ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ v PRAZE

Fakulta dopravní
Katedra aplikované matematiky

Úlohy

z předmětu

Matematické algoritmy

Autor práce: **Ondra Havel**
Školní rok: **zima 2004/2005**

leden 2005

Obsah

1 Faktorizace Fermatových čísel

1.1 Algoritmy pro faktorizaci

CFRAC. Continued fraction factorization je obecný algoritmus pro faktorizaci celých čísel. V roce 1975 jej vyvinuli Michael A. Morrison a John Brillhart. CFRAC je založen na Dixonově faktorizační metodě (modifikace Fermatova faktorizačního algoritmu). CFRAC používá nejlepších přiblížení řetězového zlomku

$$\sqrt{kn}, \quad k \in Z^+$$

Z algoritmu pro rozklad odmocniny z celého čísla do řetězového zlomku je patrné, že rozvoj je periodický. Např. pro $\sqrt{F_6}$ dostaneme

$$\sqrt{2^{64}+1} = 2^{32} + \sqrt{2^{64}+1} - 2^{32} = 2^{32} + \frac{1}{2^{32} + \sqrt{2^{64}+1}}$$

Odsud $\sqrt{2^{64}+1} = [2^{32}, \overline{2^{33}}]$.

GNFS. General number field sieve je v současné době algoritmus s nejnižší časovou složitostí pro nynější počítače.

$$O\Big(\exp\Big(\Big(\frac{64}{9}n\Big)^{\frac{1}{3}}(\log n)^{\frac{2}{3}}\Big)\Big)$$

Shorův algoritmus. V roce 1994 objevil Peter Shor algoritmus, který pracuje v polynomiálním čase na kvantových strojích. Jeho časová složitost je $O(n^3)$, pamětová náročnost $O(n)$. V roce 2001 došlo k první praktické faktorizaci čísla 15 na experimentalním 7mi quibitovém počítači.

1.2 Faktorizace F_6

O F_6 víme, že má 20 cifer, 2 faktory (6, 14) – Landry 1880. Faktor má tvar $2^{n+1}k + 1$, $(k \in Z)$ – Euler, 1770. Přesněji $2^{n+2}l + 1$, $l \in Z$ – Lucas, 1878. K faktorizaci F_6 použijeme pro jednoduchost algoritmus hrubé síly, budeme testovat čísla $2^8 \cdot l + 1$ pro $l \in [2^0; 2^{24}]$. Skript je napsán v Perlu s využitím knihovny pro interpretaci a operace s čísly s libovolnou přesností GMP[1].

```
#!/usr/bin/perl

use strict;
use warnings;
use GMP::Mpz qw/:all/;
```

[1] http://swox.com/gmp

```
my $k = mpz(2) ** 64 + 1;
print "F(6) = 2^64 + 1 = $k\n";

my($d,$f);

for (1 .. 2**24) {
    $d = mpz($_) * 2 ** 8 + 1;
    next if $k % $d;
    $f=$d;
    last;
}

if(!defined $f) {
    print "No factors found\n";
}

print "F(6) = $d * ", $k/$d, "\n";
```

Výsledek dostaneme už pro $l = 1071$.
Výstup programu:

```
F(6) = 2^64 + 1 = 18446744073709551617
F(6) = 274177 * 67280421310721
```

2 Řetězové zlomky

2.1 Nejlepší přiblížení

Převedeme čísla $\frac{1+\sqrt{5}}{2}$, $\sqrt{3}$, $\sqrt{2}$ a π. Pro výrazy obsahující odmocniny můžeme koeficienty řetězových zlomků spočítat ručně, posloupnost bude periodická. Pro číslo π získáme dostatečně dlouhé vyjádření. Vzhledem k použití standardních datových typů pro reprezentaci čísel bude stačit zhruba 14 desetinných míst[2]. Koeficienty čísla π snadno vygenerujeme skriptem uvedeným níže.

Pro generování postupných přiblížení a spočtení chyby použijeme následují skript. Na vstupu jsou koeficienty pravého řetězového zlomku a "správná hodnota" (hodnota vzhledem ke které se počítá chyba) je uvedena jako argument.

Pro výpočet jednotlivých postupných přiblížení použijeme vzorec vyplývající z maticového zápisu pro výpočet koeficientů. Pro čitatel i jmenovatel je vzorec shodný, liší se pouze startovací členy posloupnosti. Pro čitatel $(1, a_1)$, pro jmenovatel $(0, 1)$, kde a_i je i-tý koeficient pravého řetězového zlomku.

$$x_i = a_i \cdot x_{i-1} + x_{i-2}$$

```
#!/usr/bin/perl
#Generate continued fraction approximation from given coefficients
#input: comma separated list of coefficients on standard input

sub log10 {
    log(shift)/log(10);
}

my $r=shift;

my @a=split ',',<STDIN>;
my($p0,$q0,$p1,$q1)=(1,0,$a[0],1);
my $i=1;
my($pn,$qn);
for (1 .. $#a) {
    print "$_ ".($p1/$q1)." $p1/$q1";
    print ' ',log10(abs($r-$p1/$q1)) if defined $r;
    print "\n";
    $pn=$a[$_]*$p1+$p0;
    $qn=$a[$_]*$q1+$q0;
    $p0=$p1;
    $q0=$q1;
    $p1=$pn;
```

[2]Použijeme například 3.14159265358979323846264338327950.

```
    $q1=$qn
}
```

Skript spustíme pro zlatý řez: `$ ./gencf3 1.61803398874989 < fibocoef.txt` Soubor fibocoef.txt obsahuje koeficienty oddělené čárkami (v tomto případě dostatečný počet jedniček).

Prvních 13 řádek výstupu (ve formátu: číslo aproximace, hodnota, zlome přiblížení, $\log_{10}$ z odchylky) vypadá následovně.

```
1 1 1/1 -0.208987640249982
2 2 2/1 -0.417975280499952
3 1.5 3/2 -0.927992916413935
4 1.66666666666667 5/3 -1.31307181571953
5 1.6 8/5 -1.74390820558603
6 1.625 13/8 -2.15701582849151
7 1.61538461538462 21/13 -2.57685683405749
8 1.61904761904762 34/21 -2.99412041673167
9 1.61764705882353 55/34 -3.41236767929751
10 1.61818181818182 89/55 -3.83023909197989
11 1.61797752808989 144/89 -4.24825404943149
12 1.61805555555556 233/144 -4.66621417499722
13 1.61802575107296 377/233 -5.08419524453323
```

Graf znázorňuje vývoj chyby přiblížení (log_{10} z odchylky) pro n-té přiblížení.

2.2 Astronomický rok

Délku roku nejprve převedeme na jedno desetinné číslo. Víme, že den má 24 hodin, hodina 60 minut a minuta 60 vteřin. Dostaneme

$$365 + \frac{5 + \frac{48 + \frac{55}{60}}{60}}{24} \doteq 365.2423.$$

Pomocí následujícího skriptu převedeme číslo na koeficienty pravého řetězového zlomku.

```
#!/usr/bin/perl

$_=shift;

for(;;) {
    print int $_;
    $_-=int $_;
    last if $_<0.0000001;
    $_=1/$_;
```

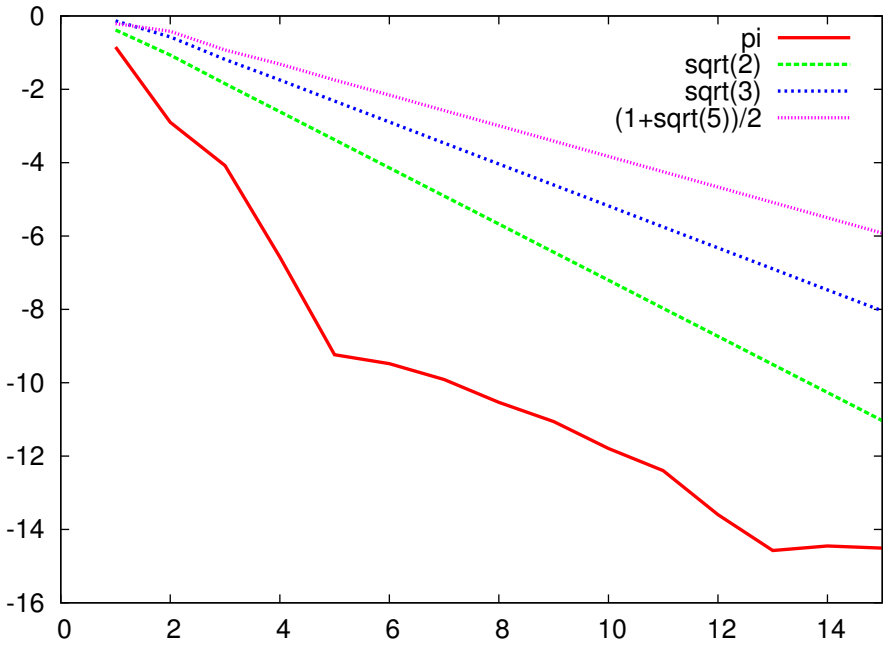
```
    print ',';
}

print "\n";
```

Výsledkem je: 365,4,7,1,6,1,1,19,1.

$$365 + \cfrac{1}{4 + \cfrac{1}{7 + \cfrac{1}{6 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{19 + \cfrac{1}{1}}}}}}}$$

Φ	$[1, \overline{1}]$
$\sqrt{2}$	$[1, \overline{2}]$
$\sqrt{3}$	$[1, \overline{1, 2}]$
π	$[3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, \ldots]$

Obrázek 1: Vývoj chyby nejlepšího přiblížení.



3 Čebyševovy polynomy

Máme $f_{n+2}(x) - 2xf_{n+1}(x) + f_n(x) = 0$ a počáteční podmínky $f_0(x) = T_0(x) = 1$, $f_1(x) = T_1(x) = x$ a $f_0(x) = 0$, $f_1(x) = U_1(x) = 2x$. Diferenční rovnici vyřešíme pomocí z-transformace a pomocí podmínek určíme $T(x)$ a $U(x)$.

$$z^2F(z) - z^2f_0(x) - z\ f_1(x) - 2x(zF(z) - zf_0(x)) + F(z) = 0$$
$$F(z)(1 - 2xz^{-1} + z^{-2}) = (1 - 2xz^{-1})f_0(x) + z^{-1}f_1(x).$$

Odtransformováním a dosazením počátečních podmínek...

$$\begin{aligned}\sum_{n=0}^{\infty} T_n(x)z^{-n} &= \frac{1 - xz^{-1}}{1 - 2xz^{-1} + z^{-2}} = \\ &= \frac{1}{2}\left(\frac{1}{1 - z_1z^{-1}} + \frac{1}{1 - z_2z^{-1}}\right) \\ T_n(x) &= \frac{1}{2}(z_1^n + z_2^n) = \frac{1}{2}((x + \sqrt{x^2 - 1})^n + (x - \sqrt{x^2 - 1})^n)\end{aligned}$$

$$\begin{aligned}\sum_{n=0}^{\infty} U_n(x)z^{-n} &= \frac{1}{1 - 2xz^{-1} + z^{-2}} = \\ &= \frac{z_1}{z_1 - z_2} \cdot \frac{1}{1 - z_1z^{-1}} - \frac{z_2}{z_1 - z_2} \cdot \frac{1}{1 - z_2z^{-1}} \\ U_n(x) &= \frac{z_1^{n+1} - z_2^{n+1}}{z_1 - z_2} = \\ &= \frac{(x + \sqrt{x^2 - 1})^n - (x - \sqrt{x^2 - 1})^{n+1}}{\sqrt{x^2 - 1}}\end{aligned}$$

3.1 Interpolace Čebyševovým polynomem

Pomocí implementace algoritmu převodu v jazyce Python aproximujeme funkci e^{-x^2}. Zjistíme největší chybu přiblížení.

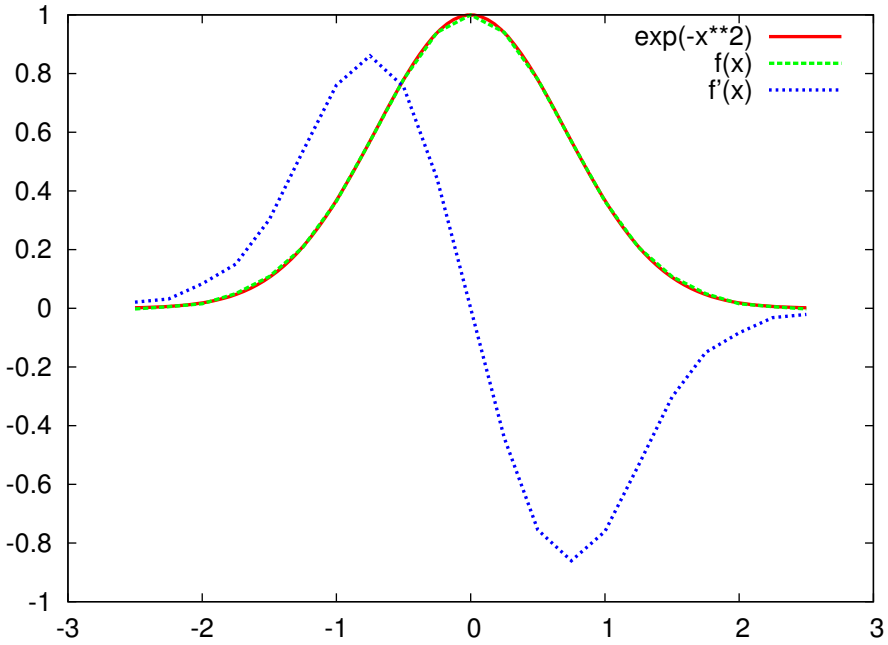
Maximální chyba nastala pro funkční hodnotu $x = 2.5$, absolutní chyba je 0.0041003842775973, relativní 2.124051641863847

```
#!/usr/bin/python

from math import *
import sys
import string
```

```
def chebft(func, a, b, n):
    n = n + 1 # n + 1 koeficientu
    f = range(n) # inicializace poli
    c = range(n)
    bma = 0.5 * (b - a) # transformace intervalu
    bpa = 0.5 * (b + a) # (a,b) ==> (-1,1)

    # transformace funkce
    for k in range(n):
    y = cos(pi * (k + 0.5) / n)
    f[k] = func(y * bma + bpa)

    # vycisleni koeficientu
    fac = 2.0 / n

    for j in range(n):
    sum = 0.0

    for k in range(n):
    sum = sum + f[k] * cos(pi * j * (k + 0.5) / n)
    c[j] = fac * sum

    return c

def chebev(a, b, c, m, x):
    if ((x - a) * (x - b) > 0.0):
print "x nelezi v <a,b>"
return "null"

    if m > len(c) - 1:
    print "m prilis velke"
    return "null"

    y  = (2.0 * x - a - b) / (b - a) # zmena intervalu na (-1,1)

    t0 = 1
    t1 = y
    f = 0.5 * c[0] + c[1] * t1
    k = 1

    while k < m:
    k = k + 1
    t2 = 2. * y * t1 - t0
    f  = f + c[k] * t2
    t0 = t1
    t1 = t2
```

```
    return f

def showfun(func, a, b, c, n, h):
    x = a
    maxerr = 0.
    while x <= b:
    fax    = chebev(a, b, c, n, x)
    fx     = func(x)
    err    = abs(fax - fx)

            if fx:
    relerr = abs(fax / fx - 1)

    print "%f %f" % (x,fax)

    if err > maxerr:
    maxerr    = err
    maxrelerr = relerr
    xmax      = x

    x += h

    if maxerr != 0:
    print "(n=%d) MAX_E x=%.4f AE=%.4f RE=%.24f" % (n, xmax, maxerr, maxrelerr)

    return

def myfunc(x):
    return exp(-pow(x, 2))

if(len(sys.argv)<2):
n=10
else:
n=string.atoi(sys.argv[1])

print "CHEBFT n = " + repr(n)
c = chebft(myfunc, -2.5, 2.5, n)
showfun(myfunc, -2.5, 2.5, c, len(c) - 1, 0.25)
```

Obrázek 2: Aproximace funkce e^{-x^2}.



4 Autogenerátor

Následující program vygeneruje a vypíše do souboru STDOUT svůj obsah. Tento program využívá zajímavé vlastnosti některých programovacích jazyků a sice schopnosti programu obsahovat sebe sama. S podobným mechanismem se setkáme u zlatého řetězce.

```
char *a="char *%c";char *b="main(){printf(a,'a');putchar('=');putchar(34);\
printf(a,'%');putchar('c');putchar(34);putchar(';'); printf(a,'b');\
putchar('=');putchar(34);printf(b); putchar(34);putchar(';');printf(b);}";
main(){printf(a,'a');putchar('=');putchar(34);
printf(a,'%');putchar('c');putchar(34);putchar(';');
printf(a,'b');putchar('=');putchar(34);printf(b);
putchar(34);putchar(';');
printf(b);}
```

5 Hanoiské věže

Algoritmus byl implementován v jazyce Perl. Celý problém řeší rekurzivně volaná procedura *hanoi*. Ta přemístí $n-1$ disků ze zdrojového místa na dočasné místo s využitím cílového místa. Poté přemístí n-tý disk ze zdrojového na cílové místo. K dokončení zbývá přemístit $n-1$ disků z dočasného místa na cílové.

5.1 Implementace algoritmu

Program představuje implementaci pro n disků. Libovolný počet disků lze zadat pomocí parametru. Výchozí hodnota pro počet disků je 3.

```
#!/usr/bin/perl

sub hanoi;

sub hanoi {
    my ($n,$from,$via,$to)=@_;

    $n or return;

    hanoi $n-1,$from,$to,$via;
    print "$from ==> $to\n";
    hanoi $n-1,$via,$from,$to;
}

$_=shift;
/^\d+$/ or $_=3;
hanoi $_,1,2,3;
```

5.2 Počet kroků

Nechť a_n značí minimální počet tahů pro přemístění n disků z jednoho místa na druhé. Ze zdrojového kódu programu je zřejmé, že $a_n = 2a_{n-1} + 1$.

Rekurentní zadání posloupnosti převedeme do explicitního tvaru.

Uvažujeme dvě stejné posloupnosti zadané odlišně.

$$a_n - 2a_{n-1} = 1$$
$$a_{n+1} - 2a_n = 1$$

Rovnice odečteme.

$$a_{n+1} - 3a_n + 2a_{n-1} = 0$$

Vyřešíme charakteristickou rovnici.

$$\lambda^2 - 3\lambda + 2 = 0$$
$$(\lambda - 2)(\lambda - 1) = 0$$

Dostáváme, že $a_n = \alpha 2^n + \beta$. Zřejmě $a_0 = 0$ a $a_1 = 1$. Explicitní vyjádření tedy je

$$a_n = 2^n - 1$$

6 Dělitelnost číslem 7

Všimneme si, že mechanismus postupného násobení a sčítání cifer připomíná zápis čísla ve trojkové soustavě. Algoritmus popíše tvrzení: Číslo z je dělitelné sedmi právě tehdy když

$$\sum_{n=1}^{k} c_n 3^n \ \% \ 7 = 0$$

kde k je počet cifer čísla, c_n (dekadická) cifra čísla a c_1 nejméně významná cifra. (% je odlišně zapsaná operace mod.)

Důkaz:

Všimneme si, že posloupnosti $a_n = 10^n$ a $b_n = 3^n$ mají pro dělitel 7 stejné charakteristické posloupnosti a tedy i stejné zbytky po dělení číslem 7.

$$\begin{aligned}
10^0 \ \% \ 7 &= 3^0 \ \% \ 7 = 1 \\
10^1 \ \% \ 7 &= 3^1 \ \% \ 7 = 3 \\
10^2 \ \% \ 7 &= 3^2 \ \% \ 7 = 2 \\
10^3 \ \% \ 7 &= 3^3 \ \% \ 7 = 6 \\
10^4 \ \% \ 7 &= 3^4 \ \% \ 7 = 4 \\
10^5 \ \% \ 7 &= 3^5 \ \% \ 7 = 5
\end{aligned}$$

Dělitelnost čísla k obecně ověříme spočtením sumy $\sum c_n(B^n \ \% \ d)$ (c_n je n-tá cifra, B číselná báze a d dělitel). Je-li výsledek této sumy dělitelný dělitem d beze zbytku, je jím takto dělitelné i číslo k. Tato vlastnost vyplývá z vlastností kongruence pro sčítání a násobení.

6.1 Dělitelnost čísly 7, 11 a 13

Pro případ zjišťování dělitelnosti čísly 7, 11 a 13 můžeme využít skutečnosti, že číslo 1001 je dělitelné těmito beze zbytku. Libovolné celé číslo můžeme zapsat jako $k \cdot 1001 + l$, $k, l \in Z_0$ a omezit se na zjišťování dělitelnosti čísla l.

$$\begin{aligned}
946988875 &= 946988 \cdot 1000 + 946988 - 946988 + 875 = \\
&= 946988 \cdot 1001 - (946988 - 875)
\end{aligned}$$

7 Házení kostek

7.1 Pravděpodobnosti pro 3 a 4 kostky

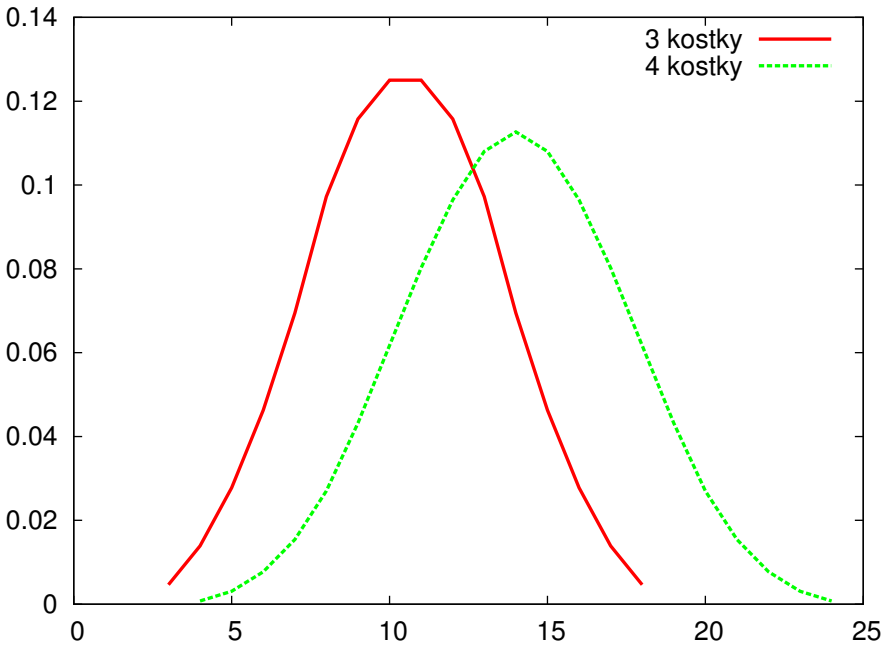
Následujícím skriptem spočteme počet různých kombinací pro jednotlivé součty. Počet kostek je zadán jako vstupní argument. Získané hodnoty násobíme $1/n$, kde n je počet kostek. Ze získaných hodnot nakreslíme graf. Algoritmus prochází všechny možné kombinace, tj. čísla z intervalu $[0; 6^n - 1]$ pro každé spočítá součet cifer čísla reprezentovaného v šestkové soustavě a výsledek pro každý součet uloží do pole `@res`.

```
#!/usr/bin/perl

my $n=shift;
$n=3 if $n!~/^\d+$/;
my $p=shift;

my @res;

sub getsum {
    my($N,$n)=@_;
    my $r=0;
    while($N--) {
        $r+=$n%6;
        $n/=6;
    }
    $r;
}

for my $i (0 .. (6**$n)-1) {
    my $sum=getsum $n,$i;
    $res[$sum]++;
}

for (my $i=0;$i<@res;$i++) {
    print "".($i+$n)." ".(defined $p?$res[$i]/(6**$n):$res[$i])."\n";
}
```

Například pro $n = 3$ dostaneme následující hodnoty.

3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	3	6	10	15	21	25	27	27	25	21	15	10	6	3	1

Obrázek 3: Graf pravděpodobností pro 3 a 4 kostky.



7.2 Hodnota pravděpodobnosti

Hodnota pravděpodobnosti nejvyššího součtu při házení n kostkami nastane pouze v případě, že na všech kostkách padne šestka. Pravděpodobnost šestky na jedné kostce je $\frac{1}{6}$, na n kostkách současně tedy $\left(\frac{1}{6}\right)^n$.

8 Zlatý řetězec

Motivací je implementace algoritmu, který vygeneruje libovolně dlouhý podřetězec z libovolného místa zlatého řetězce. Vyjdeme z jednoduchého rekurzivní funkce pro n-té Fibonacciho číslo.

```
sub genf {
    my $a=shift;
    return 0 if !$a;
    return 1 if $a==1;
    return genf($a-1)+genf($a-2);
}
```

8.1 Implementace algoritmu

Rozhraní představuje funkce getgs, jejíž argumenty udávají pozici (*pos*) a délku (*len*) podřetězce. K dispozici je cache, která obsahuje řetězec odpovídající řetězcové AB reprezentaci nějakého n-tého Fibonacciho čísla. Je-li $pos + len$ menší než délka cachového řetězce, funkce okamžitě vrací výsledek, jinak určí poslední Fibonacciho číslo menší než $pos + len$. Na základě porovnání vrací výsledek pomocí rekurzivního volání. Součet $pos + len$ je v každém vnořeném volání nižší, tím je zaručena konečnost algoritmu.

```
#!/usr/bin/perl

use strict;
use warnings;

sub lrf {
    my($n,$l,$r)=@_;
    my($a,$b)=(1,1);
    while($n>$b) {
        my $c=$b;
        $b+=$a;
        $a=$c;
    }
    return $a;
}

my $cache_str='ab';
my $cache_len=length $cache_str;

sub getgs {
    my($pos,$len)=@_;
    return substr $cache_str,$pos,$len if $pos+$len<=$cache_len;
    my $l=lrf $pos+$len;
```

```
    my($lpart,$rpart);
    if($pos<$l) {
        $lpart=getgs($pos,$l-$pos);
        $rpart=getgs(0,$len-($l-$pos));
    } else {
        $lpart='';
        $rpart=getgs($pos-$l,$len);
    }
    return $lpart.$rpart;
}

for my $i (qw/1 2 3 5 8 13/) {
    print getgs(0,$i)."\n";
}
```

Výstup programu vypadá následovně (podřetězce odpovídající $F_1(=1), \ldots, F_6(=13)$).

```
a
ab
aba
abaab
abaababa
abaababaabaab
```

9 Katalánská čísla

Jednou z interpretací posloupnosti katalánských čísel je počet všech kombinací správného zápisu párů závorek, tzn. součet počtu levých závorek musí být vždy větší nebo roven součtu počtu pravých závorek v libovolném podřetězci začínajícím na počátku závorkového řetězce. Pro počet takových kombinací lze na základě znalosti správného zápisu závorek odvodit rekurentní formuli.

- $n = 0, 1$

  Pro $n = 0, 1$ je zřejmě $A_0 = A_1 = 1$.

- $n = n + 1$

  Vždy musíme začít levou závorkou (resp. skončit pravou), příslušnou pravou závorku můžeme umístit na libovolné neobsazené místo. Mezi první dvojici závorek můžeme umístit buď žádný nebo více párů závorek, maximálně n. Umístíme-li tedy mezi závorky i párů, potom za tyto závorky musíme umístit $n - i$ párů. Pro každou volbu i existuje právě A_i způsobů vyplnění vnitřku a A_{n-i} pro vyplnění zbytku. Odtud pro $n + 1$ závorek dostáváme rekurentní vzorec

$$A_{n+1} = \sum_{i=0}^{n} A_i \cdot A_{n-i}$$

1	2	3	4	5	6	7	8	9	10
1	2	5	14	42	132	429	1430	4862	16796

9.1 Interpretace Katalánských čísel

- explicitně

$$C_n = \frac{\binom{2n}{n}}{n+1}$$

- rekurentně

$$C_0 = 1, \quad C_{n+1} = \sum_{i=0}^{n} C_i \cdot C_{n-i}$$

- počet všech možných binárních stromů s n uzly
- počet všech možných triangulací konvexního mnohoúhelníku s n+2 hranami
- počet všech možných lomených čar obtahujících hrany na rohu čtverečkovaného papíru, kde jednotlivé úseky čáry vedou vždy na pravý horní nebo pravý dolní sousední průsečík přičemž není překročen dolní horizont, který je určen počátečním průsečíkem (roh papíru je orientován směrem vzhůru)