1. előadás

11. 11.

Algoritmus hatékonysága:

a; végrehajtási idő: annál hatékonyabb egy program, minél kisebb a végrehajtási idő

b; program tárigénye:

- adat tárigénye
- kód tárigénye

Belső vezérlésű: a gépet az operatív tárban tárolt program vezérli, míg előzőleg csak adatokat tároltak

Végrehajtási idő esetében 2 lehetőség van. Nem mindegy, h egy adathalmazt milyen lépésekkel dolgozunk fel és hogy milyen adattípusokat ábrázoljuk.

1; eljárás szorzás_1 (a, b, c: egész);

  c := a * b;

e vége;

a végrehajtási idő az a, b, c típusától függ

2; eljárás szorzás_2 (a, b, c: egész);

  változó i: egész;

  c := ∅;

  ciklus i := 1..a

    c := c + b;

  c vége;

e vége.

· Végrehajtási időt az operandusok értéke is befolyásolja.

Nem mindegy hány összeadást végez 500x3, 3x500

eltér a végrehajtási idő

nem csak az értékétől függ, hanem a sorrendjétől is.

· Ha eltekintünk attól, h. a hatékonyság eltér az operandusok nagyságától és sorrendjétől, akkor jobb a 2. eljárás.

· A rövidebb algoritmust könnyebb átlátni, nehezebb hibát véteni. Ha nem működik a program, meg kell keresni a hibát, ki kell javítani => rövidebbet könnyebb javítani.

· Fontos a tagolás -> jobban érthetőbb, átláthatóbb

TÁRIGÉNY

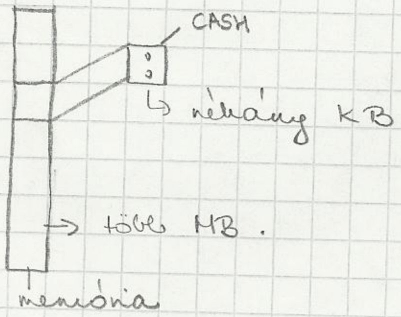
VÉGREHAJTÁSI IDŐ

BONYOLULTSÁG

· Egy program mire teljesül => ált. a tulajdonságok egyikére jól „hat" a többi ellen dolgozik. (U. gyors és nagy a tárigény...)

· A COMMODORE 64 processzora nem tud szorozni és nincs lebegőpontos számábrázolás

Végrehajtási idő csökkentése:



• Betesszük a programkódnak azt a részletét, ami belefér ⇒ innen olvassa ki a proc. az adatokat.

• A CASH drága, ezért kicsi. A tár és a CASH között gyors az adatforgalom

• Nem végigolvassa, hanem kering egy darabig ⇒ CIKLUSOKAT tartalmaz.

• Annak egy részét emeljük be a CASH-be, ami fontosabb, ciklusokból áll

• A sebesség növelésének szempontjából a ciklusoknál kell keresni.

Tfh: Adva van egy ciklus, végrehajtási idő ⇒ t.

n-szer végrehajtódik : $T = n \cdot t$

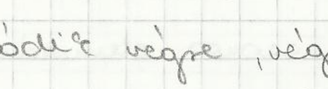
• Vagy a ciklusmag végrehajtási számát csökkentjük (találunk olyan $m < n$ ⇒ $T > T_1$, ha a ciklus m-szer hajtódik végre, végrehajtási idő $t \cdot m \cdot t$)

vagy $t' < t$ (↳ új idő) $T > T_2 = n \cdot t'$

Ha sikerül olyan ΔT-t találni, ahol különböznek a ciklus végrehajtási idői ⇒ $n \cdot \Delta t$ időt takaríthatunk meg.

$[\Delta t + t' = t \quad n \cdot \Delta t]$

BUBORÉK rendezés:


szomszédos elemek összehasonlítása

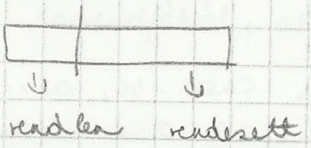
BUBORÉK rendezés I.

végrehajtás számának csökkentése

ha nincs csere ⇒ VÉGE

BUBORÉK rendezés II.

azt figyeli, hol volt az utolsó csere.



Végrehajtási idő csökkentése: lineáris, strázsás keresés

Sorozatok részre osztása (ciklusmag lefutási száma)

- egy sorozathoz 1 értéket rendelő algoritmus
- spec tulajdonságok

· Tudjuk, v könnyen el tudjuk dönteni, hogy a keresett elem a sorozat mely részében található. Esetleg egymásba ágyazott részsorozatokban (lin keresés)

· A sorozat átrendezhető, felosztható kb. azonos elemszámú részekre (Quicksort)

Feladat:

Hat. meg egy x elem A(n) sorozatban elfoglalt helyét!
(tudjuk benne van egyszer)

Mego: lin kiválasztás, a keresett elem helye k

```
eljárás kiválasztás_1 (k: egész, x: elem);
    k := 1;
    ciklus amíg A(k) ≠ x
        k := k+1;
      a vége;
    c vége;
  e vége
```

	max	átlagos	min
összehasonlítások száma	n	$\frac{n}{2}$	1

rendezett sorozat

```
eljárás bin_2 (e:egész; x:elem);
c:=1; u:=n;
ciklus
    k:=(e+u) div 2;
    ha A(k)<x akkor e:=k+1;
    ha A(k)>x akkor u:=k-1;
c vége A(k)=x esetén;
e vége e
```

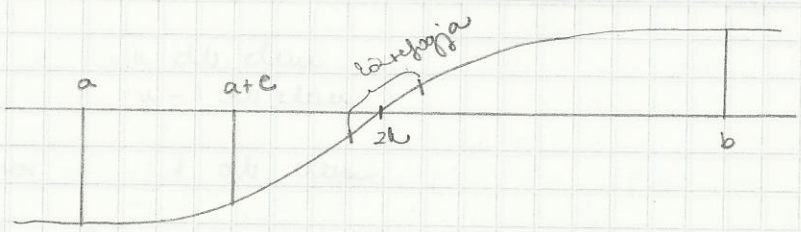
	max	átlagos	min
összehas. száma	$\log_2 n$	$\log_2 n/2$	1

---

f fgv $[a,b]$-n folytonos

Kat megy az intervallum 1 pontját, amely e-nél kevésbé tér el a zk-tól.

lin



eljárás gyökkeresés_1 (a, b , ε, e : valós);

k := a + e ;   { e : közelítés hibája, ε : keresőérték }

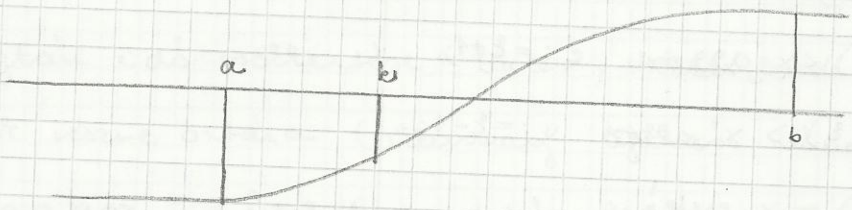
ciklus

amíg f(ε) * f(ε−e) > 0   { amíg jelet vált }

k := ε + e ;

c vége;

eljárás vége;



a feltételben vizsgálja és ahhoz viszonyítja ⇒ lineárishez hasonlít

eljárás gyökkeresés _2 (a, b, ε, e : valós);

ciklus

k := (a+b) /2;

ha f(a) * f(ε) > 0 akkor a := k

különben ha f(b) * f(ε) > 0 akkor b := ε ;

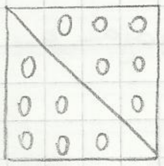
h vége

c vége (b−a ≤ e) vagy f(ε) = 0 esetén

c vége;

2. előadás

Elemszám csökkentése – lefutás csökkentése

– 

Diagonális mátrix

Van alsó- és felső Δ mátrix ⇒ biztosan tudjuk, h 0.

Hiányosan kitöltött struktúráknak nevezzük.

- Mindenhol a főátlón kívül K elem van. Spec. eset: K=0.
- A sorozatságban elfoglalt helye szerint ő meg tudjuk határozni az értékét.

$M[i,j] = 0$, ha $i \neq j$

- egyszerű megfeleltetni egy mátrixnak egy vektort.

Írnak egy fgvt, aminek 2 paramétere van: $M[i,j]$

ha $i = j$ akkor

$M := v[i]$

különben → nem kell sorfolytonosan bejárni, elég

$M := \emptyset$; a v vektort bejárni.

h vége; $n^2 - n$ db elem elhagyható.

- A felső v. alsó Δ mátrixot is szokás vektorban tárolni.

1 sor	n db elem
2. sor	n − 1 db elem
⋮	
utolsó sor	1 db elem

- ábrázolható olyan vektorban, amelynek van $\frac{n(n+1)}{2}$ eleme van.

- normál mátrix esetén n^2 elemet kenne a vektornak, így csak $\frac{n(n+1)}{2}$

- megadunk egy (i,j) indexpárt. Ha az főátló alatti elem ⇒ megadja, ha nem, akkor ő állítja elő, és visszaadja a mátrixban a megfelelő értéket.

- $C := A + B \quad \Rightarrow \quad c_{ij} := a_{ij} + b_{ij}$

  Elegendő csak megfelelő elemeket lejátszani

- A adattárolásban fontosak a prímszámok.

  ↓

  Fontos a gyakorlati jelentőssége
  (aki a legnagyobb prímet ismeri, több adatot tud feltörni)

„Def": Azok a pozitív egészek a prímek, amiknek nincs valódi osztójuk.
(2 osztója van: 1 és önmaga)

↓

így lehet eldönteni, h. prím-e vagy nem.

Számoljuk meg hány osztója van!

Venni kell a szám lehetséges osztóinak sorozatát 1-től a-ig
(ha a szám „a")

- A def. miatt vegyük csak 2-től a-1-ig ⇒ összehasonlítjuk az elemünkkel 2-vel.

- Tovább szűkítem a sorozat elemeinek számát:

„Def." Ha van valódi osztója ⇒ NEM PRÍM. Keressünk neki legalább 1 valódi osztót.
Ezáltal könnyebb a nem prímeket meghatározni.)

- Ha „a" $\in \mathbb{Z}$ -nél akarjuk eldönteni, h. prím-e, és feltételezzük, h. $\left(\frac{a}{2} < k < a\right)$ „k" osztója a

$$\frac{a}{2} < k < a \qquad /\cdot 2 \Rightarrow \text{legkisebb szám, aminek van}$$

$$a < 2k < 2a$$

ZED a , k nem lehet osztója ezáltal az „a"-nak

⇓

elegendő az osztókat a 2-től

a szám feléig egészéig vizsgálni.

$\sqrt{64}$

64 osztói : $2 < 4 < 8 < 16 < 32$

$\sqrt{24}$

24 osztói : $2 < 3 < 4 < 6 < 8 < 12$

⇓

ez azt jelenti, hogy nem kell tovább menni, mint a szám

$\sqrt{\ }$-e, mert megtaláltunk minden osztót. Vagy ténylegesen, vagy a párján keresztül.

· Ha találtunk egy osztószámot, növelhetjük 2-nesére, mert a párja is ott van.

I.

```
i := 2;
ciklus  míg (i < n) ∧ (n mod i ≠ ∅)
    i := i + 1;
c vége;
```

sorra vettük az első valódi osztóig a számokat.

- Ha az azt i-vel való osztási maradék ∅ ⇒ találtunk 1 valódi osztót.
- ha i < = n ⇒ n nem prím, megállt n mod i = 0 miatt ⇒ van valódi osztója
- prím, ha i = n (önmaga az osztója)

II. módosíthatjuk az i < n feltételt

$2 * i < n$ a feléig menjen a ciklus ($i < n/2$)

III. $2 * i < n$ -t módosítsuk:

$i * i \leq n$ ($i \leq \sqrt{n}$)

Feladat: 2-től n-ig írassuk ki a prímeket!

2..n

Lássuk be, h. elegendő a √-ig nézni!

~~(2)~~	(3)	4	(5)	6	(7)	8	9	10	(11)	12	(13)	14	15	16	(17)	18	(19)	20	21	22	(23)	24	25	26

- A 2-ről tudjuk, h. prím. Húzzuk ki a 2-vel osztható számokat!

  ⋮

  stb.

- A program megírásához nem kell eltárolni a számokat. Elegendő egy logikai érték sorozata → csak azt kell tudni egy számról, h. kihúztuk-e vagy nem.
- Ha nincs áthúzva: TRUE
  ha át van húzva: FALSE
- Töltsük fel 2..n a logikai sorozatot igaz értékekkel!
- A 2 prím → 2-es lépésközönként állítsuk át hamisra az értéket.
- Keressük meg a sorozaton belüli első elemet, ami igaz. haladjunk ‚k' lépésközzel, állítsuk át az értéket FALSE-ra...
  ... STB.
- Meddig kell keresni a következő igaz elemet?
- A felén túl már nem érdemes menni, mert az túlmutat a sorozat végén

```
E tömb [2..n] logikai;
CIKLUS i:=2..n
    e[i]:= igaz;
c vége;
i:=2;
ciklus míg (i<n)
```

Hf: ide milyen feltételt lehet írni