S. 68.

Szabó 928 Attila 11. o. t.

Leőwey Klára Gimnázium, Pécs

A program C nyelven, a Dev-C++ 4.9.9.2 verziójával készült (fordító: MinGW-GCC).

A program első lépésben végighalad a bemenetként kapott fájlon, és egy regiszterben megszámolja az egyes bájtok számát. Az állományban előforduló bájtokat aztán fába fűzi, majd az így adódó Huffman-kódokat a kimenetre írja. Ezeket a műveleteket rendre a stat, huffman, writeout függvények végzik el.

A program felhasználja a stdio.h, string.h (a kódok összeállításához) és az inttypes.h (64 bites egészek használata) fejállományokat. A faszerkezetet a node típusnevű struktúrával hozom létre. Ez tartalmazza az elem

által leírt bájtot (ha az elem levél, azaz a fájl egyik bájtját jelképezi), az elem gyakoriságát, kapcsolatait (ősét, bal és jobb leányát; ezek alkalmasak a befűzöttség jelzésére is) és Huffman-kódját (szintén csak leveleknél). Legfeljebb 256-féle bájt lehetséges, amelyekhez könnyen meggondolhatóan 255 szülőelem kell a fába fűzéshez, így a fának legfeljebb 511 eleme lehet: ezeket tömbben tárolja a tree tömb.

A stat eljárás a bemeneti inp fájlt bájtonként elolvassa, s ennek alapján az egyes bájtok gyakoriságát meghatározza (ez a freqs tömb megfelelő elemeibe kerül).

A huffman eljárás a Huffman-fát építi fel. A freqs tömb elemein végighaladva, a nemnulla elemekhez hozzárendel egy faelemet a tree tömb elején, eközben meghatározza a levelek leaves számát. Alapértelmezésként az ős sorszámát −1-re állítja, ezzel azonosíthatók a fába még nem szervezett elemek. A faelemeken ezután a feladatban részletezett algoritmust hajtjuk végre. Mivel minden lépésben eggyel kevesebb szabad faelem marad, és az algoritmust egy elemig (a gyökérelem) kell végezni, azért leaves−1 lépést kell végrehajtani. Ezeket indexeli 0-tól (leaves−2)-ig a program. Minden lépésben az addig már létrejött elemek közül (ezek közül az utolsó

könnyen végiggondolhatóan a leaves+i−1 indexű, ennek megfelelő a ciklusindexelés) keresi meg a két legkisebb gyakoriságú szabad (−1 ősű) elemet, ezekhez hoz létre egy közös őselemet, ez lesz a fa új eleme. Mivel a szabad elemek száma minden lépésben eggyel csökken, így az eljárás végén adódó, utolsó elem lesz a gyökér, amelynek gyakorisága az összes bájt gyakoriságának összege, azaz az állomány hossza: ez kerül a len változóba.

A writeout eljárás készíti el a kimenetet. Első lépésben a fa minden elemének meghatározza a Huffman-kódját.

A C nyelv stringkezelő függvényeinek használatával ennek legegyszerűbb módja, hogy az adott elemtől a gyökér felé haladva minden lépésnél feljegyezzük a lépés irányát és azokat ilyen sorrendben fűzzük össze. Nyilvánvaló, hogy így az elem Huffman-kódjának a fordítottját kapjuk meg, ezért azt meg kell fordítani. A rendezési elvet egy egész kifejezés szerinti növekvő rendezéssel meg lehet valósítani: ezt a bájt értékének és a kódhossz 256-szorosának összegeként kaphatjuk meg: mivel egy bájt értéke legfeljebb 255, így a kód hosszának változásakor 256-os növekedés magasabb

prioritású lesz a bájt értékénél. Az ord segédtömb 1. oszlopába ezek a kódok kerülnek, a 0. oszlopba pedig a hozzátartozó levél sorszáma. A tömböt ezután rendezzük a kódok szerinti növekvő sorrendbe, így a 0. oszlopban a levelek indexei a kiíráskor elvárt sorrendbe kerülnek. A kimeneti fájlba ezután írjuk ki az elvárt formában a kódokat (nyomtathatónak a 33–126 ASCII kódokat tekintjük). A kódolás utáni bitek számát bájtonként külön számolva könnyen megkaphatjuk, a kódolás előtti bitszám pedig a bájtok számának 8-szorosa.

A főprogram megnyitja az állományokat (a bemenetet binárisan, a kimenetet szövegesen), sorban végrehajtja a fenti eljárásokat, majd az állományok lezárása után kilép.

A program annak érdekében, hogy nagy fájlokat is kezelhessünk, a gyakoriságértékeket és a bitek számát 64 bites egész típusban tárolja (uint64_t), így bármekkora, reálisan előforduló fájl kezelhető. A 64 bites egészek használatához szükséges inttypes.h fejállomány teszi lehetővé a minimumkereséshez szükséges maximális kezdőérték megadását és a 64 bites egészek kiíratását.

Mivel a stat eljáráson kívül százas nagyságrendű adatot kezelünk, így a kevésbé hatékony, egyszerűbb algoritmusok (egyszerű cserés rendezés,. . . ) is helytállóak. Nagyobb fájlok esetén a futásidő döntő részét a statisztika elkészítése teszi ki: a stat eljárás kb. 6 MB adatot dolgozott fel másodpercenként az általam tesztelésre használt számítógépen (2,66 GHz, 2 GB RAM).