CS 460, HOMEWORK 3

INSTRUCTOR: HOA VU

- Each question is worth 25 points. You can work in groups of 2.
- When you are asked to design an algorithm, do the following: a) describe the algorithm, b) explain (or more rigorously prove) why it is correct, and c) provide the running time.
- **DO NOT** look up solutions online. Any violation will be reported. I will be happy to provide some hints or ideas if you get really stuck during class or office hours.
- [Erickson] denotes the book by Jeff Erickson (available for free online at `http://jeffe.cs.illinois.edu/teaching/algorithms/`).
- [DPV] denotes the book by Dasgupta, Papadimitriou, and Vazirani (the required textbook).

(1) **Question 1:** 4.3 of [DPV]. Your algorithm should run in $O(|V|^3)$ time. Hint: use the matrix representation of the graph.

(2) **Question 2:** Given an undirected graph G and an edge uv in G. Design an algorithm that runs in $O(|E|+|V|)$ time that decides if there is a cycle that contains uv.

(3) **Question 3:** Design an algorithm to tell if an input graph can be colored using 3 colors (the constraint is such that no two adjacent vertices have the same color). Hint: use BFS.

(4) **Question 4:** Problem 3 in page 245 of [Erickson]. Hint: Use topological sort.

(5) **Question 5 (Extra credit)** Implement the edit distance algorithm on Leetcode. Follow the following link `https://leetcode.com/problems/edit-distance/`. You need to do the following: 1) provide your code in the homework submission, 2) after your code successfully passed all the test cases when you click on the "submit" button, click on the "details" button on the top left, you will see a report of the performance of your code (which would look similar to the picture below). Include that in your submission.

Question 1: Squares. Design and analyze an algorithm that takes as input an undirected graph G = (V, E) and determines whether G contains a simple cycle (that is, a cycle which doesn't intersect itself) of length four. Its running time should be at most O(|V|^3).
You may assume that the input graph is represented either as an adjacency matrix or with adjacency lists, whichever makes your algorithm simpler.
Hint: use the matrix representation of the graph.
Answer: Let name the vertices from 1 to n = |V| and the n by n matrix called M. If there is an edge between u and v, M[u,v] = 1, otherwise, M[u,v] = 0. Since G has a simple cycle of length four, then it must be the case that having two distinct neighbors in common.
The pseudocode:

```
Counter = 0
 For i = 1, 2, 3, ..., n:
    for j = 1, 2, 3, ..., n:
      For k = 1, 2, 3, ..., n:
         If M[i,k] = M[j,k] = 1:
            Counter += 1
      If counter == 2:
          Return "There is a square!!"
 Return "No squares!"
```

The running time is O(|V|^3).

Question 2: Give an undirected graph G and an edge uv in G. Design an algorithm that runs in O(|E| + |V|) time that decides if there is a cycle that contains uv.
Answer: Let edge e = {u, v} in the undirected graph G. G has a cycle containing edge e if and only if the undirected G without edge e has a path from u to v. Firstly, removing the edge e from the graph G takes

linear times which is editing the adjacency list of nodes. Then using DFS which is explore(G-e, u) and check if v is visited by the DFS. If node v gets visited, return yes, otherwise return no. The DFS algorithm runs in linear time. Hence, the running time of this design algorithm is O(|E| + |V|).

Question 3: Design an algorithm to tell if an input graph can be colored using 2 colors (the constraint is such that no two adjacent vertices have the same color). Hint: use BFS.

- Answer:

The graph G is given.
We start with an arbitrary vertex s, color it Red.
Then run BFS from there in order to color all vertices in the same connected component with s - neighbors (level 1). It is necessary that all neighbors of s are colored Blue, and all neighbors of these neighbors are colored Red (level 2), etc. If at some point we detect a vertex that is colored by 2 different colors, then the graph is NOT 2-colorable.
We can do this by storing current fringe in a queue.
If there are uncolored vertices in G after using BFS, we can choose one of them and repeat the same process.

1. Initialize a queue, an integer color array with all values 0. 0 means not color and not visited, 1 means Red and visited, 2 means Blue and visited.
2. Start from the first note to the end note with using BFS.
3. Add the current node to the queue and color the note with color 1 (Red). Mark it inside the integer array.

4. Repeat until our queue is empty.
5. Remove the top element from the queue and traverse through all the adjacent nodes of the removed node. If the node is not colored, check the color of the removed node. Color the child node in such a

way that color[removed] = 1 means store color[child] = 2 else color[child] = 1. Then add the child to the queue.
6. Make sure to check if the colors of both nodes are different, otherwise the graph is not 2 colorable.
Running Time: O(|V| + |E|)

Question 4: Suppose we are given a directed acyclic graph G with a unique source s and a unique sink t. A vertex $v \notin \{s,t\}$ is called an (s, t) - cut vertex if every path from s to t passes through v, or equivalently, if deleting v makes t unreachable from s. Describe and analyze an algorithm to find every (s, t) - cut vertex in G.

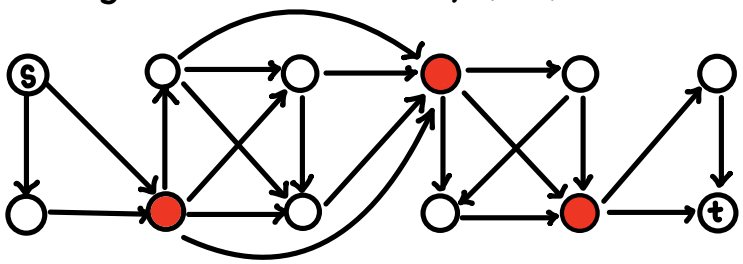


Figure 6.20. A directed acyclic graph with three (s, t) - cut vertices

Hint: Use topological sort

Answer:

The graph G is given.

- First, using DFS starting from s and remove all the vertices which are not reachable from s, then we have a new graph with all nodes are reachable from s. Since using DFS, running time will be $O(|V| + |E|)$.
- Let reverse all the edge direction, we have the new graph H. Also using DFS in graph H starting from t and remove all the vertices which are not reachable from t, then we get a new graph with all nodes are reachable from t. Therefore, all nodes can not reach t in G are also removed. By using DFS, it takes $O(|V| + |E|)$ time.
- We can use topological sort on this acyclic graph which is arrange vertices from left to right such that no edge goes from a vertex to another to the left of it. The total running time is $O(|V|\log|V| + |E|)$. We obviously have s is the first element and t is the last element in this list and the (s, t) cut-vertex meaning no edge over it.

- Design Algorithm:

i = 1

last = 0

while i < n:

    For each edge $v_i \longrightarrow v_j$:

        if j > last:

            last = j

        i = last

Return all the unmark vertices

Since all vertices between v_i and v_j are jumped over so they can't be cut vertices.

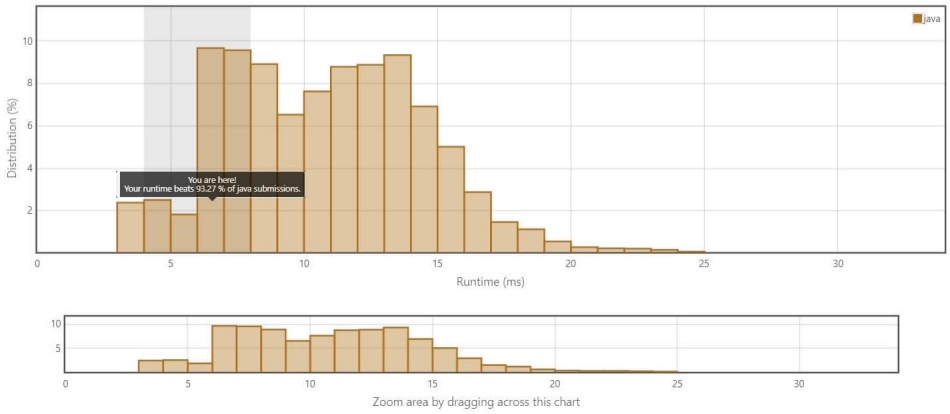
- Running time: O(|V| + |E|)

Question 5: Implement the edit distance algorithm on Leetcode.

Answer:

```java
class Solution {
    public int minDistance(String word1, String word2) {
        int a = word1.length();
        int b = word2.length();

        if(word1 == null && word2 == null)
            return -1;
        if(word1 == null || a == 0)
            return b;
        if(word2 == null || b == 0)
            return a;

        int [][] table = new int [a + 1][b+1];

        for (int i = 1; i <= a; i ++){
            table [i][0] = i;
        }

        for (int j = 1; j <= b; j++){
            table[0][j] = j;
        }

        for (int i = 1; i <= a; i++){
            for (int j = 1; j <= b; j++){
                if (word1.charAt(i-1) == word2.charAt(j -1)){
                    table[i][j] = table[i-1][j-1];
                }
                else {
                    int del = table[i-1][j];
                    int rep = table[i-1][j-1];
                    int ins = table[i][j-1];

                    table[i][j] = Math.min(rep, Math.min(ins,del)) + 1;
                }
            }
        }
        return table[a][b];

    }
}
```

Accepted Solutions Runtime Distribution



Accepted Solutions Memory Distribution