Szegedi Tudományegyetem

Bolyai Intézet

Geometria Tanszék

A Gao-féle algoritmus Reed-Solomon-féle hibajavító kódok dekódolására

MSc szakdolgozat

Készítette:
Táborosi Andor Zsolt
alkalmazott matematikus
szakos hallgató

Témavezető:
Dr. Nagy Gábor Péter
egyetemi docens

Szeged
2015

Tartalomjegyzék

Bevezetés

A kódelmélet egy következő kommunikációs modellel foglalkozik. A feladó valamilyen csatornán keresztül adatokat akar továbbítani egy címzettnek, a továbbítás során pedig az adatok egy része megváltozik, hibák keletkez(het)nek. Például a Marson lévő kutatórobotot kell irányítani a földi űrbázisról (feladó), és üzenetet küldenek neki egy paranccsal, de az üzenet az űrben való utazás alatt megváltozik, így előfordulhat, hogy a robot a hibás parancs hatására ellentétes irányban indul el. A kérdés az, hogy hogyan tudja a címzett (a robot) kiválasztani, hogy mely adatok változtak meg, és hogyan tudja a megváltozott adatokat kijavítani. Ebben az üzenetek különböző kialakításai, kódolásai segíthetnek.

A Reed-Solomon-féle kódokat Irving S. Reed és Gustave Solomon találták fel 1960-ban, de ekkor még nem találtak hatékony dekódoló algoritmust. Ezek a kódok az egyik legjobb hibajavító kódok, mert a kód elemei a lehető legtávolabb vannak egymástól, ami a dekódolás lehetőségeit növeli. Az ilyen kódokat MDS (maximal distance separable) kódoknak nevezzük. Ezáltal kiválóan kezelik a véletlenszerű és szándékos hibákat is. Egy korábbi dolgozatomban bemutattam Peterson 1960-ban kifejlesztett tünetalapú dekódolását, amit Elwyn Berlekamp és James Massey továbbfejlesztett 1969-ben, és ezért ezt a változatot Berlekamp-Massey-féle dekódoló algoritmusnak nevezik. Ezek az algoritmusok először a tüneteket számolják ki, majd meghatározzák a hibák helyeit és mértékét. A Berlekamp-Massy algoritmus négyzetes futási idejével hatékony a gyakorlatban, és így más lineáris kódok dekódolására is próbálják alkalmazni.

A Reed-Solomon-féle kódokat 1977-ben a Voyager-programban is használták, de manapság is a legnépszerűbb kódok több területen is: a digitális háttértárolók, CD-k, DVD-k, a digitális kommunikációs szerkezetek, a digitális videók vagy az űrfelderítésben használt technológiák területén.

Dolgozatomban bemutatok egy új megközelítését a Reed-Solomon-féle kódok dekódolásának, amit Shuhong Gao fedezett fel, illetve fejlesztett ki 2002-ben. A módszer hasonlít a Berlekamp-Massey algoritmusra, itt is a hibák helyének és értékének explicit kifejezése nélkül találjuk meg az üzenetpolinomot. Ez az üzenetpolinom egy bizonyos értelemben "látható", "észrevehető", "elkapható" az egész dekódoló folyamat során, és az algoritmus közvetlenül a polinomot keresi. A fő műveletek a polinomok interpolációja, részleges legnagyobb közös osztója és maradékos osztása. Ezeket mind lehet implementálni gyors Fourier-transzformáltakon alapuló algoritmusokkal.

1. Alapfogalmak

1.1. Kódelméleti alapfogalmak

Példa egy vektortérre: Legyen $m, n \in \mathbb{N}$, p prím és $q = p^m$. Ekkor $\mathbb{F}_q^n$ vektortér $\mathbb{F}_q$ felett, ahol

$$\mathbb{F}_q^n = \{(x_1, x_2, \ldots, x_n) | \forall i = 1, \ldots, n : x_i \in \mathbb{F}_q\}.$$

A továbbiakban q-ra és n-re mindig az előzőek szerint hivatkozunk.

Megjegyzés: A fenti vektortérre úgy is tekinthetünk, mint $\mathbb{F}_q$ feletti $n \times 1$-es, vagy $1 \times n$-es mátrixok, így alkalmasan felírva a mátrixszorzás is értelmezhető.

1.1. Definíció. *Ha F egy véges test, akkor a $C \subset V = F^n$ részhalmazt n hosszú **kódnak** nevezzük. V egy lineáris alterét egy n hosszú **lineáris kódnak** nevezzük. Ha $F = \mathbb{F}_2$, akkor pedig C-t bináris kódnak hívjuk.*

*A C kód elemeit **kódszavaknak** vagy kódvektoroknak nevezzük.*

A következőekben definiált fogalmakat egy általános F véges test esetén is lehetne definiálni, de mivel most az $\mathbb{F}_q^n$ véges testre koncentrálódik a témánk, így a fogalmak során erre a véges testre fogunk hivatkozni.

1.2. Definíció. *Legyen $\boldsymbol{v} = (v_1, v_2, \ldots, v_n), \boldsymbol{u} = (u_1, u_2, \ldots, u_n) \in \mathbb{F}_q^n = V$. A két vektor Hamming-távolságán azon koordináták számát értjük, amelyekben $\boldsymbol{v}$ és $\boldsymbol{u}$ eltér, azaz:*

$$d(\boldsymbol{u}, \boldsymbol{v}) = |\{i : 1 \leq i \leq n, v_i \neq u_i\}|.$$

*A $d\colon V \times V \to \mathbb{N}$ függvényt Hamming-metrikának nevezzük, a $\boldsymbol{v} \in V$ vektor **súlyán** pedig a $d(\boldsymbol{v}, \boldsymbol{0})$ távolságot értjük, és $|\boldsymbol{v}|$-vel jelöljük.*

Vegyük észre, hogy $\forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}_q^n$ vektor esetén

$$d(\boldsymbol{u}, \boldsymbol{v}) = |\{i | 1 \leq i \leq n, u_i - v_i \neq 0\}| = d(\boldsymbol{u} - \boldsymbol{v}, \boldsymbol{0}) = |\boldsymbol{u} - \boldsymbol{v}|.$$

Ezt a tulajdonságot felhasználva könnyen belátható a következő lemma.

1.3. Lemma. *Az $\mathbb{F}_q^n$ vektortéren a d leképezés metrika, azaz a következők teljesülnek $\forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_q^n$ esetén:*

a) $d(\boldsymbol{x}, \boldsymbol{y}) \geq 0$ *és* $d(\boldsymbol{x}, \boldsymbol{y}) = 0 \Leftrightarrow \boldsymbol{x} = \boldsymbol{y}$,

b) $d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x})$,

c) $d(\boldsymbol{x}, \boldsymbol{y}) \leq d(\boldsymbol{x}, \boldsymbol{z}) + d(\boldsymbol{z}, \boldsymbol{y})$.

1.4. Definíció. *A $\boldsymbol{c} \in \mathbb{F}_q^n$ középpontú $r \in \mathbb{R}_+$ sugarú gömbnek nevezzük a következő halmazt:*

$$\mathbb{B}_r(\boldsymbol{c}) = \{\boldsymbol{x} \in \mathbb{F}_q^n | d(\boldsymbol{c}, \boldsymbol{x}) \leq r\}.$$

Mivel $\mathbb{F}_q^n$ véges, ezért ez a gömb is véges sok elemet tartalmaz, és ezt az elemszámot egy kis kombinatorikával kiszámolhatjuk. Észrevehető, hogyha $r \geq n$, akkor a $\mathbb{B}_r(\boldsymbol{c})$ gömb bármely $c \in \mathbb{F}_q^n$ esetén $\mathbb{F}_q^n$ összes elemét tartalmazza, mivel a gömbben azok az elemek vannak, amelyek legfeljebb r koordinátában térnek el a középponttól, és triviális, hogy bármely elemre igaz, hogy egy másik elemtől legfeljebb az összes koordinátában különbözik. Másrészt ha $r = 0$, akkor a gömbnek csak a középpont az eleme.

Tehát feltehető, hogy $1 \leq r \leq n$. A $\mathbb{B}_r(\boldsymbol{c})$ gömb elemszámának meghatározásához definiáljuk a következő i-sugarú gömbhéjakat a $\boldsymbol{c}$ középpont körül ($0 \leq i \leq r$):

$$\mathbb{B}(i, \boldsymbol{c}) = \{\boldsymbol{x} \in \mathbb{F}_q^n | d(\boldsymbol{c}, \boldsymbol{x}) = i\}.$$

Ezekben a gömbhéjakban azok az elemei vannak $\mathbb{F}_q$-nak, amik pontosan i távolságra vannak $\boldsymbol{c}$-től. Világos, hogy:

$$|\mathbb{B}_r(\boldsymbol{c})| = \sum_{i=0}^{\lfloor r \rfloor} |\mathbb{B}(i, \boldsymbol{c})|.$$

Egy ilyen gömbhéj elemszáma pedig adott i-re $\binom{n}{i}(q-1)^i$. Ez onnan adódik, hogy $\binom{n}{i}$ választás van arra, hogy melyik koordinátákban különbözzön az adott elem $\boldsymbol{c}$-től, egy koordináta esetén pedig $q-1$ lehetőség van egy különböző koordinátát találni, mivel $|\mathbb{F}_q| = q$; és i darab koordinátaválasztás van, amik függetlenek, ezért a koordináták értékeinek kiválasztására $(q-1)^i$ lehetőség van. Tehát:

$$|\mathbb{B}_r(\boldsymbol{c})| = \sum_{i=0}^{\lfloor r \rfloor} |\mathbb{B}(i, \boldsymbol{c})| = \sum_{i=0}^{\lfloor r \rfloor} \binom{n}{i} (q-1)^i.$$

1.5. Definíció. *A*

$$d(C) = \min\{d(\boldsymbol{x}, \boldsymbol{y}) | \boldsymbol{x}, \boldsymbol{y} \in C, \boldsymbol{x} \neq \boldsymbol{y}\}$$

számot ***a C kód minimális távolságának*** *nevezzük.*

1.6. Tétel. *Ha C egy lineáris kód és* $\boldsymbol{d}$*=d(C), akkor*

$$\boldsymbol{d} = \min\{d(\boldsymbol{v}, \boldsymbol{0}) | \boldsymbol{v} \in C \setminus \{\boldsymbol{0}\}\}$$

Vagyis egy lineáris kód minimális távolsága megegyezik a legkisebb súlyú nem-zéró kódvektor súlyával.

1.7. Definíció. *Egy $t \in \mathbb{N}$ esetén a $C \subseteq \mathbb{F}_q^n$ kódot* ***t-hibajavító kódnak*** *nevezzük, ha $\forall$ különböző $\boldsymbol{u}, \boldsymbol{v} \in C$ esetén*

$$d(\boldsymbol{u}, \boldsymbol{v}) \geq 2t + 1.$$

Megjegyzés. Világos, hogy bármely C t-hibajavító kód esetén $d(C) \geq 2t + 1$, ebből pedig az következik, hogy a t sugarú gömbök diszjunktak, amit a következő lemma is megfogalmaz.

1.8. Lemma. *Ha C t-hibajavító kód, akkor tetszőleges $\boldsymbol{v} \in \mathbb{F}_q^n$ vektorhoz legfeljebb egy olyan $\boldsymbol{c} \in C$ kódszó létezik, amelyre $d(\boldsymbol{c}, \boldsymbol{v}) \leq t$.*

Ez könnyen belátható indirekt módon. Tegyük fel, hogy 2 különböző ilyen kódszó létezik egy adott $\boldsymbol{v}$-re: $\boldsymbol{c_1}$ és $\boldsymbol{c_2}$, azaz $d(\boldsymbol{c_1}, \boldsymbol{v}) \leq t$ és $d(\boldsymbol{c_2}, \boldsymbol{v}) \leq t$. Ekkor a háromszög-egyenlőtlenség alapján:

$$2t \geq d(\boldsymbol{c_1}, \boldsymbol{v}) + d(\boldsymbol{c_2}, \boldsymbol{v}) \geq d(\boldsymbol{c_1}, \boldsymbol{c_2}) \geq 2t + 1, \text{ mivel } C \text{ t-javító kód.}$$

Ezzel azt kaptuk, hogy $2t \geq 2t + 1$, ami ellentmondás, tehát a lemma igaz.

1.9. Definíció. *Ha a $C \subseteq \mathbb{F}_q^n$ lineáris kód dimenziója k, akkor* ***lineáris*** *$[n, k]$* ***kódnak*** *nevezzük.*

1.10. Definíció. *Ha $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_k$ a C lineáris $[n, k]$ kód egy bázisa, akkor azt a $k \times n$-es $\boldsymbol{G}$ mátrixot, melynek i-edik sora a $\boldsymbol{c}_i$ vektor ($i = 1, \ldots, k$), C* ***generátor mátrixának*** *nevezzük.*

1.11. Tétel (Singleton-korlát)**.** *Ha valamely C lineáris $[n, k]$-kód minimális távolsága d, akkor*

$$d \leq (n - k) + 1.$$

1.12. Definíció. *Ha egy lineáris $[n, k]$-kód minimális távolsága d, és teljesül a*

$$d = n - k + 1$$

egyenlőség, akkor a kódot ***MDS-kódnak*** *nevezzük.*

Megjegyzés. Az MDS a 'Maximal distance separable code' angol kifejezés rövidítése, ami magyarul maximális elválasztási távolságú kódot jelent. Ezt azért hívjuk így, mert az adott vektortér lineáris kódjai között erre (is) teljesül, hogy a kódszavak a lehető legtávolabb vannak egymástól, ez pedig a dekódolásnál bizonyul hasznosnak, ugyanis mivel a kijavítható hibák a kódszavak távolságától függenek ($d \geq 2t + 1$), így az MDS-kódok a lehető legtöbb hibát ki tudják javítani. Más szóval nincs olyan kód az adott vektortérben adott dimenzióval, ami több hibát tudna kijavítani.

2. Reed-Solomon-féle hibajavító kódok

A továbbiakban a következő jelöléseket rögzítem: q egy prímhatvány, $\mathbb{F}_q$ a q elemű véges test, n, k, d egészek a következő tulajdonságokkal: $1 \leq k < n \leq q$ és $d = n - k + 1$.

2.1. Felépítés

Legyenek $a_1, \ldots, a_n \in \mathbb{F}_q$ különböző rögzített elemei az $\mathbb{F}_q$ testnek. Egy rögzített $1 \leq k \leq n$ esetén a következőképp építünk fel egy (n, k, d) Reed-Solomon-féle hibajavító kódot.

Adott egy k hosszú $(m_1, m_2, \ldots, m_k)$ információ, ahol $m_i \in \mathbb{F}_q$ $\forall i$-re. Először képezzük az ehhez tartozó üzenetpolinomot:

$$f(x) = m_1 + m_2 x + m_3 x^2 + \ldots + m_k x^{k-1} \in \mathbb{F}_q[x], \tag{1}$$

majd kiszámítjuk a

$$c_i = f(a_i) \in \mathbb{F}_q, i = 1, 2, \ldots, n \tag{2}$$

értékeket. Ekkor a megfelelő kódszó:

$$\mathbf{c} = (c_1, c_2, \ldots, c_n).$$

Ha az összes lehetséges információt lekódoljuk, azaz a rögzített a_1, $a_2, \ldots, a_n$ elemeket behelyettesítjük a k-nál kisebb fokú polinomokba, akkor q^k darab n hosszú kódszót kapunk, és ezek fogják alkotni a Reed-Solomon-féle kódunkat. Erről egyszerűen belátható, hogy egy $\mathbb{F}_q$ feletti, $d = n - k + 1$ minimális távolsággal rendelkező lineáris $[n, k]$-kód, és ezt nevezzük Reed-Solomon (n, k, d)-kódnak.

2.2. Dekódolás Shuhong Gao algoritmusával

Tegyük fel, hogy a $\mathbf{b} = (b_1, b_2, \ldots, b_n) \in \mathbb{F}_q^n$ üzenetet kaptuk, és hogy ezt a $\mathbf{c}$ kódszó t helyen való elrontásával/elromlásával keletkezett, ahol $t \leq (d-1)/2$. A célunk az (1)-beli $f(x)$ üzenetpolinom megtalálása, ami a $\mathbf{c}$ kódszót definiálta.

Először is kiszámítjuk előre a következő polinomot:

$$g_0 = \prod_{i=1}^{n} (x - a_i) \in \mathbb{F}_q[x].$$

Vegyük észre, hogy g_0 sok esetben ismert. Például $g_0 = x^q - x$, ha $n = q$, és $g_0 = x^n - 1$, ha $n \mid q - 1$ és $a_1, a_2, \ldots, a_n$ egy multiplikatív csoportot alkotnak $\mathbb{F}_q$-ban.

A **b** üzenet dekódolását végző algoritmust a következőképp definiáljuk.

1. Algoritmus: Reed-Solomon-féle kódok dekódolása

Input: Egy beérkezett $\mathbf{b} \in \mathbb{F}_q^n$ vektor.

Output: Az $m_1 + m_2 x + \ldots + m_k x^{k-1}$ üzenetpolinom, vagy "Dekódolási hiba".

1. lépés: (**Interpoláció**) Találjuk meg az egyetlen, legfeljebb $n-1$-fokú $g_1(x) \in \mathbb{F}_q[x]$ polinomot, melyre

$$g_1(a_i) = b_i, 1 \leq i \leq n.$$

2. lépés: (**Részleges legnagyobb közös osztó**) Alkalmazzuk a kiterjesztett euklideszi algoritmust a $g_0(x)$ és $g_1(x)$ polinomokra. Azaz elkezdjük az euklideszi algoritmus lépéseit, és a futás közben minden maradékhoz kiszámítjuk azokat a polinomokat, amik szükségesek a $g_0(x)$ és $g_1(x)$ azon lineáris kombinációjához, ami az adott maradékot előállítja (részletesebben lásd lentebb). Viszont nem számítjuk végig ezt az algoritmust, hanem megállunk akkor, amikor a maradék, jelöljük most $g(x)$-szel, foka kisebb, mint $\frac{1}{2}(n+k)$. Tegyük fel, hogy ekkor

$$u(x)g_0(x) + v(x)g_1(x) = g(x).$$

3. lépés: (**Maradékos osztás**) Osszuk el $g(x)$-et $v(x)$-szel, és tegyük fel, hogy

$$g(x) = f_1(x)v(x) + r(x),$$

ahol $\deg r(x) < \deg v(x)$. Ha $r(x) = 0$ és $f_1(x)$ foka kisebb k-nál, akkor az output $f_1(x)$, különben az output "Dekódolási hiba" (amiből az következik, hogy $(d-1)/2$-nél több hiba történt).

Később látni fogjuk, hogy a $v(x)$ polinom valójában a hibakereső polinom, aminek a gyökei megadják a hibák helyeit. A fenti dekódoló algoritmusnak viszont nem szükséges ismernie a hibák helyeit és nagyságát.

A következőkben bemutatjuk, hogy miért képes a hibahatáron belül a dekódolásra a Gao-féle algoritmus.

Először idézzük fel a kiterjesztett euklideszi algoritmust, amit két, nemzéró, $r_0, r_1 \in \mathbb{F}_q[x]$ polinomra alkalmazunk. Legyenek

$$u_0 = 1, u_1 = 0, v_0 = 0, v_1 = 1.$$

A kiterjesztett euklideszi algoritmus maradékos osztások sorozatából áll:

$$r_{i-1} = q_i r_i + r_{i+1}, \deg r_{i+1} < \deg r_i, i = 1, 2, \ldots, m,$$

ahol $r_i \neq 0$, $1 \leq i \leq m$, és $r_{m+1} = 0$. Továbbá egyidejűleg kiszámítjuk a következőket:

$$\begin{aligned} u_{i+1} &= u_{i-1} - q_i u_i \\ v_{i+1} &= v_{i-1} - q_i v_i, \end{aligned}$$
minden $1 \leq i \leq m$-re. (3)

Indukcióval könnyen belátható, hogy $\text{lnko}(r_0, r_1) = r_m$ (ha szükséges, főpolinommá alakítva), és hogy

$$r_i = u_i r_0 + v_i r_1, 0 \leq i \leq m + 1. \tag{4}$$

2.1. Lemma. *Legyen r_0 és r_1 két nem-zérus polinom $\mathbb{F}_q$ felett. Tegyük fel, hogy a kiterjesztett euklideszi algoritmus a fenti műveletekből áll. Ekkor*

$$u_{m+1} = (-1)^{m+1} \frac{r_1}{r_m}, v_{m+1} = (-1)^m \frac{r_0}{r_m}.$$

Bizonyítás. A (3)-ból következik, hogy

$$\begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix} \begin{bmatrix} u_{i-1} & v_{i-1} \\ u_i & v_i \end{bmatrix}.$$

Ezt a mátrixegyenletet i-szer ismételve kapjuk, hogy

$$\begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -q_{i-1} \end{bmatrix} \cdots \begin{bmatrix} 0 & 1 \\ 1 & -q_1 \end{bmatrix} \begin{bmatrix} u_0 & v_0 \\ u_1 & v_1 \end{bmatrix}.$$

Emlékezzünk arra, hogy az $\begin{bmatrix} u_0 & v_0 \\ u_1 & v_1 \end{bmatrix}$ mátrix a 2×2-es egységmátrix, így a determinánsok szorzástétele alapján

$$u_i v_{i+1} - v_i u_{i+1} = \det \begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} = \prod_{j=1}^{i} \det \begin{bmatrix} 0 & 1 \\ 1 & -q_j \end{bmatrix} = (-1)^i.$$

A (4) egyenletből tudjuk, hogy

$$\begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix} = \begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \end{bmatrix}, 0 \leq i \leq m.$$

Összevetve a két eredményt

$$\begin{bmatrix} r_0 \\ r_1 \end{bmatrix} = \begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix}^{-1} \begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix} = (-1)^i \begin{bmatrix} v_{i+1} & -v_i \\ -u_{i+1} & u_i \end{bmatrix} \begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix}, 0 \leq i \leq m.$$

Mivel $r_{m+1} = 0$, a fenti egyenlet $i = m$ esetéből következik a lemma állítása.

□

A következő lemma elengedhetetlen a dekódoló algoritmusunkhoz.

2.2. Lemma. *Legyen* $g_0(x) = w_0(x) \cdot r_0(x) + \epsilon_0(x)$ *és* $g_1(x) = w_0(x) \cdot r_1(x) + \epsilon_1(x)$, *ahol* $lnko(r_0(x), r_1(x)) = 1$ *és*

$$\deg r_i(x) \leq t, \deg \epsilon_i(x) \leq l, i = 1, 2.$$

Tegyük fel, hogy d_0*-ra teljesül, hogy*

$$\deg w_0(x) \geq d_0 > l + t.$$

Alkalmazzuk a kiterjesztett euklideszi algoritmust $g_0(x)$*-re és* $g_1(x)$*-re úgy, hogy megállunk, ha a maradék* $g(x)$ *foka kisebb* d_0*-nál. Tegyük fel, hogy ennél a megállásnál*

$$u(x)g_0(x) + v(x)g_1(x) = g(x).$$

Ekkor

$$u(x) = -\alpha r_1(x),\ v(x) = \alpha r_0(x)$$

valamely nullától különböző $\alpha \in \mathbb{F}_q$*-ra.*

Bizonyítás. Először belátjuk, hogy a kiterjesztett euklideszi algoritmus ugyanazt a hányadossorozatot állítja elő az r_0 és r_1 párra, mint a g_0, g_1 párra. Pontosítva: tegyük fel, hogy

$$r_{i-1} = q_i r_i + r_{i+1},\ \deg r_{i+1} < \deg r_i,\ i = 1, 2, \ldots, m,$$

ahol $r_{m+1} = 0$ és $r_m \in \mathbb{F}_q \setminus \{0\}$ (mivel $\text{lnko}(r_0, r_1) = 1$, és az algoritmus végén csak a legnagyobb közös osztó skalárszorosait kaphatjuk eredményként). Legyenek

$$u_0 = 1, \quad u_1 = 0, \quad u_{i+1} = u_{i-1} - q_i u_i$$
$$v_0 = 0, \quad v_1 = 1, \quad v_{i+1} = v_{i-1} - q_i v_i,$$
$$\forall 1 \leq i \leq m\text{-re}.$$

Ekkor

$$r_i = u_i r_0 + v_i r_1, 0 \leq i \leq m+1,$$

továbbá

$$\deg u_i \leq \deg r_1 \leq t, \ \deg v_i \leq \deg r_0 \leq t.$$

A 2.1 lemma alapján

$$u_{m+1} = (-1)^{m+1} r_1 / r_m, \ v_{m+1} = (-1)^m r_0 / r_m. \tag{5}$$

Definiáljuk a következőket:

$$g_i = u_i g_0 + v_i g_1, 2 \leq i \leq m+1.$$

Világos, hogy ekkor

$$g_{i-1} = q_i g_i + g_{i+1}, 1 \leq i \leq m.$$

Még be kell látnunk, hogy a $g_1, \ldots, g_{m+1}$ polinomok fokai szigorúan csökkennek. Valóban, minden $0 \leq i \leq m+1$-re $g_i = u_i(w_0 r_0 + \epsilon_0) + v_i(w_0 r_1 + \epsilon_1) = w_0(u_i r_0 + v_i r_1) + (u_i \epsilon_0 + v_i \epsilon_1) = w_0 r_i + (u_i \epsilon_0 + v_i \epsilon_1)$.

Mivel $\deg u_i \epsilon_0 + v_i \epsilon_1 \leq l + t < d_0 \leq \deg w_0$ minden $0 \leq i \leq m+1$-re, így

$$\deg g_i = \deg w_0 + \deg r_i \geq \deg w_0 \geq d_0 > l + t, 0 \leq i \leq m,$$

illetve

$$\deg g_{m+1} = \deg (u_{m+1} \epsilon_0 + v_{m+1} \epsilon_1) \leq l + t.$$

A fenti feltevésünk szerint $r_1, \ldots, r_m$ fokai szigorúan csökkennek, és így $g_1, \ldots, g_{m+1}$ fokai is. Ez azt jelenti, hogy g_0 és g_1 hányadossorozata pontosan $q_1, \ldots, q_m$, legalább az m-edik lépésig, azaz r_0 és r_1 sorozatával megegyezik. Ebből pedig az következik, hogy az u és v sorozatok is ugyanazok a g_0, g_1 párra, mint az r_0, r_1 párra. Hasonlóan, csak az m-edik lépésben fordul elő először, hogy a g_{m+1} maradék foka kisebb d_0-nál, és ebben a lépésben

$$u_{m+1}g_0 + v_{m+1}g_1 = g_{m+1},$$

ahol u_{m+1} és v_{m+1} az (5) -beli alakban áll elő. Ezzel pedig beláttuk a lemmát.

□

Most térjünk vissza az algoritmusunk helyességéhez.

2.3. Tétel. *Ha a kapott* **b** *üzenet legfeljebb* $(d-1)/2$ *távolságra van az* $f(x)$ *üzenetpolinom által definiált* **c** *kódszótól, akkor (1)-ben az* ***1. Algoritmus*** *eredménye* $f(x)$*, különben pedig "Dekódolási hiba".*

Bizonyítás. Tegyük fel, hogy a kapott vektor, $\mathbf{b} = (b_1, b_2, \ldots, b_n)$, $t \leq (d-1)/2$ távolságra van egy adott, **c** kódszótól, ami egy $f(x)$ üzenetpolinom által van definiálva az (1) és a (2)-beli módon. Definiáljuk a hibakereső polinomot a következőképp:

$$w(x) = \prod_{\substack{1 \leq i \leq n \\ c_i \neq b_i}} (x - a_i),$$

így $\deg w = t$. Legyen $w_0(x)$ a társtényezője $w(x)$-nek $g_0(x)$-ben, azaz

$$g_0(x) = w_0(x)w(x).$$

Továbbá legyen $\bar{w}(x) \in \mathbb{F}_q[x]$ az az egyetlen polinom, melyre $\deg \bar{w} < t$ és

$$\bar{w}(a_i) = (b_i - c_i)/w_0(a_i), \text{ minden } 1 \leq i \leq n\text{-re, ahol } c_i \neq b_i.$$

Ekkor $\text{lnko}(w(x), \bar{w}(x)) = 1$, és

$$g_1(x) = w_0(x) \cdot \bar{w}(x) + f(x),$$

mivel mindkét oldalnak kisebb a foka n-nél és ugyanazokat a b_i értékeket veszi fel a két oldal az a_i helyeken kiértékelve, $1 \leq i \leq n$-re.

Legyen $d_0 = (n+k)/2$. Vegyük észre, hogy

$$\deg(w_0(x)) = n - t \geq d_0 > k - 1 + t \geq \deg(f(x)) + \deg(w(x)).$$

Ekkor a 2.2 lemma alapján megkapjuk a következő polinomokat:

$$u(x) = -\alpha\bar{w}(x) \text{ és } v(x) = \alpha w(x)$$

valamely $\alpha \in \mathbb{F}_q \setminus \{0\}$ esetén. Így

$$g(x) = u(x)g_0(x) + v(x)g_1(x) = v(x)f(x).$$

Ez azt jelenti, hogy az **1. Algoritmus** 3. lépésében a maradéknak zérónak kellene lennie és a hányados $f_1(x)$ megegyezik az $f(x)$-szel (aminek k-nál kisebb a foka), mint ahogy azt vártuk.

Másrészről tegyük fel, hogy az algoritmus eredménye egy $f_1(x)$ polinom a 3. lépésben. Nyilvánvalóan az $f_1(x)$ polinom is egy kódszóhoz tartozó üzenetpolinom (mivel a foka kisebb k-nál). A 2. lépésbeli egyenletből következik, hogy

$$u(x)g_0(x) = v(x)(f_1(x) - g_1(x)),$$

így

$$v(a_i)(f_1(a_i) - g_1(a_i)) = 0, 1 \leq i \leq n.$$

De $v(x)$ foka $t \leq (d-1)/2$, amiből látszik, hogy $f_1(a_i) = g_1(a_i)$ legalább $n - t \geq n - (d-1)/2$ értékére az i-nek. Így a $\mathbf{b}$ az $f_1(x)$ által definiált kódszótól legfeljebb $(d-1)/2$ távolságra van. Következésképp, ha $\mathbf{b}$ minden kódszótól $(d-1)/2$-nél nagyobb távolságra van, akkor az algoritmus kénytelen "Dekódolási hibát" visszaadni a 3. lépésben.

□

A 2.3 tétel megmutatja, hogy az egyetlen lehetséges dekódolási hiba akkor jelentkezik, amikor a küldés során fellépő túl sok hiba miatt az eredeti kódszó egy másik kódszó $(d-1)/2$ sugarú gömbjébe kerül. Ez egy olyan eset, amit csak a kapott szó alapján lehetetlen észrevenni.

A fenti bizonyítás jobban megvilágítja a dekódoló algoritmust. Vegyük észre, hogy

$$g_0 = w_0(x)w(x) \tag{6}$$
$$g_1 = w_0(x)\bar{w}(x) + f(x), \tag{7}$$

ahol $w(x)$ a hibakereső polinom. A 2. lépésben kiszámított $u(x)$ és $v(x)$ polinomok tulajdonképpen rendre $-\alpha\bar{w}(x)$ és $\alpha w(x)$ valamely nem-nulla $\alpha \in \mathbb{F}_q$-ra. Így

$$f(x) = g_1(x) + \frac{g_0(x)}{v(x)}u(x). \tag{8}$$

Az $f(x)$ üzenetpolinom el van rejtőzve $g_1(x)$-ben, bár "látható" az algoritmusunk számára, és már az elején megcélozhatjuk.

Egy másik megközelítéssel, a 2.2 lemmából megfigyelhetjük, hogy $u(x)$ és $v(x)$ kiszámíthatóak csak a g_0 és g_1 együtthatóinak felső részeiből. Valóban, az $\epsilon_0(x)$ és $\epsilon_1(x)$ polinomok (az alsó részek) tetszőlegesek lehetnek, így nem befolyásolják az $u(x)$-hez és $v(x)$-hez vezető számításokat. Ez alapján egyszerűen figyelmen kívül hagyhatjuk az együtthatókat az alsó részeken. Az l foknak (az $\epsilon_i(x)$-ek fokának) teljesítenie kell, hogy

$$n - l \geq l + t,$$

amiből következik, hogy

$$l \leq n - 2t - 1.$$

Mivel $2t + 1 \leq d$, feltehető, hogy $l = n - d$. A 2. lépés kezdeténél újraírhatjuk $g_0(x)$-et és $g_1(x)$-et mint

$$g_0(x) = (\text{legfeljebb } n-d\text{-fokú tagok}) + x^{n-d+1}s_0(x), \tag{9}$$

$$g_1(x) = (\text{legfeljebb } n-d\text{-fokú tagok}) + x^{n-d+1}s_1(x). \tag{10}$$

Azaz $s_0(x)$ és $s_1(x)$ nem mások, mint rendre a $g_0(x)$ és $g_1(x)$ együtthatóinak felső részei. Például $s_0(x) = x^{d-1}$, ha $g_0 = x^q - x$ vagy $x^n - 1$. A 2.2 lemma alapján alkalmazhatjuk az euklideszi algoritmust közvetlenül az $s_0(x)$-re és $s_1(x)$-re, hogy megkapjuk $u(x)$-et és $v(x)$-et. Itt $s_1(x)$-nek legfeljebb $d - 2$ a foka (a $d - 1$ felső együtthatója g_1-nek) és a kapott szó tüneteként szolgál. Így az 1. Algoritmus a következőképp módosítható:

1a. Algoritmus: Reed-Solomon-féle kódok dekódolása (módosítva)

Input: Egy beérkezett $\mathbf{b} = (b_1, b_2, \ldots, b_n) \in \mathbb{F}_q^n$ vektor.

Output: Egy $m_1 + m_2x + \ldots + m_kx^{k-1}$ üzenetpolinom, vagy "Dekódolási hiba."

1. lépés: (**Interpoláció**) Keressük meg az egyetlen $g_1(x) \in \mathbb{F}_q[x]$ legfeljebb $n - 1$-ed fokú polinomot, amelyre

$$g_1(a_i) = b_i, 1 \leq i \leq n.$$

2. lépés: (**Részleges legnagyobb közös osztó**) Alkossuk meg az $s_0(x)$ és $s_1(x)$ polinomokat (9) és (10) alapján $g_0(x)$-ből és $g_1(x)$-ből. Alkalmazzuk a kiterjesztett euklideszi algoritmust $s_0(x)$-re és $s_1(x)$-re. Álljunk meg, amikor a maradék, legyen $g(x)$, foka $< (d+1)/2$. Tegyük fel, hogy ebben a lépésben

$$u(x)s_0(x) + v(x)s_1(x) = g(x).$$

3. lépés: (**Osztás és szorzás**) Osszuk el $g_0(x)$-et $v(x)$-szel, és tegyük fel, hogy

$$g_0(x) = h_1(x)v(x) + r(x),$$

ahol $\deg r(x) < \deg v(x)$. Ha $r(x) \neq 0$, akkor legyen az output "Dekódolási hiba", különben számítsuk ki a következőt:

$$f_1(x) := g_1(x) + h_1(x)u(x).$$

Ha $f_1(x)$ foka kisebb k-nál, akkor legyen az output $f_1(x)$, különben pedig "Dekódolási hiba".

Evvel a módosított algoritmussal a legnagyobb közös osztót számoló lépés olcsóbb, de az utolsó lépés drágább. A 2.3 tétel erre a módosított verzióra is teljesül.

2.3. Dekódolás hibákkal és törlésekkel

Egyszerű az algoritmusunkat átalakítani úgy, hogy törlésekkel is tudjon dolgozni. Ez egy kívánatos dolog, mivel a gyakorlatban gyakran van lehetőségünk ismerni a hibák elhelyezkedését, és ez az információ jelentősen megnöveli a kód teljesítményét.

Legyen $\mathbf{b} = (b_1, b_2, \ldots, b_n) \in \mathbb{F}_q^n$ egy kapott üzenet s darab törléssel. Ekkor ezeken felül t hibát javíthatunk, ahol

$$2t + s < d = n - k + 1.$$

Legyen S a törlési pozíciók halmaza (azaz azon helyek, ahol hibát észleltünk), és a törlési polinom a következő:

$$s(x) = \prod_{i \in S}(x - a_i).$$

Ha az eredeti Reed-Solomon kódszavakban az S-beli helyeket figyelmen kívül hagyjuk, akkor egy új Reed-Solomon-féle kódot kapunk $n - s$ hosszal, k dimenzióval és $d = n - s - k + 1$ távolsággal. Ahhoz, hogy $\mathbf{b}$-t dekódoljuk, elég egyszerűen az S-beli pozíciókat figyelmen kívül hagynunk és a kapott vektorra alkalmaznunk az 1. vagy az 1a. Algoritmust. Pontosabban az 1. vagy az 1a. Algoritmusban csak le kell cserélnünk az n-et $n - s$-re, a d-t $d - s$-re, $g_0(x)$-et $g_0(x)/s(x)$-re, illetve az interpolációs lépésben $g_1(x)$-től megkövetelni, hogy legfeljebb $n - s - 1$-ed fokú legyen, és teljesítse, hogy $g_1(a_i) = b_i$ minden $i \notin S$-re. Ez még mindig működni fog, mivel az átalakított kódra is teljesülnek a kívánalmak, azaz Reed-Solomon-féle kód.

3. A Shuhong Gao dekódoló algoritmus programozása

Az **1. Algoritmust** a SAGE online programcsomag segítségével implementáltam egy futtatható programbeli függvénybe. Azt a speciális esetet vizsgáltam, amikor a Reed-Solomon-féle kód a q elemű véges test egy multiplikatív részcsoportja által van generálva, azaz a kódot meghatározó $a_1, \ldots, a_n \in \mathbb{F}_q$-k egy multiplikatív részcsoport elemei. A függvény paramétereinek a kód alapjául szolgáló véges testet ($\mathbb{F}_q$), a kód dimenzióját (k), a multiplikatív részcsoport egy generáló elemét (a; mivel ciklikus, így elég egyet megadni) és a megérkezett, dekódolandó üzenetet (b) kell megadnunk. A függvény az **1. Algoritmus** lépéseit hajtja végre, és ha ez az algoritmus az üzenetpolinomot adja vissza, akkor a függvény az ezen polinom segítségével megkonstruált kódszót eredményezi. Ha viszont az algoritmus hibát jelez, akkor a függvény is. Ha a futás közben egyéb hiba történt, például rossz paramétert adtak meg, akkor azt jelzi a függvény, általában az 1 érték felvételével.

3.1. Forráskód

```
def decoder(F,k,a,b):
    if not(a in F):
        print "Az adott kódgeneráló elem nem a test eleme."
        return 1
    if a.multiplicative_order()!=F(a).multiplicative_order
       ():
        a=F(a) #arra az esetre, ha
        #alapértelmezetten más testnek az eleme
    R.<x>=PolynomialRing(F)
    o=a.multiplicative_order()
    if o<=k:
        print "A kód dimenziója nagyobb, mint a hossza,
                de ez nem lehetséges."
        return 1
    if b.__len__()!=o:
        print "A dekódolandó vektor hossza nem
                egyezik meg a kód hosszával."
        return 1
    #paraméterek ellenôrizve
    M=[a^j for j in range(o)]
    print "A(z)", a, "elem által generált multiplikatív
        részcsoport a ", F, " testben:"
    print M
    C=codes.ReedSolomonCode(o,k,F,M)
```

```
24      print "A C kód:", C
25      g0=R(x**o-1)
26      print "0. lépés: g0(x)=", g0
27      L=[(a^j,b[j]) for j in range(o)]
28      E=R.divided_difference(L)
29      g1=sum(E[l]*prod((x-a^j) for j in range(l))
30          for l in range(o))
31      print "1. lépés: interpolációs polinom: g1="
32      print g1
33      g,u,v=xeucalg(g0,g1,o,k,R)
34      print "2. lépés: a kiterjesztett euklideszi alg.
35          eredménye: g="
36      print g
37      print "és a g1 együtthatója abban a lineáris
38          kombinációban, ami g1-bôl és g0-ból elôállítja g-t:
               "
39      print "v=",v
40      r=g.quo_rem(v)
41      f1=r[0]
42      print "3. lépés: leosztjuk g-t maradékosan v-vel,
43          és annak a hányadosa, ill. maradéka:"
44      print r[0]
45      print r[1]
46      print "Ha a maradék 0 és a hányados foka k-nál kisebb,
47           akkor a hányados együtthatói a dekódolt kódszó."
48      if ((r[1]!=0) | (f1.degree()>=k)):
49          return "Dekódolási hiba."
50      EH=f1.coeffs()
51      print "Együtthatói a hányadosnak:", EH
52      m=[f1(a^j) for j in range(o)]
53      print "A dekódolás után kapott kódszó:", m
54      return m
```

3.2. Magyarázat

A program 2-17. sorában egyrészt a paraméterek helyességét ellenőrizzük, azt feltételezve, hogy a testet jól adta meg a felhasználó. Másrészt az 5-7. sorban azt a fellépő problémát oldottam meg, hogy az a generáló elem néhány választása esetében (pl. ha $a = 2$) előfordult, hogy a multiplikatív rendjét a beépített függvényétől lekérdezve nem az $\mathbb{F}_q$-beli multiplikatív rendjét kaptuk eredményül. Ezt a multiplikatív rendet az o változóban tároltuk el.

Az **1. Algoritmusbeli** $g_0(x)$ felépítésénél kihasználtuk, hogy a kód alapjául szolgáló elemek multiplikatív részcsoportot alkotnak, és mivel o ennek a csoportnak az elemszáma, így

$$g_0(x) = x^o - 1.$$

Ezek után az **1. Algoritmus** interpolációs lépése következik. Ehhez a SAGE polinomgyűrűkhöz tartozó *divided_difference* függvényét használtuk. Ennek az argumentumába egy kétdimenziós vektorokból álló listát kell megadni ($[(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)]$, a mi esetünkben $[(a^1, b_1)$, $(a^2, b_2), \ldots, (a^o, b_o)]$), és az eredmény szintén egy lista ($E = [E_i | i = 0, \ldots, o - 1]$), aminek segítségével azt a Lagrange-interpolációs polinomot tudjuk előállítani ($L(x)$), amit a vektorok első koordinátájának helyén véve a második koordinátát kapjuk eredményül ($L(x_i) = y_i$). A polinomot pedig, amit az **1. Algoritmus** jelöléseit követve g_1-gyel jelölünk a mi esetünkben, a kapott lista elemeit felhasználva a következőképp kapjuk meg:

$$g_1(x) = \sum_{i=0}^{o-1} E_i \prod_{j=0}^{i-1} (x - a^j).$$

Ezek után a kiterjesztett euklideszi algoritmust kell végrehajtani, amit egy külön, *xeucalg* nevű függvényben definiáltam.

3.2.1. A *xeucalg* függvény forráskódja

```
1  def xeucalg(f,g,n,k,R):
2      r0=R(f);r1=R(g)
3      u0=1;u1=0
4      v0=0;v1=1
5      while True:
6          if (r1==0) or (r1.degree()<(n+k)/2):
7              return [r1,u1,v1]
8          X=r0.quo_rem(r1)
9          q=X[0]
10         r2=X[1]
11         u2=u0-q*u1
12         v2=v0-q*v1
13         r0,r1=r1,r2
14         u0,u1=u1,u2
15         v0,v1=v1,v2
```

3.2.2. A *xeucalg* függvény magyarázata

A *xeucalg* függvény a kiterjesztett euklideszi algoritmust hajtja végre a paraméterként megadott $f, g \in R$ polinomokra, ahol az R polinomgyűrűt is paraméterként kérjük be. Az algoritmus akkor áll meg, amikor az **1. Algoritmus** által meghatározott $\frac{n+k}{2}$ határ alá csökken a maradék foka. A futás alatt pedig hasonlóan a fenti algoritmus lépéseit hajtja végre, és ahhoz, hogy ez a SAGE-ben megfelelően lefusson, biztosítani kell, hogy a program tényleg az adott gyűrűbeli polinomokként dolgozzon a paraméterként megadott polinomokkal, erről gondoskodtam a 2. sorban.

3.2.3. A főprogram magyarázatának folytatása

A kiterjesztett euklideszi algoritmus végrehajtása után (35. sor) az **1. Algoritmus** további lépéseit implementáltam. Ehhez felhasználtam a polinomok maradékos osztását végrehajtó *quo_rem* beépített függvényt (40. sor), majd a *coeffs* függvényt (51. sor), ami az adott polinomnak az együtthatóit adja vissza a legnagyobb nem 0 kitevőig. Azaz például ha $f(x) = a_0 + a_2x^2 + a_5x^5 + a_6x^6$, akkor $f.coeffs() = [a_0, 0, a_2, 0, 0, a_5, a_6]$. Itt az **1. Algoritmus** véget is ér, hiszen megkaptuk a kódszót meghatározó üzenetpolinomot. A kapott üzenet dekódolásához ezt úgy használhatjuk fel, hogy sorra kiszámoljuk ennek a polinomnak a multiplikatív részcsoport elemein vett behelyettesítési értékét, amit az 52. sorban hajtottam végre. Így pedig megkaptuk végül a kívánt kódszót, ha az **1. Algoritmus** implementált változata hibátlanul lefutott.

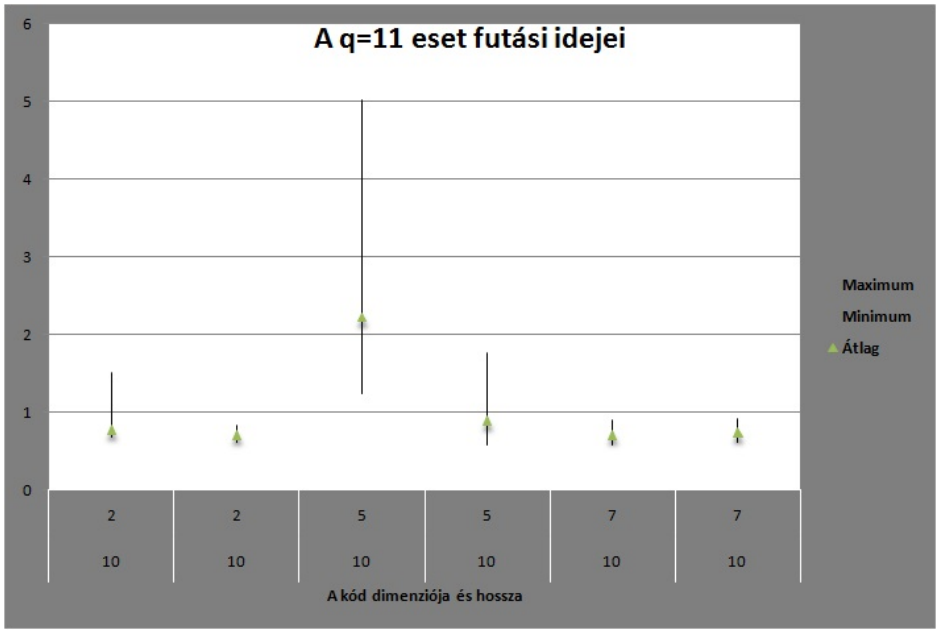
Ha viszont az üzenetpolinom együtthatóira vagyunk kíváncsiak, amit az **1. Algoritmus** is keres, akkor óvatosnak kell lennünk. Az $f.coeffs()$ függvény ugyanis csak az üzenetpolinom főegyütthatójáig adja vissza az együtthatókat, és ha az f foka $k-1$-nél kisebb volt, akkor az információs vektor végén legalább egy darab nulla szerepel, de az $f.coeffs$ ezt már nem tárolja el és egy k-nál rövidebb vektort ad vissza. Ezt a SAGE-ben a listák összevonásával meg lehet oldani úgy, hogy egy $k - len(f.coeffs())$ elemű, csak nullákból álló listát hozzáadunk az $f.coeffs()$-hez, így kiegészül egy k hosszú listává, ami már a kódolt információt tartalmazza.

3.3. Teszteredmények

A dolgozatom elkészítése során a megírt programot néhány rögzített paraméter esetén teszteltem gyorsaságra, illetve hibajavítási képességre. A paramétereket a következőképp választottam: $11 \leq q \leq 121$, n a $q-1$ osztói közül került ki egy bizonyos korlátot meghaladva, k-nak pedig a következő értékeket választottam kerekítve: $\frac{n}{5}$, $\frac{n}{2}$, $\frac{3n}{4}$. Ezek után azt vizsgáltam, hogy egy a paraméterektől függő t hibamennyiséget ki tud-e javítani. Mivel egy (n, k, d) Reed-Solomon-féle hibajavítókód $\lfloor \frac{n-k}{2} \rfloor$ hibát ki tud javítani, amit jelöljünk H-val, így t-t a $H, H+1$ értékek közül választottam. Minden ilyen paraméterre generáltam 100 véletlen kódszót, majd t helyen generáltam bennük hibákat, és a kapott vektorokra végrehajtottam a programot. Ezeket a teszteredményeket tartalmazzák az alábbi ábrák.

3.3.1. Rögzített paraméterek: $q = 11$, $n = 10$

Kezdetben a 11 elemű test feletti, a 2 által generált multiplikatív részcsoport által meghatározott $(15, k, d)$ Reed-Solomon-féle kódokat vizsgáltam. Ekkor $d = n - k + 1$, ami alkalmas k-kra kiszámolható, ez alapján pedig a hibajavítási képesség is a fentiek alapján.



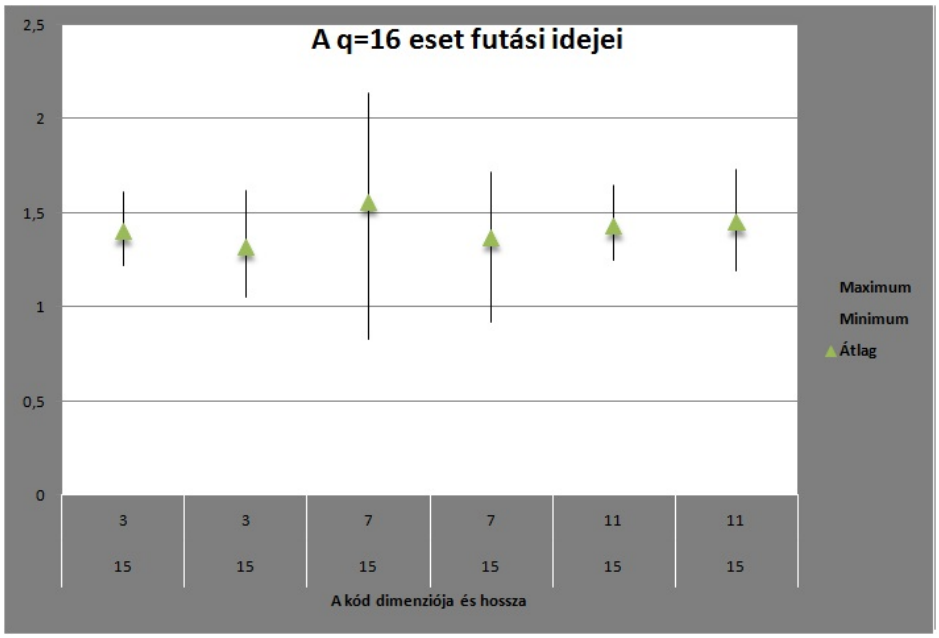
1. ábra. $q = 11$, $n = 10$

A vízszintes tengelyen az alsó sorban az n értékek, azaz a vizsgált kód hossza szerepel. A felső sorban pedig a k értékek, azaz a dimenziók, amikből azért szerepel mindegyik kétszer, mert minden k értékre kétszer futtattam le a tesztet. Először úgy, hogy a véletlen generált kódszavak elrontott változataiban a hibák még éppen a korláton belül voltak (azaz a fenti H-t vették fel), majd másodjára már olyan rontott üzeneteket generáltam, amikben éppen átlépjük a korlátot, azaz a fenti $H + 1$ hiba szerepel bennük.

A függőleges tengely a futási időt mutatja másodpercben. Mint láthatjuk, általában 0,8 és 1,5 mp között mozgott a futási idő, az átlag pedig 1 mp alatt maradt egy teszt kivételével.

3.3.2. Rögzített paraméterek: $q = 16$, $n = 15$

Következőleg a 16 elemű test feletti, c által generált multiplikatív részcsoport által meghatározott $(10, k, d)$ Reed-Solomon-féle kódokat vizsgáltam, ahol c-vel a test kiterjesztése közben kapott komplex elemet jelöltem. Alkalmas k dimenziókra itt is a minimális távolság és a hibajavítási képesség kiszámítható.

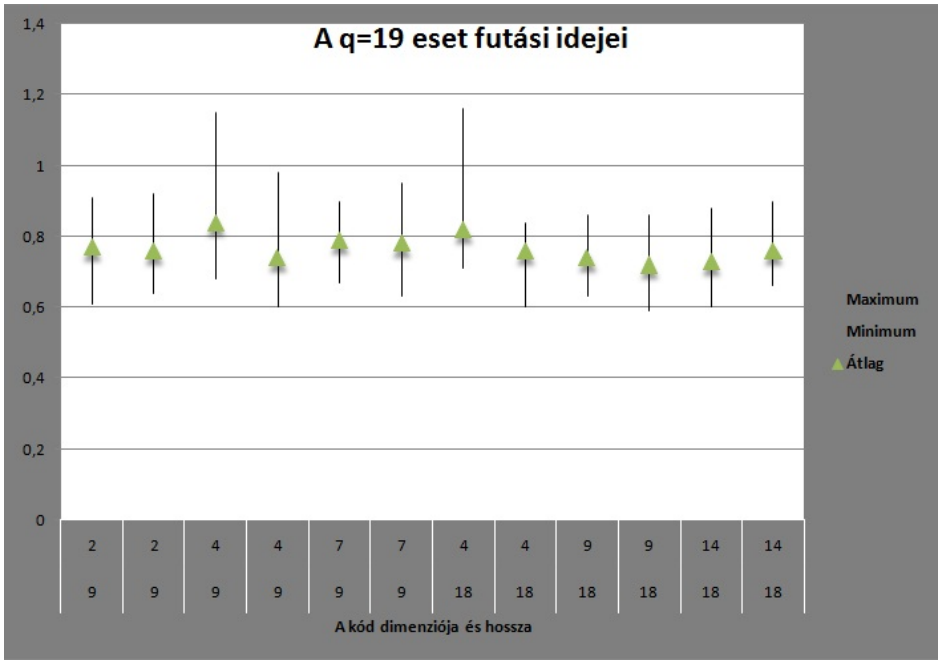


2. ábra. $q = 16$, $n = 15$

A tengelyek az előző ábrához hasonló adatokat mutatnak, azaz a futási időt, a kódok dimenziót és hosszait. Az előzőhöz képest itt az átlagos futási idő 1,5 mp körül mozgott, ez valószínűleg annak köszönhető, hogy a 16 nem prím, csak prímhatvány, így a test egy bővített test.

3.3.3. Rögzített paraméterek: $q = 19$, $n \in \{9, 18\}$

A következő tesztnél a 19 elemű test feletti, 4, ill. 2 által generált multiplikatív részcsoportok által meghatározott (n, k, d) Reed-Solomon-féle kódokat vizsgáltam. Hasonlóan az eddigiekhez a kódok hosszait az elemek multiplikatív rendjei határozták meg, jelen esetben a 2-nek 18 a rendje, s ezek után a k-kat a fejezet elején leírt módon számoltam ki. Ezek után adott dimenziókra a minimális távolság és a hibajavítási képesség könnyen kiszámítható.

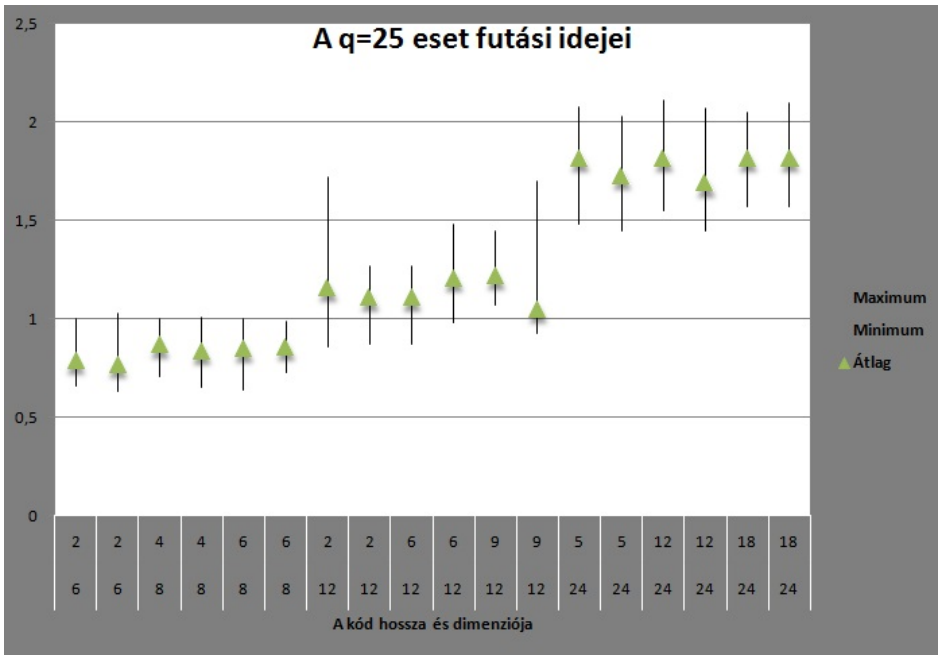


3. ábra. $q = 19$, $n \in \{9, 18\}$

Az ábra az eddigiekhez hasonló adatokat tartalmaz. A futási idők átlagai 0,8 mp körül mozognak, ami kisebb a 16 elemű test esetében végzett kísérletek során, ezért következtettem, hogy gyorsabb a program, ha a test elemszáma prím, mintha egy prímhatvány.

3.3.4. Rögzített paraméterek: $q = 25$, $n \in \{6, 8, 12, 24\}$

Következőleg a 25 elemű test feletti, c, ill. alkalmas hatványai által generált multiplikatív részcsoportok által meghatározott (n, k, d) Reed-Solomon-féle kódokat vizsgáltam (c a 16 esetéhez hasonlóan definiált). Ugyanis c rendje 24, és ha egy n rendű elemet szeretnénk kapni, ahol $n|24$, akkor a $c^{24/n}$ elem megfelelő választás. A további számításokat az előzőekhez hasonlóan végeztem.

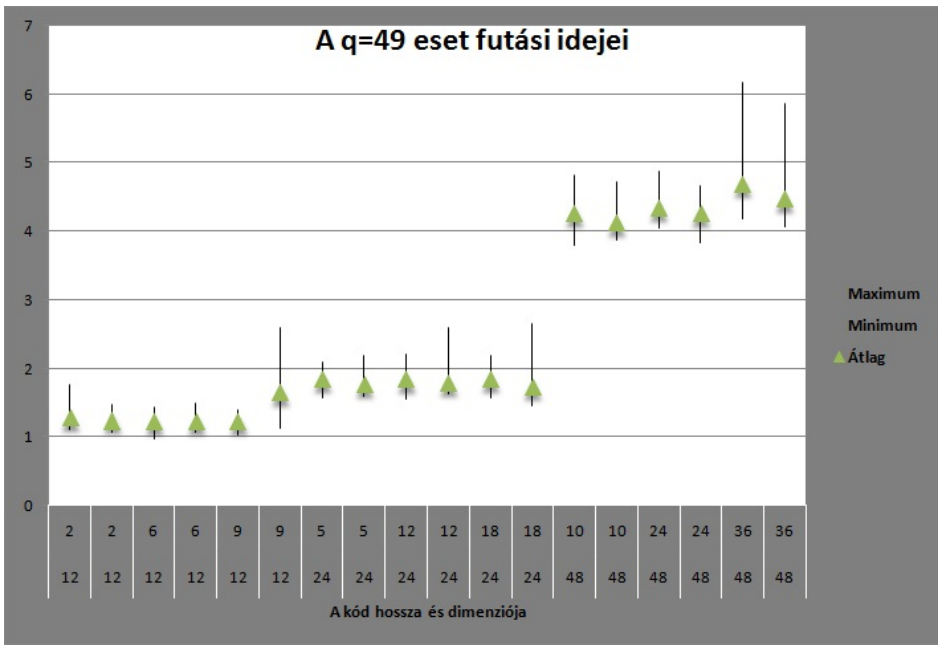


4. ábra. $q = 25$, $n \in \{6, 8, 12, 24\}$

Itt már több lehetőség volt a $q - 1$-es érték osztóira, és az eredmények is változatosabbak. A 12-nél kisebb hosszú kódok esetén a dekódolás megint viszonylag gyorsan lefutott, a 12-es hosszúságnál már "lelassult" átlagosan 1,2 mp-re, de leginkább a legnagyobb, 24-es hosszúságú kódnál tartott a legtovább, átlagosan 1,75 mp-ig a dekódolás. Bár itt csak százalékosan nagy a különbség, a későbbiekben érdemes lesz emlékezni ezekre.

3.3.5. Rögzített paraméterek: $q = 49$, $n \in \{12, 24, 48\}$

Ezek után a 49 elemű test feletti, c, ill. adott rendű hatványai által generált multiplikatív részcsoportok által meghatározott (n, k, d) Reed-Solomon-féle kódokat vizsgáltam (c ismét a kiterjesztett test komplex eleme). A további számításokat az előzőekhez hasonlóan végeztem.

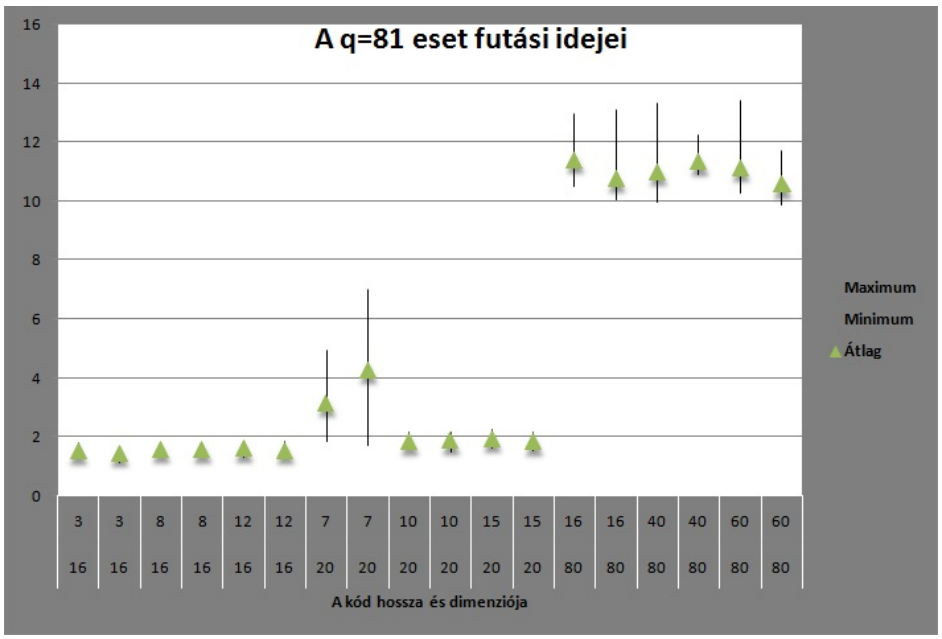


5. ábra. $q = 49$, $n \in \{12, 24, 48\}$

Hasonlóan az előzőhöz, itt is amikor a kód hossza megegyezett $q - 1$-gyel, megugrott a futási idő, de itt már nagyobb mértékben, ugyanis átlagos 1,2-1,9 mp helyett a 48 hosszú kódok dekódolása átlagosan 4,1-4,8 mp-ig tartott.

3.3.6. Rögzített paraméterek: $q = 81$, $n \in \{16, 20, 80\}$

Ezek után a 81 elemű test feletti, c, ill. alkalmas rendű hatványai által generált multiplikatív részcsoportok által meghatározott (n, k, d) Reed-Solomon-féle kódokat vizsgáltam (c hasonlóan az előzőekhez a kiterjesztett test komplex eleme). A további számításokat az előzőekhez hasonlóan végeztem.

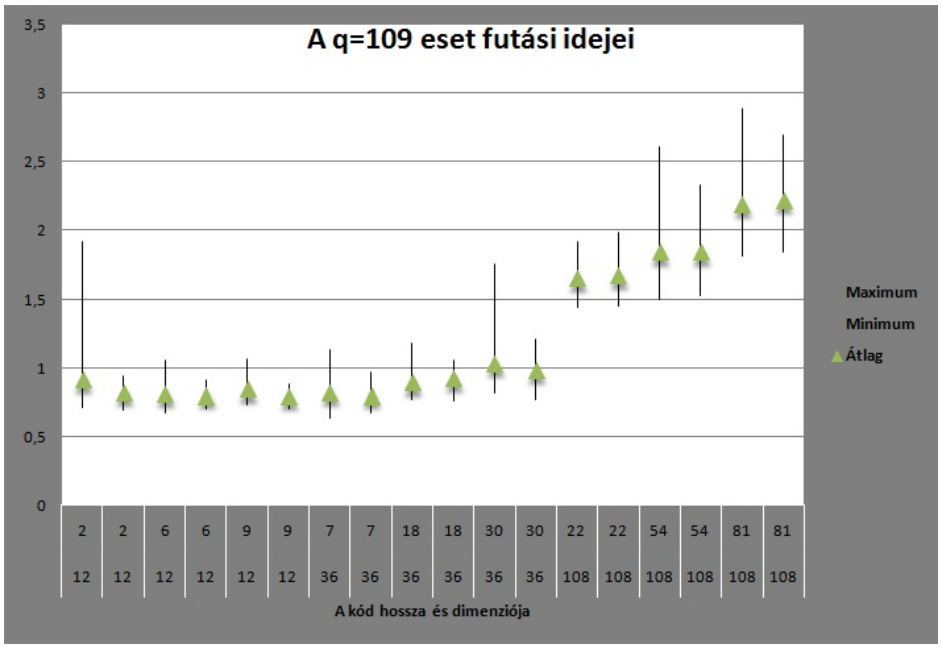


6. ábra. $q = 81$, $n \in \{16, 20, 80\}$

Néhány kiugró adattól eltekintve itt is érvényesül az a tendencia, hogy minél nagyobb a hossz, annál tovább tart a programnak lefutnia, különösen, ha a hossz megegyezik a test multiplikatív csoportjának a rendjével.

3.3.7. Rögzített paraméterek: $q = 109$, $n \in \{12, 36, 108\}$

A következő teszt során a 109 elemű test feletti, 10, ill. alkalmas rendű hatványai által generált multiplikatív részcsoportok által meghatározott (n, k, d) Reed-Solomon-féle kódokat vizsgáltam. Ugyanis 10 multiplikatív rendje 108, és így a fentiekhez hasonlóan megkapható a többi generáló elem. A további számításokat az előzőekhez hasonlóan végeztem.

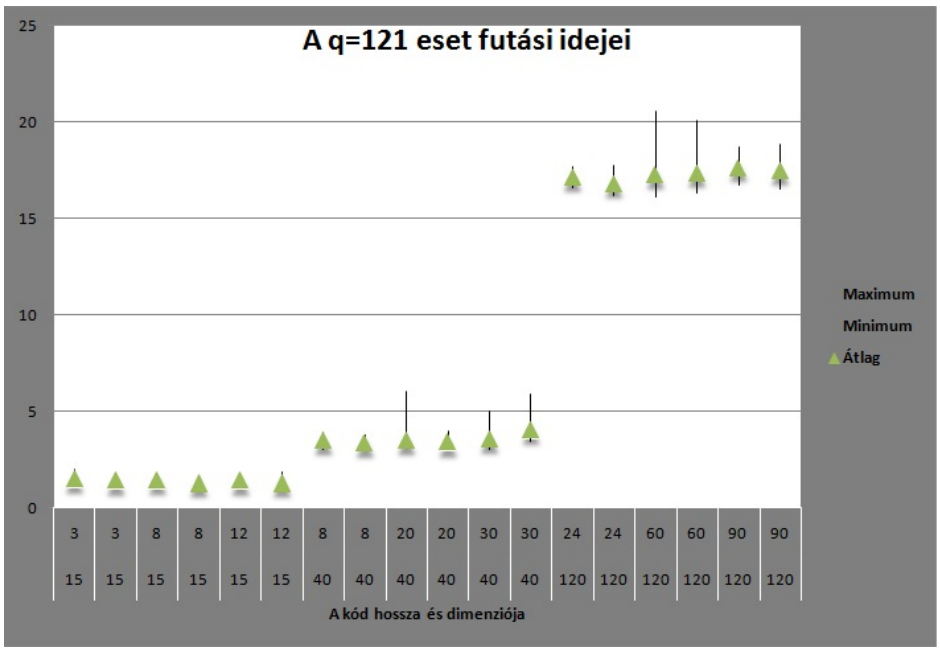


7. ábra. $q = 109$, $n \in \{12, 36, 108\}$

Az ábra alapján itt is az eddig megfigyelt jelenségek tükröződnek, minél hosszabb a kód, annál tovább fut a program. Viszont az is észrevehető, hogy ez ismét egy prím elemű test, és pl. a $q = 81$ esethez képest sokkal gyorsabban lefutott a program.

3.3.8. Rögzített paraméterek: $q = 121$, $n \in \{15, 40, 120\}$

Az utolsó prímhatványnak a 121-et választottam, és itt is a testbővítés során létrejött c komplex elemet és hatványait választottam azon részcsoportok generáló elemének, amik a Reed-Solomon-féle kódokat meghatározták. A további számításokat az előzőekhez hasonlóan végeztem.

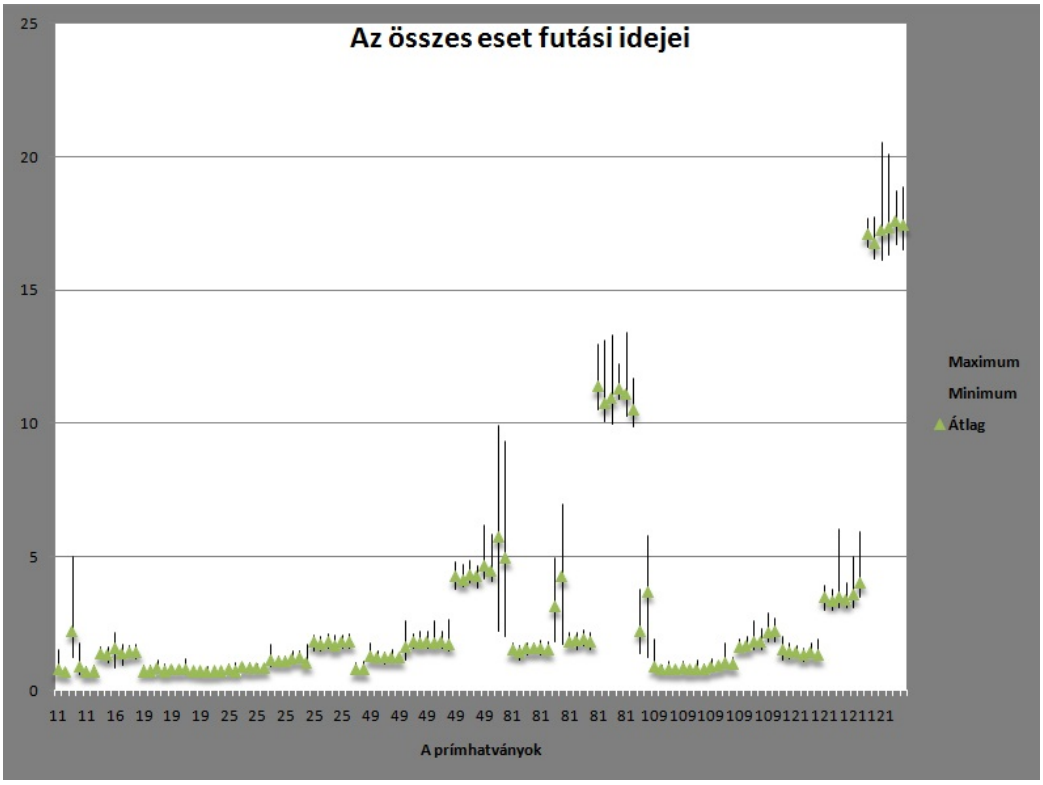


8. ábra. $q = 121$, $n \in \{15, 40, 120\}$

Kis kódhosszakra itt is megesett, hogy az átlagsebesség 2 mp körül mozgott. A $q - 1$ hosszú kódok viszont itt is kiemelkedőek voltak, az átlagos 17 mp-es futási idővel.

3.3.9. Összegzett ábra

Néhány prím esetén csak egy pár tesztet sikerült futtatni, de a következő összesített diagramban azoknak a futási idejét is ábrázoltam. Az alábbi ábrából látszik, hogy a prímhatványok növekedésével arányosan növekszik a futási idő is, illetve azt lehet sejteni, hogy egy adott prímhatványnál a hossz függvényében exponenciális a futási idő növekvése.



9. ábra. Áttekintő a $q \in \{11, 16, 19, 25, 49, 81, 109, 121\}$ értékekre

Hivatkozások

[1] S. Gao, A New Algorithm for Decoding Reed-Solomon Codes, in *Communications, Information and Network Security*, eds. V.Bhargava, H.V.Poor, V.Tarokh, and S.Yoon, 2002, 55–68, Kluwer.

[2] D. Joyner, R. Kreminski, J. Turisco, *Applied Abstract Algebra*, The Johns Hopkins University Press, Baltimore, 2004.

[3] Kiss Gy. - Szőnyi T., *Véges Geometriák*, Polygon Könyvtár, Szeged, 2001.

[4] `http://en.wikipedia.org/wiki/Reed-Solomon_error_correction` 2013.04.16.

[5] Korchmáros Gábor, *Véges geometriák és kódok (előadásjegyzet)*, Szegedi Tudományegyetem, 2012.

[6] W. A. Stein et al., *Sage Mathematics Software (Version 6.5.0)*, The Sage Development Team, 2015, `http://www.sagemath.org`.

Nyilatkozat

Alulírott Táborosi Andor Zsolt, kijelentem, hogy a dolgozatomat más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Bolyai Intézetének könyvtárában a kölcsönözhető könyvek között helyezik el, és az interneten is nyilvánosságra hozhatják.

Szeged, 2015. május 16. .....

aláírás

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani **Dr. Nagy Gábor Péternek**, amiért a dolgozatom témaválasztásában, kidolgozásában és a szükséges kutatómunkában a kezdetektől fogva segített irányításával, tanácsaival, illetve azért is szeretnék köszönetet mondani, hogy erre a projektre áldozta az idejét.