CSE 2321 Homework 10 Template

In Latex the lstlisting environment is one of the best ways to write code or pseudocode. Python is syntactically nice and similar to how I like to write pseudocode, so I use that option even though this isn't valid Python code. This is how I formatted the Bipartite algorithm from lecture, you can copy/paste and replace my code with your own:

```
Bipartite(G):
    bipartite = true
    for each v in V
        v.color = None
    for each v in V
        if v.color = None
            bipartite = bipartite and two_color(G, v)
    return bipartite

two_color(G, s):
    s.color = 0
    q.enqueue(s)
    while q.length != 0
        v = q.dequeue()
        for each w such that {v, w} in E
            if w.color = v.color
                return false
            else if w.color = None
                w.color = (v.color + 1) mod 2
                q.enqueue(w)
    return true
```

Problem 1

There is a method alphabet that takes a list of strings which contains all of the words and returns a string, which is a possible alphabet.
It creates an empty set of characters which contains all of the letters of the alphabet.
Using an iterator, it iterates through all words in the set and adds all of the unique characters to the set using the contains() method.

Then, the set of characters is used to create vertices for every character in the alphabet.
While the list is not empty,
Remove the first two entries in the list. These will be strings word1 and word2.
Then, using a for loop that goes from i = 0 to i ¡ length of the shorter word,
It compares the first characters of word 1 and word 2 (at index 0).
If they (the first character) are not equal, then an edge is drawn from the vertex of the first character of word 1 to the vertex of the first character of word 2.
The for loop will continue to compare every character at index i until a differing character is found or the loop terminates.
Then, word2 = word 1, and a new entry is removed from the list that becomes word2.
This process (inside of the while loop) will continue until the list is empty and all of the verifiable orders between characters are represented by edges on the directed graph.
Any remaining vertices in the graph without edges will be randomly connected until all of the vertices are connected.
Then, the graph is put into the topological sort algorithm to order the vertices.
The resulting string that represents the alphabet will be derived starting from the first to the final vertex.

Problem 2

There is an empty stack of nodes that will contain the vertices of the graph.
There is a current time, represented by an int called time.
Each vertice is represented by a node that contains data to represent the parent pi (char), the discovery time d (int), and the finishing time (int). These data are initially null/empty.
The starting vertex s is pushed into the stack and is marked as visited with a discovery time d = time = 0. It does not have a parent node.
While the stack is not null,
Explore the next adjacent node if it is not already visited, and push that

node into the stack.
The current node's parent node pi will be the node that was visited right before it.
The current node will have a discovery time d = 1 + time.
The current node is marked as visited.
The current time t will be incremented by 1.
If there is another adjacent unvisited node, it becomes the new current node and it is pushed into the stack and the above steps are repeated
Else, pop the current node from the stack and set its finishing time f = time + 1. Then retreat back to the parent node of the current node, and continue until the current node has an adjacent unvisited node.
This while loop will continue until the current node retreats back to the starting node again, and the starting node is popped from the stack, leaving a null stack.
At this point, all nodes have been visited, and have a parent node (with the exception of the starting node, a discovery time, and a finishing time.

Problem 3

There is a method alphabet that takes a list of strings which contains all of the words and returns a string, which is a possible alphabet.
It creates an empty set of characters which contains all of the letters of the alphabet.
Using an iterator, it iterates through all words in the set and adds all of the unique characters to the set using the contains() method.
Then, the set of characters is used to create vertices for every character in the alphabet.
While the list is not empty,
Remove the first two entries in the list. These will be strings word1 and word2.
Then, using a for loop that goes from i = 0 to i ¡ length of the shorter word,
It compares the first characters of word 1 and word 2 (at index 0).
If they (the first character) are not equal, then an edge is drawn from the vertex of the first character of word 1 to the vertex of the first character of word 2.

The for loop will continue to compare every character at index i until a differing character is found or the loop terminates.
Then, word2 = word 1, and a new entry is removed from the list that becomes word2.
This process (inside of the while loop) will continue until the list is empty and all of the verifiable orders between characters are represented by edges on the directed graph.
Any remaining vertices in the graph without edges will be randomly connected until all of the vertices are connected.
Then, the graph is put into the topological sort algorithm to order the vertices.
The resulting string that represents the alphabet will be derived starting from the first to the final vertex.