P-úplnost: těžko paralelizovatelné problémy

Zdeněk Sawa

20. srpna 2000

1 Úvod

Paralelní algoritmus je algoritmus, ve kterém pracuje na řešení určitého problému nikoli jeden, ale více navzájem spolupracujících procesorů. Intuitivně je možné očekávat, že čím je počet těchto procesorů větší, tím je doba potřebná k vyřešení daného problému kratší.

Při návrhu paralelního algoritmu je vhodné postupovat tak, jako kdyby počet procesorů, které jsou k dispozici, byl potenciálně neomezený, aby bylo co největší množství operací prováděno současně (concurently). Při analýze algoritmů pak analyzujeme vedle množství potřebného času a potřebné paměti také počet potřebných procesorů a vzájemné vztahy mezi těmito hodnotami. Počet procesorů je pak vyjádřen jako funkce velikosti instance problému, což znamená, že analyzujeme, jak počet procesorů roste v závislosti na velikostí instance. Podobně jako při analýze časové nebo paměťové složitosti algoritmu často kvůli zjednodušení abstrahujeme od detailů a růst počtu procesorů vyjadřujeme pouze asymtoticky.

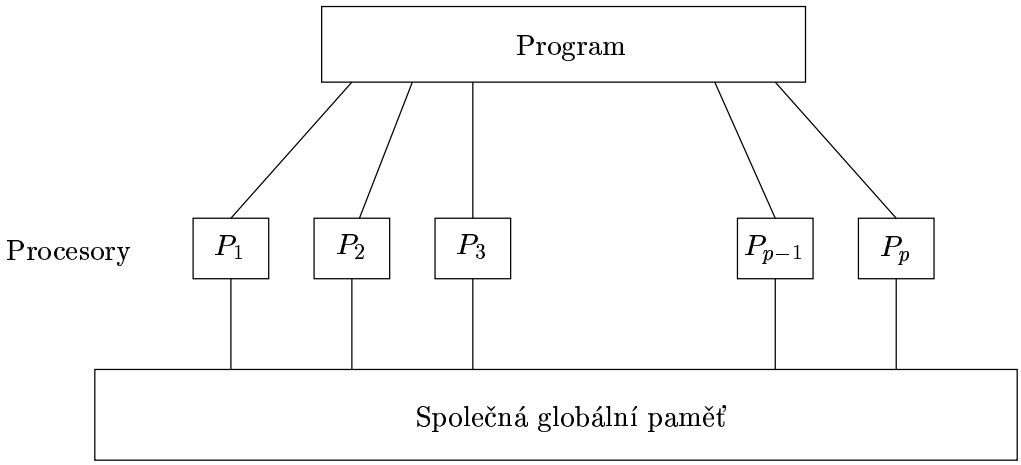
Z praktického hlediska jsou považovány za rozumné pouze paralelní algoritmy, u kterých je počet použitých procesorů polynomiální, což znamená, že existuje konstanta k taková, že počet procesorů je v $O(n^k)$, kde n je velikost instance problému.

Časová složitost algoritmu závisí na použitém modelu výpočtu. V případě sekvenčních algoritmů existuje poměrně dobrá shoda ohledně tohoto modelu. Bývá používán tzv. RAM (random-access machine), který se skládá z globální paměti tvořené jednotlivými paměťovými buňkami a procesoru, který provádí instrukce podle zadaného programu, přičemž tento procesor je schopen provádět jednoduché aritmetické a logické operace (sčítání, odčítání, násobení, dělení, porovnávání čísel apod.). Procesor může přistupovat k libovolné buňce paměti a doba čtení nebo zápisu z nebo do paměťové buňky je konstantní.

V případě paralelních algoritmů není situace takto jednoduchá, neboť existuje celá řada různých modelů, které jsou různě vhodné pro různé účely. Jedním z modelů často používaných při návrhu algoritmů je tzv. P-RAM (parallel random-access machine). Ten je opět tvořen globální společnou pamětí a určitým počtem procesorů, které pracují nad touto společnou pamětí. Tyto procesory jsou očíslovány přirozenými čísly a všechny vykonávají tentýž program (viz obrázek 1). Ačkoliv provádějí všechny procesory tytéž instrukce, mohou pracovat nad různými daty (uloženými na různých místech v paměti).

Existuje několik variant P-RAMu, které se liší například v tom, zda je možné, aby několik procesorů najedou četlo nebo zapisovalo z nebo do jedné paměťové buňky, a jakým způsobem jsou takovéto situace řešeny. Tyto jednotlivé varianty jsou ekvivalentní v tom smyslu, že je možné simulovat jednu pomocí druhé, ovšem časová složitost algoritmu může být pro různé varianty různá.

P-RAM je vhodný model pro návrh algoritmů na obecné abstraktní úrovni, neboť je to poměrně jednoduchý a dosti obecný model, ovšem při realizaci těchto algoritmů na konkrétní



Obrázek 1: P-RAM

architektuře již nemusí plně odrážet realitu. Jedná se zejména o to, že v případě použití P-RAMu je čas spotřebovaný na vzájemnou komunikaci mezi procesory nulový, neboť tyto procesory spolu komunikují výhradně pomocí společné globální paměti a doba přístupu ke kterékoliv buňce paměti je konstantní a stejná pro všechny procesory. Takový předpoklad ovšem obecně neplatí, neboť paměť může být rozdělena mezi jednotlivé procesory, přičemž doba přístupu k paměti jiného procesoru může být o několik řádů větší než doba přístupu k lokální paměti a navíc nemusí být tato doba konstantní, ale může záviset na vzájemné poloze a způsobu propojení procesorů.

Definice 1.1 Třída NC (Nics's Class) je definována jako množina všech problémů, pro které existuje paralelní algoritmus, který má při použití polynomiálního počtu procesorů polylogaritmickou časovou složitost, což znamená, že existuje konstanta k taková, že časová složitost je v $O(\log^k n)$, kde n je velikost problému.

NC je v podstatě třída dobře paralelizovatelných problémů, to jest problémů, pro něž existují efektivní paralelní algoritmy.

Navíc je tato třída robustní, neboť to, zda do ní daný problém patří nebo ne, nezávisí na použitém modelu výpočtu. Výpočet, který v jednom modelu vyžaduje polylogaritmický čas a polynomiální počet procesorů, totiž může být v jiných modelech simulován tak, že výpočet opět vyžaduje polylogaritmický čas a polynomiální počet procesorů, i když konkrétní časová složitost a počet procesorů se může změnit.

Definice 1.2 Třída P je definována jako třída problémů, pro které existuje sekvenční algoritmus s polynomiální časovou složitostí, což znamená, že existuje nějaká konstanta k taková, že časová složitost algoritmu patří do $O(n^k)$, kde n je velikost problému.

Třída P je v podstatě třída prakticky zvládnutelných problémů.

Nemůžeme očekávat, že by existovaly paralelní algoritmy s polynomiální časovou složitostí při použití polynomiálního počtu procesorů pro problémy, které nepatří do třídy P (a které tedy mají při použití sekvenčního stroje RAM alespoň exponenciální časovou složitost). Jinými slovy řečeno, nemůžeme očekávat, že problém, který není zvládnutelný při použití jednoho procesoru, by mohl být zvládnutelný při použití polynomiálního počtu procesorů.

Je totiž celkem zřejmé, že výpočet paralelního stroje P-RAM můžeme simulovat pomocí sekvenčního stroje RAM (RAM může např. provést jednu instrukci prvního procesoru, jednu instrukci druhého procesoru atd. až jednu instrukci p-tého procesoru, poté druhou instrukci prvního procesoru, druhou instrukci druhého procesoru atd.). Časová složitost výsledného sekvenčního algoritmu je pak úměrná celkovému počtu všech instrukcí provedených paralelním strojem P-RAM, přičemž tento počet instrukcí je maximálně roven součinu počtu procesorů a maximálního počtu instrukcí provedených jedním procesorem, přičemž tento počet může být maximálně roven časové složitosti paralelního algoritmu. Pokud by tedy existoval paralelní algoritmus s polynomiální časovou složitostí při použití polynomiálního počtu procesorů, snadno bychom mohli najít sekvenční algoritmus s polynomiální časovou složitostí (neboť součin dvou polynomů je zase polynom) a problém by tedy patřil do třídy P.

Zůstává otázkou, zda pro každý problém, který patří do třídy P platí, že pro něj existuje paralelní algoritmus, který je při použití polynomiálního počtu procesorů podstatně rychlejší než jakýkoliv sekvenční algoritmus. Tuto otázku je možné formulovat přesněji tak, že se ptáme, jaký je vztah mezi třídami NC a P. Zjevně platí, že $NC \subseteq P$, neboť každý paralelní algoritmus s polylogaritmickou časovou složitostí při polynomiálním počtu procesorů můžeme simulovat sekvenčním algoritmem s polynomiální časovou složitostí.

Zda však platí $P \subseteq NC$ (a tedy, zda platí $P = NC$) zůstává otevřeným problémem. Panuje obecné přesvědčení, že $P \neq NC$ (a tedy $NC \subset P$, že tedy existují problémy, které patří do $P - NC$), zatím se to však nikomu nepodařilo dokázat.

Pokud existují nějaké problémy patřící do $P - NC$, pak do této množiny budou určitě patřit tzv. ***P-úplné*** problémy.

Definice 1.3 Problém L nazýváme P-úplným problémem, jestliže tento problém patří do třídy P a jestliže každý jiný problém patřící do P může být transformován na problém L v polylogaritmickém čase za požití polynomiálního počtu procesorů (takováto transformace se nazývá NC-redukce).

Pokud by se pro nějaký P-úplný problém L podařilo dokázat, že $L \in NC$, pak by platilo, že $P = NC$, neboť každý problém v P by bylo možné převést pomocí NC-redukce na L a poté vyřešit v polylogaritmickém čase s polynomiálním počtem procesorů.

Kdyby se naopak pro nějaký problém z P podařilo dokázat, že tento problém nepatří do NC, pak by do NC nemohl patřit žádný z P-úplných problémů, neboť v opačném případě by bylo možné převést tento problém na P-úplný problém patřící do NC a problém, o kterém by bylo dokázáno, že nepatří do NC, by patřil do NC, což by byl spor.

P-úplné problémy tedy tvoří třídu nejhůře paralelizovatelných problémů mezi zvládnutelnými problémy. Pro žádný z problémů, pro které se podařilo dokázat, že patří mezi P-úplné problémy, se nepodařilo najít efektivní paralelní algoritmus (algoritmus patřící do NC). Na druhou stranu se pro žádný z P-úplných problémů nepodařilo dokázat, že takový algoritmus neexistuje. Z praktického hlediska je však důkaz P-úplnosti nějakého problému dostatečným důvodem k přesvědčení, že pro tento problém zřejmě neexistuje efektivní paralelní algoritmus, který by měl podstatně menší časovou složitost než algoritmus sekvenční.

V následujícím textu bude uvedeno několik příkladů P-úplných problémů včetně důkazů jejich P-úplnosti. Je poměrně obtížné najít nějaký problém a dokázat jeho P-úplnost podle definice. Pokud se nám však podaří dokázat pro nějaký problém L, že se jedná o P-úplný problém, pak jako důkaz P-úplnosti nějakého dalšího problému L' stačí ukázat, že $L' \in P$ a že existuje NC-redukce problému L na problém L'.

2 První P-úplný problém — Generovatelnost

První problém, jehož P-úplnost ukážeme, je problém nazvaný Generovatelnost (Generability):

Generovatelnost

Instance: Konečná množina X a binární operace $\circ$ na této množině. Dále musí být specifikována podmnožina $T \subset X$ (tzv. počáteční prvky) a tzv. cílový prvek $s \in X$.

Otázka: Patří s do uzávěru (vzhledem k $\circ$) T ?

Pokud máme danou množinu S a binární operaci $\circ$, je množina $S \circ S$ definována takto: $S \circ S = \{a \circ b \mid a, b \in S\}$. S využitím této definice je možné uzávěr (closure) množiny T vypočítat následujícím (sekvenčním) algoritmem:

$T' \leftarrow T$
while $T' \neq T' \cup T' \circ T'$ **do**
  $T' \leftarrow T' \cup T' \circ T'$
end while
$closure \leftarrow T'$

Tělo cyklu **while** je provedeno nejvýše $|X - T|$ krát, neboť během každé iterace je do T' přidán nejméně jeden prvek a platí, že uzávěr nemůže obsahovat více prvků než množina X ($closure(T) \subseteq X$ a tedy $|closure(T)| \leq |X|$). Provedení jedné iterace vyžaduje zřejmě polynomiální čas, problém tedy určitě patří do třídy P.

Dá se ukázat, že pokud přidáme podmínku, že $\circ$ musí být asociativní operace, pak problém patří do NC, bez této podmínky se však jedná o P-úplný problém.

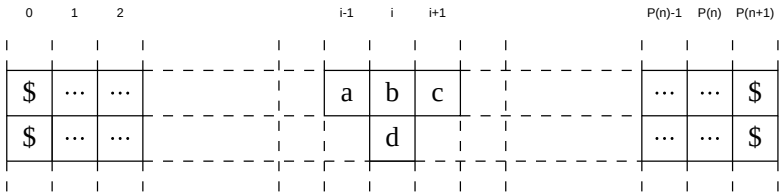
Při důkazu P-úplnosti použijeme pomocný problém gen', který je definován stejně jako problém Generovatelnost s tím rozdílem, že binární operace $\circ$ je nahrazena ternální operací $next(u, v, w)$. Ukážeme P-úplnost problému gen' a tento problém pak převedeme na problém Generovatelnost.

Věta 2.1 Problém gen' je P-úplný.

Nyní je třeba ukázat, že libovolný problém L ze třídy P je možné převést na problém gen'. Využijeme známého faktu, že pokud je možné problém řešit v polynomiálním čase při použití stroje RAM, pak je ho možné řešit v polynomiálním čase i při použití jednopáskového Turingova stroje. Pro jednoduchost a bez ztráty na obecnosti se můžeme omezit na problémy typu ANO/NE.

Pokud vezmeme libovolný problém L typu ANO/NE takový, že $L \in P$, pak musí existovat deterministický Turingův stroj M, který tento problém řeší, přičemž počet kroků, které Turingův stroj M provede při výpočtu nad slovem velikosti n je shora omezen nějakým polynomem $P(n)$. Budeme předpokládat, že tento polynom je explicitně znám.

O Turingovu stroji M budeme předpokládat, že se jedná o Turingův stroj s jednou jednostranně omezenou páskou, jejíž políčka jsou očíslována $1, 2, 3, \ldots$. Stroj M bude mít jediný koncový stav odpovídající odpovědi ANO (označíme ho q_{ANO}) a při dosažení tohoto stavu bude vždy platit, že se čtecí hlava nachází na políčku 1, přičemž toto políčko musí obsahovat symbol blank (#). (Pokud by Turingův stroj M toto nesplňoval, můžeme ho upravit odpovídajícím způsobem.)



Obrázek 2: Dvě po sobě jdoucí konfigurace Turingova stroje

Během výpočtu navštíví Turingův stroj maximálně prvních $P(n)$ políček na pásce (ve skutečnosti až $P(n)+1$, pokud provede přesně $P(n)$ kroků, ale jedničku můžeme pro jednoduchost zahrnout do polynomu $P(n)$), ostatní políčka pásky obsahují symbol blank (#) a pro výpočet nejsou důležitá.

Výpočet Turingova stroje, což je posloupnost konfigurací, může být zapsán do tabulky s řádky očíslovanými $0 \ldots P(n)$ a sloupci očíslovanými $0 \ldots P(n)+1$. Řádky této tabulky odpovídají jednotlivým konfiguracím (řádek s číslem t, odpovídá konfiguraci po provedení t kroků výpočtu, řádek s číslem 0 tedy odpovídá počáteční konfiguraci), sloupce odpovídají jednotlivým políčkům pásky s tím, že políčka ve sloupcích 0 a $P(n)+1$ obsahují speciální symbol \$. Ostatní políčka obsahují buď symboly páskové abecedy Turingova stroje M nebo dvojice tvořené symbolem páskové abecedy a stavem řídící jednotky Turingova stroje. V každém řádku (tj. v každé konfiguraci) je právě jedno políčko obsahující symbol a stav, je to políčko, na kterém je v této konfiguraci nastavena čtecí/zapisovací hlava Turingova stroje. Pokud je výpočet kratší než $P(n)$ kroků, opakuje se konfigurace, ve které se výpočet zastavil i v dalších řádcích tak, aby byla zaplněna celá tabulka.

Je zřejmé, že pro $1 \leq i \leq P(n)$ závisí obsah i-tého políčka v řádku $t+1$ pouze na obsahu políček ve sloupcích $i-1$, i a $i+1$ v řádku t a přechodové funkci Turingova stroje, neboť dvě po sobě jdoucí konfigurace se mohou lišit pouze v obsahu políček v bezprostřední blízkosti čtecí/zapisovací hlavy (viz obrázek 2, kde obsah políčka označeného d závisí na obsahu políček označených a, b a c).

Instanci problému gen' sestrojíme k problému L následujícím způsobem: Množina X bude tvořena všemi trojicemi (t,p,sym), kde $0 \leq t \leq P(n)$, $0 \leq p \leq P(n)+1$ a $sym \in \Gamma \cup (\Gamma \times Q)$, přičemž Γ je pásková abeceda Turingova stroje a Q je množina stavů jeho řídící jednotky. Trojice (t,p,sym) označuje, že v tabulce se v řádku t ve sloupci p nachází symbol sym.

Množina T v instanci problému gen' bude tvořena těmi trojicemi z množiny X, které odpovídají počáteční konfiguraci Turingova stroje M v řádku 0 tabulky. Ternální operaci $next(u,v,w)$ definujeme tak, že $next(u,v,w) = z$ právě tehdy, když u, v a w jsou trojice odpovídající políčkům $p-1$, p a $p+1$ v řádku t tabulky a z je trojice odpovídající obsahu políčka p v řádku $t+1$, přičemž tento obsah musí odpovídat správně provedenému kroku Turingova stroje M podle jeho přechodové funkce.

Je celkem zřejmé, že do uzávěru množiny T bude patřit trojice (t,p,sym) právě tehdy, když se v tabulce korektně popisující výpočet stroje M nachází v políčku p na řádku t symbol sym.

Když tedy v instanci problému gen' položíme s rovno $(P(n),1,(q_{ANO},\#))$, bude platit, že s bude patřit do uzávěru T právě tehdy, když Turingův stroj M dá při výpočtu odpověď ANO.

Výše popsaná konstrukce může být snadno realizována v polylogaritmickém čase při použití polynomiálního počtu procesorů. Důležité je, že velikost množiny X i velikost operace

next (uložené jako tabulka hodnot pro všechny možné trojice operandů) je možné omezit polynomem.

Věta 2.2 Problém Generovatelnost je P-úplný.

Nyní popíšeme, jakým způsobem je možné k instanci $(X, T, next, s)$ problému gen' sestrojit instanci problému Generovatelnost.

Definujeme $X' = X \cup (X \times X)$ a dále $u \circ v = (u, v)$ pro všechna u, v z množiny X a $(u, v) \circ w = next(u, v, w)$. Instance problému Generovatelnost bude mít stejnou množinu T i prvek s. Instance problému Generovatelnost tedy bude $(X', T, \circ, s)$. Je možné snadno ověřit, že prvek s patří do uzávěru množiny T vzhledem k operaci $\circ$ nad množinou X' právě tehdy, když patří do uzávěru množiny T vzhledem k operaci *next* nad množinou X. Vytvoření instance problému Generovatelnost k instanci problému gen' se dá snadno realizovat jako NC-redukce.

3 Další P-úplné problémy

V této části budou popsány některé další P-úplné problémy včetně naznačení důkazů jejich P-úplnosti. Podstatou všech těchto důkazů bude popis NC-redukce nějakého problému, o němž již bude známo, že je P-úplný, na daný problém.

3.1 Hodnota na výstupu booleovského obvodu

Dalším P-úplným problémem je problém zjištění hodnoty na výstupu booleovského obvodu (circuit-value problem — CVP). Tento problém je definován takto:

Hodnota na výstupu booleovského obvodu (CVP)

Instance: Booleovský obvod a specifikovaná množina vstupních hodnot

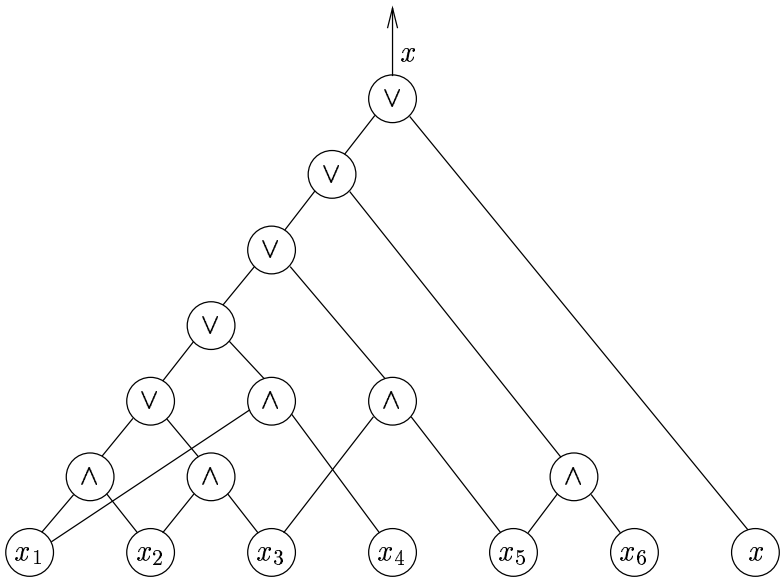
Otázka: Je na výstupu obvodu hodnota **true**?

Booleovský obvod je acyklický orientovaný graf, jehož vrcholy jsou hradla realizující různé booleovské funkce (jako **and**, **or** nebo **not**). Vrcholy grafu, do kterých nevstupují žádné hrany tvoří vstupy booleovského obvodu. Pokud jsou těmto vstupům přiřazeny hodnoty z množiny {**true**, **false**} je možné ohodnotit i ostatní vrcholy grafu hodnotami z této množiny. Ohodnocení každého vrcholu (hradla) závisí na ohodnocení vrcholů, ze kterých vedou hrany (vodiče) do tohoto vrcholu a na booleovské funkci, kterou toto hradlo realizuje. Většinou se omezujeme na obvody, ve kterých do každého hradla vstupují maximálně dvě hrany (vodiče). Jedno z hradel v obvodu musí být označeno jako výstupní, výstupem celého obvodu je pak pak hodnota na výstupu tohoto hradla.

Věta 3.1 Problém CVP je P-úplný problém.

Tento problém je možné řešit sekvenčním algoritmem v polynomiálním čase, je tedy třeba pouze ukázat, že je možné pomocí NC-redukce převést libovolný problém na problém CVP.

Ukážeme NC-redukci problému Generovatelnost na problém CVP. Nechť $(X, T, \circ, s)$ je instance problému Generovatelnost. Vytvoříme booleovský obvod, který v podstatě realizuje



Obrázek 3: Příklad konstrukce booleovského obvodu

algoritmus pro výpočet uzávěru množiny T, který byl popsán v části textu věnované problému Generovatelnost.

Tento obvod se bude skládat z několika „vrstev“. Celkový počet těchto vrstev bude $|X| - |T|$, tedy stejný jako maximální počet iterací v dříve uvedeném algoritmu pro výpočet uzávěru množiny T. Každá tato vrstva bude tvořena $|X|$ hradly přičemž každé hradlo bude odpovídat jednomu prvku množiny X. Přiřazení hodnot **true** a **false** jednotlivým hradlům bude odpovídat tomu, zda v dané iteraci patří daný prvek do množiny T' (pak bude na výstupu hradla hodnota **true**) nebo ne (pak bude na výstupu hradla hodnota **false**). Hodnoty na vstupech obvodu budou nastaveny podle toho, jestli daný prvek patří do množiny T nebo ne.

Jednotlivé vrstvy budou propojeny následujícím způsobem: Pro každý prvek x z množiny X najdeme všechny dvojice $(y_1, z_1), (y_2, z_2), \ldots, (y_k, z_k)$ takové, že $y_i \circ z_i = x$ pro všechna i, $1 \leq i \leq k$. Část obvodu, která propojuje hradlo odpovídající prvku x v nějaké vrstvě s prvky v předchozí vrstvě pak realizuje výpočet $(y_1 \wedge z_1) \vee (y_2 \wedge z_2) \vee \cdots \vee (y_k \wedge z_k) \vee x$, přičemž v tomto zápisu odpovídá y_i, z_i nebo x hodnotě, kterou má hradlo odpovídající prvku y_i, z_i nebo x v předchozí vrstvě.

Na obrázku 3 je příklad konstrukce takové části obvodu, kde platí, že (x_1, x_2), (x_2, x_3), (x_1, x_4), $(x_3 x_5)$ a (x_5, x_6) jsou všechny dvojice prvků (y, z) takové, že $y \circ z = x$.

Výstupem celého obvodu je výstup hradla, které odpovídá prvku s v poslední vrstvě. Snadno ověříme, že výstup booleovského obvodu bude **true** právě tehdy, když prvek s patří do uzávěru množiny T.

Výše popsanou konstrukci je možné realizovat jako NC-redukci, přestože detaily této konstrukce nejsou v tomto textu popsány.

Dokázat P-úplnost problému CVP je též možné použitím podobných myšlenek, které byly použity v důkazu P-úplnosti problému Generovatelnost. Můžeme sestrojit obvod, který v podstatě realizuje výpočet libovolného Turingova stroje, nebo lépe řečeno realizuje tabulku obsahující jednotlivé konfigurace Turingova stroje, podobně jako tomu bylo při důkazu P-

úplnosti problému Generovatelnost.

Problém CVP zůstává P-úplným problém i když se omezíme jen na tzv. monotónní obvody, které mohou obsahovat pouze hradla typu **and** a **or** (nemohou tedy obsahovat hradla typu **not**). To je snadno vidět z toho, že ve výše popsané konstrukci byla použita pouze hradla typu **and** nebo **or**.

Pokud platí, že hradla v obvodu jsou uspořádána do posloupnosti takovým způsobem, že v této posloupnosti předcházejí všechna hradla, ze kterých vedou vodiče do nějakého hradla, tomuto hradlu, nazýváme takový obvod topologicky uspořádaný (topologically sorted). Platí, že problém CVP zůstává P-úplný i když přidáme podmínku, že vstupem musí být monotónní topologicky uspořádaný obvod.

To se dá ukázat tak, že výše popsanou konstrukci rozšíříme o utvoření takovéhoto uspořádání. Vytvoření takového uspořádání zde nebudeme detailně popisovat, ale princip spočívá v tom, že hradla uspořádáme podle vrstev, ve kterých se nacházejí (v rámci jedné vrstvy libovolně). Toto je opět možné realizovat jako NC-redukci.

3.2 Neprázdnost bezkontextového jazyka

Mějme následující problém nazvaný problém neprázdnosti bezkontextového jazyka (context-free emptiness — CFE):

Neprázdnost bezkontextového jazyka (CFE)

Instance: Bezkontextová gramatika G.

Otázka: Obsahuje jazyk $L(G)$ generovaný gramatikou G alespoň jedno slovo ?

Pokud uvažujeme gramatiku G bez terminálních symbolů, pak je tento problém ekvivalentní problému jestli do $L(G)$ patří prázdné slovo ε.

Věta 3.2 CFE je P-úplný problém.

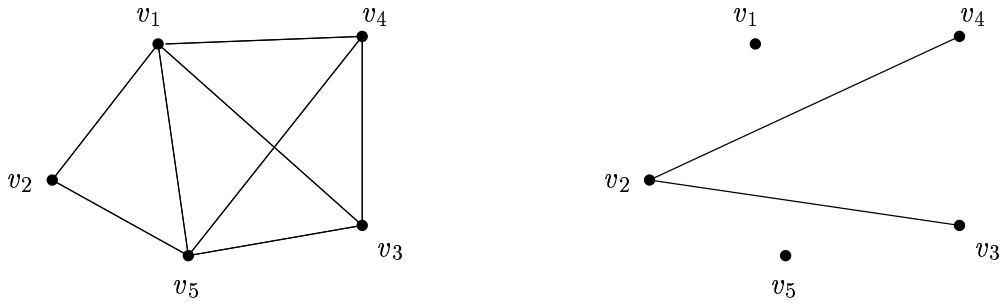
Problém je snadno řešitelný v polynomiálním čase. Jako důkaz P-úplnosti stačí ukázat NC-redukci nějakého P-úplného problému na tento problém. K tomuto účelu použijeme opět problém Generovatelnost.

Nechť $(X, T, \circ, s)$ je instance problému Generovatelnost. Zkonstruujeme bezkontextovou gramatiku G, kde X bude množina neterminálních symbolů, množina terminálních symbolů bude prázdná, s bude počátečním neterminálním symbolem a množina přepisovacích pravidel bude obsahovat pravidla:

1. $x \to yz$ pokud platí, že $y \circ z = x$,
2. $x \to \varepsilon$ pokud platí, že $x \in T$.

V této gramatice je možné libovolný neterminál x přepsat na prázdné slovo ε právě tehdy, když x patří do uzávěru T. Konstrukce gramatiky se dá velmi snadno paralelizovat, jedná se tedy o NC-redukci.

I když přidáme podmínku, že gramatika nesmí obsahovat ε-pravidla, problém bude stále P-úplný, neboť místo pravidel typu $x \to \varepsilon$ můžeme přidat pravidla typu $x \to a$, kde a je nějaký terminální symbol, pro všechna x patřící do T.



Obrázek 4: Příklad grafu a jeho doplňkového grafu

3.3 Lexikograficky první maximální klika a lexikograficky první nezávislá množina

Problém lexikograficky první maximální kliky (lexicographically first maximum clique — LFMC) je definován takto:

Lexikograficky první maximální klika (LFMC)

Instance: Neorientovaný graf G s uspořádanou množinou vrcholů $\{v_1, v_2, \ldots, v_n\}$.

Problém: Najít lexikograficky první maximální kliku grafu G.

Klika grafu je množina vrcholů, pro které platí, že každé dva vrcholy z této množiny jsou spolu incidentní. Maximální klika je klika, pro kterou platí, že žádná její vlastní nadmnožina není klika. (Maximální klika je něco jiného než největší klika, což je klika, pro kterou platí, že žádná jiná klika neobsahuje větší počet vrcholů. Problém nalezení největší kliky patří mezi NP-úplné problémy.)

Lexikograficky první maximální kliku je možné vypočítat následujícím (sekvenčním) algoritmem:

$clique \leftarrow \emptyset$
for $i \leftarrow 1$ to n **do**
  if v_i je incidentní se všemi vrcholy v $clique$ **then**
    $clique \leftarrow clique \cup \{v_i\}$
  end if
end for

Pokud například použijeme tento algoritmus pro graf na obrázku 4(a), dostaneme jako výsledek kliku $\{v_1, v_2, v_5\}$.

S pojmem klika úzce souvisí pojem nezávislá množina. Nezávislá množina je množina vrcholů, pro kterou platí, že žádné dva vrcholy z této množiny nejsou spolu incidentní.

Problém lexikograficky první maximální nezávislé množiny (lexicographically first maximal independent set — LFMIS) je definován takto:

Lexikograficky první maximální nezávislá množina (LFMIS)

Instance: Neorientovaný graf G s uspořádanou množinou vrcholů $\{v_1, v_2, \ldots, v_n\}$.

Problém: Najít lexikograficky první maximální nezávislou množinu grafu G.

Algoritmus pro výpočet lexikograficky první maximální nezávislé množiny se bude lišit od algoritmu pro výpočet lexikograficky první maximální kliky pouze v podmínce: vrchol bude do nezávislé množiny přidán pouze tehdy, když nebude incidentní se žádným vrcholem, který již byl do nezávislé množiny přidán.

Když máme dán graf $G = (V, E)$, nazýváme doplňkovým grafem grafu G graf $G' = (V, E')$, kde pro každou dvojici vrcholů $u, v \in V$ platí, že hrana (u, v) patří do E', právě tehdy když nepatří do E. Například doplňkový graf grafu na obrázku 4(a) je na obrázku 4(b) (a naopak doplňkový graf grafu na obrázku 4(b) je na obrázku 4(a)).

Platí, že klika v grafu tvoří nezávislou množinu v jeho doplňkovém grafu a naopak. Snadno se dá ukázat, že určitá množina vrcholů je řešením problému LFMC pro graf G právě tehdy, když je řešením problému LFMIS pro doplňkový graf grafu G.

Konstrukce doplňkového grafu je NC-redukce, a tak pokud je LFMC P-úplný problém, pak je P-úplný problém i LFMIS a naopak. Stačí tedy ukázat P-úplnost pouze jednoho z nich.

Věta 3.3 LFMC a LFMIS jsou P-úplné problémy.

Následuje popis NC-redukce problému CVP na problém LFMIS. Předpokládáme, že instance problému CVP je monotónní topologicky uspořádaný obvod s množinou hradel $V = \{v_1, \ldots, v_n\}$, přičemž vstupy jsou vrcholy s nejnižšími indexy a výstupem je v_n. Zkonstruujeme neorientovaný graf $G' = (V', E')$, kde $V' = \{v'_1, v''_1, v'_2, v''_2, \ldots, v'_n, v''_n\}$. Vrcholy z množiny V' budou očíslovány od 1 do $2n$ v naznačeném pořadí, pouze v případě, že v_i je hradlo typu **or** nebo když v_i je vstupem s hodnotou **false**, bude pořadí vrcholů v'_i a v''_i opačné, tj. v''_i, v'_i.

Dále přidáme ke grafu G' hrany a to takovým způsobem, že zajistíme, že lexikograficky první maximální nezávislá množina bude odpovídat přiřazení booleovských hodnot jednotlivým hradlům. Pokud na výstupu hradla v_i bude hodnota **true**, bude do nezávislé množiny patřit vrchol v'_i, pokud bude na výstupu hradla hodnota **false**, bude do nezávislé množiny patřit vrchol v''_i.

Ke grafu G' přidáme následující hrany:

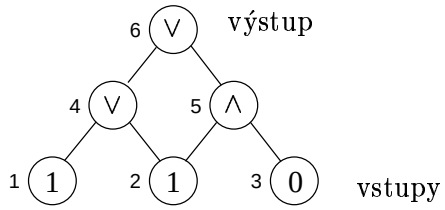
(a) Pro každé hradlo v přidáme hranu mezi vrcholy v' a v'', čímž zajistíme, že nanejvýš jeden z těchto vrcholů může patřit do nezávislé množiny.

(b) Pokud je v hradlo typu **and** a jeho vstupy jsou u a w, pak přidáme hrany (u'', v') a (w'', v'), takže v' může patřit do nezávislé množiny jen tehdy, když tam nepatří u'' ani w''.

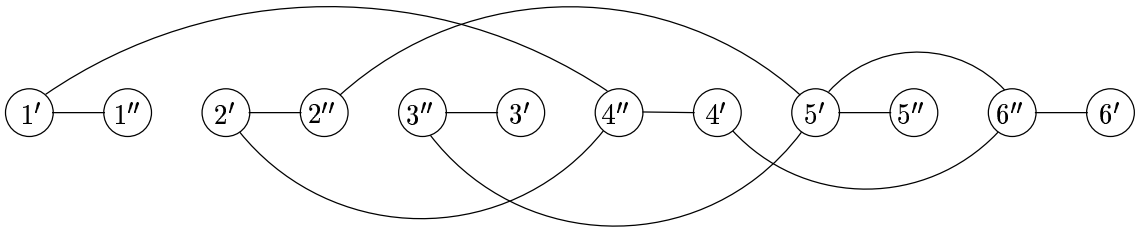
(c) Pokud je v hradlo typu **or**, jehož vstupy jsou u a w, pak přidáme hrany (u', v'') a (w', v''), takže v'' bude patřit do nezávislé množiny jen pokud tam nepatří u' ani w', čili v' bude patřit do nezávislé množiny pokud tam patří alespoň jeden z vrcholů u' nebo w'.

Příklad takové konstrukce grafu k obvodu na obrázku 5 je na obrázku 6.

Problém, jaká hodnota je na výstupu booleovského obvodu, se redukuje na otázku, jestli vrchol v'_n patří do lexikograficky první maximální nezávislé množiny. Konstrukci grafu G' je možné provést pro každý vrchol paralelně.



Obrázek 5: Obvod G



Obrázek 6: Graf G' odpovídající obvodu G na obrázku 5

3.4 Prohledávání grafu do hloubky

Poslední ze zde uvedených problémů se týká prohledávání grafu do hloubky (depth-first search), což je způsob průchodu grafem používaný v řadě sekvenčních grafových algoritmů.

V následujícím textu uvažujeme pouze orientované grafy. Předpokládejme, že graf G je reprezentován pomocí seznamů incidentních vrcholů, což znamená, že pro každý vrchol v máme dán seznam vrcholů $ADJ(v)$, což je seznam vrcholů, do kterých vedou hrany z vrcholu v.

Následující algoritmus přiřadí každému vrcholu v číslo, které bude odpovídat pořadí, ve kterém byl vrchol v navštíven při průchodu grafem. Toto číslo bude označeno jako index vrcholu (depth-first index) $DFI(v)$. Algoritmus je rekurzivní. Když je zavolán jako $dfs(v_0)$, přiřadí odpovídající indexy všem vrcholům dosažitelným z v_0, přičemž $DFI(v_0) = 1$. Proměnná *counter* je globální proměnná nastavená na začátku na nulu. Rovněž se předpokládá, že na začátku je hodnota $DFI(v)$ rovna nule pro všechna v.

procedure $dfs(v)$
begin
if $DFI(v) = 0$ **then**
  $counter \leftarrow counter + 1$
  $DFI(v) \leftarrow counter$
  for $\forall u \in ADJ(v)$ v pořadí podle seznamu **do**
    $dfs(u)$
  end for
end if
end

Formálně je problém prohledávání do hloubky (depth-first search — DFS) definován takto:

Prohledávání do hloubky (DFS)

Instance: Orientovaný graf G reprezentovaný pomocí seznamů incidentních vrcholů a specifikovaný počáteční vrchol v_0.

Problém: Najít indexy odpovídající pořadí, ve kterém by byly navštíveny při prohledávání do hloubky, pro všechny vrcholy grafu G dosažitelné z vrcholu v_0 počínaje tímto vrcholem ($DFI(v_0) = 1$).

Věta 3.4 Problém DFS je P-úplný problém.

Ukážeme NC-redukci problému CVP na problém DFS. K danému booleovskému obvodu sestrojíme graf G tak, že každému hradlu v původním booleovském obvodu bude odpovídat určitá část grafu G, přičemž v této části budou určeny dva vrcholy s a t tak, že bude platit, že na výstupu daného hradla bude hodnota **true** právě tehdy, když bude platit $DFI(s) < DFI(t)$ (v případě, že na výstupu hradla bude hodnota **false**, bude platit $DFI(s) > DFI(t)$).

Budeme předpokládat, že booleovský obvod bude topologicky uspořádaný $(v_1, v_2, \ldots, v_n)$, že hodnota všech všech vstupů bude **true** a že obvod bude obsahovat pouze hradla typu **nor**, to znamená hradla realizující funkci **not**(x **or** y). Dá se snadno ověřit, že problém CVP zůstává i za těchto podmínek P-úplný. K libovolnému obvodu můžeme sestrojit obvod realizující stejnou booleovskou funkci a přitom obsahující pouze hradla typu **nor**, neboť určitou kombinací hradel typu **nor** můžeme realizovat různé booleovské funkce, jako jsou **and**, **or** nebo **not**. Počet hradel nového obvodu je úměrný počtu hradel původního obvodu. U vstupů, které mají hodnotu **false** přidáme před vstup hradlo typu **not** (realizované pomocí hradla typu **nor**). Když teď budou mít všechny vstupy hodnotu **true**, bude na výstupu stejná hodnota jako v původním obvodu při původních vstupních hodnotách.

Předpokládejme, že booleovský obvod obsahuje hradla $(1, 2, \ldots, n)$. Ke každému hradlu i sestrojíme graf G_i. Jednotlivé grafy G_i nebudou mít žádné společné hrany, ale budou mít některé společné vrcholy. Spojením všech těchto grafů (pomocí společných vrcholů) vznikne graf G.

Ke vstupnímu hradlu i, které nemá žádné vstupy a jehož výstupy vedou do hradel j_1, j_2, …, j_k, sestrojíme graf G_i, který bude obsahovat vrcholy *first*(i), *last*(i), $s(i)$, $t(i)$, $i\#j_1$, …, $i\#j_k$ a dalších k pomocných vrcholů. Struktura tohoto grafu, pro případ kdy $k = 3$, je naznačena na obrázku 7.

Ke hradlu i typu **nor**, do kterého vstupují vodiče z hradel i_1 a i_2 a jehož výstupy vedou do hradel $j_1, j_2, \ldots, j_k$, sestrojíme graf G_i, který bude obsahovat vrcholy *first*(i), *last*(i), $s(i)$, $t(i)$, $i\#j_1$, …, $i\#j_k$ a dalších k pomocných vrcholů. Struktura tohoto grafu, pro případ kdy $k = 3$, je naznačena na obrázku 8.

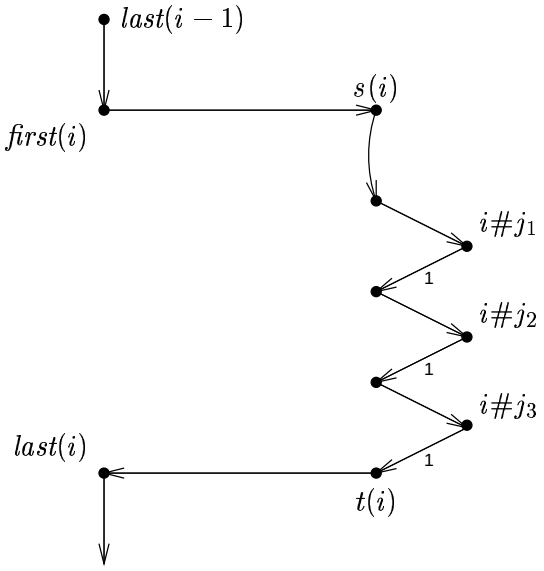
Vrcholy označené $i\#j$ jsou společné vrcholy sdílené podgrafy G_i a G_j. Žádné jiné vrcholy nejsou sdíleny více podgrafy.

Na obou obrázcích udávají čísla u hran pořadí vrcholů, do kterých tyto hrany vedou, v seznamech incidentních vrcholů. Pokud číslo není u hrany uvedeno, na pořadí vrcholu nezáleží.

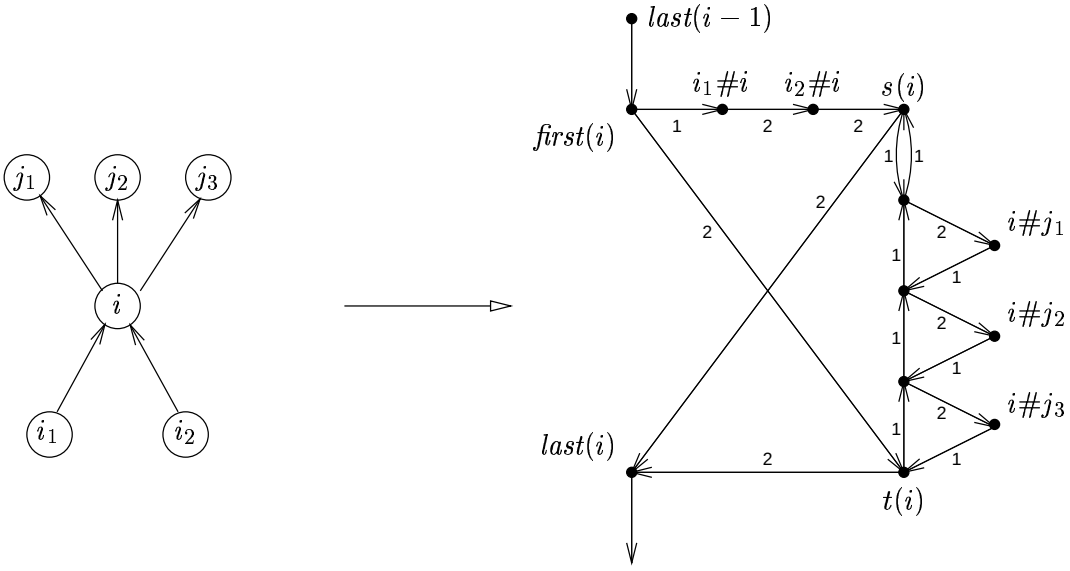
Při průchodu grafem G začneme vrcholem *first*(1) a projdeme podgrafem G_1, pak hranou (*last*(1), *first*(2)) přejdeme do grafu G_2 a poté projdeme podobně podgrafy G_2, G_3, …, G_n (při tomto průchodu však nemusíme nutně navštívit všechny vrcholy daných podgrafů). Když dojdeme do vrcholu *last*(n), začneme se vracet zpět, přičemž navštívíme všechny vrcholy, které dosud nebyly navštíveny.

Pro každé hradlo i platí, že pokud je na výstupu tohoto hradla hodnota **true**, navštívíme při prvním průchodu podgrafem G_i všechny vrcholy $i\#j_1, i\#j_2, \ldots, i\#j_k$, kde $j_1, j_2, \ldots, j_k$ jsou hradla, do kterých vedou výstupy z hradla i. Pokud je naopak na výstupu hradla i hodnota **false** nenavštívíme při prvním průchodu žádný z těchto vrcholů.

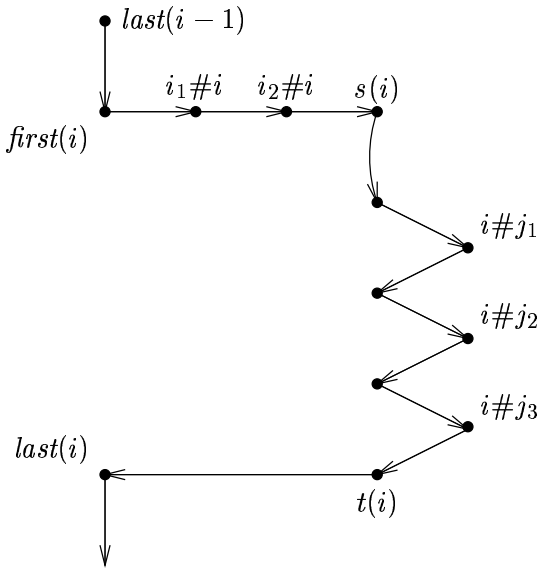
Pokud je na výstupu hradla i hodnota **true**, musíme grafem G_i projít způsobem naznačeným na obrázku 9. U vstupního hradla nemáme jinou možnost, protože neobsahuje žádné



Obrázek 7: Podgraf G_i pro vstupní hradlo i (vstup musí mít hodnotu **true**)



Obrázek 8: Hradlo i typu **nor** a podgraf G_i odpovídající tomuto hradlu



Obrázek 9: Průchod podgrafem G_i v případě, že hradlo i má na výstupu hodnotu **true**

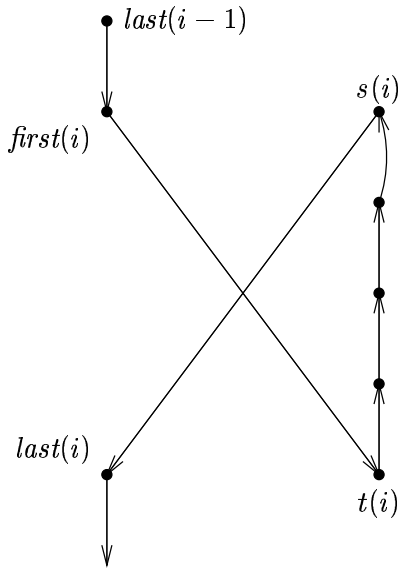
jiné hrany. V případě hradla typu **nor** bude na výstupu tohoto hradla hodnota **true** pouze v případě, že na vstupech tohoto hradla jsou hodnoty **false**, což znamená, že dosud nebyl navštíven ani jeden z vrcholů $i_1\#i, i_2\#i$. V tom případě projdeme těmito dvěma vrcholy a pokračujeme způsobem naznačeným na obrázku 9.

Pokud je na výstupu hradla i hodnota **false**, musíme grafem G_i projít způsobem naznačeným na obrázku 10. U vstupního hradla tato možnost nemůže nastat, protože na vstupech může být pouze hodnota **true**. V případě hradla typu **nor** bude na výstupu tohoto hradla hodnota **false** pouze v případě, že alespoň jeden vstup tohoto hradla má hodnotu **true**, což znamená, že alespoň jeden z vrcholů $i_1\#i, i_2\#i$ již byl navštíven. V tom případě se musíme z tohoto již navštíveného vrcholu vrátit do vrcholu *first*(i) a pokračovat z něj do vrcholu $t(i)$.

Reference

[1] Gibbons, A., Rytter, W.: *Efficient parallel algorithms.* Cambridge University Press, 1988.

[2] Sipser, M.: *Introduction to the theory of computation.* PWS Publishing Company, 1997.



Obrázek 10: Průchod podgrafem G_i v případě, že hradlo i má na výstupu hodnotu **false**

Obsah