Secvența lui Kolakoski

Rachieru Gheorghe Gabriel
Facultatea de Matematică și Informatică, Grupa 141

March 8, 2025

Această lucrare explorează secvența lui Kolakoski, încercând să răspundă la trei întrebări fundamentale: **Ce este? De ce este importantă? Cum poate fi implementată?**

1. **Ce este secvența lui Kolakoski?**
   Secvența lui Kolakoski este un șir infinit autodescriptiv, format doar din cifrele 1 și 2, în care fiecare termen (sau pereche de termeni) determină lungimea următorului grup de cifre. Secvența începe cu 1, ceea ce indică faptul că următorul termen trebuie să fie 2 și să apară o singură dată. De exemplu:
   1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 2, 1, 2, 2, 1, 1, 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1

2. **Particularități ale secvenței.**
   Acest șir conține doar cifrele 1 și 2, iar distribuția lor pare să fie aproape egală. Totuși, nu se cunoaște dacă, la un moment dat, numărul de 1-uri este exact egal, mai mare sau mai mic decât numărul de 2-uri. Simulările computerizate indică faptul că densitatea acestor cifre tinde spre egalitate.
   În secvența lui Kolakoski apar doar șase tipuri de triplete: 112, 121, 122, 211, 212 și 221. Din această observație rezultă că există doar optsprezece grupuri distincte de șase cifre, care includ atât combinații de două triplete identice, cât și combinații în care ordinea tripletelor este inversată.

   În ceea ce privește distribuția cifrei 1 în secvență, toate aceste grupuri de șase cifre au o densitate de 1/2 pentru cifra 1. De asemenea, atunci când se aplică regulile secvenței Kolakoski asupra unei duble triplete, proporția dintre cifre rămâne aceeași. De exemplu, secvența 112112 se transformă în 12112122, în care raportul dintre 1 și 2 este menținut. Acest lucru se întâmplă deoarece a doua parte a grupului inversează distribuția generată de prima parte.

   Astfel, secvența poate fi privită ca fiind compusă din două tipuri de segmente: unele care, după transformare, păstrează densitatea exactă de 1/2 pentru cifra 1 și altele care deja au această densitate de la început.

3. **Cum poate fi implementată secvența lui Kolakoski?**
Am ales să implementez script-ul pentru generarea șirului în C++, deoarece eram familiarizat cu acesta și, pentru mine cel puțin, este mai intuitiv în a lucra la limite și a măsura performanța.Am ales sa lucrez pe tipul de date int deoarece îi știam destul de bine limitele. O primă abordare în C++ a fost generarea secvenței într-un vector static. Deși acest algoritm funcționa pentru valori mici, am fost interesat să analizez distribuția cifrelor și să obțin statistici detaliate. Astfel, am testat implementarea la limita arhitecturii calculatorului meu.
Totuși, o abordare cu un vector static are limitări legate de dimensiunea memoriei. Citind despre implementări recursive sau cu complexitate logaritmică în memorie, am observat că partea de început a șirului devine inutilă după ce secvența a fost generată. Acest lucru m-a condus la utilizarea unei structuri de coadă pentru optimizarea memoriei.

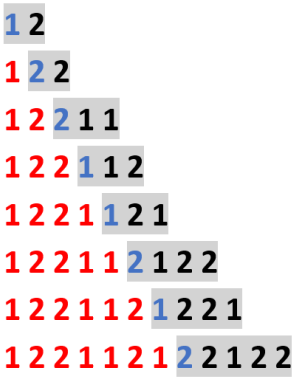

Figure 1: Reprezentarea unei cozi pentru generarea secvenței

M-am intors la algoritmica de clasa a 11-a, cand am scris cod ultima data pentru altceva decat queries în baze de date pentru atestat sau probleme banale de bac, si mi-am adus aminte de o structura de date relevanta, coada din STL. Exact ce aveam nevoie, sa pastrez cel mai din stanga element si sa adaug unele noi in dreapta tuturor ce stau la coada pentru a fi verificate. Am implementat problema cu aceasta noua abordare, dar acum trebuia testata pentru a vedea daca este cu adevarat mai buna decat cea cu un array static. Chiar daca la fiecare pas, primul element din coada primeste pop, in cel mai rau caz, sunt adaugate alte doua elemente, deci nici aceasta abordare nu imi ofera un calcul independent de memorie, pentru a genera la infinit acest sir (daca as avea tot curentul si timpul din lume). Conditia de oprire a acestui algoritm este data de lungimea cozii. Pentru comparatia acestui algoritm cu cel vechi, am introdus acelasi numar n ca lungimea vectorului static si ca lungimea cozii. Am observat ca pentru acelasi n, implementarea cu coada m-a lasat sa generez, aproximativ, de 3 ori mai multe numere din sir. Sa consider asta un succes?. Nefiind la fel de intuitiv ca la un vector, am încercat sa aflu care este numarul maxim de elemente dintr-o astfel de coada. Pentru a fi mai bun decat versiunea anterioara, aceasta valoare ar trebui sa fie macar 2147483647/3=715827882 Inca este un numar mult prea mare, dar o sa sper ca

o coada poate avea mai mult de 715827882 elemente. O alta optimizare interesanta ar fi lucrul cu variabile de tip bool, caci ocupa fix un bit, minunat pentru codificarea de 1 sau 2.

4. **La ce rezultate am ajuns?**
Căutând pe internet rezultate despre generarea șirului, am ajuns la aceste statistici densitatea cifrelor.

n	1s	2s	% 1s	% 2s	% difference
100	49	51	49.0000%	51.0000%	2.0000%
1,000	502	498	50.2000%	49.8000%	0.4000%
10,000	4,996	5,004	49.9600%	50.0400%	0.0800%
100,000	49,972	50,028	49.9720%	50.0280%	0.0560%
1,000,000	499,986	500,014	49.9986%	50.0014%	0.0028%
2,000,000	999,952	1,000,048	49.9976%	50.0024%	0.0048%
3,000,000	1,500,021	1,499,979	50.0007%	49.9993%	0.0014%
4,000,000	1,999,967	2,000,033	49.9992%	50.0008%	0.0017%

Figure 2: Statisticii

Însă, după ce am rulat si eu script-ul aproximativ 3 ore pentru a testa daca implementarea cu coada este mai bună decât cea cu un vector static, am ajuns la următoarele rezultate complementare:

n	1s	2s	%1s	%2s	% Diferența
6.000.000	3.000.065	2.999.935	50.0010%	49.9989%	0.0021%
9.000.000	4.500.007	4.499.993	50.00007%	49.99992%	0.00015%
12.000.000	5.999.964	6.000.036	49.999700%	50.000299%	0.0005%
15.000.000	7.500.181	7.499.819	50.001206%	49.998793%	0.002%
19.192.354	9.096.407	9.095.947	50.00126%	49.99873%	0.002%

Figure 3: Statisticii

Putem observa astfel că densitatea cifrelor tinde spre egalitate, dar nu este atinsă cu siguranță.

5. **Alte abordări mai eficiente.**
Majoritatea algoritmilor funcționează liniar, dar din cauză ca generarea șirului presupune o referință anterioara, trebuie păstrată o bucată din secvență tot timpul, bucată a cărei lungime nu crește liniar. O alternativă cu timp liniar și spațiu logaritmic presupune generarea mai multor copii la viteze diferite. Compute the Kolakoski sequence

Există totuși două formule care definesc complet secvența lui Kolakosi:

$$a(1) = 1, \quad a(2) = 2, \tag{1}$$

$$a\left(\sum_{i=1}^{k} a(i)\right) = \frac{3 + (-1)^k}{2}, \tag{2}$$

$$a\left(\sum_{i=1}^{k} a(i) + 1\right) = \frac{3 - (-1)^k}{2}. \tag{3}$$

6. **Referințe:**

https://github.com/rachdocx/LFA-FMI-2025/tree/main/Kolakoski%20Sequence
https://oeis.org/A000002
https://en.wikipedia.org/wiki/Kolakoski_sequence
https://metatutor.co.uk/the-kolakoski-sequence/
https://codegolf.stackexchange.com/questions/157403/compute-the-kolakoski-sequence
https://cs.uwaterloo.ca/journals/JIS/VOL9/Steinsky/steinsky5.pdf
https://mathworld.wolfram.com/KolakoskiSequence.html