Programovací strategie: Jak pobrat odměnu

Tomáš Maršálek

6. října 2011

1 Zadání

Svojí nebojácností a schopnostmi jste zvítězili nad zlým čarodějem a teď si můžete buď odvést princeznu anebo si vybrat některý z krásných zlatých předmětů, které kouzelník vlastní. Princezna se vám nelíbí, proto jste se rozhodl pro zlato. Můžete si vybrat spoustu velkých a krásných předmětů, jedinou podmínkou je, že vybrané objekty musíte být schopen odnést v batohu, jehož nosnost (i vaše, koneckonců) je omezena. Pokud se vám to nepovede, nezískáte nic. Předměty jsou různorodé – svícny, sošky apod., každý stojí jinak a váží jinak. Teď se vám hodí praxe v dynamickém programování z PRO.

Vstupem je množina položek $P = \{p_1, p_2, \ldots, p_n\}$, kde položka p_i má velikost s_i a hodnotu v_i, batohová kapacita (tj. velikost batohu) je C. Vašim úkolem je najít podmnožinu s maximální hodnotou zjištěnou jako součet hodnot prvků podmnožiny takovou, aby součet velikostí prvků podmnožiny nepřekročil C (tj. všechny vybrané položky se musí vejít do batohu). Velikosti položek i jejich hodnoty jsou kladná čísla do 1000.

2 Problém

Problém je známý jako KNAPSACK PROBLEM, zde přesněji DISCRETE KNAPSACK, jelikož počet předmětů, které můžeme pobrat není spojitá hodnota. V opačném případě by se jednalo o LIQUID KNAPSACK, pro který se dá použít jednoduchý greedy algoritmus. Ještě speciálnější vymezení je 0-1 KNAPSACK, protože počet předmětů, které můžeme vzít od každého druhu je právě 0 nebo 1.

3 Řešení

Algoritmus bude na principu dynamického programování. Snažíme se maximalizovat hodnotu batohu, tak abychom se vešli do limitu C.

$$knapsack(N, C) = \textit{maximální hodnota, kterou je možno narvat do batohu tak, že nepřekročíme mez } C$$

3.1 Myšlenka

Optimální substruktura bude založená na jednoduché myšlence, že N-tý předmět můžeme do batohu vložit nebo naopak ne. Ta z možností, která bude mít větší hodnotu nás bude zajímat. Batoh, který překročil hmotnostní limit bude mít hodnotu 0.

Rekurzivní vyjádření bude vypadat zhruba následovně

$$krapsack(N, C) = \max(N - tou\ položku\ přijmout, \\ N - tou\ položku\ vyhodit)$$

Formálně zapsáno

$$knapsack(N, C) = \max(knapsack(N - 1, C - hmotnost(N)) + hodnota(N), \\ knapsack(N - 1, C))$$

```
def knapsack(N, C):
        if N == 0:
                return 0
        i = N - 1                               # začínáme od nuly
        if hmotnost[i] > C:
                return knapsack(N - 1, C)
        return max(knapsack(N - 1, C - hmotnost[i]) + hodnota[i],
                   knapsack(N - 1, C))
```

3.2 DP provedení

Uvedenou rekurzi vypočítáme zdola nahoru tabulární metodou (klasické dynamické programování).

Funkce závisí na dvou parametrech N a C, proto složitost bude $\Theta(NC)$. Na první pohled by se mohlo zdát, že se jedná o polynomiální složitost, ale je třeba si uvědomit, že limit C může dosahovat exponenciálních hodnot. KNAPSACK PROBLEM je na seznamu Karpových 21 NP-úplných problémů. [2]

```
def knapsack(N, C):
        tmp = [[0 for j in range(C + 1)] for i in range(N)]  # DP tabulka
        for j in range(C + 1): # base case
                if hmotnost[0] <= j:
                        tmp[0][j] = hodnota[0]

        for i in range(1, N):
                for j in range(C + 1):
                        if hmotnost[i] > j:
                                tmp[i][j] = tmp[i - 1][j]
                        else:
                                tmp[i][j] = max(tmp[i - 1][j - hmotnost[i]] + hodnota[i],
                                                tmp[i - 1][j])
        return tmp[-1][-1]        # vrátit poslední prvek tabulky
```

Položky najdeme zpětným průchodem tabulkou

```
polozky = []
while i >= 0 and j >= 0:
        if j >= hmotnost[i] \
        and tmp[i - 1][j - hmotnost[i]] + hodnota[i] == tmp[i][j]:
                polozky.append(i)
                j -= hmotnost[i]
                i -= 1
        elif tmp[i - 1][j] == tmp[i][j]:
                i -= 1
        else:
                j -= 1
polozky = polozky[::-1] # obrátit položky
```

4 Implementace

Použitý jazyk je python 2.7.2. Implementace obsahuje generátor vstupu `gen.py` a vlastní program `knapsack.py`.

4.1 Příklad použití

```
# vygenerovat 100 polozek s maximalni hodnotou a maximalni vahou 1000
./gen.py 100 1000 1000 > test.data

# pouzit vygenerovana data pro knapsack s limitem 5000 (C = 5000)
cat test.data | ./knapsack.py 5000
```

Reference

[1] *Shai Simonson*
02-20-01: Knapsack, Bandwidth Min. Intro: Greedy Algs. (video) 2001
http://aduni.org/courses/algorithms/index.php?view=cw
http://video.google.com/videoplay?docid=-8586312179390822765

[2] *Wikipedia contributors*
"Knapsack problem," Wikipedia, The Free Encyclopedia
http://en.wikipedia.org/w/index.php?title=Knapsack_problem&oldid=452859011
(accessed October 5, 2011).