CS106B
Fall 2016

Chris and Chris
Dec 12th, 2016

CS106B Final Exam Answer Booklet

Please put **all** your answers in the space provided in this booklet. **No answers outside of these pages will be accepted**.

Last Name: _____**KEY**_____

First Name: _____

Sunet ID (eg jdoe): _____

Section Leader: _____

I accept the letter and spirit of the honor code. I've neither given nor received aid on this exam. I pledge to write more neatly than I ever have in my entire life.

(signed) _____

		Score
1. Short Answer	[10]	_____
2. Tries	[14]	_____
3. Lyft Line	[15]	_____
4. Code Search Engine	[18]	_____
5. Epidemic	[15]	_____
Total	**[xx]**	**_____**

Question 1: Short Answer (10 points)

A. (2 points) Best Case: Big O(**n log n**) Wost Case: Big O(**n^2**)

B. (2 points) Big O(**n**)

C. (2 points) **A Map<> keeps the elements sorted by key.**

D. (1 point) **pre-order** / **in-order** / **post-order** / **level-order** (circle the correct answer).

E. (3 points)

```
Node *concatenate(Node *listA, Node* listB) {
     // Concatenate listB onto the end of listA: (listA->listB)
     if (listA == NULL) {
         return listB;
     }
     // traverse to end of A
     Node *curr = listA;
     while (curr->next != NULL) {
         curr = curr->next;
     }
     // now curr is the tail of listA
     curr->next = listB;
     return listA;
}
```

Question 2: Tries (14 points)

```
void Trie::insert(string word) {
    if (word.size() == 0) {
        isWord = true;
    } else {
        char firstChar = word[0];
        int index = firstChar - 'a';
        Trie *child;
        if (children[index] == NULL) { // does not exist yet
            child = new Trie();
            children[index] = child;
        } else {
            child = children[index];
        }
        child->insert(word.substr(1)); // insert rest of word
    }
}
```

Alternate solution (for loops):

```
void Trie::insert(string word) {
    Trie *curr = this;
    int wordLen = word.size();
    for (int i=0; i < wordLen; i++) {
        char firstChar = word[i];
        int index = firstChar - 'a';
        Trie *child;
        if (curr->children[index] == NULL) { // does not exist yet
            child = new Trie();
            curr->children[index] = child;
        } else {
            child = curr->children[index];
        }
        curr = child;
    }
    curr->isWord = true;

}
```

```
bool Trie::containsPrefix(string prefix) {
    if (prefix.size() == 0) {
        return true;
    }

    // traverse Trie until you find a true isWord marker or NULL
    char firstChar = prefix[0];
    int index = firstChar - 'a';
    Trie *child = children[index];
    if (child == NULL) {
        return false;
    }
    return child->containsPrefix(prefix.substr(1));
}
```

Alternate solution (for loops):

```
bool Trie::containsPrefix(string prefix) {
    Trie *curr = this;
    int wordLen = prefix.size();
    for (int i=0; i < wordLen; i++) {
        // traverse Trie until you find a true isWord marker or NULL
        char nextChar = prefix[i];
        int index = nextChar - 'a';
        Trie *child = curr->children[index];
        if (child == NULL) {
            return false;
        }
        curr = child;
    }
    return true;
}
```

```
bool Trie::contains(string word) {
    if (word.size() == 0) {
        return isWord;
    }

    // traverse Trie until you find a true isWord marker or NULL
    char firstChar = word[0];
    int index = firstChar - 'a';
    Trie *child = children[index];
    if (child == NULL) {
        return false;
    }
    return child->contains(word.substr(1));
}

Alternate solution (for loops):
bool Trie::contains(string word) {
    Trie *curr = this;
    int wordLen = word.size();
    for (int i = 0; i < wordLen; i++) {
        // traverse Trie until you find a true isWord marker or NULL
        char nextChar = word[i];
        int index = nextChar - 'a';
        Trie *child = curr->children[index];
        if (child == NULL) {
            return false;
        }
        curr = child;
    }
    return curr->isWord;
}
```

Question 2, Part B: Big O of `Trie::contains(string word)` function? (2 points)

Big O(**m**)

3. Lyft Line (15 points)

```
double lyftLine(RoadGraph & g, Set<Ride> & rides, Vertex * driverStart) {

	Set<string> inCar;
	return carpool(g, rides, driverStart, inCar)

}

double rideShare(Graph & g, Set<Ride> &rides, Vertex * curr, Set<string> &inCar) {

	if(rides.isEmpty()) return 0;

	double minCost = -1;
	for(Ride r : rides) {
		// end a ride
		if(inCar.contains(r.rideId)) {
			Set<Ride> remaining = rides;
			remaining.remove(r);

			inCar.remove(r.rideId);
			double cost = rideShare(g, remaining, r.end, inCar);
			inCar.add(r.rideId);

			// factor in the cost to get there
			double totalCost = cost + pathCost(curr, r.end);
			minCost = update(totalCost);
		}

		// start a ride
		else if(inCar.size() < TOTAL_SEATS) {
			inCar.add(r.rideId);
			double cost = rideShare(g, rides, r.start, inCar);
			inCar.remove(r.rideId);

			// factor in the cost to get there
			double totalCost = cost + pathCost(curr, r.start);
			minCost = update(totalCost);
		}
	}
}
```

4. Code Search Engine (18 points)

4a) Tree equality.

```
bool treeEqual(Tree * a, Tree * b) {
     if(a == NULL) return b == NULL;
     if(b == NULL) return a == NULL;
     if(a->name != b->name) return false;
     if(!treeEqual(a->left, b->left)) return false;
     return treeEqual(a->right, b->right);
}
```

4b, TreeHash

```
int treeHash(Tree * tree) {
     int index = 0;
     return hash(t, index);
}

int hash(Tree * t, int & index) {
     int h = stringHash(t->value) * pow(PRIME, index);
     for(Tree * child : children) {
          h += hash(child, index++);
     }
     return h;
}
```

4c. Fast Equivalence Search:

```
void pairwiseMatch(Vector<Tree*> trees) {
     Map<int, Vector<Tree*>> hashes;
     for(Tree * t : trees) {
          hashes[hashTree(t)].add(t);
     }
     for(int key : hashes) {
          for(Tree * a : hashes[key]) {
               for(Tree * b : hashes[key]) {
                    if(treeEquals(a, b)) {
                         cout << a << ", " << b << endl;
                    }
               }
          }
     }
}
```

5. Modelling Infectious Disease

```
Set<string> modelDisease(Person * patientZero, double maxDays) {
     Path first;
     first.add(patientZero)
     PriorityQueue<Path> todo;
     todo.add(first);
     Set<string> seen;

     while(!todo.isEmpty()) {
          double cost = todo.peekPriority();
          Path currPath = todo.dequeue();
          Vertex * curr = currPath[currPath.size() - 1];
          if(cost > maxTime) break;
          if(seen.contains(curr->name)) continue;
          seen.add(curr->name);

          for(Edge * edge : curr->outgoing)) {
               double newCost = cost + edge->cost;
               Vertex * neighbor = edge->end;
               Path newPath = currPath;
               newPath.add(neighbor);
               todo.add(newPath, newCost);
          }
     }

     return seen;
}
```

Scratch work only

Scratch work only

Scratch work only

Scratch work only