Hatékonyság

Szempontok

- Idő - mennyi idő alatt fut le a program, illetve annak adott része?
  A végrehajtási idő nagy részben a hardvertől, illetve a használt algoritmusoktól függ
  Egy algoritmus vagy eljárás végrehajtási ideje függhet a bemeneti paraméterektől (például a számok nagyságától), így beszélhetünk egy program minimális, maximális és átlagos végrehajtási idejéről.
- Helyfoglalás (mennyi helyet foglal a program a merevlemezen),
  A helyfoglalás tekintetében beszélhetünk a programkód méretéről, vagyis a háttértáron való helyfoglalásról, valamint a változók memóriában való helyfoglalásáról

  A helyfoglalás lényegesen függ a használt adatszerkezetektől illetve a feldolgozó algoritmustól.

  A programkód hosszát az algoritmikus elemek (elágazások, ciklusok) számának csökkentésével, vagy metódusok alkalmazásával (a többször leírt részeket csak egyszer írjuk le) csökkenthetjük.
- Bonyolultság - mennyire átlátható a forráskód?
  Elsősorban a programozó felkészültségétől függ.

Megközelítés

- Lokális (részegységek vizsgálata)
  A programkód egyes részleteinek formális átalakításával kapcsolatosan lokális hatékonyságról beszélünk
- Globális (teljes program vizsgálata)
  Az egész algoritmus átgondolásával, módosításával, foglalkozunk ebben az esetben. Mivel ez költséges dolog, érdemes már a tervezésnél figyelembe venni, átgondolni a hatékonyság szempontjait

A hatékonyság mérése:

- A végrehajtási idő mérése történhet akár egy beágyazott objektum segítségével.
- A memóriaigényt az operációs rendszer szervizprogramjaival, pl. a Feladatkezelővel mérhetjük.
- A fájlok méretét minden fájlkezelő program jelzi.
- A bonyolultság mérése a legösszetettebb dolog: általában mondhatjuk, hogy minél rövidebb a forráskód és minél könnyebb megérteni, annál kisebb a bonyolultsága.

  Ezenkívül jellemezhetjük:
  - az elágazások, ciklusok számával
  - a vezérlési szerkezetek egymásba ágyazottságának mélységével
  - a feltételekben a logikai műveletek számával

A hatékonyság szempontjából fontos tevékenységek:

- **Keresések:**
  - A **lineáris** keresések végrehajtási ideje arányos a sorozat elemszámával, ezt nagyságrendileg nem tudjuk javítani formális átalakításokkal. Rendezett sorozatban a lineáris keresés is gyorsabb, de itt is a sorozat elemszámával arányos. A bináris keresés ennél nagyságrenddel gyorsabb: a sorozat elemszámának logaritmusával arányos (bár kis elemszám esetén lehet, hogy lassabb, mint a lineáris keresés, hiszen a ciklusmagban több feltétel-vizsgálat szerepel).
  - A **bináris** keresés algoritmusa először a tömb középső elemét hasonlítja össze a keresett értékkel (ha páros számú elem van, akkor a két középső közül az egyiket vesszük). Ha a középső elem megegyezik a keresett értékkel, akkor befejeztük a keresést. Ha a középső elem nagyobb, mint a keresett érték, akkor a tömb első felében (a középső elem előtt) folytatjuk a keresést, egyébként a második felében (a középső elem után) vizsgálódunk. A maradék elemekre ugyanezt az eljárást ismételgetjük, így a részsorozatok mérete egyre csökken. Közben vagy rátalálunk a keresett értékre, vagy elfogynak az elemek.

  A keresés tehát rendezett sorozatban sokkal gyorsabb lehet, viszont maga a rendezés költséges algoritmus.

- **Rendezések:**

  Többféle rendezési algoritmus ismert, a módszerek gyorsasága, erőforrásigénye függ az adatok rendezetlenségétől. Az algoritmusok hatékonyságánál a helyfoglalást, az összehasonlítások számát és a mozgatások számát szoktuk mérni. Általában a tárigény és a végrehajtási idő egymás rovására csökkenthető. A konkrét feladat dönti el, hogy melyik jellemzőt tekintjük fontosabbnak. Egy hordozható program esetén a helyfoglalás fontosabb szempont lehet, mint a végrehajtási idő. Rendezési algoritmusok a teljesség igénye nélkül:
  - egyszerű cserés rendezés,
  - minimumkiválasztásos rendezés,
  - beillesztéses (kártyás) rendezés,
  - buborék rendezés,
  - keveréses rendezés,
  - gyorsrendezés (quicksort).
    A quicksort algoritmusban a rendezendő sorozatot helyben kettéválogatjuk. A sorozat eredetileg első eleme lesz az úgynevezett „elválasztó” elem, nevezzük ezt a továbbiakban X-nek. A szétválogatás során a sorozat azon elemeit, amelyek kisebbek X-nél, X elé helyezzük, a sorozat azon elemeit, amelyek nagyobbak X-nél, X mögé helyezzük. A szétválogatás végén X új helyre kerül, és ez a hely a rendezettségnek megfelelő, végleges hely. A továbbiakban folytatjuk ugyanezt az eljárást a sorozat bal és jobb oldali részsorozatával. Ezekben a szétválogatást követően szintén a helyére kerülnek az elválasztó elemek, és már a részsorozatok bal és jobb oldali részeivel folytatjuk tovább a rendezést, és így tovább, amíg még van rendezhető bal és jobb oldali rész. (Rekurzív algoritmus, egy sorozat rendezését két kisebb részének a rendezésére vezetjük vissza.)

  A C# Array.Sort() metódusa 16-nál kevesebb elemszám esetén a beillesztéses rendezést használja, nagyobb elemszám esetén a quicksort rendezést.

Hatékonyság növelése

- **Végrehajtási idő csökkentése:**

  A végrehajtási idő csökkentése érdekében azokat a program-részeket érdemes vizsgálni, amelyeket nem csak egyszer-kétszer hajtunk végre, hanem sokszor, mert ezek gyorsításával érhetünk el jelentősebb hatást. Ezek általában ciklusok belsejében, vagy rekurzív függvényhívásokban szerepelnek.

  Egy ciklus átlagos futási idejét a következő képlet határozza meg:

  futási idő = lépésszám*egyszeri lefutás ideje

  A gyorsabbra írást tehát kétféleképp érhetjük el:

  - a ciklusok lépésszámát csökkentjük
    A keresés/kiválogatás során egy sorozathoz rendelünk egy értéket/sorozatot. Ha van a sorozatnak olyan része, amelyben a keresett elem garantáltan nem lehet, akkor csökkenhet a ciklus lépésszáma. Ezt csak a konkrét feladat ismeretében lehet meghatározni. A feladatban szereplő naplófájlban az adatok természetes módon időrendben követik egymást, ezért ha lehetőségünk van a keresésnél a dátumra szűrő feltételt adni, akkor jelentősen lecsökkenthetjük annak a sorozatnak a méretét, amit a memóriában tárolni kell, és amiben keresni fogunk. Így egyszerre csökkenthetjük a helyfoglalást és a végrehajtási időt
  - a ciklusmagok egyszeri végrehajtási idejét csökkentjük
    Elágazás kiküszöbölése indexeléssel: pl. megszámlálási feladatoknál: számoljuk meg az egyes IP-címekhez tartozó bejegyzéseket – megoldható úgy, hogy egy asszociatív tömbben (C#-ban Dictionary az IP-címek a kulcsok, és a hozzá tartozó értékek a bejegyzések darabszámai. Ekkor a megfelelő darabszámot a megfelelő kulccsal közvetlenül tudjuk növelni.

    Kivételes eset kiküszöbölése: kiválasztás, eldöntés esetén „ütköző” használatával elkerülhetjük az össszetett feltétel alkalmazását. Ilyenkor a tömb végére elhelyezünk egy olyan elemet, ami megfelel a keresett tulajdonságnak így a feltételes ciklus biztos megáll a tömb utolsó eleménél, így nem szükséges a tömbindex ellenőrzése.

    Eldöntés tétele ütköző használatával:

```
Tömb(végsőindex+1)=ütköző
i=kezdőindex
ciklus amíg Tömb(i) nem T tulajdonságú
    i=i+1
ciklus
vége
Van=(i<=végsőindex)
```

    A futási idő 20-30%-kal csökkenthető.

- **A helyfoglalás csökkentése:**

1. a program változóinak helyigényét csökkentjük

   A változók helyfoglalásának csökkentése szempontjából a sorozatokra/kollekciókra kell koncentrálni (tömbök, listák). A helyfoglalást csökkenthetjük:

   - a sorozat elemszámának csökkentésével
     - **Az indexes változók kiküszöbölése** akkor lehetséges, ha a bennük tárolt adatokra a programnak nincs tartósan szüksége, csak valamely számításban vesznek részt, de végül csak az utolsóként kiszámolt elemre van szükségünk. Például, ha azt szeretnénk meghatározni, hogy egy adott számítógépen milyen gyakorisággal következtek be bizonyos események.
     - **Feldolgozás helyben:** Előfordul, hogy egy sorozatból egy másik sorozatot kell készíteni úgy, hogy a régire a feldolgozás végén már nincs szükség: pl. rendezés, kiválogatás, szétválogatás).

       Ekkor olyan algoritmusokat használjunk, amelyek a feldolgozásokat helyben végzik.
     - **Ciklusok összevonása:** a beolvasás ciklusa összevonható a különböző feldolgozási ciklusokkal (pl. megszámolás, kiválogatás) így kiküszöbölhetők az indexes változók vagy csökkenthető az elemszámuk
   - a sorozat egy elemének a helyfoglalását csökkentve
     - A leghatásosabb módszer, amikor egy adat tárolását megszüntethetjük, mert mások értékeiből meghatározható. Pl. személyi számból a születési dátum. Ebben a feladatban nincs olyan adat, amire ezt alkalmazhatnánk.
     - Ha tárolnunk kell egy adatot, akkor válasszunk olyan adatreprezentációt, amelyben egy adatelem a lehető legkisebb helyet foglalja el. Pl. dátum tárolása szöveg helyett számszerű típusban pl. C#ban DateTime struct-ban – ennek mérete 8 byte, míg egy string helyfoglalása annyiszor 2 byte, ahány karakterből áll

2. a programkód helyfoglalását csökkentjük

   A programkód méretét akkor tudjuk csökkenteni, ha bizonyos részek a programban többször szerepelnek: ilyenkor úgy alakítjuk át a kódot, hogy az ismétlődő kódrészeket csak egyszer írjuk le.

   - Az azonos funkciókat eljárásba foglaljuk
   - Ciklusok összevonása
   - Programkód adattá transzformálása: programok gyakori része a futásuk elején tájékoztató szöveg kiírása, vagy igény szerint help megjelenítése. Ha a kiírandó szöveget a programkódban tároljuk (literálként vagy konstansként), akkor egy karakteres képernyőnyi szöveg (2000 karakter) is kb. 2000 byte-ot foglalhat a memóriában. Sokkal jobb megoldás, ha ezeket a szövegeket háttértáron tároljuk, és szükség esetén olvassuk be és írjuk a képernyőre.