Kovács Bertalan (KZI8JV) – Nagy Házi Feladat c nyelven

LZW Tömörítő

A projektben szereplő fájlok tartalma és szerepe:

- main.c: Ebben a fájlban vannak a fő függvények meghívva (compress, decompress) és alapszintű hibakezelések vannak implementálva (a feldolgozott argumentumok alapján elindítja a megfelelő funkciót, amennyiben hiba lépett fel, leállítja a program futását). Ebben a fájlban logika nincs megfogalmazva, függvények nincsenek a main-t kivéve.
- compress.h (header) és compress.c (implementáció): A tömörítéshez szükséges függvények és globális változók (compress, initDecompressDict és a globális scope-ban lévő compressDictionary) vannak itt deklarálva és implementálva. Itt van még a tömörítéshez szükséges struktúra (compressData).
- decompress.h (header) és decompress.c (implementáció): A kitömörítéshez szükséges függvények és globális változók (decompress, initDecompressDict, freeDecompressDict, appendToDecompressDict és a globális scope-ban lévő decompressDict) és a kitömörítéshez szükséges struktúra (decompressData) vannak deklarálva és implementálva.
- includes.h: Alap dolgok include-olása szerepel benne. A célja az, hogy minden fájl elején csak 1x kelljen beinclude-olni 1 fájlt, így szebben néz ki a fájlok eleje és kevesebb a kódismétlés.
- IOUtility.h (header) és IOUtility.c (implementáció): An N bitre való kiírás és N bitről való beolvasáshoz szükséges függvények és az ehhez szükséges adatstuktúra itt van deklarálva és implementálva (IOData, printInNBits, printRemaining, readNBits és egy tesztfüggvény: testReadWrite). Itt van még definiálva két globális változó: MAX_BITS (a szótárakban lévő elemek maximum hány bitből állnak) és a MAX_VALUE (mi a legnagyobb MAX_BITS biten tárolható szám + 1).
- parseArgs.h (header) és parseArgs.c (implementáció): A futtatási paraméterek (argumentumok) feldolgozásáért felelős függvények és az argumentumok adatstruktúrája ezekben a fájlokban van deklarálva és implementálva (commands, parseArguments).

A függvények szerepe és működése:

- main (int argc, char** argv): Meghívja a *parseArguments* függvényt, majd ennek eredménye alapján meghívja a *compress*, vagy a *decompress* függvényeket a megfelelő paraméterekkel.
- compress (FILE* inputFile, FILE* outputFile): Az inputFile adatait tömöríti és az eredményt az outputFile-ba kiírja. Inicializálja a *compressDictionary*-t, majd végigmegy a bemeneti fájlon és byte-onként beolvassa. Minden elemre megnézi, hogy adtunk-e már értéket az {eddigi bemenet + jelenlegi karakter} stringnek. Amennyiben igen, akkor továbbmegyünk ezen az úton. Ezt faként kell elképzelni, aminek az egyes ágain megyünk lefelé. Minden string-nek adunk egy egyedi értéket, amit kitömörítésnél ugyanezzel a módszerrel elő tudunk állítani. Amikor elértünk egy ág végére (azaz nem létezik az {eddigi bemenet + jelenlegi karakter} string a listában), akkor értéket adunk ennek az elemnek (mindig a lista hossza lesz az új elem értéke, majd növeljük a lista hosszát egyel), majd kiírjuk az eddigi bemenethez rendelt értéket N biten. Akkor növeljük az N értékét, ha nem tudjuk már kiírni a lista elemszámát N biten (azaz a {lista elemszáma} > 2^N). Amikor tele van a lista, akkor újra inicializáljuk a listát és kezdjük a folyamatot elölről. Amikor nincs több bemenetünk, akkor a maradék eltárolt adatot ki kell írnunk, mert lehet olyan eset, hogy van még eltárolt adat, csak még nem volt lehetőség kiírni.
- initCompressDict (void): Inicializálja a compressDictionary-t. Feltölti 0-val az összes elemet és beállítja az első 256 elemet értékét {0-255}-re. Mivel tudjuk, hogy egy karakter {0-255} között van, ezért tudjuk, hogy minden string után maximum 256 -féle karakter következhet. Azt is tudjuk, hogy a 0. helyen lévő elem 1 hosszú, ezért {string + következő elem} nem mutathat a 0. helyre. Ebből az következik, hogy használhatjuk a 0-t, mint NULL jelzés, azaz végpontként.
- decompress (FILE* inputFile, FILE* outputFile): Az inputFile adatait kitömöríti és az eredményt az outputFile-ba kiírja. Inicializálja a *decompressDict*-et, majd beolvassa az első karaktert. Az első karakter mindig 8, a következő mindig 9 byte-on van tárolva, a kezdő értékeket ennek megfelelően állítjuk be. A tömörítéssel ellentétben, itt el kell tárolnunk azt is, hogy az adott kódú stringhez milyen string tartozik. Beolvasunk N bitről adatot, majd megnézzük, hogy szerepel-e ez az adat már a szótárunkban. Ha nem az még nem jelent hibát, megnézzük, hogy az a speciális eset-e (lásd: [1]). Ha ezek közül egyik sem, akkor hibás a bejövő adat, leállítjuk a kitömörítést. Az N-et a tömörítéshez hasonlóan növeljük és az újrainicializálás is hasonlóan folyik. Az egyetlen kivétel a stringek, mert azokat dinamikusan foglaljuk, ezért lista ürítésnél azokat free-elni kell. A kitömörítés végén free-eljük a lefoglalt memóriát.
- initDecompressDict (void): A függvény hasonló az *initCompressDict*-hez, egy kivétellel: lefoglal memóriát egy karakternek az első 256 helyen (az egy hosszú stringet) és beállítja a hosszukat 1-re.
- freeDecompressDict (void): Felszabadítja az összes manuálisan foglalt memóriát (a stringek, amelyek a *decompressDict*-ben vannak).
- appendToDecompressDict (int lastIndex, int sourceIndex, char lastChar): A decompressDict lastIndex. pozíciójára létrehoz egy új stringet, olyan hosszút, mint ami a sourceIndex. helyen van + 1 hosszút. A +1. helyre kerül a lastChar. Az új string felépítése {decompressDict[sourceIndex].word + lastChar}. Megjegyzés:

1 https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch#Decoding_2

Ezek a stringek nem null-terminálták, ezért el van tárolva a hosszuk is. Pont ezért nem használhatóak string-re kitalált műveletek. A másolást memcpy-vel végezzük.

- printInNBits (IOData* data, int number, int bits): Kiírja *number*-t, ami egy *bits* bites szám. A kimeneti fájl a *data*-ban van eltárolva. A *data*-t pointerként adjuk át, mert oda lesz eltárolva a maradék adat és az, hogy ez hány biten van eltárolva.
- printRemaining (IOData* data): Kiírja a maradék adatot a kimenetre, jobb oldalt 0-val kitöltve.
- readNBits (IOData* data, int bits): Beolvas egy számot, ami *bits* biten van eltárolva. A maradék beolvasott adatot a *data*-ban tárolja el, ezért kell azt pointerként megadni.
- testReadWrite (void): Teszteli az itt lévő függvények működését úgy, hogy random számokat random biten kiír, majd beolvas.
- parseArguments (int argc, char** argv): A függvény megkapja a main-be beérkező argumentumokat és feldolgozza őket.

A használt adatstruktúrák:

- decompressData
  - char* word: Manuálisan lefoglalt string, az adott értékhez rendelt string. Nem null-terminált, ezért meg kell adni a string hosszát is (lásd: *length*)
  - int length: A string hossza
  - uint16_t next: Egy adott elem .next[{karakter}]-je egy listaindex. Ezen a helyen található a {string + {karakter}}. Ez egyben ennek az új stringnek a kiírt értéke is.
- compressData:
  - uint16_t value: A jelenlegi string kódja.
  - uint16_t next: Lásd: *decompressData*
- IOData:
  - FILE* file: A jelenlegi be- / kimeneti fájl.
  - dataStored: Az utolsó beolvasásból / kiírásból maradt adat.
  - bitsStored: A dataStored hány biten van tárolva.
- commands:
  - bool isCcompress: Megszabja a műveletet. Amennyiben true, tömörítés, ha false akkor kitömörítés.
  - char* inputFileName, outputFileName: A ki- és bemeneti fájlok nevei. Csak a bemeneti fájlnév megadása kötelező.
  - bool error: Ha nincs minden kötelező adat megadva, akkor ez true lesz, ellenkező esetben false. Ha ez true, akkor olyan adat hiányzik, ami kötelező a program futásához.
  - bool info: Amennyiben a program a „--help” paraméterrel lett futtatva, nem hiba, ha nincs meg minden kötelező paraméter. Ebben az esetben ez a bool true lesz és a program hibaüzenet nélkül áll le.

A program használata:

lzw.exe <paraméterek>

Paraméterek:

- -c, -d: Tömörítés, vagy kitömörítés a művelet. -c a tömörítés (compress), -d a kitömörítés (decompress).
- -I, -O: Megadják a be- és kimeneti fájlok neveit. Használatuk: -I <fájlnév>, -O <fájlnév>

A paramétereket tetszőleges sorrendben lehet megadni. A kimeneti fájl nevét nem kötelező megadni.