Oblikovanje i analiza algoritama
9. predavanje

Saša Singer

singer@math.hr

web.math.pmf.unizg.hr/~singer

PMF – Matematički odsjek, Zagreb

Sadržaj predavanja

- Fibonaccijevi brojevi:
  - Uvod, problem FIB, DeMoivreova formula.
  - FIB — rekurzivni algoritam.
  - FIB — aditivni algoritam.
  - Brzo potenciranje i brzo računanje n-tog člana rekurzije.
  - FIB — brzi algoritam.
  - FIB — sve znamenke i stvarna složenost.

Informacije — web stranica

Moja web stranica za Oblikovanje i analizu algoritama je

https://web.math.pmf.unizg.hr/~singer/oaa/

ili, skraćeno

https://web.math.hr/~singer/oaa/

Kopija je na adresi

http://degiorgi.math.hr/~singer/oaa/

Službena web stranica za Oblikovanje i analizu algoritama je

https://web.math.pmf.unizg.hr/nastava/oaa/

Fibonaccijevi brojevi — Uvod, DeMoivreova formula

Pojednostavljenje DeMoivreove formule

Egzaktna DeMoivreova formula je

$$F_n = \frac{1}{\sqrt{5}} \left(\Phi^n - (-\Phi)^{-n} \right), \quad n \in \mathbb{N}_0.$$

Kako je $\Phi^{-1} < 1$, onda je $\Phi^{-n}/\sqrt{5} < 1/2$, za svaki $n \in \mathbb{N}_0$, pa drugi član možemo zanemariti, tj. vrijedi

$$F_n = \left[\frac{1}{\sqrt{5}} \Phi^n \right], \quad n \in \mathbb{N}_0,$$

gdje $[x]$ označava "najbliže cijelo" od x.

To odgovara korektnom zaokruživanju realnog broja $\Phi^n/\sqrt{5}$ na najbliži cijeli broj!

Implementacija svih algoritama za F_n

Kod implementacije i usporedbe različitih algoritama za računanje F_n, za prikaz Fibonaccijevih brojeva koristimo

- "najveći" realni tip, koji je korektno podržan arhitekturom i programskom bibliotekom u danom jeziku.

U nastavku, standardno koristim tip `double` — za Intel C.

Mogu još koristiti `extended` — za GNU C, ili Pascal/Delphi (ali samo na ia32).

Razlozi za izbjegavanje cjelobrojnih tipova = F_n brzo rastu:

- mali raspon prikazivih cijelih brojeva (do F_{93}, sa 64 bita),
- modularna aritmetika cijelih brojeva — dobivamo samo donje bitove pravog rezultata.

Implementacija DeMoivreove formule

U implementaciji DeMoivreove formule u realnoj artimetici,

- izbacujemo i "najbliže cijelo", tj. vraćamo realni broj.

Kod ispisa, rezultat se ionako "zaokružuje" na zadani broj decimala, tj. ispis s 0 decimala (`%.0f`) radi to što treba!

Funkcija za računanje F_n po DeMoivreovoj formuli:

```
double Fib(int n)
{
    double Sqrt5 = sqrt(5.0), Fi = (1 + Sqrt5) / 2;

    return pow(Fi, n) / Sqrt5;
}
```

Rezultati za $n = 78$ (Intel C 18.0.1, ia32)

Najveći egzaktno prikazivi Fibonaccijev broj u tipu `double` je

$$F_{78} = 8\ 944\ 394\ 323\ 791\ 464 \approx 8.9 \cdot 10^{15}.$$

Međutim, zbog grešaka zaokruživanja u realnoj aritmetici i u funkcijama `sqrt`, `pow` iz `<math.h>`,

- izračunati rezultat je za 24 prevelik (gubitak točnosti od 4–5 bitova, od ukupno 53 bita).

Izračunati rezultat i pripadno vrijeme je:

```
     F(n) = F_n                          T(n) [s]
=============================================================
 F(   78) =   8944394323791488     2.260000e-08
=============================================================
```

Rezultati za $n = 2^k$ (Intel C 18.0.1, ia32)

```
Repeat count:  Rpt = 10000000 = 10^7

                                          Rpt * T(n)     c_n =
     F(n) = F_n                              [s]         T(n)
=================================================================
 F(    2) =   1.1708203932e+00              0.226      2.260000e-08
 F(    4) =   3.0652475842e+00              0.228      2.280000e-08
 F(    8) =   2.1009519494e+01              0.226      2.260000e-08
 F(   16) =   9.8700020263e+02              0.228      2.280000e-08
 F(   32) =   2.1783090000e+06              0.227      2.270000e-08
 F(   64) =   1.0610209858e+13              0.226      2.260000e-08
 F(  128) =   2.5172882568e+26              0.226      2.260000e-08
 F(  256) =   1.4169381771e+53              0.294      2.940000e-08
 F(  512) =  4.4893845313e+106              0.293      2.930000e-08
 F( 1024) =  4.5066996337e+213              0.293      2.930000e-08
=================================================================
```

FIB — rekurzivni algoritam

Implementacija rekurzivnog algoritma za F_n

Rekurzivna funkcija za računanje F_n:

```
double Fib(int n)
{
    if (n > 1)
            /* Rekurzivni pozivi. */
        return Fib(n - 1) + Fib(n - 2);

    else    /* n <= 1. */
            /* Pretpostavljamo da je n = 0, 1,
               tj., n >= 0. */
        return n;
}
```

Rezultati za $n = 34, \ldots, 45$ (Intel C 18.0.1, ia32)

F(n) = F_n		T(n) [s]	c_n = T(n) / F(n)
F(34) =	5702887	0.045	7.890740e-09
F(35) =	9227465	0.074	8.019537e-09
F(36) =	14930352	0.120	8.037319e-09
F(37) =	24157817	0.193	7.989132e-09
F(38) =	39088169	0.312	7.981955e-09
F(39) =	63245986	0.505	7.984696e-09
F(40) =	102334155	0.816	7.973877e-09
F(41) =	165580141	1.320	7.971971e-09
F(42) =	267914296	2.134	7.965234e-09
F(43) =	433494437	3.452	7.963193e-09
F(44) =	701408733	5.581	7.956844e-09
F(45) =	1134903170	9.029	7.955745e-09

Predviđanje za $n = 78$ (Intel C 18.0.1, ia32)

Uzmimo da je u modelu složenosti $c = 7.9 \cdot 10^{-9}$ (sekundi).

Iz prethodne tablice, to je prilično točna aproksimacija (na 2 dekadske znamenke, za još veće n).

Koristeći tu vrijednost, izračunajmo (približno) potrebno vrijeme rekurzivnog algoritma za računanje

$$F_{78} = 8\ 944\ 394\ 323\ 791\ 464 \approx 8.9 \cdot 10^{15}.$$

Dobivamo

$$T(78) = c \cdot F_{78} \approx (7.9 \cdot 10^{-9}) \cdot (8.9 \cdot 10^{15}) \approx 7.03 \cdot 10^{7}.$$

U prijevodu, algoritam treba oko 70 milijuna sekundi.

Kad uzmemo da je broj sekundi u jednoj godini oko $3.05 \cdot 10^{7}$, slijedi da rekurzivni algoritam za F_{78} traje oko 2.3 godine!

FIB — aditivni algoritam

Implementacija aditivnog algoritma za F_n

Aditivna funkcija za računanje F_n:

```
double Fib(int n)
{
    double F, F1, F2;
    int i;

    if (n <= 1)     /* Pretp. da je n = 0, 1. */
        return n;

    else {  /* n > 1. Inicijaliziraj ‘‘prozor’’
               (prva dva clana). */
        F1 = 0;     // Fib(0) = 0.
        F  = 1;     // Fib(1) = 1.
```

Aditivna funkcija za F_n (nastavak)

```
            /* Pomak (klizanje) ‘‘prozora’’
               od 3 susjedna clana. */

        for (i = 2; i <= n; ++i) {
            F2 = F1;          // Pomak F1 u F2.
            F1 = F;           // Pomak F u F1.
            F = F1 + F2;      // F = Fib(i) = F1 + F2.
        }

        return F;
    }
}
```

Rezultati za $n = 78$, 2^k (Intel C 18.0.1, ia32)

Repeat count: Rpt = 10000000 = 10^7

F(n) = F_n			Rpt * T(n) [s]	c_n = T(n) / n
F(78)	=	8944394323791464	0.505	6.474359e-10
F(4)	=	3.0000000000e+00	0.025	6.250000e-10
F(8)	=	2.1000000000e+01	0.042	5.250000e-10
F(16)	=	9.8700000000e+02	0.069	4.312500e-10
F(32)	=	2.1783090000e+06	0.164	5.125000e-10
F(64)	=	1.0610209858e+13	0.371	5.796875e-10
F(128)	=	2.5172882568e+26	0.924	7.218750e-10
F(256)	=	1.4169381771e+53	2.024	7.906250e-10
F(512)	=	4.4893845313e+106	4.219	8.240234e-10
F(1024)	=	4.5066996337e+213	8.614	8.412109e-10

Brzo potenciranje i brzo računanje n-tog člana rekurzije

Implementacije bez množenja s 1

Umjesto inicijalizacije `pot = 1`,

- koja služi tome da algoritam radi i za $n = 0$ (kraći kod),

inicijalizaciju `pot` treba napraviti "na pravom mjestu".

Za potenciranje množenjem, inicijalizacija je `pot = x`, čim znamo da je $n \geq 1$.

Za binarno potenciranje, inicijalizacija je `pot = b`,

- u trenutku kad naiđemo na prvu binarnu znamenku jednaku 1 u (trenutnom) n.

Potenciranje množenjem — bez množenja s 1

```
double pow_mul(double x, int n)
{
    double pot;
    int i;

    if (n > 0) {
        pot = x;
        for (i = 2; i <= n; ++i)
            pot *= x;
        return pot;
    } else
        return 1.0;
}
```

Binarno potenciranje — bez množenja s 1

```
double pow_bin(double x, int n)
{
    double pot, b = x;

        /* Prvo rijesimo specijalni slucaj n = 0. */
    if (n == 0) return 1.0;

        /* Sve dok je (trenutni) n paran (zadnja
           binarna znamenka je 0), kvadriraj b i
           obrisi zadnju znamenku u n. */
    while ((n & 1) == 0) {    // (n % 2 == 0)
        b *= b;      // Kvadriraj b.
        n >>= 1;     // n /= 2;
    }
```

Binarno potenciranje — bez množ. s 1 (nast.)

```
        /* Zadnja binarna znamenka (trenutnog) n je
           jednaka 1, tj. n je neparan.
           Inicijaliziraj pot na b, obrisi zadnju
           znamenku (jedinicu). */

pot = b;
n >>= 1;     // n /= 2;
```

Binarno potenciranje — bez množ. s 1 (nast.)

```
        /* Petlja za preostale binarne znamenke
           od n. */
    while (n > 0) {
        b *= b;      // Kvadriraj b.

            /* Ako je n neparan, onda akumuliraj
               produkt u pot (pomnozi pot s b). */
        if ((n & 1) != 0) pot *= b;

        n >>= 1;     // n /= 2;
    }

    return pot;
}
```

FIB — brzi algoritam

Implementacija brzog algoritma za F_n

Koristimo binarno potenciranje, bez dodatnog množenja s 1, i

- za parne n — eksplicitno postavljamo prvu matricu na F^2, a ne na F.

Rezultati za $n = 78$, 2^k (Intel C 18.0.1, ia32)

Repeat count: Rpt = 100000000 = 10^8

F(n) = F_n			Rpt * T(n) [s]	c_n = T(n) / lg(n)
F(78)	=	8944394323791464	1.144	3.788280e-09
F(4)	=	3.0000000000e+00	0.286	2.976358e-09
F(8)	=	2.1000000000e+01	0.372	2.580898e-09
F(16)	=	9.8700000000e+02	0.472	2.456015e-09
F(32)	=	2.1783090000e+06	0.598	2.489317e-09
F(64)	=	1.0610209858e+13	0.725	2.514988e-09
F(128)	=	2.5172882568e+26	0.891	2.649285e-09
F(256)	=	1.4169381771e+53	1.050	2.731797e-09
F(512)	=	4.4893845313e+106	1.254	2.900041e-09
F(1024)	=	4.5066996337e+213	1.455	3.028392e-09

Rezultati za $n = 2^k - 1$ (Intel C 18.0.1, ia32)

Repeat count: Rpt = 100000000 = 10^8

F(n) = F_n			Rpt * T(n) [s]	c_n = T(n) / lg(n)
F(3)	=	2.0000000000e+00	0.373	4.898227e-09
F(7)	=	1.3000000000e+01	0.567	4.203730e-09
F(15)	=	6.1000000000e+02	0.843	4.491024e-09
F(31)	=	1.3462690000e+06	1.166	4.898628e-09
F(63)	=	6.5574703198e+12	1.461	5.087398e-09
F(127)	=	1.5557697022e+26	1.813	5.399474e-09
F(255)	=	8.7571595343e+52	2.149	5.595027e-09
F(511)	=	2.7745922289e+106	2.521	5.831973e-09
F(1023)	=	2.7852935507e+213	2.866	5.966044e-09

Usporedba algoritama (Intel C 18.0.1, ia32)

Pregled potrebnih vremena za računanje F_{1023} i F_{1024}, za razne algoritme:

algoritam	$T(1024)$	$T(1023)$
`f_moivre`	$2.930 \cdot 10^{-8}$	$2.920 \cdot 10^{-8}$
`f_summa`	$8.614 \cdot 10^{-7}$	$8.607 \cdot 10^{-7}$
`f_binpow`	$1.878 \cdot 10^{-8}$	$3.066 \cdot 10^{-8}$
`f_binpow_w`	$1.825 \cdot 10^{-8}$	$3.024 \cdot 10^{-8}$
`f_binpow_1`	$1.577 \cdot 10^{-8}$	$2.870 \cdot 10^{-8}$
`f_binpow_2`	$1.455 \cdot 10^{-8}$	$2.866 \cdot 10^{-8}$

Uočite da je dobro implementirano binarno potenciranje brže od DeMoivreove formule!

FIB — sve znamenke i stvarna složenost