# Clasa P

1. Clasa P; timpul polinomial
2. Exemple de probleme cu timp de calcul polinomial determinist

# Clasa P

Complexitatea timp a problemelor se modifică

- cu un factor polinomial (o ridicare la pătrat) atunci când trecem de la **MT** cu mai multe benzi la **MT** cu o singură bandă (ambele deterministe);
- cu un factor exponenţial atunci când trecem de la **MT** nedeterministe la **MT** deterministe (ambele cu o singură bandă).

# Clasa P

Conceptul de calculabilitate si nu un model de calculabilitate in particular

- diferenta de comportament intre clasa functiilor de tip polinomial si clasa functiilor de tip exponential;
- diferentele de timp de lucru de tip polinomial sunt aproape nesemnificative in timp ce diferentele de tip exponential sunt importante

# Clasa P

Toate modelele de calculabilitate deterministe “rezonabile” sunt POLINOMIAL ECHIVALENTE,

adică simularea unuia cu instrumentele celuilalt antrenează doar o creştere polinomială a timpului de lucru.

# Clasa P

Definitia 1

Notăm cu *P* clasa limbajelor decidabile în timp polinomial determinist de către MT deterministe cu o singură bandă de intrare:

$$P = \bigcup_{k \in N} TIME\,(n^k)$$

Această clasă de probleme are următoarele proprietăţi:

**(a)** este invariantă faţă de toate modelele de calculabilitate care sunt polinomial echivalente cu MT deterministe cu o singură bandă de intrare;

**(b)** corespunde clasei de probleme practic rezolvabile cu un calculator real.

# Clasa P

1. Clasa P; timpul polinomial
2. Exemple de probleme cu timp de calcul polinomial determinist

# Clasa P

Câteva precizări

- vom descrie algoritmii tot cu ajutorul etapelor şi paşilor, ca şi în cazul **MT**;
- vom calcula timpul de lucru al algoritmilor în 2 trepte:
  - ✓ vom căuta o limită superioară a numărului de etape şi paşi executaţi de algoritm pe o intrare oarecare n $\in$ ***N***,
  - ✓ vom examina fiecare pas al algoritmului pentru a determina dacă poate fi implementat în timp polinomial, cu ajutorul unui model de calculabilitate determinist, “rezonabil”,
  - ✓ întrucât compunerea a 2 polinoame este încă un polinom, vom putea conchide că algoritmul rulează în timp polinomial;

# Clasa P

- vom utiliza o metodă “rezonabilă” de codificare a problemelor:
  - ✓ vom folosi tot codificarea sub forma unei secvenţe pe care o vom nota cu **< >**
  - ✓ vom aprecia că o metodă de codificare este “rezonabilă” dacă ea foloseşte un timp de lucru polinomial .

# Clasa P

Teorema 1

Fie limbajul *PATH* = { <G,s,t> | G este un digraf în care există un drum de la nodul s la nodul t } => **PATH** $\in$ ***P*.**

# Clasa P

*demonstratie*

Următorul algoritm rezolvă problema *PATH* în timp polinomial:

M = “Fie secvenţa de intrare <G,s,t>, unde G este un digraf oarecare iar s şi t două noduri oarecare ale sale:

1. Se marchează nodul s.
2. Se repetă Pasul 3 atât timp cât mai pot fi marcate noi noduri:
   3. Se examinează toate arcele din G : dacă există un arc (a,b) de la nodul <u>marcat</u> a la nodul <u>nemarcat</u> b atunci se marchează nodul b.
4. Dacă t este marcat atunci M acceptă secvenţa de intrare <G,s,t>, altfel respinge.”

# Clasa P

Analizăm complexitatea timp a acestui algoritm:

(i)

- evident, prima şi ultima etapă se execută o singură dată;
- etapa a 3a se execută de cel mult m ori ;

⇒ numărul total de paşi este 1 + m + 1, deci O(m).

(ii)

- prima şi ultima etapă se implementează uşor în timp polinomial, în oricare dintre modelele deterministe "rezonabile";
- etapa a 3a presupune o scanare a nodurilor şi o testare a stării acestora: marcat / nemarcat; deci şi aceste operaţii se pot implementa în timp polinomial

⇒ complexitatea timp a acestui algoritm de rezolvare a problemei PATH este polinomială în raport cu numărul de noduri ale grafului.

# Clasa P

| Etapa | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Nr. pasi | 1 | $2m^2$ | | m |
| Nr. executii | 1 | m | | 1 |

$$\Rightarrow O(1).O(1) + O(m).O(m^2) + O(1).O(m) = O(m^3).$$

# Clasa P

Teorema 2

Fie limbajul *RELPRIME*= { <x,y> | (x,y)=1 }

=> *RELPRIME* $\in$ *P*.

*demonstratie*

*(i) construim o MT, E, care calculeaza cmmdc{x,y} cu algoritmul lui Euclid*

# Clasa P

E = “Fie secventa de intrare <x,y>, $\forall$ x,y$\in$N, reprezentate in baza 2:

1. Se repeta pasii 2 si 3 pana cand y=0:
   2. Se atribuie x$\leftarrow$x mod y.
   3. Se interschimba x si y.
4. Se returneaza x.”

*(ii) construim o MT, R, care rezolva problema RELPRIME apeland MT, E, ca subrutina*

R = “Fie secventa de intrare <x,y>, $\forall$ x,y$\in$N, reprezentate in baza 2:

1. Se ruleaza E pe intrarea <x,y>.
2. Daca rezultatul returnat de E este 1 atunci R accepta, altfel: respinge.”

# Clasa P

*(a) demonstram ca fiecare executie a ciclului format din etapa a 2-a si etapa a 3-a (eventual <u>cu exceptia</u> primei executii) reduce valoarea lui x cel putin la jumatate*

se executa etapa 2 => obtinem x<y (cf. def. operatorului mod)

se executa etapa 3 => obtinem x>y (pt ca x si y se interschimba)

se executa din nou etapa 2 => obtinem din nou x<y etc.

Distingem 2 cazuri:

- daca $x/2 \geq y \rightarrow x \bmod y < y \leq x/2$ => x se reduce cel putin la jumatate;
- daca $x/2 < y \rightarrow x \bmod y = x-y < x/2$ => x se reduce cel putin la jumatate.

# Clasa P

*(b) calculam numarul de executii ale ciclului format din etapa a 2-a si a 3-a*

fiecare executie a etapei 3 → valorile lui x si y se interschimba

=> fiecare executie a ciclului → valorile initiale ale x si y se reduc cel putin la jumatate, alternativ

=> numarul maxim de executii ale ciclului format din etapa 2 si 3 este

$$\min \{ 2\log_2 x, 2\log_2 y \}.$$

Dar $\log_2 x \sim |x_{baza=2}|$

=> numarul de executii ale ciclului format din etapa 2 si 3 este de ordin O(n).

Fiecare etapa se executa intr-un singur pas

=> fiecare etapa utilizeaza un timp polinomial

=> timpul total de executie al algoritmului este polinomial

# Clasa P

<u>Teorema 3</u>

$\forall$ L$\in$LIC => L$\in$P.

*Observaţie*:

Teorema anterioara: $\forall$ L$\in$LIC este decidabil dar algoritmul folosit în demonstraţie rulează în timp exponenţial.

*Justificare*:

Fie L $\in$ LIC => $\exists$G$\in$FNC astfel încât L=L(G) (cf. teorema anterioara).

Fie w$\in$L, |w|=n, n$\in$N; orice derivare în G pentru w va avea exact 2n-1 paşi

=> este suficient să verificăm toate derivările de lungine 2n-1:

dacă găsim o derivare care produce w, atunci MT accepta, altfel respinge.

Dar:

- lungimea derivărilor creşte polinomial cu lungimea cuvintelor dar
- numărul derivărilor de lungime k, k $\in$ N; creşte exponenţial cu lungimea derivărilor

=> trebuie căutat un algoritm care să lucreze în timp polinomial;

Metoda programării dinamice.

# Clasa P

***Informal*,**

metoda programării dinamice constă în rezolvarea problemei date prin rezolvarea subproblemelor de dimensiuni mari pe baza acumulării de informaţii despre subproblemele de dimensiuni mici.

***Formal*:**

metoda programării dinamice constă din determinarea soluţiei prin luarea unui şir de decizii $d_1,d_2,\ldots,d_n$ (unde $d_i$=transformă problema dată din starea $s_{i-1}$ în starea $s_i$, $\forall 2\leq i\leq n$) respectând <u>principiul de optim</u> (de minim sau de maxim).

Conform acestui principiu, dacă $d_1,d_2,\ldots,d_n$ este un şir optim de decizii care transformă problema dată din starea iniţială $s_0$ în starea finală $s_n$, atunci este îndeplinită una dintre condiţiile:

**a**) $d_k,\ldots,d_n$ este un şir optim de decizii care duce problema din starea $s_{k-1}$ în starea $s_n$, $\forall 1\leq k\leq n$;

**b**) $d_1,\ldots,d_k$ este un şir optim de decizii care duce problema din starea $s_0$ în starea $s_k$, $\forall 1\leq k\leq n$;

**c**) $d_{k+1},\ldots,d_n$ şi $d_1,\ldots,d_k$ sunt şiruri optime de decizii care duc problema din starea $s_k$ în starea $s_n$, respectiv din starea $s_0$ în starea $s_k$, $\forall 1\leq k\leq n$.

Cazul (a): metoda "înainte“, ordinea deciziilor: $d_n$, $d_{n-1},\ldots$, $d_{1;}$ $d_i$ depinde de $d_{i+1},\ldots,d_n$.

Cazul (b): metoda "înapoi“, ordinea deciziilor: $d_1$, $d_2,\ldots$, $d_n$. $d_i$ depinde de $d_1,d_2,\ldots,d_{i-1}$.

În cazul (c) spunem că aplicăm metoda mixtă.

# Clasa P

*ideea demonstraţiei*

Vom considera următoarele subprobleme:

"fie o variabilă oarecare A din G∈FNC şi fie un subcuvânt oarecare v al lui w, w∈L;

este adevărat că A ==>* v ?"

Următorul algoritm va memora intr-o matrice pătratică de ordin n x n, n=|w|, soluţiile acestor subprobleme astfel:

∀ $1 \le i \le j \le n$: celula (i,j) din matrice conţine mulţimea variabilelor din G care generează subcuvântul $w_i w_{i+1} \ldots w_j \subseteq w$.

Algoritmul completează celulele matricei pentru fiecare subcuvânt al lui w, indiferent de lungimea acestuia:1,2,…,k,…,n=|w|;

- incepe cu celulele pentru cuvintele de lungime 1: (i,i), $1 \le i \le n = |w|$;
- continuă cu celulele pentru cuvintele de lungime 2: (i,i+1), $1 \le i \le n-1$;
- apoi cu celulele pentru cuvintele de lungime 3 etc.,

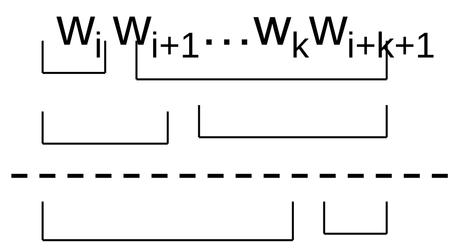
folosind la completarea celulelor pentru subcuvintele de lungime k, $2 \le k \le n$, informaţia din celulele deja completate pentru subcuvintele de lungime 1,2,...,k-1.

# Clasa P

Să presupunem, de exemplu, că algoritmul a determinat ce variabile din $G$ generează toate subcuvintele lui $w$ de lungime cel mult $k$.

Pentru a determina dacă o variabilă oarecare $A$ generează un subcuvânt oarecare $v$ de lungime $k+1$ al lui $w$, algoritmul:

(i) împarte acel subcuvânt $v$ în alte 2 subcuvinte nevide în toate cele $k$ moduri posibile;



# Clasa P

(ii) pentru fiecare divizare a lui v, algoritmul examinează fiecare regulă de tip A→BC pentru a verifica dacă:

- B generează $1^0$ subcuvânt al lui v,
- C generează al 2-lea subcuvant al lui v

şi face acest lucru folosindu-se de celulele deja calculate din matrice.

Dacă atât B cât şi C generează respectivele subcuvinte ale v,

=> A generează v

=> algoritmul adaugă neterminalul A în celula corespunzătoare din matrice;

(iii) algoritmul începe acest proces cu cuvintele de lungime 1, prin examinarea matricei pentru producţiile de tipul A → b.

# Clasa P

*demonstraţie*

Prezentăm algoritmul, notat D:

Fie $G \in GIC$, G în FNC, care generează $L \in LIC$ şi fie S simbolul de start al G.

D = "Fie cuvântul de intrare $w = w_1 w_2 \ldots w_n$:

1. Dacă $w = \lambda$ şi dacă producţia $S \rightarrow \lambda$ se află în G, atunci D acceptă. // este tratat cazul special al cuvantului vid

2. Pentru $\forall$ $i = 1, 2, \ldots, n$:

3. Pentru oricare variabila A din G:

4. Se verifică dacă producţia $A \rightarrow b$ unde $b = w_i$ se află în G.

5. Dacă da, atunci variabila A este depusă în celula (i,i) din matrice.

# Clasa P

6. Pentru ∀ k=2,…,n: // k = lungimea unui subcuvânt v al lui w
   7. Pentru ∀ i=1,2,…,n-k+1: // i = indicele iniţial (de start) al subcuvântului v din w
      8. Fie j=i+k-1: // j = indicele final al subcuvântului v din w
         9. Pentru ∀ t=i,…,j-1 // t = indicele final al primului subcuvânt al lui v ⊆ w
            10. Pentru orice producţie A → BC:
                11. Dacă celula (i,t) din matrice conţine variabila B
                    iar celula (t+1,j) din matrice conţine variabila C
                    atunci se depune variabila A în celula (i,j).
12. Dacă variabila S apare în celula (1,n) din matrice, atunci D acceptă, altfel D respinge."

# Clasa P

Analizăm algoritmul din punct de vedere al complexităţii timp; observăm că el se compune din 4 etape: 1, 2,…, 6,…, 12.

Complexitatea alg. se obţine prin însumarea complexităţilor acestor etape:

- Etapele 1 şi 12 se execută, evident, 1! dată şi au câte 1! pas;
- Pentru a calcula numărul de execuţii ale etapelor 2-5, procedăm din aproape în aproape:
  - ✓ etapa a 2-a se executa de $n=|w|$ ori;
  - ✓ pentru fiecare execuţie a etapei a 2-a, etapa a 3-a se execută de *m* ori, $m=card(V)$ = nr. de variabile din G =>
  - ✓ paşii 4 şi 5, cei mai "interiori" din etapa a 2-a, se execută de *n.m* ori,

    intrucât m = o constantă fixată, independentă de n

    → aceste etape sunt de ordin O(n);

# Clasa P

- Pentru a calcula numărul de execuţii ale etapelor 6-11, procedăm ca mai sus:
  - etapa a 6-a se execută de cel mult n ori;
  - pentru fiecare execuţie a etapei a 6-a, etapa a 7-a se execută tot de cel mult n ori;
  - pentru fiecare execuţie a etapei a 7-a, etapa a 8-a se execută 1! dată iar etapa a 9-a se execută de cel mult n ori;
  - pentru fiecare execuţie a etapei a 9-a, etapele a 10-a şi a 11-a se execută de cel mult r ori , unde r= numărul de producţii din G;

  intrucât r = o constantă fixată, independentă de n

  → etapa a 11-a – cea mai "interioară" din algoritm – se execută de $O(n^3)$ ori;

=> algoritmul este de ordin $O(1)+O(n)+O(n^3)+O(1) = O(n^3)$.

=> $L \in P$.

# Complexitatea timp

1. Clasa P; timpul polinomial
2. Exemple de probleme cu timp de calcul polinomial determinist