CS 473: Algorithms, Spring 2021
HW 8 (due Wednesday, April 7th at 8pm)

This homework contains three problems. **Read the instructions for submitting homework on the course webpage**.

Collaboration Policy: For this home work, each student can work in a group with up to three members. Only one solution for each group needs to be submitted. Follow the submission instructions carefully.

For problems that use maximum flows as a black box, a full-credit solution requires the following.

- A complete description of the relevant flow network, specifying the set of vertices, the set of edges (being careful about direction), the source and target vertices s and t, and the capacity of every edge. (If the flow network is part of the original input, just say that.)
- A description of the algorithm to construct this flow network from the stated input. This could be as simple as "We can construct the flow network in $O(n^3)$ time by brute force."
- A description of the algorithm to extract the answer to the stated problem from the maximum flow. This could be as simple as "Return TRUE if the maximum flow value is at least 42 and False otherwise."
- A proof that your reduction is correct. This proof will almost always have two components. For example, if your algorithm returns a Boolean, you should prove that its TRUE answers are correct and that its FALSE answers are correct. If your algorithm returns a number, you should prove that number is neither too large nor too small.
- The running time of the overall algorithm, expressed as a function of the original input parameters, not just the number of vertices and edges in your flow network.
- You may assume that maximum flows can be computed in $O(VE)$ time. Minimum-cost flows can be computed in $O(E^2 \log^2 V)$ time. Do *not* regurgitate the maximum flow algorithm itself.

Reductions to other flow-based algorithms described in class or in the notes (for example: edge-disjoint paths, maximum bipartite matching, minimum-cost circulation) or to other standard graph problems (for example: reachability, minimum spanning tree, shortest paths) have similar requirements.

---

1. Consider a bipartite (undirected) graph represented by $G = (L \cup R, E)$ where L and R are sets of n nodes each, and for each edge $(u, v) \in E$, $u \in L$ and $v \in R$. Matching $M \subseteq E$ in G is a set of edges such that every node has at most one edge incident on it from M; the nodes covered by the edges of M are called matched nodes. M is said to be a *perfect matching* if every node of L and R are matched. See Figure **??** for an example.

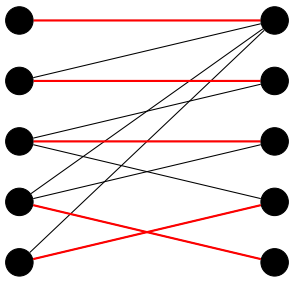


Figure 1: Bipartite Graph where the red edges represent a perfect matching

Hall's theorem states that G has a perfect matching if and only if for every subset $S \subseteq L$, $|S| \leq |N(S)|$ where $N(S) = \{v \in R \mid \exists u \in S, (u, v) \in E\}$ is the neighborhood of S (or equivalently $N(S) = S \times R \cap E$).

Prove Hall's theorem using the maxflow-mincut theorem.

[Hint: Use the construction covered in the class to compute maximum matching using max-flow]

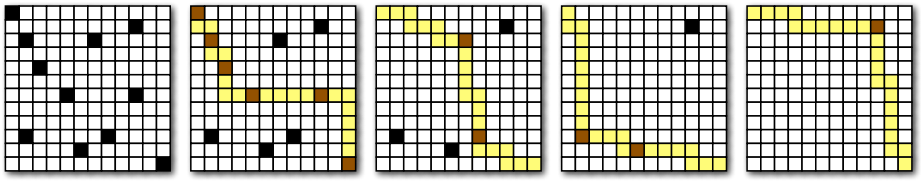
2. Let $G = (V, E)$ be a *directed* graph and let $\mathcal{C} = \{C_1, C_2, \ldots, C_h\}$ be a collection of cycles in G. We say that $\mathcal{C}$ is a *cycle partition* of G if each vertex of V is in exactly one of the cycles. In other words the cycles of $\mathcal{C}$ are vertex disjoint and together contain all vertices. Describe an algorithm that given G decides whether G contains a cycle partition. Follow the two steps below.

   (a) Argue that a set of edges $E' \subseteq E$ forms a cycle partition if and only if each vertex v has exactly one incoming edge and one outgoing edge in E'.

   (b) Use bipartite matching to check if there is an $E' \subseteq E$ satisfying the property in the previous part.

3. Suppose we are given an $n \times n$ grid, some of whose cells are marked; the grid is represented by an array $M[1..n, 1..n]$ of booleans, where $M[i, j] =$ True if and only if cell (i, j) is marked. A monotone path through the grid starts at the top-left cell, moves only right or down at each step, and ends at the bottom-right cell. Our goal is to cover the marked cells with as few monotone paths as possible.

   - **Not to submit:** Describe an algorithm to find a monotone path that covers the largest number of marked cells.
   - **Not to submit:** There is a natural greedy heuristic to find a small cover by monotone paths: If there are any marked cells, find a monotone path Π that covers the largest



Greedily covering the marked cells in a grid with four monotone paths.

number of marked cells, unmark any marked cells covered by Π, and recurse. Show that this algorithm does *not* always compute an optimal solution.

- Describe and analyze an efficient algorithm to compute the smallest set of monotone paths that covers every marked cell.