1 - Exam preparation

We consider the following problem :

Problem 1. *Taking the right set of courses for each semester is always challenging – in this problem you will design an algorithm for this task.*
Suppose that there are n courses available for the next semester and the exam season for the next semester starts at day 1. Also, for any $j \in 1, 2, ..., n$, the j'th course has c_j credits and its exam is on day d_j. In order to pass a course you need to allocate exactly one day to prepare for its exam. You want to obtain the maximum number of credits possible in the next semester. Design an efficient algorithm that finds a set of optimal courses (with the maximum credits in total) that you can pass and prove the correctness of your algorithm, assuming the following:

- *You can only prepare for the exams after the exam season starts (it starts on day 1).*
- *On each day, you can only study one course the whole day (no multitasking).*
- *The exams take place at the end of each day. Assume that they do not take any time, i.e., if you have any number of exams on any day, you can still study on that day.*
- *If you do not study for a course before its exam, you are guaranteed to fail that course. For example, to pass an exam that is on day d_j, you need to study for that exam on a day i, where $1 \leq i \leq d_j$.*
- *It is possible to have more than one exam on a day.*

Proposed Solution

Consider the bipartite graph $G = (T \cup S, E)$ that we construct as such :

- For each day of the session we add a vertex t_i to T (so $T = \{t_1, t_2, ...t_m\}$) for a m days long session.
- For each possible course we add a vertex s_i to S (so $S = \{s_1, s_2, ...s_n\}$) for a semester where n courses are available.
- We add an edge $e = (t_j, s_i)$ between two vertices $\iff$ $i \leq d_j$ (for a given course-vertex s_i we add an edge to every day-vertex t_j when it would still be possible to prepare for the exam).

We then define the following cost function $w : V \to \mathbb{R}$

$$w_{(t_j, s_i)} = c_j.$$

The problem as formulated becomes a *maximum-weight bipartite matching* problem on G with respect to w that we can solve in polynomial time (as it is a matroid intersection problem).

2 - Safe Travels

0.1 3 - Implementation

Just do max-flow/min-cut ????????