FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2110211 Introduction to Data Structures

YEAR II, Second Semester, Final Examination, May 18, 2017, Time 13:00 – 16:00

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

หมายเหตุ

1. ข้อสอบมีทั้งหมด 11 ข้อในกระดาษคำถามคำตอบจำนวน 8 แผ่น 8 หน้า คะแนนเต็ม 73 คะแนน
2. ไม่อนุญาตให้นำตำราและเครื่องคำนวณต่างๆ ใดๆ เข้าห้องสอบ
3. ควรเขียนตอบด้วยลายมือที่อ่านง่ายและชัดเจน สามารถใช้ดินสอเขียนคำตอบได้
4. ห้ามการหยิบยืมสิ่งใดๆ ทั้งสิ้น จากผู้สอบอื่นๆ เว้นแต่ผู้คุมสอบจะหยิบยืมให้
5. ห้ามนำส่วนใดส่วนหนึ่งของข้อสอบออกจากห้องสอบ ข้อสอบเป็นทรัพย์สินของราชการซึ่งผู้ลักพาอาจมีโทษทางคดีอาญา
6. ผู้ที่ประสงค์จะออกจากห้องสอบก่อนหมดเวลาสอบ แต่ต้องไม่น้อยกว่า 45 นาที
7. เมื่อหมดเวลาสอบ ผู้เข้าสอบต้องหยุดการเขียนใดๆ ทั้งสิ้น
8. ผู้ที่ปฏิบัติเข้าข่ายทุจริตในการสอบ ตามประกาศคณะวิศวกรรมศาสตร์
   มีโทษ คือ ได้รับ สัญลักษณ์ F ในรายวิชาที่ทุจริต และพักการศึกษาอย่างน้อย 1 ภาคการศึกษา

รับทราบ

ลงชื่อนิสิต (.....)

---

1. (5 คะแนน) จงวาด AVL Tree ที่เกิดขึ้นจากการเพิ่มและลบข้อมูลดังต่อไปนี้ทีละขั้น

เพิ่ม 1	เพิ่ม 2	เพิ่ม 3
เพิ่ม 4	เพิ่ม 5	เพิ่ม 6
เพิ่ม 7	ลบ 3	ลบ 5

2. (4 คะแนน) จงเขียนอธิบายสั้นๆว่าทำไมข้อความดังต่อไปนี้ที่เกี่ยวกับ hash table ถึงเป็นสิ่งที่ควรปฏิบัติตาม

   a. ขนาดของ hash table ควรเป็นตัวเลขจำนวนเฉพาะ

   _____

   _____

   b. Load factor ห้ามเกิน 0.5 เมื่อใช้ quadratic probing กับ open addressing hash table

   _____

   _____

ชื่อ_____หมายเลขประจำตัว_____เลขที่ใน CR58_____

3. (5 คะแนน) จงวิเคราะห์เวลาการทำงานของฟังก์ชันข้างล่างนี้ (ในรูปของ Θ ของฟังก์ชัน n)

`a1(int n) {`
`  vector<int> v;`
`  for (int i = 0;i < n;i++)`
`  v.push_back(i);`
`  for (int i = 0;i < n;i++)`
`  v.erase(v.end()-1);`
`}`
	a1(n) ใช้เวลา เป็น $\Theta(\ ?\ )$

`a2(int n) {`
`  s = 0`
`  for (i=1; i<=n; i++) {`
`  for (j=i; j>=0; j--) {`
`  s = i+j`
`  }`
`  }`
`  return s`
`}`
	a2(n) ใช้เวลา เป็น $\Theta(\ ?\ )$

`a3(int n) {`
`  list<int> l;`
`  for (int i = 0;i < n;i++)`
`  l.insert(l.begin(),i);`
`  while (l.size() > 0)`
`  l.erase(l.begin());`
`}`
	a3(n) ใช้เวลา เป็น $\Theta(\ ?\ )$

`a4(int n) {`
`  map<int,int> m;`
`  for (int i = 0;i < n;i++)`
`  m[0] = i;`
`  s = 0;`
`  for (auto &x : m) {`
`  s += x.second;`
`  }`
`}`
	a4(n) ใช้เวลา เป็น $\Theta(\ ?\ )$

`a5(int n) {`
`  if (n > 0) {`
`  a5(n – 2);`
`  a5(n – 2);`
`  }`
`}`
	a5(n) ใช้เวลา เป็น $\Theta(\ ?\ )$

4. (5 คะแนน) ให้ t คือ Binary Tree ต้นหนึ่ง มี 8 ปม เมื่อเราแวะผ่านปมใน t แบบในลำดับ (inorder traversal) ได้ลำดับดังนี้ <D, B, E, H, A, C, G, F> และเมื่อแวะผ่านปมแบบหลังลำดับ (postorder traversal) ใน t ได้ลำดับ <D, H, E, B, G, F, C, A> จงเขียนลำดับของชื่อปมที่ถูกแวะผ่าน เมื่อแวะผ่านปมใน t แบบก่อนลำดับ (preorder traversal)

_____

_____

************** สำคัญ!!! **************

ตั้งแต่ข้อ 4 เป็นต้นไปเป็นการเขียนโปรแกรม การเขียนโปรแกรมจะต้องไม่ผิด syntax ในส่วนที่สำคัญ โดยเฉพาะอย่างยิ่งการเรียก function และ argument ของฟังก์ชันจะต้องถูกต้อง นอกจากนี้คะแนนจะแปรผันตาม "ประสิทธิภาพ" ของโปรแกรมที่เขียนมา

******** ให้สังเกตถึงข้อกำหนดต่าง ๆ ในโจทย์ให้รอบคอบ <u>การไม่ทำตามข้อกำหนดจะมีผลต่อคะแนนเป็นอย่างมาก</u> ********

5. (5 คะแนน) คลาส CP::queue มี implementation ดังแสดงในช่องข้างล่างนี้ จงเขียนฟังก์ชัน void remove_kth(size_t k) เพิ่มเติมให้กับคลาสดังกล่าว ซึ่งเป็นการเอาข้อมูลลำดับที่ k (นับจากข้อมูลตัวแรกที่อยู่ที่หัวคิวเป็นลำดับที่ 0) ออกจากคิว ตัวอย่างเช่น การเรียก remove_kth(0) นั่นจะให้ผลเหมือนกับการเรียก pop() และการเรียก remove_kth(3) จากคิวซึ่งมีข้อมูลเป็น {A,B,C,D,E} (เมื่อ A คือหัวคิว และ E คือข้อมูลตัวสุดท้ายที่พึ่งใส่เข้าไป) คิวของเราจะต้องเปลี่ยนไปเป็น {A,B,C,E} ในกรณีที่ k มีค่ามากกว่าจำนวนข้อมูลในคิว ไม่ต้องทำอะไร

```
template <typename T>
class queue {
  protected:
    T*     mData;
    size_t mCap;
    size_t mSize;
    size_t mFront; // ตำแหน่งบอกว่าข้อมูลแรกอยู่ช่องไหน คิวจะวนทบอาร์เรย์ด้วยนะ

  public:
    void remove_kth(size_t k) {
      // เติม C++ code ตรงนี้









    }
}
```

6. (5 คะแนน) โครงสร้างข้อมูลแบบ Binary Heap ตามที่เรียนในห้องเรียนนั้นใช้อาร์เรย์ในการเก็บต้นไม้ Binary Heap โดยที่ข้อมูลตัวที่มีค่ามากที่สุดอยู่ที่ปมรากของฮีบ (ซึ่งก็คือช่องหมายเลข 0 ใน อาเรย์) จงเขียนฟังก์ชัน bool isAHeap(vector<int> v) ซึ่งจะทำการตรวจสอบว่าข้อมูลใน v นั้นเก็บข้อมูลที่มีลักษณะตรงตามเงื่อนไขของ Binary Heap หรือไม่ (ให้ถือว่าการเปรียบเทียบข้อมูลนั้นสามารถทำได้โดยใช้เครื่องหมาย <, >, == ได้โดยตรง)

```
bool isAHeap(vector<int> v) {









}
```

7. (4 คะแนน) ในข้อนี้ให้นิสิตเขียนฟังก์ชั่น size_t find_position(const KeyT& k) สำหรับคลาสประเภทเดียวกับ CP::unordered_map ที่ใช้หาตำแหน่งใน hash table แบบ double hashing

นิสิตสามารถดูตัวอย่างของ ของ find_position สำหรับการตำแหน่งแบบ linear probing ตามที่ที่นิสิตเรียนในคาบดังต่อไปนี้

```
size_t find_position(const KeyT& key) {
  size_t homePos = hash_to_bucket(key);
  size_t pos = homePos, m = mBuckets.size(), col_count = 0;
  while ( !mBuckets[pos].empty() &&
          !mEqual(mBuckets[pos].value.first, key) ) {
     col_count++;
     pos = (homePos + col_count) % m;
  }
  return pos;
}
```

โดยให้ฟังก์ชั่น g(key) เป็นอีกแฮชฟังก์ชั่นที่ใช้ในการกระโดด

```
int g(const KeyT& key) {
    // g เป็น has function
}
size_t find_position(const KeyT& key) {
  size_t homePos = hash_to_bucket(key);
  size_t pos = homePos, m = mBuckets.size(), col_count = 0;
  while ( !mBuckets[pos].empty() &&
          !mEqual(mBuckets[pos].value.first, key) ) {
               col_count++;
               // เติม C++ code ตรงนี้
  }
  return pos;
}
```

8. (10 คะแนน) ในข้อนี้ให้นิสิตเขียนเพิ่มคำสั่ง eraseEveryOther() เพื่อทำการลบข้อมูลใน circular doubly linked list with header แบบตัวเว้นตัว กล่าวคือ ฟังก์ชันนี้จะลบข้อมูลตัวแรก ไม่ลบข้อมูลตัวที่สอง ลบข้อมูลตัวที่สาม ไม่ลบข้อมูลตัวที่สี่ สลับกันไปจนครบทุกตัว **โดยในข้อนี้ห้ามนิสิตเรียก บริการใดๆของ list และห้ามใช้ iterator โดยเด็ดขาดแต่สามารถเรียกบริการของ node ได้**

```
template <typename T>
class list {
  protected:
    class node {
      protected:
      T     data;  node *prev;    node *next;
      node() : data( T() ), prev(this), next(this) { }
      node(const T& data, node* prev, node* next) : data( T(data) ), prev(prev), next(next) { }
     };

    node   *mHeader; // pointer to a header node
    size_t  mSize;
    // ..............
    void eraseEveryOther() {
      // เติม C++ code ตรงนี้
    }
};
```

9. (10 คะแนน) กำหนดให้มี Binary Search Tree (BST) ที่เก็บข้อมูลประเภท int ที่ไม่ซ้ำกันเลยอยู่ต้นหนึ่ง ปมแต่ละปมของต้นไม้นี้เป็นตัวแปรประเภท node* โดยที่คลาส node นั้นเป็นดังที่เขียนไว้ด้านล่างนี้ ให้สังเกตว่าคลาส node นั้นมี member คือ size ซึ่งระบุจำนวนปมทั้งหมดที่เป็นลูกหลานของปมดังกล่าว (นับปมดังกล่าวด้วย) สมมติว่าปมทั้งหมดในต้นไม้นี้มีค่าที่ถูกต้องตามนิยามของ BST (ซึ่งหมายความว่าค่า size ของแต่ละปมมีการคำนวณให้เป็นค่าที่ถูกต้องเรียบร้อยแล้ว) จงเขียนฟังก์ชัน node* get_kth(node *r, size_t k) ซึ่งจะคืนปมที่มีเก็บค่าลำดับที่ k เมื่อเรียงจากน้อยไปมากของต้นไม้ BST ที่มีปมรากอยู่ที่ r หรือคืนค่า NULL เมื่อไม่มีปมดังกล่าว (ให้ปมที่มีค่าน้อยที่สุดเป็นลำดับที่ 0)

```
class node {
   public:
     int     data;  node *left;    node *right;
     size_t  size; // จำนวนปมทั้งหมดที่อยู่ภายใต้ปมนี้ รวมถึงตัวเองด้วย
};

node* get_kth(node *r, size_t k) {









}
```

10. (10 คะแนน) จงเขียนฟังก์ชั่น int count_leaves() เพื่อเพิ่มบริการให้กับคลาส CP::map_avl เพื่อนับว่าในต้นไม้ AVL ดังกล่าวนั้น มีปมที่เป็นใบอยู่กี่ปม **โดยในข้อนี้ห้ามนิสิตใช้ iterator โดยเด็ดขาดแต่สามารถเรียกบริการอื่น ๆ ของ CP::map_avl ได้และนิสิตสามารถเขียนฟังก์ชันอื่นเพิ่มเติมเข้าไปในคลาสนี้ได้เช่นกัน**

```
template <typename KeyT,
          typename MappedT,
          typename CompareT = std::less<KeyT> >
class map_avl {
protected:
    class node {
       friend class map_avl;
       protected:
            ValueT data; int    height;
            node  *left, *right, *parent;
            //......
    }
    node     *mRoot;      CompareT  mLess;      size_t    mSize;
    // สามารถ เติม function อื่น ๆ เพิ่มเติมได้ตรงนี้









public:
    int count_leaves() {
           // เติม C++ code ตรงนี้









    }
};
```

11. (10 คะแนน) ในข้อนี้เราจะพิจารณาโครงสร้างข้อมูลแบบใหม่ที่มีชื่อว่า Bloom Filter โดย Bloom Filter นั้นเป็นโครงสร้างข้อมูลที่มีบริการหลักอยู่สองบริการคือ void insert(const &T x) ซึ่งทำหน้าที่เอาข้อมูล x ใส่ลงไปในโครงสร้างข้อมูล และ bool exist(const &T y) ซึ่งใช้ตรวจสอบว่ามีข้อมูล y อยู่ในโครงสร้างข้อมูลนี้หรือไม่ Bloom Filter ไม่รองรับการลบข้อมูล และไม่มี iterator

Bloom Filter มีความแปลกประหลาดอยู่อย่างหนึ่งคือบริการ exist นั้นอาจจะคืนค่า true สำหรับข้อมูลที่ไม่เคยถูกใส่ไปในโครงสร้างข้อมูลมาก่อนเลย (กล่าวคือ เมื่อ exists คืนค่า true มันอาจจะไม่ได้แปลว่ามีข้อมูลดังกล่าวอยู่ในโครงสร้างข้อมูลก็เป็นได้) แต่เมื่อ exist คืนค่า false นั้น เรามั่นใจได้แน่นอนว่า ไม่มีข้อมูลดังกล่าวอยู่ในโครงสร้างข้อมูลจริง ๆ

หลักการทำงานของ Bloom Filter คือ มันจะสร้างตารางขนาด K ช่อง แต่ละช่องเก็บค่าประเภท bool เท่านั้น และมี hash function จำนวน M hash function ที่แตกต่างกัน กำหนดให้ $h_i(x)$ คือ hash function ตัวที่ i เมื่อเราใส่ข้อมูล x ลงไปใน Bloom Filter เราจะทำการ mark ช่องหมายเลข $h_1(x), h_2(x), \dots, h_m(x)$ ในตารางให้มีค่าเป็น true และเมื่อเราต้องการตรวจสอบว่า Bloom Filter นี้มีข้อมูล y อยู่หรือไม่ เราก็จะไปตรวจสอบว่าช่อง $h_1(y), h_2(y), \dots, h_m(y)$ ทุกช่องนั้นมีค่าเป็น true ทั้งหมดหรือไม่ ความผิดพลาดของฟังก์ชัน exists นั้นเกิดจากการที่ช่องหมายเลข $h_1(y), \dots, h_m(y)$ นั้นเป็น true จากการ insert ข้อมูล x1 ร่วมกับ x2 ที่ทำให้ช่อง $h_1(x1), \dots, h_m(x1)$ กับช่อง $h_1(x2), \dots, h_m(x2)$ นั้นซ้ำกับช่อง $h_1(y), \dots, h_m(y)$ เป็นต้น

ตัวอย่างต่อไปนี้แสดงตาราง Bloom Filter ที่ค่า K = 9 และ M = 2 ที่มีการใส่ค่า 1 และ 3 เข้าไป โดยที่ hash ฟังก์ชันมีค่าเป็นดังตารางถัดมา จะเห็นว่า exists(5) นั้นจะเป็น true ถึงแม้เราจะไม่เคยใส่ค่า 5 ลงไปก็ตาม (เนื่องจาก $h_1(5) = 7$ และ $h_2(5) = 2$ และช่อง 2 และ 7 นั้นมีค่าเป็นจริง แต่ว่า exists(2) นั้นเป็น false

ช่อง 0	ช่อง 1	ช่อง 2	ช่อง 3	ช่อง 4	ช่อง 5	ช่อง 6	ช่อง 7	ช่อง 8
True	False	True	False	False	False	False	True	False

ค่า x	1	2	3	4	5
$h_1(x)$	2	1	0	1	7
$h_2(x)$	0	6	7	3	2

จงเขียนคลาส BloomFilter ซึ่งเป็นโครงสร้างข้อมูล Bloom Filter ที่ทำงานตามหลักการข้างต้น โดยคลาส Bloom Filter จะต้องมีฟังก์ชันดังต่อไปนี้

- BloomFilter(size_t K, vector<std::hash<T> > hash) เป็น constructor ซึ่งรับค่า K และ hash function จำนวน M function ในรูปแบบ vector ของ std::hash<T>

  สมมติให้ x เป็นตัวแปรประเภท T และให้ h เป็น hash function ในรูปแบบตัวแปรประเภท std::hash<T> การคำนวณค่า hash ของ x ทำได้โดยเรียก h(x) ซึ่งจะคืนค่าเป็นตัวเลขจำนวนเต็มตั้งแต่ 0 ถึง 2^{32}-1
- void insert(const T& data) เป็นฟังก์ชันที่ใส่ข้อมูล data ลงไปใน BloomFilter
- bool exists(const T& value) เป็นฟังก์ชันที่ตรวจสอบว่ามีค่า value อยู่ใน BloomFilter นี้หรือไม่ (โดยที่อาจจะคืนค่า true ที่ผิดมาก็เป็นได้ แต่ไม่เคยคืนค่า false ที่ผิด)

(ย่อหน้านี้ไม่เกี่ยวกับข้อสอบ มันเพียงแค่อธิบายว่า Bloom Filter นั้นมีประโยชน์อย่างไร) หลักการของ Bloom Filter ทำให้เราสามารถใช้ช่องเพียง K ช่องในการเก็บข้อมูลจำนวนกี่ตัวก็ได้ โดยที่ประสิทธิภาพในการทำงานของ insert และ exist นั้นเป็น $\Theta(M)$ อย่างไรก็ตาม ถ้าค่า K และ M นั้นน้อย ๆ มันมีโอกาสที่ exist จะคืนค่า true ที่ไม่ถูกต้องมากยิ่งขึ้น ประโยชน์ของ Bloom Filter คือเราสามารถเก็บข้อมูลจำนวนมาก ๆ ได้โดยที่ใช้ค่า K น้อยกว่าจำนวนข้อมูล โดยการใช้งานหนึ่งของ Bloom Filter คือใช้สำหรับตรวจสอบว่าค่าดังกล่าวอยู่ใน cache ของหรือไม่ โดยใช้ Bloom Filter มาเป็นตัวกรองเบื้องต้นอย่างรวดเร็ว โดยที่ถ้า exist คืนค่า false จะหมายความว่าข้อมูลดังกล่าวไม่อยู่ใน cache และระบบจะต้องไปหาค่านั้นมาจากที่อื่น แต่ถ้า Bloom Filter คืนค่า true ระบบจะทดลองไปหาข้อมูลดังกล่าวจาก cache ก่อน เมื่อไม่เจอข้อมูลจริง ๆ จึงไปหาค่าจากที่อื่น

11.1 จงอธิบาย data member ต่าง ๆ ที่ใช้ในคลาสที่ออกแบบขึ้น ว่ามี member อะไรบ้าง และ แต่ละตัวทำหน้าที่อะไร ทำไมถึงเลือกใช้ประเภทตัวแปรดังกล่าว

11.2 จงเขียนคลาส Bloom Filter ตามข้อกำหนดที่ได้ระบุไว้ข้างต้น ถ้าหากเนื้อที่ไม่พอเขียน ให้เขียนไว้ด้านหลังของ หน้า 7 เท่านั้น

STL Reference

(be noted that the underlined method is not available in CP version)

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>

Element Access	ValueT& operator[] (size_t n); ValueT& at(inti dx);

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);
	void clear();
	void resize(size_t n);

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

Class set<ValueT>

Operation	iterator find (const ValueT& val); size_t 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_t erase (const ValueT& val);

Class map<KeyT, MappedT>

Element Access	MappedT& operator[] (const KeyT& k);
Operation	iterator find (const KeyT& k); size_t 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_t 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 functions

```
iterator find (iterator first, iterator last, const T& val);
void sort (iterator first, iterator last, Compare comp);
pair<T1,T2> make_pair (T1 x, T2 y);
```