Unyi Péter Álmos

Nagy házifeladat – Huffman-kódolás

program dokumentációja

**Rövid leírás:**

A nagy házifeladatom egy parancssorból futtatható, szövegeket veszteségmentesen tömörítő program, amely a Huffman-algoritmust használja fel a tömörítésre.

**Felhasznált könyvtárak és függőségek:**

* stdio.h;
* stdlib.h;
* stdbool.h;
* string.h;
* ctype.h;
* debugmalloc.h;

**Típusok:**

* Gyakorisag: struct a szövegben található karakterek gyakoriságának a tárolására.
  1. char betu: a karakter, aminek a gyakoriságát tárolja a .gyakorisag paraméter;
  2. int gyakorisag: a karakter előfordulásának a száma az adott szövegben.
* Binfa: bináris fa tárolására;
  1. int szam: a bináris fában található levél esetén a levél által tárolt karakter előfordulásának a száma, belső adatpont esetén pedig a két alatta lévő gyerek .szam paramétereinek az összege;
  2. char betu: a bináris fában található levél esetén a kódolt karakter, belső adatpont esetén pedig null;
  3. char \*ut: a Huffman-kódolt fában az adatpontok elérési útvonala bináris kódban kifejezve, ami alapján el lehet jutni a kódolt ponthoz a gyökértől kezdve;
  4. struct Binfa \*bal, \*jobb: a bináris fa egy gyökerének a két levelére mutató pointer, melyeknek típusa szintén Binfa.
* Kod: a kitömörítéshez használt struct, amiben a kódolt karakterek bináris kódjait („elérési útvonalait”) lehet tárolni;
  1. char betu: a kódolt karakter;
  2. char \*kod: a bináris kód, ami a karakterhez tartozik és ami alapján megkereshető a fában.

**Fájlok és funkciók:**

**huffman.c**

* Főprogram, kommunikáció a felhasználóval, ez választja szét a betömörítés és a kitömörítéshez szükséges programokat, illetve kezeli az esetleges felmerülhető problémákat a programhoz beírt utasítások kapcsán.
* Argumentumok: bemeneti opciók, char \*argv[] array-ben tárolva;
* Visszatérési érték: 0, a felmerülő problémákat kiírja a képernyőre.

**betomorites.c**

1. ***FILE \*fajl\_letrehoz(char \*fajlnev)***

* Függvény fájl létrehozására. Létrehoz egy fájl, aminek az elérési útvonalát argumentumként kapta meg, majd visszatér a létrehozott filepointerrel.
* Argumentumok:
  + char \*fajlnev: a fájl elérési útvonala.
* Visszatérési érték:
  + sikertelen létrehozáskor: NULL;
  + sikeres létrehozáskor: FILE \*fajl: a fájlt tartalmazó pointer.

1. ***int fajlmeret(FILE \*fajlp)***

* Függvény egy fájl méretének a meghatározására. Eltárolja és visszatér az argumentumként kapott filepointer byteokban számolt méretével.
* Argumentumok:
  + FILE \*fajlp: a fájlt tartalmazó pointer.
* Visszatérési érték:
  + int fajlmeret: a fájl mérete byteokban.

1. ***char \*fajl\_olvas(char \*fajlnev)***

* Függvény egy file tartalmának az olvasására. Megnyitja az argumentumként kapott elérési útvonalon lévő file-t és karakterenként kiolvassa belőle a tartalmát, ezt eltárolja egy char pointerben, majd ezzel tér vissza.
* Argumentumok:
  + char \*fajlnev: a file elérési útvonala.
* Visszatérési érték:
  + sikertelen file megnyitáskor: NULL;
  + sikeres olvasáskor: char \*string: a szöveget tartalmazó char pointer.

1. ***int fajl\_ir(char \*fajlnev, char \*tartalom)***

* Függvény egy tartalom file-ba írására. Megnyitja az argumentumként kapott elérési útvonalon található file-t és beleírja az argumentumként kapott tartalmat, majd visszatér a file-ba írt karakterek számával.
* Argumentumok:
  + char \*fajlnev: a file elérési útvonala
  + char \*tartalom: a file-ba írandó szöveges tartalom char pointere
* Visszatérési érték:
  + sikertelen file megnyitáskor: NULL;
  + sikeres file-ba íráskor: int kar\_count: a file-ba írt karakterek száma

1. ***void bemenet\_eldont(char opcio, char \*arg1, char \*arg2, char \*arg3)***

* A betomorites.c file vezérlő függvénye, ez vezérli a betömörítés folyamatát. Az opcio argumentumból megkapja a betömörítés típusát, ami f (fileból olvas be) és s (begépelt szövegből olvas be) típus lehet, ez alapján eldönti, honnan kell beolvasni a tömörítendő szöveget. Megszámolja a karakterek gyakoriságát a beolvasott sztringben, létrehoz egy bináris fát, és megállapítja a karakterek tömörített kódjait, majd a kódolt szöveget is létrehozza, ezen utóbbiakat a binaris\_fa.c file-ból vett függvények segítségével.
* Argumentumok:
  + char opcio: a beolvasás típusa, értékei: f (fileból olvas be), s (begépelt szövegből olvas be);
  + char \*arg1: opcio = f esetén: a beolvasandó file elérési útvonala, opcio = s esetén a begépelt szöveg;
  + char \*arg2: az egyedi kimeneti file választó opciója, ha NULL, nem kell létrehozni egyedi file-t;
  + char \*arg3: ha szükséges létrehozni egyedi kimeneti file-t, akkor a program az arg3 paraméterként kapott elérési útvonalon található file-ba tömörít, alapértelmezése: tomoritett.hcf.

**binaris\_fa.c**

1. ***Gyakorisag \*rendezes\_struct(Gyakorisag \*t, int meret)***

* Függvény, ami egy Gyakorisag típusú tömböt tud rendezni, az előfordulások száma alapján növekvő sorrendben;
* Argumentumok:
  + Gyakorisag \*t: az előfordulásokat tartalmazó Gyakorisag típusú tömb;
  + int meret: a tömb mérete.
* Visszatérési érték:
  + Gyakorisag \*t: a rendezett tömb.

1. ***Gyakorisag tombelem\_torol\_struct(Gyakorisag \*t, int p, int meret)***

* Funkció egy tömbelem törlésére egy Gyakorisag típusú listából, a p-edik pozícióról.
* Argumentumok:
  + Gyakorisag \*t: maga a tömb, ahonnan törlésre kerül egy elem;
  + int p: a törlendő elem pozíciója;
  + int meret: a tömb mérete.
* Visszatérési érték:
  + Gyakorisag \*t: a tömb, a törölt elem nélkül.

1. ***void sorban\_kiir(Binfa \*gyoker)***

* Rekurzívan végigmegy a bináris fán, és kiírja az elemeit preorder algoritmussal. Debug függvény.
* Argumentumok:
  + Binfa \*gyoker: a bináris fa első elemét tartalmazó pointer.
* Visszatérési érték: -

1. ***Binfa \*lefoglal(int szam)***

* Lefoglal egy új elemet a bináris fában, a következő adatokkal:
  + int .szam = szam (a függvény paramétere);
  + char .betu = NULL;
  + char .ut = NULL;
  + Binfa \*bal, \*jobb = NULL;
* Argumentumok:
  + int szam: a bináris fa adatpontjához tartozó gyakorisági szám.
* Visszatérési érték:
  + Binfa \*uj: a létrehozott és memóriában lefoglalt memóracím.

1. ***bool level(Binfa \*gyoker)***

* Megvizsgálja, hogy levél-e az adott adatpont a bináris fában és bool értékkel tér vissza.
* Argumentumok:
  + Binfa \*gyoker: a bináris fa egy adatpontja;
* Visszatérési érték:
  + bool level: igaz, vagy hamis, hogy az adatpont levél-e.

1. ***char \*tomb\_megnovel(char \*t)***

* Funkció, ami megnöveli char tömb méretét, úgy hogy újabb elemeket lehessen beszúrni.
* Argumentumok:
  + char \*t: a megnövelendő tömb.
* Visszatérési érték:
  + char \*uj: az lefoglalt memória, amibe további elemeket tudunk beszúrni.

1. ***Binfa \*beszur(Binfa \*gyoker, int osszeg, int a, int b, char ca, char cb)***

* Függvény az összegfa felépítésére. Megkeres két gyakoriság számának az összegét a fában és ha létezik, továbbá az adott adatpont levele a fának, beszúrja a két adatot, a gyakoriságokhoz tartozó karakterekkel együtt.
* Argumentumok:
  + Binfa \*gyoker: a bináris fa első elemét tartalmazó pointer;
  + int a, int b: a két gyakorisági szám;
  + char ca, char cb: a két gyakorisághoz tartozó karakterek.
* Visszatérési érték:
  + Binfa \*gyoker: a bináris fa első elemét tartalmazó pointer.

1. ***Binfa \*rekurziv\_tomb(Gyakorisag \*n, int meret)***

* Rekurzívan végigmegy egy tömbön, és minden rekurzióban rendezi a tömböt, összedja az első két elem értékét, beszúrja a tömbbe és amikor elfogynak az elemek, vagy egy elem marad, létrehoz egy bináris fát és meghívva a beszur függvényt, beszúrja a bináris fába a tömb első két elemét, visszatér a létrehozott és adatokkal feltöltött bináris fával.
* Argumentumok:
  + Gyakorisag \*n: a karaktereket és gyakoriságukat tartalmazó tömb;
  + int meret: az előző tömb mérete.
* Visszatérési érték:
  + Binfa \*r: a létrehozott és adatokkal feltöltött fa.

1. ***void binaris\_fajl\_ir(char \*t, FILE \*f)***

* Egy tetszőleges méretű, 1-es és 0-s elemű tömböt ír fájlba, az eredeti tömböt byte méretű rész-sztrintgekre felosztva és a benne található 1-es, 0-s elemek szerint. Ha a rész-sztring kisebb méretű mint 8, feltölti 0-s karakterekkel.
* Argumentumok:
  + char \*t: a bináris kódot tartalmazó tömb;
  + FILE \*f: a fájlra mutató pointer.
* Visszatérési érték: -

1. ***void kodok\_fileba(Binfa \*gyoker, FILE \*f)***

* Fájlba írja a Huffma-kódolt betűk bináris reprezentációját. Rekurzívan, preorder sorrendben megy végig a fán, ha az adott pont levél a fában, fájlba írja a karaktert és a hozzá tartozó „elérési útvonalat” a fában.
* Argumentumok:
  + Binfa \*gyoker: a bináris fa első elemét tartalmazó pointer;
  + FILE \*f: a fájlra mutató pointer.
* Visszatérési érték: -

1. ***char \*faban\_keres(Binfa \*gyoker, char c, char \*t, FILE \*f)***

* Végigmegy rekurzívan a bináris fán és megkeres benne egy karaktert, ha megtalálta, a paraméterként átadott \*t tömbbe másolja az „elérési utat” és visszatér ugyanezzel a tömbbel.
* Argumentumok:
  + Binfa \*gyoker: a bináris fa első elemét tartalmazó pointer;
  + char c: a fában keresett karakter;
  + char \*t: a tömb, amit címként vesz át a függvény, hozzáfűzi a karakter „elérési útvonalát”;
  + FILE \*f: a fájlra mutató pointer.
* Visszatérési érték:
  + char \*t: a karaktertömb, belemásolva a keresett karakter „elérési útvonala”.

1. ***Gyakorisag \*gyak\_szamol(char \*szoveg, int \*cel\_meret)***

* Függvény, ami megszámolja egy szövegben található karakterek gyakoriságát, majd ezeket eltárolja egy Gyakorisag típusú tömbben és visszatér ezzel.
* Argumentumok:
  + char \*szoveg: a szövegtömb, ahol meg kell számolni a kraktereket;
  + int \*cel\_meret: a kimeneti tömb mérete, amit a függvény címként kap meg és ezt később a program felhasználja.
* Visszatérési érték:
  + char \*cel: a céltömb, ahová bemásolásra kerültek a betűk és gyakoriságaik.