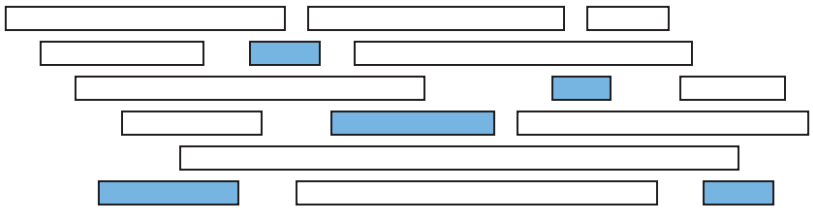
CS 4349.400 Homework 5

Due Wednesday October 17th, in class

Please answer each of the following questions.

1. Recall the class scheduling problem from lecture: We are given two arrays $S[1..n]$ of start times and $F[1..n]$ of finish times (with $0 \le S[i] < F[i]$ for each i). Our goal is to find a conflict-free schedule $X \subseteq \{1, \cdots, n\}$ of maximum size such that for any pair $i, j \in X$, either $S[i] > F[j]$ or $S[j] > F[i]$.



A maximal conflict-free schedule for a set of classes.

   (a) Some seemingly good greedy algorithms may not actually give the best solution. For example, suppose we choose a course x that *conflicts with the fewest other courses*, discard all classes that conflict with x, and recurse.[1] Describe an input example where this greedy strategy *does not* produce a maximal conflict-free schedule. *[Hint: The remaining parts do not depend upon this one.]*

   (b) However, the following strategy *does* produce a maximal conflict-free schedule: Choose the course x that *starts last*, discard all classes that conflict with x, and recurse.

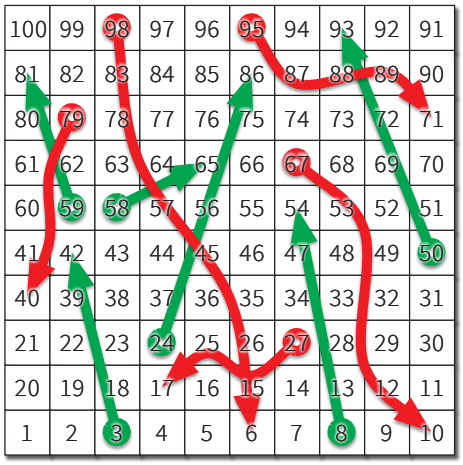
   Prove there is a maximal conflict-free schedule that includes the course x that starts last. Your proof should use an exchange argument similar to those given in the lecture notes or textbook.

   (c) Now, give a short induction proof that the strategy does produce a maximal conflict-free schedule. You may assume your proof from part (b) is correct; i.e., there is a maximal conflict-free schedule that includes the course x that starts last.

   (d) Finally, describe a simple iterative algorithm to implement the greedy strategy discussed above. You may want to use Erickson's GreedySchedule$(S[1..n], F[1..n])$ procedure as a template. What is the running time of your algorithm?

[1] Thanks, Tyler!

2. ***Snakes and Ladders*** is a classic board game, originating in India no later than the 16th centure. The board consists of an $n \times n$ grid of squares, numbered consecutively from 1 to n^2, starting in the bottom left corner and proceeding row by row from bottom to top, with rows alternating to the left and right. Certain pairs of squares in this grid, always in different rows, are connected by either "snakes" (leading down) or "ladders" (leading up). ***Each square can be an endpoint of at most one snake or ladder.***



A typical Snakes and Ladders board. Upward straight arrows are ladders; downward wavy arrows are snakes.

You start with a token in cell 1, in the bottom left corner. In each move, you advance your token up to k positions, for some fixed constant k. If the token ends the move at the *top* end of a snake, it slides down to the bottom of that snake. Similarly, if the token ends the move at the *bottom* end of a ladder, it climbs up to the top of that ladder.

We want to design an algorithm to determine if the token can reach the last square of the grid. It turns out we can solve this problem using a simple reduction to WHATEVERFIRSTSEARCH. Specifically, we can build a graph, call WHATEVERFIRSTSEARCH(s) for some vertex s, and then use the output in some useful way.

(a) What vertices should we use for our graph? *[You may want to read about configuration graphs in the lecture notes.]*

(b) What edges should we use in our graph? Are they directed or undirected edges?

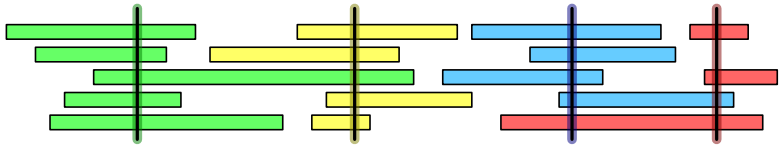
(c) We want to call WHATEVERFIRSTSEARCH(s) on our graph. Which vertex should we use for s?

(d) WHATEVERFIRSTSEARCH(s) has marked several vertices. How do we use this output to decide whether or not the last square is reachable?

(e) Suppose we use a bag with $O(1)$ time insertion and removal of edges so that WHATEVERFIRSTSEARCH runs in $O(V + E)$ time. What is the running time of our algorithm ***in terms of n***.

(f) Finally, suppose we want to compute the smallest number of moves required for the token to reach the last square of the grid. Which bag data structure should we use? Describe a simple way to extract the minimum number of moves from the output of WHATEVERFIRSTSEARCH.

3. Let X be a set of n intervals on the real line. We say that a set P of points *stabs* X if every interval in X contains at least one point in P. Describe and analyze a greedy algorithm to compute the smallest set of points that stabs X. Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$, representing the left and right endpoints of the intervals in X.



A set of intervals stabbed by four points (shown here as vertical segments).

A substantial portion of your credit will come from proposing a point to include in your output and giving an exchange argument that your choice appears in some optimal solution. *[Hint: There is a good greedy strategy that looks a lot like the one we used for class scheduling.]*[2]

[2]In fact, the maximal number of conflict-free intervals from X is *equal* to the minimum number of points for a stabbing set. You do not need to prove this fact.