CS 577: Introduction to Algorithms — Homework 9 Solutions

1. **Part (a)**

We reduce the construction of an optimal fair schedule to an instance of MAX-FLOW. Add a source s, a sink t to the network. For each person j, we add a node P_j and an edge (s, P_j) of capacity Δ_j to enforce the constraint that person j may drive no more than Δ_j times. For each day i, we add a node S_i and an edge (S_i, t) to enforce the constraint that there is at most one driver on any given day. Finally, for each pair (P_j, S_i) such that person j goes to work on day i, we add an edge (P_j, S_i) of capacity 1 to indicate that person j can drive once on the i-th day.

We now show that there is a one-to-one correspondence between integral flow in the network and fair schedules.

Claim 1. *There exists a fair schedule with a driver on k days if and only if there is an* integral *flow of value k.*

Proof. Suppose there is fair schedule that has a driver on k days. Then one can obtain an integral flow of value k as follows: If person j drives on day i, send 1 unit of flow along the path s, P_j, S_i, t. We do so for all days with a driver. Since we started with a fair schedule with a driver on k days, all capacity constraints in the network are satisfied (why?) and we send k units of flow from s to t. By the construction of the flow, it is integral.

Now for the other direction, let us suppose that we have an integral flow of value k. We obtain a fair driving schedule with a driver on k days as follows: If an edge (P_j, S_i) carries a unit of flow, we have person j drive on day i. By the capacity constraints on edges leaving the source and edges entering the sink, each person j drives at most Δ_j times and each day has at most one driver. This tells us that the schedule we constructed is fair. Now, since the flow has value k, there are k days i such that the edge (S_i, t) carries a unit of flow. Since the flow into an vertex is equal to the flow out of a vertex, this means there are k days i for which there exists a person j such that (P_j, S_i) carries a unit of flow. This tells us that there are k days that have a driver in the schedule we constructed. □

By the claim, to find the optimal fair schedule, we only need to find an integral max-flow in the network. One can use the Ford-Fulkerson algorithm to do so. Given an integral max-flow, we construct a fair schedule as we did in the proof of the claim.

Running Time The running time of our algorithm is given by the time required to construct the network plus the time to find the max-flow in the network plus the time to retrieve a fair schedule from the max-flow. The size of the network we construct is $O(nd)$. As a result, the time required to construct the network is $O(nd)$, and one can retrieve a fair schedule given the max-flow in $O(nd)$ time. The running time of Ford-Fulkerson is $O(|E|F)$, where $|E|$ is the number of edges in the graph and F is the value of the maximum flow. Since there are d edges entering the sink each of capacity 1, the max-flow in the graph has value at most d. Therefore, finding the max-flow takes $O(nd^2)$ time (the number of edges in the graph is $O(nd)$) . It follows that our algorithm for finding the optimal fair schedule runs in $O(nd^2)$ time.

Part (b)

By the claim, to show that there is a fair schedule that has a driver on each day, it is sufficient to exhibit an integral flow of value d in the network. To show that there is an integral flow of value d, it is sufficient to show that there is an s-t flow of value d (why?). We construct such a flow as follows: For each person j and for each day i on which j goes to work, send $\frac{1}{|S_i|}$ units of flow along the path s, P_j, S_i, t. The amount of flow through an edge (s, P_j) is exactly $\sum_{i:j \in S_i} \frac{1}{|S_i|} \leq \Delta_j$. The flow along an edge (P_j, S_i) is $\frac{1}{|S_i|}$. And the amount of flow along an edge (S_i, t) is $\sum_{j \in S_i} \frac{1}{|S_i|} = 1$. Therefore, each edge (S_i, t) carries one unit of flow and the total flow into the sink is d. This shows that there is a flow of value d. It follows that there is a fair schedule with a driver on all days.

2. **(a)**

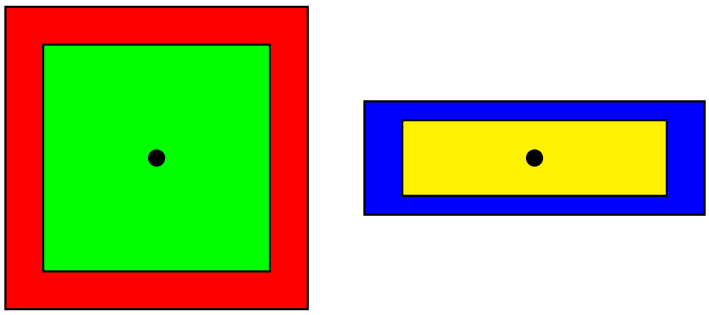
A simple counterexample that relies on a specific tie breaking rule has 2 pegs and 4 discs that are rectangles with the following lengths and widths,

$$\{(5,5),(3,3),(6,2),(4,1)\}.$$

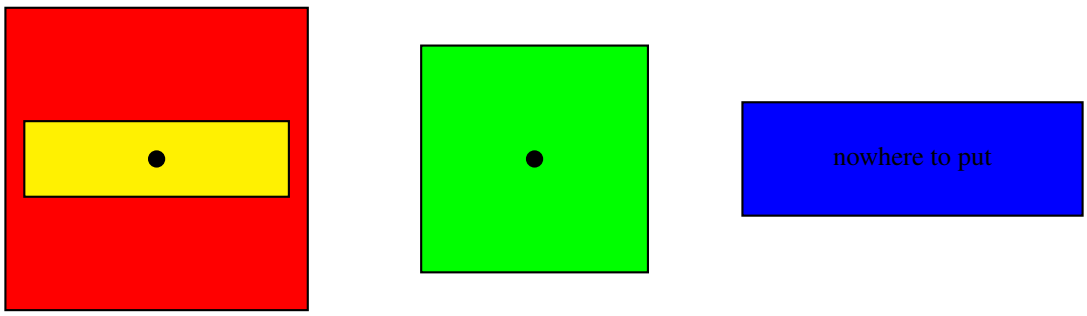
The corresponding C is,

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

The algorithm returns FALSE because the discs can be placed as follows,
On peg 1 : $\{(5,5),(4,1)\}$,
On peg 2: one of $\{(3,3)\}$, or $\{(6,2)\}$ because none of them can be placed on top of another.
The correct solution to the task is TRUE because the discs are placed as follows,
On peg 1 : $\{(5,5),(3,3)\}$,
On peg 2 : $\{(6,2),(4,1)\}$.



correct top view



greedy top view

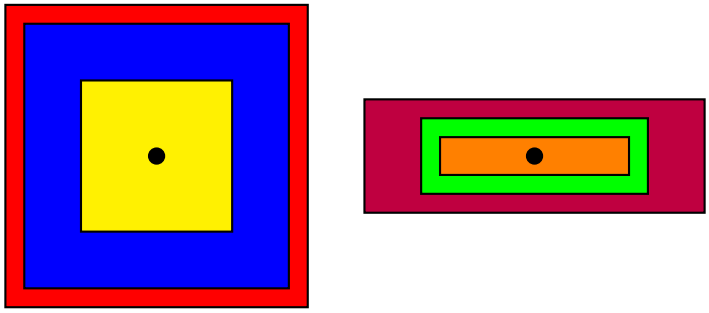
A counterexample that does not rely on a specific tie breaking rule has 2 pegs and 6 discs that are rectangles with the following lengths and widths,

$$\{(8,8),(7,7),(4,4),(9,3),(6,2),(5,1)\}.$$

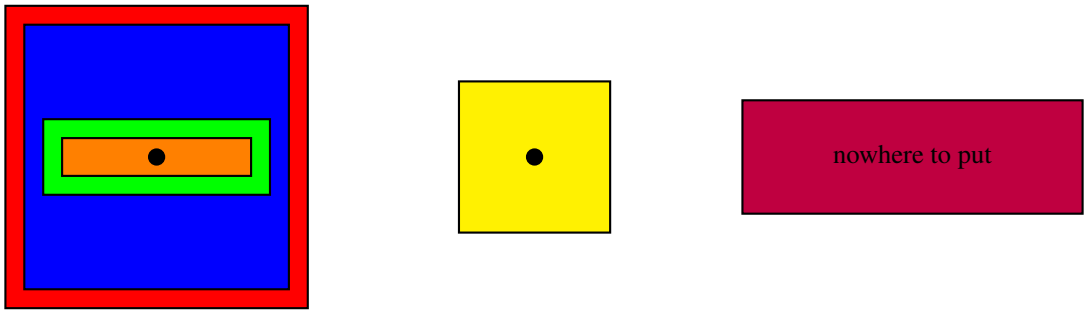
The corresponding C is,

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

The algorithm returns FALSE because the discs are placed as follows,
On peg 1 : $\{(8,8),(7,7),(6,2),(5,1)\}$,
On peg 2: only one of $\{(4,4)\}$ or $\{(9,3)\}$ because none of them can be placed on top of the other.
The correct solution to the task is TRUE because the discs are placed as follows,
On peg 1 : $\{(8,8),(7,7),(4,4)\}$,
On peg 2 : $\{(9,3),(6,2),(5,1)\}$.



correct top view



greedy top view

(b)

The task is to find the maximum number of discs that can be placed on k or fewer pegs. This is equivalent to placing at least $n-k$ discs on top of other discs: every time a disc i is put on top of a disc j, the total number is increased by 1, regardless of whether disc i is already on top of some other discs. The constraints are that each disc is only on top of one other disc, or no two discs on placed directly on top of the same third disc at the same time. These constraints are similar to those of a maximum matching problem.

This problem can be modelled by maximum bipartite matching. Given n discs with their compatibility matrix C, construct a bipartite graph G with $2n$ vertices, one on the left and one on the right for each disc. There is a

edge from disc i on the left to disc j on the right if disc i fits on top of disc j. Formally, $G = (L; R, E)$, where,

$$L = R = \{1, 2, ...n\},$$
$$E = \{(i, j) : i \in L, j \in R, \text{ and } C_{ij} = 1\}.$$

Algorithm 1: Correct

Input: The list of discs D, the compatibility matrix M, and the number of pegs k

```
Generate G = (L; R, E) as described above;
Find M = maximum bipartite matching of G using the algorithm from the lectures;
if | M | ≥ n − k then
    return TRUE
else
    return FALSE
```

To show that any matching M of G corresponds to a valid placement of the discs: no two edges share an endpoint, enforcing the conditions that no disc has two or more discs directly on top of it and the condition that no disc is placed directly on top of two or more discs at the same time. Furthermore, each edge of the matching represents a placement of a disc on top of another disc, and so decreases the total number of discs placed on the k pegs by 1. This shows that a matching M corresponds to a placement of discs on $n - |M|$ pegs. To show that any valid placement of discs on top of other discs corresponds to a matching in this graph: given a valid placement, A, of discs, the matching of G can be constructed by choosing the edge from $i \in L$ to $j \in R$ if disc i is placed directly on top of disc j in A. These edges always exist in G since disc i fits on top of disc j, and the set of such edges taken together is a matching, as no two of them can share an endpoint (since this would mean that either some disc i is directly on top of two or more discs or more than one disc is placed directly on top of some disc j). Furthermore, if m pegs are used in $A, n - m$ discs must have been placed on top of other discs. So, $|M| = n - m$.

This means that there is a placement of discs using at most k pegs if and only if there is a matching of G with at least $n - k$ edges. Therefore, to determine whether the n discs can be placed on no more than k pegs, it suffices to check whether the maximum matching in G has size at least $n - k$.

The running time of the algorithm is equal to the time required to construct the graph $G, O\left(n^2\right)$, plus the time to find the maximum matching, $O\left(|V||E|\right) = O\left(n^3\right)$, plus the time to check whether the maximum matching is at least $n - k, O\left(1\right)$. Therefore, the algorithm runs in $O\left(n^3\right)$ time.