Homework 7: Graph Algorithms

Due: November 7, 2024

Problem 1. Given an arbitrary undirected graph $G = (V, E)$, applying DFS on a given vertex will create a tree. The tree can be used to detect the separating edges and vertices of the graph.

First, some notation: Given $G = (V, E)$, a Depth-First Tree rooted at $r \in V$, and a pair of distinct vertices $u, v \in V$ such that there is a directed path comprised only of forward (discovery) edges of the Depth-First Tree from u to v, we say that u is an *ancestor* of v and v is a *descendant* of u. If u is an ancestor of v and the edge (u, v) is part of the Depth-Fist Tree then we say that u is a direct ancestor of v and v is a direct descendant of u.

(a) Show that the root vertex of a DFS tree for G is a separating vertex of G if and only if if it has more than one direct descendant in the DFS tree.

(b) Show that any non-root vertex v of a DFS tree is a separating vertex of G if and only there exists a child of v, w, such that none of w and w's descendants (if they exist) in the DFS tree have a back-edge to an ancestor of v in the DFS tree.

Solution.

(a) The root vertex of a DFS tree has no back-edges. If it only has one child, then removing the root of the tree keeps the DFS tree connected, so it is not a separating vertex.

If the root vertex of a DFS tree is a separating vertex, then we know that it splits the graph into at least two connected components. For the removal of this vertex to split the DFS tree, it must have degree of at least 2, and since no back-edges of the root vertex can exist, it must have at least 2 children.

(b) Let v be a non-root vertex of a DFS tree that has a child v' where there are no descendants of v' has a back-edge to a proper ancestor of v in the DFS tree. Then we know that the removal of v will separate a proper ancestor of v (which must exist because v is not a root) and v' so it is a separating vertex. If v' has no descendants and no back edge, then again we disconnect v' and an ancestor of v, resulting in v separting in both cases.

On the other hand, if v is a separating non-root vertex of a DFS tree, we know that there must only be one path to get from a proper ancestor of v to any descendant of v. So there must not be any back-edges from any of the descendants of v (excluding itself) to proper ancestors of v in the DFS tree.

□

Problem 2. The city of Irvine, California, allows residents to own a maximum of three dogs per household without a breeder's license. Imagine you are running an online pet adoption website for the city for n Irvine residents and m puppies.

Describe an efficient algorithm for assigning puppies to residents that provides for the maximum number of puppy adoptions possible while satisfying the constraints that each resident will only adopt puppies that he or she likes and that no resident can adopt more than **three** puppies. Provide a proof of the correctness of your algorithm along with an analysis of runtime and space complexity.

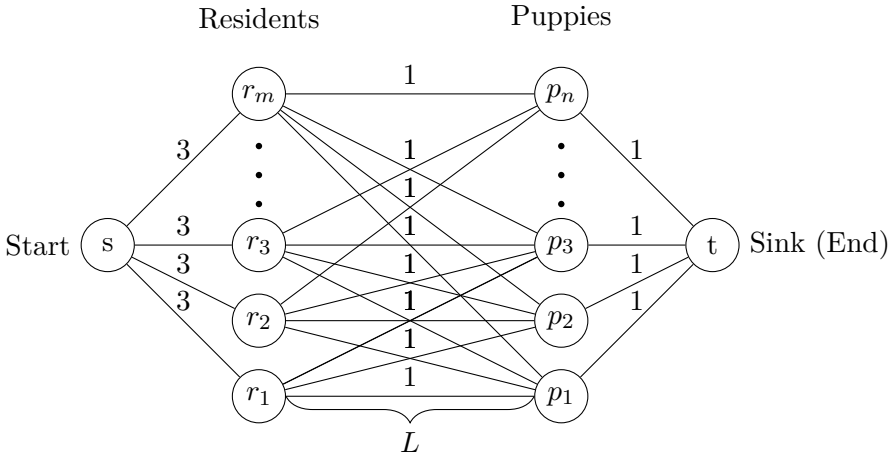
Solution. The structure of this algorithm is as follows:

(a) Assume that we are given (along with the residents $r_1, r_2, \ldots, r_n$ and puppies $p_1, p_2, \ldots, p_m$) a list of the puppies that each resident likes, given in the form:

$$L = \{(i, j) : \text{resident } i \text{ likes puppy } j\} \tag{1}$$

So that we have an explicit representation of these relationships.

(b) Construct a graph G with the following structure and edge capacities:



(1) The flow through this graph will represent which puppies get adopted (i.e. if there is flow between r_i and p_j, then that represents r_i adopting puppy p_j.

(2) The edge from the start to each resident r_i has capacity 3 (since each resident can adopt at most 3 puppies).

(3) The edge from each puppy p_j to the sink has capacity 1 (since each puppy can only be adopted once, and a single adoption is one single flow).

(4) We **only** construct an interior edge between r_i and p_j if $(i, j) \in L$ (the graph drawn above represents the worst case, where every resident likes every puppy).

We can then apply Ford-Fulkerson on this graph to find the maximum flow f^*. From this max flow, we can construct our solution: if there is an edge with flow 1 between p_i and r_j, then we say that person p_i has adopted puppy p_j. We can repeat this for all of L to construct a solution list S of adoptions of the form (i, j), and return it.

It remains to check this flow network satisfies the conditions of the problem. In constructing G, we restrict the flow to a maximum of 3 puppies per resident (which aligns with the given condition), and only have an edge with flow (indicating an adoption) between r_i, p_j if r_i likes p_j (from our construction of L) and it is part of the optimal adoption flow.

Further, since have verified that our initial conditions in the construction of G is correct, and we assume (since we proved it in class) that Ford-Fulkerson produces a correct flow (and this flow is what precisely determines the optimal adoptions), we know that this must produce the optimal adoptions for these puppies.

The space complexity of this algorithm is the size of the graph: there are $m + n + 2$ vertices, and at most $m + mn + n$ edges (where mn comes from resident/puppy pairs), resulting in $O(mn)$ space. The time complexity is $O(|f^*||E|)$, where E is the number of edges. We just showed $O(|E|) = O(mn)$. We can bound the flow by $3m$ and $3n$: the former comes from the flow through the start to the residents, the latter is from puppies to the sink. This results in $|f^*| \in O(\min(m, n))$, so we get a time complexity of $O(mn \min(m, n))$.

Remark: If we wanted to get rid of the minimum, then we can replace with either one of m, n, or $m + n$ since these all bound $\min(m, n)$ from above. □