Implementácia Diehard testov pre testovanie generátorov pseudonáhodných čísel

Bc. Andrej Naď

Diplomová práca
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Andrej Naď**
Osobní číslo:	**A16294**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Bezpečnostní technologie, systémy a management**
Forma studia:	**kombinovaná**
Téma práce:	**Implementace Diehard testů pro testování generátorů pseudonáhodných čísel**
Téma anglicky:	**The Implementation of Diehard Tests for the Testing of Pseudo-random Number Generators**

Zásady pro vypracování:

1. **Popiště teorii problematiky tvorby generátorů čísel.**
2. **Seznamte se s nejznámějšími zástupci generátorů čísel (random.org, Mersenne-Twister, vestavěné generátory programovacích jazyků, apod.)**
3. **Rozeberte a porovnejte možnosti testování generátorů čísel, včetně Diehard testů.**
4. **Implementujte sadu Diehard testů ve zvoleném programovacím jazyce.**
5. **Otestujte implementaci na reálně používaných generátorech čísel.**
6. **Vhodně vyhodnoťte získané výsledky.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **KNUTH, Donald Ervin. Umění programování. Vyd. 1. Brno: Computer Press, 2010, 763 s. ISBN 978-80-251-2898-5.**
2. **SUMMERFIELD, Mark. Python 3: výukový kurz. Brno: Computer Press, 2010. ISBN 978-80-251-2737-7.**
3. **PRESS, William H. FORTRAN numerical recipes. 2nd ed. New York: Cambridge University Press, 1999. ISBN 0-521-43064-x.**
4. **GENTLE, James E. Random number generation and Monte Carlo methods. 2nd ed. New York: Springer, c2003. ISBN 978-0387001784.**
5. **MATSUMOTO, Makoto a Takuji NISHIMURA. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation[online]. 8(1), 3-30 [cit. 2017-11-24]. DOI: 10.1145/272991.272995. ISSN 10493301. Dostupné z: http://portal.acm.org/citation.cfm?doid=272991.272995**
6. **ZACEK, Petr, Roman JASEK, Lukas KRALIK, David MALANIK a Petra HOLBIKOVA. Analysis of the Chaotic Pseudo-Random Generator of the PM-DC-LM Mode Based on the Position of the Returned Numbers. LISS 2017. Kyoto, Japonsko: IEEE Xplore, 2017. ISBN 978-1-5386-1047-3**

Vedoucí diplomové práce: **Ing. Petr Žáček**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **8. prosince 2017**

Termín odevzdání diplomové práce: **28. května 2018**

Ve Zlíně dne 8. prosince 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

L.S.

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

doc. RNDr. Vojtěch Křesálek, CSc.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis diplomanta

ABSTRAKT

Táto práca sa zaoberá problematikou testovania generátorov náhodných čísel, popisuje pätnásť štandardizovaných testov spolu s príkladmi testovania, podmienky pre maximálne spoľahlivý výsledok a vyhodnotenie výsledkov týchto testov. Výsledkom práce je program s grafickým rozhraním, ktorý môže používať aj technicky menej zdatný užívateľ. Program dokáže otestovať ľubovoľnú sekvenciu bitov a vrátiť výsledok pre túto sekvenciu. Práca ďalej popisuje ako získať náhodné čísla z najpoužívanejších služieb a programov pre generovanie náhodných a pseudonáhodných čísel. Výsledkom testovania týchto generátorov sú tabuľky s hodnotami a slovné vyhodnotenie výsledkov dosiahnutých pri testovaní týchto generátorov.

Kľúčová slová: Náhodné číslo, generátor náhodných čísel, generátor pseudonáhodných čísel, testovanie

ABSTRACT

This work deals with testing techniques of random number generators, it describes fifteen standardized tests with basic examples of testing and conditions, when result from testing random sequences will be reliable. Work provides a computer program with graphic user interface, which can be used by all users. A Program can test arbitrary sequence of random bits and return results of testing. Work also describes how to get random or pseudorandom numbers from various generators and services. Result of testing these generators and services are tables with values and verbal evaluation of results obtained while testing these generators.

Keywords: Random number, random number generator, pseudorandom number generator, testing

Týmto by som chcel poďakovať školiteľovi mojej diplomovej práce Ing. Petrovi Žáčkovi za odborné vedenie, metodickú pomoc a cenné rady, ktoré mi poskytol pri jej vypracovávaní.

OBSAH

ÚVOD

Potreba získať náhodné čísla, ktoré sa nedajú vypočítať alebo uhádnuť je kľúčová pre zabezpečenie spoľahlivo šifrovanej komunikácie. Pomocou náhodných čísel sa zabezpečuje napríklad komunikácia cez protokol HTTPS a certifikáty SSL, taktiež bezdrôtové spojenia zabezpečené napríklad pomocou WPA2 WiFi sietí, ďalej sa náhodné čísla používajú pri SSH spojení, posielaní šifrovaných emailov, ale aj pri obyčajnom Skype video hovore. Bolo by nezodpovedné tieto náhodné čísla generovať tak, že potenciálny útočník môže tieto náhodné čísla vypočítať alebo uhádnuť, a získať tak možnosť rozšifrovať komunikáciu.

Základné overenie dokáže každý – ak generátor nonstop generuje iba jedno rovnaké číslo, alebo je hneď viditeľný jednoduchý vzorec, pomocou ktorého sa dá odhadnúť ďalšie číslo, nebudú takto vygenerované čísla bezpečné. Takéto overenie je však absolútne nedostačujúce, a preto sa postupne vyvinulo niekoľko setov testov, ktoré náhodnosť zisťujú. Medzi tieto sety testov patrí aj set s názvom Diehard z roku 1995, ktorý však už nie je oficiálne podporovaný ani udržovaný. Set vylepšených Diehard testov momentálne udržiava americká štandardizačná spoločnosť NIST (National Institut of Standards and Technology).

Tieto testy sú síce voľne dostupné, ale zatiaľ nie je prístupný žiadny ucelený program, ktorý sa jednoducho používa a obsahuje celú túto sadu testov. Tento fakt ma viedol k tomu, aby som vypracoval riešenie, ktoré bude ľahko použiteľné pre takmer každého užívateľa, bude ucelené a dodrží všetky požiadavky a štandardy NIST-u pre testovanie generátorov náhodných čísiel. Samozrejmosťou je, že sám program bude otestovaný tak, aby pre testovaciu vstupnú sekvenciu vrátil identický výsledok podľa štandardu. Týmto bude zaručené, že na výsledky môjho programu budú kvalitné a sa bude dať na spoľahnúť.

V tejto práci je možné nájsť popis najznámejších zástupcov generátorov náhodných čísel, rozdelenie setov testov spolu s autormi týchto testov, všetkých pätnásť testov zo setu spoločnosti NIST s popisom, praktickým príkladom na zjednodušenej sekvencii, vyhodnotením výsledkov a doporučenými vstupmi pre test. Ďalej je tu kompletná dokumentácia ku každému naprogramovanému testu, jeho kód a vysvetlenie, čo ktorý príkaz znamená. Samozrejmosťou je popísanie ovládania testovacieho programu s názornou ukážkou funkčnosti. V závere práce je popísané, ako získať náhodné čísla z každého z testovaných generátorov, otestovanie získaných sekvencií náhodných čísel, zobrazenie a vyhodnotenie dosiahnutých výsledkov testov.

I. TEORETICKÁ ČÁST

1 PROBLEMATIKA GENERÁTORU NÁHODNÝCH ČÍSEL

Pri hlbšom zamyslení to môže vyznieť zvláštne, že počítač pracujúci presne podľa deterministických princípov môže generovať náhodné čísla. Veď pre rovnaké vstupné podmienky počítač vždy vráti rovnaký výstup. Tento problém môže vyriešiť pripojiteľné externé zariadenie, ktoré generuje náhodné čísla pomocou rôznych fyzikálnych javov ako je napríklad šum. Tento postup je však nepraktický, a tak sa dnes vo väčšine prípadov používajú náhodné čísla vygenerované za pomoci algoritmu. Ďalšími faktami, ktoré nahrávajú generátorom pseudonáhodných čísel je znalosť a stálosť ich distribúcie. Aby náhodné čísla boli užitočné, musia byť nezávisle a identicky distribuované. [1]

1.1 Čo je to náhodnosť

Existuje spojenie medzi náhodnou a náhodnosťou, a to, že náhoda je produktom náhodnosti. Pokiaľ je reč o naozaj náhodnom jave, potom každý výstup z tohto javu musí byť jedinečný. Toto sa dá dobre predstaviť na kocke, ktorá má nekonečne veľa hrán a každý hod touto kockou má za následok novú, nikdy predtým nezískanú hranu. S reálnou kockou je to však inak. Výsledok hodu môže byť iba jedna zo šiestich možností, a aby sa jednalo o platné pokusy, frekvenčné rozloženie výsledkov musí splňovať určité zákonitosti. Avšak jedna z týchto zákonitostí je prítomnosť náhod, ako napríklad že padne číslo 6 desaťkrát za sebou. [2]

Náhodnosť by sa dala jednoducho definovať ako chaos, nepredvídateľnosť, alebo niečo bez štruktúry či vzoru. V prenesení na sekvenciu čísel sa dá náhodnosť charakterizovať ako jav, kedy nie je možné s určitosťou vypočítať ďalšie číslo pomocou nejakého algoritmu, a zároveň nie je možné získať rovnakú sekvenciu čísiel. Pred kvantovou mechanikou obecne platilo, že nič nie je náhodné, náhoda neexistuje, a rozdielne výsledky pokusov (napríklad hod kockou alebo mincou) sú zapríčinené zanedbaním nejakej sily. Kvantová mechanika odhalila, že náhoda existuje, napríklad pri pozorovaní hmoty na mikroskopickej úrovni. Napríklad nevieme s presnosťou určiť, kedy sa rádioaktívne jadro rozpadne. Aj z tohto dôvodu sú kvantové javy vynikajúce pre vytváranie sekvencie skutočne náhodných čísel. Objavenie deterministického chaosu ukázalo, že náhodnosť a nepredvídateľnosť sa vyskytuje aj v klasickej fyzike. Momentálne vieme získavať náhodné čísla, ale nevieme predpovedať aké budú ďalšie. Toto nám dovoľuje vygenerovať sekvenciu pseudonáhodných čísiel pomocou algoritmu, ktorá na prvý pohľad vyzerá úplne rovnako ako sekvencia pravých náhodných čísiel. [2]

1.2 Rozdelenie generátorov náhodných čísel

Generátory náhodných čísel sa rozdeľujú na generátory pseudonáhodných čísel (PRNG) a generátory pravých náhodných čísel (TRNG). Na prvý pohľad nie je možné rozoznať, ktorý z týchto generátorov vygeneroval skúmanú sekvenciu čísel, avšak pomocou rôznych štatistických analýz a testov je možné odhaliť nespoľahlivé generátory pseudonáhodných čísel. [1]

1.2.1 PRNG – Generátor pseudonáhodných čísel

Tento druh generátoru náhodných čísel predstavuje algoritmus, ktorý používa matematické vzorce na generovanie náhodných čísel. Takto vygenerovaná sekvencia čísel sa svojimi vlastnosťami blíži k sekvencii náhodných čísel. Generátor začína generovať náhodné čísla z ľubovoľného štartovacieho stavu pomocou stavu semien (seed state). V krátkom čase generátor vygeneruje veľké množstvo čísiel, ktoré môžu byť znovu vygenerované, ak budú použité rovnaké podmienky pri spustení generátoru. Problém u takéhoto generátoru je, že po určitom množstve náhodne vygenerovaných čísel sa tieto náhodné čísla začnú opakovať. Napriek tomu, že táto vlastnosť je silne nechcená, moderné generátory majú dobu opakovania tak veľkú, že je prakticky zanedbateľná vo väčšine prípadov. Takto vygenerované pseudonáhodné čísla nájdu svoje uplatnenie najmä v aplikáciách, ktoré vyžadujú mnoho náhodných čísel v krátkom čase, ako napríklad programy pre simulovanie a modelovanie situácii. Naopak, tieto generátory nie sú najvhodnejšou voľbou pre programy vyžadujúce naozaj nepredvídateľné čísla ako napríklad hazardné hry alebo šifrovanie dát. Najznámejšími algoritmami na generovanie pseudonáhodných čísel sú: LFG – Lagged Fibonacci Generator, LFSR – Linear-feedback shift register a BBS – Blum Blum Shub. [3]

1.2.2 TRNG – Generátor pravých náhodných čísel

O číslach vygenerovaných týmto generátorom sa dá povedať, že sú naozaj náhodné, a je prakticky nemožné získať rovnakú sekvenciu čísel i pri rovnakých vstupných podmienkach. Princíp tohto generátoru spočíva v meraní určitého fyzikálneho javu, ktorý sa deje náhodne (napríklad rozklad rádioaktívnych látok alebo šum). Najznámejší je pravdepodobne biely šum, ktorý má konštantnú výkonovo spektrálnu hustotu (rovnako široké frekvenčné pásma majú rovnakú energiu). Generovať šum sa dá napríklad pomocou Zenerovej diódy v záver-

nom smere, pomocou bipolárneho tranzistoru na prechode emitor-báza, pomocou A/D prevodníku nastaveného na vysokú citlivosť bez pripojeného vstupu, alebo aj obyčajným rádiom, ktoré je naladené na nepoužívanú frekvenciu. [4]

Získať takto vygenerovane čísla je možné buď vlastným hardwarovým zariadením, alebo pomocou rôznych online služieb ako napríklad random.org. [5]

2 NAJVÝZNAMNEJŠÍ ZÁSTUPCOVIA GENERÁTOROV NÁHODNÝCH ČÍSEL

V tejto kapitole budú podrobnejšie predstavené niektoré najznámejšie a najpoužívanejšie generátory náhodných aj pseudonáhodných čísel.

2.1 Random.org

Najznámejšia služba, ktorá poskytuje náhodne vygenerované čísla pomocou generátoru pravých náhodných čísel je webstránka www.random.org. Služba bola založená v roku 1998 Dr. Madsom Haarom zo Školy informatiky a štatistiky na Univerzite Trinity Dubline v Írsku. Dnes random.org prevádzkuje spoločnosť Randomness and Integrity Services Ltd. Generátor na tejto webstránke používa na generovanie pravých náhodných čísel atmosférický šum, ktorý podľa je podľa nich lepším zdrojom náhodnosti ako algoritmy generujúce pseudonáhodné čísla. Táto služba je široko využívaná online lotériami a hazardnými webmi, rôznymi online hrami, umelcami a v neposlednom rade ju taktiež používajú vedecké aplikácie. [5]

Podľa štatistík vedených službou random.org od spustenia služby použitý generátor vygeneroval k dátumu písania tejto vety približne 1,6 bilióna náhodných bitov, priemernou rýchlosťou 2675 bit/s, čo je podľa štatistík uvedených na jednej z ich podstránok približne 191GiB. Tento objem dát sa dá preniesť na cca 301 CD-ROMoch alebo 42 DVD-ROMoch. [6]

2.2 Mersenne Twister

Mersenne Twister je najpoužívanejším univerzálnym generátorom pseudonáhodných čísel. Bol vyvinutý v rokoch 1996 a 1997 dvojicou Makoto Matsumoto a Takuji Nishimura, ktorý ho pomenovali po Mersennových prvočíslach (Mersennovo prvočíslo je také prvočíslo, ktoré je o jedno menšie, ako celočíselná mocnina dvojky, napr.: $2^3 - 1 = 7$, $2^7 - 1 = 127$) a po jeho predchodcovi generátoru „Twisted GFSR“ (twisted generalised feedback shift register, navrhnutý Matsumotom a Kuritom v roku 1992). Tento generátor je prvý generátor pseudonáhodných čísel, ktorý dokáže generovať vysokokvalitné celočíselné pseudonáhodné čísla a zároveň odstraňuje takmer všetky nedostatky starších PRNG. Najpoužívanejšia verzia tohto generátoru (MT19937) pracuje s Mersennovým prvočíslom číslom $2^{19937}-1$ a používa slovo o dĺžke 32 bitov. Tento generátor samozrejme prešiel všetkými Diehard testami, ďalej

prešiel load testami a tiež ultimate load testami. Na používanie tohto generátoru sa neviaže žiadna platená licencia a pravdepodobne je tento generátor aj najrozšírenejší generátor kvalitných pseudonáhodných čísel. Avšak aj tento generátor ma svoje nevýhody. Jednou z nich je, že vygenerované pseudonáhodné čísla nie sú kryptograficky bezpečné. Pri znalosti dostatočne veľkej vzorky z výstupu sa pomocou jednoduchej lineárnej transformácie stane výstup lineárnou opakujúcou sa sekvenciou. Túto nevýhodu odstraňuje generátor CryptMT, avšak tento generátor už nie je dostupný zadarmo. [7]

Generátor Mersenne Twister je možné nájsť napríklad v programoch Matlab či Wolfram Mathematica, v programovacích jazykoch PHP, Python, Ruby, C++ a ďalších.

2.3 Generátor náhodných čísel v programe Microsoft Excel

Najjednoduchšie získanie náhodného čísla v Exceli je pomocou funkcie *RAND().* Tá vráti pseudonáhodné číslo v intervale <0, 1>. Po dôkladnom testovaní pomocou štandardných Diehard testov náhodnosti sa ukázalo, že generátor v Exceli pred rokom 2003 je nedostatočný, pretože dlhšia sekvencia náhodných čísel sa po určitom počte vygenerovaných pseudonáhodných číslach začne opakovať. Aj keď tento nedokonalý generátor neprechádzal testami len pri vysokom počte volaní (milión a viac), a problém sa taktiež netýkal všetkých užívateľov, Microsoft poveril vývojom nového generátora B.A. Wichmana a I.D. Hilla. Nový generátor funguje tak, že vygeneruje 3 náhodné čísla *X (0 – 30268)*, *Y (0 – 30306)* a *Z (0 - 30322).* Následne sa spočíta zvyšok po delení takto: $X = MOD(171X, 30269)$, $Y = MOD(172 * Y, 30307)$, $Z = MOD(170 * Z, 30323)$. Keďže kombinácia náhodných čísel vráti znovu náhodné číslo, tak poslednom kroku sa tieto pseudonáhodné čísla skombinujú v rovnici, ktorej výstupom je pseudonáhodné číslo z intervalu <0, 1>. Microsoft túto rovnicu uvádza na svojich webových stránkach takto:

$$RANDOM = AMOD\left(\frac{FLOAT(X)}{30269{,}0} + \frac{FLOAT(Y)}{30307{,}0} + \frac{FLOAT(Z)}{30323{,}0}, 1.0\right)$$

Keďže sa nejedná o pravé náhodné čísla, ale o pseudonáhodné čísla, sekvencia sa po určitom počte volaní začne opakovať. Microsoft tvrdí, že algoritmus Wichman-Hill vygeneruje minimálne 10^{13} pseudonáhodných čísiel predtým, ako sa sekvencia začne opakovať. [8]

2.4 Generátor náhodných čísel pre programovací jazyk C

Programovací jazyk C obsahuje knižnicu *stdlib.h*, v ktorej sa nachádza funkcia *rand()*. Táto funkcia vracia pseudonáhodné hodnoty od 0 do *RAND_MAX*, maximálna hodnota sa môže

líšiť, ale nikdy nie je menšia ako *32767 (2^{15}).* Keďže funkcia *rand()* pri každom volaní vráti úplne rovnaké hodnoty, nie je vhodné ju volať samostatne. Pred zavolaním tejto funkcie sa musí inicializovať začiatok sekvencie pomocou funkcie *srand()* a parameter s ktorým sa funkcia volá sa nazýva seed. Pri vynechaní tejto funkcie sa použite ako seed jednotka, teda *srand(1).* Parameter tejto funkcie by však nemal byť konštanta, pretože program bude znovu vracať rovnakú sekvenciu po každom zavolaní. Ako parameter sa môže použiť napríklad systémový čas, ktorý sa neustále mení. [9]

Zápis veľmi jednoduchého programu, ktorý vypíše sekvenciu desiatich náhodných čísiel od *0* po *99* vyzerá takto:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    int mytime = time(0);
    // mytime = 1546300800; // 1.1.2019 00:00:00 UTC
    srand(mytime);
    for(int i = 0; i<10; i++)
        printf(" %d,", rand()%100);

    return 0;
}
```

Výstup tohto programu pre čas v okamžiku písania tohto odstavca, teda *1521389492* sekúnd od *1.1.1970 00:00:00 UTC*, potom bude: *39, 69, 15, 68, 24, 10, 19, 68, 0, 97*. Použitím rovnakého času sa dá dokázať, že program vráti rovnakú sekvenciu, a teda pridávaním alebo odoberaním sekúnd vieme jednoducho získať výstupy tejto funkcie v budúcnosti alebo v minulosti. Napríklad, *1.1.2019 o 0:00:00 UTC* uplynie presne *1546300800* sekúnd od *1.1.1970 00:00:00 UTC*, po dosadení tohto čísla do funkcie *srand()* program vráti hodnoty *80, 76, 47, 0, 14, 42, 7, 62, 49, 32,* ktoré by program vypísal presne na prelome rokov 2018 a 2019. Ďalšou zaujímavou vlastnosťou je, že pri požiadavke na vygenerovanie 10 čísel od 0 do 999 v rovnakom čase, program vráti *980, 76, 847, 600, 314, 842, 107, 762, 549, 332*. Na prvý pohľad je zreteľné, že generátor generuje mnohonásobne väčšie čísla a následne z nich odstrihne a vráti posledné číslice. Ak sa program ďalej upraví do takej podoby, aby vypísal pseudonáhodnú sekvenciu desiatich čísel od 0 do 50, výstupom v rovnakom čase bude *30, 26, 47, 0, 14, 42, 7, 12, 49, 32.* Tu sa dá pozorovať, že pokiaľ je číslo väčšie ako maximálna požadovaná hodnota, generátor túto maximálnu hodnotu od čísla odpočíta. Napríklad číslo *76* na druhej pozícii generátor upravil na *26*, alebo z čísla *62* na ôsmej pozícii je teraz *12*. Z týchto dôvodov je absolútne nevhodné použiť takto naprogramovaný generátor

pri generovaní šifrovacích kľúčov, pri online hazardných hrách, športke, alebo kdekoľvek inde, kde sa vyžaduje náhodnosť.

2.5 Generátor náhodných čísel pre programovací jazyk Python 3.6

Generátor náhodných čísiel pre Python je obsiahnutý v základnom inštalačnom balíčku, a nainštaluje v podobe knižnice *random.py*. Táto knižnica používa ako základ generátor MersenneTwister s periódou $2^{19937}-1$, ktorý vracia desatinné čísla s presnosťou 53 bitov. Keďže je tento generátor plne deterministický, nemal by sa používať na kryptografické účely. Generátor pseudonáhodných čísel z knižnice *random.py* dokáže:

- vybrať pseudonáhodné celé číslo z intervalu pomocou funkcií
- vybrať pseudonáhodný prvok z nejakej sekvencie (napríklad zo zoznamu alebo slovníku) pomocou funkcií
- vybrať pseudonáhodné reálne číslo pomocou funkcií
- vyžiadať náhodné číslo od operačného systému
- vrátiť sekvenciu pseudonáhodných bitov
- získať a uložiť stav generátoru

Princíp funkčnosti je podobný ako pri generátore z jazyka C, to znamená že pokiaľ je na vstup privedený rovnaký seed, výstupná sekvencia bude identická. Hodnota seed-u sa však nezadáva priamo z programu, ale je vypočítaná priamo v tejto knižnici. Pre opakovanie sekvencie je teda potrebné uložiť si stav generátoru. [10][11]

3 ŠTATISTICKÉ TESTY PRE TESTOVANIE NÁHODNÝCH ČÍSEL

Testovať generátory náhodných čísel sa dá pomocou niekoľkých sérii testov. V tabuľke 1 je uvedený prehľad dostupných setov štatistických testov používaných na testovanie generátorov náhodných čísel. [12]

Tabuľka 1: Prehľad setov štatistických testov [13]

Autor	Názov setu	Zoznam testov
Donald Knuth	The Art Of Computer Programming Vol. 2 Seminumerical Algorithms	frequency, serial, gap, poker, coupon collector's, permutation, run, maximum-of-t, collision, birthday spacings, and serial correlation
George Marsaglia	Diehard	birthday spacings, overlapping permutations, ranks of 31x31 and 32x32 matrices, ranks of 6x8 matrices, monkey tests on 20-bit Words, monkey tests OPSO, OQSO, DNA, count the 1's in a stream of bytes, count the 1's inspecific bytes, parking lot, minimum distance, random spheres, squeeze, overlapping sums, runs, craps
Helen Gustafson	Crypt-XS	frequency, binary derivative, change point, runs, sequence complexity, linear complexity
Alfred Menezes	Handbook of Applied Cryptography	
Andrew Rukhin	NIST Statistical Test Suite	monobit, block frequency, cumulative sums, runs, long runs, rank, spectral, nonoverlapping template matchings, overlapping template matchings, Maurer's universal statistical, approximate entropy, random excursions, complexity, linear complexity, serial

Set testov Diehard bol predstavený v roku 1995 a momentálne už nie je udržovaný. Set testov od NIST-u vychádza zo setu testov Diehard, a tieto testy zdokonaľuje. Všetky testy v oboch setoch rozhodujú o náhodnosti pomocou hodnoty *p* rovnako, a to tak, že sekvencia

sa považuje za náhodnú iba vtedy, keď je hodnota $p < 0,01$. Nakoľko k setu testov Diehard nie je viac dostupná oficiálna dokumentácia, z ktorej je nutné pri implementácii testov vychádzať, budú implementované modernejšie a aktuálnejšie testy od organizácie NIST, ktoré vyberajú a vylepšujú tie najlepšie testy naprieč všetkými setmi.[5][12][13]

3.1 Frekvenčný test: Monobit

Test sa zameriava na rozloženie jednotiek a núl v náhodne generovanej binárnej sekvencii. Predpoklad pre úplne náhodné čísla je, že jednotky a nuly budú mať približne rovnaké zastúpenie. Výstup testu je zlomok, v ideálnom prípade blížiaci sa k ½. Presnosť výsledku stúpa s dĺžkou vstupnej sekvencie. Pokiaľ generátor neprejde týmto testom, je pravdepodobné, že neprejde ani ostatnými testami. [12]

Test bude demonštrovaný na sekvencii $S = 10011000$ s dĺžkou $n = 8$. Podľa organizácie NIST je tento test vhodný pre sekvencie o dĺžke 100 a viac bitov. [12]

Postup testovania podľa organizácie NIST

1. prevod všetkých núl v sekvencii na -1 a následné sčítanie:
$$S_n = 1 + (-1) + (-1) + 1 + 1 + (-1) + (-1) + (-1) = -2 \quad (1)$$
2. vypočítanie štatistického testu:
$$S = \frac{|S_n|}{\sqrt{n}} = \frac{|-2|}{\sqrt{8}} \quad (2)$$
3. vypočítanie hodnoty P pomocou funkcie *erfc*:
$$P = erfc\left(\frac{S}{\sqrt{2}}\right) \cong 0,4795 \quad (3)$$
Hodnotu funkcie *erfc(z)* dokáže rýchlo a presne spočítať voľne dostupná online verzia softwaru Wolfram Alpha.
4. Rozhodnutie, či sa jedná o náhodnú sekvenciu. Pokiaľ platí, že hodnota $P \geq 0,01$, tak testovaná sekvencia pomocou tohto testu sa dá považovať za náhodnú. Keďže testovaná sekvencia prešla testom s hodnotou $P = 0,4795$, dá sa o tejto sekvencii tvrdiť, že je podľa tohto testu náhodná.[12]

3.2 Frekvenčný test v rámci bloku

Tento test odhaľuje počty jednotiek v n-bitových blokoch. Predpoklad pre náhodné čísla je, že pomer jednotiek a núl bude približne 1:1. Pri testovaní jednobitových blokov tento test vracia rovnaké výsledky ako predošlí frekvenčný test monobit.[12]

Ako vstup do testu bude použitý reťazec (ε) o dĺžke n bitov, ktorý bude rozdelený na M rovnako dlhých blokov. Pre spoľahlivé otestovanie generátoru organizácia NIST odporúča aby mal vstupný reťazec dĺžku aspoň *100* bitov a malo by platiť pravidlo $n \geq MN$. Počet blokov M by mal byť minimálne *20*, a zároveň by mal byť väčší ako stotina dĺžky reťazca n. Počet blokov v teste N by mal byť menší ako *100*.[12]

Pre potreby demonštrácie testu budú použité nasledujúce hodnoty:

$\varepsilon = 1001110101$ $n = 10$ $M = 4$

Postup testovania podľa organizácie NIST:

1. Vypočítanie počtu možných blokov z reťazca. Bity na konci reťazca, ktoré sú príliš krátke na vytvorenie reťazca budú vynechané.

$$N = \left[\frac{n}{m}\right] = \left[\frac{10}{4}\right] = 2 \tag{4}$$

   Zo zadaného reťazca ε je teda možné vytvoriť 2 štvorbitové bloky: $n_1=1001$ a $n_2=1101$. Bity *01* z konca reťazca nebudú použité.

2. V tomto kroku sa vypočítajú jednotlivé pomery jednotiek v každom z blokov.

$$\pi_i = \frac{\sum_{j=1}^{M} \varepsilon_{(i-1)M+j}}{M} \tag{5}$$

   Rovnica sa počíta pre $1 \leq i \leq N$. Výsledkom rovnice je pomer jednotiek ku nulám v každom z blokov: $\pi_1 = 1/2$ a $\pi_2 = ¾$

3. Porovnanie výsledkov so referenčnou distribúciou $\chi^2 = 0,5$.

$$\chi^2 = 4M \sum_{i=1}^{N} \left(\pi_i - \frac{1}{2}\right)^2 \tag{6}$$

$$\chi^2 = 4 * 4 * \left(\left(\frac{1}{2} - \frac{1}{2}\right)^2 + \left(\frac{3}{4} - \frac{1}{2}\right)^2\right) = 1 \tag{7}$$

4. Vypočítanie hodnoty P pomocou funkcie *igamc*. Túto funkciu taktiež dokáže spoľahlivo vypočítať dostupná online verzia softwaru Wolfram Alpha.

$$P = igamc\left(\frac{N}{2}, \frac{\chi^2}{2}\right) = 0{,}606531 \tag{8}$$

5. Rozhodnutie, či sa jedná o náhodnú sekvenciu. Pokiaľ platí, že hodnota $P \geq 0,01$, tak testovaná sekvencia pomocou tohto testu sa dá považovať za náhodnú. Keďže testovaná sekvencia prešla testom s hodnotou $P = 0,606531$, dá sa o tejto sekvencii tvrdiť, že je podľa tohto testu náhodná.[12]

3.3 Test postupností

Tento test vyhodnocuje postupnosti rovnakých znakov v sekvencii. Postupnosťou sa rozumie opakovanie rovnakého znaku bez prerušenia, teda postupnosť o dĺžke z obsahuje presne z identických bitov a je ohraničená zľava aj sprava opačným bitom. Test určuje či počet postupností jednotiek a núl rôznych dĺžok je rovnaký ako pri náhodnom reťazci. Test postupností teda testuje, či oscilácia jednotiek a núl je príliš rýchla alebo pomalá.[12]

Ako vstup je použitá sekvencia bitov ε o dĺžke n. Premenná V_n udáva súčet počtu postupností jednotlivých bitov v postupnosti ε. Organizácia NIST odporúča minimálnu dĺžku sekvencie $n = 100$[12]

Ako názorná ukážka postupu je zvolená sekvencia $\varepsilon = 1001101011$ s dĺžkou $n = 10$

Postup testovania podľa organizácie NIST:

1. Výpočet pomeru jednotiek k dĺžke sekvencie

$$\pi = \frac{\sum j^{\varepsilon} j}{n} = \frac{6}{10} \tag{9}$$

2. Zistenie počtu blokov

$$V_n = \sum_{k-1}^{n-1} r(k) + 1 \tag{10}$$

kde $r(k) = 0$ ak $\varepsilon_k = \varepsilon_{k+1}$ a $r(k) = 0$ pre ostatné prípady.

V tomto prípade teda zápis vyzerá takto:

$$\varepsilon = 1\ 00\ 11\ 0\ 1\ 0\ 11 \tag{11}$$

$$V_{10} = (1 + 0 + 1 + 0 + 1 + 1 + 1 + 1 + 0) + 1 = 7 \tag{12}$$

3. Vypočítanie hodnoty P.

$$P = erfc\left(\frac{|V_n - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right) \tag{13}$$

$$P = erfc\left(\frac{\left|7 - 2 * 10 * \frac{6}{10} * \left(1 - \frac{6}{10}\right)\right|}{2 * \sqrt{2 * 10} * \frac{6}{10} * \left(1 - \frac{6}{10}\right)}\right) = 0{,}147232 \tag{14}$$

4. Rozhodnutie, či sa jedná o náhodnú sekvenciu. Pokiaľ platí, že hodnota $P \geq 0{,}01$, tak testovaná sekvencia pomocou tohto testu sa dá považovať za náhodnú. Keďže testovaná sekvencia prešla testom s hodnotou $P = 0{,}147232$, dá sa o tejto sekvencii tvrdiť, že je podľa tohto testu náhodná. [12]

3.4 Najdlhšia sekvencia jednotiek v bloku

Tento test hľadá najdlhšiu nepretržitú sekvenciu jednotiek v bloku. Následne sa sekvencia porovná s očakávanou najdlhšou sekvenciou pri náhodnom reťazci.[12]

Ako vstup do tejto funkcie sa použije reťazec ε o dĺžke n a dĺžka bloku M. Z uvedených vstupov sa vypočíta počet všetkých blokov N.[12]

Pre potreby demonštrácie boli zvolené nasledujúce vstupy:

ε = 11110001 00111110 11001100 11010111 01100000 11001001 00111001 00011010

n = 64 *M = 8* *N = 8*

Postup testovania poľa organizácie NIST:

1. Rozdelenie sekvencie ε do blokov rovnakej dĺžky
2. Vypočítanie frekvencií v_i najdlhších sekvencií jednotiek v každom bloku a rozdelenie do podľa najdlhšej sekvencie

   11110001 = 4 *01100000 = 2*

   00111110 = 5 *11001001 = 2*

   11001100 = 2 *00111001 = 3*

   11010111 = 3 *00011010 = 2*
3. Vypočítanie funkcie χ^2. Hodnoty π_i boli pre tento test stanovené nasledovne:

   Tabuľka 2: Pravdepodobnosti pre dĺžku sekvencie jednotiek

Dĺžka sekvencie jednotiek	Pravdepodobnosť π	Počet
≤1	0,2148	0
2	0,3672	4
3	0,2305	2
≥4	0,1875	2

$$\chi^2 = \sum_{i=1}^{K} \frac{(v_i - N\pi_i)^2}{N\pi_i} \tag{15}$$

$$\chi^2 = \frac{(0 - 16 * 0{,}2148)^2}{16 * 0{,}2148} + \frac{(4 - 16 * 0{,}3672)^2}{16 * 0{,}3678} + \frac{(2 - 16 * 0{,}2305)^2}{16 * 0{,}2305} + \frac{(2 - 16 * 0{,}1875)^2}{16 * 0{,}1875} = 5{,}140267214 \tag{16}$$

4. Vypočítanie hodnoty P za pomoci funkcie *igamc*. Hodnota K je daná, a pre tento prípad je rovná 3.

Tabuľka 3: Hodnoty premennej K a N pre dĺžku bloku

M	K	N
8	3	16
128	5	49
10000	6	75

$$P = igamc\left(\frac{K}{2}, \frac{\chi^2}{2}\right) = 0{,}1434 \tag{17}$$

5. Vypočítaná hodnota P rozhoduje o náhodnosti sekvencie. Pokiaľ je P ≤ 0,01 generátor negeneruje náhodné čísla.[12]

3.5 Test binárnej hodnoty matíc

Úloha tohto testu spočíva v zistení lineárnej závislosti medzi sekvenciami s rovnakou dĺžkou vytvorených z pôvodného reťazca bitov.[12]

Do testu vstupuje sekvencia *ε* o dĺžke *n*. Tá sa rozdelí do matice s počtom riadkov *R* a stĺpcov *S*. Generátor vygeneroval sekvenciu bitov *ε = 110 010 100 111 010 011 010 110* o dĺžke *n = 24*. Počet riadkov *R* a stĺpcov *S* bol stanovený na 3.[12]

Postup testovania podľa organizácie NIST je tento:

1. Sekvencia sa rozdelí do matíc podľa nasledujúceho vzorca. Bity ktoré sa do matíc nezmestia sa zanedbajú.

$$N = \left[\frac{n}{RS}\right] = 2 \tag{18}$$

2. Bity sa rozdelia do matíc, ako prvý sa naplní prvý riadok prvej matice potom druhý riadok tej istej matice. Po naplnení prvej matice sa postupuje na druhú, logika napĺňania zostáva nezmenená. Zo zadanej sekvencie teda vzniknú tieto matice:

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} a\ B = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \tag{19}$$

3. Výpočet hodnosti matíc. Hodnosť oboch matíc, teda počet unikátnych nenulových riadkov je *3*.
4. Spočítanie matíc s hodnosťou rovnou počtu riadkov F_R, matíc s hodnosťou o jedno menšou ako počet riadkov F_{R-1}, a ostatných matíc.

$$F_R = F_3 = 2 \tag{20}$$

$$F_{R-1} = F_2 = 0 \tag{21}$$

$$Ostatné\ matice = N - F_R - F_{R-1} = 0 \tag{22}$$

5. Vypočítanie funkcie χ^2

$$\chi^2 = \frac{(F_R - 0{,}2888N)^2}{0{,}2888N} + \frac{(F_{R-1} - 0{,}5776N)^2}{0{,}5776N} + \frac{(N - F_R - F_{R-1} - 0{,}1336N)^2}{0{,}1{,}336N} \tag{23}$$

$$\chi^2 = \frac{(2 - 0{,}2888 * 2)^2}{0{,}2888 * 2} + \frac{(0 - 0{,}5776 * 2)^2}{0{,}5776 * 2} + \frac{(2 - 2 - 0 - 0{,}1336 * 2)^2}{0{,}1336 * 2} \tag{24}$$

$$\chi^2 = 4{,}92521 \tag{25}$$

6. Výpočet hodnoty P

$$P = igamc\left(1, \frac{\chi^2}{2}\right) = 0{,}08521 \tag{26}$$

7. Vyhodnotenie, či sa jedná o náhodnú sekvenciu za pomoci vypočítanej hodnoty *P*. Pokiaľ platí, že $P \leq 0{,}01$, tak sekvencia náhodná nie je. V ostatných prípadoch sa jedná o náhodnú sekvenciu.[12]

3.6 Test pomocou diskrétnej fourierovej transformácie – Spektrálny test

Tento test sa pokúša nájsť periodické vlastnosti v testovanej sekvencii bitov, ktoré by naznačovali odchýlky od predpokladu náhodnosti. Do testu vstupuje sekvencia bitov ε o dĺžke n. Je doporučené aby vstupná sekvencia mala dĺžku aspoň 1000 bitov.[12]

Postup testovania podľa organizácie NIST:

1. Nuly v sekvencii sa zmenia na *-1* a jednotky na *+1*. Následne sa vytvorí sekvencia X kde $X = x_1, x_2, \ldots, x_n$, kde $x_i = 2\ \varepsilon_i - 1$. Pre sekvenciu ε = 1001010011 s dĺžkou $n = 10$ bude sekvencia X vyzerať nasledovne:

$$X = 1, -1, -1, 1, -1, 1, -1, -1, 1, 1 \tag{27}$$

2. Na sekvenciu X sa aplikuje diskrétna fourierova transformácia. Výstupom je sekvencia komplexných premenných predstavujúca periodické komponenty sekvencie bitov na rôznych frekvenciách
3. Výpočet funkcie $M = modulus(S') = |S^*|$, kde S' je prvá polovica reťazca S. Výstup funkcie *modulus* je sekvencia výšok vrcholov.
4. Výpočet $T = \sqrt{\ln\left(\frac{1}{0{,}05}\right) * n}$. Pokiaľ sa jedná o generátor náhodných čísiel, *95%* hodnôt získaných z testu nesmie presiahnuť túto hodnotu.

$$T = 5{,}473328 \tag{28}$$

5. Výpočet hodnoty $N_0 = \frac{0,95n}{2}$. Hodnota N_0 je očakávaná hodnota vrcholov ktoré sú menšie ako T

$$N_0 = 4,75 \tag{29}$$

6. Výpočet hodnoty N_1. Jedná sa o odpozorovaný počet vrcholov v M, ktoré sú menšie ako T.

$$N_1 = 4 \tag{30}$$

7. Výpočet hodnoty d

$$d = \frac{(N_1 - N_0)}{\sqrt{\frac{n * 0,95 * 0,05}{4}}} = -2,17642875 \tag{31}$$

8. Výpočet hodnoty P za pomoci funkcie *erfc*

$$P = erfc\left(\frac{|d|}{\sqrt{2}}\right) = 0,029523 \tag{32}$$

9. Rozhodnutie. Pokiaľ je hodnota P menšia ako $0,01$, sekvenciu nie je možné považovať za náhodnú. Z toho vyplýva, že testovaná sekvencia je náhodná.[12]

3.7 Non-overlapping template matching test

Pri tomto teste sa sleduje počet výskytov určených reťazcov, ktoré by odhalili generátory produkujúce nadmerné množstvo danej neperiodickej sekvencie. Počas testovania sa použije m-bitové okno pomocou ktorého sa vyhľadávajú m-bitové sekvencie. Ak hľadaná sekvencia nie je v okne nájdená, okno sa posunie o jeden bit. Ak je vzor v okne nájdený, okno sa vynuluje a vyhľadávanie sa obnoví. Ako vstup teda poslúži sekvencia náhodných bitov ε o dĺžke n a dĺžka šablóny m. NIST odporúča voliť $m = 9$ alebo $m = 10$, ďalej $N \leq 100$ aby vypočítaná hodnota P v predposlednom kroku dávala zmysel. Posledné podmienky pre dosiahnutie ideálnych výsledok sú, $M > 0,01 * n$ a $N = [n/M]$.[12]

Postup testovania podľa organizácie NIST:

1. Rozdelenie sekvencie náhodných bitov do N nezávislých blokov o bitovej dĺžke M

   ε = *1010 0100 1011 1001 0110*

   $n = 20$ $M = 10$ $N = 2$

   Z takto danej sekvencie a jej parametrov sa dajú zostaviť dva bloky:

   B1 = 1010 0100 10 *B2 = 1110 0101 10*

2. Zvolí sa skúmaná šablóna B s dĺžkou m. Ďalej je zavedená premenná W_{blok}, ktorá vyjadruje počet objavení sa šablóny v bloku.

Pre tento prípad je zvolená šablóna *B = 001* s dĺžkou *m = 3*. Pokiaľ sa sekvencia vyskytne v bloku, premenná *W* sa zvýši o *1* a ostatné bity sa preskočia.

Tabuľka 4: Zobrazenie bitov a ich porovnávanie so šablónou

Pozícia bitov v bloku	Blok 1		Blok 2	
	Bity	W_1	Bity	W_2
1-3	101	0	111	0
2-4	010	0	110	0
3-5	100	0	100	0
4-6	001	1	001	1
5-7	Preskočí sa	-	Preskočí sa	-
6-8	Preskočí sa	-	Preskočí sa	-
7-9	001	2	011	1
8-10	010	2	110	1

Výstup tohto kroku je, že šablóna *B* sa vyskytuje v prvom bloku dvakrát a v druhom bloku raz.

3. Vypočítanie teoretického priemeru μ a rozptylu σ^2.

$$\mu = \frac{M - m + 1}{2^m} = \frac{10 - 3 + 1}{2^3} = 1 \tag{33}$$

$$\sigma^2 = M\left(\frac{1}{2^m} - \frac{2m - 1}{2^{2m}}\right) = 10\left(\frac{1}{2^3} - \frac{2 * 3 - 1}{2^{2*3}}\right) = 0{,}46875 \tag{34}$$

4. Výpočet χ^2

$$\chi^2 = \sum_{j=1}^{N} \frac{(W_j - \mu)^2}{\sigma^2} = \frac{(2-1)^2 + (1-1)^2}{0{,}46875} = 2{,}1333333 \tag{35}$$

5. Výpočet hodnoty *P* pomocou funkcie *igamc*.

$$P = igamc\left(\frac{N}{2}, \frac{\chi^2}{2}\right) = igamc\left(\frac{2}{2}, \frac{2{,}1333333}{2}\right) = 0{,}344154 \tag{35}$$

6. Rozhodnutie, či sa jedná o náhodnú sekvenciu. Pokiaľ platí, že hodnota $P \geq 0{,}01$, tak testovaná sekvencia pomocou tohto testu sa dá považovať za náhodnú. Keďže testovaná sekvencia prešla testom s hodnotou $P = 0{,}344154$, dá sa o tejto sekvencii tvrdiť, že je podľa tohto testu náhodná.[12]

3.8 Overlapping template matching test

Tento test je veľmi podobný s predchádzajúcim, tiež sa používa šablóna a okno o dĺžke *m* a sekvencia sa posúva o jeden bit v bloku. Jediný rozdiel je v tom, že pokiaľ sa šablóna bude zhodovať s oknom, tieto bity sa nepreskočia ako v minulom prípade, ale bude sa pokračovať ďalej. NIST odporúča použiť hodnoty, ktoré spĺňajú nasledovné podmienky:

- $n \geq MN$
- $N * (\min \pi_i) > 5$
- $\lambda = \frac{(M-m+1)}{2^m} \approx 2$
- $m \approx log_2 M$ [12]

Vstupom do funkcie je sekvencia náhodných bitov ε s dĺžkou *n*, a šablóna *B* s dĺžkou *m*. Test bude demonštrovaný na sekvencii $\varepsilon = 1011\ 1011\ 1100\ 1011\ 0110\ 0111\ 0010\ 1110\ 1111\ 1000\ 0101\ 1010\ 01$ s dĺžkou $n = 50$. [12]

Postup testovania podľa NIST:

1. Rozdelenie sekvencie do blokov kde $M = 10$ a $N = 5$.
   B1 = 1011 1011 11 *B3 = 0111 0010 11* *B5 = 0101 1010 01*
   B2 = 0010 1101 10 *B4 = 1011 1110 00*
2. Zvolí sa šablóna *B*, a bude sa skúmať koľko krát sa nachádza v danom bloku. Postupuje sa po okne ktoré ma rovnakú dĺžku ako šablóna a po každom kroku sa okno posunie o jeden bit. V tomto prípade bola zvolená šablóna $B = 11$ s dĺžkou $m = 2$.

Tabuľka 5: Zobrazenie rozdelenej sekvencie a hľadanie šablóny

Pozícia bitov v bloku	Blok 1		Blok 2		Blok 3		Blok 4		Blok 5	
	Bity	W_1	Bity	W_2	Bity	W_3	Bity	W_4	Bity	W_5
1-2	10	0	00	0	01	0	10	0	01	0
2-3	01	0	01	0	11	1	01	0	10	0
3-4	11	1	10	0	11	2	11	1	01	0

4-5	11	2	01	0	10	2	11	2	11	1
5-6	10	2	11	1	00	2	11	3	10	1
6-7	01	2	10	1	01	2	11	4	01	1
7-8	11	3	01	1	10	2	10	4	10	1
8-9	11	4	11	2	01	2	00	4	00	1
9-10	11	5	10	2	11	3	00	4	01	1

Výstupom z tohto kroku je, že prvý blok obsahuje šablónu 5 krát, druhý blok 2 krát, tretí blok 3 krát, štvrtý blok 4 krát a piaty blok raz. V súlade s týmto, premenné v_i budú vyzerať takto: $v_0 = 0, v_1 = 2, v_2 = 0, v_3 = 1, v_4 = 1, v_5 = 1$.

3. Výpočet hodnôt λ a η ktoré budú použité na dopočítavanie teoretických pravdepodobností π_i.

$$\lambda = \frac{(M - m + 1)}{2^m} = \frac{(10 - 2 + 1)}{2^3} = 2 \tag{36}$$

$$\eta = \frac{\lambda}{2} = 1{,}125 \tag{37}$$

4. Výpočet χ^2. Hodnoty π_i sú dané nasledovne:

$\pi_0 = 0{,}324652$ $\pi_2 = 0{,}142670$ $\pi_4 = 0{,}077147$

$\pi_1 = 0{,}182617$ $\pi_3 = 0{,}106645$ $\pi_5 = 0{,}166269$

$$\chi^2 = \sum_{i=0}^{5} \frac{(v_i - N\pi_i)^2}{N\pi_i} \tag{38}$$

$$\chi^2 = \frac{(0 - 5 * 0{,}324652)^2}{5 * 0{,}324652} + \frac{(2 - 5 * 0{,}182617)^2}{5 * 0{,}182617} + \frac{(0 - 5 * 0{,}142670)^2}{5 * 0{,}142670} + \frac{(1 - 5 * 0{,}106645)^2}{5 * 0{,}106645} + \frac{(1 - 5 * 0{,}077147)^2}{5 * 0{,}077147} + \frac{(1 - 5 * 0{,}166269)^2}{5 * 0{,}166269} = 5{,}05146 \tag{39}$$

5. Výpočet hodnoty P pomocou funkcie *igamc*.

$$P = igamc\left(\frac{N}{2}, \frac{\chi^2}{2}\right) = 0{,}40963 \tag{40}$$

6. Rozhodnutie, či sa jedná o náhodnú sekvenciu. Pokiaľ platí, že hodnota $P \geq 0{,}01$, tak testovaná sekvencia pomocou tohto testu sa dá považovať za náhodnú. Keďže testovaná sekvencia prešla testom s hodnotou $P = 0{,}40963$, dá sa o tejto sekvencii tvrdiť, že je podľa tohto testu náhodná. [12]

3.9 Maurerov univerzálny štatistický test

Maurerov univerzálny štatistický test sa zameriava na počet bitov medzi opakujúcimi sa vzormi. Test tiež odhalí, či je možne sekvenciu bitov výraznejšie skomprimovať bez straty informácie. Sekvenciu, ktorú možno výrazne skomprimovať bez straty informácie, nie je možné považovať za náhodnú. Ako vstup do testu poslúži sekvencia *ε*, dĺžka bloku *L*, počet inicializačných blokov Q a dĺžka bitovej sekvencie *n*. Organizácia NIST odporúča voliť vstupné hodnoty nasledovne:

- $n \geq L(Q + K)$
- $6 \leq L \leq 16$
- $Q = 10 * 2^L$
- $K = \left[\frac{n}{L}\right] - Q \approx 1000 * 2^L$ [12]

Hodnoty *L, Q* a *n* by mali byť vybrané podľa nasledovnej tabuľky:

Tabuľka 6: Hodnoty L a Q podľa dĺžky sekvencie

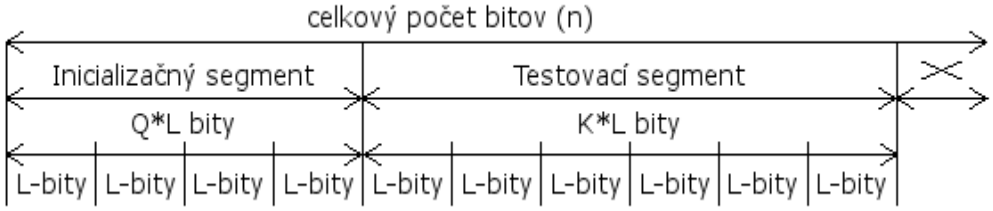
n	**L**	**Q = 10 * 2^L**
≥ 387 840	6	640
≥ 904 960	7	1280
≥ 2 068 480	8	2560
≥ 4 654 080	9	5120
≥ 10 342 400	10	10240
≥ 22 753 280	11	20480
≥ 49 643 520	12	40960
≥ 107 560 960	13	81920
≥ 231 669 760	14	163840
≥ 496 435 200	15	327680
≥ 1 059 061 760	16	655360

Postup testovania bude demonštrovaný na sekvencii bitov *ε = 01011010011101010111.* Dĺžka sekvencie je *n* = 20, dĺžka bloku bola zvolená *L* = 2 a inicializačný segment *Q* má dĺžku 4. [12]

Postup testovania podľa NIST:

1. Vstupná sekvencia *ε* o dĺžke *n* sa rozdelí na dva segmenty. Prvý segment pozostáva s *Q L*-bitových neprekrývajúcich sa blokov, druhý segment je nazývaný testovací segment a pozostáva s *K L*-bitových neprekrývajúcich sa blokov. Bity, ktoré sú na

konci a nepodarí sa ich použiť do blokov sú zmazané. Prvých *Q*-blokov je použitých na inicializáciu testu a ostatných *K* blokov je použitých na samotné testovanie. Obrázok nižšie zobrazuje rozdelenie sekvencie do blokov.



Obrázok 1: Rozdelenie sekvencie bitov

Zo zadaných hodnôt sa dá vypočítať dĺžka testovacieho segmentu takto:

$$K = \left[\frac{n}{L}\right] - Q = \left[\frac{20}{2}\right] - 4 = 6 \tag{41}$$

Následne je možné sekvenciu rozdeliť na inicializačný segment *01011010* a testovací segment: *011101010111*. Testovacia sekvencia teda vyzerá takto:

Tabuľka 7: Rozdelenie sekvencie bitov

Blok	Typ	Obsah
1	Inicializačný segment	01
2		01
3		10
4		10
5	Testovací segment	01
6		11
7		01
8		01
9		01
10		11

2. Pomocou inicializčného segmentu sa vytvorí tabuľka obsahujúca každú možnú kombináciu *L*-bitov. Následne sa do tabuľky uloží číslo bloku, v ktorom sa daná kombinácia nachádza.

Tabuľka 8: Inicializačný segment

	Možné L-bitové kombinácie			
	00 T_0	01 T_1	10 T_2	11 T_3
Inicializačný segment	0	2	4	0

3. Preskúmanie každého bloku v testovacom segmente určenie a počtu blokov od posledného výskytu rovnakého bloku. Hodnota v tabuľke sa nahradí polohou aktuálneho bloku, vypočíta sa vzdialenosť medzi znovu objavujúcimi sa blokmi, a tá sa použije ako argument pre log_2. Výsledná vzdialenosť sa pripočíta k ostatným.

Tabuľka 9: Testovací segment

Blok	L-bity	Blok posledného výskytu	Výpočet vzdialenosti	Kumulatívny súčet vzdialeností
5	01	2	$Log_2(5-2) = 1{,}584962501$	1,584962501
6	11	X	$Log_2(6-0) = 2{,}584962501$	4,169925002
7	01	5	$Log_2(7-5) = 1$	5,169925002
8	01	7	$Log_2(8-7) = 0$	5,169925002
9	01	8	$Log_2(9-8) = 0$	5,169925002
10	11	6	$Log_2(10-6) = 2$	7,169925002

Zmeny bitov sú dobre viditeľné aj v tejto tabuľke:

Tabuľka 10: Kombinácie bitov a ich obsadenie v sekvencii

Blok	Možné L-bitové kombinácie			
	00	01	10	11
Inicializačný segment	0	2	4	0
5	0	5	4	0
6	0	5	4	6
7	0	7	4	6
8	0	8	4	6
9	0	9	4	6
10	0	9	4	10

4. Vypočítanie štatistiky testovania pomocou dolu uvedenej rovnice. T_j znamená číslo posledného bloku obsahujúceho rovnakú hodnotu ako *i.* Suma v tejto rovnici reprezentuje kumulatívny súčet vzdialeností

$$f_n = \frac{1}{K}\sum_{i=Q+1}^{Q+K} log_2(i - T_j) = \frac{1}{6} * 7{,}16992502 = 1{,}1949875 \tag{42}$$

5. Vypočítanie hodnoty *P* pomocou funkcie *erfc*. NIST poskytuje pre tento krok tieto tabuľky s hodnotami, ktoré treba dosadiť do tejto funkcie.

Tabuľka 11: Dané očakávané hodnoty a odchýlky

L	Očakávaná hodnota	Odchýlka
2	1,5374383	1,338
6	5,2177052	2,954

L	Očakávaná hodnota	Odchýlka
11	10,170032	3,384
12	11,168765	3,401

7	6,1962507	3,125	13	12,168070	3,410
8	7,1836656	3,238	14	13,167693	3,416
9	8,1764248	3,311	15	14,167488	3,419
10	9,1723243	3,356	16	15,167379	3,421

$$c = 0{,}7 - \frac{0{,}8}{L} + \left(4 + \frac{32}{L}\right)\frac{K^{-\frac{3}{L}}}{15} \tag{43}$$

$$\sigma = c\sqrt{\frac{odchýlka(L)}{K}} \tag{44}$$

$$P = erfc\left(\left|\frac{f_n - OčakávanáHodnota(L)}{\sqrt{2}\sigma}\right|\right) \tag{45}$$

$$P = erfc\left(\left|\frac{1{,}1949875 - 1{,}5374383}{\sqrt{2} * 0{,}1845}\right|\right) = 0{,}06345 \tag{46}$$

6. Rozhodnutie, či sa jedná o náhodnú sekvenciu. Pokiaľ platí, že hodnota $P \geq 0{,}01$, tak testovaná sekvencia pomocou tohto testu sa dá považovať za náhodnú. Keďže testovaná sekvencia prešla testom s hodnotou $P = 0{,}06345$, dá sa o tejto sekvencii tvrdiť, že je podľa tohto testu náhodná. [12]

3.10 Test lineárnej komplexnosti

Tento test sa zameriava na dĺžku lineárneho spätného posuvného registru (linear feedback shift register - LFSR). Test má určiť, či je vygenerovaná sekvencia dostatočne komplexná na to, aby sa dala považovať za náhodnú. Náhodné sekvencie sú charakteristické dlhším LFSR. Aby výsledok testovania bol spoľahlivý, NIST odporúča voliť nasledovné vstupné hodnoty:

- dĺžka vstupnej sekvencie: $n \geq 10^6$
- dĺžka bitov v bloku: $500 \leq M \leq 5000$
- počet blokov $N \geq 200$ [12]

Postup testovania bude znázornený na bloku bitov *1101011110001, M = 13*:

1. Rozdelenie *n*-bitovej sekvencie na *N* nezávislých blokov o dĺžke *M* bitov
2. Použitím Berlenkamp-Masseyovho algoritmu sa určí lineárna komplexnosť L_i každého z *N* blokov *(i = 1,...,N)*. L_i reprezentuje dĺžka najkratšej LFSR sekvencie, ktorá generuje všetky bity v bloku *i*. V každej *Li* bitovej sekvencii sa nachádza kombinácia bitov, ktorej výsledok po sčítaní a následnom module 2 dá ďalší bit v sekvencii.

Tabuľka 12: Berlenkamp-Masseyov algoritmus

	Bit 1	Bit 2	Bit 3	Bit 4	**Bit 5**
Prvé 4 bity ktoré dajú bit č. 5	1	1	0	1	**0**
Bity 2-5 ktoré dajú bit č. 6	1	0	1	0	**1**
Bity 3-6 ktoré dajú bit č. 7	0	1	0	1	**1**
Bity 4-7 ktoré dajú bit č. 8	1	0	1	1	**1**
Bity 5-8 ktoré dajú bit č. 9	0	1	1	1	**1**
Bity 6-9 ktoré dajú bit č. 10	1	1	1	1	**0**
Bity 7-10 ktoré dajú bit č. 11	1	1	1	0	**0**
Bity 8-11 ktoré dajú bit č. 12	1	1	0	0	**0**
Bity 9-12 ktoré dajú bit č. 13	1	0	0	0	**1**

Pre túto testovanú sekvenciu platí, že *1.* a *2.* bit v štvorbitovej podsekvencii vytvoria piaty bit. Preto pre tento blok platí, že $L_i = 4$.

3. Výpočet teoretického priemeru μ

$$\mu = \frac{M}{2} + \frac{(9 + (-1)^{M+1})}{36} - \frac{(\frac{M}{3} + \frac{2}{9})}{2^M} \tag{47}$$

$$\mu = \frac{13}{2} + \frac{(9 + (-1)^{13+1})}{36} - \frac{\left(\frac{13}{3} + \frac{2}{9}\right)}{2^{13}} = 6{,}777222 \tag{48}$$

4. Výpočet hodnoty T_i pre každý blok

$$\mathrm{T}_i = (-1)^M * (L_i - \mu) + \frac{2}{9} = (-1)^{13} * (4 - 6{,}777222) + \frac{2}{9} = 2{,}999444 \tag{49}$$

5. Uloženie hodnôt T_i nasledovne:

Tabuľka 13: počítanie hodnôt *T*

$\mathrm{T}_i \leq -2{,}5$	$v_0 += 1$
$-2{,}5 < \mathrm{T}_i \leq -1{,}5$	$v_1 += 1$
$-1{,}5 < \mathrm{T}_i \leq -0{,}5$	$v_2 += 1$
$-0{,}5 < \mathrm{T}_i \leq 0{,}5$	$v_3 += 1$
$0{,}5 < \mathrm{T}_i \leq 1{,}5$	$v_4 += 1$
$1{,}5 < \mathrm{T}_i \leq 2{,}5$	$v_5 += 1$
$2{,}5 < \mathrm{T}_i$	$v_6 += 1$

6. Výpočet χ^2, kde hodnoty π_0 až π_6 sú dané nasledovne:

   $\pi_0 = 0{,}010417$ $\pi_3 = 0{,}5$ $\pi_6 = 0{,}020833$

   $\pi_1 = 0{,}03125$ $\pi_4 = 0{,}25$

   $\pi_2 = 0{,}125$ $\pi_5 = 0{,}0625$

$$\chi^2 = \sum_{i=0}^{K} \frac{(v_i - N\pi_i)^2}{N\pi_i} \tag{50}$$

7. Výpočet hodnoty *P* pomocou funkcie *igamc*

$$P = igamc\left(\frac{K}{2}, \frac{\chi^2}{2}\right) \tag{51}$$

8. Rozhodnutie, či sa jedná o náhodnú sekvenciu. Pokiaľ platí, že hodnota $P \geq 0{,}01$, tak testovaná sekvencia pomocou tohto testu sa dá považovať za náhodnú.[12]

3.11 Sériový test

Test sa sústreďuje na frekvenciu všetkých prekrývajúcich sa *m*-bitových vzorov naprieč celou sekvenciou bitov. Cieľom je určiť, či počet výskytov 2^m *m*-bitových prekrývajúcich sa vzorov je približne rovnaké ako pri pravej náhodnej sekvencii. Pri náhodných sekvenciách platí, že každý *m*-bitový vzor má rovnakú šancu na výskyt ako všetky ostatné *m*-bitové vzory. Pokiaľ platí, že $m = 1$, test degraduje na frekvenčný test popísaný ako druhý test z Diehard testov. Do testu vstupuje sekvencia pseudonáhodných bitov ε o dĺžke *n* a dĺžka blokov *m*. NIST odporúča voliť vstupnú sekvenciu a dĺžku blokov tak, aby platilo:

$$m < [log_2 n] - 2 \tag{52}$$

Test bude demonštrovaný na sekvencii $\varepsilon = 0011011101$ s dĺžkou $n = 10$, a dĺžka bloku je stanovená na $m = 3$. [12]

Postup testovania podľa organizácie NIST:

1. Vytvorenie rozšírenej sekvencie ε' tak, že na koniec sekvencie sa pridajú bity *m-1* z prvého bloku

$$\varepsilon = 0011011101 \tag{53}$$

$$m = 3: \varepsilon' = 001101110100 \tag{54}$$

$$m = 2: \varepsilon' = 00110111010 \tag{55}$$

$$m = 1: \varepsilon = 0011011101 - p\hat{o}vodn\acute{a}\ sekvencia \tag{56}$$

2. Určenie frekvencie všetkých možných prekrývajúcich sa *m*-bitových blokov, všetkých možných prekrývajúcich sa *m-1*-bitovýh blokov a všetkých možných prekrývajúcich sa *m-2*-bitových blokov.

Tabuľka 14: Frekvencia výskytov m-bitových blokov

m = 3		m = 2		m = 1	
$v_{i_1 \dots i_m}$	Počet výskytov	$v_{i_1 \dots i_{m-1}}$	Počet výskytov	$v_{i_1 \dots i_{m-2}}$	Počet výskytov
000	0	00	1	0	4
001	1	01	3	1	6
010	1	10	3		
011	2	11	3		
100	1				
101	2				
110	2				
111	1				

3. Výpočet funkcie ψ_m^2

$$\psi_m^2 = \frac{2^m}{n} \sum_{i_1 \dots i_m} \left(v_{i_1 \dots i_m} - \frac{n}{2^m} \right)^2 = \frac{2^m}{n} \sum_{i_1 \dots i_m} v_{i_1 \dots i_m}^2 - n \tag{57}$$

$$\psi_{m-1}^2 = \frac{2^{m-1}}{n} \sum_{i_1 \dots i_{m-1}} \left(v_{i_1 \dots i_{m-1}} - \frac{n}{2^{m-1}} \right)^2 = \frac{2^{m-1}}{n} \sum_{i_1 \dots i_{m-1}} v_{i_1 \dots i_{m-1}}^2 - n \tag{58}$$

$$\psi_{m-2}^2 = \frac{2^{m-2}}{n} \sum_{i_1 \dots i_{m-2}} \left(v_{i_1 \dots i_{m-2}} - \frac{n}{2^{m-2}} \right)^2 = \frac{2^{m-2}}{n} \sum_{i_1 \dots i_{m-2}} v_{i_1 \dots i_{m-2}}^2 - n \tag{59}$$

$$\psi_3^2 = \frac{2^3}{10}(0 + 1 + 1 + 4 + 1 + 4 + 4 + 1) - 10 = 2{,}8 \tag{60}$$

$$\psi_2^2 = \frac{2^2}{10}(1 + 9 + 9 + 9) - 10 = 1{,}2 \tag{61}$$

$$\psi_1^2 = \frac{2}{10}(16 + 36) - 10 = 0{,}4 \tag{62}$$

4. Výpočet $\nabla\psi_m^2$ a $\nabla^2\psi_m^2$

$$\nabla\psi_m^2 = \psi_m^2 - \psi_{m-1}^2 = \psi_3^2 - \psi_2^2 = 2{,}8 - 1{,}2 = 1{,}6 \tag{63}$$

$$\nabla^2\psi_m^2 = \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2 == \psi_3^2 - 2\psi_2^2 + \psi_1^2 = 2{,}8 - 2(1{,}2) + 0{,}4 = \tag{64}$$

5. Výpočet hodnoty *P* pomocou funkcie *igamc*.

$$P_1 = igamc\left(2^{m-2}, \frac{\nabla\psi_m^2}{2}\right) \qquad P_2 = igamc\left(2^{m-3}, \frac{\nabla^2\psi_m^2}{2}\right) \tag{65}$$

$$P_1 = igamc\left(2, \frac{1,6}{2}\right) \qquad P_2 = igamc\left(1, \frac{0,8}{2}\right) \tag{66}$$

$$P_1 = 0,8088 \qquad P_2 = 0,6703 \tag{67}$$

6. Rozhodnutie, či sa jedná o náhodnú sekvenciu. Pokiaľ platí, že hodnoty $P_x \geq 0,01$, tak testovaná sekvencia pomocou tohto testu sa dá považovať za náhodnú. Keďže testovaná sekvencia prešla testom s hodnotami $P_1 = 0,9057$ a $P_2 = 0,8805$, dá sa o tejto sekvencii tvrdiť, že je podľa tohto testu náhodná. [12]

3.12 Test približnej entropie

Podobne ako pri sériovom teste v predošlej kapitole, aj v tento test sa zameriava na frekvenciu všetkých možných prekrývajúcich sa *m*-bitových vzorov naprieč celou sekvenciou. Rozdiel je v tom, že tento test porovnáva frekvenciu prekrývajúcich sa blokov dvoch po sebe idúcich dĺžok (*m, m+1)*. Do testu vstupuje sekvencia pseudonáhodných bitov ε o dĺžke n a dĺžka blokov m. NIST odporúča voliť vstupnú sekvenciu a dĺžku blokov tak, aby platilo:

$$m < [log_2 n] - 5 \tag{68}$$

Test bude demonštrovaný na sekvencii $\varepsilon = 0100110101$ s dĺžkou $n = 10$ a dĺžkou bloku je stanovená $m = 3$. [12]

Postup testovania podľa organizácie NIST:

1. Vytvorenie rozšírenej sekvencie ε' tak, že na koniec sekvencie sa pridajú bity *m-1* z prvého bloku. Nová sekvencia vyzerá takto: $\varepsilon' = 010011010101$
2. Zistenie počtu vzorov v sekvencii pre $m = 3$

Tabuľka 15: Kombinácie bitov pre m = 3

#1	#2	#3	#4	#5	#6	#7	#8
000	001	010	011	100	101	110	111
0	1	3	1	1	3	1	0

3. Výpočet hodnoty C^m pre každú kombináciu

$$C_i^m = \frac{\#i}{n} \tag{69}$$

Tabuľka 16: Výpočet C pre m = 3

$C_{000}^3 = \frac{0}{10} = 0$	$C_{001}^3 = \frac{1}{10} = 0{,}1$	$C_{010}^3 = \frac{3}{10} = 0{,}3$	$C_{011}^3 = \frac{1}{10} = 0{,}1$
$C_{100}^3 = \frac{1}{10} = 0{,}1$	$C_{101}^3 = \frac{3}{10} = 0{,}3$	$C_{110}^3 = \frac{1}{10} = 0{,}1$	$C_{111}^3 = \frac{0}{10} = 0$

4. Výpočet hodnoty φ^m

$$\varphi^m = \sum_{i=0}^{2^{m-1}} \pi_i \log \pi_i \,;\; \pi_i = C_j^3 \; a \; j = log_2 i \tag{70}$$

$$\varphi^3 = 0(\log 0) + 0{,}1(\log 0{,}1) + 0{,}3(\log 0{,}3) + 0{,}1(\log 0{,}1) + 0{,}1(\log 0{,}1) + 0{,}3(\log 0{,}3) + 0{,}1(\log 0{,}1) + 0(\log 0) = -1{,}64341772 \tag{71}$$

5. Zopakovanie krokov 1-4 pre hodnoty *m+1* a sekvenciu $\varepsilon' = 0100110101010$

Tabuľka 17: Výpočet C pre m = 4

$C_{0000}^4 = 0$	$C_{0001}^4 = 0$	$C_{0010}^4 = 0$	$C_{0011}^3 = 0{,}1$
$C_{0100}^4 = 0{,}1$	$C_{0101}^4 = 0{,}2$	$C_{0110}^4 = 0{,}1$	$C_{0111}^3 = 0$
$C_{1000}^4 = 0$	$C_{1001}^4 = 0{,}1$	$C_{1010}^4 = 0{,}3$	$C_{1011}^3 = 0$
$C_{1100}^4 = 0$	$C_{1101}^4 = 0{,}1$	$C_{1110}^4 = 0$	$C_{1111}^3 = 0$

$$\varphi^4 = -1{,}83437197 \tag{72}$$

6. Výpočet χ^2

$$\chi^2 = 2n[\log 2 - ApEn(m)];\; ApEn(m) = \varphi^3 - \varphi^4 \tag{73}$$

$$\chi^2 = 2 * 10[\log 2 - 0{,}190954] = 0{,}502193 \tag{74}$$

7. Výpočet hodnoty *P* pomocou funkcie *igamc*

$$P = igamc\left(2^{m-1}, \frac{\chi^2}{2}\right) = igamc\left(2^2, \frac{0{,}502193}{2}\right) = 0{,}261961 \tag{75}$$

8. Rozhodnutie, či sa jedná o náhodnú sekvenciu. Pokiaľ platí, že hodnota $P \geq 0{,}01$, tak testovaná sekvencia pomocou tohto testu sa dá považovať za náhodnú. Keďže testovaná sekvencia prešla testom s hodnotou $P = 0{,}261961$, dá sa o tejto sekvencii tvrdiť, že je podľa tohto testu náhodná. [12]

3.13 Test kumulatívnych súčtov – cusum

Účelom tohto testu je určiť, či kumulatívny súčet čiastkových sekvencii nachádzajúcich sa v hlavnej sekvencii je priveľký alebo primalý. Pre náhodnú sekvenciu by mal byť tento súčet

blízky nule. NIST odporúča, aby vstupná sekvencia mala aspoň 100 bitov. Test bude demonštrovaný na sekvencii $\varepsilon = 1011010111$ s dĺžkou $n = 10$. [12]

Postup testovania podľa organizácie NIST:

1. Nahradenie logickej nuly *-1* a logickej jednotky *+1*

$$X = +1, -1, +1, +1, -1, +1, -1, +1, +1, +1 \tag{76}$$

2. Výpočet čiastkových súčtov S_i odpredu alebo odzadu

Tabuľka 18: Počítanie čiastkových súčtov

Dopredu	Dozadu
$S_1 = X_1$	$S_1 = X_n$
$S_2 = X_1 + X_2$	$S_2 = X_n + X_{n-1}$
$S_3 = X_1 + X_2 + X_3$	$S_3 = X_n + X_{n-1} + X_{n-2}$
...	...
$S_n = X_1 + X_2 + ... + X_n$	$S_n = X_n + X_{n-1} + ... + X_1$

$$\begin{aligned}
&S_1 = 1 \\
&S_2 = 1 + (-1) = 0 \\
&S_3 = 1 + (-1) + 1 = 1 \\
&S_4 = 1 + (-1) + 1 + 1 = 2 \\
&S_5 = 1 + (-1) + 1 + 1 + (-1) = 1 \\
&S_6 = 1 + (-1) + 1 + 1 + (-1) + 1 = 2 \\
&S_7 = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) = 1 \\
&S_8 = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 = 2 \\
&S_9 = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + 1 = 3 \\
&S_{10} = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + 1 + 1 = 4
\end{aligned} \tag{77}$$

3. Nájdenie maxima *z* z absolútnej hodnoty všetkých súčtov

$$z = \max_{1 \le k \le n} |S_k| = 4 \tag{78}$$

4. Výpočet hodnoty *P*

$$\begin{aligned}
P = 1 - &\sum_{k=\frac{\left(\frac{-n}{z}+1\right)}{4}}^{\frac{\left(\frac{n}{z}-1\right)}{4}} \left[\Phi\left(\frac{(4k+1)z}{\sqrt{n}}\right) - \Phi\left(\frac{(4k-1)z}{\sqrt{n}}\right)\right] \\
&+ \sum_{k=\frac{\left(\frac{-n}{z}-3\right)}{4}}^{\frac{\left(\frac{n}{z}-1\right)}{4}} \left[\Phi\left(\frac{(4k+3)z}{\sqrt{n}}\right) - \Phi\left(\frac{(4k+1)z}{\sqrt{n}}\right)\right]
\end{aligned} \tag{79}$$

$$P = 0{,}4116588 \tag{80}$$

5. Rozhodnutie, či sa jedná o náhodnú sekvenciu. Pokiaľ platí, že hodnota $P \geq 0{,}01$, tak testovaná sekvencia pomocou tohto testu sa dá považovať za náhodnú. Keďže testovaná sekvencia prešla testom s hodnotou $P = 0{,}4116588$, dá sa o tejto sekvencii tvrdiť, že je podľa tohto testu náhodná. [12]

3.14 Random Excurions Test

Tento test sa zameriava na počet jednotlivých stavov, ktoré sekvencia dosiahne pri počítaní kumulatívnych súčtov. Do testu vstupuje sekvencia ε s dĺžkou n. NIST odporúča, aby vstupná sekvencia mala aspoň 1 000 000 bitov. Test bude demonštrovaný na sekvencii $\varepsilon = 0110110101$ s dĺžkou $n = 10$. [12]

Postup testovania podľa organizácie NIST:

1. Nahradenie logickej nuly *-1* a logickej jednotky *+1*

$$X = -1, +1, +1, -1, +1, +1, -1, +1, -1, +1 \tag{81}$$

2. Výpočet kumulatívnych súčtov rovnako, ako v predchádzajúcom teste v kroku 2

$S_1 = -1$ $S_2 = 0$ $S_3 = 1$ $S_4 = 0$ $S_5 = 1$

$S_6 = 2$ $S_7 = 1$ $S_8 = 2$ $S_9 = 1$ $S_{10} = 2$

3. Vytvorenie sekvencie *S'* začínajúcu pridanou nulou, pokračujúcu $S_{1...n}$, a končiacu taktiež pridanou nulou.

$$S' = 0, -1,0,1,0,1,2,1,2,1,2,0 \tag{82}$$

4. Rozdelenie sekvencie *S'* na cykly začínajúce a končiace nulou. Číslo *J* bude vyjadrovať počet týchto cyklov.

$$J = 3 = \{0, -1,0\}, \{0,1,0\}, \{0,1,2,1,2,1,2,0\} \tag{83}$$

Pokiaľ je hodnota $J < 500$, NIST nariaďuje prerušiť test.

5. Výpočet frekvencii výskytov stavov $-4 \leq x \leq -1$ *a* $1 \leq x \leq 4$

Tabuľka 19: Výpočet frekvencii pre konkrétne hodnoty x

Stavy *x*	Cykly		
	(0, -1, 0)	(0, 1, 0)	(0, 1, 2, 1, 2, 1, 2, 0)
-4	0	0	0
-3	0	0	0
-2	0	0	0
-1	1	0	0

1	0	1	3
2	0	0	3
3	0	0	0
4	0	0	0

6. Výpočet $v_k(x)$, počet cyklov v ktorých sa stav x nachádza presne k krát. Pre $k > 5$, sa hodnota k upraví na $k = 5$.

Tabuľka 20: Výpočet hodnôt v

Stavy x	Počet výskytov v cykloch - k					
	0	1	2	3	4	5
-4	3	0	0	0	0	0
-3	3	0	0	0	0	0
-2	3	0	0	0	0	0
-1	2	1	0	0	0	0
1	1	1	0	1	0	0
2	2	0	0	1	0	0
3	3	0	0	0	0	0
4	3	0	0	0	0	0

Stav *-1* sa teda vyskytuje raz v jednom cykle, stav *1* sa vyskytuje raz v jednom cykle a tri krát v treťom cykle a stav *2* sa vyskytuje tri krát v treťom cykle. Ostatné stavy sa nevyskytujú v žiadnom z cyklov.

7. Pre každý z ôsmych stavov sa vypočíta χ^2

$$\chi^2 = \sum_{k=0}^{5} \frac{(v_k(x) - J\pi_k(x))^2}{J\pi_k(x)} \tag{84}$$

$\pi_k(x)$ je pravdepodobnosť, že sa stav x objaví k krát v náhodnom rozložení.

Pre $x = 1$:

$$\chi^2 = \frac{(1 - 3(0{,}5))^2}{3(0{,}5)} + \frac{(1 - 3(0{,}25))^2}{3(0{,}25)} + \frac{(0 - 3(0{,}125))^2}{3(0{,}125)} + \frac{(1 - 3(0{,}0625))}{3(0{,}0625)} \tag{85}$$
$$+ \frac{(1 - 3(0{,}0312))^2}{3(0{,}0312)} + \frac{(1 - 3(0{,}0312))^2}{3(0{,}0312)} = 4{,}333033$$

8. Výpočet hodnoty P pomocou funkcie igamc pre každý stav x.

$$P = igamc\left(\frac{5}{2}, \frac{\chi^2}{2}\right) \tag{86}$$

Pre $x = 1$:

$$P = igamc\left(\frac{5}{2}, \frac{4{,}333033}{2}\right) = 0{,}502529 \tag{87}$$

9. Rozhodnutie, či sa jedná o náhodnú sekvenciu. Pokiaľ platí, že hodnota $P \geq 0{,}01$, tak testovaná sekvencia pomocou tohto testu sa dá považovať za náhodnú. Keďže testovaná sekvencia prešla testom pre $x = 1$ s hodnotou $P = 0{,}502529$, dá sa o tejto sekvencii tvrdiť, že je pre $x = 1$ náhodná. [12]

3.15 Random Excursions Variant Test

Test sa zameriava na súčet stavov, kedy bol dosiahnutý určitý súčet pri kumulatívnom súčte. Výstupom tohto testu je osemnásť výstupov *(-9,...,-1 a 1,...,9)*, pomocou ktorých sa dá určiť náhodnosť sekvencie a to tak, že sa tieto výstupy blížia k číslam, ktoré sa predpokladajú pri náhodných sekvenciách. Do tohto testu vstupuje len sekvencia ε s dĺžkou n. NIST odporúča, aby táto sekvencia mala dĺžku aspoň 1 000 000 bitov. Test bude predvedený na sekvencii $\varepsilon = 0110110101$ s dĺžkou $n = 10$. [12]

Postup testovania podľa organizácie NIST:

1. Nahradenie logickej nuly *-1* a logickej jednotky *+1*

$$X = -1, +1, +1, -1, +1, +1, -1, +1, -1, +1 \tag{88}$$

2. Výpočet kumulatívnych súčtov rovnako, ako v predchádzajúcich dvoch testoch kroku 2

$S_1 = -1$ $S_2 = 0$ $S_3 = 1$ $S_4 = 0$ $S_5 = 1$

$S_6 = 2$ $S_7 = 1$ $S_8 = 2$ $S_9 = 1$ $S_{10} = 2$

3. Vytvorenie sekvencie *S'* začínajúcu pridanou nulou, pokračujúcu $S_{1...n}$, a končiacu taktiež pridanou nulou.

$$S' = 0, -1,0,1,0,1,2,1,2,1,2,0 \tag{89}$$

4. Pre každý z osemnástich nenulových stavov x sa vypočíta celkové množstvo rovnakých súčtov $\xi(x)$ naprieč všetkými J cyklami

   Počet cyklov $J = 3$. Číslo sa spočíta rovnako ako v prechádzajúcom teste.

   $\xi(-1) = 1$ $\xi(1) = 4$ $\xi(2) = 3$ *Ostatné stavy* $\xi(x) = 0$

5. Výpočet osemnástich P hodnôt pomocou funkcie *erfc*

   Pre $x = 1$:

$$P = erfc\left(\frac{|\xi(x) - J|}{\sqrt{2J(4|x| - 2)}}\right) = erfc\left(\frac{|4 - 3|}{\sqrt{2 * 3(4|1| - 2)}}\right) = 0{,}683091 \tag{89}$$

6. Rozhodnutie, či sa jedná o náhodnú sekvenciu. Pokiaľ platí, že hodnota $P \geq 0{,}01$, tak testovaná sekvencia pomocou tohto testu sa dá považovať za náhodnú. Keďže testovaná sekvencia prešla testom pre $x = 1$ s hodnotou $P = 0{,}683091$, dá sa o tejto sekvencii tvrdiť, že je pre $x = 1$ náhodná. [12]

II. PRAKTICKÁ ČÁST

4 VYTVORENIE PROGRAMU

Program na testovanie generátorov pseudonáhodných čísel som sa rozhodol naprogramovať v programovacom jazyku Python 3.6.4. Ako vývojové prostredie som si zvolil PyCharm 2017.3.3 vo verzii Community, ktorá je dostupná zadarmo pre plné použitie. Na inštaláciu externých knižníc som použil program Pip. Do každého z pätnástich testov bude vstupovať sekvencia bitov, ktoré budú na začiatku overené, či spĺňajú odporučenia od NISTu. Pokiaľ tieto odporúčania nespĺňajú, užívateľ bude upozornený, že výsledok testu môže byť skreslený. Tieto testy budú uložené v knižnici diehard.py.

4.1 Inštalácia voľne dostupných knižníc

Keďže každý z testov sa vyhodnocuje pomocou hodnoty *p*, ako prvé som sa rozhodol získať funkcie ktoré túto hodnotu vypočítajú. Tieto funkcie sú dve: *igamc* a *erfc*. Obidve tieto funkcie sa nachádzajú v knižnici *scipy.special* a do programu sa vložia pomocou príkazov:

- *from scipy.special import erfc*
- *from scipy.special import gammaincc as igamc*

Z knižnice *scipy.stats* bude potrebná funkcia *norm* a z knižnice *scipy.fftpack* funkcia *fft*, ktorá bude počítať fourierové transformácie. Ostatné matematické funkcie poskytne veľmi známa knižnica *numpy*. Program z používa tieto funkcie:

- from numpy import array – vytváranie polí
- from numpy import exp – počítane exponenciálnej funkcie
- from numpy import sqrt – počítanie odmocnín
- from numpy import log – počítanie prirodzeného logaritmu
- from numpy import where – výber elementov pomocou podmienky
- from numpy import log2 – výpočet logaritmu so základom 2
- from numpy import append – pridanie elementu na koniec zoznamu
- from numpy import transpose – transponovanie matíc
- from numpy import clip – hodnoty menšie ako zadané minimum v intervale sa upravia na minimálnu hodnotu, naopak hodnoty väčšie ako zadané maximum sa upravia na maximum v intervale
- from numpy import histogram – vytvorenie histogramu

4.2 Pomocné funkcie

Pri programovaní je dobrou praxou, že sa kód vykonávajúci tie isté operácie sa nahradí funkciou. Nielenže sa takto dá funkcionalita tohto bloku kódu zmeniť na jednom mieste, ale program potom býva podstatne prehľadnejší a jednoduchší na pochopenie. Inak to nie je ani v tomto prípade a väčšina opakujúceho sa kódu je nahradená funkciami. Takýchto funkcii som vytvoril 6 a nachádzajú sa vo zvlášť súbore *functions.py*. Funkcia na výpočet binárnej hodnosti matíc je vo zvlášť triede uloženej v súbore *binary_rank.py*.

4.2.1 Check_sequence

Do funkcie vstupuje jeden parameter, a to vstupná sekvencia bitov. Tá sa následne prechádza znak po znaku pomocou *for* cyklu. Ak sa prečíta znak ktorú je rôzny od *0* a od *1*, tak sa bit preskočí. Ak táto podmienka splnená nie je, prečítaný bit sa priradí do zoznamu nakoniec. Výstupom funkcie je zoznam číslic *0* alebo *1*.

4.2.2 Change_zeros_to_minus_ones

Táto funkcia je veľmi podobná predošlej. Vstupuje do nej len sekvencia ktorá sa spracováva po jednom čísle. Každé prečítané číslo sa vynásobí dvomi a následne sa z neho odčíta 1. Takto sa dá zo vstupnej sekvencie jednotiek a núl získať sekvencia jednotiek a mínus jednotiek. Po dokončení cyklu sa získaná sekvencia vráti.

4.2.3 To_blocks

Funkcia má dva vstupné parametre – sekvenciu a dĺžku jedného bloku. V prvom bloku sa celočíselným delením získa počet možných blokov. Sekvencia sa následne prechádza pomocou dvoch *for* cyklov. Vnútorný *for* cyklus sa opakuje tak dlho, až vznikne zoznam o požadovanej dĺžke bloku. Ten sa následne pripojí k ostatným blokom. Vonkajší *for* cyklus beží pokiaľ sa nedosiahne vypočítaný počet blokov. Časť sekvencie, ktorá je prikrátka na vytvorenie bloku sa stratí.

4.2.4 Linear complexity

Síce je táto funkcia potrebná len pri jednom teste, je elegantnejšie ju osamostatniť a následne volať. Do funkcie vstupuje len sekvencia, konkrétne blok jednotiek a núl. Najskôr sa vypočíta dĺžka tejto sekvencie, potom sa vytvoria dve jednorozmerné polia núl o dĺžke tejto sekvencie a prvý znak v týchto poliach sa zmení na jednotku. V ďalšom riadku sa premennej

lx, ktorá sa v poslednom kroku vráti ako výsledok, priradí nula, pomocným premenným *m* a *i* sa priradí *-1* a *1*. Nasleduje cyklus *while*, ktorý sa bude opakovať pokiaľ nebude splnená podmienka, že pomocná premenná *i* je menšia ako spočítaná dĺžka sekvencie. V tomto cykle sa následne počíta dĺžka lineárneho spätného posuvného registru (LSFR). Vypočítaná hodnota sa následne vráti.

4.2.5 Cumulative_sums

Táto funkcionalita je požadovaná niekoľkými testami. Do funkcie vstupuje zoznam jednotiek a mínus jednotiek a premenná, podľa ktorej sa uloží buď vypočítaná hodnota alebo absolútna vypočítaná hodnota. Zoznam sa prechádza po jednom prvku pomocou *for* cyklu. Tento prvok sa pripočíta k sume, ktorá je pred začatím cyklu rovná nule. Následne sa vypočítaná suma pripojí do zoznamu vypočítaných súm z predošlých cyklov vo forme, akú určila vstupná premenná *do_abs*.

4.2.6 To_overlapping_blocks

Do funkcie vstupujú dve premenné – vstupné sekvencia a premenná m ktorá reprezentuje veľkosť posuvného okna. Najskôr sa zistí sekvencia bitov zo začiatku sekvencie a tá sa priradí na koniec sekvencie. Následný *for* cyklus posúva okno po sekvencii a hodnoty z tohto okna ukladá do zoznamu. Keď sa cyklus dostane na koniec sekvencie a už nie je dostatok elementov v zozname na vytvorenie okna, cyklus skončí a vráti sa zoznam blokov.

4.2.7 Binary_matrix

Keďže NIST nikde neuvádza algoritmus, pomocou ktorého by sa mala táto hodnota počítať, tak som túto funkciu vyhľadal na githube, kde je voľne dostupná na použitie. Triedu vytvoril pred tromi rokmi používateľ StuartGordonReid a po otestovaní a následnom upravení niekoľkých drobností je vhodná na použitie v tomto programe. Trieda sa inicializuje pomocou matice a čísiel reprezentujúcich stĺpce a riadky. Po zavolaní funkcie *compute_ranks* sa z triedy vráti požadovaná správna hodnota.

5 IMPLEMENTÁCIA DIE HARD TESTOV

Pre testy som vytvoril súbor *diehard.py*, ktorý obsahuje triedu *DieHardTests*. Každý z testov vracia buď jednu hodnotu *p* alebo zoznam *p* hodnôt. Vypočítané medzivýsledky sa ukladajú len v logu, ktorý je používateľovi prístupný priamo v programe. Zapisovanie hodnôt do logu nebude pri testoch popisované, nakoľko sa nejedná o výpočtovú časť.

Prvou funkciou je predpísaná funkcia *__init__*, ktorá vytvorí objekt *diehard*. V tejto funkcii sa ako prvé overí, že v sekvencii sa nachádzajú len platné znaky. Následne sa vypočíta dĺžka sekvencie.

5.1 Frekvenčný test: Monobit

Po zavolaní tejto funkcie sa sekvencia pošle do funkcie *change_zeros_to_minus_ones*. Všetky elementy zoznamu upravenej sekvencie sa spočítajú pomocou funkcie *sum*, a výsledok sa uloží do premennej *sn*. Nasleduje výpočet premenne *sobs* a nakoniec hodnoty *p* pomocou funkcie *erfc*.

Zápis funkcie vyzerá nasledovne:

```
def monobit(self):
    s = change_zeros_to_minus_ones(self.e)
    sn = sum(s)
    sobs = abs(sn) / sqrt(self.n)
    p = erfc(sobs / sqrt(2))
    return p
```

Ako testovací vstup poslúži sekvencia *10011000* z príkladu k tejto funkcii.

Očakávaný výsledok testu tejto funkcie je vrátenie hodnoty približne $p = 0{,}4795$.

Výsledok testu dopadol nasledovne: $p = 0{,}4795001221869536$.

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.2 Frekvenčný test v rámci bloku

Do tejto funkcie vstupuje premenná *m*, ktorá reprezentuje dĺžku bloku. Najskôr sa inicializujú premenná *chisqr*, ktorá reprezentuje χ^2. Nasleduje rozdelenie sekvencie do blokov a pomocou *for* cyklu sa pre každý blok spočíta počet jednotiek. Pre každý blok sa taktiež

počíta hodnota χ^2 ktorá sa pripočíta k predošlej hodnote. Nakoniec sa vypočíta hodnota *p* ktorá sa vráti. Zápis funkcie vyzerá nasledovne:

```
def frequency_in_block(self, m):
    chisqr = 0
    blocks = to_blocks(self.e, m)
    for block in blocks:
        ones = sum(block)
        chisqr += 4 * m * ((int(ones) / m) - (1 / 2)) ** 2
    p = igamc(len(blocks) / 2, chisqr / 2)
    return p
```

Ako testovací vstup poslúži sekvencia *1001110101* a *m = 4* z príkladu k tejto funkcii.

Očakávaný výsledok testu tejto funkcie je vrátenie hodnoty približne *p = 0,606531*.

Výsledok testu dopadol nasledovne: *p = 0,606530659712633*.

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.3 Test postupností

Na začiatku funkcie sa premenným *ones* (počet jednotiek), *blocks* (počet blokov) a *previous_bit* (predchádzajúci znak) priradia konštanty. Premenná *previous_bit* má hodnotu *2* preto, lebo v zvalidovanej sekvencii sa nikdy takáto hodnota nachádzať nebude, a je tak zaručené, že sa po prečítaní prvého bitu vytvorí prvý blok. Sekvencia sa následne pomocou *for* cyklu prechádza po jednotlivom elemente, ktorý je skúmaný pomocou dvoch podmienok. Pokiaľ je skúmaný znak rozdielny ako predošlý, počet blokov sa zvýši o jedno. Pokiaľ je skúmaný znak rovný jednej, počet jednotiek sa zvýši o jedna. Nasleduje výpočet hodnoty π, ku ktorému dôjde len vtedy, ak je dĺžka sekvencie rôzna od nuly. Ak je π spočítané, vypočíta sa hodnota *p*, ktorá sa následne vráti. Zápis funkcie vyzerá nasledovne:

```
def runs(self):
    ones, blocks, previous_bit = 0, 0, 2
    for bit in self.e:
        if bit != previous_bit:
            blocks += 1
        if bit == 1:
            ones += 1
        previous_bit = bit
    try:
        pi = ones / self.n
```

```
    except ZeroDivisionError:
        return 'Zero Division Error'
    p = erfc((abs(blocks - 2 * self.n * pi * (1 - pi))) / (2 * sqrt(2 *
self.n) * pi * (1 - pi)))
    return p
```

Ako testovací vstup poslúži sekvencia *1001101011* z príkladu k tejto funkcii.

Očakávaný výsledok testu tejto funkcie je vypísanie hodnoty približne $p = 0,147232$

Výsledok testu dopadol nasledovne: $p = 0,1472322553636658$

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.4 Najdlhšia sekvencia jednotiek v bloku

Tento test má tri vstupné parametre: dĺžka bloku, stupeň voľnosti a zoznam hodnôt π. Na začiatku sa sekvencia rozdelí do blokov, nasledujú dva *for* cykly, výstupom ktorých je zoznam obsahujúci dĺžky najdlhších sekvencií jednotiek v jednotlivých blokoch. V ďalšom kroku sa pomocou funkcie *histogram* vytvorí zoznam obsahujúci počty najdlhších behov jednotiek. Nasleduje *for* cyklus, ktorý berie hodnoty zo zoznamov obsahujúcich hodnoty π a hodnoty počtov najdlhších sekvencii jednotiek v bloku. Po spočítaní hodnoty χ^2 sa vypočíta hodnota *p*, a tá sa nakoniec vráti.

Zápis funkcie vyzerá nasledovne:

```
def longest_run(self, m, k, pi):
    chisqr, max_runs = 0, []
    sequence_blocks = to_blocks(self.e, m)
    for block in sequence_blocks:
        runs, ones_in_block  = [], 0
        for bit in block:
            if bit == 1:
                ones_in_block += 1
            elif bit == 0:
                runs.append(ones_in_block)
                ones_in_block = 0
            runs.append(ones_in_block)
        max_runs.append(max(runs))
    if m == 8:
        v = histogram(max_runs, bins=[0, 1, 2, 3, 4, 9999999999])[0]
    elif m == 128:
        v = histogram(max_runs, bins=[0, 5, 6, 7, 8, 9, 9999999999])[0]
```

```
    else:
        v = histogram(max_runs, bins=[0, 11, 12, 13, 14, 15, 16,
9999999999])[0]
    for value_v, value_pi in zip(v, pi):
        chisqr += (((value_v - len(sequence_blocks) * value_pi) ** 2) /
(len(sequence_blocks) * value_pi))
    p = igamc(k / 2, chisqr / 2)
    return p
```

Ako testovací vstup poslúži sekvencia:
1100110001010101101100010011001111000000000001001001101010100010001001111010110100000001101011111001100111001101101100010110010.

Očakávaný výsledok testu tejto funkcie je: $p = 0{,}180609$

Výsledok testu dopadol nasledovne: $p = 0{,}18059797678555792$

Keďže dosiahnutý výsledok sa líši len o približne *0,000011*, môžem o tejto funkcii tvrdiť, že funguje správne. Chyba pravdepodobne nastala použitím jemne rozdielnych hodnôt π, keďže hodnoty *v* sú v testovanom prípade totožné s očakávanými hodnotami: $v_0 = 4$, $v_1 = 9$, $v_2 = 3$, $v_4 = 0$.

5.5 Test binárnej hodnoty matíc

Do testu vstupujú dva parametre *m* a *q* (riadky a stĺpce matice), ktoré sú štandardne rovné *32*. Funkcia potom rozdelí sekvenciu znakov na zoznamy o dĺžke *m* * *q* (štandardne je dĺžka *1024*). Pomocou *for* cyklu sa z každého z týchto zoznamov vytvorí pole (array), pomocou triedy *BinaryMatrix* a jej funkcie *compute_rank* sa vypočíta binárna hodnosť matice a tá sa podľa svojej veľkosti uloží na dané miesto v zozname *ranks*. Po dobehnutí cyklu a získaní všetkých binárnych hodností sa funkcia pokúsi vypočítať hodnotu χ^2. Ak náhodou zo vstupnej sekvencie nebolo možné vytvoriť ani jednu maticu, tak nie je možné ani spočítať hodnotu χ^2 a funkcia skončí. Ak sa χ^2 vypočítať podarí, vypočíta sa aj hodnota *p* ktorá sa v poslednom riadku funkcie vráti. Zápis funkcie vyzerá nasledovne:

```
def binary_matrix_rank_test(self, m=32, q=32):
    ranks = [0, 0, 0]
    list_matrices = to_blocks(self.e, m*q)
    for mx in list_matrices:
        matrix = array(to_blocks(mx, m))
        binary_rank = BinaryMatrix(matrix, q, m)
        rank = binary_rank.compute_rank()
        if rank == m:
```

```
            ranks[0] += 1
        elif rank == m - 1:
            ranks[1] += 1
        else:
            ranks[2] += 1
    try:
        chisqr = (((ranks[0] - 0.2888 * len(list_matrices)) ** 2) / (0.2888
* len(list_matrices))) + (((ranks[1] - 0.5776 * len(list_matrices)) ** 2) /
(0.5776 * len(list_matrices))) + (((ranks[2] - 0.1336 * len(list_matrices))
** 2) / (0.1336 * len(list_matrices)))
    except ZeroDivisionError:
        return'Zero Division Error'
    p = exp(-chisqr/2)
    return p
```

Ako testovací vstup poslúži sekvencia *01011001001010101101* z príkladu k tejto funkcii, *m* a *q* budú *3*.

Očakávaný výsledok testu tejto funkcie je: $p = 0{,}741948$

Výsledok testu dopadol nasledovne: $p = 0{,}7419477513283345$

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.6 Test pomocou diskrétnej fourierovej transformácie – Spektrálny test

Vstupom pri tejto funkcii je iba samotná sekvencia jednotiek a núl, ktorá sa hneď v prvom kroku zmení na sekvenciu jednotiek a mínus jednotiek. Funkcia sa ďalej pokúsi spočítať fourierovu transformáciu každého znaku v sekvencii. Sekvencia fourierových transformácii sa následne rozdelí na polovicu a spočíta sa hodnota *t*. V ďalších dvoch krokoch sa počítajú hodnoty *n0* a *n1*, z nich sa vypočíta hodnota *d* a z nej sa dostane konečná hodnota *p* ktorá sa vráti. Zápis tejto funkcie vyzerá nasledovne:

```
def spectral_test(self):
    s = change_zeros_to_minus_ones(self.e)
    try:
        ft = fft(s)
    except ValueError:
        return 'Value Error'
    half = abs(ft)[1:int(self.n // 2):]
    t = sqrt(log(1/0.05) * self.n)
    n1 = len(where(half < t)[0])
    n0 = 0.95 * self.n / 2
```

```
    d = (n1 - n0)/(sqrt(self.n * 0.95 * 0.05 / 4))
    p = erfc(abs(d) / sqrt(2))
    return p
```

Ako testovací vstup poslúži sekvencia:

1001001000011111101101010100010001000010110100011000010001101001100010011000110011000101000101110000

Očakávaný výsledok testu tejto funkcie je: $p = 0{,}168669$

Výsledok testu dopadol nasledovne: $p = 0{,}16866861888781504$

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.7 Non-overlapping template matching test

Do testu vstupuje niekoľko preddefinovaných premenných (sekvencia, *b, M, m, N*), ktoré sú rovnako označené aj v teoretickej časti tejto práce. Najskôr sa spočítajú premenné *avg* a *var*, následne sa program pokúsi rozdeliť sekvenciu do blokov s dĺžkou *M*. Test pokračuje *for* cyklom, v ktorom sa bloky pretypujú na reťazec, vypočíta sa počet výskytov šablóny *b* v týchto blokoch a hodnota sa uloží do zoznamu (premenná *counts*). Zároveň sa postupne vypočítava hodnota χ^2, ktorá je nutná pre vypočítanie hodnoty *p*. Táto hodnota sa v poslednom kroku vráti. Zápis tohto testu vyzerá nasledovne:

```
def non_overlapping_template_matching_test(self, b, M, m, N):
    chisqr = 0
    avg = (M - m + 1) / (2 ** m)
    var = M * ((1 / (2 ** m)) - ((2 * m - 1) / (2 ** (2 * m))))
    try:
        blocks = to_blocks(self.e, M)
    except ZeroDivisionError:
        return 'Zero Division Error'
    for block in blocks:
        block = ''.join(map(str, block[:]))
        counts = block.count(b)
        wj.append(counts)
        chisqr += ((counts - avg) ** 2) / var
    p = igamc(len(blocks) / 2, chisqr / 2)
    return p
```

Ako testovací vstup poslúži sekvencia *10100100101110010110* z príkladu k tejto funkcii, premenná *N* bude nastavená na *2*, premenná *M* na *10*, *m* bude 3 a *b* bude 001.

Očakávaný výsledok testu tejto funkcie je: $p = 0{,}344154$

Výsledok testu dopadol nasledovne: $p = 0{,}3441537868654125$

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.8 Overlapping template matching test

Rovnako ako v predchádzajúcom teste, aj do testu vstupuje niekoľko preddefinovaných premenných (sekvencia, *b, M, m, N*), Test začína tým, že sa vstupná sekvencia rozdelí na bloky o dĺžke *M*. Následne sa z týchto blokov vyberie prvých *N* ktoré sa budú testovať. Každý tento blok sa prevedie na reťazec, a potom sa v ňom hľadá šablóna *B* tak, že po každom výskyte sa zo začiatočného indexu skráteného bloku odoberie jeden znak zo začiatku. Ak sa šablóna v bloku nájde, premenná *count* sa zvýši o jedno, naopak, ak sa šablóna nenájde, cyklus *while* skončí. Po skončení *while* cyklu sa výsledok zapíše do zoznamu výsledkov, ktorý sa v ďalšom *for* cykle prehliada spolu s preddefinovanými hodnotami π. V tomto *for* cykle sa počíta hodnota χ^2, z ktorej sa nakoniec vypočíta hodnota *p*. Posledný príkaz v tomto teste hodnotu *p* vráti. Zápis tohto testu vyzerá nasledovne:

```
def overlapping_template_matching_test(self, m, B, M, N, pi):
    start, chisqr, v = 0, 0, [0] * 6
    blocks = to_blocks(self.e, M)[:N]
    for block in blocks:
        count = 0
        block = ''.join(map(str, block[:]))
        while True:
            start = block.find(B, start) + 1
            if start > 0:
                count += 1
            else:
                break
        if count < 5:
            v[count] += 1
        else:
            v[5] += 1
    for vi, pi in zip(v, pi):
        chisqr += ((vi - N * pi) ** 2)/(N * pi)
    p = igamc(5 / 2, chisqr / 2)
    return p
```

Ako testovací vstup poslúži sekvencia z príkladu k tejto funkcii:

10111011110010110100011100101110111110000101101001

Premenná *N* bude nastavená na *5*, premenná *M* na *10*, *m* bude *2* a *b* bude *11*.

Očakávaný výsledok testu tejto funkcie je: $p = 0,40963$

Výsledok testu dopadol nasledovne: $p = 0,40963260237780935$

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.9 Maurerov univerzálny štatistický test

Vstupné parametre tohto testu sú: sekvencia, *L, Q, expected* (očakávaná hodnota), *variance* a stupeň voľnosti *K*. Testovanie začína rozdelením sekvencie na bloky o dĺžke *L*, následne sa bloky rozdelia do testovacieho a inicializujúceho segmentu. Spočítajú sa všetky možné kombinácie bitov pri dĺžke *L* a uložia sa do zoznamu. Na ďalšom riadku sa vytvoria premenné a program pokračuje spracovávaním inicializačného segmentu pomocou *for* cyklov. Pokiaľ sa v inicializačných blokoch nájde nejaká kombinácia, uloží sa jej pozícia do zoznamu *last_occured*. Nasleduje vynulovanie premennej *count*, a vytvorenie ďalších premenných. Teraz sa bude prechádzať dlhší testovací segment rovnako ako predchádzajúci inicializačný. Pokiaľ sa vyskytne zhoda medzi možnou kombináciou a aktuálnym stavom, spočíta sa pomocou funkcie *log2* vzdialenosť a do zoznamu *last_occured* sa uloží číslo bloku. Po dobehnutí cyklov sa spočíta premenná *fn* a *c*, a po získaní týchto hodnôt sa vypočíta hodnota *sigma*. Test má teraz všetky hodnoty potrebné pre vypočítanie a vrátenie hodnoty *p*. Zápis tohto testu vyzerá nasledovne:

```
def maurers_universal_test(self, L, Q, expected, variance, K):
    blocks, combinations = to_blocks(self.e, L)
    init_segment, test_segment = blocks[:Q], blocks[Q:]
    combinations = [list(i) for i in product([0, 1], repeat=L)]
    last_occured, count, count_blocks = [0] * len(combinations), 0, 0
    for block in init_segment:
        count_blocks += 1
        for state in combinations:
            if block == state:
                last_occured[count] = count_blocks
                break
            count += 1
        count = 0
    count, count_blocks, state_no, distance = 0, Q, 0, 0
```

```
    for block in test_segment:
        count_blocks += 1
        for state in combinations:
            state_no += 1
            if block == state:
                distance += log2(count_blocks-last_occured[count])
                last_occured[count] = count_blocks
                break
            count += 1
        count = 0
    fn = 1 / K * distance
    c = 0.7 - (0.8 / L) + (4 + (32 / L)) * ((K ** (-3 / L)) / 15)
    sigma = c * sqrt(variance / K)
    p = erfc(abs((fn - expected)/(sqrt(2) * sigma)))
    return p
```

Ako testovací vstup poslúži sekvencia *01011010011101010111* z príkladu k tejto funkcii.

Premenná *L* bude nastavená na *2*, premenná *Q* na *4*, *expected* bude *1,1949875*, *variance* bude *1,338* a *K* bude *6*.

Očakávaný výsledok testu tejto funkcie je: $p = 0{,}06345$

Výsledok testu dopadol nasledovne: $p = 0{,}0634535024151588$

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.10 Test lineárnej komplexnosti

Do testu vstupujú parametre *M, N, K* a zoznam obsahujúci hodnoty π. V prvom kroku sa vypočíta premenná *avg*. Následne sa pomocou *for* cyklu vyjme blok, tomuto bloku sa vypočíta lineárna komplexnosť a do zoznamu *t* sa vloží vypočítaná hodnota zaokrúhlená na dve desatinné miesta. Na koniec cyklu sa začiatočná a konečná hodnota nasledujúceho cyklu zväčší o *M*. Po skončení cyklu sa pomocou funkcie *histogram* roztriedia výsledky v zozname *t* do zoznamu *vg* podľa nadefinovaných hodnôt. Nasleduje ďalší *for* cyklus, ktorý postupne spočíta hodnotu χ^2, po jeho skončení sa vypočíta a vráti hodnota *p*. Zápis tohto testu vyzerá nasledovne:

```
def linear_complexity_test(self, M, N, K, pi):
    avg = M / 2 + ((9 + (-1) ** (M + 1)) / 36) - (((M / 3) + (2 / 9)) / (2
** M))
    block_end, block_start, chisqr, t = M, 0, 0, []
```

```
    for i in range(N):
        block = (self.e[block_start:block_end])
        lcn = linear_complexity(block)
        t.append(float(format(((-1) ** 13) * (lcn - avg) + (2 / 9), '.2f')))
        block_start += M
        block_end += M
    vg = histogram(t, bins=[-9999999999, -2.5, -1.5, -0.5, 0.5, 1.5, 2.5,
9999999999])[0][::-1]
    for pii, v in zip(pi, vg):
        chisqr += ((v - N * pii) ** 2) / (N * pii)
    p = igamc(K / 2, chisqr / 2)
    return p
```

Ako testovací vstup poslúži sekvencia *1101011110001* z príkladu k tejto funkcii.

Premenná *M* bude nastavená na *13*, premenná *N* na *1*, *K* bude *6* a *pi* bude zoznam *[0.010417, 0.03125, 0.125, 0.5, 0.25, 0.0625, 0.020833]*.

Očakávaný výsledok testu tejto funkcie je: $p = 0{,}32085$

Výsledok testu dopadol nasledovne: $p = 0{,}32084719886213414$

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.11 Sériový test

Vstupné parametre tohto testu sú dva: sekvencia a dĺžka bloku. Celý test sa opakuje dva krát pomocou *for* cyklu. V tomto cykle sa najskôr priradí na koniec sekvencie vypočítaný počet bitov, následne sa získajú všetky kombinácie jednotiek a núl pre dané *m*. Ďalší krok rozdelí sekvenciu na bloky, potom sa vytvorí zoznam núl s rovnakou dĺžkou ako je dĺžka zoznamu s kombináciami a do tohto zoznamu sa postupne uloží počet výskytov rovnakých reťazcov pomocou *for* cyklu prechádzajúceho oba zoznamy zároveň. Teraz sa test pokúsi spočítať χ^2 pomocou ďalšieho for cyklu simulujúceho sumu, a keďže sa na konci sumy od tejto hodnoty odčítava dĺžka sekvencie, tak tento krok musí byť mimo cyklu. Vypočítaná hodnota χ^2 sa na konci hlavného cyklu pridá do zoznamu hodnôt nazvaného *allpsisqr*. Po skončení hlavného cyklu sú v teste dostupné všetky hodnoty na spočítanie premenných d_1 a d_2 predstavujúce $\nabla\psi_m^2$ a $\nabla^2\psi_m^2$. Z týchto hodnôt sa nakoniec vypočítajú a vrátia dve hodnoty *p*. Zápis tohto testu vyzerá nasledovne:

```
def serial_test(self, m):
    d, p, vs, allpsisqr = [], [], [], []
    for n_of_runs in range(m-2, m+1):
        ext_e = self.e + self.e[0:n_of_runs - 1:]
        combinations = [list(i) for i in product([0, 1], repeat=n_of_runs)]
        blocks = [ext_e[x:n_of_runs+x:] for x in range(self.n)]
        v = [0] * len(combinations)
        for block in blocks:
            for c, z in zip(combinations, range(0, len(v))):
                if block == c:
                    v[z] += 1
        vs.append(v)
        psisqr = 0
        try:
            for vi in v:
                psisqr += ((2 ** n_of_runs) / self.n) * (vi ** 2)
        except ZeroDivisionError:
            return ['Zero Division Error']
        psisqr -= self.n
        allpsisqr.append(float('{0:.5f}'.format(psisqr)))
    d.append(allpsisqr[2] - allpsisqr[1])
    d.append(allpsisqr[2] - 2 * allpsisqr[1] + allpsisqr[0])
    p.append(igamc((2**(m-2)), d[0]/2))
    p.append(igamc((2**(m-3)), d[1]/2))
    return p
```

Ako testovací vstup poslúži sekvencia *0011011101* z príkladu k tejto funkcii.

Premenná *m* bude nastavená na *3*.

Očakávaný výsledok testu tejto funkcie je*:* $p_1 = 0{,}8088$ a $p_2 = 0{,}6703$

Výsledok testu dopadol nasledovne*:* $p_1 = 0{,}8087921354109989$ a $p_2 = 0{,}6703200460356394$

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.12 Test približnej entropie

Aj to tohto testu, rovnako ako v predošlom, vstupuje sekvencia a dĺžka bloku. Podobné je tiež to, že veľká časť testu sa bude opakovať, preto sa nachádza vo *for* cykle, ktorý sa bude opakovať dva krát pre *m* a pre *m + 1*. Na začiatok sa sekvencia rozdelí do blokov a potom sa vytvorí zoznam možných kombinácii jednotiek a núl s potrebnou dĺžkou. Nasleduje inicializácia alebo vynulovanie premenných (podľa toho či je cyklus v prvom alebo v druhom behu). Po inicializácii sa pokračuje ďalším *for* cyklom, ktorý pôjde blok po bloku, v ňom je

vnorený ďalší *for* cyklus, ktorý ide po zozname blokov a kombinácii. V tomto cykle je podmienka, že ak sa nájde blok s rovnakým obsahom ako je aktuálna kombinácia, tak sa pozícia *p* v zozname *v* zvýši o *1*. Po skončení týchto dvoch cyklov sa test pokúsi vypočítať φ. Na konci hlavného cyklu sa hodnota φ priradí do zoznamu *both_phis*. Po získaní oboch φ je možné vypočítať premennú *apen*, pomocou nej potom χ^2 a nakoniec hodnotu *p*. Tá sa v poslednom kroku vráti. Zápis tohto testu vyzerá nasledovne:

```
def approximate_entropy_test(self, m):
    both_phis = []
    for i in range(m, m + 2):
        blocks = to_overlapping_blocks(self.e, i)
        combinations = [list(i) for i in product([0, 1], repeat=i)]
        cs, phi, v = [], 0,  [0] * len(combinations)
        for block in blocks:
            for c, p in zip(combinations, range(0, len(v))):
                if block == c:
                    v[p] += 1
        try:
            for vi in v:
                c = vi/self.n
                if c != 0:
                    phi += c * log(c)
        except ZeroDivisionError:
            return 'Zero Division Error'
        both_phis.append(phi)
    apen = both_phis[0] - both_phis[1]
    chisqr = 2 * self.n * (log(2) - apen)
    p = igamc(2 ** (m - 1), chisqr / 2)
    return p
```

Ako testovací vstup poslúži sekvencia *0100110101* z príkladu k tejto funkcii.

Premenná *m* bude nastavená na 3.

Očakávaný výsledok testu tejto funkcie je: $p = 0{,}261961$

Výsledok testu dopadol nasledovne: $p = 0{,}2619611048816654$

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.13 Test kumulatívnych súčtov – cusum

Okrem sekvencie do tohto testu vstupuje aj smer počítania kumulatívnych súčtov. Na začiatok sa pomocou funkcie prevedú všetky nuly na mínus jednotky. Pokiaľ chce užívateľ počítať kumulatívne súčty odzadu, sekvencia sa zrkadlovo otočí. Zo sekvencie test spočíta pomocou funkcie absolútne kumulatívne súčty a pokúsi sa vybrať maximum. Toto maximum sa uloží do premennej *z*. Nasleduje vypočítanie začiatku a konca sumy prvej časti hodnoty *p*. Následne sa spočíta suma pomocou *for* cyklu, ktorý beží od vypočítaného začiatku po vypočítaný koniec. Keďže druhá časť výpočtu hodnoty *p* má iný začiatok, tak sa spočíta znovu a zopakuje sa *for* cyklus počítajúci sumu. Po získaní oboch častí sa postupuje podľa vzorca a to tak, že sa od jednotky odpočíta súčet dvoch vypočítaných súm. Výsledná hodnota je konečná hodnota *p*, ktorá sa v poslednom riadku testu vráti. Zápis tohto testu vyzerá nasledovne:

```
def cusum(self, direction='F'):
    p_part1, p_part2 = 0, 0
    e = change_zeros_to_minus_ones(self.e)
    if direction != 'F':
        e = e[::-1]
    sums = cumulative_sums(e, 'Y')
    try:
        z = max(sums)
    except ValueError:
        return 'Value Error'
    start = int(round((-self.n / z) + 1) / 4)
    stop = int(round((self.n / z) - 1) / 4)
    for k in range(start, stop + 1):
        p_part1 += norm.cdf(((4 * k + 1) * z)/(sqrt(self.n))) - norm.cdf(((4
* k - 1) * z)/(sqrt(self.n)))
    start = int(round((-self.n / z - 3) / 4))
    for k in range(start, stop + 1):
        p_part2 += norm.cdf(((4 * k + 3) * z)/(sqrt(self.n))) - norm.cdf(((4
* k + 1) * z)/(sqrt(self.n)))
    p = 1 - p_part1 + p_part2
    return p
```

Ako testovací vstup poslúži sekvencia:

1100100100001111110110101010001000100001011010001100001000110100110001001
1000110011000101000101110000

Očakávaný výsledok testu tejto funkcie pri počítaní spredu je: *p = 0,219194*

Očakávaný výsledok testu tejto funkcie pri počítaní zozadu je: $p = 0,114866$

Výsledok testu spredu dopadol nasledovne: $p = 0,21919399348562665$

Výsledok testu zozadu dopadol nasledovne: $p = 0,1148662153025217$

Keďže dosiahnuté výsledky sú rovnaké ako očakávané, môžem o tejto funkcii tvrdiť, že funguje správne.

5.14 Random Excurions Test

Vstupné parametre tohto testu sú hodnoty π a hodnoty x. Na začiatok sa inicializujú premenné a potom sa pomocou funkcie zmenia všetky nuly vo vstupnej sekvencii na jednotky. Teraz sa spočítajú kumulatívne súčty a na začiatok aj koniec zoznamu týchto súčtov sa pridá *0*. Ďalší krok má za úlohu zistiť pozície núl. Pomocou tohto zoznamu núl sa zoznam kumulatívnych súčtov rozdelí do blokov tak, že každý blok je ohraničený nulou na začiatku aj konci. Po dokončení tohto delenia sa spočíta počet blokov a uloží sa do premennej *j*. Nasledujúci *for* cyklus, ktorý prechádza blok po bloku, uloží do tabuľky stavov (premenná *state_table*) počet navštívení konkrétneho stavu. Po skončení cyklu sa táto tabuľka transponuje. Ďalší *for* cyklus vytvorí tabuľku, ktorá ma v sebe uložené údaje o tom, v koľkých cykloch sa daný stav nachádza. Na konci cyklu sa aj táto tabuľka transponuje. Nasledujúci *for* cyklus spočíta z tejto tabuľky 8 hodnôt χ^2 a pre každú spočíta hodnotu p. Tento zoznam p hodnôt sa v poslednom kroku vráti. Zápis tohto testu vyzerá nasledovne:

```
def random_excursions_test(self, x_values, pi_values):
    e = change_zeros_to_minus_ones(self.e)
    p_values, state_table, occurred_in_cycles, chisqr, rows, cols = [], [],
[], 0, 0, 0
    sums = cumulative_sums(e)
    sums = append(0, sums)
    sums = append(sums, 0)
    zero_places = where(sums == 0)[0]
    blocks = ([sums[zero_places[x]:zero_places[x + 1] + 1] for x in
range(len(zero_places) - 1)])
    j = len(blocks)
    for block in blocks:
        state_table.append(([len(where(block == x)[0]) for x in x_values]))
    state_table = transpose(clip(state_table, 0, 5))
    for i in range(6):
        occurred_in_cycles.append([(x == i).sum() for x in state_table])
    occurred_in_cycles = transpose(occurred_in_cycles)
    for row in occurred_in_cycles:
        for value in row:
```

```
            chisqr += ((value - (j * pi_values[rows][cols])) ** 2)/(j *
(pi_values[rows][cols]))
            cols += 1
        p = igamc(5 / 2, chisqr/2)
        p_values.append(p)
        rows += 1
        cols = 0
    return p_values
```

Ako testovací vstup poslúži sekvencia *0110110101* z príkladu k tejto funkcii.

Očakávaný výsledok testu tejto funkcie pre $x = 1$ je: $p = 0{,}502529$

Výsledok testu pre $x = 1$ dopadol nasledovne: $p = 0{,}502528742447265$

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.15 Random Excursions Variant Test

Vstupná sekvencia sa prevedie na sekvenciu jednotiek a mínus jednotiek, rovnako ako v predchádzajúcom teste sa spočítajú kumulatívne súčty a na začiatok a koniec zoznamu týchto súčtov sa pridá nula. Test ďalej zistí miesta, kde je kumulatívny súčet rovný *0* a podľa týchto miest sekvenciu rozdelí na bloky rovnako, ako v prechádzajúcom teste. Do premennej *j* sa uloží počet týchto blokov. Dvojica *for* cyklov prechádza všetky vypočítané kumulatívne súčty a všetky skúmané hodnoty *x*. Ak nastane zhoda, počet výskytov sa na konkrétnej pozícii sa zvýši o jedno. Po skončení cyklov sa odstráni zo zoznamu výskytov hodnota na pozícii *9* (táto hodnota odpovedá hodnote *x* rovnej *0* ktorá sa pri teste neskúma). Posledný *for* cyklus vypočíta hodnoty *p* pre každú hodnotu výskytov. Tieto hodnoty sa nakoniec vrátia. Zápis tohto testu vyzerá nasledovne:

```
def random_excursions_variant_test(self, x_values):
    e = change_zeros_to_minus_ones(self.e)
    sums = cumulative_sums(e)
    sums = append(0, sums)
    sums = append(sums, 0)
    zero_places = where(sums == 0)[0]
    blocks = ([sums[zero_places[x]:zero_places[x + 1] + 1] for x in
range(len(zero_places) - 1)])
    j = len(blocks)
    counts = 19 * [0]
    for cs in sums:
        for x in x_values:
```

```
            if cs == x:
                counts[x+9] += 1
    p_values = []
    del counts[9]
    for count, x in zip(counts, x_values):
        p = erfc((abs(count-j))/(sqrt(2*j*(4*abs(x)-2))))
        p_values.append(p)
    return p_values
```

Ako testovací vstup poslúži sekvencia *0110110101* z príkladu k tejto funkcii.

Očakávaný výsledok testu tejto funkcie pre $x = 1$ je: $p = 0{,}683091$

Výsledok testu pre $x = 1$ dopadol nasledovne: $p = 0{,}6830913983096087$

Keďže dosiahnutý výsledok je rovnaký ako očakávaný, môžem o tejto funkcii tvrdiť, že funguje správne.

5.16 Podmienky pre dôveryhodný výsledok

Organizácia NIST ku každému testu priložila niektoré vstupné hodnoty a podmienky, ktoré je nutné dodržať aby malo testovanie dôveryhodný výsledok. Všetky tieto podmienky sú uložené v súbore *test_boundaries.py* v triede *Boundaries*. Tieto podmienky pracujú iba s dĺžkou sekvencie a podľa nej vrátia zoznam hodnôt. Ak je podmienka splnená, na prvom mieste v zozname je reťazec *„OK“* a nasledujú vstupné hodnoty pre testovanie sekvencie, pokiaľ je sekvencia prikrátka, na prvom a jedinom mieste je reťazec *„Sequence Too Short“*.

Pokiaľ užívateľ použije ako vstupný súbor sekvenciu s dĺžkou sekvencie nula, overovanie hodnôt bude vracať rôzne chyby. Toto by sa dialo preto, lebo dĺžka sekvencie sa používa ako menovateľ pri delení pri rôznych testoch. Z tohto dôvodu je v inicializácii triedy podmienka, že ak dĺžka sekvencie je rovná nule, tak sa do premennej dĺžka sekvencie uloží číslo jedna. Takto overenie testov bude fungovať aj pre takýto neočakávaný vstup.

Overenie, či test vráti dôveryhodný výsledok pre konkrétny test vyzerá nasledovne:

- Monobit: dĺžka sekvencie musí byť väčšia alebo rovná ako 100

```
def monobit(self):
    if self.length >= 100:
        return ['OK']
    else:
        return ['Sequence Too Short']
```

- Frekvenčný test v rámci bloku: dĺžka sekvencie musí byť väčšia alebo rovná ako 100, zároveň sa vypočíta vhodná hodnota m.

```
def freq(self):
    if self.length >= 100:
        M = (0.01 * self.length)
        if M < 20:
            M = 20
        N = self.length // M
        while N >= 100:
            M += 2
            N = self.length // M
        return ['OK', int(M)]
    else:
        return ['Sequence Too Short']
```

- Test postupností: dĺžka sekvencie musí byť väčšia alebo rovná ako 100

```
def runs(self):
    if self.length >= 100:
        return ['OK']
    else:
        return ['Sequence Too Short']
```

- Najdlhšia sekvencia jednotiek v bloku: sekvencia musí byť dlhá aspoň 128 znakov, zároveň sa podľa dĺžky sekvencie vrátia hodnoty *M, K* a zoznam pravdepodobností π.

```
def long_run_ones(self):
    if self.length < 128:
        return ['Sequence Too Short']
    elif self.length < 6272:
        M, K, pi = 8, 3, [0.2148, 0.3672, 0.2305, 0.1875]
        return ['OK', M, K, pi]
    elif self.length < 750000:
        M, K, pi = 128, 5, [0.1174, 0.2430, 0.2493, 0.1752, 0.1027,
0.1124]
        return ['OK', M, K, pi]
    else:
        M, K, pi = 10000, 6, [0.0882, 0.2092, 0.2483, 0.1933, 0.1208,
0.0675, 0.0727]
        return ['OK', M, K, pi]
```

- Test binárnej hodnoty matíc: dĺžka sekvencie musí byť tak dlhá, aby sa dalo zostaviť 38 štvorcových matíc o dĺžke strany 32, teda aspoň *38912* znakov.

```
def binary_rank(self):
    if self.length < 38*32*32:
        return ['Sequence Too Short']
    else:
        return ['OK']
```

- Test pomocou diskrétnej fourierovej transformácie: dĺžka sekvencie musí byť väčšia alebo rovná ako 1000

```
def fourier_trans(self):
    if self.length >= 1000:
        return ['OK']
    else:
        return ['Sequence Too Short']
```

- Non-overalpping template matching test: šablóna *B* je daná (je možné ju zmeniť v tomto súbore na inú), vypočíta sa jej dĺžka, ktorá je pri nezmenení šablóny *9*. Počet blokov *N* je tiež daný na *8*. Následne sa vypočíta dĺžka bloku pomocou celočíselného delenia dĺžky sekvencie počtom blokov. Pokiaľ je dĺžka šablóny väčšia ako dĺžka bloku, sekvencia je prikrátka.

```
def non_overlapping(self):
    B = '000000001'
    m = len(B)
    n = self.length
    N = 8
    M = n // N
    if m > M:
        return ['Sequence Too Short']
    else:
        return ['OK', B, M, m, N]
```

- Overlapping template matching test: šablóna *B* je daná (je možné ju zmeniť v tomto súbore na inú), vypočíta sa jej dĺžka, ktorá je pri nezmenení šablóny *9*. Počet blokov ako aj dĺžka bloku je taktiež daná. Pokiaľ je sekvencia kratšia, ako *998976 (M * N)* znakov, test nevráti spoľahlivý výsledok.
-

```
def overlapping(self):
    B = '111111111'
    m = len(B)
    M, N = 1032, 968
    if self.length < M * N:
        return ['Sequence Too Short']
    else:
        return ['OK', m, B, M, N]
```

- Maurerov univerzálny štatistický test: dĺžka sekvencie musí byť aspoň *387840* znakov, ak je táto podmienka splnená, premenným *L, Q, expected a variance* sa priradia odpovedajúce hodnoty a nakoniec sa vypočíta hodnota *K*. V poslednom riadku funkcie sa zoznam hodnôt vráti.

```
def maurer(self):
    if self.length < 387840:
        return ['Sequence Too Short']
    elif self.length < 904960:
        L, Q, expected, variance = 6, 640, 5.2177052, 2.954
    elif self.length < 2068480:
        L, Q, expected, variance = 7, 1280, 6.1962507, 3.125
    elif self.length < 4654080:
        L, Q, expected, variance = 8, 2560, 7.1836656, 3.238
    elif self.length < 10342400:
        L, Q, expected, variance = 9, 5120, 8.1764248, 3.311
    elif self.length < 22753280:
        L, Q, expected, variance = 10, 10240, 9.1723243, 3.356
    elif self.length < 49643520:
        L, Q, expected, variance = 11, 20480, 10.170032, 3.384
    elif self.length < 107560960:
        L, Q, expected, variance = 12, 40960, 11.168765, 3.401
    elif self.length < 231669760:
        L, Q, expected, variance = 13, 81920, 12.168070, 3.410
    elif self.length < 496435200:
        L, Q, expected, variance = 14, 163840, 13.167693, 3.416
    elif self.length < 1059061760:
        L, Q, expected, variance = 15, 327680, 14.167488, 3.419
    else:
        L, Q, expected, variance = 16, 655360, 15.167379, 3.421
    K = (self.length / L) - Q
    return ['OK', L, Q, expected, variance, int(K)]
```

- Test lineárnej komplexnosti: dĺžka sekvencie musí byť aspoň jeden milión bitov, hodnoty *M, K*, a zoznam pravdepodobností π sú dané.

```
def lct(self):
    pi = [0.01047, 0.03125, 0.125, 0.5, 0.25, 0.0625, 0.020833]
    K = 6
    M = 500
    if self.length < 1000000:
        return ['Sequence Too Short']
    else:
        return ['OK', M, K, pi]
```

- Sériový test: dĺžka sekvencie musí byť väčšia alebo rovná ako 100, zároveň hodnotu *m*, vyjadrujúcu dĺžku bloku, som stanovil ako *8*. Je možné použiť aj vyššie hodnoty, avšak pre hodnotu *16* program kvôli obrovskému počtu kombinácii počítal výsledok približne 10 hodín.

```
def serial(self):
    if self.length >= 100:
        m = 8
        return ['OK', m]
    else:
        return ['Sequence Too Short']
```

- Test približnej entropie: dĺžka sekvencie musí byť väčšia alebo rovná ako 100, zároveň hodnotu m som stanovil ako *10*.

```
def aet(self):
    if self.length >= 100:
        m = 10
        return ['OK', m]
    else:
        return ['Sequence Too Short']
```

- Test kumulatívnych súčtov – cusum: dĺžka sekvencie musí byť väčšia alebo rovná ako 100.

```
def cusum(self):
    if self.length >= 100:
        return ['OK']
    else:
        return ['Sequence Too Short']
```

- Random excursions test: dĺžka sekvencie musí byť aspoň jeden milión bitov, zároveň sa vracia zoznam hodnôt *x*, pre ktoré sa počíta hodnota *p*.

```
def ret(self):
    x_values = [-4, -3, -2, -1, 1, 2, 3, 4]
    if self.length >= 1000000:
        return ['OK', x_values]
    else:
        return ['Sequence Too Short']
```

- Random excursion variant test: dĺžka sekvencie musí byť aspoň jeden milión bitov, zároveň sa vracia zoznam hodnôt *x*, pre ktoré sa počíta hodnota *p*.

```
def retv(self):
    x_values = [-9, -8, -7, -6, -5, -4, -3, -2, -1, 1, 2, 3, 4, 5, 6,
7, 8, 9]
    if self.length >= 1000000:
        return ['OK', x_values]
    else:
        return ['Sequence Too Short']
```

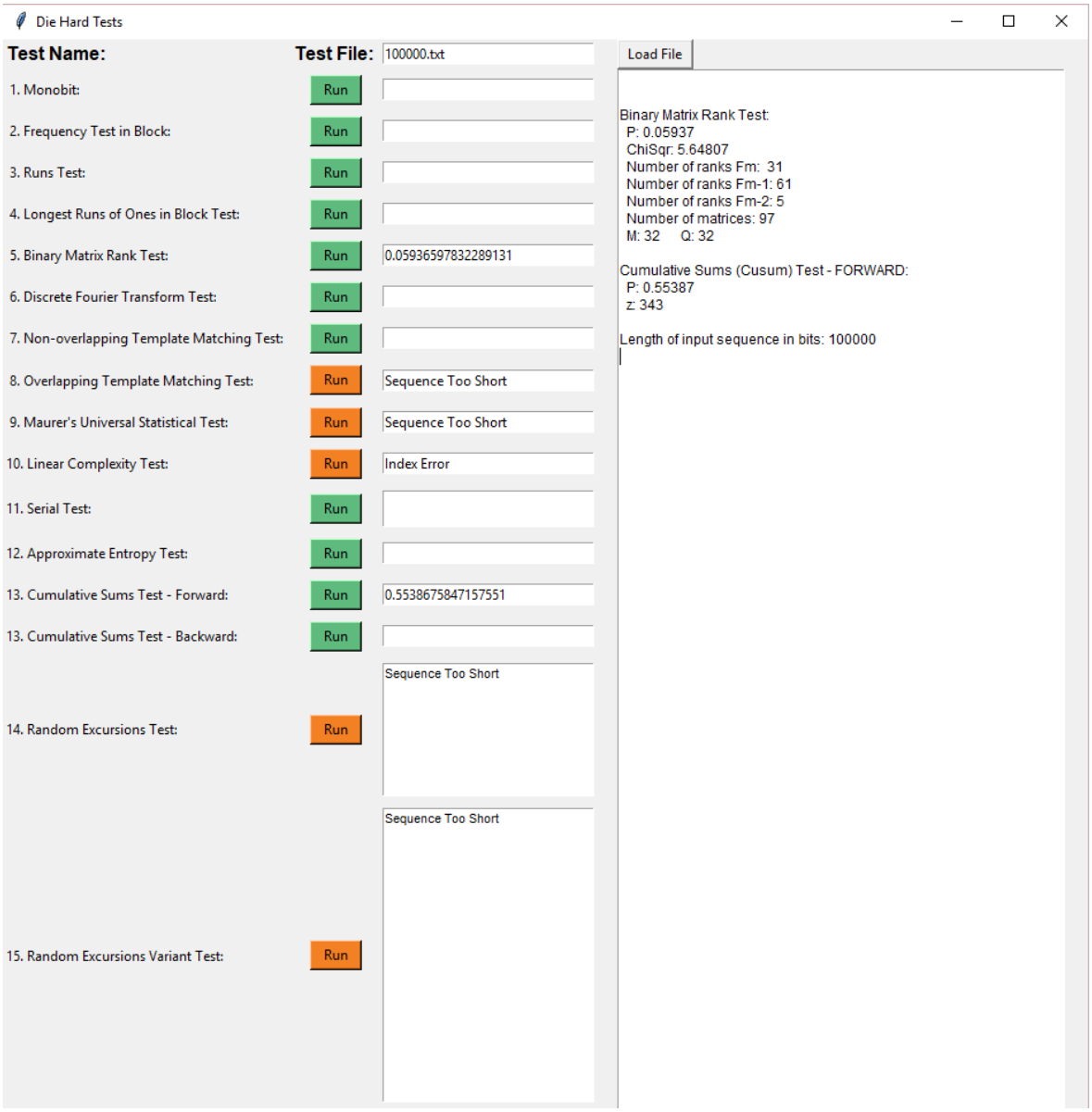
- Na konci tejto triedy sa nachádza funkcia *return_result*, ktorá zavolá všetky overovacie funkcie, ich výsledok priradí na koniec zoznamu a ten po získaní všetkých výsledkov vráti.

```
def return_result(self):
    self.result = []
    self.result.append(self.monobit())
    self.result.append(self.freq())
    self.result.append(self.runs())
    self.result.append(self.long_run_ones())
    self.result.append(self.binary_rank())
    self.result.append(self.fourier_trans())
    self.result.append(self.non_overlapping())
    self.result.append(self.overlapping())
    self.result.append(self.maurer())
    self.result.append(self.lct())
    self.result.append(self.serial())
    self.result.append(self.aet())
    self.result.append(self.cusum())
    self.result.append(self.ret())
    self.result.append(self.retv())
    return self.result
```

5.17 Užívateľské rozhranie

Aby bol tento program jednoduchý na použitie, vytvoril som jednoduché grafické rozhranie ktoré obsahuje 16 testov (test cusum je obsiahnutý dvakrát – raz sa kumulatívne súčty počítajú spredu a druhý krát zozadu). Na začiatku je potrebné načítať sekvenciu jednotiek a núl v textovej podobe. Po zadaní adresy súboru a následnom stlačení tlačidla *Load File* sa sekvencia načíta a zistí sa, ktoré testy je možné spustiť. Pokiaľ sekvencia spĺňa požiadavky na spustenie testu, tlačidlo *Run* pri konkrétnom teste zozelenie, ak sekvencia nespĺňa minimálne požiadavky pre test, tlačidlo bude mať oranžovú farbu. Test sa dá spustiť aj napriek neodporučeniu, ale jeho výsledok nemusí byť presný. Vedľa tlačidla *Run* sa nachádza pole, v ktorom sa po dokončení testu zobrazí výsledná hodnota *p*. Na pravej strane sa nachádza log, v ktorom je možné nájsť medzivýsledky pre prípadné overenie správnosti počítania, prípadne priebežné správy užívateľovi.

Na nasledujúcom obrázku je vidieť ako vyzerá program po načítaní sekvencie s dĺžkou 100000 znakov. Keďže pre testy 8, 9, 10, 14 a 15 je sekvencia prikrátka, ich tlačidlá majú oranžovú farbu. Ostatné testy je možné spustiť bez obáv. Testy číslo 5. a 13F. vrátili spoľahlivé hodnoty, test 10 aj po nedoporučení spustenia vrátil *Index Error*. V logu vpravo je možné vidieť výsledky aj medzivýsledky testov 5 a 13F.



Obrázok 2: Grafické rozhranie programu

5.18 Test programu s reálnymi dátami

Organizácia NIST v prílohe B ponúka niekoľko vstupných sekvencii a očakávaných výsledkov, aby bolo možné spoľahlivo otestovať implementáciu testov. Svoj program som teda otestoval pomocou týchto vstupov a dosiahnuté výstupy som porovnal s očakávanými.

Ako prvý vstupný súbor som použil prvých milión binárnych čísel z desatinného rozvoja čísla π.

Tabuľka 21: Test programu pomocou binárneho rozvoja čísla π

Názov testu	Očakávaná hodnota p	Dosiahnutá hodnota p	Rozdiel
Monobit	0,578211	0,578210855	0,0000001452
Frekvenčný test v bloku (m = 128)	0,380615	0,380615198	0,0000001976
Test postupností	0,419268	0,41926842	0,0000004204
Najdlhšia sekvencia jednotiek v bloku	0,02439	0,024389699	0,0000003015
Test binárnej hodnoty matíc	0,083553	0,083553143	0,0000001430
Test pomocou diskrétnej fouriérovej transformácie	0,010186	0,010185826	0,0000001738
Non-overlapping template matching test (m = 9; B = 000000001)	0,165757	0,165757457	0,0000004575
Overlapping template matching test (m = 9)	0,296897	0,296897123	0,0000001234
Maurerov univerzálny štatistický test	0,669012	0,669012438	0,0000004381
Test lineárnej komplexnosti (M = 500)	0,255475	0,255474574	0,0000004259
Sériový test (m = 16)	0,143005	0,143005234	0,0000002343
Test približnej entropie (m = 10)	0,361595	0,36159493	0,0000000695
Cusum (odpredu)	0,628308	0,628308085	0,0000000854
Cusum (odzadu)	0,663369	0,663368609	0,0000003910
Random Excursion Test (x = 1)	0,844143	0,844143101	0,0000001008
Random Excursion Variant Test (x = -1)	0,760966	0,760966325	0,0000003253

Druhá vstupná sekvencia pozostáva z prvého miliónu binárnych čísel z desatinného rozvoja čísla *e*.

Tabuľka 22: Test programu pomocou binárneho rozvoja čísla e

Názov testu	Očakávaná hodnota p	Dosiahnutá hodnota p	Rozdiel
Monobit	0,953749	0,953748629	0,0000003715
Frekvenčný test v bloku (m = 128)	0,211072	0,211071544	0,0000004563
Test postupností	0,561917	0,561916885	0,0000001150
Najdlhšia sekvencia jednotiek v bloku	0,718945	0,71894533	0,0000003299
Test binárnej hodnoty matíc	0,306156	0,306155838	0,0000001625
Test pomocou diskrétnej fouriérovej transformácie	0,847187	0,847186705	0,0000002949
Non-overlapping template matching test (m = 9; B = 000000001)	0,07879	0,078790133	0,0000001327
Overlapping template matching test (m = 9)	0,110434	0,110433685	0,0000003146
Maurerov univerzálny štatistický test	0,282568	0,282567948	0,0000000522
Test lineárnej komplexnosti (M = 500)	0,826335	0,82633477	0,0000002296
Sériový test (m = 16)	0,766182	0,766181644	0,0000003556
Test približnej entropie (m = 10)	0,700073	0,700073387	0,0000003871
Cusum (odpredu)	0,669887	0,669886464	0,0000005358
Cusum (odzadu)	0,724266	0,72426531	0,0000006900
Random Excursion Test (x = 1)	0,786868	0,786867905	0,0000000948
Random Excursion Variant Test (x = -1)	0,826009	0,826009013	0,0000000128

Ako je možné pozorovať z tabuliek, mnou naprogramovaný test vracia takmer rovnaké výsledky ako sú očakávané. Drobné rozdiely sú spôsobené tým, že môj program vracia presnejšie výsledky, ako sú očakávané. Po zaokrúhlení na rovnaký počet desatinných miest, sú dosiahnuté výsledky identické s očakávanými.

6 ZÍSKANIE DÁT PRE TESOTVANIE

Existuje mnoho služieb a spôsobov ako získať náhodné čísla. Ja som zvolil nasledujúce možnosti, nakoľko ich považujem za najpoužívanejšie pre generovanie náhodných čísel. Samozrejme existuje veľké množstvo programovacích jazykov, preto som zvolil programovací jazyk C, nakoľko je to jeden z najpoužívanejších programovacích jazykov a jazyk v ktorom je napísaný program na testovanie spolu s jeho knižnicou *Crypto.py*, ktorá by mala generovať kvalitné pseudonáhodné čísla.

6.1 Programovací jazyk C

Zápis generátora v programovacom jazyku C vyzerá nasledovne:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    FILE *f = fopen("1", "w");
    srand(time(NULL));
    for(int i = 0; i<1048576; i++)
        fprintf(f, "%d", rand()%2);

    fclose(f);
    //char c = getchar();
    return 0;
}
```

Tento generátor najskôr vytvorí súbor s názvom *1*, následne vytvorí seed pre generátor pomocou príkazu *srand*, a keďže chcem aby bol seed zakaždým iný, použil som ako seed aktuálny čas. Generátor následne pomocou *for* cyklu vygeneruje presne 1MB súbor jednotiek a núl. Nakoniec sa súbor uloží a generátor skončí. Takto som vygeneroval 5 rôznych súborov s veľkosťou 1MB.

6.2 Programovací jazyk Python 3.6

Zápis generátoru v programovacom jazyku Python 3.6 vyzerá nasledovne:

```
import random
def normal():
    sample = str(random.getrandbits(1048576))
    mystring = "{0:b}".format(int(sample))
    file = open('1', 'w')
    file.write(mystring)
    file.close()
normal()
```

Pre generovanie pseudonáhodných čísel je potrebné najskôr naimportovať knižnicu *random*. Táto knižnica obsahuje funkciu *getrandbits(n)* kde *n* je počet bitov, ktoré ma funkcia vrátiť. Vygenerované bity sa následne pretypujú na reťazec a uložia sa do súboru *1*. Takto som vygeneroval 5 rôznych súborov s veľkosťou 1MB.

6.3 Knižnica Crypto.py pre programovací jazyk Python

Zápis generátoru v programovacom jazyku Python 3.6 s použitím knižnice *Crypto.py* vyzerá nasledovne:

```
from Crypto.Random import get_random_bytes

def crypto():
    numbers = ''
    for i in range(0, 131072):
        sample = get_random_bytes(1)
        sample = int.from_bytes(sample, byteorder='little')
        sample = bin(sample)[2:]
        sample = sample.zfill(8)
        numbers += sample
    file = open('1', 'w')
    file.write(numbers)
    file.close()

crypto()
```

Pre generovanie kvalitnejších pseudonáhodných čísel je potrebné túto knižnicu najskôr nainštalovať a následne naimportovať. Keďže táto funkcia vracia náhodné bity, bude stačiť ak *for* cyklus pobeží 8 krát menej ako pri generovaní bitov. Každý vygenerovaný byte sa prevedie na číslo v desiatkovej sústave, to sa prevedie na binárne číslo a doplnia sa nuly na začiatok tak, aby malo výsledné binárne číslo dĺžku 8. Výsledok sa pridá do reťazca a po dobehnutí cyklu sa tento reťazec vpíše do súboru *1*. Takto som vygeneroval 5 rôznych súborov s veľkosťou 1MB.

6.4 Generátor Mersenne-Twister

Z webovej stránky tvorcov tohto generátoru som si stiahol voľne dostupnú verziu zdrojového kódu tohto generátoru v programovacom jazyku C. Následne som tento kód prepísal do jazyka Python 3.6 a odskúšal či pri zadaní rovnakých vstupných podmienok generuje rovnaké výstupy. Keďže generátor je navrhnutý tak, aby vracal 32 bitové čísla, je tieto čísla nutné prekonvertovať do binárnej podoby. K tomuto som vytvoril krátky program s nasledujúcim zápisom:

```
def main():
    sgenrand(876846456764)
    count, numbers = 0, ''
    while count <= 1050000: #
        number = genrand()
        binary = bin(number)
        binary = (binary[2:])
        binary = binary.zfill(32)
        seq = str(binary)
        numbers += seq
        count += len(binary)
    numbers = numbers[:1048576]
    with open('1', 'w') as f:
        f.write(numbers)
main()
```

Program najskôr inicializuje generátor so seedom (ten je potreba meniť pri každom zavolaní, vyriešilo by to volanie s aktuálnym časom). Nasleduje for cyklus ktorý sa bude opakovať pokiaľ výstup nebude mať viac ako 1MB dát. V cykle sa najskôr získa náhodné číslo, to sa prekonvertuje na binárne číslo a následne sa doplní tak aby malo dĺžku 32 bitov. Binárne číslo sa zapíše do sekvencie a premenná *count* sa zvýši o dĺžku tohto binárneho čísla. Po získaní dostatočne dlhého výstupu sa sekvencia oreže na dĺžku odpovedajúcu presne 1MB dát. V poslednom kroku sa sekvencia uloží do súboru *1*. Takto som vygeneroval 5 rôznych súborov s veľkosťou 1MB.

6.5 Program Microsoft Excel

Na získanie testovacích dát som použil program Microsoft Excel 2013 Student vo verzii 15.0.4981.1001. Náhodný bit som získal pomocou príkazu *RANDBETWEEN(0;1)*. Tento

príkaz som zavolal na tabuľke o veľkosti 256 * 4096 buniek. Výsledok som uložil do textového súboru a odstránil z neho nechcené znaky ako napríklad medzeru alebo nový riadok. Takto som získal 5 rôznych súborov s veľkosťou 1MB.

6.6 Webový portál Random.org

Tento webový portál vracia pravé náhodné čísla, avšak používateľ ma k dispozícii len 1 000 000 bitov. Po prečerpaní je možné vygenerovať 200 000 bitov každý deň. Binárne dáta sa dajú získať na adrese https://www.random.org/bytes/ po zakliknutí možnosti „Binary“. Počet dát, ktoré je možné získať sa dá overiť na webstránke https://www.random.org/quota/. Zber dát mi trval približne mesiac a nakoniec sa mi podarilo získať sekvenciu dosť dlhú na to, aby sa dala rozložiť na 5 súborov s veľkosťou presne 1MB.

7 TESTOVANIE GENERÁTOROV NÁHODNÝCH ČÍSEL

Každý z generátorov poskytol 5 sekvencii pre testovanie. Každá sekvencia má presne má presne 2^{20} bitov.

7.1 Výsledky generátoru pre programovací jazyk C

Ako prvý je testovaný generátor pre jazyk C. Výsledky jeho testov sú zaokrúhlené na 3 desatinné miesta a dopadli nasledovne:

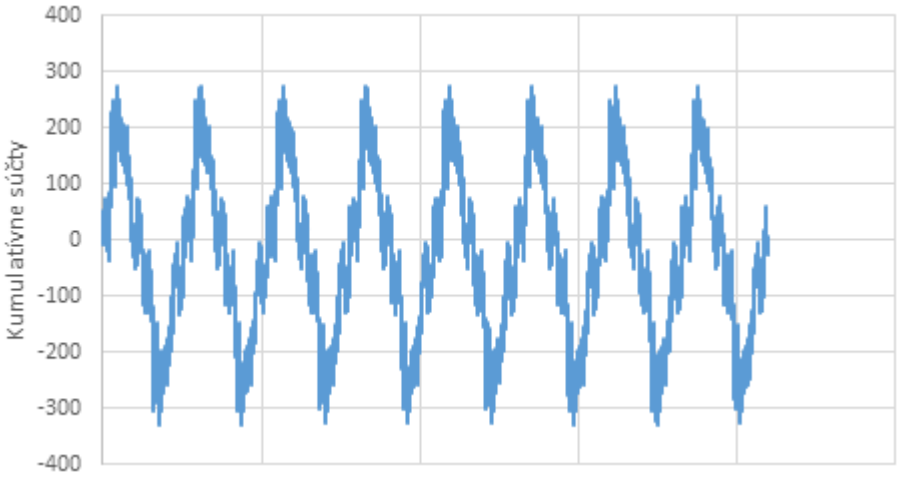
Tabuľka 23: Výsledky testovania generátoru pre programovací jazyk C

Č.	Názov testu		Číslo sekvencie				
			1	2	3	4	5
1	Monobit		1,000	1,000	1,000	1,000	1,000
2	Frekvenčný test v bloku (m = 128)		0,392	0,169	0,616	0,795	0,744
3	Test postupností		0,925	0,927	0,927	0,927	0,925
4	Najdlhšia sekvencia jednotiek v bloku		**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
5	Test binárnej hodnoty matíc		**0,001**	**0,000**	**0,000**	0,539	0,791
6	Test pomocou dis. four. transf.		**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
7	Non-overlapping template matching test (m = 9; B = 000000001)		0,795	0,795	0,795	0,795	0,795
8	Overlapping temp. match. test (m = 9)		**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
9	Maurerov univerzálny štatistický test		0,012	0,023	**0,006**	0,019	0,022
10	Test lineárnej komplexnosti (M = 500)		0,650	0,826	0,131	0,734	0,384
11	Sériový test (m = 8)	1	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
		2	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
12	Test približnej entropie (m = 10)		**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
13	Cusum (odpredu)	F	1,000	0,993	0,988	0,998	1,000
	Cusum (odzadu)	B	1,000	0,993	0,988	0,998	1,000
14	Random Excursion Test	-4	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
		-3	**0,000**	**0,000**	**0,006**	**0,000**	**0,000**
		-2	**0,000**	0,020	**0,006**	**0,000**	0,312
		-1	0,069	**0,000**	**0,001**	**0,000**	0,012
		1	**0,002**	**0,000**	**0,000**	**0,000**	**0,000**
		2	0,032	**0,000**	**0,000**	**0,000**	**0,000**
		3	**0,000**	**0,000**	0,131	**0,000**	**0,000**
		4	**0,000**	**0,001**	**0,005**	**0,000**	**0,000**
15	Random Excursion Variant Test	-9	**0,007**	**0,003**	0,678	0,406	0,560
		-8	**0,008**	**0,008**	0,972	0,397	0,300
		-7	**0,002**	0,070	0,428	0,799	0,027
		-6	**0,000**	0,106	0,252	0,968	**0,005**
		-5	**0,000**	0,023	0,073	0,233	0,043

		-4	**0,000**	0,038	0,023	0,994	0,020
		-3	**0,000**	0,318	**0,003**	0,633	**0,009**
		-2	**0,000**	0,155	**0,002**	0,122	0,030
		-1	**0,006**	0,121	0,738	**0,000**	0,062
		1	0,619	**0,000**	0,859	**0,000**	0,663
		2	0,994	**0,001**	0,577	**0,000**	0,029
		3	0,685	0,023	0,320	**0,000**	**0,000**
		4	0,791	0,086	0,899	**0,005**	**0,000**
		5	0,637	0,961	0,404	0,421	**0,002**
		6	0,025	0,690	0,321	0,826	**0,002**
		7	**0,001**	0,171	0,164	0,363	**0,006**
		8	**0,002**	0,037	0,023	0,248	**0,008**
		9	**0,001**	0,077	0,052	0,311	**0,002**

V prvom teste generátor dosiahol pri každej sekvencii rovnakú hodnotu. To znamená, že každá sekvencia obsahuje presne rovnaký počet jednotiek a núl, teda je možné so zvýšenou pravdepodobnosťou odhadnúť nasledujúci znak. Druhým testom generátor prešiel bez problémov. V treťom teste generátor dosiahol podozrivo rovnakých výsledkov, ktoré sú navyše teste pod jednotkou. Toto ukazuje, že v sekvencii sa striedajú jednotky a nuly veľmi často a pravidelne. Predchádzajúce tvrdenie potvrdzuje štvrtý test, ktorý ako prvý dopadol tak, že ani jedna hodnota *p* nie je väčšia ako *0,01*. Výsledok tohto testu sa dá interpretovať tak, že v testovacej sekvencii sa nenachádzajú takmer žiadne postupnosti jednotiek. Piaty test dopadol pre niektoré sekvencie priaznivo, pre niektoré nie. Šiesty test je pre každú sekvenciu rovný nule. To znamená, že v sekvencii nie je dostatok hodnôt väčších ako referenčná hodnota *T*. Siedmi test taktiež dopadol pre každú jednu sekvenciu rovnako, znamená to, že v každej sekvencii sa testovaná šablóna vyskytuje rovnako často. Test číslo osem taktiež potvrdil, že sekvencia nie je náhodná. Deviatym testom prešli len 4 z 5 sekvencii a desiatym všetky sekvencie. Jedenásty test vrátil pre všetky sekvencie nulu, značí to, že v sekvencii je príliš rovnomerné rozloženie blokov. Dvanásty test potvrdzuje to, čo zatiaľ viem o tomto generátore, a to že naprieč sekvenciou vládne silná pravidelnosť. Trinásty test ukázal, že rozdelenie jednotiek a núl je veľmi pravidelné, je totižto jedno z ktorej strany sa začne, vždy je dosiahnutý rovnaký výsledok. Štrnásty a pätnásty test nakoniec potvrdili, že v sekvenciách generovaných céčkovým generátorom vládne príliš málo náhodnosti.

To, že generátor je absolútne nespoľahlivý sa dá pozorovať aj na nasledujúcom grafe, kde sú zobrazené kumulatívne súčty. Tu je jasne vidieť, že náhodné čísla sa opakujú, a uhádnuť ďalšie číslo zo sekvencie je v prípade tohto generátoru veľmi jednoduché.



Obrázok 3: Kumulatívne súčty 1. testovanej sekvencie

7.2 Výsledky generátoru pre programovací jazyk Python

Výsledky zaokrúhlené na 3 desatinné miesta dopadli pre vstavaný pythonovský generátor náhodných čísel nasledovne:

Tabuľka 24: Výsledky testovania generátoru pre programovací jazyk Python

Č.	Názov testu		Číslo sekvencie				
			1	2	3	4	5
1	Monobit		0,610	0,831	0,430	0,437	0,966
2	Frekvenčný test v bloku (m = 128)		0,598	0,904	0,939	0,438	0,203
3	Test postupností		0,988	0,527	0,875	0,376	0,338
4	Najdlhšia sekvencia jednotiek v bloku		0,983	0,686	0,423	0,190	0,217
5	Test binárnej hodnoty matíc		0,663	0,579	0,306	0,917	0,022
6	Test pomocou dis. four. transf.		0,716	0,222	0,153	0,637	0,983
7	Non-overlapping template matching test (m = 9; B = 000000001)		0,276	0,969	0,414	0,393	0,406
8	Overlapping temp. match. test (m = 9)		0,984	0,730	0,706	0,835	0,567
9	Maurerov univerzálny štatistický test		0,134	0,728	0,683	0,957	0,048
10	Test lineárnej komplexnosti (M = 500)		0,710	0,816	0,468	0,858	0,140
11	Sériový test (m = 8)	1 2	0,878 0,987	0,800 0,527	0,363 0,218	0,161 0,714	0,630 0,881
12	Test približnej entropie (m = 10)		0,175	0,222	0,509	0,429	0,639
13	Cusum (odpredu) Cusum (odzadu)	F B	0,927 0,831	0,366 0,245	0,539 0,785	0,792 0,719	0,913 0,881

14	Random Excursion Test	-4	0,572	0,609	0,271	0,883	0,836
		-3	0,601	0,397	0,287	0,930	0,859
		-2	0,877	0,373	0,087	0,366	0,802
		-1	0,754	0,527	0,774	0,241	0,701
		1	0,895	0,797	0,919	0,868	0,749
		2	0,378	0,783	0,820	0,768	0,904
		3	0,847	0,767	0,767	0,643	0,619
		4	0,053	0,419	0,475	0,657	0,959
15	Random Excursion Variant Test	-9	0,304	0,082	0,458	0,890	0,969
		-8	0,276	0,054	0,325	0,984	0,780
		-7	0,333	0,062	0,206	0,693	0,735
		-6	0,391	0,048	0,242	0,477	0,669
		-5	0,492	0,024	0,188	0,393	0,488
		-4	0,547	0,043	0,095	0,384	0,463
		-3	0,352	0,121	0,162	0,862	0,754
		-2	0,544	0,167	0,255	0,491	0,533
		-1	0,856	0,285	0,196	0,378	0,124
		1	1,000	0,665	0,586	0,378	0,497
		2	0,944	0,607	0,783	0,788	0,695
		3	0,928	0,900	0,670	0,473	0,610
		4	0,933	0,878	0,817	0,512	0,970
		5	0,925	0,973	0,928	0,569	0,936
		6	0,884	0,690	0,682	0,558	0,630
		7	0,691	0,708	0,428	0,620	0,441
		8	0,628	0,793	0,334	0,411	0,505
		9	0,534	0,911	0,458	0,340	0,491

Z dosiahnutých výsledkov je viditeľné, že žiadna sekvencia v žiadnom teste nedosiahla hodnotu p menšiu ako *0,01*. Generátor teda generuje spoľahlivé pseudonáhodné čísla. K tomuto tvrdeniu som dospel aj vďaka tomu, že hodnoty p sa pri jednotlivých testoch podstatne líšia.

7.3 Výsledky generátoru pre Microsoft Excel 2013

Výsledky testov zaokrúhlené na 3 desatinné miesta dopadli pre excelovský generátor nasledovne:

Tabuľka 25: Výsledky testovania generátoru pre program Microsoft Excel 2013

Č.	Názov testu		Číslo sekvencie				
			1	2	3	4	5
1	Monobit		0,523	0,724	0,699	0,670	0,488
2	Frekvenčný test v bloku (m = 128)		0,087	0,358	0,870	0,201	0,406

3	Test postupností		0,354	0,618	0,346	0,679	0,281
4	Najdlhšia sekvencia jednotiek v bloku		0,221	0,075	0,730	0,077	0,566
5	Test binárnej hodnoty matíc		0,021	0,447	0,663	0,726	0,219
6	Test pomocou dis. four. transf.		0,097	0,201	0,861	0,731	0,156
7	Non-overlapping template matching test (m = 9; B = 000000001)		0,399	0,916	0,301	0,345	0,775
8	Overlapping temp. match. test (m = 9)		0,025	0,059	0,742	0,404	0,059
9	Maurerov univerzálny štatistický test		0,986	0,985	0,771	0,715	0,394
10	Test lineárnej komplexnosti (M = 500)		0,720	0,266	0,039	0,494	0,924
11	Sériový test (m = 8)	1	0,138	0,325	0,640	0,689	0,639
		2	0,198	0,010	0,594	0,674	0,741
12	Test približnej entropie (m = 10)		0,231	0,102	0,396	0,305	0,549
13	Cusum (odpredu)	F	0,925	0,536	0,981	0,489	0,668
	Cusum (odzadu)	B	0,614	0,856	0,695	0,734	0,539
14	Random Excursion Test	-4	0,425	0,505	0,483	0,535	0,799
		-3	0,138	0,765	0,770	0,980	0,624
		-2	0,406	0,323	0,879	0,946	0,970
		-1	0,200	0,940	0,401	0,090	0,921
		1	0,864	0,935	0,572	0,125	0,332
		2	0,954	0,171	0,448	0,638	0,083
		3	0,983	0,832	0,388	0,592	0,979
		4	0,364	0,308	0,501	0,264	0,749
15	Random Excursion Variant Test	-9	0,883	0,608	0,413	0,838	0,375
		-8	0,879	0,420	0,464	0,976	0,483
		-7	0,665	0,241	0,548	0,991	0,852
		-6	0,756	0,446	0,414	0,850	0,750
		-5	0,824	0,806	0,571	0,530	0,551
		-4	0,579	0,652	0,687	0,472	0,453
		-3	0,364	0,651	0,968	0,516	0,227
		-2	0,726	0,979	0,434	0,308	0,133
		-1	0,449	0,927	0,871	0,637	0,303
		1	0,844	0,696	0,134	0,860	0,502
		2	0,713	0,586	0,150	0,892	0,824
		3	0,699	0,992	0,167	0,617	0,754
		4	0,788	0,555	0,293	0,244	0,996
		5	0,369	0,362	0,470	0,084	0,856
		6	0,233	0,375	0,579	0,090	0,799
		7	0,226	0,683	0,561	0,253	0,584
		8	0,247	0,877	0,414	0,441	0,709
		9	0,304	0,701	0,295	0,644	0,695

Z dosiahnutých výsledkov je viditeľné, že žiadna sekvencia v žiadnom teste nedosiahla hodnotu p menšiu ako *0,01*. Generátor teda generuje spoľahlivé pseudonáhodné čísla. K tomuto tvrdeniu som dospel aj vďaka tomu, že hodnoty p sa pri jednotlivých testoch podstatne líšia.

7.4 Výsledky generátoru mersenne twister

Výsledky testov zaokrúhlené na 3 desatinné miesta dopadli pre excelovský generátor nasledovne:

Tabuľka 26: Výsledky testovania generátoru mersenne twister

Č.	Názov testu		Číslo sekvencie				
			1	2	3	4	5
1	Monobit		0,397	0,170	0,716	0,788	0,026
2	Frekvenčný test v bloku (m = 128)		0,529	0,367	0,097	0,045	0,875
3	Test postupností		0,668	0,639	0,137	0,378	0,012
4	Najdlhšia sekvencia jednotiek v bloku		0,028	0,280	0,126	0,420	0,396
5	Test binárnej hodnoty matíc		0,367	0,691	0,689	0,421	0,865
6	Test pomocou dis. four. transf.		0,257	0,813	0,819	0,100	0,046
7	Non-overlapping template matching test (m = 9; B = 000000001)		0,236	0,056	0,829	0,109	0,774
8	Overlapping temp. match. test (m = 9)		0,939	0,459	0,514	0,200	0,594
9	Maurerov univerzálny štatistický test		0,695	0,040	0,382	0,357	0,910
10	Test lineárnej komplexnosti (M = 500)		0,590	0,718	0,656	0,924	0,034
11	Sériový test (m = 8)	1	0,037	0,954	0,700	0,524	0,621
		2	0,256	0,645	0,820	0,474	0,960
12	Test približnej entropie (m = 10)		0,177	0,483	0,668	0,433	0,215
13	Cusum (odpredu)	F	0,627	0,225	0,883	0,883	0,050
	Cusum (odzadu)	B	0,487	0,166	0,556	0,640	0,019
14	Random Excursion Test	-4	0,920	0,856	0,668	0,488	0,408
		-3	0,641	0,361	0,667	0,920	0,739
		-2	0,958	0,693	0,827	0,850	0,826
		-1	0,803	0,197	0,531	0,921	0,321
		1	0,966	0,287	0,146	0,273	0,831
		2	0,460	0,934	0,738	0,183	0,747
		3	0,970	0,508	0,736	0,057	0,895
		4	0,762	0,649	0,803	**0,009**	0,772

15	Random Excursion Variant Test	-9	0,339	0,765	0,383	0,570	0,640
		-8	0,322	0,800	0,442	0,852	0,549
		-7	0,239	0,895	0,567	0,924	0,520
		-6	0,292	0,753	0,836	0,957	0,631
		-5	0,320	0,809	0,970	0,798	0,608
		-4	0,379	0,896	0,910	0,985	0,519
		-3	0,488	0,562	0,759	0,524	0,336
		-2	0,612	0,412	0,483	0,584	0,170
		-1	0,791	0,297	0,169	0,948	0,219
		1	0,919	0,825	0,070	0,154	0,328
		2	0,724	0,898	0,362	0,110	0,498
		3	0,609	0,843	0,975	0,312	0,431
		4	0,671	0,599	0,815	0,447	0,258
		5	0,536	0,528	0,450	0,723	0,183
		6	0,349	0,487	0,456	0,855	0,186
		7	0,267	0,545	0,589	0,690	0,258
		8	0,273	0,579	0,500	0,748	0,204
		9	0,347	0,685	0,339	0,769	0,169

Po testovaní tohto generátoru som dostal jednu hodnotu menšiu ako *0,01*. Túto hodnotu dosiahla testovacia sekvencia číslo 4 pri štrnástom teste pre $x = 4$. Táto hodnota je zobrazená v tabuľke červenou farbou. Keďže všetky ostatné hodnoty sú väčšie ako minimálna hranica, je možné tvrdiť, že generátor generuje kvalitné pseudonáhodné čísla a výskyt nepriaznivého výsledku je len ojedinelá náhoda.

7.5 Výsledky generátoru z webovej služby random.org

Výsledky testov zaokrúhlené na 3 desatinné miesta dopadli pre excelovský generátor nasledovne:

Tabuľka 27: Výsledky testovania generátoru z webovej služby random.org

Č.	Názov testu		Číslo sekvencie				
			1	2	3	4	5
1	Monobit		0,705	0,562	0,219	0,210	0,371
2	Frekvenčný test v bloku (m = 128)		0,834	0,125	0,232	0,875	0,514
3	Test postupností		0,463	0,169	0,283	0,966	0,775
4	Najdlhšia sekvencia jednotiek v bloku		0,941	0,080	0,303	0,841	0,513
5	Test binárnej hodnoty matíc		0,010	0,826	0,153	0,392	0,975

6	Test pomocou dis. four. transf.		0,143	0,854	0,997	0,763	0,199
7	Non-overlapping template matching test (m = 9; B = 000000001)		0,178	0,011	0,121	0,807	0,012
8	Overlapping temp. match. test (m = 9)		0,988	0,178	0,032	0,299	0,764
9	Maurerov univerzálny štatistický test		0,044	0,609	0,035	0,396	0,014
10	Test lineárnej komplexnosti (M = 500)		0,519	0,275	0,595	0,736	0,544
11	Sériový test (m = 8)	1	0,568	0,658	0,259	0,951	0,815
		2	0,228	0,797	0,074	0,868	0,784
12	Test približnej entropie (m = 10)		0,565	0,864	0,121	0,662	0,611
13	Cusum (odpredu)	F	0,536	0,691	0,401	0,389	0,318
	Cusum (odzadu)	B	0,877	0,710	0,148	0,150	0,043
14	Random Excursion Test	-4	0,128	0,590	0,256	0,076	0,649
		-3	0,724	0,280	0,193	0,053	0,354
		-2	0,229	0,089	0,123	0,063	0,722
		-1	0,346	0,586	0,319	0,244	0,354
		1	0,912	0,395	0,682	0,681	0,155
		2	0,601	0,135	0,805	0,746	0,094
		3	0,298	0,188	0,881	0,074	0,823
		4	0,544	0,803	0,609	0,260	0,897
15	Random Excursion Variant Test	-9	0,163	0,347	0,237	0,313	0,596
		-8	0,153	0,188	0,214	0,334	0,700
		-7	0,200	0,147	0,191	0,379	0,869
		-6	0,287	0,140	0,197	0,585	0,728
		-5	0,339	0,131	0,139	0,383	0,795
		-4	0,418	0,320	0,063	0,105	0,763
		-3	0,545	0,753	0,106	0,051	0,363
		-2	0,889	0,928	0,330	0,082	0,153
		-1	0,824	1,000	0,768	0,733	0,049
		1	0,657	0,466	0,894	0,351	0,635
		2	0,449	0,300	0,699	0,396	0,992
		3	0,521	0,358	0,598	0,102	0,940
		4	0,927	0,485	0,731	0,032	0,949
		5	0,952	0,690	0,964	0,038	0,914
		6	0,812	0,814	0,735	0,030	0,503
		7	0,741	0,840	0,705	0,052	0,296
		8	0,782	0,717	0,604	0,082	0,338
		9	0,957	0,695	0,516	0,159	0,344

Z dosiahnutých výsledkov je viditeľné, že žiadna sekvencia v žiadnom teste nedosiahla hodnotu p menšiu ako *0,01*. Generátor teda generuje spoľahlivé pseudonáhodné čísla. K tomuto tvrdeniu som dospel aj vďaka tomu, že hodnoty p sa pri jednotlivých testoch podstatne líšia.

7.6 Výsledky generátoru pre knižnicu Crypto.py

Výsledky testov zaokrúhlené na 3 desatinné miesta dopadli pre excelovský generátor nasledovne:

Tabuľka 28: Výsledky testovania generátoru pre knižnicu Crypto.py

Č.	Názov testu		Číslo sekvencie				
			1	2	3	4	5
1	Monobit		0,206	0,411	0,882	0,100	0,439
2	Frekvenčný test v bloku (m = 128)		0,387	0,979	0,556	0,311	0,052
3	Test postupností		0,193	0,109	0,216	0,442	0,434
4	Najdlhšia sekvencia jednotiek v bloku		0,095	0,503	0,976	0,250	0,338
5	Test binárnej hodnoty matíc		0,511	0,890	0,479	0,843	0,716
6	Test pomocou dis. four. transf.		0,413	0,981	0,422	0,750	0,403
7	Non-overlapping template matching test (m = 9; B = 000000001)		0,465	0,469	0,837	0,049	0,325
8	Overlapping temp. match. test (m = 9)		0,532	0,153	0,447	0,838	0,826
9	Maurerov univerzálny štatistický test		0,386	0,660	0,609	0,091	0,788
10	Test lineárnej komplexnosti (M = 500)		0,900	0,667	0,086	0,678	0,494
11	Sériový test (m = 8)	1	0,412	0,679	0,121	0,621	0,591
		2	0,510	0,374	0,295	0,332	0,874
12	Test približnej entropie (m = 10)		0,502	0,204	0,775	0,781	0,862
13	Cusum (odpredu)	F	0,147	0,683	0,600	0,133	0,828
	Cusum (odzadu)	B	0,276	0,194	0,475	0,123	0,420
14	Random Excursion Test	-4	0,463	0,676	0,961	0,856	0,265
		-3	0,542	0,388	0,100	0,648	0,209
		-2	0,449	0,592	0,741	0,605	0,273
		-1	0,240	0,562	0,932	0,999	0,799
		1	0,641	0,079	0,741	0,391	0,570
		2	0,622	0,153	0,716	0,696	0,763
		3	0,701	0,284	0,777	0,592	0,943
		4	0,205	0,445	0,292	0,921	0,931
15	Random Excursion Variant Test	-9	0,672	0,813	0,694	0,468	0,257
		-8	0,797	0,986	0,667	0,326	0,151

		-7	0,992	1,000	0,798	0,132	0,197
		-6	0,830	0,983	0,662	0,043	0,309
		-5	0,568	0,577	0,804	0,046	0,308
		-4	0,590	0,221	0,967	0,103	0,477
		-3	0,667	0,175	0,739	0,339	0,981
		-2	0,209	0,717	0,723	0,675	0,729
		-1	0,030	0,807	0,776	0,864	0,657
		1	0,212	0,163	0,510	0,442	0,875
		2	0,174	0,243	0,990	0,605	0,821
		3	0,067	0,963	0,695	0,804	0,674
		4	0,057	0,813	0,908	0,771	0,502
		5	0,083	0,889	0,781	1,000	0,559
		6	0,053	0,793	0,905	0,817	0,631
		7	0,050	0,706	0,971	0,713	0,487
		8	0,109	0,808	0,816	0,414	0,373
		9	0,149	0,939	0,710	0,443	0,539

Z dosiahnutých výsledkov je viditeľné, že žiadna sekvencia v žiadnom teste nedosiahla hodnotu p menšiu ako *0,01*. Generátor teda generuje spoľahlivé pseudonáhodné čísla. K tomuto tvrdeniu som dospel aj vďaka tomu, že hodnoty p sa pri jednotlivých testoch podstatne líšia.

ZÁVER

Táto práca sa zaoberá problematikou testovania náhodných čísel pomocou Diehard testov a vytvorením samotnej aplikácie na testovanie generátorov náhodných čísel.

Prvá kapitola pojednáva o tom, čo je to náhodnosť a rozdieloch medzi generátormi náhodných a pseudonáhodných čísel. Druhá kapitola predstavuje najvýznamnejšie generátory náhodných a pseudonáhodných čísel, ich vlastnostiach, použití a určení. Tretia kapitola sa venuje konkrétnym testom. Jedná sa o kompletnú sadu Diehard testov tak, ako ju štandardizovala organizácia NIST. Každý jeden test je popísaný tak, že na začiatku sa dá nájsť čo si test všíma, potom sú vysvetlené podmienky, čo by mala spĺňať testovaná sekvencia a ostatné vstupné premenné aby sa na výsledok testovania dalo spoľahnúť. Každý test taktiež obsahuje popis každého kroku od začiatku až po koniec a zároveň je test demonštrovaný na zjednodušenom príklade pre lepšie pochopenie.

Praktická časť obsahuje kapitolu, ktorá objasní, v akom prostredí som aplikáciu vyvíjal, aké som použil knižne, a popis funkcii, ktoré som si naprogramoval sám. Ďalšia kapitola praktickej časti obsahuje pythnovský kód každého jedného testu spolu s popisom každého príkazu a test na jednoduchej sekvencii. V tejto kapitole sa taktiež dá nájsť, ako sú riešené podmienky vhodnosti testov pre konkrétnu sekvenciu, ako vyzerá a ako sa ovláda užívateľské rozhranie a test naprogramovaného generátoru. Výsledky testov sú porovnané s očakávanými výsledkami v prehľadnej tabuľke.

Šiesta kapitola priblíži, ako som získal dáta s generátorov pseudonáhodných čísel a z webovej stránky random.org. Nachádzajú sa tu kódy jednoduchých aplikácii v konkrétnych programovacích jazykoch, vzorec pre generovanie pseudonáhodných bitov v MS Excel a návod ako získať sekvenciu náhodných bitov zo stránky random.org.

V poslednej kapitole sú zobrazené a zhodnotené výsledky testovania náhodných a pseudonáhodných sekvencii pomocou môjho programu. Každý zdroj dát bol otestovaný pomocou piatich vstupných súborov a výsledky boli zaznamenané v prehľadných tabuľkách. Nevyhovujúce hodnoty boli zvýraznené a každý generátor bol zhodnotený slovne.

Hlavný cieľ tejto práce bol v naprogramovaní a následnom otestovaní najpoužívanejších generátorov pseudonáhodných čísel. Tento program taktiež môže pomôcť pri vývoji a testovaní nových generátorov náhodných alebo pseudonáhodných čísel.

ZOZNAM POUŽITEJ LITERATÚRY

[1] *Numerical recipes in fortran: the art of scientific computing*. 2nd ed. Cambridge: Cambridge University Press, 1992. ISBN 05-214-3064-X.

[2] What is randomness? *Zitogiuseppe* [online]. 2012 [cit. 2018-03-13]. Dostupné z: http://zitogiuseppe.com/randomness.html

[3] Pseudo Random Number Generator (PRNG). *Geeksforgeeks* [online]. [cit. 2018-03-11]. Dostupné z: https://www.geeksforgeeks.org/pseudo-random-number-generator-prng/

[4] Hardwarový generátor náhodných čísel aneb náhoda z atomů. *Root.cz* [online]. 2010 [cit. 2018-03-11]. Dostupné z: https://www.root.cz/clanky/hardwarovy-generator-nahodnych-cisel-aneb-nahoda-z-atomu/

[5] What's this fuss about true randomness?. *Random.org* [online]. 2018 [cit. 2018-03-14]. Dostupné z: https://www.random.org/

[6] Bit Tally. *Random.org* [online]. 2018 [cit. 2018-03-14]. Dostupné z: https://www.random.org/bit-tally/

[7] MATSUMOTO, Makoto a Takuji NISHIMURA. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*. 8(1), 3-30. DOI: 10.1145/272991.272995. ISSN 10493301. Dostupné také z: http://portal.acm.org/citation.cfm?doid=272991.272995

[8] Description of the RAND function in Excel. *Microsoft Support* [online]. 2017 [cit. 2018-03-15]. Dostupné z: https://support.microsoft.com/en-us/help/828795/description-of-the-rand-function-in-excel

[9] Rand() and srand() in C/C++. *GeeksforGeeks*[online]. [cit. 2018-03-18]. Dostupné z: https://www.geeksforgeeks.org/rand-and-srand-in-ccpp/

[10] random — Generate pseudo-random numbers. *Python 3.6.5rc1 documentation*[online]. [cit. 2018-03-18]. Dostupné z: https://docs.python.org/3/library/random.html

[11] SUMMERFIELD, Mark. *Python 3: výukový kurz*. Brno: Computer Press, 2010. ISBN 978-802-5127-377.

[12] RUKHIN, Andrew L. *A statistical test suite for random and pseudorandom number generators for cryptographic applications*. Washington, D.C.: For sale by the Supt. of Docs., U.S. G.P.O., 2000. NIST special publication, 800-22.

[13] SOTO, Juan. *Statistical Testing of Random Number Generators*. 1999, National Institute of Standards & Technology

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

BBS Blum Blum Shub

GFSR Generalised Feedback Shift Register

LFG Lagged Fibonacci Generator

LSFR Linear Feedback Shift Register

NIST National Institut of Standards and Technology

PRNG Pseudorandom Number Generator

TRNG True Random Number Generator

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK