FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2110211 Introduction to Data Structures

YEAR II, Second Semester, Midterm Examination, May 6, 2015, Time 13:00 – 16:00

ชื่อ-นามสกุล_____เลขประจำตัว | | | | | | | | | 2 | 1 | CR58_____

หมายเหตุ

1. ข้อสอบมีทั้งหมด 12 ข้อในกระดาษคำถามคำตอบจำนวน 10 แผ่น 10 หน้า คะแนนเต็ม 100 คะแนน
2. ไม่อนุญาตให้นำตำราและเครื่องคำนวณต่างๆ ใดๆ เข้าห้องสอบ
3. ควรเขียนตอบด้วยลายมือที่อ่านง่ายและชัดเจน สามารถใช้ดินสอเขียนคำตอบได้
4. ห้ามการหยิบยืมสิ่งใดๆ ทั้งสิ้น จากผู้สอบอื่นๆ เว้นแต่ผู้คุมสอบจะหยิบยืมให้
5. ห้ามนำส่วนใดส่วนหนึ่งของข้อสอบออกจากห้องสอบ ข้อสอบเป็นทรัพย์สินของราชการซึ่งผู้ลักพาอาจมีโทษทางคดีอาญา
6. ผู้เข้าสอบสามารถออกจากห้องสองได้ หลังจากผ่านการสอบไปแล้ว 45 นาที
7. เมื่อหมดเวลาสอบ ผู้เข้าสอบต้องหยุดการเขียนใดๆ ทั้งสิ้น
8. ผู้ที่ปฏิบัติเข้าข่ายทุจริตในการสอบ ตามประกาศคณะวิศวกรรมศาสตร์
   มีโทษ คือ ได้รับ สัญลักษณ์ F ในรายวิชาที่ทุจริต และพักการศึกษาอย่างน้อย 1 ภาคการศึกษา

รับทราบ

ลงชื่อนิสิต (.....)

หมายเหตุเพิ่มเติม: ในข้อที่มีการเขียนโปรแกรมนั้น จะมีการพิจารณาประสิทธิภาพในการทำงานของโปรแกรมที่เขียนมาด้วย ให้พยายามเขียนโปรแกรมที่ทำงานได้เร็วที่สุดและใช้หน่วยความจำน้อยที่สุดเท่าที่เป็นไปได้

0. ในวิชานี้ อัตราส่วนของคะแนนสอบย่อย (quiz) ครั้งสุดท้ายอยู่ที่ 25% และข้อสอบปลายภาคอยู่ที่ 30% นิสิตสามารถเลือกปรับลดน้ำหนักของคะแนนสอบย่อยครั้งสุดท้ายให้เป็นได้ตั้งแต่ 15% ถึง 25% โดยค่าน้ำหนักในส่วนที่ลดไปนั้นจะถูกนำไปคิดเพิ่มให้กับน้ำหนักของข้อสอบปลายภาคแทน จงระบุอัตราส่วนของคะแนนสอบย่อยครั้งสุดท้ายที่ต้องการ โดยใส่เป็นตัวเลขจำนวนเต็มตั้งแต่ 15 ถึง 25 ลงในช่องด้านขวานี้ (ถ้าไม่ระบุหรือระบุค่าที่ไม่ถูกต้องจะถือว่าระบุเลข 25 ไว้) ******(ตัวอย่างการคิดคะแนน ถ้ากรอกเลข 17 หมายความว่าคะแนนสอบย่อยครั้งสุดท้ายจะคิดเป็น 17% และสอบปลายภาคเป็น 38%) ******

1. (10 คะแนน) จงวิเคราะห์อัตราการเติบโตของส่วนของโปรแกรมต่อไปนี้ **ให้ตรงกับความเป็นจริงมากที่สุด** โดยให้เขียนอยู่ในรูปสัญกรณ์เชิงเส้นกำกับในตัวแปร n

`int a(int n) {`
`  if (n > 0) n = 3*a(n/2) + a(n/2);`
`  return n;`
`}`
	มีอัตราการเติบโต เป็น $\Theta($ _____ $)$

`// x is a vector of n elements`
`void ana1(vector<int> &x) {`
`  x.insert(x.end()-1, 99);`
`}`
	มีอัตราการเติบโต เป็น $\Theta($ _____ $)$

`// x is a list of n elements`
`//  (circular-doubly-linked with header)`
`void ana2(list<int> &x) {`
`  x.push_back( x.front() );  x.pop_front();`
`}`
	มีอัตราการเติบโต เป็น $\Theta($ _____ $)$

`// x is a priority queue of n elements (min binary heap)`
`void ana3(priority_queue<int> &x) {`
`  x.push( x.top() );  x.pop();`
`}`
	มีอัตราการเติบโต เป็น $\Theta($ _____ $)$

`// x is a map of n elements (AVL tree)`
`int ana4(map_avl<int,int> &x) {`
`  return x.begin()->first;`
`}`
	มีอัตราการเติบโต เป็น $\Theta($ _____ $)$

ชื่อ_____หมายเลขประจำตัว_____เลขที่ใน CR58_____

2. (5 คะแนน) จงเติมคำตอบลงในช่องว่าง เพื่อให้ฟังก์ชันต่อไปนี้ มีอัตราการเติบโตของเวลาการทำงานเป็นไปตามที่กำหนดให้

`void b(int n) {`
`  if (n > 0)`
`  b(_____);`
`}`
	มีอัตราการเติบโต เป็น $\Theta(\log n)$

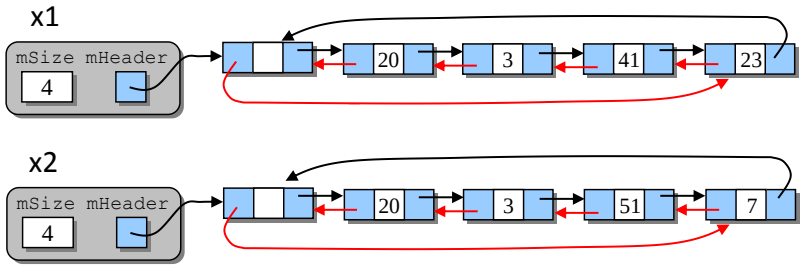
`// x is an empty`
`//  (non-circular-singly-linked with header)`
`void ana5(list<int> &x, int n) {`
`  for (int i = 0; i < n; i++) {`
`  x.push_back( _____ );`
`  }`
`}`
	มีอัตราการเติบโต เป็น $\Theta(n)$

`// x is an empty priority queue (min binary heap)`
`void ana6(priority_queue<int> &x, int n) {`
`  for (int i = 0; i < n; i++) {`
`  x.push( _____ );`
`  }`
`}`
	มีอัตราการเติบโต เป็น $\Theta(n)$

`// x is an empty map_bst (binary search tree)`
`void ana7(map_bst<int,int> &x, int n) {`
`  for (int i = 0; i < n; i++) {`
`  x.insert( make_pair( _____ , 99) );`
`  }`
`}`
	มีอัตราการเติบโต เป็น $\Theta(n^2)$

`// x is an empty unordered_map`
`//  (separate chaining hash table`
`//  with the table size of 13 and no rehashing)`
`void ana8(unordered_map<int,int> &x, int n) {`
`  for (int i = 0; i < n; i++) {`
`  x[ i*13 ] = 99;  // assume no rehashing`
`  }`
`  for (int i = 0; i < n; i++) {`
`  find( _____ );`
`  }`
`}`
	มีอัตราการเติบโต เป็น $\Theta(n^2)$

3. (10 คะแนน) ngong1 และ ngong2 เป็นฟังก์ชันอยู่ใน class list ที่นิสิตได้เรียนโครงสร้างและการทำงานภายในกันในวิชานี้ ถ้ากำหนดให้ x1 และ x2 เป็น list มีโครงสร้างภายในดังรูปข้างล่างนี้



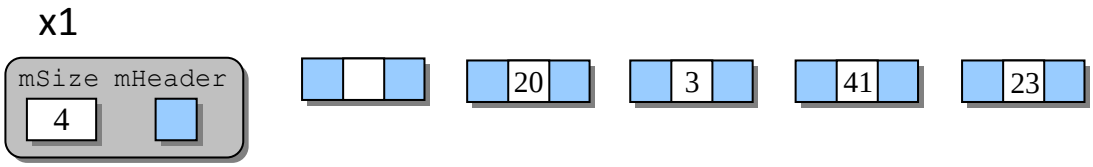
`void ngong1() {`
`  node *p = mHeader;`
`  do {`
`  node *q = p->prev;`
`  p->prev = p->next;`
`  p->next = q;`
`  p = q;`
`  } while(p != mHeader);`
`}`
	ตอบข้อ 3.3

```
void ngong2() {
  if (size()<=1) return;
  node *p1 = mHeader->prev;
  while(p1 != mHeader) {
    node *q = mHeader->next;
    while(q->next != mHeader) {
      if (q->data < q->next->data) {
        node *p  = q->prev;
        node *n  = q->next;
        node *nn = n->next;
        p->next  = n;
        nn->prev = q;
        q->prev  = n;
        q->next  = nn;
        n->prev  = p;
        n->next  = q;
      } else {
        q = q->next;
      }
    }
    p1 = p1->prev;
  }
}
```

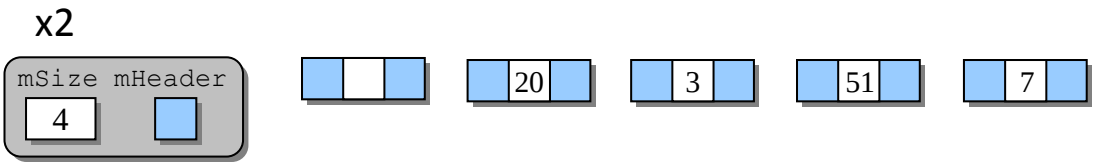
ตอบข้อ 3.3

อยากทราบว่า

3.1 หลังจากคำสั่ง **x1.ngong1()**ทำงาน จะเปลี่ยนโครงสร้างภายใน **x1** อย่างไร (วาดเส้นเชื่อมต่าง ๆ ให้ครบในรูปข้างล่างนี้)

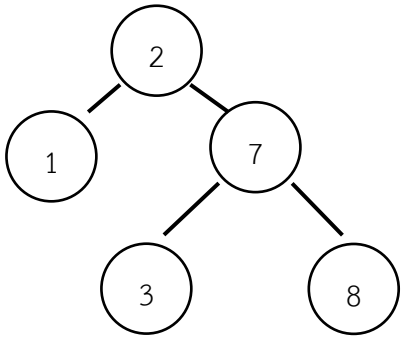


3.2 หลังจากคำสั่ง **x2.ngong2()** ทำงาน จะเปลี่ยนโครงสร้างภายใน **x2** อย่างไร (วาดเส้นเชื่อมต่าง ๆ ให้ครบในรูปข้างล่างนี้)



3.3 เขียนสรุปสั้น ๆ ว่า ngong1 และ ngong2 ทำอะไร และจงตั้งชื่อใหม่ให้กับฟังก์ชันทั้งสองที่สื่อความหมาย (เขียนตอบในด้านขวาของตารางข้างบน)

4. (4 คะแนน) จากต้นไม้ AVL ดังรูปนี้



จงวาดผลลัพธ์หลังจากการเพิ่มปม 9, 4, 5, 6 เข้าไปในต้นไม้ AVL นี้ตามลำดับ

(1) หลังจากเพิ่ม 9	(2) หลังจากเพิ่ม 4
(3) หลังจากเพิ่ม 5	(4) หลังจากเพิ่ม 6

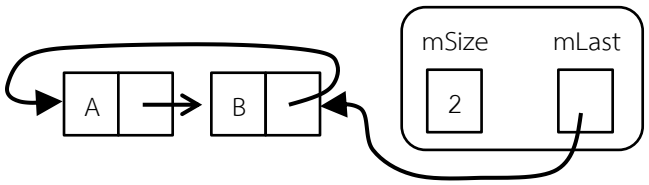
5. (10 คะแนน) จงเติมตารางแฮชแบบที่อยู่เปิด (Open Addressing Hash Table) โดยใช้ฟังก์ชั่นแฮชดังต่อไปนี้

   h(x) = (x % 13) โดยข้อมูลคือ 1, 10, 3, 9, 24, 50, 14, 27, 23, 40

   ตามลำดับโดยใช้การแก้การชนแบบกำลังสอง (Quadratic Probing) โดยที่ตารางเริ่มต้นเป็นตารางว่าง โดยให้เขียนผลของการใส่ข้อมูลทีละตัวลงไปในแต่ละแถวของตารางข้างล่างนี้

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
(ตัวอย่าง) หลังจาก เพิ่ม 1		1											
หลังจาก เพิ่ม 10		1											
หลังจาก เพิ่ม 3		1											
หลังจาก เพิ่ม 9		1											
หลังจาก เพิ่ม 24		1											
หลังจาก เพิ่ม 50		1											
หลังจาก เพิ่ม 14		1											
หลังจาก เพิ่ม 27		1											
หลังจาก เพิ่ม 23		1											
หลังจาก เพิ่ม 40		1											

6. (5 คะแนน) สำหรับโครงสร้างข้อมูลแบบ Singly Circular Linked List ซึ่งมีโครงสร้างดังตัวอย่างในรูป จงเขียนฟังก์ชัน void push_back(T &e) ซึ่งทำการเพิ่มข้อมูล e ไปยังด้านท้ายของ list นี้ ในฟังก์ชัน push_back ห้ามเรียก insert




```
template <typename T>
class list {
  protected:
    class node {
      public:
        T data;
        node *next;
        node() : data(T()), next(NULL) { }
        node(T e, node* a_next) : data(e), next(a_next) { }
    };
  node *mLast;
  size_t mSize;
  //คลาสนี้มีฟังก์ชันอื่น ๆ ตามปรกติทุกอย่าง และสามารถเรียกใช้ได้ ให้ทำการแก้ไขเฉพาะฟังก์ชัน  push_back เท่านั้น
  public:
```

```
    void push_back(T &e) {









   }
};
```

7. (6 คะแนน) จงเติมส่วนของโปรแกรมด้านล่างของคำสั่ง rotate_left_child, rotate_right_child และ rebalance ของ map_avl ซึ่งเป็น AVL Tree ให้สมบูรณ์ โดยให้นิสิตเติมเฉพาะบริเวณที่ขีดเส้นใต้เท่านั้น

```
node *rotate_left_child(node *r) {
   node *new_root = r->left;
   r->set_left(new_root->_______);
   ____________________
   new_root->right->set_height();
   new_root->set_height();
   return new_root;
 }
 node *rotate_right_child(node * r) {
   node * new_root = r->right;
   r->set_right(new_root->_______);
   ____________________
   new_root->left->set_height();
   new_root->set_height();
   return new_root;
 }
```

```
node *rebalance(node *r) {
    if (r == NULL) return r;
    int balance = r->balanceValue();
    if (balance == -2) {
      if (r->left->balanceValue() == 1)
        ______________________________
      ____________________
    } else if (balance == 2) {
      if (r->right->balanceValue() == -1)
        r->set_right(rotate_left_child(r->right));
      r = rotate_right_child(r);
    }
    r->set_height();
    return r;
 }
```

8. (5 คะแนน) ฮีปแบบทวิภาค (Binary Heap) นั้นมีโครงสร้างเป็นต้นไม้ทวิภาค (binary tree) ซึ่งแต่ละปมมีลูกไม่เกินสองปม ให้ทำการดัดแปลงฮีปแบบทวิภาค โดยเปลี่ยนจากการใช้ต้นไม้ทวิภาค เป็นต้นไม้ไตรภาค (3-ary tree) ซึ่งต้นไม้ดังกล่าวจะมีลูกจำนวน 3 ลูก แทนที่จะมีแค่ 2 ลูก โดยที่หลักการทำงานของฮีปนั้นยังเป็นเหมือนเดิม จงเขียนฟังก์ชัน FixDown ของฮีปไตรภาคนี้

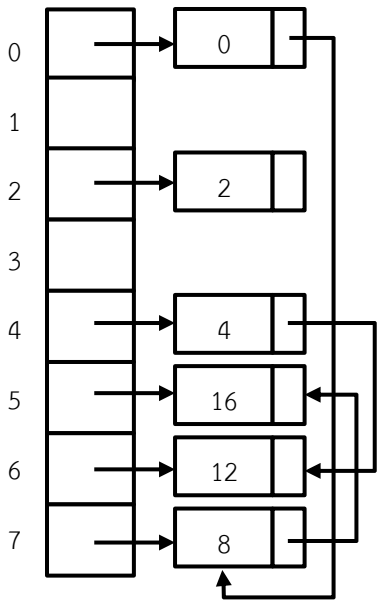
```
template <typename T,typename Comp = std::less<T> >
class priority_queue {
  protected:
    T *mData;    size_t mCap;    size_t mSize;    Comp mLess;
    //คลาสนี้มีฟังก์ชันอื่น ๆ ตามปรกติทุกอย่าง และสามารถเรียกใช้ได้ ให้ทำการแก้ไขเฉพาะฟังก์ชัน FixDown เท่านั้น
    void fixDown(size_t idx) {









    }
};
```

9. (10 คะแนน) จงเขียนคลาสที่ทำงานแบบ priority queue แบบค่ามากสำคัญ ของข้อมูลจำนวนเต็ม โดยให้ใช้อาเรย์ในการเก็บข้อมูล กำหนดให้สมาชิกของคลาสนี้ได้แก่ mData เป็นอาเรย์ของ int สำหรับเก็บข้อมูล mSize เป็นจำนวนข้อมูลใน priority queue ของเรา และ mCap เป็นขนาดของอาเรย์ที่ได้จองไว้แล้ว จงเขียนฟังก์ชันต่อไปนี้ให้ทำงานด้วยประสิทธิภาพที่กำหนดให้ <u>นอกจากนี้ ให้สังเกตว่ามันเป็นไปได้ที่จะมีการใส่ข้อมูลเข้ามาจนมีจำนวนข้อมูลมากกว่า mCap เริ่มต้น คลาสที่เขียนขึ้นนี้จะต้องรองรับการทำงานดังกล่าวด้วย</u>

- void push(int x): เอา x มาเก็บ priority queue ต้องใช้เวลา O(n) (ต้องพิจารณากรณีที่ใส่ข้อมูลเพิ่มตอนที่มีข้อมูลอยู่เต็มด้วย)
- int top(): ดึงค่าที่สำคัญที่สุดในอาร์เรย์ออกมา ต้องใช้เวลา O(1)
- void pop(): เอาค่าที่สำคัญที่สุดในอาร์เรย์ทิ้งไป ต้องใช้เวลา O(1)

```
class MyPriorityQueue {
  protected:
    int* mData;
    int  mSize, mCap;

  public:
    MyPriorityQueue() { mData = new int[1]; mCap = 1; mSize = 0;}
    ~MyPriorityQueue() { delete mData; }
    void push(int x){









    }

    int top(){


    }

    void pop() {


    }
};
```

10. (10 คะแนน) โครงสร้างข้อมูลแบบ Hash Table นั้นมีวิธีการแก้การชนกันอยู่สองแบบคือ Open Addressing (OA) และ Separate Chaining (SC) ซึ่งต่างก็มีข้อดีและข้อเสียแตกต่างกันไป ในข้อนี้เราจะพิจารณาการแก้การชนแบบใหม่ที่รวมเอาการแก้การชนทั้งสองแบบเข้าไว้ด้วยกัน เรียกว่า Coalesced Hashing (CH) ซึ่งมีวิธีการทำงานดังนี้ หนึ่งช่องในตารางแฮชจะเก็บข้อมูลหนึ่งตัวเช่นเดียวกับ OA แต่ว่าในแต่ละช่องนั้นแทนที่จะเป็นตัวแปรประเภท T เราจะใช้ตัวแปรที่เป็น pointer ไปยัง class node ของ singly linked list (คือมี pointer ชื่อ next เอาไว้ชี้ไปยังข้อมูลตัวถัดไป) แทน โดยกำหนดให้ถ้า pointer ดังกล่าวเป็น null จะแสดงว่าช่องนั้นว่าง แต่ถ้า pointer ดังกล่าวไม่ใช่ null ก็แสดงว่าช่องดังกล่าวมีข้อมูลอยู่ สำหรับช่องที่มีข้อมูลนั้น ตัวแปร next ของปมในช่องดังกล่าวจะชี้ไปยัง “ช่องที่เก็บข้อมูลตัวถัดไปที่ hash มาลงช่องเดียวกัน” หรือกล่าวอีกนัยหนึ่งคือ CH นั้นก็คือ separate chaining ที่ใช้ช่องต่าง ๆ เก็บปม 1 ปม แทนที่จะยอมให้หนึ่งช่องเก็บได้หลายปมนั่นเอง รูปทางขวามือแสดงตัวอย่างของ Coalesced Hashing สำหรับการใส่ข้อมูลนั้น ให้ h(x) คือค่า hash ของข้อมูล x ถ้าช่อง h(x) ว่าง เราก็จะสร้างปมที่มีข้อมูล x และมี next เป็น null แล้วให้ pointer ของช่อง h(x) นั้นชี้มาที่ปมดังกล่าว แต่ถ้าช่อง h(x) ไม่ว่าง เราจะหาช่องว่าง โดยเริ่มหาจาก “ช่องสุดท้าย” ของตารางแฮช แล้วไล่ดูทีละช่องย้อนลงมาเรื่อย ๆ จนกระทั่งเจอช่องว่าง สมมติให้ a คือหมายเลขช่องที่ว่าง เราจะสร้างปมใหม่ที่มีข้อมูลที่ต้องการจะใส่ แล้วให้ช่อง a นั้นชี้



มาที่ปมใหม่นี้ นอกจากนั้นแล้ว เราจะกลับไปไล่ดูปมจากช่อง h(x) โดยจะวิ่งตาม pointer next เริ่มต้นที่ h(x) ไปเรื่อย ๆ จนกระทั่ง next มีค่าเป็น NULL แล้วจึงให้ next ของช่องดังกล่าวชี้ไปยังปมในช่อง a แทน เราจะเห็นได้ว่า การใส่ปมลงไปใน CH นั้นจะคล้ายคลึงกับการเพิ่มปมลงไปในตำแหน่งท้ายสุดของ list นั่นเอง รูปในหน้าที่แล้วเกิดจากการใส่ข้อมูล 0, 2, 4, 8, 12, 16 ตามลำดับ เมื่อให้ h(x) = x % 8 จงเขียนฟังก์ชันสำหรับ class coalesced_hash สำหรับเก็บข้อมูลประเภท int โดยให้เขียนรายละเอียดของฟังก์ชัน void insert(int e) สำหรับการเพิ่มข้อมูล e ลงไปใน CH และฟังก์ชัน bool find(int e) สำหรับค้นหาว่ามีข้อมูล e อยู่ใน CH หรือไม่ โดยให้ทำงานตามรูปแบบของ CH ที่ได้อธิบายไว้ข้างต้น กำหนดให้ hash function h(x) = x % ขนาดของตาราง ในข้อนี้ ให้ถือว่าตาราง hash มีขนาดใหญ่เพียงพอ (ไม่จำเป็นต้องพิจารณากรณีที่ต้องขยายตาราง hash)

```
class coalesced_hash {
  protected:
    class node {
      friend class coalesced_hash;
      protected:  int data;  node  *next;
      public:
        node() { }
        node(int a_data,node *a_next) : data(a_data), next(a_next) { }
    };
    vector<node*> mBuckets;
    int mSize;

  public:
    coalesced_hash() { mBuckets.resize(1007); mSize = 0; }

    bool find(int e) {









    }
    void insert(int e) {









    }
};
```

11. (10 คะแนน) สำหรับต้นไม้ค้นหาแบบทวิภาค (Binary Search Tree) นั้น เราเรียกปม x ว่าเป็น “ปมบรรพบุรุษ” ของปม a ก็ต่อเมื่อเส้นทางจากปมรากไปยังปม a นั้นจะต้องผ่านปม x ก่อน กำหนดให้ “ปมบรรพบุรุษร่วมลึกสุด” ของปม a และ b คือปม x ที่เป็นปมบรรพบุรุษของทั้งปม a และ ปม b โดยที่ปม x นั้นจะต้องอยู่ห่างจากปมรากมากที่สุด สำหรับคลาส CP::map_bst นั้น จงเขียนฟังก์ชัน iterator find_lca(iterator ia, iterator ib) ซึ่งจะคืน iterator ที่ชี้ไปยังปม “บรรพบุรุษร่วมลึกสุด” ของปม a และ b นอกจากนี้ ฟังก์ชันนี้จะต้องไม่เปลี่ยนแปลงข้อมูลใด ๆ ภายในต้นไม้โดยเด็ดขาด รับประกันว่ามีปม a, b อยู่ในต้นไม้

```
template <typename KeyT, typename MappedT, typename CompareT = std::less<KeyT> >
class map_bst { //คลาส map_bst, iterator และ node มีฟังก์ชันอื่น ๆ ตามปรกติทุกอย่าง และสามารถเรียกใช้ได้ตามปรกติ
protected:
    typedef std::pair<KeyT,MappedT> ValueT;
    class node {
      protected: ValueT data;  node  *left; node  *right;  node  *parent;
    };
    class iterator {
      protected: * ptr;
      public:
        iterator() : ptr( NULL ) { }
        iterator(node *a) : ptr(a) { }
    };
    node     *mRoot;    CompareT  mLess;    size_t    mSize;
public:
  iterator find_lca(iterator ia, iterator ib) {









};
```

12. (15 คะแนน) โจทย์ข้อนี้เป็นการออกแบบโครงสร้างข้อมูลสำหรับเว็บไซต์อ่านหนังสือออนไลน์ ชื่อ CPReadr โดย CPReadr นั้นจะเก็บข้อมูลหนังสืออยู่เป็นจำนวนมาก (ประมาณ 1,000,000 เล่ม) หนังสือแต่ละเล่มจะมีข้อมูลดังต่อไปนี้ ชื่อหนังสือเป็น string, ชื่อผู้แต่งเป็น string, ข้อมูล pdf ของหนังสือเป็น vector<unsigned char> (unsigned char เป็นข้อมูลขนาด 1 byte), ข้อมูลคะแนนรีวิวของหนังสือเล่มดังกล่าว และสุดท้ายข้อมูล “ประเภท” ของหนังสือ เนื่องจากหนังสือแต่ละเล่มอาจจะเป็นได้หลายประเภท สำหรับหนังสือแต่ละเล่มนั้นข้อมูลประเภทจะประกอบด้วย string หลายตัว โดยที่ข้อมูล string แต่ละตัวคือชื่อประเภทของหนังสือ (เช่น หนังสือ “Linear Algebra” ของ “Jim Hefferon” อาจจะถูกจัดเป็นประเภท “textbook” และ “math” พร้อม ๆ กัน)
เราทราบรายละเอียดของหนังสือแต่ละเล่มว่าเป็นดังนี้

- เราสามารถแยกแยะหนังสือแต่ละเล่มได้ด้วยชื่อหนังสือ**<u>พร้อมด้วย</u>**ชื่อผู้แต่ง ให้พิจารณาว่ามีหนังสือหลายเล่มที่มีชื่อหนังสือเหมือนกันแต่ชื่อผู้แต่งต่างกัน และมีหนังสือหลายเล่มที่ชื่อผู้แต่งเหมือนกันแต่ชื่อหนังสือไม่เหมือนกัน
- หนังสือแต่ละเล่ม**<u>มีประเภทอยู่ไม่เกิน 3 ประเภท</u>**และหนังสือทั้งหมด มี**<u>ประเภทที่แตกต่างกันทั้งหมดไม่เกิน 999 ประเภท</u>**
- ข้อมูล pdf นั้นมีความยาวมาก โดยเฉลี่ยอยู่ที่ 5MB
- ข้อมูลรีวิวนั้นเป็นจำนวนจริง มีค่าตั้งแต่ 0.00 ถึง 10.00 **<u>รับประกันว่าไม่มีหนังสือสองเล่มใด ๆ มีคะแนนรีวิวเท่ากัน</u>**

เว็บไซต์ CPReadr นั้นจะให้บริการค้นหาหนังสือด้วยชื่อหนังสือพร้อมด้วยชื่อผู้แต่ง และต้องแสดงข้อมูลต่าง ๆ ของหนังสือเหล่านั้น นอกจากนี้ เมื่อผู้ใช้ค้นหาหนังสือเจอ เว็บไซต์นี้จะแนะนำหนังสือเล่มอื่น ๆ ให้อีกด้วย โดยเว็บไซต์จะแนะนำหนังสือที่มีประเภทเหมือนกันทั้งหมดเท่านั้น (ตัวอย่างเช่น หนังสือเล่มหนึ่งมีประเภทเป็น {“text”, “romance”, “kid”} จะถือว่ามีประเภทตรงกับ {“romance”, “text”, “kid”} แต่ไม่ตรงกับ {“text”, “romance”, “kid”, “math”} โดยจะแนะนำหนังสือที่มีประเภทตรงกันที่มีคะแนนรีวิวสูงสุด K อันดับแรก นอกจากนี้ ระบบจะต้องสามารถ เพิ่ม, ลด หนังสือได้ด้วย

ก) จงอธิบายการออกแบบคลาส Book ที่มีหน้าที่เก็บข้อมูลหนังสือ**<u>หนึ่งเล่ม</u>**ตามข้อมูลข้างต้น โดยให้เขียน code ของคลาสดังกล่าวขึ้นมาพร้อมระบุว่าคลาสนั้นมีสมาชิกและฟังก์ชันอะไรบ้าง และทำหน้าที่หรือให้บริการอะไร

```
class Book {









};
```

ข) จงออกแบบโครงสร้างข้อมูล CPReadr โดยให้เขียนคลาส CPReadr ที่ใช้งานได้จริง ซึ่งต้องมีบริการดังต่อไปนี้

- CPReadr() เป็น constructor ที่จะต้องเรียกก่อนใช้งานเสมอ
- void add_book(string title, string author, vector<unsigned char> pdf, vector<string> type, double score) เป็นการเพิ่มหนังสือเข้าไปในระบบ โดยเมื่อเพิ่มหนังสือแล้วจะไม่มีการเปลี่ยนแปลงข้อมูลใด ๆ ของหนังสือ
- Book* search(string title, string author) เป็นการค้นหาหนังสือจากชื่อเรื่องและชื่อผู้แต่ง ถ้าไม่มีให้คืนค่า NULL;
- void remove_book(string title, string author) เป็นการลบหนังสือออกจากระบบ (เป็นไปได้ที่จะไม่มีหนังสือดังกล่าวอยู่)
- vector<Book*> recommend(Book &b, int K) เป็นการหาหนังสือแนะนำที่มีประเภทตรงกับหนังสือ b ที่มีคะแนนรีวิวสูงสุด K อันดับ (ให้ระวังด้วยว่าต้องไม่แนะนำหนังสือ book และอาจจะมีหนังสือที่มีประเภทตรงกันไม่ถึง K)

```
class CPReadr { //ให้เขียนอธิบายด้วยว่าสมาชิกแต่ละตัวในคลาสนี้ทำหน้าที่เก็บข้อมูลอะไร




};
```

Common

All classes support these two capacity functions;

Capacity	size_t size(); // return the number of items in the structure
	bool empty(); // return true only when size() == 0

Container Class

All classes in this category support these two iterator functions.

Iterator	iterator begin(); // an iterator referring to the first element
	iterator end(); // an iterator referring to the *past-the-end* element

Class vector<ValueT>, list<ValueT>

Element Access	operator[] (size_t n);

Modifier ที่ใช้ได้ทั้ง list และ vector	void push_back(const ValueT& val);
	void pop_back();
	iterator insert(iterator position, const ValueT& val);
	iterator insert(iterator position, InputIterator first, InputIterator last);
	iterator erase(iterator position);
	iterator erase(iterator first, iterator last);

Modifier ที่ใช้ได้เฉพาะ list	void push_front(const ValueT& val);
	void pop_font;
	void remove(const ValueT& val);

Class set<ValueT,CompareT = less<ValueT> >,
unordered_set<ValueT, HashT = hash< ValueT >, EqualT = equal_to< ValueT > >

Operation	iterator find (const ValueT& val); size_type count (const ValueT& val);
Modifier	pair<iterator,bool> insert (const ValueT& val); void insert (InputIterator first, InputIterator last); iterator erase (iterator position); iterator erase (iterator first, iterator last); size_type erase (const ValueT& val);

Class map<KeyT, MappedT, CompareT = less<KeyT> >
unordered_map<KeyT, MappedT, HashT = hash<KeyT>, EqualT = equal_to<KeyT> >

Element Access	MappedT& operator[] (const KeyT& k);
Operation	iterator find (const KeyT& k); size_type count (const KeyT& k);
Modifier	pair<iterator,bool> insert (const pair<KeyT,MappedT>& val); void insert (InputIterator first, InputIterator last); iterator erase (iterator position); iterator erase (iterator first, iterator last); size_type erase (const KeyT& k);

Container Adapter

These three data structures support the same data modifiers but each has different strategy. These data structures do not support iterator.

Modifier	void push (const ValueT& val); // add the element
	void pop(); // remove the element

Class queue<ValueT>

Element Access	ValueT front(); ValueT back();

Class stack<ValueT>

Element Access	ValueT top();

Class priority_queue<ValueT, ContainerT = vector<ValueT>, CompareT = less<ValueT> >

Element Access	ValueT top();

Useful function

```
iterator find(iterator first, iterator last, const T& val);  // find by iteration, using O(N)
void sort (iterator first, iterator last, Compare comp);  // sort, using O( N log(N) )
pair<T1,T2> make_pair (T1 x, T2 y);
```