

# CS624: Analysis of Algorithms

## Assignment 4

Due: Thursday, Nov. 7, 2024

**Topics:** Dynamic Programming, Greedy algorithms

1. Let  $T$  be a rooted tree.  $T$  is not necessarily a binary tree. That is, each node may have any finite number of children. We are going to consider a method of visiting each node of the tree. The method works like this:

At Step 1, we visit the root. The root is then marked as “visited”.

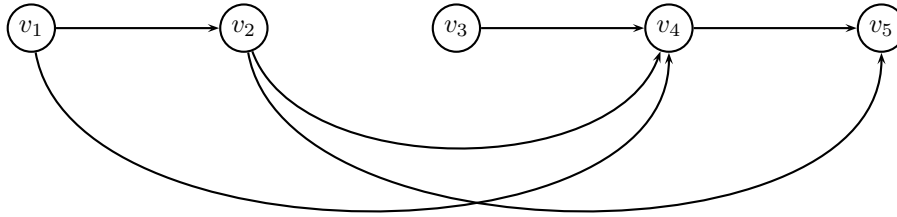
At Step  $n$ , all nodes that have already been marked “visited” can visit at most one of their children. Of course if all the children of a node have already been visited, there is nothing for that node to do. But other nodes may still do something at that step. (And so in particular, more than one node might be visited in a single step of the process.)

The question is this: what is the minimum number of steps necessary to visit every node in the tree?

- (a) Draw a picture of a tree in which more than one node is visited at some step of this process, and show why that happens.
  - (b) Show that this problem contains optimal substructure. Please be careful when you do this. You have to state precisely and clearly what the substructure is and show why it is optimal.
  - (c) Use that result to write a recursive algorithm to solve this problem.
  - (d) Turn that algorithm into a dynamic programming algorithm.
  - (e) What is the cost of the algorithm? (By that, I mean, “What is the cost of the algorithm that computes the minimum cost?”, not “What is the minimum cost?”. In doing this problem, please write as little pseudo-code as possible. I would much rather read something that is clearly explained using ordinary language.
2. Determine an LCS of  $\langle 1, 0, 0, 1, 0, 1, 0, 1 \rangle$  and  $\langle 0, 1, 0, 1, 1, 0, 1, 1, 0 \rangle$ . Of course you could probably solve this problem easily enough just by looking at it. What I want you to do is to explicitly use the algorithm presented in class. Write out your derivation neatly, including the table.
  3. Exercise 3.2 in the Lecture 8 handout (on page 12).
  4. Exercise 3.3 in the Lecture 8 handout (on page 12).
  5. Let  $G = (V, E)$  be a directed graph with vertices  $V = \{v_1, v_2, \dots, v_n\}$ . The set  $E$  of edges has the following property:
    - Each edge goes from a vertex of lower index to a vertex of higher index. For instance, there might be an edge  $(v_1, v_7)$  from  $v_1$  to  $v_7$ . But there definitely will *not* be an edge  $(v_7, v_1)$  from  $v_7$  to  $v_1$ .

- Each vertex except  $v_n$  has at least one edge leaving it. That is, for every vertex  $v_i$  ( $1 \leq i \leq n - 1$ ), there is at least one edge of the form  $(v_i, v_j)$  (of course with  $i < j$ ).

Here is an example of a graph with this property:



It is called a DAG (directed acyclic graph). We will discuss them later on in the course. By a *path* we mean as usual a sequence of vertices  $\{v_{i_1}, v_{i_2}, \dots, v_{i_k}\}$ ,

such that for each successive pair in the sequence there is an edge from the first to the second. (That is, there is an edge from  $v_{i_1}$  to  $v_{i_2}$ , another edge from  $v_{i_2}$  to  $v_{i_3}$ , and so on.)

The *length* of a path is just the number of edges in it. Equivalently, it is 1 less than the number of vertices in it.

We want to find an algorithm to solve the following problem, given such a graph:

Find the length of the longest path that begins at  $v_1$  and ends at  $v_n$ .

Here is a “greedy” algorithm that it is natural to think of at first:

---

**Algorithm 1** FindPath

---

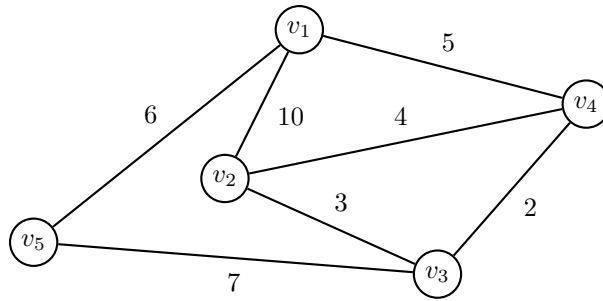
```

1:  $w \leftarrow v_1$   $w$  is the current node we are considering.
2:  $L \leftarrow 0$   $L$  will hold the greatest length of any path so far.
3: while there is an edge out of  $w$  do
4:   Choose the edge  $(w, v_j)$  for which  $j$  is as small as possible.
5:    $w \leftarrow v_j$ 
6:    $L \leftarrow L + 1$ 
7: end while
8: return  $L$ 

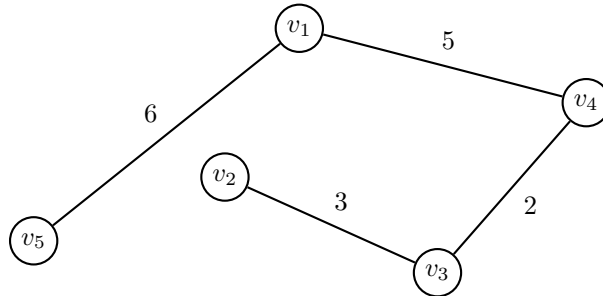
```

---

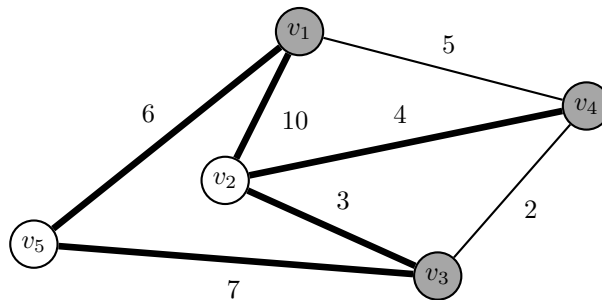
- Show that this algorithm gives the correct answer for the graph drawn above.
  - Show that this algorithm does not give the correct answer in general.
  - One could of course solve the problem in general by considering all possible paths from  $v_1$  to  $v_n$ . Why is this not an efficient algorithm?
  - Construct an efficient algorithm that does give the correct answer in general. (Hint: use dynamic programming. You have to prove that the algorithm is correct.)
  - How efficient is your algorithm?
6. The minimum spanning tree problem is defined on undirected, weighted graphs as follows: a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. The minimum spanning tree is a spanning tree of minimum weight. For example, given the following graph:



Its minimum spanning tree is:



- (a) Show that the minimum spanning tree has an optimal substructure by showing that given an MST for a graph  $G = (V, E)$ , any sub-tree of the MST is a minimum spanning tree with respect to the subset of vertices and their edges in  $G$ .
- (b) A cut in a graph  $G = (V, E)$  is a division of the vertices into two disjoint non-empty subsets,  $S$  and  $T$ . An edge that crosses the cut is an edge that has one vertex in  $S$  and one vertex in  $T$ . An example here shows a cut in the graph above, where  $S = \{v_1, v_3, v_4\}$  and  $T = \{v_2, v_5\}$ . Edges crossing the cut are shown in bold. The greedy choice property says that for any cut in the graph, the lightest edge crossing the cut is a part of a minimum spanning tree. Prove it is true for any cut (not only this example!). You can show by a simple exchange argument.



- (c) Show that the lightest edge in the graph is always a part of an MST (there may be more than one MST if the weights are not unique).
7. Given a tree  $T = (V, E)$  (not necessarily binary) an *independent set* is subset of nodes in the tree such that no two nodes are adjacent (in a tree it means, no parent and child can be in the same independent set). The problem of finding the maximum independent set in a tree involves finding an independent set of maximum size (obviously).
- (a) Show that the problem has an optimal substructure by showing that given a maximum independent set on a tree, every subset of it represents a maximum independent set with respect to its subtree.

- (b) Every tree has at least one leaf. Show that any leaf node  $v$  in a tree must be a part of a maximum size independent set. **Hint:** Assume we have a maximum independent set  $S$  on the tree. It either contains  $v$  or not, take it from there.
  - (c) Give a linear time algorithm to obtain a maximum size independent set in a tree.
8. Prove that a binary tree that is not full (i.e., that at least one of its nodes has only one child) cannot correspond to an optimal prefix code.
9. Suppose that a data file contains a sequence of 8-bit characters such that all 256 characters are about equally common: the maximum character frequency is less than twice the minimum character frequency. Prove that Huffman coding in this case is no more efficient than using an ordinary 8-bit fixed-length code.