Úvod do informatiky

přednáška jedenáctá

Miroslav Kolařík

Zpracováno dle

P. Martinek: Základy teoretické informatiky,
http://phoenix.inf.upol.cz/esf/ucebni/zti.pdf

Obsah

Obsah

Složitost každého algoritmu může být studována buď z hlediska **paměťové náročnosti** nebo z hlediska **časové náročnosti**. Paměťovou náročností rozumíme požadavek na velikost paměti počítače, jež je zapotřebí k provedení výpočtu. Podobně časovou náročností rozumíme čas potřebný pro výpočet. Tento čas se obvykle neměří v časových jednotkách, ale počtem provedených elementárních kroků algoritmu.

Poznámka: Dále se budeme věnovat zejména časové složitosti.

Poznámka: Složitost je funkce závislá na velikosti vstupních dat algoritmu. U funkcí popisujících časovou složitost budeme uvažovat pouze jejich **řádovou velikost**, tedy například složitosti lišící se konstantním násobkem budeme považovat za stejné.

Definice – řádové porovnávání funkcí

Nechť f, g jsou dvě funkce, které přiřazují nezáporným celým číslům reálná čísla. Pak řekneme, že funkce f **roste řádově nejvýše** jako funkce g, píšeme $f(n) = O(g(n))$, právě když existují čísla $K > 0$ a $n_0 \in \mathbb{N}$ taková, že pro každé přirozené číslo $n \geq n_0$ platí $f(n) \leq K \cdot g(n)$.

Příklad

$3n^3 - n^2 + 2n = O(n^3)$, neboť $3n^3 - n^2 + 2n \leq Kn^3 \Leftrightarrow$
$n^3(K-3) + n^2 - 2n \geq 0$, což pro $K = 4$ dává $n^3 + n^2 - 2n \geq 0 \Leftrightarrow$
$n(n+2)(n-1) \geq 0$, což platí $\forall n \geq n_0 = 1$.

Definice

Řekneme, že algoritmus má **polynomickou časovou složitost**, právě když existuje polynom p takový, že $f(n) \leq p(n)$ pro všechna $n \in \mathbb{N}_0$.

Poznámka: Základním kritériem pro určování časové složitosti výpočetních problémů je jejich algoritmická zvládnutelnost. Je třeba si uvědomit, že existují **algoritmicky neřešitelné** problémy, pro které nemá smysl zkoušet algoritmy konstruovat. Příkladem je problém sestrojení algoritmu, který by o každém algoritmu uměl rozhodnout, zda jeho činnost skončí po konečném počtu kroků či nikoliv.

Poznámka: V praxi používané algoritmy mívají většinou některou z následujících složitostí: $O(\log N)$, $O(N)$, $O(N \log N)$, $O(N^2)$, $O(N^2 \log N)$, $O(N^3)$, $\ldots$, $O(2^N)$. Přitom stupeň polynomu bývá poměrně nízký.

Algoritmům se složitostí $O(N)$ říkáme lineární, se složitostí $O(N^2)$ kvadratické, se složitostí $O(N^3)$ kubické. Všechny algoritmy, jejichž funkci časové složitosti můžeme shora omezit polynomem v N, označujeme jako algoritmy **polynomiální**.

Příklad

Některé typické příklady časové složitosti (od nejrychlejší po nejpomalejší)

- $O(1)$ – **konstantní** (indexování prvků v poli)
- $O(\log_2 N)$ – **logaritmická** (vyhledání prvku v seřazeném poli metodou půlení intervalu)
- $O(N)$ – **lineární** (vyhledání prvku v neseřazeném poli lineárním vyhledáváním)
- $O(N \log N)$ – **lineárnělogaritmická** (seřazení pole N čísel dle velikosti; třídění sléváním či třídění haldou či quicksort)
- $O(N^2)$ – **kvadratická** (třídění N čísel dle velikosti; přímý výběr či bublinkové třídění)
- ...
- $O(2^N)$ – **exponenciální** (Fibonacciho posloupnost řešená pomocí stromové rekurze)

Definice

Nechť $f, g : \mathbb{N} \to \mathbb{R}$ jsou funkce. Pak píšeme

- $f(n) = O(g(n))$, právě když $\exists K > 0$, $\exists n_0 \in \mathbb{N}$, $\forall n \geq n_0 : f(n) \leq K \cdot g(n)$
  (Funkce f roste řádově nejvýše jako funkce g.)
- $f(n) = \Omega(g(n))$, právě když $\exists k > 0$, $\exists n_0 \in \mathbb{N}$, $\forall n \geq n_0 : f(n) \geq k \cdot g(n)$
  (Funkce f roste řádově aspoň jako funkce g.)
- $f(n) = \Theta(g(n))$, právě když $f(n) = O(g(n))$ a $f(n) = \Omega(g(n))$.
  (Funkce f roste řádově stejně jako funkce g; f a g jsou řádově ekvivalentní neboli asymptoticky ekvivalentní.)

Příklad

Dokažte, že $\frac{n^2-1}{n+1} = \Theta(n)$.

Řešení:

$$k \cdot n \leq \frac{n^2-1}{n+1} \leq K \cdot n$$

$$k \cdot n \leq n-1 \leq K \cdot n$$

$$k \leq 1 - \frac{1}{n} \leq K$$

Je to splněno např. pro $k = \frac{1}{2}$, $K = 1$ a pro $\forall n \geq n_0 = 2$.

Příklad

Dokažte, že platí

a) $5n^3 - 3n^2 + 7 = O(n^3)$,

b) $\ln n = O(n)$.

Řešení: Jednoduché.

(Srovnejte s knihou: L. Kučera – Kombinatorické algoritmy, SNTL, Praha 1983.)

Uvažme počítač, u nějž provedení 1 instrukce trvá 1 nanosekundu. Následující tabulka ukazuje délky trvání výpočtu, spustíme-li na takovém počítači algoritmus o řádové složitosti $f(n)$ se vstupními daty velikosti n.

$f(n)$	$n=20$	$n=40$	$n=60$	$n=80$	$n=100$	$n=1000$
n	$20ns$	$40ns$	$60ns$	$80ns$	$0,1\mu s$	$1\mu s$
$n\log n$	$86ns$	$0,2\mu s$	$0,35\mu s$	$0,5\mu s$	$0,7\mu s$	$10\mu s$
n^2	$0,4\mu s$	$1,6\mu s$	$3,6\mu s$	$6,4\mu s$	$10\mu s$	$1ms$
n^4	$0,16ms$	$2,56ms$	$13ms$	$41ms$	$0,1s$	$16,8min$
2^n	$1ms$	$16,8min$	$36,6let$			
$n!$	$77let$					

Předchozí tabulka potvrzuje oprávněnost představy: prakticky použitelný algoritmus = algoritmus s nejvýše polynomickou časovou složitostí.

Nelze to však brát jako dogma. Viz
$f_1(n) = 2^{100} \cdot n$
$f_2(n) = 2^{n^{0,0001}}$ ($= 2^{10}$ pro $n = 10^{10^4}$).

Předchozí představu rámcově potvrzuje i další tabulka, která popisuje, jak se zvětší rozsah zpracovatelných úloh v případě zvětšení výpočetní rychlosti použitého počítače 100x a 1000x, jestliže původně bylo možno v daném časovém limitu zpracovat vstupní data o velikosti $n = 100$.

$f(n)$	zrych. výp. $1x$	zrych. výp. $100x$	zrych. výp. $1000x$
n	100	10000	100000
$n \log n$	100	5362	43150
n^2	100	1000	3162
n^4	100	316	562
2^n	100	106	109
$n!$	100	100	101

Z tabulek je vidět, že už pro exponenciální algoritmy je typická existence mezní velikosti vstupních dat, nad níž je úloha prakticky neřešitelná i při zvýšení rychlosti počítače o několik řádů.

Obsah

Definice

Úlohu (algoritmus) nazveme **řešitelnou v polynomiálním čase**, jestliže její časovou složitost můžeme shora ohraničit polynomem. Třídu úloh s polynomiální složitostí označíme P.

Například třídění na haldě je úloha třídy P, neboť má složitost $n \log n \leq n^2$.

Úlohy třídy P považujeme za řešitelné (v přiměřeném čase).

Definice

Úlohu nazveme **nedeterministicky polynomiální**, jestliže existuje nedeterministický algoritmus, který ji řeší v polynomiálním čase. Tuto třídu úloh označíme NP.

Úloha batohu

Příklad

Máme množinu různých přirozených čísel $a_1, \ldots, a_n$ a máme zadáno přirozené číslo A. Úkolem je vybrat z čísel $a_1, \ldots, a_n$ podmnožinu jejíž součet bude A. Tedy $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ je řešením úlohy, jestliže $a_{i_1} + a_{i_2} + \cdots + a_{i_k} = A$.

Algoritmus:

1. Položíme $X = 0$.
2. Pro $i = 1, 2, \ldots, n$ buď číslo a_i přičteme k X nebo číslo a_i nepřičteme k X.
3. Jestliže $A = X$, máme řešení.

Poznámka: Při řešení úloh třídy NP je jediný známý způsob řešení postupně projít všechny možnosti připadající v úvahu, což u úlohy batohu je 2^n možností.

Poznámka: Vzhledem k časovým nárokům považujeme NP-úplné problémy za neřešitelné pro $n \geq 60$.

Skutečnost, že r je řešením úlohy U označíme $r \in U$.

Definice

Říkáme, že **úloha U_1 je redukovatelná na úlohu U_2**, jestliže existuje deterministický polynomiální algoritmus M, který řešení úlohy r převádí na výsledek $M(r)$ tak, že $r \in U_1$ právě když $M(r) \in U_2$. Označujeme $U_1 \lhd U_2$.

Definice

Úlohu U nazveme **NP-úplnou**, jestliže na ni lze redukovat libovolnou úlohu z třídy NP, tj. pro každou úlohu $U' \in NP$ platí $U' \lhd U$. Třídu NP-úplných úloh označujeme NPC.

Poznámka: NP-úplné problémy patří mezi ty nejsložitější problémy z třídy NP.

Cookova věta

Problém splnitelnosti booleovských formulí je NP-úplný.

Poznámka: NP-úplnost zde byla dokázána jako první.

P–NP problém

Je $P = NP$ nebo je $P \neq NP$?

Poznámka: Nevěříme v platnost $P = NP$ (důkaz však stále chybí). Očekáváme, že $P \neq NP$, tj. $P \subset NP$.

Poznámka: Ten kdo první správně vyřeší P-NP problém dostane (kromě doživotního věhlasu) milión amerických dolarů. (Vyhlašovatelem soutěže je Clay Mathematics Institut, více informací lze nalézt na jeho webových stránkách.)

Některé NP-úplné úlohy

1. **Úloha batohu**.
2. **Chromatické číslo grafu**. Určení nejmenšího počtu barev, jimiž lze obarvit vrcholy grafu G tak, že žádné dva sousední vrcholy nemají stejnou barvu; značíme $\chi(G)$.
3. **Nalezení Hamiltonovy kružnice**. Hamiltonova kružnice je kružnice, která obsahuje všechny vrcholy grafu.
4. **Problém kliky**. Klikou v neorientovaném grafu rozumíme takovou podmnožinu vrcholů, jejíž každé dva vrcholy jsou navzájem propojeny hranami. Problémem kliky rozumíme problém stanovení, zda má daný neorientovaný graf kliku o k vrcholech.
5. Hledání nejkratší (resp. nejdelší) cesty v orientovaném ohodnoceném grafu z vrcholu x do vrcholu y.
   (Též odpověď na otázku zda existuje cesta délky $|m|$.)
6. Úloha rozhodnout zda dva konečné automaty rozpoznávají stejný jazyk.

Dalším NP-úplným problémem je tzv. **problém obchodního cestujícího:**
Může obchodní cestující projet všechna města tak, aby každé navštívil právě jednou, na závěr se vrátil do výchozího města a přitom urazil vzdálenost menší než K?
($=$ Existuje v (úplném) neorientovaném grafu s ohodnocenými hranami hamiltonovská kružnice, v níž je součet ohodnocení jejích hran menší než předem daná hodnota K?)

Poznámka: NP-úplných problémů je více než 2000. Víme, že všechny NP-úplné problémy jsou mezi sebou "převoditelné" (tam i zpět) v polynomiálním čase, neboli jsou na sebe redukovatelné.

Poznámka: Typickým představitelem třídy NP je problém, který je řešen algoritmem (s exponenciální časovou složitostí) probírajícím všechny varianty, kde ověření správnosti každé z těchto variant vyžaduje pouze polynomický čas.

Poznámka: K žádnému NP-úplnému problému není znám algoritmus řešící jej v polynomickém čase. Pokud by někdo dokázal příslušnost některého NP-úplného problému k třídě P, dokázal by rovnost P=NP.

Poznámka: V praxi se NP-úplné úlohy (s většími vstupy) obvykle řeší pouze přibližně (heuristickými algoritmy, genetickými algoritmy, ...). Tím se (za cenu vzdání se nároků na nalezení přesného řešení) dosahuje prakticky použitelných časů.

Poznámka: Obtížně řešitelné problémy mají využití například v oblasti šifrování.

Využití výpočetně obtížných úloh v šifrování

Při utajené komunikaci požadujeme "nemožnost" nebo alespoň enormní časovou náročnost odhalení klíče potřebného k dešifrování zašifrované zprávy. Jestliže systém tvorby klíče propojíme s některými ze známých obtížně řešitelných problémů, pak úkol dešifrovat odeslanou zprávu bez znalosti příslušného klíče odpovídá úloze nalézt konkrétní řešení souvisejícího problému. Víme-li, že v současnosti nikdo takový problém efektivně řešit neumí a uvedený problém je výpočetně náročný, pak jsme pro utajení zprávy udělali maximum.

Za typický příklad využití výpočetně obtížné úlohy v šifrování lze považovat metodu RSA používanou při šifrování s veřejným klíčem.

Metoda RSA je založena na využití součinu velkých prvočísel. Poznamenejme, že testování prvočíselnosti zadaného čísla je úloha zvládnutelná v rozumném čase, naproti tomu úloha nalézt rozklad zadaného přirozeného čísla na součin prvočinitelů je výpočetně velmi náročná (a to nebyla dokázána ani její příslušnost k NP-úplným úlohám). Úkol rozšifrovat zasílanou zprávu tak odpovídá úkolu nalézt přijatelně rychlý algoritmus pro rozklad přirozeného čísla na součin prvočinitelů.

Idea šifrování s veřejně přístupným klíčem

Jeden účastník utajené komunikace (např. ústředí banky) vygeneruje dvojici klíčů A_1, A_2, mezi nimiž je určitá matematická závislost. Klíč A_1 dostačuje k zašifrování posílaných zpráv, klíč A_2 je potřebný k rozšifrování přijatých zpráv. Klíč A_2 utají, A_1 naproti tomu rozešle ostatním účastníkům utajené komunikace (např. bankovním pobočkám) prostřednictvím veřejně přístupného média (např. telefonní linkou). S pomocí klíče A_1 tedy může každý zašifrovat svou zprávu, ale její rozšifrování lze provést pouze s využitím klíče A_2, jenž je známý výhradně iniciátoru celé komunikace.