Evan Edelstein

EN.605.621.84.FA25

HW 5

The work in this exercise is mine alone without un-cited help. No AI was used to answer these questions.

1. Let *T* be a tree produced by DFS at the root of a graph *G* and *R* be an isomorphic tree *T* produced by BFS on the root of the graph *G*. We want to prove that , that is *G* cannot contain any edges that are not also in *T*. We can prove this by contradiction. Since the tree produced by DFS and BFS are isomorphic to each other we know that each vertex has the same predecessor/successor relationship in both trees (i.e. edge order is preserved). Let’s assume there exists an edge *(v1,v2)* that is in *G* but not *T*. This could only happen if the edge *(v1,v2)* was missed by DFS, which could happen if *(v2,v1)* was seen first and added to *T* instead (or if *G* is disconnected, which by the problem statement it is not). But by the invariant of isomorphism the ordering of the edges in *T* must be equal for DFS and BFS, so no reverse edge should be in *T*. Thus, having an edge in *G* and not *T* is a contradiction.
   1. We can describe a bipartite graph *G* that maps each person to a night to cook . If the mapping is a perfect matching, then we know each night has a person to cook dinner and no nights are without a cook. This is because we have an equal number of cooks to nights, so a perfect matching would cover all nights with a cook. If it was not perfectly matching than there could be a night that does not have an assigned cook. Figure 1 visualizes the mapping graph *G*.

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 1:** An example perfect matching bipartite graph *G* that maps each person to a night to cook . [made with draw.io]

* 1. We will use the mechanics of flow graphs to help construct a perfect matching bipartite graph *G* that can be a valid cooking schedule. We note that by finding the maximum flow through the graph we can construct the perfect matching. We start by creating a start and sink vertex and creating an edge from the start to each person with a flow of 0 and capacity of 1, and from each day to the sink with a flow of zero and capacity of 1. We can then add the properly scheduled connections to the graph by creating an edge from the person to their day to cook with a flow of 1 and capacity of 1. We then add the two mismatched edges to the graph with flow of zero and capacity of 1.

We can then use the Ford-Fulkerson method (with BFS) to find an augmented path, and thus a proper matching between the two remained cooks and days.

Pseudocode: {todo}

* 1. We can construct a flow graph to help solve this problem. We create a flow graph F with a source, sink and all vertices and edges in G with capacity 1. For each populated city c in X we create an edge (S,c) with a capacity of 1. Similarly, all we create an infinite capacity edge (S,y) for all safe vertices in S. A set of evacuation routes exists if and only if the maximum flow value equals the number of vertices in X (populated cities). We can use the Ford-Fulkerson algorithm to do compute this in polynomial time, .