีอาญา
8. ผู้ที่ประสงค์จะออกจากห้องสอบก่อนหมดเวลาสอบ แต่ต้องไม่น้อยกว่า 45 นาที ให้นั่งอยู่กับที่นั่งสอบและยกมือ เพื่อให้ผู้คุมสอบเก็บข้อสอบและอนุญาตจึงจะออกจากห้องสอบได้และต้องทำโดยสงบ
9. เมื่อหมดเวลาสอบ ผู้เข้าสอบต้องหยุดการเขียนใดๆ ทั้งสิ้น และนั่งอยู่กับที่นั่งสอบด้วยความสงบจนกว่าผู้คุมสอบจะเก็บข้อสอบจากผู้สอบได้ทั้งหมดและอนุญาตให้ลุกขึ้นและออกจากห้องได้
10. ผู้ที่ไม่ปฏิบัติตามข้อ 5 - 8 จะถือว่าเข้าข่ายทุจริตในการสอบ ตามประกาศคณะวิศวกรรมศาสตร์ ณ วันที่ 6 มกราคม 2546
11. ส่อเจตนาทุจริต จะได้รับ F ในวิชาที่ส่อเจตนาทุจริต และพักการศึกษา 1 ภาคการศึกษา
12. เจตนาทุจริต จะได้รับ F ในวิชาที่เจตนาทุจริต และพักการศึกษา 1 ปีการศึกษา

รับทราบ

ลงชื่อนิสิต (.....)

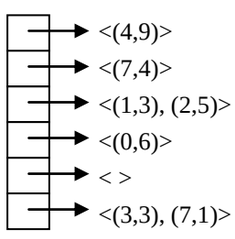
1 (15 คะแนน) เมื่อใดที่เราต้องการใช้เวกเตอร์ในโปรแกรม ก็ย่อมนึกถึงอาเรย์ เช่นเวกเตอร์ (0,3,0,5) ก็แทนด้วยอาเรย์ double[ ] v = {0, 3, 0, 5} เวกเตอร์ที่ยาว *n* ก็ใช้อาเรย์ *n* ช่อง แต่ถ้าเราทราบก่อนล่วงหน้าว่าข้อมูลส่วนใหญ่ในเวกเตอร์มีค่าเป็นศูนย์ ซึ่งเรียกกันว่าเวกเตอร์มากเลขศูนย์ (sparse vector) เราก็สามารถจัดเก็บเวกเตอร์แบบนี้ด้วยรายการที่เก็บเฉพาะข้อมูลที่ไม่ใช่ศูนย์ ซึ่งประหยัดเนื้อที่ได้มาก แถมอาจทำให้การประมวลผลเวกเตอร์บางอย่างเร็วขึ้นด้วย โดยต้องเก็บหมายเลขดัชนีกำกับข้อมูลแต่ละตัวด้วย เช่น เวกเตอร์ (0, 0, 7, 0, 3, 0) ก็เก็บเป็นรายการ ⟨ (2,7), (4,3) ⟩ เพราะช่องที่ 2 และ 4 ของเวกเตอร์มีค่า 7 และ 3 ตามลำดับ นอกนั้นเป็นศูนย์หมด กำหนดให้เรามีคลาส `SparseVector` แทนการจัดเก็บเวกเตอร์มากเลขศูนย์ด้วยรายการที่ได้กล่าวมา โดยมี public methods ต่าง ๆ ดังแสดงในตารางข้างล่างนี้

เมท็อด	หน้าที่	ตัวอย่าง v = (0,3,0,5)
`int length()`	คืนความยาวของเวกเตอร์	`v.length()` ได้ `4`
`double get(int index)`	คืนค่าในช่องที่ `index` ของเวกเตอร์	`v.get(2)` ได้ `0`, `v.get(3)` ได้ `5`
`void set(int index, double value)`	ตั้งค่า `value` ให้ช่องที่ `index`	`v.set(2,9)` ได้ `v = (0, 3, 9, 5)`
`SparseVector add(SparseVector v)`	คืนผลบวกของเวกเตอร์นี้กับ `v`	`v.add(v)` ได้ (0, 6, 0, 10)
`double dot(SparseVector v)`	คืน dot product ของเวกเตอร์นี้กับ `v`	`v.dot(v)` ได้ 0+3×3+0+5×5 = 34
`SparseVector multiply(double c)`	คืนผลคูณของเวกเตอร์นี้กับจำนวนจริง `c`	`v.multiply(2)` ได้ (0, 6, 0, 10)
`SparseVector multiply(SparseMatrix m)`	คืนผลคูณของเวกเตอร์นี้กับเมทริกซ์ `m`	

หมายเหตุ เมท็อด `add`, และ `multiply` คืนเวกเตอร์ใหม่ที่เป็นผลลัพธ์ ไม่มีการเปลี่ยนแปลงเวกเตอร์ที่ถูกเรียก

เราสามารถขยายแนวคิดของการสร้างเวกเตอร์มากเลขศูนย์มาสร้างเมทริกซ์มากเลขศูนย์ (sparse matrix) ได้ง่าย ๆ โดยจัดเก็บเมทริกซ์ด้วยอาเรย์มิติเดียวของเวกเตอร์มากเลขศูนย์หลาย ๆ แถว ดังตัวอย่างในรูปข้างล่างนี้

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 3 & 5 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 1 \end{bmatrix}$$



จงเขียน constructor และเมท็อดที่เว้นว่างไว้ของคลาส `SparseMatrix` ในหน้าถัดไปให้สมบูรณ์ (พยายามใช้ประโยชน์จากเมท็อดของคลาส `SparseVector` จะได้เขียนโปรแกรมได้สั้น เข้าใจง่าย และน่าจะทำงานได้รวดเร็ว)

```
public class SparseMatrix {
    SparseVector [] rows;

    public SparseMatrix(int r, int c) {
```

2 คะแนน

```
    }
    public int numRows() { return rows.length; }
    public int numCols() { return rows[0].length(); }
    public double get(int r, int c) {
        if (r < 0 || r >= numRows() || c < 0 || c >= numCols())
            throw new IndexOutOfBoundsException(r + "," + c);
        return rows[r].get(c);
    }
    public void set(int r, int c, double v) {  // set m[r][c] = v
```

2 คะแนน

```
    }
    public SparseMatrix add(SparseMatrix m) {  // this + m
```

3 คะแนน

```
    }
    public SparseVector multiply(SparseVector v) {  // this × v
```

3 คะแนน

$$\text{for example: } m = \begin{bmatrix} 4 & 0 \\ 5 & 1 \\ 0 & 3 \end{bmatrix},\ v = \begin{bmatrix} 1 \\ 2 \end{bmatrix},$$

$$m.\text{multiply}(v) = \begin{bmatrix} 4 & 0 \\ 5 & 1 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 6 \end{bmatrix}$$

```
    }
    public SparseMatrix multiply(SparseMatrix m) { // this × m
```

5 คะแนน

```
    }
```

2 (2 คะแนน) ต้นไม้ AVL ที่สูง 9 มีได้กี่ nodes (แสดงวิธีทำด้วย) กำหนดให้ต้นไม้ว่างสูง –1

.....

.....

.....

3 (5 คะแนน) คลาส `ArrayQueue` ที่แสดงข้างล่างนี้ คือคลาสที่เราได้ศึกษากันมาในชั้นเรียน มีตัวแปร `front back` และ `size` อยู่ภายใน จงปรับปรุงเมท็อดในคลาสข้างล่างนี้ใหม่ โดยให้<u>ตัดหนึ่งในตัวแปรนี้ออก</u> แล้วยังคงมีพฤติกรรมการทำงานและประสิทธิภาพเชิงเวลาเหมือนของเดิม (เขียนปรับปรุงทับลงใน code ข้างล่างนี้เลย)

```
public class ArrayQueue implements Queue  {
  private Object[] elementData;
  private int size;
  private int front;
  private int back;

  public ArrayQueue() {
    elementData = new Object[10];
    front = 0; back = -1; size = 0;
  }
  public boolean isEmpty() { return size == 0; }
  public int size() { return size; }
  public Object peek() {
    if (isEmpty()) throw new IllegalStateException ();
    return elementData[front];
  }
  public void enqueue(Object e) {
    if (size == elementData.length) {
      Object[] newA = new Object[2 * elementData.length];
      for (int i = 0, j = front; i < size; i++, j = inc(j)) {
        newA[i] = elementData[j];
      }
      front = 0; back = size - 1; elementData = newA;
    }
    back = inc(back);
    elementData[back] = e;
    ++size;
  }
  public Object dequeue() {
    Object e = peek();
    elementData[front] = null; front = inc(front);
    --size;
    return e;
  }
  private int inc(int index) {
    return (index + 1) % elementData.length;
  }




}
```

เลือกตัดออกหนึ่งตัว ขอตัด ..... ออก

4 (5 คะแนน) จงเขียนเมท็อด `equals` ให้กับคลาส `AVLSet` เพื่อเปรียบเทียบว่า<u>เซตสองเซตมีข้อมูลเหมือนกันหรือไม่</u> (ลักษณะของต้นไม้ไม่จำเป็นต้องเหมือนก็ได้) โดยใช้เวลาการทำงานเป็น O(ขนาดของเซตทั้งสองรวมกัน)

```
public class AVLSet implements Set  {
  ...
  public boolean equals(Object obj) {
    if (!(obj instanceof AVLSet)) return false;
    AVLSet that = (AVLSet) obj;
```

5 (5 คะแนน) ก) เมท็อด add ของคลาส BSTSet ที่ได้ศึกษากันมาจะนำข้อมูลใหม่มาสร้างเป็นใบใหม่ของต้นไม้ จงเขียน add ใหม่ ให้หลังการเพิ่มแล้วข้อมูลใหม่ปรากฏที่รากของต้นไม้ โดยใช้เวลาการทำงานเป็น O(ความสูงของต้นไม้)

(2 คะแนน) ข) ภายในคลาส BSTSet ข้างล่างนี้มีเมท็อด whatIsThis ซึ่งเรียกใช้ add ของข้อ ก อยากทราบว่า whatIsThis ทำอะไร สรุปหน้าที่ของเมท็อดนี้สั้น ๆ

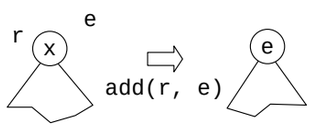
```
public class BSTSet implements Set  {
  private static class Node {
    Object element;
    Node left, right;
    Node(Object e, Node l, Node r) { element = e; left = l; right = r; }
  }
  private Node root;
  private int size;
  ...
  public void whatIsThis(BSTSet s) {
    root = whatIsThis(root, s.root);
  }
  private Node whatIsThis(Node a, Node b) {
    if (a == null) return b;
    if (b == null) return a;
    add(b, a.element);
    b.left = whatIsThis(a.left, b.left);
    b.right = whatIsThis(a.right, b.right);
    return b;
  }
  public void add(Object e) {
    root = add(root, e);
  }
  private Node add(Node r, Object e) {
```

whatIsThis ทำอะไร ?

.....

.....

.....



6 (3 คะแนน) จงเขียนเมท็อด equals ให้กับคลาส QuadraticProbingHashSet เพื่อเปรียบเทียบว่าเซตสองเซตมีข้อมูลเหมือนกันหรือไม่

```
public class QuadraticProbingHashSet implements Set  {
  private static Object deleted = new Object();
  private Object[] table;
  private int size;
  private int numOccupied;
  ...
  public boolean equals(Object obj) {
    if (!(obj instanceof QuadraticProbingHashSet)) return false;
    QuadraticProbingHashSet that = (QuadraticProbingHashSet) obj;
```

7 (5 คะแนน) จงออกแบบคลาสที่ `implements PriorityQueue` ที่เมท็อด `isEmpty`, `size`, `peek`, และ `enqueue` ใช้เวลาคงตัว O(1) ยกเว้น `dequeue` ที่ใช้เวลาเป็น O(*n*)

```
public class PQ implements PriorityQueue  {
```

8 (3 คะแนน) ถ้าเราอยากให้ผู้ใช้สามารถนำออปเจกต์ของ `ArrayList` ไปเก็บในแต่ละช่องของ hash table ได้ เราก็ต้องเขียน `hashCode` ให้กับคลาส `ArrayList` เมท็อดที่เขียนในกรอบทางขวาข้างล่างนี้เป็น `hashCode` "ที่ไม่ค่อยดี" จงเขียน `hashCode` "ที่ดี" ให้กับคลาส `ArrayList` อธิบายด้วยว่า `hashCode` ที่ออกแบบ "ดี" อย่างไร

```
public class ArrayList implements List  {
  private Object[] elementdata;
  private int size;
  ...
  public int hashCode() {

  }
```

```
public int hashCode() {
  int h = 0;
  for(int i=0; i<elementData.length; i++)
    h += elementData[i].hashCode();
  return h;
}
```

9 (10 คะแนน) จงเขียนเมท็อด `shellSort`, `mergeSort` และ `quickSort` ข้างล่างนี้ให้สมบูรณ์

```
public class ArrayUtil {
  private static void merge(Object[] a, int al, int ar,
          Object[] b, int bl, int br, Object[] c, int cl ) {
    int i = al, j = bl, n = (ar-al+1)+(br-bl+1);
    for (int k = cl; k < cl+n; k++) {
      if (i > ar) { c[k] = b[j++]; continue; }
      if (j > br) { c[k] = a[i++]; continue; }
      c[k] = lessThan(a[i], b[j]) ? a[i++] : b[j++];
    }
  }
  private static int partition(Object[] d, int left, int right) {
    Object p = d[left];
    int i = left, j = right + 1;
    while (i < j) {
      while (lessThan(p, d[--j])) ;
      while (lessThan(d[++i], p)) if (i == right) break;
      if (i < j) swap(d, i, j);
    }
    swap(d,left,j);
    return j;
  }
  public static void shellSort(Object[] d, int left, int right) {
    int h;
    for (h = 1; h <= (right-left)/9; h = 3*h+1);   // Knuth's sequence
    for ( ;h > 0; h /= 3) {
      for (int i = left+h; i <= right; i++) {
```

เหมือนที่เรียนทุกประการ

เหมือนที่เรียนทุกประการ

4 คะแนน

```
      }
    }
  }
  public static void quickSort(Object[] d, int left, int right) {
```

3 คะแนน

```
  }
  public static void mergeSort(Object[] d, int left, int right) {
```

3 คะแนน

```
 }
```