

# Tetris Attack

Ostatnimi czasy w Bajtocji bardzo popularną grą stała się lamiglówka „Tetris Attack”. Jej uproszczona wersja ma następującą postać: Gracz otrzymuje do dyspozycji stos, na którym umieszczono  $2n$  elementów (jeden na drugim), oznaczonych  $n$  różnymi symbolami. Przy tym każdym symbolem są oznaczone dokładnie dwa elementy na stosie. Ruch gracza polega na zamianie dwóch sąsiednich elementów miejscami. Jeśli w wyniku zamiany na stosie sąsiadują ze sobą elementy oznaczone identycznymi symbolami, to w „magiczny” sposób znikają, a elementy znajdujące się powyżej spadają w dół (być może powodując kolejne zniknięcia). Celem gracza jest opróżnienie stosu w jak najmniejszej liczbie ruchów.

## Zadanie

Napisz program, który:

- wczyta ze standardowego wejścia opis początkowej zawartości stosu,
- obliczy rozwiązanie wymagające minimalnej liczby ruchów,
- wypisze znalezione rozwiązanie na standardowe wyjście.

## Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna liczba całkowita  $n$ ,  $1 \leq n \leq 50\,000$ . W kolejnych  $2n$  wierszach zapisana jest początkowa zawartość stosu. Wiersz  $i + 1$ -szy zawiera jedną liczbę całkowitą  $a_i$  — symbol elementu znajdującego się na wysokości  $i$  ( $1 \leq a_i \leq n$ ). Każdy symbol występuje na stosie **dokładnie** 2 razy. Na początku żadne dwa identyczne symbole nie występują obok siebie. Ponadto dane testowe są tak dobrane, że istnieje rozwiązanie zawierające nie więcej niż 1 000 000 ruchów.

## Wyjście

Na standardowym wyjściu należy wypisać opis rozwiązania, wymagającego minimalnej liczby ruchów. Pierwszy wiersz powinien zawierać jedną liczbę całkowitą  $m$  — długość najkrótszego rozwiązania. Kolejne  $m$  wierszy powinno zawierać opis rozwiązania, czyli ciąg  $m$  liczb całkowitych  $p_1, \dots, p_m$ , po jednej w każdym wierszu. Wartość  $p_i$  oznacza, że w  $i$ -tym ruchu gracz zdecydował o zamianie elementów, znajdujących się na wysokościach  $p_i$  oraz  $p_i + 1$ .

Jeżeli istnieje wiele rozwiązań, to Twój program powinien wypisać dowolne z nich.

**Przykład**

*Dla danych wejściowych:*

5  
5  
2  
3  
1  
4  
1  
4  
3  
5  
2

*poprawnym wynikiem jest:*

2  
5  
2

*Natomiast dla danych wejściowych:*

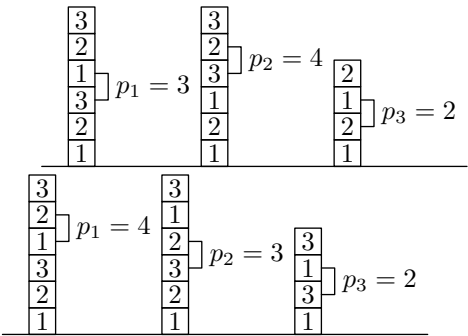
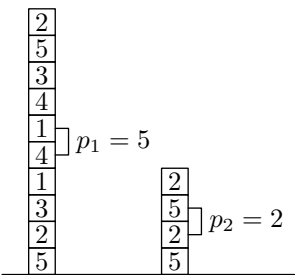
3  
1  
2  
3  
1  
2  
3

*poprawnym wynikiem jest:*

3  
3  
4  
2

*jak również:*

3  
4  
3  
2  
1



**Rozwiązanie**

**O zadaniu**

Nazwa i pomysł zadania pochodzą od gry *Tetris Attack*<sup>1</sup> (gra występuje również pod nazwami *Pokemon Puzzle League* lub *Panel de Pon*). Oczywiście oryginalna gra jest bardziej

<sup>1</sup>Więcej informacji o grze można znaleźć na stronie <http://www.tetrisattack.net/>

skomplikowana: klocki są ułożone w dwóch wymiarach, liczba klocków oznaczonych takimi samymi symbolami nie jest ograniczona itd.

## Podstawowe pojęcia

Rozpocznijmy od wprowadzenia kilku definicji, które ułatwią nam prezentację rozwiązania. Przyjmiemy konwencję, zgodnie z którą stos będziemy zapisywać od elementu na dole stosu do elementu na szczycie; na przykład ciąg: 1 3 2 1 2 3 oznacza stos, w którym element 1 znajduje się na samym dole, natomiast 3 — na szczycie stosu.

Symbole odróżniające klocki będziemy nazywać *kolorami*, zgodnie z konwencją przyjętą w oryginalnych grach.

Powiemy, że para kolorów  $(a, b)$  tworzy *inwersję*, jeśli klocki w kolorach  $a, b$  występują na stosie w następującej kolejności:

$$\dots a \dots b \dots a \dots b \dots$$

Przykładowo na stosie:

$$1\ 2\ 3\ 2\ 1\ 3$$

występują inwersje  $(1, 3)$  oraz  $(2, 3)$ .

Układ na stosie nazwiemy *stabilnym*, jeśli na sąsiednich pozycjach nie występują jednakowe kolory.

W stabilnym układzie o  $n$  kolorach możemy mieć od 1 do  $n(n-1)/2$  inwersji. Najmniejsza liczba inwersji występuje w układzie:

$$1\ 2\ 3\ \dots\ n-2\ n\ n-1\ n\ n-1\ n-2\ \dots\ 2\ 1$$

natomiast największą liczbę inwersji ma układ:

$$1\ 2\ 3\ \dots\ n-1\ n\ 1\ 2\ 3\ \dots\ n-1\ n$$

## Rozwiązanie wzorcowe

Dla rozwiązania zadania kluczowe są następujące spostrzeżenia:

- pojedynczy ruch zmniejsza liczbę inwersji co najwyżej o 1;
- jeśli stos zawiera układ stabilny i nie jest pusty, to zawsze można wykonać ruch, który zmniejszy liczbę inwersji o 1;
- wykonując tylko ruchy zmniejszające liczbę inwersji o 1, otrzymamy rozwiązanie optymalne, czyli wymagające minimalnej liczby ruchów.

Pierwsze, a zatem także i trzecie z powyższych stwierdzeń, są oczywiste. Pozostaje wykazać stwierdzenie drugie — poniżej udowodnimy twierdzenie, w którym pokazujemy, jak znajdować ruchy zmniejszające liczbę inwersji. Twierdzenie to stanie się podstawą przedstawionego dalej algorytmu rozwiązania wzorcowego.

**Twierdzenie 1** Niech  $a_1, \dots, a_{2n}$  oznacza stos elementów, w którym każdy z  $n$  kolorów występuje dokładnie 2 razy. Zakładamy, że układ jest stabilny, tzn.  $a_i \neq a_{i+1}$ , dla  $1 \leq i < 2n$ . Istnieje ruch, którego wykonanie powoduje zmniejszenie liczby inwersji o 1.

**Dowód** W większości wypadków istnieje wiele ruchów o takiej własności, jednak my wyznaczymy ruch, którego wykonanie pozwoli na efektywne rozwiązanie zadania.

Niech  $j$  oznacza maksymalny indeks  $1 \leq j < 2n$  taki, że każdy z elementów  $a_1, \dots, a_j$  ma inny kolor. Oczywiście w ciągu  $a_1, \dots, a_{j+1}$  jakiś kolor musi się już powtórzyć, czyli istnieje indeks  $1 \leq i \leq j$  taki, że  $a_i = a_{j+1}$ .

Łatwo zauważyć, że kolory  $a$  oraz  $b$  występujące odpowiednio na pozycjach  $a_i$  oraz  $a_j$  tworzą inwersję:

kolor:	...	$a$	...	$b$	$a$	...	$b$	...
indeks:		$i$		$j$	$j+1$		$> j+1$	

Możemy ją zlikwidować, zamieniając sąsiednie elementy o indeksach  $j$  oraz  $j+1$ . ■

W rozwiązaniu wzorcowym będziemy kolejno eliminować inwersje, w których występuje pierwszy powtarzający się kolor — jak w dowodzie twierdzenia. Aby efektywnie przeprowadzić tę procedurę, stworzymy dwa stosy, pomiędzy które rozdzielimy zadane elementy:

- $D$  — stos ten będzie zawierał początkowe, zbadane elementy układu, wśród których nie występują powtarzające się kolory — początkowo stos ten jest pusty.
- $S$  — stos ten będzie zawierał resztę elementów tworzących aktualny układ — początkowo stos ten zawiera wszystkie elementy, ułożone w kolejności odwrotnej do wejściowej, tzn.  $a_1$  znajduje się na szczycie stosu  $S$ .

W trakcie algorytmu analizujemy kolejno elementy  $x$  pobierane ze stosu  $S$ . Jeśli kolor elementu  $x$  nie występuje jeszcze w  $D$ , to odkładamy element  $x$  na szczyt stosu  $D$ . W przeciwnym przypadku możemy zastosować twierdzenie, wykonując ruch dla  $x$  i jego poprzednika, likwidując w ten sposób jedną inwersję. W kolejnych ruchach eliminujemy kolejne inwersje, w których występuje kolor  $x$ , przesuwając ten element w głąb stosu  $D$  (w rzeczywistości będziemy przekładać elementy ze stosu  $D$  do stosu  $S$ ), aż do momentu, gdy spotka on swoją „parę” i zniknie z układu. Warto zauważyć, że wykonywane przy okazji przesuwania  $x$  zamiany są (podobnie jak pierwsza) również zamianami „pierwszego powtarzającego się koloru” i każda z nich powoduje zmniejszenie liczby inwersji o 1. Opisane postępowanie jest realizowane w poniższym algorytmie:

```

1:  $D := \emptyset$ ;
2:  $S := \{ \text{stos elementów w kolejności } a_{2n}, a_{2n-1}, \dots, a_1 \}$ ;
3: while  $S \neq \emptyset$  do
4:   begin
5:      $x := S.Pop$ ; { kolejny element z  $S$  }
6:     if  $x \notin D$  then { nowy kolor, więc dodajemy do  $D$  }
7:        $D.Push(x)$ ;
8:     else { kolor  $x$  już występuje w  $D$  }
9:       begin
```

```

10:    $j := \text{LiczbaElementów}(D);$ 
11:    $y := D.Pop;$ 
12:   if  $x \neq y$  then { jeśli kolory są różne, to wykonujemy zamianę }
13:   begin
14:     { print: zamień  $(j, j+1);$  }
15:      $S.Push(y); S.Push(x);$  { odłóż elementy  $x$  i  $y$  z powrotem na stos  $S$  }
16:   end
17: end
18: end

```

Procedury *Pop* i *Push* powodują odpowiednio pobranie elementu ze szczytu stosu i włożenie elementu na szczyt stosu.

Czas działania powyższego algorytmu wynosi  $O(n+k)$ , gdzie  $k$  to liczba inwersji w ciągu podanym na wejściu — przypomnijmy, że  $k$  może wynosić od  $O(1)$  do  $O(n^2)$ . Jest to jednak algorytm o bardzo dobrych parametrach, ponieważ składnika  $k$  w złożoności i tak uniknąć się nie da — jest to rozmiar informacji, które trzeba wypisać jako wynik działania algorytmu.

## Inne rozwiązania

Istnieje wiele rozwiązań opartych na schemacie:

```

1: while  $\text{LiczbaInwersji} > 0$  do
2:   begin
3:     wyznacz ruch, który zmniejsza liczbę inwersji o 1;
4:     wykonaj ten ruch;
5:     zaktualizuj zawartość stosu (w szczególności usuń znikające elementy);
6:   end

```

W schemacie tym kryją się dwa potencjalne źródła nieefektywności algorytmu. Po pierwsze, czasochłonne może być „naiwne” poszukiwanie ruchu zmniejszającego liczbę inwersji. Po drugie, nieprzemysłane symulowanie aktualnego stanu stosu, w szczególności usuwanie “znikających elementów”, także może powodować wydłużenie obliczeń. Częściowo niedogodności te można zmniejszyć, odpowiednio dobierając kolejne ruchy, na przykład poszukując ich możliwie blisko szczytu stosu. Niestety są to tylko heurystyki i w pesymistycznym przypadku bazujące na nich rozwiązania mają jednak złożoność  $O(n^2 + k)$ .

## Testy

Poniższa tabelka przedstawia zestawienie testów użytych do oceny rozwiązań. Parametr  $w$  oznacza rozmiar optymalnego rozwiązania, a  $n$  — liczbę kolorów w danych wejściowych.

Nazwa	w	n	Opis
<i>tet1.in</i>	78	40	test losowy

Nazwa	w	n	Opis
<i>tet2.in</i>	406	50	test losowy
<i>tet3.in</i>	6 624	200	test losowy
<i>tet4.in</i>	19 956	1 000	test losowy
<i>tet5.in</i>	135 926	1 200	test losowy
<i>tet6.in</i>	663 589	4 000	test losowy
<i>tet7.in</i>	657 197	4 500	test losowy
<i>tet8.in</i>	998 360	5 000	test losowy
<i>tet9.in</i>	999 986	5 000	test losowy
<i>tet10a.in</i>	999 614	5 000	test losowy
<i>tet10b.in</i>	1 000 000	2 000	test z największą odpowiedzią
<i>tet11a.in</i>	830 998	42 000	test specjalny
<i>tet11b.in</i>	998 991	10 000	test specjalny
<i>tet12.in</i>	592 528	50 000	test specjalny
<i>tet13.in</i>	759 921	50 000	test specjalny
<i>tet14.in</i>	472 199	50 000	test specjalny
<i>tet15a.in</i>	905 241	50 000	test specjalny
<i>tet15b.in</i>	1	50 000	maksymalny test z odpowiedzią 1

Testy specjalne to losowe testy uzupełnione o duży fragment typu  $1\ 2\ \dots\ k\ \dots\ k\ \dots\ 2\ 1$  oraz wiele (krótkich) fragmentów typu  $1\ 2\ \dots\ p\ 1\ 2\ \dots\ p$ .

# Zawody III stopnia

opracowania zadań

