ZADANIE #67 – NAJDŁUŻSZY NIEMALEJĄCY CIĄG

Dla ciągu liczb naturalnych na standardowym wejściu, wypisz na standardowe wyjście długość najdłuższego niemalejącego podciągu liczb.

Dane wejściowe

Pierwsza linia zawiera jedną liczbę `N`, która wyznacza liczbę ciągów liczb do przetworzenia.

W każdej z `N` kolejnych linii znajduje się do `K` liczb, oddzielonych od siebie pojedynczym znakiem spacji.

Założenia

- `N` – liczba naturalna z zakresu `<1, 5>`
- `K` – liczba naturalna z zakresu `<1, 1000>`
- Zakres wartości każdej liczby naturalnej w każdej z `K` linii: `<1, 100>`

Dane wyjściowe

Dla każdego wejściowego ciągu liczb na wyjściu powinna znaleźć się jedna linia z jedną liczbą. Ta liczba to długość najdłuższego niemalejącego podciągu znalezionego we wczytanym ciągu liczb.

Przykładowe dane i spodziewany wynik

Dla danych:

```
5
1 2 3 4 5 6 7 8 9 10
5 4 3 2 1
10 10 30 10 10 20
1 2 1 2 3 4 1 2 3
100
```

Spodziewanym wynikiem jest:

```
10
1
3
4
1
```

Opis wyniku

Zgodnie z pierwszą linią danych wejściowych, mamy spodziewać się pięciu ciągów liczb:

- Pierwsza linia z wynikami zawiera liczbę `10`, ponieważ ciąg wejściowych danych zawiera dziesięć następujących po sobie niemalejących liczb. W tym przypadku znaleziony najdłuższy podciąg liczb jest po prostu całym ciągiem danych wejściowych.
- Druga linia z wynikami zawiera liczbę `1`, ponieważ nie możemy znaleźć dłuższego podciągu, który byłyby niemalejący, niż zawierającego jedną liczbę.
- Trzecia linia z wynikami zawiera liczbę `3`, ponieważ najdłuższy niemalejący podciąg ma trzy liczby. W trzecim zestawie danych są dwa takie podciągi: `10`, `10`, `30` oraz `10`, `10`, `20`.
- Czwarta linia z wynikami zwiera liczbę `4`, ponieważ najdłuższy niemalejący podciąg liczb, jaki możemy znaleźć w czwartym zestawie danych, to `1`, `2`, `3`, `4`.
- W ostatniej, piątej linii z wynikami, znajduje się liczba `1`, bo najdłuższy podciąg w ciągu składającym się z jednej liczby ma długość jeden.

Rozwiązanie

Zanim spojrzymy na rozwiązanie, zastanówmy się jak sprawdzić, czy dany ciąg liczb jest niemalejący. Jeżeli porównamy dwie kolejne liczby i ta druga jest nie mniejsza niż pierwsza, to te dwie liczby stanowią podciąg liczb niemalejących. Długość tego podciągu wynosi `2`, ponieważ zawiera dwie liczby. Kontynuując ten proces, możemy porównać drugą i trzecią liczbę – jeżeli ta trzecia liczba także nie jest mniejsza niż liczba ją poprzedzająca, to nasz podciąg ma już trzy elementy. Możemy kontynuować w ten sposób, aż nie znajdziemy takich kolejnych dwóch liczb, gdzie ta druga będzie mniejsza od liczby ją poprzedzającej. Rozważmy ten algorytm dla następującego ciągu: `1`, `10`, `10`, `15`, `4`, `6`, `8`.

W tym ciągu liczb najdłuższym podciągiem niemalejących liczb jest `1`, `10`, `10`, `15`. Jeżeli taki byłby zestaw danych wejściowych, to wynikiem powinna być liczba `4`, ponieważ najdłuższy podciąg niemalejących liczb ma cztery elementy.

To już prawie wszystko, co będzie nam potrzebne do implementacji rozwiązania – musimy tylko jeszcze pamiętać, żeby gdzieś zapisać, jaka była długość znalezionego najdłuższego podciągu liczb niemalejących, oraz aby „zerować” aktualnie rozważaną długość kolejnego podciągu, gdy znajdziemy liczbę, która spowoduje zakończenie aktualnie rozważanego podciągu.

Spójrzmy na proponowane rozwiązanie tego zadania. Tak, jak w poprzednich zadaniach, korzystamy z klas `BufferedReader` i `BufferedWriter`. Opisałem je przy okazji omawiania pierwszego zadania w tym rozdziale.

W pierwszej kolejności wczytujemy liczbę zestawów danych, w linii (1), a następnie rozpoczynamy przetwarzanie danych w linii (2). Dane będziemy odczytywać ze standardowego wejścia linia po linii, co czynimy w linii (3). Odczytane dane dzielimy na poszczególne elementy, które odseparowane są od siebie znakiem spacji. W wyniku otrzymuje tablicę z wartościami, które mamy przetworzyć. Tę tablicę przypisujemy do zmiennej `liczbyJakoTekst`:

Plik: 11_algorytmy/zadanie67/NajdluzszyNiemalejacyCiag.java

```
import java.io.*;

public class NajdluzszyNiemalejacyCiag {
  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(
        new InputStreamReader(System.in)
    );
    BufferedWriter bw = new BufferedWriter(
        new OutputStreamWriter(System.out)
    );

    int liczbaZestawowWejsciowych =
        Integer.parseInt(br.readLine()); // 1

    for (int i = 0; i < liczbaZestawowWejsciowych; i++) { // 2
      String[] liczbyJakoTekst = br.readLine().split(" "); // 3

      int aktualnaDlugosc = 1; // 4
      int znalezionaNajwiekszaDlugosc = 1;
      int poprzedniaLiczba = Integer.parseInt(liczbyJakoTekst[0]);

      for (int k = 1; k < liczbyJakoTekst.length; k++) { // 5
        // 6
        int aktualnaLiczba = Integer.parseInt(liczbyJakoTekst[k]);

        if (poprzedniaLiczba <= aktualnaLiczba) { // 7
          aktualnaDlugosc++; // 8
        } else { // 9
          znalezionaNajwiekszaDlugosc = Math.max(
              znalezionaNajwiekszaDlugosc, aktualnaDlugosc
          );
          aktualnaDlugosc = 1;
        }
```

```
            poprzedniaLiczba = aktualnaLiczba; // 10
        }

        bw.write(String.valueOf( // 11
            Math.max(znalezionaNajwiekszaDlugosc, aktualnaDlugosc)
        ));
        bw.newLine();
    }

    br.close();
    bw.close();
  }
}
```

Następnie, zaczynając w linii `(4)`, definiujemy trzy pomocnicze zmienne:

- `aktualnaDlugosc` – ta zmienna będzie przechowywała długość aktualnie rozważanego podciągu w danych wejściowych,
- `znalezionaNajwiekszaDlugosc` – ta zmienna będzie przechowywała długość najdłuższego znalezionego do tej pory niemalejącego podciągu liczb w danych wejściowych,
- `poprzedniaLiczba` – ta zmienna będzie zawsze przechowywać poprzednią liczbę, do której będziemy porównywać liczbę, która po niej następuje, aby ustalić, czy podciąg jest nadal niemalejący lub czy natrafiliśmy na koniec aktualnego podciągu. Tę zmienną inicjalizujemy pierwszą liczbą z wejściowego ciągu liczb, ponieważ jako jedyna nie ma poprzedniczki. Z racji tego, że wartości w tablicy `liczbyJakoTekst` są typu `String`, musimy zamienić wartość na liczbę typu `int` korzystając z metody `Integer.parseInt`.

Kolejnym krokiem jest przeprocesowanie danych z wejściowego ciągu liczb w pętli zaczynającej się w linii `(5)`. Zauważ, że zmienną `k` inicjalizujemy liczbą `1`, a nie `0`, bo chcemy rozpocząć od porównywania drugiej liczby do pierwszej – gdybyśmy zaczęli od początku, to nie mielibyśmy poprzedniej liczby dla pierwszego elementu tablicy, bo żaden element go nie poprzedza.

W linii `(6)` przypisujemy do pomocniczej zmiennej aktualną liczbę.

Następnie, w linii `(7)`, sprawdzamy, czy poprzednia liczba jest mniejsza bądź równa aktualnie rozważanej liczbie. Jeżeli tak, oznacza to, że te dwie liczby formują niemalejący podciąg, więc należy zwiększyć długość aktualnie rozważanego podciągu o `1`, co robimy w linii `(8)`.

```
for (int k = 1; k < liczbyJakoTekst.length; k++) { // 5
  int aktualnaLiczba = Integer.parseInt(liczbyJakoTekst[k]); // 6

  if (poprzedniaLiczba <= aktualnaLiczba) { // 7
    aktualnaDlugosc++; // 8
  } else { // 9
    znalezionaNajwiekszaDlugosc = Math.max(
        znalezionaNajwiekszaDlugosc, aktualnaDlugosc
    );
    aktualnaDlugosc = 1;
  }

  poprzedniaLiczba = aktualnaLiczba; // 10
}
```

Jeżeli jednak aktualna liczba jest mniejsza, niż poprzedzająca ją liczba, którą to sytuację obsługujemy w sekcji `else` w linii `(9)`, to musimy wykonać dwie operacje:

- Jeżeli długość aktualnie rozważanego ciągu jest większa, niż dotychczasowo znaleziona i zapamiętana w zmiennej `znalezionNajwiekszaDlugosc`, to przypiszemy do tej zmiennej tę większą długość. Jeżeli jednak aktualna długość jest mniejsza od długości znalezionego wcześniej niemalejącego podciągu, to w zmiennej `znalezionaNajwiekszaDlugosc` pozostawimy jej dotychczasową wartość. Korzystamy w tym celu z metody `Math.max`, która zwraca największy ze swoich argumentów.
- „Zerujemy" wartość zmiennej służącej do przetrzymywania długości aktualnie rozważanego podciągu, przypisując zmiennej `aktualnaDlugosc` wartość `1`. W kolejnym obiegu pętli rozpoczniemy szukanie kolejnego niemalejącego ciągu.

Na końcu pętli, w linii `(10)`, przypisujemy do zmiennej `poprzedniaLiczba` wartość aktualnej liczby. Ta aktualna liczba w następnym obiegu pętli posłuży za poprzednią liczbę.

Po przejściu przez wszystkie liczby możemy wypisać wynik na standardowe wyjście:

```
bw.write(String.valueOf( // 11
    Math.max(znalezionaNajwiekszaDlugosc, aktualnaDlugosc)
));
```

Zauważ, że ponownie korzystamy z metody `Math.max`. Jest to wymagane w szczególnym przypadku, gdy wejściowy ciąg liczb jest w całości niemalejącym ciągiem, na przykład `1`, `2`, `3`, `4`, `5`. W takim przypadku warunek z linii `(7)` zawsze będzie spełniony

i nigdy nie wejdziemy do sekcji `else` w linii `(9)`, a tym samym nie przypiszemy do zmiennej `znalezionNajwiekszaDlugosc` długości najdłuższego niemalejącego ciągu. Dodatkowe użycie `Math.max` w linii `(11)` ma na celu obsłużenie właśnie tego konkretnego przypadku.

Testy rozwiązania tego zadania znajdziesz w klasie `NajdluzszyNiemalejacyCiagTest` w katalogu `11_algorytmy/zadanie67` w repozytorium z rozwiązaniami. Podczas omawiania rozwiązania pierwszego zadania w tym rozdziale opisałem, jak możesz użyć tej klasy do przetestowania własnego rozwiązania.