ELEKTROTEHNIŠKI VESTNIK 80(1-2): 34–38, 2013
PREGLEDNI ZNANSTVENI ČLANEK

Algoritmi za vsoto potenc naravnih števil

Jurij Mihelič

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Tržaška 25, 1000 Ljubljana, Slovenija
E-pošta: jurij.mihelic@fri.uni-lj.si

Povzetek. Zaporedja imajo ključni pomen v matematiki, teoretičnem računalništvu, analizi algoritmov, teoriji izračunljivosti, računski zahtevnosti in številnih drugih natančnih znanostih. V članku se posvetimo zaporedjem k-tih potenc prvih n naravnih števil. Za takšna zaporedja za poljubna k in n izpeljemo več različnih formul za izračun delne vsote, tj. vsote prvih n členov. Na podlagi izpeljanih formul razvijemo pet algoritmov, ki za podana n in K izračunajo vse delne vsote za k med 0 in K. Prva dva izmed predstavljenih algoritmov temeljita neposredno na definiciji delne vsote, preostali trije pa na rekurenčni enačbi, ki vsoto izračuna iz vsot nižjih redov. Za vse algoritme zapišemo implementacijo in podamo asimptotično časovno zahtevnost.

Ključne besede: zaporedje, vrsta, delna vsota, rekurenčna enačba, algoritem

Algorithms for sum of powers of integers

Sequences are of key importance in mathematics, theoretical computer science, analysis of algorithms, computability theory, computational complexity and many other exact sciences. In this paper we focus on the sequences of k-th powers of the first n natural numbers. For any k and n we derive several formulas to calculate the partial sum of the sequence, i.e., the sum of the first n terms. Based on these formulas, we develop five algorithms which for any n and K compute all partial sums for any k between 0 and K. The first two algorithms are based on the definition of the partial sum and the remaining three on the recurrence equation which calculates the sum from the lower order sums. For all the algorithms we give implementations and analyze asymptotic time complexity.

1 UVOD

Zapredja spadajo med najbolj raziskovane objekte v matematiki. Prav tako imajo velik pomen v (teoretičnem) računalništvu, še zlasti pri analizi algoritmov, teoriji izračunljivosti in zahtevnosti.

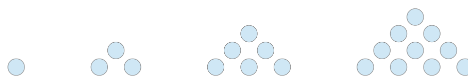
Z zaporedjem lahko opišemo časovno ali prostorsko zahtevnost algoritma, pri čemer n-ti člen zaporedja podaja zahtevnost algoritma pri vhodnih podatkih velikosti n [1]. Zaporedje lahko opišemo z *rodovno funkcijo*, področje, ki se s tem poglobljeno ukvarja, se imenuje *analitična kombinatorika* [2].

V nadaljevanju se bomo ukvarjali s posplošeno obliko enega najbolj znanih zaporedij, tj. z zaporedjem potenc reda k prvih n naravnih števil. Če je k celo število, so tudi vsi členi takšnega zaporedja cela števila. V članku bomo namenili pozornost algoritmom za izračun vsote takšnega zaporedja za poljubna k in n.

Najbolj znano in tudi enostavno je verjetno zaporedje naravnih števil $1, 2, 3, 4, 5 \ldots$, katerega vsoto do poljubnega $n \in \mathbb{N}$ izračunamo po dobro znani formuli [3]:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}. \tag{1}$$

Pri zaporednih $n \in \mathbb{N}$ delne vsote $\sum_{i=1}^{n} i$ tvorijo novo zapredje $1, 3, 6, 10, 15, \ldots$, tako imenovanih *trikotniških števil*. Ta pomenijo število objektov, ki jih lahko razporedimo v obliko enakostraničnega trikotnika (glej Sliko 1).



Slika 1: Trikotniška števila $1, 3, 6$ in 10

Formula (1) koristi pri analizi algoritmov. Na primer: preštejmo, kolikokrat se izvede tretja vrstica v naslednjem algoritmu.

```
1 for i = 1 to n do
2   for j = 1 to i do
3     print(i + j)
```

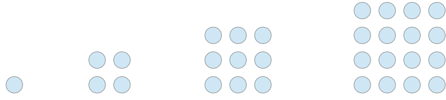
Obe zanki for preprosto spremenimo v vsoti in ju seštejemo:

$$\sum_{i=1}^{n} \sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

Dobro znano je tudi zaporedje kvadratov naravnih števil oz. *kvadratnih števil*, tj. $1, 4, 9, 16, 25, \ldots$, ki podajajo število objektov, razmeščenih v obliko kvadrata (glej sliko 2). Vsoto prvih n kvadratnih števil izračunamo po formuli:

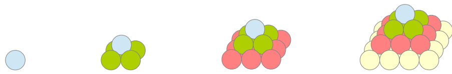
$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}. \tag{2}$$

Prejet 4. april, 2013
Odobren 15. april, 2013



Slika 2: Kvadratna števila $1, 4, 9$ in 16

Nadalje, za zaporedne $n \in \mathbb{N}$ vsote po enačbi (2) tvorijo novo zaporedje $1, 5, 14, 30, 55, \dots$ tako imenovanih *kvadratnih piramidnih števil*, ki podajajo število objektov, razmeščenih v obliko štiristrane piramide s kvadratno osnovno ploskvijo (glej sliko 3). Prav tako n-to kvadratno piramidno število npr. podaja število vseh mogočih kvadratov v šahovnici velikosti $n \times n$.



Slika 3: Kvadratna piramidna števila $1, 5, 14$ in 30

V nadaljevanju članka se bomo ukvarjali s posplošeno vrsto prvih n potenc reda k naravnih števil. Ta je natančneje definirana s formulo:

$$S_k(n) = \sum_{i=1}^{n} i^k = 1^k + 2^k + 3^k + \ldots + n^k. \quad (3)$$

Vsote različnih redov so med seboj večkrat povezane z enačbami, npr. $S_3(n) = (S_1(n))^2$ (vsoti kubov zato pravimo tudi *kvadrat trikotniškega števila*).

V analizi algoritmov pogosto srečamo tudi *harmonično vrsto*:

$$H(n) = S_{-1}(n) = \sum_{i=1}^{n} \frac{1}{i}.$$

In ne nazadnje neskončna hiperharmonična vrsta, ki je v matematični analizi znana kot *Riemanova zeta funkcija*:

$$\zeta(s) = S_{-s}(\infty) = \sum_{i=1}^{\infty} i^{-s}.$$

V literaturi [4] zasledimo tudi posplošitev vrste (3), kjer namesto zaporedja prvih n števil nastopa aritmetično zaporedje $(b + ia)$.

V nadaljevanju članka bomo predstavili več algoritmov, ki za podana n in K izračunajo vse vsote $S_k(n)$ za k med 0 in K. Najprej predstavimo algoritma, ki temeljita na definiciji $S_k(n)$ (enačba (3)). Nato sledita algoritma, zasnovana na rekurenčnih enačbah, ki $S_k(n)$ podajata kot vsoto predhodno izračunanih $S_i(n)$, kjer je $i < k$. In kot zadnjega opišemo še algoritem, ki ravno tako temelji na rekurenčni enačbi, le da za izračun potrebuje vsak drugi člen. Na koncu opišemo še nekaj mogočih optimizacij predstavljenih algoritmov.

2 Izračun po definiciji

Najprej si oglejmo algoritem, ki ga dobimo, če vsote $S_k(n)$ računamo neposredno po definiciji, podani s formulo (3). Izračun po formuli je treba izvesti $(K+1)$-krat; za vsak k od 0 do K. Implementacija samega algoritma je preprosta. Potrebujemo dve zanki: prvo prek parametra k in drugo za sam izračun vsote.

```
1  sums = [n, 0, 0, ..., 0]
2  for k = 1 to K do
3     sum = 1
4     for i = 2 to n do
5        sum += i^k
6     sums[k] = sum
```

V vrstici 1 ustvarimo tabelo sums dolžine $K + 1$, pri čemer prvi element tabele neposredno nastavimo na $S_0(n) = n$ (seštevek n enic). Vrstica 5 vsebuje izračun i^k, ki ga po potrebi lahko implementiramo tudi z dodatno zanko. Po izvedbi algoritma k-ti element tabele sums vsebuje vrednost $S_k(n)$.

Asimptotična časovna zahtevnost algoritma je $O(K^2 n)$, prostorska pa $O(K)$. Pri tem smo upoštevali, da operacija potenciranja zahteva $O(k)$ množenj. Opozorimo še, da gre pravzaprav za eksponentno časovno zahtevnost, ker za zapis vhoda, tj. števil K in n, potrebujemo le $\log(Kn)$ bitov.

3 Optimizacija potenciranja

Operacijo potenciranja v algoritmu iz predhodnega razdelka je mogoče pohitriti. Vsote računamo postopoma po naraščajočih k: potence reda k pa seveda ni treba vsakič računati znova, ampak je mogoče k-to potenco izračunati iz $(k-1)$-ve. Velja namreč:

$$x^k = x \cdot x^{k-1}.$$

Nov algoritem potrebuje dodaten prostor, tj. tabelo velikosti $n - 1$ za hranjenje potenc reda k za vse $2, 3, 4, \dots, n$. Implementacija je podobna algoritmu iz predhodnega razdelka.

```
1  sums = [n, 0, 0, ..., 0]
2  pows = [2,3,4,...,n]
3  for k = 1 to K do
4     sum = 1
5     for i = 2 to n do
6        sum += pows[i - 2]
7        pows[i - 2] *= i
8     sums[k] = sum
```

V drugi vrstici se nahaja inicializacija tabele pows (dolžine $n - 1$), ki hrani potence reda k. Dodana je še vrstica 7, ki iz predhodne potence reda $k - 1$ izračuna potenco reda k.

Asimptotična časovna zahtevnost algoritma je $O(Kn)$. S preprosto optimizacijo smo za red velikosti zmanjšali časovno zahtevnost. Prostorska zahtevnost algoritma je $O(K + n)$.

4 Alternirajoča vrsta

V tem razdelku razvijemo algoritem za zadani problem s pomočjo enačbe, ki med seboj povezuje vsote $S_i(n)$,

kjer je $0 \leq i \leq k$. Enačba temelji na alternirajoči vrsti členov $S_i(n)$. Najprej zapišimo enačbo, nato jo bomo izpeljali:

$$\sum_{i=0}^{k} \binom{k+1}{i} (-1)^{k-i} S_i(n) = n^{k+1}. \tag{4}$$

V nadaljevanju bomo večkrat potrebovali *binomski izrek*, zato ga zapišimo:

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i}, \tag{5}$$

pri čemer je *binomski koeficient* definiran kot:

$$\binom{n}{i} = \frac{n!}{(n-i)!\, i!}. \tag{6}$$

Izpeljavo rekurenčne formule za $S_k(n)$ začnemo iz definicije za vsoto $S_{k+1}(n)$. Najprej izrazimo zadnji, tj. n-ti člen, nato premaknemo sumacijski indeks za ena in upoštevamo, da je člen pri $j = 1$ enak 0. Nadaljujemo z uporabo binomskega izreka, nato obrnemo vrstni red vsot.

$$\begin{aligned} S_{k+1}(n) &= \sum_{j=1}^{n-1} j^{k+1} + n^{k+1} \\ &= \sum_{j=1}^{n} (j-1)^{k+1} + n^{k+1} \\ &= \sum_{j=1}^{n} \sum_{i=0}^{k+1} \binom{k+1}{i} j^i (-1)^{k+1-i} + n^{k+1} \\ &= \sum_{i=0}^{k+1} \binom{k+1}{i} (-1)^{k+1-i} S_i(n) + n^{k+1}. \end{aligned}$$

V zadnjem izrazu je člen v vsoti pri $i = k+1$ enak $S_{k+1}(n)$, kar se odšteje s členom na levi strani enačbe. Preostanek je lepše zapisan v obliki enačbe (4).

Lotimo se implementacije algoritma po izpeljani enačbi. Najprej izrazimo člen $S_k(n)$, ki ga želimo izračunati:

$$S_k(n) = \frac{1}{k+1} \left(n^{k+1} + \sum_{i=0}^{k-1} \binom{k+1}{i} (-1)^{k+1-i} S_i(n) \right). \tag{7}$$

Preden dokončno zapišemo psevdokodo algoritma, naštejmo nekaj ugotovitev. Potenco n^{k+1} lahko izračunavamo sproti ob samem računanju vsote. Pri računanju vsote je treba upoštevati vse predhodno izračunane $S_i(n)$, kjer je $0 \leq i < k$. Vrednosti $S_i(n)$ alternirajoče enkrat prištevamo, drugič odštevamo; to je odvisno od $(-1)^{k+1-i}$. Začetni člen pri $i = 0$ je lahko pozitiven ali negativen, kar je odvisno od k. Zadnji člen pri $i = k-1$ pa je $(-1)^2 = 1$, torej vedno pozitiven. Če vsoto začnemo računati od $k-1$ proti 0, lahko vedno začnemo s prištevanjem. Poleg tega je tako tudi laže računati binomske koeficiente, kot bomo videli v nadaljevanju. Pri seštevanju je treba posamezne člene $S_i(n)$, kjer $0 \leq i < k$ še množiti z binomskim koeficientom $\binom{k+1}{i}$.

Oglejmo si še izračun binomskih koeficientov. Izračun po definiciji (6) se pogosto izjalovi zaradi faktoriele oz. prevelikih števil. Poleg tega je počasen, ker izračun $n!$ vsebuje $n-2$ množenj. Spomnimo se, da je binomski koeficient mogoče izračunati tudi po naslednji rekurenčni enačbi [3]:

$$\binom{k+1}{i} = \binom{k}{i} + \binom{k}{i-1}.$$

Takšen izračun je za naš algoritem pravzaprav zelo priročen. Rekurenčna enačba nam da dobro znani *Pascalov trikotnik* (glej tabelo 1). Pri računanju $S_k(n)$ potrebujemo prvih k koeficientov vrstice $(k+1)$ trikotnika. Vedno potrebujemo le eno vrstico, ki jo z uporabo rekurenčne enačbe izračunamo iz predhodne. Celotne vrstice ni treba naračunati vnaprej, ampak lahko ustrezni koeficient izračunamo natanko takrat, ko ga potrebujemo.

k\i	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
6	1	6	15	20	15	6	1

Tabela 1: Pascalov trikotnik za $0 \leq k \leq 6$: vsaka vrstica vsebuje števila $\binom{k}{i}$ za $0 \leq i \leq k$

Na voljo imamo dovolj informacij, da lahko zapišemo implementacijo. Zopet potrebujemo dve zanki: prvo prek parametra k in drugo za sam izračun vsote po formuli (7).

```
1  binoms = [1, 0, 0, ..., 0]
2  sums = [n, 0, 0, ..., 0]
3  nk1 = n
4  for k = 1 to K do
5      sum = 0, neg = 1
6      for i = k − 1 to 0 by −1 do
7          if i > 0 then
8              binoms[i] += binoms[i−1]
9          sum += neg*binoms[i]*sums[i]
10         neg = −neg
11     nk1 *= n
12     sums[k] = (sum+nk1) / (k+1)
13     binoms[k] = k + 1
```

Vrstice od 1 do 3 vsebujejo inicializacijo spremenljivk. Tabela binoms (dolžine $K+1$) pomeni aktivno vrstico Pascalovega trikotnika. Spremenljivka nk1 vsebuje potenco n^{k+1}, ki jo, kot je že bilo rečeno, računamo sproti v prvi zanki. Obe tabeli, tako binoms kot sums, imata svoj prvi element že nastavljen na pravo vrednost.

Zunanja zanka v vrstici 4 je namenjena izračunu $S_k(n)$ za vse k od 1 do K. V akumulatorju sum se

računa skupna vsota, spremenljivka neg je indikator prištevanja oz. odštevanja. Notranja zanka z začetkom v vrstici 6 je implementacija zgoraj zapisane rekurenčne formule. Najprej v vrstici 8 izračunamo nov binomski koeficient, nato ga pomnožimo (vrstica 9) z ustrezno že predhodno izračunano vsoto, rezultat pa prištejemo oz. odštejemo v akumulator. Po zanki dokončamo izračun vsote in rezultat shranimo, nato sledi le še priprava na morebitni naslednji korak zanke.

Asimptotična časovna zahtevnost algoritma je $O(K^2)$, prostorska pa $O(k)$. Algoritem je torej asimptotično hitrejši od algoritma, opisanega v predhodnem razdelku.

5 Nealternirajoča vrsta

Algoritem po alternirajoči vrsti zahteva nekaj dodatnega dela in pazljivosti pri seštevanju členov. Že Blaise Pascal naj bi poznal rekurenčno enačbo, v kateri se členi v vrsti le prištevajo:

$$\sum_{i=0}^{k}\binom{k+1}{i}S_i(n)=(n+1)^{k+1}-1. \tag{8}$$

Veljavnost enačbe je mogoče pokazati s pomočjo indukcije, kar pa je ne pomaga izpeljati. To lahko storimo na več načinov: z rodovnimi funkcijami [7], z uporabo kombinatorike [8] ali z uporabo algebraičnih metod [9], [10]. Izpeljavo povzemamo po [10]. Začnimo z naslednjo enačbo:

$$\sum_{j=1}^{n}\left((j+1)^{k+1}-j^{k+1}\right)=(n+1)^{k+1}-1$$

Da enačba res drži, vidimo, če zapišemo nekaj členov vrste:

$$(2^{k+1}-1^{k+1})+(3^{k+1}-2^{k+1})+\cdots+((n+1)^{k+1}-n^{k+1})$$

Večina členov se odšteje, ostaneta le: $(-1)^{k+1}$ in $(n+1)^{k+1}$.

Lotimo se leve strani identitete. Najprej člen $(j+1)^{k+1}$ razvijemo po binomskem izreku, nato se člen j^{k+1} razvoja odšteje z $-j^{k+1}$. V preostanku obrnemo vrstni red seštevanja vrst, upoštevamo definicijo za $S_i(n)$ in enačba (8) je izpeljana:

$$\begin{aligned}\sum_{j=1}^{n}\Big(\sum_{i=0}^{k+1}\binom{k+1}{i}j^i-j^{k+1}\Big)&=\\ &=\sum_{j=1}^{n}\sum_{i=0}^{k}\binom{k+1}{i}j^i\\ &=\sum_{i=0}^{k}\binom{k+1}{i}S_i(n).\end{aligned}$$

Lotimo se še implementacije algoritma. Najprej enačbo (8) preoblikujmo, da bo $S_k(n)$ na levi strani, vse drugo pa na desni:

$$S_k(n)=\frac{1}{k+1}\Big((n+1)^{k+1}-1-\sum_{i=0}^{k-1}\binom{k+1}{i}S_i(n)\Big) \tag{9}$$

Dobljena enačba je podobna enačbi (7), le da vsota ne vsebuje člena $(-1)^{k+1-i}$, ki povzroči alternacijo predznaka členov, kar malce poenostavi algoritem. Pri implementaciji se zgledujemo po algoritmu iz predhodnega razdelka.

```
1  binoms = [1, 0, 0, ..., 0]
2  sums = [n, 0, 0, ..., 0]
3  n1k1 = n + 1
4  for k = 1 to K do
5      sum = 0
6      for i = k − 1 to 0 by −1 do
7          if i > 0 then
8              binoms[i] += binoms[i−1]
9          sum += binoms[i] ∗ sums[i]
10     n1k1 ∗= n + 1
11     sums[k] = (n1k1−1−sum) / (k+1)
12     binoms[k] = k + 1
```

Asimptotična časovna zahtevnost algoritma je enaka kot pri algoritmu iz predhodnega razdelka, tj. $O(K^2)$.

6 Prepolovitev dela

V tem razdelku najprej izpeljemo rekurenčno enačbo, ki za izračun $S_k(n)$ ne potrebuje vseh vrednosti $S_i(n)$, kjer $i<k$, ampak le vsako drugo. Nato zapišemo še implementacijo algoritma po izpeljani enačbi.

Vzemimo enačbi (4) in (8) in ju seštejmo, tako dobimo:

$$\sum_{i=0}^{k}\binom{k+1}{i}S_i(n)\big((-1)^{k-i}+1\big)=(n+1)^{k+1}+n^{k+1}-1.$$

V vsoto vpeljimo nov indeks $j=k-i$ in upoštevajmo simetričnost binomskega koeficienta $\binom{n}{k}=\binom{n}{n-k}$:

$$\sum_{j=0}^{k}\binom{k+1}{j+1}S_{k-j}(n)\big((-1)^{j}+1\big)=(n+1)^{k+1}+n^{k+1}-1.$$

Opazimo, da je vsak drugi člen vrste enak nič oz. natančneje, da velja:

$$(-1)^j+1=\begin{cases}0 & j \text{ je lih}\\ 2 & j \text{ je sod.}\end{cases}$$

Vpeljimo nov indeks i, pri čemer $j=2i$, in dobimo:

$$2\sum_{i=0}^{\lfloor k/2\rfloor}\binom{k+1}{2i+1}S_{k-2i}(n)=(n+1)^{k+1}+n^{k+1}-1.$$

Dobljeno enačbo preoblikujmo, da bo primerna za uporabo v algoritmu:

$$\begin{aligned}S_k(n)&=\frac{1}{k+1}\Big(\frac{(n+1)^{k+1}+n^{k+1}-1}{2}\\ &-\sum_{i=1}^{\lfloor k/2\rfloor}\binom{k+1}{2i+1}S_{k-2i}(n)\Big).\end{aligned} \tag{10}$$

Zapišimo še implementacijo. Na vsakem koraku je treba izračunati celotno vrstico Pascalovega trikotnika, čeprav gre vsota v enačbi (10) le do $\lfloor k/2 \rfloor$. Zato v algoritmu nastopata dve notranji zanki (vrstici 9 in 12). Prva skrbi za izračun binomskih koeficientov, druga pa dodaja še izračun vsote po enačbi (10).

```
1  binoms = [1, 1, 0, 0, ..., 0]
2  sums = [n, 0, 0, ..., 0]
3  n1k1 = n + 1
4  nk1 = n
5  for k = 1 to K do
6      sum = 0
7      binoms[k + 1] = 1
8      i = k
9      while i > k/2 do
10         binoms[i] += binoms[i - 1]
11         i -= 1
12     while i >= 1 do
13         binoms[i] += binoms[i - 1]
14         sum += binoms[2*i + 1] *
15                sums[k - 2*i]
16         i -= 1
17     n1k1 *= n + 1
18     nk1 *= n
19     tmp = (n1k1 + nk1 - 1) / 2
20     sums[k] = (tmp - sum) / (k + 1)
```

Asimptotična časovna zahtevnost je enaka $O(K^2)$, vendar je število množenj prepolovljeno. Opozorimo, da gre pravzaprav za množenja velikih števil – časovna zahtevnost takega množenja pa ni konstantna, ampak je odvisna od velikosti števila.

7 Optimizacije

Omenimo še nekaj mogočih splošnih optimizacij, ki jih lahko uporabimo v zgoraj opisanih algoritmih. Vsote za majhne k lahko vnajprej izračunamo po dobro znanih formulah. S tem lahko prihranimo nekaj iteracij zunanje zanke v vseh omenjenih algoritmih.

Vrstice v Pascalovem trikotniku so simetrične po pravilu $\binom{n}{i} = \binom{n}{n-i}$. Z upoštevanjem simetričnosti lahko izračunamo le polovico vrstice Pascalovega trikotnika. Druge polovice nam niti ni treba hraniti, v algoritmu pa moramo zaradi tega dodati preverjanje parametra i binomskega koeficienta.

Simetričnost lahko izrabimo tudi tako, da zmanjšamo število množenj velikih števil. Pri izračunu vsote namreč lahko upoštevamo obe S_i in S_{k+1-i} hkrati. Kot primer, predelajmo enačbo (9) tako, da zmanjšamo število množenj za polovico:

$$\begin{aligned} S_k(n) &= \frac{1}{k+1}\Big((n+1)^{k+1} - 1 \\ &- S_0(n) - (k+1)S_1(n) - \\ &- \sum_{i=2}^{k/2}\binom{k+1}{i}\big(S_i(n) + S_{k+1-i}(n)\big)\Big) \end{aligned}$$

Zapisana enačba velja pri sodih vrednostih k, pri lihih k ravnamo podobno, le srednji člen moramo posebej upoštevati.

8 Sklep

Zaporedja in njihove vsote so pomembni na številnih znanstvenih področjih. V članku smo se ukvarjali z vsoto zaporedja potenc reda k prvih n naravnih števil. Opisali smo več algoritmov za izračun te vsote. Vse algoritme smo matematično utemeljili. Poleg tega smo zapisali tudi njihovo implementacijo, na podlagi katere smo podali tudi asimptotično časovno zahtevnost.

Literatura

[1] Robert Sedgewick, Philippe Flajolet, *An Introduction to the Analysis of Algorithms*, 2nd ed. Addison-Wesley, 2013.

[2] Philippe Flajolet, Robert Sedgewick, *Analytic Combinatorics*, Cambridge University Press, 2009.

[3] Ronald Graham, Donald Knuth, and Oren Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Addison–Wesley, 1994.

[4] N. Gauthier, "Sum of the m-th powers of n successive terms of an arithmetic sequence: $bm + (a+b)m + (2a+b)m + \ldots + ((n-1)a+b)m$", *International Journal of Mathematical Education in Science and Technology*, vol. 37, no. 3, pp.

[5] Michael Z. Spivey, "The Euler-Maclaurin Formula and Sums of Powers", *Mathematics Magazine*, vol. 79, no. 1, pp. 61–65, 2006.

[6] David M. Bloom, "An Old Algorithm for the Sums of Integer Powers", *Mathematics Magazine*, vol. 66, no. 5, pp. 304–305, 1993.

[7] J. Riordan, *Combinatorial Identities*, Wiley, 1968.

[8] J. L. Paul, "On the Sum of the k-th Powers of the First n Integers", *The American Mathematical Monthly*, vol. 78, no. 3, pp. 271–272, 1971.

[9] Clive Kelly, "An Algorithm for Sums of Integer Powers", *Mathematics Magazine*, vol. 57, no. 5, pp. 296–297, 1984.

[10] Dumitru Acu, "Some Algorithms for the Sums of Integer Powers", *Mathematics Magazine*, vol. 61, no. 3, pp. 189–191, 1988.

Jurij Mihelič je leta 2006 doktoriral s področja prilagodljivosti v optimizacijskih problemih na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Je asistent in raziskovalec na omenjeni fakulteti. Njegovo področje raziskovanja vključuje načrtovanje in analizo algoritmov ter podatkovnih struktur, kombinatorično optimizacijo, teorijo grafov.