1. You are running a web site that is visited by the same set of people every day. When a user registers with your site, the provide their membership in one or more demographic groups. Your site is supported by advertisers. Each advertiser has told you which demographic groups should see its ads and how many of its ads you must show each day. Altogether, there are n visitors, k demographic groups, and m advertisers. Describe an efficient algorithm to determine whether you can show each visitor exactly one ad per day, so that every advertiser has its desired number of ads displayed, and every ad is seen by someone in an appropriate demographic group.

**Sol**: Certainly! Let’s tackle this problem step by step. We’ll formulate it as a **maximum flow problem** to efficiently determine whether we can show each visitor exactly one ad per day while satisfying the advertisers’ requirements.

1. **Problem Statement**:
   * We have **n visitors**, **k demographic groups**, and **m advertisers**.
   * Each advertiser specifies a set of demographic groups (denoted as (X\_i)) and a minimum number of users they want to show their ads to (denoted as (r\_i)).
   * Our goal is to find a way to show no more than one ad to each user such that every advertiser’s contract is satisfied, and every ad is seen by someone in an appropriate demographic group.
2. **Graph Representation**:
   * We’ll create a directed graph with the following layers:
     + **Source (s)**: Represents the ads.
     + **Advertisers (A)**: Each advertiser corresponds to a node in this layer.
     + **Demographic Groups (G)**: Each demographic group corresponds to a node in this layer.
     + **Users (U)**: Each user corresponds to a node in this layer.
     + **Sink (t)**: Represents the end of the flow.
   * The edges between layers will have capacities:
     + From **s** to **A**: Capacity equal to the number of ads each advertiser wants to show ((r\_i)).
     + From **A** to **G**: Capacity equal to infinity (since an advertiser can target any demographic group).
     + From **G** to **U**: Capacity equal to 1 (since each user can see at most one ad).
     + From **U** to **t**: Capacity equal to 1 (to ensure each user sees at most one ad).
3. **Flow Constraints**:
   * The flow from **s** to **A** represents the number of ads shown to users.
   * The flow from **A** to **G** represents the assignment of demographic groups to advertisers.
   * The flow from **G** to **U** represents the assignment of users to demographic groups.
   * The flow from **U** to **t** ensures that each user sees at most one ad.
4. **Maximum Flow Algorithm**:
   * We can use the **Ford-Fulkerson** algorithm or **Edmonds-Karp** algorithm to find the maximum flow in the graph.
   * Start with an initial flow of 0.
   * While there exists an augmenting path from **s** to **t**:
     + Find the minimum capacity along the path.
     + Update the flow along the path.
   * The maximum flow represents the maximum number of ads we can show while satisfying the constraints.
5. **Solution**:
   * If the maximum flow equals the total number of ads required by all advertisers, we can show each visitor exactly one ad per day.
   * Otherwise, it’s not possible to satisfy all advertisers’ requirements.
6. **Pairing Users with Advertisements**:
   * To determine which user sees which ad, traverse the flow from **U** to **s**.
   * Assign each user to the corresponding ad based on the flow.

Remember that this approach assumes that the demographic group classification is already done. The graph structure and capacities ensure that each user sees at most one ad and that advertisers’ contracts are met. [The maximum flow algorithm efficiently solves this problem, allowing you to optimize ad placements while meeting advertiser demands1](https://stackoverflow.com/questions/66543822/making-facebook-ads-a-maximum-flow-problem).

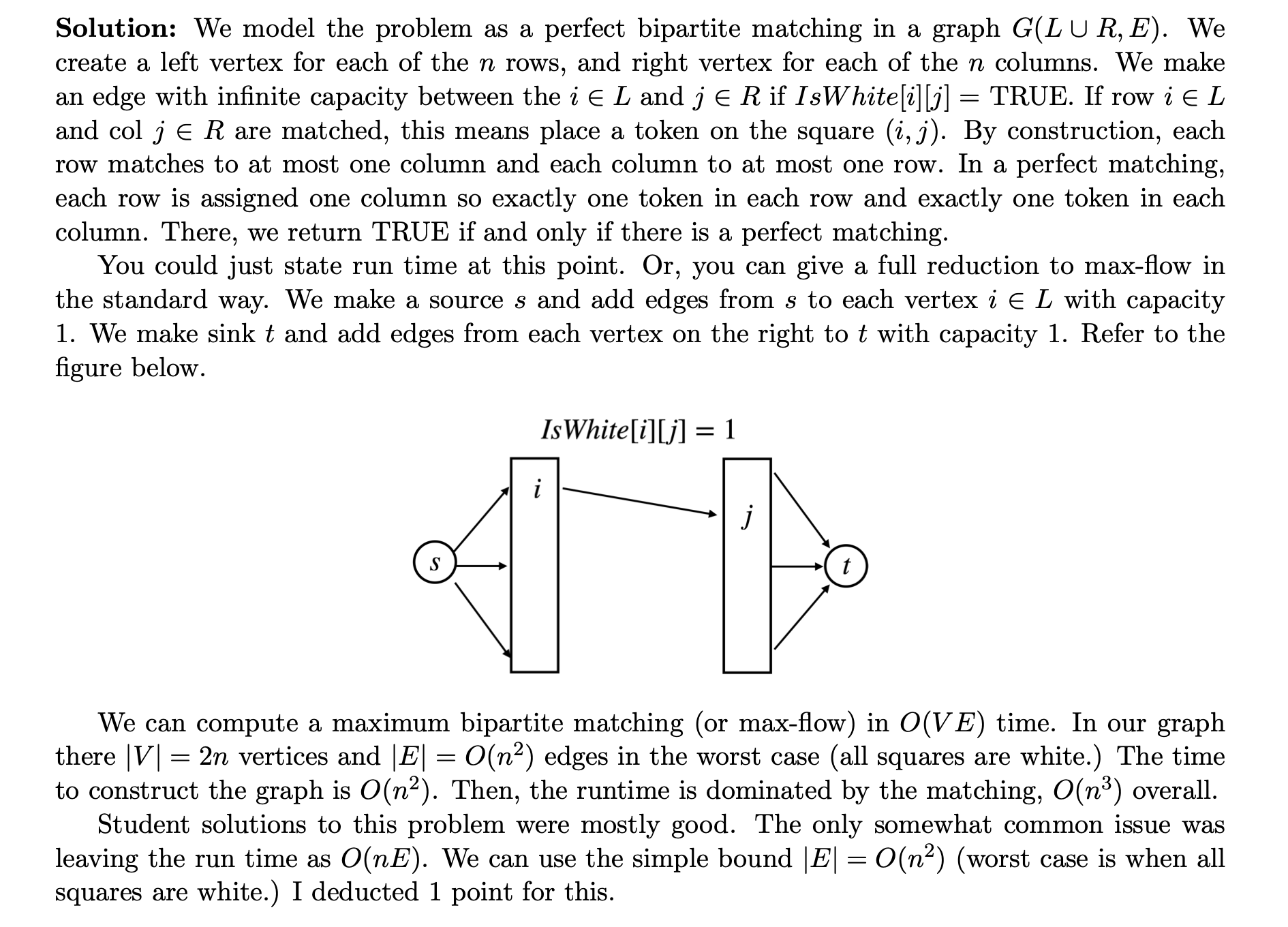
O(VE) to calculate max flow = V = n users + k demographics + m advertisers + 1 source + 1 sink , let it be O(N) and E = O(nk + km)= O(N^2), so O(VE) = O(N^3)

2.

Suppose we are given an n × n square grid, some of whose squares are colored black and the rest white. Describe and analyze an algorithm to determine whether tokens can be placed on the grid so that

• every token is on a white square  
 • every row of the grid contains exactly one token  
 • every column of the grid contains exactly one token.

Your input is a two dimensional array is IsWhite[1...n][1...n] of booleans, indicating which squares are white. Your output is a single boolean. For example, given the grid in the figure below, your algorithm should return True.

**Sol: **

3. Ad-hoc networks are made up of low-powered wireless devices. In principle, these networks can be used on battlefields, in regions that recently suffered natural disasters, and in other hard to reach areas. The idea is that a large collection of cheap, simple devices could be distributed throughout the area of interests (for example, by dropping them from an air- plane); the devices would then automatically configure themselves into a functioning wireless network.

These devices can communicate only within a limited range. We assume all devices are identical; there is a distance D such that two devices can communicate if and only if the distance between them is at most D.

We would like our ad-hoc network to be reliable, but because the devices are cheap and low-powered, they frequently fail. If a device detects that it is likely to fail, it should transmit its information to some other backup device within its communication range. We require each device x to have k potential backup devices, all within distance D of x. We call these k devices the backup set of x. Also, we do not want any device to be in the backup set of too many other devices, otherwise, a single failure might affect a large fraction of the network.

Suppose we are given the communication radius D, parameters b and k, and an array d[1 . . . n][1 . . . n] of distances, where d[i][j] is the distance between distance devices i and j. Describe an algorithm that either computes a backup set of size k for each of the n devices, such that no device appears in more than b backup sets, or reports that no such collection of backup sets exists.

**Sol:** Certainly! The problem you’ve described can indeed be framed as a maximum flow problem in the context of ad-hoc networks. Let’s delve into an algorithmic approach to address this challenge.

1. Graph Representation:
   * Construct a graph where each device corresponds to a node.
   * Connect nodes with an edge if their distance is at most *D* (communication radius).
2. Flow Network:
   * Create a flow network from the graph:
     + Add a source node *S* and a sink node *T*.
     + Connect *S* to all devices with an edge capacity of *k* (representing the desired backup set size).
     + Connect each device to *T* with an edge capacity of *b* (limiting the number of backup sets a device can appear in).
3. Edge Capacities:
   * Assign capacities to edges within the graph:
     + Set the capacity of each edge between devices to 1 or 0 depending on where its within the range, connect if d[i][j]<=D(since each device can be part of at most one backup set).
     + Set the capacity of edges from *S* to devices to *k*.
     + Set the capacity of edges from devices to *T* to *b*.
4. Maximum Flow:
   * Compute the maximum flow in the network using a flow algorithm (e.g., Ford-Fulkerson, Edmonds-Karp, or Dinic).
   * The flow represents the assignment of backup sets to devices.
5. Backup Sets:
   * For each device *x*, identify the devices that have flow from *S* to *x*.
   * These devices form the backup set for *x*.
6. Validity Check:
   * Verify that no device appears in more than *b* backup sets.
   * If any device violates this constraint, report that no valid collection of backup sets exists.

* The algorithm ensures that each device has a backup set of size *k* while avoiding excessive overlap.
* The flow network captures the constraints on backup set sizes and inter-device communication.
* The algorithm can be implemented efficiently using standard flow algorithms.

**Or**

Yes, you can formulate this problem as a maximum flow problem. Here's how you can approach it:

1. \*\*Graph Representation\*\*: Represent the network as a graph where each device is a node, and there is an edge between two nodes if they are within communication range (i.e., the distance between them is at most D).

2. \*\*Flow Network\*\*: Transform this graph into a flow network where each node (device) has a source node and a sink node. Connect the source to each device with an edge representing its need for backup devices and connect each device to the sink with an edge representing its capability to serve as a backup device.

3. \*\*Capacity\*\*: Assign capacities to the edges between the source and the devices based on the required number of backup devices (k), and capacities to the edges between the devices and the sink based on their ability to serve as backup devices.

4. \*\*Objective\*\*: The objective is to maximize the flow from the source to the sink while satisfying the capacity constraints and the constraint that no device appears in more than b backup sets.

5. \*\*Algorithm\*\*: Use a maximum flow algorithm such as Ford-Fulkerson or Edmonds-Karp to find the maximum flow in the network.

6. \*\*Backup Set\*\*: After finding the maximum flow, the backup set for each device can be determined based on the flow through the network. Each device will have a set of backup devices that are connected to it in the flow network.

7. \*\*Constraint Check\*\*: Ensure that no device appears in more than b backup sets. If this constraint is violated, report that no such collection of backup sets exists.

This approach allows you to find the backup sets for each device while ensuring that the specified constraints are met.

Runtime:

O(VE) = V total nodes = 2\*n

E = O(n^2)

So O(VE) = O(n^3)

Or O(Ef\*) where f\* = kxn and E = O(n^2)