Dawid Pawliczek
Lista 2, Zadanie 4

Treść zadania

Masz początkowo do dyspozycji m monet o nominale 1 i nieskończoną liczbę monet o nominale 100. W kolejnych n dniach robisz zakupy (w i-tym dniu za kwotę c_i). Jeśli nie odliczysz dokładnie c_i w jedynkach, kasa wyda resztę minimalną liczbą monet, używając wyłącznie nominałów 1 i 100, a Twoja „atrakcyjność kliencka" zmniejszy się o $x \cdot w_i$, gdzie x to liczba wydanych monet, zaś w_i to zadany współczynnik. (Zakładamy $1 \leq c_i, w_i \leq n$). Wyznacz, o ile *co najmniej* spadnie łączna atrakcyjność po wszystkich zakupach.

Rozwiązanie dynamiczne

Niech

$$\mathrm{dp}[i][m] \;=\; \text{największa możliwa atrakcyjność po } i \text{ dniach przy stanie portfela } m.$$

Dla uproszczenia przyjmujemy $c_i < 100$, bo zawsze w pierwszej kolejności płacimy pełnymi setkami.

$$\begin{aligned}
\mathrm{dp}[i+1][m-c_i] &= \max\bigl(\mathrm{dp}[i+1][m-c_i],\, \mathrm{dp}[i][m]\bigr), && \text{jeśli } c_i \leq m; \\
\mathrm{dp}[i+1][m+100-c_i] &= \max\bigl(\mathrm{dp}[i+1][m+100-c_i],\, \mathrm{dp}[i][m] - (100-c_i)w_i\bigr), && \text{jeśli } c_i > m.
\end{aligned}$$

Liczba możliwych stanów rośnie najwyżej o 100 dziennie, więc złożoność wynosi $O(n^2)$.

Obserwacja prowadząca do rozwiązania zachłannego

W każdym dniu wybór sprowadza się do:

$$\boxed{\text{zapłacić dokładnie}} \quad \text{lub} \quad \boxed{\text{zapłacić 100 i przyjąć koszt } (100-c_i)w_i.}$$

Jeżeli w i-tym dniu zabraknie jedynek, możemy „cofnąć" którąś z wcześniejszych decyzji (zamienić płatność dokładną na setkę), ale warto to zrobić tylko tam, gdzie koszt $(100 - c_j)w_j$ jest najmniejszy.

Algorytm zachłanny

Algorithm 1 GreedyMin

```
Require: m, tablice c[1...n], w[1...n]
Ensure: łączny spadek atrakcyjności
loss ← 0,   heap ← min-kopiec
for i ← 1 to n do
    m ← m − c_i                                 ▷ próbujemy zapłacić dokładnie
    heap.PUSH((100 − c_i)w_i)
    loss ← loss + heap.POPMIN()                 ▷ najtańsza „cofka”
    m ← m + 100
end for
return loss
```

Złożoność. Każdy element wstawiamy i usuwamy z kopca co najwyżej raz, stąd czas $O(n \log n)$, pamięć $O(n)$.

Poprawność

Inwariant. Po obsłużeniu pierwszych i dni algorytm utrzymuje:

1. portfel zawiera co najmniej 0 monet 1,
2. z wszystkich dotychczas rozważonych dni do zbioru płatności 100 wybrano *najtańsze* (koszt $(100 - c_t)w_t$) tak, aby liczba jedynek nie była ujemna.

Punkt (1) jest spełniany, bo w razie deficytu algorytm natychmiast „cofa” kolejne dni, dopisując po 100 jednostek do stanu portfela, aż $m \geq 0$. Punkt (2) zachodzi, gdyż cofamy zawsze *najmniejszą* dostępną wartość z kopca.

Greedy → optymalny. Załóżmy, że istnieje rozwiązanie OPT o mniejszym łącznym koszcie od wyniku algorytmu. Prześledźmy dni w kolejności i wybierzmy pierwszy dzień, w którym zbiory „cofnięć” algorytmu i OPT różnią się. W tym momencie algorytm wybrał koszt minimalny spośród wszystkich niedotychczasowych dni, a OPT — koszt co najmniej tak duży. Zamieniając w OPT jego droższy dzień na tańszy dzień algorytmu, poprawiamy lub nie zmieniamy kosztu i **nie** psujemy stanu portfela (bo liczba cofnięć się nie zmniejsza). Wykonując taką operację każdorazowo, stopniowo przekształcamy OPT w rozwiązanie identyczne z wynikiem algorytmu, bez zwiększania ceny. Sprzeczność z założeniem, że algorytm był droższy.

Konkluzja. Zawsze, gdy brakuje jedynek, trzeba dołożyć co najmniej jedną płatność 100, a w każdym takim momencie algorytm wybiera najtańszą możliwą opcję — dlatego łączny koszt nie może być mniejszy. □