Tomasz Idziaszek
Treść zadania, Opracowanie

Alan Kutniewski
Program

Dostępna pamięć: 64 MB.

OI, etap III, dzień drugi, 29.03.2012

Hurtownia

Bajtazar prowadzi hurtownię z materiałami budowlanymi. W tym sezonie hitem są panele podłogowe i większość dochodu hurtowni pochodzi z ich sprzedaży. Niestety, dość często zdarza się sytuacja, że odwiedzający hurtownię klient składa zamówienie, którego nie da się zrealizować, gdyż w magazynie jest zbyt mało paneli. Żeby nie tracić klientów, Bajtazar postanowił zminimalizować liczbę takich przypadków.

W tym celu przygotował plan pracy na najbliższe n dni. Przeanalizował umowy z producentami paneli i na ich podstawie wyznaczył ciąg $a_1, a_2, \ldots, a_n$. Liczba a_i oznacza, że rano i-tego dnia do magazynu zostanie dostarczonych a_i opakowań paneli.

Bajtazar zrobił też zestawienie ofert, które zgłosili klienci hurtowni, i na jego podstawie wyznaczył ciąg $b_1, b_2, \ldots, b_n$. Liczba b_i oznacza, że w południe i-tego dnia w hurtowni zjawi się klient, który będzie chciał zakupić b_i opakowań paneli. Jeśli Bajtazar zdecyduje się na realizację zamówienia klienta, to będzie je musiał zrealizować w całości. Jeśli w momencie wizyty klienta w magazynie jest mniej opakowań paneli niż potrzebuje klient, to Bajtazar będzie zmuszony odrzucić takie zamówienie. Jeśli natomiast paneli jest wystarczająco dużo, to Bajtazar może zdecydować, czy zrealizować zamówienie klienta, czy też nie.

Na podstawie powyższych danych Bajtazar chce stwierdzić, które zamówienia powinien zrealizować, a które odrzucić, tak by liczba odrzuconych zamówień była jak najmniejsza. Zakładamy, że na początku pierwszego dnia magazyn hurtowni jest pusty.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się liczba całkowita n ($1 \leqslant n \leqslant 250\,000$). W drugim wierszu znajduje się ciąg liczb całkowitych $a_1, a_2, \ldots, a_n$ ($0 \leqslant a_i \leqslant 10^9$). W trzecim wierszu znajduje się ciąg liczb całkowitych $b_1, b_2, \ldots, b_n$ ($0 \leqslant b_i \leqslant 10^9$). Liczby w drugim i trzecim wierszu są pooddzielane pojedynczymi odstępami.

W testach wartych 50% punktów zachodzi dodatkowy warunek $n \leqslant 1\,000$.

Wyjście

W pierwszym wierszu standardowego wyjścia Twój program powinien wypisać liczbę całkowitą k oznaczającą maksymalną liczbę zamówień, które uda się zrealizować Bajtazarowi. W drugim wierszu należy wypisać rosnący ciąg k liczb pooddzielanych pojedynczymi odstępami, oznaczający numery klientów, których zamówienia trzeba w tym celu zrealizować, lub pusty wiersz, gdy nie można zrealizować żadnego zamówienia. Klientów numerujemy od 1 do n w kolejności przychodzenia do hurtowni. Jeśli istnieje więcej niż jedna poprawna odpowiedź, Twój program powinien wypisać dowolną z nich.

Przykład

Dla danych wejściowych:	*jednym z poprawnych wyników jest:*
`6`	`3`
`2 2 1 2 1 0`	`1 2 4`
`1 2 2 3 4 4`	

Rozwiązanie

Podejście pierwsze – programowanie dynamiczne

Nasze zadanie ma dość naturalne rozwiązanie oparte na metodzie programowania dynamicznego. Zauważmy, że jeśli w ciągu pierwszych i dni zrealizowaliśmy pewną liczbę zamówień, to aby wyznaczyć, które zamówienia powinny zostać zrealizowane w kolejnych dniach, wystarczy znać liczbę paneli, które zostały w magazynie po i dniach (czyli nie musimy wiedzieć, które konkretnie zamówienia z pierwszych i dni zostały zrealizowane).

Ta obserwacja prowadzi do następującego rozwiązania. Będziemy wypełniać tablicę d. Dla $0 \leqslant i \leqslant n$ i $A \geqslant 0$ mamy

$$d[i, A] = \text{maksymalna liczba zamówień z dni } \{1, \ldots, i\}, \text{ które można zrealizować, aby w magazynie zostało dokładnie } A \text{ paneli.}$$

Jeśli nie da się tak realizować zamówień, aby na końcu zostało dokładnie A paneli, to przyjmujemy, że $d[i, A] = -\infty$. Dla wygody przyjmujemy także $d[i, A] = -\infty$ dla dowolnego $A < 0$.

Jak wyznaczyć $d[i, A]$? Skoro na początku pierwszego dnia magazyn jest pusty, to $d[0, 0] = 0$ oraz $d[0, A] = -\infty$ dla $A > 0$. Załóżmy zatem, że $i \geqslant 1$. Aby wyznaczyć wartość $d[i, A]$, musimy zdecydować, czy realizować i-te zamówienie. Jeśli go nie realizujemy, to po $i - 1$ dniach musiało nam zostać $A - a_i$ paneli, w przeciwnym wypadku musiało zostać $A - a_i + b_i$ paneli. Zatem

$$d[i, A] = \max(d[i - 1, A - a_i], 1 + d[i - 1, A - a_i + b_i]).$$

Wynikiem jest maksymalna wartość spośród $d[n, A]$ dla wszystkich $A \geqslant 0$.

Problemem w tym rozwiązaniu jest to, że A może być bardzo duże. Po prostu niewłaściwie wybraliśmy, po czym indeksujemy tablicę. Spróbujmy inaczej; dla $0 \leqslant i \leqslant n$ i $0 \leqslant k \leqslant n$ mamy

$$d[i, k] = \text{maksymalna liczba paneli, które mogą zostać w magazynie po zrealizowaniu } k \text{ zamówień w dniach } \{1, \ldots, i\}.$$

Przyjmujemy, że $d[i, k] = -\infty$, jeśli w dniach $\{1, \ldots, i\}$ nie da się zrealizować k zamówień. Znowu $d[0, 0] = 0$ oraz $d[0, k] = -\infty$ dla $k > 0$. Gdy $i \geqslant 1$, musimy znów zdecydować, czy realizujemy i-te zamówienie. Jeśli nie, to w ciągu pierwszych $i - 1$ dni musimy zrealizować k zamówień, a następnie dokładamy do magazynu a_i paneli. A jeśli tak, to w ciągu pierwszych $i-1$ dni musimy zrealizować $k-1$ zamówień,

a i-tego dnia dokładamy a_i paneli i zabieramy b_i. Należy pamiętać, że drugi z tych scenariuszy jest możliwy tylko wtedy, gdy maksymalna liczba paneli, które zostaną po zrealizowaniu $k-1$ zamówień w ciągu pierwszych $i-1$ dni, powiększona o a_i jest co najmniej taka, jak b_i.

Ostatecznym wynikiem jest maksymalna wartość k taka, że $d[n,k] \neq -\infty$, tzn. taka, dla której w ciągu n dni da się zrealizować k zamówień. Poniżej przedstawiamy to rozwiązanie zapisane w postaci pseudokodu.

```
1:  d[0,0] := 0;
2:  for k := 1 to n do
3:    d[0,k] := −∞;
4:  for i := 1 to n do
5:    for k := 0 to n do begin
6:      d[i,k] := d[i−1,k] + a[i];
7:      if k > 0 and d[i−1,k−1] + a[i] ⩾ b[i] then
8:        d[i,k] := max(d[i,k], d[i−1,k−1] + a[i] − b[i]);
9:    end
10: wynik := n;
11: while d[n,wynik] = −∞ do
12:   wynik := wynik − 1;
13: return wynik;
```

Wyznaczenie na podstawie tablicy d, które zamówienia należy zrealizować, zostawiamy jako ćwiczenie dla Czytelnika. Złożoność czasowa i pamięciowa tego rozwiązania to $O(n^2)$. Takie rozwiązanie wystarczało, by zaliczyć połowę testów przygotowanych przez jurorów. Jego implementacje można znaleźć w plikach `hurs1.cpp` i `hurs2.pas`.

Podejście drugie – algorytm zachłanny

Pierwsze pytanie, które moglibyśmy sobie zadać, próbując rozwiązać nasze zadanie, jest następujące: jeśli będziemy zgadzać się na realizację każdego zamówienia, które jest możliwe do zrealizowania, to czy w ten sposób zawsze uzyskamy optymalne rozwiązanie? Dość szybko powinniśmy dojść do wniosku, że odpowiedź na to pytanie jest negatywna: dla $n = 3$ i ciągów $a = (2,0,0)$, $b = (2,1,1)$ taka strategia przydzieli obydwa panele pierwszemu klientowi, natomiast rozwiązanie optymalne przydzieli po jednym panelu drugiemu i trzeciemu klientowi. Nic w tym jednak dziwnego — zadanie nie byłoby ciekawe, gdyby takie rozwiązanie było poprawne. Zostało ono zaimplementowane w pliku `hurb1.cpp`. Nie przechodzi żadnego testu.

Nie porzucajmy jednak myśli o rozwiązaniu zachłannym. Spróbujmy nieco zmienić zadanie. Powiedzmy, że zanim jeszcze pierwszy klient przyjdzie do hurtowni, wszyscy klienci postanowili potwierdzić możliwość realizacji swojego zamówienia telefonicznie. Pytanie brzmi: czy istnieje taka kolejność telefonów, że jeśli będziemy odpowiadali klientom zachłannie (tzn. zamówienie akceptujemy, jeśli da się je zrealizować, biorąc pod uwagę wszystkie poprzednio zaakceptowane zamówienia), to uzyskane rozwiązanie będzie optymalne? Okazuje się, że tak — jeśli klienci będą dzwonić w kolejności niemalejących wielkości zamówień, to strategia zachłanna da optymalne rozwiązanie.

Dowód poprawności strategii zachłannej

Na początek zastanówmy się, jak należałoby sprawdzać, czy da się zrealizować kolejne zamówienie. Najłatwiej byłoby dla każdego klienta, którego zamówienie zdecydujemy się zrealizować, oznaczać panele, które mu sprzedamy. To prowadzi do pytania: które to będą panele? Dla ustalenia uwagi załóżmy, że dzwoni do nas klient o numerze i. Możemy mu przydzielić dowolny podzbiór b_i nieoznaczonych jeszcze paneli z dni $\{1, \ldots, i\}$. Zauważmy, że lepiej przydzielać mu panele z jak najpóźniejszych dni. Istotnie, jeśli później zadzwoni jakiś klient $j < i$, to nie będą go interesowały panele z dni $\{j+1, \ldots, i\}$. A dla klientów $j > i$ nie ma różnicy.

Niech ciąg $a'_1, \ldots, a'_i$ oznacza liczbę nieoznaczonych dotąd paneli spośród paneli dostarczonych do magazynu w poszczególnych dniach, a ciąg $c_1, \ldots, c_i$ — liczbę paneli, które oznaczymy dla klienta i (oczywiście, $0 \leqslant c_j \leqslant a'_j$ dla każdego j). Powiemy, że klientowi i przydzielamy *najświeższe* panele, jeśli $c_1 + \ldots + c_i = b_i$ oraz istnieje takie k, że

$$c_j = \begin{cases} 0 & \text{dla } j < k, \\ a'_j & \text{dla } j > k. \end{cases}$$

Niech $X \subseteq \{1, \ldots, n\}$ będzie podzbiorem klientów. Powiemy, że X *da się zrealizować*, jeśli można zrealizować zamówienia wszystkich klientów z X. Zamówienie klienta o numerze i możemy zrealizować wtedy, gdy i-tego dnia mamy w magazynie, po realizacji poprzednich zamówień, co najmniej b_i paneli. Oczywiście zadanie polega na znalezieniu najliczniejszego zbioru X, który da się zrealizować.

Niech X będzie podzbiorem klientów, a $\prec$ pewnym uporządkowaniem (porządkiem liniowym) elementów zbioru X. Powiemy, że X *da się zrealizować zachłannie względem* $\prec$, jeśli przydzielając kolejnym klientom z X według porządku $\prec$ najświeższe panele, zrealizujemy zamówienia wszystkich klientów z X.

Poniższy lemat pokazuje, że przydział według najświeższych paneli jest zawsze optymalny, niezależnie od wybranego porządku realizacji zamówień:

Lemat 1. Niech X będzie podzbiorem klientów, który da się zrealizować. Wtedy X da się zrealizować zachłannie względem dowolnego uporządkowania $\prec$.

Dowód: Załóżmy przeciwnie, że dla pewnego uporządkowania $\prec$, zbioru X nie udało się zrealizować zachłannie względem $\prec$. Zatrzymajmy się w momencie, gdy dla pewnego klienta j zabrakło paneli. Oznaczmy przez J zbiór klientów x, takich że $x \prec j$ lub $x = j$. Czyli do zbioru J należą ci klienci, którym już zrealizowaliśmy zamówienia, oraz klient j, któremu przydzieliliśmy część zamówionych przez niego paneli.

Niech i będzie największą liczbą, taką że wszystkie panele z dni $\{1, \ldots, i\}$ są oznaczone, ale część paneli z dnia $i+1$ pozostała nieoznaczona (oczywiście $i < n$, bo inaczej znaczyłoby, że X nie da się zrealizować). Wszystkie panele z dni $\{1, \ldots, i\}$ przydzielaliśmy tylko klientom ze zbioru $J_i = J \cap \{1, \ldots, i\}$, gdyż w przeciwnym przypadku wszystkie panele z dnia $i+1$ musiałyby być oznaczone. Również $j \in J_i$, bo inaczej można byłoby mu przydzielić kolejny panel z dnia $i+1$. Ponieważ ostatecznie zabrakło nam paneli, więc

$$\sum_{x \in J_i} b_x > a_1 + \ldots + a_i,$$

zatem $J_i \subseteq X$ nie da się zrealizować. Sprzeczność. ■

Twierdzenie 1. *Niech $\prec$ będzie uporządkowaniem wszystkich klientów niemalejąco względem ich zapotrzebowań (tzn. $i \prec j$ jeśli $b_i \leqslant b_j$; remisy rozstrzygamy dowolnie). Wtedy realizowanie zamówień zachłanne względem $\prec$ daje optymalne rozwiązanie.*

Dowód: Niech ALG to zbiór klientów wyznaczonych do realizacji ich zamówień przez nasz algorytm, zaś OPT to zbiór klientów wyznaczonych w pewnym rozwiązaniu optymalnym. Niech $OPT = \{i_1, i_2, \ldots, i_m\}$, przy czym $i_1 \prec i_2 \prec \ldots \prec i_m$. Niech l będzie największą liczbą, taką że $\{i_1, \ldots, i_l\} \subseteq ALG$. Zakładamy, że OPT, spośród wszystkich rozwiązań optymalnych, daje największe l (tzn. ma możliwie najdłuższy wspólny prefiks z ALG).

Na początek rozważmy przypadek, gdy OPT jest właściwym nadzbiorem ALG. Mamy wówczas $i_{l+1} \notin ALG$, choć z lematu wynika, że $ALG \cup \{i_{l+1}\} \subseteq OPT$ da się zrealizować zachłannie względem $\prec$. Sprzeczność.

Załóżmy zatem, że $ALG \setminus OPT \neq \emptyset$. Niech x będzie pierwszym klientem względem uporządkowania $\prec$, takim że $x \in ALG$ i $x \notin OPT$. Pokażemy teraz inne rozwiązanie optymalne, które będzie zawierało $\{i_1, \ldots, i_l, x\} \subseteq ALG$, a zatem znowu dojdziemy do sprzeczności, gdyż to rozwiązanie będzie miało dłuższy prefiks wspólny z ALG.

Na początek zrealizujmy zachłannie $\{i_1, \ldots, i_l\}$ względem $\prec$. Oznaczmy teraz panele, które zachłannie przydzielilibyśmy klientowi x, ale faktycznie ich nie przydzielajmy. Następnie zrealizujmy zachłannie $\{i_{l+1}, \ldots, i_m\}$ względem uporządkowania $>$, czyli w kolejności *malejących numerów klientów*. Na mocy lematu, w ten sposób przydzielimy panele wszystkim klientom z OPT.

Zauważmy teraz, że $x \prec i_{l+1}, \ldots, i_m$, gdyż w przeciwnym razie w rozwiązaniu ALG zamiast klienta x wybralibyśmy klienta i_{l+1}. Innymi słowy, zamówienia klientów $i_{l+1}, \ldots, i_m$ są co najmniej tak liczne, jak zamówienie klienta x. To pozwala nam dokonać kluczowej obserwacji: panele, które oznaczyliśmy dla klienta x, zostały przydzielone co najwyżej dwóm klientom ze zbioru $\{i_{l+1}, \ldots, i_m\}$.

Jeśli nie przydzieliliśmy tych paneli nikomu, to mamy sprzeczność — przydzielając je teraz klientowi x, dostajemy rozwiązanie liczniejsze niż OPT.

Jeśli przydzieliliśmy je jednemu klientowi $u \in \{i_{l+1}, \ldots, i_m\}$, to anulujemy realizację jego zamówienia i przydzielamy oznaczone panele klientowi x.

Jeśli przydzieliliśmy je dwóm klientom $u, v \in \{i_{l+1}, \ldots, i_m\}$, $u < v$, to anulujemy realizację zamówienia u, przydzielamy oznaczone panele klientowi x, a z nieoznaczonych paneli u rekompensujemy stratę klienta v. W ten sposób klientowi x przydzieliliśmy b' paneli klienta v, a temu drugiemu daliśmy ich w zamian $b_u - (b_x - b') \geqslant b'$.

Tak więc jedyna możliwość, która pozostała, to $ALG = OPT$. Zatem rozwiązanie wyznaczane przez nasz algorytm jest optymalne. ■

Powyższy dowód poprawności został zaproponowany przez Jakuba Pachockiego.

Implementacja algorytmu: rozwiązanie wzorcowe

W algorytmie zachłannym przeglądamy zatem klientów w kolejności niemalejących zapotrzebowań i realizujemy zamówienia zawsze, kiedy się da.

Będziemy trzymać tablicę $a[1..n]$, gdzie $a[i]$ oznacza liczbę nieoznaczonych jeszcze paneli z i-tego dnia. Zamówienie klienta i można zrealizować, gdy:

$$a[1] + a[2] + \ldots + a[i] \geqslant b_i.$$

Jeśli warunek jest spełniony, to musimy oznaczyć b_i paneli, czyli uaktualnić tablicę a. Robimy to, przeglądając kolejno elementy od $a[i]$ do $a[1]$ i za każdym razem odejmując tyle, ile możemy, dopóki w sumie nie odejmiemy b_i. Całość została zapisana w poniższym pseudokodzie (zignorujmy na razie wiersz 12).

```
 1: z := (1, ..., n);
 2: Posortuj ciąg z[i] w kolejności niemalejących zapotrzebowań b[z[i]];
 3: wynik := 0;
 4: for i := 1 to n do begin
 5:    B := b[z[i]];
 6:    A := a[1] + ... + a[z[i]];
 7:    if A ⩾ B then begin
 8:       wynik := wynik + 1;
 9:       zgłoś zlecenie z[i] do realizacji;
10:       j := z[i];
11:       while B > 0 do begin
12:          j := ostatniNiezerowy(j);
13:          if B ⩽ a[j] then begin
14:             a[j] := a[j] − B;
15:             B := 0;
16:          end else begin
17:             B := B − a[j];
18:             a[j] := 0;
19:          end
20:          j := j − 1;
21:       end
22:    end
23: end
24: return wynik;
```

Przeanalizujmy złożoność czasową naszego algorytmu. Sortowanie w wierszu 2 realizujemy w czasie $O(n \log n)$. Wyznaczenie wartości A z wiersza 6 możemy zrealizować w czasie $O(z[i])$; również pętla z wiersza 11 wykona co najwyżej $z[i]$ obrotów, gdyż warunek z wiersza 7 gwarantuje, że j będzie zawsze dodatnie. Tak więc cała pętla **for** z wiersza 4 wykonuje się w czasie $O(1 + 2 + \ldots + n) = O(n^2)$. Takie rozwiązanie zostało zaimplementowane w plikach `hurs3.cpp` i `hurs4.pas`.

No cóż, nie widać zysku w porównaniu z algorytmem dynamicznym. Spróbujmy więc zoptymalizować nasz algorytm. Potrzebujemy struktury danych, która dla tablicy $a[1..n]$ umożliwi nam obliczanie sum prefiksowych (wiersz 6) i uaktualnianie wartości elementów (wiersze 14 i 18). Stosując odpowiednie *drzewo przedziałowe* (patrz dalej), obie te operacje można zrealizować w czasie $O(\log n)$. Wciąż jednak wykonanie pętli z wiersza 11 może za każdym razem wymagać pełnych $z[i]$ obrotów (jest tak np. w sytuacji, gdy wszystkie panele zostały dostarczone pierwszego dnia). Przeanalizujmy to dokładniej. Jeśli warunek w wierszu 13 jest prawdziwy, to zerujemy B i pętla się kończy. Wobec tego cała praca jest wykonywana w wierszach 17–18. Zauważmy, że wiersz 18 zeruje element $a[j]$. Wynika z tego, że podczas całego programu ten wiersz wykona się co najwyżej n razy dla dodatniego elementu $a[j]$. Ale przypadek $a[j] = 0$

jest dla nas mało interesujący — oznacza, że wszystkie panele z j-tego dnia zostały już przydzielone. Możemy więc ograniczyć się do rozpatrywania tylko tych wartości j, dla których $a[j] > 0$. To właśnie wykonuje wiersz 12: funkcja *ostatniNiezerowy*(j) zwraca największą liczbę $x \leqslant j$, dla której $a[x] > 0$. Jak pokażemy poniżej, funkcję *ostatniNiezerowy* możemy również zrealizować na drzewie przedziałowym w czasie $O(\log n)$.

Podsumowując, pętla z wiersza 11 sumarycznie będzie miała co najwyżej $2n$ obrotów. Tak więc cały algorytm będzie miał złożoność czasową $O(n \log n)$. Został on zapisany w plikach `hur.cpp` i `hur1.pas`.

Implementacja drzewa przedziałowego

Niech 2^m będzie najmniejszą potęgą dwójki większą niż n. Trzymamy dwie tablice $sum[1..2^{m+1}-1]$ i $ost[1..2^{m+1}-1]$. Element $sum[2^l+i]$ dla $0 \leqslant i < 2^l$ oznacza sumę wartości a na przedziale $[i \cdot 2^{m-l}, (i+1) \cdot 2^{m-l})$. Natomiast element $ost[2^l+i]$ oznacza największe $j \in [i \cdot 2^{m-l}, (i+1) \cdot 2^{m-l})$, takie że $a[j] > 0$ (lub $-\infty$, gdy takie j nie istnieje). Na początku tablicę *sum* wypełniamy zerami, a tablicę *ost* wartościami $-\infty$.

Wykonanie przypisania $a[i] := v$ realizuje procedura *ustaw*(i, v). Można ją też wykorzystać do zainicjowania drzewa początkowym ciągiem a.

```
 1: procedure ustaw(i, v)
 2: begin
 3:   i := i + 2^m;
 4:   sum[i] := v;
 5:   if v > 0 then ost[i] := i; else ost[i] := −∞;
 6:   while i > 1 do begin
 7:     i := i div 2;
 8:     sum[i] := sum[2i] + sum[2i + 1];
 9:     ost[i] := max(ost[2i], ost[2i + 1]);
10:   end
11: end
```

Obliczenie sumy $a[1] + \ldots + a[i]$ realizuje poniższa funkcja *sumaPrefiksowa*(i). Ponieważ funkcja *ostatniNiezerowy*(i) jest analogiczna, w komentarzach zaznaczono, czym różnią się obie funkcje.

```
 1: function sumaPrefiksowa(i)          { ostatniNiezerowy(i) }
 2: begin
 3:   i := i + 2^m;
 4:   wynik := sum[i];                  { wynik := ost[i]; }
 5:   while i > 1 do begin
 6:     if i mod 2 = 1 then
 7:       wynik := wynik + sum[i − 1];  { wynik := max(wynik, ost[i − 1]); }
 8:     i := i div 2;
 9:   end
10:   return wynik;
11: end
```

Wszystkie trzy funkcje działają w czasie $O(m) = O(\log n)$. Dodajmy, że można zrezygnować z utrzymywania tablicy *ost*, jednak wtedy funkcja *ostatniNiezerowy* będzie musiała zostać przepisana w innej formie:

```
1:  function ostatniNiezerowy(i)
2:  begin
3:    s := sumaPrefiksowa(i);
4:    j := 1;
5:    while j < 2^m do begin
6:      j := 2j;
7:      if sum[j] < s then begin
8:        s := s − sum[j];
9:        j := j + 1;
10:     end
11:   end
12:   return j;
13: end
```

Epilog

Warto dodać, że jeśli zamiast maksymalizować liczbę zrealizowanych zamówień, będziemy chcieli maksymalizować liczbę sprzedanych paneli, to problem stanie się NP-zupełny, wobec tego prawdopodobnie nie da się go rozwiązać w czasie wielomianowym. Zainteresowani Czytelnicy mogą to łatwo udowodnić przez sprowadzenie do niego *problemu sumy podzbioru* (patrz książka [23]):

> Mając dany zbiór liczb $X = \{b_1, \ldots, b_n\}$ oraz liczbę S, sprawdzić, czy istnieje podzbiór zbioru X sumujący się do S.

Testy

Rozwiązania zawodników były sprawdzane na 12 zestawach testowych. W większości testów ciąg a składa się głównie z niewielkich wartości, a ciąg b jest losowy. Rozwiązania wykładnicze, które sprawdzają wszystkie możliwe wyniki (patrz pliki `hurs5.cpp` i `hurs6.pas`), przechodziły maksymalnie dwa pierwsze testy.

Nazwa	n
hur1a.in	10
hur1b.in	1
hur1c.in	10
hur2.in	20
hur3.in	80
hur4.in	290

Nazwa	n
hur5.in	742
hur6.in	1 000
hur7.in	9 200
hur8.in	25 400
hur9.in	52 023

Nazwa	n
hur10a.in	85 000
hur10b.in	85 000
hur11a.in	140 123
hur11b.in	140 123
hur12a.in	250 000
hur12b.in	250 000