Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozási Nyelvek és Fordítóprogramok Tanszék

Osztott rendszerek specifikációja és implementációja

IP-08bORSIG

Dokumentáció az 1. beadandóhoz

Mikus Márk István
CM6TSV

2017. október 02.

1. Kitűzött feladat

Adott egy input fájl, melyben bizalmas adatokat tartalmazó szöveget találhatunk. Arról azonban nem tudunk közvetlen meggyőződni, hogy az információk helyesek és nem kerültek módosításra, így a kriptográfiában megismertek módjára hashelnünk kell az adatokat, hogy ellenőrizhessük, hogy az a hivatalos ellenőrzőkóddal megegyezik e, hogy nyugodtan támaszkodjunk az ott olvasottra.

Az inputfájl felépítése az alábbi: Az első sorban egy N, nemnegatív egész szám olvasható, ezt követően összesen N sornyi szöveges információt találhatunk. Arra vonatkozóan, hogy egy sorban pontosan hány szó található, nincs közvetlen információnk, de mindegyikben legalább 1, és legfeljebb 100.

A fent említett hash függvényt szavakra definiáljuk. Az itt említett szavakba nem tartoznak bele a whitespace karakterek. A függvényünk minden szóra az alábbi módon működik:

```
'hashérték' : Nemnegatív egész szám := 0;
```

A szó egy 'c' betűjének hashkódja (`'kód'`) az alábbi módon áll elő:

```
'kód' : Nemnegatív egész szám := 1638; (0x666)
ha 'c' ASCII értéke páratlan:
    a 'kód'-ot bitenként shifteljük balra 11-et
egyébként:
     'kód'-ot shifteljük 6-ot bitenként balra.
'kód' := 'kód' XOR ('c' ASCII értéke BITENKÉNTI ÉSELVE 255-el); (0xFF)
Ha az így kapott 'kód' prím szám, akkor bitenként 'vagy'-oljuk,
amennyiben nem prím, 'és'-eljük össze 305419896-tal. (0x12345678)
```

Egy szó hashértéke a benne szereplő betűk hashértékének összegeként áll elő. Egy sor/szöveg hashértéke a benne szereplő szavak hashértékének szóközzel vett konkatenációjaként definiálható.

Feltehetjük, hogy a szövegben az angol ábécé betűit használó szavak (tehát magyar/orosz/koreai stb. készletbe tartozó karakterek nem), valamint általános írásjelek (pont, vessző, kérdő- és felkiáltójel, aposztróf, kettőspont stb.) találhatóak.

Annak érdekében, hogy hatékonyan működjön a programunk, a hasht ne szavanként, hanem soronként állítsuk elő, így egyszerre több sorhoz tartozó értéket párhuzamosan tudunk kiszámolni. A program olvassa be az adatokat, majd N folymatot indítva számítsa ki az egyes sorokhoz tartozó hash-értékeket, majd az így kapott adatokat írja ki az "*output.txt*" fájlba. A fő szál hasheléshez tartozó számítást ne végezzen!

Példa bemenet (*input.txt*):

4
Never gonna give you up, never gonna letyou down
Never gonna run around and desert you
Never gonna make you cry, never gonna say goodbye
Never gonna tell a lie and hurt you

Az ehhez tartozó kimenet (*output.txt*):

6312424 311932945 9453976 9449808 3158280 6312456 311932945 3158328 9449808 6308256
6312424 311932945 3158344 311937161 305633089 311937147 9449808
6312424 311932945 617549246 9449808 6308208 6312456 311932945 614399340 15758024
6312424 311932945 3162528 305624697 6304048 305633089 3162552 9449808

2. Felhasználói dokumentáció

A program egy konzolból futtatható alkalmazás, amellyet elindítva egy input.txt nevű, bemenő adatállományt feldolgozva előállít egy kimenő, output.txt nevű fájlt, benne az eredménnyel.

2.1. Rendszer-követelmények, telepítés

A programunk több platformon is futtatható, amely támogatja a (.exe) kiterjesztésű futtatható alkalmazások futtatását. Dinamikus függősége nincsen, bármelyik, manapság használt PC-n működik. Külön telepíteni nem szükséges, elég a futtatható állományt elhelyezni a számítógépen.

2.2. A program használata

A program használata egyszerű, külön paramétereket nem vár, így intézőből és parancssorból is indítható. A parancsorból való futtatáshoz, elég csak a parancssort megnyitni (*Windows-on: Start Menü > cmd*), majd a parancssorba belenavigálni abba a mappába ahol a futtatható állomány és az input.txt el van helyezve (Pl.: `C:\Never`). Ha a megfelelő mappában állunk elég begépelni a következőt: `<a futtatható állomány neve>.exe` (Pl.: Ha a futtatható állomány neve 'a' , akkor: `a.exe`).

Az intézőből való futtatáshoz elég kitallózni a megfelelő mappát, majd dupla kattintással elindítani a programot. A futtatható állomány mellett kell elhelyezni az *input.txt* nevű fájlt, mely a bemeneti adatokat tartalmazza, a fenti specifikációnak megfelelően. Figyeljünk az ebben található adatok helyességére és megfelelő tagolására, mivel az alkalmazás külön ellenőrzést nem végez erre vonatkozóan. A futás során az alkalmazás mellett

található *output.txt* fájl tartalmazza a kapott eredményt, ahol az *i*-ik sor a bemeneti fájl *i+1*-ik sorában található szöveg *hash*-elt alakját jelenti.

3. Fejlesztői dokumentáció

A program lényegét tekintve egy egyszerűbb C++ kód (C++ 11. szabvánnyal), egyetlen állományba szervezve benne a főporgram gericét adó `main()` továbbá a segédmetódusok:

- `isPrime(int number)`
- `hash_char(char& c)`
- `(hash_word(std::string& string)`
- `hash_line(std::string data)`

3.1. Megoldási mód

A kódunkat logikailag két részre bonthatjuk, egy fő-, illetve több alfolyamatra. A fő folyamatunkat a `main()` függvény fogja megvalósítani, mely beolvassa az inputként kapott fájl tartalmát, majd egy N méretű *string* vektort tölt fel annak adataiból. Az alfolyamatokhoz ennek a vektornak egy-egy elemét társítjuk, melyek elvégzik a szükséges számításokat, majd az így kapott eredményeket a főfolyamat fogja a kimeneti fájlba írni. Az alfolyamatok a kapott szöveget feldolgozza szavakra, majd a szavakat karakterekre, és sorrendben visszafelé kiszámítják a helyes hasheket. (Szavaknál hash-összeg, szövegnél hashek konkatenációja)

3.2. Implementáció

Az említett vektort `std::vector<std::string>` típussal fogjuk megvalósítani, míg az alfolyamatokat egy `std::future<std:string>` típusparaméterű vektorban fogjuk tárolni. Az egyes objektumok a kívánt visszatérési értéket is tartalmazni fogják, ezeknek külön memóriát nem kell foglalni. A szükséges N folyamatot az `std::async()` függvény segítségével, azonnal fogjuk új szálon indítani, paraméterül a végrehajtandó `hash_line()` függvényt, illetve a vektor egy elemét fogjuk átadni. Ez a függvény az egy sorhoz tartozó hash-t állítja elő úgy, hogy feldarabolja szavakra, majd egyesével hash-eli a szavakat a `hash_word()` metódussal, s ezek eredményét összefűzi szóközökkel elválasztva és ezt adja vissza a főfolyamatnak majd. A `hash_word()` függvény miután megkapja a kódolandó szót, összegzi a karaktereinek hash-ét (`hash_char()`) majd visszatér az összeggel. A `hash_char()` függvény a feladat leírásának megfelelő működéssel bír.

A feladat egyszerűsége révén egyetlen forrásfájlban, a `bead.cpp`-ben található a teljes implementációs kód.

3.3. Fordítás menete

A programunk forráskódját a `bead.cpp` fájl tartalmazza. A fordításhoz elengedhetetlen egy `C++11` szabványt támogató fordítóprogram a rendszeren. Ehhez használhatjuk az *MSVC*, *g++* és *clang* bármelyikét. A fordítás menete (6.4.0-s verziójú g++ használata esetén) a következő: `'g++ bead.cpp'`. A speciális, `-std=c++11` kapcsoló nem szükséges, mert alapértelmezés szerint ez a verziójú fordítóprogram már a C++11-es szabványt követi. Miután a kód lefordult, létrejön a forráskód melett egy *a.exe* fájl, amely a kész futtatható állomány.

3.4. Tesztelés

A program tesztelése során különböző méretű bemeneti fájlokkal futtattam. Az így kapott fájlok mindig a specifikációnak megfelelően. A programom minden esetben a tőle elvárt kimenetet állította elő, így a tesztesetek alapján helyesnek gondolhatjuk a működését.

A számítógépemben található 4 magból 3-at kikapcsolva, (időben) szekvenciális lefutást tudtam előállítani, 2 tesztesettel (4 soros és egy 40 soros). 1 processzormag esetén a 40 soros fájlra a futásidő ∼2.23 mp, míg a 4 soros fájlra ∼0.04 mp volt. Sorban visszakapcsolva a magokat az alábbiakat tapasztaltam:

	4 soros fájl	40 soros fájl
1 mag	∼0.04 mp	∼2.23 mp
2 mag	∼0.1 mp	∼0.23 mp
3 mag	∼0.11 mp	∼0.20 mp
4 mag	∼0.04 mp	∼0.43 mp

A magokat isszakapcsolása során a ∼2.23 mp egészen ∼0.43 mp-ig csökkent, így megállapíthatjuk, hogy a párhuzamosított program tényleg gyorsabban futott, mint a szekvenciális változata (Intel i5-4590 @ 3.30GHz processzorral).