KMI/VCS1 – Vyčíslitelnost a složitost

Paměťová složitost, Savitchova věta, třída PSPACE, PSPACE-úplné problémy, a jako bonus: Bremermannova mez

Jan Konečný

3. prosince 2013

Paměťová složitost

Zabýváme náročností výpočetních problémů z hlediska paměti, která je potřeba k jejich řešení.
Připomínka:

Definice

Paměťová složitost TS T je funkce $f : N \to N$, kde $f(n)$ je maximální počet políček použitých při výpočtu nad jakýmkoli vstupem délky n.

a pro NTS:

Definice

Paměťová složitost NTS T je funkce $f : N \to N$, kde $f(n)$ je maximální počet políček použitých při výpočtu nad jakýmkoli vstupem délky n v jakékoli větvi výpočtu.

Paměťová složitost

Definice

Jazyk A nazveme *rozhodovaný v paměti* $f(n)$ pokud existuje TS, který má paměťovou složitost $f(n)$.

Analogicky pro NTS:

Definice

Jazyk A nazveme *nedeterministicky rozhodovaný v paměti* $f(n)$ pokud existuje NTS, který má paměťovou složitost $f(n)$.

Třídy paměťové složitosti:

Definice

$\mathrm{SPACE}(f(n)) = \{A \mid \text{Jazyk } A \text{ je rozhodovaný v paměti } \mathcal{O}(f(n))\}$.
$\mathrm{NSPACE}(f(n)) = \{A \mid \text{Jazyk } A \text{ je nedet. rozhodovaný paměti } \mathcal{O}(f(n))\}$.

Příklad

$\mathrm{SAT} \in SPACE(n)$;

TS M_1 pro $[\phi]$, kde ϕ je Booleovská formule:

1. *Pro každé ohodnocení proměnných $x_1, \ldots, x_m$ z ϕ:*
   Vyhodnoť ϕ v tomto ohodnocení.
2. *Pokud ϕ je vyhodnoceno (aspoň v jednom případě) na 1,* přijmi, *jinak* zamítni.

M_1 pracuje v lineární paměti, protože každá iterace toho cyklu může znovuvyužít tu samou část pásky. Stroj si potřebuje pamatovat pouze atuální ohodnocení, což může být provedeno v paměti $\mathcal{O}(m)$. Počet proměnných m je nejvýše n (délka vstupu), takže stroj pracuje v paměti $\mathcal{O}(n)$.

Savitchova věta

Říká, že DTS může simulovat NTS s použitím celkem malého počtu paměti. Přesněji, NTS, který používá $f(n)$ paměti může být zkonvertován na DTS, který používá $f^2(n)$ paměti.

Věta (Savitchova věta)

Nechť $f : \mathbb{N} \to \mathbb{R}$, *t.ž.* $n \leq f(n)$. *Pak platí*

$$\mathrm{SPACE}(f^2(n)) \supseteq \mathrm{NSPACE}(f(n))$$

Idea důkazu (část I)

První idea je asi použít naše dosavadní znalosti o simulaci NTS a DTS (věta o ekvivalenci těchto tříd strojů). Tam jsme ale generovali celý strom výpočtů NTS. Ovšem větev využívající $f(n)$ paměti může běžet až po $2^{f(n)}$ kroků. V každém z nich může udělat nedeterministický krok (větvení). K prohledávání všech historií sekvenčně by bylo potřeba si tyto kroky pamatovat, tedy použít. $\mathcal{O}(2^{f(n)})$ paměti. Tudy tedy ne.

Idea důkazu (část II)

Namísto toho vytvoříme TS, který bude řešit obecnější problém:

Problém odvoditelnosti v NTS M
Instance: $\mathcal{C}_1, \mathcal{C}_2$ – konfigurace M, $t \in \mathbb{N}$
Otázka: Může se M dostat během t kroků z $\mathcal{C}_1$ do $\mathcal{C}_2$?

Pokud budeme umět řešit odvoditelnost v polynomické paměti, můžeme ho pak řešit pro $\mathcal{C}_0^w$,$\mathcal{C}_+$, t, kde t je maximum kroků, které může NTS použít.

Vytvoříme rekurzivní algoritmus, který pracuje tak, že hledá mezilehlou konfiguraci $\mathcal{C}_m$, t.ž. M se může dostat

- z $\mathcal{C}_1$ do $\mathcal{C}_m$ během $t/2$ kroků,
- z $\mathcal{C}_m$ do $\mathcal{C}_2$ během $t/2$ kroků.

Algoritmus bude potřebovat paměť pro zásobník rekurze, každá úroveň rekurze zabere $\mathcal{O}(f(n))$ prostoru pro uložení konfigurace.
Hloubka rekurze je $\log t$, kde $t = 2^{\mathcal{O}(f(n))}$, a tedy $\log t = \mathcal{O}(f(n))$.
Celkově tedy $\mathcal{O}(f^2(n))$.

Důkaz (část I)

Nechť N je NTS rozhodující A v paměti $f(n)$.
Zkonstruujeme deterministický TS M, který rozhoduje A.
TS M bude používat proceduru (TS) CanYield (na následujícím slajdu).

- ta testuje jestli jedna z konfiguraci NTS N může odvodit druhou v daném počtu kroků.
- pro konfigurace c_1 a c_2 NTS N a číslo $t \in \mathbb{N}$, CanYield(c_1, c_2, t) přijme, pokud c_2 lze odvodit z c_1 v t krocích, jinak zamítne. (popsáno v Idei důkazu)

Důkaz (část II)

TS CanYield *pro vstupní slovo* $[c_1, c_2, t]$*:*

1. *Pokud* $t = 1$*, testuje jestli* $c_1 = c_2$ *nebo* $c_1 \vdash_N c_2$ *pokud ano, přijme, jinak zamítne.*
2. *Pokud* $t > 1$ *tak pro každou konfiguraci* c_m*, která využívá prostor* $f(n)$ *: Spustí* CanYield *pro* $[c_1, c_m, \lceil \frac{t}{2} \rceil]$ *a spustí* CanYield *pro* $[c_m, c_2, \lceil \frac{t}{2} \rceil]$*, pokud obě spuštění skončí přijetím, přijme.*
3. *Pokud neexistuje taková konfigurace* c_m*, pro které by došlo k přijetí v předchozím bodu, zamítne.*

(Značení $\lceil \cdot \rceil$ *představuje zaokrouhlení nahoru).*

Důkaz (část III)

Nyní definujeme TS M (který simuluje N) takto:

- Modifikujeme N tak aby po sobě uklízel.
- Zvolíme konstantu d, tak aby N neměl více než $2^{df(n)}$ konfigurací používajících pamět $f(n)$, kde n je délka w.
- Víme že $2^{df(n)}$ představuje horní hranici pro čas použitý v každé větvi výpočtu N nad w.

TS M *pro* w *spustí* CanYield *pro* $[\mathcal{C}_0^w, \mathcal{C}_+, 2^{df(n)}]$.

Paměťová složitost

- Kdykoli CanYield vyvolá rekurzivně sám sebe, uloží na zásobník trojici $\langle c_1, c_2, t\rangle$, každá úroveň rekurze použije $\mathcal{O}(f(n))$ dodatečného prostoru.
- Každá úroveň rekurze zkracuje t na polovinu, začíná se na $2^{df(n)}$. Hloubka rekurze je tedy $\mathcal{O}(\log(2^{df(n)})) = \mathcal{O}(f(n))$.
- Celkově tedy $\mathcal{O}(f^2(n))$.

Důkaz (část IV)

Ještě jedna věc zbývá dořešit: M potřebuje znát hodnotu $f(n)$, když vyvolává CanYield.

Můžeme to dořešit tak, že M bude zkoušet $f(n) = 1, 2, 3, \ldots, i, \ldots$

- jestli přijímající konfigurace $\mathcal{C}_+$ je dosažitelná.
- jestli N použije paměť alespoň i, testováním, jestli nějaká konfigurace délky i je dosažitelná z $\mathcal{C}_0^w$.

Takže pro $f(n) = i$

- Pokud je $\mathcal{C}_+$ dosažitelná, přijme.
- Pokud není, a není ani dosažitelná žádná konfigurace délky i, zamítne.
- jinak zkusí $f(n) = i + 1$.

Třída PSPACE

Definice

PSPACE je třída jazyku, které jsou rozhodnutelné v polynomickém čase na (deterministickém) TS; tedy

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k).$$

Otázka na studenty

Třídou NPSPACE se nebudeme zabývat. Proč asi?

Třída PSPACE

Definice

PSPACE je třída jazyku, které jsou rozhodnutelné v polynomickém čase na (deterministickém) TS; tedy

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k).$$

Otázka na studenty

Třídou NPSPACE se nebudeme zabývat. Proč asi?

Protože přímo ze Savitchovy věty máme:

Důsledek

$$\mathrm{PSPACE} = \mathrm{NPSPACE}.$$

Důsledek

$$\mathrm{SAT} \in \mathrm{PSPACE}$$

Poznámka

Celkově tedy máme:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME},$$

kde $\mathrm{EXPTIME} = \bigcup_k \mathrm{TIME}(2^n)$.

Alespoň jedna z těchto inkluzí je vlastní: ví se, že $\mathrm{P} \neq \mathrm{EXPTIME}$.

Věří se, že všechny ty inkluze jsou vlastní.

PSPACE-úplné problémy

Definice

Jazyk B je PSPACE-*úplný problém*, pokud

- $B \in \mathrm{PSPACE}$,
- B je PSPACE-*těžký*, tj. každý $A \in \mathrm{PSPACE}$ je redukovatelný v polynomickém čase na B.

Otázka na studenty

Proč redukovatelné v polynomickém čase? Proč ne v polynomické paměti?

PSPACE-úplné problémy

Definice

Jazyk B je PSPACE-*úplný problém*, pokud

- $B \in \mathrm{PSPACE}$,
- B je PSPACE-*těžký*, tj. každý $A \in \mathrm{PSPACE}$ je redukovatelný v polynomickém čase na B.

Otázka na studenty

Proč redukovatelné v polynomickém čase? Proč ne v polynomické paměti?

Odpověď

Páč pak by ten pojem byl triviální.

Problém pravdivosti plně kvantifikovaných proměnných (TQBF; true quantified Boolean formula).

pravdivá plně kvantifikovaná Booleovska formule:

$$\phi = \forall x \exists y [(x \vee y) \wedge (\overline{x} \vee \overline{y})]$$

nepravdivá plně kvantifikovaná Booleovska formule:

$$\phi = \exists y \forall x [(x \vee y) \wedge (\overline{x} \vee \overline{y})]$$

TQBF
Instance: ϕ – plně kvantifikovaná Booleovká formule
Otázka: Je ϕ pravdivá?

PSPACE-úplnost TQBF

Věta

TQBF *je* PSPACE*-úplný problém.*

Idea důkazu

Abychom ukázali, že $\mathrm{TQBF} \in \mathrm{PSPACE}$ uvedeme TS, který problém rozhoduje v polynomické paměti.

Abychom ukázali, že TQBF je PSPACE-úplný problém, ukážeme redukci libovolného PSPACE problému na TQBF. Tato část bude postavena podobném principu jako u důkazu NP-úplnosti SAT a u důkazu Savitchovy věty.

Důkaz (část I.)

$\mathrm{TQBF} \in \mathrm{PSPACE}$:

TS T pro $[\phi]$

1. *Pokud ϕ neobsahuje kvantifikátory, pak je to po výraz obsahující pouze konstanty, vyhodnoť ϕ, přijmi pokud je pravdivá, jinak zamítni.*
2. *Pokud je ϕ ve tvaru $(\exists x)\psi$, rekurzivně vyvolej T na ψ, nejdříve s 0 dosazenou za x, pak s 1 dosazenou za x. Pokud alespoň jedno skončilo přijetím, přijmi, jinak zamítni.*
3. *Pokud je ϕ ve tvaru $(\forall x)\psi$, rekurzivně vyvolej T na ψ, nejdříve s 0 dosazenou za x, pak s 1 dosazenou za x. Pokud obě skončila přijetím, přijmi, jinak zamítni.*

Tento algoritmus zjevně rozhoduje TQBF. Pamětová složitost odpovídá hloubce rekurze, ta odpovídá $\mathcal{O}(m)$, kde m je počet kvantifikovaných proměnných ve ϕ; $m \leq n$. A tedy T pracuje v lineární paměti.

Důkaz (část II.)

TQBF je PSPACE-těžký:
Uvažujeme, že jazyk A rozhodovaný TS M v paměti n^k pro nějako konstantu k. Provedeme redukci v polynomiálním čase na TQBF.
w budeme zobrazovat na $r(w) = \phi$, t.ž. M přijímá w p.k. ϕ je pravdivá.

Abychom ukázali konstrukci ϕ, budeme řešit obecnější problém:

Použitím dvou kolekcí c_1, c_2 proměnných reprezentující konfigurace $\mathcal{C}_1$ a $\mathcal{C}_2$, a číslo $t > 0$ zkonstruujeme formuli $\phi_{c_1,c_2,t}$.

Pokud přiřadíme c_1, c_2 konfigurace, formula bude pravdivá p.k. $\mathcal{C}_1 \vdash_M \mathcal{C}_2$ v nejvýše t krocích.
Pak naše hledaná formule bude $\phi_{c_0^w,c_+,h}$, kde $h = 2^{df(n)}$, pro konstantu d vybranou tak, aby M nemělo více jak $2^{df(n)}$ možných konfigurací nad vstupem délky n.

(podobná finta jako v důkazu Savichovy věty.)

Důkaz (část III.)

Pro $t = 1$, můžeme snadno vytvořit $\phi_{c_1,c_2,t}$ podobně, jako se to dělalo v důkazu Cook-Levinovy věty.

Pokud $t > 1$ zkonstruujeme $\phi_{c_1,c_2,t}$ rekurzivně.

$$\phi_{c_1,c_2,t} = \exists m_1[\phi_{c_1,m_1,\lceil \frac{t}{2} \rceil} \wedge \phi_{m_1,c_2,\frac{t}{2}\rceil}].$$

Toto je teprve „zahřívačka“, tohle nebude fungovat. . .

m_+ reprezentuje kolekci proměnných, která kóduje konfiguraci.

Význam formule je tedy: „Existuje mezilehlá konfigurace m_1 t.ž. ji lze odvodit z c_1 v nejvýše $t/2$ krocích a z m_1 lze odvodit c_2 v nejvýše $t/2$ krocích.“

Formule $\phi_{c_1,m_1,\lceil \frac{t}{2} \rceil}$ a $\phi_{m_1,c_2,\lceil \frac{t}{2} \rceil}$ zkonstruujeme rekurzivně.

Důkaz (část IV.)

$$\phi_{c_1,c_2,t} = \exists m_1 [\phi_{c_1,m_1,\lceil \frac{t}{2} \rceil} \wedge \phi_{m_1,c_2,\lceil \frac{t}{2} \rceil}].$$

Takto vytvořená formule bude určite korektní, ale bude příliš velká:

- každá úroveň rekurze rozdělí t na poloviny,
- ale zhruba zdvojnásobí velikost formule, takze dostaneme formuli délky zhruba t, tedy $2^{df(n)}$.

Můžeme ji ale zkrátit takto

$$\phi_{c_1,c_2,t} = \exists m_1 \forall (c_1, c_2) \in \{(c_1, m_1), (m_1, c_2)\} [\phi_{c_3,c_4,\lceil \frac{t}{2} \rceil}].$$

Stanovení velikosti formule $\phi_{c_1,c_2,t}$, kde $t = 2^{df(n)}$:

- každý krok rekurze jen přidá část, která je lineárně závislá na velikosti konfigurací, tedy $\mathcal{O}(f(n))$,
- počet kroků rekurze je $\log(2^{df(n)})$, tedy $\mathcal{O}(f(n))$,
- celkove tedy $\mathcal{O}(f^2(n))$.

Hra dle logické formule

Nechť $\phi = \exists x_1 \forall x_2 \exists x_3 \ldots Q x_k[\psi]$ je kvantifikovaná formule v prenexní normální formě. Q reprezentuje buďto $\forall$ nebo $\exists$.

Hru hrají 2 hráči E a A následovně:

- hráči se střídají, a volí hodnoty proměnných $x_1, x_2, \ldots, x_k$,
- A volí hodnoty pro proměnné vázané ke kvantifikátorům $\forall$,
- E volí hodnoty pro proměnné vázané ke kvantifikátorům $\exists$.
- střídání hráčů je prováděno podle pořadí, jak jsou kvantifikátory ve ϕ.
- hra končí jakmile jsou zvoleny hodnoty pro všechny proměnné: pokud má ψ pod takto vytvořeném ohodnocení hodnotu 1, jinak vyhrává A.

Vítězná strategie

Příklad

$$\phi = \exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})].$$

Hráč E vždy vyhraje, pokud zvolí $x_1 = 1$ a x_3 je negace toho, co zvolil A za x_2.

V takovém případě říkáme, že E má vítěznou strategii.
Obecně, hráč má vítěznou strategii, pokud tento hráč vyhraje, pokud obě strany hrají optimálně.

Příklad

$$\phi = \exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3})]$$

Zde má zase víteznou strategii A, protože mu stačí zvolit $x_2 = 0$ (bez ohledu na to, co volí E).

$$\text{FORMULAGAME} = \{[\phi] \mid \text{Hráč } E \text{ má vítěznou strategii ve hře podle } \phi\}$$

Věta

FORMULAGAME *je* PSPACE*-úplný.*

Důkaz.

Stačí si všimnout, že $\text{FORMULAGAME} = \text{TQBF}$. □

Grafová hra

Mějme orientovaný graf G s vyznačeným počátečním uzlem u_1.

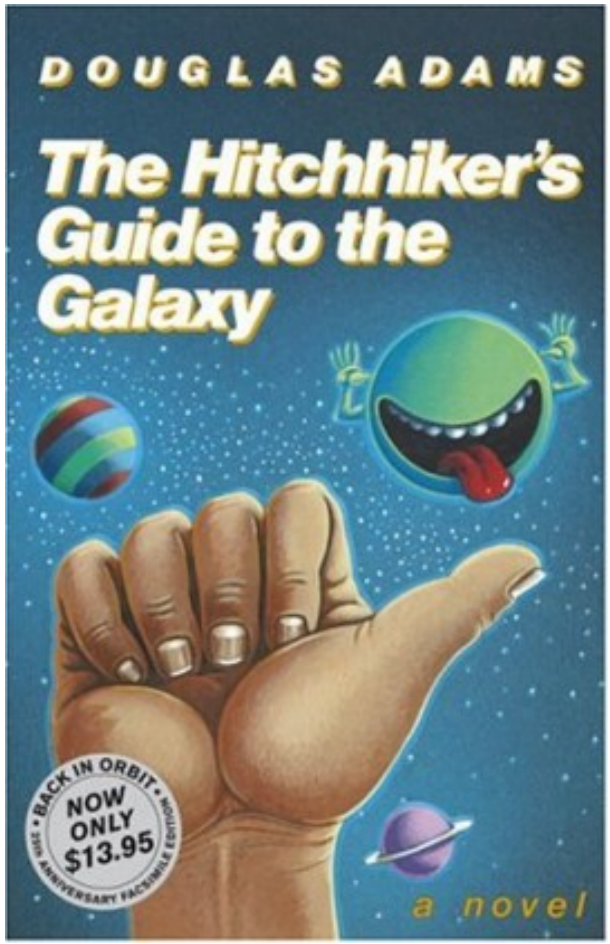
- Aktuální uzel je na začátku hry u_1.
- Hráči se střádají v tazích.
- Hráč na tahu zvolí souseda u_1, ten se stává aktuálním uzlem, nelze vybrat uzel, který už byl předtím navštíven.
- Hráč, který uvízne prohrává.

$$\mathrm{GG} = \{[G, b] \mid \text{hráč 1 má vítěznou strategii pro } G \text{ a poč. b}\}$$

Věta

GG *je* PSPACE*-úplný.*

Bremermannova mez (Bremermann's limit)



Bremermannova mez (Bremermann's limit)

No data processing system, whether artificial or living, can process more than 2×10^{47} *bits per second per gram of its mass.*

Bremermann, H.J. (1962) Optimization through evolution and recombination In: Self-Organizing Systems 1962, edited M.C. Yovitts et al., Spartan Books, Washington, D.C. pp. 93–106.

Proč?

Je jasné, že informace, která být zpracovávána strojem, musí být nějakým způsobem fyzicky zakódována.

Předpokládáme, že je zakódována pomocí úrovní energie v intervalu $[0, E]$ (nějaké) energie.

Na E se můžeme dívat jako celkovou energii dostupnou pro tento účel.

Dále předpokládáme, že tyto úrovně energie mohou být měřeny s přesností $\triangle E$.

Pak, nejjemnější kódování je definováno značkami, kterými je celý interval rozdělen do $N = E/\triangle E$ podintervalů, každý asociovaný s množstvím energie $\triangle E$.

Extrémní případ je vyjádřen **Heisenbergovým principem nejistoty**: energie může být měřena s přesností $\triangle E$, pokud platí nerovnost

$$\triangle E \triangle t \geq h,$$

kde

- $\triangle t$ je čas trvání měření,
- h je Planckova konstanta $10^{-34}\mathrm{J} \cdot \mathrm{s} = 10^{-27}\mathrm{erg} \cdot \mathrm{s}$,
- $\triangle E$ je definováno jako průměrná odchylka od očekáváné energie.

To znamená, že

$$N \leq \frac{E \triangle t}{h}.$$

Dostupná energie E může být vyjádřena equivalentním množstvím hmoty m dle **Einsteinovy formule**:

$$E = mc^2,$$

kde $c = 3 \cdot 10^8 m \cdot s^{-1}$ je rychlost světla ve vakuu.

Pokud vezmeme horní (nejoptimističtější) hranici N v té nerovnosti na přechozím slajdu, dostáváme:

$$N = \frac{mc^2 \triangle t}{h}$$

$$N = \frac{mc^2 \triangle t}{h}$$

Po dosazení číselných hodnot dostáváme

$$N = 1.36 m \triangle t \cdot 10^{47}.$$

Pro hmotnost 1g ($m = 1$) a čas 1s ($\triangle t = 1$), dostáváme

$$N = 1.36 \cdot 10^{47}.$$

Odtud pak tvrzení.

Země jako počítač

Předpokládáné stáří Země: 10^{10} let , každý rok má asi $3.14 \cdot 10^7$ sekund.

Hmotnost Země: 6×10^{27}g

Tedy Země byla za dobu své existence schopná zpracovat maximálně $2.56 \cdot 10^{92}$ bitů informace.

Zaokrouhleno nahoru na mocninu 10, dostáváme číslo 10^{93}.
Toto číslo je známo jako **Bremermannova mez**.