

TDK-dolgozat

Varga András

REKURZÍV SZÁMSOROZATOK

Név: Varga András

Évfolyam: 1. évfolyam, mester képzés

Felsőoktatási intézmény: Univerzita Komenského v Bratislave

Kar: Fakulta matematiky, fyziky a informatiky

Konzulensek: RNDr. Martin Sleziak (Egyetemi tanár)

Konzulensek: PhD. és RNDr. Bukor József, PhD (Egyetemi tanár, dékán)

Ezennel kijelentem, hogy eme pályamunkát egyedül
készítettem a felsorolt irodalomra támaszkodva,
vezetőim felügyelete alatt.

.....

Szeretném megköszönni vezetőimnek segítségüket, melyet a munka elkészítése során nyújtottak.

Absztrakt

Ebben a munkában egy olyan algoritmus leírása található, amely egy adott másodrendű rekurzív számsorozat n -edik elemének kiszámításához $O(\log n)$ számtani műveletet használ fel. Ezen felül tartalmazza az adott algoritmus helyességének bizonyítását, valamint az algoritmus implementálása is a munka részét képezi. A téma megválasztása azért is érdekes, mivel a rekurzív számsorozat definíciója önmaga is egyfajta algoritmus a sorozat n -edik elemének kiszámítására, mely elvégzéséhez azonban $O(n)$ számtani műveletet kell elvégeznünk. Így tehát a munka részét képezi azon matematikai összefüggések levezetése is, melyek az itt bemutatott algoritmus alapját képezik. Az algoritmus implementálása bemutatja annak működőképességét, és lehetőséget nyújt arra, hogy az itt leírt algoritmust összehasonlítsuk más, ugyanezt a feladatot szolgáló algoritmusokkal is.

KULCSSZAVAK: Fibonacci-sorozat, Másodrendű rekurzív számsorozatok, Számsorozat n -edik elemének kiszámítása, Effektív algoritmusok, Dinamikus programozás

Abstract

The main goal of this article is derivation, proof and implementation of some fast algorithms for linear recurrent sequences. The basic problem is the evaluation of n -th member of given sequence. This problem is solvable in time $O(n)$. In this work, we are trying to create algorithms using $O(\log n)$ arithmetic operations. The work also includes proofs of correctness of algorithms and of all necessary mathematical statements. Implementation of these algorithms also serves as a test of their correctness. Implementation of algorithms also gives the possibility to experimentally compare the effectiveness of individual algorithms compared to the ones that already exist.

KEYWORDS: Fibonacci sequence, Linear recurrent sequences, Evaluation n -th number of sequence, Effective algorithms, Dynamic programming

Tartalomjegyzék

Bevezetés	1
Célok és a motiváció	1
Az algoritmus létrehozásának lépései	1
1. A másodrendű rekurzív számsorozatok	3
1.1. A rekurzív számsorozatokról röviden	3
Az m -edrendű rekurzív számsorozatok definíciója	3
1.2. Az algoritmus matematikai alapjai	4
A munkában használt rekurzív számsorozatok definíciói	4
Tétel a Fibonacci számsorozatról	5
Tétel az R_n sorozatról	6
Tétel a P_n sorozatról	7
Tétel a Q_n sorozatról	8
1.3. A Fibonacci számsorozat n -edik elemének kiszámítása	9
1.3.1. Iteráló algoritmus a Fibonacci számsorozat n -edik elemének kiszámítására	11
1.4. A többi számtani sorozat n -edik elemének kiszámítása	13
1.5. Az abszolút taggal rendelkező másodrendű rekurzív számsorozatok n -edik elemének kiszámítása	14
Tétel az S_n sorozatról	16
2. Analízis és összehasonlítás	18
2.1. A felhasznált algoritmusok	18
2.2. Az algoritmusok összehasonlítása a felhasznált aritmetikai operációk, és azok száma alapján	20
2.3. Az algoritmusok összehasonlítása a mért futási idők alapján	21
Befejezés	22

Bevezetés

Célok és a motiváció

Ez a munka a rekurzív számsorozatok n -edik elemének kiszámításával foglalkozik. Célja, hogy leírjon egy algoritmust a probléma megoldására, bizonyítsa annak helyességét s levezesse azon matematikai összefüggéseket, melyek az algoritmus alapjául szolgálnak. Célunk továbbá, hogy ez az algoritmus a rekurzív számsorozatok lehető legszélesebb csoportjára alkalmazható legyen, legrosszabb esetben $O(\log_2 n)$ számtani műveletet végezzen, s ezen feltételek mellett az algoritmus memóriaigénye is optimális maradjon.

Az algoritmus alapötletét Daisuke Takahashi adta, aki [16] cikkében leír egy speciális algoritmust a Fibonacci számsorozat n -edik elemének kiszámítására. Ez az algoritmus szintén $O(\log_2 n)$ számtani művelet hajt végre, de csak a Fibonacci számsorozatra alkalmazható.

Ezzel szemben az itt leírt algoritmus a rekurzív számsorozatok nagyobb halmazára alkalmazható, mégpedig azon másodrendű rekurzív számsorozatokra, melyek együtthatói valós számok, mindemellett továbbra is elégséges $O(\log_2 n)$ számtani művelet végrehajtása, s nem nőtt meg az algoritmus memóriaigénye sem. Viszont a túlzott általánosítás miatt nem sikerült az algoritmus további optimalizációja, s úgy gondoljuk, hogy a konkrét számsorozatok n -edik elemének kiszámítására létezhet gyorsabb, speciálisan az adott számsorozat jellegzetességein alapuló algoritmus, mely azonban aszimptotikusan azonos mennyiségű számtani műveletet végez el.

Az algoritmus létrehozásának lépései

Elsősorban a Fibonacci számsorozat ismert összefüggéseit kellett általánosítani, bebizonyítani helyességüket, levezetni azokat, hogy a rekurzív számsorozatok minél nagyobb csoportjára érvényesek legyenek. Ez a feladat a már említett valós együtthatókkal rendelkező másodrendű rekurzív számsorozatokra járt sikerrel. A levezetett összefüggések valójában egy Oszd meg és uralkodj al-

goritmust írnak le, mely az adott problémát több hasonló problémára osztja kisebb bemenettel. Mivel azonban a dinamikus programozás időigénye sok esetben (ebben a konkrét esetben is) jobb, mint a klasszikus Oszd meg és uralkodj elvű algoritmus, ezért a végső algoritmust ez alapján a programozói elv alapján készítettük el. Ezzel a módszerrel sikerült elérni a kívánt $O(\log n)$ -es időigényt, azonban általánosságban növekszik az algoritmus memóriaigénye. Viszont a megfelelő rekurzív összefüggés kiválasztásával, valamint a rekurzív hívások elemzésével sikerült optimalizálni a memóriaigényt. Az itt bemutatott algoritmus ugyanis 6 változót használ a szükséges segédváltozókon (különböző indexek és ciklusváltozók) kívül.

1. fejezet

A másodrendű rekurzív számsorozatok

1.1. A rekurzív számsorozatokról röviden

Mivel ebben a munkában olyan számsorozatokkal foglalkozunk, melynek tagjai valós számok, ezért az alábbi módon definiálhatjuk a rekurzív számsorozatokot:

1.1.1. Definíció. *[] Az m -edrendű rekurzív számsorozat olyan sorozat $\{r_i\}_0^\infty$, melyre $\forall i \in \mathbb{N} : r_i \in \mathbb{R}; r_0, r_1, \dots, r_{m-1}$ valós számok,*

$$\forall n \in \mathbb{N}, n \geq m : r_n = F(r_{n-1}, r_{n-2}, \dots, r_{n-m}, n),$$

ahol $F : \mathbb{R}^{m+1} \rightarrow \mathbb{R}$ egy függvény.

Szabadon tehát az m -edrendű rekurzív számsorozat olyan sorozat, melynek minden tagját az előző m tag alapján lehet kiszámítani. Természetesen a sorozat első m tagja adott, és meghatározza az egész sorozatot. Tehát egy rekurzív számsorozat pontos leírásához szükségünk van a kezdeti feltételekre (első m tag) és a rekurzív összefüggésre, mely alapján a sorozat többi tagját kiszámítjuk. A következőkben olyan sorozatokkal fogunk foglalkozni, melyek együtthatói valós számok, vagyis az F függvény a sorozat előző m tagját úgy adja össze, hogy előbb mindegyiket megszorozza a helyüknek megfelelő együtthatóval. Több ismert probléma is visszavezethető a rekurzív számsorozatokra, ilyenek például Hanoi tornyainak problémája, a sík maximális felosztása n egyenes segítségével. És természetesen ide tartozik a Fibonacci számsorozat is, mely a világ leghíresebb számtani sorozata.

1.2. Az algoritmus matematikai alapjai

Mivel az algoritmus csak a másodrendű rekurzív számsorozatokra alkalmazható, szükséges, hogy bevezessünk néhány ezzel összefüggő matematikai fogalmat s levezessük a felhasználandó összefüggéseket. Ezzel foglalkozik az alábbi fejezet.

Az eddigiekből látható, hogy a másodrendű rekurzív számsorozat olyan számsorozatot takar, melyben minden tagot az őt megelőző két tagból számítjuk ki a rekurzív összefüggés alapján.

1.2.1. Definíció. *[/ Értelmezzük az alábbi rekurzív számsorozatokat:*

$$\begin{aligned} F_0 &= 0, F_1 = 1 \text{ a } F_n = F_{n-1} + F_{n-2} \\ L_0 &= 2, L_1 = 1 \text{ a } L_n = L_{n-1} + L_{n-2} \\ R_0 &= a, R_1 = b \text{ a } R_n = R_{n-1} + R_{n-2} \\ P_0 &= 0, P_1 = 1 \text{ a } P_n = cP_{n-1} + dP_{n-2} \\ Q_0 &= a, Q_1 = b \text{ a } Q_n = cQ_{n-1} + dQ_{n-2} \\ \text{pre } \forall n \in \mathbb{N}: n &\geq 2 \text{ a, } b, c, d \in \mathbb{R} \end{aligned}$$

Az ilyen módon definiált F_n -t Fibonacci számsorozatnak nevezik, vagy csak egyszerűbben Fibonacci sorozatnak, az L_n sorozatot pedig Lucas sorozatnak.

Az $L_n = F_{n-1} + F_{n+1}$ összefüggés matematikai indukcióval bizonyítható. A továbbiakban átvesszük Beáta Štupáková munkájában [15] használt mátrixot, mely a Fibonacci számsorozatot írja le. Ezen mátrixot általánosítjuk a többi rekurzív számsorozatra is. A [10] készítői szintén általánosítják a Fibonacci számsorozatra érvényes fontosabb összefüggéseket. Ugyanebben a műben megjelenik a P_n sorozatot leíró mátrix is.

1.2.2. Definíció. *[/*

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

1.2.3. Lemma. *[/*

$$\mathbf{A}^n = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n = \begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix}$$

Bizonyítás. Matematikai indukcióval az n alapján:
 $n = 1$ esetben érvényes:

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} F_0 & F_1 \\ F_1 & F_2 \end{pmatrix}$$

Most tételezzük fel, hogy az állításunk igaz n -re, bizonyítsuk be $n + 1$ -re:

$$\mathbf{A}^{n+1} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n+1} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} =$$

az indukciós feltevés alapján

$$= \begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} F_n & F_{n+1} \\ F_{n-1} + F_n & F_n + F_{n+1} \end{pmatrix} = \begin{pmatrix} F_n & F_{n+1} \\ F_{n+1} & F_{n+2} \end{pmatrix}$$

□

1.2.4. Tétel. []

$$F_{m+n} = F_m F_{n+1} + F_{m-1} F_n$$

Bizonyítás.

$$\mathbf{A}^{m+n} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^m \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n =$$

Az előző lemma 1.2.3 alapján:

$$= \begin{pmatrix} F_{m-1} & F_m \\ F_m & F_{m+1} \end{pmatrix} \begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix} =$$

$$= \begin{pmatrix} F_{m-1}F_{n-1} + F_m F_n & F_{m-1}F_n + F_m F_{n+1} \\ F_m F_{n-1} + F_{m+1} F_n & F_m F_n + F_{m+1} F_{n+1} \end{pmatrix}$$

S mindemelett érvényes

$$\mathbf{A}^{m+n} = \begin{pmatrix} F_{m+n-1} & F_{m+n} \\ F_{m+n} & F_{m+n+1} \end{pmatrix}$$

Tehát a mátrixok egyenlőségéből adódik: $F_{m+n} = F_{m-1}F_n + F_m F_{n+1}$. □

Az előző tételnek van egy fontos következménye: Táto veta má jeden dôležitý dôsledok:

1.2.5. Következmény. []

$$F_{2n} = F_n(F_{n+1} + F_{n-1})$$

$$F_{2n+1} = F_{n+1}^2 + F_n^2$$

Bizonyítás. Az első esetben n -t helyettesítünk az m helyére, a második egyenlőség esetén pedig $n + 1$ -et. □

1.2.6. Lemma. []

$$\mathbf{A}^n \begin{pmatrix} a & b \\ b & a+b \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} a & b \\ b & a+b \end{pmatrix} = \begin{pmatrix} R_n & R_{n+1} \\ R_{n+1} & R_{n+2} \end{pmatrix}$$

Bizonyítás. Matematikai indukcióval az n alapján:

$n = 0$ esetében érvényes az egyenlőség. $n = 1$ esetében:

$$\mathbf{A} \begin{pmatrix} a & b \\ b & a+b \end{pmatrix} = \begin{pmatrix} b & a+b \\ a+b & a+2b \end{pmatrix} = \begin{pmatrix} R_1 & R_2 \\ R_2 & R_3 \end{pmatrix}$$

Most tételezzük fel, hogy az állításunk igaz n -re, bizonyítsuk be $n+1$ -re:

$$\mathbf{A}^{n+1} \begin{pmatrix} a & b \\ b & a+b \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} a & b \\ b & a+b \end{pmatrix} =$$

az indukciós feltevés alapján

$$= \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} R_n & R_{n+1} \\ R_{n+1} & R_{n+2} \end{pmatrix} = \begin{pmatrix} R_{n+1} & R_{n+2} \\ R_n + R_{n+1} & R_{n+1} + R_{n+2} \end{pmatrix} = \\ = \begin{pmatrix} R_{n+1} & R_{n+2} \\ R_{n+2} & R_{n+3} \end{pmatrix}$$

A lemma helyességéről úgy is meggyőződhetünk, hogy $a = 0$ és $b = 1$ behelyettesítése esetén a 1.2.3 lemmát kapjuk. $\square$

1.2.7. Tétel. []

$$R_n = aF_{n-1} + bF_n$$

Bizonyítás.

$$\begin{pmatrix} R_n & R_{n+1} \\ R_{n+1} & R_{n+2} \end{pmatrix} = \mathbf{A}^n \begin{pmatrix} a & b \\ b & a+b \end{pmatrix} = \begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix} \begin{pmatrix} a & b \\ b & a+b \end{pmatrix} = \\ = \begin{pmatrix} aF_{n-1} + bF_n & bF_n + (a+b)F_{n+1} \\ aF_n + bF_{n+1} & bF_n + (a+b)F_{n+1} \end{pmatrix}$$

A mátrixok egyenlőségéből adódik: $R_n = aF_{n-1} + bF_n$

$\square$

1.2.8. Lemma. []

$$\begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^{n-1} \begin{pmatrix} 0 & 1 \\ 1 & c \end{pmatrix} = \begin{pmatrix} P_{n-1} & P_n \\ P_n & P_{n+1} \end{pmatrix}$$

Bizonyítás. Matematikai indukcióval:

$n = 1$ esetén érvényes az egyenlőség, $n = 2$ esetében nemkülönben.

Most tételezzük fel, hogy az állításunk igaz n -re, bizonyítsuk be $n + 1$ -re:

$$\begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix} \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^{n-1} \begin{pmatrix} 0 & 1 \\ 1 & c \end{pmatrix} =$$

az indukciós feltevés alapján

$$\begin{aligned} &= \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix} \begin{pmatrix} P_{n-1} & P_n \\ P_n & P_{n+1} \end{pmatrix} = \begin{pmatrix} P_n & P_{n+1} \\ dP_{n-1} + cP_n & dP_n + cP_{n+1} \end{pmatrix} = \\ &= \begin{pmatrix} P_n & P_{n+1} \\ P_{n+1} & P_{n+2} \end{pmatrix} \end{aligned}$$

A lemma helyességéről úgy is meggyőződhetünk, hogy $c = 1$ és $d = 1$ behelyettesítése esetén a 1.2.3 lemmát kapjuk. $\square$

1.2.9. Tétel. []

$$P_{m+n} = dP_{n-1}P_m + P_nP_{m+1}$$

Bizonyítás. A $\begin{pmatrix} 0 & 1 \\ 1 & c \end{pmatrix}$ mátrix inverze a $\begin{pmatrix} -c & 1 \\ 1 & 0 \end{pmatrix}$

Ezen kívül igaz még:

$$\begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^{m+n-1} \begin{pmatrix} 0 & 1 \\ 1 & c \end{pmatrix} = \begin{pmatrix} P_{m+n-1} & P_{m+n} \\ P_{m+n} & P_{m+n+1} \end{pmatrix}$$

Miközben:

$$\begin{aligned} &\begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^{m+n-1} \begin{pmatrix} 0 & 1 \\ 1 & c \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^n \mathbf{I} \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^{m-1} \begin{pmatrix} 0 & 1 \\ 1 & c \end{pmatrix} = \\ &= \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^n \begin{pmatrix} 0 & 1 \\ 1 & c \end{pmatrix} \begin{pmatrix} -c & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^{m-1} \begin{pmatrix} 0 & 1 \\ 1 & c \end{pmatrix} = \end{aligned}$$

Az előző lemma 1.2.8 alapján:

$$\begin{aligned} &= \begin{pmatrix} P_n & P_{n+1} \\ P_{n+1} & P_{n+2} \end{pmatrix} \begin{pmatrix} -c & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} P_{m-1} & P_m \\ P_m & P_{m+1} \end{pmatrix} = \\ &= \begin{pmatrix} -cP_n + P_{n+1} & P_n \\ -cP_{n+1} + P_{n+2} & P_{n+1} \end{pmatrix} \begin{pmatrix} P_{m-1} & P_m \\ P_m & P_{m+1} \end{pmatrix} = \\ &= \begin{pmatrix} (-cP_n + P_{n+1})P_{m-1} + P_nP_m & (-cP_n + P_{n+1})P_m + P_nP_{m+1} \\ (-cP_{n+1} + P_{n+2})P_{m-1} + P_{n+1}P_m & (-cP_{n+1} + P_{n+2})P_m + P_{n+1}P_{m+1} \end{pmatrix} \end{aligned}$$

A mátrixok egyenlősége alapján: $P_{m+n} = (-cP_n + P_{n+1})P_m + P_nP_{m+1}$ $\square$

Ennek a tételnek van egy számunkra fontos következménye:

1.2.10. Következmény. []

$$P_{2n} = -cP_n^2 + 2P_nP_{n+1}$$

$$P_{2n+1} = -cP_nP_{n+1} + P_{n+1}^2 + P_nP_{n+2}$$

Bizonyítás. Az első esetben n -t helyettesítünk az m helyére, a második egyenlőség esetén pedig $n + 1$ -et. $\square$

1.2.11. Lemma. []

$$\begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^{n-1} \begin{pmatrix} a & b \\ b & bc + ad \end{pmatrix} = \begin{pmatrix} Q_{n-1} & Q_n \\ Q_n & Q_{n+1} \end{pmatrix}$$

Bizonyítás. Matematikai indukció segítségével:

$n = 1$ és $n = 2$ esetén az egyenlőség érvényes.

Most tételezzük fel, hogy az állításunk igaz n -re, bizonyítsuk be $n + 1$ -re:

$$\begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix} \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^{n-1} \begin{pmatrix} a & b \\ b & bc + ad \end{pmatrix} =$$

ami az indukciós feltevés alapján

$$\begin{aligned} &= \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix} \begin{pmatrix} Q_{n-1} & Q_n \\ Q_n & Q_{n+1} \end{pmatrix} = \begin{pmatrix} Q_n & Q_{n+1} \\ dQ_{n-1} + cQ_n & dQ_n + cQ_{n+1} \end{pmatrix} = \\ &= \begin{pmatrix} Q_n & Q_{n+1} \\ Q_{n+1} & Q_{n+2} \end{pmatrix} \end{aligned}$$

$\square$

1.2.12. Tétel. []

$$Q_n = bP_n + adP_{n-1}$$

Bizonyítás.

$$\begin{aligned} \begin{pmatrix} Q_n & Q_{n+1} \\ Q_{n+1} & Q_{n+2} \end{pmatrix} &= \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^n \begin{pmatrix} a & b \\ b & bc + ad \end{pmatrix} = \\ &= \begin{pmatrix} P_n & P_{n+1} \\ P_{n+1} & P_{n+2} \end{pmatrix} \begin{pmatrix} -c & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a & b \\ b & bc + ad \end{pmatrix} = \\ &= \begin{pmatrix} P_n & P_{n+1} \\ P_{n+1} & P_{n+2} \end{pmatrix} \begin{pmatrix} -ac + b & -bc + bc + ad \\ a & b \end{pmatrix} = \end{aligned}$$

$$= \begin{pmatrix} (-ac + b)P_n + aP_{n+1} & adP_n + bP_{n+1} \\ (-ac + b)P_{n+1} + aP_{n+2} & adP_{n+1} + bP_{n+2} \end{pmatrix}$$

A mátrixok egyenlőségéből adódik:

$$Q_n = (-ac + b)P_n + aP_{n+1} = -acP_n + bP_n + acP_n + adP_{n-1} = bP_n + adP_{n-1}$$

□

1.3. A Fibonacci számsorozat n -edik elemének kiszámítása

A Fibonacci számsorozatról szóló tétel következménye 1.2.5 egy kész módszert ad a számsorozat n -edik elemének kiszámítására. Ezt a módszert szeretnénk a következőkben hatékonyabbá tenni. Mivel F_{2n} a már említett következmény alapján a sorozat $n - 1$ -edik, n -edik és $n + 1$ -edik tagjától függ, de F_{n+1} szintén összefüggésben áll F_{n-1} -el és F_n -el a 1.2.5 következményben leírt összefüggéseket az alábbi módon egyszerűsíthetjük:

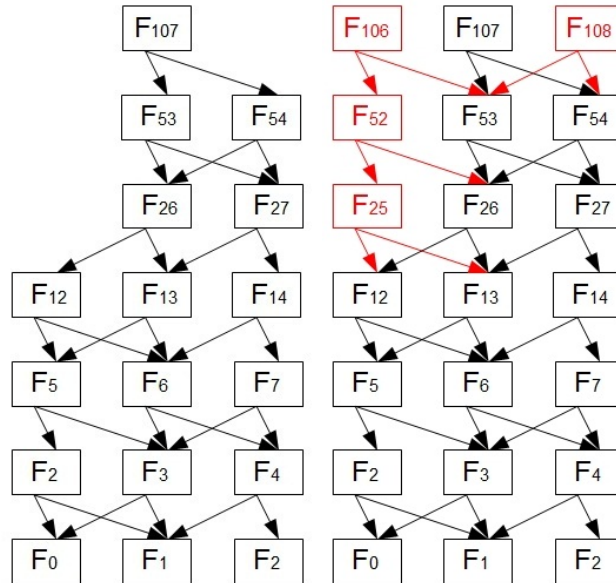
$$F_{2n} = F_n(F_{n+1} + F_{n-1}) = F_n(F_n + F_{n-1} + F_{n-1}) = 2F_{n-1}F_n + F_n^2$$

Hasonlóan egyszerűsíthetjük a másik összefüggést is, tehát:

$$F_{2n} = 2F_{n-1}F_n + F_n^2$$

$$F_{2n+1} = F_{n+1}^2 + F_n^2$$

Az alábbi séma az F_{107} kiszámításához szükséges rekurzív hívásokat mutatja:



Elsősorban észlelhetjük, hogy minimális változtatással képesek vagyunk az F_{107} mellett az F_{106} és az F_{108} értékeit is kiszámítani (második kép). Ez azért is fontos számunkra, mivel az R_n kiszámításához a 1.2.7 tétel alapján egyszerre van szükség az F_n és az F_{n-1} értékeire. Azt a szintet, ahol a bemeneti index található, nevezzük el a legmagasabbnak. Ugyanígy azon szintet, ahol a rekurzív hívások abbamaradnak, nevezzük el a legalacsonyabb szintnek. A legalacsonyabb szint tehát azon tagjait tartalmazza a sorozatnak, melyeket ismertnek nyilvánítunk. (ha szükséges ismertnek nyilváníthatjuk a sorozat harmadik és negyedik tagját is) Másodsorban észrevehetjük, hogy minden szint maximálisan 3 elemet tartalmaz, a sorozat 3 egymást követő elemét. Fokozatosan tehát bizonyítsuk be észrevételeinket.

1.3.1. Állítás. *[[A rekurzív hívások minden szintje pontosan három elemet tartalmaz, a Fibonacci számsorozat három egymást követő elemét.*

Bizonyítás. Az állítást matematikai indukcióval igazoljuk. A legmagasabb szint három általunk választott elemet tartalmaz, az indukció kezdete tehát biztosítva van. A bemeneti indexet válasszuk meg a szint középső elemének. A szomszédos két elemet úgy válasszuk meg, hogy az állításunk érvényes maradjon, tehát a sorozat két szomszédos tagját válasszuk meg.

Ezek után tételezzük fel, hogy az állítás érvényes egy bizonyos szintre, s ebből kiindulva bizonyítsuk be, hogy az eggyel alacsonyabb szintre is érvényes lesz az állításunk. Tegyük fel, hogy az a szint, melyre állításunk érvényes a sorozat $n - 1$ -edik, n -edik és $n + 1$ -edik tagját tartalmazza. Két lehetőségünk van, n lehet páros és páratlan.

Tegyük fel, hogy n páros. Tehát $n = 2k$, vagyis az indexek a következő módon módosulnak $2k - 1$, $2k$ és $2k + 1$. A módosított rekurzív összefüggések alapján tehát a sorozat alábbi tagjainak ismeretére van szükség az adott indexek kiszámítására:

$$F_{2k-1} = F_{2(k-1)+1} = F_{(k-1)+1}^2 + F_{k-1}^2 = F_k^2 + F_{k-1}^2$$

$$F_{2k} = 2F_{k-1}F_k + F_k^2$$

$$F_{2k+1} = F_{k+1}^2 + F_k^2$$

Ebből következik, hogy az eggyel alacsonyabb szint a sorozat $k - 1$ -edik, k -adik és $k + 1$ -edik tagját kell, hogy tartalmazza, s egyben semmilyen más elemet nem kell tartalmaznia. Ha n páratlan a bizonyítás hasonlóan történik. Ezzel be is fejeztük a matematikai indukciót, s bebizonyítottuk a kívánt állítást. $\square$

1.3.2. Állítás. *[[A rekurzív hívások szintjeinek száma $O(\log_2 n)$, ahol n a bemeneti index.*

Bizonyítás. Tekintsünk a szintek középső elemire, tehát a középső oszlopra a képeken. A legmagasabb szinten a középső elem a bemeneti index, a legalacsonyabb szinten egy olyan tagja a Fibonacci sorozatnak, melynek értékét ismerjük, tehát az első vagy második tag. (A legalacsonyabb szint az F_0 , F_1 , F_2 és az F_1 , F_2 , F_3 lehet, ezen tagok értékét ugyanis ismerjük.) A rekurzív összefüggésből látható, hogy minden szint középső elemének indexe nagyságrendben kétszer akkora, mint az eggyel alacsonyabb szint középső elemének indexe. Tehát érvényes:

$$z * 2^k \approx n,$$

ahol z vagy 1 vagy 2. k a szintek számát jelöli n pedig a bemeneti index.

$$2^k \approx n$$

$$k \approx \log_2 n$$

□

1.3.3. Állítás. *[[A számsorozat n -edik elemének kiszámítása közben a sorozat $(n-1)$ -ső és $(n+1)$ -ső eleme is kiszámítható, maximálisan $O(\log n)$ rekurzív hívás hozzáadásával.*

Bizonyítás. Ez az állítás valójában az előző állítás következménye. A szintek száma nagyságrendben $O(\log_2 n)$. A rekurzív hívásokból kitűnik, hogy a második legmagasabb szinttől az összes szint tartalmaz legalább két elemet. Ahhoz, hogy minden szint három elemet tartalmazzon tehát elég csak egy elemet hozzáadnunk minden szinthez. (Természetesen a legmagasabb szinthez kettőt.) Tehát annyi rekurzív hívást kell hozzátennünk, amennyi a rekurzív hívások szintjének száma. □

Ebben a fejezetben valójában egy egyszerű Oszd meg és uralkodj elvű algoritmust írtunk le. A következő fejezet eme algoritmus további optimalizálását tartalmazza az eddigi ismeretek alapján. Ezt az optimalizációt azzal érjük el, hogy a dinamikus programozás elvének megfelelő algoritmust alkotunk, miközben az algoritmus csak állandó mennyiségű változóra van szüksége.

1.3.1. Iteráló algoritmus a Fibonacci számsorozat n -edik elemének kiszámítására

Most leírjuk az algoritmus végleges formáját a Fibonacci számsorozat n -edik elemének kiszámítására.

Algoritmus(n) $aktindex \leftarrow getbit(n, 1) * 2 + getbit(n, 2)$ $P_0[0, 1, 2] \leftarrow [P_{aktindex-1}, P_{aktindex}, P_{aktindex+1}]$ $aktomb \leftarrow 0$ $inaktomb \leftarrow 1$ **for** $i = 3$ **to** $\log_2 n$ **do** $aktindex \leftarrow aktindex * 2 + getbit(n, i)$ $aktomb \leftarrow (aktomb + 1) \bmod 2$ $inaktomb \leftarrow (inaktomb + 1) \bmod 2$ **if** $aktindex$ páros **then** $P_{aktomb}[0] \leftarrow P_{inaktomb}[0]^2 + P_{inaktomb}[1]^2$ $P_{aktomb}[1] \leftarrow 2 * P_{inaktomb}[0] * P_{inaktomb}[1] + P_{inaktomb}[1]^2$ $P_{aktomb}[2] \leftarrow P_{inaktomb}[1]^2 + P_{inaktomb}[2]^2$ **else** $P_{aktomb}[0] \leftarrow 2 * P_{inaktomb}[0] * P_{inaktomb}[1] + P_{inaktomb}[1]^2$ $P_{aktomb}[1] \leftarrow P_{inaktomb}[1]^2 + P_{inaktomb}[2]^2$ $P_{aktomb}[2] \leftarrow 2 * P_{inaktomb}[1] * P_{inaktomb}[1] + P_{inaktomb}[2]^2$ **end if** $i \leftarrow i + 1$ **end for****return** P_{aktomb}

A $getbit(x, y)$ funkció az x y -dik legnagyobb bitjét adja vissza konstans időben. Ha az általunk választott programnyelv ezt nem teszi lehetővé, akkor az algoritmus futtatása előtt a számot szét kell bontanunk bitekre, hogy konstans időben hozzáférhetőek legyenek. Természetesen, ha már létezik ilyen funkció, használhatjuk azt. A szétbontás szintén $\log_2 x$ operáció végrehajtását igényli hiszen a szám kettes számrendszerbeli alakja ilyen hosszúságú.

1.3.4. Állítás. *[[A fenti algoritmus a Fibonacci számsorozat n -edik elemének kiszámításához $O(\log_2 n)$ számtani műveletet hajt végre, pontosabban $O(\log_2 n)$ szorzást.*

Bizonyítás. Az algoritmus egyetlen ciklust tartalmaz $\log_2 n$ -es felső határral. Egy ciklus leforgása alatt csak állandó mennyiségű szorzást, összeadást hajt végre. Az összeadás időigénye elhanyagolható a szorzásé mellett. $\square$

1.3.5. Állítás. *[[Az algoritmus $O(n)$ memóriát igényel.*

Bizonyítás. A memóriában állandó mennyiségű változót kell tárolni. Mivel azonban a számsorozat n -edik eleme 2^n nagyságú ez a számítógépen n hosszúságú egyesek és nullák sorozataként szerepel. Tehát az algoritmus végeztével $O(n)$ memóriát igényel. $\square$

1.3.6. Állítás. *[[A bemutatott algoritmus a Fibonacci számsorozat n -edik elemét számítja ki.*

Bizonyítás. Ahhoz, hogy megbizonyosodjunk arról, hogy az általunk leírt algoritmus valóban a 1.3 összefüggéseket implementálja, elég ha bebizonyítjuk, hogy az *aktindex* nevű változó a ciklus minden lefutása alatt egy bizonyos szintnek a középső elemét tartalmazza (pontosabban annak sorszámát). Belátható, hogy a

$$\text{aktindex} \leftarrow \text{aktindex} * 2 + \text{getbite}(n, i)$$

parancssor az aktuális szint középső eleméből (indexe legyen k) $2k$ vagy $2k + 1$ értékét veszi fel. Ez az összefüggés megfelel a 1.3 összefüggések elmentettjének, tehát megfelel az alacsonyabb szintektől a magasabb szintekig való számításnak. $\square$

1.4. A többi számtani sorozat n -edik elemének kiszámítása

A fejezet elején definiáltuk az R , P és Q számsorozatokat. (1.2.1 definíció) Az R számsorozat n -edik elemének kiszámításához alkalmazhatjuk az előző fejezetben bemutatott algoritmust. A 1.2.7 tétel alapján ugyanis ezen számsorozat n -edik elemét úgy kapjuk meg, ha elvégezzük a $aF_{n-1} + bF_n$ műveleteket. Mivel F_n és F_{n-1} egyidejűleg kiszámítható, a képletben használt a és b értékek állandók, ezért nem változik az előző algoritmus időigénye, sem pedig memóriaigénye.

A P számsorozat esetében hasonlóan járhatunk el, mint a Fibonacci számsorozat esetén. A 1.2.10 következmény azonos szerepet tölt be a P sorozatnál, mint a 1.2.5 a Fibonacci számsorozatnál. Így tehát apró változtatásokkal általánosíthatjuk a fenti algoritmust a P számsorozatnak megfelelőre. Mivel ez az algoritmus szinte azonos az előzővel, további bizonyítás nélkül mutatjuk be az így megváltoztatott algoritmust:

Algoritmus(n)

$$\text{aktindex} \leftarrow \text{getbite}(n, 1) * 2 + \text{getbite}(n, 2)$$

$$P_0[0, 1, 2] \leftarrow [F_{\text{aktindex}-1}, F_{\text{aktindex}}, F_{\text{aktindex}+1}]$$

$$\text{akttomb} \leftarrow 0$$

$$\text{inakttomb} \leftarrow 1$$

for $i = 3$ **to** $\log_2 n$ **do**

$$\text{aktindex} \leftarrow \text{aktindex} * 2 + \text{getbite}(n, i)$$

$$\text{akttomb} \leftarrow (\text{akttomb} + 1) \bmod 2$$

$$\text{inakttomb} \leftarrow (\text{inakttomb} + 1) \bmod 2$$

```

if aktindex páros then
     $P_{aktomb}[0] \leftarrow d * P_{inaktomb}[0]^2 + P_{inaktomb}[1]^2$ 
     $P_{aktomb}[1] \leftarrow 2 * d * P_{inaktomb}[0] * P_{inaktomb}[1] + c * P_{inaktomb}[1]^2$ 
     $P_{aktomb}[2] \leftarrow d * P_{inaktomb}[1]^2 + P_{inaktomb}[2]^2$ 
else
     $P_{aktomb}[0] \leftarrow 2 * d * P_{inaktomb}[0] * P_{inaktomb}[1] + c * P_{inaktomb}[1]^2$ 
     $P_{aktomb}[1] \leftarrow d * P_{inaktomb}[1]^2 + P_{inaktomb}[2]^2$ 
     $P_{aktomb}[2] \leftarrow 2 * d * P_{inaktomb}[1] * P_{inaktomb}[2] + c * P_{inaktomb}[2]^2$ 
end if
 $i \leftarrow i + 1$ 
end for
return  $P_{aktomb}$ 

```

A Q számsorozat esetén pedig hasonlóan járunk el, mint az R esetében. A 1.2.12 tétel alapján a Q sorozat n -edik elemét a P sorozat két egymást követő tagjából számíthatjuk ki. (Melyeket az előző algoritmus segítségével párhuzamosan kiszámíthatunk.)

1.5. Az abszolút taggal rendelkező másodrendű rekurzív számsorozatok n -edik elemének kiszámítása

Definiáljuk a $\bar{Q}_n$ sorozatot a következő módon: $\bar{Q}_n = c\bar{Q}_{n-1} + d\bar{Q}_{n-2} + e$, $\forall n \in \mathbb{N}$: $n \geq 2$ ahol $\bar{Q}_0 = a$, $\bar{Q}_1 = b$, és $a, b, c, d, e \in \mathbb{R}$. Az alábbiakban bemutatjuk, hogy az előző alfejezetben bemutatott algoritmus segítségével képesek vagyunk kiszámítani a $\bar{Q}_n$ sorozat n -edik elemét is $O(\log n)$ számtani művelet végrehajtása árán. Látható, hogy a $\bar{Q}_n$ sorozat a másodrendű rekurzív számsorozatok egy az eddigieknél még nagyobb halmazát jelöli. Elsőként néhány lemmát kell bebizonyítanunk:

1.5.1. Lemma. []

$$\begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^n = \begin{pmatrix} -cP_n + P_{n+1} & P_n \\ -cP_{n+1} + P_{n+2} & P_{n+1} \end{pmatrix}$$

Bizonyítás. A 1.2.8 lemmából következik:

$$\begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^n \begin{pmatrix} 0 & 1 \\ 1 & c \end{pmatrix} = \begin{pmatrix} P_n & P_{n+1} \\ P_{n+1} & P_{n+2} \end{pmatrix}$$

A 1.2.9 tétel bizonyítása során már kiszámítottuk a $\begin{pmatrix} 0 & 1 \\ 1 & c \end{pmatrix}$ mátrix inverzét. Az egyenlet mindkét oldalát kiszorozzuk ezzel a mátrixszal:

$$\begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^n = \begin{pmatrix} P_n & P_{n+1} \\ P_{n+1} & P_{n+2} \end{pmatrix} \begin{pmatrix} -c & 1 \\ 1 & 0 \end{pmatrix}$$

Az egyenlet jobb oldalát is kiszorozzuk a fent említett mátrixszal, s ezzel meg is kapjuk a kívánt egyenlőséget. $\square$

1.5.2. Lemma. *[Jelöljük $M_n := \sum_{i=1}^n \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^i$ Az M_n mátrix zárt alakja:*

$$M_n = \frac{1}{c+d-1} \begin{pmatrix} (c^2-c)P_{n+1} + (1-2c)P_{n+2} + P_{n+3} - d & (1-c)P_{n+1} + P_{n+2} - 1 \\ -cdP_{n+1} + (d-c)P_{n+2} + P_{n+3} - d & bP_{n+1} + P_{n+2} - c - d \end{pmatrix}$$

Bizonyítás.

$$M_n = \sum_{i=1}^n \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^i \quad (1.1)$$

$$\begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix} M_n = \sum_{i=2}^{n+1} \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^i \quad (1.2)$$

A (1.2)-(1.1) elvégzése után az alábbiakat kapjuk:

$$\begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix} M_n - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} M_n = \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^{n+1} - \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}$$

$$\begin{pmatrix} -1 & 1 \\ d & c-1 \end{pmatrix} M_n = \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^{n+1} - \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}$$

A $\begin{pmatrix} -1 & 1 \\ d & c-1 \end{pmatrix}$ mátrix inverze a $\begin{pmatrix} \frac{1-c}{c+d-1} & \frac{1}{c+d-1} \\ \frac{1}{c+d-1} & \frac{1}{c+d-1} \end{pmatrix}$ mátrix. Az előző lemma alapján tehát:

$$\begin{aligned} \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^{n+1} - \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix} &= \begin{pmatrix} -cP_{n+1} + P_{n+2} & P_{n+1} \\ -cP_{n+2} + P_{n+3} & P_{n+2} \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix} = \\ &= \begin{pmatrix} -cP_{n+1} + P_{n+2} & P_{n+1} - 1 \\ -cP_{n+2} + P_{n+3} - d & P_{n+2} - c \end{pmatrix} \\ \begin{pmatrix} -1 & 1 \\ d & c-1 \end{pmatrix} M_n &= \begin{pmatrix} -cP_{n+1} + P_{n+2} & P_{n+1} - 1 \\ -cP_{n+2} + P_{n+3} - d & P_{n+2} - c \end{pmatrix} \end{aligned}$$

Ezt az egyenletet szorozzuk meg a már említett mátrix inverzével.

$$M_n = \begin{pmatrix} \frac{1-c}{c+d-1} & \frac{1}{c+d-1} \\ \frac{1}{c+d-1} & \frac{1}{c+d-1} \end{pmatrix} \begin{pmatrix} -cP_{n+1} + P_{n+2} & P_{n+1} - 1 \\ -cP_{n+2} + P_{n+3} - d & P_{n+2} - c \end{pmatrix}$$

Szebb alakra hozva:

$$M_n = \frac{1}{c+d-1} \begin{pmatrix} (c^2 - c)P_{n+1} + (1 - 2c)P_{n+2} + P_{n+3} - d & (1 - c)P_{n+1} + P_{n+2} - 1 \\ -cdP_{n+1} + (d - c)P_{n+2} + P_{n+3} - d & bP_{n+1} + P_{n+2} - c - d \end{pmatrix}$$

Tehát a lemma érvényes. $\square$

Definiáljunk még egy számsorozatot: $S_n = cS_{n-1} + dS_{n-2} + e$, $\forall n \in \mathbb{N}$: $n \geq 2$ ahol $S_0 = 0$, $S_1 = 0$, és $c, d, e \in \mathbb{R}$.

1.5.3. Tétel. []

$$\bar{Q}_n = Q_n + S_n$$

Bizonyítás. Matematikai indukció segítségével. $n = 0$ és $n = 1$ esetében az állítás triviálisan igaz. Ezek után tételezzük fel, hogy igaz n -re, s ebből bizonyítsuk be, hogy igaz $n + 1$ -re is.

Tehát $\bar{Q}_n = c\bar{Q}_{n-1} + d\bar{Q}_{n-2} + e$ az indukciós feltevés alapján $\bar{Q}_n = cQ_{n-1} + cS_{n-1} + dQ_{n-2} + dS_{n-2} + e$, továbbá

$$\bar{Q}_n = cQ_{n-1} + dQ_{n-2} + cS_{n-1} + dS_{n-2} + e$$

$$\bar{Q}_n = Q_n + S_n$$

Tehát állításunk igaz. $\square$

Mivel a Q_n sorozat tagjait már ki tudjuk logaritmikus időben számítani az S_n sorozatot kell még - szintén logaritmikus időben - kiszámítanunk. Tehát szükségünk van a következő tételre:

1.5.4. Tétel. []

$$S_{n+2} = e \frac{dP_{n+1} + P_{n+2} - 1}{c + d - 1} + eP_{n+1}$$

Bizonyítás. Könnyen belátható, hogy

$$\begin{pmatrix} S_n & S_{n+1} \\ S_{n+1} & S_{n+2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ d & c \end{pmatrix}^n \begin{pmatrix} 0 & 0 \\ 0 & e \end{pmatrix} + M_n \begin{pmatrix} 0 & 0 \\ e & e \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ e & e \end{pmatrix}$$

Ezek után a 1.5.1 és a 1.5.2 lemmákat alkalmazva kapjuk:

$$\begin{pmatrix} S_n & S_{n+1} \\ S_{n+1} & S_{n+2} \end{pmatrix} = M_n \begin{pmatrix} 0 & 0 \\ e & e \end{pmatrix} + \begin{pmatrix} 0 & eP_n \\ 0 & eP_{n+1} + e \end{pmatrix}$$

Az M_n behelyettesítése után a mátrixok egyenlőségéből adódik:

$$S_{n+2} = e \frac{dP_{n+1} + P_{n+2} - d - c}{c + d - 1} + eP_{n+1} + e$$

Mely bizonyítja a tétel helyességét.. □

Mivel P_{n+1} és P_{n+2} értékeit képesek vagyunk logaritmikus időben kiszámítani, az előző tétel alapján S_n értékét is képesek vagyunk logaritmikus időben kiszámítani.

2. fejezet

Néhány az adott problémát oldó algoritmus analízise és összehasonlítása

2.1. A felhasznált algoritmusok

Ebben a fejezetben 4 algoritmust hasonlítunk össze, melyek a Fibonacci számsorozat n -edik elemét számítják ki. Ezek közül hármat különböző publikációkból vettünk át, negyedik pedig az itt bemutatott algoritmus.

Az első algoritmus a számsorozat rekurzív összefüggése alapján számítja ki a Fibonacci számsorozat n -edik elemét, nevezhetjük ezt naiv vagy természetes hozzáállásnak is a probléma megoldásához. A többi bemutatott algoritmustól több tekintetben is eltér. Mivel semmilyen optimalizációt nem tartalmaz, időigénye lineáris, tehát valószínűleg ez lesz a leglassúbb itt bemutatott algoritmus. Azonban helyességéhez nem férhet kétség, mivel a Fibonacci számsorozat definícióján alapul. Ugyanezért a számsorozat három egymást követő tagját kell elraktározunk a memóriában. Az első kettőből számítjuk ki a harmadik értékét. Ezt az algoritmust a **algoritmus1.cpp** fájlban implementáltuk.

A második algoritmus a Daisuke Takahashi által a [16] cikkben megjelentetett módszere. Ezt a módszert a **algoritmus4.cpp** fájlban implementáltuk.

A harmadik algoritmus neve Alternate, an integer iteration, mely - többek közt - a [8] cikkben is megtalálható. Ez az algoritmus hasonlít az itt bemutatott algoritmushoz, hiszen ugyanazon összefüggéseken alapul - 1.2.5. Ezt az algoritmust a **algoritmus5.cpp** fájlban implementáltuk.

A negyedik algoritmus az ebben a munkában bemutatott módszer, melyet a

algoritmus2.cpp fájlban implementáltuk.

Az utolsó három algoritmus több szempontból is hasonlít egymásra, mindegyik algoritmus egy $O(\log_2 n)$ hosszúságú ciklus segítségével számítja ki a sorozat újabb és újabb tagjait. Így összehasonlítható, hogy egyes algoritmusok a ciklusmag egy lefolyása alatt milyen fajta és mennyi számtani műveletet végeznek el. Azon változók, melyek a számsorozat tagjait tartalmazzák a program futása során a bemeneti indexnél nagyságrendekkel nagyobb értékeket is felvehetnek. (A Fibonacci számsorozat i -edik eleme 2^i nagyságrendű szám.)¹ Ezekkel a változókkal háromféle műveletet végeznek el: kiszorozhatják egymással a sorozat két tagját, négyzetre emelhetik a sorozat tagját, vagy összeadhatják a sorozat két tagját. Ezeken a számtani műveleteken kívül a nagyságrendekkel kisebb értékű változókkal, és a konstans értékekkel is végeznek számtani műveleteket. Ezeket közös gyűjtőnéven nevezzük el műveleteknek a kis számokon. Tehát négyféle műveletet különböztetünk meg:

- A sorozat két tagjának kiszorzása
- A sorozat valamely tagjának négyzetre emelése
- A sorozat két tagjának összeadása
- Műveletek a kis számokkal

Ezen műveleteket sorba tudjuk állítani az alapján, hogy elvégzésükhöz hány bitműveletre van szükség. A [16] alapján a négyzetre emelés végrehajtása kevesebb bitoperációt követel, mint a szorzás végrehajtása. Ugyancsak a [16] alapján az összeadás még a négyzetre emelésnél is kevesebb bitműveletet igényel. A kis számokkal végzett műveletek ezek mellett elhanyagolható mennyiségű ideig tartanak. Tehát az említett műveleteket az alábbi módon állíthatjuk sorrendbe:

szorzás > négyzetre emelés > összeadás $\gg$ számtani műveletek a kis számokon

¹A Binet-formula alapján bebizonyítható, hogy $F_n \sim \frac{\varphi^n}{\sqrt{5}}$ nagyságú szám, ahol φ az aranymetszés arányát jelöli, tehát $\varphi = \frac{1+\sqrt{5}}{2}$. A Binet-formula például a [15] bakalármunka is tartalmazza, de szinte mindenhol említik, ahol a Fibonacci számsorozatról írnak.

2.2. Az algoritmusok összehasonlítása a felhasznált aritmetikai operációk, és azok száma alapján

Ebben a fejezetben az egy ciklus lefolyása alatt végrehajtott számtani műveletek száma és fajtája alapján hasonlítjuk össze az algoritmusokat. Ugyan a számtani műveletek a kis számokon aszimptotikusan gyorsabbak, mint a többi felsorolt számtani művelet, a teljesség kedvéért azokat is feltüntetjük.

	szorzás	négyzetre emelés	összeadás	kis számok
D. T. algoritmus	0	2	6	8
Alternate	2	0	$4 - 5^2$	0
Az 1. fejezetben leírt algoritmus	$1 - 2^3$	$5 - 4^3$	3	$1 - 2^3$

2.1. táblázat. Egyes műveletek mennyisége egy ciklus lefolyása alatt

Az előző táblázat alapján képesek vagyunk felállítani egy sorrendet az algoritmusok közt. A D. Takahashi által bemutatott algoritmus lesz várhatóan a leggyorsabb, utána az Alternate, s végül az első fejezetben bemutatott algoritmus. Természetesen leglassúbb a fejezetben elsőként említett algoritmus lesz, mely a definíció alapján számítja ki a rekurzív számsorozatok elemeit. Hogy meggyőződjünk a felállított sorrend helyességéről, megmértük az algoritmusok futási idejét néhány bemeneti indexre. Mindegyik algoritmust C++ nyelvben implementáltunk az NTL könyvtár használatával. A mért eredmények a következő fejezetben lesznek kiértékelve.

²A markOdd(j) aktuális értéke alapján

³Az aktindex aktuális paritása alapján

2.3. Az algoritmusok összehasonlítása a mért futási idők alapján

Bemeneti	Ez egyes bemenetek futási ideje (sec)			
index	D. T. algoritmus	Alternate	1. fejezet	Naiv hozzáállás
1 000	0.000108957	9.20296e-05	0.00010705	0.000324965
10 000	0.000193834	0.000263929	0.000449181	0.00822306
100 000	0.00518703	0.00907183	0.019192	0.424006
1 000 000	0.125531	0.229387	0.487688	40.391
2 500 000	0.540508	0.97237	2.0968	251.345
5 000 000	1.61849	2.92191	6.27913	1006.04
10 000 000	4.84948	8.7129	18.9938	-

2.2. táblázat. Az algoritmusok futási ideje egyes bemenetekre másodpercekben

A mérések tehát megerősítették az elméleti vizsgálat alapján felállított sorrendet.

Befejezés

Elsősorban arra szeretném felhívni a figyelmet, hogy a munkában többször előforduló $O(\log_2 n)$ kifejezés természetesen ugyanazt a halmazt jelképezi, mint az $O(\log n)$. A kifejezésnek ez a módja arra szeretné felhívni a figyelmet, hogy minden bemutatott algoritmus felezésen alapul.

Fontos továbbá, hogy az itt leírt algoritmusok természetesen valós számokkal is működnek, s az implementációjukkor csak az egyszerűség kedvéért választottuk az egész számok halmazát.

Mint ahogy azt a második fejezetben bemutattuk, az első fejezetben leírt algoritmus nem a leghatékonyabb a probléma megoldására. Egy konkrét, adott számsorozat esetén készíthető gyorsabb algoritmus is. Ennek az algoritmusnak az előnye, hogy a sorozatok széles skálájára működik, ahogy azt az első fejezetben bemutattuk.

Végezetül pedig szeretnék még egyszer megköszönni minden segítséget.

Irodalomjegyzék

- [1] Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory - Volume I: Efficient Algorithms*. The MIT Press, London, 1997.
- [2] Daniel J. Bernstein. Detecting perfect powers in essentially linear time. *Mathematics of Computation*, 67:1253–1283, 1998.
- [3] Pavol Ďuriš. Tvorba efektívnych algoritmov. 2009.
- [4] Ira Gessel. Fibonacci is square (problem and solution h-187). *Fibonacci Quarterly*, 10(4):417, 1972.
- [5] V. E. Hoggatt, Jr. and Marjorie Bie Knell-Johnson. A primer for the Fibonacci numbers xvii. *Fibonacci Quarterly*, 16. <http://www.fq.math.ca/Scanned/16-2/hoggatt.pdf>.
- [6] Ross Honsberger. *Mathematical Gems III*. The Mathematical Association of America, USA, 1985.
- [7] Phillip James. When is a number Fibonacci? www.cs.swan.ac.uk/~csulrich/ftp/PhilJames1008.pdf, January 25, 2009.
- [8] L. F. Johnson. Golden an Alternating, fast simple $O(\lg n)$ algorithms for Fibonacci. 2010. <http://arxiv.org/abs/1011.0148>.
- [9] James P. Jones. Representation of solutions of Pell equations using Lucas sequences. *Sectio Mathematicae*, 30:75–86, 2003.
- [10] Dan Kalman and Robert Mena. The Fibonacci Numbers – Exposed. *Math. Mag.*, 76(3):167–181, June 2003.
- [11] Refik Keskin. Solutions of some quadratic Diophantine equations. *Computers and Mathematics with Applications*, 60:2225–2230, 2010.
- [12] Marco-Felipe King. *Fibonacci Numbers and Mathematical Phyllotaxis*. Department of Mathematics, University of Durham, Durham, 2008. bakalárska práca.

- [13] Péter Kiss. Diophantine representation of generalized Fibonacci numbers. *Elemente der Mathematik*, 34:129–132, 1979.
- [14] C. S. Simons and M. Wright. Fibonacci imposters. *International Journal of Mathematical Education in Science and Technology*, 38:677–682, 2006.
- [15] Beáta Štupáková. *Fibonacciho a Lucasove čísla*. FMFI UK, Bratislava, 2008. bakalárska práca.
- [16] Daisuke Takahashi. A fast algorithm for computing large Fibonacci numbers. *Information Processing Letters*, 75:243–246, 2000.