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 2023

Sadržaj

1 Uvod

Steinhouse-Johnson-Trotter algoritam je algoritam za generisanje svih permutacija n elemenata. Originalni algoritam datira iz 1962 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. $1234 \to 1243$). Ovaj algoritam je zanimljiv i po tome što pronalazi Hamiltonov put u permutoedru.

2 Algoritam

2.1 Rekurzivna struktura

Niz permutacija n brojeva se moze dobiti od niza permutacija $n-1$ brojeva tako što broj n umetnemo na svaku moguću poziciju u svakoj kraćoj permutaciji. Niz permutacija podelimo na $(n-1)!$ blokova tako da je poredak brojeva od 1

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

do $n-1$ isti i razlikuje se samo na poziciji gde je broj n. 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
Sada možemo staviti 2 u opadajućem poretku recimo i dobijamo:
1**2**
21

Dalje, broj 3 možemo staviti na tri različite pozicije za svaku od ove dve permutacije, u opadajućem poretku za prvu permutaciju 12 pa onda u rastućem poretku za permutaciju 21:
12**3**
1**3**2
312
321
2**3**1
21**3**

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

2.2 Prvobitna verzija algoritma

Generisanje sledeće permutacije od date permutacije π se odvija na sledeći nacin:

- Za svako i od 1 do n, neka je x_i pozicija broja i u permutaciji π. Ako poredak brojeva od 1 do $i-1$ cini parnu permutaciju 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 je $O(n)$ po permutaciji.

2.3 Evenovo ubrzanje

Izraelski informaticar Shimon Even zaslužan je za poboljšanje vremena izvršavanja algoritma. Njegova 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.

1. Pronađi najveći pokretni broj. Broj je pokretan ako je veći od svog suseda prema kom pokazuje
2. Zameni pokretni broj sa tim susedom
3. Promeni smer svim elementima čija vrednost je veća od trenutnog pokretnog broja
4. Ponavljaj korak 1 sve dok nema više pokretnih brojeva

Na primer, n=4:

$< 1 < 2 < 3 < 4$
Pokretni brojevi su 2,3 i 4. Najveći među njima je 4. Zamenimo 3 i 4:
$< 1 < 2 < 4 < 3$

4 je najveći pokretni, zamenimo ga sa 2:
$< 1 < 4 < 2 < 3$

4 je najveći pokretni, zamenimo ga sa 1:

$< 4 < 1 < 2 < 3$

Sada je 3 najveći pokretni, zamenimo ga sa 2 i menjamo smer svim većim elementima.

$4 >< 1 < 3 < 2$

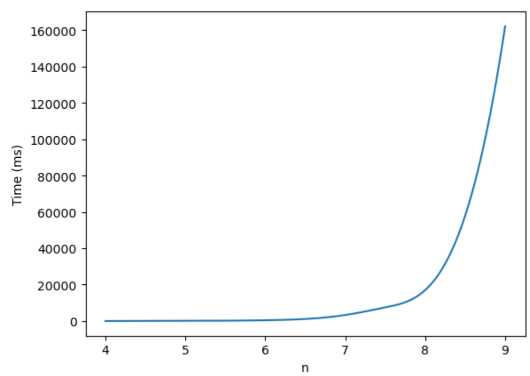
4 je najveći pokretni, zamenimo ga sa 1:

$< 14 >< 3 < 2$
i tako dalje...

Na kraju dobijamo:
$< 2 < 13 > 4 >$
i konstatujemo da više nema pokretnih brojeva.
Složenost algoritma je $O(n \cdot n!)$

Postoji i nešto komplikovanija verzija algoritma 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 Grafički prikaz



Slika 2: Grafik zavisnosti vremena izvršavanja algoritma (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: Tabelarni prikaz

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

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

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