Tomasz Syposz Treść zadania
Jakub Radoszewski Opracowanie
Aleksander Łukasiewicz Program

Dostępna pamięć: 32 MB. *OI, etap II, dzień pierwszy, 10.02.2016*

Zająknięcia

Bitek zapadł ostatnio na dziwną chorobę: strasznie się jąka, a przy tym jedyne słowa, które wypowiada, to liczby. Jego starszy brat, Bajtek, zauważył jednak dziwną powtarzalność w zająknięciach Bitka. Podejrzewa, że Bitek tak naprawdę udaje, żeby nie chodzić do szkoły i móc więcej grać na komputerze. Bajtek nie może przez to uczyć się programowania i jest z tego powodu bardzo smutny. Postanowił więc zdemaskować młodszego brata i liczy, że w nagrodę będzie miał tyle czasu na programowanie, ile dusza zapragnie.

Opiszmy formalnie podejrzenia Bajtka. Załóżmy, że mamy dany ciąg liczb A.

- **Podciągiem** *A nazywamy ciąg powstały przez wyrzucenie z A dowolnych wyrazów, np. $1, 1, 7, 5$ jest podciągiem ciągu $1, 3, 1, 7, 6, 6, 5, 5$.*
- **Zająknięciem** *A nazywamy podciąg A, który składa się z ustawionych po kolei par takich samych wyrazów, np. $1, 1, 1, 1, 3, 3$ jest zająknięciem ciągu $1, 2, 1, 2, 1, 2, 1, 3, 3$.*

Mając dane dwie wypowiedzi Bitka jako ciągi liczb, pomóż Bajtkowi stwierdzić, jaka jest długość najdłuższego zająknięcia, które występuje w każdym z tych ciągów, a nagroda Cię nie ominie.

Wejście

Pierwszy wiersz standardowego wejścia zawiera dwie liczby całkowite n oraz m ($n, m \geqslant 2$) oddzielone pojedynczym odstępem, oznaczające długości ciągów A i B, które reprezentują wypowiedzi Bitka. W drugim wierszu wejścia znajduje się n liczb całkowitych $a_1, a_2, \ldots, a_n$ oddzielonych pojedynczymi odstępami, czyli kolejne wyrazy ciągu A ($1 \leqslant a_i \leqslant 10^9$). W trzecim wierszu wejścia znajduje się m liczb całkowitych $b_1, b_2, \ldots, b_m$ oddzielonych pojedynczymi odstępami, czyli kolejne wyrazy ciągu B ($1 \leqslant b_i \leqslant 10^9$).

Wyjście

Twój program powinien wypisać na standardowe wyjście jedną nieujemną liczbę całkowitą oznaczającą długość najdłuższego wspólnego zająknięcia ciągów A i B. Jeśli ciągi nie mają żadnego wspólnego zająknięcia, poprawnym wynikiem jest 0.

Przykład

Dla danych wejściowych:
```
7 9
1 2 2 3 1 1 1
2 4 2 3 1 2 4 1 1
```

poprawnym wynikiem jest:
```
4
```

Wyjaśnienie do przykładu: *Szukanym ciągiem jest 2, 2, 1, 1.*

Testy „ocen”:

`1ocen`: *$n = 5$, $m = 4$, wszystkie liczby to 42,*

`2ocen`: *$n = 9$, $m = 13$, ciągi to słowa* `OLIMPIADA` *i* `INFORMATYCZNA` *zapisane w kodzie ASCII,*

`3ocen`: *$n = 15\,000$, $m = 15\,000$, ciąg A składa się z par rosnących liczb $(1, 1, 2, 2, 3, 3, \ldots, 7500, 7500)$, natomiast B powstał w wyniku odwrócenia A,*

`4ocen`: *$n = 10\,000$, $m = 5000$, oba ciągi składają się z par naprzemiennych liczb 13 oraz 37 $(13, 37, 13, 37, \ldots)$.*

Ocenianie

Zestaw testów dzieli się na podzadania spełniające poniższe warunki. Testy do każdego podzadania składają się z jednej lub większej liczby osobnych grup testów.

Podzadanie	**Warunki**	**Liczba punktów**
1	$n, m \leqslant 2000$	30
2	$n, m \leqslant 15\,000$ *i każda liczba w każdym ciągu występuje co najwyżej dwa razy*	28
3	$n, m \leqslant 15\,000$	42

Rozwiązanie

W zadaniu dane są dwa ciągi liczb $A = (a_1, \ldots, a_n)$ i $B = (b_1, \ldots, b_m)$. W opisie rozwiązania zamiast o wartościach elementów ciągów wygodniej nam będzie mówić o ich kolorach, tak więc np. a_i oznaczać będzie kolor i-tego elementu ciągu A. Naszym celem jest znaleźć najdłuższe wspólne zająknięcie ciągów A i B, czyli najdłuższy ciąg kolorów złożony z parami powtarzających się elementów, który jest podciągiem każdego z ciągów A i B. Co istotne, w odpowiedzi wystarczy podać długość najdłuższego wspólnego zająknięcia.

Najdłuższy wspólny podciąg

Nasze zadanie ewidentnie ma związek z problemem znajdowania najdłuższego wspólnego podciągu dwóch ciągów. Rozważania zacznijmy więc od przypomnienia klasycznego rozwiązania tego problemu za pomocą programowania dynamicznego (patrz np. książka [6]). Wyznacza się w nim dwuwymiarową tablicę NWP rozmiaru $(n+1) \times (m+1)$, taką że $NWP[i, j]$ oznacza długość najdłuższego wspólnego podciągu ciągów $a_1, \ldots, a_i$ oraz $b_1, \ldots, b_j$. Szukanym wynikiem jest oczywiście $NWP[n, m]$. Mamy następującą zależność rekurencyjną:

$$NWP[i, j] = \begin{cases} 0 & \text{jeżeli } i = 0 \text{ lub } j = 0 \\ NWP[i-1, j-1] + 1 & \text{jeżeli } a_i = b_j \\ \max(NWP[i-1, j], NWP[i, j-1]) & \text{w przeciwnym przypadku.} \end{cases}$$

Z zależności tej wynika następujący algorytm o złożoności $O(nm)$, w którym wypełniamy kolejne pola tablicy $NWP[i,j]$ dla $i = 0,\ldots,n$ oraz $j = 0,\ldots,m$.

```
procedure ObliczNWP
begin
  for j := 0 to m do NWP[0, j] := 0;
  for i := 1 to n do begin
    NWP[i, 0] := 0;
    for j := 1 to m do
      if a[i] = b[j] then
        NWP[i, j] := NWP[i − 1, j − 1] + 1
      else
        NWP[i, j] := max(NWP[i − 1, j], NWP[i, j − 1]);
  end
  return NWP[n, m];
end
```

Warto dodać, że choć złożoność pamięciowa powyższego algorytmu to także $O(nm)$, to można ją łatwo zredukować do $O(n+m)$. Wystarczy mianowicie pamiętać tylko dwa ostatnie wiersze tablicy, tj. $NWP[i-1,\star]$ oraz $NWP[i,\star]$. W tym celu można po prostu we wszystkich odwołaniach do pól tablicy NWP w powyższym pseudokodzie na pierwszej współrzędnej brać resztę z dzielenia przez 2.

Pierwsze rozwiązanie

Nasze pierwsze rozwiązanie będzie naśladować opisaną wyżej metodę. Niech $NWZ[i,j]$ oznacza długość najdłuższego wspólnego zająknięcia ciągów $a_1,\ldots,a_i$ oraz $b_1,\ldots,b_j$.

Jeśli do wyznaczenia tablicy $NWZ[i,j]$ chcielibyśmy zastosować metodę z powyższego pseudokodu, to przypadki brzegowe oraz przypadek $a_i \neq b_j$ pozostaną bez zmian. Natomiast w sytuacji, gdy $a_i = b_j$, powinniśmy do zająknięcia dołożyć jeszcze jeden element koloru a_i. Odpowiada to wybraniu pary elementów: $a_{i'} = a_i$ dla $i' < i$ oraz $b_{j'} = b_j$ dla $j' < j$, co zwiększa długość zająknięcia o dwa elementy. Może się też okazać, że w tym przypadku któryś z szukanych elementów $a_{i'}$ oraz $b_{j'}$ nie istnieje lub znajduje się bardzo wcześnie w ciągu; wówczas lepiej jest wybrać, podobnie jak w przypadku $a_i \neq b_j$, większą z wartości NWZ dla krótszych fragmentów ciągów.

Sprecyzujmy, że jako indeks i' – jeśli istnieje – najlepiej wybrać indeks wskazujący najbliższy wcześniejszy element o tym samym kolorze co a_i. Rzeczywiście, gdyby w najdłuższym wspólnym zająknięciu w ciągu A występowała jako para kolejnych jednokolorowych elementów para $a_{i'}$ oraz a_i, a istniałby indeks i'' taki że $i' < i'' < i$ oraz $a_{i''} = a_i$, to moglibyśmy równie dobrze zamiast $a_{i'}$ wziąć do zająknięcia element $a_{i''}$. Podobnie rzecz ma się w przypadku ciągu B.

Dla danego indeksu $i \in \{1,\ldots,n\}$ przez $prev_A[i]$ oznaczmy indeks najbliższego wcześniejszego elementu o kolorze a_i w ciągu A. Jeśli element $prev_A[i]$ nie istnieje, przyjmujemy, że $prev_A[i] = 0$. Wprowadźmy też analogiczne oznaczenie $prev_B[j]$ dla elementu b_j w ciągu B. Pozwala nam to zapisać następujący pseudokod wyznaczania tablicy $NWZ[i,j]$.

```
procedure ObliczNWZ
begin
  for j := 0 to m do NWZ[0, j] := 0;
  for i := 1 to n do begin
    NWZ[i, 0] := 0;
    for j := 1 to m do begin
      if a[i] = b[j] and prev_A[i] > 0 and prev_B[j] > 0 then
        NWZ[i, j] := NWZ[prev_A[i] − 1, prev_B[j] − 1] + 2
      else
        NWZ[i, j] := 0;
      NWZ[i, j] := max(NWZ[i, j], NWZ[i − 1, j], NWZ[i, j − 1]);
    end
  end
  return NWZ[n, m];
end
```

Algorytm ten ma złożoność czasową i pamięciową $O(nm)$, pod warunkiem, że będziemy mieli do dyspozycji tablice $prev_A[i]$ oraz $prev_B[j]$. Tablice te można wyznaczyć zupełnie siłowo w czasie $O(n^2 + m^2)$, co było wystarczające w tym zadaniu.

Całe rozwiązanie ma zatem złożoność czasową $O((n + m)^2)$ i pamięciową $O(nm)$. Przykładowe implementacje można znaleźć w plikach `zajb2.cpp`, `zajb3.cpp` i `zajb7.pas`. Tego typu rozwiązania przechodziły pierwsze podzadanie, natomiast w pozostałych podzadaniach przekraczały limit pamięciowy. Rzeczywiście, dla maksymalnych wartości n i m z zadania (tj. 15 000) tablica *NWZ* musiałaby mieć $(15\,000)^2 = 225\,000\,000$ komórek, co nie ma możliwości zmieścić się w 32 MB pamięci.

Dodajmy jeszcze, że implementację rozwiązania o złożoności czasowej $O(n^2m^2)$, które nie zapamiętuje wartości tablic $prev_A[i]$ oraz $prev_B[j]$, tylko każdorazowo sprawdza wszystkich kandydatów na elementy $a_{i'}$ i $b_{j'}$, można znaleźć w pliku `zajb4.cpp`.

Rozwiązanie wzorcowe

W naszym zadaniu nie jest niestety tak łatwo zmniejszyć złożoność pamięciową rozwiązania jak w przypadku problemu najdłuższego wspólnego podciągu. Moglibyśmy zastosować sztuczkę z traktowaniem pierwszej współrzędnej modulo 2, gdyby nie konieczność odwoływania się do wartości $NWZ[prev_A[i] - 1, prev_B[j] - 1]$, która teoretycznie może znajdować się w zupełnie dowolnym miejscu tablicy *NWZ*.

Przyjrzyjmy się jednak dokładniej, które komórki tablicy występują w tych kłopotliwych odwołaniach w poszczególnych momentach obliczeń. Gdy w zewnętrznej pętli rozpatrujemy konkretny indeks i, wartość $prev_A[i]$ jest oczywiście ustalona i wskazuje na wcześniejszy element tego samego koloru co i; niech będzie to kolor c. W tym obrocie pętli interesują nas tylko indeksy $prev_B[j]$ dla j takich, że $b_j = c$. Elementy o tych indeksach w ciągu B mają także kolor c.

Gdy w algorytmie przechodzimy do kolejnych indeksów i, takich że $a_i \neq c$, to interesujące nas indeksy $prev_B[j]$ są zatem zupełnie inne. Natomiast kiedy napotkamy pierwszy indeks $i' > i$, taki że $a_{i'} = c$, będziemy mieli $prev_A[i'] = i$, a interesujące nas wartości $prev_B[j]$ będą znów odpowiadały elementom koloru c.

Wprowadźmy do rozwiązania pomocniczą tablicę jednowymiarową *memo* indeksowaną parametrem j. Zauważmy, że gdybyśmy podczas rozpatrywania indeksu i zapamiętali, jako $memo[j]$, wartości $NWZ[i-1, j-1]$ dla wszystkich indeksów j takich że $b_j = c$, to wówczas, rozpatrując indeks i', moglibyśmy jako $NWZ[prev_A[i']-1, prev_B[j]-1]$ wziąć dokładnie wartość $memo[prev_B[j]]$. Mamy wówczas gwarancję, że obliczenia dla indeksów pomiędzy i a i' nie nadpiszą pól $memo[j]$ dla indeksów j, na których w ciągu B znajdują się elementy koloru c.

Ostatecznie musimy wprowadzić w pseudokodzie stosunkowo niewielkie zmiany.

```
procedure ObliczNWZ2
begin
  for j := 0 to m do NWZ[0, j] := memo[j] := 0;
  for i := 1 to n do begin
    NWZ[i mod 2, 0] := 0;
    for j := 1 to m do begin
      if a[i] = b[j] and prev_A[i] > 0 and prev_B[j] > 0 then
        NWZ[i mod 2, j] := memo[prev_B[j]] + 2
      else
        NWZ[i mod 2, j] := 0;
      NWZ[i mod 2, j] := max(NWZ[i mod 2, j], NWZ[(i − 1) mod 2, j],
                             NWZ[i mod 2, j − 1]);
    end
    for j := 1 to m do
      if a[i] = b[j] then
        memo[j] := NWZ[(i − 1) mod 2, j − 1];
  end
  return NWZ[n mod 2, m];
end
```

Otrzymane rozwiązanie ma ewidentnie złożoność pamięciową $O(n+m)$, a jego złożoność czasowa nie uległa zmianie. Implementację tego typu rozwiązania można znaleźć w plikach `zaj.cpp`, `zaj3.pas`, `zaj4.cpp` i `zaj6.cpp`.

Dodatkowe optymalizacje

W naszych rozwiązaniach tablice $prev_A$ i $prev_B$ wyznaczaliśmy, odpowiednio, w czasie $O(n^2)$ i $O(m^2)$. Programujący w języku C++ mogli obliczyć je efektywniej np. z użyciem kontenera `map`. Faktycznie, przeglądając ciąg A za pomocą indeksu i, dla każdego koloru wystarczy pamiętać w strukturze danych ostatnio napotkany indeks elementu tego koloru. Wówczas $prev_A[i]$ wyznaczamy jako indeks zapamiętany w strukturze danych pod kolorem a_i, a odtąd w strukturze pamiętamy dla tego koloru indeks i. Złożoność pojedynczej operacji na kontenerze `map` to $O(\log n)$, więc cały proces zajmuje czas $O(n \log n)$. Tak samo w czasie $O(m \log m)$ można wyznaczyć elementy tablicy $prev_B[j]$. W identycznej złożoności tablice te można wypełnić także bez użycia wspomnianego kontenera – w przypadku tablicy $prev_A[i]$ wystarczy przejrzeć wszystkie pary postaci (a_i, i), posortowawszy je niemalejąco po współrzędnych.

Warto też wspomnieć o pewnym prostym usprawnieniu, które można było zastosować na początku każdego z omawianych rozwiązań. Otóż jeśli elementy jakiegoś koloru występują w którymś z ciągów A, B mniej niż dwukrotnie, to możemy usunąć wszystkie elementy tego koloru z obydwu ciągów. Optymalizację tę łatwo przeprowadzić w czasie $O((n+m)\log(n+m))$. Choć nie zmniejsza ona pesymistycznej złożoności czasowej rozwiązania, w przypadku wielu typów testów pozwala istotnie zmniejszyć długość ciągów.

Rozwiązanie drugiego podzadania

W drugim podzadaniu mieliśmy gwarancję, że w obu ciągach każdy z kolorów występuje co najwyżej dwukrotnie. Przy tym założeniu zadanie można rozwiązać w inny sposób, stosując metodę programowania dynamicznego z liniową liczbą stanów.

Dla każdego koloru elementu, który w każdym z ciągów występuje dwukrotnie (pozostałe kolory możemy w ogóle odrzucić na podstawie opisanej powyżej optymalizacji), znajdujemy indeks i późniejszego wystąpienia elementu tego koloru w ciągu A i zapamiętujemy dla niego indeks $odp[i]$ późniejszego wystąpienia elementu tego koloru w ciągu B. Pozostałe pola tablicy odp inicjujemy zerami. Dla każdego indeksu i w ciągu A wyznaczymy, w tablicy $NWZ'[i]$, długość najdłuższego wspólnego zająknięcia ciągów A i B, które kończy się w ciągu A elementem a_i. Widzimy, że $NWZ'[i] > 0$ tylko dla indeksów i takich że $odp[i] > 0$.

Aby obliczyć $NWZ'[i]$, wystarczy przejrzeć wszystkie wcześniejsze pozycje j i sprawdzić, czy kończące się na nich najdłuższe wspólne zająknięcia (jeśli istnieją) można przedłużyć o elementy znajdujące się pod indeksami $prev_A[i]$, i w ciągu A oraz te pod indeksami $prev_B[odp[i]]$, $odp[i]$ w ciągu B. W ten sposób uzyskujemy poniższy pseudokod.

```
procedure ObliczNWZ'
begin
  for i := 0 to n do begin
    NWZ'[i] := 0;
    if odp[i] > 0 then
      for j := 0 to prev_A[i] − 1 do
        if odp[j] < prev_B[odp[i]] then
          NWZ'[i] := max(NWZ'[i], NWZ'[j] + 2);
  end
  return max{NWZ'[1], ..., NWZ'[n]};
end
```

Indeksy $odp[i]$ można wyznaczyć siłowo; nie będziemy się na ten temat szczegółowo rozpisywać. Otrzymane rozwiązanie ma złożoność czasową $O((n+m)^2)$ i pamięciową $O(n+m)$. Przykładową implementację można znaleźć w pliku `zajb1.cpp`.

Dodajmy na koniec, że podzadanie 2 można także rozwiązać efektywniej, bo w czasie $O((n+m)\log(n+m))$. Rozwiązanie to wykorzystuje drzewo przedziałowe i jest podobne do rozwiązania problemu najdłuższego wspólnego podciągu w przypadku, gdy każdy kolor występuje w każdym z ciągów co najwyżej raz. Dopracowanie jego szczegółów pozostawiamy Czytelnikowi.