Zadanie 9 lista 2

Dominik Szczepaniak

```
gdzie-chce = []
gdzie-jest = []
for i in pi:
    gdzie-jest[pi[i]] = i
for i in sigma:
    gdzie-chce[sigma[i]] = i
```

Fakt 1.
Jeśli idąc po kolei w sigma (gdzie poprzednie elementy już naprawiliśmy) napotkamy element który nie jest na swoim miejscu to chce on iść gdzieś na prawo.
Dowód:
Jeśli naprawiliśmy wszystkie poprzednie elementy, to są one na swoich pozycjach, a obecna pozycja nie jest pozycją na którą chce iść liczba która stoi na tej pozycji, więc musi ona chcieć iść gdzieś na prawo.

Przykład algorytmu:
pi = 1, 5, 2, 3, 4
sigma = 2, 1, 3, 4, 5
1. Idziemy forem po sigmie
Trafiamy na 2:
Na miejscu 2 jest 1
Czy 1 chce się zamienić z 2? Nie, bo 2 stoi dalej niż 1 chce iść.
1 przekieruje 2 na 5.
Czy 5 chce się zamienić z 2? Tak, bo 5 chce iść dalej niż stoi 2, więc jest jej to bez różnicy.
pi = 1, 2, 5, 3, 4
sigma = 2, 1, 3, 4, 5
Czy teraz 1 chce się zamienić z 2? Tak, bo 2 jest na miejscu gdzie 1 chce

dotrzeć.
Skończyły nam się liczby, więc zwiększamy fora.
Mamy:
pi = 2, 1, 5, 3, 4
sigma = 2, 1, 3, 4, 5
Trafiamy na 1:
1 jest na swoim miejscu więc idziemy dalej.
Trafiamy na 3:
Na miejscu 3 stoi 5.
Czy 5 chce się zamienić z 3? Tak, bo 3 stoi bliżej niż chce iść 5.
Zamieniamy
Mamy:
pi = 2, 1, 3, 5, 4
sigma = 2, 1, 3, 4, 5
Idziemy dalej forem:
Trafiamy na 4:
Na miejscu 4 stoi 5.
Czy 5 chce się zamienić z 4?
Tak, bo 4 stoi bliżej miejsca 5 niż obecnie stoi 5.
Zamieniamy 4 i 5.
4 jest na swoim miejscu, dalej for. Trafiamy na 5:
5 stoi na swoim miejscu - idziemy dalej.
Kończy się for - kończy się algorytm.

czyli algos:

```
gdzie-chce = []
gdzie-jest = []
for i in pi:
    gdzie-jest[pi[i]] = i
for i in sigma:
    gdzie-chce[sigma[i]] = i
wynik = 0
for i in sigma:
    if(pi[i] == sigma[i]):
        continue
    obecny = pi[i]
```

```
odwiedzone = stack()
odwiedzone.push(pi[i])
#odwiedzone musi chciec isc w prawo, wynika to z faktu 1
while(!odwiedzone.empty()):
    rozwazamy = odwiedzone.top()
    if(gdzie-chce[rozwazany] >= gdzie-jest[obecny]):
        wynik += abs(gdzie-jest[obecny] - gdzie-jest[rozwazany])
        odwiedzone.pop()
        swap(pi[rozwazany], pi[obecny])
        swap(gdzie-jest[rozwazany], gdzie-jest[obecny])
    else: #jesli rozwazany chce skonczyc gdzies blizej niz jest obec
        odwiedzone.push(gdzie-chce[rozwazany])
print(wynik)
```

Dowód:

1. Czy się skończy?
   Zauważmy, że jedyna sytuacja w której nasz algorytm się nie skończy, to gdy odwiedzone nigdy nie będzie pusty w jakimś momencie.

   Zastanówmy się czy to możliwe.

   Nasz algorytm dla jakiejś liczby X chcę się zamienić z liczbą Y która jest na jego miejscu. Jeśli liczba Y nie chce się zamienić z liczbą X (bo liczba X stoi dalej niż liczba Y musi przejść), to przekierowuje liczbę X do zamiany z jakąś liczbą Z która stoi na polu liczby Y.

   No ale skoro Z stoi na polu liczby Y, to musi stać gdzieś pomiędzy liczbami Y i X (bo pole Y było bliżej niż stoi obecnie X - dlatego Y nie zgodziło się na wymiane). W takim razie możemy pójść sobie indukcyjnie i za każdym razem będziemy skracać przedział między Y i X, aż dojdziemy do sytuacji w której potencjalnie X może przesunąć się tylko jedno miejsce w prawo, ale późniejsze wymiany wszystkie będą przebiegać pomyślnie, ponieważ jakaś liczba którą wcześniej rozważaliśmy wskazała na to pole, więc będzie się chciała wymienić. W takim razie nigdy nie dojdzie do sytuacji, że stack odwiedzone nigdy się nie zwolni.

2. Czy zwraca optymalny wynik? Załóżmy, że mamy jakiś wynik A, który jest lepszy od naszego wyniku B.
   Jeśli wynik A jest lepszy od wyniku B to musiała nastąpić jakaś wymiana między liczbami X i Y w A która była tańsza niż wymiana między liczbami X i Y w B.
   Zauważmy, że wymiana X i Y w B zawsze będzie przybliżać obie liczby do swojego miejsca docelowego. W takim razie każda z liczb Z w B przejdzie dokładnie dystans który dzieli ją między miejscem w którym się znajduje a miejscem docelowym. W takim razie mamy sprzeczność, że istnieje wynik A który jest lepszy od wyniku B.

Jaka złożoność?
Nasza złożoność wynosi O(n). Dlaczego?
Zauważmy, że dla każdej pozycji w forze chcemy umieścić liczbę docelową na to miejsce.
Rozważmy przypadki

1. Liczba X i Y chętnie się ze sobą wymieniają.
   W takim razie dla liczby X wykonujemy tylko jedną operację - więc dla wszystkich liczb wykonamy ich n - stąd złożoność O(n).

2. Liczby X i Y nie chcą się wymienić.
   W tym przypadku musimy użyć kolejnej liczby - Z. Jeśli X i Z chcą się wymienić w takim razie zamienimy je, a później zamienimy X i Y i w ten sposób będziemy mieć gotowe dwie liczby - X oraz Y, a wykonaliśmy tylko dwie zamiany - Z i X oraz X i Y.

Zauważmy, że punkt drugi możemy rozszerzać indukcyjnie, tj. jeśli a1 nie chce się zamienić z a2, oraz a3 (liczba wskazywana przez a2) nie chce się zamienić z liczbą a1, to możemy przejść do indukcji, w której a1 = X oraz a2 = Y i wtedy musimy znaleźć sobie kolejnego Z. Wiemy, że kiedyś się to skończy i wtedy mamy, że X i Z się zamienią, później zamieni się jakieś Y_k, później Y_{k-1}, ..., Y_1, więc zrobimy poprawnie k liczb, czyli dokładnie tyle zamian ile chcieliśmy. Co dowodzi, że algorytm działa liniowo.