1 Longest Prefix

Fill in the `longestPrefixOf(String word)` method below such that it returns the longest prefix of `word` that is also a prefix of a key in the trie.

For example, if a `TrieSet t` contains keys `{"cryst", "tries", "cr"}`, then `t.longestPrefixOf("crystal")` returns `"cryst"` and `t.longestPrefixOf("crys")` returns `"crys"`.

The code uses the `StringBuilder` class to build strings character-by-character. To add a character to the end of the `StringBuilder`, use the `append(char c)` method. Once all characters have been appended, the resulting String is returned by the `toString()` method.

```
StringBuilder sb = new StringBuilder();
sb.append('a');
sb.append('b');
System.out.println(sb.toString()); // "ab"
```

```
public class TrieSet {
    private Node root;
    private class Node {
        boolean isKey;
        Map<Character, Node> map;
        private Node() {
            isKey = false;
            map = new HashMap<>();
        }
    }

    public String longestPrefixOf(String word) {
        int n = word.length();
        StringBuilder prefix = new StringBuilder();
        Node curr = root;
        for (int i=0; i<n; i++) {
            if(curr.map.containskey(word.charAt(i))){
                prefix.append(word.charAt(i));
                curr=curr.map.get(word.charAt(i));
            }else{
                break;
            }
        }
        return prefix.toString();
    }
}
```

2 A Tree Takes On Graphs

Your friend at Stanford has come to you for help on their homework! For each of the following statements, determine whether they are true or false; if false, provide counterexamples.

(a) "A graph with edges that all have the same weight will always have multiple MSTs."

True X False

(b) "No matter what heuristic you use, A* search will always find the correct shortest path."

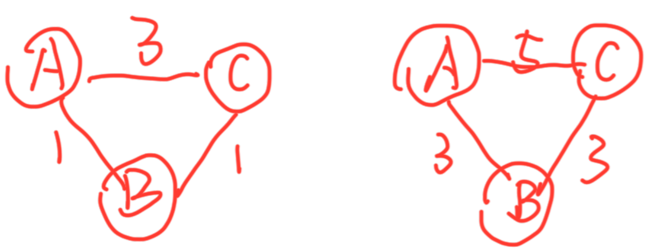
False, it depends on wether h(n) is resonable.

(c) "If you add a constant factor to each edge in a graph, Dijkstra's algorithm will return the same shortest paths tree."

✓X

False, it will be false if the factor is negative.

refer to solution



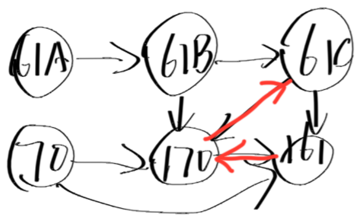
3 Class Enrollment

DAG (Directed Acyclic Graph)

You're planning your CS classes for the upcoming semesters, but it's hard to keep track of all the prerequisites! Let's figure out a valid ordering of the classes you're interested in. A valid ordering is an ordering of classes such that every prerequisite of a class is taken before the class itself. Assume we're taking one CS class per semester.

(a) The list of prerequisites for each course is given below (not necessarily accurate to actual courses!). Draw a graph to represent our scenario.

- CS 61A: None
- CS 61B: CS 61A
- CS 61C: CS 61B ,CS170
- CS 70: None
- CS 170: CS 61B, CS 70 ,CS161
- CS 161: CS 61C, CS 70



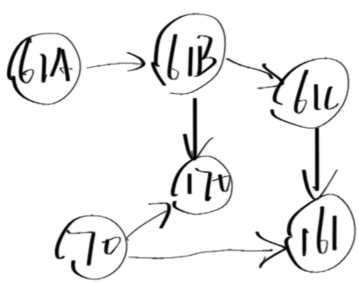
(b) Suppose we added a new prerequisite where the student must take CS 161 before CS 170 and CS 170 before CS 61C. Is there still a valid ordering of classes such that no prerequisites are broken? If no, explain.

161, 170, 61C, 61B, 61A, 70

~~Yes~~

~~70, 61A, 61B, 61C, 170, 161~~

(c) With the original graph, perform a topological sort to find a valid ordering of the 6 classes. Break ties by going to the lower course number first.

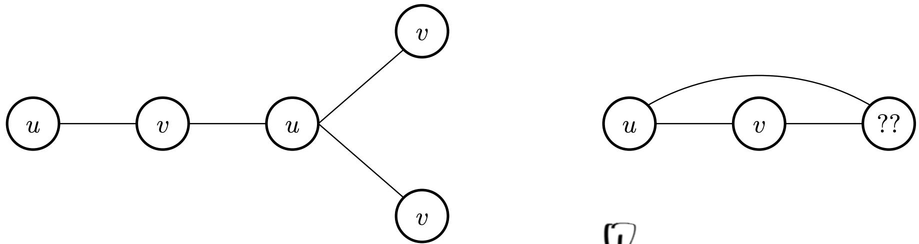


161, 61C, 170, 61B, 61A, 70

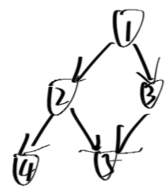
70, 61A, 61B, 170, 61C, 161

4 Graph Algorithm Design

(a) An undirected graph is said to be bipartite if all of its vertices can be divided into two disjoint sets U and V such that every edge connects an item in U to an item in V. For example below, the graph on the left is bipartite, whereas on the graph on the right is not. Provide an algorithm which determines whether or not a graph is bipartite. What is the runtime of your algorithm?
Hint: Can you modify an algorithm we already know (ie. graph traversal)?



BFS



See solution

(b) Consider the following implementation of DFS, which contains a crucial error:

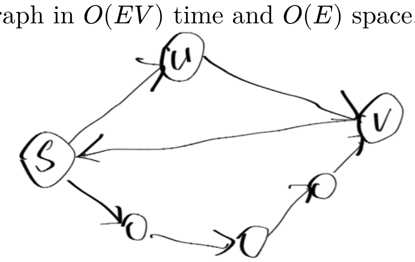
```
create the fringe, which is an empty Stack
push the start vertex onto the fringe and mark it
while the fringe is not empty:
    pop a vertex off the fringe and visit it
    for each neighbor of the vertex:
        if neighbor not marked:
            push neighbor onto the fringe
            mark neighbor
```

mark[ ]

First, identify the bug in this implementation. Then, give an example of a graph where this algorithm may not traverse in DFS order.

X the judgement in while loop is not correct because when it arrive at the node that haven't any neighbors any more it will stop though there also other node which haven't been visited yet. See solution

(c) *Extra:* Provide an algorithm that finds the shortest cycle (in terms of the number of edges used) in a directed graph in $O(EV)$ time and $O(E)$ space, assuming $E > V$.



See solution