

Computability

1. Prove that decidable languages are closed under
 - complement
 - intersection
 - union
2. Prove that recognizable languages are closed under
 - union
 - intersection
3. Prove that recognizable languages are not closed under complement.
4. Prove that the class of languages decidable/recognizable by non-deterministic Turing machines is the same as the class of languages decidable/recognizable by deterministic Turing machines.
5. Let S be an infinite countable set, and let $\mathcal{P}(S)$ denote all the subsets of S . Prove that $\mathcal{P}(S)$ is uncountable.
6. Consider the problem of deciding whether a Turing machine runs in polynomial time.

$$\text{POLYTIME} = \{\langle M \rangle \mid \exists k \text{ s.t. } T_M(n) \in O(n^k)\}$$

Prove that POLYTIME is unrecognizable.

7. For a language $B \subseteq \Sigma^*$, define

$$L_B = \{\langle M \rangle \mid L(M) \subseteq B\}$$

Prove that if $B \neq \Sigma^*$, then L_B is unrecognizable.

Time Complexity

8. Prove that \mathbf{P} is closed under
 - complement
 - union
 - intersection

9. Prove that **NP** is closed under

- union
- intersection

10. Prove that if **NP** \neq **co-NP**, then **P** \neq **NP**.

11. A dominating set in a graph $G = (V, E)$ is a set $S \subseteq V$ of vertices such that every $v \in V$ has a neighbour in S . The DOMINATING-SET decision problem is defined below.

Instance: A graph $G = (V, E)$, and a number $k \in \mathbb{N}$

Question: Does G have a dominating set of size k or smaller?

Either prove that DOMINATING-SET is in **P**, or prove that it is **NP**-complete, or prove that it is **co-NP**-complete.

12. The decision problem UNARY-PARTITION is defined below.

Instance: A set $p_1, \dots, p_n \in \mathbb{N}$, represented in unary

Question: Does there exist $S \subseteq \{1, \dots, n\}$ such that $\sum_{i \in S} p_i = \sum_{j \notin S} p_j$?

Either prove that UNARY-PARTITION is in **P**, or prove that it is **NP**-complete, or prove that it is **co-NP**-complete.

13. The decision problem UNARY-CLIQUE is defined below.

Instance: A graph $G = (V, E)$, and a number $k \in \mathbb{N}$ in unary

Question: Does G have a clique of size k or larger?

Either prove that UNARY-CLIQUE is in **P**, or prove that it is **NP**-complete, or prove that it is **co-NP**-complete.

14. If $G = (V, E)$ is a graph, and $V' \subseteq V$, then the induced subgraph of G on V' is the graph $G' = (V', E')$, where $E' = \{\{u, v\} \in E \mid u, v \in V'\}$. That is, the induced subgraph on V' is the graph obtained by removing all vertices not in V' , and removing all edges that are not fully contained in V' . A graph H is an induced subgraph of G if there exists $V' \subseteq V$ such that the induced subgraph of G on V' is (isomorphic to) H . A graph G is H -free if H is not an induced subgraph of G .

Instance: A graph $G = (V, E)$, and a graph H

Question: Is G H -free?

Either prove that SUBGRAPH-FREENESS is in **P**, or prove that it is **NP**-complete, or prove that it is **co-NP**-complete.

15. The problem HAM-PATH-SEARCH is: given a directed graph G , find a Hamiltonian path in G , or output \perp if no Hamiltonian path exists. Prove that if HAM-PATH $\in \mathbf{P}$, then there is a polynomial-time algorithm for HAM-PATH-SEARCH.

Approximation Algorithms

16. Let $\beta \leq 1$, let L be a language, and let Π be a maximization problem. Prove that if there is a β -gap-introducing reduction from L to Π , and there is a polynomial-time, β -approximation algorithm for Π , then $L \in \mathbf{P}$.

17. The problem SIMPLE-KNAPSACK-OPT is defined below (it is called “simple knapsack” because it is equivalent to the knapsack problem restricted to the case where each item has weight equal to its value).

Instance: Numbers w_1, \dots, w_n, W

Solution: A subset $I \subseteq \{1, \dots, n\}$ such that $\sum_{i \in I} w_i \leq W$

Objective: Maximize $\sum_{i \in I} w_i$

The algorithm GREEDY sorts the items from largest to smallest, and then (starting with the largest item) takes each item if doing so would not exceed the capacity.

(a) Prove that for every $\varepsilon > 0$ there is an instance of SIMPLE-KNAPSACK-OPT for which GREEDY achieves an approximation ratio of $1/2 + \varepsilon$ or worse.

(b) Prove that GREEDY is a $1/2$ -approximation algorithm for SIMPLE-KNAPSACK-OPT.

(c) Prove that if the algorithm does not sort the items, then it does not achieve any constant approximation ratio.

18. The problem BIN-PACKING-OPT is defined below. Informally, the problem is: given a collection of items of different sizes and a number B representing the “bin size”, pack all the items into bins of size B and use as few bins as possible.

Instance: $b_1, \dots, b_n, B \in \mathbb{N}$

Solution: A number $M \in \mathbb{N}$ and an allocation $\sigma : \{1, \dots, n\} \mapsto \{1, \dots, M\}$ such that

$$\max_{1 \leq j \leq M} \left\{ \sum_{i: \sigma(i)=j} b_i \right\} \leq B$$

Objective: Minimize M

A greedy algorithm for BIN-PACKING-OPT is: examine each item in turn, and places the current item into the first bin into which it fits (if it does not fit into any bin, then start a new bin).

Prove that this greedy algorithm is a 2-approximation for BIN-PACKING-OPT.

This algorithm is actually a 1.7-approximation algorithm. Can you think of an example where it requires three bins, but only two are necessary?