Рачунарска провера Курепине хипотезе за уопштене леве факторијеле

–Мастер теза–

Александар Тошић

ментор:
др. Миодраг Живковић

Математички факултет
Универзитет у Београду
2012

Садржај

1 Увод **5**

2 Ератостеново сито **6**

3 Леви факторијели, хипотезе и уопштења **7**

3.1 Леви факторијел 7

3.2 Алтернирајуће суме факторијела 8

3.3 Уопштења левог факторијела 8

3.4 Суме степенова факторијела 10

4 Програмска реализација и резултати **12**

4.1 Ератостеново сито 12

4.2 Паралелизација - нити 13

4.3 Леви факторијели и алтернирајуће суме 18

4.4 Уопштења и суме степенова факторијела 23

4.5 Добијени резултати 24

5 Закључци и даљи рад **39**

1 Увод

Појам левог факторијела увео је Ђ. Курепа [1]. У истом раду Курепа је поставио следећу хипотезу: највећи заједнички делилац за $!n$ и $n!$ је 2. Ова хипотеза, са својим еквивалентима, је предмет многих истраживања, а појављује се и у књизи Р. Гаја (*R. Guy "Unsolved problems in Number Theory"* [2]) као проблем Б44. Хипотеза је уз помоћ рачунара више пута проверавана, али контрапример није пронађен. Један од циљева овог рада је конструкција програма који користи предности савремене технологије, као што су процесори са више језгара и 64-битна архитектура, за проверу Курепине хипотезе на што већем почетном интервалу скупа природних бројева.

Поред поменутог програма конструисана су још три слична који се баве испитивањем алтернирајуће суме факторијела, уопштењем левог факторијела и сумама степенова факторијела. Хипотеза о алтернирајућим сумама факторијела $A_n = n! - (n-1)! + (n-2)! - + \ldots - (-1)^n 1!$ се такође налазе у књизи [2] као проблем Б43. Како су неки бројеви A_n прости, постављено је питање да ли међу њима бескоачно много простих. Проблем је решио М. Живковић [3]. Пошто решење проблема већ постоји онда је циљ овог сегмента рада конструкција програма који користи тренутно доступну технологију ради упоређивања брзине обраде података са технологијом доступном у 1999.

Када су у питању уопштење левог факторијела и суме степенова факторијела, циљ је испитивање постојања сличних правилности као код левог факторијела, као и било каквих других правилности међу добијеним резултатима.

2 Ератостеново сито

Ератостеново сито је алгоритам за генерисање простих бројева од два до задате границе. Улаз у алгоритам представља скуп природних бројева, у опсегу од два до задате границе, из кога се затим избацују бројеви који нису прости како би на крају у скупу остали само прости бројеви. Алгоритам се одвија у пар једноставних корака који се циклично понављају. На почетку првог циклуса број два се означава као прост број и затим се из скупа редом избацују сви бројеви који су дељиви са два. У сваком наредном циклусу бира се први не избачени број који се налази одмах после означеног простог броја из претходног циклуса и тако изабран број се означава као прост. Даље се избацују сви бројеви коју су дељиви бројем који је означен као прост за тај циклус. Циклуси се понављају све док се не дође до задате границе тј. док у скупу не остану само прости бројеви.

На основу следеће теореме овај поступак може да се скрати, довољно је да циклуси иду до квадратног корена горње границе уместо до саме границе.

Теорема 1. Ако је позитиван цео број n сложен број, тада n има прост фактор p, такав да је $p^2 \leq n$.

Доказ Нека је p најмањи прост фактор броја n. Ако су r и s нетривијални фактори броја $n = rs$, онда је $p \leq r$ и $p \leq s$, из чега следи да је $p^2 \leq rs = n$.

3 Леви факторијели, хипотезе и уопштења

3.1 Леви факторијел

Дефиниција 1. (Курепа [1])За сваки ненегативан цели број n нека је леви факторијел $!n$ дефинисан са

$$!n = 0! + 1! + 2! + \ldots + (n-1)!$$

У истом раду Курепа је поставио питање да ли је $!n \not\equiv 0 \pmod{n}$, за сваки природан број $n \geq 2$. Пошто је релација задовољена за бројеве 3,4,5,6,7, а самим тим и за њихове умношке, следи да има бесконачно много бројева n за које је $!n \not\equiv 0 \pmod{n}$. Шта више, у раду постављена хипотеза да је 2 највећи заједнички делилац за $!n$ и $n!$.

Нека је

$$r(n) = !n \pmod{n} \tag{1}$$

Један од циљева овог рада је конструкција програма који проверава да ли постоји такво n за које $r(n) = 0$, а то је уједно и проблем Б44 у књизи [2]. До сада је било више рачунарских провера овог проблема, као што су: Ж. Мијајловић за $n < 311009$ [5], М. Живковић за $n < 2^{23}$ [3], И. Галот (*Y. Gallot*) за $n < 2^{26}$ [6], П Цоблинг (*P. Jobling*) за $n < 144000000$ [7] и М. Татаревић за $n < 10^9$ [8].

За проверу овог проблема није потребно вршити испитивање за све природне бројеве, већ је то довољно урадити за просте бројеве. Заиста, ако за неки сложени број n важи $!n \equiv 0 \pmod{n}$, тј. $n|!n$, онда за сваки прости број p који дели n важи следеће:

Теорема 2. Ако за природан број n важи $n|!n$ онда за сваки прост број p, који дели n, важи $p|!p$.

Доказ Ако $p|n$ и $n|!n$ то повлачи да $p|!n$. Како p дели $p!$, $(p+1)!$, ..., $(n-1)!$ и p дели $!n$, а $!n = !p + p! + (p+1)! + \ldots + (n-1)!$, онда p дели $!p$

3.2 Алтернирајуће суме факторијела

Дефиниција 2. (Р. Гај [2], проблем Б43) Алтернирајуће суме факторијела су дефинисане на следећи начин:

$$A_n = n! - (n-1)! + (n-2)! - + \ldots - (-1)^n 1!$$

Пошто су неки прости бројеви облика A_n ($3! - 2! + 1! = 5$, итд.) постављено је питање да ли постоји бесконачно много таквих простих бројева. Нека је

$$a(n) = A_{n-1} \pmod{n} \tag{2}$$

Ако постоји n такво да $a(n) = 0$ онда n дели и A_m за све $m \geq n$, па постоји само коначно много простих бројева облика A_n. Одговор на ово питање дао је М. Живковић [3] показавши да ако је $p = 3612703$ онда $p|A_n$, $\forall n \geq p$.

3.3 Уопштења левог факторијела

Леви факторијел се могу уопштити на више начина. Једна могућност је да се уведу низови $L(n,k)$ као уопштени факторијели реда k. При томе су $L(n,0) = !n$ сами леви факторијели, а уопштени факторијели реда k су кумулативне суме уопштених левих факторијела реда $k-1$ за $k \geq 1$.

Дефиниција 3. За ненегативне целе бројеве n, k нека је уопштење левог факторијела дефинисано са

$$L(n,k) = \left\{ \begin{array}{ll} 1, & n = 1 \\ !n, & k = 0 \\ L(n-1,k) + L(n,k-1), & n > 1 \end{array} \right\}$$

У Табели 1 наведени су бројеви $L(n,k)$ за $n, k \leq 8$.

$n \setminus k$	$0(!p)$	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1	1
2	2	3	4	5	6	7	8	9	10
3	4	7	11	16	22	29	37	46	56
4	10	17	28	44	66	95	132	178	234
5	34	51	79	123	189	284	416	594	828
6	154	205	284	407	596	880	1296	1890	2718
7	874	1079	1363	1770	2366	3246	4542	6432	9150
8	5914	6993	8356	10126	12492	15738	20280	26712	35862

Табела 1: $L(n,k), 1 \leq n \leq 8, 0 \leq k \leq 8$

Уопштени факторијли реда k могу се представити као суме производа факторијела и одговарајућих полинома $L(n,k) = \sum_{i=1}^{n} i! P_k(i)$.

Специјално, $P_0(i) = 1$ јер је $L(n,0) = !n = \sum_{i=0}^{n-1} i! = \sum_{i=1}^{n}(n-i)!$. У следећим редовима биће изведене формуле за прва три уопштења.

Уопштени леви факторијели реда 1 могу се представити у облику:

$$\begin{aligned}
&L(n,1)\\
&=!1+!2+\ldots+!n\\
&=(0!)+(0!+1!)+\ldots+(0!+1!+\ldots+(n-1)!)\\
&=n\cdot 0!+(n-1)1!+\ldots+1(n-1)!\\
&=\textstyle\sum_{i=0}^{n-1}(n-i)i!=\sum_{i=1}^{n} i(n-i)!
\end{aligned}$$

Дакле, види се да је $P_1(i) = i = \binom{i}{1}$

Уопштени леви факторијели реда 2 могу се представити у облику:

$$\begin{aligned}
&L(n,2)\\
&=(!1)+(!1+!2)+\ldots+(!1+!2+\ldots+!n)\\
&=n\cdot!1+(n-1)!2+\ldots+1\cdot!n\\
&=n(0!)+(n-1)(0!+1!)+\ldots+1\cdot(0!+1!+\ldots+(n-1)!)\\
&=\tfrac{n(n+1)}{2}0!+\tfrac{(n-1)n}{2}1!+\ldots+\tfrac{1(1+1)}{2}(n-1)!\\
&=\textstyle\sum_{i=0}^{n-1}\frac{(n-i)(n-i+1)}{2}i!=\sum_{i=1}^{n}\frac{i(i+1)}{2}(n-i)!
\end{aligned}$$

Дакле, види се да је $P_2(i) = \frac{i(i+1)}{2} = \binom{i+1}{2}$

Уопштени леви факторијели реда 3 могу се представити у облику:

$$\begin{aligned}
&L(n,3)\\
&=(!1)+((!1)+(!1+!2))+\ldots+((!1)+(!1+!2)+\ldots+(!1+!2+\ldots+!n))\\
&=n\cdot!1+(n-1)(!1+!2)+\ldots+1\cdot(!1+!2+\ldots+!n)\\
&=\tfrac{n\cdot(n+1)}{2}!1+\tfrac{(n-1)n}{2}!2+\ldots+\tfrac{1\cdot(1+1)}{2}!n\\
&=\tfrac{n\cdot(n+1)}{2}0!+\tfrac{(n-1)n}{2}(0!+1!)+\ldots+\tfrac{1\cdot(1+1)}{2}(0!+1!+\ldots+(n-1)!)\\
&=(\tfrac{n\cdot(n+1)}{2}+\tfrac{(n-1)n}{2}+\ldots+\tfrac{1\cdot(1+1)}{2})0!+\\
&+(\tfrac{(n-1)n}{2}+\tfrac{(n-2)(n-1)}{2}+\ldots+\tfrac{1\cdot(1+1)}{2})1!+\ldots+\tfrac{1\cdot(1+1)}{2}(n-1)!\\
&=\textstyle\sum_{i=0}^{n-1} i!\cdot\sum_{j=1}^{n-i}\frac{j(j+1)}{2}=\sum_{i=1}^{n}(n-i)!\cdot\sum_{j=1}^{i}\frac{j(j+1)}{2}
\end{aligned}$$

Дакле, види се да је $P_3(i) = \sum_{j=1}^{i}\frac{j(j+1)}{2} = \binom{i+2}{3}$

Полиноми $P_k(i)$ могу се представити као биномни коефицијенти па је тако $P_0(i) = i = \binom{i-1}{0}$, $P_1(i) = i = \binom{i}{1}$, $P_2(i) = \frac{i(i+1)}{2} = \binom{i+1}{2}$, $P_3(i) = \sum_{j=1}^{i}\frac{j(j+1)}{2} = \binom{i+2}{3}$, итд. Може се дакле претпоставити да важи следећа теорема.

Теорема 3. Бројеви $L(n,k)$ могу се изразити у облику $\sum_{i=1}^{n}(n-i)!\binom{i+k-1}{k}$.

Доказ Тврђење теореме доказује се индукцијом. Непосредно се проверава да је тврђење теореме тачно за $k = 0$ и n произвољно, односно за $n = 1$ и k произвољно. Ово је база доказа индукцијом и покрива прву врсту и колону табеле са уопштеним лавим факторијелима. Да се теорема докаже, довољно је доказати да она важи за $L(n,k)$ ако важи за $L(n-1,k)$ и $L(n,k-1)$. Према дефиницији је $L(n,k) = L(n,k-1) + L(n-1,k)$. Како је $L(n,k-1) = \sum_{i=1}^{n}(n-i)!\binom{i+k-2}{k-1}$ и $L(n-1,k) = \sum_{i=1}^{n-1}(n-1-i)!\binom{i+k-1}{k}$ сабирањем се добија:

$$
\begin{aligned}
&L(n,k-1) + L(n-1,k)\\
&= \textstyle\sum_{i=1}^{n}(n-i)!\frac{i(i+1)\ldots(i+k-2)}{(k-1)!} + \sum_{i=1}^{n-1}(n-1-i)!\frac{i(i+1)\ldots(i+k-1)}{k!}\\
&= \textstyle\sum_{i=2}^{n}(n-i)!\frac{i(i+1)\ldots(i+k-2)}{(k-1)!} + (n-1)!\frac{1\cdot(1+1)\cdot\ldots\cdot(1+k-2)}{(k-1)!}\\
&+ \textstyle\sum_{i=1}^{n-1}(n-1-i)!\frac{i(i+1)\ldots(i+k-1)}{k!}\\
&= \textstyle\sum_{i=1}^{n-1}(n-1-i)!\frac{(i+1)(i+2)\ldots(i+k-1)}{(k-1)!} + (n-1)!\\
&+ \textstyle\sum_{i=1}^{n-1}(n-1-i)!\frac{i(i+1)\ldots(i+k-1)}{k!}\\
&= \textstyle\sum_{i=1}^{n-1}(n-1-i)!\frac{(i+1)(i+2)\ldots(i+k-1)}{(k-1)!}(1+\frac{i}{k}) + (n-1)!\\
&= \textstyle\sum_{i=1}^{n-1}(n-1-i)!\frac{(i+1)(i+2)\ldots(i+k)}{k!} + (n-1)!\\
&= \textstyle\sum_{i=2}^{n}(n-i)!\frac{i(i+1)\ldots(i-1+k)}{k!} + (n-1)!\\
&= \textstyle\sum_{i=1}^{n}(n-i)!\frac{i(i+1)\ldots(i+k-1)}{k!} = \sum_{i=1}^{n}(n-i)!\binom{i+k-1}{k} = L(n,k)
\end{aligned}
$$

Нека је

$$R(p,k) = L(p,k) \mod p \tag{3}$$

У вези са уопштеним левим факторијелима поставља се питање да ли за њих важе слична тврђења као за обичне леве факторијеле односно да ли су могуће једнакости $R(p,k) = 0$.

3.4 Суме степенова факторијела

Друго могуће уопштење левих факторијела су суме степенова факторијела.

$$S(n,k) = 0!^k + 1!^k + \ldots + (n-1)!^k = \textstyle\sum_{i=0}^{n-1} i!^k$$

За $k = 1$ су то обични леви факторијели. Како су овој суми сви сабирци сем прва два парни, сума је парна за $n \geq 2$ и специјално 2 дели !2.

Ова особина се може уопштити. Како за $0 \leq i \leq p-1$ према малој Фермаовој теореми важи $i!^{p-1} \equiv 1 \pmod p$, закључује се да је $S(p,p-1) = 0!^{p-1} + 1!^{p-1} + \ldots + (p-1)!^{p-1} \equiv 0 \pmod p$ па је $S(n,p-1)$ дељиво са p за свако $n \geq p$. Дакле, ако је $k = p-1$ где је p прост број, онда међу бројевима $S(n,k)$ може да буде само коначно много простих.

Природно је разматрати остатке

$$Q(p,k) = S(p,k) \pmod{p} \tag{4}$$

и потенцијалне једнакости $Q(p,k) = 0$. Из изложеног се види да је увек $Q(p,p-1) = 0$, па остаје да се провери постојање осталих, нетривијалних једнакости $Q(p,k) = 0$.

4 Програмска реализација и резултати

За проналажење простих бројева p за које су $r(p)(1), a(p)(2), R(p,k)$ (3) за неко k, и $Q(p,k)$ (4) за неко k једнаки 0, направљена су четири програма. Ови програми се разликују по финкцијама које обрађују просте бројеве док су генерисање простих бројева и паралелизација реализовани на исти начин у сва четири програма.

4.1 Ератостеново сито

За генерисање простих бројева користи се алгоритам који се базира на Ератостеновом ситу. Простих бројева мањих од 2^{25} има 2063689 и они се смештају у статички низ (*nizProstih*). Поред овог низа креира се и низ квадрата (*nizKvadrata*) простих бројева који се користи у складу са Теоремом 1. Пошто је 5801 први прост број чији је квадрат већи од 2^{25} онда је овај квадрат узет као последњи за упис у *nizKvadrata*. На интервалу $[2, 5801]$ има 761 простих бројева па се зато *nizKvadrata* иницијализује на ту дужину. Кôда овог алгоритма је следећи:

```
int nizProstih[2063689];
nizProstih[0] = 2;
int nizKvadrata[761];
nizKvadrata[0] = 4;
bool prost;
pozicijaSledecegProstog = 1;
duzinaNizaKvadrata = 1;
for( int i = 3; i <= 33554432; i++ )
{
   prost = true;
   pozicijaKvadrata = 0;
   while( pozicijaKvadrata < duzinaNizaKvadrata )
   {
      if( i < nizKvadrata[pozicijaKvadrata] )
      {
         break;
      }
      /* pozicijaKvadrata уједно представља и позицију
         одговарајућег простог броја из ниске nizProstih */
      else if( i % nizProstih[pozicijaKvadrata] == 0 )
      {
         prost = false;
         break;
      }
      pozicijaKvadrata++;
   }
   if( prost )
   {
```

```
            nizProstih[pozicijaSledecegProstog] = i;
            pozicijaSledecegProstog++;
            if( duzinaNizaKvadrata < 761)
            {
               nizKvadrata[duzinaNizaKvadrata] = i * i;
               duzinaNizaKvadrata++;
            }
         }
      }
```

На почетку се у *nizProstih* уписује 2 као први прост број, а у *nizKvadrata* се уписује 4 као квадрат од 2. Испитују се сви бројеви од 3 до 2^{25}. По уласку у главну петљу претпостави се да је број који испитујемо прост тако што се буловској променљивој *prost* додели вредност **true**. За испитивање се користе претходно нађени прости бројеви ≤ 5801, као и њихови квадрати. Прво се проверава да ли је број, који се испитује, мањи од квадрата простог броја помоћу кога се врши тренутно испитивање. Ако је то случај онда према Теореми 1 нема потребе да се испитује даље јер се дошло до простог броја чији је квадрат већи од броја који се испитује, а ниједан претходни прост број није делио број који се испитује. Помоћу **break** наредбе се излази из петље и број се уписује на одговарајуће место у *nizProstih* (за случај да је испитивани број ≤ 5801 онда се и његов квадрат уписује на одговарајуће место у *nizKvadrata*). У случају да је испитивани број већи од квадрата простог броја, којим се врши испитивање, прелази се на другу проверу. Другом провером се утврђује да ли прост број, којим се врши провера, дели број који се проверава. Ако се утврди да прост број дели број који се проверава тиме је доказано да испитивани број није прост, па се тако и означава (променљивој *prost* додели се вредност **false**), након чега се излази из петље помоћу команде **break**. У супротном испитивање се наставља следећим нађеним простим бројем мањим од 5801. Овај процес се наставља док се не нађу сви прости бројеви мањи од 2^{25} и упишу у *nizProstih*.

4.2 Паралелизација - нити

Сва четири програма намењена су да обраде велики број простих бројева. Већи број језгара на данашњим процесорима омогућавају да се ова обрада убрза тако што се више простих бројева паралелно обрађују помоћу тих језгара. У ову сврху коришћене су нити. Програми су извршавани на процесору са два језгра (Intel Core 2 Duo 2.2GHz), али су написани тако да се могу извршавати и на рачунарима са више језгара. Како би се одредио оптималан број нити потребно је доћи до информације о броју језгара. У зависности од оперативног система разликују се команде којима се добијају системске информације (број језгара у овом случају). Пошто су програми предвиђени за рад под Linux-ом користи се следећа команда:

```
int brProcesora = ( int ) sysconf( _SC_NPROCESSORS_ONLN );
```

Функција *sysconf* враћа целобројну вредност типа **long** и служи за добијање вредности неког системског лимита или опције, док се помоћу параметра *_SC_NPROCESSORS_ONLN* добија број активних језгара.

Паралелизација је решена тако што се једна нит користи за обраду једног простог броја. Поред тога што свака нит има другачији прост број за обраду оне имају и посебан простор за складиштење резултата па је тиме обезбеђено да њихов рад не зависи од других нити јер нема дељених података међу њима. Колико ће простих бројева паралелно бити обрађивано тј. колико ће нити истовремено бити креирано у једном блоку зависи од броја процесора. Следећи блок нити се креира када све нити из претходног блока заврше обраду и када се њихови резултати упишу на одговарајућа места. Прости бројеви који се истовремено обрађују су суседни па су зато разлике у временима потребним за њихову обраду занемарљиве. Постоји могућност да број простих бројева није сразмеран броју језгара. У том случају блокови се креирају док има довољно простих бројева да се сваки од њих попуни, а за просте бројеве који на крају преостану креирају се додатне нити (нпр. ако треба да се обради девет простих бројева помоћу четири језгра то значи да ће се након два блока од четири нити креирати још једна нит за последњи прост број). Овако реализована паралелизација омогућава да се максимално искористе сва језгра процесора за убрзавање извршавања програма уз минималне губитке као што су различита времена обрада простих бројева унутар једног блока и циклуси креирања и гашења нити. Кôд за рад са нитима је представљен у следећим редовима:

```
int nitiNaRedu = 0;
pthread_t niti[brProcesora];
rezultatNiti = new string[brProcesora];
zaObradu zo[brProcesora];
for( int i = 0; i < 2063689 ; i++ )
{
   /* donja_g и gornja_g чувају вредности претходно задатих
      граница интервала над којим се врши испитивање */
   if( nizProstih[i] >= donja_g
       && nizProstih[i] <= gornja_g )
   {
      // припрема простих бројева за обраду
      if( nitiNaRedu < brProcesora )
      {
         zo[nitiNaRedu].prostBr = nizProstih[i];
         zo[nitiNaRedu].redniBr = nitiNaRedu;
         nitiNaRedu++;
      }
      // блоковска обрада простих бројева
      else if( nitiNaRedu == brProcesora )
      {
```

```
            // креирање нити
            for( int k = 0; k < brProcesora; k++ )
            {
               while( pthread_create( &niti[k], NULL,
                      nit, &zo[k] ) );
            }
            // провера завршетка рада нити и очување редоследа
            for( int k = 0; k < brProcesora; k++ )
            {
               if( pthread_join( niti[k], NULL ) )
               {
                  cout << "Greska prilikom racunanja niti za
                           broj " << zo[k].prostBr << endl;
                  exit( 1 );
               }
               // upis u odgovarajući fajl
               if( rezultatNiti[k] != "" )
               {
                  izlazniFajl << rezultatNiti[k] << endl
                  << endl;
               }
            }
            nitiNaRedu = 0;
            /* у овом кораку итерације nizProstih[i] није
               припремљен за обраду већ су обрађивани
               претходно припремљени прости бројеви па је
               потребно умањити променљиву i како не би
               дошло до прескакања */
            i--;
         }
      }
      else if( nizProstih[i] >= gornja_g )
      {
         break;
      }
   }
   // обрада преосталих простих бројева
   if( nitiNaRedu != 0 )
   {
      for( int k = 0; k < nitiNaRedu; k++ )
      {
         while( pthread_create( &niti[k], NULL,
                nit, &zo[k] ) );
      }

      for( int k = 0; k < nitiNaRedu; k++ )
      {
         if( pthread_join( niti[k], NULL ) )
         {
```

```
            cout << "Greska prilikom racunanja niti za
                     broj " << zo[k].prostBr << endl;
            exit( 1 );
         }
         if( rezultatNiti[k] != "" )
         {
            izlazniFajl << rezultatNiti[k] << endl << endl;
         }
      }
   }
```

При раду са нитима коришћена је POSIX (Portable Operating System Interface) фамилија стандарда [9]. Укључивањем заглавља *pthread.h* на почетку кôда омогућен је рад са овим нитима. За њихово креирање користи се функција *pthread_ create()*. Приликом креирања нити могуће је проследити само један аргумент. Овај проблем је решен тако што је нитима прослеђена следећа структура:

```
struct zaObradu
{
   int prostBr; // прост број за који се врши обрада
   int redniBr; /* редни број знаковне ниске у коју
                   се уписује резултат */
};
```

Нитима се прослеђује показивач на аргумент тј. на структуру, али како се очекује да је то показивач на тип **void** потребно је да се у оквиру саме нити преведе у показивач на одговарајући тип. *pthread_ create()* враћа нулу ако је нит успешно креирана или број који означава грешку до које је дошло. Нека је са А означена нит из које се позива функција *pthread_ join()*, а са Б нит која се позива овом функцијом, таде се нит А суспендује док се нит Б не изврши. У случају да је нит Б већ завршила онда не долази до суспензије нити А. На овај начин се обезбеђује да све нити заврше свој задатак, као и да се очува редослед резултата. Функција *pthread_ join()* враћа нулу ако је циљана нит успешно завршила или број који означава грешку до које је дошло.

У оквиру програма за леве факторијеле и алтернирајуће суме за један прост број генерише се један резултат, док се у оквиру програма за уопштења левог факторијела и сума степенова факторијела за један прост број генерише низ резултата. Због овога постоји мала разлика у самим имплементацијама нити у оквиру ова четири програма јер се у прва два користи само једна целобројна променљива за резултат обраде док друга два програма користи низ целобројних променљивих за складиштење резултата обраде. Пошто је други случај општији овде ће бити приказана његова имплементација, конкретно кôд нити за програм који обрађује суме степенова факторијела.

```
void *nit(void *zo)
{
   stringstream ss;
   long sumeS[maxStepen];
   bool stampa = false;
   zaObradu *z = (zaObradu *) zo;

   sumStepenova( z->prostBr, sumeS );

   //излазни фајлови су у LaTeX-у
   ss << "\\textbf{" << z->prostBr << "}";
   for( int i = 0; i < maxStepen; i++ )
   {
      if( sumeS[i] <= kriterijum )
      {
         ss << " " << i + 1 << ":" << sumeS[i];
         stampa = true;
      }
      else if( sumeS[i] >= z->prostBr - kriterijum )
      {
         ss << " " << i + 1 << ":" << sumeS[i] - z->prostBr;
         stampa = true;
      }
   }

   if( !stampa )
   {
      ss.str( "" );
      ss.clear();
   }
   s[z->redniBr] = ss.str();
   return ( NULL );
}
```

У нитима се прво прослеђени показивач преводи у показивач на одговарајући тип, а потом се позива одговарајућа функција за обраду простог броја. Ове функције су карактеристичне за сваки програм и биће касније описане. Након добијања резултата проверава се да ли он задовољава критеријум исписа (о коме ће такође бити више речи касније) и, ако је то случај, смешта се у одговарајућу знаковну ниску која је резервисана за ту нит и из које се уписује у излазни фајл.

4.3 Леви факторијели и алтернирајуће суме

Програмом за леви факторијел се проверава да ли постоји прост број $p > 2$ такав да $p|!p$, односно $r(p) = 0$. Позивање овог програма се врши тако што се у командној линији, поред извршног фајла програма, уносе три аргумента који представљају доњу границу интервала над којим се врши испитивање, горњу границу тог интервала и критеријум исписа.

```
./leviFaktorijeli.out donja_g gornja_g kriterijum
```

Интервал над којим је предвиђено да се врши испитивање је $[2, 2^{25}]$, па тако није могуће унети вредности за променљиве *donja_g* и *gornja_g* ван овог интервала. Променљива *kriterijum* може да има вредности из интервала $[1, 100]$.

Нека је *rezultat* променљива која садржи вредности $r(p)$. Променљива *kriterijum* одређује које вредности $r(p)$ треба да се штампају, где се разликују следећи случајеви:

1. Ако променљива *rezultat* припада интервалу [0, *kriterijum*]
2. Ако променљива *rezultat* припада интервалу [p - *kriterijum, p*], где је p прост број који испитујемо (за ове случајеве у излазни фајл се уписује *rezultat* - p)

За израчунавање $r(p)$ користи се функција **int** *leviF*(**int** p) где је p прост број који се обрађује. Рачунање унутар ове функције се врши помоћу следећег кôда:

```
unsigned long  faktorijel = 1, rezultat = 1;

for( int i = 1; i < p; i++ )
{
   faktorijel = ( faktorijel * i ) % p;
   rezultat += faktorijel;
}
rezultat %= p;
```

Променљиве *faktorijel* и *rezultat* су типа **unsigned long** из следећа два разлога:

1. Квадрат од 2^{25} је већи од $2^{32} - 1$ (максимум за **unsigned int**), а под 64-битним Linux-ом **long** је величине 64-бита што је сасвим довољно за потребе израчунавања
2. Користи се **unsigned** јер вредности могу да буду само позитивне, а и дељење неозначених бројева је брже од дељења означених зато што се не троши време на проверу знака

Променљива *faktorijel* се иницијализује на 1 јер је $0! = 1$, а променљива *rezultat* се иницијализује на 1 јер је $!1 = 0! = 1$. Како вредности факторијела не би пробила опсег типа **unsigned long** модуо се не рачуна после добијања левог факторијела, већ у сваком кораку итерације приликом рачунања новог факторијела. За резултат није потребно рачунати модуо у сваком кораку петље јер се у променљивој *rezultat* сабирају факторијели по модулу p, а пошто тих сабирања има укупно $p - 1$ онда је загарантовано да њена вредност неће пробити поменути опсег. Шта више елиминацијом овог дељења у петљи добија се на брзини при рачунању.

Помоћу 64-битног компајлера добијен је асемблерски кôд који је искоришћен за даљу оптимизацију. Идеја је да се не користи стек приликом рачунања већ само регистри. У ту сврху написан је следећи асемблерски кôд за функцију *leviF* (под Linux-ом се користи AT & T синтакса за асемблерски језик):

```
    pushq   %r13
    pushq   %r12
       # p се проширује на 64 бита и смешта у r13
    movslq  %edi, %r13
       # у r8 се чува вредност променљиве faktorijel
    movq    $1, %r8
       # у r9 се чува вредност променљиве rezultat
    movq    $1, %r9
       # у r10 се чува вредност променљиве i
    movq    $1, %r10
    movl    $0, %r11d
    movq    $1, %r12
    jmp     .L56
.L57:
    movq    %r8, %rax
       #  i * t faktorijel
    imulq   %r10, %rax
       # доњих 32 бита се припрема за неозначено дељење
    movl    %r11d, %edx
    divq    %r13
       # у rax се смешта количник, а у rdx остатак
    movq    %rdx, %r8
       # rezultat += faktorijel
    addq    %r8, %r9
       # i++
    addq    %r12, %r10
.L56:
       # поређење p и i
    cmpq    %r13, %r10
       # ако нису једнаки враћа се у петљу
    jne     .L57
    movq    %r9, %rax
    movl    %r11d, %edx
```

```
   # рачунање модула ван петље
divq    %r13
movq    %rdx, %rax
popq    %r12
popq    %r13
ret
```

Неколико аутора је спровело проверу хипотезе за већи опсег бројева него у овом раду.

- И. Галот [6] је 2000. године извршио испитивање до 2^{26}. Тада је за израчунавање коришћен сличан алгоритам. Разлика се огледа у томе што су уместо целих бројева коришћени бројеви у покретном зарезу. Поред тога проверавала су се два проста броја у истој петљи.
- П. Цоблинг [7] је 2004. године наставио истраживање са горњом границом $p < 144000000$ при чему је коришћен исти начин провере као и код Галота.
- М. Татаревић [8] је 2011. године подигао границу до 10^9. За ово истраживање коришћена је 64-битна архитектура, проверавана су четири броја истовремено, а кôд је писан у u C-u и асемблеру.

Програмом за алтернирајуће суме се проверава да ли $p|A_{p-1}$ (односно да ли је $a(p) = 0$) где је p један од простих бројева из задатог интервала. Позивање из командне линије се врши на исти начин као и код програма за леви факторијел, тј. поред назива извршног фајла очекују се три аргумента који означавају доњу и горњу границу интервала, као и критеријум исписа.

```
./alternirajuceSume.out donja_g gornja_g kriterijum
```

Разлика између програма за леве факторијеле и програма за алтернирајуће суме се огледа у томе што је функција **int** *leviF*(**int** p) замењена функцијом **int** *altSuma*(**int** p), а њен кôд је:

```
unsigned long faktorijel = 1;
long rezultat = 0;
bool oduzimanje;

if( p == 2 )
   oduzimanje = false;
else
   oduzimanje = true;

for( int i = 1; i < p; i++ )
{
   faktorijel = ( faktorijel * i ) % p;
```

```
      if( oduzimanje )
      {
         rezultat -= faktorijel;
         oduzimanje = false;
      }
      else
      {
         rezultat += faktorijel;
         oduzimanje = true;
      }
   }
   rezultat %= p;
   if( rezultat < 0 )
      rezultat += p;
```

Променљива *rezultat* се поставља на нулу јер алтернирајуће суме крећу од 1! тј. сви факторијел се рачунају унутар петље па нема потребе да се *rezultat* иницијализује на неку претходну вредност. *rezultat* је овде типа **long** јер може да има позитивне и негативне вредности. Уведена је буловска променљива *oduzimanje* којом се одређује да ли ће доћи до одузимања у корацима итерације. Како су сви прости бројеви p, где је $p > 2$, непарни онда су сви $p - 1$ парни. Ова чињеница се користи за иницијализацију буловске променљиве *oduzimanje* јер ако је $p-1$ паран број онда се у првом кораку итерације долази до одузимања у формирању алтернирајуће суме A_{p-1}. Коначан резултат рачунања, користећи описани алгоритам, може да има и негативну вредност па се, пошто је рачунање по модулу p, на негативан резултат додаје вредност p како би се он превео у позитиван домен.

Ова функција је такође оптимизована помоћу асемблера:

```
pushq   %r14
pushq   %r13
pushq   %r12
   # p се проширује на 64 бита и смешта у r13
movslq  %edi, %r13
movq    $0, %r12
movq    $1, %r14
   # у r8 се чува вредност променљиве faktorijel
movq    $1, %r8
   # у r9 се чува вредност променљиве rezultat
movq    $0, %r9
   # провера да ли је p == 2
cmpl    $2, %edi
jne     .L56
   # доњих 8 бита r11 чува вредност променљиве oduzimanje
movb    $0, %r11b
jmp     .L57
```

```
.L56:
   movb    $1, %r11b
.L57:
      # у r10 се чува вредност променљиве i
   movq    $1, %r10
   jmp     .L58
.L61:
   movq    %r8, %rax
      # i * faktorijel
   imulq   %r10, %rax
      # припрема за неозначено дељење факторијела
   movl    %r12d, %edx
   divq    %r13
      # у rax се смешта количник, а у rdx остатак
   movq    %rdx, %r8
      #провера да ли је одузимање 0 тј. false
   cmpb    %r11b, %r12b
   je      .L59 &
      # rezultat -= faktorijel
   subq    %r8, %r9
      # одузимање постаје 0 тј. false
   movb    %r12b, %r11b
   jmp     .L60
.L59:
      # rezultat += faktorijel
   addq    %r8, %r9
      # одузимање постаје 1 тј. true
   movb    %r14b, %r11b
.L60:
      # i++
   addq    %r14, %r10
.L58:
      # поређење p и i
   cmpq    %r13, %r10
      # враћа се у петљуа ко нису једнаки
   jne     .L61
   movq    %r9, %rax
      # доњих 32 бита rdx се припрема за означено дељење
   movq    %rax, %rdx
      # шифтовање у десно како би у rdx остао само знак
   sarq    $63, %rdx
      # означено рачунање модула ван петље
   idivq   %r13
   movq    %rdx, %rax
      # поређење 0 са резултатом
   cmpq    %rax, %r12
      # ако је 0 <= rezultat онда нема потребе да се дода p
   jle     .L62
      # rezultat += p
   addq    %r13, %rax
```

```
.L62:
   popq    %r12
   popq    %r13
   popq    %r14
   ret
```

4.4 Уопштења и суме степенова факторијела

Следећа два програма приликом позивања из командне линије очекују, поред описана три аргумента за претходне програме, и четврти аргумент. Програм за уопштења левог факторијела, односно за израчунавање бројева $R(n,k)$, као четврти аргумент очекује ред уопштења до кога ће се генерисати резултати, смешта га у променљиву *maxRed*, а његове вредности могу бити из интервала [1, 100].

```
./uopsteni.out donja_g gornja_g kriterijum maxRed
```

Код програма за суме степенова факторијела, односно за израчунавање бројева $Q(n,k)$, четврти аргумент представља степен до кога ће ићи израчунавања, његова вредност се смешта у променљиву *maxStepen* и може бити из интервала [1, 256].

```
./sumeStepenova.out donja_g gornja_g kriterijum maxStepen
```

Алгоритам израчунавања уопштења левог факторијела је надоградња алгоритма за рачунање левог факторијела.

```
unsigned long faktorijel = 1;

for( int i = 0; i <= maxRed; i++ )
{
   redovi[i] = 1;
}
for( int i = 1; i < p; i++ )
{
   faktorijel = ( faktorijel * i ) % p;
   redovi[0] = ( redovi[0] + faktorijel ) % p;
   for( int j = 1; j <= maxRed; j++ )
   {
      redovi[j] = ( redovi[j] + redovi[j - 1] ) % p;
   }
}
```

У помоћном низу *redovi* чувају се резултати израчунавања редова уопштења левог факторијела броја p по модулу тог броја. Дужина овог низа је *maxRed*+1 јер се на првом месту у низу чува вредност левог

факторијела броја p, по модулу тог броја, док се на преосталим местима чувају редови уопштења левог факторијела. У сваком кораку итерације редови уопштења се рачунају тако што им се додаје вредност претходно израчунатог реда (почетно се сви иницијализују на вредност 1 јер је $L(1,k)=1$).

У основи алгоритма за рачунање сума степенова факторијела налази се алгоритам за рачунање левог факторијела броја p по модулу тог броја. Низ променљивих *sumeS* чува резултате израчунавања сума степена факторијела (од првог степена до степена чија се вредност налази у променљивој *maxStepen*) по модулу броја p. Уведен је и низ променљивих *faktorijel* у коме се чувају сви степенови (од један до *maxStepen*) тренутног факторијела у итерацији.

Алгоритам за рачунање сума степенова факторијела је следећи:

```
unsigned long faktorijel[maxStepen], ij;

for( int i = 0; i < maxStepen; i++ )
{
   sumeS[i] = 1;
   faktorijel[i] = 1;
}
for( int i = 1; i < p; i++ )
{
   ij = 1;
   for( int j = 0; j < maxStepen; j++ )
   {
      ij = ( ij * i ) % p;
      faktorijel[j] = ( faktorijel[j] * ij ) % p;
      sumeS[j] = ( sumeS[j] + faktorijel[j] ) % p;
   }
}
```

4.5 Добијени резултати

Овде су представљени резултати добијени радом претходно описаних програма. Сви резултати су добијени по модулу простог броја за који су рачунати, па се због једноставности записа користе већ споменуте ознаке:

- $r(p)=\sum_{i=0}^{p-1} i! \mod p$ (1)
- $a(p)=\sum_{i=1}^{p-1}(-1)^{p-1-i} i! \mod p$ (2)
- $R(p,k)=\sum_{i=1}^{p}(p-i)!\binom{i+k-1}{k} \mod p$ (3)
- $Q(p,k)=\sum_{i=0}^{p-1} i!^k \mod p$ (4)

Програм за леве факторијеле, тј. за испитивање постојања простог броја $p>2$ за који важи $p|!p$, је пуштен у рад над интервалом $[2,2^{25}]$, са критеријумом исписа 10. Време рада програма је било око 95 сати, а добијени резултати приказани су у Табели 2.

p	$r(p)$	p	$r(p)$	p	$r(p)$
2	0	**71**	-3	**4337**	-5
3	1	**113**	-4	**7717**	7
5	4	**139**	-5	**10331**	-2
7	6	**151**	10	**11813**	6
11	1	**163**	4	**33703**	9
13	10	**197**	9	**77687**	-3
17	-4	**227**	-2	**126323**	-8
19	9	**277**	7	**274453**	-1
23	-2	**349**	-6	**2275843**	3
31	2	**373**	2	**3467171**	5
37	5	**467**	3	**4709681**	-9
41	4	**661**	-10	**11477429**	9
67	-2	**2437**	-5		

Табела 2: Прости бројеви p за које $r(p)$ и $r(p) - p$ имају апсолутне вредности ≤ 10

Према очекивањима прости број, већи од 2, који дели свој леви факторијел није пронађен, док су резултати претходних истраживања на овом интервалу верификовани.

Алтернирајуће суме факторијела су провераване за просте бројеве из интервала $[2, 2^{25}]$, критеријум исписа је био 10, а резултати се налазе у Табели 3.

p	$a(p)$	p	$a(p)$	p	$a(p)$
2	1	**67**	5	**983**	-3
3	1	**71**	-7	**1109**	2
5	4	**79**	4	**1439**	-2
7	3	**109**	-5	**2741**	3
11	4	**131**	-3	**3559**	3
13	-1	**157**	6	**9439**	-10
17	8	**191**	6	**11119**	-3
19	-5	**197**	-2	**16007**	-4
23	-5	**229**	-9	**22619**	-3
29	-10	**307**	5	**32833**	-6
31	9	**367**	-4	**52201**	10
37	-1	**463**	-1	**394249**	1
41	1	**641**	3	**2934901**	1
43	5	**647**	5	**3515839**	-2
47	6	**691**	-2	**3612703**	0

Табела 3: Прости бројеви p за које $a(p)$ и $a(p) - p$ имају апсолутне вредности ≤ 10

Добијени резултати се поклапају са резултатима из [3]. Прост број $p = 3612703$ дели своју алтернирајућу суму $(\sum_{i=1}^{p-1}(-1)^{p-1-i}i!)$ и представља решење проблема Б43 у књизи "Нерешени проблеми у теорији бројева" [2]. Програму је било потребно око сат и четрдесет минута да обради просте бројеве из интервала $[2, 3612703]$, а око 102 сата да обради просте бројеве из интервала $[2, 2^{25}]$.

Испитивање уопштења левог факторијела је извршено за просте бројеве из интервала $[2, 10^7]$. Критеријум исписа и максимални ред уопштења левог факторијела су били постављени на 10, а време обраде простих бројева је било око 139 сати. Добијени резултати приказани су у Табели 4, при чему су изостављени редови у којима је само $R(p, 1) = 1$.

$p \setminus k$	$0(!p)$	1	2	3	4	5	6	7	8	9	10
2	0	1	0	1	0	1	0	1	0	1	0
3	1	1	2	1	1	2	1	1	2	1	1
5	4	1	4	3	4	4	1	4	3	4	4
7	6	1	5	6	0	5	6	6	1	5	6
11	1	1	6	0	1	6	0	9	2	9	10
13	10	1	9	7	7	3	10	9	7	3	3
17	-4	1	3	9	7	4	-2	-2	-6	10	3
19	9	1	6	1	-3	2	9	-8	9	2	5
23	-2	1	2	2	-10	10	2	8	-5	7	1
29		1	7	-1		-10	-5	0	8		1
31	2	1	0		-1	1		-1	-2		-3
37	5	1				0	-3			-5	
41	4	1	-1	-2	8	-2	-10	8	-8	6	
43		1	-7	-10	1	1	8		-6		
47		1	-8			4	8			-6	
53		1		-8							-8
61		1	-10				-1	-5		4	
67	-2	1	2	2					-2		
71	-3	1				6					-5
73		1	10				-10			-7	
79		1	-9								0
89		1				0	6				
97		1					10		-5		-8
101		1		-8							
103		1				4					-4
107		1								-7	
109		1									0
113	-4	1	3					-1	-6		
127		1			-3		-6				
131		1				-3					
137		1				-7					
139	-5	1		4							

Табела 4: Прости бројеви p за које $R(p, k)$ и $R(p, k) - p$ имају апсолутне вредности ≤ 10

$p \setminus k$	$0(!p)$	1	2	3	4	5	6	7	8	9	10
151	10	1	-4	-6	-6		-3				
157		1							10		
163	4	1	-1	-2		-2					
167		1				-3					
173		1				9					
181		1		8		5					
191		1		-5					9	-8	
197	9	1									
199		1				-4					
223		1				10				6	
227	-2	1	2	2							
229		1								-3	
233		1							0		
239		1	-5								
269		1	-7	-10							
271		1					4				
277	7	1		-4			-8				
281		1					6				
313		1	8	10	9			-9			
317		1			-9						
337		1					-10				
349	-6	1	4		4						
359		1								5	
367		1						-1			
373	2	1	0		-1						
379		1			2						
383		1							4		
389		1				6	-4				
421		1						-9			
463		1						-10			
467	3	1									
479		1			3						
503		1		5			-4				
523		1		8		5					
541		1		-5					-8		
587		1				0			-6		
601		1					6				
613		1						-7			
647		1	10								
659		1				4	-6				0
661	-10	1	6								
673		1									2
677		1					-8				
691		1			8						
733		1						-10			

Табела 4: (наставак)

$p \setminus k$	$0(!p)$	1	2	3	4	5	6	7	8	9	10
787		1						6			
823		1	-10								
919		1		-5							
941		1		7							
977		1					-6				
997		1						7			
1061		1									-5
1129		1								-4	
1291		1							-2		
1301		1			-7						
1423		1							7		
1439		1									7
1499		1	-8								
1523		1						-1			
1531		1				-8					
1559		1	-5								
1657		1								-2	
1723		1			-7						
2089		1							0		
2099		1							-4		
2207		1								-10	
2347		1	7								
2389		1								1	
2417		1		7							
2437	-5	1		4							
2503		1				1					
2621		1	-7	-10							
2659		1									-6
2777		1			-9						
2917		1					1				
2963		1			-8						
3041		1					-1				
3137		1					-2				
3257		1									6
3697		1								9	
3889		1					6				
4243		1									4
4337	-5	1		4							
4421		1							-9		
4817		1						-7			
4877		1				-1					
5437		1									0
5801		1							0		
6163		1								4	
6317		1						5			

Табела 4: (наставак)

$p \setminus k$	$0(!p)$	1	2	3	4	5	6	7	8	9	10
6701		1							-8		
6703		1					-5				
6779		1			-2						
6961		1					5				
6997		1									-10
7127		1									-5
7207		1									4
7639		1					9				
7717	7	1		-4							
7753		1			10						
8543		1						-10			
8893		1							1		
8941		1				7					
9127		1					-4				
9137		1							6		
9521		1		7							
9721		1	-10								
10331	-2	1	2	2							
11329		1							-1		
11813	6	1	-2								
11933		1							-4		
13327		1							-8		
15031		1				-1					
15391		1		8		5					
16229		1	10								
17137		1	-5								
19963		1								0	
22123		1									8
22817		1	-6								
24043		1						3			
25579		1		-5							
25603		1				6					
25997		1				-10					
26693		1				2					
30241		1								6	
31033		1		3							
32831		1								6	
33703	9	1									
37561		1							-4		
37781		1								6	
41011		1	7								
41617		1							2		
47569		1									5
52289		1	8	10	9						
53597		1				-9					

Табела 4: (наставак)

$p \setminus k$	$0(!p)$	1	2	3	4	5	6	7	8	9	10
53629		1								10	
56417		1					-6				
57383		1			-2						
57557		1							5		
58693		1							-7		
59393		1						-3			
61729		1				1					
68087		1						-6			
70583		1					7				
73643		1							-2		
77687	-3	1									
89087		1							-1		
92957		1					7				
96017		1		5							
97547		1									10
100019		1								10	
109379		1			-8						
121021		1						8			
125423		1						-4			
126323	-8	1	5	6							
151471		1			6						
162221		1						-9			
191537		1									-2
195089		1					8				
217001		1				-6					
218479		1					4				
240587		1						4			
265123		1								4	
274453	-1	1									
286163		1			10						
337367		1							4		
360749		1								-9	
390743		1									6
402037		1							6		
429673		1						0			
455291		1							10		
513839		1	-7	-10							
524969		1								0	
536423		1			1						
550691		1					-4				
554977		1								-1	
861547		1						-3			
880687		1									7
905381		1						7			
929717		1					2				

Табела 4: (наставак)

$p \setminus k$	$0(!p)$	1	2	3	4	5	6	7	8	9	10
964889		1				4					
1038671		1								5	
1064269		1				2					
1111711		1							7		
1215367		1		-8							
1309591		1						-3			
1318931		1									9
1886657		1				4					
2240449		1						1			
2256731		1			2						
2275843	3	1									
2240449		1						1			
2256731		1			2						
2275843	3	1									
3467171	5	1									
3579691		1		3							
3606661		1		7							
3617059		1					0				
3784447		1								5	
3936827		1	10								
4709681	-9	1									
4910579		1					2				
5713709		1							-4		
5926457		1								-3	
5933167		1					-10				
6672947		1		8		5					
7497593		1					1				
7728067		1				-3					
7895207		1								7	
8542519		1							-2		

Табела 4: (наставак)

Добијени резултати показују да за свако проверено p за уопштење левих факторијела првог реда $L(p,1)$ важи $L(p,1) \pmod p = 1$. Испоставља се да је ово тврђење које важи за све природне бројеве.

Теорема 4. За сваки природан број n важи $R(n,1) = 1$.

Доказ Израз $L(n,1)$ може се раздвојити у два сабирка на следећи начин:

$\sum_{k=1}^{n} k(n-k)! = \sum_{k=0}^{n-1}(n-k)k! = n\sum_{k=0}^{n-1} k! - \sum_{k=0}^{n-1} kk!$
Како је $\sum_{k=0}^{n-1} kk! = \sum_{k=0}^{n-1}(k+1)! - k! = n! - 0!$
добија се да је
$\sum_{k=1}^{n} k(n-k)! = n\sum_{k=0}^{n-1} k! - n! + 0! = 1 + n\sum_{k=0}^{n-2} k! \equiv 1 \pmod n$.

$p \setminus k$	$0(!p)$	1	2	3	4	5	6	7	8	9	10
2	0		0		0		0		0		0
7					0						
11				0			0				
29								0			
31			0								
37						0					
79											0
89						0					
109											0
233									0		
373			0								
587						0					
659											0
2089									0		
5437											0
5801									0		
19963										0	
429673								0			
524969										0	
4709681							0				

Табела 5: Прости бројеви p за које је $R(p,k) = 0$

Распоред нула у Табели 5 нема никакву видљиву правилност, што је слично понашању левих факторијела и алтернирајућих сума факторијела.

k	$0(!p)$	1	2	3	4	5	6	7	8	9	10
p			31	11	7	37	11	29	233	19963	79

Табела 6: Први прости бројеви $p > 2$ за које је $R(p,k) = 0$ и $0 \leq k \leq 10$

Суме степенова факторијела $S(p,k)$, односно њихови остаци $Q(p,k)$ су испитивани за просте бројеве из интервала $[2, 10^6]$, са критеријумом исписа 10 и максималним степеном 256. Време потребно за обраду простих бројева је било око 74 сата, а резултати овог експеримента за $k \leq 16$ се налазе у Табели 7.

$p \setminus k$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
5	4	3	0	0	4	3	0	0	4	3	0	0	4	3	0	0
7	6	4	1	4	3	0	6	4	1	4	3	0	6	4	1	4
11	1	4	7	3	10	3	3	4	6	0	1	4	7	3	10	3
13	10	5	-1	1	2	-1	5	1	9	5	0	0	10	5	-1	1
17	-4	-1	-1	3	-5	-6	0	5	2	-6	8	3	7	-1	8	0
19	9	-5	-2	8	9	-2	8	-7	7	-7	9	-2	5	8	-2	-5
23	-2	9	4	1	-2	0	-4	2	8	3	7	3	-5	2	-9	0
29		10	-4	-7				-7	0			-5			-1	-5
31	2	10	9	-2	-9			-5		7	-10	-9		5	-1	5
37	5	-8		4		-3		4		10		1	-5	4	-2	4
41	4		7		10		-8			1	1			7		
43		-8	6			-6	-3	-9	-3				9	4		1
47				10	5		10		0	10		6	7	-7	0	
53		8	-10		-9							-4	-6	-6		
59		6	7	-4					1		7	6	-10		-8	
61			4		-6		-5		6	-1						
67	-2		5	6					-3	2			-6	-5	-10	
71	-3		2	-9					-1		-10	-10				
73		-4				6	5	8		0		5				
79						5	-2				6					
83		7		7				10				-3				
89		4					-8	3				2			-5	
97		-3												-10	6	-2
101		10					6			-10		-8			4	
103			-3		-7							-4	-4			
107		4		-6					4	-9						
109															-3	
113	-4				10							-7	-1			
127													5			
131		2					-6									
137					5	-9										
139	-5									7						
149				0					6				-4			
151	10							-10		-9						
157		9				-10										
163	4															9
167													3			
173					6				7							
179		8									-1					
181						0								-2		
191								3	-4							

Табела 7: Прости бројеви p за које $Q(p,k)$ и $Q(p,k)-p$ имају апсолутне вредности ≤ 10

$p \setminus k$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
193													-5			4
197	9					7			9			1				
211			7					-10					-9	-6		
223							-8									
227	-2					2	-1		6							
233															-8	
239		4	4													
241							6	5						-2	1	
251						3										
257							-3									
263						-2			-1		-3					
269									6							
271															1	
277	7					9										
293		-4				-9			6					-5		
311				4		-8							6			
317															6	
331													-8			
347							-8							-9		
349	-6		5													
353										1	-4					
367								0								
373	2		6		-9										1	
401										-1						
421												-7				
433															1	
443						0								-10		
449									2			-9			2	
463							-6		10					1		
467	3	5														
487											-2					
499															-2	
521					9	2										
523														8		
541												-6				
557									-6							
563								0								
571		4														
577												4		-1		
599												5				4
607						-10										
643											-6	-6				
659																8
661	-10															

Табела 7: (наставак)

$p \setminus k$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
673			-10									-2				
751								-8								
761						2										
769						-5										
773		6														
829		-4														
881				5					9							
971										8						
1009												-5				
1013			3								0					
1019													3			
1049												0			-3	
1051										7						
1087								4								
1091						1										
1097					10											
1129			3							-9						
1181									6							
1213					2											
1303						0										
1373			9													
1433		9														
1453		3														
1487							-7									
1499					1											
1621									-6							
1637											-4					

Табела 7: (наставак)

Запажа се периодичност у врстама Табеле 7: редом за $p = 2, 3, 5, 7, 11, 13$ дужина периода је $1, 2, 4, 6, 10, 12$. Поред тога, последњи број у периоди је у наведеним случајевима 0. Појаву објашњава следећа теорема.

Теорема 5. За сваки прост број p низ бројева $Q(p,k), k = 1, 2, 3, \ldots$ је периодичан са периодом дужине $p-1$, односно $Q(p, k+p-1) = Q(p,k)$. Поред тога важи једнакост $Q(p, p-1) = 0$.

Доказ Према малој Фермаовој теореми за прост број p и цео број a узајамно прост са p важи $a^{p-1} \equiv 1 \pmod{p}$. Поменути низ је периодичан јер за произвољно k важи
$Q(p, k+p-1) = \sum_{i=0}^{p-1} i!^{k+p-1} = \sum_{i=0}^{p-1} i!^k = Q(p,k)$
За суму степенова факторијела, чији је степен облика $k = j(p-1)$, за произвољно $j \in N$, важи следеће:
$S(p,k) = 0!^{j(p-1)} + 1!^{j(p-1)} + \ldots + (p-1)!^{j(p-1)} = p \cdot 1 \equiv 0 \pmod{p}$ тј. у низу $Q(p,k)$, $k = 1, 2, \ldots$ нуле су чланови са индексима $j(p-1)$ за сваки природан број j.

У наредној табели приказани су парови (p, k) за које је $Q(p, k) = 0$.

p	k	p	k	p	k	p	k
1747	134	**5297**	201	**29077**	142	**118709**	118
1787	214	**5431**	161	**29209**	29	**124489**	65
1867	155	**5683**	188	**30187**	221	**140761**	135
1987	132	**5779**	243	**30529**	115	**152843**	14
1993	124	**5987**	244	**30881**	144	**161459**	48
1999	98	**6197**	75	**34549**	200	**163601**	129
2099	103	**6361**	177	**36269**	44	**169837**	207
2137	24	**6637**	35	**36389**	140	**181087**	245
2207	158	**7417**	69	**38119**	42	**191531**	7
2309	159	**7561**	37	**42943**	250	**199819**	253
2309	238	**7793**	56	**43063**	166	**210827**	103
2393	122	**7817**	206	**44617**	75	**213091**	186
2477	80	**7867**	143	**47309**	120	**236381**	80
2551	96	**8081**	215	**47339**	198	**251539**	82
2551	149	**8377**	189	**48073**	16	**267961**	249
2617	46	**8447**	79	**51869**	221	**290057**	185
2671	103	**9619**	240	**54377**	10	**290369**	204
2719	194	**10433**	162	**55399**	62	**298159**	207
2833	43	**10499**	28	**56263**	87	**306739**	157
3023	6	**10607**	135	**60089**	108	**321109**	155
3049	81	**10631**	37	**62143**	78	**322769**	49
3191	179	**11057**	127	**62213**	137	**340709**	27
3307	69	**11827**	250	**65609**	156	**341179**	76
3331	19	**11981**	173	**67957**	250	**342203**	177
3529	102	**12503**	37	**73523**	224	**343163**	199
3539	228	**13171**	129	**73897**	161	**469939**	155
3613	31	**13789**	116	**74189**	170	**500671**	119
3617	142	**13859**	249	**74323**	133	**553607**	99
3709	135	**14293**	29	**75337**	226	**586541**	34
3823	100	**15373**	164	**81647**	57	**599891**	193
4027	250	**16231**	98	**86753**	214	**602081**	173
4111	156	**16361**	176	**88117**	19	**616157**	191
4339	106	**17093**	197	**90917**	241	**672029**	37
4373	159	**17203**	17	**91457**	206	**687977**	150
4451	79	**18211**	97	**97423**	193	**782501**	217
4457	72	**19841**	56	**97967**	194	**786673**	58
4567	125	**20771**	10	**98999**	197	**796541**	28
4673	168	**20879**	2	**100669**	119	**799873**	87
4759	82	**22279**	170	**104207**	89	**822589**	249
4759	235	**23333**	49	**104383**	229	**823309**	45
4871	80	**25409**	67	**104597**	50	**842267**	209
4967	227	**26723**	62	**104831**	194	**843043**	220
4969	202	**27479**	16	**111593**	248	**987851**	113

Табела 8: Парови (p, k) такви да је $Q(p, k) = 0$, $p \geq 1747$, $1 \leq k \leq 256$

Наредна табела за свако k садржи први прост број p за који је $Q(p,k)$ нетривијална нула (нула која није последица Теореме 5).

p	k	p	k	p	k	p	k
	1	**89**	33	**29**	65	**18211**	97
20879	2	**586541**	34		66	**313**	98
5	3	**5**	35	**5**	67	**5**	99
149	4	**479**	36	**211**	68	**3823**	100
	5	**29**	37	**3307**	69	**47**	101
23	6	**23**	38		70	**3529**	102
5	7	**5**	39	**5**	71	**5**	103
367	8		40	**4457**	72	**23**	104
29	9	**487**	41		73	**131**	105
73	10	**313**	42		74	**757**	106
5	11	**5**	43	**5**	75	**5**	107
1049	12	**103**	44	**296**	76	**60089**	108
	13	**233**	45		77		109
152843	14	**2617**	46	**62143**	78	**271**	110
5	15	**5**	47	**5**	79	**5**	111
23	16	**161459**	48	**1087**	80		112
17203	17	**859**	49	**229**	81	**987851**	113
109	18	**23**	50	**23**	82		114
5	19	**5**	51	**5**	83	**5**	115
	20		52		84	**23**	116
	21	**59**	53	**61**	85		117
1009	22		54		86	**118709**	118
5	23	**5**	55	**5**	87	**5**	119
2137	24	**7793**	56		88	**47309**	120
61	25	**109**	57	**104207**	89	**29**	121
149	26	**103**	58	**109**	90	**149**	122
5	27	**5**	59	**5**	91	**5**	123
23	28	**23**	60	**367**	92	**1993**	124
14293	29	**47**	61	**29**	93	**4567**	125
	30	**73**	62	**23**	94	**23**	126
5	31	**5**	63	**5**	95	**5**	127
	32		64	**2551**	96	**1259**	128

Табела 9: Први прости бројеви $p > 2$ за које је $Q(p,k) = 0$, $1 \leq k \leq 256$, $k \neq j(p-1) \forall j \in N$

p	k	p	k	p	k	p	k
13171	129	**5431**	161	**47**	193	**83**	225
	130	**113**	162	**733**	194	**23**	226
5	131	**5**	163	**5**	195	**5**	227
1987	132	**15373**	164		196	**13**	228
631	133	**109**	165	**17093**	197	**104383**	229
1747	134	**43063**	166	**109**	198		230
5	135	**5**	167	**5**	199	**5**	231
	136	**1381**	168	**34549**	200	**1093**	232
62213	137	**59**	169	**281**	201	**29**	233
23	138	**23**	170	**4969**	202	**109**	234
5	139	**5**	171	**5**	203	**5**	235
521	140		172	**23**	204	**23**	236
	141	**131**	173	**29**	205		237
211	142	**181**	174	**73**	206	**2309**	238
5	143	**5**	175	**5**	207	**5**	239
149	144	**16361**	176		208	**9619**	240
61	145	**29**	177	**89**	209	**90917**	241
103	146		178	**1013**	210		242
5	147	**5**	179	**5**	211	**5**	243
23	148		180		212	**953**	244
29	149		181		213	**47**	245
687977	150	**23**	182	**23**	214		246
5	151	**5**	183	**5**	215	**5**	247
149	152		184	**571**	216	**23**	248
47	153	**290057**	185	**782501**	217	**283**	249
73	154	**181**	186		218	**4027**	250
5	155	**5**	187	**5**	219	**5**	251
4111	156	**5683**	188	**373**	220		252
863	157	**8377**	189	**30187**	221	**521**	253
2207	158		190		222		254
5	159	**5**	191	**5**	223	**5**	255
23	160	**23**	192	**73523**	224		256

Табела 9: (наставак)

Распоред нула у Табели 9 нема никакву видљиву правилност.

5 Закључци и даљи рад

У овом раду су представљена четири програма за испитивање левог факторијела, алтернирајуће суме факторијела, уопштења левог факторијела и суме степенова факторијела. Резултати за леви факторијел и алтернирајуће суме факторијела су били очекивани тј. на испитиваном интервалу није пронађен прост број $p > 2$ такав да $p|!p$, а резултати из [3] су потврђени. Код уопштења левог факторијела откривено је да за сваки природан број n важи $L(n, 1) \equiv 1 \pmod{n}$ (Теорема 4), док је за суме степенова факторијела потврђена периодичност резултата (Теорема 5). Слично као код левог факторијела, ни код уопштења левог факторијела, ни код сума степенова факторијела није уочена никаква правилност у вези са дељивостима одговарајућим хипотези о левом факторијелу.

Могући правац даљег рада у овој области могао би да буде коришћење вишепроцесорских рачунара. У основи овог правца истраживања је огроман број израчунавања па се тако са повећањем сирове процесорске снаге смањује време потребно за та израчунавања. Обрада великих бројева, коју ограничава данашњи хардвер, би требала да буде могућа у будућности.

Литература

[1] Đuro Kurepa - *On the left factorial function*, Mathematica Balkanica, **1** (1971), 147-153.

[2] Richard Guy - *Unsolved Problems in Number Theory*, Third Edition, New York: Springer-Verlag, 2004.

[3] Miodrag Živković - *The number of primes* $\sum_{i=1}^{n}(-1)^{n-i}i!$ *is finite*, Math. Comp. **68** (1999), no. 225, 403–409.

[4] James Anderson - *Diskretna matematika sa kombinatorikom*, prevod drugog izdanja, Beograd: CET, 2005.

[5] Žarko Mijajlović - *On some formulas involving !n and the verification of the !n hypothesis by use of computers*, Publ. Inst. Math (Beograd) **47**(61), 1990, 24-32.

[6] Yves Gallot - *Is the number of primes* $\frac{1}{2}\sum_{i=0}^{n-1} i!$ *finite?*, 2000. `http://yves.gallot.pagesperso-orange.fr/papers/lfact.html`

[7] Paul Jobling - *A couple of searches*, 2004. `http://tech.groups.yahoo.com/group/primeform/message/5095`

[8] Miloš Tatarević - *Searching for a counterexample to the Kurepa's left factorial hypothesis* ($p < 10^9$), 2011. `http://mtatar.wordpress.com/2011/07/30/kurepa/`

[9] POSIX `http://pubs.opengroup.org/onlinepubs/9699919799`

[10] Randall Hyde - *The Art of Assembly Language Programming*, 1996 `http://maven.smith.edu/~thiebaut/ArtOfAssembly/artofasm.html`