Steinhouse-Johnson-Trotter algoritam za permutacije

Seminarski rad u okviru kursa
Konstrukcija i analiza algoritama 2
Matematički fakultet

Jovan Ranđelović
mr231088@alas.bg.ac.rs

Februar 2024

Sadržaj

1 Uvod

Steinhouse-Johnson-Trotter algoritam je algoritam za generisanje svih permutacija n elemenata. Originalni algoritam datira iz 1962. godine kad su istraživači otkrili da je moguće generisati svih $n!$ permutacija koristeći $n! - 1$ zamena susednih elemenata. Svake dve susedne permutacije u nizu se razlikuju po tome što su im neka dva susedna elementa zamenila mesta (npr. $1234 \rightarrow 1243$). Ovaj algoritam je zanimljiv i po tome što pronalazi Hamiltonov put 1 u permutoedru.[1]

[1] Permutoedar je poliedar koji ima $n!$ temena koja predstavljaju permutacije elemenata $1, 2, ...n$ 1

Slika 1: Hamiltonov put u Kejlijevom grafu simetrične grupe generisane Steinhouse-Johnson-Trotter algoritmom

2 Algoritam

2.1 Rekurzivna struktura

Niz permutacija n brojeva se može dobiti od niza permutacija $n-1$ brojeva tako što broj n umetnemo na svaku moguću poziciju u svakoj kraćoj permutaciji.

Formiramo $(n-1)!$ blokova i u svakom bloku važi da brojevi $1, ..., n-1$ formiraju rastući ili opadajući niz.

Svaki od tih blokova je dobijen rekurzivno za jedan broj manje. U okviru svakog bloka, pozicija broja n ili raste ili opada dok je u susednom bloku obrnuto. Na primer, u prvom bloku opada, u drugom raste, u trećem opet opada itd.

Za permutaciju jednog elementa imamo:
1
Da bismo dobili sve permutacije 2 elementa, formiramo $(2-1)! = 1$
blok u kome umećemo broj 2 opadajuće (svejedno je da li opadajuće ili rastuće)
1**2**
21

Za $n = 3$ imaćemo $(3-1)! = 2$ bloka velicine 3. U prvom bloku broj 3 umećemo opadajuće, a u drugom rastuće

12**3**	**3**21
1**3**2	2**3**1
312	21**3**

Primetimo da se svake dve susedne permutacije razlikuju samo za transpoziciju neka dva susedna elementa. Takođe, prvi i poslednji element niza se razlikuju za transpoziciju elemenata 1 i 2 što se može dokazati indukcijom.

2.2 Prvobitna verzija algoritma

Generisanje sledeće permutacije brojeva $1, ..., n$ od date permutacije π se odvija na sledeći način:

- Za svako i od 1 do n, neka je x_i pozicija broja i u permutaciji π. Ako poredak brojeva od 1 do $i-1$ čini parnu permutaciju [2] uzimamo $y_i = x_i - 1$, inace $y_i = x_i + 1$.
- Pronađi najveće i za koje y_i definiše validnu poziciju u permutaciji π koja sadrži broj manji od i. Zameni vrednosti na pozicijama x_i i y_i.

Kada više ne postoji i za koje je ispunjen drugi korak algoritma, to znači da smo stigli do završne permutacije i procedura se zaustavlja. Složenost generisanja jedne permutacije je $O(n)$.[3]

Ključna ideja ovog algoritma je zamena susednih elemenata sa ciljem da nova permutacija bude različite parnosti od trenutne.

Neka je tekuća permutacija $\pi = [1, 2, 3]$.

$$\begin{array}{ccc} i = 1 & \text{i=2} & i = 3 \\ x_1 = 1 & x_2 = 2 & x_3 = 3 \\ y_1 = 0 & y_2 = 1 & y_3 = 2 \end{array}$$

Najveće i za koje je ispunjeno $\pi[y_i] < i$ je $i = 3$, stoga vršimo zamenu $\pi[x_3]$ i $\pi[y_3]$ i tako dobijamo sledeću permutaciju $\pi' = [1, 3, 2]$.

2.3 Evenovo ubrzanje

Izraelski informatičar Shimon Even zaslužan je za poboljšanje vremena izvršavanja prethodnog algoritma. Nova ideja je bila da čuvamo dodatnu informaciju za svaki element u permutaciji, smer (nalevo ili nadesno) u kom se element kreće. Na početku, svi elementi imaju smer nalevo. [7]

1. Pronađi najveći po vrednosti pokretni broj. Broj je pokretan ako je veći od svog suseda prema kom pokazuje
2. Zameni pokretni broj sa tim susedom

[2] Za permutaciju kazemo da je parna ukoliko se moze zapisati kao proizvod parnog broja transpozicija

3. Promeni smer svim elementima čija je vrednost veća od trenutnog pokretnog broja

4. Ponavljaj korak 1 sve dok nema više pokretnih brojeva

Razmotrimo slučaj kada je n=3:

$< 1 < 2 < 3$ (sa $<$ i $>$ označavamo smer kretanja elemenata)
Pokretni brojevi su 2 i 3. Najveći među njima je 3. Vrši se zamena 2 i 3:
$< 1 < 3 < 2$

3 je najveći pokretni, menja se sa 1:
$< 3 < 1 < 2$

Sada je 2 najveći pokretni, vršimo zamenu sa 1 i menja se smer svim većim elementima.
$3 >< 2 < 1$

3 je najveći pokretni, vršimo zamenu sa 2:

$< 23 >< 1$
Ponovo je 3 najveći pokretni, menja se sa 1:
$< 2 < 13 >$
Na kraju konstatujemo da više nema pokretnih brojeva i tu je kraj algoritmu.

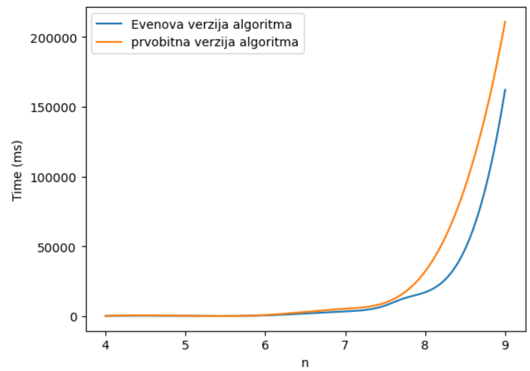
Postoji i nešto složenija verzija algoritma [6] koja ne koristi petlje, pogodna za funkcionalno programiranje i koja osigurava konstantnu vremensku složenost po permutaciji, ali, ispostavlja se da te modifikacije koje uklanjaju petlje na kraju prouzrokuju da je algoritam sporiji nego u iterativnoj verziji.

3 Analiza vremena izvršavanja

Sva testiranja su izvršena na računaru sa procesorom AMD Ryzen 5 4500 6-Core Processor 3.60 GHz. Uspešno je testirano zaključno sa $n = 9$.
Na slici 2 prikazani su grafici dva navedena algoritma. Vidimo da oba grafika jako brzo rastu što je u skladu sa njihovom ocenom slozenosti $O(n \cdot n!)$, ali da poboljšana verzija algoritma ima za nijansu bolje vreme izvršavanja.

4 Implementacija

Implementacija Steinhouse-Johnson-Trotter algoritma u jeziku C++ nalazi se na adresi https://github.com/jocaran/Steinhaus-Johnson-Trotter-algorithm/blob/main/SJT.cpp
U nastavku je dat pseudo kod algoritma



Slika 2: Grafici zavisnosti vremena izvršavanja algoritama (u milisekundama) od veličine ulaza

n	Time (ms)
4	5.275
5	62.289
6	437.32
7	3312.538
8	17032.518
9	162177.775

Slika 3: Leva kolona tabele sadrži dužinu permutacije a desna vreme izvršavanja Evenove verzije algoritma izraženo u milisekundama

```
funkcija SJT(permutacija,smer,n):
pokretni_broj = getPokretni(permutacija,smer,n)
poz = getPozicija(permutacija,n,pokretni_broj)

if(smer[permutacija[poz-1]-1] == NADESNO) then
    zameni(permutacija[poz-1],permutacija[poz-2])

else if (smer[permutacija[poz-1]-1] == NALEVO) then
    zameni(permutacija[poz],permutacija[poz-1])

for i=1 to n do
    if permutacija[i] > pokretni_broj then
        promeni_smer(permutacija[i]-1)
```

Literatura

[1] Sedgewick, Robert (1977), "Permutation generation methods"

[2] Lenstra, J. K.; Rinnooy Kan, A. H. G. (September 1979), "A recursive approach to the implementation of enumerative methods"

[3] Johnson, Selmer M. (1963), "Generation of permutations by adjacent transposition"

[4] Knuth, Donald (2011), "Generating All Permutations", The Art of Computer Programming, volume 4A: Combinatorial Algorithms, Part 1

[5] Even, Shimon (1973), Algorithmic Combinatorics, Macmillan

[6] Bird, Richard (2010), Chapter 29: The Johnson–Trotter algorithm, Pearls of Functional Algorithm Design

[7] Even, Shimon (1973), Algorithmic Combinatorics, Macmillan