FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2110211 Introduction to Data Structures

Year 2nd, First Semester, Final Examination 12 Dec 2019 8:30-11:30

ชื่อ-นามสกุล_____เลขประจำตัว_____ CR61_____

หมายเหตุ

1. ข้อสอบมีทั้งหมด 11 ข้อในกระดาษคำถามคำตอบจำนวน 11 แผ่น 11 หน้า คะแนนเต็ม 103 คะแนน
2. ไม่อนุญาตให้นำตำราและเครื่องคำนวณต่างๆ ใดๆ เข้าห้องสอบ
3. ห้ามการหยิบยืมสิ่งใดๆ ทั้งสิ้น จากผู้สอบอื่นๆ เว้นแต่ผู้คุมสอบจะหยิบยืมให้
4. ห้ามนำส่วนใดส่วนหนึ่งของข้อสอบออกจากห้องสอบ ข้อสอบเป็นทรัพย์สินของราชการซึ่งผู้ลักพาอาจมีโทษทางคดีอาญา
5. ผู้เข้าสอบสามารถออกจากห้องสองได้ หลังจากผ่านการสอบไปแล้ว 45 นาที
6. เมื่อหมดเวลาสอบ ผู้เข้าสอบต้องหยุดการเขียนใดๆ ทั้งสิ้น
7. นิสิตกระทำผิดเกี่ยวกับการสอบ ตามข้อบังคับจุฬาลงกรณ์มหาวิทยาลัย **มีโทษ คือ พ้นสภาพการเป็นนิสิต หรือ ได้รับ สัญลักษณ์ F ในรายวิชาที่กระทำผิด และอาจพิจารณาให้ถอนรายวิชาอื่นทั้งหมดที่ลงทะเบียนไว้ในภาคการศึกษานี้**

ห้ามนิสิตพกโทรศัพท์และอุปกรณ์สื่อสารไว้กับตัวระหว่างสอบ
หากตรวจพบจะถือว่านิสิตกระทำผิดเกี่ยวกับการสอบ อาจต้องพ้นสภาพการเป็นนิสิต

* ร่วมรณรงค์การกระทำผิด หรือทุจริตการสอบเป็นศูนย์ที่คณะวิศวกรรมศาสตร์ *

ข้าพเจ้ายอมรับในข้อกำหนดที่กล่าวมานี้ ข้าพเจ้าเป็นผู้ทำข้อสอบนี้ด้วยตนเองโดยมิได้รับการช่วยเหลือ หรือให้ความช่วยเหลือ ในการทำข้อสอบนี้

ลงชื่อนิสิต.....

วันที่.....

หมายเหตุเพิ่มเติม:

1. หากที่ไม่พอเขียนคำตอบให้เขียนใน**ด้านหลังของข้อดังกล่าว**ได้แต่ต้องระบุไว้ให้ชัดเจน
2. ต้องเขียนตอบด้วยลายมือที่อ่านง่ายและชัดเจน สามารถใช้ดินสอเขียนคำตอบได้
3. ให้เขียนชื่อ-นามสกุล รหัสนิสิต และ หมายเลข CR ให้ครบทุกหน้า
4. ในข้อที่มีการเขียนโปรแกรมนั้น จะมีการพิจารณาประสิทธิภาพในการทำงานของโปรแกรมที่เขียนมาด้วย ให้พยายามเขียนโปรแกรมที่ทำงานได้เร็วที่สุด

ชื่อ-นามสกุล_____เลขประจำตัว_____ CR61_____

1. (10 คะแนน) จงระบุว่าข้อความต่อไปนี้เป็นจริงหรือเท็จ โดยเขียนตัวอักษร T สำหรับข้อที่เป็นจริง และ F สำหรับข้อที่เป็นเท็จ แต่ละข้อมีคะแนน 1 คะแนน หากตอบผิด จะติดลบ 0.5 คะแนนต่อข้อ (คะแนนรวมทั้งข้อนี้ จะไม่ต่ำกว่า 0 คะแนน)

   1.1 ____ CP::list นั้น มี data member ชื่อ mLess เพื่อใช้สำหรับการเปรียบเทียบข้อมูล

   1.2 ____ CP::list เป็น linked list แบบ circular singly linked list with header

   1.3 ____ CP::priority_queue มี method ชื่อ begin() ซึ่งจะคืน iterator ที่ชี้ไปยังข้อมูลตัวเดียวกับ top() ถ้าหากมีข้อมูลนั้นอยู่

   1.4 ____ CP::priority_queue สามารถเก็บข้อมูลที่มีค่าซ้ำกันได้มากกว่า 1 ตัว

   1.5 ____ std::priority_queue<int,vector<int>,std::greater<int>> pq นั้นเป็นการประกาศตัวแปร pq ที่ทำให้ข้อมูลใน pq นั้นเรียงจากมากไปน้อย (กล่าวคือ pq.top() จะคืนค่าข้อมูลที่มีค่ามากสุดใน pq)

   1.6 ____ operator++() ของ CP::map_avl นั้นใช้เวลาเป็น O(n) เมื่อ n คือจำนวนข้อมูลใน map ดังกล่าว

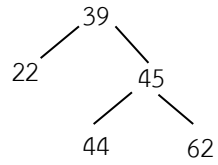
   1.7 ____ operator--() ของ CP::list นั้นใช้เวลาเป็น O(n) เมื่อ n คือจำนวนข้อมูลใน list ดังกล่าว

   1.8 ____ การเก็บข้อมูลจำนวน 1 ตัวใน CP::map_avl<int,int> นั้นมีจำนวนตัวแปรประเภท node* มากกว่า CP::list<int> ที่มีข้อมูล 1 ตัว

   1.9 ____ จำนวนช่องของ mData ใน CP::priority_queue นั้นเท่ากับ mSize เสมอ

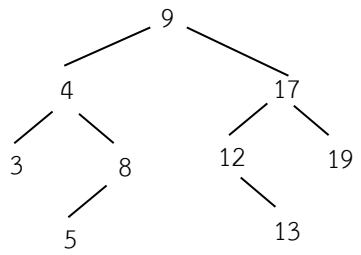
   1.10____ ฟังก์ชัน rotate_left_child ของ CP::map_avl นั้น ใช้เวลาเป็น O(1) เสมอ

2. (10 คะแนน) (a) จงวาด AVL Tree ที่เกิดจากการเพิ่มข้อมูล 90 91 92 93 94 (เริ่มจากต้นไม้ต่อไปนี้) ตามลำดับ



(1) เพิ่ม 90	(2) เพิ่ม 91	(3) เพิ่ม 92	(4) เพิ่ม 93	(5) เพิ่ม 94

(b) เริ่มจากต้นไม้ในช่องซ้ายสุด แล้วลบ 3 4 5 8 9 ตามลำดับ

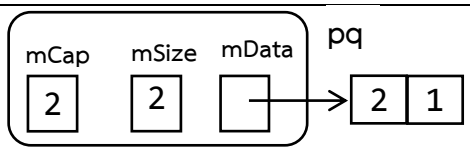
ต้นไม้ AVL ที่มีอยู่	(1) ลบ 3	(2) ลบ 4
		

(3) ลบ 5	(4) ลบ 8	(5) ลบ 9

3. (5 คะแนน) ให้พิจารณาตาราง Hash แบบ open addressing ที่ใช้วิธีการแก้การชนกันแบบ Quadratic Probing โดยใช้ Hash Function เป็น h(x) = x % 13 จงเขียนข้อมูลในตาราง Hash ดังกล่าว ซึ่งเป็นผลของการดำเนินการต่อไปนี้ตามลำดับ (ช่องที่เคยมีข้อมูลให้เขียนเครื่องหมาย X สำหรับช่องที่ไม่มีข้อมูลให้ปล่อยว่างไว้)

การทำงาน	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
(ตัวอย่าง) เพิ่ม 10											10		
(ตัวอย่าง) เพิ่ม 24											10	24	
เพิ่ม 1													
เพิ่ม 11													
ลบ 24													
เพิ่ม 37													
เพิ่ม 23													

4. (5 คะแนน) จงวาดแผนภูมิเพื่อแสดงลักษณะและค่าของ data member ในโครงสร้างข้อมูล หลังจากที่โปรแกรมได้ทำงานจนจบคำสั่งสุดท้ายในแต่ละข้อดังต่อไปนี้ (สำหรับโครงสร้างข้อมูลที่มีการใช้ mLess ไม่จำเป็นต้องเขียน mLess)

5.0	`priority_queue<int> pq;`
	`pq.push(1);`
	`pq.push(2);`
	// ตัวอย่าง
		

5.1	`priority_queue<int> pq;`
	`pq.push(1);`
	`pq.push(2);`
	`pq.push(4);`
	`pq.push(3);`
	`pq.pop();`	

5.2	`priority_queue<int> pq;`
	`pq.push(1); pq.push(2);`
	`pq.push(4); pq.push(3);`
	`pq.pop(); pq.pop();`
	`pq.pop(); pq.pop();`	

5.3	CP::list<int> l;
	l.push_back(5);
	l.push_front(4);
	l.pop_back();	

5.4	CP::list<int> l;
	l.push_back(5);
	l.push_front(4);
	auto it = l.insert(l.end(),3);
	it--;
	l.insert(it,2);	

5.5	CP::map_bst<int,int> m;
	m[1] = 2;
	m[2] = 3;
	m[3] = 4;	

5. (10 คะแนน) ใน CP::priority_queue นั้น การเปรียบเทียบข้อมูลว่าใครมีค่าน้อยกว่ากันจะทำโดยใช้ตัวแปร mLess ซึ่งเป็นประเภท Comp (ซึ่ง Comp เป็น typename ของ template) เราต้องการเปลี่ยนให้ priority_queue ของเรานั้นเรียงข้อมูลใหม่ โดยให้เพิ่มบริการ CP::priority_queue::set_priority(Comp new_mLess) เพื่อเปลี่ยนให้ priority_queue นั้นใช้ new_mLess ในการเรียงข้อมูลและทำการแก้ไขการเรียงข้อมูลภายในใหม่ จงเขียนฟังก์ชันดังกล่าว (ควรจะทำให้ฟังก์ชันนี้ใช้เวลาเป็น O(n))

```
template <typename T,typename Comp = std::less<T> >
class priority_queue {
  protected:
    T *mData;     size_t mCap;  size_t mSize; Comp mLess;
    void fixUp(size_t idx) {…}
    void fixDown(size_t idx) {…}
  public:

    void set_priority(Comp newMLess) {
      // your code here









    }
};
```

6. (10 คะแนน) ในการลบข้อมูลของตารางแฮชแบบที่อยู่เปิด (Open Addressing) โดยใช้การตรวจแบบเชิงเส้น (Linear Probing) ที่เรียนในคาบนั้นมีการเก็บสถานะของช่องเพื่อระบุว่ามีข้อมูลอยู่หรือไม่เคยมีข้อมูลอยู่ หรือเคยมีข้อมูลอยู่แต่โดนลบไปแล้ว แต่จริงๆแล้วเราสามารถที่จะไม่ต้องเก็บสถานะของช่องก็ได้ (แต่การลบข้อมูลจะมีความซับซ้อนขึ้น) ในข้อนี้ให้นิสิต เขียนคำสั่ง erase เพื่อลบข้อมูลออกจากตารางแฮช (พร้อมทั้งคืนค่า 1 ถ้ามีการลบเกิดขึ้น และ คืนค่า 0 ถ้าไม่มีการลบ) ของตารางแฮชแบบที่อยู่เปิดที่ใช้การตรวจแบบเชิงเส้น โดยช่องที่ไม่มีข้อมูลถือว่ามีค่าเป็น T() (และรับประกันว่าไม่มีข้อมูลใดที่มี key เป็น T()) โดยจะต้องพิจารณาช่องให้น้อยช่องที่สุดเท่าที่ทำได้, นิสิตสามารถเพิ่มบริการให้กับ unordered_map ได้แต่ห้ามเพิ่ม member variable

```
template <typename KeyT,  typename MappedT,typename HasherT = std::hash<KeyT>,
          typename EqualT = std::equal_to<KeyT> >
class unordered_map {
  protected:
    typedef pair<KeyT,MappedT> BucketT;
//-------------------------------------------------------------
    std::vector<BucketT> mBuckets;
    size_t               mSize;
    HasherT              mHasher;
    EqualT               mEqual;
    float                mMaxLoadFactor;

  // default constructor
   unordered_map() : mBuckets( std::vector<BucketT>(11) ), mSize(0),
   mHasher( HasherT() ), mEqual( EqualT() ), mMaxLoadFactor(0.7) { }
   size_t bucket_count() { return mBuckets.size();  }
   size_t hash_to_bucket(const KeyT& key) {  return mHasher(key) % mBuckets.size();   }
   size_t find_position(const KeyT& key) {
      size_t homePos = hash_to_bucket(key);
      size_t pos     = homePos;
      while ( (!mBuckets[pos].first == T()) &&
              (!mEqual(mBuckets[pos].first, key)) ) {
          pos = (pos + 1) % mBuckets.size();
      }
      return pos;
  }
     // เติมโค้ดตรงนี้ได้ เผื่อประกาศบริการเพิ่ม









   size_t erase(const KeyT & key) {
     // เติมโค้ดตรงนี้









  }
};
```

7. (10 คะแนน) ในงานทางด้าน Data Science หรือ Data Mining นั้น เรามีความจำเป็นที่จะต้องตรวจสอบความคล้ายคลึงกันของ set จำนวน 2 set (ตัวอย่างเช่น การจะตรวจสอบว่า ผู้ใช้ Netflix สองคนมีความชอบเหมือนกันหรือไม่ เราจะตรวจสอบความคล้ายกันของ set ของภาพยนต์ที่ทั้งสองคนได้ดูจบไปแล้ว เป็นต้น) วิธีการหนึ่งในการตรวจสอบความคล้ายคือการคำนวณ Jaccard Index ของเซ็ต A และ B (เขียนด้วย J(A,B)) ซึ่งเป็นการคำนวณค่าความคล้ายออกมาเป็นจำนวนจริงตั้งแต่ 0 (ไม่เหมือนกันเลย) จนถึง 1 (A และ B เหมือนกัน 100%) โดยมีนิยามคือ $J(A,B) = |A \cap B|/|A \cup B|$ การคำนวณ Jaccard Index นั้นต้องใช้เวลามาก เนื่องจากจำเป็นจะต้องหาผลของ intersect และ Union ก่อน ถ้าหากเรามีผู้ใช้จำนวน n คน ก็จะยิ่งช้ามากยิ่งขึ้น

ทางออกหนึ่งของปัญหานี้คือ การใช้เทคนิคของ Hash Function เข้ามาช่วยในการประมาณค่า Jaccard Index ด้วยวิธีการที่เรียกว่า MinHash ซึ่งอธิบายได้ดังนี้ กำหนดให้มี hash function ที่แตกต่างกันอยู่จำนวน k ฟังก์ชัน ได้แก่ hs[0] ถึง hs[k-1] กล่าวคือ hs[i](x) นั้นจะคำนวณค่า hash ของ x ได้

หากเราต้องการคำนวณ Jaccard Index ของเซ็ต A ใด ๆ เราจะคำนวณ MinHash signature ของเซ็ตดังกล่าว โดยกำหนดให้ MinHash signature ของ A เป็น vector<size_t> ขนาด k ช่อง โดยช่องหมายเลข i ต้องเก็บค่า hs[i](x) โดย x เป็นสมาชิกใน A ที่ทำให้ hs[i](x) มีค่าน้อยสุด หลังจากนั้น เมื่อได้ MinHash signature ของเซ็ต A และ B แล้ว เราสามารถประมาณ J(A,B) ด้วยการคำนวณจำนวนสมาชิกใน MinHash signature ของ A ที่ปรากฏอยู่ใน MinHash signature ของ B แล้วนำจำนวนดังกล่าวหารด้วย k

จงเขียนฟังก์ชัน vector<size_t> min_hash_signature(set<int> A, vector<HasherT> hs) ที่คำนวณ MinHash signature ของเซ็ต A โดยใช้ hash function ใน hs โดยเราสามารถเรียกใช้ hash function ต่าง ๆ ใน hs ได้โดยการเรียก hs[i](x) เมื่อ x เป็น int และ hs[i](x) จะคืนค่าเป็นประเภท size_t และจงเขียนฟังก์ชัน float min_hash_estimate(vector<size_t> as, vector<size_t> bs) ซึ่งคำนวณการประมาณค่า Jaccard Index ด้วยวิธี MinHash เมื่อ as และ bs คือ MinHash signature ของเซ็ต A และ B

```
vector<size_t> min_hash_signature(set<int> A, vector<HasherT> hs) {









}
float min_hash_estimate(vector<size_t> as, vector<size_t> bs) {









}
```

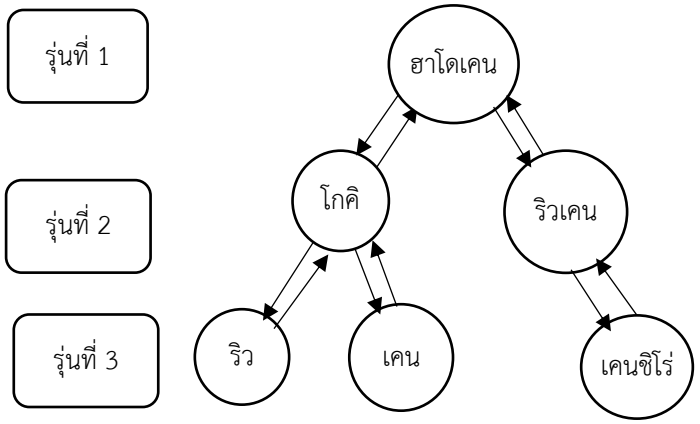
8. (10 คะแนน) ในข้อนี้ให้นิสิตเพิ่มบริการของคลาส CP::list คือ void append_k_list(vector<list<T>> &Ls) ที่รับ list<T> มา k อัน (โดยที่ k = Ls.size()) แล้วนำเอาข้อมูลทั้งหมดที่เกิดจากการนำรายการ Ls[0], Ls[1], Ls[2], ... ,Ls[k-1] มาต่อกันตามลำดับ มาต่อท้ายข้อมูลของ list ที่เรียก โดยหลังจากการเรียกบริการนี้แล้ว Ls[0], Ls[1], Ls[2], ..., Ls[k-1] จะเป็นรายการว่าง ***** โดยบริการนี้จะต้องใช้เวลา O(k) และ ให้นิสิตทำการแก้ไข member variable ของ list หรือ node โดยตรงเท่านั้น (ห้ามนิสิตเรียกบริการใดๆของ list<T> หรือ node แต่เรียกบริการของ vector<list<T>> ได้) *****

ตัวอย่างเช่น	**หลังเรียก `L.append_k_list(Ls)` จะได้ผลเป็น**

`L = <2, 1, 7, 3>`
`Ls[0] = <4, 3, 6>`
`Ls[1] = <9, 0>`
	`L = <2, 1, 7, 3, 4, 3, 6, 9, 0>`
	`Ls[0] = <>`
	`Ls[1] = <>`

```
template <typename T>
class list {
  protected:
    class node {
      friend class list;
      protected:
        T     data; node *prev; node *next;
    }
    node   *mHeader; // pointer to a header node
    size_t  mSize;
  public:
    void append_k_list(vector<list<T>> &Ls) {
        // เติมโค๊ดตรงนี้
   }
}
```

9. (10 คะแนน) ผู้สืบทอดวิทยายุทธสำนักอุดรเทวะ สามารถรับลูกศิษย์เป็นผู้สืบทอดต่อได้อย่างมากที่สุดสองคน ปัจจุบันมีแผนผังผู้สืบทอดดังรูปด้านขวานี้



ในอนาคตอาจมีผู้สืบทอดอีกหลายรุ่น ดังนั้นเพื่อให้สามารถหาได้ง่ายว่าใครอยู่รุ่นไหน เคน ผู้ซึ่งเป็นคนดูแลระบบคอมพิวเตอร์ของสำนัก จึงขอให้คุณเขียนโค้ดที่พิมพ์รายชื่อของรุ่นที่กำหนดออกมาให้ได้ โดยเคนให้โครงสร้างข้อมูลคุณมาดังด้านล่างนี้

จงเขียนโค้ด ฟังก์ชั่น void printIthLevel(int k) ซึ่งพิมพ์ผู้สืบทอดรุ่นที่ k ออกมาทั้งหมด อนุญาตให้ใช้ stl ใดๆ ช่วยก็ได้ และนิสิตสามารถเขียนฟังก์ชันอื่น ๆ เพิ่มเติมขึ้นมาได้ (แนะนำว่า ควรใช้คิวสองคิว หรือ พยายามเขียนโปรแกรมแบบ recursive) จากแผนผังข้างบน การเรียกฟังก์ชันprintKThLevel(3) ฟังก์ชั่นต้องพิมพ์ ริว เคน เคนชิโร่ ออกมา (ไม่จำเป็นต้องเรียงลำดับตามนี้ แต่ต้องมีสามคนนี้เท่านั้น)

```
class north_star_tree {
  class node {
    public:
      string name;
      node *left, *right, *parent;
      node(string s, node *left, node *right, node *parent) :
        name(s), left(left), right(right), parent(parent){}
  };
  node *root;
  tree(string s) {
    root* = new node(s,NULL,NULL,NULL);
```

```
  }
  // นิสิตสามารถเขียนฟังก์ชันอื่น ๆ เพิ่มเติมได้
  void printKThLevel(int k) {
    // your code here









  }
};
```

10. (8 คะแนน) จงเพิ่มบริการให้กับ CP::map_bst โดยให้เขียนฟังก์ชัน iterator find_middle_value(iterator a, interator b) ซึ่งจะต้องคืนค่า iterator ที่ระบุตำแหน่งของข้อมูลที่มีค่าคีย์อยู่ตรงกลางระหว่างค่าคีย์ที่ระบุด้วย a กับ b (รับประกันว่า a กับ b เป็น iterator ที่ระบุถึงข้อมูลที่อยู่ในต้นไม้ของเราแน่ๆ และคีย์ของ a จะอยู่ก่อนคีย์ของ b) กำหนดให้ค่ากลางระหว่างคีย์ k1 และ คีย์ k2 ของ map_bst ใด ๆ คือ ค่าที่อยู่ในลำดับที่ (p+q)/2 ปัดเศษทิ้ง เมื่อ p และ q คือ ลำดับของคีย์ k1 และ k2 เมื่อเรียงข้อมูลในคีย์ทั้งหมดของ map_bst ตามลำดับ **<u>ตัวอย่างเช่น</u>** ถ้า map_bst มีเลข 10,20,30,40,50,60,70 เป็นค่าของ data.first ในแต่ละโหนด (ลักษณะของต้นไม้ไม่เกี่ยว ที่เกี่ยวคือค่าจากน้อยไปมาก)

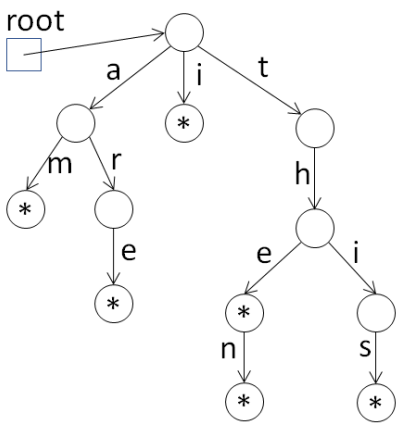
- ค่ากลางระหว่าง ตำแหน่งที่มี 30 เป็นคีย์กับตำแหน่งที่มี 70 เป็นคีย์คือตำแหน่งที่มี 50 เป็นคีย์
- ค่ากลางระหว่าง ตำแหน่งที่มี 30 เป็นคีย์กับตำแหน่งที่มี 60 เป็นคีย์คือตำแหน่งที่มี 40 เป็นคีย์

```
template <typename KeyT, typename MappedT, typename CompareT = std::less<KeyT> >
class map_bst {
  protected:
    typedef std::pair<KeyT,MappedT> ValueT;
    class node {
      friend class map_bst;
      protected:
        ValueT data; node  *left; node  *right; node  *parent;
    };
    class tree_iterator {
      protected:
        node* ptr;
      public:
        tree_iterator() : ptr( NULL ) { }
        tree_iterator(node *a) : ptr(a) { }
        tree_iterator& operator++() { …}
        tree_iterator& operator--() {…}
        tree_iterator operator++(int) {…}
        tree_iterator operator--(int) {…}
        ValueT& operator*()  { return ptr->data;    }
        ValueT* operator->() { return &(ptr->data); }
        bool    operator==(const tree_iterator& other) { return other.ptr == ptr; }
        bool    operator!=(const tree_iterator& other) { return other.ptr != ptr; }
    };
    node     *mRoot;    CompareT  mLess;    size_t    mSize;
  public:
    typedef tree_iterator iterator;
```

```
    iterator find_middle_value(iterator a, iterator b) {









    }
};
```

11. (15 คะแนน) ในข้อนี้ เราจะพิจารณาโครงสร้างข้อมูลแบบใหม่ ซึ่งมีชื่อว่า trie โดย trie นั้นสามารถเพิ่ม, ลบ และ หา ข้อมูลประเภท string ได้โดยใช้เวลาเป็น O(l) เมื่อ l คือความยาวของ string ดังกล่าวโดยไม่ขึ้นอยู่กับจำนวนข้อมูลที่มีอยู่ใน trie นิสิตจะต้องเขียนโครงสร้างข้อมูลดังกล่าวนี้ ดังที่จะอธิบายต่อไป เพื่อความง่าย กำหนดให้ string ที่จะเก็บใน trie นั้นประกอบด้วยตัวอักษรภาษาอังกฤษตัวพิมพ์เล็กเท่านั้น

```
class trie_node {
  public:
    bool valid;
    trie_node* child[25];
    //child[0] is child "a",
    //child[1] is child "b",....,
    //child[25] is child "z"
    trie_node() : valid(false) {
      for (int i = 0;i<26;i++)
        child[i] = NULL;
    }
};
```



โครงสร้างข้อมูลประเภท trie นั้นเป็นต้นไม้ประเภทหนึ่ง โดยที่ปมแต่ละปมนั้นเก็บข้อมูล valid เป็นประเภท boolean และแต่ละปม มีลูกได้มากถึง 26 ปม ปมลูกแต่ละปมนั้นจะถูกเรียกว่า ลูก a, ลูก b,..., ลูก z สำหรับปมใด ๆ ก็ตามมันอาจจะไม่มีลูกครบทั้ง 26 ลูกก็เป็นได้ เช่น ปมรากอาจจะมีปมลูกเพียงแค่ ลูก a, ลูก i และ ลูก t เป็นต้น โค้ดด้านบนแสดงถึงคลาสของปมดังกล่าว การตรวจสอบว่ามี string s อยู่ใน ต้นไม้ของเราหรือไม่ จะเริ่มต้นที่ปม ราก หลังจากนั้นจะเดินทางไปยังปมลูกที่ชื่อ s[0] และเดินทางไปยังปมลูกที่มีชื่อว่า s[1] ไปเรื่อย ๆ จนถึง s[s.length()-1] ถ้าหากเราสามารถเดินทางไปตามปมดังกล่าวได้ตามที่ระบุไว้จนครบทั้งหมด และปมสุดท้ายมีค่า valid เป็น true ก็จะแสดงว่า trie นั้นมี string s อยู่ แต่ถ้าหากเดินไม่ได้ หรือหากปมสุดท้ายมีค่า valid เป็น false ก็จะถือว่าไม่มีปมนั้น รูปด้านซ้ายนี้แสดงถึง trie ที่เก็บคำดังต่อไปนี้ไว้ “the”, “then”, “this”, “are”, “am”, “i” โดยที่เครื่องหมาย * ในปมต่าง ๆ หมายถึงปมนั้นมีค่า valid เป็น true ให้สังเกตว่าต้นไม้นี้ไม่มีคำว่า “a” หรือคำว่า “th” เป็นต้น

Trie นั้นเริ่มต้นด้วยปมรากที่มีค่า valid เป็น false และปมรากดังกล่าวไม่มีปมลูกใด ๆ เลย การเพิ่มข้อมูล string s เข้าไปใน trie จะกระทำโดยทดลองเดินไปตามปมต่าง ๆ เหมือนกับการหาว่ามี s อยู่ใน trie หรือไม่ ถ้าหากเดินไม่ได้ ณ จุดใดก็ตาม ก็จะทำการสร้างปมลูกใหม่ตามทางที่ควรจะไป การลบคำออกจาก trie ทำโดยเดินไปตามปมต่าง ๆ เหมือนกับการหาว่ามี s อยู่ ถ้าหากการเดินสำเร็จ ก็จะเปลี่ยนค่า valid ในปมสุดท้ายให้เป็น false จงเขียนโครงสร้างข้อมูล Trie ดังกล่าว ซึ่งต้องมีบริการอย่างน้อยดังต่อไปนี้

- trie() เป็น constructor ซึ่งสร้าง trie ว่างที่ไม่มีคำใดอยู่เลย
- ~trie() เป็น destructor ซึ่งต้อง delete หน่วยความจำที่จองไว้ทั้งหมด
- void insert(string s) เพิ่มคำ s เข้าไปใน trie ถ้าหากมีคำ s ดังกล่าวอยู่แล้ว คำสั่งนี้จะไม่มีผลอะไร
- bool find(string s) ค้นหาว่าใน trie เก็บคำ s อยู่หรือเปล่า โดยให้คืนค่า true ก็ต่อเมื่อมีคำ s อยู่ใน trie นี้เท่านั้น
- void erase(string s) ลบคำ s ออกจาก trie (ให้ระวังว่า s อาจจะไม่มีอยู่ใน trie ก็เป็นได้)

เพื่อความสะดวก ให้ถือว่ามีฟังก์ชัน to_i(char c) ซึ่งสามารถแปลงอักขระใน string ให้เป็นตัวเลขตามลำดับตัวอักษรได้ เช่น to_i(‘a’) จะคืนค่า 0, to_i(‘e’) จะคืนค่า 4 หรือถ้าให้ s = “somchai" การเรียก to_i(s[2]) จะได้ค่า 11 (เนื่องจาก s[2] คือ ‘m’)

STL Reference

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> **และ** list<ValueT>

Element Access สำหรับ vector	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);
```