CS345 : Algorithms II
Semester I, 2020-21, CSE, IIT Kanpur

Assignment 4

Deadline : 11:55 PM, 11st October 2020.

Most 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. Remember - **Before cheating the instructor, you are cheating yourself**. The onus of learning from a course lies first on you. So act wisely while working on this assignment.
- **Grading policy:**
  There will be no penalty for submission based on the time of submission as long as the submission is before the deadline.
- Refrain from collaborating with the students of other groups. If any evidence is found that confirms copying, the penalty will be very harsh. Refer to the website at the link: https://cse.iitk.ac.in/pages/AntiCheatingPolicy.html regarding the departmental policy on cheating.

General guidelines

1. There are two problems in this assignment: Difficult and Easy. The difficult one carries 100 marks, the easy one carries 65 marks. Attempt **only** one of them.
2. You are strongly discouraged to submit the scanned copy of a handwritten solution. Instead, you should prepare your answer using any text processing software (LaTex, Microsoft word, ...). The final submission should be a single pdf file.
3. You need to justify any claim that you make during the analysis of the algorithm. But you must be formal, concise, and precise. You may use the results proved in the class. But, if you wish to use any homework problem in your solution, you must provide its solution as well.
4. If you are asked to design an algorithm, you may state the algorithm either in plain English or a pseudocode. But it must be formal, complete, unambiguous, and easy to read. You must not submit any code (in C++ or C, python, ...).
5. **Naming the file**:
   The submission file has to be given a name that reflects the information about the assignment number, version attempted (difficult/moderate/esay), and the roll numbers of the 2 students of the group. For example, you should name the file as **D_i_Rollnumber1_Rollnumber2.pdf** if you are submitting the solution for the difficult problem of the ith assignment. In a similar manner, the name should be **E_i_Rollnumber1_Rollnumber2.pdf** if you are submitting the solution for the easy problem of the ith assignment.
6. **Each student of a group** has to upload the submission file separately unlike in the past when only one submission file per group had to be uploaded. Be careful during the submission of an assignment. Once submitted, it can not be re-submitted.
7. Deadline is strict. Make sure you upload the assignment well in time to avoid last minute rush.
8. Contact TA at the email address: **spandey@cse.iitk.ac.in** for all queries related to the submission of this assignment. Avoid sending any such queries to the instructor.

Difficult

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 such 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.

Nearly 25 years ago, a program analysis tool was designed. At the core, it solves an algorithmic problem on a directed acyclic graph (DAG). This result appeared in one of the most prestigious conference in systems area of computer science, and it has been a milestone in the area of program analysis. You will be solving the algorithmic problem lying at the core of this result, which is neatly described as follows.

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. Design the most efficient algorithm to accomplish this task. Full marks will be given only if your algorithm achieves the best time complexity known for this problem. You must also provide a concise and formal proof of correctness of the algorithm.

Easy

Searching for a special path in a DAG

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 from V. 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. You must also provide a concise and formal proof of correctness of the algorithm.

Sketch of Solution and Grading Policy for the Difficult Problem

Objective

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

Algorithm Design [40pts]

- Compute a topological numbering for the vertices of the given DAG and store it in an array T such that $T[u]$ stores the topological number assigned to vertex u.
- Let $P[u]$ denotes the total number of paths from u to t. Notice that $P[t] = 1$. $P[u]$ for all vertices other than t can be computed efficiently by processing vertices in the decreasing order of their topological numbering. This order ensure that while processing u we would already have computed value of $P[v]$ for each v such that $T[v] > T[u]$. Thus, $P[u] = \sum_{v:(u,v)\in E} P[v]$.
- We now show how we can assign weights to the outgoing edges of a vertex u. Let $T[u] = i$ and it has outgoing edges to vertices $w_1, w_2, \ldots, w_k$. Since $T[w_j] > i$, where $j \in [k]$, assume that all such vertices are processed and have correct $P[w_j]$ values and weight assigned to their outgoing edges such that *pathids* of all path from w_j to t are unique and in range $[0, P[w_j] - 1]$.
- Following steps describe strategy for assigning weights to edges of type (u, w_j) such that *pathids* of all path from u to t will be unique and in range $[0, P[u] - 1]$.
  - For edge (u, w_1), we assign weight 0. This means that we have discovered $P[w_1]$ paths from u to t passing through w_1 and now they will have *pathids* in range $[0, P[w_1] - 1]$.
  - For *pathids* to be unique, any *pathid* of remanining paths must be greater than $P[w_1] - 1$, therefore for edge (u, w_2) we will assign weight $P[w_1]$ which means that now *pathids* of paths from u to t passing through w_2 will be in the range $[P[w_1], P[w_1] + P[w_2] - 1]$. Combining them we can see that we have found $P[w_1] + P[w_2]$ different paths from u to t and they will have *pathids* in the range $[0, P[w_1] + P[w_2] - 1]$.
  - In general, we can say that weight assigned to edge (u, w_j) will be equal to $\sum_{l=1}^{j-1} P[w_l]$ if $j > 1$ and 0 if $j = 1$. And when we finish processing all edge (u, w_j), then $P[u] = \sum_{j=1}^{k} P[w_j]$ and all *pathids* will be unique and will be in range $[0, P[u] - 1]$.

Correctness Proof [50pts]

It suffices if we can establish the vality of the following claim.
Claim: While we are processing vertex u, weights are assigned to its outgoing edges such that the *pathids* of all the paths from u to t are unique and in the range $[0, P[u] - 1]$.

An easy way to establish the above claim is by induction on the vertices in the decreasing order of their topological numbering.

Time Complexity [10pts]

Let n be the number of vertices & m be the number of edges in the given DAG.

- Topological Ordering of the DAG will take $O(m+n)$ time.
- Then we are traversing right to left through every vertex once and doing constant amount of work for each of its outgoing edge, i.e, $O(outDeg(u))$ time for each vertex. Hence, this step also takes just $O(m+n)$ time.
- Hence total time complexity of the algorithm is $\mathbf{O(m+n)}$.

For claiming the above time complexity, one must state the assumption that the number of paths from vertex s to t is polynomial in n. The reason being that while analyzing the time complexity of an algorithm, we work with **Word RAM** model of computation which allows us to assume that any arithmetic operation involving numbers that can be stored in $O(1)$ words takes $O(1)$ time. For a graph on n vertices and m edges, one word stores $\Theta(\log n)$ bits.

In order to realize the importance of this point, suppose the number of paths from any vertex to t is exponential in n. Then it will take $O(n)$ bits to represent the number of the paths. Hence, our assumption that it takes $O(1)$ time for any operations (e.g. addition operation during the algorithm) involving these numbers will fail. So, although the number of operations are $O(m+n)$, it will be wrong to infer that the time complexity of the algorithm is $O(m+n)$ in this case.
[5 marks will be deducted for not stating the assumption that the number of paths from s to t is polynomial in n]