Acest fişier conține o introducere în generalizarea clasificării problemelor conform unor clase de complexitate diferite de P şi NP. Fişierul 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** şi **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 şi, implicit, relații între complexitatea temporală şi cea spațială. Sunt considerate doar probleme de decizie, fără ca generalitatea să fie afectată. În ceea ce priveşte metrica folosită pentru a măsura dimensiunea datelor şi consumul de timp şi 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 şiruri de simboluri dintr-un alfabet Σ. În particular, putem alege $\Sigma=\{0,1\}$, fără a afecta generalitatea rezultatelor relativ la posibilitățile curente ale maşinilor 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 Q_1 este mai dificilă decât cea a unei alte probleme Q_2 doar pentru că dimensiunea datelor lui Q_1 este mai mare decât dimensiunea datelor lui Q_2. Din alt punct de vedere, datele de intrare ale unui algoritm pot fi stocate în "afara" algoritmului şi pot fi considerate nemodificabile (*read-only*).

3.4.1 Clase de probleme rezolvate determinist

Fie $\mathtt{Q:I}\rightarrow\{0,1\}$ o problemă de decizie, rezolvabilă prin mulțimea algoritmilor determinişti $\mathtt{Alg_Q}$, şi $\mathtt{f}:\mathrm{N}\rightarrow\mathtt{R}_+$ o funcție pe care o privim ca limită a complexității algoritmilor. Notăm:

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

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

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

- `stări(Alg,i)` stările din secvența execuției `Alg(i)`. Fiecare stare este asociată unui moment de timp `t` al execuției şi desemnează mulțimea variabilelor existente şi instrucțiunea executată la momentul `t`.

- `spațiu_stare(s)` spațiul consumat de variabilele din starea `s`, excluzând variabilele care desemnează datele de intrare ale algoritmului.

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

Mulțimea $\mathtt{Alg}_Q^T(\mathtt{f})$ conține acei algoritmi determinişti care rezolvă problema `Q` în $O(\mathtt{f(n)})$ unități de timp, iar $\mathtt{Alg}_Q^S(\mathtt{f})$ conține acei algoritmi determinişti care rezolvă problema `Q` folosind $O(\mathtt{f(n)})$ unități de spațiu de lucru.

Definiția 3.26 Fie $\mathtt{f}:\mathrm{N}\rightarrow\mathtt{R}_+$ o funcție totală peste N. Clasele de probleme rezolvabile prin algoritmi determinişti, cu limitare `f`, temporală şi, respectiv, spațială sunt:

$$\mathtt{TIME(f)} =_{\mathtt{def}} \{\mathtt{Q} \mid \mathtt{Alg}_Q^T(\mathtt{f}) \neq \varnothing\}$$

$$\mathtt{SPACE(f)} =_{\mathtt{def}} \{\mathtt{Q} \mid \mathtt{Alg}_Q^S(\mathtt{f}) \neq \varnothing\}$$

În particular,

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

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

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

unde notația $\lambda\mathtt{n.R(n)}$ desemnează o funcție cu parametrul `n` şi rezultatul `R(n)`.

Ca exemplu, să clasificăm problema **GAP** (**P**roblema **A**ccesibilităţii într-un **G**raf). Pentru un graf G cu n noduri să se decidă existenţa unui drum între două noduri date, i şi j, din G. Considerăm nodurile numerotate 1,2,...,n.

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

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

```
GAP_S(G,i,j) { // G are nodurile V(G) şi 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 şi n sunt vizibile în funcţia 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 şi 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 observaţie: un drum x..y de lungime l există dacă şi numai dacă este satisfăcută una din condiţiile:

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

Spaţiul consumat de algoritmul drum este $O(lg(n)^2)$. Într-adevăr, lungimea maximă a lanţului de apelurilor recursive este limitată la $lg(n)$, iar fiecare apel consumă spaţiu pentru 6 variabile şi spaţiu pentru înregistrarea de activare a apelului. Deoarece se lucrează cu întregi de valoare cel mult n, spaţiul consumat la apelul k este $k\,O(lg(n))$ biţi, astfel încât la apelul $lg(n)$ se consumă $O(lg(n))^2$ biţi.

În partea GAP_S a algoritmului, datele G, i şi j nu contribuie la complexitatea spaţială, fiind *read-only*. Se adaugă doar spaţiul consumat de variabilele n şi lung şi de apelul drum(i,j,n), anume $O(lg(n))$ biţi. Spaţiul 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 foloseşte recurenţa $T(l) \le 2n\ T(l/2)+\Theta(n)$ pentru calculul timpului consumat de apelul drum(i,j,l), considerând că operaţia $(i,j) \in E(G)$ este în $\Theta(1)$. Soluţia recurenţei este $O(l\ n^{lg(l)})$, astfel încât timpul consumat de GAP_s este $O(n^{lg(n)+2})$. Problema poate fi însă rezolvată mai rapid, parcurgând graful în adâncime.

```
GAP_T(G,i,j){ // G are nodurile V(G) şi 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.

Lanţul apelurilor recursive ale algoritmului explorare este $O(n)$, iar fiecare apel consumă un spaţiu de memorie $O(lg(n))$. Spaţiul total consumat de algoritm este $O(n\ lg(n))$, mai mare decât cel al algoritmului GAP_s. În schimb, timpul consumat este $O(n^2)$. Deci $GAP \in TIME(n^2)$ şi, totodată, $GAP \in PIME$. ■

3.4.2 Clase de probleme rezolvate nedeterminist

Fie $N\_Alg: I \to \{0,1\}$ un algoritm nedeterminist de decizie. Spaţiul de stare al algoritmului pentru datele $i \in I$ este un graf orientat $G(i)=(V,E,s_0,s_f)$, unde nodurile V reprezintă stări ale algoritmului, iar arcele E desemnează tranziţii între stări. O stare este asociată unui moment t al execuţiei N_Alg(i) şi corespunde valorilor variabilelor algoritmului la momentul t, precum şi instrucţiunii executate de algoritm la momentul t. O tranziţie din starea u în starea v este guvernată de instrucţiunea executată în starea u. Nodul rădăcină s_0 corespunde stării iniţiale a execuţiei algoritmului, iar nodul destinaţie s_f desemnează starea corespunzătoare terminării cu succes a algoritmului. Starea s_f colectează arcele de la nodurile success, fie ele $\{s_{t_1}, s_{t_2}, \ldots, s_{t_r}\}$, ale execuţiei N_Alg(i).

Notăm:

– căi(N_Alg,i) mulţimea drumurilor simple $s_0..s_f$ din G(i). Evident, $căi(N\_Alg,i) \neq \emptyset$ doar dacă $Q(i)=1$ (problema are soluţie pentru datele i).

– **stări(d)** stările (nodurile) de pe o cale **d∈căi(N_Alg,i)**. Fiecare stare **s** din **stări(d)** este asociată unui moment de timp **t** al execuției seriale a căii **d** şi desemnează mulțimea variabilelor existente şi instrucțiunea executată la momentul **t** al execuției căii **d**.

– **timp_cale(d)** timpul total necesar execuției seriale a tuturor prelucrărilor din calea **d** (a tranzițiilor între stările din **stări(d)**). Prin convenție, timpul tranziției (s_{t_i}, s_f), $i=1,r$, unde s_{t_i} este un nod **success**, este **0**.

– **spațiu_stare(s)** spațiul consumat de variabilele corespunzătoare stării **s**, excluzând datele de intrare **i**. In particular, **spațiu_stare**$(s_f)=0$.

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

N_timp(N_Alg,i) = min {∀d∈ căi(N_Alg,i) • timp_cale(d)}

N_spațiu(N_Alg,i) = min {∀d∈ căi(N_Alg,i) • spațiu_cale(d)}, unde
spațiu_cale(d) = max {∀s∈stări(d) • spațiu_stare(s)}

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

Să notăm **N_timp(N_Alg,n)** şi **N_spațiu(N_Alg,n)**, complexitatățile angelice ale algoritmului nedeterminist **N_Alg** din punctul de vedere al timpului şi spațiului consumat în raport cu dimensiunea **n** a datelor algoritmului. Spunem că **N_Alg** are complexitatea angelică temporală/spațială limitată în raport cu o funcție $f:N\to R_+$ şi scriem **N_timp(N_Alg,n)=O(f(n))**, respectiv **N_spațiu(N_Alg,n)=O(f(n))**, dacă:

∀i∈I | Q(i)=1 ∧ dim(i)=n • N_timp(N_Alg,i)= O(f(n))

∀i∈I | Q(i)=1 ∧ dim(i)=n • N_spațiu(N_Alg,i)= O(f(n))

Măsurile angelice **N_timp(N_Alg,n)** şi **N_spațiu(N_Alg,n)** ale timpului şi spațiului consumate de un algoritm nedeterminist sunt suficiente pentru a caracteriza performanța algoritmului în orice situație, inclusiv în atunci când **Q** nu are soluție. De exemplu, pentru orice date **i**, astfel încât problema **Q** are soluție, deci **Q(i)=1**, timpul rezolvării este limitat în raport cu **f(n)**, **n=dim(i)**, deci **N_timp(N_Alg,i)≤k f(n)**, $n\geq n_0$, unde n_0, şi **k** sunt constante, atunci depăşirea timpului **k f(n)** opreşte execuția algoritmului pentru orice date cu dimensiunea **n**, rezultatul algoritmului fiind **0**, aşa cum se sugerează în figura 3.14.

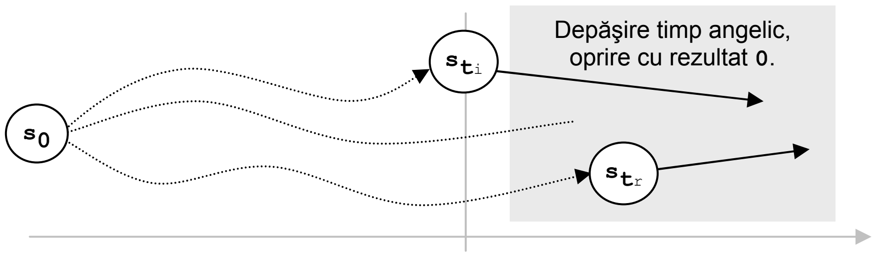

Figura 3.14 Spaţiul stărilor execuţiei `N_Alg(i)` pentru rezultat `0`

Ca şi în cazul algoritmilor determinişti putem să definim clase de algoritmi nedeterminişti cu complexitate impusă. Fie $Q:I \to \{0,1\}$ o problemă de decizie, rezolvabilă prin mulţimea algoritmilor nedeterminişti N_Alg_Q, şi $f:N \to R_+$ o funcţie totală peste N. Notăm:

$$N\_Alg_Q^T(f) = \{Alg \in N\_Alg_Q \mid N\_timp(Alg,n) = O(f(n))\},$$

$$N\_Alg_Q^S(f) = \{Alg \in N\_Alg_Q \mid N\_spaţiu(Alg,n) = O(f(n))\},$$

unde $n=dim(i)$ este dimensiunea datelor i. Mulţimea $N\_Alg_Q^T(f)$ conţine acei algoritmi nedeterminişti care rezolvă problema Q în $O(f(n))$ unităţi de timp, iar $N\_Alg_Q^S(f)$ conţine acei algoritmi nedeterminişti care rezolvă problema Q folosind $O(f(n))$ unităţi de spaţiu de memorie (biţi).

Definiţia 3.28 Fie $f:N \to R_+$ o funcţie totală peste N. Clasele de probleme rezolvabile prin algoritmi nedeterminişti cu limitare f, temporală şi, respectiv, spaţială sunt:

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

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

În particular,

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

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

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

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

Propoziția 3.12 **GAP** $\in$ **NLOGSPACE** şi **GAP** $\in$ **NPTIME**.

Construim algoritmul nedeterminist **N_GAP** de mai jos şi arătăm că are complexitate spațială O(lg(n)) şi complexitate temporală O(n).

```
N_GAP(G,i,j) { // G are nodurile V(G) şi 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∈NLOGSPACE**. Fiecare stare a algoritmului conține doar patru variabile, anume **n**, **lung**, **u** şi **v**, care contribuie la consumul de spațiu al algoritmului (variabilele **G**, **i** şi **j** reprezintă date de intare ale algoritmului). 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∈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 opreşte 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 $f(n)=O(g(n)) \Rightarrow TIME(f) \subseteq TIME(g)$ şi discutăm despre relații între clase cu natură diferită, anume relații timp-spațiu şi 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 aceeaşi complexitate. Prin urmare, $Alg_Q^T(f) \subseteq N\_Alg_Q^T(f)$ şi $Alg_Q^S(f) \subseteq N\_Alg_Q^S(f)$ pentru orice problemă **Q**. ■

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

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

Se demonstrează că:

LOGSPACE $\subseteq$ **NLOGSPACE** $\subseteq$ **PTIME** $\subseteq$ **NPTIME** $\subseteq$ **PSPACE** = **NPSPACE**.

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