Ali je vrsta skladna permutacija druge vrste?

Anže Ozimek

Fakulteta za matematiko in fiziko
Univerza v Ljubljani

anze.ozimek1@gmail.com

4.1.2022

- Kaj je permutacija s skladom?

- Kaj je permutacija s skladom?
- Pravila:

- Kaj je permutacija s skladom?
- Pravila:
  1. Iz vhodne vrste lahko le odvzemamo elemente

- Kaj je permutacija s skladom?
- Pravila:
  1. Iz vhodne vrste lahko le odvzemamo elemente
  2. V skladu lahko uporabljamo vgrajene funkcije za polnenje in odstranjevanje

- Kaj je permutacija s skladom?
- Pravila:
  1. Iz vhodne vrste lahko le odvzemamo elemente
  2. V skladu lahko uporabljamo vgrajene funkcije za polnenje in odstranjevanje
  3. Sklad in vhodna vrsta morata biti na koncu prazna

- Kaj je permutacija s skladom?
- Pravila:
  1. Iz vhodne vrste lahko le odvzemamo elemente
  2. V skladu lahko uporabljamo vgrajene funkcije za polnenje in odstranjevanje
  3. Sklad in vhodna vrsta morata biti na koncu prazna
  4. V izhodno vrsto lahko le dodajamo elemente

- Kaj je permutacija s skladom?
- Pravila:
  1. Iz vhodne vrste lahko le odvzemamo elemente
  2. V skladu lahko uporabljamo vgrajene funkcije za polnenje in odstranjevanje
  3. Sklad in vhodna vrsta morata biti na koncu prazna
  4. V izhodno vrsto lahko le dodajamo elemente

Primer

- Vhodna vrsta: $[1, 2, 3]$

Primer

- Vhodna vrsta: $[1, 2, 3]$
- Izhodna vrsta 1: $[2, 1, 3]$

Primer

- Vhodna vrsta: $[1, 2, 3]$
- Izhodna vrsta 1: $[2, 1, 3] \rightarrow$ True

Primer

- Vhodna vrsta: $[1, 2, 3]$
- Izhodna vrsta 1: $[2, 1, 3] \rightarrow$ True
- Izhodna vrsta 2: $[3, 1, 2]$

Primer

- Vhodna vrsta: $[1, 2, 3]$
- Izhodna vrsta 1: $[2, 1, 3] \rightarrow$ True
- Izhodna vrsta 2: $[3, 1, 2] \rightarrow$ False

Rešitev

- Opazimo, da sklad polnimo, dokler vanj ne vstavimo elementa, ki je začetek permutacije

Rešitev

- Opazimo, da sklad polnimo, dokler vanj ne vstavimo elementa, ki je začetek permutacije
- Primer:
  - Vhodna vrsta: $[2, 4, 3, 1]$
  - Izhodna vrsta: $[3, 1, 4, 2]$

Rešitev

- Opazimo, da sklad polnimo, dokler vanj ne vstavimo elementa, ki je začetek permutacije
- Primer:
  - Vhodna vrsta: $[2, 4, 3, 1]$
  - Izhodna vrsta: $[3, 1, 4, 2]$
  - V sklad dajemo števila vhodne vrste dokler ne naletimo na število 3

Opis algoritma

- Algoritem:

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada
  3. Preverimo ali je naslednji element v skladu enak prvemu v izhodni vrsti in naredimo enako kot v točki 2

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada
  3. Preverimo ali je naslednji element v skladu enak prvemu v izhodni vrsti in naredimo enako kot v točki 2
  4. Če naslednjega elementa v skladu še ni, nadaljujemo s točko 1

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada
  3. Preverimo ali je naslednji element v skladu enak prvemu v izhodni vrsti in naredimo enako kot v točki 2
  4. Če naslednjega elementa v skladu še ni, nadaljujemo s točko 1
- Vhodna vrsta prazna

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada
  3. Preverimo ali je naslednji element v skladu enak prvemu v izhodni vrsti in naredimo enako kot v točki 2
  4. Če naslednjega elementa v skladu še ni, nadaljujemo s točko 1
- Vhodna vrsta prazna
- Sklad ni nujno prazen:

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada
  3. Preverimo ali je naslednji element v skladu enak prvemu v izhodni vrsti in naredimo enako kot v točki 2
  4. Če naslednjega elementa v skladu še ni, nadaljujemo s točko 1
- Vhodna vrsta prazna
- Sklad ni nujno prazen:
  1. Sklad je prazen

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada
  3. Preverimo ali je naslednji element v skladu enak prvemu v izhodni vrsti in naredimo enako kot v točki 2
  4. Če naslednjega elementa v skladu še ni, nadaljujemo s točko 1
- Vhodna vrsta prazna
- Sklad ni nujno prazen:
  1. Sklad je prazen $\rightarrow$ True (permutacija obstaja)

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada
  3. Preverimo ali je naslednji element v skladu enak prvemu v izhodni vrsti in naredimo enako kot v točki 2
  4. Če naslednjega elementa v skladu še ni, nadaljujemo s točko 1
- Vhodna vrsta prazna
- Sklad ni nujno prazen:
  1. Sklad je prazen $\rightarrow$ True (permutacija obstaja)
  2. Sklad ni prazen

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada
  3. Preverimo ali je naslednji element v skladu enak prvemu v izhodni vrsti in naredimo enako kot v točki 2
  4. Če naslednjega elementa v skladu še ni, nadaljujemo s točko 1
- Vhodna vrsta prazna
- Sklad ni nujno prazen:
  1. Sklad je prazen $\rightarrow$ True (permutacija obstaja)
  2. Sklad ni prazen $\rightarrow$ Flase (permutacija ne obstaja)

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada
  3. Preverimo ali je naslednji element v skladu enak prvemu v izhodni vrsti in naredimo enako kot v točki 2
  4. Če naslednjega elementa v skladu še ni, nadaljujemo s točko 1
- Vhodna vrsta prazna
- Sklad ni nujno prazen:
  1. Sklad je prazen $\rightarrow$ True (permutacija obstaja)
  2. Sklad ni prazen $\rightarrow$ Flase (permutacija ne obstaja)
- Kaj pa izhodna vrsta?

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada
  3. Preverimo ali je naslednji element v skladu enak prvemu v izhodni vrsti in naredimo enako kot v točki 2
  4. Če naslednjega elementa v skladu še ni, nadaljujemo s točko 1
- Vhodna vrsta prazna
- Sklad ni nujno prazen:
  1. Sklad je prazen $\rightarrow$ True (permutacija obstaja)
  2. Sklad ni prazen $\rightarrow$ Flase (permutacija ne obstaja)
- Kaj pa izhodna vrsta?
  1. Izhodna vrsta je prazna $\rightarrow$

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada
  3. Preverimo ali je naslednji element v skladu enak prvemu v izhodni vrsti in naredimo enako kot v točki 2
  4. Če naslednjega elementa v skladu še ni, nadaljujemo s točko 1
- Vhodna vrsta prazna
- Sklad ni nujno prazen:
  1. Sklad je prazen $\rightarrow$ True (permutacija obstaja)
  2. Sklad ni prazen $\rightarrow$ Flase (permutacija ne obstaja)
- Kaj pa izhodna vrsta?
  1. Izhodna vrsta je prazna $\rightarrow$ True
  2. izhodna vrsta ni prazna $\rightarrow$

Opis algoritma

- Algoritem:
  1. Sproti gremo po elemtih v vhodni vrsti in jih dajemo v sklad, dokler ne naletimo na prvi element v izhodni vrsti
  2. Ko najdemo prvi element izhodne vrste, ga odstranimo tako iz vrste kot iz sklada
  3. Preverimo ali je naslednji element v skladu enak prvemu v izhodni vrsti in naredimo enako kot v točki 2
  4. Če naslednjega elementa v skladu še ni, nadaljujemo s točko 1
- Vhodna vrsta prazna
- Sklad ni nujno prazen:
  1. Sklad je prazen $\rightarrow$ True (permutacija obstaja)
  2. Sklad ni prazen $\rightarrow$ Flase (permutacija ne obstaja)
- Kaj pa izhodna vrsta?
  1. Izhodna vrsta je prazna $\rightarrow$ True
  2. izhodna vrsta ni prazna $\rightarrow$ False (izhodna vrsta ima več elementov kot vhodna)

Še nekaj primerov

- Vhodna vrsta: [”danes”, ”je”, ”lep”, ”sončen”,”dan”]
- Izhodna vrsta: [”lep”, ”sončen”, ”dan” ,”je” ,”danes”] $\rightarrow$

Še nekaj primerov

- Vhodna vrsta: ["danes", "je", "lep", "sončen","dan"]
- Izhodna vrsta: ["lep", "sončen", "dan" ,"je" ,"danes"] $\rightarrow$ True

Še nekaj primerov

- Vhodna vrsta: [”danes”, ”je”, ”lep”, ”sončen”,”dan”]
- Izhodna vrsta: [”lep”, ”sončen”, ”dan” ,”je” ,”danes”] $\rightarrow$ True
- Vhodna vrsta: [2 , 4, 5, 6, 7, 11]
- Izhodna vrsta: [4 , 11, 2, 7, 5, 6] $\rightarrow$

Še nekaj primerov

- Vhodna vrsta: [”danes”, ”je”, ”lep”, ”sončen”,”dan”]
- Izhodna vrsta: [”lep”, ”sončen”, ”dan” ,”je” ,”danes”] $\rightarrow$ True
- Vhodna vrsta: [2 , 4, 5, 6, 7, 11]
- Izhodna vrsta: [4 , 11, 2, 7, 5, 6] $\rightarrow$ False

Še nekaj primerov

- Vhodna vrsta: ["danes", "je", "lep", "sončen","dan"]
- Izhodna vrsta: ["lep", "sončen", "dan" ,"je" ,"danes"] $\rightarrow$ True
- Vhodna vrsta: [2 , 4, 5, 6, 7, 11]
- Izhodna vrsta: [4 , 11, 2, 7, 5, 6] $\rightarrow$ False
- Vhodna vrsta: [8, 17, 47, 38, 29, 39, 100]
- Izhodna vrsta: [38, 47, 29, 17, 39, 100, 8] $\rightarrow$

Še nekaj primerov

- Vhodna vrsta: ["danes", "je", "lep", "sončen","dan"]
- Izhodna vrsta: ["lep", "sončen", "dan" ,"je" ,"danes"] $\rightarrow$ True
- Vhodna vrsta: [2 , 4, 5, 6, 7, 11]
- Izhodna vrsta: [4 , 11, 2, 7, 5, 6] $\rightarrow$ False
- Vhodna vrsta: [8, 17, 47, 38, 29, 39, 100]
- Izhodna vrsta: [38, 47, 29, 17, 39, 100, 8] $\rightarrow$ True

Koda

```python
from sklad import Sklad
from vrsta import Vrsta


def preveri(vhodna,izhodna):
    '''Funkcija nam vrne True, če se vrsto izhodna da dobiti kot permutacijo s skladom vrste vhodna''
    pomozni_sklad = Sklad()
    while not vhodna.prazna():
        element = vhodna.zacetek()
        vhodna.odstrani()
        if element == izhodna.zacetek():
            izhodna.odstrani()
            while not pomozni_sklad.prazen():
                if pomozni_sklad.vrh() == izhodna.zacetek():
                    pomozni_sklad.odstrani()
                    izhodna.odstrani()
                else:
                    break
        else:
            pomozni_sklad.vstavi(element)
    return vhodna.prazna() and pomozni_sklad.prazen() and izhodna.prazna()

```

Viri

- https://www.geeksforgeeks.org/stack-permutations-check-if-an-array-is-stack-permutation