

Kapitola 1

Státnice - Metody tvorby algoritmů

1.1 Popis složitosti algoritmů

Definice (Velikost dat, krok algoritmu)

Velikost dat je obvykle počet bitů, potřebných k jejich zapsání, např. data D jako pro čísla a_1, \dots, a_n je velikost dat $|D| = \sum_{i=1}^n \lceil \log a_i \rceil$.

Krok algoritmu je jedna operace daného abstraktního stroje (např. Turingův stroj, stroj RAM), zjednodušeně jde o nějakou operaci, proveditelnou v konstantním čase (např. aritmetické operace, porovnání hodnot, přiřazení – pro jednoduché číselné typy).

Definice (Časová složitost)

Časová složitost je funkce $f : \mathbb{N} \rightarrow \mathbb{N}$ taková, že $f(|D|)$ udává počet kroků daného algoritmu, pokud je spuštěn na datech D .

Definice (Asymptotická složitost)

Řekneme, že funkce $f(n)$ je asymptoticky menší nebo rovna než $g(n)$, značíme $f(n)$ je $O(g(n))$, právě tehdy, když

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

Funkce $f(n)$ je asymptoticky větší nebo rovna než $g(n)$, značíme $f(n)$ je $\Omega(g(n))$, právě tehdy, když

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

Funkce $f(n)$ je asymptoticky stejná jako $g(n)$, značíme $f(n)$ je $\Theta(g(n))$, právě tehdy, když

$$\exists c_1, c_2 > 0 \exists n_0 \forall n > n_0 : 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

Funkce $f(n)$ je asymptoticky ostře menší než $g(n)$ ($f(n)$ je $o(g(n))$), když

$$\forall c > 0 \exists n_0 \forall n > n_0 : 0 \leq f(n) < c \cdot g(n)$$

Funkce $f(n)$ je asymptoticky ostře větší než $g(n)$ ($f(n)$ je $w(g(n))$), když

$$\forall c > 0 \exists n_0 \forall n > n_0 : 0 \leq c \cdot g(n) < f(n)$$

Poznámka

Asymptotická složitost zkoumá chování algoritmů na velkých datech, zařazuje je podle toho do kategorií. Zanedbává multiplikační a aditivní konstanty.

1.2 Rozděl a panuj

Definice (Metoda rozděl a panuj)

Rozděl a panuj je metoda návrhu algoritmů (ne strukturované programování), která má 3 kroky:

1. rozděl – rozdělí úlohu na několik podúloh stejného typu, ale menší velikosti
2. vyřeš – vyřeší podúlohy a to buď přímo pro dostatečně malé, nebo rekurzivně pro větší
3. sjednot – sjednotí řešení podúloh do řešení původní úlohy

Aplikace

- QUICKSORT
- Hledání mediánu

Analýza složitosti algoritmů rozděl a panuj**Poznámka (Vytvoření rekurentní rovnice)**

Pro časovou složitost algoritmů typu rozděl a panuj zpravidla dostávám nějakou rekurentní rovnici.

- $T(n)$ budiž doba zpracování úlohy velikosti n , za předpokladu, že $T(n) = \Theta(1)$ pro $n \leq n_0$.
- $D(n)$ budiž doba na rozdělení úlohy velikosti n na a podúloh stejné velikosti $\frac{n}{c}$.
- $S(n)$ budiž doba na sjednocení řešení podúloh velikosti $\frac{n}{c}$ na jednu úlohu velikosti n . Dostávám rovnici

$$T(n) = \begin{cases} D(n) + aT(\frac{n}{c}) + S(n) & n > n_0 \\ \Theta(1) & n \leq n_0 \end{cases}$$

Poznámka

Při řešení rekurentních rovnic:

- Zanedbávám celočíselnost ($\frac{n}{2}$ místo $\lceil \frac{n}{2} \rceil$ a $\lfloor \frac{n}{2} \rfloor$)
- Nehledím na konkrétní hodnoty aditivních a multiplikativních konstant, asymptotické notace používám i v zadání rekurentních rovnic, i v jejich řešení.

Věta (Substituční metoda)

1. Uhodnu asymptoticky správné řešení
2. Indukcí ověřím správnost (zvláště horní a dolní odhad)

Věta (Metoda “kuchařka” (Master Theorem))

Nechť $a \geq 1, c > 1, d \geq 0 \in \mathbb{R}$ a nechť $T : \mathbb{N} \rightarrow \mathbb{N}$ je neklesající funkce taková, že $\forall n$ tvaru c^k platí

$$T(n) = aT(\frac{n}{c}) + \Theta(n^d)$$

Potom

1. Je-li $\log_c a \neq d$, pak $T(n)$ je $\Theta(n^{\max\{\log_c a, d\}})$
2. Je-li $\log_c a = d$, pak $T(n)$ je $\Theta(n^d \log_c n)$

Věta (Master Theorem, varianta 2)

Nechť $0 < a_i < 1$, kde $i \in \{1, \dots, k\}$ a $d \geq 0$ jsou reálná čísla a nechť $T : \mathbb{N} \rightarrow \mathbb{N}$ splňuje rekurenci

$$T(n) = \sum_{i=1}^k T(a_i \cdot n) + \Theta(n^d)$$

Nechť je číslo x řešením rovnice $\sum_{i=1}^k a_i^x = 1$. Potom

1. Je-li $x \neq d$ (tedy $\sum_{i=1}^k a_i^d \neq 1$), pak $T(n)$ je $\Theta(n^{\max\{x, d\}})$
2. Je-li $x = d$ (tedy $\sum_{i=1}^k a_i^d = 1$), pak $T(n)$ je $\Theta(n^d \log n)$

1.3 Dynamické programování

Dynamické programování je metoda řešení problémů, které v sobě obsahují překrývající se subproblémy.

Příklad

Typickým příkladem je výpočet fibonaciho čísla. Fib. posloupnost je definována jako:

$$f(0) = 1, f(1) = 1$$

$$f(n+2) = f(n+1) + f(n)$$

Výpočet třeba 4. čísla by pak byl $f(4) = f(3) + f(2) = f(2) + f(1) + f(1) + f(0) = f(1) + f(0) + f(1) + f(1) + f(0) = 5$. Vidíme, že jsme $f(2)$ počítali dvakrát, $f(1)$ třikrát a $f(0)$ dvakrát. Ve větším měřítku toto zbytečně počítání vede k exponenciální složitosti algoritmu. Lepší cestou je počítat “od spodu”, kdy pomoci $f(0)$ a $f(1)$ spočteme $f(2)$, pak s jeho pomoci $f(3)$ nakonec $f(4)$ s lineární složitostí.

V dynamickém programování se například vytvoří mapa Fibonaciho čísel a jejich hodnot. Před tím, než začnu počítat hodnotu nějakého Fibonaciho čísla, se podívám do mapy.

Nutné podmínky

V dynamickém programování využíváme:

1. Překrývání podproblému — problém lze rozdělit na podproblémy, jejichž řešení se využívá opakovaně.
2. Optimální podstruktury — optimální řešení lze zkonstruovat z optimálních řešení podproblémů.

Postupy

Obvykle se používá jeden ze dvou přístupů k dynamickému programování:

- Top-down — problém se rekurzivně dělí na podproblémy a po jejich vyřešení se zapamatují výsledky, které se použijí při případném opětovném řešení daného podproblému.
- Bottom-up — nejdříve se spočítají všechny možné podproblémy (viz příklad s Fibonaciho posloupností), které se potom skládají do řešení větších problémů. Tento přístup je výhodnější z hlediska počtu volání funkci a místa na zásobníků, ale ne vždy musí být zřejmé, které všechny subproblémy je třeba předem spočítat.

Použití

Problém batohu – máme věci různé váhy a batoh určité nosnosti. Které věci máme dát do batohu, abychom jej co nejlépe zaplnili (jednorozměrná varianta problému batohu)? Speciální případ je součet podmnožiny (SP). Algoritmus je popsán v sekci o NP-úplnosti.

Uzávorkování součinu matic tak, aby počet skalárních součinů byl co nejmenší. Dělá se pomocí 2 čtvercových matic (M a K) řádu rovného počtu násobených matic, kde pracujeme jen nad diagonálou. Hodnota na M_{ij} udává minimální počet skalárních součinů při nejlepší uzávorkování matic i až j , hodnota K_{ij} určuje matici, která rozděluje závorkování na dvě podmnožiny matic. Hodnotu M_{ij} lze zkonstruovat ze všech M_{ik} a M_{kj} pro $i < k < j$.

1.4 Hladové algoritmy

Motivace

Problém 1

Dán souvislý neorientovaný graf $G = (V, E)$ a funkce $d : E \rightarrow \mathbb{R}^+$, udávající délky hran. Najděte minimální kostru grafu G , tj. kostru $G' = (V, E')$ tak, že $\sum_{e \in E'} d(e)$ je minimální.

Problém 2

Je dána množina $S = \{1, \dots, n\}$ úkolů jednotkové délky. Ke každému úkolu je dána lhůta dokončení $d_i \in \mathbb{N}$ a pokud $w_i \in \mathbb{N}$, kterou je úkol i penalizován, není-li hotov do své lhůty. Najděte rozvrh (permutaci úkolů od času 0 do času n), který minimalizuje celkovou pokutu.

Problém 3

Je dána množina $S = \{1, \dots, n\}$ úkolů. Ke každému úkolu je dán čas jeho zahájení s_i ukončení f_i , $s_i \leq f_i$. Úkoly i a j jsou kompatibilní, pokud se intervaly $[s_i, f_i)$ a $[s_j, f_j)$ nepřekrývají. Najděte co největší množinu (po dvou) kompatibilních úkolů.

Matroid

Definice (Matroid)

Matroid je uspořádaná dvojice $M = (S, I)$, splňující:

- S je konečná neprázdná množina (prvky matroidu M)
- I je neprázdná množina podmnožin S (nezávislé podmnožiny), která má
 1. dědičnou vlastnost: $B \in I \wedge A \subseteq B \Rightarrow A \in I$,
 2. výměnnou vlastnost: $A, B \in I \wedge |A| < |B| \Rightarrow \exists x \in B \setminus A : A \cup \{x\} \in I$.

Věta (O velikosti maximálních nezávislých podmnožin)

Všechny maximální (maximální vzhledem k inkluzi) nezávislé podmnožiny v matroidu mají stejnou velikost.

Důkaz

Z výměnné vlastnosti, nechť $A, B \in I$ maximální, $|A| < |B|$, pak $A \cup \{x\} \in I, x \notin A$, což je spor.

Definice (Vážený matroid)

Matroid $M = (S, I)$ je vážený, pokud je dána funkce $w : S \rightarrow \mathbb{R}^+$ a její rozšíření na podmnožiny množiny S je definováno předpisem:

$$A \in S \Rightarrow w(A) = \sum_{x \in A} w(x)$$

Problém 1 a 2 – zobecněný

Pro daný vážený matroid nalezněte nezávislou podmnožinu s co největší vahou (optimální množinu). Protože váhy jsou kladné, vždy se bude jednat o maximální nez. množinu.

Problém 1 je spec. případ tohoto, protože můžeme uvažovat grafový matroid (nad hranami) – $M_G = (S, I)$, kde $S = E$ a pro $A \subseteq E$ platí $A \in I$, pokud hrany z A netvoří cyklus (tj. indukovaný podgraf tvoří les).

Dědičná vlastnost této struktury je zřejmá, výměnnost se dá dokázat následovně: mějme $A, B \subseteq E, |A| < |B|$. Pak lesy z A, B mají $n - a > n - b$ stromů (vč. izolovaných vrcholů). V B musí být strom, který se dotýká ≥ 2 různých stromů z A . Ten obsahuje hranu, která není v žádném stromě A a netvoří cyklus a tak ji můžeme k A přidat.

Váhovou funkci převedu na hledání maxim: $w(e) = c - d(e)$, kde c je dost velká konstanta, aby všechny váhy byly kladné. Algoritmus pak najde max. množinu, kde hrany netvoří cyklus. Implicitně bude mít $n - 1$ hran, takže půjde o kostru a její w bude maximální, tedy původní váha minimální.

Pro problém 2 zavedeme pojem kanonického rozvrhu – takového rozvrhu, kde jsou všechny včasné úkoly rozvrženy před všemi zpožděnými a uspořádány podle neklesající lhůty dokončení (tohle na celkové pokutě nic nezmění a máme bijekci mezi množinami včasných úkolů a kanonickými rozvrhy).

Optimální množinu pak lze hledat jen nad kanonickými rozvrhy – nezávislou množinou úkolů nazvu takovou, pro kterou existuje kanonický rozvrh tak, že žádný úkol v ní obsažený není zpožděný. Potom hledání max. nezávislé množiny při ohodnocení pokutami je hledání nejmenší pokuty (odeberu co nejvíc z možné celkové pokuty).

Pak zbývá dokázat, že takto vytvořená struktura je matroid. Dědičná vlastnost je triviální – vyhodím-li něco z nezávislé množiny, nechám v rozvrhu mezery a bude platit stále. Pro výměnnou vlastnost zavedu pomocnou funkci $N_t(C) = |\{i \in C | d_i \leq t\}|$, udávající počet úkolů z množiny C se lhůtou do času t . Pak množina C je nezávislá, právě když $\forall t \in \{1, \dots, n\} : N_t(C) \leq t$.

Pak máme-li 2 nezávislé $A, B, |B| > |A|$, označíme k největší okamžik takový, že $N_k(B) \leq N_k(A)$, tj. od $k + 1$ dál platí $N_t(A) < N_t(B)$. To skutečně nastane, protože $|B| = N_n(B) > N_n(A) = |A|$. Pak určitě v B je ostře víc úkolů s $d_i = k + 1$ než v A , tj. $\exists x \in B \setminus A$ se lhůtou $k + 1$. $A \cup \{x\}$ je nezávislá, protože $N_t(A \cup \{x\}) = N_t(A)$ pro $t \leq k$ a $N_t(A \cup \{x\}) \leq N_t(B)$ pro $t \geq k + 1$.

Hladový algoritmus na váženém matroidu

Algoritmus (Greedy)

Mám zadaný matroid $M = (S, I)$, kde $S = \{x_1, \dots, x_n\}$ a $w : S \rightarrow \mathbb{R}^+$. Potom

1. $A := \emptyset$, seříd' a přeznač S sestupně podle vah
2. pro každé x_i zkoušej: je-li $A \cup \{x_i\} \in I$, tak $A := A \cup \{x_i\}$
3. vrať A jako maximální nezávislou množinu

Pokud přidám nějaké x_i , nikdy nezruším nezávislost množiny; s už přidanými prvky se nic nestane. Časová složitost je $\Theta(n \log n)$ na setřídění podle vah, testování nezávislosti množiny závisí na typu matroidu ($f(n)$), takže celkem něco jako $\Theta(n \log n + n \cdot f(n))$.

Důkaz

- Vyhození prvků, které samy o sobě nejsou nezávislé množiny, nic nezkaží (z dědičné vlastnosti).
- První vybrané x_i je “legální” (nezablokuje mi cestu), protože mezi max. nez. množinami určitě existuje taková, která jím mohla vzniknout (z výměnné vlastnosti – vezmeme první prvek, který je sám o sobě nez. mn. a s pomocí nějaké max. nez. množiny B ho doplníme, vzniklé $\{x_i\} \cup (B \setminus \{x_j\})$ musí být také maximální).
- Nalezení optimální množiny obsahující pevné (první vybrané) x_i v $M = (S, I)$ je ekvivalentní nalezení optimální množiny v $M' = (S', I')$, kde $S' = \{y \in S \mid \{x_i, y\} \in I\}$ a $I' = \{B \subset S' \setminus \{x_i\} \mid B \cup \{x_i\} \in I\}$ (je-li S' neprázdná, je to matroid, vlastnosti se přenesou z původního; pokud je S' prázdná, tak algoritmus končí) a tedy když vyberu x_i , nezatarasím si cestu k optimu – stačí ukázat, že $A \cup \{x\}$ je optimální v M právě když A je optimální v M' .
- Takže když budu vybírat (indukcí opakovaně) x_i podle váhy, dojdou k optimální množině.

Algoritmus (Hladový algoritmus na problém 3)

Máme $S = \{1, \dots, n\}$ množinu úkolů s časy startů s_i a konců f_i . Provedeme:

1. $A := \emptyset, f_0 := 0, j := 0$
2. setříd' úkoly podle f_i vzestupně a přeznač
3. pro každé i zkoušej: je-li $s_i \geq f_j$, pak $A := A \cup \{i\}$ a $j := i$
4. vrať A jako max. množinu nekryjících se úkolů

Složitost je $n \log n$ na setřídění, zbytek už lineární. Sice to funguje, ale tahle struktura NENÍ matroid – nesplňuje výměnnou vlastnost. Algoritmus má ale stejný důkaz jako předchozí na matroidech (legálnost hladového výběru – existuje max. množina obsahující prvek 1 – a existenci optimální množiny – převod na “menší” zadání).