Véletlenség-extraktorok implementálása és tesztelése pszeudo-véletlen bemeneteken

Szász Kristóf Róbert

# 

[**Összefoglaló 5**](#_sya2u0kgwlcb)

[**Abstract 6**](#_nm6oife4qaa7)

[**Bevezetés 7**](#_s7c3yfjaicol)

[**Extraktorok 8**](#_29gxucf7rnu)

[Hatékonyság 8](#_1g5utyqca646)

[XOR 8](#_qdbm7qm66926)

[Shift XOR 8](#_9tahbxu3nybl)

[Von Neumann 9](#_opb7mssqlcls)

[Iterating Von Neumann 9](#_smco15b9cwaw)

[N bit Von Neumann 9](#_7i2xrob52klm)

[H function 9](#_tppkwqpayd2c)

[SHA-1 10](#_kcjbtln9vnnq)

[S-box 10](#_gsajaojuhfgt)

[**Tesztek 11**](#_fhusfic4c22k)

[Először: hipotézisvizsgálat és statisztikai tesztelés 11](#_v9ip2k3jie86)

[Frequency (Monobit) Test 11](#_qka3unter4j)

[Frequency Test within a Block 11](#_5by26dx1xa3x)

[Runs Test 11](#_fb4dwbdvwq7i)

[Test for the Longest Run of Ones in a Block 11](#_p529dcswbejn)

[Binary Matrix Rank Test 12](#_ytv9v1ax6gax)

[Discrete Fourier Transform (Spectral) Test 12](#_m4lyo8b3iu4r)

[Non-overlapping Template Matching Test 12](#_bsy1dxbpevdp)

[Overlapping Template Matching Test 12](#_tzdzfq6zgunb)

[Maurer’s “Universal Statistical” Test 12](#_5f9hyk3co56e)

[Linear Complexity Test 12](#_yr6htetheqmj)

[Serial Test 13](#_y774re8v6hqh)

[Approximate Entropy Test 13](#_v6obzj9ljcxf)

[Cumulative Sums (Cusum) Test 13](#_awm6rl459nr3)

[Random Excursions Test 13](#_p0h8h9co8xx1)

[Random Excursions Variant Test 13](#_v4tv6ijix5lf)

[**Gui 14**](#_qnib6e7nxa4b)

[Motiváció 14](#_gyavdy3vlfkl)

[Kezelőfelület 15](#_fmecu9caqkt7)

[Telepítés 18](#_lsk51fnzhrps)

[Felépítés és működés 21](#_r38rzen3ew7r)

[App 21](#_7j4kqd619bg4)

[FileManager 21](#_kdpnekykiq5z)

[Card 21](#_435rdhl2k852)

[NextCard 22](#_mvpygjp9ptxg)

[Execute 22](#_qq5iyzi7x4u5)

[Kártyák 22](#_qtexm6xd7niw)

[ExtractorCard 22](#_pqmnovd1g2i5)

[NISTCard 22](#_irw33m9murzb)

[Felhasznált technológia/külső könyvtárak 24](#_2koitf525uzd)

[**Tesztelés 25**](#_d4xu5b80xxy5)

[Álvéletlen sorozatok 25](#_kkdv1aqc09dm)

[Angol betűeloszlás 26](#_1l2gsec15gyt)

[XOR extraktor 26](#_ixuo2p8zd634)

[Shift XOR extraktor 27](#_jhst9rqknb2e)

[Von Neumann extraktor 28](#_qz3kp8z6mgmf)

[Iterating Von Neumann extraktor 29](#_z1ky60v5ze92)

[N bites Von Neumann 30](#_ofmz6cqeh0o0)

[H function 31](#_r9uam54gfwgl)

[Sha-1 hash 32](#_u2xchxnflyhp)

[S-box 33](#_4xt1dauojc8y)

[Angol felezés 34](#_aj4677bgf5ak)

[XOR extraktor 34](#_suuzg8j6ut4x)

[Shift XOR extraktor 35](#_v7glgnmt849)

[Von Neumann extraktor 36](#_f8zsrh7jtw16)

[Iterating Von Neumann extraktor 37](#_qut9q8bxfcpn)

[N bites Von Neumann 38](#_2ab2anpaao2h)

[H function 39](#_yd399u943yrg)

[Sha-1 hash 40](#_czrgrg5iorsd)

[S-box 41](#_rpbaeqlbi1hd)

[Angol hasonlítás 42](#_6o6iyqy4r86u)

[XOR extraktor 42](#_cwgsq0xmohng)

[Shift XOR extraktor 43](#_ra4vtv54u9fy)

[Von Neumann extraktor 44](#_bv94tdgqah6a)

[Iterating Von Neumann extraktor 45](#_l7gb2n8huu43)

[N bites Von Neumann extraktor 46](#_6em5ub4ydkq8)

[H function 47](#_f6bs3vpbx7j4)

[Sha-1 hash 48](#_uitswobz417)

[S-box 49](#_1z9mpogphlq2)

[Magyar betűeloszlás 50](#_oxhhpg6xthie)

[XOR extraktor 50](#_dk6sz5d8ryo3)

[Shift XOR extraktor 51](#_n2cw6wbgwhsu)

[Von Neumann extraktor 52](#_icxt415gbeis)

[Iterating Von Neumann extraktor 53](#_pqjj63i7be08)

[N bites Von Neumann extraktor 53](#_kqg8lasdjwj3)

[H function 54](#_qpb7ew4pk3h4)

[Sha-1 hash 55](#_2lh7oid3wf6g)

[S-box 56](#_q8rrq8520rd)

[Időjárás adatok 57](#_w7j1ex2nv9xd)

[XOR extraktor 57](#_b4jf69yecy81)

[Shift XOR extraktor 58](#_73sfnp7qbjly)

[Von Neumann extraktor 59](#_rhbhubjkglj)

[Iterating Von Neumann extraktor 60](#_ea9hte5bvgs6)

[N bites Von Neumann extraktor 60](#_7ymw88350hdo)

[H function 61](#_of3gi6abm595)

[Sha-1 hash 62](#_ugqxgwywrj6)

[S-box 63](#_2pytr3pzu91w)

[Extraktorok hatása 64](#_918jbggzqjzm)

[XOR extraktor 64](#_ce8j0qd2sek)

[Shift XOR extraktor 64](#_ay5cupo4rkue)

[Von Neumann extraktor 64](#_kqm5u4156bv6)

[Iterating Von Neumann extraktor 64](#_hg0yub1fuvyx)

[N bites Von Neumann extraktor 64](#_1fuahymqt54q)

[H function 64](#_aoxlpypwju2i)

[Sha-1 hash 64](#_mnl9b0dh8esd)

[S-box 65](#_2c1y2zp3fqbm)

# 

# Összefoglaló

A következő dolgozat véletlenség extraktorok implementálásáról, és teszteléséről fog szólni. Így a dolgozat első fejezetei ezeket az extraktorokat és teszteket hivatottak bemutatni. Az extraktorokhoz a dolgozat során saját implementációkat készítettem, míg a tesztelést a NIST Statistical Test Suite segítségével végeztem.

Az extraktorok és tesztek elmagyarázása után a következő fejezetben a projekthez fejlesztett alkalmazást mutatom be. Az alkalmazásban futtathatók az implementált extraktorok, és a bemutatott NIST tesztek. Az alkalmazásról szóló fejezetekben szót ejtek az alkalmazás mögötti motivációról, az alkalmazás használatáról, telepítéséről, valamint a belső működéséről.

A dolgozat utolsó nagyobb fejezetében az extraktorok tesztelésének eredményeit mutatom be. Ebben a fejezetben főleg a szemléletesebb eredményekre koncentrálok, más szóval bemutatom a legjobban, legrosszabbul, de legalább is érdekesen teljesítő extraktor forrás párokat. A dolgozat végén megpróbálok konklúziókat levonni az implementált extraktorok működésével kapcsolatban. Viszont észben tartandó, hogy nem lehetünk biztosak ezen záró konklúziók pontosságában, mivel az extraktorok futtatása során történő bitveszteségek, és az eredeti források nem túl nagy mérete miatt viszonylag kevés adat állt a rendelkezésemre.

# Abstract

In this thesis I will be discussing the implementation and testing of some randomness extractors. Because of this the first two chapters will aim to showcase these extractors and tests. The implementations for the extractors discussed were made by me, and the tests were conducted with the help of the NIST Statistical Test Suite.

In the following chapter I will discuss the application created for this project. In this application both the implemented extractors, and showcased NIST tests can be run. In this chapter I will explain the motivation behind the app, how one can use and install the app, and I will discuss how the app works.

In the last big chapter of the thesis I will focus on the results of testing the implemented extractors. I will show the most interesting results, in other words the surprisingly good, the hilariously bad, and the hard-to-explain. At the end I will attempt to draw some conclusions from the data collected. Keep in mind however, that we can not be certain of the accuracy of these conclusions, since due to the bit loss of the extractors, and the size of the original source files I did not have enough data to come up with anything conclusive.

# Bevezetés

Az önálló laboratórium és témalabor projektem álvéletlen számok előállítása és statisztikai elemzése volt. Az első labor során egy terminál alkalmazást készítettem, ami szöveget és időjárás adatokból tudott álvéletlen bitsorozatokat előállítani. A szöveges forrásaim magyar és angol nyelvű könyvek voltak, míg az időjárásadatokat az irodalomjegyzékben felsorolt weboldalról szereztem be. A források feldolgozásáról ebben a dolgozatban is szót ejtek majd, mert ezeket a bitforrásokat használtam kiindulásul a szakdolgozatban is.   
Ebben a dolgozatban az előző laborok munkájára építve véletlenség extraktorokkal fogok foglalkozni. Véletlen és álvéletlen számok tesztelése és felhasználása főleg titkosításban fontos, például kulcsok előállításában. Az extraktorok segítenek az álvéletlen forrásainkból kiszűrni periodikusságot korrelációkat, bitek egymástól függését, és más olyan statisztikai tulajdonságokat, amik az adott bitforrást potenciálisan jósolhatóvá teszik. Ennek az ára bitveszteség. Minden extraktornak más a várható bitvesztesége, és ez algoritmustól függően drasztikusan különböző lehet. Például az egyik felhasznált extraktor a Shift XOR. A shift xor extraktor vesztesége a bemenet méretétől független, viszont lineárisan függ az extraktorhoz megadott paramétertől. Ezzel ellentétben például a XOR extraktornak nem kell megadni paramétert, de várhatóan az extraktor kimenete feleannyi bit lesz, mint a bemenete volt. A legtöbb extraktor effektív használatához az eredeti forrásnak eleget kell tennie bizonyos előfeltételeknek. Egy gyakori ilyen előfeltétel a bitek egymástól való függetlensége. A dolgozatban az extraktorok tesztelése során ezeket az előfeltételeket figyelmen kívül hagyom, és csak az utolsó fejezetben a konklúziók levonásakor kerülnek majd újra elő. Ennek az az oka, hogy a dolgozat célja nem jó minőségű bitsorozatok előállítása, hanem a kiválasztott extraktorok implementálása, és a működésük bemutatása. Ehhez szükségem van példákra amiken a rossz működést tudom bemutatni.

A projekthez fejlesztett alkalmazás motivációja röviden a következő: Szükségem van egy alkalmazásra, amiben egyszerűen tudok a meglévő forrásaimra extraktorokat futtatni, és az extraktorok kimenetét hasonlóan egyszerűen tudom tesztelni. Az extraktorok nagy részéhez saját implementációt készítettem. Amelyik extraktorra ez nem igaz, annak a fejezetében jelzem, hogy hogyan valósítottam meg az extraktort és miért. Az extraktorok implementálásán kívül a projekt része volt egy grafikus felület létrehozása a teszteléshez használt NIST Statistical Test Suite-hoz (továbbiakban NIST). Erre azért volt szükség, hogy ugyanazon a felhasználói felületen tudjam a projek tesztelés fázisához szükséges extraktor és teszt futtatást végrehajtani. A NIST eredetileg egy terminálban futtatható program, és cél a NIST beépítése egy egyszerűbben használható felhasználói felületbe. A felhasználói felület viszonylag keveset vesz el a NIST tesztkészlet funkcionalitásából, viszont drasztikusan megkönnyíti a tesztek futtatását, főleg, ha egymás után több tesztet tervez futtatni a felhasználó. Az alkalmazás használatát, telepítését és működését a dolgozat során részletesebben taglalom.

A projekt utolsó fázisa az implementált extraktorok futtatása, és a keletkezett bitsorozatok tesztelése az alkalmazást használva. Ebben a fázisban fontos volt a megfelelő források kiválasztása. Az elsődleges szempont a méret volt a források kiválasztásánál. Mint fentebb említettem az extraktorok bitveszteség árán tudnak javítani a forrás statisztikai tulajdonságain. Viszont egy forrás teszteléséhez minél több bitre van szükség ahhoz, hogy az eredményből bármilyen érdemi konklúziót le lehessen vonni.

# Extraktorok

A következő fejezetben a projekt során implemenált extraktorokat tárgyalom. Extraktorok segítségével javítani tudjuk a bitforrásunk bizonyos statisztikai tulajdonságait. Ennek bitveszteség az ára.   
 Kétféle extraktort különböztetünk meg működés szempontjából. A determinikus extraktorok kimenete kizárólag a bemeneti bitektől függ. Azonos bemenetre mindig azonos kimenetet fognak produkálni. A másik féle extraktor a seed-elt extraktor, aminek a kimenete függ a bemeneti bitektől, és az algoritmus elvégzéséhez szükséges beállítható paraméterektől. Az implementált determinisztikus extraktorok a XOR, Von Neumann, H function, SHA-1, és S-box extraktorok. A seed-elt extraktorok a Shift XOR, ahol megadható a shiftelés mértéke (m), az Iterating Von Neumann, ahol megadható az iterációk száma, és az N bit Von Neumann, ahol megadható, hogy hány bites blokkokban dolgozza fel az algoritmus a bemeneti biteket.

## Hatékonyság

A következő részben gyakran fogom használni a hatékonyság fogalmát, szóval itt definiálom, hogy mit értek alatta. A hatékonyság egy egy és nulla közötti szám, ami az adott extraktor várható bitveszteségét fejezi ki. Például az XOR extraktor mindig két szomszédos bitből egyet ír az outputra. Feleakkora az output mérete, mint az inputé, tehát a XOR extraktor hatékonysága ½ . A legtöbb extraktornál a valós hatékonyság az inputtól, és az algoritmushoz megadott paraméterektől függően változik.

## XOR

A XOR extraktor veszi az inputon egymás mellett párban érkező biteket, és alkalmazza rájuk az XOR műveletet:

* 00 → 0
* 01 → 1
* 10 → 1
* 11 → 0

Ha páratlan számú bit érkezik az inputon, akkor az utolsó pár nélkül maradt bitet eldobjuk. Az extraktor feleannyi bitet ír az outputra, mint amennyi az inputon van, a hatékonysága ½.

## Shift XOR

Az extraktornak megadható m paraméter. Vesszük az inputot, és xor műveletet hajtunk végre az input, és az m-mel eltolt input között. Példa:

* A felső sor az input, a második sor az M-mel eltolt input, az alsó sor az output. M = 2

| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
|  |  | 1 | 0 | 0 | 1 | 0 | 1 |  |  |

A példából látható, hogy az elvesztett bitek száma mindig m kétszerese. Emiatt az inputméret növelésével konstans m mellett a hatékonyság 1 felé tart.

## Von Neumann

A Von Neumann extraktor, a XOR extraktorhoz hasonlóan párosával vizsgálja meg az input bitjeit, és a párokhoz vagy rendel output bitet, vagy eldobja őket. Ezt a következő szabály szerint teszi:

* 01 → 0
* 10 → 1
* Az extraktor a 00 és 11 bitpárokat eldobja.

¼ hatékonyságú, hiszen a bitpárjaink körülbelül felét eldobjuk, és a másik, eldobatlan fél minden két bitjéhez egy output bitet rendelünk.

## Iterating Von Neumann

Az Iterating Von Neumann extraktor több Von Neumann algoritmust futtat egymás után. Inputként megadható az iterációk maximális száma, vagy ha nem állítjuk be akkor addig iterál a program, amíg kifogy a bitekből. A Von Neumann extraktornál a 00 és 11 bitpárokat eldobtuk. Az Iterating Von Neumann ezekre az eldobott bitekre futtatja le újra a Von Neumann algoritmust. A második iteráció inputjába minden kidobott 00 után bekerül egy 0 bit, és minden kidobott 11 után egy 1 bit. Két oka van, hogy nem 00-ákat és 11-eket írunk az új inputba. Egyrészt a Von Neumann ezeket csak újra kidobná és végtelen ciklusba esne. Másrészt az első Von Neumann futtatásakor abból már információt nyertünk ki, hogy ezek egymás melletti azonos bitek, és nem szeretnénk ezt az információt az extraktor több pontján felhasználni. Az Iterating Von Neumann extraktor majdnem mindig hatékonyabb mint a Von Neumann.

## N bit Von Neumann

A von neumann általánosítása. A sima von neumann a 2 bit von neumann-nak felel meg. Az algoritmus szerint n bites blokkokban olvassuk a bemenetet. A lehetséges blokkokat csoportokba rendezzük aszerint, hogy mennyi 1 bitet tartalmaznak. Azok a blokktípusok, amik nem tartalmaznak 1 bitet, vagy csak 1 biteket tartalmaznak nem produkálnak kimenetet. A többi blokk kimenetét a következő módon döntjük el. Megfigyeljük minden blokk csoport (B) számosságát (s), és hozzá rendeljük a legnagyobb kettő hatványt () ami még kisebb, mint amennyi blokk tartozik a csoportba. A csoport minden blokkjának (b) egy különböző k bit hosszú bitsorozat lesz a kimenete. Mivel < s, így minden k hosszú bitsorozatot ki tudunk osztani egy B beli b blokktípusnak. Azok a blokktípusok, amelyeknek nem jutott k hosszú bitsorozat nem produkálnak kimenetet.   
  
Példa a 3 bites Von Neumann extraktor:  
  
A lehetséges 3 bites blokkok a következők:

| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| --- | --- | --- | --- | --- | --- | --- | --- |

A 000 és 111 nem produkálnak kimenetet, mert vagy nincs bennük 1-es bit, vagy csak 1-es bit van bennük.  
  
Az 1 darab 1-es bitet tartalmazó blokkok csoportja:

| 001 | 010 | 100 |
| --- | --- | --- |

Ezekhez a legnagyobb rendelhető 2 hatvány a , tehát a kimenetük 1 bites lesz. A kimenetek elosztása blokkon belül szabadon megválasztható, szóval legyen a következő:

| blokk | blokkhoz tartozó kimenet |
| --- | --- |
| 001 | 0 |
| 010 | 1 |
| 100 | Nem produkál kimenetet |

Az 100 blokknak nem jutott 1 bit hosszú kimenet.  
  
A 2 darab 1-es bitet tartalmazó blokkok csoportja:

| 011 | 101 | 110 |
| --- | --- | --- |

Ezekhez szintén a rendelhető, és a kimenetek hasonlóan alakulnak.

| blokk | blokkhoz tartozó kimenet |
| --- | --- |
| 011 | 0 |
| 101 | 1 |
| 110 | Nem produkál kimenetet |

Ezzel megéllapítottuk az összes blokkhoz tartozó kimenetet.

Az algoritmusról részletesebb magyarázat az irodalomjegyzékben található. Ahogy n tart a végtelenbe a hatékonyság az ½ -hez tart.

## H function

A h-function extraktor két byteot vesz inputnak, és egy byte az output. Az output úgy áll elő, hogy az egyik byteot shifteljük egy bittel, és az így kapott bytera és a másik input bytera xor műveletet alkalmazunk. Példa:

| Az első byte: | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A második byte: | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| A második byte shifteltje: | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| Az output: | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

## SHA-1

Ez az extraktor a SHA-1 hashet alkalmazza. Egyszerűen 20 byte-onként alkalmazzuk a SHA-1 hashet az inputrta. Mivel a SHA-1 outputja is 20 byte, így a veszteség csak az input végén lévő maradék bitekből jön, amik már nem voltak elég nagyok a 20 byte-os inputhoz. A hatékonyság az inputméret növekedésével 1 felé tart.

## S-box

Az s-box vagy substitution box egy szimmetrikus kulcsú titkosításra használt algoritmus. Egy nyolc bites inputhoz egy nyolc bites outputot rendel. Az átalakítás egy előre kiszámolt substitution táblázat alapján történik, amit a projektben felhasznált implementáció már tartalmaz. Ez az implementáció megtalálható az irodalomjegyzékben.

# Tesztek

Az előző fejezetben tárgyalt extraktorok tesztelésére a NIST Statistical Test Suite-ot használom. Link a teszt készlethez megtalálható az irodalomjegyzékben. A tesztek célja, hogy álvéletlen bitforrásoknak tudjuk kiszűrni bizonyos statisztikai tulajdonságait, mint például periodicitás, a bitek egymástól függése, vagy ismétlődő mintázatok. Ilyen és ehhez hasonló tesztekkel nem bitsorozatokat, hanem bitforrásokat lehet értékelni. Ha egy bitforrás átmegy a teszteken, az nem azt jelenti, hogy ez egy véletlen bitforrás, hanem azt, hogy bizonyos hibahatáron belül azt tudjuk állítani, hogy valószínűleg véletlen.

## Először: hipotézisvizsgálat és statisztikai tesztelés

* Az összes teszt egy hipotézisvizsgálat, ahol a feltett nullhipotézis az, hogy a vizsgált bitsorozat nem véletlen bitsorozat, és így a vizsgált bitgeneráló módszer nem véletlen biteket állít elő (rng). A nullhipotézis ellentettje az alternatív hipotézis, ami ezeknél a teszteknél az, hogy a bitsorozat véletlen.
* Minden tesztnek más próbafüggvénye van, ez a tesztben leírt számolás.
* A vizsgált bitsorozatból, és a teszt próbafüggvényéből p-értéket számolunk, ami a nullhipotézisre várható eredménytől való eltérést kvantifikálja.
* Minden tesz definiál egy szignifikancia szintet. Ha a p-érték ez alá esik, akkor a nullhipotézist fogadjuk el. A szignifikancia szint általában 0.01, 0.05.
* Ezeknél a teszteknél, ha a szignifikancia szint alá esik a kiszámolt p-érték, akkor a bitostozat valószínűleg nem véletlen.

## Frequency (Monobit) Test

* A teljes bitsorozaton ellenőrzi, hogy a 0-ák és 1-ek relatív gyakorisága körülbelül megegyezzen.

## Frequency Test within a Block

* Kisebb fix méretű blokkonként ellenőrzi, hogy a 0-ák és 1-ek száma körülbelül megegyezzen.

## Runs Test

* Run: egymást követő megegyező bitek
* A teszt ellenőrzi, hogy a runok száma mennyi az elvárthoz képest.
* Kiszűri ha a bitsorozat túl lassan, vagy túl gyorsan oszcillál.

## Test for the Longest Run of Ones in a Block

* A teszt ellenőrzi, hogy a leghosszabb run mekkora az elvárthoz képest.
* Elég csak az egyeseket nézni, mert a nullákra ugyanaz jönne ki.

## Binary Matrix Rank Test

* Lineáris függést szűr a bitsorozat fix hosszú alsorozatai között úgy, hogy a bitsorozatból készített mátrixokkal dolgozik.

## Discrete Fourier Transform (Spectral) Test

* Azt ellenőrzi, hogy a kimagaslóan nagy amplitúdójú frekvenciakomponensek száma ne legyen túl nagy.
* Mintákat és periodikusságot tud kiszűrni.

## Non-overlapping Template Matching Test

* Azt vizsgálja, hogy a bitsorozat hány helyen egyezik meg előre definiált, nem periodikus mintákkal.
* Ugyanaz mint az Overlapping Template Matching Test, csak ha talál egy mintát azt átugorja, vagyis a keresést a talált minta utáni bitnél folytatja.

## Overlapping Template Matching Test

* Azt vizsgálja, hogy a bitsorozat hány helyen egyezik meg előre definiált, nem periodikus mintákkal.
* Ugyanaz mint a Non-Overlapping Template Matching Test, csak ha talál egy mintát azt nem ugorja át, vagyis a keresést a talált mintán belüli következő bitnél folytatja.

## Maurer’s “Universal Statistical” Test

* Azt vizsgálja, hogy a bitsorozat tömöríthető lenne-e kevés információvesztéssel (ha igen akkor nem véletlenszerű).

## Linear Complexity Test

* Megkeresi azt a legkisebb linear feedback shift registert, ami az előállított bitsorozathoz hasonló minőségű bitsorozatot tudna előállítani. Hosszabb LFSR komplexebb sorozatot jelent. Ha túl rövid a LFSR akkor nem véletlenszerűnek ítéljük a bitsorozatot.

## Serial Test

* A lehetséges m hosszú minták előfordulási gyakoriságát veti össze az elvárttal. Véletlen sorozatoknál minden m bit hosszú minta előfordulási valószínűsége megegyezik.

## Approximate Entropy Test

* Hasonló a Serial Test-hez, de nem csak az m hosszú bitsorozatok előfordulását figyeli, hanem az m+1 hosszú bitsorozatokét is.

## Cumulative Sums (Cusum) Test

* Összegzést végez lépésenként (ha a bit 0 kivon 1-et, ha 1 hozzáad), és a legnagyobb részösszeget vizsgálja. Ha véletlen a bitsorozat ennek egy a bitsorozat hosszától függő határérték alatt kell lennie.

## Random Excursions Test

* Az előző tesztben használt módszerrel részösszegeket (állapotokat) képez. A kapott számsort ciklusokra bontja, és azt vizsgálja, hogy hány olyan ciklus van ahol “a” állapot “b”-szer jelent meg. Ezt megteszi minden lehetséges a-ra és b-re, és az eredményt összeveti az elvárttal.

## Random Excursions Variant Test

* Az előző tesztben használt módszerrel részösszegeket képez, majd megnézi, hogy a részösszegek előfordulásainak számai eltérnek-e az elvárttól.

## 

# Gui

## Motiváció

A témalabor és önálló labor alatt pszeudo-véletlen bitsorozatok előállításával, és statisztikai elemzésével foglalkoztam. Az előlállításhoz saját alkalmazást fejlesztettem, a statisztikai elemzéshez pedig a NIST statisztikai teszkészletet (későbbiekben NIST) használtam fel. A NIST tesztek ebben a feladatban is fontosak, ezeket fenetebb taglalom. A szakdolgozat fő témája a véletlen extraktorok implementálása és tesztelése a NIST tesztekkel. Ennek a folyamatnak a könnyítése érdekében az alkalmazást egy GUI-n keresztül lehet kezelni. A NIST eredetileg egy terminál alkalmazás. A felhasználói felület lehetőséget ad a NIST egyszerűbb kezelésére, fájltallózásra. Ezen kívül az extraktorokból kijövő fájlokat azonnal át lehet vezetni a NIST teszteken. Végül pedig, a NIST outputja egy fájl, amit minden futtatáskor felülír. Ezen is javít a GUI és minden futtatáshoz külön kiértékelő fájlt ment el, ami a nevét az eredeti féjl nevéből kapja, és a célmappa beállítható, ellentétben az eredeti NIST-tel..  
 A grafikus felületet java swinggel készítettem el. Java swinggel a felhasználói felület kinézetét egyszerűen lehet konfigurálni. A form nézeten felvehetők például gombok, text fieldek, framek, és más komponensek. Állítható a komponensek elrendezése, nevei, és mérete. Ez sok kódolási munkát levesz a fejlesztő válláról. Az írandó kód így csak az alkalmazás működésére vonatkozik majd. A fő ok ami miatt a java swing mellett döntöttem az volt, hogy az egyetemen már találkoztam windows formssal és swinggel, és a kettő közül a swinget találtam egyszerűbben használhatónak.

## Kezelőfelület

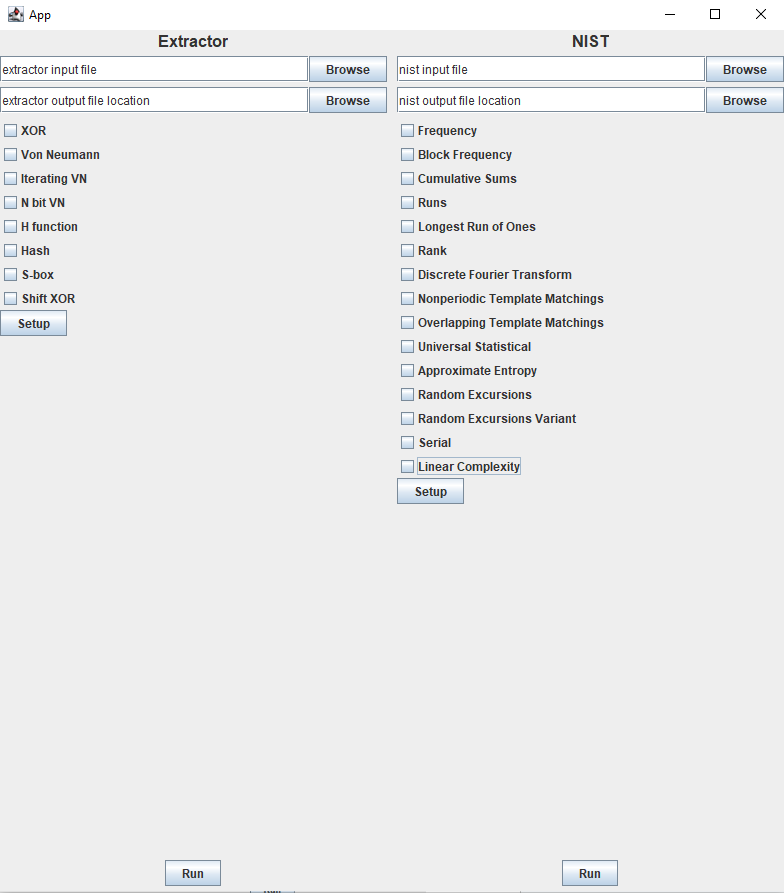
Az (1) képen a kezelőfelület látható. Középen el van választva két oldalra. A bal oldalon a projekt során implementált extraktorok futtathatók, míg a bal oldalon a NIST teszt készlet tesztjei. A két oldal kezelése teljesen amalóg.

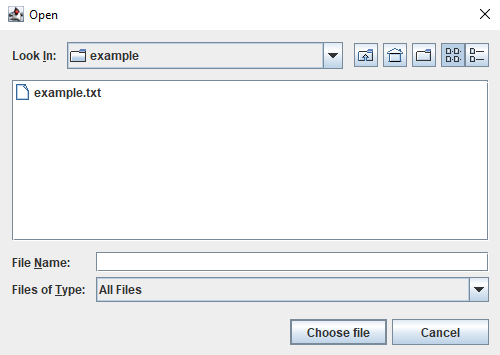
A felhasználói felület tetején látható két szöveges bemeneti mező, és mellettük egy-egy “browse” gomb.A bemeneti fájl és kimeneti mappa elérési útvonala megadható a szöveges bemeneti mezőben, vagy tallózható a “browse” gomb megnyomására megjelenő fájlböngésző ablakban. A böngésző ablak a (2) képen tekintheő meg. Az extraktorok .txt kiterjesztésű fájlokat fogadnak el bemenetként. Ennek az az oka, hogy az előző félévekben főleg ilyen formában állítottam elő bitsorozatokat. Az extraktorok kimenete néhány extraktor kivételével egy .txt és egy .bin kiterjesztésű fájl. Mindkét fájl ugyanazokat a kimeneti biteket tartalmazza, a .txt fájl karakterek formájában, míg a .bin fájl valódi bitek formájában. Minden extraktor állít elő .bin kiterjesztésű kimenetet, de nem mindegyik állít elő .txt kiterjesztésűt. A kimeneti fájlokat az alkalmazás a megadott kimeneti mappába menti. A kimeneti fájlok nevei a bemeneti fájl nevéből, és az alkalmazott extraktorból állnak össze. Például, ha a proba.txt a bemeneti fájlunk, és lefuttatjuk rá a XOR extraktort, akkor a két kimeneti fájl a proba\_xor.txt, és a proba\_xor.bin.

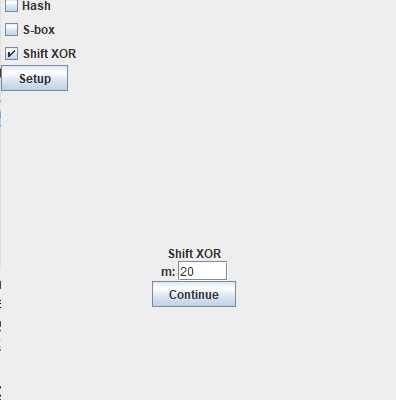
A felhasználói felületen a bemeneti mezők alatt checkboxok láthatók. Ezek segítségével lehet kiválasztani, hogy melyik extraktorokat, vagy teszteket futtassuk. Több extraktor kiválasztása esetén minden extraktor külön kimeneti fájlt generál a megadott kimeneti mappába. Például, ha a proba.txt bemeneti fájlra úgy futtatjuk az alkalmazást, hogy a xor extraktor és a hash extraktor van bepipálva, akkor a kimeneti mappában a következő fájlok keletkeznek: proba\_xor.txt, proba\_xor.bin, proba\_hash.bin. A hash extraktor egyike azoknak a fentebb említett extraktorokna, amik nem állítanak elő .txt kiterjesztésű kimenetet. A NIST tesztek nem generálnak külön kimenetet minden bepipált tesztre. Az eredeti NIST alkalmazásban is van arra opció, hogy ugyanarra a fájlra több, vagy akár az összes tesztet lefuttassuk. Az alkalmazás által elmentett kimenet a nist által generált finalAnalysisReport.txt fájl. Ezt másolja az alkalmazás a kimeneti mappába, és átnevezi a bemeneti fájl nevét felhasználva.

Ha a checkboxok sehítségével kiválasztottuk a kívánt extraktorokat, vagy teszteket, akkor a “setup” gomb megnyomásával léphetünk tovább. A setup hatására az alkalmazás sorban megjeleníti a kiválasztott tesztekhez, vagy extraktorokhoz tartozó kártyákat. Egy ilyen kártyát ábrázol a (3) ábra. A kártyákon adhatóak meg paraméterek a futtatandó extraktorhoz, vagy teszthez ha az szükséges. Ha nincs paraméter, akkor a kártya csak az extraktor, vagy tesz nevét, és egy “continue” gombot tartalmaz. Ha ezen a ponton észrevesszük, hogy nem a kívánt extraktorokat, vagy teszteket választottuk ki, akkor a checkboxok megfelelő pipálásával, és a “setup” gomb újra megnyomásával ez a lépés újból kezdhető. A NIST tesztek esetében az egyes kiválasztott tesztekhez tartozó kártyák előtt megjelenik egy kártya, amin megadható, hogy milyen hosszúak a kiértékelendő bitstreamek, és mennyi van belőlük.

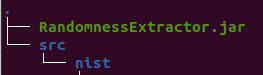
Ha elfogytak a kártyák, és elégedettek vagyunk a megadott beállításokkal, akkor az extraktorok, vagy a tesztek a legalsó “run” gombbal indíthatók. Ha a tesztek lefutottak a “run” gomb felett megjelenik üzenet, ami értesíti a felhasználót a sikeres lefutásról.



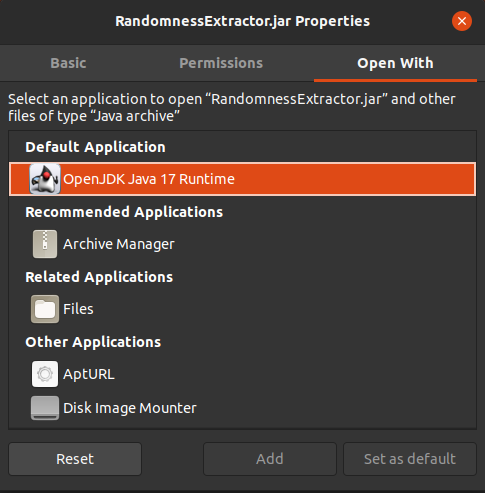


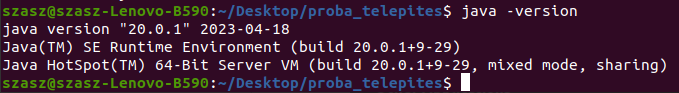
  
.

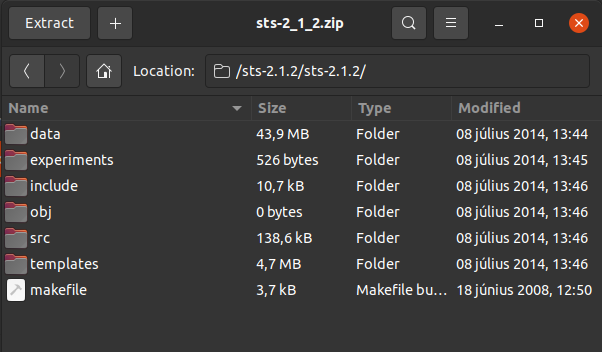
## Telepítés

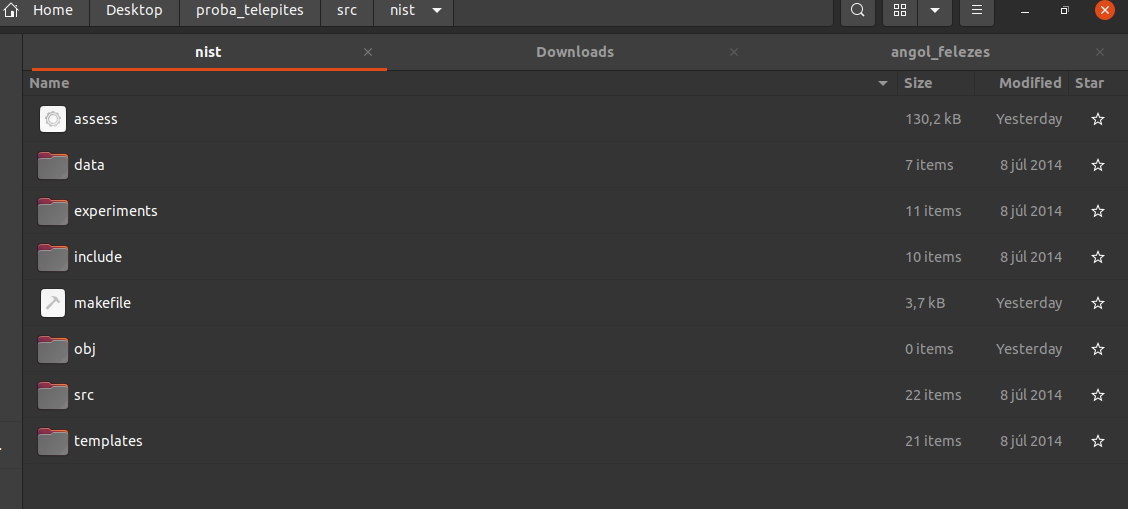
Az alkalmazás egy zip mappája fel van töltve a git repositoryba. Kicsomagolás után a mappaszerkezet a következő:  


A jar fájlt futtathatóvá kell tenni a következő paraccsal:  
chmod 777 RandomnessExtractor.jar

Ez után jobb klikk > properties:  
  
Az open with fülön ellenőrizni kell, hogy a default application Java runtime legyen.

A telepített java verziónak minimum java 19-nek kell lennie:  


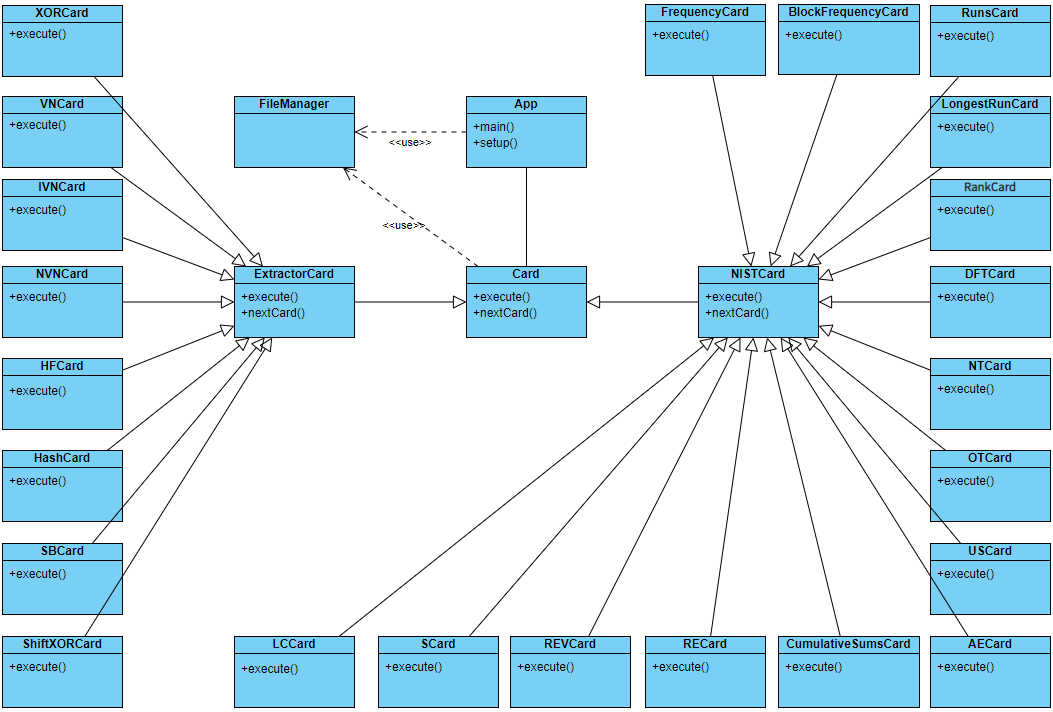
A következő lépés, a nist telepítése a nist mappába. A nist egy tömörített mappába csomagolva tölthető le. Ebben a tömörített mappában a nist fájljai két mappa mélyen vannak:  
  
A nist telepítési útmutatója megtalálható a nist dokumentációjában. A RandomnessExtractor telepítésénél figyelni kell arra, hogy a nist fő fájljai azonnal a nist mappában legyenek. Ha a nist mappában az sts-<verzió> fájlok vannak akkor nem fog működni az alkalmazás.

Telepítés után a következő a nist mappa tartalma:  


Az utolsó lépés, hogy a nist mappában létre kell hozni a, és futtathatóvá kell tenni a run\_nist.sh nevű fájlt. Ezt a következő parancsokkal lehet elérni:  
touch run\_nist.sh  
Chmod 777 run\_nist.sh

Ezek után az alkalmazás a RandomnessExtractor.jar dupla kattintásával, vagy a következő parancs kiadásával indítható:  
java -jar RandomnessExtractor.jar

## Felépítés és működés



Az alkalmazásnak fent látható az egyszerűsített osztálydiagramja. Tartalmaz minden osztályt, de nem taralmaz minden függvényt és változót, csak azokat, amik a program működésének a megértéséhez szükségesek.

### App

Az appban áll össze a grafikus felület nagy része. Ez az osztály tartalmazza a felhasználói felület komponenseit, azokat a függvényeket, amik megfelelően rendezik ezeket, a komponenseket, valamint itt találhatóak a felületen lévő gombok actionlistenerjei.

### FileManager

Ez az osztály felel a fájlműveletekért mint új fájlok létrehozása, fájlok tartalmának törlése, fájlok tartalmának másolása, valamint konvertálás bináris, és nullákat, és egyeket tartalmazó ASCII fájlok között. Az app és a card osztály is tartalmaz egy saját FileManagert.

### Card

Ez az összes extraktorért és NIST tesztért felelős card osztálynak a közös őse. A card is tartalmaz felhasználói felület elemeket, ezek a “SETUP” gomb megnyomása után megjelenő beállítás kártyák. A kártya tartalmazza az extraktor vagy NIST teszt nevét, ha kell megadható paramétereket, és a “CONTINUE” gombot. Az app tartalmaz minden levél kártya osztályból egyet. A két osztálydiagramon megjelenő függvény az execute() és a nextCard().

#### NextCard

Ez a függvény hívódik meg a “CONTINUE” gomb megnyomásakor. Ebben az osztályban a függvény absztrakt, a NISTCard és ExtractroCard implementálja.

#### Execute

Ez a függvény hajtja végre a kártya fő feladatát. Minden levél kártyának saját implementációja van, viszont az extraktorok és NIST tesztek esetén teljesen más a működési elv, szóval ezt később részletezem.

### Kártyák

A kártyák execute függvénye valósítja meg a működést. Az extraktor és NIST kártyák máshogyan érik ezt el.

### ExtractorCard

A Cardból származik le, és az összes extraktor közös őse. A nextcard függvénye megjeleníti a megfelelő extraktor kártyáját.  
  
Az extraktorok esetében a kártyák execute függvényében implementálva van az extraktor. Az implementált működést fentebb az extraktorok fejezetben tárgyaltam. Ez alól kivétel az s-box, amit hagyományosan nem extraktornak használnak, hanem Advanced Encryption Standard (AES) titkosításra. Ezt a már fentebb említett git repository felhasználásával implementáltam. Majdnem az összes extraktor kártya kettő kimeneti fájlt generál. Egy .txt kiterjesztésű fájlt, ami 0 és 1 karaktereket tartalmaz, és egy .bin kiterjesztésű fájlt, amiben a fájl bitjei a kimenet. A hash és s-box extraktorok nem generálnak .txt fájlt, mert az implementáció nem saját. Az s-box a fentebb említett git repository segítségével van megvalósítva, a hash pedig a java beépített SHA-1 függvényét használja, aminek a kimenete bináris. Minden extraktor bemenete .txt kiterjesztésű fájl. Ennek a fő oka az, hogy a témalabor és önálló labor alatt előállított álvéletlen sorozataim .txt fájlokban vannak.

### NISTCard

Az ExtractorCard-hoz hasonló, csak a nist kártyákat jeleníti meg a nextcard függvénye.

A NIST kártyák a NIST Statistical Test Suite-ot használják. Az alkalmazás a checkboxok segítségével kiválasztott NIST tesztek, és a NIST kártyákon megadott adatok alapján összerak egy Shell scriptet, majd a “RUN” gomb nyomására futtatja. Az alkalmazáson keresztül futtatott NIST mindig fájl imputtal dolgozik. Az input a GUI tetején megadott fájl. Ezen kívül a NIST-ben ki lehet választani, hogy az inputot bináris vagy .txt fájlként értelmezze a NIST. Az alkalmazáson kereszül mindig binárisként értelmez a NIST, mert minden extraktoromnak van bináris kimenete. Ezen a két beégetett beállításon kívül minden ami konfigurálható az eredeti NIST-ben konfigurálható a GUI-n keresztül is.

# Tesztelés

## Álvéletlen sorozatok

A témalabor, és önálló labor alatt álvéletlen sorozatokat állítottam elő magyar nyelvű szövegből, angol nyelvű szövegből, és időjárásadatokból. A szakdolgozat extraktorainak tesztelésére ezeket az álvéletlen sorozatokat, és a hozzájuk tartozó NIST teszteredményeket használtam fel. A következő fejezetekben röviden elmagyarázom az álvéletlen forrást, majd bemutatom a forráson futtatott extraktorok teszteredményeit. A teszteredményeket táblázatos formában ábrázolom. A táblázat minden sora egy tesztet reprezentál, és a két oszlop az eredeti és az extraktált bitsorozat eredményeit tartalmazza százalékos formában (a bitstreameknek hány százaléka ment át a teszten). Ez után egy szöveges magyarázatot is írok a teszteredményekről. A táblázatban csak azokat a teszteket jelenítem meg amiken az eredeti és az extraktált bitsorozat jelentősen különböző eredményt produkált.

### 

### Angol betűeloszlás

Az előző félévekben adatot gyűjtöttem arról, hogy az angol, és magyar nyelvben milyen gyakran fordulnak elő az egyes betűk. Az adatok alapján a betűket két csoportra osztottam. Mindkét csoportban a betűk előfordulási valószínűségeinek összege ½. Ez után a biteket állítottam elő a szövegből úgy, hogy minden betűhöz egy bitet rendeltem aszerint, hogy melyik csoportba tartoznak.

### Angol felezés

Hasonlóan az előző bitforráshoz, ez is angol nyelvű szövegből lett előállítva. Az angol ABC betűi itt nem előfordulás szerint vannak szétválasztva, hanem az alapján, hogy az ABC első vagy második felében vannak.

### Angol hasonlítás

A hasonlítás módszer két szomszédos betűből állít elő egy bitet. Ha a pár első eleme hamarabb fordul elő az ABC-ben mint a második, akkor az output bit 0, ellenkező esetben 1. Ha a két betű megegyezik, akkor nem rendelünk hozzájuk bitet.

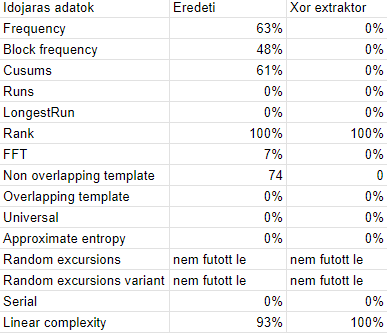
### Magyar betűeloszlás

A betűeloszlás módszer már egy előző fejezetben el lett magyarázva. A forrás abban különbözik, hogy magyar nyelvű szövegből lett előállítva.

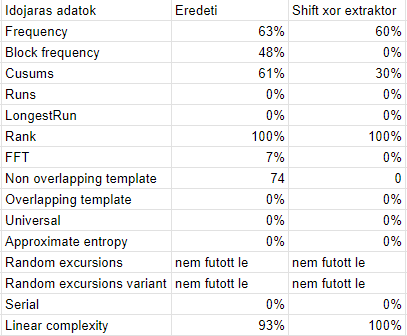
### Időjárás adatok

Az előző félévek laborjain állítottam elő biteket időjárásadatokból. Ez a forrás úgy állt elő, hogy a napi átlaghőmérsékleteket az adott hónap átlaghőmérsékletéhez hasonlítottam. Ha a napi átlghőmérséklet magasabb volt mint a havi, akkor a hozzárendelt bit 0, ellenkező esetben 1. Ha megegyeznek, akkor nem rendelünk bitet a párhoz. A szakdolgozatban implementált extraktorok teszteléséhez csak ezt az időjárás alapú forrást használtam fel, mert ezek a források általában kevés bitet produkáltak.

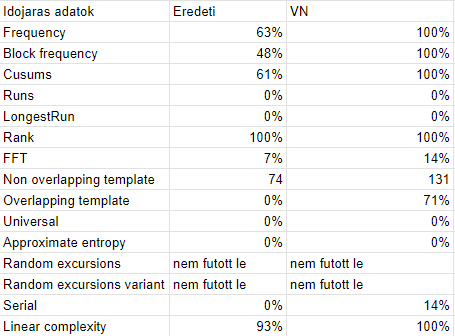
#### XOR extraktor



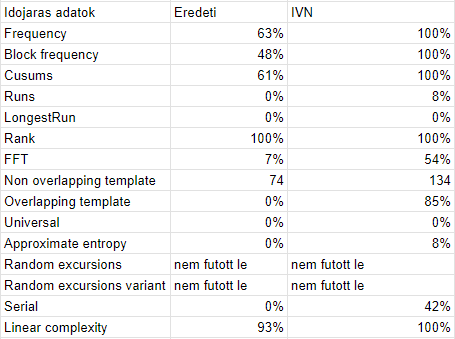
#### Shift XOR extraktor



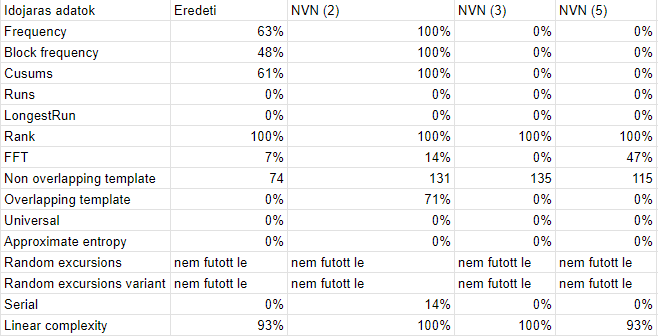
#### Von Neumann extraktor



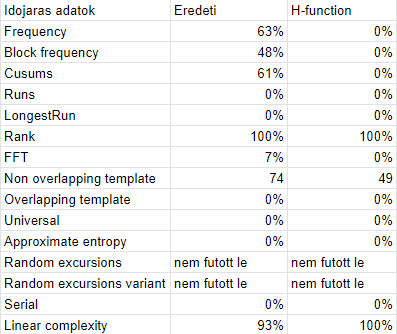
#### Iterating Von Neumann extraktor



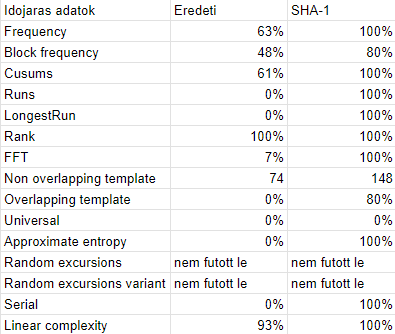
#### N bites Von Neumann extraktor



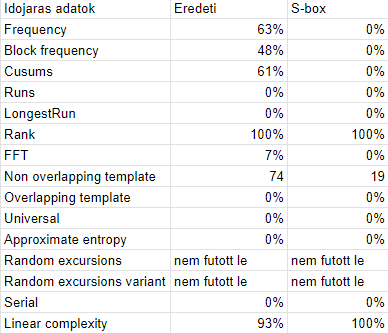
#### H function



#### Sha-1 hash



#### S-box



## Extraktorok hatása

### XOR extraktor

Az esetek többségében rontott a teszteredményeken. Ez alól két kivétel a non periodic template matching teszt, ahol a xor extraktor általában javított a teszt eredményén, és az angol felezés forrás, ahol a legrosszabb forrásból a xor extraktor drasztikusan javított a tesztek eredményein. A xor extraktor működésének előfeltétele, hogy a forrás bitjei függetlenek legyenek egymástól. Ez a nyelvi forrásoknál nyilvánvalóan nem igaz, az időjárásadatokon futtatott extraktorok, pedig általában olyan kevés bitet állítottak elő, hogy nem lehet valódi konklúziót levonni az eredményekből.

### Shift XOR extraktor

Általában javított a frequency, cumulative sums, és FFT teszteken elért eredményeken, valamint a random excursions, és random excursions variant tesztek szinte minden esetben sikeresek voltak. Ezek alapján arra lehet következtetni, hogy közeli bitek egymástól függését, és periodikusságot lehet vele kiszűrni.

### Von Neumann extraktor

Általában a frequency, block frequency és cumulative sums tesztek eredményein javított. Következtetésként leszűrhető, hogy rövid, és hosszú távon kiegyenlíti az 1 és 0 bitek számát.

### Iterating Von Neumann extraktor

Hasonlóan teljesített a sima von neumann extraktorhoz, csak nagyobb hatékonysággal dolgozta fel a forrás bitjeit.

### N bites Von Neumann extraktor

A 2 bites verzió hasonló eredményeket produkált a sima von neumannhoz (hiszen a két extraktor gyakorlatilag ugyanaz), azonban mind tesztekre való hatásban, mind hatékonyságban inkonzisztesen működött minden más n-re. A hatékonyság általában javult n növelésével, ami egyezik az elmélettel, de túl kevés adatom van ahhoz, hogy konkluzívan értelmezni tudjam az extraktor által produkált eredményeket.

### H function

Általában a longest run és a template matching tesztek eredményein javított. Hatékonynak tűnik az ismétlődő minták “elrontásában”.

### Sha-1 hash

Szinte minden teszten átment a hashelt forrás.

### S-box

Minden forrásra nagyon rossz eredményeket produkált. Arra gyanakszom, hogy az implementációban ejthettem valami hibát.

# Irodalomjegyzék

Az időjárási adatok forrása:   
<https://www.weather.gov/wrh/Climate?wfo=iln>

NIST Statistical Test Suite:   
<https://csrc.nist.gov/Projects/Random-Bit-Generation/Documentation-and-Software>   
Felhasznált S-box extraktor implementáció:  
<https://github.com/manuchandel/Advanced-Encryption-Standard>  
A fejlesztett alkalmazás git repositoryja:  
<https://github.com/szaszkristofrobert/RandomnessExtractor>  
N bites Von Neumann extraktor magyarázata:  
<https://projecteuclid.org/journals/annals-of-statistics/volume-20/issue-1/Iterating-Von-Neumanns-Procedure-for-Extracting-Random-Bits/10.1214/aos/1176348543.full>