Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Structuri de Date si Algoritmi

- suport de curs -

Dobrovat Anca - Madalina

An universitar 2019 – 2020
Semestrul I
Seriile 21 + 25

Curs 1

01/10/2019

Generalitati despre curs

1. Curs - marti, orele 12-14, Amf. Titeica

2. Seminar - o data la 2 saptamani

3. Laborator - o data la 2 saptamani

4. Prezenta la curs: nu e obligatorie

Agenda cursului

1. Regulamente UB si FMI

2. Prezentarea disciplinei

3. Primul curs

Agenda cursului

1. Regulamente UB si FMI

2. Prezentarea disciplinei

3. Primul curs

1. Regulamente UB si FMI

Lucruri bine de stiut de studenti:

regulament privind activitatea studenților la UB:

https://www.unibuc.ro/wp-content/uploads/sites/7/2018/07/Regulament-privind-activitatea-profesionala-a-studentilor-2018.pdf

regulament de etică și profesionalism la FMI:

http://fmi.unibuc.ro/ro/pdf/2015/consiliu/Regulament_etica_FMI.pdf

Se consideră incident minor cazul în care un student/ o studentă:
a. preia codul sursă/ rezolvarea unei teme de la un coleg/ o colegă și pretinde că este rezultatul efortului propriu;

Se consideră incident major cazul în care un student/ o studentă:
a. copiază la examene de orice tip;

3 incidente minore = un incident major = exmatriculare

1. Regulamente UB si FMI

Lucruri bine de stiut de studenti:

Cazuri

...toarele:
...ELENA, grupa 403, a fe... ...u găsit...
...materiale este o copiută ... . În
...locul ocupat din amfiteatru ...
...conținea o foaie scrisă de o altă
...diferit, cu o altă cerneală).
...încadrează evenimentul drept incident major.

Exmatriculare pentru fraudă
– cu drept de înscriere la admitere
– conform consiliului FMI 12.03.2016

Agenda cursului

1. Regulamente UB si FMI

2. Prezentarea disciplinei

3. Primul curs

2. Prezentarea disciplinei

2.1 Obiectivele disciplinei

Ofera o baza de pornire pentru alte cursuri

Obiectivul general al disciplinei:

"Studentii isi vor dezvolta capacitatea de a intelege si de a implementa algoritmi si structuri de date precum şi capacitatea de a analiza si rezolva probleme.

Utilizarea unor structuri de date adecvate fiecarei probleme in parte.

Elaborarea unor programe eficiente ce implementeaza algoritmii si structurile invatate." (extras din fisa disciplinei)

Obiective specifice:

Introducerea structurilor de date, de la cele mai simple la unele complexe, cu justificare pentru necesitatea utilizarii lor, cu demonstratii pentru corectitudinea si eficienta lor.

2. Prezentarea disciplinei

2.2 Programa cursului

1. Algoritmi. Complexitate. (recapitulare) Teorema Master.
2. Clasa algoritmilor de sortare bazati pe comparatii intre chei. Heap-Sort. Quick-Sort. Shell-Sort.Teorema Limitei Inferioare.
3. Structuri de date elementare: liste, stive, cozi, arbori. Aplicatii.
4. Arbori binari de cautare. Echilibrare AVL. Teorema AVL.
5. Arbori binari stricti. Proprietati. Aplicatii la codificare.
6. Tabele de dispersie. Rezolvarea coloziunilor prin inlantuire si prin adresare deschisa.
7. Grafuri. Parcurgeri cu aplicatii. Arbori partiali de cost minim: Prim si Kruskal. Reuniune si apartenenta.
8. Scurta trecere in revista a tehnicilor de programare a algoritmilor.

2. Prezentarea disciplinei

2.2 Programa cursului

1. Algoritmi

Corectitudinea algoritmilor.

Analiza performantei algoritmilor.

Cateva clase de complexitate pentru comportarea asimptotica a algoritmilor.

2. Prezentarea disciplinei

2.2 Programa cursului

2. Structuri lineare in alocare secventiala si in alocare dinamica (inlantuita)

Operatii pe liste: traversare, cautare, inserare, stergere.

Tipuri particulare de liste (cu nod marcaj, circulare, dublu inlantuite).

Aplicatii ale listelor: reprezentarea numerelor mari, reprezentari de polinoame.

Multiliste. Aplicatii: reprezentarea matricilor rare, reprezentari de grafuri.

Structuri lineare cu restrictii la intrare/iesire: stive si cozi. Aplicatii.

2. Prezentarea disciplinei

2.2 Programa cursului

3. Structuri arborescente

Arbori oarecari. Definitii, terminologie, reprezentari, parcurgeri.

Arbori binari. Reprezentari, parcurgeri.

Arbori binari stricti. Proprietati matematice. Aplicatii.

Arbori binari de cautare. Operatii: cautare, inserare, stergere.

Algoritmul de cautare binara si performanta lui.

Arbori binari echilibrati AVL. Performanta cautarii in arbori binari de cautare echilibrati AVL.

2. Prezentarea disciplinei

2.2 Programa cursului

4. Algoritmi de sortare pentru multimi statice (vectori)

Clasa algoritmilor de sortare bazati pe comparatii intre chei.

- Sortarea prin insertie.
- Sortarea prin selectie.
- Sortarea prin interschimbare.
- Sortarea Shell.
- Sortarea cu ansamble (HeapSort).
- Sortarea rapida (QuickSort).

Limita inferioara a performantei algoritmilor de sortare bazati pe comparatii intre chei.

Sortarea prin interclasare (MergeSort).
Sortarea lexicografica.

2. Prezentarea disciplinei

2.2 Programa cursului

5. Arbori binari stricti cu ponderi

Algoritmul lui Huffman.

Aplicatii la codificarea binara.

Aplicatii la interclasarea optimala a mai multor siruri.

2. Prezentarea disciplinei

2.2 Programa cursului

6. Tabele de dispersie

Functii de dispersie.

Rezolvarea coliziunilor prin inlantuire.

Rezolvarea coliziunilor prin adresare directa.

Cautare, inserare, stergere in tabele de dispersie.

Dispersie universala.

2. Prezentarea disciplinei

2.2 Programa cursului

7. Grafuri

Parcurgeri cu aplicatii.

Arbori partiali de cost minim: Prim si Kruskal.

Reuniune si apartenenta.

1. Scurta trecere in revista a tehnicilor de programare a algoritmilor.

2. Prezentarea disciplinei

2.2 Programa cursului

8. Algoritmi - tehnici de programare

Scurta trecere in revista a tehnicilor de programare a algoritmilor.

Aplicatii - Greedy, Divide et Impera, (in masura timpului si Backtracking si Programare Dinamica).

2. Prezentarea disciplinei

2.3 Bibliografie

1. A. V. Aho, J. E. Hopcroft, J. D. Ullman: "Data Structures and Algorithms", Addison-Wesley Publ. Comp., 1983.
2. R. Ceterchi: "Structuri de date. Aspecte matematice si aplicatii", Editura Univ. din Bucuresti, 2001.
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest: "Introduction to Algorithms", The MIT Press, 1990 (si editiile ulterioare).
4. D.E. Knuth :"Tratat de programarea calculatoarelor", vol. I si III.
5. I. Tomescu: "Data Structures", Editura Univ. din Bucuresti, 2006.
6. N.Wirth: "Algorithms+Data Structures = Programs", Prentice Hall Inc., 1976.
7. Orice alt material (editie printabila sau electronica) ce va ajuta!

2. Prezentarea disciplinei

2.4 Regulament de notare si evaluare

Laborator – 1/3 din nota finala
Examen – 2/3 din nota finala
- **Teorie**
- **Exercitii**

Seminar - maxim 1p

Obs.
1. Cele 2 probe de evaluare sunt obligatorii.
2. Studentii care nu obtin cel putin nota 5 pentru activitatea de Laborator nu pot intra in examen
3. Studentii care nu obtin cel putin nota 5 la examenul scris, indiferent ca media notelor obținute este >=5, vor fi considerati RESTANTIERI.
4. Studentilor care nu obtin cel putin nota 5 la examenul scris NU li se considera punctajul de la seminar.

Agenda cursului

1. Regulamente UB si FMI

2. Prezentarea disciplinei

3. Primul curs

Curs 1 - Cuprins

1. Algoritmi

Definire. Proprietati

Corectitudine.

Analiza performantei algoritmilor.

Cateva clase de complexitate pentru comportarea asimptotica a algoritmilor.

2. Structuri de date elementare

Clasificare. Operatii elementare.

Structuri elementare de date

Date simple:

- numere întregi, reale, caractere

Programele prelucrează volume mari de date => eficienta: organizarea datelor în structuri.

Obs: Dacă se face o alegere optimă a structurii de date, implementarea va conduce la un algoritm eficient, care utilizează mai puţine resurse (ca de exemplu memoria necesară şi timpul de execuţie)

Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Structuri elementare de date

Structurile de date pot fi **clasificate** după diferite criterii:

1. Dupa tipul elementelor: **omogene** si **neomogene**

2. Dupa modul de localizare a elementelor structurii: **cu acces direct** (e.g. prin numarul de ordine) si **cu acces secvential** (accesul la un element se face parcurgand toate elementele aflate inaintea lui.

3. Dupa locul unde sunt create (tipul de memorie): **interne** (in memoria interna) si **externe**

4. Dupa timpul de utilizare: **temporare** si **permanente**

5. Dupa stabilitatea structurii: **statice** si **dinamice**

Structuri elementare de date

Operatii care se pot realiza asupra structurilor de date:

1. Creare
2. Consultare
3. Actualizare
4. Sortare
5. Copiere / Mutare
6. Redenumire
7. Divizare / Reuniune
8. Stergere

Structuri elementare de date

Structuri statice:

- de tip tablou

- de tip şir de caractere

- de tip articol

- de tip fişier

Structuri dinamice:

- de tip LISTĂ

- de tip STIVĂ

- de tip COADĂ

- de tip GRAF

- de tip ARBORE

Algoritmi

Definitie Prin **algoritm** vom înțelege o secvenţă finită de comenzi explicite şi neambigue care executate pentru o mulţime de date (ce satisfac anumite condiţii iniţiale), conduce în timp finit la rezultatul corespunzător.

Caracteristici:

- Generalitate
- Claritate
- Finititudine
- Corectitudine
- Performanță
- Robustețe

Descriere: Limbaj natural / Pseudocod, Diagramă (schemă logică), program etc.

Corectitudinea algoritmilor

! Reamintire din cursul de Programarea Calculatoarelor

Corectitudine logica a unui program => Algoritmul analizat produce rezultatul dorit dupa efectuarea unui numar finit de operatii.

Modalitati de verificare a corectitudinii

- **Experimentala** (prin testare): algoritmul este executat pentru un set de date de intrare => relativ simpla dar nu garanteaza corectitudinea

- **Formala** (prin demonstrare): se demonstreaza ca algoritmul produce rezultatul corect pentru orice set de date de intrare care satisface cerintele problemei => dificila in aplicarea pentru algoritmi complecsi, dar garanteaza corectitudinea

Corectitudinea algoritmilor

Preconditii si postconditii ([1])

Preconditii = proprietati satisfacute de datele de intrare

Postconditii = proprietati satisfacute de datele de iesire (rezultate)

Exemplu:
Sa se determine valoarea maxima dintr-un sir nevid.

Preconditii: n>=1 (secventa este nevida)

Postconditii: m (valoarea maxima) = max{x[i]; 1<=i<=n} (sau m >=x[i] pentru orice i) => m contine cea mai mare valoare din x[1..n]

[1] http://web.info.uvt.ro/~dzaharie/alg/alg2013_folii3.pdf

Corectitudinea algoritmilor

Asertiuni

Asertiune = afirmatie (adevarata) privind starea algoritmului

Limbajele de programare permit specificarea unor asertiuni si generarea unor exceptii daca asertiunea nu este satisfacuta. In C: "**assert.h**" .

Etapele verificarii corectitudinii

1. Identificarea preconditiilor si a postconditiilor
2. Adnotarea algoritmului cu asertiuni astfel incat:
   – Preconditiile sa fie satisfacute
   – Asertiunea finala sa implice postconditiile
3. Fiecare pas de prelucrare asigura modificarea starii algoritmului astfel incat asertiunea urmatoare sa fie adevarata.

Corectitudinea algoritmilor

Studiu de caz – corectitudinea trecerii unui numar in baza 16

Idee initiala
– pentru un set initial programul pare corect.

main.c

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Verificarea corectitudinii transformarii in baza 16!\n");
    int n,o;
    printf("n = ");
    scanf("%d", &n);
    // se iau resturile in ordine inversa
    o = 0;
    while(n!=0)
    {
        o = o * 10 + n%16;
        n = n / 16;
    }
    // se inverseaza ordinea resturilor
    while(o!=0)
    {
        n = n * 10 + o%10;
        o = o / 10;
    }
    printf("%d", n);
```

baza16

```
Verificarea corectitudinii transformarii in baza 16!
n = 19
13
Process returned 0 (0x0)   execution time : 2.799 s
Press ENTER to continue.
```

Corectitudinea algoritmilor

Studiu de caz – corectitudinea trecerii unui numar in baza 16

```c
int main()
{
    printf("Verificarea corectitudinii transformarii in baza 16!\n");
    int n,o;
    printf("n = ");
    scanf("%d", &n);
    // se iau resturile in ordine inversa
    o = 0;
    while(n!=0)
    {
        o = o * 10 + n%16;
        n = n / 16;
    }
    // se inverseaza ordinea resturilor
    while(o!=0)
    {
        n = n * 10 + o%10;
        o = o / 10;
    }
    printf("%d", n);


    return 0;
}
```

Idee initiala
– pentru un alt set initial programul este incorect.

baza16

```
Verificarea corectitudinii transformarii in baza 1
n = 32
2
Process returned 0 (0x0)   execution time : 2.369
Press ENTER to continue.
```

Complexitatea algoritmilor

Analiza complexităţii unui algoritm => determinarea resurselor de care acesta are nevoie pentru a produce datele de ieşire.

Resurse - timpul de executare
- spatiu de memorie etc.

Obs: Modelul masinii pe care va fi executat algoritmul nu presupune existenta operatiilor paralele (operatiile se executa secvential).

Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Complexitatea algoritmilor

Analiza complexităţii unui algoritm => determinarea resurselor de care acesta are nevoie pentru a produce datele de ieşire.

Resurse - timpul de executare
- spatiu de memorie etc.

Obs: Modelul masinii pe care va fi executat algoritmul nu presupune existenta operatiilor paralele (operatiile se executa secvential).

Notatie: T(n) – timp de rulare al unui algoritm (in general masurat in nr. de comparatii sau de mutari)

Cazuri:
- cel mai favorabil
- cel mai nefavorabil
- mediu

Complexitatea algoritmilor

De ce se alege, in general, cazul cel mai defavorabil?

- este cel mai raspandit

- timpul mediu de executare este de multe ori apropiat de timpul de executare in cazul cel mai defavorabil

- ofera o limita superioara a timpului de executare (avem certitudinea ca executarea algoritmului nu va dura mai mult)

Complexitatea algoritmilor

Numararea operatiilor elementare realizate de un algoritm in cazul cel mai defavorabil [3]

Aflarea maximului unui vector de dimensiune n	# operații elementare realizate de algoritm în cazul cel mai defavorabil
`maxim = v[0];`	2 (o indexare + o atribuire)
`for (i=1; i<n;i++)`	2n (o atribuire + n comparații + (n-1) incrementări)
`    if (maxim < v[i])`	2(n-1) (n-1 indexări + n-1 comparații)
`        maxim = v[i];`	2(n-1) (n-1 indexări + n-1 atribuiri)
`return maxim`	1 (o întoarcere a rezultatului)

T(n) = 6n-1 operații elementare

[3] Alexe Bogdan – Programare procedurala (note de curs 2015)

Complexitatea algoritmilor

Notatii:

T(n) – timp de rulare al unui algoritm (in general masurat in nr. de comparatii sau de mutari)

C_{max} – numarul maxim de comparatii (obtinut in cazul cel mai defavorabil)

C_{min} – numarul minim de comparatii (cazul cel mai favorabil)

M_{max} – numarul maxim de mutari (operatii elementare) – caz defavorabil

M_{min} – numarul minim de mutari – caz favorabil

Complexitatea algoritmilor

Exemple

[2] Albeanu G – Programare procedurala (note de curs 2013)

1. Produsul a doua numere complexe ([2])

```
int main()
{
float a,b,c,d, p, q;
float t1, t2;
scanf("%f%f%f%f", &a, &b, &c, &d);
t1 = a*c; t2 = b*d; p = t1 – t2;
t1 = a*d; t2 = b*c; q = t1 + t2;
printf("Real = %f, Imaginar = %f", p, q);
}
```

- memorie pentru 8 variabile

Operatii elementare: 4 inmultiri, o adunare si o scadere

Operatia de inmultire e mai costisitoare decat adunarea / scaderea.

Complexitatea algoritmilor

Exemple

[2] Albeanu G – Programare procedurala (note de curs 2013)

1. Produsul a doua numere complexe ([2])

```
int main()
{
float a,b,c,d, p, q;
float t1, t2;
scanf("%f%f%f%f", &a, &b, &c, &d);
t1 = a*c; t2 = b*d; p = t1 – t2;
t1 = a*d; t2 = b*c; q = t1 + t2;
printf("Real = %f, Imaginar = %f", p, q);
}
```

- memorie pentru 8 variabile

Operatii elementare: 4 inmultiri, o adunare si o scadere

```
int main()
{
float a,b,c,d, p, q;
float t1, t2, t3, t4;
scanf("%f%f%f%f", &a, &b, &c, &d);
t1 = a + b; t2 = t1 * c; t1 = d – c; t3 = a * t1;
q = t2 + t3; t1 = d + c; t4 = b * t1; p = t2 – t4;
printf("Real = %f, Imaginar = %f", p, q);
}
```

- memorie pentru 10 variabile

Operatii elementare: 3 inmultiri, 3 adunari si 2 scaderi

Operatia de inmultire e mai costisitoare decat adunarea / scaderea.

Complexitatea algoritmilor

Exemple

Cate operatii se executa daca se citesc initial numerele 97 si 99?

2. Cmmdc a 2 numere

Euclid:

```
int a,b,r;
scanf("%d%d", &a, &b);
r = a % b;
while (r!=0)
{ a = b;
  b = r;
  r = a %b;
}
printf("Cmmdc = %d", b);
```

Scaderi repetate:

```
int a,b,r;
scanf("%d%d", &a, &b);
while (a != b)
        if (a >b )
                a = a – b;
        else
                b = b – a;

printf("Cmmdc = %d", a);
```

Algoritm brut:

```
int a,b,c,i,min;
scanf("%d%d", &a, &b);

if (a < b) min = a;
        else min = b;

for(i = 1 ; I <= min; i++)
        if ( %i==0 && b%i == 0)
                c = i;
printf("Cmmdc = %d", c);
```

Complexitatea algoritmilor

Exemple

3. Determinarea maximului dintr-un sir

```
int main()
{
int a[100], n, i, max;
// citire vector

max = v[1];
for(i = 2; i <= n; i++)
      if (max < v[i]) max = v[i];

printf("Maximul = %",max);
return 0;
}
```

Obs: $C_{max} = C_{min} = C_{mediu} = n - 1$

$M_{min} = 0$ // maximul se afla pe prima pozitie

$M_{max} = n - 1$ // maximul se afla pe ultima pozitie in vectorul initial

$M_{mediu} = (n - 1) / 2$

Exemplu: n = 6

v = (-32, 1, 56, 89, -20, 100)

- max = 100, gasit dupa 5 comparatii

Complexitatea algoritmilor

Exemple

1.Un program eficient pentru verificarea primalitatii unui numar.

-n comparatii

-n/2 comparatii

-sqrt(n) comparatii

-< sqrt(n)/2+1 comparatii

2. Determinati simultan maximul si minimul dintr-un sir, folosind 3n/2 + O(1) comparatii

Complexitatea algoritmilor

Exemple

4. Diferenta maxima dintre 2 elemente ale unui vector de dimensiune n [3]

```
difMaxima = 0;
for (i=0; i<n;i++)
      for (j = i+1; j< n; j++ )
            if (difMaxima < abs(v[i]-v[j]))
                  difMaxima = abs(v[i]-v[j]);
return difMaxima
```

Care este complexitate timp a algoritmului?

$T(n) = O(n^2)$

Există un algoritm de complexitate mai scăzută care rezolvă problema?

[3] Alexe Bogdan – Programare procedurala (note de curs 2015)

Complexitatea algoritmilor

Exemple

4. Diferenta maxima dintre 2 elemente ale unui vector de dimensiune n [3]

Algoritm eficient

```
//Aflăm minimul și maximul unui vector folosind 3n/2 + O(1) comparatii.
……………………
difMaxima = abs(maxim – minim)
return difMaxima
```

Care este complexitate timp a algoritmului?

$T(n) = O(n)$

[3] Alexe Bogdan – Programare procedurala (note de curs 2015)

Complexitatea algoritmilor

Exemple

5. Cautarea unei valori intr-un sir **ordonat** (Cautarea binara)

```
int main()
{
int left = 0, right = n - 1;
int mid = (left + right) / 2;
while (left <= right && val != v[mid])
       {
       if (val < v[mid]) right = mid - 1;
       else left = mid + 1;
       mid = (left + right) / 2;
       }
if (v[mid] == val) loc = mid;
else loc = UNDEFINED;
}
```

- **$O(\log_2 n)$**

Exemplu: caut (fara succes) elementul 10 in sirul v = (20, 30, 40, 50, 60, 70, 80, 90, 100)

- compar 10 cu 60 (elem din mijloc); 10 != 60
- 10 < 60 = > caut in v = (20, 30, 40, 50)

- compar 10 cu 30 (noul elem din mijloc)
- 10 < 30 = > caut in v = (20)

- compar 10 cu 20 (unicul elem)
- cautare fara succes

n = 9, $[\log_2 9] = 3$

Complexitatea algoritmilor

Exemple

6. Ordonarea unui sir folosind Interschimbarea directa

```
int main()
{
int v[100], n, i, j, aux;
// citire vector
for (i = 1; i< n; i++)
      for(j = i+1; j <= n; j++)
            If (v[i] > v[j])
                  { aux = v[i];
                    v[i] = v[j];
                    v[j] = aux;
                  }
// Afisare vector ordonat
}
```

Caz	Comparatii	Mutari
Cel mai favorabil	n(n – 1) / 2	0
Cel mai defavorabil	n(n – 1) / 2	3n(n – 1) / 2
mediu	n(n – 1) / 2	3n(n – 1) / 4

$O(n^2)$

Complexitatea algoritmilor

Exemple

7. Ordonarea unui sir folosind Insertia directa

```
int main()
{
int v[100], n, i, j, aux;
// citire vector
for (i = 2; i<=n; i++)
{      x = v[i];
            j = i – 1;
            while (j>0 && x < v[j])
                  {      v[j+1] = v[j];
                         j--;
                  }
            v[j+1] = x;
}
// Afisare vector ordonat
}
```

Caz	Comparatii	Mutari
Cel mai favorabil	**$n - 1$**	**$2(n - 1)$**
Cel mai defavorabil	**$\frac{1}{2} n^2 + \frac{1}{2} n - 1$**	$\frac{1}{2} n^2+3/2n-2$
mediu	**$(\frac{1}{2} n^2 + \frac{1}{2} n - 1)/2$**	$C_{mediu}+2(n-1)$

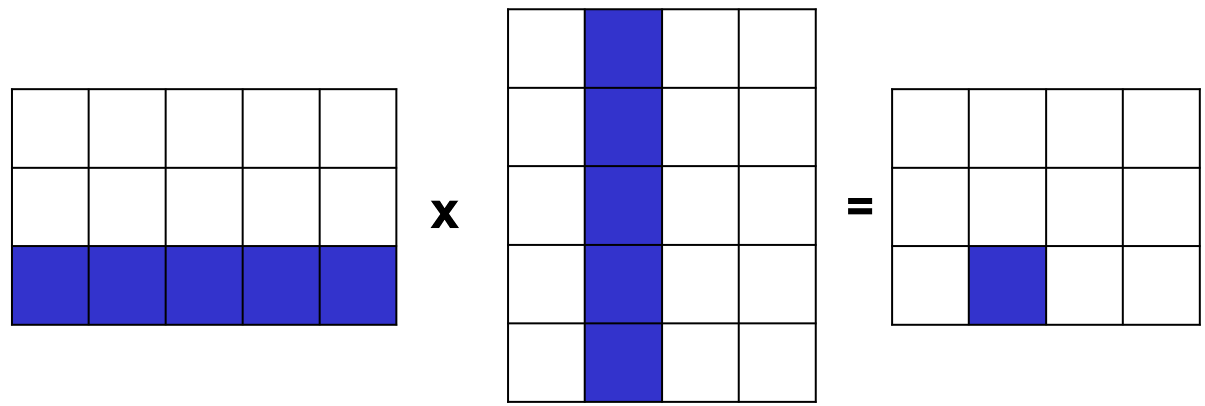
$O(n^2)$

Complexitatea algoritmilor

Exemple

8. Inmultirea a doua matrice

```
int main()
{
int a[10][20], b[20][30], c[10][30];
int n, m, p, i, j, k;
// citire matrice a si b
for(i=1; i<=n; i++)
     for(k=1; k<=p; k++)
          { c[i][k] = 0;
             for(j=1; j<=m; j++)
                  c[i][k] = c[i][k] + a[i][j] * b[j][k];
          }
// Afisare matrice produs
}
```



$$A_{n,m} \times B_{m,p} = C_{n,p}$$

O(m*n*p)

Complexitatea algoritmilor

Exemple

9. Inmultirea OPTIMA a unui sir de matrice

Avem un șir de matrice A_1, A_2, A_3, ..., A_n astfel încât numărul de coloane ale matricei A_i = numărul de linii ale matricei A_{i+1}. Considerăm matricea $A = A_1 * A_2 * A_3 * ... * A_n$. Știm că înmulțirea matricelor e asociativă, deci putem înmulți matricele în orice ordine vrem (numim o asemenea ordine = parantezare) și obținem același rezultat:

pentru n=4: $A = (A_1 * A_2) * (A_3 * A_4) = A_1 * (A_2 * A_3) * A_4$

Totuși, fiecare parantezare presupune un număr diferit de înmulțiri.
Care este parantezarea optimă ce presupune cele mai puține înmulțiri?

Exemplu: $A = A_{2,4} * A_{4,5} * A_{5,3} * A_{3,2}$ ($A_{2,4}$ – matrice cu 2 linii și 4 coloane)
$A = ((A_{2,4} * A_{4,5}) * A_{5,3}) * A_{3,2}$) – parantezare optimă
(40 + 30 + 12 = 82 de înmulțiri)

Complexitatea algoritmilor

Notatia asimptotica

T(n) – timp de rulare al unui algoritm (comparatii / mutari)

Obs: Exista un timp minim si un timp maxim de rulare

$$C_{min} <= T(n) <= C_{max}$$

Complexitatea algoritmilor

Notatia asimptotica

T(n) – timp de rulare al unui algoritm (comparatii / mutari)

Obs: Exista un timp minim si un timp maxim de rulare

$$C_{min} <= T(n) <= C_{max}$$

Margini superioare (si inferioare) ([4])

Timp de rulare T(n) - margine superioara

$T(n) <= \frac{1}{2} n^2 + (3/2)n - 2$ (expl.) => $T(n) = \mathbf{O(n^2)}$

Timp de rulare T(n) - margine inferioara

$3(n - 1) <= T(n)$ (expl) => $T(n) = \mathbf{\Omega(n)}$

Timp de rulare T(n) in cazul cel mai nefavorabil

$T(n) = a*C_{max} + b$ ($\exists$ const a ,b > 0) => $T(n) = \mathbf{\Theta(n^2)}$

[4] Ceterchi R. – Algoritmi si Structuri de Date (note de curs 2013)

Complexitatea algoritmilor

Notatia asimptotica

comportarea lui T (n) cind $n \to \infty$ ([4])

Formal

$f : N \to R_+$ (f asimptotic pozitiva)

$O(g) := \{f \mid \exists c > 0, \exists n_0$ a.i. $0 \le f(n) \le cg(n)$ oricare $n \ge n_0 \}$

$\Omega(g) := \{f \mid \exists c > 0, \exists n_0$ a.i. $0 \le cg(n) \le f(n)$ oricare $n \ge n_0 \}$

$\Theta(g) := \{f \mid \exists c_1, c_2 > 0, \exists n_0$ a.i. $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ oricare $n \ge n_0 \}$

[4] Ceterchi R. – Algoritmi si Structuri de Date (note de curs 2013)

Perspective

1. Se vor discuta directiile principale ale cursului, feedback-ul studentilor fiind foarte important!

- intelegerea notiunilor
- intrebari si sugestii

2. Cursul 2:

- Completari ale cursului 1 - notatii asimptotice
- Structuri lineare in alocare secventiala si in alocare dinamica (inlantuita).

Succes in anul universitar 2019 - 2020 !