LZW kódolás

Hogyan működik a tömörítő?

Adott egy karakterekből álló tömb, ez lesz a tömörítendő szöveg. És egy kódtába, ami tartalmazza a szöveg összes karakterét.

Jelen esetben a tömörítendő szöveg legyen ez: “dabbacdabbacdabbacdabbacdee”, a kódtábla: a,b,c,d,e értékeket fog tartalmazni.

1. Legyen egy **“csúszó ablak”**, aminek az eleje és a vége két index, azt jelzi, hogy melyik részét vizsgáljuk a tömörítendő szövegnek.
2. A csúszó ablak kezdeti indexe 0, a vég idexe 1 legyen.
3. Vegyük a csúszó ablak által mutatott rész stringet a tömörítendő szövegből. Ez most az első karakter és nézzük meg benne van-e a táblázatban? Az első karakter a 'd', ami a 4. helyen szerepel a kódtáblában.
4. Léptessük egyel a vizsgált csúszó ablak vég indexét 1-el, és most vizsgáljuk, meg az így kapott **'da'** benne van-e a kódtáblázatban?
5. Nincs benne, ezért a 'da'-t a kódtábla új 5. elemének szúrjuk be. Írjuk ki a 'd' indexét a 4 -et, azaz utoljára talált indexet.
6. A 'd'-t kiírtuk, ezért növeljük meg a csúszó ablak kezdő indexét 1-el (a 'd' hosszával).
7. Lépjünk a 3. pontra ameddig el nem fogy a kódolandó szöveg.

Az algoritmus írja ki a képernyőre a talált indexeket.

```
#include <stdio.h>
#include <string.h>

#define MAX_sxhS 100

char sxhTable[MAX_sxhS][10] = {"a","b","c","d","e"};

int tableElements = 5;

main()
{
    char text[] = "dabbacdabbacdabbacdabbacdee";
    char *p = text;
    int bufferEnd = 1;

    int lastFoundIndex = -1;
    while(*p != '\0')
    {
        char subStr[10];
        strncpy(subStr, p, bufferEnd);
        subStr[bufferEnd] = '\0';
        int foundIndex = isInsxhTable(subStr);
        if(foundIndex != -1)
```

```
        {
           bufferEnd++;
           lastFoundIndex = foundIndex;
           continue;
        }
        p += strlen(subStr) - 1;
        bufferEnd = 1;
        strcpy(sxhTable[tableElements++], subStr);
        printf("%d,", lastFoundIndex + 1);
    }
}

int isInsxhTable(char *s)
{
    for(int i = 0; i < tableElements; i++)
    {
        if(strcmp(sxhTable[i], s) == 0)
        {
            return i;
        }
    }
    return -1;
}
```

Entrópia számítás

Az informatikában, egy adatsor entrópiája kiszámítható. Ennek a nagysága arányos az adatban előforduló szimbólumok (karakterek) gyakoriságától. Ha az adatmintában sokféle elem, közel azonos gyakorisággal fordul elő, akkor az entrópia érték magasabb lesz. Ha az adatminta kis számú, azonos elemet tartalmaz, akkor az entrópia alacsonyabb érték lesz. Így tehát az entrópia kifejezi az adatsor véletlenszerűségét is. Ha az adatsor tartalmaz egy hosszabb (mondjuk 20 karakteres) jelszót, akkor az adatsor elemzésével kitalálhatjuk, hol helyezkedik el ez a jelszó. Ezért sem szabad jelszavakat tárolni egyszerű szövegként adatbázisokban, illetve forráskódban.

Írjuk egy programot, ami egy karaktertömbben elhelyezett szöveget tartalmaz. Ebbe a szövegbe másoljunk bele egy ~20 karakteres jelszót, ami egy online jelszógenerátorral készítettünk. Definiáljunk egy ablakot, aminek mentén az adatsoron végigmegyünk és minden lépésben kiírjuk az entrópia értéket. Figyeljük meg mit tapasztalaunk a jelszó közelében.

```
#include <stdio.h>
#include <math.h>

float calculateEntropy(unsigned int bytes[], int length);

char sample[] = "Hova lett a tarka szivárvány az égről?"
```

```
"Hova lett a tarka virág a mezőkről?"
"Hol van a patakzaj, hol van a madárdal,"
"S minden éke, j8g^Q!b:SFg!6#KL?hD9N"
"kincse a tavasznak s nyárnak"
"Odavan mind! csak az emlékezet által"
"Idéztetnek föl, mint halvány síri árnyak."
"Egyebet nem látni hónál és fellegnél"
"Koldussá lett a föld, kirabolta a tél";

int main()
{
    unsigned int byteCounter[256];
    const int windowWidth = 20;

    for(int i = 0; i < sizeof(sample) - windowWidth; i++)
    {
        memset(byteCounter, 0, sizeof(unsigned int) * 256);

        char *p = &sample[i];
        char *end = &sample[i + windowWidth];

        while(p != end)
        {
            byteCounter[(unsigned char)(*p++)]++;
        }
        float entropy = calculateEntropy(byteCounter, windowWidth);
        printf("%d - %.3f\n", i, entropy);
    }


}


float calculateEntropy(unsigned int bytes[], int length)
{
    float entropy = 0.0f;

    for (int i = 0; i < 256; i++)
    {
        if (bytes[i] != 0)
        {
            float freq = (float) bytes[i] / (float) length;
            entropy += -freq * log2f(freq);
        }
    }
    return entropy;
}
```