CS120: Intro. to Algorithms and their Limitations Hesterberg & Anshu

Problem Set 5

Harvard SEAS - Fall 2023 *Due: Wed Oct. 25, 2023 (11:59pm)*

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

1. (Exponential-Time Coloring) In the Github repository for PS5, we have given you basic data structures for graphs (in adjacency list representation) and colorings, an implementation of the Exhaustive-Search k-Coloring 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 ps5_tests 2`.

   (b) Implement `is_independent_set`, which checks if a given subset of nodes is an independent set.

   (c) Implement the $O(1.89^n)$-time algorithm for 3-coloring (ISET + BFS) that you studied in the SRE in the function `iset_bfs_3_coloring`, also verifying its correctness by running `python3 -m ps5_tests 3`.

   (d) Compare the efficiency of Exhaustive-Search 3-coloring and the $O(1.89^n)$-time algorithm. Specifically, identify the largest instance each algorithm is able to solve (within a time limit you specify, e.g. 10 seconds) and the smallest instance each algorithm is unable to solve (again within that same time limit).

   In addition to these numeric values, please provide a brief explanation of why these results make sense, based on your knowledge of how each algorithm finds a valid 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. More instructions can be found in `ps5_experiments`.

2. (Reductions Between Variants of IndependentSet) Consider the following three variants of the IndependentSet problem:

   - IndependentSet-OptimizationSearch: given a graph G, find the largest independent set in G.
   - IndependentSet-ThresholdSearch: given a graph G and a number $k \in \mathbb{N}$, find an independent set of size at least k in G (if one exists).
   - IndependentSet-ThresholdDecision: given a graph G and a number $k \in \mathbb{N}$, decide (by outputting YES or NO) whether or not there is an independent set of size at least k in G.

For each part below, be sure to both prove correctness and analyze runtime for the algorithms you provide.

(a) Suppose that there is an algorithm running in time $T(n, m) \geq n + m$ that solves IndependentSet-OptimizationSearch on graphs with at most n vertices and at most m edges. Prove that there is an algorithm running in time $O(T(n, m))$ that solves IndependentSet-ThresholdDecision.

(b) Suppose that there is an algorithm running in time $T(n, m) \geq n + m$ that solves IndependentSet-ThresholdSearch on graphs with at most n vertices and at most m edges. Prove that there is an algorithm running in time $O((\log n) \cdot T(n, m))$ that solves IndependentSet-OptimizationSearch. (Hint: Come up with a reduction that makes at most $\log n$ oracle calls. You might find it useful to first find one that makes at most n oracle calls.)

(c) Suppose that there is an algorithm running in time $T(n, m) \geq n + m$ that solves IndependentSet-ThresholdDecision. Prove that there is an algorithm running in time $O(n \cdot T(n, m))$ that solves IndependentSet-ThresholdSearch. (Hint: Show that G has an independent set of size at least k containing vertex v iff $G - N(v)$ has an independent set of size at least $k - 1$, where $G - N(v)$ denotes the graph obtained by removing v and all of its neighbors from G. Use this fact and the oracle to determine for each vertex v, whether or not to include v in the independent set. Be sure to update the graph appropriately after each decision.)

We remark (but you don't need to submit anything) that the combination of the three previous problem parts means that for every constant $c \in [1, 2]$, if there is an algorithm solving any one of the three problems in time $(n+m)^{O(1)} \cdot c^n$, there are algorithms solving the other two problems in $(n+m)^{O(1)} \cdot c^n$ time. The best known algorithm (by Xiao and Nagamochi, 2013) has $c \approx 1.1996$.