C2142 Návrh algoritmů pro přírodovědce

11. Přístupy k řešení problémů II.

Tomáš Raček

Jaro 2021

Sudoku

Úkol. Vyřešte následující zadání Sudoku.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Zamyšlení. Řešení je poměrně snadné pro člověka, ale jak jej algoritmizovat?

Backtracking

Backtracking je rekurzivní přístup, v rámci něhož hledám řešení následujícím způsobem:

1. Zkontroluji, zdali jsem nalezl řešení.
2. Pokud ne, zkusím pokračovat v hledání některou z možností, kterou v danou chvíli mám a ještě jsem nevyzkoušel.
3. Pokud žádné možnosti nezbývají, vracím se do posledního místa, kde jsem měl ještě na výběr.

Vlastnosti:

- garance nalezení nejlepšího/všech řešení
- potenciálně vysoká složitost

Známé příklady:

- DFS
- minimální počet mincí

Sudoku – jednoduché řešení

1. Pokud jsou obsazena všechna pole, vracím TRUE.
2. Najdu první prázdné pole.
3. Pro všechny přípustné číslice pro toto pole:
   3.1 Zapíšu zvolenou číslici.
   3.2 Celý algoritmus rekurzivně opakuji.
   3.3 Pokud je výsledkem rekurzivního volání TRUE, vracím jej, v opačném případě zkouším další číslici.
4. Všechny možnosti pro dané pole jsou neúspěšně vyčerpány, vracím FALSE.

Zamyšlení. Uvedený postup lze vylepšit, pokud budou volná pole vybírána v pořadí podle počtu možných číslic (od nejmenšího).

Branch and bound

Branch and bound je jednoduché vylepšení backtrackingu pro optimalizační problémy, kdy si během výpočtu pamatuju aktuálně nejlepší nalezené řešení (resp. jeho cenu).

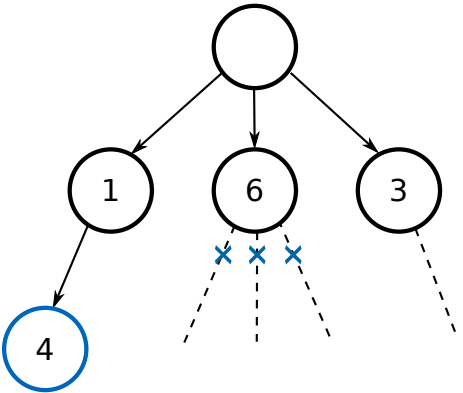
- eliminuji cesty, které už nemohou vést k lepšímu řešení (= jejich ohodnocení je větší než ohodnocení aktuálně nejlepšího nalezeného řešení)

Vlastnosti:

- vede k omezení větvení → „prořezávání větví“
- nezaručuje, že se vyhneme exponenciální složitosti
- užitečné, pokud brzy najdeme dobré řešení

Branch and bound – příklad

Ukázka. Po nalezení řešení s ohodnocením 4 neprohledávám dále neperspektivní cesty.



Dynamické programování

Dynamické programování je metoda podobná rozděj a panuj použitelná pro optimalizační úlohy.

Princip

1. Rozděl problém na menší podproblémy.
   - stejného typu
   - musí se překrývat (optimální řešení problému v sobě zahrnuje optimální řešení podproblému)
2. Vyřeš jednotlivé podproblémy v pořadí od nejmenších.
3. Zkombinuj řešení podproblémů na řešení původního problému.

Příklady

- Dijkstrův algoritmus
- Floyd-Warshallův algoritmus

Minimální počet mincí

Opakování. Pro správně zvolené hodnoty mincí je hladový přístup optimální.
V některých případech však nenalezne (nejlepší) řešení.

Optimální řešení lze nalézt pomocí dynamického programování.

- označme $C[j]$ minimální počet mincí na zaplacení částky j
- pokud známe optimální řešení pro $C[j]$ a použili jsme minci hodnoty h_i, pak máme:

$$C[j] = 1 + C[j - h_i]$$

Příklad. Pokud je $C[46]$ optimální a použili jsme minci hodnoty 20, pak $C[46] = 1 + C[26]$.

Minimální počet mincí

Zobecnění. Mějme k různých mincí hodnot h_i, kde $1 \leq i \leq k$. Pak optimální řešení pro částku j je dáno:

$$C[j] = \begin{cases} \infty & \text{pro } j < 0 \\ 0 & \text{pro } j = 0 \\ 1 + \min\limits_{1 \leq i \leq k} \{C[j - h_i]\} & \text{pro } j \geq 1 \end{cases}$$

Příklad pro částku 6 a hodnoty mincí 1, 3 a 4.

- postupujeme od nejnižších částek až po výslednou dle předchozího výrazu
- zjevně $C[0] = 0$

Minimální počet mincí

$$C[1] = \min \begin{cases} 1 + C[1-4] = \infty \\ 1 + C[1-3] = \infty \\ 1 + C[1-1] = 1 \end{cases}$$

$$C[2] = \min \begin{cases} 1 + C[2-4] = \infty \\ 1 + C[2-3] = \infty \\ 1 + C[2-1] = 2 \end{cases}$$

$$C[3] = \min \begin{cases} 1 + C[3-4] = \infty \\ 1 + C[3-3] = 1 \\ 1 + C[3-1] = 3 \end{cases}$$

$$C[4] = \min \begin{cases} 1 + C[4-4] = 1 \\ 1 + C[4-3] = 2 \\ 1 + C[4-1] = 2 \end{cases}$$

$$C[5] = \min \begin{cases} 1 + C[5-4] = 2 \\ 1 + C[5-3] = 3 \\ 1 + C[5-1] = 2 \end{cases}$$

$$C[6] = \min \begin{cases} 1 + C[6-4] = 3 \\ 1 + C[6-3] = 2 \\ 1 + C[6-1] = 3 \end{cases}$$

Optimální pořadí násobení matic

Problém. Chceme vynásobit $A_1 \cdots A_n$ s nejmenším počtem operací.

Pozorování

- násobení matic je asociativní, tj. $A(BC) = (AB)C$
- vhodným uzávorkováním lze snížit množství nutných operací

Příklad s maticemi $A_{10\times30}$, $B_{30\times5}$, $C_{5\times60}$:

- $(A_{10\times30} \cdot B_{30\times5}) \cdot C_{5\times60} = X_{10\times5} \cdot C_{5\times60}$
- $10 \cdot 30 \cdot 5 + 10 \cdot 5 \cdot 60 = 4\,500$ operací
- $A_{10\times30} \cdot (B_{30\times5} \cdot C_{5\times60}) = A_{10\times30} \cdot X_{30\times60}$
- $30 \cdot 5 \cdot 60 + 10 \cdot 30 \cdot 60 = 27\,000$ operací

Závěr. Složitost řešení pomocí přístupu rozděl a panuj je $O(3^n)$, užitím dynamického programování pak $O(n^3)$.

Heuristiky

Pozorování. Některé instance problémů mohou být z hlediska složitosti exaktně velmi těžko řešitelné.

Myšlenka

- suboptimální řešení lze nalézt často výrazně rychleji
- často není potřeba určit všechna řešení
- heuristika – forma odhadu jak vypadá/co obsahuje řešení (př. v 95 % případů platí...)

Příklad. Určete nejkratší vzdálenost mezi vrcholy s a t v grafu G.

- pro výpočet uvažujeme pouze hrany s ohodnocením $\leq k$
- zřejmě nemusí vést k (nejlepšímu) řešení

Redukce

Redukce je metoda převodu jednoho problému na jiný. Využívá se zejména v rámci teoretického porovnávání složitosti algoritmů.

Princip

1. Zadání problému A transformuji na zadání pro problém B.
2. Vyřeším zadání problému B.
3. Řešení problému B převedu zpátky na řešení původního problému A.

Příklad. Nejkratší vzdálenost v neohodnoceném grafu.

- lze převést na nejkratší vzdálenost v ohodnoceném grafu
- $\forall (u, v) \in E : w_e(u, v) = 1$
- nejkratší cesta je v novém i původním grafu stejná