Structuri de date II

Grafuri

Un graf este o pereche notată $G =< V, E >$, unde V este o mulţime de vârfuri, iar $E \subseteq V \times V$ este o mulţime de muchii. O muchie de la vârful *a* la vârful *b* este notată cu perechea ordonată (a, b).

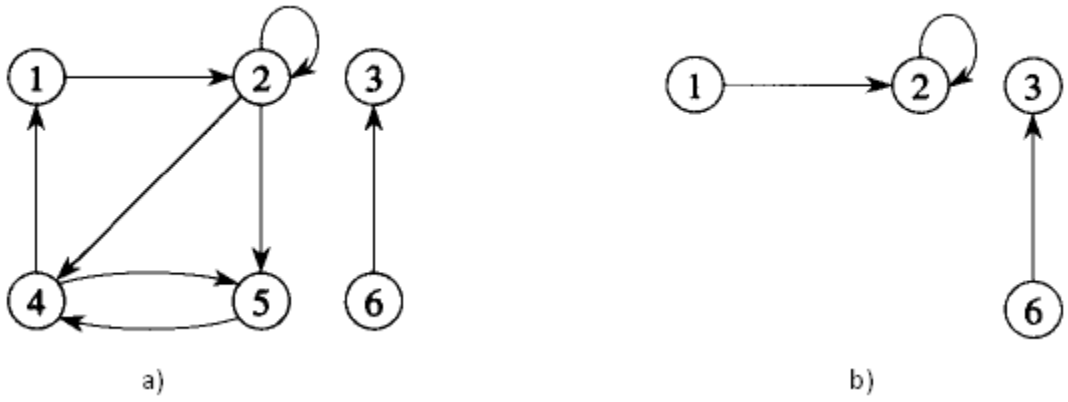

Figura 1. Exemplu de graf orientat. a) Un graf orientat $G = (V, E), cu\ V = \{1,2,3,4,6,7\}$ şi $E = \{(1,2), (2,2), (2,4), (2,5), (4,1), (4,5), (5,4), (6,3)\}$. $(2,2)$ este o buclă ciclică. b) un subgraf al celui din a) format din nodurile $\{1,2,3,6\}$.

Un graf orientat sau un digraf, este o pereche notată tot $G =< V, E >$ în care fiecare muchie este o perechea notată $\{u, v\}$ unde arcul format între nodurile u, v pleacă din u şi ajunge în v. În acest tip de graf, este permisă muchia ce are acelaşi nod ca destinaţie şi sursă. În figura de mai sus este prezentat un graf neorientat a) şi un subgraf al acestuia b).

Un subgraf este un graf $G =< V', E' >$, unde $V' \subseteq V$, iar E' este o submulţime a lui E, adică a muchiilor ce unesc nodurile din V'.

Un graf parţial, este un graf $G =< V, E'' >$, în care $E'' \subseteq E$.

Într-un graf neorientat $G =< V, E >$, mulţimea muchiilor E, constă din perechi de vârfuri-noduri, în care ordinea dispunerilor nodurilor în pereche nu contează. O muchie va fi formată din mulţimea $\{u, v\}$ unde $u, v \in V$ şi $u \neq v$. Prin convenţie, vom folosi notaţia (u, v) pentru a delimita o muchie dintr-un graf neorientat. În figura de mai jos avem o reprezentare a acestor tipuri de grafuri.

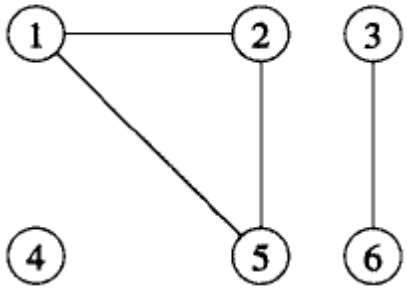

Figura 2. Un graf neorientat în care $V = \{1,2,3,4,5,6\}$ şi $E = \{(1,2), (1,5), (2,5), (3,6)\}$. Nodul 4 este izolat.

În cele ce urmează presupunem că *a* şi *b*, sunt două vârfuri diferite. Două vârfuri unite printr-o muchie se numesc *adiacente*. Un drum este o succesiune de muchii de forma:

$$(a_1, a_2), (a_2, a_3), \ldots., (a_{n-1}, a_n)$$

sau de forma

$$\{a_1, a_2\}, \{a_2, a_3\}, \ldots., \{a_{n-1}, a_n\}$$

după cum graful este orientat sau neorientat. În figura de mai jos este trasat cu roșu, un drum intre nodul 1 şi 4.

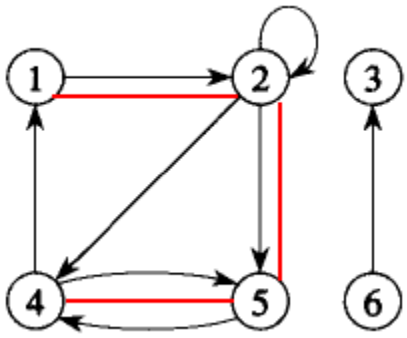

Figura 3. Un drum între două noduri $D = (1,2), (2,5), (5,4)$

Lungimea drumului este egală cu numărul muchiilor care îl constituie.

Un drum simplu este un drum în care nici un vârf se repetă .

Un ciclu este un drum care este simplu, cu excepţia primului şi a ultimului vârf, care şi coincid.

Un *graf aciclic* este un graf fără cicluri.

Un graf neorientat este *conex*, dacă între oricare două vârfuri există un drum. Pentru grafuri orientate, această noţiune este întărită: un graf orientat este tare conex, dacă între oricare două vârfuri *i* şi *j* există un drum de la *i* la *j* şi un de la *j* la *i*.

Iată mai jos un exemplu de graf neorientat conex:

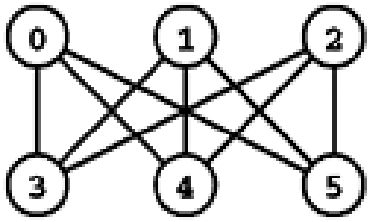

Figura 4. Graf neorientat conex

În cazul unui graf neconex, se pune problema determinării componentelor sale conexe. O *componentă conexă*, este un subgraf conex minimal, adică un subgraf conex în care nici un vârf din subgraf nu este unit cu unul din afară printr-o muchie a grafului iniţial. Împărţirea unui graf

$G =< V, E >$ în componentele sale conexe determină o partiţie a lui V şi una a lui E. Mai jos avem o reprezentare a unor componente conexe:

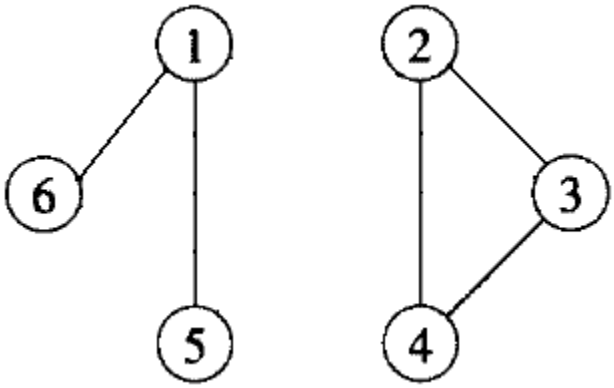

Figura 5. Componente conexe ale unui graf formate din nodurile: $\{1,5,6\}$ şi $\{2,3,4\}$

Vârfurilor unui graf neorientat li se pot atașa informații numite uneori *valori*, iar muchiilor li se pot atașa informații numite uneori *lungimi* sau *costuri*.

Un arbore este un graf neorientat aciclic conex. Altfel spus, un arbore este un graf neorientat în care există exact un drum intre oricare două vârfuri. Un graf parțial care este arbore se numește arbore parțial. Un graf aciclic, neorientat se numește o *pădure*.

Reprezentarea grafurilor

Există patru moduri mai des întâlnite de reprezentare a unui graf.

Matrice de adiacentă

Se numește matrice de adiacență *A*, o matrice de $|V| \times |V|$ în care $A[i,j] = true$ dacă vârfurile i și j sunt adiacente, iar $A[i,j] = false$ în caz contrar. O variantă alternativă ar fi să atribuim lui $A[i,j]$ valoarea lungimii muchiei dintre vârfurile i și j, considerând că $A[i,j] = +\infty$ atunci când cele două vârfuri nu sunt adiacente. Această reprezentare are un neajuns: dacă dorim să aflăm toate vârfurile adiacente unui vârf dat, trebuie să analizăm o întreagă linie din matrice. Aceasta necesită *n* operații (unde *n* este numărul de vârfuri din graf), independent de numărul de muchii care conectează vârful respectiv.

Matrice de incidență

Se numește matrice de incidență, o matrice de $|V| \times |E|$ în care $A[i,j]$ este numărul de câte ori nodul v_i întâlnește muchia e_j.

Exemplu: dat fiind graful din figura 6. Mai jos avem matricele de incidență respectiv de adiacență:

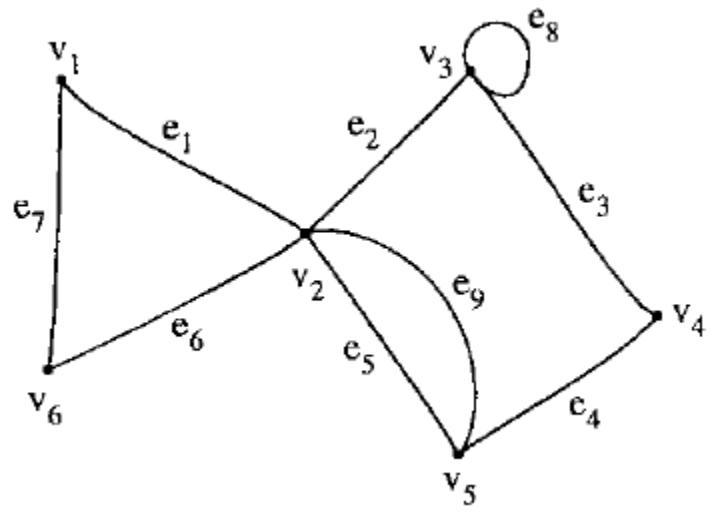

Figura 6. Un graf cu 6 noduri și 9 muchii

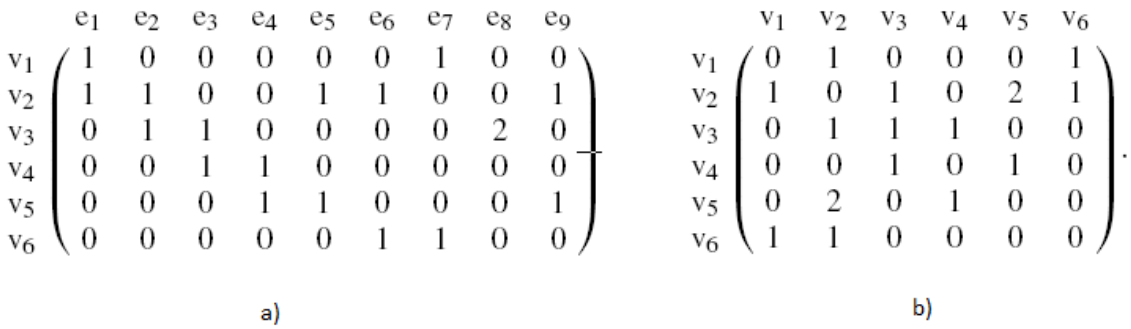

Figura 7. a) Matricea de incidență a grafului din figura 6. b) matricea de adiacență

Liste de adiacență

Prin atașarea fiecărui vârf *i* a listei de vârfuri adiacente lui (pentru grafuri orientate, este necesar ca muchia să plece din *i*) se obține o listă pentru fiecare nod. Într-un graf cu *m* muchii, suma lungimilor listelor de adiacență este *2m*, dacă graful este neorientat, respectiv *m* dacă este orientat. Dacă numărul muchiilor în graf este mic, această reprezentarea este preferabilă d.p.d.v.d. al memoriei necesare. Este posibil să examinăm toți vecinii unui vârf în mai puțin de *n* pași. Pe de altă parte, pentru a determina dacă două vârfuri *i* și *j* sunt adiacente, trebuie să analizăm lista de adiacență a lui *i*, ceea ce diminuează din eficiența acestei reprezentări. În figura de mai jos se găsesc listele de adiacență ale nodurilor din graful de mai sus.

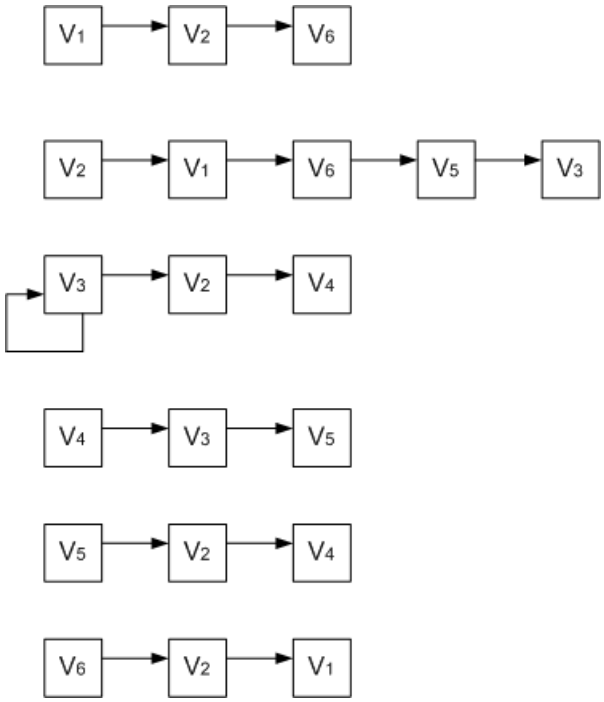

Figura 8. Listele de adiacență pentru graful din figura 6.

Listă de muchii

Această reprezentare este eficientă atunci când avem de examinat toate muchiile grafului. Practic se vor expune muchiile ca perechi de noduri.

Cum implementăm concret un graf? Putem răspunde în mai multe moduri la această întrebare, dar cel mai simplu este folosind o clasă în care nodurile sunt numerotate (în exemplul de mai sus de la 1 la 6). În aceste condiții o muchie va fi dată de cele două noduri pe care le unește:

```
class Edge
  {
        int v, w;
        Edge(int v, int w)
        {
                this.v = v;
                this.w = w;
        }
  }
```

Arbori liberi

Un arbore liber este un graf conex, aciclic, neorientat. Adjectivul *liber* este deseori omis. Dacă un graf neorientat este aciclic dar posibil neconex, atunci el se numește o pădure. Figura de mai jos prezintă un arbore, o pădure și un graf care nu este nici una nici alta, deoarece conține un ciclu.

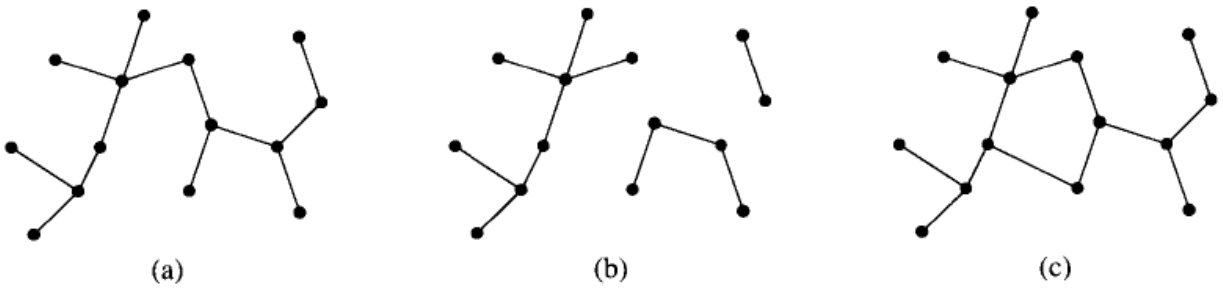

Figura 9. a) un arbore liber b) o pădure c) un graf care conține un ciclu nefiind nici arbore nici pădure

Proprietățile unui arbore

Fie $G =< V, E >$ un graf neorientat. Următoarele afirmații sunt echivalente

1. G este un arbore liber
2. Orice două noduri din G sunt conectate printr-un drum unic
3. G este conex, dar dacă orice muchie este ștearsă din E, atunci graful va fi neconex.
4. G este conex, și $|E| = |V| - 1$

5. G este aciclic și $|E| = |V| - 1$
6. G este aciclic, dar dacă o singură muchie este adăugată în E, graful rezultat va conține un ciclu.

Demonstrațiile afirmațiilor se găsesc mai jos:

$(1) \Rightarrow (2)$ Din moment ce un arbore este conex, orice două noduri din G sunt conectate printr-un drum simplu. Fie u, v cele două noduri conectate, și p_1, p_2 drumurile care le unesc după cum apare în figura de mai jos. Fie w nodul din care cele două drumuri se despart, și z nodul în care drumurile converg. Fie p' sub-drumul din drumul de la u la v, iar p'' sub-drumul din drumul de la v la u. Drumurile p' și p'' nu au în comun nici un nod, afară de capetele lor. De aceea, drumul obținut concatenând p'și inversul lui p'' este un ciclu. Aceasta este o contradicție în definiția arborelui.

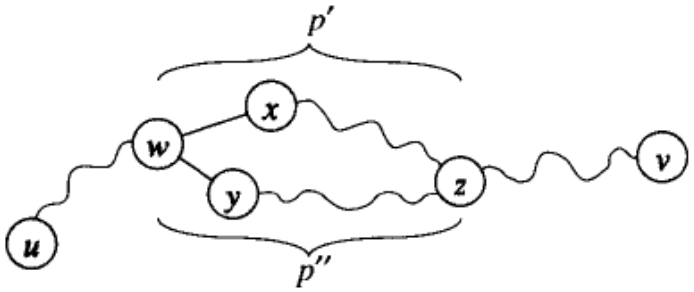

Figura 10. Demonstrația teoremei 1.

$(2) \Rightarrow (3)$ Dacă oricare două noduri din G sunt conectate printr-un drum simplu unic, atunci G este conex. Fie (u, v) o muchie oarecare din E. Această muchie este un drum de la u la v deci trebuie să fie calea unică de la u la v. Dacă ștergem (u, v) din G nu mai există cale între cele două noduri, deci G devine neconex.

$(3) \Rightarrow (4)$ Presupunând că graful este conex, și având relația $|E| \geq |V| - 1$ valabilă pentru orice graf conex vom dovedi că $|E| \leq |V| - 1$ prin inducție. Un graf conex cu $n = 1$ sau $n = 2$ noduri are $n - 1$ muchii. Dacă G are $n \geq 3$ noduri și toate grafurile ce satisfac relația 3. ce au mai puțin de n noduri vor satisface și $|E| \leq |V| - 1$. Ștergem o muchie arbitrar aleasă din G și separăm graful în două componente conexe. Fiecare componentă satisface 3. altfel G nu ar fi satisfăcut 3. De aceea numărul maxim de muchii în cele două componente va fi $|V| - 2$. Dacă adunăm la acest număr muchia ștearsă obținem că $|E| \leq |V| - 1$.

$(4) \Rightarrow (5)$ Presupunem că G este conex și că $|E| = |V| - 1$. Trebuie să arătăm că G nu conține cicluri. Presupunem că G are un ciclu ce conține *k* noduri $v_1, v_2, \ldots, v_k$. Fie $G_k =< V_k, E_k >$ subgraful lui G ce constă doar din ciclu. $|V_k| = |E_k| = k$. Dacă $k < |V|$ trebuie să existe un nod $v_{k+1} \in V - V_k$ care este adiacent cu unele noduri din V_k din moment ce G este conex. Definim $G_{k+1} =< V_{k+1}, E_{k+1} >$ ca

subgraful lui G cu $V_k \cup \{v_{k+1}\}$ și $E_{k+1} = E_k \cup \{v_i, v_{k+1}\}$. De reținut că $|V_{k+1}| = |E_{k+1}| = k + 1$. Dacă $k + 1 < n$, continuăm până când obținem $G_n =< V_n, E_n >$, unde $n = |V|$, $V_n = V$ și $|V_n| = |E_n| = |V|$.

Din moment ce G_n este un subgraf al lui G, avem $E_n \subseteq E$. și deci $|E| \geq |V|$ ceea ce contrazice presupunerea $|E| = |V| - 1$. Ca atare G este aciclic.

$(5) \Rightarrow (6)$ Presupunem că G este aciclic și că $|E| = |V| - 1$. Fie *k* numărul de componente conexe ale lui G. Fiecare componentă este un arbore, și din 5. numărul muchiilor din componentele conexe este $|V| - k$. Ca atare, trebuie ca $k = 1$. Din 1. și 2. orice două noduri sunt conectate printr-un drum unic. De aceea adăugarea unei muchii creează un ciclu.

$(6) \Rightarrow (1)$ presupunem că G este aciclic dar dacă orice muchie este adăugată la E, creează un ciclu. Trebuie să arătăm că G este conex. Fie u și v două noduri oarecare din G. Dacă u și v nu sunt deja adiacente, adăugarea unei noi muchii (u, v) creează un ciclu.

Arbori cu rădăcină

Fie un graf orientat G. Acesta se numește arbore cu rădăcina *r*, dacă există în G un vârf *r* din care se poate ajunge la oricare alt vârf printr-un drum unic.

Definiția este valabilă și pentru un graf neorientat, dar alegerea rădăcinii se va face în acest caz arbitrar. Cea mai intuitivă reprezentare a unui arbore cu rădăcină, este ca în figura 11: *a* este tatăl lui *b* si *c*. *b si c* sunt *frați* iar *b* este un *descendent* al lui *a*.

Un vârf *terminal* sau *frunză* este un nod fără descendenți. Vârfurile care nu sunt terminale se numesc *neterminale.*

Orice vârf al unui arbore cu rădăcină este rădăcina unui subarbore constând din vârful respectiv și toți descendenții săi.

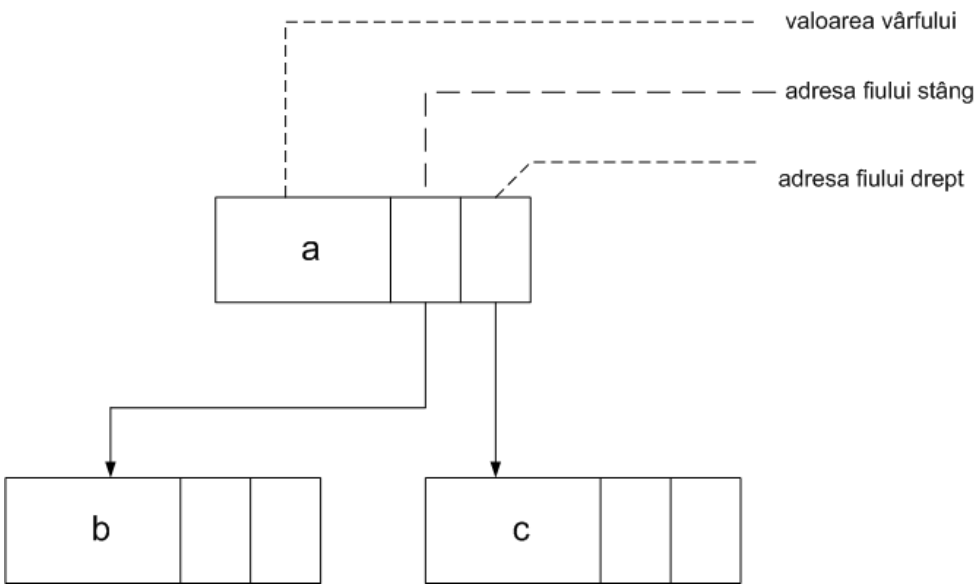

Figura 11. Reprezentarea arborilor folosind adrese

într-un arbore cu rădăcină vom adopta următoarele notații.

Adâncimea unui vârf este lungimea drumului dintre rădăcină și acest vârf.

Înălțimea unui vârf este lungimea celui mai lung drum dintre acest vârf și un vârf terminal.

Înălțimea arborelui este înălțimea rădăcinii.

Nivelul unui vârf este înălțimea arborelui minus adâncimea acelui vârf.

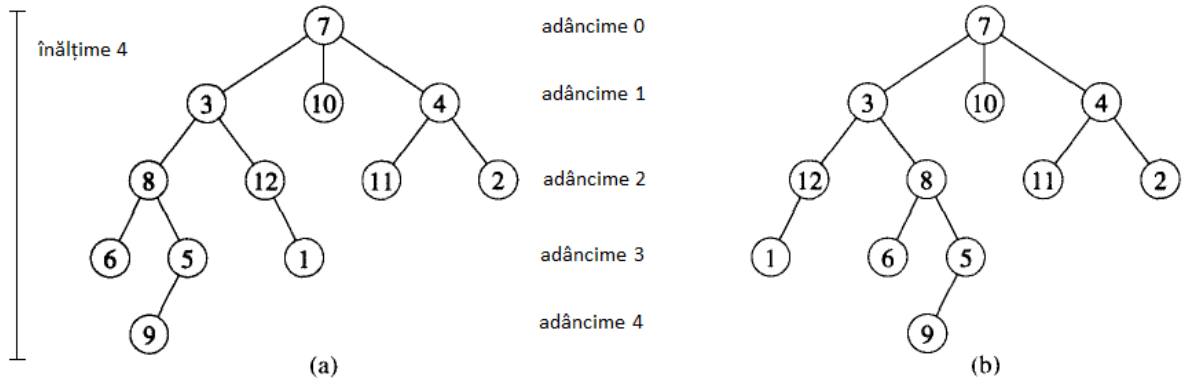

Figura 12. Arbori cu rădăcină. a) un arbore cu rădăcină cu înălțimea 4. Rădăcina 7 se află în vârful acestuia. Copii (nodurile cu adâncimea 1) sunt dedesubtul rădăcinii. Dacă arborele este ordonat, relația dintre copii și părinte contează, altfel arborele este neordonat. b) un al arbore neordonat, ordinea in subarborele cu rădăcina 3 este alta.

Arbori binari

Arborii binari sunt o structură compusă dintr-un set te noduri în care un nod fie:

- nu conține nici un nod copil, sau
- este compus din trei tipuri de noduri: *rădăcină*, ce conține un *subarbore* binar stâng, și un *subarbore* binar drept.

Arborele binar de primul tip, se numește null sau gol, și se mai notează ci NIL. Dacă subarborele stâng nu este gol, rădăcina se numește copilul stâng al rădăcinii întregului arbore. Același lucru este valabil și pentru subarborele drept. Dacă un subarbore este NIL, se spune că, copilul lipsește sau este absent.

Ceea ce este foarte important într-un arbore binar este că poziția unui copil, fie stânga șie dreapta contează, și de obicei se va decide pe baza unei comparații între valorile deținute în noduri.

În figura 13 avem reprezentarea unor arbori binari.

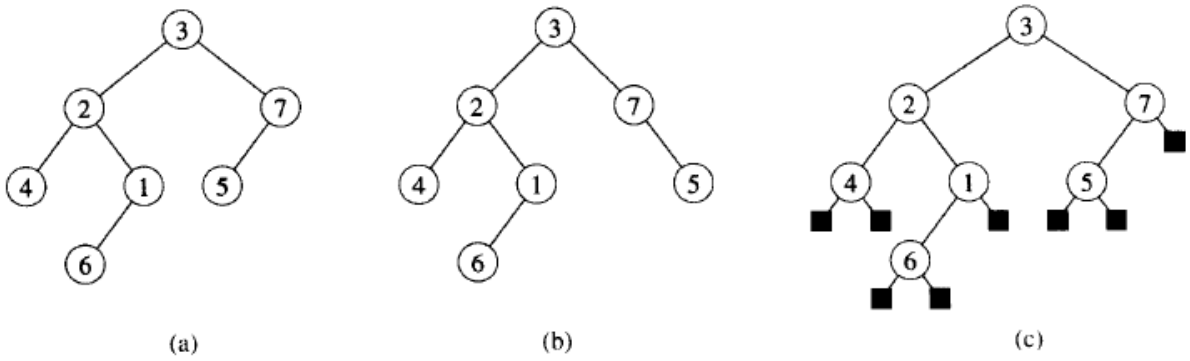

Figura 13. Arbori binari. a) un arbore binar trasat în mod normal. b) un arbore binar diferit de cel din a) deoarece nodul 7 are copilul 5 în subarborele drept de această dată. c) arborele binar din a) reprezentat ca arbor binar complet: fiecare nod intern are un grad 2. Frunzele sunt prezentate ca pătrate si au valoare NIL.

Într-un arbore binar, numărul maxim de vârfuri de adâncime *k* este 2^k.

Un arbore binar de înălțime *i* are cel mult $2^{i+1} - 1$ vârfuri, iar dacă are exact $2^{i+1} - 1$ vârfuri se numește arbore plin. Vârfurile se vor nota în ordinea adâncimii. Pentru aceeași adâncime, numerotarea se face de la stânga la dreapta.

Un arbore binar cu *n* vârfuri și de înălțime *i* este *complet*, dacă se obține din arborele binar plin, de înălțime i, prin eliminarea, dacă este cazul, a vârfurilor numerotate cu $n + 1, n + 2, \dots, 2^{i+1} - 1$.

Acest tip de arbore se poate reprezenta secvențial, folosind un tablou *T*, punând vârfurile de adâncime *k*, de la stânga la dreapta în pozițiile $T[2^k], T[2^k + 1], \dots, T[2^{k+1} - 1]$ (eventual cu excepția nivelului 0 care poate fi incomplet). În figura de mai jos avem numerotarea unui arbore binar.scu

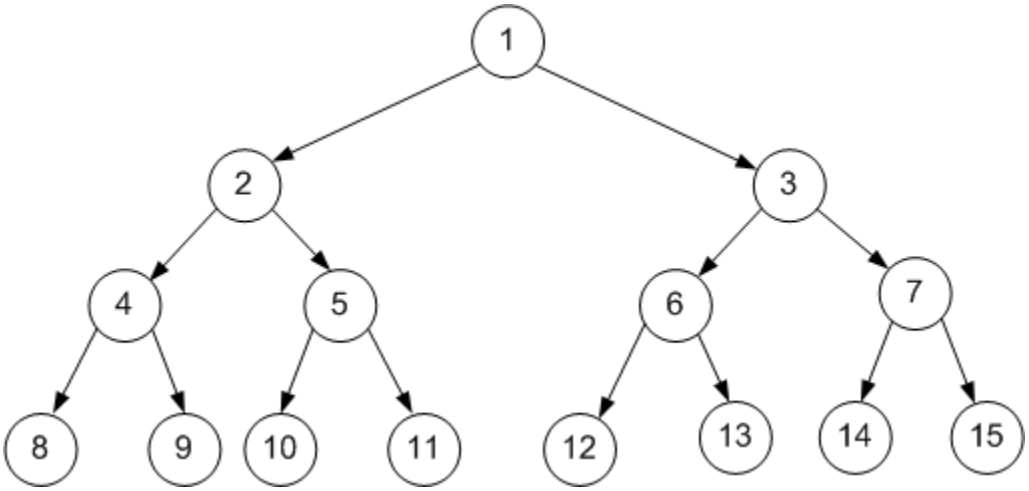

Figura 14 Numerotarea vârfurilor într-un arbore binar de înălțime 3.

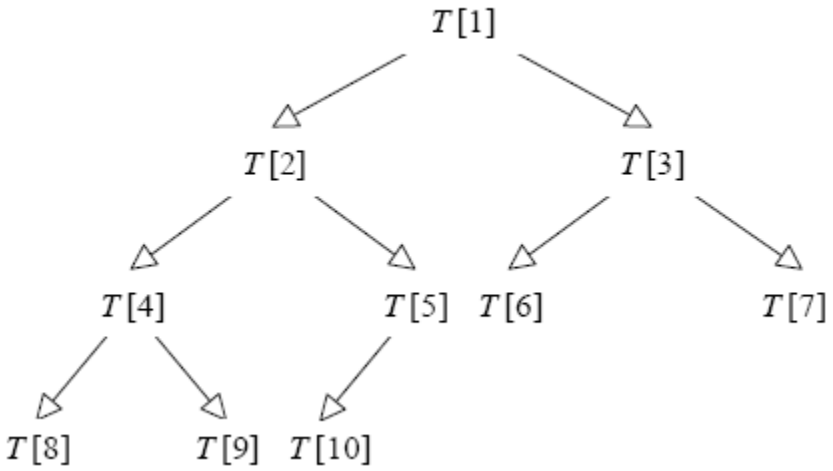

Figura 15. Un arbore binar obținut din cel de sus, prin eliminarea ultimilor 5 noduri. Fii lui $T[i]$ se vor afla, dacă ei există, în $T[2i]$ și $T[2i + 1]$

Există și alte metode de a reprezenta arbori binari. Vom analiza cum poate fi reprezentat un arbore folosind liste înlănțuite. Putem reprezenta un nod dintr-un arbore printr-un obiect. Ca și în cazul listelor, va exista un câmp *cheie* folosit pentru a stoca informația (legată de valoarea nodului). Celelalte câmpuri sunt referințe către alte noduri din arbore.

După cum urmează în figura 16 există *părinte* , *stânga* și *dreapta* pentru a stoca referințe la nodul părinte, respectiv nodurile copii din stânga și dreapta. Dacă $p[x] = NIL$ atunci x este rădăcină. se poate nota cu $root[T]$. Dacă $root[T] = NIL$ atunci arborele este gol. Dacă fiul stâng sau fiul drept nu există se va nota în figură cu "/".

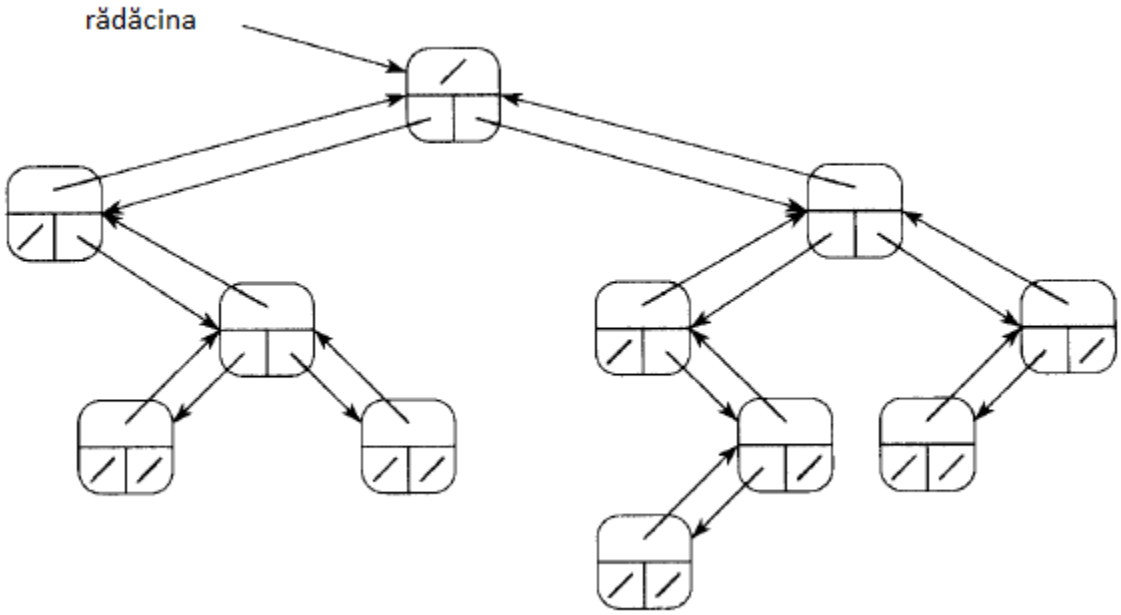

Figura 16. Reprezentarea unui arbore binar *T*. Fiecare nod *x* are referință $p[x]$ către părinte, $left[x]$ către copilul stânga, respectiv $right[x]$ către copilul dreapta. Câmpul *cheie* nu este afișat

În cele ce urmează vom face o scurtă incursiune în matematica elementară pentru a stabili câteva axiome de care ne vom folosi mai târziu.

În primul rând, numărul maxim de noduri $2^{i+1} - 1$, a fost calculat pornind de la ecuația:

$(a^n - b^n) = (a - b)(a^{n-1} + a^{n-2}b + a^{n-3}b^2 + \ \ldots + ab^{n-2} + b^{n-1})$

Pentru un număr real oarecare, partea fracționară se notează cu $\{x\}$ și este definită ca

$\{x\} = x - \lfloor x \rfloor$.

Pentru orice x $0 \leq \{x\} < 1$

Pentru un număr real oarecare *x* definim:

$\lfloor x \rfloor = max\{n | n \leq x, n \ este \ întreg\}$ și

$\lceil x \rceil = min\{n | n \geq x, n \ este \ întreg\}$

Se pot demonstra următoarele proprietăți

1. $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$ pentru orice *x* real
2. $\left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil = n$ pentru orice n întreg
3. $\left\lfloor \frac{\lceil \frac{n}{a} \rceil}{b} \right\rfloor = \left\lceil \frac{n}{ab} \right\rceil$ și $\left\lceil \frac{\lfloor \frac{n}{a} \rfloor}{b} \right\rceil = \left\lfloor \frac{n}{ab} \right\rfloor$ pentru orice *n*, *a*, *b* întregi ($a, b \neq 0$)
4. $\left\lfloor \frac{n}{m} \right\rfloor = \left\lceil \frac{n-m+1}{m} \right\rceil$ și $\left\lceil \frac{n}{m} \right\rceil = \left\lfloor \frac{n+m-1}{m} \right\rfloor$ pentru orice numere întregi pozitive n, m

Heap-uri

Un heap (în traducere aproximativă, “grămadă ordonată”), este un șir de obiecte care poate fi interpretat ca un arbore binar complet. Fiecărui nod din arbore îi corespunde un element din șir, element ce conține valoarea nodului. Arborele este complet, excepție făcând eventual, ultimul nivel, unde pot lipsi noduri terminale.

Proprietatea principală a elementelor este: valoare fiecărui vârf este mai mare sau egală cu cea a fiecărui fiu al său. În figura de mai jos avem un heap care poate fi prezentat prin următorul tablou:

10	7	9	4	7	5	2	2	1	6
$T[1]$	$T[2]$	$T[3]$	$T[4]$	$T[5]$	$T[6]$	$T[7]$	$T[8]$	$T[9]$	$T[10]$

Figura 17. Un heap implementat printr-un șir.

Caracteristica de bază a acestei structuri de dată este că modificarea valorii unui vârf se face foarte eficient, și se păstrează totodată proprietatea de heap. Dacă valoarea unui vârf crește, ai depășește valoarea tatălui este suficient să schimbăm între ele aceste valori, până când proprietatea de heap este îndeplinită.

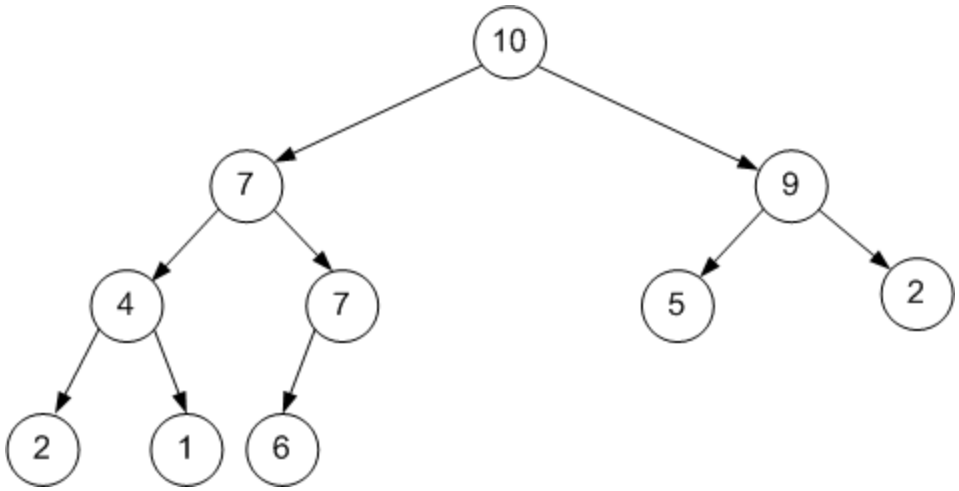

Figura 18. Un heap conform șirului din figura 17

Se spune că valoarea modificată a fost filtrată (*percolated*) către noua sa poziție. Dacă dimpotrivă valoarea vârfului scade, ai devine mai mică decât a unui fiu, este suficient să schimbăm valoarea cu cea mai mare valoare a fiilor, si apoi să continuăm procesul, până când proprietatea de heap este restabilită. Vom spune că valoarea modificată a fost cernută (*sifted down*) către noua sa poziție.

Procedurile ce urmează descriu sub formă de pseudocod, cele două operații:

$SIFT\_DOWN(T[1..n], i)$

1. $\triangleright$ *se cerne valoarea din* $T[i]$
2. $k \leftarrow i$
3. ***repeat***
4. $\quad j \leftarrow k$
5. $\quad \triangleright$ *găsește fiul cu valoarea cea mai mare*
6. $\quad$ ***if*** $2j \leq n$ ***and*** $T[2j] > T[k]$ ***then*** $k \leftarrow 2j$
7. $\quad$ ***if*** $2j < n$ ***and*** $T[2j+1] > T[k]$ ***then*** $k \leftarrow 2j+1$
8. $\quad$ ***interschimba*** $T[j]$ *și* $T[k]$
9. ***until*** $j = k$

$PERCOLATE(T[1..n], i)$

1. $\triangleright$ *se filtrează valoarea din* $T[i]$
2. $k \leftarrow i$
3. ***repeat***
4. $\quad j \leftarrow k$
5. $\quad$ ***if*** $j > 1$ ***and*** $T[j \ div \ 2] < T[k]$ ***then*** $k \leftarrow j$ ***div*** 2
6. $\quad$ ***interschimba*** $T[j]$ *și* $T[k]$
7. ***until*** $j = k$

Parametrul i al procedurilor reprezintă, poziția de început a modificărilor, practic poziția nodului vizat, în arbore.

După cum am precizat mai sus, pentru a păstra proprietatea de heap, la modificarea valorii unui nod, se va cerne în funcție de fosta valoare din nod (în susul arborelui sau in jos). Procedura care realizează aceasta se numește alter-heap:

$ALTER\_HEAP(T[1..n], i, v)$

1. $\triangleright\ T[1..n]\ este\ heap, iar\ 1 \leq i \leq n\ este\ poziția.$
2. $\triangleright\ La\ sfârșitul\ procedurii, proprietatea\ de\ heap\ este\ restabilită$
3. $x \leftarrow T[i]$
4. $T[i] \leftarrow v$
5. $\boldsymbol{if}\ v < x\ \boldsymbol{then}$
6. $\quad SIFT\_DOWN(T, i)$
7. $\boldsymbol{else}$
8. $\quad PERCOLATE(T, i)$

Parametrul v al acestei proceduri este noua valoare ce este atribuită nodului de pe poziția *i*.

Heap-ul este structura de date ideală pentru determinarea și extragerea maximului dintr-o mulțime, pentru inserarea unui vârf, sau modificarea valorii unui vârf. Sunt exact operațiile de care avem nevoie pentru a implementa o listă dinamică de priorități: valoarea unui vârf este prioritatea elementului corespunzător.Evenimentul cu cea mai mare prioritate va fi rădăcina iar aceste priorități se pot modifica dinamic. Procedurile care efectuează aceste operații sunt:

$FIND\_MAX(T[1..n])$

1. $\triangleright\ returnează\ elementul\ cel\ mai\ mare\ din\ heap$
2. $\boldsymbol{return}\ T[1]$

$DELETE\_MAX(T[1..n])$

1. $\triangleright\ șterge\ elementul\ cel\ mai\ mare\ din\ heap - ul\ T$
2. $T[1] \leftarrow T[n]$
3. $sift\_down(T[1..n-1], 1)$

$INSERT(T[1..n], v)$

1. $\triangleright\ inserează\ un\ element\ cu\ valoarea\ v\ \ in\ heap - ul\ T$
2. $T[n+1] \leftarrow v$
3. $\triangleright\ și\ restabilește\ apoi\ proprietatea\ de\ heap$
4. $sift\_down(T[1..n-1], 1)$

Există două moduri de a forma un heap, pornind de la un tablou neordonat $T[1..n]$. O soluție este de a porni cu un heap nou și să adăugăm rând pe rând, elementele în heap:

$SLOW - MAKE - HEAP(T[1..n])$

1. $\triangleright$ $formează\ în\ mod\ ineficient, din\ T\ un\ heap$
2. $\boldsymbol{for}\ i \leftarrow 2\ \boldsymbol{to}\ n\ \boldsymbol{do}\ percolate(T[1..i], i)$

Soluția nu este una eficientă, dar există o altă modalitate de a crea un heap, și anume în timp liniar.

$MAKE - HEAP(T[1..n])$

1. $\triangleright$ $formează\ din\ T\ un\ heap$
2. $\boldsymbol{for}\ i \leftarrow (n\ \boldsymbol{div}\ 2)\ \boldsymbol{downto}\ 1\ \boldsymbol{do}\ sift - down(T, i)$

Pe poziția $n\ \boldsymbol{div}\ 2$ se află tatăl lui n.

Fie tabloul neordonat :

1	6	9	2	7	5	2	7	4	10

Arborele ce corespunde acestui tablou este:

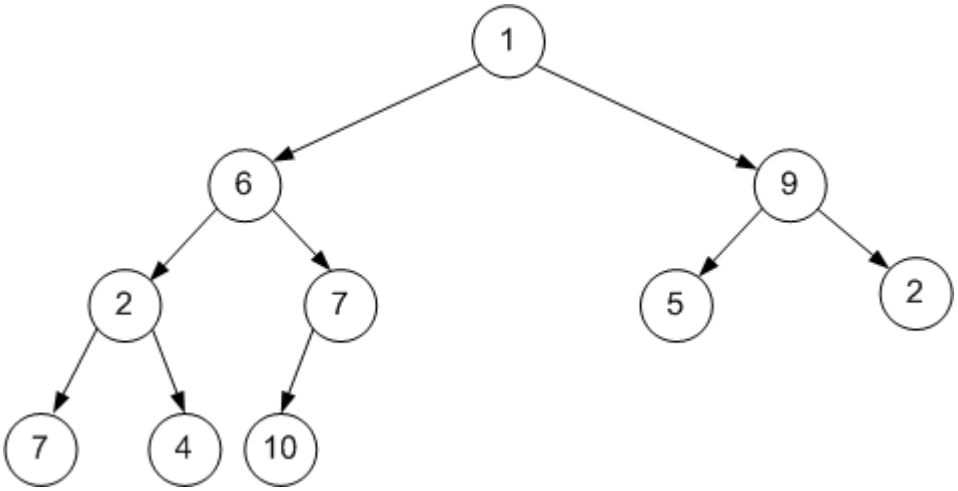

Figura 19. Tabloul prezentat ca arbore

Aplicând procedura de creare de heap, formăm subheap-uri din subarborii ce au rădăcina la nivelul 1, aplicând sift-down:

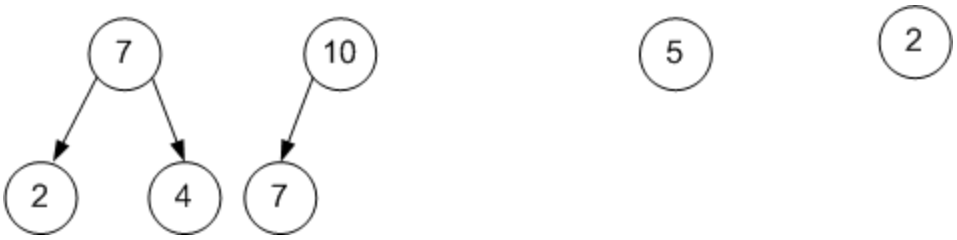

Figura 20. Subheap-uri după primul pas de sift_down

După acest pas tabloul T devine:

1	6	9	7	10	5	2	2	4	7

După aceea subarborii de la nivelul 2 sunt transformați astfel că:

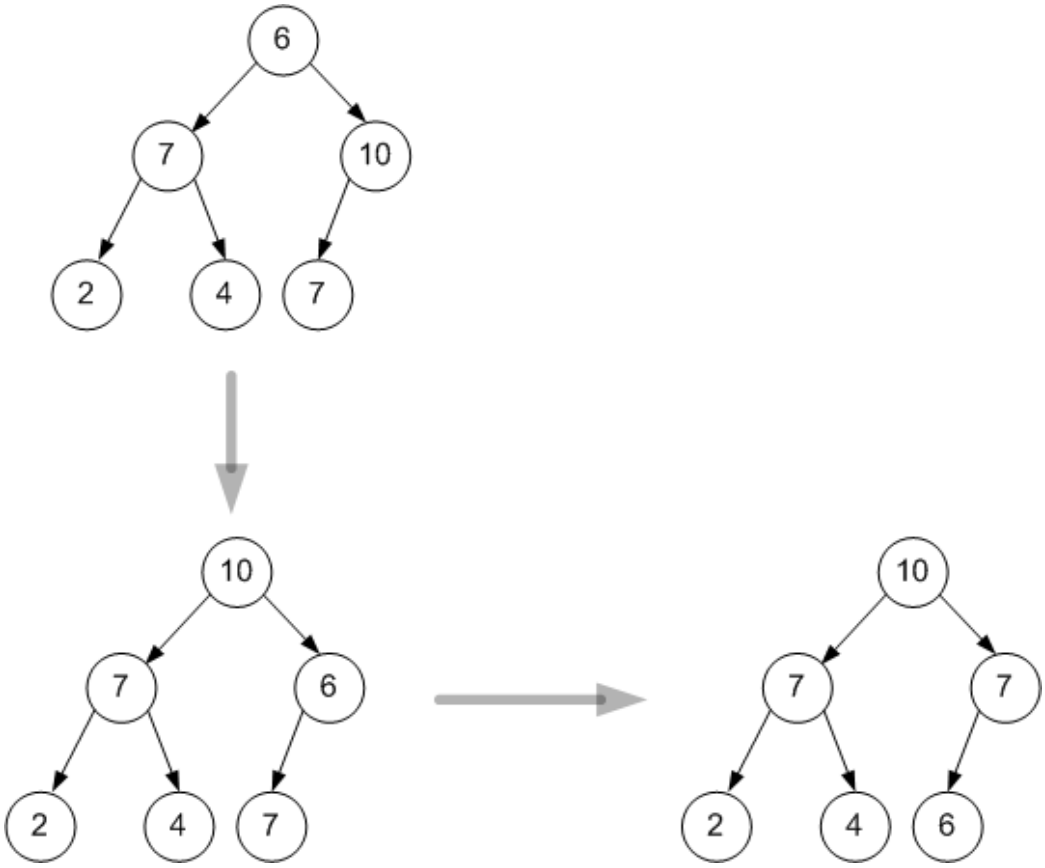

Figura 21. Transformarea pentru nivelul 2

Subarborele de nivel 2 este deja heap, iar tabloul arată astfel:

1	10	9	7	7	5	2	2	4	6

Ultimul pas va produce *interschimbarea* lui 1 cu 10 și apoi cernerea valorii 1 până când arborele va arăta ca în figura 18. Un *min_heap* este tot un heap ce are proprietatea de heap inversată: valoarea fiecărui vârf este mai mică sau egală cu valoarea fiecărui fiu al său. Evident, rădăcina va fi cel mai mic element ca valoare. Celelalte proceduri se modifică în mod corespunzător acestei condiții.

Iată mai jos, procedura de sortare a unui șir folosind conceptul de heap:

$HEAP\_SORT(T[1..n])$

1. $\triangleright$ $sortăm\ tabloul\ T$
2. $MAKE\_HEAP(T)$
3. $for\ i \leftarrow n\ downto\ 2\ do$
4. $interschimbă\ T[1]\ și\ T[i]$
5. $SIFT - DOWN(T[1..i-1], 1)$