ANSWER SHEET
These problems were generated by TAs and instructors in previous semesters. They may or may not match the actual difficulty of problems on Test4.

Dictionaries

1. The fruits in a bag of groceries are provided as a list (eg. `["apple", "oranges", "banana", "kiwis"]`). For any elements in the list that are plural, like “kiwis”, we say there are 3 of that fruit in the bag. (There are no fruits in the bag where the singular form of the word ends in an “s”.) Write a function `groceryCount` that outputs a dictionary with each fruit name and its frequency as a key-value pair.

For example:

```
groceryCount(["apple", "oranges", "banana", "kiwis", "kiwi"]) ==
{ "apple" : 1, "orange" : 3, "banana" : 1, "kiwi" : 4 }
```

```
ANSWER:
def groceryCount(L):
      d = {}
      for i in range(len(L)):
            fruit = L[i]
            quantity = 1
            if fruit[-1] == "s":
                  fruit = fruit[:-1]
                  quantity = 3
            if (fruit in d):
                  d[fruit] = d[fruit] + quantity
            else:
                  d[fruit] = quantity
      return d
```

2. What does the following code print?

```
def chainedKeys(dict, startkey):
    key = startkey
    while key in dict.keys():
        key = dict[key]
        print(key)
    return key
result = chainedKeys({3:6, 4:9, 5:7, 6:5, 7:4, 8:9, 9:2}, 3)
print(str(result) + " is missing")
```

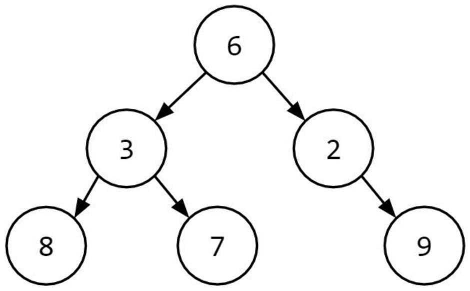
```
ANSWER:
6
5
7
4
9
2
2 is missing
```

What is one key whose **value** could be changed to make the function loop infinitely? What should the value be changed to?

ANSWER:
Many possible answers. One is:
Key: 9, Value: 3

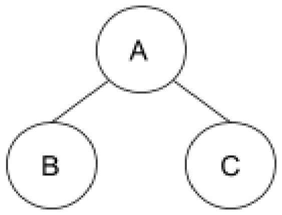
Trees

1. Write a function that calculates the height of a (possibly unbalanced) binary tree. For example, the tree below would have a height of 3.

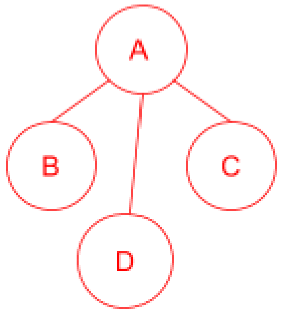


ANSWER:
```
def getHeight(tree):
    if tree == None:
        return 1
    else:
        right = getHeight(tree["right"])
        left = getHeight(tree["left"])
        return 1 + max(right, left)
```

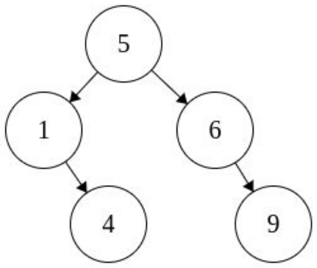
2. Add a node and edge to transform this tree from a binary tree to a general tree (i.e., it should no longer be binary).



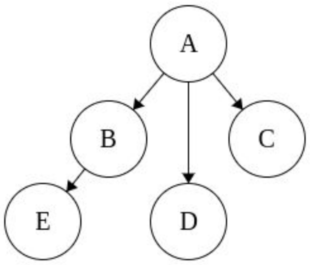
ANSWER:



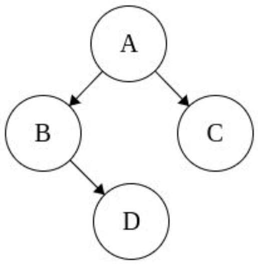
3. Label the following trees with either tree, binary tree, or binary search tree, giving the most specific term if multiple terms apply.



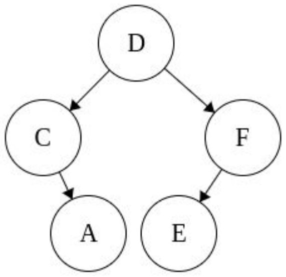
binary search tree



tree



binary tree



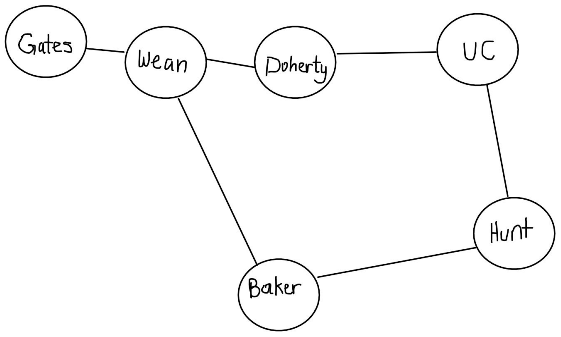
binary tree

Graphs

1. How are graphs and trees similar? How are they different?

ANSWER:
Similar: both use nodes and connect the nodes together
Different: any node can be connected to any other node in a graph; there are more restrictions with trees (hierarchical)

2. Write the dictionary that would represent the following graph.

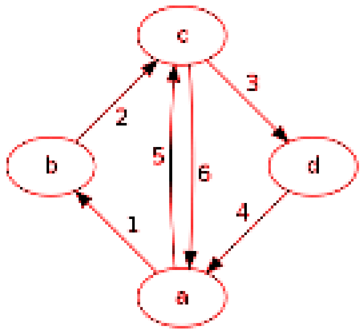


ANSWER:
```
{ "Baker"   : [ "Hunt", "Wean" ],
  "Hunt"    : [ "Baker", "UC" ],
  "Doherty" : [ "Wean", "UC" ],
  "Gates"   : [ "Wean" ],
  "UC"      : [ "Doherty", "Hunt" ],
  "Wean"    : [ "Baker", "Doherty", "Gates" ] }
```

3. Draw a directed graph with weights based on the given dictionary:

```
graph = { "a" : [ ["b", 1], ["c", 5] ],
          "b" : [ ["c", 2] ],
          "c" : [ ["a", 6], ["c", 3] ],
          "d" : [ ["a", 4] ] }
```

ANSWER:



Search Algorithms II

1. What properties does a hash function need to achieve O(1) lookup in a hashtable?

ANSWER:
Hash function results don't change over time, two unique values are likely hash to different values (collision-resistant)

2. What are some data structures that you can apply hash functions to? And what are the ones you cannot?

ANSWER:
Hash functions can only be applied to immutable data structures such as strings and integers. They cannot be applied to mutable values such as lists and dictionaries.

3. What would happen if two objects happen to have the same hash values? Is it allowed?

ANSWER:
If two objects happen to have the same hash values, collision would happen. However, in general, hash tables are designed to handle collisions. One way to handle that would be create a list of objects with the same hash values and then put them in that bucket.

4. Each student in 15-110 is asked to create a list of their favorite ice cream flavors, which they can update as their tastes change.

Students A and B say ["chocolate", "vanilla", "strawberry"],
Student C says ["mint chocolate chip", "chocolate", "strawberry"], and
Student D says ["vanilla", "mint chocolate chip", "chocolate"].
Later, Student C finds the best strawberry ice cream ever and destructively shuffles their favorite list.

Your professors want to maintain a dictionary of ice cream preferences (keys) to student counts (values) in order to keep track of which combination is the favorite at the time the preferences were first collected. How should they store the keys?

- ☐ Keys are the lists that the students submit.

- ☐ Keys are lists converted into strings using str().

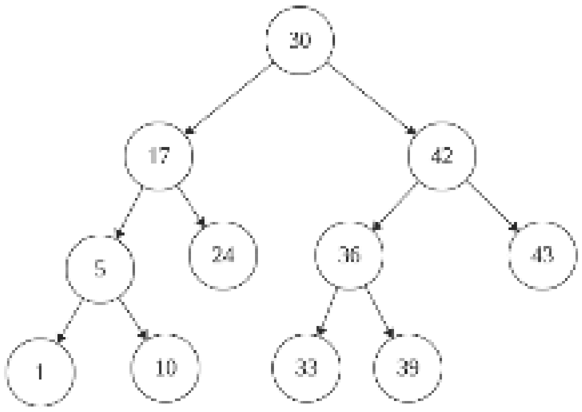
Why that approach?

ANSWER:
We cannot hash mutable values. But we only want to track the initial preferences of students, not final preferences. Converting a list to a string will capture that initial preference and make it immutable.

The professors also try to store each of the flavors in a hash table by the index of the first letter in the alphabet (i.e., almond ice cream would be index 0, birthday cake would be 1). Insert **chocolate, jalapeno, mint, and greentea** into the following hash table, using the algorithm discussed in class and practiced on your homework.

	greentea	chocolate mint		jalapeno

5. Consider the binary search tree below. What nodes would you visit while searching the tree for the value 33?



ANSWER:
Nodes visited: 30, 42, 36, 33

6. Given a description of a search algorithm, identify A) what kind of data structure is being searched, and B) what the name of that search algorithm is. Each algorithm is searching a data structure for the value `item`.

A: If the node's value equals `item`, return `True`. If `item` is less than the node's value, recurse on the left child and return the result; otherwise, recurse on the right child and return the result.

ANSWER:
Data structure: binary search tree
Algorithm: binary search

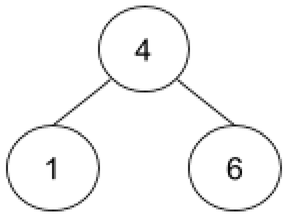
B: Go through each value sequentially, starting from the beginning. If you reach a value that equals `item`, return `True`. If you run out of values to search, return `False`.

ANSWER:
Data structure: list
Algorithm: linear search

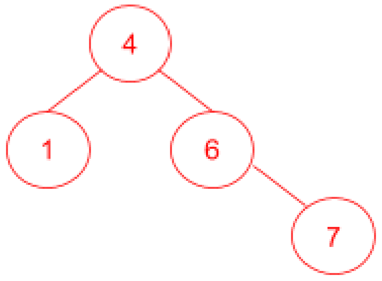
C: Begin with the start node in the to-visit list. While there are still nodes left to visit, check if the next node on the to-visit list equals `item`, and return `True` if it does. Then check if it has been visited before. If it has not, add all of the nodes connected to that node to the **end** of the to-visit list. If the to-visit list becomes empty, return `False`.

ANSWER:
Data structure: graph
Algorithm: breadth-first search

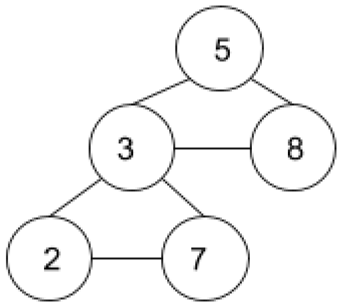
7. Add a node with value 7 to the tree so that it remains a binary search tree.



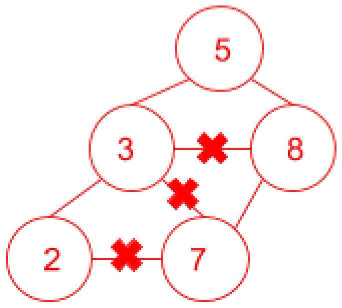
ANSWER:



8. Draw an **X** over each edge that must be removed and draw **a dotted line** for any edges that must be added to make this graph a binary search tree.



ANSWER:



Tractability

1. State True/False for the following questions and explain the answer.
   a. If we have an algorithm to solve a problem then we have a tractable solution.

ANSWER:
False. We can have an algorithm to solve an intractable problem.

   b. Any problem that runs in O(n^k) is intractable.

ANSWER:
False. n^k is polynomial. Intractable runtimes are: 2^n, k^n, and n!

   c. If any problem that can be solved in polynomial time can be verified in polynomial time, then we have proved P=NP.

ANSWER:
False. We already know every problem solved in polynomial time (every problem in P) is verifiable in polynomial time. This tells us P is a subset of NP.

2. Is exam scheduling a tractable problem? If yes, explain why / how you know. If not, explain how we still get all of our exams scheduled.

ANSWER:
No, exam scheduling is intractable. When we have these large, intractable problems, we find approximate solutions, and they just take a long time to generate.

3. For each question, check the box next to the correct answer.

☐ True or [✓ False]	NP means "Not P" so everything not in P is in NP.
[✓ True] or ☐ False	If every problem in NP can be solved in polynomial time, then P=NP
[✓ True] or ☐ False	A problem is intractable if it can be solved but it may take too long to practically get that answer.
☐ True or [✓ False]	All problems in NP are intractable to verify.