

CS340 Analysis of Algorithms

Fall 2026

Practice Set: 2

URL: <http://cs.brynmawr.edu/cs340>

1. Problem 2.7 The point of this problem is to determine how long a “song script” would be if you do not write down the repeats over and over again. You can think of this as the “encoding” of the full song. The length of this encoding is your $f(n)$.

For example, suppose the complete lyrics have a total of k lines. Some (not very clever) encoding might look like this:

```
Function DumbScript()
|   lines[1] = lyrics line 1
|   ...
|   lines[k] = lyrics line k
|   for  $i = 1$  to  $k$  do
|   |   output lines[ $i$ ]
|   end
```

Length analysis of the above encoding: recall that the lines in the lyrics are bounded by some constant c , thus the length of the complete lyrics is kc . This encoding clearly has the same number of lines, plus the 3 additional lines in the **for** loop. The initialization lines each has length $c_1 + c$, where c_1 is another constant that denotes the length of “**lines[*] =**”. Note $c_1 < c$. The length of the lines in the **for** loop is constant, and shorter than the original lyric lines and is thus $< c$. Therefore the length of the encoding is $f(n) < (k(c_1 + c) + 3c) = kc + kc_1 + 3c$, where $n = kc$. Thus $f(n)$ is $O(n)$.

Note that storage is irrelevant. We are NOT doing a time analysis. The script above used array notations, simply because it’s the easiest way for CS folks to refer to things that have a natural ordering. I could have used k variables and called them **line1**, **line2**, ..., **linek** instead and the length would still be $O(n)$. You can also interpret “output” as “sing”, if you’d like. Remember the context of the “song script” for the choir. The purpose of the encoding is to give instructions on how to reproduce the original lyrics. Big-O analysis is a relationship between two functions. In your typical time analysis, it’s the size of the input versus the time complexity of your algorithm. Here, it is the length of the original lyrics versus the length of the encoding.

Please include description of your encoding scheme and pseudo code - it should be clear that your encoding must be length-reducing, i.e. $O(n)$ is not acceptable. Focus on the length analysis. You may skip proof of correctness.

2. Problem 2.8a Full write-up. 2.8b is tricky, but feel free to give it a try if you want a challenge.
3. Problem 3.2 Full write-up. Note that this is not a decision problem. If there is a cycle, you must also devise a way to output it, either as a list of ordered vertices or edges.
4. Problem 3.10 Full write-up. You may assume that G is unweighted.
5. Full write-up. You are given a DAG (directed acyclic graph) $G = (V, E)$ in which each edge (u, v) has a label which is either 0 or 1. An alternating path is defined to be a path containing at

least one edge, such that the edges along the path alternate in label between 0 and 1. The path may begin with a 0-label edge or a 1-label edge. Note: The empty path is not an alternating path, and a path with a single edge is always an alternating path. Paths need not be of maximal length. For each vertex $u \in V$, let $alt(u)$ denote the number of alternating paths that start at u . For example, in Fig. 1, $alt(b) = 5$ because of the paths (b, c) , (b, c, e) , (b, d) , (b, d, e) , (b, e) .

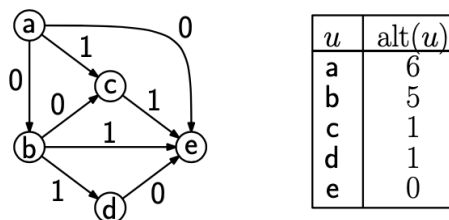


Figure 1: Alternating paths in a DAG

Present an algorithm which, given a DAG $G = (V, E)$ with 0 – 1 edge labels, computes $alt(u)$ for all $u \in V$. Ideally, your algorithm should run in $O(n + m)$ time. Note that you should work out a run of your algorithm on the example given, and show that all the correct values of $alt(u)$ are eventually obtained.