# Ciężkie życie sklepikarza, czyli problem wydawania reszty

W niniejszym artykule postaram się zaprezentować problem wydawania reszty, mając do dyspozycji pewien zbiór dostępnych nominałów. Jest on bardzo popularny i często występuje w wielu odmianach odwołując się najczęściej do dynamicznego programowania, równań modularnych czy generowania obiektów kombinatorycznych.

Przykład: mamy do dyspozycji wiele monet nominałów *2,5,7* – chcemy wydać *10*. Gdybyśmy zaczęli od najwyższego możliwego nominału to zostałoby nam do wydania *3*, co jest niemożliwe. Optymalnym rozwiązaniem jest 10=5+5. Takich właśnie rozwiązań postaramy się szukać. Ponadto, najczęściej będzie interesował nas sposób wydania danej kwoty w jak najmniejszej ilości monet.

Zbiór *S* będziemy nazywać skończonym, jeżeli dany nominał możemy wykorzystać tylko tyle razy, co występuje w *S*, a nieskończonym, jeżeli dany nominał z *S* możemy wykorzystać nieskończoną ilość razy. Kwotę, którą należy wydać będziemy oznaczać jako *K*, a liczebność zbioru *S* jako *n*.

## Przypadek ogólny

W przypadku ogólnym problem wydawania reszty sprowadza się do klasycznego problemu sumy podzbioru. Dany jest skończony zbiór *S* i pewna wartość *K*. Naszym zadaniem jest znalezienie takiego podzbioru *S'* należącego do *S*, że suma jego elementów wynosi *K*. Jest to problem NP-zupełny, a więc najprawdopodobniej nie istnieje wielomianowy algorytm rozwiązujący to zadanie.

Najbardziej oczywistym jest sprawdzenie wszystkich możliwości generując każdy możliwy podzbiór – da to algorytm o złożoności $O(2^n)$. Możemy oczywiście zastosować pewne heurystyki (przeszukiwanie z nawrotami, rozważanie szczególnych przypadków) jednakże nie zmieni to złożoności algorytmu.

Okazuję się jednak, że problem ten można rozwiązać 'trochę' szybciej. Na początku dzielimy nasz zbiór *S* na połowę. Następnie z obu połówek generujemy wszystkie możliwe podzbiory (włącznie z pustymi, otrzymujemy wtedy $2 \cdot 2^{n/2}$ podzbiorów). Dla każdego z tych podzbiorów zapisujemy jego sumę w tablicy *Q*, jeżeli jest on z pierwszej połówki, albo *W*, jeżeli jest on z drugiej połówki. Następnie sortujemy rosnąco obie tablice. Dodatkowo, jeżeli chcemy wiedzieć, suma jakich liczb będzie wynosiła *K*, należy także zapamiętać co wchodzi w skład danego podzbioru. Najlepiej jest to zrobić binarnie – ponieważ n<50 (jeżeli nie to i tak nie wyrobimy się w czasie) to zapisujemy tylko czy dana liczba z konkretnej połówki wchodzi w skład podzbioru czy nie – zajmie to 3-4 bajty dla jednego podzbioru. Niech *m* będzie długością tablicy *Q*, a *h* długością tablicy *W*. Teraz wystarczy wykonać poniższy algorytm:

```
1.  i:= 1
2.  j:= h
3.  stop := false
4.  while not stop do
5.      if  Q[i] + W[j] < K then
6.              i:= i +1
7.              if i=m+1 then
8.                      wypisz „nie znaleziono”
9.                      stop:=true
10.     else if Q[i] + W[j] > K then
```

```
11.             j:= j – 1
12.             if j=0 then
13.                     wypisz „nie znaleziono”
14.                     stop:=true
15.
16.   else
17.       wypisz „znaleziono”
18.       stop:=true
```

Otrzymujemy algorytm działający w $O(2^{n/2})$.

## Przypadki szczególne

Oczywiście cała zabawa zaczyna się, gdy na dane wejściowe narzucone są pewne ograniczenia i zadanie można rozwiązać w czasie wielomianowym.

### 1. Małe: *K*, elementy zbioru *S*, duże: *n*.

Jest to najpopularniejszy i najprostszy przypadek szczególny. Naszym zadaniem jest znalezienie najmniejszej liczby elementów z *S*, których suma wynosi *K*. Problem ten rozwiązujemy klasycznym algorytmem dynamicznym. Możemy tu wyróżnić dwa przypadki:
**a)** *S* nieskończony
**b)** *S* skończony
Bez wnikania w szczegóły algorytm w przypadku a) wygląda następująco:

Tablica *w[0..K]* gdzie *w[i]* to ilość monet potrzebnych do wydania kwoty *i*.

```
1.  for i:=1 to K do w[i]:=INFINITY
2.  w[0]:= 0
3.
4.  for i:=1 to n do
5.          czytaj nominal
6.          for j:=0 to K-nominal do
7.             if w[j]=INFINITY then continue;   // kwota j nie została dotąd uzyskana
8.             if w[j+nominal]> w[j]+1  then  w[j+ nominal]:=w[j]+1; // lepszy wynik dla
9.                                                                    kwoty j+nominal
10.
11.  if w[K]=INFINITY then wypisz „nie można wydać”
12.  else wypisz w[K]  // wypisujemy najmniejszą ilość wykorzystanych elementów
13.                    których suma wynosi K
```

Algorytm ma złożoność *O(nK)*. Przypadek b) możemy rozwiązać w podobny sposób, z tym że:
1. Należy uważać żeby nie wykorzystywać tego samego nominału więcej razy niż to możliwe.
2. Kolejne nominały należy rozpatrywać w kolejności malejącej.

### 2. n=2, duże K i elementy zbioru S

W tej wersji mamy do dyspozycji tylko dwa nominały o wartości *A* i *B* - naszym zadaniem jest oczywiście wydanie kwoty *K* używając jak najmniejszej liczby monet. Załóżmy, że *A>B*. Mamy:

p•A+q•B=K  gdzie p+q→min (p,q całkowite nieujemne )

Możemy to równanie przekształcić do:

K-p•A≡0 ( mod B ) – szukamy takiego największego *p*, że liczba *K-p•A* będzie podzielna przez *B* ( a zatem będziemy mogli wydać *K-p•A* za pomocą nominału *B* )
K≡p•A ( mod B )

Teraz wystarczy znaleźć największe możliwe i dopuszczalne *p*, za pomocą popularnego algorytmu rozwiązywania modularnych równań liniowych, który w tym przypadku będzie miał złożoność *O( lgB )*. Mając *p* bez problemu wyliczamy *q*.

### 3. Zastosowanie algorytmu zachłannego

Niektóre skończone zbiory *S* (nazwijmy je *ładnymi* zbiorami) umożliwiają zastosowanie prostego algorytmy zachłannego, który zwróci poprawną odpowiedź. W algorytmie tym wydajemy kwotę *K* zużywając największy możliwy nominał *C*, a następnie w ten sam sposób wydajemy kwotę *K-C*. Po wykonaniu algorytmu otrzymujemy najmniejszy możliwy podzbiór *S'*, którego suma wynosi *K*. *Ładnym* zbiorem jest np. zbiór, którego każdy element jest wielokrotnością każdego mniejszego elementu, w szczególności $S=\{m^a, m^b, m^c….\}$..

Czasami może się zdarzyć sytuacja, kiedy *S* będzie 'prawie' *ładny* np. *S={1,5,10,20,50,100}*. Zbiór ten byłby *ładny* gdyby nie zawierał *20* albo *50*. W takim przypadku, także można zastosować algorytm zachłanny z tym, że musimy rozważyć 2 przypadki:
**a)** element 50 używamy maksymalną ilość razy
**b)** element 50 używamy o 1 raz mniej niż w a)

Rozważmy teraz problem stwierdzenia czy dany *nieskończony* zbiór *S* jest *ładny*, to jest każdą kwotę można wydać używając jak najmniejszej ilości monet w sposób zachłanny. Zadanie to rozwiązał w 1994 roku David Pearson przedstawiając algorytm działający w czasie $O(n^3)$ który zwraca najmniejszą kwotę, której nie da wydać się optymalnie postępując zachłannie ( np. dla *S={4,3,1}* algorytm zwróci *6*- zachłannie *6=4+1+1*, a optymalnym rozwiązaniem jest *6=3+3* ) lub stwierdza że *S* jest *ładny*.

Niech będą dane nominały $c_1 > c_2 > c_3 > \ldots > c_n$, gdzie $c_n = 1$ ( inaczej zbiór nie będzie *ładny* ). Reprezentacją liczby *L* nazwiemy zbiór o *n* elementach postaci $\{ a_1, a_2, a_3 \ldots a_n \}$ taki, że $a_1 \bullet c_1 + a_2 \bullet c_2 + \ldots + a_n \bullet c_n = L$. Algorytm:

**for** i:=2 **to** n
  **for** j:=i **to** n
    **1.** policz zachłannie reprezentacje *R* liczby $c_{i-1} - 1$
    **2.** utwórz reprezentacje *M* biorąc pierwsze *j* wartości z *R*, do *j*-tej wartości dodaj *1* a resztę wypełnij zerami
    **3.** niech *x* będzie liczbą powstałą z *M* – wydaj zachłannie *x* i jeżeli okaże się, że potrzeba na to więcej monet niż w jest w *M* to *x* jest jedną z odpowiedzi – sprawdź czy *x* jest mniejsze od poprzednich wyników i jeżeli potrzeba zaktualizuj wynik

Jeżeli algorytm nie znajdzie żadnego wyniku to *S* jest *ładny*.

Przykład działania algorytmu:
*S={ 6,4,1 }; i=2;j=2;*
1. Liczymy reprezentacje *6-1=5.  R=(0,1,1)*
2. Bierzemy pierwsze *j* wartości *( M=(0,1,..) )*, do *j*-tej dodajemy *1 ( M=(0,2,..) )* a resztę uzupełniamy zerami *( M=(0,2,0) ).*
3. *x=8*; zachłannie *8=(1,0,2)* zatem *8* jest jedną z odpowiedzi