Final Exam

Your name: CS106B Rockstar

Sunet ID: cs106b-rockstar@stanford.edu

Section Leader: None

Honor Code: I hereby agree to follow both the letter and the spirit of the Stanford Honor Code. I have not received any assistance on this exam, nor will I give any. The answers I am submitting are my own work. I agree not to talk about the exam contents to anyone until a solution key is posted by the instructor.

Signature: ____ ← **YOU MUST SIGN HERE!**

Rules: (same as posted previously to class web site)

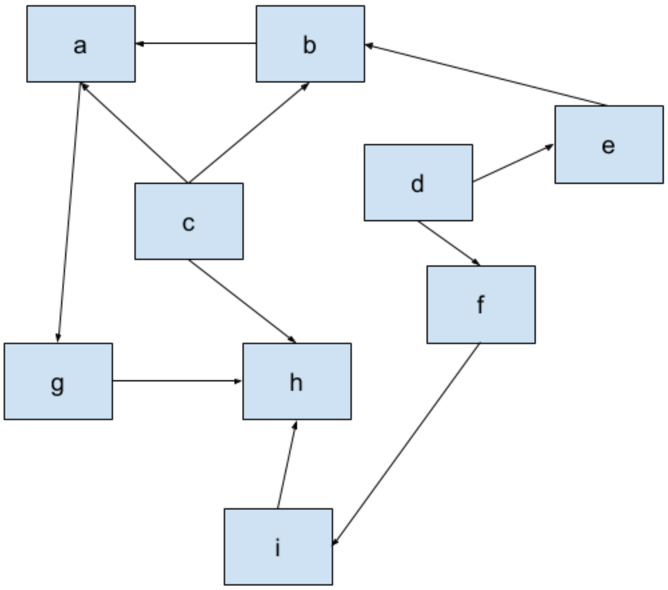
- This exam is to be completed by each student individually, with no assistance from other students.
- You have 3 hours (180 minutes) to complete this exam.
- This test is **closed-book, closed-notes**. You may only have one 8.5x11" double-sided sheet of notes.
- You may not use any computing devices, including calculators, cell phones, iPads, or music players.
- Unless otherwise indicated, your code will be graded on proper behavior/output, not on style.
- On code-writing problems, you do not need to write a complete program, prototypes, nor #include statements. Write only the code (function, etc.) specified in the problem statement.
- Unless otherwise specified, you can write helper functions to implement the required behavior. When asked to write a function, do not declare any global variables.
- Do not abbreviate code, such as writing ditto ("") or dot-dot-dot marks (...). Pseudo-code will be given no credit.
- If you wrote your answer on a back page or attached paper, please label this clearly to the grader.
- Do NOT staple or otherwise insert new pages into the exam. Changing the order of pages in the exam confuses our automatic scanning system.
- **You should write your name at the top of every page in the exam.** You will get 1 point for this.
- **Tear off the last page (reference sheet)** before submission. You will get 1 point for this.
- Follow the Stanford Honor Code on this exam and correct/report anyone who does not do so.

#	Description	Earned	Max
0	Names and removing last page		2
1	Graph Potpourri (read)		13
2	Pointer Trace (read)		11
3	Linked Lists (write)		16
4	Binary Search Tree Trace (read)		6
5	Trees (write)		15
6	Backtracking (write)		12
	Total		**75**

Name: ______________________________

1. Graph Potpourri

Parts a and b refer to the following graph.



a) Give a topological ordering for the graph above. Please do not include any punctuation.

We accepted any one of the following:

```
['cdebafgih', 'cdebafigh', 'cdebagfih', 'cdebfagih', 'cdebfaigh', 'cdebfiagh', '
   cdefbagih', 'cdefbaigh', 'cdefbiagh', 'cdefibagh', 'cdfebagih', 'cdfebaigh', '
   cdfebiagh', 'cdfeibagh', 'cdfiebagh', 'dcebafgih', 'dcebafigh', 'dcebagfih', '
   dcebfagih', 'dcebfaigh', 'dcebfiagh', 'dcefbagih', 'dcefbaigh', 'dcefbiagh', '
   dcefibagh', 'dcfebagih', 'dcfebaigh', 'dcfebiagh', 'dcfeibagh', 'dcfiebagh', '
   decbafgih', 'decbafigh', 'decbagfih', 'decbfagih', 'decbfaigh', 'decbfiagh', '
   decfbagih', 'decfbaigh', 'decfbiagh', 'decfibagh', 'defcbagih', 'defcbaigh', '
   defcbiagh', 'defcibagh', 'deficbagh', 'dfcebagih', 'dfcebaigh', 'dfcebiagh', '
   dfceibagh', 'dfciebagh', 'dfecbagih', 'dfecbaigh', 'dfecbiagh', 'dfecibagh', '
   dfeicbagh', 'dficebagh', 'dfiecbagh', 'higfabedc', 'hgifabedc', 'hifgabedc', '
   higafbedc', 'hgiafbedc', 'hgaifbedc', 'higabfedc', 'hgiabfedc', 'hgaibfedc', '
   hgabifedc', 'higabefdc', 'hgiabefdc', 'hgaibefdc', 'hgabiefdc', 'hgabeifdc', '
   higfabecd', 'hgifabecd', 'hifgabecd', 'higafbecd', 'hgiafbecd', 'hgaifbecd', '
   higabfecd', 'hgiabfecd', 'hgaibfecd', 'hgabifecd', 'higabefcd', 'hgiabefcd', '
   hgaibefcd', 'hgabiefcd', 'hgabeifcd', 'higfabced', 'hgifabced', 'hifgabced', '
   higafbced', 'hgiafbced', 'hgaifbced', 'higabfced', 'hgiabfced', 'hgaibfced', '
   hgabifced', 'higabcfed', 'hgiabcfed', 'hgaibcfed', 'hgabicfed', 'hgabcifed', '
   higabecfd', 'hgiabecfd', 'hgaibecfd', 'hgabiecfd', 'hgabeicfd', 'higabcefd', '
   hgiabcefd', 'hgaibcefd', 'hgabicefd', 'hgabciefd', 'hgabecifd', 'hgabceifd']
```

b) What is the adjacency list for the graph above? For each node, list the corresponding set in alphabetical order.

a: {g}

b: {a}

c: {a, b}

d: {e, f}

e: {b}

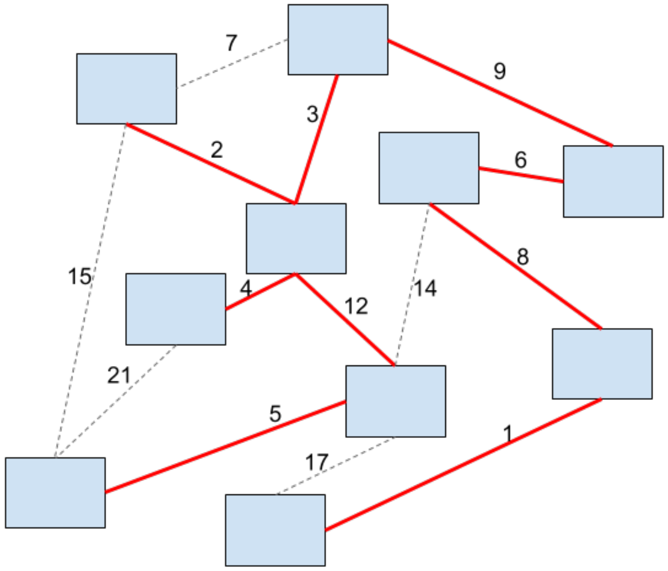
f: {i}

g: {h}

h: {}

i: {h}

c) What is a minimum spanning tree for the graph below? Please clearly draw it directly on the image by circling the relevant edges.



d) You need to determine whether an undirected graph is connected. How could you use BFS to do so?

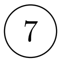
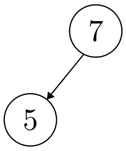
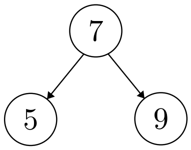
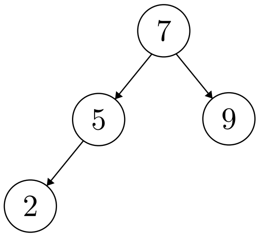
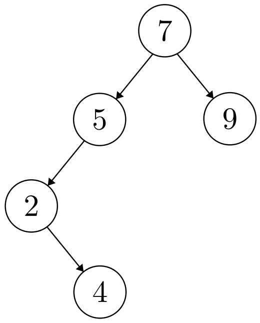
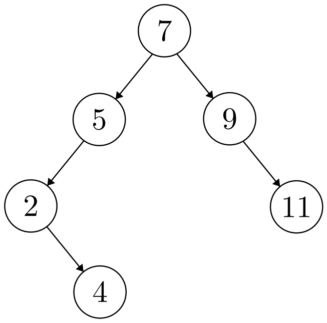
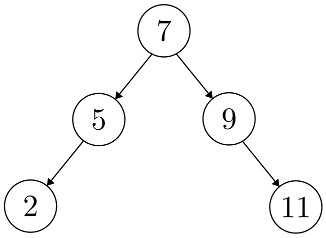
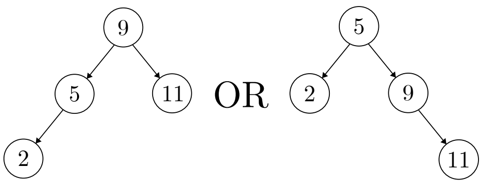
Run BFS at a node, marking all the nodes that are visited. After BFS is run, check to make sure all nodes are visited. If there are unvisited nodes, the graph is not connected.

e) Why is Dijkstra's algorithm for finding least-cost paths a worse choice than BFS for determining whether an undirected graph is connected?

Dijkstra's algorithm is less efficient than BFS (greater Big-O). We do not care about edge weights when determining whether a graph is connected or not, so BFS is a better answer.

2. Trace

Draw the resulting Binary Search Tree at each step from inserting and deleting the nodes in the following order. You should follow the algorithms for insertion and deletion that we discussed in class. The first two are completed for you. **You should not rebalance the tree.**

1. insert 7  	2. insert 5 
3. insert 9  	4. insert 2 
5. insert 4  	6. insert 11 
7. delete 4  	8. delete 7 

Name: ______________________________________

3. Pointer Trace

Draw the state of memory at the specified place in this code.

- Be careful in showing where your pointers originate and terminate (pointing at a struct versus a field in a struct). Label all objects with their type, and include names for variables.
- Mark uninitialized or unspecified areas with a question mark (?), and clearly mark nullptr values by writing NULL or drawing a slash through the box.
- Clearly indicate if any objects or memory are orphaned by labeling the relevant box as orphaned.
- Draw the components in the appropriate stack and heap areas marked for you.
- Draw struct fields adjacent to each other (held in one subdivided box), and label each field with the field name for clarity.
- If you find in the course of this trace that stack and/or heap memory no longer exists, you may just clearly cross out the relevant portion. We won't grade anything crossed out.

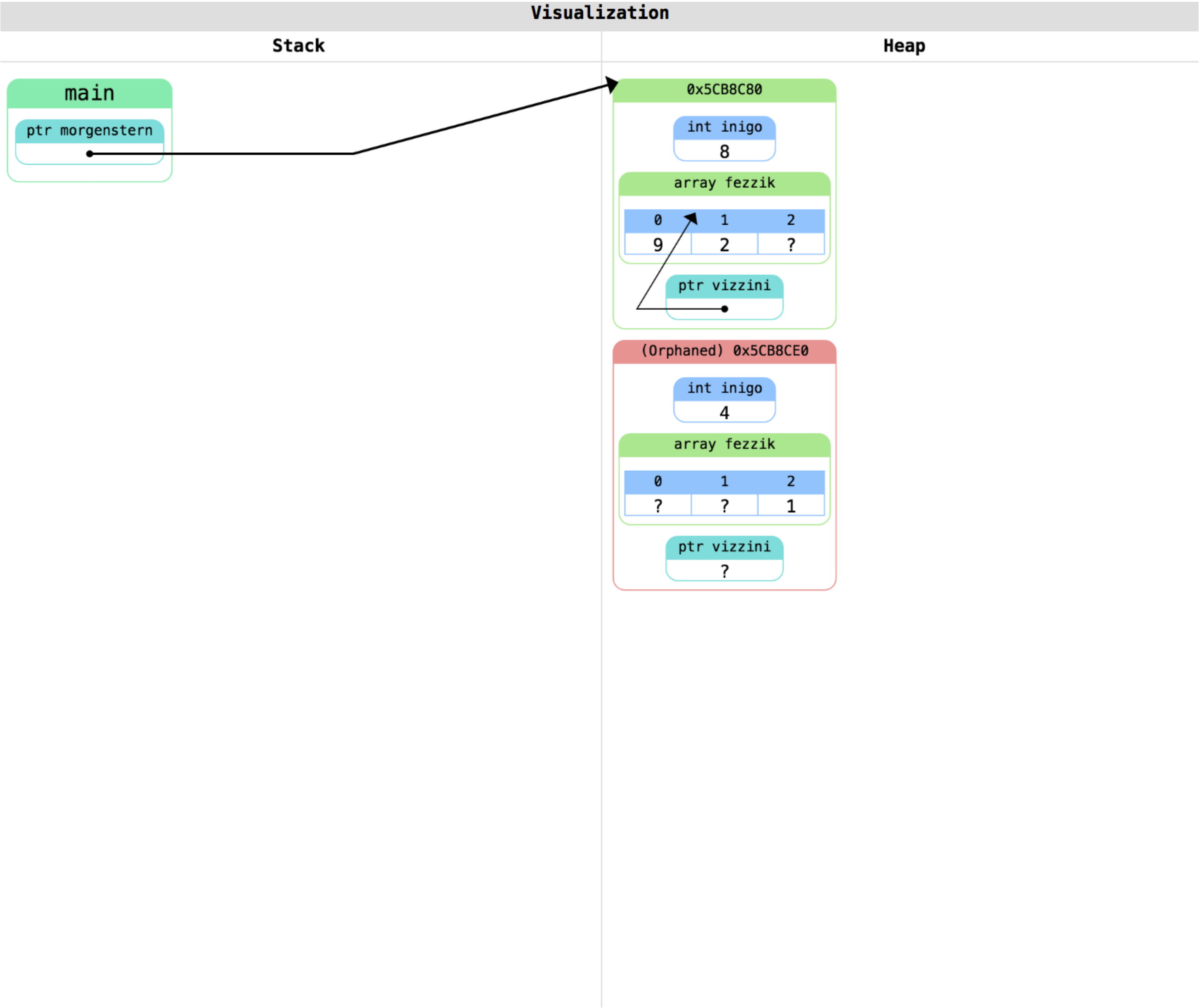
Hint: Rather than getting stuck on a particular line, draw the effects of the lines you know. We will give partial credit on this question.

```
struct Circus {
    int inigo;
    int fezzik[3];
    int *vizzini;
};
void manInBlack(Circus *rugen) {
    rugen->vizzini = &rugen->fezzik[1];
    *(rugen->vizzini) = 2;
    rugen->fezzik[0] = 9;

    rugen = new Circus;
    rugen->fezzik[2] = 1;
    rugen->inigo = 4;
}
int main() {
    Circus *morgenstern = new Circus;
    morgenstern->inigo = 8;
    morgenstern->fezzik[1] = 4;
    manInBlack(morgenstern);
    // ***DRAW THE MEMORY HERE***
    return 0;
}
```

Write your answer on the following page.

Name: ____

3. Pointer Trace (Writing Space)



Name: ____________________________________

4. Linked Lists

Recall the MergeSort algorithm we discussed in class, which recursively sorts a list and merges the sorted sublists together. We wrote the code for MergeSort for a Vector.

You will write two helper functions, culminating in writing the code for MergeSort on an unsorted linked list.

Recall the `Node` struct:

```
struct Node {
    int data;
    Node *next;
};
```

4. Part a

Write a function that, given a pointer to the front of a singly-linked list, returns the length of the linked list.

- You may not use any collections or arrays to solve this problem. You may not define your own structs or classes.
- You may not modify the list nor the nodes of the list in any way.

Name: ______________________________________

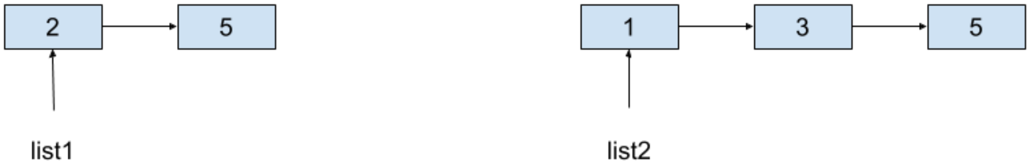
4. Part a (Writing Space)

```
int length(Node* front) {

    int result = 0;
    for (Node *curr = front; curr != nullptr; curr = curr->next) {
        result++;
    }
    return result;

}
```

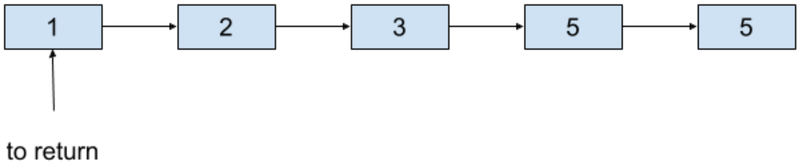
Name: ______________________________________

4. Part b

Write a function that, given pointers to two sorted linked lists, returns a pointer to the front of a list that contains all the nodes in the inputted lists sorted in increasing order of the `data` field. You will be **rewiring pointers** in the given linked lists. For example, if the inputs were:



You should return the following list:



You may make the following assumptions:

- The pointers `list1` and `list2` point to the front of linked lists whose nodes are sorted in increasing order of the `data` field.
- No nodes are in both `list1` and `list2` (though the same `data` value may appear in multiple nodes, including in both lists).
- Neither `list1` nor `list2` points to an empty list.

For full credit, your solution should adhere to the following constraints:

- You may not use any collections or arrays to solve this problem. You may not define your own structs or classes.
- Your solution should run in O(N) time, where N is the number of nodes in the resulting list.
- You may not change the `data` field of any node (though you can change the `next` field), **nor create nor delete nodes**.

Write your answer on the following page.

Name: ______________________________________

4. Part b (Writing space)

```
Node *mergeLists(Node *list1, Node *list2) {

    Node *front;
    if (list1->data < list2->data) {
        front = list1;
        list1 = list1->next;
    } else {
        front = list2;
        list2 = list2->next;
    }
    Node *curr = front;
    while (list1 && list2) {
        if (list1->data < list2->data) {
            curr->next = list1;
            list1 = list1->next;
        } else {
            curr->next = list2;
            list2 = list2->next;
        }
        curr = curr->next;
    }
    if (list1) {
        curr->next = list1;
    } else {
        curr->next = list2;
    }
    return front;

}
```

Name: ____

4. Part c

Using the helper functions from parts a and b, write a function that uses the MergeSort algorithm discussed in class to recursively sort and merge a linked list. As a refresher, we've included the following pseudocode for the MergeSort algorithm:

```
split the list in half.
recursively sort each half.
merge the halves back together.
```

At the end of your function, `front` should point to the front of the now sorted, singly-linked list.

You may make the following assumptions:

- The pointer `front` points to the front of an unsorted, singly-linked list.
- Your functions from parts a and b work as described and are available for your use.

For full credit, you solution should adhere to the following constraints:

- You may not use any collections or arrays to solve this problem. You may not define your own structs or classes.
- You may not change the data field of any node, or create or delete nodes.
- Your solution should run in O(N lg N) time, where N is the number of nodes in the resulting list.
- You must use the MergeSort algorithm as described above.

Write your answer on the following page.

Name: ________________________________

4. Part c (Writing space)

```
void mergeSort(Node *&front) {

    int listSize = size(front);
    Node *firstHalf = front;
    int index = 0;
    Node *secondHalf = front;
    while (true) {
        if (index == listSize / 2 - 1) {
            Node *temp = secondHalf;
            secondHalf = secondHalf->next;
            temp->next = nullptr;
            break;
        }
        secondHalf = secondHalf->next;
        index++;
    }
    mergeSort(firstHalf);
    mergeSort(secondHalf);
    front = merge(firstHalf, secondHalf);

}
```

Name: ______________________________________

5. Tries Redux

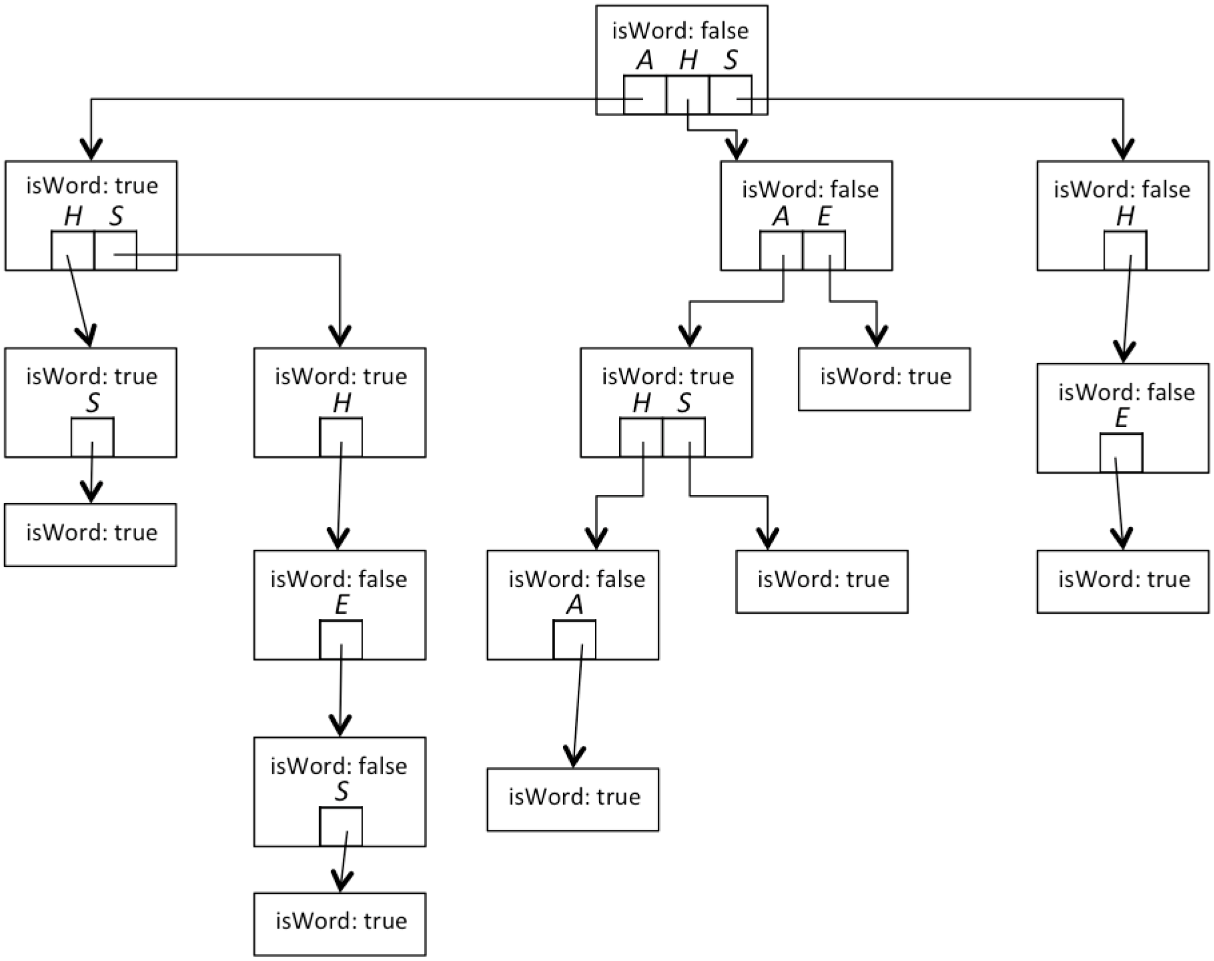
In MiniBrowser, you implemented a trie that would populate search suggestions matching a prefix that the user had typed. As we all know, humans are prone to typos. Moreover, your Trie suffers from another critical flaw in that you can't ever delete words from it, even if the corresponding article has been deleted or renamed on Wikipedia. You'll expand your Autocomplete class from MiniBrowser to address these shortcomings.

For the rest of this problem, assume you have the following .h file (the # statements are excluded for clarity):

```
class Autocomplete {
public:
    Autocomplete(); // already implemented
    ~Autocomplete(); // already implemented
    void add(string word); // already implemented

    Set<string> autocorrectedSuggestionsFor(
        string prefix, int maxHits, int maxMistakes) const; // implemented in part a
    void remove(string word); // implemented in part b

private:
    struct Node {
        bool isWord;
        Map<char, Node*> children; // notice we're using a Map, not an array
    };
    Node* root;
};
```

5. Part a

Implement the function `autocorrectedSuggestionsFor`, which takes in a user's `prefix` and `maxHits` and returns a `Set` of matching strings, as in MiniBrowser. In addition to those parameters, this function takes in a number `mistakesAllowed`, which is the maximum number of **letter substitutions** (changing one letter for another) in strings matching the prefix. For example, if the user searched for prefix "AH" in the Autocomplete represented below with a `mistakesAllowed` of 1 and `maxHits` of at least 5 for the trie below, the returned `Set` would be {AH, AHS, AS, ASHES, SHE} ("A" is too short, and the words beginning with "HA" or "HE" would require at least two changes).



Name: ______________________________________

You may make the following assumptions:

- The constructor, destructor, and `add` methods are all implemented correctly.
- You may write your own helper functions; assume they are added properly to the .h file.
- If more than `maxHits` matches exist, you may choose to return any `maxHits` of them.

For full credit, your solution should adhere to the following constraints:

- You may not define your own structs or classes.
- While your code does not need to have a particular Big-Oh, you should avoid explorations that will not lead to fruitful results. We reserve the right to deduct for inefficient or wasteful code.
- You may not use the method `suggestionsFor`. That method does not exist in the class.
- You may not create any new instance variables.
- You should not leak or orphan any memory.

Write your solution on the following page.

Name: ______________________________

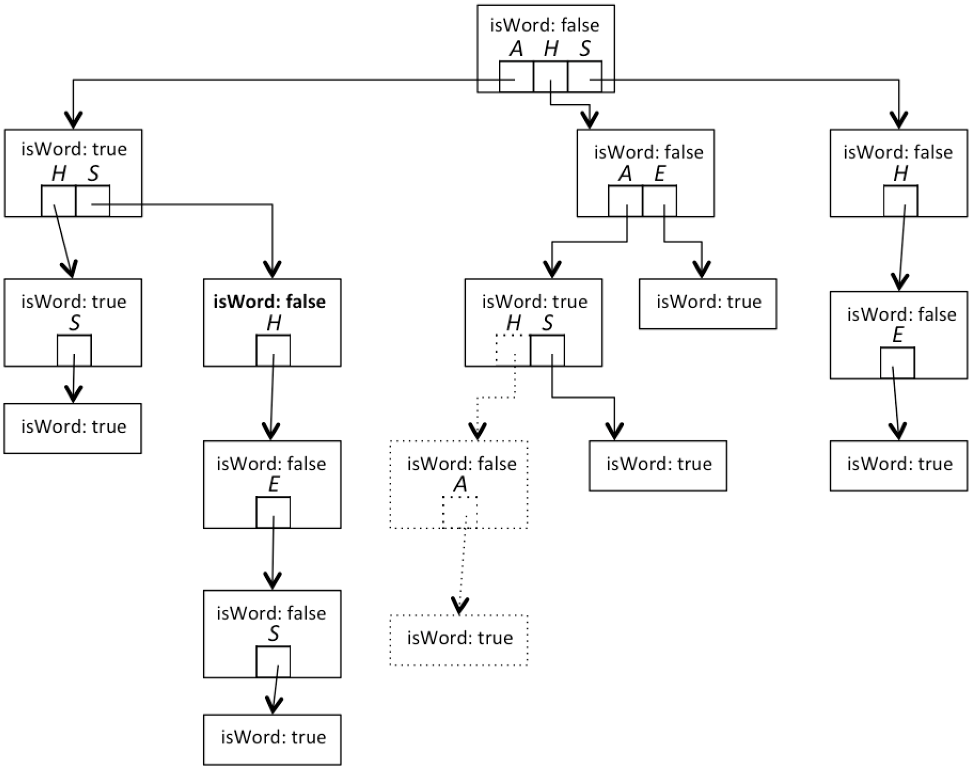
5. Part a (Writing space)

```
Set<string> Autocomplete::autocorrectedSuggestionsFor(
    string prefix, int maxHits, int mistakesAllowed) const {

    Set<string> result;
    autocorrectedSuggestionsForHelper (root, prefix, mistakesAllowed,
                                          maxHits, "", result);
    return result;

}

void Autocomplete::autocorrectedSuggestionsForHelper(
    Node *root, string prefix, int mistakesAllowed, int maxHits, string currWord,
    Set<string> &result) const {

    if (!root || numMistakes < 0 || result.size() >= maxHits) return;
    if (prefix.isEmpty()) {

        if (root->isWord) {
            result += currWord;
        }

        for (char ch : root->children) {
            suggestionsForAutocorrectHelper(root->children[ch], prefix,
                        mistakesAllowed, maxHits, currWord + ch, result);
        }

    } else {

        // If prefix[0] doesn't exist in children, then the next recursive
        // call will catch it in (!root) case and return {}
        autocorrectedSuggestionsForHelper(root->children[prefix[0]], prefix.substr(1),
                    mistakesAllowed, maxHits, currWord + prefix[0], result);

        for (char ch : root->children) {
            autocorrectedSuggestionsForHelper(root->children[ch], prefix.substr(1),
                        mistakesAllowed - 1, maxHits, currWord + ch, result);
        }
    }
}
```

Name: ______________________________________

5. Part b

Implement the function `remove` that removes a given word from the trie. In writing this function, you'll need to ensure that you maintain a key property of a trie: every leaf node in a trie represents a word. If the given word is not in the trie, the remove function should have no effect. Using the same autocomplete from part a, `remove("SHELL")` has no impact; `remove("AS")` leads to changing the relevant node to have an `isWord` value of false (bolded in the image below); and `remove("HAHA")` would result in a branch of the trie being deleted (dotted lines in the image below; notice that the key H is also removed from the node representing "HA"):



You may make the following assumptions:

- The constructor, destructor, and `add` methods are all implemented correctly.
- You may write your own helper functions and assume they are added to the .h file.

For full credit, your solution should adhere to the following constraints:

- You may not use any collections or arrays to solve this problem. You may not define your own structs or classes.
- You may not create any new instance variables.
- You should not leak or orphan any memory.

Write your solution on the following page.

Name: ______________________________________

5. Part b (Writing space)

```
void Autocomplete::remove(string word) {
    remove(root, word);
}

bool Autocomplete::remove(TrieNode *&node, string word) {
    if (!node) {
        return false;
    }
    if (word.isEmpty()) {
        if (!node->isWord) return false;
        node->isWord = false;
    } else if (remove(node->children[word[0]], word.substr(1))) {
        node->children.remove(word[0]);
    }
    if (node->children.isEmpty()) {
        delete node;
        node = nullptr;
        return true;
    }
    return false;
}
```

Name: ____________________________________

6. Recursia Event Planning Take Two

You were so successful as the planner of the coronation of the new Queen of Recursia that she has now hired you to plan her wedding reception. Instead of a large auditorium, however, the Queen has several tables, each of which can hold no more than a fixed number of guests. While there are enough spaces to seat every guest, the Queen is mindful that if any pair of enemies sit at the same table, a bloody duel may ensue.

Your job is to devise a seating arrangement, represented as a `Vector<Set<string>>`, where each `Set` represents a table. If no seating arrangement could be found, you should return an empty `Vector`. You are given a list of enemyships; you should assume that enemyships are mutual so if ″Eleanor″ is the enemy of ″Tahani,″ ″Tahani″ is also the enemy of ″Eleanor,″ though, to save space, only one of {″Eleanor″, ″Tahani″} or {″Tahani″, ″Eleanor″} will be listed.

Enemyships are stored in the following struct:

```
struct Enemyship {
    string first;
    string second;
};
```

For example, if you were given the `Vector<Enemyship>` {{″Tahani″, ″Eleanor″}, {″Eleanor″, ″Jason″}} and the `Vector` of guests (where each guest's name is represented as a string) {″Tahani, ″Eleanor″, ″Chidi″, ″Jason″}, with the constraint of two tables each seating a maximum of two guests, you could return the following seating arrangement:

```
[{"Tahani", "Jason"}, {"Eleanor", "Chidi"}]
```

Note that not every table needs to be filled, but the total number of tables in your seating arrangement cannot exceed the number of tables passed in as `numTables`, nor can the number of individuals at a table exceed `maxGuests`.

To receive full credit, your solution should adhere to the following constraints:

- You should choose an efficient solution. Choose data structures intelligently and use them properly. While your code is not required to have any particular Big-Oh, you may lose points if your code is inefficient. Specifically, if the given data structures are not well-suited for providing the information you need, you may need to restructure the data.
- You may not define any custom classes or structs; you must use the Stanford collections to solve this problem.
- All parameters should be unchanged at the end of your function.

You can make the following assumptions:

- The number of guests is no more than `maxGuests * numTables`.

Name: ______________________________________

- All citizens' names are unique.
- Enemyships are mutual, though only one direction will be represented in the Vector of enemies.
- You may use a recursive helper function if you'd like.
- You do not need to include any function prototypes or #include statements.

Write your solution on the following page.

Name: ____________________________________

6. Recursia Event Planning Take Two (Writing space)

```
Vector<Set<string>> findSeatingArrangement(Vector<Enemyship> &enemyships,
                                           Vector<string> &guests,
                                           int numTables,
                                           int maxGuests) {
    Map<string, Set<string>> enemies;
    for (Enemyship enemyship : enemyships) {
        enemies[enemyship.first] += enemyship.second;
        enemies[enemyship.second] += enemyship.first;
    }
    Vector<Set<string>> result(16);
    findSeatingArrangementHelper(enemies, guests, result, numTables, maxGuests);
    return result;
}

bool findSeatingArrangementHelper(Map<string, Set<string>> &enemies,
                                  Vector<string> &guests,
                                  Vector<Set<string>> &arrangement,
                                  int numTables,
                                  int maxGuests) {
    if (guests.isEmpty()) {
        return true;
    }
    string guest = guests[guests.size() - 1];
    guests.remove(guests.size() - 1);

    for (Set<string> table : arrangement) {
        if (table.size() < maxGuests && (table * enemies[guest]).isEmpty()) {
             table += guest;
            if (findSeatingArrangementHelper(enemies, guests, arrangement,
                    numTables, maxGuests)) {
                guests.add(guest);
                return true;
            }
            table -= guest;
        }
    }
    guests.add(guest);
    return false;
}
```