CSCI 3104: Algorithms
Spring 2022
Recitation #14 -Complexity

SOLUTIONS

TA FCQ's. Please take take a few minutes to fill out TA FCQs! They help us improve our teaching and help the department track courses. You can fill them out here: colorado.campuslabs.com/courseeval

Problem 1

Reformulate the following optimization problems as decision problems. For each, argue that the decision version is in P.

a. *Sequence Alignment.* Given strings (A, B), find the minimum value of edit operations to convert string A into string B.

> We reformulate this as the decision problem: given strings (A, B) and integer k, can string A be converted into string B with edit operations costing at most k?
>
> Using dynamic programming, we can compute the minimum number of edit operations to convert A to B in time $O(mn)$, where A is of length m and B of length n. We compute $\textsc{Opt}(m, n)$ in polynomial time and return 1 if $\textsc{Opt}(m, n) \leq k$ and 0 otherwise. Hence, Sequence Alignment $\in$ P.

b. *Minimum Spanning Tree.* Given a weighted graph $G = (V, E, w)$ with weights $w(e) \geq 0$ for all $e \in E$, find a spanning tree of minimal cost.

> We reformulate this as the decision problem: given weighted graph $G = (V, E, w)$ with weights $w(e) \geq 0$ for all $e \in E$ and integer k, is there a spanning tree for G of cost at most k?
>
> We have seen that Kruskal's algorithm computes a minimum spanning tree in time $O(|E|^2)$ (this can be refined to $O(|E| \log |V|)$). We can run Kruskal's and then compute the total weight W of the produced MST in polynomial time, and return 1 if $W \leq k$ and 0 otherwise. Thus, MST $\in$ P.

c. *Interval Scheduling.* Given a set of intervals $L = \{(s_1, e_1), \ldots, (s_n, e_n)\}$, find a maximal-size subset of intervals which do not overlap.

> We reformulate this as the decision problem: given a set of intervals $L = \{(s_1, e_1), \ldots, (s_n, e_n)\}$ and integer k, is there a subset of non-overlapping intervals of size at least k?
>
> We can use our greedy interval scheduling algorithm to compute a maximal-size set of intervals of size m in $O(n \log n)$ time (sorting by earliest finish takes $O(n \log n)$ time and selecting the edges can be done in $O(n)$ time). If $m \geq k$ return 1, otherwise return 0.

Problem 2

A *Hamiltonian cycle* on a directed graph $G = (V, E)$ is a cycle which visits each vertex in V exactly once. Recall that a *cycle* is a path with the same start and end vertices.

a. The Hamiltonian Cycle problem is: given a directed graph $G = (V, E)$, does G contain a Hamiltonian cycle?

   Show that Hamiltonian Cycle is NP-hard via a reduction from 3SAT.

> See proof presented on pages 475-477 of Kleinberg and Tardos.

b. The Hamiltonian Path problem is, similarly, does there exist a *path* which visits every vertex in the graph? (recall that a path need not start and end at the same place, while a cycle does)

   Show that Hamiltonian Cycle $\leq_p$ Hamiltonian Path, that is, Hamiltonian Path is NP-hard by a reduction to Hamiltonian Cycle.

> We reduce Hamiltonian Cycle on the directed graph $G = (V, E)$ to Hamiltonian Path on $G' = (V', E')$.
>
> We create the graph $G' = (V' = V \setminus v \cup \{v', v''\}, E')$ by replacing an arbitrary vertex $v \in V$ with the pair of vertices v' and v''. We replace all outward edges (v, u) from v with edges $(v', u) \in E'$, and all incoming edges (u, v) with edges $(u, v'') \in E'$. All other edges remain unchanged.
>
> We then solve Hamiltonian Path on G'. If there exists an Hamiltonian path in G', then it must start from v' and end at v'', since v' only has outgoing edges and v'' only has incoming edges. This path equates to a cycle which begins and ends at v in G.
>
> Similarly, if there exists a Hamiltonian cycle in G, then it is a Hamiltonian path in G'.

Problem 3 (Bonus)

The pandemic has ended, and you're having a big group of friends over for a celebratory dinner! Unfortunately, each of your m friends has very restrictive dietary needs, many of which are incompatible.

You have a large recipe book R with n recipes in it, and friend i can eat a subset $R_i \subseteq R$ of the foods in your recipe book. You've been trying to come up with a set of dishes $M \subseteq R$ to cook such that every guest can eat at least one dish ($M \cap R_i \neq \emptyset$ for all i), but you have the time to make at most k dishes before your friends arrive.

Show that the problem Meal Planning of determining whether there exists a set M of recipes you can cook such that $|M| \leq k$ and every guest can eat at least one dish is NP-complete.

(*Hint:* try reducing from Set Cover or 3SAT.)

This is a formulation of the Hitting Set problem.

We reduce from Set Cover: given a set V of vertices, a collection $S = \{S_1, \ldots, S_n\}$ of sets with $S_i \subseteq V$, and an integer k, can we select $H \subseteq S$ such that $|H| \leq k$ and for all $v \in V$, $v \in S_i$ for some $S_i \in H$?

We formulate our Set Cover instance as a Meal Planning instance as follows:

- For each set S_i, create a recipe R_i
- For each vertex $v \in V$, create a friend v who can eat any meal R_i such that $v \in S_i$
- Keep k the same

Solve the Meal Planning instance, and return the result.

We claim that this returns 1 if there is a Set Cover of size at most k and 0 otherwise.

Proof. Assume there is a set cover of size at most k, which uses $H = \{S_1, \ldots, S_j\}$, $j \leq k$. Select meals $\{R_1, \ldots, R_j\}$. Then, since each vertex v is in some set $S_{i(v)}$, friend v must be able to eat meal $R_{i(v)}$, and all friends have a meal they can eat with at most k meals.

Now, assume that our Meal Planning formulation returns 1. Then, there is some set of recipes $R = \{R_1, \ldots, R_j\}$, $j \leq k$, such that each friend can eat at least one food. Thus, each vertex v is then in at least one of $\{S_1, \ldots, S_j\}$, and $H = \{S_1, \ldots, S_j\}$ forms a set cover of size at most k. $\square$