Melyik a gyorsabb? ((i) vagy (ii) → 310)

A második.

„Megspóroluk" 1 összehasonlítást és egy egészrész fgv-t.

F: Előfordulási gyakoriságok számlálása

- ▷ Kockadobások
- ▷ Egyedi karakterek

indexelték pl.: 1-1 kockadobást 1-1 index jelentett

- ▷ Karakterek csoportjai
- ▷ Intervallumok → pl.: jegyek

pl.: Számoljuk meg! hány magas és mély hangrendű magánhangzó és hány hosszú és rövid ... stb. van?

Hosszú magán- és mássalh.-at ill. rövid magánh.- és mássalhangzókat számolunk.

	INDEX	
0		
:		
64	0	@
65	2	A
66	3	B
		: stb.
255		

Ha betűket számolunk, nem érdemes 65 kódú karakternél kisebbekkel foglalkozni. (Nem biztos, hogy lesz benne grafikus karakter...)

0: Egyéb

1: hosszú magánh.
2: rövid magánh.
3: mássalh.
4: írásjelek

$\underbrace{\hspace{6em}}_{\text{csoportosítjuk}}$

Van a szövegnek egy i-dik karaktere, ezt meg kéne számlálni.

szöveg[i];

Ord(szöveg[i]);
↓
előállítja a karakterkódját

Az index tömb karakteredik kódja ⇒ csoportba sorolja

$$\text{index}\,[\underbrace{\underbrace{\text{ord}\,(\underline{\text{szöveg}[i]})}_{@}}_{\underbrace{64}_{0}}]$$

inc (DB[index[ord(szöveg[i])]])

↳ itt tároljuk, hogy hány db van

→ eggyel léptetjük, ha találtunk oda való elemet.

Azért jó, mert nem kell feltételvizsgálattal szenvedni ⇒

az már más, hogy az indexelés mennyi időt vesz igénybe

Probléma:

- Fennáll a tévesztés a lehetősége: 1 elemet 2 halmazba

sorolunk, vagy sehová nem soroljuk

- Meg kell nézni minden halmazt, h. benne van-e.

Kell legalább 4 feltétel-kiértékelést csinálni

F: Írjunk olyan programot, amely logikai kapuk működését szimulálja!

AND

		bemenet 0	1
kimenet	0	0	0
	1	0	1

Azt szeretnénk, ha olyan rutint írnánk, ami

← így viselkedik. Van kimenete és bemenetek.

```
Eljárás And_1 (Be1, Be2, K : egész);
    Ha (Be1=1) és (Be2=1) akkor K:=1
                          különben K:=0;
e vége;
```

Tároljuk el az igazságtáblázatot egész típusú kétdimenziós tömbben!

And_: tömb [0..1, 0..1] egész;

And

	0	1
0	0	0
1	0	1

Or

	0	1
0	0	1
1	1	1

```
Eljárás And_2 (Be1, Be2, K: egész);
    K := M_And [Be1, Be2];
e vége;
```

Végrehajtási idő csökkentése — ciklusmag végrehajtási idejének csökkentése

Kivételes esetek kiküszöbölése

Máskor a ciklusmag végrehajtását olyan feltételvizsgálat lassítja, amelyek feltétele egyszer (ritkán) válik igazzá

→ az algoritmust esetleg az adatszerkezetet úgy kell átírni, hogy a kivétel megszűnjön.

T: Döntsük el, hogy a sorozatnak van-e T tulajdonságú eleme!

eldönt.: ciklus míg ($i \in N$) $\wedge$ (A[i] nem T tul) ⇒

kiválaszt: ciklus míg (A[i] nem T tul.) (strázsás is)

lineáris keresés

Kivételes eset: a sorozatnak nincs T tulajdonságú eleme

strázsás keresés

20-30%-kal csökkenhet a futási idő

F: Unió, metszet, különbség ... stb.

F: Adott n elemű $\{a_n\}$ sorozat elemeit felhasználva a $\{b_n\}$ sorozat elemeit állítsuk elő.

$$b_i := \begin{cases} a_i, & ha\ i = 1 \\ \frac{a_i + a_{i-1}}{2}, & ha\ i > 1 \end{cases}$$

```
Eljárás sorozat_1 (n : egész, A, B : sorozat);
   ciklus i := 1 .. n
       ha i = 1    akkor B[i] := A[i]
                   különben B[i] := (A[i] + A[i-1]) /2 ;
   ciklus vége;
evége ;
```

· Nincs szükség feltételvizsgálatra.

Végrehajtási idő csökkentése – ciklusmag végrehajtási idejének csökkentése

Kivételes elemek kiküszöbölése (II.)

F: Logikai kapu működésének szimulációja.

AND, OR, XOR, NOT ... stb. A NOT kivételes eset

NOT

0	1
1	0

adjunk még egy bemenetet

NOT

	0	1
0	1	1
1	0	0

NOT_[0, 1]

↳ és a második paraméter bármi lehet, nem befolyásolja

And

	0	1
0	0	0
1	0	1

1

OR

	0	1
0	0	1
1	1	1

2

XOR

	0	1
0	0	1
1	1	0

3

NOT

	x	x
0	1	1
1	0	0

4

LT : tömb [1..4, 0..1, 0..1] , ⇒ 3D-ós tömb

↓

logikai

konstans

AND_ = 1

OR_ = 2

XOR_ = 3

NOT_ = 4

LT[AND_, 1, 1] ←

Mi a kimenet 1 And 1 esetén?

k := LT[TIP, Be1, Be2]

↓

logikai művelet típusa

Végrehajtási idő csökkentése - a ciklus végrehajtási idejének csökkentése

Ciklusok szétválasztása

Az $\{a_n\}$ sorozat elemeinek értékétől függetlenül elvégezhető részsorozatok kijelölése (pl.: az elem sorozatban elfoglalt helye alapján)

Pl.:

▷ $$b_i := \begin{cases} x - a, & \text{ha } i \leq k \\ x + a, & \text{ha } i > k \end{cases}$$

$(1 < k < n)$

▷ $\{a_{k_i}\}$ $\quad i = k_i \bmod l \quad (1 < l < n)$

elemek szétválogatása l részsorozatra

F: Adott n elemű $\{a_n\}$ sorozat elemeit felhasználva a $\{b_n\}$ sorozat elemeit állítjuk elő.

```
Eljárás sorozat_2 (n: egész, A, B: sorozat);
    ciklus i := 1 .. n
        ha i ≤ k   akkor B[i] := x - A[i]
                   különben B[i] := x + A[i];
    c vége
e vége.
```

```
eljárás sorozat_3 (n: egész, A, B: sorozat);
    ciklus i := 1 .. k
        B[i] := x - A[i]
    c vége;
    ciklus i := k+1 .. n
        B[i] := x + A[i]
    c vége
e vége
```

→ nem kell feltételvizsgálat és a hatékonyabb

az értékadások az akkor ágon k-szor, a különben ágon $n-k$-szor hajtódik végre mindkettőnél, de nem történik n db feltétele-vizsgálat a sorozat_3-ban.

F. Számítsuk ki a következő összeget:

$$s := \frac{a_1 + xa_2 + ya_3 + \overset{xa_4+ya_5\ldots}{\ldots} + xa_{n-1} + a_n}{n} \qquad (n \geq 5 \wedge \text{páratlan})$$

súlyozott átlagot kell számolni

```
Eljárás összeg_1 (n: egész, A: sorozat, s: elemtípus);
  s: = 0;
  ciklus i := 1..n
    ha (i = 1) vagy (i = n) akkor s := s + A[i]
         különben
           ha (i mod 2) = 0 akkor s:= s + 4 * A[i]
                            különben s := s + 2 * A[i];
  c vége;
  s := s/n;
e vége;
```

```
Eljárás összeg_2 (n: egész; A: sorozat, s: elemtípus);
  s : = A[1] + A[n] ;  i := 2;   → első és utolsó elemet
                                   tartalmazza
  ciklus míg i ≤ n - 1;
     s := s + 4 * A[i] ;  i := i + 2;   → kettesével (páros számok)
                                          lépünk
  c vége;
  i := 3;
  ciklus míg i ≤ n - 2        (páratlan számok)
     s : = s + 2 * A[i];   i := i + 2;
  c vége;
  s : = s / n;
e vége;
```

Eljárás összeg_3 (n: egész, A: sorozat, s: elemtípus);

s := A[1] + 4*A[2] + A[n] ; i := 3;

ciklus míg i ≤ n-2

s := s+2*A[i] + 4*A[i+1] ;

i := i+2 ;

c vége;

s := s/n;

e vége;

1 | 2 | 3 4 ......

ptlan p.

páros plan

elempárokat alakíthatunk ki, de az elején az első és az utolsó elemet, valamint a 2.-at (vagy az n-1-et) kivesszük.

Vezérlési szerkezetek:

I., II., III.,

6. előadás

III. 17.

Végrehajtási idő csökkentése - ciklusmag végrehajtási idejének csökkentése:

Feltételek elhagyása

- a állapottér egyes elemeit változtatni kell a feltételtől függően

Zokák vannak az állapottérben ⇒ mindegyik milyen valószínűség szerint elpusztul: 0: rókatelen
0-tól különböző: nem rókatelen

ciklus i := 1 .. n ⇒ 1 dimenziós tömböt bejárjuk

k-szor történik meg
p valószínűséggel következik be, h. az elem 0-vá változik

ha (A[i] ≠ ∅) ∧ p < RND akkor → véletlen érték

A[i] := ∅

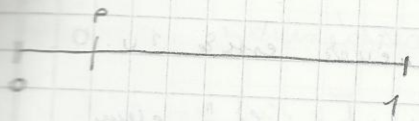
c vége

- Vizsgáljuk! Mennyire befolyásolja a kimenet állapotát, hogy milyen volt a bemenet?
- Ha bemenetkor ∅ volt az i-dik pozíció ⇒ ∅ marad
∅-tól különböző esetekben ∅-vá írja.

Függ-e a változás, hogy ∅ volt-e az i-dik pozíció, vagy nem?
- Generálunk n-nek véletlen számot. ⇒ a p értéke nem függ, hogy milyen elosztásban vannak a nullák.
- Egy tömbelemet ∅-ra állíthatunk, függetlenül attól, hogy mi az értéke.
- a p értékétől függ, hogy a művelet végrehajtódik-e.

Egyenesítés: állandó a ‚p' értéke

I. p-től független	II. p-től függ	
$k \cdot 2n$	$k \cdot n$	a; feltételek száma
$n - n(1-p)k + 1$	$p \cdot n \cdot k$	b; értékadás



ha p kicsi, nagyobb az esélye, hogy a random szám a nagyobb intervallumba esik.

· ha kihagyjuk a p-t, kevesebbszer kell megvizsgálni. (a feltétel elhagyása után)

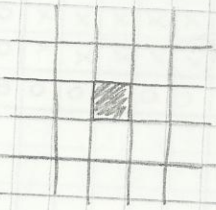
· feleslegesen adunk p-nek $\emptyset$-t értékül

2 dimenziós állapottér:

Szimuláció ⇒ 2D n×m-es állapottér

Neumann játék:

- Amennyiben 1 cellának 2 v. 3 szomszédja van, és ott laknak (vannak sejtek), ⇒ ez a sejt életben marad.



- Elpusztul, ha csak 1 szomszédja van, vagy 3-nál több.

- Amennyiben 1 üres cellának 3 szomszédja van, ott születik egy új sejt.

Rendelkezni kell azzal az információval, hogy hány szomszédja van.

· Tfh: Az üres cellát 0 jelzi, amelyikben lakó van, az 1. (Ez bizonyos mértékig a megvalósítás terhe).

"elegendő kis p-k esetén ... minél nagyobb p-k esetén"