Curs 4:

Analiza eficienței algoritmilor (I)

Structura

- In ce constă analiza eficienței algoritmilor ?
- Cum se poate măsura eficiența algoritmilor ?
- Exemple
- Analiza în cazul cel mai favorabil, în cazul cel mai defavorabil și in cazul mediu

In ce constă analiza eficienței algoritmilor?

Analiza eficienței algoritmilor înseamnă:

estimarea volumului de resurse de calcul necesare execuției algoritmilor

Observatie: uneori se folosește termenul de analiză a complexității

Utilitate: analiza eficienței este utilă pentru a compara algoritmii între ei și pentru a obține informații privind resursele de calcul necesare pentru execuția algoritmilor

In ce constă analiza eficienței algoritmilor?

Resurse de calcul:

- Spațiu de memorie = spațiul necesar stocării datelor prelucrate de către algoritm
- Timp de execuție = timp necesar execuției prelucrărilor din cadrul algoritmului

Algoritm eficient: algoritm care necesită un volum rezonabil de resurse de calcul

Dacă un algoritm utilizează mai puține resurse de calcul decât un alt algoritm atunci este considerat mai eficient

In ce constă analiza eficientei algoritmilor?

Există două tipuri de eficiență

- Eficiența în raport cu spațiul de memorie = se referă la spațiul de memorie necesar algoritmului

- Eficiența în raport cu timpul de execuție = se referă la timpul necesar execuției prelucrărilor din algoritm

Ambele tipuri de analiză a eficienței algoritmilor se bazează pe următoarea ipoteză:

volumul resurselor de calcul necesare depinde de volumul datelor de intrare = dimensiunea problemei

Scopul analizei eficienței este să răspundă la întrebarea:

Cum depinde volumul resurselor necesare de dimensiunea problemei ?

Cum se poate determina dimensiunea problemei ?

... de regulă foarte simplu pornind de la enunțul problemei și de la proprietățile datelor de intrare

Dimensiunea problemei = volumul de memorie necesar pentru a stoca toate datele de intrare ale problemei

Dimensiunea problemei este exprimată în una dintre următoarele variante:

- numărul de componente (valori reale, valori întregi, caractere etc) ale datelor de intrare
- numărul de biți necesari stocării datelor de intrare

Alegerea se face în funcție de nivelul la care se fac prelucrările ...

Cum se poate determina dimensiunea problemei ?

Exemple:

1. Determinarea minimului unui tablou x[1..n]

   Dimensiunea problemei: n

2. Calculul valorii unui polinom de ordin n

   Dimensiunea problemei: n

   Obs: numarul coeficienților este de fapt n+1

3. Calculul sumei a două matrici cu m linii și n coloane

   Dimensiunea problemei: (m,n) sau mn

4. Verificarea primalitatii unui numar n

   Dimensiunea problemei: n sau $\log_2 n$

   Obs: numărul de biți necesari stocării valorii n este de fapt $[\log_2 n]+1$ ([..] reprezintă partea întreagă inferioară)

Structura

- In ce constă analiza eficienței algoritmilor ?
- Cum se poate măsura eficiența algoritmilor ?
- Exemple
- Analiza în cazul cel mai favorabil, în cazul cel mai defavorabil și în cazul mediu

Cum se poate măsura eficiența algoritmilor ?

In continuare ne vom referi doar la analiza eficienței din punctul de vedere al timpului de execuție.

Pentru estimarea timpului de execuție este necesar să se stabilească :

- Un model de calcul
- O unitate de măsură a timpului de execuție

Cum se poate măsura eficiența algoritmilor ?

Model de calcul: mașina cu acces aleator (Random Access Machine = RAM)

Caracteristici (ipoteze simplificatoare):

- Toate prelucrările sunt executate secvențial
  (nu există paralelism în execuția algoritmului)
- Timpul de execuție al unei prelucrari elementare nu depinde de valorile operanzilor
  (timpul de execuție pentru a calcula 1+2 nu diferă de timpul de execuție pentru 12433+4567)
- Timpul necesar accesării datelor nu depinde de locația datelor în memorie (nu este diferență între timpul necesar prelucrării primului element al unui tablou și cel al prelucrării ultimului element)

Cum se poate măsura eficiența algoritmilor ?

Unitate de măsură = timpul necesar execuției unei prelucrări elementare (prelucrare de bază)

Operații elementare (de bază):

- Asignare
- Operații aritmetice
- Comparații
- Operații logice

Timp de execuție al algoritmului = numărul de operații elementare executate

Obs. Estimarea timpului de execuție exprimă dependența dintre numărul de operații elementare executate și dimensiunea problemei

Structura

- In ce constă analiza eficienței algoritmilor ?
- Cum se poate măsura eficiența algoritmilor ?
- Exemple
- Analiza în cazul cel mai favorabil, în cazul cel mai defavorabil și in cazul mediu

Exemplu 1 (calcul suma)

Precondiții: n>=1

Dim. problema: n

Postconditii: S=1+2+…+n

Algoritm:

```
Sum(n)
1:  S←0
2:  i ← 0
3:  WHILE i<n DO
4:          i ← i+1
5:          S ← S+i
6:  ENDWHILE
7: RETURN S
```

Tabel de costuri:

Operație	Cost	Nr. repetări
1	c1	1
2	c2	1
3	c3	n+1
4	c4	n
5	c5	n

Timp de execuție:

$T(n)=(c3+c4+c5)n+(c1+c2+c3)= a*n +b$

Obs: unele dintre costuri sunt identice (c1=c2), iar altele pot fi considerate diferite (c5>c1)

Exemplu 1 (calcul suma)

Observații:

- Varianta cea mai simplă este să se considere toate prelucrările simple ca având același cost
- Dacă în exemplul anterior se consideră că toate operațiile elementare au cost unitar atunci se obtine următoarea estimare pentru timpul de execuție: T(n)=3(n+1)

- Constantele ce intervin în expresia timpului de execuție nu sunt foarte importante. Elementul important este faptul că timpul de execuție depinde liniar de dimensiunea problemei.
- Algoritmul anterior este echivalent cu:

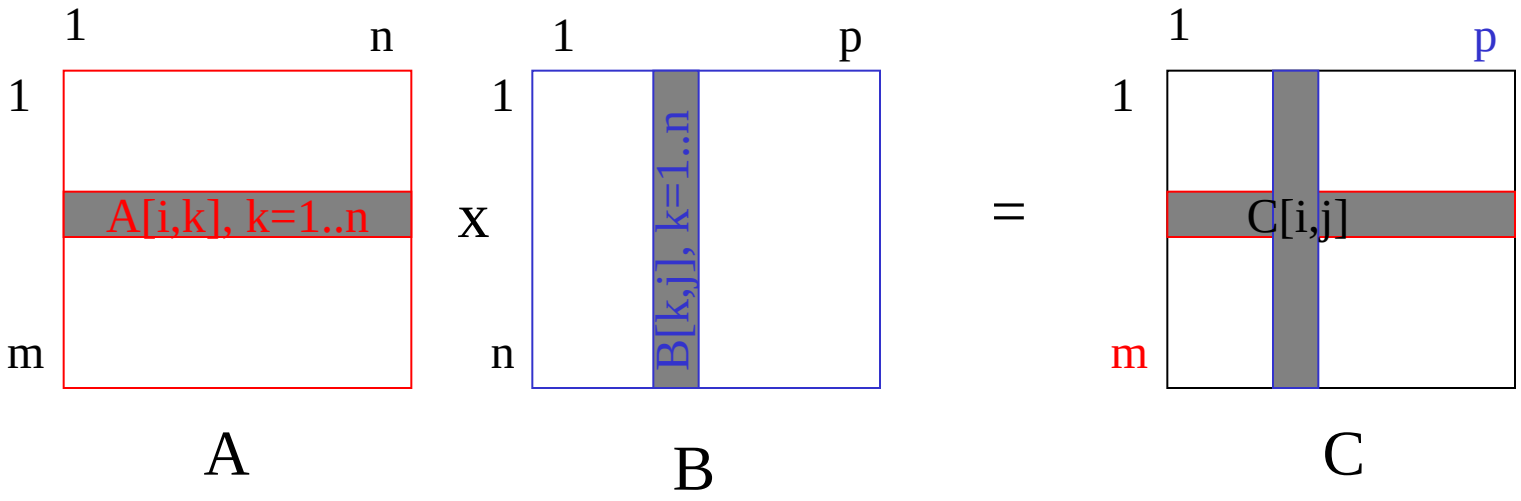
```
S ← 0
FOR i ← 1,n DO S ← S+i  ENDFOR
```

Se observă că gestiunea contorului ciclului FOR implică execuția a 2(n+1) operații; celelalte (n+1) operații corespund calculului sumei (inițializarea lui S și actualizarea de la fiecare repetare a corpului ciclului)

Exemplu 2 (produs matrici)

Preconditii: A_{m*n}, B_{n*p} Postconditii: C=A*B

Dimensiunea problemei: (m,n,p)



$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

C[i,j]=A[i,1]*B[1,j]+A[i,2]*B[2,j]+…+A[i,n]*B[n,j],
i=1..m, j=1..p

Exemplu 2 (produs matrici)

Idee de bază: pentru fiecare i=1..m și j=1..p se calculează suma după k

Algoritm:

```
Produs(A[1..m,1..n],B[1..n,1..p])
1:  FOR i ← 1,m DO
2:   FOR j ← 1,p DO
3:       C[i,j] ← 0
4:       FOR k ← 1,n DO
5:           C[i,j] ← C[i,j]+A[i,k]*B[k,j]
6:       ENDFOR
7:    ENDFOR
8:  ENDFOR
9: RETURN C[1..m,1..p]
```

Tabel de costuri

Op.	Cost	Rep.	Total
1	2(m+1)	1	2(m+1)
2	2(p+1)	m	2m(p+1)
3	1	mp	mp
4	2(n+1)	mp	2mp(n+1)
5	2	mpn	2mnp

T(m,n,p)=4mnp+5mp+4m+2

Exemplu 2 (produs matrici)

Obs: De regulă nu este necesar să se completeze întreg tabelul de costuri ci este suficient să se contorizeze doar operația dominantă

Operație dominantă: cea mai frecventă (costisitoare) operație

Algoritm:

```
Produs(A[1..m,1..n],B[1..n,1..p])
1:  FOR i ← 1,m DO
2:   FOR j ← 1,p DO
3:       C[i,j] ← 0
4:       FOR k ← 1,n DO
5:           C[i,j] ← C[i,j]+A[i,k]*B[k,j]
6:       ENDFOR
7:    ENDFOR
8:  ENDFOR
RETURN C[1..m,1..p]
```

Estimare timp de execuție:

$T(m,n,p)=mnp$

Exemplu 3 (determinare minim)

Precondiții: x[1..n], n>=1 Postcondiții: m=min(x[1..n])

Dimensiunea problemei: n

Algoritm:

```
Minim(x[1..n])
1:  m ← x[1]
2:   FOR i ← 2,n DO
3:       IF x[i]<m THEN
4:                    m ← x[i]
5:       ENDIF
6:   ENDFOR
7:RETURN m
```

Tabel de costuri:

Op.	Cost	Rep.	Total
1	1	1	1
2	2n	1	2n
3	1	n-1	n-1
4	1	t(n)	t(n)

T(n)=3n+t(n)

Obs: Timpul de execuție depinde nu doar de dimensiunea pb. ci și de proprietățile datelor de intrare

Exemplu 3 (determinare minim)

Dacă timpul de execuție depinde de proprietățile datelor de intrare atunci trebuie analizate cel puțin cazurile extreme:

- Cel mai favorabil caz : x[1]<=x[i], i=1..n => t(n)=0 => T(n)=3n
- Cel mai defavorabil caz

x[1]>x[2]>…>x[n])=>t(n)=n-1=> T(n)=4n-1

Rezultă că: 3n<=T(n)<=4n-1

Atât limita inferioară cât și cea superioară depind liniar de dimesiunea problemei

Varianta de analiză ce ia în calcul doar operația dominantă,adică comparația, conduce la

T(n) =n-1

Algoritm:

```
Minim(x[1..n])
1:  m ← x[1]
2:   FOR i ← 2,n DO
3:       IF x[i]<m THEN
4:                    m ← x[i]
5:       ENDIF
6:   ENDFOR
7: RETURN m
```

Exemplu 4 (căutare secvențială)

Preconditii: x[1..n], n>=1, v o valoare

Postconditii: variabila logică "gasit" conține valoarea de adevăr a afirmației "*valoarea v este în tabloul x[1..n]*"

Dimensiunea problemei: n

Algoritm (căutare secvențială):

```
caut(x[1..n],v)
1:  gasit ← False
2:  i ← 1
3:  WHILE (gasit==False) AND (i<=n) DO
4:      IF x[i]==v                          //t1(n)
5:          THEN gasit ← True   //t2(n)
6:          ELSE  i ← i+1       //t3(n)
7:      ENDIF
8:  ENDWHILE
9: RETURN gasit
```

Tabel de costuri

Op.	Cost
1	1
2	1
3	t1(n)+1
4	t1(n)
5	t2(n)
6	t3(n)

Exemplu 4 (căutare secvențială)

Timpul de execuție depinde de proprietățile tabloului x[1..n].

Caz 1: valoarea v aparține tabloului (considerăm k cel mai mic indice cu proprietatea că x[k]= v)

Caz 2: valoarea v nu se află în tablou

$$t1(n)=\begin{cases} k & \text{dacă v este în x[1..n]} \\ n & \text{dacă v nu este în x[1..n]} \end{cases}$$

$$t2(n)=\begin{cases} 1 & \text{dacă v este în x[1..n]} \\ 0 & \text{dacă v nu este în x[1..n]} \end{cases}$$

$$t3(n)=\begin{cases} k-1 & \text{dacă v este în x[1..n]} \\ n & \text{dacă v nu este în x[1..n]} \end{cases}$$

Algoritm (căutare secvențiala):

```
caut(x[1..n],v)
1:  gasit ← False
2:  i ← 1
3:  WHILE (gasit==False) AND (i<=n) DO
4:      IF x[i]=v                          // t1(n)
5:          THEN gasit ← True              // t2(n)
6:          ELSE  i ← i+1                  // t3(n)
7:      ENDIF
8:  ENDWHILE
9: RETURN gasit
```

Exemplu 4 (căutare secvențială)

Cel mai favorabil caz: x[1]=v

t1(n)=1, t2(n)=1, t3(n)=0

T(n)= 6

Cel mai defavorabil caz: v nu se află în x[1..n]

t1(n)=n, t2(n)=0, t3(n)=n

T(n)=3n+3

Marginile (inferioară și superioară) ale timpului de execuție sunt:

6<= T(n) <= 3(n+1)

Obs: limita inferioară este constantă iar cea superioară depinde liniar de dimensiunea pb.

$$t1(n)=\begin{cases} k & \text{dacă v este în x[1..n]} \\ n & \text{dacă v nu este în x[1..n]} \end{cases}$$

$$t2(n)=\begin{cases} 1 & \text{dacă v este în x[1..n} \\ 0 & \text{dacă v nu este în x[1..n]} \end{cases}$$

$$t3(n)=\begin{cases} k-1 & \text{dacă v este în x[1..n} \\ n & \text{dacă v nu este în x[1..n]} \end{cases}$$

Exemplu 4 (căutare secvențială II)

```
Caut2(x[1..n],v)
1:  i ← 1
2:  while x[i]!=v and i<n do
3:      i ← i+1
4:  endwhile
5: if x[i]!=v then gasit ← false
6:                else gasit ← true
7: endif
8: return gasit
```

Cel mai favorabil caz (valoarea v se află pe prima poziție):

$T(n)=4$

Cel mai defavorabil caz (valoarea v se află pe ultima poziție sau nu se află în tablou):

$T(n)=1+n+(n-1)+2=2n+2$

Daca se consideră ca operație dominantă comparația x[i]!=v și atunci:

Cel mai favorabil caz: $T(n)=2$

Cel mai defavorabil caz: $T(n)=n+1$

Structura

- In ce consta analiza eficienței algoritmilor ?
- Cum se poate măsura eficiența algoritmilor ?
- Exemple
- Analiza în cazul cel mai favorabil, în cazul cel mai defavorabil și în cazul mediu

Analiza în cel mai favorabil și cel mai defavorabil caz

Analiza în cazul cel mai favorabil:

- furnizează o margine inferioară pentru timpul de execuție

- Permite identificarea algoritmilor ineficienți (dacă un algoritm are un cost ridicat chiar și în cel mai favorabil caz atunci el nu reprezintă o soluție acceptabilă)

Analiza în cazul cel mai defavorabil:

- Furnizează cel mai mare timp de execuție în raport cu toate datele de intrare de dimensiune n (reprezintă o margine superioară a timpului de execuție)

- Marginea superioară a timpului de execuție este mai importantă decat marginea inferioară

Analiza în cazul mediu

Pentru anumite probleme cazul cel mai favorabil si cel mai defavorabil sunt cazuri rare (excepții)

Astfel … timpul de execuție în cazul cel mai favorabil respectiv în cazul cel mai defavorabil nu furnizează suficientă informație

In aceste cazuri se efectuează o altă analiză… analiza cazului mediu

Scopul acestei analize este să furnizeze informații privind comportarea algoritmului în cazul unor date de intrare arbitrare (care nu corespund neapărat nici celui mai favorabil nici celui mai defavorabil caz)

Analiza în cazul mediu

Aceasta analiză se bazează pe cunoașterea distribuției de probabilitate a datelor de intrare

Aceasta înseamnă cunoașterea (estimarea) probabilității de apariție a fiecăreia dintre instanțele posibile ale datelor de intrare (cât de frecvent apare fiecare dintre posibilele valori ale datelor de intrare)

Timpul mediu de execuție este valoarea medie (în sens statistic) a timpilor de execuție corespunzători diferitelor instanțe ale datelor de intrare

Analiza în cazul mediu

Ipoteze. Presupunem că sunt satisfăcute următoarele ipoteze:

- datele de intrare pot fi grupate în clase astfel încât timpul de execuție corespunzător datelor din aceeași clasă este același
- sunt m=M(n) clase cu date de intrare
- Probabilitatea de apariție a unei date din clasa k este P_k
- Timpul de execuție al algoritmului pentru date de intrare aparținând clasei k este $T_k(n)$

In aceste ipoteze timpul mediu de execuție este:

$$T_{mediu}(n)= P_1T_1(n)+P_2T_2(n)+\ldots+P_mT_m(n)$$

Observație: dacă toate clasele de date au aceeași probabilitate atunci timpul mediu de execuție este:

$$T_{mediu}(n)=(T_1(n)+T_2(n)+\ldots+T_m(n))/m$$

Analiza în cazul mediu

Exemplu: căutare secvențială (operație dominantă: comparația)

Ipoteze privind distribuția de probabilitate a datelor de intrare:

- Probabilitatea ca valoarea v să se afle in tablou: p
  - valoarea v apare cu aceeași probabilitate pe fiecare dintre pozițiile din tablou
  - probabilitatea ca valoarea v să se afle pe poziția k este 1/n

- Probabilitatea ca v să nu se afle in tablou: 1-p

$T_{mediu}(n)=p(1+2+\ldots+n)/n+(1-p)n=p(n+1)/2+(1-p)n=(1-p/2)n+p/2$

Daca $p=0.5$ se obține $T_{mediu}(n)=3/4\ n+1/4$

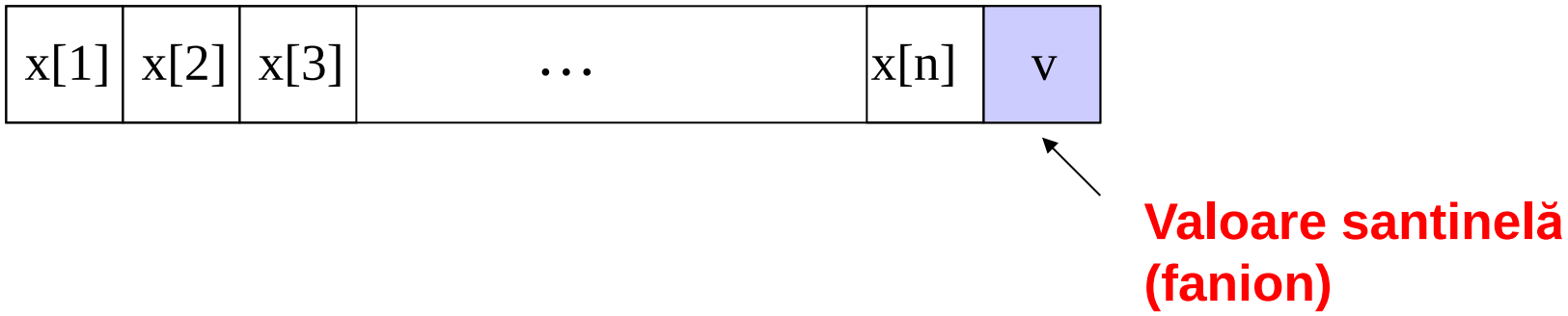
Concluzie: timpul mediu de execuție al algoritmului de căutare secvențială depinde liniar de dimensiunea datelor de intrare

Analiza în cazul mediu

Exemplu: căutare secvențială (varianta cu santinelă sau fanion)

Idee de bază:

- Tabloul este extins cu un element suplimentar (pe poziția n+1) a cărui valoare este v
- Tabloul este parcurs până la întâlnirea lui v (valoarea va fi găsită în cel mai rău caz pe poziția n+1 – corespunde situației când tabloul x[1..n] nu conține valoarea v)



Analiza în cazul mediu

Algoritm:

```
Căutare_fanion(x[1..n],v)
i ← 1
WHILE x[i]!=v DO
    i ← i+1
ENDWHILE
RETURN i
```

Operație dominantă: comparația

Ipoteză: Probabilitatea ca valoarea v să fie pe pozitia k din {1,2,…,n+1} este 1/(n+1)

Timpul mediu de execuție:

$T_{mediu}(n)=(1+2+\ldots+(n+1))/(n+1)$

$=(n+2)/2$

Observație:

- Schimbând ipoteza privind distribuția datelor de intrare valoarea timpului mediu se modifică (în cazul căutarii secvențiale dependența de dimensiunea problemei rămâne liniară)

Observație. Timpul mediu de execuție NU este in mod necesar media aritmetică dintre timpii de execuție corespunzători cazurilor extreme (cel mai favorabil si cel mai defavorabil)

Rezumat: etapele analizei eficienței

- Pas 1: Identificarea dimensiunii problemei
- Pas 2: Stabilirea operației dominante
- Pas 3: Determinarea numărului de execuții ale operatiei dominante
- Pas 4. Dacă numărul de execuții ale operației dominante depinde de proprietățile datelor de intrare atunci se analizează
  - Cel mai favorabil caz
  - Cel mai defavorabil caz
  - Cazul mediu
- Pas 5. Se stabilește ordinul (clasa) de complexitate (vezi cursul următor)

Următorul curs va fi …

.. tot despre analiza eficienței algoritmilor.

Mai concret, vor fi prezentate:

- Ordin de creștere
- Notații asimptotice
- Clase de complexitate

Intrebare de final

Se consideră un algoritm pentru calculul produsului dintre o matrice A[1..m,1..n] si un vector x[1..n]:

y[i]=A[i,1]*x[1]+A[i,2]*x[2]+…+A[i,n]*x[n] pentru i=1..m

```
Produs(A[1..m,1..n],x[1..n])
FOR i ←1,m DO
  y[i] ←0
  FOR j ←1,n DO
     y[i] ←y[i]+A[i,j]*x[j]
  ENDFOR
ENDFOR
RETURN y[1..m]
```

Numărul de operații de înmulțire efectuate este:

a) m
b) n
c) m+n
d) m*n
e) (m-1)*(n-1)