Suppose you and your friend Alanis live, together with n−2 other people, at a popular off-campus cooperative apartment. Over the next n nights, each of you is supposed to cook dinner for the co-op exactly once, so that someone cooks on each of the nights.

Of course, everyone has scheduling conflicts with some of the nights, so deciding who should cook on which night becomes a tricky task. For concreteness, let's label the people $\{p_1, ..., p_n\}$ and the nights $\{d_1, ..., d_n\}$; and for person p_i, there's a set of nights $S_i \subset \{d_1, ..., d_n\}$ when they are *not* able to cook.

A *feasible dinner schedule* is an assignment of each person in the co-op to a different night, so that each person cooks on exactly one night, there is someone cooking on each night, and if p_i cooks on night d_j, then $d_j \notin S_i$.

(a) Describe a bipartite graph G so that G has a perfect matching if and only if there is a feasible dinner schedule for the co-op.

(b) Your friend Alanis takes on the task of trying to construct a feasible dinner schedule. After great effort, she constructs what she claims is a feasible schedule and then heads off to class for the day.

Unfortunately, when you look at the schedule she created, you notice a problem. $n-2$ of the people at the co-op are assigned to different nights on which they are available: no problem there. But for the other two people, p_i and p_j, and the other two days d_k and d_l, you discover that she has accidentally assigned both p_i and p_j to cook on night d_k, and assigned no one to cook on night d_l.

You want to fix Alanis's mistake but without having to recompute everything from scratch. Show that it's possible, using her "almost correct" schedule, to decide in only $O(n^2)$ time whether there exists a feasible dinner schedule for the co-op. (If one exists, you should also output it.)

ANSWER

(a) Construct a bipartite graph of $G(P \cup D, E)$, where each node $p_i \in P$ represents a person; each node $d_i \in D$ represents a day and an edge $\{p_i, d_j\} \in E$ represents the availability of p_i on d_j to cook ($d_j \notin S_i$). *Note:* $|P| = |D|$.

A perfect matching in G gives a *feasible dinner* since every person will be assigned to a day when they are available and no two person will be assigned to the same day.

A *feasible dinner* is a perfect matching in G since an assignment A is a set of (p_i, d_j), where every person is only used exactly once and no two person is assigned to the same endpoint (day) in G.

(b) First construct G in the following ways: create the nodes for P and D then for every pair (p_i, d_j) decide whether $d_j \notin S_i$ and edge accordingly. This takes $O(n^2)$, because we check $n \cdot n$ edges and each query to check elements in a set can be done in $O(1)$ time.

Use Alanis's matching except that remove p_i's or p_j's assignment to d_k. This way Alanis has a valid matching of size $n-1$. Now look for an augmenting path in G. If there is an augmenting path, there is a perfect matching and using that path it is possible to find the assignment. If there is no augmenting path, then the perfect matching does not exist (using theorem from class) because $n-1$ is the maximum matching in G. We can find augmenting path in $O(n^2)$ thus the algorithm runs in $O(n^2)$.

Note: You can use Ford-Fulkerson algorithm, assign capacities and flows to G as discussed in class. This approach has the same asymptotic running time in this case as looking directly for augmenting paths.