Rozhodovací problémy

Definice

Rozhodovací problém je takový, kde je množina možných výstupů dvouprvková $\{\text{ANO}, \text{NE}\}$.

- V této a následující přednášce se budeme zabývat výhradně rozhodovacími problémy

Nedeterministický Turingův stroj

Podobný princip jako byl použit pro nedeterministické konečné (resp. zásobníkové) automaty můžeme použít i na Turingovy stroje.

Definice

Nedeterministický Turingův stroj je definován jako šestice $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{ANO}}, q_{\text{NE}})$ kde:

- Q je konečná množina **stavů**
- Γ je konečná množina **páskových symbolů**
- $\Sigma \subseteq \Gamma, \Sigma \neq \emptyset$ je konečná množina **vstupních symbolů**
- $\delta : (Q - F) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{-1, 0, +1\})$ je **přechodová funkce**
- $q_0 \in Q$ je **počáteční stav**
- $q_{\text{ANO}}, q_{\text{NE}} \in Q$ jsou **koncové stavy** (přijímající a nepřijímající)

Nedeterministický Turingův stroj

- Požadujeme, aby každý výpočet skončil buď ve stavu q_{ANO} nebo ve stavu q_{NE}.
- Odpověď nedeterministického TS na vstup w je ANO, jestliže alespoň jeden z možných výpočtů skončí v q_{ANO}.
- Složitost takového nedeterministického stroje můžeme definovat jako počet kroků nejdelšího možného vypočtu stroje nad daným vstupem

Nedeterministický stroj RAM

- Je definován stejně jako deterministický RAM
- Navíc je jen jedna instrukce CHOOSE GOTO x OR GOTO y, která umožňuje stroji vybrat si jedno z možných pokračování
- Pokud by ze všech možných výpočtů takového stroje nad zadaným vstupem alespoň jeden skončil s odpovědí ANO, je odpověď ANO (bez ohledu na další možné výpočty)
- Pokud všechny možné výpočty končí odpovědí NE, je odpověď NE
- Složitost nedeterministického stroje RAM definujeme jako délku nejdelšího možného výpočtu nad zadaným vstupem
- Podobně můžeme definovat nedeterministické verze jiných výpočetních modelů

Nedeterminismus

Na nedeterminismus můžeme nahlížet dvěma způsoby:

1. Ve chvíli, kdy dochází k nejednoznačnému pokračování, se stroj rozhodne pro tu nejlepší vedoucí nejrychleji k odpovědi ANO (pokud taková možnost existuje). Odpověď je ANO, právě když tento jediný výpočet skončí odpovědí ANO.
2. Ve chvíli, kdy dochází k nejednoznačnosti se stroj rozdvojí (případně rozdělí na více identických kopií) a každá kopie pokračuje jedním možným pokračováním. Odpověď je ANO právě tehdy, když alespoň jedna z kopií stroje odpoví ANO.

- Oba přístupy jsou ekvivalentní.
- Možnost 1. uhádne to nejlepší pokračování zatímco možnost 2. jej najde mezi všemi možnými. Pokud je odpověď ne, tak ani při výběru nejlepší varianty ani prozkoumání všech možných variant nikdy nedostaneme odpověď ANO.

Význam nedeterminismu

- Z hlediska rozhodnutelnosti nemá nedeterminismus význam. Pokud nějaký problém řeší nedeterministický RAM nebo TS, tak ho dokáže řešit i deterministický, který postupně vyzkouší všechny možné výpočty.
- Význam je při rozdělování problémů do složitostních tříd, kdy nám díky nedeterminismu vznikají třídy mezi těmi deterministickými.

Nedeterministické třídy složitosti

Definice

Pro funkci $f : \mathbb{N} \to \mathbb{N}$ rozumíme **třídou časové složitosti** $\mathcal{NT}(f)$ množinu těch problémů, které jsou řešeny nedeterministickými RAMy s časovou složitostí v $O(f(n))$.

Definice

Pro funkci $f : \mathbb{N} \to \mathbb{N}$ rozumíme **třídou prostorové složitosti** $\mathcal{NS}(f)$ množinu těch problémů, které jsou řešeny nedeterminictickými RAMy s prostorovou složitostí v $O(f(n))$.

Třída NPTIME

Definice

$$\text{NPTIME} = \cup_{k=0}^{\infty} \mathcal{NT}(n^k)$$

- Existují rozhodovací problémy, pro které známe exponenciální algoritmus (s polynomiální prostorovou složitostí), ale nevíme, jestli existuje polynomiální algoritmus.
- Mnoho takových problémů patří právě do třídy NPTIME
- Každý problém patřící do PTIME patří i do NPTIME– algoritmus řešící problém z PTIME můžeme prohlásit za nedeterministický (který má 0 instrukcí CHOOSE) a složitost takového nedeterministického algoritmu je stejná jako původního deterministického, protože nejdelší větev výpočtu je ta jediná možná, která určovala i složitost deterministického.

Třída NPTIME

Problém „Barvení grafu k barvami“

Vstup: Neorientovaný graf G a přirozené číslo k

Výstup: ANO, pokud lze vrcholy grafu obarvit k barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu. NE pokud to takto obarvit nelze.

- RAM prochází postupně všechny vrcholy a u každého má volbu z k barev. Po přiřazení barvy zkontroluje sousední vrcholy, jestli již některý nemá přiřazenu stejnou barvu.
- Pokud má být odpověď ANO, tak existuje správné obarvení. Jestliže se při každé volbě mezi více barvami stroj rozdělí na několik kopií, tak ta kopie, kde se přiřadily barvy odpovídající správnému řešení odpoví ANO a tedy i celý nedeterministický RAM.
- Pokud má být odpověď NE, tak žádné možné přiřazení nesplňuje požadavky na barvení a všechny kopie RAMu musí odpovědět NE, tedy i nedeterministický RAM odpoví NE.

Třída NPTIME

Problém „Barvení grafu k barvami“

Vstup: Neorientovaný graf G a přirozené číslo k

Výstup: ANO, pokud lze vrcholy grafu obarvit k barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu. NE pokud to takto obarvit nelze.

- Počet kroků kterékoliv kopie algoritmu odpovídá počtu vrcholů a hran v grafu. Složitost tohoto algoritmu je tedy polynomiální.
- Problém barvení grafu tedy patří do třídy NPTIME.
- Nejlepší známý deterministický algoritmus pracuje v exponenciálním čase a polynomiálním prostoru.

Třídy složitosti

- Podobně můžeme definovat třídu NPSPACE.
- Obdobně jako v deterministickém případě zřejmě platí NPTIME $\subseteq$ NPSPACE.
- Již dříve jsme zdůvodnili PTIME $\subseteq$ NPTIME.
- Nedeterministický RAM můžeme simulovat deterministickým, který zkusí všechny možné výpočty. Každý z možných výpočtů použije maximálně polynomiálně mnoho paměti a jednotlivé výpočty si nic nepředávájí, takže můžeme používat pro všechny stejnou oblast paměti. Platí tedy NPTIME $\subseteq$ PSPACE.
- Savitch ukázal, že pokud je problém rozhodován nedeterministickým Turingovým strojem s prostorovou složitostí $f(n)$, tak je rozhodován i nějakým deterministickým Turingovým strojem s prostorovou složitostí $f^2(n)$. Z toho plyne NPSPACE $\subseteq$ PSPACE.

$$\text{PTIME} \subseteq \text{NPTIME} \subseteq \text{PSPACE} = \text{NPSPACE}$$

Třída NPTIME

Třída NPTIME je nejdůležitější z tříd definovaných s pomocí nedeterminismu, protože obsahuje mnoho praktických problémů. Tuto třídu můžeme alternativně definovat následujícím způsobem.

Definice

NPTIME je třídou všech rozhodovacích problémů $\mathcal{P}$ se vstupem x takových, že existuje algoritmus $\mathcal{A}$ splňující:

- Složitost $\mathcal{A}$ je v $O(p(|x|))$.
- Vstupem $\mathcal{A}$ je (x, y), kde $|y| \in O(p(|x|))$.
- Odpověď P na x je ANO právě tehdy, když existuje y takové, že $\mathcal{A}$ vrací ANO pro vstup (x, y).

- y je jakási "nápověda", která se často nazývá **svědek**.
- Není důležité, jak je těžké svědka najít. Jen když je znám, musí běžet algoritmus v polynomiálním čase.

Třída NPTIME

Problém „Isomorfismus grafů“

Vstup: Neorientované grafy $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$

Výstup: ANO, jestliže jsou grafy isomorfní, tedy existuje-li zobrazení $\rho : V_1 \to V_2$ takové, že $(x, y) \in E_1 \Leftrightarrow (\rho(x), \rho(y)) \in E_2$. NE pokud takové zobrazení neexistuje.

Svědkem je v tomto případě zobrazení ρ. Algoritmus pouze ověří, jestli hrany mezi vrcholy v G_1 odpovídají příslušným hranám v G_2.

Třída NPTIME

Problém „loupežníka“

Vstup: Množina přirozených čísel $A = \{a_1, a_2, \ldots, a_n\}$

Výstup: ANO, jestliže je možné najít množinu X tak, že $X \subset A$, $\sum_{a_i \in X} a_i = \sum_{a_j \in A - X} a_j$.

Svědkem je v tomto případě množina X a algoritmus kontroluje, že je podmnožinou A a součty se rovnají.

Třída NPTIME

- Obě defnice třídy NPTIME (deterministický algoritmus s využitím svědků a nedeterministický algoritmus) jsou ekvivalentní
- Převod z nedeterministického algoritmu na deterministický se svědky
  - Pokud máme nedeterministický algoritmus, tak se může rozhodovat mezi více variantami a vybere vždy tu vedoucí nejrychleji k řešení.
  - Rozhoduje se maximálně polynomiálně často (víc kroků ani neudělá) a tak můžeme jako svědka zvolit posloupnost voleb algoritmu.
  - Potom nám již stačí deterministický algoritmus, který díky svědkovi vždy ví, jak pokračovat.
- Převod z deterministického algoritmu se svědky na nedeterministický
  - Nedeterministický algoritmus si na začátku může nedeterministicky nagenerovat svědka (v polynomiálním čase)
  - Dále již nedeterministický algoritmus pracuje s vygenerovaným svědkem stejně, jako deterministický

Polynomiální převoditelnost

Definice

Problém P_1 je **polynomiálně převeditelný** na problém P_2, jestliže existuje algoritmus s polynomiální časovou složitostí, který pro libovolný vstup w problému P_1 sestrojí vstup w' problému P_2, přičemž platí, že odpověď na otázku problému P_1 pro vstup w je stejná jako odpověď na otázku problému P_2 pro vstup w'.

- Pokud převedeme polynomiálním algoritmem w na w', kde $|w| = n$ a $|w'| = m$, tak $m \in O(n^k)$. Pokud je $P_2 \in \text{PTIME}$, tak jeho složitost je v $O(m^l)$. Pokud převedeme w na w' a na to spustíme algoritmus pro P_2 vyřešíme v $O((n^k)^l) = O(n^{kl}) = O(n^r)$ problém P_1. Tedy $P_1 \in \text{PTIME}$.
- Podobně, převedeme-li P_1 na P_2 a $P_2 \in \text{NPTIME}$, tak i $P_1 \in \text{NPTIME}$

Polynomiální převoditelnost

S několika příklady převodu jste se setkali v předmětu Diskrétní matematika.

Tok v síti

Vstup: Orientovaný ohodnocený graf s vyznačenou dvojicí vrcholů zdroj a stok a konstanta l

Výstup: ANO, pokud existuje v grafu tok o velikosti l, NE jinak

Párování v bipartitním grafu

Vstup: Neorientovaný bipartitní graf a konstanta k

Výstup: ANO, pokud existuje párování o velikosti k (k hran takových, že žádné dvě z nich nemají stejný koncový vrchol)

Polynomiální převoditelnost

- Je znám polynomiální algoritmus řešící problém toku v síti
- Problém párování dokážeme převést na problém toku v síti v polynomiálním čase (dokonce lineárním) a tedy složením dvou algoritmů (převodu a řešení problému toku) dostaneme algoritmus řešící problém párování v polynomiálním čase.
- Při převodu si označíme partity grafu, který je vstupem pro párování, jako V_1 a V_2. Přidáme nové dva vrcholy z a s. Přidáme orientované hrany ze z do každého vrcholu z V_1. Hrany mezi vrcholy z V_1 a V_2 budou mít orientaci ve směru z V_1 do V_2. Z každého vrcholu z V_2 přidáme orientovanou hranu do s.

NP-úplnost

Definice

Problém Q je NP-**těžký**, pokud každý problém ve třídě NP (neboli NPTIME) lze na Q převést polynomiálním převodem.

Definice

Problém Q nazveme NP-**úplným**, pokud je NP-těžký a současně patří do třídy NP.

- Pokud bychom znali polynomiální algoritmus pro kterýkoliv NP-úplný problém P, tak dokážeme polynomiálně řešit všechny.
- Každý NP-úplný problém P' totiž můžeme převést na P a složením dvou polynomiálních algoritmů (převodu a řešení P) dostaneme zase polynomiální algoritmus řešící P'.

NP-úplnost

Věta

Nechť problém Q je NP-těžký. Pokud existuje polynomiální převod problému Q na nějaký problém P, pak také P je NP-těžký.

Důkaz: Protože je Q NP-těžký, je možné na něj převést všechny problémy z třídy NP. Spojení algoritmů převodu na Q a z Q na P dostaneme pro jakýkoliv problém z NP algoritmus převodu na P. Protože tedy jakýkoliv problém z NP umíme převést na P, je P NP-těžký.

- Dokazovat NP-těžkost hledáním převodů všech problémů z NP není možné
- Na základě této věty nám stačí ukázat převod z jednoho problému, o kterém již dříve bylo dokázáno, že je NP-těžký.

NP-úplnost

- NP-těžké problémy jsou vlastně nejtěžší problémy ze třídy NP
- Navržení polynomiálního algoritmu pro jeden NP-úplný problém znamená, že známe polynomiální algoritmus pro každý problém v NP
- Navržení polynomiálního algoritmu pro problém z NP, který není NP-těžký, neznamená nic pro složitost NP-těžkých problémů
- Otázka jestli $\mathrm{P} = \mathrm{NP}$ je jednou z nejznámějších dlouhodobě otevřených otázek teoretické informatiky
- Předpokládá se $\mathrm{P} \neq \mathrm{NP}$, tedy že NP-úplné problémy nepatří mezi efektivně řešitelné
- Otázka ale zní, jestli vůbec nějaký NP-úplný problém existuje. Jak ukázat, že všechny problémy z NP jdou na něj převést, když jich je nekonečně mnoho?

NP-úplnost

SAT - Splnitelnost logických formulí

Vstup: Logická formule Φ v konjunktivní normální formě

Výstup: ANO, pokud existuje ohodnocení μ proměnných takové, aby pravdivostní hodnota formule $\mu(\Phi) = 1$

Příklad: Mějme formuli
$\Phi = (a_1 \vee \neg a_2 \vee a_4) \wedge (\neg a_1 \vee a_3) \wedge (\neg a_1 \vee \neg a_3 \vee \neg a_4)$
Řešení: Pokud ohodnotíme např. $\mu(a_1) = \mu(a_3) = 1$, $\mu(a_2) = \mu(a_4) = 0$ dostaneme $\mu(\Phi) = 1$, tedy odpověď je, že formule je splnitelná.

- a_1, a_2, a_3, a_4 nazýváme **atomické proměnné**.
- Proměnné i jejich negace jsou **literály**.
- Disjunkce, např. $(a_1 \vee \neg a_2 \vee a_4)$, nazýváme **klauzule**.
- Aby byla formule splněna, musí ohodnocení přiřazovat 1 alespoň jednomu literálu v každé klauzuli.

Cookova věta

Věta (Cook)

Existuje NP-úplný problém.

- V důkazu se ukáže, že SAT je NP-úplný
- Příslušnost SATu do NP je zřejmá. Nedeterministický algoritmus hádá ohodnocení proměnných a jen ověří pravdivostní hodnotu formule (popř. využijeme svědka v podobě ohodnocení proměnných a algortimus jen kontroluje, jesli je ohodnocení správné).
- Přesný důkaz NP-obtížnosti není úplně snadný, musí se ošetřit mnoho drobností

Cookova věta

- Každý problém z NP má být převoditelný na SAT
- Pro jeden konkrétní problém, pro který chceme převoditelnost zrovna ukázat, existuje nějaký nedeterministický Turingův stroj, který jej rozhoduje
- Ukážeme konstrukci, která pro tento Turingův stroj $\mathcal{M}$ a vstupní slovo w sestrojí formuli $\phi_{\mathcal{M},w}$
- Formule $\phi_{\mathcal{M},w}$ bude splnitelná právě tehdy, když TS přijme slovo w
- Následující konstrukce funguje pro všechny Turingovy stroje, tedy bychom mohli postupně ukázat převod všech problémů z NP na SAT (což nemůžeme udělat explicitně, protože jich je nekonečně mnoho)

Cookova věta

Výpočet Turingova stroje $\mathcal{M}$ na vstupním slově w ($|w| = n$) můžeme reprezentovat tabulkou

	0	1	2				n^k
0	#	(q_0, w_1)	w_2	$\cdots$	w_n	# $\cdots$ #	#
i	#	α_1	α_2	$\cdots$	α_n	$\alpha_{n+1}\cdots$	#
n^k	#	β_1	β_2	$\cdots$	β_n	$\beta_{n+1}\cdots$	#

Cookova věta

- Zavedeme proměnné $x_{i,j,s}$ pro každé $0 \leq i \leq n^k$, $0 \leq j \leq n^k$, $s \in \Gamma$. Hodnota 1 takové proměnné znamená, že v i-tém řádku a j-tém sloupci se nachází symbol s, tedy stroj má v i-té konfiguraci na j-té pozici pásky symbol s.
- Zavedeme proměnné $x_{i,j,(q,s)}$ pro každé $0 \leq j \leq n^k$, $0 \leq j \leq n^k$, $s \in \Gamma$ a $q \in Q$. Hodnota 1 takové proměnné znamená, že v i-tém řádku a j-tém sloupci se nachází symbol (q, s), tedy stroj má v i-té konfiguraci na j-té pozici pásky symbol s, na tento symbol ukazuje hlava a stroj se nachází ve stavu q.
- Formuli $\phi_{\mathcal{M},w}$ vytvoříme jako

$$\phi_{\mathcal{M},w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$$

Cookova věta

ϕ_{cell} je formule, která musí zajistit, že v každé buňce tabulky je právě jeden symbol.

$$\phi_{cell} = \bigwedge_{0 \leq i,j \leq n^k} \left[\left(\bigvee_{s \in \Gamma \cup (Q \times \Gamma)} x_{i,j,s} \right) \wedge \left(\bigwedge_{r \neq t \in \Gamma \cup (Q \times \Gamma)} (\neg x_{i,j,r} \vee \neg x_{i,j,t}) \right) \right]$$

$\bigvee_{s \in \Gamma \cup (Q \times \Gamma)} x_{i,j,s}$ zajistí, že každá buňka obsahuje alespoň jeden symbol.
$\bigwedge_{r \neq t \in \Gamma \cup (Q \times \Gamma)} (\neg x_{i,j,r} \vee \neg x_{i,j,t})$ zajistí, že žádná buňka neobsahuje více různých symbolů.

Cookova věta

ϕ_{start} musí zajistit, že první řádek odpovídá počáteční konfiguraci. Tedy na pásce je slovo w, hlava ukazuje na první symbol tohoto slova a stav je q_0.

$$\phi_{start} = x_{0,0,\#} \wedge x_{0,1,(q_0,w_1)} \wedge x_{0,2,w_2} \wedge \ldots \wedge x_{0,n,w_n} \wedge x_{0,n+1,\#} \wedge \ldots \wedge x_{0,n^k,\#}$$

Cookova věta

ϕ_{accept} musí zajistit, že na posledním řádku tabulky se vyskytuje stav q_{ANO}. Předpokládáme, že pokud Turingův stroj skončí výpočet dřív než po n^k krocích, všechny následující konfigurace (řádky tabulky) jsou stejné.

$$\phi_{accept} = \bigvee_{1 \leq j \leq n^k, w \in \Gamma} x_{n^k, j, (q_{\text{ANO}}, w)}$$

Cookova věta

ϕ_{move} musí zajistit, že každá následující konfigurace vznikne z předchozí spravným způsobem.

- Musíme prozkoumat všechna "okna" o dvou řádcích a třech sloupcích.
- Pokud horní řádek neobsahuje stav, muže v dolním přibýt na kraji stav, ale symboly musí zůstat stejné.
- Pokud horní řádek obsahuje stav, může na spodním být stav posunutý a symbol na pozici, kde stav byl, může být jiný.
- Konkrétní správná okna záleží jen na přechodové funkci stroje $\mathcal{M}$. Vůbec nezáleží na délce vstupu w, takže počet správných oken je konstantní vzhledem k velikosti vstupu.
- Označme SO množinu správných oken, tedy šestic symbolů z $\Gamma \cup (Q \times \Gamma)$ takových, že tvoří správné okno

$$\phi_{move} = \bigwedge_{1 \leq i,j \leq n^k} \bigvee_{(a_1,\ldots,a_6) \in SO} (x_{i-1,j-1,a_1} \wedge \ldots x_{i,j+1,a_6})$$

Cookova věta

- Formule $\phi_{\mathcal{M},w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$ je polynomiální vzhledem k n.
- Jestliže je splnitelná tato formule, dokážeme vyplnit tabulku tak, že reprezentuje přijímající výpočet Turingova stroje $\mathcal{M}$ na slově w.
- Jestliže je formule nesplnitelná, tak neexistuje přijímající výpočet Turingova stroje $\mathcal{M}$ na slově w, protože se nám nepodaří vyplnit tabulku reprezentující takový výpočet.
- Ukázali jsme tedy polynomiální převod libovolného problému z NP na problém SAT

NP-úplnost

Důkazy NP-úplnosti problémů můžeme rozdělit na:

1. Přímé - podobně jako pro SAT se ukáže přímo převoditelnost všech možných problémů z NP na zadaný problém. Nejčastěji se využívá posloupnosti konfigurací Turingova stroje - odpověď na problém pro který dokazujeme NP-úplnost bude ANO právě tehdy, když existuje přijímající výpočet daného Turingova stroje na daném slově.
2. Nepřímé - převodem z nějakého již známého NP-úplného problému. Vzniká vlastně sekvence převodů začínající nějakým s přímo dokázanou NP-úplností a končící problémem, pro který NP-úplnost právě dokazujeme.

Všechny důkazy, které budeme dále dělat, budou vedeny způsobem 2. Ten je obecně běžnější a častěji používaný.