1 Graph algorithms (1.5 point)

The police department in the city of Computopia has made all streets one-way. The mayor contends that there is still a way to drive legally from any intersection in the city to any other intersection, but the opposition is not convinced. A computer program is needed to determine whether the mayor is right. However, the city elections are coming up soon, and there is just enough time to run a linear-time algorithm.

1. Formulate this problem graph-theoretically, and explain why it can indeed be solved in linear-time.

2. Suppose it now turns out that the mayor’s original claim is false. She next claims something weaker: if you start driving from town hall, navigating one-way streets, then no matter where you reach, there is always a way to legally drive back to the town hall. Formulate this weaker property as a graph-theoretic problem, and carefully show how it too can be checked in linear time.

2 Divide and conquer (1.5 point)

Suppose you’re consulting for a bank that’s concerned about fraud detection, and they come to you with the following problem. They have a collection of $n$ bank cards that they’ve confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic stripe with some encrypted data, and it corresponds to a unique account in the bank. Each account can have many bank cards corresponding to it, and we’ll say that two bank cards are equivalent if they correspond to the same account.

It’s very difficult to read the account number off a bank card directly, but the bank has a high tech “equivalence tester” that takes two bank cards and, after performing some computations, determines whether they are equivalent.

Their question is the following: among the collection of $n$ cards, is there is set of more than $n/2$ of them that are all equivalent to one another? Assume that the only feasible operation you can do with the cards are to pick two of them and plug them in to the equivalence tester. Show how to decide the answer to their question with only $O(n \log n)$ invocations of the equivalence tester.
3 Network flow (1.5 points)

Suppose you and your friend Alanis live, together with \( n - 2 \) other people, at a popular off-campus cooperative apartment, the Upson Collective. 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 (e.g. exams, concerts, etc.), so deciding who should cook on which night becomes a tricky task. For convenience, let’s label the people

\[
\{p_1, \ldots, p_n\},
\]

the nights

\[
\{d_1, \ldots, d_n\},
\]

and for person \( p_i \), there’s a set of nights \( S_i \subseteq \{d_1, \ldots, d_n\} \) 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, and if \( p_i \) cooks on night \( d_j \) then \( d_j \not\in S_i \).

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

2. 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 big problem. \( n - 2 \) 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’ mistake but without having to recompute everything from scratch. Show that it is 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.)

4 Red-black trees (1.5 points)

1. Give an efficient algorithm which, given a list of \( n \) nodes, sorted on their keys, constructs a red-black tree with a minimal number of red nodes.

2. Given \( n \), what is the minimal number of red nodes in a red-black tree?

5 Greedy algorithms (1.5 points)

Alice wants to throw a party and is deciding whom to call. She has \( n \) people to choose from, and she has made up a list of which pairs of these people know each other. She wants to pick a many people as possible, subject to two constraints: at the party, each person should have at least five other people whom they know and five other people whom they don’t know.

Give an efficient algorithm that takes as input the list of \( n \) people and the list of pairs who know each other and outputs the best choice of party invitees. Show the correctness of your algorithm and give the running time in terms of \( n \).
6 Dynamic programming (1.5 points)

You are given a rectangular piece of cloth with dimensions $X \times Y$, where $X$ and $Y$ are positive integers, and a list of $n$ products that can be made using the cloth. For each product $i \in [1, n]$ you know that a rectangle of cloth of dimensions $a_i \times b_i$ is needed and that the final selling price of the product is $c_i$. Assume the $a_i$, $b_i$, and $c_i$ are all positive integers. You have a machine that can cut any rectangular piece of cloth into two pieces either horizontally or vertically. Design an algorithm that determines the best return on the $X \times Y$ piece of cloth, that is, a strategy for cutting the cloth so that the products made from the resulting pieces give the maximum sum of selling prices. You are free to make as many copies of a given product as you wish, or none if desired.