CSCI-1200 Data Structures
Test 3 — Practice Problems

Note: This packet contains selected practice problems from Test 3 from three previous years. Your test will contain approximately one third as many problems (totalling ~100 pts).

1 Intensely Overloading on TA Office Hours [ / 41 ]

Cameron (who helped develop YACS) is thinking about an extension to assist students in maximizing use of TA office hours for their registered classes. He has access to the following tables of data from the registrar using STL maps and vectors (shown here with sample data):

<table>
<thead>
<tr>
<th>ta_assignments</th>
<th>office_hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abby</td>
<td>Monday 3pm, AE Thursday 2pm, Folsom</td>
</tr>
<tr>
<td>Eric</td>
<td>Tuesday 5pm, J-Rowl</td>
</tr>
<tr>
<td>Milo</td>
<td>Thursday 4pm, Folsom Wednesday 6pm, Folsom</td>
</tr>
<tr>
<td>Ryan</td>
<td>Thursday 4pm, Folsom</td>
</tr>
<tr>
<td>Sinclair</td>
<td>Thursday 4pm, Folsom</td>
</tr>
<tr>
<td>Tim</td>
<td>Thursday 4pm, Folsom</td>
</tr>
<tr>
<td>Wendy</td>
<td>Thursday 4pm, Folsom</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>students</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben</td>
</tr>
<tr>
<td>Louis</td>
</tr>
</tbody>
</table>

Cameron proposes we write a function GetHelp that takes in these 3 tables of data, and the name of a student, and returns an STL set of strings with the day, time, and location of all relevant office hours. For example, by printing the output of:

GetHelp(ta_assignments,students,office_hours,"Ben");

We will learn that Ben should attend the following office hours:

Thursday 4pm, Folsom
Tuesday 5pm, J-Rowl
Wednesday 6pm, Folsom

1.1 Just the typedefs, Ma’am [ / 3 ]

After looking ahead through the rest of this problem, let’s define a few helpful typedefs:

typedef

type_t;

typedef
type_s;

typedef
type_o;
1.2 Let’s get Ben some help in Office Hours!

Write the `GetHelp` function below. Perform some basic error checking. We suggest you use `assert` to verify any assumptions as you work through the implementation.

```cpp
std::set<std::string> GetHelp(const type_t &ta_assignments,
                               const type_s &students,
                               const type_o &office_hours,
                               const std::string &student) {
    // sample solution: 18 line(s) of code
}
```
Milo notices someone forgot their laptop charger in office hours. In addition to posting a message on the Submitty Discussion Forum, Milo would like to message students in the other courses that share his assigned office hour times and locations. Let’s write another function, `GetShared`, that returns an STL `set` of the names of other courses that share an office hour time and location with the specified TA. For example, if we print the return value of:

```cpp
GetShared(ta_assignments, office_hours, "Milo");
```

We will learn that Milo’s Data Structures office hours overlap with these other courses:

```
   bio
csi
```

```cpp
def GetShared(ta_assignments, office_hours, ta) {
    // sample solution: 19 lines of code
}
```
1.4 Who’s Afraid of the Big Bad ’O’ Notation? [ 7 ]

What is the running time for each of the functions you wrote? Let’s say that we have \( s \) students, each student takes on average \( k \) courses, the university has \( t \) TAs, each TA is assigned on average to \( h \) office hour slots per week, the university has \( c \) total courses, and the output contains \( x \) values. Justify your answers.

GetHelp

GetShared

1.5 Making some Mappy Improvements [ 3 ]

Is this the best organization of data to accomplish these tasks? Describe a minor change to one of these tables (still sticking with STL maps and vectors) that will simplify the algorithm(s) you wrote above and/or improve the big ’O’ notation. Write 2-3 sentences describing the change and why it’s an improvement.

1.6 Are Hash Tables always better than Binary Search Trees? [ 3 ]

If we switched the maps and sets in this problem to be hash tables, what would be one advantage/improvement? What would be one disadvantage/loss? Write 2 concise, technical, and well-written sentences.
2 I am the Lorax, I speak for the Trees 

In this problem we will inspect the structure of a memory diagram composed of Node objects, as declared on the right. We will decide if the current arrangement, contents, and coloring of nodes is a valid, well-balanced, binary search tree. Note: “red” nodes are visualized white. On the next two pages you will complete the implementation of the recursive functions used in the fragment of code below:

```cpp
template <class T> class Node {
public:
    T value;
    bool is_black;
    std::vector<Node*> children;
};
```

```cpp
if (!is_tree(root)) std::cout << "NOT A TREE"
else if (!is_binary(root)) std::cout << "NOT A BINARY TREE"
else if (!is_bst(root)) std::cout << "NOT A BINARY SEARCH TREE"
else if (!is_red_black(root)) std::cout << "NOT A RED BLACK TREE"
else std::cout << "you're a well-balanced binary search tree!"
```

2.1 Diagrams

For each of the 4 diagrams below, write the statement that will print when the diagram is passed to the code fragment above. *Hint: Each of the four "NOT A ..." phrases will be used exactly once.*

![Diagram 1](image1)

![Diagram 2](image2)

![Diagram 3](image3)

![Diagram 4](image4)

Now, draw a diagram that contains the integers 1-9 that will print the statement “you’re a well-balanced binary search tree!”

![Diagram 5](image5)
2.2 \textbf{is\_tree Implementation} [/ 9 ]

\begin{verbatim}
template <class T> bool is_tree(Node<T> *n) {
    std::set<Node<T>*> already_seen;
    return is_tree(n,already_seen);
}
\end{verbatim}

2.3 \textbf{is\_binary Implementation} [/ 6 ]

\begin{verbatim}
template <class T> bool is_binary(Node<T> *n) {
}
\end{verbatim}
2.4 is_bst Implementation [ 6 ]

```cpp
template <class T>
bool is_bst(Node<T> *n, Node<T> *lower=NULL, Node<T> *upper=NULL) {
    sample solution: 6 line(s) of code
}
```

2.5 is_red_black Implementation [ 7 ]

```cpp
template <class T> bool double_red(Node<T> *n) {
    sample solution: 6 line(s) of code
}

template <class T> int black_count(Node<T> *n) {
    sample solution: 6 line(s) of code
}

template <class T> bool is_red_black(Node<T> *n) {
    if (double_red(n)) return false;
    if (black_count(n) == -1) return false;
    return true;
}
```
Write a function named `construct_breadth` that takes in an STL list of STL strings and creates and returns a pointer to a well-balanced binary tree of Nodes with breadth-first traversal order that matches the input.

```cpp
class Node {
public:
    Node(const std::string &v)
    { value=v; left=right=NULL; }
    std::string value;
    Node* left;
    Node* right;
};
```

Sample solution: 20 line(s) of code

Give an input list of 10 RPI building names that when passed to `construct_breadth` will create a valid binary search tree. Also, draw the produced tree.
Louis B. Reasoner has started a new job in the RPI housing office and is tasked with re-writing their dorm assignment software. An example of the input text file is shown on the right. He has already written code to parse the input file and store the data:

```cpp
4 Dorm Assignment Maps [ / 45 ]

Crockett 320 Beth Collins
Barton 201 Sally Morris
Barton 201 Alice Williams
Barton 201 Jessica Smith
Nugent 316 George Davis
Nugent 316 Fred Harrison
Crockett 106 Chris Thompson
Nugent 112 Erin Lee
Nugent 112 Kathy Newton

type_b dorms;
type_c people;
istr("dorm_data.txt");
dorm, first, last;
int room;
while (istr >> dorm >> room >> first >> last) {
    dorms[dorm][room].push_back(std::make_pair(first,last));
    people[std::make_pair(last,first)] = std::make_pair(dorm,room);
}

He wrote a helper function to email students with their room assignment and roommate information:

```cpp
void Info(/*?*/ dorms, /*?*/ people, /*?*/ first, /*?*/ last) {
    std::string dorm;
    int room;
    LookupRoomAssignment(people,first,last,dorm,room);
    if (dorm == ") { 
        std::cout << first"" << last"" does not have a room assignment.""<<std::endl;
    } else {
        std::cout<<first"" << last"" is assigned to ""<<dorm"" ""<<room"".""<<std::endl;
        PrintRoommates(dorms,first,last,dorm,room);
    }
}
```

And here is sample usage of his code:

```cpp
Info(dorms.people,"Fred","Harrison");
Info(dorms.people,"Sally","Harrison");
Info(dorms.people,"Sally","Morris");
Info(dorms.people,"Chris","Thompson");
std::cout << "Barton has " << NumRoomsOccupied(dorms,"Barton",3)
<< " triple(s)." << std::endl;
std::cout << "Crockett has " << NumRoomsOccupied(dorms,"Crockett",1)
<< " single(s)." << std::endl;
```

Resulting in this output:

Fred Harrison is assigned to Nugent 316.
Fred's roommate(s) are: George
Sally Harrison does not have a room assignment.
Sally Morris is assigned to Barton 201.
Sally's roommate(s) are: Alice Jessica
Chris Thompson is assigned to Crockett 106.
Chris has no roommates.
Barton has 1 triple(s).
Crockett has 2 single(s).

You will write a few functions to help Louis finish the implementation and then analyze the running time.
4.1 The typedefs

First, let’s fill in these typedef declarations. Note: type_a is optional, but it may help simplify your code.

typedef type_a;

typedef type_b;

typedef type_c;

4.2 Data Structure Sketch

Sketch the contents of the dorms variable for the sample input text file. Follow the conventions from lecture for this diagram.
4.3 Implementation of NumRoomsWithOccupancy

sample solution: 12 line(s) of code

4.4 Implementation of LookupRoomAssignment

sample solution: 10 line(s) of code
4.5 Implementation of PrintRoommates

sample solution: 17 line(s) of code

4.6 Order Notation

If the housing system contains $s$ students and $d$ dorms, with $r$ rooms per dorm, and an average/max of $k$ students per room, what is the running time for each of the functions above?

- NumRoomsWithOccupancy
- LookupRoomAssignment
- PrintRoommates
Alyssa P. Hacker has augmented our binary search tree with an additional member variable, the height of each node. She argues that this small change will allow erase to help minimize the overall tree height. She has defined a leaf node to have height = 1. Fill in the missing pieces to finish the implementation of erase.

```cpp
class Node {
public:
    int value;
    int height;
    Node* left;
    Node* right;
};

bool erase(Node* &n, int v) {
    bool answer = true;
    if (n == NULL) { return false; }
    if (v < n->value) { answer = erase(n->left, v); }
    else if (v > n->value) { answer = erase(n->right, v); }
    else {
        if (n->right == NULL && n->left == NULL) { delete n; n = NULL; }
        else if (n->right == NULL) { Node* tmp = n->left; delete n; n = tmp; }
        else if (n->left == NULL) { Node* tmp = n->right; delete n; n = tmp; }
        else { // handle the case of 2 children
            // sample solution: 11 line(s) of code
        }
    }
    if (n != NULL) { // update the height
        // sample solution: 5 line(s) of code
    }
    return answer;
}
```


Draw an exactly balanced binary search tree that contains the characters: b, e, r, n, a, d, and i.

In order for a binary tree to be exactly balanced, what must be true about \( n \), the number of elements?

Using the simple binary search tree insertion algorithm from lecture, give two different orderings for insertion of these elements that will result in an exactly balanced binary search tree.

Suppose we have an STL set named data, whose size allows construction of an exactly balanced tree (as you specified earlier). Let’s write a recursive function named print_balanced_order that will print to std::cout an ordering of these elements for insertion to make an exactly balanced binary search tree.

print_balanced_order(data.begin(), data.end());
Upside-Down Binary Search Tree

Ben Bitdiddle has come up with another wacky tree scheme (with questionable usefulness). He proposes to represent a binary search tree not with a single pointer to the tree root, but instead with an STL list of the leaf nodes. And then it follows that each `Node` will store only a pointer to its parent.

```cpp
class Node {
public:
  Node(int v) : value(v), parent(NULL) {}
  int value;
  Node* parent;
};
```

Ben is sure that because his new representation only has one pointer per `Node` this structure will be much more memory efficient than the typical binary tree. Here’s how he proposes to construct the tree you drew above:

```cpp
std::list<Node*> leaves;
Insert(leaves,13); Insert(leaves,8); Insert(leaves,20); Insert(leaves,4); Insert(leaves,18); Insert(leaves,25); Insert(leaves,2); Insert(leaves,16); Insert(leaves,21); Insert(leaves,6); Insert(leaves,23);
assert (leaves.size() == 5);
```

BelongsInSubtree

Rather than jumping straight into the implementation of the `Insert` function, Alyssa P. Hacker suggests that Ben start by implementing the `BelongsInSubtree` function. This recursive function takes in two arguments: `node`, a pointer to a `Node` already in the upside down tree, and `value`, an element we would like to add. The function returns false if placing `value` within a subtree of `node` violates the binary search tree property of the whole tree and true otherwise. Note: Ben’s tree does not allow duplicate elements.
bool BelongsInSubtree(Node* node, int value) {

}  

The implementation of Insert will call the BelongsInSubtree function on each Node in the tree. Note: This function will return true for at least one, but possibly many nodes in the tree! Of these possible choices, Insert will select the node that is furthest (in number of parent pointer links) from the root of the tree. For example, if we’d like to insert the value 15 into our example tree, there are four nodes that will return true for the BelongsInSubtree function above. What values are stored in those nodes? Which of these nodes will be selected by Insert as the immediate parent for ’15’?
Now, let’s write the `DestroyTree` function, which cleans up all of the dynamically allocated memory associated with Ben’s upside-down tree leaving a valid empty tree.

```cpp
void DestroyTree(std::list<Node*> &leaves) {
    // sample solution: 14 line(s) of code
}
```

If the tree contains \( n \) elements, and is approximately balanced, what is the order notation of your implementation of `destroy tree`? Write 2-3 sentences justifying your answer.
The costume shop owner from Homework 7 has asked for help predicting what costumes their indecisive customers might choose in the future. Looking at the history of costume rentals they suspect there might be a pattern when a customer changes their mind about their Halloween costume.

For the example data on the left, we can see two instances where a customer switched from a pirate costume to a doctor costume, and only once did a customer switch from a pirate costume to a zombie costume. Here is the output we expect from the sample ‘r’ = rental and ‘h’ = print costume history commands:

```
history for pirate
   next rental was doctor 1 time(s)
   next rental was zombie 2 time(s)
no next rental history for elf
history for doctor
   next rental was pirate 1 time(s)
```

The shop owner emphasizes the need for fast performance in this implementation, since the system will be handling the records for thousands of customers and costumes in many different cities.

### 8.1 Data Structure Sketch

Let’s store this data in two variables, one with the current customer information, and the second with the costume history. (You’ll specify the typedefs in the next part). Sketch the contents of these variables after the rental commands above. Follow the conventions from lecture for your diagrams.
8.2 The typedefs

Next, fill in these typedef declarations.

```cpp
typedef PEOPLE_TYPE;

typedef HISTORY_TYPE;
```

8.3 Implementation of the Rental Command

Now, complete the implementation:

```cpp
int main() {
    PEOPLE_TYPE people;
    HISTORY_TYPE history;
    std::string first, last, costume;
    char c;
    while (std::cin >> c) {
        if (c == 'r') {
            std::cin >> first >> last >> costume;

    sample solution: 6 line(s) of code
```
8.4 Implementation of the History Command [ / 10 ]

```cpp
else {
    assert (c == 'h');
    std::cin >> costume;
}
```

8.5 Order Notation [ / 5 ]

If the shop has $p$ customers, $c$ costumes, and $r$ total rental events, what is the order notation for performing a single rental (the 'r' command)? Write 1-2 sentences justification.

What is the order notation for performing a history query (the 'h' command)? (justify your answer)
Complete the functions below without using any additional \textit{for} or \textit{while} expressions. Given an STL vector of words, find all pairs of those words that share at least one common letter. For example, if \textit{words} contains: apple boat cat dog egg fig then \textit{common(words)} should return: (apple,boat) (apple,cat) (apple,egg) (boat,cat) (boat,dog) (dog,egg) (dog,fig) (egg,fig)

typedef std::set<std::pair<std::string, std::string>> set_of_word_pairs;

bool common(const std::string &a, const std::string &b) {
    for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < b.size(); j++)
            if (a[i] == b[j]) return true;
    return false;
}

void common(set_of_word_pairs &answer, const std::vector<std::string>& words, int a, int b) {
    \textit{sample solution: 6 line(s) of code}
}

void common(set_of_word_pairs &answer, const std::vector<std::string>& words, int a) {
    \textit{sample solution: 4 line(s) of code}
}

set_of_word_pairs common(const std::vector<std::string>& words) {
    \textit{sample solution: 5 line(s) of code}