A félév során előkerülő témakörök

- rekurzív algoritmusok
- rendező algoritmusok
- alapvető adattípusok, adatszerkezetek, és kapcsolódó algoritmusok
- dinamikus programozás
- mohó algoritmusok
- gráf algoritmusok
- keresőfák

Algoritmus

Az algoritmus olyan elemi műveletekből kompozíciós szabályok szerint felépített összetett művelet, amelyet megadott feltételt teljesítő bemeneti adatra végrehajtva, a megkívánt kimeneti adatot eredményezi.

Számítási Probléma

A számítási probléma bemenet/kimenet feltételpárral meghatározott követelmény. Azt írja elő, hogy a bemeneti feltételt teljesítő adatra a kimeneti feltételt teljesítő adatot kell kiszámítani. Probléma bemenete: egy olyan (esetleg összetett) adat, amely teljesíti a bemeneti feltételt. Például, a rendezési probléma a következőt jelenti. Bemenet: Azonos típusú adatok $H = \{a_1, ..., a_n\}$ halmaza, amelyeken értelmezett egy $\leq$ lineáris rendezési reláció. Kimenet: A H halmaz elemeinek egy rendezéstartó felsorolása, tehát olyan $\{b_1, ..., b_n\}$ sorozat, amelyre $b_1 \leq b_2 \leq \cdots \leq b_n$, és $H = \{b_1, ..., b_n\}$.

Specifikáció

$$\{B\}A\{K\}$$

- B logikai formula, a bemeneti feltétel,
- K logikai formula, a kimeneti feltétel,
- A az algoritmus, amelyre az állítás vonatkozik.

Az A algoritmus helyes a $\{B\}A\{K\}$ specifikációra nézve, ha minden X bemeneti adatra, amelyre teljesül a B bemeneti feltétel, az algoritmus végrehajtása véges sok elemi művelet végrehajtása után befejeződik, és a keletkezett kimeneti adatra teljesül a K kimeneti feltétel.

Beszúró Rendezés

```
BESZÚRÓ-RENDEZÉS (A)
 For j= 2 to hossz(A)
        {kulcs:=A[j]
         i:=j-1
         while i>0 és A[i]>kulcs
                 {A[i+1]:=A[i]
                 i:=i-1}
         A[i+1]:=kulcs}
```

Példa
$(5,2,4,6,1,3)$
$(5,2^j,4,6,1,3)$
$(2,5,4,6,1,3)$
$(2,5,4^j,6,1,3)$
$(2,4,5,6,1,3)$
$(2,4,5,6^j,1,3)$
$(2,4,5,6,1^j,3)$
$(1,2,4,5,6,3)$
$(1,2,4,5,6,3^j)$
$(1,2,3,4,5,6)$
További példa:
http://www.youtube.com/watch?v=fdGvEZzYYRcfeature=related

Futási idő elemzése

Legyen A az $e_1,\ldots,e_m$ elemi műveletekből felépített algoritmus, és jelölje t_i az e_i művelet futási idejét (konkrét, vagy hipotetikus gépen). A t_i futási idő függhet az e_i művelet argumentumaitól. A továbbiakban feltételezzük, hogy minden e_i elemi művelet futási ideje c_i konstans. Adott x bemenetre jelölje $T(A,x)$ az A algoritmus tényleges futási idejét, ami az x bemeneti adatra ténylegesen végrehajtott elemi műveletek futási idejének az összege. Jelölje $|x|$ az x bemeneti adat méretét. Ez a méret összetett adat (pl. halmaz, vagy sorozat) esetén általában az adatok száma. Nem összetett adat esetén pedig általában az adat értéke. Az e_i elemi műveletek t_i futási ideje és az $|x|$ méret függvény együttesen adja a bonyolultsági mértéket. Algoritmusnak más költsége is van, nevezetesen a memóriaigény, és elosztott algoritmusok esetén fontos a kommunikációs költség is.

Legjobb eset:
$T_{lj}(A,n) = \min\{T(A,x) : |x| = n\}$
Legrosszabb eset:
$T_{lr}(A,n) = \max\{T(A,x) : |x| = n\}$
Átlagos eset:
Jelölje $Pr(x)$ annak valószínűségét, hogy x bemeneti adata lesz az A algoritmusnak.
$T_a(A,n) = \sum_{|x|=n} Pr(x)T(A,x)$

Elemzés

Legyen t_j a for ciklus j-edik végrehajtásában a while ciklus tesztelések száma.

```
BESZÚRÓ-RENDEZÉS (A)                költség végrehajtások száma
 For j= 2 to hossz(A)                 c1     n
        {kulcs:=A[j]                  c2     n-1
         i:=j-1                       c3     n-1
         while i>0 és A[i]>kulcs      c4     t2+t3+...+tn
                 {A[i+1]:=A[i]        c5     (t2-1)+...+(tn-1)
                 i:=i-1}              c6     (t2-1)+...+(tn-1)
         A[i+1]:=kulcs}               c7     n-1
```

A teljes költség:

$$c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

Legjobb eset:
Ha a tömb már rendezett, akkor minden j-re $t_j = 1$, tehát

$$T_{lj}(A,n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$

Ez n lineáris függvénye.

Legrosszabb eset:
Ha a tömb már fordított sorrendben rendezett, akkor minden j-re $t_j = j$, tehát

$$T_{lr}(A,n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n(n+1)/2 - 1) +$$

$$c_5((n-1)n/2 - 1) + c_6((n-1)n/2 - 1) + c_7(n-1),$$

mivel $\sum_{j=2}^{n} j = n(n+1)/2 - 1$.
A legrosszabb eset n négyzetes függvénye.

Aszimptotikus jelölések

O-jelölés

$$O(g(n)) = \{f(n) : (\exists c, n_0 \geq 0)(\forall n \geq n_0)(0 \leq f(n) \leq cg(n))\}$$

Az $f(n) = O(g(n))$ jelentése: $f(n) \in O(g(n))$.

Ω-jelölés

$$\Omega(g(n)) = \{f(n) : (\exists c, n_0 > 0)(\forall n \geq n_0)(cg(n) \leq f(n))\}$$

Az $f(n) = \Omega(g(n))$ jelentése: $f(n) \in \Omega(g(n))$.

Θ-jelölés

$$\Theta(g(n)) = \{f(n) : (\exists d_1, d_2, n_0 > 0)(\forall n \geq n_0)(d_1 g(n) \leq f(n) \leq d_2 g(n))\}$$

Az $f(n) = \Theta(g(n))$ jelentése: $f(n) \in \Theta(g(n))$.

Aszimptotikus jelölések

o-jelölés

$$o(g(n)) = \{f(n) : (\forall c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)(0 \leq f(n) \leq cg(n))\}$$

Az $f(n) = o(g(n))$ jelentése: $f(n) \in o(g(n))$, ekkor $f(n)/g(n) \to 0$.

ω-jelölés

$$\omega(g(n)) = \{f(n) : (\forall c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)(cg(n) \leq f(n))\}$$

Az $f(n) = \omega(g(n))$ jelentése: $f(n) \in \omega(g(n))$, akkor $f(n)/g(n) \to \infty$.

Példák

- $2n + 20 = \Theta(n)$

- $n^2 = \omega(n \log n)$
- $\log_2(n) = \Theta(\log_{10} n)$
- $n^2 + 20n + 10 = \Theta(n^2)$
- $2^n = \omega(n^{100000})$

A konstans költségű elemi műveletek mindegyikét számolhatjuk egységnyi költségűnek.

Fő tulajdonságok

- A Θ reláció tranzitív, reflexív és szimmetrikus, így egy osztályozás tartozik hozzá.
- A Θ, O, Ω, o, ω relációk mindegyike tranzitív.
- Ha $f(n) = O(g(n))$, akkor $g(n) = \Omega(f(n))$.
- Ha $f(n) = o(g(n)$, akkor $f(n) = O(g(n))$.
- Ha $f(n) = \omega(g(n))$, akkor $f(n) = \Omega(g(n))$.

Rekurzív algoritmusok

Egy objektum definícióját rekurzívnak nevezünk, ha a definíció tartalmazza a definiálandó objektumot.

Egy P eljárást (vagy függvényt) rekurzívnak nevezünk, ha P utasításrészében előfordul magának a P eljárásnak a hívása.

```
Faktor(n)
  If n=1 return 1
  Else return n*Faktor(n-1)
```

Példa:
Faktor(5)=5*Faktor(4)
Faktor(4)=4*Faktor(3)
Faktor(3)=3*Faktor(2)
Faktor(2)=2*Faktor(1)
Faktor(1)=1

Hatványozás

```
Hatvany(a,n)
  If n=0 return 1
  else return a*Hatvany(a,n-1)
```

```
Hatvany2(a,n)
  If n=0 return 1
  else If odd(n) return a*Hatvany2(a,(n-1)/2)*Hatvany2(a,(n-1)/2)
       else return Hatvany2(a,n/2)*Hatvany2(a,n/2)
```

Euklideszi algoritmus

- Ha b=0, akkor $lnko(a,b) = a$.
- Egyébként ha $a \geq b$, akkor $lnko(a,b) = lnko(b, a \mod b)$.

Tehát, ha $a \geq b$, akkor az alábbi algoritmus kiszámolja a legnagyobb közös osztót.

```
Euklidesz(a,b)
  If b=0 return a
  Else return Euklidesz(b, a mod b)
```

$$Euklidesz(64,28) = Euklidesz(28,8) = Euklidesz(8,4) = Euklidesz(4,0) = 4$$

Fibonacci számok

Fibonacci 1202-ben vetette fel a kérdést: hány nyúlpár születik n év múlva, ha feltételezzük, hogy

- az első hónapban csak egyetlen újszülött nyúl-pár van;
- minden nyúlpár, amikor szaporodik egy újabb nyúlpárt hoz létre;
- a nyulak a születésük után a következő két évben egyszer-egyszer szaporodnak

```
Fibonacci(n)
  If n=1 or n=2 return 1
  Else return Fibonacci(n-1)+Fibonacci(n-2)
```

A hanoi torony probléma

Három pálca egyikén n korong van a többi üres. A korongok nagyság szerinti sorrendben helyezkednek el, alul van a legnagyobb. Át akarjuk helyezni a korongokat egy másik pálcára a következő szabályok alapján. Egyszerre csak egy korong mozgatható. A korong vagy üres pálcára vagy egy nála nagyobb korongra helyezhető. Oldjuk meg a feladatot egy rekurzív algoritmussal!

Legyen a hanoi eljárás egy olyan algoritmus, amelynek három argumentuma van, az első argumentum azt adja meg, hogy hány korongot helyezünk át, a második megadja, hogy melyik toronyról a harmadik, hogy melyik toronyra. Ekkor az eljárás az $(n,1,2)$ argumentummal megoldja a feladatot.

Amennyiben $i-1$ korongot már át tudunk helyezni, i korongot a következőképpen helyezhetünk át. Elsőként $i-1$ korongot áthelyezünk az oszlopról egy másik oszlopra. Utána az i-edik korongot rárakjuk a kimaradó üres oszlopra. Végül ezen korong tetejére felrakjuk az $i-1$ korongot. Ezt a rekurziót írja le a következő eljárás (a megengedett lépést a mozgat függvény írja le).

```
Hanoi(n, rol, ra)
   If n=1 Then
         mozgat(rol, ra)
   Else  {hanoi(n-1, rol, 6-rol-ra)
         mozgat(rol, ra)
         Hanoi(n-1, 6-rol-ra, ra)}
```

Lépések száma: Az i-edik korong átrakásához kétszer kell $i-1$ korongot áthelyezni és egy további mozgatás szükséges. Tehát $T(i)$-vel jelölve az i korong átrakásához szükséges mozgatások számát a $T(i) = 2T(i-1)+1$ rekurzív összefüggés áll fenn. $T(1) = 1$, így $T(2) = 3$, $T(3) = 7$. Azt sejthetjük, hogy $T(i) = 2^i - 1$ egyenlőség, amely teljes indukcióval egyszerűen igazolható.