1. gyakorlat

Téma:

Algoritmusok műveletigényének meghatározása, hatékonyság, hatékonyság jellemzése (aszimptotikus korlátok bevezetése). Az anyag úgy lett összeállítva, hogy akkor is elvégezhető, ha még nem volt előtte előadás.

Javasolt feladatok:

1. Polinom helyettesítési értékének kiszámítása. Adott egy n-ed fokú polinom, határozzuk meg egy adott x helyen felvett értékét: $a_n*x^n+a_{n-1}*x^{n-1}+ \dots. a_1*x+a_0$
   (Tfh nagyon sok polinomunk van, és nagyon sok helyen kell kiszámítani az értékét, ezért készítsünk minél hatékonyabb megoldást.)

A polinom együtthatóit egy nullától indexelt, n+1 méretű tömbben helyezzük el. (Megállapodás: ha a tömböt nem nullától indexeljük, a deklarációnál és a specifikációnál jelezzük, pl. A/1:T[n]. Most tehát Z:R[] ugyanaz, mint Z/0:R[].) A Z tömb mérete: Z.length (hangsúlyozzuk, hogy: Z.length=n+1).

A megoldásoknál írjuk fel, hogy az egyes lépések hányszor hajtódnak végre. Vizsgáljuk meg a ciklusiterációk *it(n)*, a szorzások *S(n)* és az összeadások *Ö(n)* számát, a polinom fokszámának függvényében.

Feltehető, hogy n≥0, azaz Z.length>0

Első megoldás, az összegzés tételéből származik:

Polinom1(Z:R[]; x:R) :R	*Hányszor fut le (Z.lenght=n+1)*
y:= Z[0]	*1*
i = 1 to Z.length-1	*n+1 (ciklusfeltétel kiértékelés)*
h:=x	*n*
j= 1 to i-1	*1+2+3+...+n-1+n*
h:=h*x	*0+1+2+3+...+n-1*
y:=y+h*Z[i]	*n*
return y	*1*

$$S(n) = \frac{n*(n+1)}{2} = \frac{n^2+n}{2}$$

$\text{Ö}(n) = n$ *it(n) = S(n)*

Második megoldás, x hatványait rekurzívan számoljuk a h változóban: $x^i=x^{i-1}*x$, ha i>0, $x^0=1$

Rekurzív(Z:R[]; x:R) :R	*Hányszor fut le (Z.lenth=n+1)*
y:=Z[0] h:=1	*1*
i = 1 to Z.length-1	*n+1*
h:=h*x	*n*
y:=y+h*Z[i]	*n*
return y	*1*

$S(n) = 2*n$ *it(n) = n*

$\text{Ö}(n) = n$

Harmadik megoldás, a Horner séma:

$y=(\ldots(a_n*x+a_{n-1})*x+a_{n-2})*x+\ldots+a_1)*x+a_0$

Horner(Z:R[]; x:R) :R	*Hányszor fut le (Z.length=n+1)*
y:=Z[Z.length-1]	*1*
i= Z.length-2 downto 0	*n+1*
y:=y*x+Z[i]	*n*
return y	*1*

$S(n) = n$ *it(n) = n*

$\ddot{O}(n) = n$

Jellemezzük a három megoldást a Θ aszimptotikus korlát segítségével. Írjuk fel a definíciót, rajzoljunk szemléltető ábrát, majd készítsük el az alábbi táblázatot! *it(n)* a futási idő nagyságrendjét általában, minden nemrekurzív program esetében is megadja:

	Polinom1	Rekurzív	Horner
S(n)	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Ö(n)	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
it(n)	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$

2. Buborék rendezés (ezt tanulták programozásból, de nem biztos, hogy mindegyik csoportnál vették). Készítsük el az alap algoritmust, majd a javított változatot. Elemezzük itt is, hogy a struktogram egyes lépései hányszor hajtódnak végre. Nézzük meg az összehasonlítások Öh(n) és cserék számát Cs(n). Cserék elemzésénél használjuk a mCs(n), MCs(n) ACs(n) jelöléseket. Átlagos csere számot nem kell pontosan kiszámolni, elég csak a „megérzés”-re támaszkodni.

Mutassuk be a rendezés menetét egy rövid példán, majd írjuk fel a struktogramot.

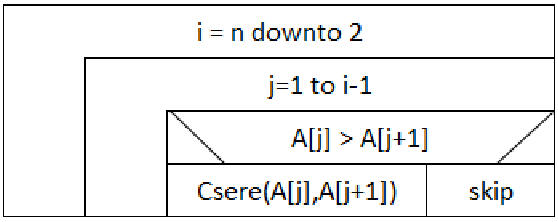
Buborék példa:					Csere
3	5	2	4	1	0
3	5	2	4	1	1
3	2	5	4	1	1
3	2	4	5	1	1
3	2	4	1	5	*1. menet vége, 5 a helyén van*
3	2	4	1	5	1
2	3	4	1	5	0
2	3	4	1	5	1
2	3	1	4	5	*2. menet vége*
2	3	1	4	5	0
2	3	1	4	5	1
2	1	3	4	5	*3. menet vége*
2	1	3	4	5	1
1	2	3	4	5	*4. menet vége, rendezett a tömb*

Csere összesen: **7**

Összehasonlítás összesen: **10**

A rendezendő kulcsokat (és a hozzájuk tartozó adatokat) egy A nevű tömbben helyeztük el. A.length = n, a rendezendő kulcsok darabszáma.

Buborék(A/1:T[n])



Hányszor fut le (A.length=n)

n

n+n-1+ … +2

n-1+n-2+ … +1

Csereszám?

Az összehasonlítások száma $Ö(n) = \sum_{i=1}^{n-1} i = \frac{n*(n-1)}{2} = \frac{n^2-n}{2} \in \Theta(n^2)$

Cserék számát hogyan tudjuk meghatározni?

Cserék száma a rendezendő adatsorban található inverziók számával egyenlő. Lásd a példában 7 inverzió van:
3,2 3,1 5,2 5,4 5,1 2,1 4,1

Ebből adódik, hogy mCs(n)=0 (nincs inverzió, azaz növekvően rendezett a bemenet)

MCs(n)= Ö(n) (minden összehasonlítást csere követ, azaz fordítottan rendezett a tömb)

ACs(n)= $\frac{n*(n-1)}{4} = \Theta(n^2)$ Ezt nem kell pontosan levezetni, a lejjebb megadott linken megtalálható.

Vezessük be az Ω és O aszimptotikus korlátokat, és használjuk a csere számra:
mCs(n)=0, MCs(n)=$\Theta(n^2)$ azaz Cs(n)=$O(n^2)$

Az átlagos futási idő kiszámítása részletesen megtalálható dr Fekete István jegyzetében:
https://people.inf.elte.hu/fekete/algoritmusok_jegyzet/01_fejezet_Muveletigeny.pdf

Említsük meg a buborék rendezés javítási módszereit:

- figyelhetjük egy logikai változóval, hogy volt-e csere, ha nem volt akkor a külső ciklus álljon le,
- megjegyezhetjük az utolsó csere helyét: ha ez u és u+1 indexen történt, akkor u+1-től már a tömb rendezett, a külső ciklus változót u-ra lehet csökkenteni (ezt a változatot én fel szoktam írni). Itt elég csak a legkedvezőbb és legrosszabb esetet vizsgálni mÖ(n)$\in\Theta(n)$, MÖ(n)$\in\Theta(n^2)$. Megemlíthetjük a futási idő jelölést: mT(n)$\in\Theta(n)$, MT(n)$\in\Theta(n^2)$; azaz mT(n),MT(n)$\in\Omega(n)$, mT(n),MT(n)$\in O(n)$

Példa:

Javított buborék példa:

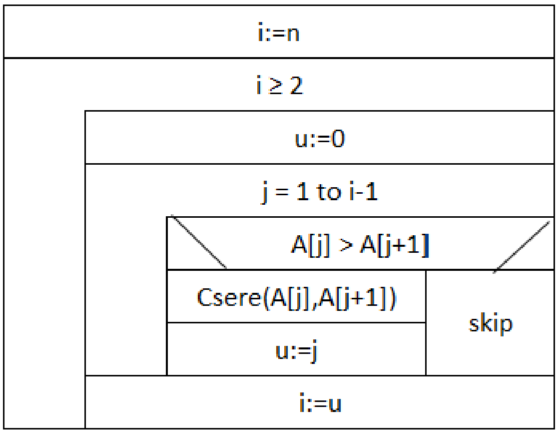
					Csere	
2	3	1	4	5	0	
2	3 ⟺	1	4	5	1	u=2
2	1	3	4	5	0	
2	1	3	4	5	0	
2	1	3	4	5	*1. menet vége 3,4,5 rendezett*	
2 ⟺	1	3	4	5	1	u=1
1	2	3	4	5	*kész*	

Csere összesen: 2

Összehasonlítás összesen: 5

Struktogramja, elemzés nélkül:

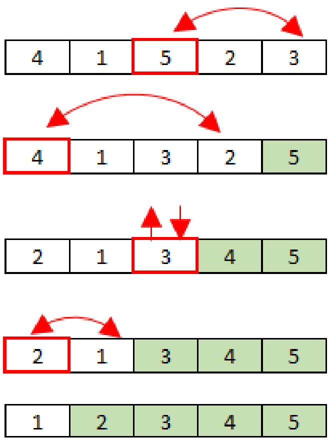
JavítottBuborék(A/1:T[n])



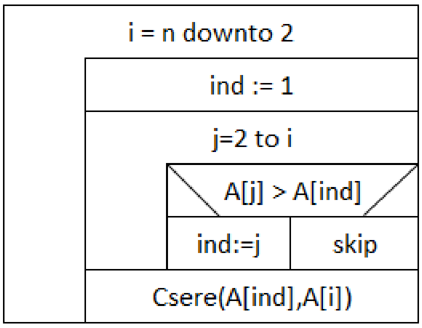
3. A maximum kiválasztásos rendezés struktogramjának elkészítése, elemzése.

Mutassuk be a rendezést egy rövid példán, majd írjuk fel a struktogramot. Beszéljük meg, miért nem érdemes a cserét egy elágazásba tenni? (Maximum értéket azért nem használunk, mert rendezésnél mindig feltesszük, hogy nem csak kulcs, hanem a kulcshoz tartozó rekord is tárolva van a tömbben, melynek mozgatása költséges lenne.)

Példa:



MaxKivRend(A/1:T[n])



Hányszor fut le (A.length=n)

n

n-1

n+n-1+ … +2

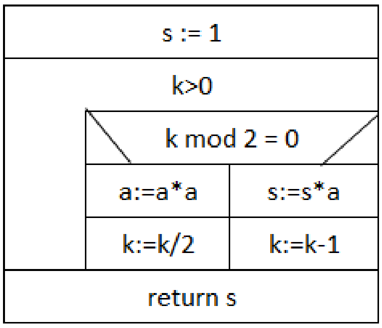
n-1+n-2+ … +1

n-1

Házi feladatok:

1. Legendre algoritmus műveletigényének meghatározása k függvényében:
   (Az algoritmus az a^k hatványt számolja ki, ezt nem szokták tudni, így elsőként fel lehet adni kitalálós feladatnak, hogy mit csinál az algoritmus. $0^0=1$, definíció szerint.)

Legendre(a:R,k:N):R



a' és k' : a és k kezdeti értéke

Ciklus invariáns:

s*(a^k) = a'^k' és k in 0..k'

(a^k : hatványozás)

Útmutatás:

Érdemes lejátszani egy példán: 3^{11}

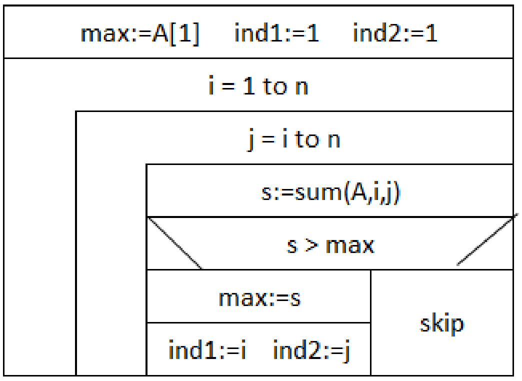
Menet	a	s	k
Kezdés	3	1	11
1. k páratlan	3	3	10
2. k páros	3^2	3	5
3. k páratlan	3^2	3^3	4
4. k páros	3^4	3^3	2
5. k páros	3^8	3^3	1
6. k páratlan	3^8	3^{11}	0
k=0, vége az algoritmusnak, s –ben a megfelelő hatvány van			

- Legkedvezőbb eset, amikor k 2 hatványa, ekkor bináris alakja: 100…00, azaz a legutolsó menet kivételével mindig a páros ágon fut az algoritmus, k mindig feleződik. A legutolsó menetben k=1 esetében egyszer fut le a páratlan ág. Ilyenkor a ciklus meneteinek száma: $T(k) = \log_2 k + 1$
- Legkedvezőtlenebb az az eset amikor felváltva fut a páratlan, majd páros ágon. Ilyenkor 2 hatvány-1 alakú a k, aminek bináris alakja csupa 1-es: 111…11. amikor a „k:=k-1” ágon fut páros lesz a szám, majd kettővel osztva ismét páratlan. Ilyenkor a ciklus meneteinek száma: $T(k)= 2*\lfloor \log_2 k \rfloor + 1$ (vagy $T(k)= 2*\log_2(k+1)-1$ is jó megoldás)
- Mivel ugyanazt a nagyságrendet kaptuk, azaz $mT(k),MT(k)\in\Theta(\log_2 k) = \Theta(\log k)$, így az átlagos eset is e kettő közé kell essen, tehát $AT(k) \in \Theta(\log k)$
- Említsük meg, esetleg később vissza lehet rá térni, hogy $\log_a n\in\Theta(\log_b n)$ és fordítva, azaz ha a,b>1 akkor a logaritmusok ugyanazt a nagyságrendet képviselik, így a logaritmus alapszáma elhanyagolható.
- Megemlíthető, hogy a k:=k/2 nem jelent osztást, ez csak a bitek shiftelése jobbra, így ez a Horner sémához hasonlóan a szorzások számát tekintve egy nagyon hatékony algoritmus.

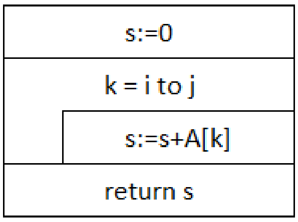
2. Adott egy *n* hosszú, egész számokat tartalmazó tömb. Keressük a tömb azon szakaszát, melynek összege a lehető legnagyobb. (Legyen a tömb neve: A, adjuk meg az a két indexet: 1≤ind1≤ind2≤n, melyre a $\sum_{i=ind1}^{ind2} A[i]$ a maximális.). Elemezzük a megoldás műveletigényét, készítsünk minél hatékonyabb algoritmust! A „Brute-Force” megoldás $\Theta(n^3)$, könnyen javítható $\Theta(n^2)$-re, de van $\Theta(n)$-es megoldás is!

Megoldások:

BruteForce(A/1:Z[n])

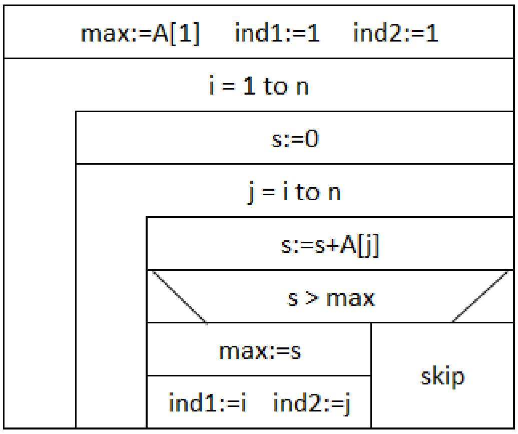


sum(A/1:Z[], i:N, j:N): Z



A két főciklus négyzetes műveletigényű (mintha egy négyzetes mátrix felső háromszögét járnánk be):$\Theta(n^2)$, a kiemelt összegző függvény pedig i-től j-ig előállítja a vektor elemeinek összegét, ami O(n), összességében belátható, hogy $\Theta(n^3)$ a műveletigény.

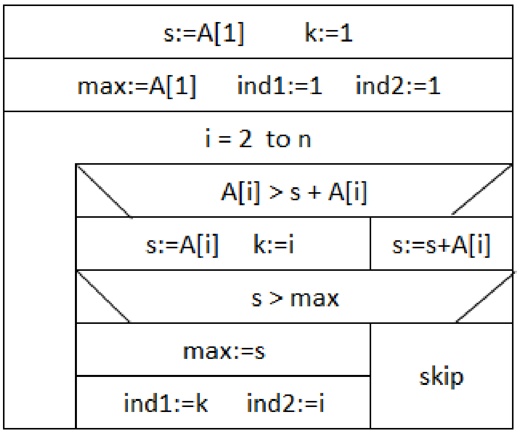
Négyzetes(A/1:Z[n])



Ugyanúgy a „felső háromszöget” járjuk be, de az összeg előállítását az előző összegből egy összeadással állítjuk elő (mint a hatványok számítása a polinomos feladatban), így a műveletigény: $\Theta(n^2)$

A legügyesebb megoldás lineáris:

Lineáris(A/1:Z[n])



Illusztráció:

A	1	2	3	4	5	6	7	8	9	10
	4	5	-3	-7	4	-1	2	3	-1	10
s	4	9	6	-1	4	3	5	8	7	17
k	1				5					
max	4	9								17
ind1	1	1								5
ind2	1	2								10

Egy adott k kezdőindextől (kezdetben 1-től) elkezdjük összeadni a vektorban lévő számokat. Mindig a következő (i-dik) elemmel növeljük az összeget. Ha az így kapott összeg nagyobb, mint az A[i], akkor megyünk tovább az összeg számolással. Amikor az összeg negatívvá válik (lásd a példában a 4. elem, akkor a következő körben s+A[i]<A[i] teljesül, így nem érdemes a részösszeget folytatni, egy új részösszeget kezdünk s-ben, és megjegyezzük az új kezdőindexet k-ban. Ha az adott körben s értéke nagyobb lesz, mint az eddigi részösszegek maximuma, akkor max-ot, és ind1, ind2 változókat is megfelelően átállítjuk.

Tulajdonképpen ez egy rekurzív függvényen végzett maximum keresés[1], ahol a rekurzív függvény értéke két komponensből áll, egy összegből (s), és egy kezdőindexből (k). A rekurzív függvény egyenlete pedig:

$$szum(1) = (A[1], 1)$$
$$i > 1\ esetben: szum(i) = \begin{cases} (A[i], i), ha\ A[i] > szum(i-1)_1 + A[i] \\ (szum(i-1)_1 + A[i], szum(i-1)_2),\ \ egyébként \end{cases}$$

A ciklusmag elsőként meghatározza a rekurzív függvény i-dik értékét, majd a maximum kiválasztás tétel ezeken keresi a maximumot.

Készült az „Integrált kutatói utánpótlás-képzési program az informatika és számítás-tudomány diszciplináris területein" című EFOP 3.6.3-VEKOP-16-2017-00002 azonosítójú projekt támogatásával.

[1] A megoldás dr. Gregorics Tibor Programozás kurzusán szerepelt: a rekurzív függvény kiszámolása iterációval progtétel kapcsán.