Programozás I.

1. előadás: Algoritmusok alapjai

Sergyán Szabolcs
sergyan.szabolcs@nik.uni-obuda.hu

Óbudai Egyetem
Neumann János Informatikai Kar
Alkalmazott Informatikai Intézet

2016. szeptember 12.

Tartalom

Tartalom

1. példa

Autómosás

1. Előmosás. Az autót mosószeres lével bespriccelik.
2. Kefés mosás. A forgó kefék letisztítják az autót.
3. Öblítés. Az autót tiszta vízzel leöblítik.
4. Szárítás. Az autót levegő áramoltatással megszárítják.

Egymást követő utasítások sorozata.

2. példa

„Haladó” autómosás

1. Ha aktívhabos mosásra fizettünk elő, akkor az autót bespriccelik aktív habbal. Különben csak mosószeres lével spriccelik be az autót.
2. Ha alváz mosásra is előfizettünk, akkor az alvázat is végigspriccelik aktív habbal.
3. Ha kerékmosásra előfizettünk, akkor forgó kefék letisztítják az autót, a kerekeknél pedig speciális keréktisztító kefék mossák a kerekeket. Különben csak a forgó kefék letisztítják az autót.
4. Az autót tiszta vízzel leöblítik.
5. Ha előfizettünk viaszvédelemre, akkor az autót forró viasz réteggel bevonják.
6. Az autót levegőáramoltatással megszárítják.

Megjelennek döntési pontok, ahol elágazhat a végrehajtás.

Euklideszi algoritmus

Két pozitív egész szám legnagyobb közös osztója

1. Adott két pozitív egész szám, jelöljük ezeket m-mel és n-nel. A kettő közül legyen m a nagyobbik.
2. Osszuk el m-et n-nel, az osztás maradékát jelöljük r-rel.
3. Ha r értéke 0, azaz m osztható n-nel, akkor az algoritmus végére értünk. Ilyenkor a két szám legnagyobb közös osztója az n értékével egyezik meg, az algoritmus pedig befejeződik.
4. Ha r értéke nem 0, akkor m-be tároljuk el az n jelenlegi értékét, n-be pedig az r értékét. Majd ugorjunk vissza a 2. pontra.

Egyes lépések többször is végrehajtásra kerülnek.

Euklideszi algoritmus

Két pozitív egész szám legnagyobb közös osztója

1. Adott két pozitív egész szám, jelöljük ezeket m-mel és n-nel. A kettő közül legyen m a nagyobbik.
2. Osszuk el m-et n-nel, az osztás maradékát jelöljük r-rel.
3. Ha r értéke 0, azaz m osztható n-nel, akkor az algoritmus végére értünk. Ilyenkor a két szám legnagyobb közös osztója az n értékével egyezik meg, az algoritmus pedig befejeződik.
4. Ha r értéke nem 0, akkor m-be tároljuk el az n jelenlegi értékét, n-be pedig az r értékét. Majd ugorjunk vissza a 2. pontra.

Konkrét példa

m	**n**	**r**
150	36	6
36	6	0

Algoritmus fogalma

Mi az algoritmus?

- Az algoritmus egy olyan „gép”, amely valamilyen bemenetekből meghatározott lépéseken keresztül előállít valamilyen kimenetet.
- Bemenetek: Az algoritmus elején már ismertek.
- Kimenetek: A bemenetek által egyértelműen meghatározottak.
- Az algoritmus működése jól meghatározott, azaz determinisztikus.
- Az algoritmus egyértelmű, jól definiált lépésekből áll, melyek száma minden esetben véges.

Algoritmus alkotás

Legfontosabb lépések

- A folyamatot elemi lépésekre bontjuk.
- Figyelembe vesszük az összes felmerülő lehetőséget.
- Ügyelünk, hogy az algoritmus véges sok lépésben véget érjen.

Tartalom

Változók

Mi az a változó?

- Az algoritmusban használt értékeket tárolni kell az algoritmus futása során.
- Változók a bemenetek, a kimenetek és az algoritmus megvalósításához szükséges lokális változók.
- Minden változónak van neve, amellyel egyértelműen tudunk rá hivatkozni.
- Minden változónak van típusa is.

Típusok

Algoritmus leírásnál használt típusok

- Egészek (pszeudokódban jelölés: **egész**)
- Logikaiak (pszeudokódban jelölés: **logikai**)
- Általános típus (pszeudokódban jelölés: **T**)

Értékadás

$\leftarrow$

- Az a változónak értékül adjuk az 5-öt: $a \leftarrow 5$
- Az értékadás bal oldalán egyetlen változó állhat.
- A jobb oldalon egy érték, egy változó vagy egy kifejezés állhat.
- Mindig a baloldali változó kapja meg a jobboldali értéket, visszafelé nem történik értékadás.

Csere

$\leftrightarrow$

- Az a és b változó megcserélésének lépései:

  $seg\acute{e}d \leftarrow a$

  $a \leftarrow b$

  $b \leftarrow seg\acute{e}d$

- Rövidebb jelölés: $a \leftrightarrow b$

Kifejezések

Mi az a kifejezés?

- Számkifejezésben számokat tartalmazó változók, konkrét számértékek és ezeket összekapcsoló matematikai műveletek szerepelhetnek.
- Példa:
$$\frac{bal - jobb}{2} + 3,$$
ahol *bal* és *jobb* egy-egy változó.
- A számkifejezések kiértékelése a matematikában megismert szabályok szerint történik.

Logikai típus

Értékek és műveletek

- Két lehetséges érték:
    - **igaz**
    - **hamis**
- Műveletek:
    - Negálás: $\neg$
    - És kapcsolat: $\wedge$
    - Vagy kapcsolat: $\vee$

Logikai típus

Negálás

l logikai értéke	$\neg l$ logikai értéke
hamis	**igaz**
igaz	**hamis**

És valamint vagy kapcsolat

l_1 log. értéke	l_2 log. értéke	$l_1 \wedge l_2$ log. értéke	$l_1 \vee l_2$ log. értéke
hamis	**hamis**	**hamis**	**hamis**
hamis	**igaz**	**hamis**	**igaz**
igaz	**hamis**	**hamis**	**igaz**
igaz	**igaz**	**igaz**	**igaz**

Logikai típus

Rövidzár kiértékelés

- Ha egy és kapcsolat első tagja **hamis**, akkor a második tag már nem is kerül kiértékelésre, mert a teljes és kapcsolat **hamis** lesz.
- Ha egy vagy kapcsolat első tagja **igaz**, akkor a második tag már nem is kerül kiértékelésre, mert a teljes vagy kapcsolat **igaz** lesz.

Tartalom

Tömbök

Mi a tömb?

- Több azonos típusú értéket lehet egy változóban eltárolni.
- A tömb elemeire indexeléssel hivatkozunk.
  A harmadik elem elérése: $x[3]$.
- Az indexelés 1-től indul.

x:	2	3	5	7	11	13	17	19
	↑	↑	↑	↑	↑	↑	↑	↑
	1	2	3	4	5	6	7	8

Tömb létrehozása

$$x \leftarrow \text{LÉTREHOZ}(\mathbf{egész})[8]$$

Kétdimenziós tömbök

Példa

	1	2	3	4	5	6
1	3	5	8	4	7	1
2	2	9	3	0	6	4
3	6	8	2	1	1	5
4	4	1	0	2	7	8

Elemre hivatkozás: $x[2,5]$

Kétdimenziós tömb létrehozása

$$x \leftarrow \textsc{Létrehoz}(\textbf{egész})[4,6]$$

Tartalom

Szekvencia

Utasítások egymás utáni végrehajtása

$x \leftarrow$ Létrehoz(**egész**)[3]
$x[1] \leftarrow 5$
$x[2] \leftarrow 8$
$x[3] \leftarrow 11$

Utasítás feltételes végrehajtása

ha $a < 0$ **akkor**

  $a \leftarrow -a$

elágazás vége

$b \leftarrow \sqrt{a}$

Elágazás

Kétirányú elágazás

ha $a < 0$ **akkor**

$b \leftarrow \sqrt{-a}$

különben

$b \leftarrow \sqrt{a}$

elágazás vége

Elágazás

Többirányú elágazás

ha $a > 0$ **akkor**

$\quad b \leftarrow \sqrt{a}$

különben ha $a < 0$ **akkor**

$\quad b \leftarrow \sqrt{-a}$

különben

$\quad b \leftarrow 0$

elágazás vége

Ciklus

Elöltesztelős ciklus

ciklus amíg feltétel

    utasítások

ciklus vége

Példa

ciklus amíg $r \neq 0$

    $m \leftarrow n$

    $n \leftarrow r$

    $r \leftarrow m \mod n$

ciklus vége

Ciklus

Hátultesztelős ciklus

ciklus
    utasítások
amíg feltétel

Ciklus

Számlálós ciklus

ciklus *ciklusváltozó* $\leftarrow$ *kezdetiérték*-től *végérték*-ig
    utasítások
ciklus vége

Példa

$x \leftarrow$ LÉTREHOZ(**egész**)[5]
ciklus $i \leftarrow$ 1-től 5-ig
    $x[i] \leftarrow i^2$
ciklus vége

Tartalom

Strukturált programozás

- Strukturált programnak tekintjük azokat a programokat, amelyek csak a megengedett elemi programokat tartalmazzák a megengedett programkonstrukciók (vezérlési szerkezetek) alkalmazásával.
- Elemi programok
  - üres program
  - értékadás (állapot változtatás)
    jele: $\leftarrow$
- Megengedett konstrukciók
  - szekvencia
  - elágazás
  - ciklus
- Bizonyítható, hogy a fenti szabályok megtartásával minden algoritmussal megoldható feladatra adható is megoldás.

- Az elkészítendő programot egyetlen utasítással szeretnénk megoldani.
- Ha ez nem sikerül, akkor több utasítással próbálkozunk, melyek egyes részfeladatokat valósítanak meg.
- Addig folytatjuk ezt a részfeladatokra bontást, míg minden részfeladatot sikerül megvalósítani elemi utasítások felhasználásával.

Összefoglalva: A teljes feladatot részekre bontjuk, majd ezeket a visszavezetés módszerével megoldjuk.

Függvények vs. eljárások

Függvény

- Egy konkrét algoritmust valósít meg.
- Más függvények vagy eljárások meghívhatják.
- Vannak bemenetei.
- Van visszatérési értéke (kimenete). Ezt a **vissza** kulcsszóval adjuk meg.

Eljárás

- Egy konkrét algoritmust valósít meg.
- Más függvények vagy eljárások meghívhatják.
- Vannak bemenetei.
- Kimenete csak paramétereken keresztül lehet. Ehhez **címszerint** adjuk át a paramétert.

Euklideszi algoritmus

Függvénnyel megvalósítva

Bemenet: m – **egész**, n – **egész**
Kimenet: n – **egész**

```
függvény LNKO(m, n)
    r ← m mod n
    ciklus amíg r ≠ 0
        m ← n
        n ← r
        r ← m mod n
    ciklus vége
    vissza n
függvény vége
```

Euklideszi algoritmus

Eljárással megvalósítva

Bemenet: m – **egész**, n – **egész**

Kimenet: n – **egész**

eljárás $\mathrm{LNKO}(m,$ **címszerint** $n)$
    $r \leftarrow m \mod n$
    ciklus amíg $r \neq 0$
        $m \leftarrow n$
        $n \leftarrow r$
        $r \leftarrow m \mod n$
    ciklus vége
eljárás vége

Függvény hívása

Relatív prím vizsgálat

Bemenet: x – **egész tömb**, n – **egész** (*tömb mérete*), *value* – **egész**
Kimenet: y – **logikai tömb**

függvény RelatívPrímVizsgálat(x, n, *value*)
    $y \leftarrow$ Létrehoz(**logikai**)[n]
    ciklus $i \leftarrow$ 1-től n-ig
        ha LNKO($x[i]$, *value*) $= 1$ **akkor**
            $y[i] \leftarrow$ **igaz**
        különben
            $y[i] \leftarrow$ **hamis**
        elágazás vége
    ciklus vége
függvény vége

Tartalom

Mikor „jó” egy algoritmus?

- Megbízható: Helyesen működik.
- Kiszámítható: Adott bemenet esetén mindig ugyanazt a kimenetet szolgáltatja.
- Egyszerű: Könnyen megérthető.
- Hatékony:
  - Kevés memória igénye van.
  - „Kevés” a futási ideje.

Előadásra járó hallgatók számának meghatározása

- Hányan vannak itt a teremben?
- Hogyan tudnánk ezt hatékonyan megszámolni?

Szó keresése szótárban

Hogyan tudunk hatékonyan szót keresni egy szótárban?

Algoritmus lépésszáma

- Egy algoritmus lépésszáma alatt azt értjük, hogy hány elemi műveletet (értékadás, összehasonlítás, stb.) kell végrehajtani adott bemenet mellett.
- Egy algoritmus futási ideje függ az algoritmust megvalósító programtól, a programozási nyelvtől, a számítógéptől, stb.
- Algoritmuselméletben a futási időt a lépésszámmal jellemezzük.

Futási idő

Hány lépésben fut le az algoritmus?

Bemenet: x – **egész tömb**, n – **egész**
Kimenet: db – **egész**

```
függvény NullátAdóElempárokSzáma(x, n)
    db ← 0
    ciklus i ← 1-től (n − 1)-ig
        ciklus j ← (i + 1)-től n-ig
            ha x[i] + x[j] = 0 akkor
                db ← db + 1
            elágazás vége
        ciklus vége
    ciklus vége
    vissza db
függvény vége
```

Futási idő

Feltétel kiértékelések száma

$$(n-1)+(n-2)+\ldots+2+1=\frac{(n-1)+1}{2}\cdot(n-1)=\frac{n\,(n-1)}{2}$$

Értékadások száma

- Legjobb esetben: 0-szor
- Legrosszabb esetben: $\frac{n(n-1)}{2}$-szer

Algoritmus futási ideje

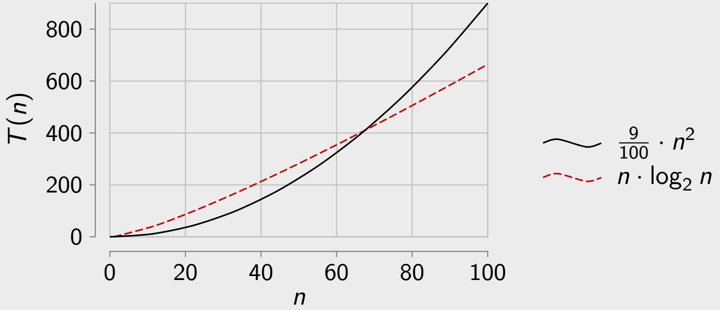
$$O\left(n^2\right)$$

Futási idő analízisének fajtái

- Legrosszab eset analízis
  - $T(n)$: a maximális futási idő, amely bármely n elemű sorozat esetén a rendezéshez legfeljebb szükséges
- Átlagos eset analízis
  - $T(n)$: a várható futási idő, amely az n elemű sorozatok rendezéséhez szükséges
- Legjobb eset analízis
  - Magunkat verjük át, ha ezzel foglalkozunk

Futási idő

Futási idők összehasonlítása



Nagy ordó jelölés

Azt mondjuk, hogy a $T(n)$ futási idő $O(f(n))$-es, ha létezik olyan c konstans, hogy elég nagy n értékek esetén

$$T(n) \leq c \cdot f(n).$$

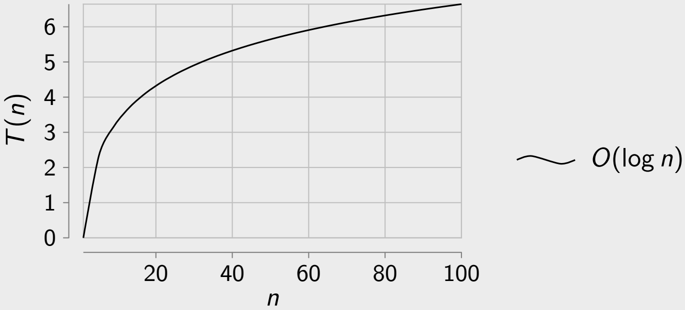
Futási idő

Gyakran előforduló nagyságrendek

Futási idő nagyságrendje	Futási idő nagyságrendjének elnevezése
$O(1)$	Konstans
$O(\log n)$	Logaritmikus
$O(n)$	Lineáris
$O(n \log n)$	Logaritmetikus
$O(n^2)$	Négyzetes
$O(n^3)$	Köbös
$O(2^n)$	Exponenciális

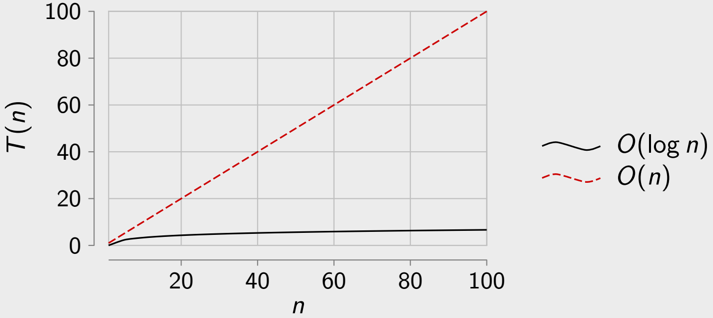
Futási idő

Gyakori nagyságrendek



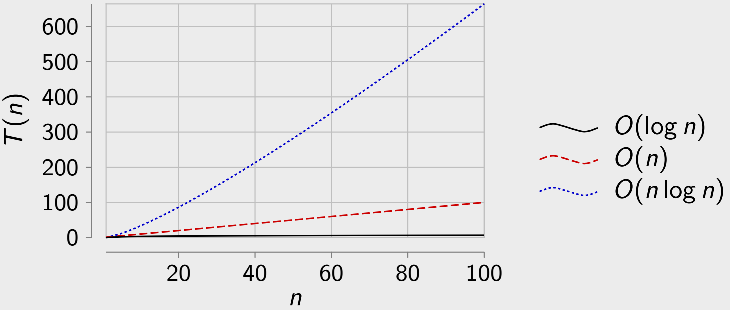
Futási idő

Gyakori nagyságrendek



Futási idő

Gyakori nagyságrendek



Gyakori nagyságrendek