Complexity (SLOa) - Homework 2

Šimon Stupinský - xstupi00@stud.fit.vutbr.cz

Obtained points:

1. TASK (3.5 POINTS)

Assignment: Let the problem `MOD - PARTITION` is defined as follows: The input is a finite set of items S, a weight function $v : S \to \mathbb{N}$ and a number $k \in \mathbb{N}$. The problem asks whether there exists a set A such that:

$$\left(\sum_{i \in A} v(i)\right) \mod k = \left(\sum_{i \in S \setminus A} v(i)\right) \mod k$$

Prove that `MOD-PARTITION` is **NP**-complete.

NP-Completeness proving template

Firstly, we will attempt to give a general template to construct the proofs of *NP-Completeness* for an arbitrary decision problems. The concept of *completeness* is one of the most important in complexity theory, therefore we remind its definition. Let **C** be a complexity class, in this task **NP**, then we call a language **L**:

- **C-hard** if for all $L' \in \mathbf{C}$, $L' \leq L$,
- **C-complete** if L is **C**-hard and $L \in \mathbf{C}$.

We note, that $L' \leq L$ denote a *polynomial reduction* from L_1 to L_2, thus that there exists a **PTIME** Turing Machine computing a reduction function $R : \Sigma^* \to \Sigma^*$ s.t. $w \in L' \iff R(w) \in L$. This means that **C**-complete problems are the *harder* problems of **C**. To prove, that the given decision problem is **NP**-complete, we will need to show that it is in **NP**, and in **NP**-hard.

To show that `MOD-PARTITION` problem is **NP**-complete, we need to show the following things. There have to exist a non-deterministic polynomial-time algorithm that solves `MOD-PARTITION`, i.e., `MOD-PARTITION` $\in$ **NP**. Since it has to be **NP**-hard, any **NP**-complete problem B have to be reduced to it and the constructed reduction must be in polynomial time. With respect to the definition of the *reduction* function, the `MOD-PARTITION` problem has a solution if and only if B has a solution.

Proof of the NP-Completeness of `Mod-Partition` problem

`Mod-Partition` $\in$ NP. Initial, we show, that the problem is in **NP**. It is sufficient to show that a *certificate* (an encoding of a solution to the problem) can be *verified* to represent a solution to the decision problem in polynomial time. We can guess the two partitions and verify that both satisfies the equations of the given problem. Given input finite set of items S, our certificate is a set **A** which is a solution to the problem. The verification algorithm checks that $A \subseteq S$ and that: $\left(\sum_{i \in A} v(i)\right) \mod k = \left(\sum_{i \in S \setminus A} v(i)\right) \mod k$. Clearly that the certificate for instance of this problem can be verified in polynomial time.

`Mod-Partition` $\in$ NP $-$ hard. To prove this, it is sufficient to show that any instance of the already known decision problem that is **NP**-hard can be reduced to an instance of this problem in polynomial time. We will show that this problem is **NP**-hard by constructing the reduction from decision problem called `SET-PARTITION`, in other words, that the given problem is at least hard as `SET-PARTITION` problem. This problem is defined as follows, given a set S of numbers and v be the weight function $v : T \to \mathbb{Z}$, determine whether S can be partitioned into two sets A and $\overline{A} = S \setminus A$ of equal total weight, i.e.: $\sum_{i \in A} v(i) = \sum_{i \in \overline{A}} v(i)$. For clarity, we recall, that this problem is **NP**-complete.

Let (S, v) be the instance of the `SET-PARTITION` and $s = \sum_{i \in S} v(i)$. As we already knew, this problem determines whether the S can be partitioned into two sets A and $S \setminus A$ such that have equal sums $(s/2)$. We construct an instance of the `Mod-Partition` with the following items: $(\mathbf{S}, \mathbf{v}, \mathbf{s+1})$. The finite set of items S is unchanged, as the same as the given vector of weights v and the *modulus* number k we set to the sum of the set S plus one $(k = s + 1)$. We note that

this instance can be constructed in $\mathcal{O}(|\mathbf{S}|)$ by summing the elements in S, which proves our reduction in performed in polynomial time.

Now it remains to show that $\mathbf{S}$ can be partitioned into two distinct subsets ($\mathbf{A}$, $\mathbf{S} \setminus \mathbf{A}$) of equal weights ($\mathbf{s}/\mathbf{2}$) if and only if there is a subset $\mathbf{A}$ whose sum modulo by $(\mathbf{s}+\mathbf{1})$ is equal to the sum of the set $\mathbf{S} \setminus \mathbf{A}$ modulo by $(\mathbf{s}+\mathbf{1})$. In other word, we will prove that $(\mathbf{S}, \mathbf{v}) \in \texttt{SET} - \texttt{PARTITION} \iff (\mathbf{S}, \mathbf{v}, \mathbf{s}+\mathbf{1}) \in \texttt{MOD} - \texttt{PARTITION}$.

$\Rightarrow$ First, suppose that $\mathbf{A} \subseteq \mathbf{S}$ and the sum of elements in $\mathbf{A}$ is $\mathbf{s}/\mathbf{2}$, thus the $\mathbf{A}$ is a solution to the `SET-PARTITION` problem. Since there exist two sets of numbers in $\mathbf{S}$ whose sum is equal to $\mathbf{s}/\mathbf{2}$, therefore their result by modulus $(\mathbf{s}+\mathbf{1})$ will be equal, it will be more exactly $(\mathbf{s}/\mathbf{2}) \mod (\mathbf{s}+\mathbf{1}) = (\mathbf{s}/\mathbf{2})$. Thus, the partition into $\mathbf{A}$ and $\mathbf{S} \setminus \mathbf{A}$ is a solution to the `MOD-PARTITION` problem with modulus $\mathbf{s}+\mathbf{1}$.

$\Leftarrow$ Now, suppose that there exists $\mathbf{A} \subseteq \mathbf{S}$ such that $\left(\sum_{i \in A} v(i)\right) \mod (s+1) = \left(\sum_{i \in S \setminus A} v(i)\right) \mod (s+1)$, where $s = \sum_{i \in S} v(i)$. The large enough value of modulus $(\mathbf{s}+\mathbf{1})$ ensures that when two sets have equal results, then they have to have the same sums ($\mathbf{s}/\mathbf{2}$). No other two distinct subsets of $\mathbf{S}$, the sum of which is not equal, will not achieve an equal result after modulo by $(\mathbf{s}+\mathbf{1})$ due to the features of *least residue system modulo*. Since the set $\mathbf{A}$ is a solution to such `MOD-PARTITION` problem with modulus $(\mathbf{s}+\mathbf{1})$, the sum in both sets has to be equal. Therefore, we proved, that there exists a partition of $\mathbf{S}$ into two such that each partition sums to $\mathbf{s}/\mathbf{2}$.

Conclusion. As we proved above, due to a suitably constructed reduction to the instance of the `MOD-PARTITION` problem, we have shown the validity of the instance preservation within both problems in the reduction. The main trick has consisted of the appropriate choice of the *modulus*, which ensures the described requirement. We have chosen the value $\mathbf{s}+\mathbf{1}$ and not $\mathbf{s}$, to avoid collisions, that would arise when selecting a set $\mathbf{A} = \mathbf{S}$, which would never be the solution to the `SET-PARTITION` problem. Note that every larger number than $\mathbf{s}$ would have the same effect as chosen *modulus* $\mathbf{s}-\mathbf{1}$. We proved that the `MOD-PARTITION` problem is **NP**-complete. □

2. TASK (2 POINTS)

Assignment: Let $L_t = \{0\}$ be a language over the alphabet $\{0,1\}$. Prove (provide a ground ideas of the proof) the following statement: $\mathbf{P} = \mathbf{NP} \Longrightarrow L_t$ is **NP**-complete.

Solution

Assume that $\mathbf{P} = \mathbf{NP}$ according to the left side of the given implication. Clearly that language $\mathbf{L_t} \in \mathscr{L}_\mathbf{3}$, and since the regular languages are in **P** ($\mathscr{L}_\mathbf{3} \subseteq \mathbf{P}$), because a deterministic finite automaton is a restricted deterministic Turing machine that runs in linear time, then also $\mathbf{L_t} \in \mathbf{P}$: $\mathbf{L_t} \in \mathscr{L}_\mathbf{3} \wedge \mathscr{L}_\mathbf{3} \subseteq \mathbf{P} \Rightarrow \mathbf{L_t} \in \mathbf{P}$. This language $\mathbf{L_t}$ has the only one word, we mark this word as $\mathbf{w_{acc}} = \mathbf{0}$ and mark another word $\mathbf{w_{rej}} = \mathbf{1}$, which is over the given alphabet $\{\mathbf{0},\mathbf{1}\}$ and it does not belong to the $\mathbf{L_t}$: $\mathbf{w_{rej}} \notin \mathbf{L_t}$. We want to show that L_t is **NP**-complete.

From definition **NP**-completeness we have to prove, that $\mathbf{L_t} \in \mathbf{NP}$ and choose the instance **L** of **NP**-complete class and prove that: $\mathbf{L} \in \mathbf{NP} \Rightarrow \mathbf{L} \leq \mathbf{L_t}$. According to our assumption $\mathbf{N} = \mathbf{NP}$, then surely the language $\mathbf{L_t}$ is also in **NP**: $\mathbf{P} = \mathbf{NP} \wedge \mathbf{L_t} \in \mathbf{P} \Rightarrow \mathbf{L_t} \in \mathbf{NP}$, by which we are satisfying the first condition. Let $\mathbf{L_{NP}}$ be an arbitrary language from **NP**, then is clearly that: $\mathbf{L_{NP}} \in \mathbf{NP} \wedge \mathbf{NP} = \mathbf{P} \Rightarrow \mathbf{L_{NP}} \in \mathbf{P}$. Therefore, for $\mathbf{L_{NP}}$ have to exist a polynomial-time algorithm that for every input **s** solves[1] this problem within $\mathbf{p}(|\mathbf{s}|)$ steps, where $p(\cdot)$ is some polynomial function. Such an algorithm, that runs in polynomial time for every input, can be modelled for instance on a deterministic Turing machine. Denote such deterministic Turing machine, the decider of the $\mathbf{L_{NP}}$, as $\mathbf{M_{L_{NP}}}$.

Now, we need to argue that $\mathbf{L_{NP}} \leq \mathbf{L_t}$. We note, that this denotes there exists the polynomial reduction from $\mathbf{L_{NP}}$ to $\mathbf{L_t}$. i.e. that exists a polynomial-time Turing machine computing the reduction function **R**, which ensures the preservation within both problems: $w \in L_{NP} \Longleftrightarrow R(w) \in L_t$. Now, we will describe the construction of the polynomial-time reduction **f** from $\mathbf{L_{NP}}$ to $\mathbf{L_t}$. Let **w** be an instance of the $\mathbf{L_{NP}}$ and also the input for the deterministic Turing machine $\mathbf{M_{L_{NP}}}$. Firstly, we run $\mathbf{M_{L_{NP}}}$, the decider for $\mathbf{L_{NP}}$, on its input **w** and then check the result of the simulation:

- If $\mathbf{M_{L_{NP}}}$ accepted the input **w** then we set the output to the $\mathbf{f}(\mathbf{w}) = \mathbf{w_{acc}}$.
- If $\mathbf{M_{L_{NP}}}$ rejected the input **w** then we set the output to the $\mathbf{f}(\mathbf{w}) = \mathbf{w_{rej}}$.

Since $\mathbf{f}(\mathbf{w}) \in \mathbf{L_t}$ if and only if $\mathbf{w} \in \mathbf{L_{NP}}$, we showed that the reduction preserves membership in a problems. This is a polynomial-time reduction from $\mathbf{L_{NP}}$ to $\mathbf{L_t}$, and hence $\mathbf{L_t}$ is **NP**-complete since we have proved also the second condition of **NP**-completeness definition. □

---

[1] By solving we understand that the algorithm provides an answer YES or NO for each input.

3. TASK (1 POINT)

Assignment: Give reason why from statement in point 2 follows: $\mathbf{P} = \mathbf{NP} \Longrightarrow$ each language $\mathbf{L} \in \mathbf{NP}$ is **NP**-complete (with an exception of empty and universal language).

Proof

We want to show that if $\mathbf{P} = \mathbf{NP}$, then every language $\mathbf{L} \in \mathbf{NP}$, except $\mathbf{L} = \emptyset$ and $\mathbf{L} = \Sigma^*$, is **NP**-complete. We remind, that to show **L** is **NP**-complete, we need to show it satisfies two conditions: $\mathbf{L} \in \mathbf{NP}$, and every $\mathbf{A} \in \mathbf{NP}$ is polynomial-time reducible to **L**. The first part should be simple, since we are given language $\mathbf{L} \in \mathbf{NP}$. The second part is a reduction, specifically $\mathbf{A} \leq \mathbf{L}$ for all $\mathbf{A} \in \mathbf{NP}$, the assumption the that $\mathbf{P} = \mathbf{NP}$ will help us with it.

Clearly, $\mathbf{P} = \mathbf{NP}$ and $\mathbf{A} \in \mathbf{NP}$ implies that $\mathbf{A} \in \mathbf{P}$. To check whether input $\mathbf{w} \in \mathbf{A}$, we run the polynomial-time algorithm, that deciding **A**. When algorithm answer `YES`, output constant $\mathbf{w_{acc}} \in \mathbf{L}$, otherwise, output constant $\mathbf{w_{rej}} \notin \mathbf{L}$. Strings $\mathbf{w_{acc}}$ and $\mathbf{w_{rej}}$ necessarily exist because both **L** and $\overline{\mathbf{L}}$ are non-empty. Since the outputs $\mathbf{w_{acc}}$ and $\mathbf{w_{rej}}$ are constant and the deciding algorithm runs in the polynomial-time, this reduction is also in the polynomial-time. Therefore, we can conclude, that we have proved that **L** is **NP**-complete.

Empty and universal language

We show why it is needed $\mathbf{L} \neq \emptyset$ and $\mathbf{L} \neq \Sigma^*$ to solve this problem. When this assumption is satisfied, then we have a guarantee that there exists two different instances $\mathbf{w_1} \in \mathbf{L}$ and $\mathbf{w_2} \notin \mathbf{L}$. Both trivial languages ($\emptyset$ and Σ^*) cannot be **NP**-complete, because they do not accomplished the condition of the reduction. To reduce a language $\mathbf{A} \in \mathbf{NP}$ to a given language **L**, we need to map instances in **A** to instances in **L** and those outside **A** to outside **L**. However, when $\mathbf{L} = \emptyset$, then there are no instances in **L** and when $\mathbf{L} = \Sigma^*$ there are no instances outside **L**. Therefore, there cannot be constructed the reduction from any language $\mathbf{A} \neq \emptyset$, respectively $\mathbf{A} \neq \Sigma^*$.

Reasoning

The **NP**-complete problems allow us to solve any other **NP**-problem in polynomial time. When $\mathbf{P} = \mathbf{NP}$, then we can already solve any **NP**-problem in polynomial time, thus all **NP**-problems are **NP**-complete.

---

4. TASK (3.5 POINT)

Assignment: Let us consider the GRAPH_COLOURING problem defined in slides of series no. 5. Let us define optimisation problem OPT_GRAPH_COLOURING as follows: For a graph $G = (V,E)$ and a finite set of colours C, a feasible solution is any mapping set $A : V \to C$. The cost of the solution is defined as $c(A) = |\{(v_1,v_2) \in E \,|\, A(v_1) = A(v_2)\}|$, i.e. the number of edges with adjacent vertices coloured by an equal colour. The optimal solution is the one with a minimal cost. Prove that if $\mathbf{P} \neq \mathbf{NP}$ then there is no absolute approximation for OPT_GRAPH_COLOURING.

Problem definition

GRAPH COLOURING: Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and $\mathbf{p} \in \mathbb{N}$, can the vertices of $\mathbf{G}$ be coloured using $\mathbf{p}$ colours such that no two adjacent vertices are assigned the same colour?

OPTIMISATION PROBLEM: Each optimisation problem is defined as a triple $(\mathbf{I}, \mathbf{F}, \mathbf{c})$, where $\mathbf{I}$ is a set of *instances* (possible inputs), $\mathbf{F}(\mathbf{x})$ is a set of *feasible solutions* for the instance $\mathbf{x} \in \mathbf{I}$ and $\mathbf{c} : \mathbf{F}(\mathbf{x}) \to \mathbf{Q}^+$ is a *cost function*.

OPTIMAL GRAPH COLOURING: Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and $\mathbf{p} \in \mathtt{N}$ as the instance $\mathbf{I}$ of this optimisation problem (possible input). Feasible solutions for the instance $\mathbf{x} \in \mathbf{I}$ is the set of any mapping set $\mathbf{F}(\mathbf{x}) = \{\mathbf{A} \,|\, \mathbf{A} : \mathbf{V} \to \mathbf{C}\}$. The cost function $\mathbf{c} : \mathbf{F}(\mathbf{x}) \to \mathbf{Q}^+$ of the individual solution is defined as $\mathbf{c}(\mathbf{A}) = |\{(\mathbf{v_1}, \mathbf{v_2}) \in \mathbf{E} \,|\, \mathbf{A}(\mathbf{v_1}) = \mathbf{A}(\mathbf{v_2})\}|$. The optimal solution of this optimisation problem is the one with a minimal cost, thus it represents the *Minimisation problem*. It has given the instance $\mathbf{x} \in \mathbf{I}$ and it wants to find a feasible solution $\mathbf{r} \in \mathbf{F}(\mathbf{x})$ such that $\forall \mathbf{q} \in \mathbf{F}(\mathbf{x}) : \mathbf{c}(\mathbf{r}) \leq \mathbf{c}(\mathbf{q})$. The price of *optimal solution* is denoted as **OPT(x)**.

Absolute Approximation Algorithm: $\mathbf{A}$ is a **k-absolute approximation algorithm** for the optimisation problem Π, when exists a constant $\mathbf{k} > \mathbf{0}$ such, that for instances $\forall \mathbf{x} \in \Pi$ is valid that: $|\mathbf{c}(\mathbf{A}(\mathbf{x})) - \mathtt{OPT}(\mathtt{x})| \leq \mathbf{k}$. The number $\mathbf{k}$ is called the (absolute) error of the algorithm $\mathbf{A}$.

Proving by Scaling method

It is proved, that GRAPH_COLOURING problem is **NP**-complete for $\mathbf{p} \geq \mathbf{3}$, which implies that it is also **NP**-hard for such instance. In fact, for most of **NP**-hard problems, we can prove that an absolute approximation algorithm cannot exist unless **P equals NP**. Such proofs us a technique called **scaling**. In this method, we first increase (scale) certain parameters of the problem instance. Subsequently, we show that if an absolute approximation algorithm existed, the solution it would provide for the modified instance could be re-scaled to yield an optimum solution for the original instance. However, this would imply the existence of an efficient algorithm for an **NP**-hard problem, and this **P would equal NP**.

Solution

Theorem. There is no k-absolute approximation algorithm for OPTIMAL_GRAPH_COLOURING, for any number $\mathbf{k}$ when $\mathbf{P} \neq \mathbf{NP}$.

Lemma. The existence of the deterministic polynomial-time algorithm for an arbitrary **NP**-hard optimisation problem Π implies the existence of the polynomial-time algorithm to solve **NP**-complete language associated with it. For the given **NP**-hard optimisation problem Π thus exists the polynomial-time algorithm, which for each instance determines the optimal solution, only with the assumption that $\mathbf{P} = \mathbf{NP}$.

Proof. Suppose for the sake of contradiction that we have a k-absolute approximation algorithm $\mathbf{A}$ for OPTIMAL_GRAPH_COLOURING whose absolute error is $\mathbf{k}$, where $\mathbf{k} \in \mathbb{N}$. With use this algorithm we will design the new (polynomial) algorithm, which for each instance of the OPTIMAL_GRAPH_COLOURING problem finds its optimal solution. By application the above-defined Lemma we get $\mathbf{P} = \mathbf{NP}$, so we have a contradiction with our assumption. Although there are no numbers within the problem instance to scale up, there is still a scaling trick we can use.

The searched polynomial-time algorithm for the given instance $(\mathbf{G}, \mathbf{p})$ of the OPTIMAL_GRAPH_COLOURING problem firstly constructs the new instance $(\mathbf{G}', \mathbf{p})$ and then simulates the computation of the algorithm $\mathbf{A}$ on the input $(\mathbf{G}', \mathbf{p})$. For an instance $\mathbf{G}$, we make a new instance $\mathbf{G}'$ out of $\mathbf{k}+\mathbf{1}$ copies of $\mathbf{G}$ that are **not** connected to each other. Then each copy of $\mathbf{G}$ can be coloured individually with the same number of colour within all copies such, that the *cost function* within each copy will be equal. Then an overall *cost function* of $\mathbf{G}'$ is composed of partial sub-results within each copy, and in particular, $\mathtt{OPT}((\mathtt{G}', \mathtt{p}))$ is $\mathbf{k}+\mathbf{1}$ times as large as $\mathtt{OPT}((\mathtt{G}, \mathtt{p}))$.

The algorithm $\mathbf{A}$ on the input $(\mathbf{G}', \mathbf{p})$ computes the solution $\mathbf{A}((\mathbf{G}', \mathbf{p}))$. Since the absolute error of the algorithm $\mathbf{A}$ is $\mathbf{k}$, then is valid: $|\,\mathbf{c}(\mathbf{A}((\mathbf{G}', \mathbf{p}))) - \mathtt{OPT}((\mathtt{G}', \mathtt{p}))\,| \leq \mathbf{k} = |\,\mathbf{c}(\mathbf{A}((\mathbf{G}', \mathbf{p}))) - (\mathbf{k}+\mathbf{1})\,\mathtt{OPT}((\mathtt{G}, \mathtt{p}))\,| \leq \mathbf{k}$ and therefore we

can conclude, that the value of the *cost function* $\mathbf{c}(\mathbf{A}((\mathbf{G}', \mathbf{p})))$ is at most $(\mathbf{k}+\mathbf{1})\,\texttt{OPT}(\texttt{G})+\mathbf{k}$. This approximation consists of independent coloured sub-graphs which represents each copy of $\mathbf{G}$. Just one of the copies must represent the solution of the `OPTIMAL_GRAPH_COLOURING` problem (i.e., its value of *cost function* is equal to `OPT((G,p))`), because otherwise, the total value of partial *cost functions* within sub-graphs in the approximation would be at least $(\mathbf{k}+\mathbf{1})\,\texttt{OPT}(\texttt{G})+(\mathbf{k}+\mathbf{1})$, which is more than the maximum possible value $(\mathbf{k}+\mathbf{1})\,\texttt{OPT}(\texttt{G})+\mathbf{k}$. Therefore, it is in contradicting with the assumption, that we have the *k*-absolute approximation algorithm $\mathbf{A}$. Hence, we can obtain an exact solution to `OPTIMAL_GRAPH_COLOURING` problem in polynomial time, which we presume is impossible.

Let us summarise it at the end, as we have said above, we know that the optimal solution must belong to the interval $<(\mathbf{k}+\mathbf{1})\,\texttt{OPT}(\texttt{G}), (\mathbf{k}+\mathbf{1})\,\texttt{OPT}(\texttt{G})+\mathbf{k}>$. Since we solve the *minimisation* problem we do not have to consider the first half of the interval $<(\mathbf{k}+\mathbf{1})\,\texttt{OPT}(\texttt{G})-\mathbf{k}, (\mathbf{k}+\mathbf{1})\,\texttt{OPT}(\texttt{G}))$, because there would be a sub-optimal solution of this problem. Since the solution must be a multiple of $\mathbf{k}+\mathbf{1}$, then there exists only one value from this interval. Thus we can choose the optimal solution for that new instance $(\mathbf{G}', \mathbf{p})$ from this interval and then determine the solution for the original instance from it. □
