CS345 : Algorithms II
Semester I, 2016-17, CSE, IIT Kanpur

Assignment 3

Deadline : 5:00 PM, Sunday, 4th September 2016

Important Guidelines:

- It is only through the assignments that one learns the most about the algorithms and data structures. You are advised to refrain from searching for a solution on the net or from a notebook or from other fellow students. Before cheating the instructor, you are cheating yourself. The onus of learning from a course lies first on you and then on the quality of teaching of the instructor. So act wisely while working on this assignment
- There are two exercises in this assignments. Each exercise has two problems- one *easy* and one *difficult*. Submit exactly one problem per exercise. It will be better if a student submits a correct solution of an easy problem that he/she arrived on his/her own instead of a solution of the difficult problem obtained by hints and help from a friend. Do not try to be so greedy :-).
- Do not feel uncomfortable on seeing the details of the problem 1.2 These details are to motivate you through some real life application. You must read it with a lot of interest and enthusiasm (this is also a part of the assignment). The exact problem definition is given at the end.
- On the last page, there is an optional problem. This is for those students who love algorithms without expectations. This problem is just for fun and won't be graded and hence you are not supposed to submit it.

1 DAG

Attempt exactly one of the two problems.

1.1 Searching for a special path

(*marks=25*)

Let $G = (V, E)$ be a directed acyclic graph on n vertices and m edges. Let $x_1, x_2, \cdots x_k$ be a sequence of k vertices. There is a source vertex s and a destination vertex t. Our aim is to determine if there exists any path from s to t which looks like:

$$s \rightsquigarrow x_1 \rightsquigarrow x_2 \rightsquigarrow \cdots \; x_k \rightsquigarrow t$$

Design an $O(m + n)$ time algorithm to do this task.

1.2 A real life application of Directed Acyclic Graphs

(*marks=40)*

Control Flow Graphs(CFG) are representations of the control flow of a program during its execution. A path profile describes how many times a path is executed. But, there may be infinitely many paths in CFG (if the program contains some loop). However, researchers found a technique to achieve path profiling by converting a CFG to a directed acyclic graph (DAG). Observe that total number of execution paths in a DAG is finite. A CFG is converted to the corresponding DAG(with the start node s and the end node t) as follows: a dummy *root* node and a dummy *exit* node are added to existing nodes, edges $root \rightarrow s$ and $t \rightarrow exit$ are added to existing edges list. Then, each back-edge $u \rightarrow v$ in the CFG is replaced by edges $root \rightarrow u$ and $v \rightarrow exit$ are added. For example, the program mentioned below prints sum of `n` elements stored in an array, `arr`. The corresponding CFG and DAG are shown in Figure 1.

```
//Print the sum of n elements in an array arr[0..n-1]
sum=i=0;
while(i<n)
    sum += arr[i++];
print sum;
```

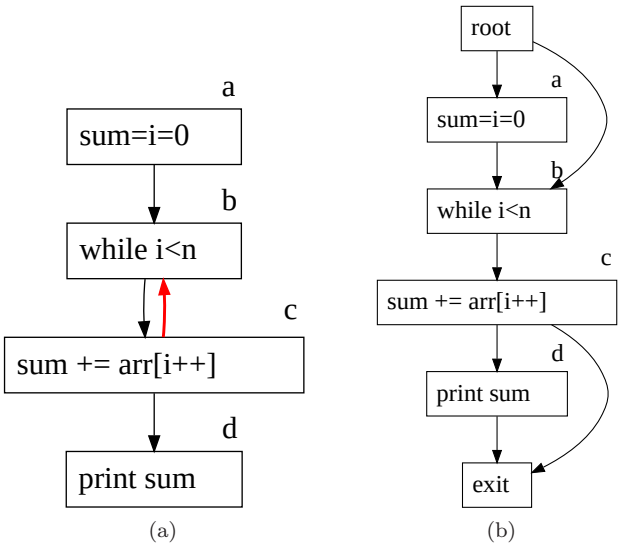

Figure 1: (a) The weighted directed acyclic graph (b) *pathid*s for all paths from s to t

The basic principle of a path profiler is: each edge in the DAG is assigned an integral weight such that sum of edge-weights along any path from *root* to *exit* is unique. The sum of edge-weights of the edges

along a path is called *pathid* of that path. A counter is initialized to 0, it is incremented by edge-weights of edges as they are traversed. At the end of a path p, counter stores the *pathid* of p. It is crucial that each path get unique pathid, otherwise we can not resolve which path get executed. If there are total of N such paths, an array of size N can be employed to keep track of the count of the paths which are executed.

A profiler is a program analysis tool. *Program analysis tools are extremely important for understanding program behavior. Computer architects need such tools to evaluate how well programs will perform on new architectures. Software writers need tools to analyze their programs and identify critical pieces of code. Compiler writers often use such tools to find out how well their instruction scheduling or branch prediction algorithm is performing or to provide input for profile-driven optimizations.*

Problem Definition: Given a directed acyclic graph $G = (V, E)$ with the vertex s as the root vertex and the vertex t as the exit vertex, *pathid* of a path p from s to t is defined as the sum of weights of edges along p. The problem is to assign integral weights to the edges in G such that all paths from s to t get unique *pathid*s from 0 to $N - 1$, where N is the total number of paths from s to t.

2 DFS in directed graph

Attempt exactly one of the following problems.

2.1 Computing least label vertex

(*marks=25*)

We discussed about SCC graph during Lecture 13 (You are advised to revisit it before attempting this problem). It follows from the $O(m+n)$ time algorithm for strongly connected components that we can compute SCC graph of a given directed graph in $O(m+n)$ time. Let $G = (V, E)$ be a directed graph on n vertices and m edges. Each vertex v has a label $L(v)$ which is a real number. Let $minL(v)$ denote the label of the smallest label vertex reachable from v. Our aim is to compute $minL(v)$ for all $v \in V$.

1. Give suitable arguments to show that the above problem can be solved if we solve it on the corresponding SCC graph.

2. Give an $O(m+n)$ time algorithm to solve the problem on SCC.

Remark: Note that the above problem can also be solved in a single DFS traversal of the graph. All you will have to do will be to suitably augment the existing code of DFS traversal. Interestingly, there is an alternate algorithm for this problem that does not require any knowledge of DFS in directed graphs or any algorithm for DAG. It is based on carrying out any traversal (even BFS) on the reversed graph.

2.2 At least one path

(*marks=35*)

Let $G = (V, E)$ be a directed graph on n vertices and m edges. Someone has claimed that for each pair of vertices $u, v \in V$, either there is a path from u to v or there is a path from v to u. Note that here "or" does not mean exclusive-or. So there can be a path from u to v as well a path from v to u. Design an $O(m+n)$ time algorithm to determine if the claim is valid or not.

3 OPTIONAL - Only for fun

Let $G = (V, E)$ be a directed graph on n vertices and m edges. Further, you are told that there exists some vertex which has a path to every other vertex. However, you don't know that vertex. Design an $O(m + n)$ time algorithm to determine if G is unique-path graph. You must explicitly state a theorem that is being exploited by you in the design of this algorithm. You must also provide a proof of this theorem.

Hint: Once you are able to locate the *desired* vertex, perform a DFS traversal from it and make careful observations about it in terms of DFS traversal.