Eötvös Loránd Tudományegyetem

Informatikai Kar

Komputeralgebra Tanszék

Pszeudovéletlen sorozatok mértékei és konstrukciói

Szerző:

Kovács Levente

programtervező informatikus BSc

Témavezető:

Dr. Tóth Viktória

egyetemi adjunktus

Komputeralgebra Tanszék

Budapest, 2019

Tartalomjegyzék

1 Bevezető

1.1 Történeti betekintés

Véletlen sorozatok generálására sok modern területnek igénye van (pl.: statisztika, szimulációk, kriptográfia). Számítógéppel azonban nem lehetséges valódi véletlen számokat generálni, ezért pszeudovéletlen eljárásokra hagyatkozunk. Cél az, hogy olyan eljárásokat használjunk melyek statisztikailag véletlennek tűnnek, attól függetlenül, hogy egy determinisztikus rendszer generálja az eredményt.

Az ilyen sorozatokra való igény először Gilbert Vernam munkájának következménye. 1917-ben Vernam feltalálta a one-time pad titkosítási eljárást. Az üzeneteket telegráfon bitenként továbbították, viszont az eredeti adatot valamilyen kulcs segítségével megváltoztatták (amit aztán a fogadó fél a kulcs ismeretében vissza tud fejteni). Ha le is hallgatták a küldött adatot, akkor a kulcs nélkül nem lehetett tudni, hogy mi volt az eredeti üzenet.

1949-ben Claude Shannon, matematikus, bebizonyította a fenti eljárásról, hogyha egy jó sorozatot használunk kulcsnak, akkor az eredeti és a titkosított bitsorozat között nem lehet semmiféle összefüggést találni. Ennek következménye, hogy a Vernam-féle one-time pad eljárást nem lehet feltörni, ha megfelelő, egyszer használatos kulcsokat használunk.

Az is igaz azonban, hogyha nem megfelelő kulcsokat használunk (pl. többször használjuk a kulcsokat), akkor a titkosítás könnyen törhető. A fő nehézség tehát a one-time pad használatánál, hogy nagy méretű véletlen bitsorozatokat kell előállítani kulcsoknak.

Az ilyen sorozatok előállításához régen fizikai eszközöket használtak, pl. diódákat. A problémája az ilyen eszközöknek, hogy külső tényezők is befolyásolják a véletlenséget, és meghibásodás is történhet. Ezért nagyon fontos, a véletlenség vizsgálása különböző statisztikai tesztekkel, ami azonban időigényes. Emiatt jobb, hogyha "csak" pszeudovéletlen sorozatokat állítunk elő. Így elkerülhető a statisztikai tesztelés, mivel

bizonyítottan jó véletlen tulajdonságokkal rendelkező sorozatot kapunk.

A pszeudovéletlen tulajdonságok mérésével sok évtizede foglalkoznak matematikusok. Elsőként Émile Borel definiálta végtelen bináris sorozatokra a pszeudovéletlenség normalitás mértékét. Később Solomon Wolf Golomb és Donald Knuth próbálkoztak a pszeudovéletlenség matematikai fogalmának definiálására. Ennek a definíciónak alapvető gyengeségei voltak, ezért ezt ma már nem nagyon használják. Ezután Andrej Nyikolajevics Kolmogorov és Gregory John Chaitin bonyolultságelméleti irányból közelítették meg a problémát, de a gyakorlatban ezek a fogalmak nem kaptak nagy jelentőséget, mivel alkalmazásuk nem jól kivitelezhető.

Ezen próbálkozások inspiráltak egy új, konstruktív megközelítést a pszeudovéletlenség vizsgálatára. Christian Manduit és Sárközy András megközelítése az volt, hogy léteznek olyan sorozatok, melyekről bizonyítható, hogy statisztikailag erős véletlen tulajdonságokkal rendelkeznek (más szóval: erős pszeudovéletlen tulajdonságúak), így az utólagos tesztelés elkerülhető. Ez a fajta vizsgálat az évek során egyre szélesebb körben alkalmazott.

1.2 A program célja

A programom célja, hogy a fent említett konstrukció keretén belül vizsgáljak különböző pszeudovéletlen sorozatokat. A vizsgált konstrukciókat két csoportból válogattam. Egyik kategóriába matematikai hátterű sorozatok tartoznak, melyeknek működése főként számelméleti ismereteken alapulnak. A másik csoport az iparban széles körben használt pszeudovéletlen generátorok, amik nem feltétlenül matematikai hátterűek.

A programban lehetőség nyílik a pszeudovéletlen konstrukciók kipróbálására, és a generált sorozatok vizsgálatára a statisztikai mértékek szerint. A sorozatokat ki is próbálhatjuk a Vernam-féle one-time pad titkosítási eljárással.

2 Felhasználói dokumentáció

A program pszeudovéletlen sorozatkonstrukciók kipróbálására és a sorozatok vizsgálatára készült a statisztikai mértékekkel.

2.1 Rendszerkövetelmények

2.2 Futattás

2.3 Menürendszer

A menürendszer pontjainak 3 fő csoportja van, ezek:

- Alapvető funkciók: Kilépés, vissza a főmenübe
- Konstrukciók: A konstrukciók kipróbálása
- Tesztelés: Mértékek és titkosítás

2.3.1 Főmenü

Amikor a program elindul a felhasználót a főmenü fogadja. Itt elérhető az összes megvalósított konstrukció almenüpontja és a mértékek, illetve titkosításé is.

Kilépni a program jobb felső sarkában található "X" gombbal, vagy a főmenüben található "Kilépés" gombbal lehet.

Konstrukciók

Ez a pont csak alapvető segítséget nyújt a különböző konstrukciók kipróbálásához. Részletes leírása az algoritmusoknak a fejlesztői dokumentációban találhatóak.

2.3.2 Legendre szimbólumos konstrukció

Ez a konstrukció a Legendere szimbólumon alapul, melynek véletlen tulajdonságait régóta vizsgálják.

Szükséges paraméterek:

1. **A sorozat hossza**

   A sorozat hosszát bájtokban kell megadni, tehát az 1 bájt hosszú sorozat 8 bitet foglal magába. Bármennyi lehet.

2. **Prímszám**

   Egy olyan prímszámra van szükségünk a konstrukcióhoz, mely legalább 16-szor nagyobb mint a sorozat hossza (bitben megadott méret esetén 2-szer nagyobb). Az is kell, hogy a 2 legyen primitív gyök modulo prím. A "Generálás" gombbal a program megkeresi a legkisebb olyan prímet mely a fenti feltételeknek megfelel. A "Következő" gombbal ugorhatunk a következő legkisebb jó prímre.

3. **A konstrukcióhoz használt polinom fokszáma**

   A fokszám nem lehet túl nagy függően a prímszámtól, de túl kicsi sem. Az alsó határ jelen megvalósításban 2-nek, a felső $5 \cdot \sqrt[10]{p}$-nek lett megszabva, ahol p a használt prímszám. Polinom fokszámot generálhatunk a fenti intervallumban a "Generálás" gombbal.

4. **A polinom**

   A konstrukció jó működéséhez szükséges, hogy olyan polinomot válasszunk, amelynek csak egyszeres gyökei vannak. Emiatt a polinomot a gyökeinek felsorolásával reprezentáljuk, ezeket a szóközzel elválasztva kell felsorolni a szövegmezőben. Lehetőség van polinom generálására is, ekkor a program legenerál egy az előző pontban megadott fokszámú polinomot.

Ha fenti paramétereket megadtuk, akkor a "Sorozat generálása" gombra kattintva megkapjuk az eredményét a Legendre konstrukciónak a kapott bemenetre.

2.3.3 RC4 konstrukció

Az RC4 konstrukciót Ron Rivest fejlesztette ki az 1987-ben viszont eredetileg üzleti titok volt. 1994-ben azonban kiszivárgott az algoritmus leírása. A működése egyszerű és gyors, ezért használata széles körben elterjedt, viszont vannak rossz tulajdonságai.

Szükséges paraméterek:

1. **A sorozat hossza**
   A sorozat hosszát bájtokban kell megadni. Bármennyi lehet.
2. **A kulcs**
   A kulcs változó méretű lehet, ez alapján fog történni a generálás. Tipikusan 40-2048 bit hosszú a kulcs. Ez a megvalósítás egy szöveget vár kulcsnak, így átfogalmazva a feltételt legalább 5, maximum 256 karakter hosszú lehet a kulcs.

A fenti paraméterek megadása után megtekinthetjük az RC4 konstrukció által generált sorozatot a "Sorozat generálása" gombbal.

3 Fejlesztői dokumentáció

3.1 Konstrukciók

3.1.1 Legendre konstrukció

Ismertető

Def.: Ha p egy prímszám, és $a \in \mathbb{Z}$, akkor az $\left(\frac{a}{p}\right)$ Legendre-szimbólum értéke:

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{ha } a \text{ kvadratikus maradék modulo } p \text{ és } a \not\equiv 0 \pmod{p} \\ -1 & \text{ha } a \text{ nem kvadratikus maradék modulo } p \\ 0 & \text{ha } a \equiv 0 \pmod{p} \end{cases}$$

A Legendre-szimbólum véletlenségét már régóta vizsgálták, ez volt a motiváció a konstrukció megvalósítására.

Algoritmus

1. Vegyünk egy olyan p prímet, hogy a 2 legyen primitív gyök modulo p
2. Vegyünk egy olyan $f \in \mathbb{F}_p[x]$ polinomot, melynek csak egyszeres gyökei vannak, és foka nem túl nagy p-hez képest
3. A $(e_0, e_1, ..., e_{k-1})$ sorozat n. tagja: $e_n = \begin{cases} \left(\frac{f(i)}{p}\right) & \text{, ha } f(i) \nmid p \\ 1 & \text{, ha } f(i) \mid p \end{cases}$

Extra követelmények

Ha egy n hosszú sorozatot szeretnénk generálni, akkor a használt p prímre legyen igaz, hogy $p > 2n$.

3.1.2 RC4 konstrukció

Ismertető

Az RC4 stream cipher algoritmus a bájt hosszúságú bitsorozatok egy permutációjának előállításával kezdődik, ami egy kulcssorozat segítségével történik. Ez a key-scheduling fázisa az algoritmusnak.

Ez után következik a tényleges sorozat generálás. Ameddig kell a sorozatot generálni (pl. hossz alapján), addig az algoritmus lépésenként keveri a bájtok sorrendjét, és közben egy bájtnyi kimenetet generál.

Ebben a megvalósításban a klasszikus, módosítások nélküli algoritmust valósítottam meg, hogy ez hogyan viszonyul a megvalósított mértékekhez. A módszernek több változata van, amik javíthatják a véletlenségét a generált sorozatnak.

Algoritmus

1. Töltsünk fel egy 256 elemű S tömböt úgy, hogy i. indexére: $S[i] := i$. 0-tól indexelünk, és gyakorlatilag az összes 8 hosszúságú bitsorozatot kell előállítani
2. Adjunk meg egy n méretű K (ebben a megvalósításban egy n karakterű ASCII kódolású szöveg) kulcstömböt, ahol $1 \leq n \leq 256$
3. Ez az algoritmus key-scheduling fázisa. Legyen $i, j := 0$, majd:
   (a) Legyen $j := (j + S[i] + K[i \bmod n]) \bmod 256$
   (b) Cseréljük ki $S[i]$ és $S[j]$ értékét
   (c) Inkrementáljuk i-t, ha $i < 256$, akkor ugorjunk vissza (a)-ra
4. Ez a sorozatgeneráló fázis. Legyen $i, j := 0$, n-szer kell generálni 1 bájtnyi kimenetet, tehát ennek követésére legyen $l := 0$, majd:
   (a) Legyen: $i := (i + 1) \bmod 256$, majd $j := (j + S[i]) \bmod 256$
   (b) Cseréljük ki $S[i]$ és $S[j]$ értékét

(c) Legyen $T := S[(S[i] + S[j]) \bmod 256]$, T lesz a mostani generálási lépés 1 bájtnyi kimenete

(d) Inkrementáljuk l-t, ha $l < n$, akkor ugorjunk vissza (a)-ra

Egyéb követelmények

Elvárjuk, hogy a kulcssorozat ne legyen túl rövid (általában 40 bit az alsó határ). Jelen megvalósításban: $n \geq 5$, vagyis legalább 5 karakter hosszú kulcs kell.