CS1200: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Problem Set 6

Harvard SEAS - Fall 2025 *Due: Wed Oct. 29, 2025 (11:59pm)*

Please see the syllabus for the full collaboration and generative AI policy, as well as information on grading, late days, and revisions.

All sources of ideas, including (but not restricted to) any collaborators, AI tools, people outside of the course, websites, ARC tutors, and textbooks other than Hesterberg–Vadhan must be listed on your submitted homework along with a brief description of how they influenced your work. You need not cite core course resources, which are lectures, the Hesterberg–Vadhan textbook, sections, SREs, problem sets and solutions sets from earlier in the semester. If you use any concepts, terminology, or problem-solving approaches not covered in the course material by that point in the semester, you must describe the source of that idea. If you credit an AI tool for a particular idea, then you should also provide a primary source that corroborates it. Github Copilot and similar tools should be turned off when working on programming assignments.

If you did not have any collaborators or external resources, please write 'none.' Please remember to select pages when you submit on Gradescope. A problem set on the border between two letter grades cannot be rounded up if pages are not selected.

Your name:
Collaborators:
No. of late days used on previous psets:
No. of late days used after including this pset:

1. (`GreedyColoring` for IntervalScheduling) The IntervalScheduling–Optimization problem we studied in class finds the largest group of nonintersecting intervals. In many applications, it is also natural to consider the *coloring* version of the problem, where we want to partition the input intervals into as few groups as possible so that each group is nonintersecting. For example, in the original motivating problem about radio station airtime, solving the coloring version would allow us to figure out the minimum number of radio station channels we need to allow all of the input segments to be scheduled.

   In this problem, you will prove that `GreedyColoring` in order of *increasing start time* gives an optimal solution to IntervalScheduling–Coloring. (Note the contrast with the *increasing finish time* ordering we used for the version studied in class. It is a common phenomenon that different orderings are better for coloring vs. independent-set problems; for example decreasing vertex degree is a good heuristic for greedy coloring of general graphs, while increasing vertex degree is a good heuristic for independent set.) Let $x = (x_0, \ldots, x_{n-1})$ be an instance of IntervalScheduling, where each x_i is an interval $[a_i, b_i]$ with $a_i, b_i \in \mathbb{Q}$. You may assume for simplicity that all of the numbers $a_0, a_1, \ldots, a_{n-1}, b_0, b_1, \ldots, b_{n-1}$ are distinct.

   Let k be the maximum number of input intervals that contain any value $t \in \mathbb{Q}$. That is,

   $$k = \max_{t \in \mathbb{Q}} |\{i \in [n] : t \in x_i\}|.$$

   (a) Prove that every coloring for the IntervalScheduling instance x uses at least k colors.

(b) Show that `GreedyColoring` in order of *increasing start time* uses at most k colors. (To develop your intuition, carry out the algorithm on a few examples.)

(c) Show that the `GreedyColoring` in order of increasing start time can be correctly implemented in time $O(n \log n + nk)$. (Hint: keep track of the end times of the most recently scheduled intervals assigned to each color. For correctness, ensure that your algorithm faithfully implements `GreedyColoring`.)

(d) (Extra credit[1]) Using an appropriate data structure, improve the runtime to $O(n \log n + n \log k) = O(n \log n)$. (Hint: You may instead implement a *variant* of `GreedyColoring` in which, at every step, you can assign a vertex either the smallest available color or *any already-used color that is not assigned to any of its neighbours* (as opposed to standard `GreedyColoring` in which we always assign the smallest color).)

2. (Maximal Independent Sets) Let $G = (V, E)$ be a graph. A set $S \subseteq V$ is a *maximal* independent set if we cannot add any vertices to S while it remains an independent set. That is, for every vertex $v \in V \setminus S$, $S \cup \{v\}$ is not an independent set.

   (a) Show that given a graph G, a maximal independent set can be found in time $O(n + m)$. Note that this is in sharp contrast to *maximum-size* independent sets, for which we do not know any subexponential-time algorithms. (Hint: be greedy.)

   (b) Show that if G is 3-colorable, then it has a 3-coloring f in which the set of vertices of color 2 (i.e. $f^{-1}(2)$) is a maximal independent set.

   (c) It is known that every graph G has at most $3^{n/3}$ maximal independent sets, and there is an algorithm (the Bron–Kerbosch algorithm) that enumerates all of the maximal independent sets in time $O(3^{n/3})$. Use this fact to conclude that 3-coloring can be solved in time $O((n + m) \cdot 3^{n/3}) = O(1.45^n)$, improving the runtime of $O(1.89^n)$ from SRE4.

3. (Exponential-Time Coloring) In the Github repository for PS6, we have given you basic data structures for graphs (in adjacency list representation) and colorings, an implementation of `ExhaustiveSearch` k-Coloring algorithm, an implementation of the `BronKerbosch` algorithm, and a variety of test cases (graphs) for coloring algorithms.

   (a) Implement the $O(n + m)$-time algorithm for 2-coloring that we covered in class in the function `bfs_2_coloring`, verifying its correctness by running `python3 -m ps6_tests 2`. Your implementation of BFS should follow the presentation and notation that we used in class (with the loop over distance d and the sets F and S), which may be different than presentation of BFS in other sources (online or in the optional textbooks).

   (b) Implement the $O((n + m) \cdot 3^{n/3})$-time algorithm for 3-coloring (`MaximalIS+BFS`) from Problem 2 above in the function `iset_bfs_3_coloring`, also verifying its correctness by running `python3 -m ps6_tests 3`.

[1]Solutions to this part will not affect differences between N, L, R-, and R grades, but can help achieve an R+

(c) Compare the efficiency of `ExhaustiveSearch` 3-coloring and the `MaximalIS+BFS`. Specifically, identify and write down the largest instance size n each algorithm is able to solve (within a time limit you specify, e.g. 10 seconds) and the smallest instance size n each algorithm is unable to solve (again within that same time limit).

Within the `ps6_experiments.py` file, the experiments generate two types of graphs: line of rings and randomized cluster connections. More detailed explanations of these graphs are found in the `ps6_experiments.py` file, but each graph type can be briefly explained as follows:

- **Line of Rings:** A ring is structured as a cycle of n nodes, with an edge between each pair of consecutive nodes and an edge closing the cycle between node n-1 and node 0. For instance, for n=5, the edges would be as follows: $(0 \to 1, 1 \to 2, 2 \to 3, 3 \to 4, 4 \to 0)$. A line of rings is defined as a collection of rings such that a node from a new ring forms an edge with the last vertex from the existing collection of rings.
- **Randomized Cluster Connections:** These graphs create clusters of independent sets, where for each pair of nodes that are in different clusters, we add an edge between them with probability p.

An example response (with dummy values) for `MaximalIS+BFS` would be:

With a one-second timeout and no parameters changed, the largest instance `MaximalIS+BFS` could solve was 10 rings of size 300, or $n = 3000$. The smallest instance it could not solve was 4 clusters of size 10, or $n = 40$.

In addition to these numeric values, please provide a brief explanation of why these results make sense, based on your knowledge of both the algorithms' runtime and how each algorithm goes about finding a coloring. For this part, there is no need to go through every combination of parameters; feel free to give just the largest and smallest instances each algorithm can solve and speak generally as to why one algorithm performs better than the other. Make sure to also include a comparison between the types of graphs (i.e. do your algorithms perform better on the Line of Rings or the Randomized Cluster Connections? Why?). More instructions can be found in `ps6_experiments`.