# CS 170 Homework 3

Due **9/20/2021, at 10:00 pm**

## 1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer "Yes", "Yes but anonymously", or "No"

## 2 Updating Labels

You are given a tree $T = (V, E)$ with a designated root node $r$, and a non-negative integer label $l(v)$ for each node $v$. We wish to relabel each vertex $v$, such that $l_{\text{new}}(v)$ is equal to $l(w)$, where $w$ is the $k$th ancestor of $v$ in the tree for $k = l(v)$. We follow the convention that the root node, $r$, is its own parent. Give a linear time algorithm to compute the new label, $l_{\text{new}}(v)$ for each $v$ in $V$

Slightly more formally, the *parent* of any $v \neq r$, is defined to be the node adjacent to $v$ in the path from $r$ to $v$. By convention, $p(r) = r$. For $k > 1$, define $p^k(v) = p^{k-1}(p(v))$ and $p^1(v) = p(v)$ (so $p^k$ is the $k$th ancestor of $v$). Each vertex $v$ of the tree has an associated non-negative integer label $l(v)$. We want to find a linear-time algorithm to update the labels of all vertices in $T$ according to the following rule: $l_{\text{new}}(v) = l(p^{l(v)}(v))$.

Describe the algorithm and give a runtime analysis; no proof of correctness is necessary.

## 3 Disrupting a Network of Spies

Let $G = (V, E)$ denote the "social network" of a group of spies. In other words, $G$ is an undirected graph where each vertex $v \in V$ corresponds to a spy, and we introduce the edge $\{u, v\}$ if spies $u$ and $v$ have had contact with each other. The police would like to determine which spy they should try to capture, to disrupt the coordination of the group of spies as much as possible. More precisely, the goal is to find a single vertex $v \in V$ whose removal from the graph splits the graph into as many different connected components as possible. This problem will walk you through the design of a linear-time algorithm to solve this problem. In other words, the running time will be $O(|V| + |E|)$.

In the following, let $f(v)$ denote the number of connected components in the graph obtained after deleting vertex $v$ from $G$. Also, assume that the initial graph $G$ is connected (before any vertex is deleted), has at least two vertices, and is represented in an adjacency list format.

For each part, prove that your answer is correct (some parts are simple enough that the proof can be a brief justification; others will be more involved).

1. 3037534078 Ye Tian

3037534676 Shenghan Zheng

2.

① Alg Suppose the $T$ is a adjacency matrix $A$

input: label for every node, $T=(V, E)$

output: new label for every node

run DFS and return an array $B$ of nodes

(every node stores old and new labels)

for $n$: 0 to $B.len-1$

find $p^{(i B[n])}_{(B[n])}$ through $B$

$l_{new}(B[n]) = l_{old}(p^{(i B[n])}_{(B[n])})$

② Runtime

DFS $O(|V|+|E|)$

Loop $O(\sum_{v \in G} l(v)) = O(c)$

⇒ total Runtime $O(|V|+|E|)$

3.

(a) $f(r)$ = number of $r$'s children

Correctness:

Assume $r$ has $k$ children $v_1, \cdots, v_k$

then the connected components for any 2 of them is isolated. Otherwise, add $r$ and there will be a circle. Also, there are only $k$ connected components because by deleting $r$, we get $k$ trees with root $v_1, \cdots, v_k$ respectively

(b) $f(r)$ = 1 + number of $r$'s children

Correctness:

After running DFS, there is no cross edge

Also from definition of $v$, there is no back edge.

So there are only forward edges.

Thus, after removing $r$, from a) the connected component contains at most 1 $r$'s children.

Denote $k :=$ number of $r$'s children

Denote connected components containing $r$'s children

$G_1, \cdots, G_k$

"$\Leftarrow$" there is no cross edge and back edge

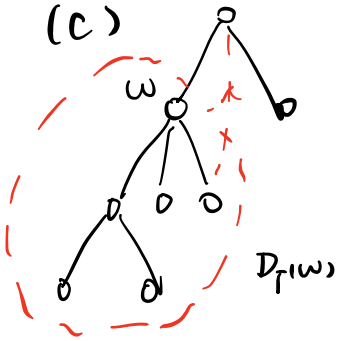
$\Rightarrow$ $G_i$ isn't connected to $G / \{G_1, \cdots, G_k\}$ $(i = 1, \cdots, k)$

$G_i$ is connected because its root is $r$

and there is no edge deleted after removing $r$

proved!

(c)




$$f(v) = k+1 - \sum_{i=1}^{k} \text{Bool}(up_T(w_i) > depth_T(v))$$

first, notice that we don't have cross edges by DFS

and $depth_T(w) \in up_T(w)$

if $depth_T(w) < up_T(w)$, then there must be a back edge containing w's ancestor and one node in $D_T(w)$

① - if $depth_T(w) + 1 = up_T(w)$ and w isn't root

then ∵ no cross edge, the graph is undirected

∴ $up_T(w)$ which stands for min depth comes from w's parent node v. In this case, $D_T(w)$ forms a connected component if v is deleted.

② - if $depth_T(w) + 1 < up_T(w)$

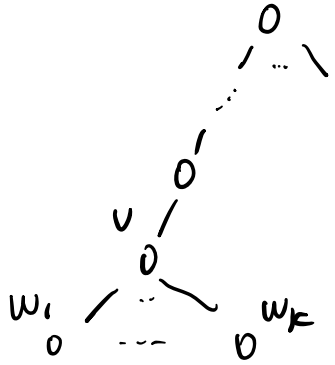
$up_T(w)$ comes from w's ancestor excluding parent.

In this case, delete v will not make $D_T(w)$ a connected component

Denote the subtree $T'$ with root v in $T$

If $\forall v \in D_T(v)$ satisfies (b), we get the max number of $f(v)$ which is $k+1$. For k children, if ②

happens, it decreases 1

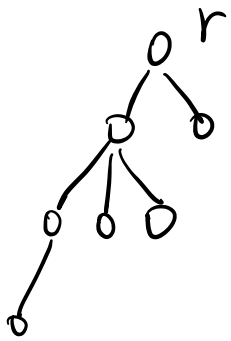



denote $Bool(x)$ $\begin{cases} 1 & x \text{ is true} \\ 0 & x \text{ is false} \end{cases}$

$\Rightarrow$

$$f(v) = k+1 - \sum_{i=1}^{k} Bool(up_T(w_i) > depth_T(v))$$

(d)




① Alg

input : $G, T$

output: $up_T(v)$ for $\forall\ v \in V$

Start from root in $T$, traverse $T$ and label each node's depth and estimated value of $up_T$

$up_T(r) = 0$ $\quad up_T(v) = depth(v\text{'s parent})$

go through $T$. When at node $u$, if there is $(u,v)$ in $G$ but not in $T$, Then $up_T(v) = \min(depth(u), up_T(v))$

② Correctness

$T$ is returned by DFS

denote $v$ to be node $u$'s parent

Then $up_T(u)$ is depth$(v)$ unless

its decendant has backedge to its ancestor

Because $T$ is DFS $\Rightarrow$ when at node $u$

if there is $(u,v)$ in $G$ but not in $T$ then $v$ must be $u$'s decendant ($(u,v)$ is back edge). Renew $up_T(u)$.

Thus, When we finish traversing $T$, each $up_T(u)$ is correct.

③ Runtime

Label depth and set estimation for $up_T$

$O(|V|+|E|)$

Traverse $T$ and find back edge, then renew $up_T$

$O(|V|+|E|+|E|) \Rightarrow$ total runtime $= O(2|V|+3|E|)$

(e)

① Alg

run DFS on $G$, get $T$ and label their depth

Compute each $up_T(v)$ using (d)

Calculate $f(v)$ through equation in (c)

② Correctness

it's correct from correctness in (c)

③ runtime

run DFS $O(|V|+|E|)$

label depth : in the loop, we add 1 to depth when we push element in stack and minus 1 when pop a element $O(|V|)$

Compute $up_T(v)$ given $G.T$: $O(|V|+|E|)$

Calculate $f(v)$ using (c): for every node we need to find their children and do compare

$O(\sum_{v \in G} \text{children of } v \times 2) = O(|V|)$

$\Rightarrow$ total runtime $= O(|V| + |E|)$

which is linear

(a) Let $T$ be a tree produced by running DFS on $G$ with root $r \in V$. (In particular, $T = (V, E_T)$ is a spanning tree of $G$: a tree which connects all the nodes of $G$ using a subset of its edges.) Given $T$, find an efficient way to calculate $f(r)$.

(b) Let $v \in V$ be some vertex that is not the root of $T$ (i.e., $v \neq r$). Suppose further that there is no edge $\{u, w\}$ such that $u$ is a descendant of $v$ and $w$ is an ancestor of $v$. How could you calculate $f(v)$ from $T$ in an efficient way?

(c) For $w \in V$, let $D_T(w)$ be the set of descendants of $w$ in $T$ including $w$ itself. For a set $S \subseteq V$, let $N_G(S)$ be the set of *neighbors* of $S$ in $G$, i.e. $N_G(S) = \{y \in V : \exists x \in S \text{ s.t. } \{x, y\} \in E\}$. We define $\mathsf{up}_T(w) := \min_{y \in N_G(D_T(w))} \mathsf{depth}_T(y)$, i.e. the smallest depth in $T$ of any neighbor in $G$ of any descendant of $w$ in $T$.

Now suppose $v$ is an arbitrary non-root node in $T$, with children $w_1, \ldots, w_k$. Describe how to compute $f(v)$ as a function of $k$, $\mathsf{up}_T(w_1), \ldots, \mathsf{up}_T(w_k)$, and $\mathsf{depth}_T(v)$.

Hint: Think about what happened in part (b); think about what changes when we can have non-tree edges that go up from one of $v$'s descendants to one of $v$'s ancestors, and think about how you can detect it from the information provided.

(d) Design an algorithm which, on input $G, T$, computes $\mathsf{up}_T(v)$ for all vertices $v \in V$, in linear time.

(e) Given $G$, describe how to compute $f(v)$ for all vertices $v \in V$, in linear time.

## 4 Where's the graph?

Each of the following problems can be solved with techniques taught in lecture. Construct a simple directed graph and write an algorithm for each problem by black-boxing algorithms taught in lecture and in the textbook.

(a) Sarah wants to do an extra credit problem for her math class. She is given three numbers: 1, $x$, and $y$. Starting from $x$, she needs to find the shortest sequence of additions, subtractions, and divisions (only possible when the number is divisble by $y$) using 1 and $y$ to get to 2021. If there are multiple sequences with the shortest length, return any one of them. She can use 1 and $y$ multiple times. Give an algorithm that Sarah can query to get this sequence of arithmetic operations.

(b) There are $n$ different species of Gem Berry, all descended from the original Stone Berry. For any species of Gem Berry, Emily knows all of the species directly descended from it. Emily wants to write a program. There would be two inputs to her program: $a$ and $b$, which represent two different species of Gem Berries. Her program will then output one of three options in constant time (the time complexity cannot rely on $n$):

(1) $a$ is descended from $b$.

(2) $b$ is descended from $a$.

(3) $a$ and $b$ share a common ancestor, but neither are descended from each other.

Give an algorithm that Emily's program could use to do this.

$2021 = 43 \times 47$

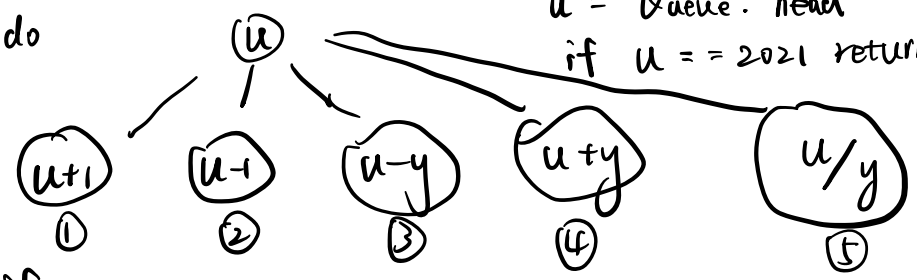
4.

(a) Start with node $x$ Queue.push(x) depth(x) = 0

while ! Queue.empty

do u = Queue.head

if u == 2021 return depth(u)




if u/y isn't integer fill this node

depth(①) :=
depth(②) :=
depth(③) :=
depth(④) :=
} depth(u)+1

for i := 1 to 4 Queue.push(ⓘ)

else

depth(①) :=
depth(②) :=
depth(③) :=
depth(④) :=
depth(⑤) :=
} depth(u)+1

for i := 1 to 5 Queue.push(ⓘ)

(b) Construct the graph which is composed of different species of Gem Berry.

(1) $\Leftrightarrow$ $(b,a) \in E$

(2) $\Leftrightarrow$ $(a,b) \in E$

$G_1$ is a DAG because there is no edge from descendant to ancestor. $G_1$ is a tree because they share the same ancestor and species other than root comes from another.

Thus, if (1), (2) aren't true then (3) is true

Say we have $n$ vertices, $v_1, \cdots, v_n$

Construct a matrix $A \in R^{n \times n}$

$$A_{ij} = \begin{cases} 0 & (v_i, v_j) \in E \quad (1) \\ 1 & (v_j, v_i) \in E \quad (2) \\ 2 & \text{else} \quad (3) \end{cases}$$

To fill the matrix, run 2 DFS with $v_i$, $v_j$ as root respectively

① $v_i$ as root: if we reach $v_j$ in the same

connected component with $v_i$, fill $A_{ij}$ with 0

② $v_j$ as root: if we reach $v_i$ in the same connected component with $v_j$, fill $A_{ij}$ with 1

if ①, ② don't happen, fill $A_{ij}$ with 2

Now for $\forall\ v_i, v_j \in V \quad i \neq j$

check $A_{ij}$ and return their relation

runtime: $\Theta(1)$

# 5 2-SAT

In the 2SAT problem, you are given a set of clauses, where each clause is the disjunction (OR) of two literals (a literal is a Boolean variable or the negation of a Boolean variable). You are looking for a way to assign a value true or false to each of the variables so that all clauses are satisfied – that is, there is at least one true literal in each clause. For example, here's an instance of 2SAT:

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

This instance has a satisfying assignment: set $x_1$, $x_2$, $x_3$, and $x_4$ to `true, false, false, and true`, respectively.

The purpose of this problem is to lead you to a way of solving 2SAT efficiently by reducing it to the problem of finding the strongly connected components of a directed graph. Given an instance $I$ of 2SAT with $n$ variables and $m$ clauses, construct a directed graph $G_I = (V, E)$ as follows.

- $G_I$ has $2n$ nodes, one for each variable and its negation.
- $G_I$ has $2m$ edges: for each clause $(\alpha \vee \beta)$ of $I$ (where $\alpha$, $\beta$ are literals), $G_I$ has an edge from from the negation of $\alpha$ to $\beta$, and one from the negation of $\beta$ to $\alpha$.

Note that the clause $(\alpha \vee \beta)$ is equivalent to either of the implications $\overline{\alpha} \implies \beta$ or $\overline{\beta} \implies \alpha$. In this sense, $G_I$ records all implications in $I$.

(a) Show that if $G_I$ has a strongly connected component containing both $x$ and $\overline{x}$ for some variable $x$, then $I$ has no satisfying assignment.

(b) Now show the converse of (a): namely, that if none of $G_I$'s strongly connected components contain both a literal and its negation, then the instance $I$ must be satisfiable. (*Hint*: Assign values to the variables as follows: repeatedly pick a sink strongly connected component of $G_I$ . Assign value `true` to all literals in the sink, assign `false` to their negations, and delete all of these. Show that this ends up discovering a satisfying assignment.)

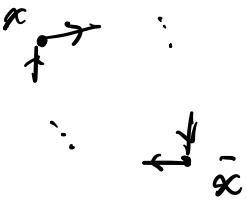
(c) Conclude that there is a linear-time algorithm for solving 2SAT.

# 6 Introduction To Coding Homework

In this iteration of CS170, we will be introducing coding problems that will be conducted on an Online Judge platform in the style of programming contests (e.g Leetcode, Codeforces). This will only be a small part of the homework (roughly 1 problem on only 3 of the homeworks, so 3 total in the semester).

We will be using QDUOJ, an open-source online judge platform, to test your solutions. To access this platform, visit hellfire.ocf.berkeley.edu).

5.

(a) 


Let $I$ be satisfiable $(x \cdot y) \in G$

Then, if $x$ is true, then $y$ is true

$\because (x \cdot y) \in G \quad \therefore I$ contains $\bar{x} \vee y$ or $x \vee \bar{y}$

WLOG, let's say $I$ contains $\bar{x} \vee y$

if $x$ = true $\Rightarrow \bar{x}$ = false $y$ = true

and there is a path in $G$ s.t connects $x$ to $\bar{x}$

denote it $x\, x_1 \cdots x_k \bar{x}$ which shows

$x \Rightarrow x_1 \Rightarrow - \Rightarrow x_k \Rightarrow \bar{x}$

$\because x$ is true $\therefore x_1$ is true $\cdots \therefore \bar{x}$ is true

Contradiction !

if $\bar{x}$ = true, likewise

$\Rightarrow$ $I$ has no satisfying assignment

(b) Notice that $I$ is satisfiable if each clause in $I$ is true

$(\alpha \vee \beta) \iff \bar{\alpha} \Rightarrow \beta$ or $\bar{\beta} \Rightarrow \alpha$

use algorithm in class to find a sink strongly connected component of $G_1$.

( find by DFS, also notice that SCC is a node in larger DAG )

Denote it $S$

Assign value true to all literals

Notice that there is no edge from $S$ to outside

$\Rightarrow$ no edge from outside to $\bar{S}$ which means $\bar{S}$ is a source SCC

$\bar{S}$ = false

remove $S$ and $\bar{S}$

$\because$ none of $G_i$'s SCC contains both a literal and its negation

$\therefore$ we can repeat until we get a empty graph

proved!

(c)

① Alg

Use algorithm in class to find one node in sink scc. Set literals in them to be true peel it off (which is in reverse topological order) peel its reverse off and do above peperatedly We end up with empty set and get satisfiable I with a truth table. Otherwise I isn't satisfiable

② Correctness

discussed in class

③ Runtime

O(m+n)

## Rules

(a) Any rules described on the Syllabus are in effect.

(b) You may refer to online resources for minor implementation details, such as searching up documentation for a heap class. You may not consult any resources that give away the main idea or implementation of the algorithm.

(c) Similar to the written homework, you may not collaborate on the implementation details and must submit your own code.

(d) You may not exploit bugs on the platform, access otherwise not publicly visible test cases, or submit malicious code.

## Some Details

(a) Languages
For fairness, we will only allow Python 3.

(b) Input & Output
We will use stdin and stdout for input and output. The format will be given and you do not need to consider invalid inputs.

(c) If you receive a "Time Limit Exceeded!" message, it is most likely an issue with the efficiency of your implementation, as all time limits are tested on the same hardware.

## Action items for this homework

(a) Log in to the contest system and register **with your bMail address as it appears on Gradescope**. You can use any nickname you wish as long as the email address matches your bMail address on Gradescope. Note that this username may be displayed to other students if you wish to opt-in to the leaderboard.

(b) Make a test submission to the problem below to familiarize yourself with the platform.

### A+B Problem

Access the problem at: https://hellfire.ocf.berkeley.edu/contest. Choose the appropriate contest (Homework 3 Coding Problems, or the DSP 150% or DSP 200% alternative), navigate to "Problems" on the toolbar on the right and choose the **standard** version of the A+B problem. Feel free to ignore the Contest version of the problem.
Your task is to make any submission. You will not be penalized if your solution is wrong, but you have to submit some code for this problem. In the case if you encounter a bug with the online contest system, for example, you are having problems with registration or problems with submitting, please, report the bug to this form: https://forms.gle/17aykrtBas5CmiV26.

### Description

To practice basic input/output and to familiarize with this OJ platform.
Input $n$, then on the following $n$ lines two nonnegative integers $A$ and $B$, output the sum of these two nonnegative integers $A + B$.

### Input

The first line will be one positive integer The following Of 10% of the test cases $0 < A, B \leq 10$, and it is guaranteed that $n > 0 > 200$.
Of 100% of the test cases $0 < A, B \leq 200$, and it is guaranteed that $n > 0 > 200$.

### Output

On each line, print out the sum of the two integers given.

### Sample Input

3
114 51
4 19
198 10

### Sample Output

165
23
208

## Submission

Make a screenshot of your submission and attach it to your homework PDF, where you can submit it on gradescope with the rest of the problems. In the future, we intend on building a gradescope integration, so you may not need to do this for future homeworks.