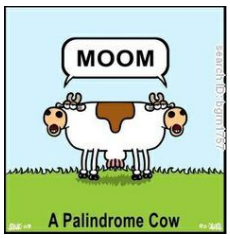
Taneczny palindrom



Wskazówki do rozwiązania zadania

Wczytujemy informacje o kostiumach do tablicy k na pozycje $i = 1, 2, \ldots, n$. Cenę do zapłacenia za zakupienie nowych kostiumów oznaczymy przez *price* (na początku równa 0).

Będziemy przeglądać komórki tej tablicy parami: $k[i]$ oraz $k[j]$, gdzie $i = 1, 2, 3, \ldots$, zaś $j = n, n-1, n-2, \ldots$. Mamy tutaj kilka możliwości:

- $k[i] = 0$, $k[j] \neq 0$ – musimy dokupić taki kostium, jaki ma tancerz o numerze j, czyli *price* powiększamy o jego cenę;
- $k[i] \neq 0$, $k[j] = 0$ – musimy dokupić taki kostium, jaki ma tancerz o numerze i, czyli *price* powiększamy o jego cenę;
- $k[i] = k[j] \neq 0$ – nie trzeba nic kupować;
- $k[i] = k[j] = 0$ – dokupujemy dwie sztuki tańszego kostiumu;
- $k[i] \neq k[j]$, przy czym $k[i], k[j] \neq 0$ – kostiumów nie da się dobrać (piszemy „NIE” i kończymy program).

Sprawdzanie parami kontynuujemy dla $i < j$. Jeśli pętla obejdzie spokojnie do końca, wtedy dobranie kostiumów dla całego zespołu jest możliwe. Trzeba tylko uwzględnić sytuację, kiedy n jest nieparzyste i mamy dodatkową osobę na środku, czyli na pozycji $m = \lceil n/2 \rceil = \lfloor (n+1)/2 \rfloor$. Jeśli $k[m] \neq 0$, nie trzeba nic dokupywać, w przeciwnym razie dokupujemy tańszy kostium.

Wynikiem programu jest wartość zmiennej *price*.

Kod źródłowy rozwiązania w języku C++

```
#include <iostream>
#include <algorithm>
using namespace std;

int main()
{
  int k[1001], n, a, b;
  cin >> n >> a >> b;
  for(int i = 1; i <= n; i++)
    cin >> k[i];
  int price = 0;
  if(n % 2 && k[(n + 1) / 2] == 0)
```

```
    price = min(a, b);
  int i = 1, j = n;
  while(i < j)
  {
    if(k[i] && k[j])
    {
      if(k[i] != k[j])
      { cout << "NIE" << endl; return 0;}
      else continue;
    }
    if(k[i] == 0 && k[j] == 0)
      price += 2 * min(a, b);
    else if(k[i] == 1 || k[j] == 1)
      price += a;
    else
      price += b;
    i++; j--;
  }
  cout << price << endl;
  return 0;
}
```

Uwagi

Rozmiar tablicy $k[\,]$ wynosi o 1 więcej niż maksymalny rozmiar danych n. Dokładna dyskusja tego problemu znajduje się w scenariuszu do zajęć z zadaniem *Plan lekcji*.

W kodzie programu wykorzystujemy fakt, że liczba 0 występująca jako wartość logiczna jest traktowana tak samo jak `false`, zaś liczba różna od zera jest traktowana jak `true`.

Nie jest konieczne używanie dwóch indeksów i oraz j w celu sprawdzenia, czy układ jest (może stać się) palindromem, gdyż dla danego i wartość j wynosi zawsze $n - i + 1$. Jednak w ten sposób łatwiej i przejrzyściej skonstruować kod wewnątrz pętli, no i unika się ewentualnego błędu w wyliczeniu wartości drugiego indeksu (łatwo zapomnieć o dodaniu 1).

Użycie instrukcji `return 0` pozwala na szybkie zakończenie działania programu (funkcji `main`), gdy już wiadomo, że nie uda się uzyskać palindromu.

Użycie instrukcji `continue` umożliwia zapisanie kodu wewnątrz pętli bez dodatkowego poziomu nawiasów klamrowych.