Výpočetní složitost algoritmů

Slajdy pro výuku na KS

Ondřej Čepek

Sylabus

1. Definice časové a prostorové složitosti algoritmů. Příklady na konkrétních algoritmech. Prostředky pro popis výpočetní složitosti algoritmů (asymptotická notace), příklady použití asymptotické notace.

2. Optimalizační úlohy a rozhodovací problémy. Problémy řešitelné deterministicky v polynomiálním čase (třída P), problémy řešitelné nedeterministicky v polynomiálním čase (třída NP), polynomiální převoditelnost problémů.

3. NP-těžkost a NP-úplnost. Definice, důkazové postupy, příklady NP-úplných problémů.

4. Pseudopolynomiální algoritmy a jejich příklady. Silná NP-úplnost, příklady silně NP-úplných problémů.

5. Aproximační algoritmy pro NP-těžké optimalizační úlohy. Neaproximovatelnost, příklady neaproximovatelných úloh.

Jak porovnávat kvalitu různých algoritmů?

časová složitost algoritmu
prostorová složitost algoritmu

} oboje závisí na „velikosti“ vstupních dat

Jak měřit velikost vstupních dat?

rigorózně: počet bitů nutných k zapsání vstupních dat

Příklad: vstupem jsou (přirozená) čísla $a_1, a_2, \dots, a_n$ která je třeba setřídit

velikost dat D v binárním zápisu je $|D| = \lceil \log_2 a_1 \rceil + \dots + \lceil \log_2 a_n \rceil$

časová složitost = funkce $f(|D|)$ udávající počet kroků algoritmu v závislosti na velikosti vstupních dat

intuitivně: není podstatný přesný tvar funkce f (multiplikativní a aditivní konstanty), ale pouze to, do jaké „třídy“ funkce f patří (lineární, kvadratická, exponenciální, …)

Příklad: $f(|D|) = a\,|D| + b$ lineární algoritmus

$f(|D|) = a\,|D|^2 + b\,|D| + c$ kvadratický algoritmus

$f(|D|) = k\,2^{\text{‘}D\text{‘}}$ exponenciální algoritmus

Co je to krok algoritmu?

rigorózně: operace daného abstraktního stroje (Turingův stroj, stroj RAM)

zjednodušení (budeme používat): krok algoritmu = operace proveditelná v konstantním (tj. na velikosti dat nezávislém) čase

- aritmetické operace (sčítání, odčítání, násobení, …)
- porovnání dvou hodnot (typicky čísel)
- přiřazení (pouze pro jednoduché datové typy, ne pro pole …)

→ tím se zjednoduší i měření velikosti dat (čísla mají pevnou maximální velikost)

Příklad: setřídit čísla $a_1, a_2, \ldots, a_n$ → velikost dat je $|D| = n$

Toto zjednodušení nevadí při porovnávání algoritmů, ale může vést k chybě při zařazování algoritmů do tříd složitosti

Proč měřit časovou složitost algoritmů?

stačí přeci mít dostatečně rychlý počítač …

Doba provádění f(n) operací (délka běhu algoritmu) pro vstupní data velikosti n za předpokladu že použitý hardware je schopen vykonat 1 milion operací za sekundu

	n						
f(n)	20	40	60	80	100	500	1000
n	20μs	40μs	60μs	80μs	0.1ms	0.5ms	1ms
n log n	86μs	0.2ms	0.35ms	0.5ms	0.7ms	4.5ms	10ms
n^2	0.4ms	1.6ms	3.6ms	6.4ms	10ms	0.25s	1s
n^3	8ms	64ms	0.22s	0.5s	1s	125s	17min
2^n	1s	11.7dní	36tis.let				
n!	77tis.let						

Růst rozsahu zpracovatelných úloh, tj. „zvládnutelné“ velikosti vstupních dat, díky zrychlení výpočtu (lepší hardware) za předpokladu, že na „stávajícím“ hardware lze řešit úlohy velikosti x

	Zrychlení výpočtu			
f(n)	původní	10 krát	100 krát	1000 krát
n	x	10x	100x	1000x
n log n	x	7.02x	53.56x	431.5x
n^2	x	3.16x	10x	31.62x
n^3	x	2.15x	4.64x	10x
2^n	x	x+3	x+6	x+9

Asymptotická (časová) složitost

Intuitivně: zkoumá „chování“ algoritmu na „velkých“ datech, tj. nebere v úvahu multiplikativní a aditivní konstanty, pouze zařazuje algoritmy do „kategorií“ podle jejich skutečné časové složitosti

Rigorózně:

$f(n)$ je asymptoticky menší nebo rovno $g(n)$, značíme $f(n) \in O(g(n))$, pokud

$$\exists c>0\ \exists n_0>0\ \forall n \geq n_0 : 0 \leq f(n) \leq c\, g(n)$$

$f(n)$ je asymptoticky větší nebo rovno $g(n)$, značíme $f(n) \in \Omega(g(n))$, pokud

$$\exists c>0\ \exists n_0>0\ \forall n \geq n_0 : 0 \leq c\, g(n) \leq f(n)$$

$f(n)$ je asymptoticky stejné jako $g(n)$, značíme $f(n) \in \Theta(g(n))$, pokud

$$\exists c>0\ \exists d>0\ \exists n_0>0\ \forall n \geq n_0 : 0 \leq c\, g(n) \leq f(n) \leq d\, g(n)$$

Úlohy, optimalizační úlohy, rozhodovací problémy

Úloha: pro dané zadání najít strukturu s danými vlastnostmi

Příklady:

• v daném orientovaném grafu najdi cyklus

• vynásob dvě dané čtvercové matice

Optimalizační úloha: pro dané zadání najít optimální (většinou nejmenší nebo největší) strukturu s danými vlastnostmi

Příklady:

• v daném neorientovaném grafu najdi největší (počtem vrcholů) úplný podgraf (kliku)

• pro danou množinu úkolů najdi nejkratší rozvrh

Rozhodovací problém: pro dané zadání odpovědět ANO/NE

Příklady:

• existuje v daném neorientovaném grafu Hamiltonovská kružnice?

• je daná čtvercová matice regulární?

My se v následujícím omezíme jen na rozhodovací problémy, což lze (více méně) udělat bez újmy na obecnosti - v tom smyslu, že k většině (optimalizačních) úloh existuje „stejně těžký“ rozhodovací problém.

Třída P

Definice (vágní): Třída P je třída rozhodovacích problémů, pro které existuje (deterministický sekvenční) algoritmus běžící v polynomiálním čase (vzhledem k velikosti zadání), který správně rozhodne ANO/NE (který řeší daný problém).

- je daný orientovaný graf silně souvislý?
- obsahuje daný neorientovaný graf trojúhelník? (speciální případ „kliky“)
- je daná matice regulární?

Poznámka: Problémy patřící do třídy P se považují za efektivně řešitelné, u praktických problémů nebývá stupeň polynomu omezujícího časovou složitost algoritmu příliš vysoký (typicky ≤ 3 výjimečně ≤ 5) i když samozřejmě existují (umělé) problémy řešitelné polynomiálními algoritmy s libovolně vysokým stupněm polynomu.

Třída NP

Nedeterministický algoritmus = algoritmus, který v každém svém kroku může libovolně volit z několika možností

Nedeterministický algoritmus řeší daný rozhodovací problém ⇔ pro každé kladné zadání problému (odpověď ANO) existuje posloupnost voleb vedoucí k tomu, že algoritmus odpoví ANO, pro žádné záporné zadání taková posloupnost voleb neexistuje.

Definice (vágní): Třída NP je třída rozhodovacích problémů, pro které existuje nedeterministický sekvenční algoritmus běžící v polynomiálním čase (vzhledem k velikosti zadání), který řeší daný problém.

Jiný model nedeterministického algoritmu: dopředu provede volby (do paměti zapíše vektor čísel) a pak už provádí jednotlivé kroky původního algoritmu deterministicky.

Alternativní definice (opět vágní): Rozhodovací problém patří do třídy NP, pokud pro každé jeho kladné zadání existuje (polynomiálně velký) certifikát, pomocí něhož lze v polynomiálním čase (deterministicky) ověřit, že zadání je skutečně kladné (že odpověď na dané zadání je skutečně ANO).

Poznámka: Problémy patřící do třídy NP se považují za efektivně ověřitelné.

Příklady problémů ze třídy NP:

- KLIKA (úplný podgraf): Je dán neorientovaný graf G a číslo k.

  Otázka: Existuje v G úplný podgraf velikosti alespoň k?

- HK (Hamiltonovská kružnice): Je dán neorientovaný graf G.

  Otázka: Existuje v G Hamiltonovská kružnice?

- TSP (obchodní cestující): Je dán ohodnocený úplný neorientovaný graf G a číslo k.

  Otázka: Existuje v G Hamiltonovská kružnice celkové délky nejvýše k?

- SP (součet podmnožiny): Jsou dána přirozená čísla $a_1, a_2, \ldots, a_n, b$.

  Otázka: Existuje podmnožina čísel $a_1, a_2, \ldots, a_n$, jejíž součet je přesně b?

- ROZ (rozvr. na paralel. strojích): Je dán počet úkolů, jejich délky, počet strojů a číslo k.

  Otázka: Existuje přípustný rozvrh délky nejvýše k?

Ukážeme, že HK → TSP a SP → ROZ, kde A → B znamená, že pokud existuje polynomiální algoritmus řešící B potom také existuje polynomiální algoritmus řešící A, neboli vyřešit B je alespoň tak „těžké“ jako vyřešit A.

Převody (redukce) mezi rozhodovacími problémy

Nechť A,B jsou dva rozhodovací problémy. Říkáme, že A je polynomiálně redukovatelný na B, pokud existuje zobrazení f z množiny zadání problému A do množiny zadání problému B s následujícími vlastnostmi:

1. Nechť X je zadání problému A a Y zadání problému B takové, že Y = f(X). Potom je X kladné zadání problému A tehdy a jen tehdy, když je Y kladné zadání problému B.

2. Nechť X je zadání problému A. Potom je zadání f(X) problému B (deterministicky sekvenčně) zkonstruovatelné v polynomiálním čase vzhledem k velikosti X.

Poznámka: Z 2. také vyplývá, že velikost f(X) je polynomiální vzhledem k velikosti X.

NP-těžkost a NP-úplnost

Definice: Problém B je NP-těžký pokud pro libovolný problém A ze třídy NP platí, že A je polynomiálně redukovatelný na B.

Definice: Problém B je NP-úplný pokud 1) patří do třídy NP a 2) je NP-těžký.

Důsledek 1: Pokud je A NP-těžký a navíc je polynomiálně redukovatelný na B, tak je B také NP-těžký.

Důsledek 2: Pokud existuje polynomiální algoritmus pro nějaký NP-těžký problém, pak existují polynomiální algoritmy pro všechny problémy ve třídě NP.

Věta (Cook-Levin 1971): Existuje NP-úplný problém (dokázáno pro problém SAT).

Pseudopolynomiální algoritmy - příklad

Algoritmus pro SP: předpokládejme, že platí $a_1 \geq \ldots \geq a_n$, a že A je pole délky b.

```
for j := 1 to b do {A[j] := 0; a_0 := b+1};
for i := 1 to n do
        A[a_i] := 1;
        for j := b downto a_{i-1} do if (A[j] = 1) and (j+a_i ≤ b) then A[j+a_i] := 1;
SP := (A[b] = 1).
```

Platí: Po i-tém průchodu hlavním cyklem obsahuje pole A jedničky právě u těch indexů, které odpovídají součtům všech neprázdných podmnožin množiny $\{a_1, \ldots, a_i\}$, které jsou nejvýše rovny b.

Časová složitost: O(nb), což je

- exponenciální časová složitost vzhledem k binárně (ale také ternárně, dekadicky, …) kódovanému vstupu, ale
- polynomiální časová složitost vzhledem k unárně kódovanému vstupu.

Algoritmy s těmito vlastnostmi se nazývají pseudopolynomiální.

Pseudopolynomiální algoritmy - definice

Nechť je dán rozhodovací problém Q a jeho instance X. Definujme:

kód(X) = délka zápisu (počet bitů) instance X v binárním (či „vyšším“) kódováním

max(X) = největší číslo v X (velikost čísla, NE délka jeho binárního zápisu)

Definice: Algoritmus řešící Q se nazývá pseudopolynomiální, pokud je jeho časová složitost při spuštění na vstupu X omezena polynomem v proměnných kód(X) a max(X).

Poznámka: každý polynomiální algoritmus je samozřejmě také pseudopolynomiální.

Pozorování: Pokud je Q takový, že pro každou jeho instanci X platí $max(X) \leq p(kód(X))$ pro nějaký (pevný) polynom p, tak pro Q pojem polynomiálního a pseudopolynomiálního algoritmu splývá. Problémy, kde toto nenastává budeme nazývat číselné problémy.

Definice: Rozhodovací problém Q se nazývá číselný, pokud neexistuje polynom p takový, že pro každou instanci X problému Q platí $max(X) \leq p(kód(X))$.

Věta: Nechť Q je NP-úplný problém, který není číselný. Potom pokud $P \neq NP$, tak Q nemůže být řešen pseudopolynomiálním algoritmem.

Otázka: Je každý číslený problém řešitelný nějakým pseudopolynomiálním algoritmem?

Odpověď: NE (a typickým představitelům takových problémů se říká silně NP-těžké)

Silně NP- úplné problémy

Nechť je Q rozhodovací problém a p polynom. Symbolem Q_p označíme množinu instancí problému Q (tj. podproblém problému Q), pro které platí $max(X) \leq p(kód(X))$, tj.

$$Q_p = \{X \in Q \mid max(X) \leq p(kód(X))\}$$

Věta: Nechť je A pseudopolynomiální algoritmus řešící Q. Potom pro každý polynom p je A polynomiálním algoritmem řešícím Q_p.

Definice: Rozhodovací problém Q se nazývá silně NP-úplný, pokud $Q \in$ ***NP*** a existuje polynom p takový, že podproblém Q_p je NP-úplný.

Věta: Nechť Q je silně NP-úplný problém. Potom pokud ***P ≠ NP***, tak Q nemůže být řešen pseudopolynomiálním algoritmem.

Příklady číselných silně NP-úplných problémů:

Obchodní Cestující (TSP) :

- je to číselný problém (váhy na hranách mohou být libovolně velké)
- je silně NP-úplný neboť zůstává NP-úplný i když váhy omezíme (malou) konstantou

3-partition (3-P) – toto je „čistě“ číselný problém :

Zadání: $a_1, \ldots, a_{3m}, b \in N$, taková že $\forall j : ¼\, b < a_j < ½\, b$ a platí $\Sigma_{j=1}^{3m} a_j = mb$.

Otázka: $\exists S_1, \ldots, S_m$ disjunktní rozdělení množiny $\{1, \ldots, 3m\}$ takové, že $\forall i : \Sigma_{j \in Si} a_j = b$?

Aproximační algoritmy

Aproximační algoritmy jsou typicky používány na řešení "velkých" instancí NP-těžkých optimalizačních problémů, pro které je nalezení optimálního řešení "beznadějné", tj. časově příliš náročné (pro "malé" instance lze nalézt optimální řešení "hrubou silou" v exponenciálním čase). Aproximační algoritmus má následující tři vlastnosti:

1. Vrací (většinou) suboptimální řešení (někdy ale může vrátit i optimum).

2. Dává odhad kvality vráceného řešení vzhledem k optimu.

3. Běží v polynomiálním čase vzhledem k velikosti zadání.

Odhad kvality vráceného řešení

Značení: OPT = optimální řešení

APR = řešení vrácené aproximačním algoritmem

f(Z) = hodnota řešení Z (předpokládáme, že je vždy nezáporná)

Definice: Aproximační algoritmus A řeší optimalizační problém X s poměrovou chybou r(n), pokud pro všechna zadání problému X velikosti n platí

$$\max \{ f(APR) / f(OPT) , f(OPT) / f(APR) \} \leq r(n).$$

Definice: Aproximační algoritmus A řeší optimalizační problém X s relativní chybou e(n), pokud pro všechna zadání problému X velikosti n platí

$$|f(APR) - f(OPT)| / f(OPT) \leq e(n).$$

Příklad maximalizačního problému (optimalizační verze Kliky):

Pro daný neorientovaný graf najít největší (měřeno počtem vrcholů) kliku v daném grafu.

Aproximační algoritmus by musel poskytovat odhad (záruku kvality) tohoto typu

$$f(APR) \geq \frac{3}{4} f(OPT),$$

kde $f(X)$ je v tomto případě počet vrcholů (tj. velikost kliky) v řešení X.

Příklad minimalizačního problému (optimalizační verze Obchodního cestujícího):

Pro daný úplný vážený neorientovaný graf najít nejkratší Hamiltonovskou kružnici (měřeno součtem délek hran) v daném grafu.

Aproximační algoritmus by musel poskytovat odhad (záruku kvality) tohoto typu

$$f(APR) \leq 2 f(OPT),$$

kde $f(X)$ je v tomto případě délka Hamiltonovské kružnice v řešení X.

Příklady aproximačních algoritmů

Úloha vrcholového pokrytí (optimalizační verze):

Vstup: Neorientovaný graf $G = (V,E)$.

Úloha: Najít vrcholové pokrytí minimální velikosti, tj. najít $V' \subseteq V$ takové, že pro každé $(u,v) \in E$ platí $u \in V'$ nebo $v \in V'$ (nebo oboje), a navíc V' má minimální možnou kardinalitu.

Algoritmus A: opakovaně vyber v grafu vrchol nejvyššího stupně, přidej ho do postupně konstruovaného vrcholového pokrytí a odstraň ho z grafu spolu se všemi incidentními (a tedy pokrytými) hranami dokud nezbývá v grafu žádná hrana.

Má Algoritmus A konstantní relativní (poměrovou) chybu?

Algoritmus B: opakovaně vyber v grafu libovolnou hranu (u,v) dej jak u tak v do postupně konstruovaného vrcholového pokrytí a odstraň jak u tak v z grafu spolu se všemi incidentními (a tedy pokrytými) hranami dokud nezbývá v grafu žádná hrana.

Má Algoritmus B konstantní poměrovou chybu?

Úloha obchodního cestujícího (optimalizační verze):

Vstup: Úplný vážený neorientovaný graf $G = (V,E)$ a váhová funkce $c : E \rightarrow Z^+ \cup \{0\}$

Úloha: Najít v G Hamiltonovskou kružnici nejmenší celkové váhy (délky).

Obchodní cestující s trojúhelníkovou nerovností

Platí: $\forall\ u,v,w \in V : c(u,w) \leq c(u,v) + c(v,w)$

Napřed nutno zjistit: Je tento podproblém vůbec NP-těžký (a má tedy vůbec cenu uvažovat o aproximačních algoritmech)??

Algoritmus C:

a) Najdi minimální kostru grafu G.

b) Vyber libovolný vrchol grafu G a spusť z něj na nalezené kostře DFS, které očísluje vrcholy v preorder pořadí

c) Výsledná Hamiltonovská kružnice je dána pořadím (permutací) z bodu b)

Poznámka: Pokud je v bodě a) použit Primův (Jarníkův) algoritmus, tak celý algoritmus běží v čase $O(|E|) = O(|V|^2)$.

Věta: Algoritmus C má konstantní poměrovou chybu $r(n) \leq 2$.

Úloha rozvrhování na paralelních strojích:

Vstup: Je dán počet úkolů n, jejich délky $x_1, \ldots, x_n$ a počet strojů m.

Úloha: Zkonstruovat nejkratší možný rozvrh.

Naivní aproximační algoritmus FRONTA : bere úkoly postupně podle jejich čísel a každý úkol vždy umístí na stroj, který je volný nejdříve.

Značení: OPT = optimální rozvrh, Q = rozvrh zkonstruovaný algoritmem FRONTA,
délka(OPT) = o, délka(Q) = q

Věta: Pokud m je počet strojů, tak $q \leq ((2m - 1) / m)o$ a tento odhad již nelze zlepšit.

Důsledek: Aproximační algoritmus FRONTA má poměrovou chybu 2.

1. Těsnost odhadu: Pro každé m zkonstruujeme zadání, pro které platí v dokazované nerovnosti rovnost, a to následujícím způsobem

   $x_1 = x_2 = \ldots = x_{m-1} = m-1$ (m–1 úkolů délky m–1)

   $x_m = x_{m+1} = \ldots = x_{2m-2} = 1$ (m–1 úkolů délky 1)

   $x_{2m-1} = m$ (1 úkol délky m)

2. Platnost nerovnosti: Nechť j je úkol končící jako poslední v rozvrhu Q (končící v čase q) a nechť t je okamžik zahájení úkolu j. Potom žádný procesor nemá prostoj před časem t a platí $mq \leq (2m - 1)o$.

Lepší aproximační algoritmus USPOŘÁDANÁ FRONTA pro optimalizační verzi ROZ: pracuje stejně jako FRONTA, ale na začátku úkoly setřídí do nerostoucí posloupnosti podle jejich délek.

Značení: OPT = optimální rozvrh,
U = rozvrh zkonstruovaný algoritmem USPOŘÁDANÁ FRONTA,
délka(OPT) = o, délka(U) = u

Věta: Pokud m je počet strojů, tak $u \leq ((4m - 1) / 3m)o$ a tento odhad již nelze zlepšit.

Důsledek: Aproximační algoritmus USPOŘÁDANÁ FRONTA má poměrovou chybu 4/3.

Důkaz: Těsnost odhadu: Pro každé liché m zkonstruujeme zadání, pro které platí v dokazované nerovnosti rovnost, a to následujícím způsobem

$x_1 = x_2 = 2m-1$ (2 úkoly délky $2m-1$)

$x_3 = x_4 = 2m-2$ (2 úkoly délky $2m-2$)

$x_{2m-3} = x_{2m-2} = m+1$ (2 úkoly délky $m+1$)

$x_{2m-1} = x_{2m} = x_{2m+1} = m$ (3 úkoly délky m)

Lemma: Pokud pro všechny úkoly platí $x_i > 1/3o$ pak $u = o$.

Dokončení důkazu: Nechť j je úkol končící jako poslední v rozvrhu U (končící v čase u). Pokud $x_j > 1/3o$ tak použijeme Lemma, v opačném případě je důkaz velmi podobný jako pro algoritmus FRONTA.

Příklad neaproximovatelného problému

Obchodní cestující bez trojúhelníkové nerovnosti

Věta: Nechť $R \geq 1$ je libovolná konstanta. Potom pokud $P \neq NP$, tak neexistuje polynomiální aproximační algoritmus řešící obecný případ obchodního cestujícího s poměrovou chybou nejvýše R.

Důsledek (o existenci neaproximovatelných úloh): Existují NP-těžké optimalizační úlohy, pro které neexistují polynomiální aproximační algoritmy s konstantní poměrovou chybou (pokud $P \neq NP$).

Opačný případ: Existují NP-těžké optimalizační úkoly, které lze aproximovat s libovolně malou relativní chybou (poměrovou chybou libovolně blízko 1) s tím, že čím menší je požadovaná chyba tím vyšší je typicky časová složitost aproximačního algoritmu. Algoritmy mající na vstupu zadání optimalizační úlohy a požadovanou relativní chybu a vracející na výstupu řešení dané úlohy s požadovanou relativní chybou se nazývají aproximační schémata (AS, PAS, ÚPAS)

Poznámka: NP-těžké optimalizační úlohy jsou „všechny stejné“ (vzájemně na sebe převoditelné) z hlediska obtížnosti získání optimálního řešení, ale mohou se zcela diametrálně lišit z hlediska obtížnosti aproximace (získání přibližného řešení).

Vztah determinismu a nedeterminismu pro prostorovou složitost

Definice (vágní): Třída PSPACE je třída rozhodovacích problémů, pro které existuje (deterministický sekvenční) algoritmus běžící v polynomiálním prostoru (vzhledem k velikosti zadání), který správně rozhodne ANO/NE (který řeší daný problém).

Definice (vágní): Třída NPSPACE je třída rozhodovacích problémů, pro které existuje nedeterministický sekvenční algoritmus běžící v polynomiálním prostoru (vzhledem k velikosti zadání), který řeší daný problém.

Věta (Savičova): Nechť $S : N \rightarrow N$ je polynomiální funkce funkce taková, že $\forall n$ platí $S(n) \geq n$. Potom lze každý nedeterministický výpočet probíhající v prostoru $S(n)$ odsimulovat deterministickým sekvenčním výpočtem v prostoru $O(S^2(n))$.

Důsledek: PSPACE = NPSPACE