1. Uvažujme abecedu Σ, t.ž., symbol $R \notin \Sigma$, a následující kódování deterministického konečného automatu do Turingova stroje: pro $A = (Q, \Sigma, \delta, q_0, F)$ sestrojíme TS $M_{sim}(A) = (Q \cup \{q_0^M, q_f^M\}, \Sigma, \Sigma \cup \{\Delta\}, \delta_M, q_0^M, q_f^M)$, kde $Q \cap \{q_0^M, q_f^M\} = \emptyset$ a δ_M je definována následovně:

   - $\delta_M(q_0^M, \Delta) = (q_0, R)$
   - $\forall f \in F : \delta_M(f, \Delta) = (q_f^M, R)$
   - $\delta_M(q, a) = (p, R) \Leftrightarrow \delta(q, a) = p$

   Množina kódů turingových strojů vzniklých transfomací DKA $KA = \{\langle M\rangle \mid M = M_{sim}(A)$ pro nějaký DKA $A\}$ je rozhodnutelná, protože lze jednoduše ověřit tvar přechodové funkce.

   Rozhodněte a dokažte, zda následující jazyky jsou (resp. nejsou) rekurzivní (resp. rekursivně vyčíslitelné):

   - $L_1 = \{\langle M\rangle\#\langle w\rangle \mid w \notin L(M) \wedge \langle M\rangle \in KA\}$
   - $L_2 = \{\langle M\rangle\#\langle w\rangle \mid w \notin L(M) \wedge \langle M\rangle \notin KA\}$

   20 bodů

Jazyk L_1:

- Pro rozhodnutí L_1 sestavíme Turingův stroj T_{L_1}, který pracuje následovně:
  1. T_{L_1} zkontroluje, zda je jeho vstup ve formátu ⟨*M*⟩#⟨*w*⟩, kde ⟨*M*⟩ je kód DKA a ⟨*w*⟩ je kód řetězce *w*. Pokud není, T_{L_1} odmítne.
  2. T_{L_1} ověří, zda ⟨*M*⟩ ∈ *KA*, tj. zda *M* reprezentuje DKA. Pokud ne, T_{L_1} odmítne.
  3. T_{L_1} spustí simulaci stroje *M* na *w*.
  4. Pokud *M* přijme *w*, T_{L_1} odmítne. Pokud *M* odmítne *w* (tj. *w* ∉ *L(M)*), T_{L_1} přijme.
- Všechny kroky jsou algoritmicky proveditelné, protože kontrola, zda ⟨*M*⟩ ∈ *KA*, je rozhodnutelná, a rozhodnutí, zda *w* ∉ *L(M)* pro DKA, je také rozhodnutelné.
- Jazyk L_1 **je tedy rekurzivní (rozhodnutelný)**.

Jazyk L_2 (kostra přebrána z redukce_příklady.pdf):

- Nemůžeme obecně rozhodnout, zda *M* ∉ *KA*, ani zda *w* ∉ *L(M)*, protože rozhodování o příslušnosti *w* k jazyku obecného Turingova stroje *M* zahrnuje problém zastavení. Jazyk tedy není rekurzivně vyčíslitelný. Dokažme to níže.
- Jazyk L_2 není rekurzivně vyčíslitelný, což dokážeme redukcí z co-HP, tedy ukážeme, že co-HP ≤ L_2.
- Redukční funkce ψ bude mít následující signaturu: ψ : {0, 1, #}* → {0, 1, #}*.
- Pokud vstup redukční funkce nebude ve tvaru ⟨*M*⟩#⟨*w*⟩, funkce ψ vrátí pevný řetězec ⟨M_0⟩#⟨w_0⟩, kde M_0 je pevně zvolený DKA a w_0 ∈ *L*(M_0). Tím je zajištěno, že ⟨M_0⟩#⟨w_0⟩ ∉ L_2, neboť M_0 ∈ *KA*.
- V případě, že vstup redukční funkce je ve tvaru ⟨*M*⟩#⟨*w*⟩, funkce vytvoří nový kód ⟨*M'*⟩#⟨*w*⟩, kde *M'* je Turingův stroj, který pracuje následovně:
  1. Turingův stroj *M'* ignoruje svůj vstup a místo toho simuluje *M* na *w*.
  2. Pokud *M* cyklí na *w*, *M'* cyklí také.
  3. Pokud *M* zastaví na *w*, *M'* se zachová podle výsledku:
     - Pokud *M* přijme *w*, *M'* cyklí.
     - Pokud *M* odmítne *w*, *M'* přijme.
- Tím je zajištěno, že *M'* ∉ *KA*, protože *M'* není deterministický konečný automat.
- Pokud ⟨*M*⟩#⟨*w*⟩ ∈ co-HP, pak *M* nezastaví na *w*. Stroj *M'* v tomto případě cyklí na všech vstupech, což splňuje podmínky ⟨*M'*⟩#⟨*w*⟩ ∈ L_2.

- Pokud $\langle M \rangle\#\langle w \rangle \notin$ co-HP, tedy *M* zastaví na *w*, výstupní instance $\langle M' \rangle\#\langle w \rangle \notin L_2$, protože *M′* přijme *w*, což porušuje podmínky jazyka L_2.
- Můžeme tedy psát: $\langle M \rangle\#\langle w \rangle \in$ co-HP ⇔ $\psi(\langle M \rangle\#\langle w \rangle) \in L_2$.
- Redukce je validní, protože ψ je totální, rekurzivně vyčíslitelná funkce zachovávající příslušnost. Protože co-HP není rekurzivně vyčíslitelný, jazyk L_2 také není rekurzivně vyčíslitelný.
- Jazyk L_2 **není rekurzivní ani rekurzivně vyčíslitelný**.

2. Jan a Eliška si vymysleli novou hru. Mají barevné křídy o b ($b \geq 2$) barvách. Na chodník si nakreslili křídou kolečka a některá z nich propojili čárami. Teď by rádi do každého kolečka namalovali x ($x \geq 2$) barevných značek (barvy značek v jednom kolečku se mohou opakovat) tak, aby kolečka propojená čárou nebyla označená stejně. Pořadí značek v kolečku nehraje roli. Otázka zní, jestli je takového označení možné.

   Formálně definujme hru Jana a Elišky jako n-tici $H = (K, C, b, x)$, kde

   - K je konečná množina koleček,
   - $C \subseteq \{\{a, b\} \mid a, b \in K\}$ je množina čar,
   - $b \geq 2$ je počet barev,
   - $x \geq 2$ požadovaný počet značek.

   Hra H má řešení, pokud existuje zobrazení $O : K \times \langle 1, b \rangle \to \mathbb{N}$ takové, že

   - $\forall a \in K : \sum_{i=1}^{b} O(a, i) = x$ (počet značek v jednom kolečku je roven x)
   - $\forall \{a, b\} \in C\ \exists i : O(a, i) \neq O(b, i)$ (označení dvojice míst spojených čárou se liší)

   Dokažte, že problém existence řešení pro hru H je NP-úplný.

   (Pozn: Pomůže Vám NP-úplnost některého z problémů uvedených zde:
   `https://en.wikipedia.org/wiki/NP-completeness#NP-complete_problems`
   v odstavci „*NP-complete problems*".) 15 bodů

- Pro prokázání, že problém existence řešení pro hru *H* je NP-úplný, je třeba provést následující kroky:
  1. Prokázat, že problém *H* náleží do třídy NP.
  2. Provést redukci známého NP-úplného problému na problém *H*.

1. Problém *H* = (*K, C, b, x*) patří do třídy NP, protože je možné ověřit platnost navrhovaného řešení (tj. konkrétní přiřazení značek jednotlivým kolečkům) v polynomiálním čase vzhledem k velikosti vstupu. Proces ověření probíhá ve dvou fázích:
   a. Kontrola počtu značek v jednotlivých kolečkách:
      - Pro každé kolečko *a* ∈ *K* spočítáme součet hodnot indikátorových funkcí $O(a,\ i)$ (*pro i = 1, …, b*), které označují, zda kolečku *a* byla přiřazena barva *i*. Tento součet by měl být roven požadovanému počtu značek *b*. Tedy ověřujeme, že platí:
        $$\sum_{i=1}^{b} O(a,\ i)\ =\ x$$
      - Tento krok lze provést s časovou složitostí $O(|K| \cdot b)$
   b. Ověření konfliktů mezi sousedními kolečky:
      - Pro každou dvojici sousedních koleček (*a, b*) ∈ *C* (tedy pro každou hranu grafu) ověříme, že kolečka *a* a *b* nemají žádnou společnou barvu. Formálně kontrolujeme, že pro každou barvu *i* platí:
        $O(a,\ i)\ +\ O(b,\ i)\ =$ 0 (pro každou barvu *i*)
      - Tento krok má časovou složitost $O(|C| \cdot b)$

   Protože oba kroky jsou proveditelné v polynomiálním čase vzhledem k velikosti vstupu, problém *H* splňuje podmínky pro zařazení do třídy NP.

2. Známý NP-úplný problém:
   - 3-obarvitelnost grafu (3-coloring):
     - Máme graf *G* = (*V, E*) a chceme zjistit, zda je možné obarvit jeho vrcholy pomocí tří barev tak, aby žádné dva sousední vrcholy neměly

stejnou barvu. Tento problém je NP-úplný.

Redukce na problém *H*:

- Vezmeme graf *G* = (*V, E*) a převedeme jej na instanci *H* = (*K, C, b, x*) následujícím způsobem:
    a. Množina koleček: Kolečka *K* odpovídají množině vrcholů *V* grafu *G*.
    b. Množina čar: Čáry *C* odpovídají hranám *E* grafu *G*.
    c. Počet barev *b*: *b* = 3, protože se zaměřujeme na 3-obarvitelnost grafu.
    d. Pravidla obarvování: Každé kolečko může mít maximálně jednu ze tří barev. Pokud existuje mezi dvěma kolečky (odpovídajícími vrcholům) hrana, kolečka nesmí mít stejnou barvu.

- Pokud je graf *G* 3-obarvitelný: Každý vrchol grafu lze obarvit jednou ze tří barev, a pro každé dvojice sousedních vrcholů (tedy koleček *a* a *k*) bude platit, že mají různé barvy. Toto rozdělení odpovídá pravidlům problému *H*.
- Pokud graf *G* není 3-obarvitelný: Není možné obarvit graf tak, aby každý pár sousedních vrcholů měl různé barvy, což znamená, že není možné přiřadit barvy kolečkům tak, aby splňovaly podmínky problému *H*.
- Problém *H* je v NP, protože ověření jeho řešení lze provést v polynomiálním čase. Redukce z 3-obarvitelnosti na problém *H* ukazuje, že problém *H* je NP-těžký. Protože *H* je zároveň v NP, je **NP-úplný**.

3. Uvažujeme funkci get_next, která má na vstupu řetězec nad abecedou $\Sigma = \{A, ..., Z\}$ a jeho délku l. Funkce vrácí následující řetězec vzhledem k lexikografickému uspořádání. Funkce $next\_char$ vrací následující znak latinské abecedy. Analyzujte a zdůvodněte amortizovanou časovou složitost libovolné posloupnosti n operací $str := get\_next(str, l)$. Na začátku je zafixována konstanta $l > 0$ a počáteční hodnota $str = A^l$ (řetězec obsahující l symbolů A).

Předpokládejme uniformní cenové kriterium, kde každý řádek má cenu 1 (vyjímkou je řádek 7 s cenou 0).

```
1  Function get_next(char [] str, int l)
2      fin := false;
3      while ¬(fin) ∧ l > 0 do
4          l := l − 1;
5          if str[l] = Z then
6              str[l] := A;
7          else
8              next_char(str[l]);
9              fin := true;
10     return str;
```

15 bodů

- V zadání je explicitně řečeno, že funkce začíná s libovolně dlouhou posloupností symbolů *A*.
- To znamená, že bude trvat minimálně 26 iterací (za předpokladu použití anglické abecedy Σ bez diakritiky), než se narazí na symbol *Z*.
- Při každém volání funkce *get_next* se vždy vykonají řádky 2 a 10 (**cena: 2**)
- Pokud však symbol na pozici *str[l]* není *Z* (tedy v případě větve **else**), vykonají se řádky 3, 4, 5, 8, 9 a znovu 3 (**cena: 6**).
- Naopak, pokud je na této pozici symbol *Z*, provádí se pouze řádky 3, 4, 5, 6 (**cena: 4)**
- Z toho plyne, že náklady na případ, kdy se vyskytne symbol *Z* (4), musíme předplatit během předchozích 25 iterací. Tedy náklady jsou rovny $\frac{4}{25}$ na každou iteraci.
- Celkový počet iterací je tedy roven:
  n · max{všechny hodnoty ze sloupce **kredity**} = (2 + 6 + $\frac{4}{25}$) · n
- Pro lepší názornost je zde vložena tabulka:

Operace	**Cena** (aktuální volání)	**Kredity** (předplacení)
Inicializace	2	2
A → *X*, kde $X \in \Sigma - \{Z\}$	6	6 + $\frac{4}{25}$
A → *Z*	4	0

4. Uvažujme funkci *find_suffix*, která má na vstupu pole čísel $array$ o velikosti $size$ (chybné vstupy neuvažujte) a která se snaží nalést v rámci pole suffix takový, že součet čísel v tomto suffixu je roven hodnotě $final$.

- Analyzujte časovou složitost funkce *find_suffix* v nejlepším případě
- Analyzujte časovou složitost funkce *find_suffix* v nejhorším případě.
- Navrhněte funkci *find_opt*, která bude dávat stejný výsledek jako funkce *find_suffix*, ale bude mít lepší asymptotickou složitost v nejhorším případě.
- Analyzujte časovou složitost funkce *find_opt* v nejhorším případě.

Předpokládejme uniformní cenové kriterium, kde každý řádek má cenu 1.

```
1  Function find_suffix(int * array, int size, int final)
2      int i, j;
3      i := 0;
4      while i < size do
5          j := i;
6          int tmp := 0;
7          while j < size do
8              tmp := tmp + array[j];
9              j := j + 1;
10         if tmp = final then
11             return ANO
12         i := i + 1;
13     return NE
```

15 bodů

- **Nejlepší případ** časové složitosti funkce *find_suffix* je $O(size)$ a nastává tehdy, když součet prvků celého pole o délce *size* je roven hodnotě *final*. V tomto případě se vykoná vnitřní smyčka (řádky 7–9) a při první kontrole **if** *tmp* = *final* se podmínka splní (řádek 10). Funkce následně vrátí hodnotu *ANO* (řádek 11) a tím okamžitě končí, protože byl nalezen prefix, jehož součet je roven hodnotě *final.*
- **Nejhorší případ** časové složitosti funkce *find_suffix* je $O(size^2)$ a nastává, když poslední prvek pole *array* o délce *size* je roven hodnotě *final*. Algoritmus v tomto případě projde všechny možné suffixy počínaje indexem *i=0* až po *i=size-1*, přičemž pro každý z nich počítá sumu (vnitřní smyčka **while** se vykonává od *j=i* po *j=size*). Celkový počet iterací je roven:
  $$\sum_{i=0}^{size-1} (size - i) = (size - 0) + (size - 1) + \ldots + (size - (size - 1)) = \frac{size\cdot(size+1)}{2}$$
  Tento výraz vyjadřuje kvadratickou časovou složitost, což znamená, že v nejhorším případě funkce projde všechny možné suffixy a spočítá jejich součet, což má složitost $O(size^2)$.
- **Funkce** *find_opt*:

```python
def find_opt(array, final):
    tmp = 0  # Akumulátorový součet
    for i in range(len(array) - 1, -1, -1):  # Procházíme pole od konce
        tmp += array[i]  # Přičítáme aktuální prvek
        if tmp == final:  # Kontrola výsledku
            return 1  # Nalezeno
    return 0  # Nenalezeno
```

- **Nejhorší případ** časové složitosti funkce *find_opt* je $O(size)$ a nastává, když součet všech prvků pole *array* není roven hodnotě *final*, což znamená, že funkce musí projít celé pole, aby zjistila, že žádný suffix nemá součet, který by se rovnal hodnotě *final*.

5. Mějme teorii T se signaturou $\langle\{Kral_{/0}, Dama_{/0}, Vez_{/0}, Kun_{/0}, Pesec_{/0}\}, \{ohrozuje_{/2}, =_{/2}\}\rangle$ (= je standardní rovnost) se speciálními axiomy

$$\forall x(x = Kral \lor x = Dama \lor x = Vez \lor x = Kun \lor x = Pesec)$$
$$\forall x\, \neg ohrozuje(x, Kral)$$
$$\forall x, y(ohrozuje(x, y) \land ohrozuje(y, x) \Rightarrow x \neq Kun)$$
$$\forall x(ohrozuje(Pesec, x) \Rightarrow ohrozuje(x, Pesec) \lor x = Kun)$$

i) Rozhodněte a stručně zdůvodněte, zda T je: a) bezesporná, b) úplná a c) rozhodnutelná (tj. množina důsledků T je rozhodnutelná).

15 bodů

a) Bezespornost:
- Teorie je bezesporná, pokud neexistuje žádná formule ϕ, pro kterou $T \vdash \phi$ a zároveň $T \vdash \neg\phi$ .Teorie je bezesporná právě tehdy, když má model (viz přednáška).
- Po prozkoumání všech čtyř axiomů se nejeví, že by teorie obsahovala jakékoli vnitřní rozpory nebo protichůdnosti. Proto se pokusíme vytvořit interpretační model. Příkladem interpretačního modelu může být následující: I_1: D = {Kral, Dama, Vez, Kun, Pesec}

  α(Kral) = {Kral}, α(Dama) = {Dama}, α(Vez) = {Vez}, α(Kun) = {Kun}, α(Pesec) = {Pesec}

  α(ohrozuje): ohrozuje(Kral, Kral) = ∅
  ohrozuje(Kral, Dama) = {Kral, Dama}
  ohrozuje(Kral, Vez) = {Kral, Vez}
  ohrozuje(Kral, Kun) = {Kral, Kun}
  ohrozuje(Kral, Pesec) = {Kral, Pesec}

  ohrozuje(Dama, Kral) = ∅ (splnění axiomu č. 2)
  ohrozuje(Dama, Dama) = ∅
  ohrozuje(Dama, Vez) = {Dama, Vez}
  ohrozuje(Dama, Kun) = {Dama, Kun}
  ohrozuje(Dama, Pesec) = {Dama, Pesec}

  ohrozuje(Vez, Kral) = ∅ (splnění axiomu č. 2)
  ohrozuje(Vez, Dama) = {Vez, Dama}
  ohrozuje(Vez, Vez) = ∅
  ohrozuje(Vez, Kun) = {Vez, Kun}
  ohrozuje(Vez, Pesec) = {Vez, Pesec}

  ohrozuje(Kun, Kral) = ∅ (splnění axiomu č. 2)
  ohrozuje(Kun, Dama) = ∅ (splnění axiomu č. 3)
  ohrozuje(Kun, Vez) = ∅ (splnění axiomu č. 3)
  ohrozuje(Kun, Kun) = ∅
  ohrozuje(Kun, Pesec) = {Kral, Pesec}

  ohrozuje(Pesec, Kral) = ∅ (splnění axiomu č. 2)
  ohrozuje(Pesec, Dama) = {Pesec, Dama}
  ohrozuje(Pesec, Vez) = {Pesec, Vez}

ohrozuje(Pesec, Kun) = ∅ (splnění axiomu č. 4)
ohrozuje(Pesec, Pesec) = ∅

- Na základě tohoto modelu můžeme říci, že teorie *T* **je bezesporná**.

b) Úplnost

- Teorie je úplná, pokud pro každou formuli ϕ platí, že buď $T \vdash \phi$, nebo $T \vdash \neg\phi$. To znamená, že pro jakoukoli formuli v rámci teorie musí být buď prokázána její pravdivost, nebo nepravdivost. Dále, teorie je úplná, pokud existuje jediný interpretační model, který splňuje všechny axiomy teorie.
- Axiomy poskytují značný prostor pro interpretaci, přičemž není explicitně určeno, zda konkrétní objekty ohrožují jiné objekty. Z tohoto důvodu je vhodné zaměřit se na otázku, zda existuje více než jeden interpretační model, který je v souladu s těmito axiomy.
- Příklad dalšího validního interpretačního modelu je:
  I_2: *D* = {Kral, Dama, Vez, Kun, Pesec}

  α(Kral) = {Kral}, α(Dama) = {Dama}, α(Vez) = {Vez}, α(Kun) = {Kun}, α(Pesec) = {Pesec}

  α(ohrozuje): ohrozuje(Kral, Kral) = ∅
  ohrozuje(Kral, Dama) = {Kral, Dama}
  ohrozuje(Kral, Vez) = {Kral, Vez}
  ohrozuje(Kral, Kun) = {Kral, Kun}
  ohrozuje(Kral, Pesec) = {Kral, Pesec}

  ohrozuje(Dama, Kral) = ∅ (splnění axiomu č. 2)
  ohrozuje(Dama, Dama) = ∅
  ohrozuje(Dama, Vez) = {Dama, Vez}
  ohrozuje(Dama, Kun) = ∅ (**změna**)
  ohrozuje(Dama, Pesec) = {Dama, Pesec}

  ohrozuje(Vez, Kral) = ∅ (splnění axiomu č. 2)
  ohrozuje(Vez, Dama) = {Vez, Dama}
  ohrozuje(Vez, Vez) = ∅
  ohrozuje(Vez, Kun) = {Vez, Kun}
  ohrozuje(Vez, Pesec) = {Vez, Pesec}

  ohrozuje(Kun, Kral) = ∅ (splnění axiomu č. 2)
  ohrozuje(Kun, Dama) = ∅ (splnění axiomu č. 3)
  ohrozuje(Kun, Vez) = ∅ (splnění axiomu č. 3)
  ohrozuje(Kun, Kun) = ∅
  ohrozuje(Kun, Pesec) = {Kral, Pesec}

  ohrozuje(Pesec, Kral) = ∅ (splnění axiomu č. 2)
  ohrozuje(Pesec, Dama) = {Pesec, Dama}
  ohrozuje(Pesec, Vez) = {Pesec, Vez}
  ohrozuje(Pesec, Kun) = ∅ (splnění axiomu č. 4)
  ohrozuje(Pesec, Pesec) = ∅
- V důsledku toho teorie *T* **není úplná**.

c) Rozhodnutelnost

- Teorie je rozhodnutelná, pokud existuje algoritmus, který dokáže rozhodnout, zda daná formule ϕ je důsledkem teorie *T*, tedy zda platí $\phi \in \mathrm{Th}(T)$, kde Th(*T*) označuje množinu všech důsledků *T*. Pokud je teorie efektivní, bezesporná a úplná, pak je rozhodnutelná. V případě, že teorie není úplná, ale je efektivní a bezesporná, lze ji považovat za částečně rozhodnutelnou (viz přednáška).
- Teorie *T* se zabývá konečnou množinou objektů – konkrétně pěti šachovými figurami: Král, Dáma, Věž, Kůň a Pěšec. Díky konečné doméně je možné vyhodnotit všechny možné konfigurace relace „ohrožuje" mezi těmito objekty.
- Relace „ohrožuje" je definována jako binární vztah, který se váže na pevně daný počet objektů. Použití rovnosti (=) zjednodušuje proces ověřování tvrzení, protože se jedná o standardní operátor.
- Teorie obsahuje omezený počet axiomů s pevně danou strukturou, což umožňuje vytvořit algoritmus, který ověří, zda dané tvrzení φ z *T* ($T \vdash \varphi$). Algoritmus dokáže systematicky prozkoumat všechny možné situace (např. které figury koho ohrožují) a rozhodnout, zda je axiom nebo formule platná.
- Teorie nevyužívá složité konstrukty jako kvantifikátory přes nekonečné množiny nebo prvky aritmetiky, což by mohlo vést k nerozhodnutelnosti.
- Z toho plyne, že teorie *T* **je rozhodnutelná**.