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 (ie 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 searched and added to T instead. 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. Essentially, BFS and DFS only produce the same tree if G is already a tree, which means all edges would be traversed by DFS.
   1. We can describe a bipartite graph G that maps each person Pi to a night to cook Di. If the mapping has a perfect matching, then we know each night has exactly one 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 visaulzes 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 Pi to a night to cook Dj. [made with draw.io]

* 1. We will use the mechanics of flow graphs to help contruct a perfect matching bipartite graph G that can be a valid cooking schedule.

We start by constructing the bipartite graph G from the valid schedule made by the friend. We assume Pi can cook on Dk and add that to G. We now have a perfect matching the bipartite graph of size n-1 that can be a valid schedule. We can use the Ford-Fulkerson method (with DFS or BFS) to find an augmented path in G. If it exists, we can use that to pair off the missing cook. If not we cannot form a perfect matching and thus cannot have a proper cooking schedule.

Pseudocode: {todo}

* 1. We can construct a flow graph to help solve this problem. We creat 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, .