Bartosz Walczak
Treść zadania, Opracowanie

Jakub Radoszewski
Program

OI, Etap II, dzień 1, 9–02–2005

Banknoty

Bajtocki Bank Bitowy (w skrócie BBB) ma największą w Bajtocji sieć bankomatów. BBB postanowił usprawnić swoje bankomaty i zwrócił się do Ciebie o pomoc. Środkiem płatniczym w Bajtocji są banknoty o nominałach $b_1, b_2, \ldots, b_n$. *BBB postanowił, że bankomaty powinny wypłacać żądaną kwotę w jak najmniejszej łącznej liczbie banknotów.*

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia opis zapasu banknotów, które posiada bankomat, oraz kwotę do wypłacenia,*
- *obliczy minimalną łączną liczbę banknotów, za pomocą jakiej bankomat może wypłacić żądaną kwotę, oraz znajdzie pewien sposób jej wypłacenia,*
- *wypisze wynik na standardowe wyjście.*

Wejście

W pierwszym wierszu wejścia znajduje się liczba nominałów n, $1 \leqslant n \leqslant 200$. *Drugi wiersz zawiera* n *liczb całkowitych* $b_1, b_2, \ldots, b_n$, $1 \leqslant b_1 < b_2 < \ldots < b_n \leqslant 20\,000$, *pooddzielanych pojedynczymi odstępami. Trzeci wiersz zawiera* n *liczb całkowitych* $c_1, c_2, \ldots, c_n$, $1 \leqslant c_i \leqslant 20\,000$, *pooddzielanych pojedynczymi odstępami;* c_i *jest liczbą banknotów o nominale* b_i *znajdujących się w bankomacie. W ostatnim, czwartym wierszu wejścia znajduje się jedna liczba całkowita* k *— kwota, którą bankomat ma wypłacić,* $1 \leqslant k \leqslant 20\,000$. *Możesz założyć, dla danych testowych, że kwotę* k *można wypłacić za pomocą dostępnych banknotów.*

Wyjście

Pierwszy wiersz wyjścia powinien zawierać jedną dodatnią liczbę całkowitą równą minimalnej łącznej liczbie banknotów, za pomocą których bankomat może wypłacić kwotę k. *Drugi wiersz wyjścia powinien zawierać* n *liczb całkowitych, oddzielonych pojedynczymi odstępami i oznaczających liczby sztuk poszczególnych banknotów użytych do wypłacenia kwoty* k. *W przypadku, gdy istnieje więcej niż jedno rozwiązanie, program powinien wypisać którekolwiek.*

Przykład

Dla danych wejściowych:

```
3
2 3 5
2 2 1
10
```

poprawnym wynikiem jest:

```
3
1 1 1
```

Rozwiązanie

Podstawowy algorytm dynamiczny

Zadanie należy do grupy problemów *wydawania reszty*, które często rozwiązuje się metodą *programowania dynamicznego*. Metoda ta polega na stopniowym konstruowaniu coraz większych częściowych rozwiązań, aż do uzyskania rozwiązania całego problemu.

W naszym zadaniu będziemy konstruować rozwiązania kolejno, dla coraz większych zbiorów nominałów $\{b_1\}, \{b_1, b_2\}, \ldots, \{b_1, \ldots, b_n\}$. Najpierw obliczymy, jak optymalnie wypłacić kwoty z przedziału $[0, k]$ korzystając tylko z banknotów o nominale b_1. Potem ponownie obliczymy optymalne rozwiązania dla tych kwot, ale korzystając już z banknotów o nominałach $\{b_1, b_2\}$. Następnie powtórzymy obliczenia dla nominałów $\{b_1, b_2, b_3\}$ itd.

Rozważmy sytuację, w której bierzemy pod uwagę tylko banknoty o nominałach $b_1, \ldots, b_i$ dla pewnego $0 \leqslant i \leqslant n$, i wprowadźmy następujące oznaczenia:

- Przez $R_{i,j}$ oznaczymy minimalną liczbę banknotów potrzebną do wypłacenia kwoty j ($0 \leqslant j \leqslant k$), za pomocą banknotów $\{b_1, \ldots, b_i\}$. Jeżeli kwoty j nie można wypłacić za pomocą banknotów $\{b_1, \ldots, b_i\}$, przyjmujemy $R_{i,j} = \infty$.
- Jeśli kwotę j można wypłacić za pomocą rozważanych banknotów, czyli $i \geqslant 1$ oraz $R_{i,j} \neq \infty$, to przez $W_{i,j}$ oznaczymy liczbę banknotów o nominale b_i, które są użyte do wypłacenia kwoty j za pomocą $R_{i,j}$ banknotów ($0 \leqslant W_{i,j} \leqslant c_i$).

Podstawową własnością rozważanego problemu, dzięki której do jego rozwiązania możemy zastosować metodę programowania dynamicznego, jest własność *optymalnej podstruktury*. Pozwala ona konstruować rozwiązanie optymalne problemu, opierając się *tylko* na rozwiązaniach optymalnych podproblemów. W naszym przypadku zasada ta mówi, że jeżeli $R_{i,j}$ (dla $i \geqslant 1$ oraz $R_{i,j} \neq \infty$) jest minimalną liczbą banknotów $b_1, \ldots, b_i$ potrzebną do wypłacenia kwoty j, to $R_{i-1,j'}$, gdzie $j' = j - b_i \cdot W_{i,j}$, jest minimalną liczbą banknotów $b_1, \ldots, b_{i-1}$ potrzebną do wypłacenia kwoty j'. Stąd mamy następujący wzór pozwalający wyznaczać wartości $R_{i,j}$ dla $i \geqslant 1$:

$$R_{i,j} \quad = \quad \min\{R_{i-1,j'} + w : 0 \leqslant w \leqslant c_i \wedge 0 \leqslant j' = j - w \cdot b_i\} \tag{1}$$

Przy tym jeżeli $R_{i,j} \neq \infty$, to $W_{i,j}$ jest tą wartością w, dla której występuje minimum. Dla $i = 0$ mamy oczywiście:

$$R_{0,j} = \begin{cases} 0 & \text{gdy } j = 0 \\ \infty & \text{gdy } j \geqslant 1 \end{cases}$$

Stosując powyższe wzory, możemy w czasie $\Theta(n \cdot k^2)$ obliczyć wszystkie wartości $R_{i,j}$ — dla każdej z $\Theta(n \cdot k)$ wartości $R_{i,j}$ obliczamy minimum (1) w czasie $\Theta(k)$. W ten sposób mamy algorytm znajdujący poszukiwaną wartość $R_{n,k}$. Jednak dla ograniczeń na n i k podanych w treści zadania działa on zbyt długo na dużych testach.

Ulepszony algorytm dynamiczny

Zauważmy, że do obliczenia wartości $R_{i,j}$ ze wzoru (1) wykorzystujemy tylko te wartości $R_{i-1,j'}$, w których $j' = j - w \cdot b_i$ dla pewnego w. Zatem dla liczby $r \in [0, b_i)$ wartości $R_{i,r}, R_{i,r+b_i}, \ldots, R_{i,r+l \cdot b_i}$ obliczamy tylko na podstawie wartości $R_{i-1,r}, R_{i-1,r+b_i}, \ldots, R_{i-1,r+l \cdot b_i}$. To spostrzeżenie pozwoli nam lepiej zorganizować i przyśpieszyć obliczanie minimów według wzoru (1).

Rozważmy ustalone $i \in [1,n]$ oraz $r \in [0,b_i)$ i niech kwota do wydania będzie postaci $j = r + l \cdot b_i$ dla pewnego l. Wtedy ze wzoru (1), po przedstawieniu liczby wydawanych banknotów o nominale b_i w postaci $w = l - l'$, otrzymujemy:

$$R_{i,j} = \min\{R_{i-1,r+l' \cdot b_i} + (l - l') : l - c_i \leqslant l' \leqslant l \wedge 0 \leqslant l'\}$$

Wprowadzając dla l, takich że $0 \leqslant r + l \cdot b_i \leqslant k$, pomocnicze oznaczenie:

$$M_l = R_{i-1,r+l \cdot b_i} - l$$

możemy wzór (1) zapisać następująco:

$$R_{i,j} \quad = \quad \min\{M_{l'} : l - c_i \leqslant l' \leqslant l \wedge 0 \leqslant l'\} + l \qquad (2)$$

Stąd obliczanie kolejno wartości $R_{i,r+l \cdot b_i}$ dla $l = 0, 1, \ldots$ sprowadza się do wyliczania minimów zbiorów $S_l = \{M_{l'} : \max\{l - c_i, 0\} \leqslant l' \leqslant l\}$ dla kolejnych wartości l.

Zauważmy, że jeżeli $\max\{l - c_i, 0\} \leqslant l'' < l' \leqslant l$ oraz $M_{l''} \geqslant M_{l'}$, to wartość $M_{l''}$ nie ma wpływu na wartości $R_{i,r+l \cdot b_i}$, gdyż jest zawsze przysłaniana przez wartość $M_{l'}$. Stąd ze zbioru S_l wystarczy pamiętać tylko takie elementy $Q_l = (M_{l_1}, \ldots, M_{l_m})$, że $\max\{l - c_i, 0\} \leqslant l_1 < \ldots < l_m \leqslant l$ oraz $M_{l_1} < \ldots < M_{l_m}$. Wówczas $\min(S_l) = \min(Q_l) = M_{l_1}$. Co więcej, ciąg Q_{l+1} można łatwo wyznaczyć na podstawie ciągu Q_l. W tym celu:

- jeśli $l_1 = l - c_i$, to usuwamy z początku Q_l element M_{l_1};
- usuwamy z końca ciągu Q_l wszystkie wartości większe lub równe M_{l+1};
- wstawiamy na koniec ciągu Q_l element M_{l+1} otrzymując w ten sposób ciąg Q_{l+1}.

Implementacja algorytmu

Zauważmy, że wszystkie operacje wykonywane na ciągach Q_l dotyczą elementów stojących na początku lub na końcu ciągu. Stąd Q_l możemy zaimplementować jako kolejkę o dwóch końcach (ang. *double-ended queue*, w skrócie *deque*).

W poniższym pseudokodzie tablica `R` jest przeznaczona na wartości $R_{i,j}$ dla aktualnie rozważanego i (jest uaktualniana w każdym kroku pętli), a `Q` jest kolejką o dwóch końcach, w której znajdują się indeksy elementów tworzących Q_l: $l_1, \ldots, l_m$. Liczba `k` jest kwotą, którą mamy wydać, a tablice `b` i `c` zawierają dane nominały banknotów i ich liczbę. Operacje `pop_front` i `push_front` oznaczają odpowiednio usunięcie i wstawienie elementu na początek kolejki. Analogicznie, operacje `pop_back` i `push_back` oznaczają odpowiednio usunięcie i wstawienie elementu na koniec kolejki. Funkcja `front` zwraca pierwszy element kolejki.

```
1:  R[0] := 0;
2:  R[1..k] := ∞;
3:  for i := 1 to n do
4:     for r := 0 to b[i] − 1 do
5:        Q.clear;
6:        l := 0;
7:        while r + l · b[i] ⩽ k do
8:           M[l] := R[r + l · b[i]] − l;
9:           while Q.not_empty and M[Q.back] ⩾ M[l] do Q.pop_back;
10:          Q.push_back(l);
11:          R[r + l · b[i]] := M[Q.front] + l;
12:          W[i, r + l · b[i]] := l − Q.front;
13:          if Q.front = l − c[i] then Q.pop_front;
14:          l := l + 1;
```

Dokładne odtworzenie sposobu wypłacenia kwoty k nie stanowi problemu:

```
j := k;
for i := n downto 1 do
   wypłać W[i,j] banknotów o nominale b[i];
   j := j − W[i, j] · b[i];
```

Aby oszacować złożoność pierwszej procedury przyjrzyjmy się bliżej zewnętrznej pętli `while` (wiersze 7–14). Jeśli pominiemy wewnętrzną pętlę `while` (wiersz 9), to złożoność pozostałych operacji jest stała. Natomiast sumaryczną złożoność wszystkich iteracji wewnętrznej pętli `while` (wykonywanych we wszystkich zewnętrznych pętlach `while`) możemy oszacować zauważając, że dla ustalonych i i r każde l jest raz wstawiane do kolejki, więc co najwyżej raz może zostać z niej usunięte. Dlatego całkowita złożoność zewnętrznej pętli `while` dla nominału b_i wynosi $\Theta(k/b_i)$. Tym samym złożoność czasowa algorytmu wynosi:

$$\Theta(k) + \sum_{i=1}^{n} b_i \cdot \Theta(k/b_i) = \Theta(n \cdot k).$$

Złożoność pamięciowa algorytmu jest także równa $\Theta(n \cdot k)$. Gdybyśmy jednak chcieli obliczyć tylko optymalną liczbę banknotów potrzebnych do wypłacenia kwoty, to moglibyśmy zmodyfikować algorytm tak, by działał w pamięci $\Theta(n + k)$ (nie byłaby potrzebna tablica `W`).

Przedstawione rozwiązanie zostało zaimplementowane w `ban.cpp` (z STL), `ban0.cpp` (bez STL) i `ban1.pas`.

Testy

Rozwiązania zawodników były oceniane na zestawie 18 testów. Testy 1–4 to proste testy poprawnościowe (w szczególności, testy 1–2 pozwalają sprawdzić pewne przypadki brzegowe), testy 5–7 to niewielkie testy losowe, testy 8–15 to duże losowe testy wydajnościowe, natomiast testy 16–18 to specyficzne testy wydajnościowe.

Nazwa	n	k	Opis
ban1.in	5	28	
ban2.in	1	104	
ban3.in	3	5 000	
ban4.in	10	500	
ban5.in	46	1 000	
ban6.in	40	2 000	
ban7.in	181	6 982	
ban8.in	113	10 034	
ban9.in	127	16 933	dokładnie jeden nieparzysty nominał, nieparzysta kwota
ban10.in	21	19 998	małe nominały, duża kwota
ban11.in	170	19 989	gęste rozmieszczenie nominałów na małym przedziale
ban12.in	190	19 123	
ban13.in	130	19 999	kwota daje resztę 49 z dzielenia przez 50, dokładnie jeden z nominałów daje resztę 1 (pozostałe 0), więc musi zostać użyty 49-krotnie
ban14.in	185	18 888	
ban15.in	175	20 000	banknoty wyraźnie podzielone na grupy: o małych i dużych nominałach, przy czym żadnego z dużych nie można wziąć do rozkładu wynikowego
ban16.in	15	19 999	test z maksymalną liczbą banknotów o nominałach będących potęgami 2
ban17.in	34	19 999	
ban18.in	20	19 999	