

Acest fișier conține o introducere în generalizarea clasificării problemelor conform unor clase de complexitate diferite de P și NP. Fișierul conține fragmente revăzute din capitolul 3 din "Introducere în Analiza Algoritmilor".

3.4 O ierarhie spațiu-timp a problemelor

În secțiunile precedente, dificultatea rezolvării problemelor a fost analizată în raport cu timpul consumat de algoritmi. Clasele P și NP sunt definite pe această bază. O altă resursă importantă este spațiul de memorie consumat. Astfel, ierarhizarea P - NP poate fi extinsă din perspectiva complexității spațiale impuse de rezolvarea problemelor.

Secțiunea prezintă, intuitiv, o ierarhie posibilă spațiu-timp a problemelor și, implicit, relații între complexitatea temporală și cea spațială. Sunt considerate doar probleme de decizie, fără ca generalitatea să fie afectată. În ceea ce privește metrica folosită pentru a măsura dimensiunea datelor și consumul de timp și spațiu, se reamintesc următoarele convenții:

- Dimensiunea datelor algoritmilor este un număr întreg pozitiv.
- Măsura timpului este unitară. Algoritmul efectuează operațiile considerate elementare (adunări, înmulțiri, comparații etc.) într-un număr limitat de unități de timp, adică în $\Theta(1)$.
- Datele sunt memorate ca șiruri de simboluri dintr-un alfabet Σ . În particular, putem alege $\Sigma = \{0, 1\}$, fără a afecta generalitatea rezultatelor relativ la posibilitățile curente ale mașinilor de calcul. Astfel, dacă dimensiunea unei date d este cel mult n , sunt necesari $O(\lg(n))$ biți pentru a stoca d .

De asemenea, convenim ca spațiul ocupat de datele nemodificate de un algoritm, de obicei datele de intrare, să nu participe în calculul complexității spațiale a algoritmului. Decizia este justificată dacă ne gândim că unul din scopurile analizei complexității este clasificarea problemelor în raport cu resursele consumate. Ar fi nepotrivit să considerăm că rezolvarea unei probleme P_1 este mai dificilă decât cea a unei alte probleme P_2 doar pentru că dimensiunea datelor lui P_1 este mai mare decât dimensiunea datelor lui P_2 . Din alt punct de vedere, astfel de date *read-only* pot fi stocate în "afara" algoritmului.

3.4.1 Clase de probleme rezolvate determinist

Fie $Q: I \rightarrow \{0,1\}$ o problemă de decizie, rezolvabilă prin mulțimea algoritmilor determinați Alg_Q , și $f: N \rightarrow R_+$ o funcție pe care o privim ca limită a complexității algoritmilor. Notăm:

$$Alg_Q^T(f) = \{Alg \in Alg_Q \mid \forall i \in I \bullet \text{timp}(Alg, i) = O(f(n))\}$$

$$Alg_Q^S(f) = \{Alg \in Alg_Q \mid \forall i \in I \bullet \text{spațiu}(Alg, i) = O(f(n))\},$$

unde $n = \dim(i)$ este dimensiunea datelor i , $\text{timp}(Alg, i)$ este timpul consumat de calculul serial $Alg(i)$, iar $\text{spațiu}(Alg, i)$ este spațiul maxim de lucru consumat în cursul execuției $Alg(i)$. Mai precis, să notăm:

- $\text{stări}(Alg, i)$ stările din secvența execuției $Alg(i)$. Fiecare stare este asociată cu un moment de timp t al execuției și desemnează mulțimea variabilelor existente la momentul t al execuției precum și instrucțiunea curent executată.
- $\text{spațiu_stare}(s)$ spațiul consumat de variabilele din starea s , excluzând variabilele care desemnează date nemodificabile ale algoritmului.

Deci, pentru un algoritm serial, $\text{spațiu}(Alg, i) = \max\{\forall s \in \text{stări}(Alg, i) \bullet \text{spațiu_stare}(s)\}$.

Mulțimea $Alg_Q^T(f)$ conține acei algoritmi determinați care rezolvă problema Q în $O(f(n))$ unități de timp, iar $Alg_Q^S(f)$ conține acei algoritmi determinați care rezolvă problema Q folosind $O(f(n))$ unități de spațiu de lucru.

Definiția 3.26 Fie $f: N \rightarrow R_+$ o funcție totală peste N . Clasele de probleme rezolvabile prin algoritmi determinați, cu limitare f , temporală și, respectiv, spațială sunt:

$$\text{TIME}(f) =_{\text{def}} \{Q \mid Alg_Q^T(f) \neq \emptyset\}$$

$$\text{SPACE}(f) =_{\text{def}} \{Q \mid Alg_Q^S(f) \neq \emptyset\}$$

În particular,

$$P = \text{PTIME} =_{\text{def}} \bigcup_{k \geq 0} \text{TIME}(\lambda n. n^k)$$

$$\text{PSPACE} =_{\text{def}} \bigcup_{k \geq 0} \text{SPACE}(\lambda n. n^k)$$

$$\text{LOGSPACE} =_{\text{def}} \text{SPACE}(\lambda n. \lg(n)),$$

unde notația $\lambda n. R(n)$ desemnează o funcție cu parametrul n și rezultatul $R(n)$.

Ca exemplu, să clasificăm problema **GAP** (Problema accesibilității într-un graf). Pentru un graf G cu n noduri să se decidă existența unui drum între două noduri date, i și j , din G .

Propoziția 3.11 $GAP \in SPACE(\lg(n)^2)$ și $GAP \in TIME(n^2)$ (sau $GAP \in PTIME$).

1. $GAP \in SPACE(\lg(n)^2)$. Să construim un algoritm care rezolvă problema într-un spațiu de lucru $O(\lg(n)^2)$

```
GAP_S(G,i,j) { // G are nodurile V(G) și arcele E(G)
               // Există drum i..j în G?
    n = card(V(G));
    for(lung=0; lung < n; lung++)
        if(drum(i,j,lung)) return 1;
    // G și n sunt vizibile în funcția drum
    return 0;
}

drum(i,j,lung){
    if(lung = 0) return i=j ? 1 : 0;
    if(lung = 1) return (i,j) ∈ E(G);
    l1= ⌊lung/2⌋;
    l2= ⌈lung/2⌉;
    for(k = 1; k ≤ n; k++)
        if(drum(i,k,l1) ∧ drum(k,j,l2)) return 1;
    return 0;
}
```

Algoritmul **drum** încearcă să formeze drumul $i..j$ din drumurile $i..k$ și $k..j$, pe jumătate mai scurte ca număr de arce decât drumul $i..j$. Procesul este repetat până ce lungimea unui drum devine 0 sau 1. Corectitudinea algoritmului derivă din următoarea observație: un drum $x..y$ de lungime 1 există dacă și numai dacă este satisfăcută una din condițiile:

- $l = 0$ și $x = y$;
- $l = 1$ și (x,y) este un arc al grafului;
- $l > 1$ și există drumurile $x..z$ de lungime $\lfloor l/2 \rfloor$ și $z..y$ de lungime $\lceil l/2 \rceil$.

Spațiul consumat de algoritmul **drum** este $O(\lg(n)^2)$. Într-adevăr, lungimea maximă a lanțului de apelurilor recursive este limitată la $\lg(n)$, iar fiecare apel consumă spațiu pentru 6 variabile și spațiu pentru înregistrarea de activare a apelului. Deoarece se lucrează cu întregi de valoare cel mult n , spațiul consumat la apelul k este $k \cdot O(\lg(n))$ biți, astfel încât la apelul $\lg(n)$ se consumă $O(\lg(n))^2$ biți.

În partea **GAP_S** a algoritmului, datele G , i și j nu contribuie la complexitatea spațială, fiind *read-only*. Se adaugă doar spațiul consumat de variabilele n și $lung$ și de apelul **drum**(i,j,n), anume $O(\lg(n))$ biți. Spațiul folosit de întregul algoritm este $O(\lg(n) + \lg(n)^2) = O(\lg(n)^2)$. Deci $GAP \in SPACE(\lg(n)^2)$.

2. $GAP \in TIME(n^2)$. Să construim un algoritm care rezolvă GAP în timp $O(n^2)$.

Estimarea timpului cheltuit de algoritmul GAP_s folosește recurența $T(1) \leq 2n$, $T(1/2) + \Theta(n)$ pentru calculul timpului consumat de apelul $drum(i, j, 1)$, considerând că operația $(i, j) \in E(G)$ este în $\Theta(1)$. Soluția recurenței este $O(1 \cdot n^{1g(1)})$, astfel încât timpul consumat de GAP_s este $O(n^{1g(n)+2})$. Problema poate fi însă rezolvată mai rapid, parcurgând graful în adâncime.

```
GAP_T(G,i,j){ // G are nodurile V(G) și arcele E(G)
    // Există drum i..j în G?
    for-each(u∈V(G)) stare(u)=nevizitat;
    explorare(i);
    return stare(j)=vizitat ? 1 : 0;
}

explorare(u){
    stare(u)=vizitat;
    for-each((u,v)∈E(G))
        if(stare(v)=nevizitat) explorare(v);
}
```

Algoritmul GAP_T parcurge graful în adâncime, începând din nodul i . Nodurile vizitate sunt marcate, evitându-se ciclurile. Rezultatul este decis de marca nodului j .

Lanțul apelurilor recursive ale algoritmului $explorare$ este $O(n)$, iar fiecare apel consumă un spațiu de memorie $O(1g(n))$. Spațiul total consumat de algoritm este $O(n \cdot 1g(n))$, mai mare decât cel al algoritmului GAP_s . În schimb, timpul consumat este $O(n^2)$. Deci $GAP \in TIME(n^2)$ și, totodată, $GAP \in PTIME$. ■

3.4.2 Clase de probleme rezolvate nedeterminist

Fie N_Alg un algoritm nedeterminist de decizie. Numim spațiul de stare al algoritmului pentru datele $i \in I$, un graf orientat $G(i) = (V, E, s_0, s_f)$, unde nodurile v reprezintă stări ale algoritmului, iar arcele e desemnează tranziții între stări. O stare este asociată unui moment t al execuției algoritmului și corespunde valorilor variabilelor algoritmului – inclusiv cele anonime, așa cum sunt cele ce desemnează eventualele înregistrări de activare ale algoritmului – la momentul t , precum și instrucțiunii executate de algoritm la momentul t .

O tranziție din starea u în starea v este guvernată de instrucțiunea executată în starea u . Nodul rădăcină s_0 corespunde stării inițiale a execuției algoritmului, iar nodul destinație s_f desemnează starea corespunzătoare terminării cu succes a algoritmului. Starea s_f colectează arcele de la nodurile *success* $\{s_{t_1}, s_{t_2}, \dots, s_{t_r}\}$ ale fiecărei căi corespunzătoare execuției $N_Alg(i)$. Aceste noduri corespund situației celei mai defavorabile pentru fiecare asemenea cale. Notăm:

- $c\ddot{a}i(N_Alg, i)$ mulțimea drumurilor simple $s_0 \dots s_f$ din $G(i)$.
- $st\ddot{a}ri(d)$ stările (nodurile) de pe o cale $d \in c\ddot{a}i(N_Alg, i)$. Fiecare stare s din $st\ddot{a}ri(d)$ este asociată unui moment de timp t al execuției seriale a căii d și desemnează mulțimea variabilelor existente la momentul t al acestei execuții precum și instrucțiunea curent executată.
- $timp_cale(d)$ timpul total necesar execuției seriale a tuturor prelucrărilor din calea d (a tranzițiilor între stările din $st\ddot{a}ri(d)$). Prin convenție, timpul tranziției (s_{t_i}, s_f) , $i=1, r$, este 0.
- $spa\ddot{t}iu_stare(s)$ spațiul consumat de variabilele corespunzătoare stării s , excluzând datele nemodificabile ale algoritmului, în particular, datele de intrare i . Prin convenție, $spa\ddot{t}iu_stare(s_f)=0$.

Definiția 3.27 Fie N_Alg un algoritm nedeterminist care rezolvă o problemă de decizie $Q: I \rightarrow \{0, 1\}$. Timpul și spațiul de memorie consumate de algoritm pentru datele $i \in I$ astfel încât $Q(i)=1$ (problema are soluție, iar $c\ddot{a}i(N_Alg, i) \neq \emptyset$) sunt:

$$N_timp(N_Alg, i) = \min \{ \forall d \in c\ddot{a}i(N_Alg, i) \bullet timp_cale(d) \}$$

$$N_spa\ddot{t}iu(N_Alg, i) = \min \{ \forall d \in c\ddot{a}i(N_Alg, i) \bullet spa\ddot{t}iu_cale(d) \}, \text{ unde } spa\ddot{t}iu_cale(d) = \max \{ \forall s \in st\ddot{a}ri(d) \bullet spa\ddot{t}iu_stare(s) \}$$

Se remarcă perspectiva *angelică* a măsurării resurselor consumate de algoritm pentru datele i : timpul celei mai rapide căi $s_0 \dots s_f$ din $G(i)$ și spațiul minim dintre cele maxime consumate de căile $s_0 \dots s_f$, deși spațiul total poate fi mult mai mare. Timpul și spațiul angelic se sprijină pe interpretarea: algoritmul ghicește calea cea mai rapidă sau cu consum minim de spațiu către soluție.

Să notăm $N_timp(Alg, n)$ și $N_spa\ddot{t}iu(Alg, n)$, complexitățile angelice ale algoritmului Alg din punctul de vedere al timpului și spațiului consumat în raport cu dimensiunea n a datelor algoritmului. Spunem că Alg are complexitatea angelică temporală/spațială limitată în raport cu o funcție $f: N \rightarrow R_+$ și scriem $N_timp(Alg, n) = O(f(n))$, respectiv $N_spa\ddot{t}iu(Alg, n) = O(f(n))$, dacă:

$$\forall i \in I \mid Q(i)=1 \wedge \dim(i)=n \bullet N_timp(Alg, i) = O(f(n))$$

$$\forall i \in I \mid Q(i)=1 \wedge \dim(i)=n \bullet N_spa\ddot{t}iu(Alg, i) = O(f(n))$$

Măsurile angelice $N_timp(Alg, n)$ și $N_spa\ddot{t}iu(Alg, n)$ ale timpului și spațiului consumate de un algoritm nedeterminist sunt suficiente pentru a caracteriza performanța algoritmului în orice situație. Într-adevăr, dacă pentru orice date i , astfel încât $Q(i)=1$ și $\dim(i)=n$, timpul sau spațiul rezolvării este limitat în raport cu $f(n)$, deci $N_timp(N_Alg, i) \leq k f(n)$, $n \geq n_0$, sau $N_spa\ddot{t}iu(N_Alg, i) \leq k' f(n)$, $n \geq n'_0$, unde n_0 , n'_0 , k și k' sunt constante, atunci depășirea timpului $k f(n)$ sau spațiului $k' f(n)$ oprește execuția algoritmului pentru orice date cu dimensiunea n , rezultatul algoritmului fiind 0, așa cum se sugerează în figura 3.14.

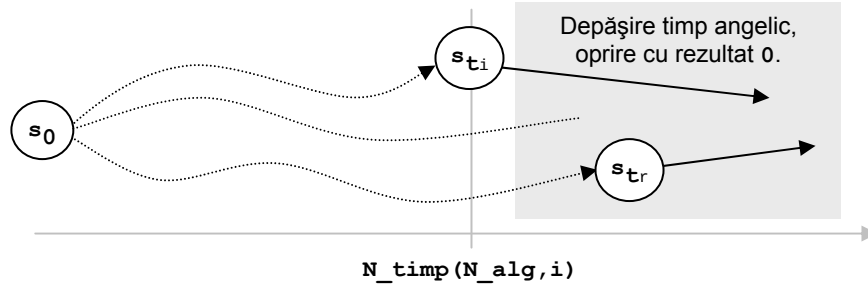


Figura 3.14 Spațiul stărilor execuției $N_Alg(i)$ pentru rezultat 0

Ca și în cazul algoritmilor determinați putem să definim clase de algoritmi nedeterminați cu complexitate impusă. Fie $Q: I \rightarrow \{0, 1\}$ o problemă de decizie, rezolvabilă prin mulțimea algoritmilor nedeterminați N_Alg_Q , și $f: N \rightarrow \mathbb{R}_+$ o funcție totală peste N . Notăm:

$$N_Alg_Q^T(f) = \{Alg \in N_Alg_Q \mid N_timp(Alg, n) = O(f(n))\},$$

$$N_Alg_Q^S(f) = \{Alg \in N_Alg_Q \mid N_spațiu(Alg, n) = O(f(n))\},$$

unde $n = \dim(i)$ este dimensiunea datelor i . Mulțimea $N_Alg_Q^T(f)$ conține acei algoritmi nedeterminați care rezolvă problema Q în $O(f(n))$ unități de timp, iar $N_Alg_Q^S(f)$ conține acei algoritmi nedeterminați care rezolvă problema Q folosind $O(f(n))$ unități de spațiu de memorie (biți).

Definiția 3.28 Fie $f: N \rightarrow \mathbb{R}_+$ o funcție totală peste N . Clasele de probleme rezolvabile prin algoritmi nedeterminați cu limitare f , temporală și, respectiv, spațială sunt:

$$NTIME(f) =_{\text{def}} \{Q \mid N_Alg_Q^T(f) \neq \emptyset\}$$

$$NSPACE(f) =_{\text{def}} \{Q \mid N_Alg_Q^S(f) \neq \emptyset\}$$

În particular,

$$NP = NPTIME =_{\text{def}} \bigcup_{k \geq 0} NTIME(\lambda n. n^k)$$

$$NPSPACE =_{\text{def}} \bigcup_{k \geq 0} NSPACE(\lambda n. n^k)$$

$$NLOGSPACE =_{\text{def}} NSPACE(\lambda n. \lg(n)),$$

unde $\lambda n. R(n)$ desemnează o funcție cu parametrul n și rezultatul $R(n)$. Ca exemplu, să reconsiderăm problema GAP , a accesibilității într-un graf, de data aceasta rezolvată folosind un algoritm nedeterminist.

Propoziția 3.12 $GAP \in NLOGSPACE$ și $GAP \in NPTIME$.

Construim algoritmul nedeterminist N_GAP de mai jos și arătăm că are complexitate spațială $O(\lg(n))$ și complexitate temporală $O(n)$.

```

N_GAP(G,i,j) { // G are nodurile V(G) și arcele E(G)
               // Există drum i..j în G?
    n = card(V(G));
    u = i;
    lung = 0;

    while(u ≠ j) {
        v = choice(V(G));
        lung++; // lungime drum (număr arce)
        if((u,v) ∉ E(G) ∨ lung ≥ n) fail;
        u = v;
    }
    success;
}

```

1. $GAP \in NLOGSPACE$. Fiecare stare a algoritmului conține doar patru variabile, anume n , $lung$, u și v , care contribuie la consumul de spațiu al algoritmului (variabilele G , i și j sunt *read-only*). Într-adevăr, prin efectul instrucțiunii *choice*, sunt construite noi copii ale algoritmului, incluzând variabilele folosite. Astfel, pot exista mai multe copii ale algoritmului funcționând simultan, dar fiecare copie conține doar patru variabile care consumă spațiu. Pentru că numerele cu care se lucrează au valoare cel mult n , spațiul de memorie folosit este $O(\lg(n))$.

2. $GAP \in NPTIME$. Pentru că un drum în spațiul stărilor algoritmului corespunde unui drum în graful G , iar lungimea drumurilor explorate în G este cel mult n , algoritmul se oprește în $O(n)$. Cu importanță marginală, putem observa că N_GAP se termină imediat ce drumul cel mai scurt $i..j$ din G este găsit. Un astfel de drum nu poate conține cicluri, deci este drum simplu. ■

3.4.3 Relații între clase de probleme

Evităm relațiile banale de tipul $f(n) = O(g(n)) \Rightarrow TIME(f) \subseteq TIME(g)$ și discutăm despre relații între clase cu natură diferită, anume relații timp-spațiu și relații determinism-nedeterminism.

Teorema 3.12 $TIME(f) \subseteq NTIME(f)$
 $SPACE(f) \subseteq NSPACE(f)$

Un algoritm determinist, fie el Alg , poate fi considerat nedeterminist, având o singură cale în spațiul stărilor. Deci dacă Alg are complexitate $O(f(n))$ înseamnă că, implicit, există un algoritm nedeterminist cu aceeași complexitate. Prin urmare, $Alg_Q^T(f) \subseteq N_Alg_Q^T(f)$ și $Alg_Q^S(f) \subseteq N_Alg_Q^S(f)$ pentru orice problemă Q . ■

Corolarul 3.3 $\text{LOGSPACE} \subseteq \text{NLOGSPACE}$
 $\text{PTIME} \subseteq \text{NPTIME}$ (sau $P \subseteq NP$)
 $\text{PSPACE} \subseteq \text{NPSPACE}$

Incluziunile derivă din definițiile (3.26), (3.28) și din teorema (3.12). ■

Se demonstrează că:

$$\text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{PTIME} \subseteq \text{NPTIME} \subseteq \text{PSPACE} = \text{NPSPACE}.$$

De asemenea, se poate generaliza conceptul de completitudine și duritate pentru diversele clase de complexitate. Aceste probleme depășesc obiectivele cursului.