

Informatyczny kącik olimpijski (85): Jeszcze dwa zadania do plecaka

Zakładamy, że początkowo tablica $d[0..M]$ jest zainicjowana wartościami $-\infty$. Po i -tej fazie algorytmu nieujemna wartość $d[j]$ oznacza największą wartość podzbioru przedmiotów o sumarycznym rozmiarze j , wybranego spośród przedmiotów ze zbioru $\{1, \dots, i\}$:

```
d[0] := 0;
for i := 1 to n do
  for j := M downto m[i] do
    if d[j - m[i]] ≠ -∞ then
      w_* := d[j - m[i]] + w[i];
      d[j] := max(d[j], w_*);
```

Naturalne jest tu zastosowanie programowania dynamicznego. Jeśli oznaczymy przez $dp[i, j]$ minimalny czas złożenia j mebli, gdy rozważamy jedynie rodzaje od 1 do i , to odpowiedzią będzie $dp[n, M]$, a rekurencją do niej prowadzącą wzór:

$$dp[i, j] = \min_{0 \leq k \leq \min(c_i, j)} (dp[i-1, j-k] + T_{i,k}),$$

gdzie $T_{i,k} = kt_i - \binom{k}{2}d_i$ jest czasem montażu k mebli rodzaju i . To rozwiązanie działa w czasie $O(nM^2)$.

Intuicyjnie rzecz biorąc, jeśli już zaczęliśmy składać meble danego rodzaju, to opłaca się nam wykorzystać zdobyte doświadczenie do maksimum. A formalnie: istnieje optymalne rozwiązanie, w którym mamy co najwyżej jeden rodzaj mebla i , którego złożymy więcej niż 0, ale mniej niż c_i sztuk. Istotnie, założymy, że mamy co najmniej dwa takie *nietrywialne* rodzaje i_1 i i_2 , których składamy odpowiednio ℓ_1 i ℓ_2 sztuk. Jeśli ostatni mebel rodzaju i_1 składamy nie dłużej niż ostatni mebel rodzaju i_2 , to składając dodatkowe $\ell = \min(c_{i_1} - \ell_1, \ell_2)$ sztuk rodzaju i_1 zamiast ℓ sztuk rodzaju i_2 , nie pogorszymy czasu, a zmniejszymy liczbę nietrywialnych rodzajów mebli.

Rozważmy zatem wszystkie możliwości wyboru nietrywialnego mebla i . Dla ustalonego i pogrupujmy pozostałe rodzaje mebli j w paczki o rozmiarach $m[j] = c_j$ i czasach montażu $w[j] = T_{j,c_j}$ i rozwiążmy dla nich problem plecakowy, w którym pytamy się o najmniejszą wartość przedmiotów całkowicie wypełniających plecak o udźwigu M . Dzięki temu w czasie $O(nM)$ dostaniemy tablicę $d[0..M]$, w której $d[j]$ oznaczać będzie minimalny czas potrzebny na złożenie dokładnie j mebli, jeśli pochodzą one z pewnej liczby w całości złożonych paczek. Zatem, dokładając do tego ℓ sztuk mebla i (dla wszystkich wartości $0 < \ell < c_i$), zmontujemy M mebli w czasie $T_{i,\ell} + d[M - \ell]$. Złożoność czasowa tego rozwiązania to $O(n^2M)$.

Zauważmy, że główny koszt algorytmu to rozwiązanie problemu plecakowego dla każdego podzbioru $n - 1$ przedmiotów. Używając techniki, o której pisaliśmy w zeszłym miesiącu, możemy zmniejszyć złożoność czasową do $O(nM \log n)$.

W kąciku kontynuujemy przygodę z zadaniami, do których rozwiązania przydaje się znajomość problemu plecakowego. Tym razem w nieco trudniejszej jego wersji, w której każdy przedmiot ma swój rozmiar $m[i]$ oraz wartość $w[i]$. Standardowe pytanie, które możemy wtedy zadać, to np. jaka jest największa sumaryczna wartość przedmiotów, które możemy zapakować do plecaka, nie przekraczając jego udźwigu M (patrz pseudokod na marginesie).

Na pierwszy ogień pójdzie zadanie *Meble z Obozu Naukowo-Treningowego im. A. Kreczmara z roku 2013*. Chcemy urządzić mieszkanie i w tym celu zakupiliśmy n rodzajów mebli do samodzielnego montażu. Mamy c_i sztuk mebla rodzaju i ; złożenie pierwszej sztuki tego rodzaju zajmie nam t_i minut, a wraz z postępem prac będziemy nabierać wprawy i złożenie każdej kolejnej sztuki zajmie nam o d_i minut krócej niż poprzedniej (zakładamy, że $t_i > (c_i - 1)d_i$). Chcemy wiedzieć, ile minimalnie czasu potrzebujemy, aby złożyć dowolnych M mebli.

Drugie zadanie pochodzi z Wiosennego Turnieju w Programowaniu Zespołowym organizowanego przez Politechnikę Poznańską (a konkretnie z 8. edycji z roku 2004) i ma tytuł (kto by się spodziewał) *Pakowanie plecaka*. W zadaniu mamy plecak o udźwigu M i n przedmiotów, z których każdy ma swój rozmiar $m[i]$ oraz użyteczność $w[i]$ (przy czym niektóre przedmioty mogą być wyjątkowo nieprzydatne i mieć ujemną użyteczność). Chcemy znaleźć maksymalne upakowanie plecaka (tzn. takie, do którego nie będzie można dołożyć żadnego przedmiotu) o największej sumie użyteczności zabranych przedmiotów (zatem, być może, będziemy zmuszeni zabrać jakiś nieprzydatny przedmiot tylko po to, by dociążyć plecak).

Zauważmy, że jeśli będziemy chcieli zostawić w plecaku puste miejsce o wielkości ℓ , to wszystkie przedmioty o rozmiarach nie większych niż ℓ będą musiały znaleźć się w plecaku (inaczej moglibyśmy dołożyć do plecaka co najmniej jeden z nich). Posortujmy zatem przedmioty względem ich rozmiarów i podzielmy na grupę j przedmiotów „dużych” i $n - j$ przedmiotów „małych”, co po ewentualnym przenumеровaniu przedmiotów da nam

$$w[1] \geq w[2] \geq \dots \geq w[j] > \ell \geq w[j+1] \geq \dots \geq w[n].$$

Do plecaka na pewno weźmiemy wszystkie „małe” przedmioty o łącznym rozmiarze $m_* = \sum_{i>j} m[i]$ i użyteczności $w_* = \sum_{i>j} w[i]$. Pozostałe przedmioty muszą zająć w plecaku miejsce o wielkości dokładnie $M - m_* - \ell$, ale poza tym możemy je wybrać dowolnie. Zatem wystarczy, że rozwiążemy dla nich problem plecakowy maksymalizujący wartość zabranych przedmiotów – odpowiedzią będzie $d[M - m_* - \ell] + w_*$.

Jeśli przeiterujemy po wartościach ℓ w kolejności malejącej, to każdy przedmiot dokładnie raz zmieni swój status z „małego” na „duży”. Możemy więc na bieżąco rozwiązywać problem plecakowy i uaktualniać wartości m_* i w_* . Wtedy koszt każdej takiej zmiany wyniesie $O(M)$, a cały algorytm będzie miał złożoność czasową $O(n \log n + nM)$.

Tomasz IDZIASZEK