Heap-uri BINOMIALE

Mihai Scorțaru

În cadrul acestui articol vom prezenta o nouă structură de date derivată din heap-urile clasice (binare). Principalul avanta al acestei structuri îl reprezintă posibilitatea de a uni două astfel de structuri într-un timp foarte scurt. Există mai multe alte avantaje pe care le veți putea descoperi în paginile acestui articol.

Un *heap binomial* este o colecție de *arbori binomiali*. Vom începe prezentarea *heap*-urilor binomiale cu descrierea arborilor binomiali; vom prezenta apoi *heap*-urile binomiale, precum și operațiile care pot fi efectuate asupra acestora.

Arbori binomiali

În cadrul acestei secțiuni vom defini arborii binomiali și vom prezenta ilustra conceptul de arbore binomial.

Definiție

Arborii binomiali sunt definiți recursiv astfel: arborele binomial de ordin 0 (B_0) constă într-un singur nod; arborele binomial de ordin k (B_k) este format din doi arbori binomiali de ordin k - 1 (B_{k-1}) care sunt legați: rădăcina unuia dintre cei doi arbori este fiul cel mai din stânga (sau cel mai din dreapta) a rădăcinii celuilalt arbore (vezi figura 1 pentru ilustrarea definiției).

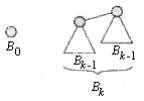

Figura 1: Definiția arborelui binomial

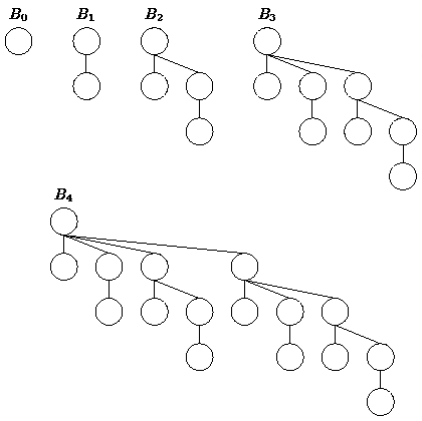

Figura 2: Arbori binomiali

În figura 2 sunt prezentați arborii binomiali B_0, B_1, B_2, B_3 și B_4.

Se poate spune că un arbore binomial de ordin k este format din rădăcină și k arbori binomial ale căror ordine sunt 0, 1, 2, ..., k - 1.

Proprietățile arborilor binomiali

Vom prezenta în continuare principalele proprietăți ale arborilor binomiali.

Vom începe cu o teoremă care indică numărul nodurilor unui arbore binomial.

Teoremă

Un arbore binomial de ordin k conține 2^k noduri.

Demonstrație

Deși intuitiv deducem destul de ușor că teorema este adevărată (pentru fiecare ordin numărul nodurilor se dublează față de ordinul imediat inferior), vom demonstra afirmație folosind metoda inducției matematice.

Vom nota numărul nodurilor unui arbore binomial de ordin k prin n_k și vom încerca să demonstrăm că avem întotdeauna $n_k = 2^k$.

Pentru k = 0, avem arborele binomial de ordin 0 care este format dintr-un singur nod, deci $n_0 = 1 = 2^0$. Așadar, ipoteza de inducție este adevărată.

Presupunem acum că pentru orice valoare $k < l$ arborele binomial de ordin k conține $n_k = 2^k$ noduri și vom încerca să demonstrăm că arborele binomial de ordin l conține $n_l = 2^l$ noduri.

Așa cum am arătat anterior un arbore binomial de ordin l este format din rădăcină și l arbori binomial ale căror ordine sunt 0, 1, 2, ..., l - 1.

Ca urmare numărul nodurilor arborelui binomial de ordine l este:

$$n_l = 1 + n_0 + n_1 + ... + n_{l-1}.$$

Conform ipotezei de inducție știm că pentru orice $k < l$ avem $n_k = 2^k$. Ca urmare, obținem:

$$n_l = 1 + 2^0 + 2^1 + ... + 2^{l-1}.$$

Folosind formula matematică $a^0 + a^1 + ... + a^n = (a^{n+1} - 1) / (a - 1)$, obținem:

$$n_l = 1 + (2^l - 1) / (2 - 1)$$
$$n_l = 1 + (2^l - 1)$$
$$n_l = 2^l$$

Așadar, am demonstrat că dimensiunile arborilor binomiali (numărul

total al nodurile) trebuie să fie puteri ale lui 2.

Mai mult, putem afirma că pentru orice putere a lui 2 dată, forma arborelui care are ca număr de noduri acea putere a lui 2 este unică.

În continuare vom prezenta o teoremă care indică înălțimea unui arbore binomial.

Teoremă

Înălțimea unui arbore binomial de ordin k este k.

Demonstrație

Din nou deducem intuitiv că pentru fiecare ordin înălțimea crește cu 1 față de ordinul imediat inferior.

Prezentăm în continuare demonstrația folosind metoda inducției matematice.

Vom nota înălțimea unui arbore binomial de ordin k prin h_k și vom încerca să demonstrăm că avem întotdeauna $h_k = k$.

Pentru $k = 0$, avem arborele binomial de ordin 0 care este format dintr-un singur nod, deci are înălțimea 0. Ca urmare, avem $h_0 = 0$, deci ipoteza de inducție este adevărată.

Presupunem acum că pentru orice valoare $k < l$ arborele binomial de ordin k are înălțimea $h_k = k$ și vom încerca să demonstrăm că arborele binomial de ordin l are înălțimea conține $h_l = l$.

Înălțimea arborelui binomial de ordin l este dată de înălțimea celui mai "înalt" dintre arborii binomiali care sunt fiii rădăcinii, la care se adaugă 1 (corespunzător rădăcinii).

Ca urmare înălțimea arborelui binomial de ordine l este:

$$h_l = 1 + \max(h_0, h_1, ..., h_{l-1}).$$

Conform ipotezei de inducție știm că pentru orice $k < l$ avem $h_k = k$. Ca urmare, obținem:

$$h_l = 1 + \max(1, 2, ..., l - 1).$$
$$h_l = 1 + (l - 1)$$
$$h_l = l$$

Următoarea teoremă indică numărul nodurilor aflate pe un anumit nivel a unui arbore binomial de ordin k.

Teoremă

Pe nivelul l al unui arbore binomial de ordin k se află C_k^l noduri.

Demonstrație

Această afirmație nu este la fel de ușor de intuit, dar poate și ea fi demonstată relativ ușor folosind metoda inducției matematice.

Vom nota numărul nodurilor aflate pe nivelul l al unui arbore binomial de ordin k prin $n_k(l)$ și vom încerca să arătam că se va respecta întotdeauna relația $n_k(l) = C_k^l$.

Pe nivelul 0 al oricărui arbore binar de ordin k se află doar rădăcina. Așadar, vom avea întotdeauna $n_k(0) = 1 = C_k^0$.

Am arătat astfel că eci ipoteza de inducție este adevărată.

Presupunem acum că pentru orice valoare $h < l$, pe nivelul h al unui arbore binomial de ordin k se află $n_k(h) = C_k^h$ noduri.

Vom încerca să demonstrăm că pe nivelul l al unui arbore binomial de ordin k se află $n_k(l) = C_k^l$ noduri.

Știm că arborele binomial de ordin k este format din doi arbori binomiali de ordin k - 1, rădăcina unuia fiind un fiu al rădăcinii celuilalt.

Ca urmare numărul nodurilor de pe nivelul l al unui arbore binomial de ordin k este egal cu numărul nodurilor de pe nivelul l - 1 al unui arbore binomial de ordin k - 1, la care se adaugă numărul nodurilor de pe nivelul l al unui arbore binomial de ordin k - 1.

Pe baza acestor afirmații avem:

$$n_k(l) = n_{k-1}(l) + n_{k-1}(l - 1).$$

Folosind ipoteza de inducție putem afirma că vom avea întotdeauna $n_{k-1}(l) = C_{k-1}^l$ și $n_{k-1}(l - 1) = C_{k-1}^{l-1}$.

Pe baza acestor valori obținem:

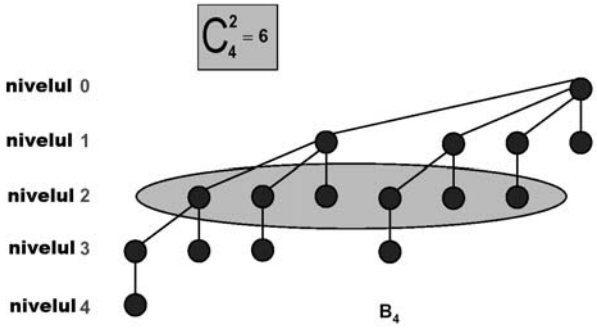

Figura 3: Numărul de noduri de pe un nivel al unui arbore binomial

$$n_k(l) = C_{k-1}^l + C_{k-1}^{l-1}$$
$$n_k(l) = \frac{(k-1)!}{l!\cdot(k-1-l)!} + \frac{(k-1)!}{(l-1)!\cdot(k-1-(l-1))!}$$
$$n_k(l) = \frac{(k-1)!}{l!\cdot(k-1-l)!} + \frac{(k-1)!}{(l-1)!\cdot(k-1-l+1)!}$$
$$n_k(l) = \frac{(k-1)!}{l!\cdot(k-1-l)!} + \frac{(k-1)!}{(l-1)!\cdot(k-l)!}$$
$$n_k(l) = \frac{(k-1)!\cdot(k-l)}{l!\cdot(k-1-l)!\cdot(k-l)} + \frac{(k-1)!\cdot l}{(l-1)!\cdot l\cdot(k-l)!}$$
$$n_k(l) = \frac{(k-1)!\cdot(k-l)}{l!\cdot(k-l)!} + \frac{(k-1)!\cdot l}{l!\cdot(k-l)!}$$
$$n_k(l) = \frac{(k-1)!}{l!\cdot(k-l)!}\cdot(k-l+l)$$
$$n_k(l) = \frac{(k-1)!\cdot k}{l!\cdot(k-l)!}$$
$$n_k(l) = \frac{k!}{l!\cdot(k-l)!}$$
$$n_k(l) = C_k^l$$

De la aceste valori care sunt cunoscute sub numele de ***coeficienți binomiali*** provine și denumirea de ***arbore binomial***. În figura 3 este ilustrată concluzia teoremei anterioare.

O altă proprietate care poate fi observată foarte ușor este faptul că gradul rădăcinii unui arbore binomial de ordin k este k și acesta este gradul maxim al unui nod al arborelui binomial.

Această proprietate rezultă imediat pe baza faptului că rădăcina are exact k fii care sunt rădăcinile unor arbori binomiali de ordine cuprinse între 0 și k - 1.

Folosindu-ne de această proprietate putem enunța următorul corolar.

Corolar

Gradul maxim al unui nod care face parte dintr-un arbore binomial cu n noduri este $\log_2 n$.

Demonstrație
Pe baza proprietății amintite rezultă că rădăcina este nodul cu cel mai mare grad, iar acest grad este k, unde k este ordinul arborelui.

Numărul total al nodurilor este, așa cum am arătat anterior, 2^k, deci avem $2^k = n$. Obținem astfel $k = \log_2 n$.

Heap-uri binomiale

Așa cum am arătat, arborii binomial pot conține doar un număr de elemente care este putere a lui 2. Totuși, am dori să avem o structură de date similară arborilor binomiali care să poată conține un număr arbitrar de elemente.

Această structură este *heap*-ul binomial care va fi prezentat în cele ce urmează.

Definiție
Un *heap* binomial este o colecție de arbori binomiali în care fiecare arbore binomial are o structură de *heap* (cheia oricărui nod este mai mare decât cheia părintelui său) și care conține cel mult un arbore binomial cu un anumit grad.

Prima parte a definiției ne arată că rădăcina unui arbore conține cea mai mică valoare din arborele respectiv.

Cea de-a doua parte a definiției arată că un heap binomial care conține n elemente, va conține cel mult $[\log_2 n]$ arbori binomiali.

Mai exact, *heap*-ul binomial va conține câte un arbore binomial pentru fiecare bit din reprezentarea binară a valorii n, iar gradul arborelui respectiv va fi dat de poziția bitului corespunzător în reprezentarea binară a valorii n.

De exemplu, un heap binomial cu 11 elemente va conține trei arbori binomiali, ale căror grade vor fi 3, 1, respectiv 0. Reprezentarea binară a lui 13 este 1101_2, iar numărul total al nodurilor din cei trei arbori este $2^3 + 2^2 + 2^0 = 8 + 4 + 1 = 13$.

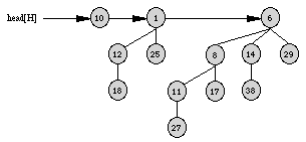

Figura 4: Heap binomial

În figura 4 este prezentat un *heap* binomial care conține 13 elemente ale căror chei sunt numere naturale. *Heap*-ul binomial este reprezentat deobicei ca o listă înlănțuită de arbori binomiali, ordinea arborilor în listă fiind dată de gradele acestora.

Operații
Vom prezenta acum principalele operații care pot fi executate asupra *heap*-urilor inferioare, vom preciza ordinul de complexitate al operației respective (fără a detalia modul în care se realizează operația) și vom compara acest ordin cu cel obținut în cazul *heap*-urilor obținuite.

Operația care se utilizează aproape întotdeuna atunci când folosim *heap*-uri de orice tip este crearea unui *heap* nou vid. Atât pentru heap-urile obișnuite, cât și pentru cele binomiale, această operație se realizează în timp constant.

Determinarea celui mai mic element al structurii este o operația pentru care, practic, au fost "inventate" acest structuri de date. Pentru *heap*-urile obișnuite operația se realizează în timp constant, iar pentru cele binomiale în timp logaritmic.

În orice structură de date trebuie să putem insera elemente. Adăugarea unui element este efectuată în timp logaritmic pentru ambele structuri de date.

O altă operație importantă este ștergerea elementului minim. Aceasta se realizează tot în timp logaritmic pentru ambele tipuri de *heap*-uri.

Pentru *heap*-urile obișnuite, operația de reuniune a două *heap*-uri se poate realiza doar în timp liniar. Acesta este unul dintre motivele pentru care au fost studiate alte tipuri de *heap*-uri. Reuniunea a două *heap*-uri binomiale se realizează în timp logaritmic.

Este cunoscut faptul că, în cazul *heap*-urilor, căutarea este o operație costisitoare. *Heap*-urile binomiale nu îmbunătățesc cu nimic ordinul de complexitate al acestei operații, ea executându-se în timp liniar atât în cazul *heap*-urilor binare, cât și în cazul celor binomiale.

În cele ce urmează vom prezenta modul în care se realizează operațiile de determinare a minimului, inserare, eliminare a minimului și reuniune pentru *heap*-urile binomiale.

Se observă clar că *heap*-urile binomiale sunt mai performante numai dacă avem nevoie de operația de reuniune a două *heap*-uri. În caz contrar, *heap*-urile clasice sunt suficiente.

Determinarea minimului

Știm că rădăcina fiecărui arbore binomial care se află într-un *heap* binomial conține cea mai mică cheie dintre toate cheile nodurile aflate în arborele respectiv.

Ca urmare, pentru a determina cel mai mic element din întregul *heap*, este suficient să determinăm cea mai mică cheie aflată în una dintre rădăcinile arborilor.

Datorită faptului că *heap*-ul binomial conține cel mult $\log_2 n$ arbori, ordinul de complexitate al acestei operații este $O(\log n)$.

Prezentăm în continuare versiunea în pseudocod a algoritmului de determinare a minimului unui *heap* binomial:

```
algoritm DeterminareMinim(H)
  y ← nil
  x ← H.vârf
  min ← ∞
  cât timp x ≠ nil execută
    dacă x.cheie < min atunci
      min ← x.cheie
      y ← x
    sfârșit dacă
    x ← x.următor
  sfârșit cât timp
  returnează y
sfârșit algoritm
```

Se observă că algoritmul va returna o referință spre rădăcina arborelui care conține valoarea minimă. Evident, algoritmul poate fi ușor modificat pentru a returna chiar valoarea minimă.

Pentru *heap*-ul din figura 4, algoritmul va returna o referință către al

doilea element al listei de arbori binomiali.

Reuniunea

Operația de reuniune a două *heap*-uri binomiale este asemănătoare cu adunarea binară a două numere. Practic, se unesc la fiecare pas arbori binomiali care au același grad, obținându-se un arbore binomial al cărui grad este imediat superior.

Unirea a doi arbori binomiali

Pentru reuniunea a două *heap*-uri binomiale avem nevoie de reuniunea a doi arbori binomiali.

Practic, din doi arbori binomiali de grad n, vom obține un arbore binomial de grad $n + 1$.

Operația este foarte simplă și constă în legarea unui arbore de rădăcina celuilalt. Astfel, rădăcina unui arbore va deveni ultimul fiu al rădăcinii celuilalt arbore.

Se observă imediat că se păstrează toate proprietățile arborilor binomial, dar trebuie să ne asigurăm că se va păstra și proprietatea de *heap*. Pentru aceasta este suficient să ne asigurăm că cheia rădăcini noului arbore este întotdeauna cea mai mică dintre cheile rădăcinilor celor doi arbori. De fapt, va trebui să decidem înainte care arbore va deveni subarbore al celuilalt.

După ce a fost luată o decizie în acest sens (pe baza comparării cheilor) poate fi folosit următorul subalgoritm:

```
subalgoritm UneșteArbori(y, z)
  y.părinte ← z
  y.următor ← z.primul_fiu
  z.primul_fiu ← y
  z.grad ← z.grad + 1
sfârșit subalgoritm
```

Astfel, nodul y devine primul fiu al nodului z în timp constant, deoarece fiecare dintre cele patru operații efectuate necesită un timp constant pentru a fi executată.

Algoritmul de reuniune

Vom prezenta în cele ce urmează algoritmul care poate fi utilizat pentru a reuni două *heap*-uri binomiale.

Vom prezenta versiunea în pseudocod a unui algoritm care realizează această operație. Există patru cazuri care trebuie tratate, fiecare fiind ilustrat în figura 5.

Pe lângă subalgoritmul de unire a doi arbori binomiali, mai este utilizat un subalgoritm care realizează interclasarea listelor arborilor din cele două *heap*-uri binomiale, astfel încât arborii să fie ordonați crescător în funcție de gradele lor. Subalgoritmul are ca parametri referințe spre vârfurile listelor de arbori din cele două *heap*-uri și returnează o referință spre vârful noii liste. Acest subalgoritm este foarte simplu, motiv pentru care nu îl mai prezentăm aici.

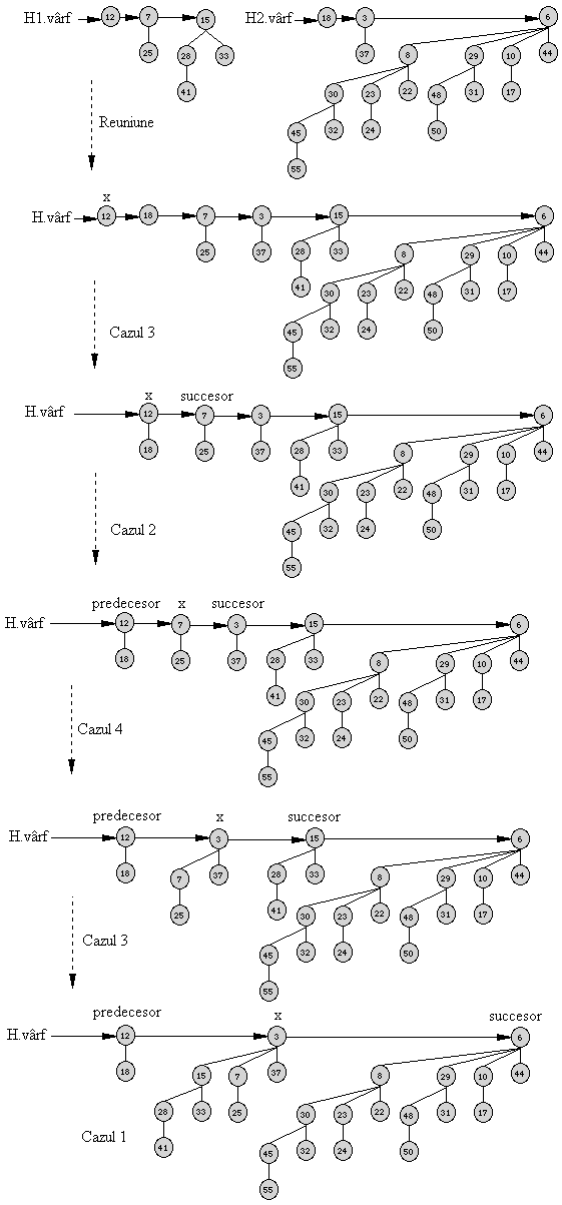

Figura 5: Exemplu pentru reuniunea a două heap-uri binomiale

```
algoritm Reuniune(H1, H2)
  H.vârf ← Interclasare
                    (H1, H2)
  dacă H.vârf = nil atunci
    returnează H
  sfârșit dacă
  predecesor ← nil
  x ← H.vârf
  succesor ← x.următor
  cât timp succesor ≠ nil
                    execută
    dacă x.grad ≠ succesor.
          grad sau (succesor.
             următor ≠ nil și
            succesor.următor.
              grad = x.grad)
                    atunci
      // cazurile 1 și 2
      anterior ← x
      x ← succesor
    altfel
      dacă x.cheie ≤
        succesor.cheie atunci
        // cazul 3
        x.următor ←
            succesor.următor
        UneșteArbori(
              succesor, x)
      altfel
        // cazul 4
        dacă predecesor =
                  nil atunci
          H.vârf ← succesor
        altfel
```

```
            predecesor.următor
                    ← succesor
        sfârșit dacă
        UneșteArbori(x,
                    succesor)
        x ← succesor
      sfârșit dacă
    sfârșit dacă
    succesor ← x.următor
  sfârșit cât timp
  returnează H
sfârșit algoritm
```

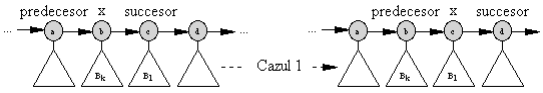

Figura 6: Primul caz pentru reuniunea a două heap-uri binomiale

Descrierea algoritmului

Algoritmul de reuniune a două heap-uri binomiale constă în două faze. Prima fază constă din apelul subalgoritmului care interclasează listele de arbori ale celor două heap-uri într-o singură listă înlănțuită care este ordonată crescător în funcție de gradele arborilor binomiali care o formează.

Pot exista cel mult doi arbori (dat nu mai mulți!) care să aibă un anumit grad. În cea de-a doua fază sunt uniți arborii care au grade egale. Datorită faptului că lista este ordonată în funcție de gradele arborilor, această operație poate fi executată foarte repede.

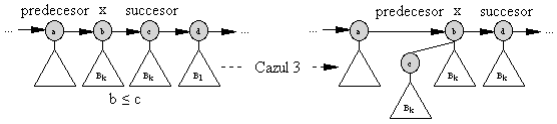

Figura 8: Al treilea caz pentru reuniunea a două heap-uri binomiale

Algoritmul continuă cu păstrarea unui referințe spre primul element al listei create. În cazul în care lista nu are nici un element, *heap*-ul rezultat este vid și algoritmul se oprește.

Dacă algoritmul continuă, știm cu siguranță că avem cel puțin un arbore. Vom păstra pentru a simplifica operațiile câze un pointer spre elementele vecine elementului curent (predecesorul și succesorul).

La început există cel mult doi arbori binomiali care au un anumit grad dat deoarece am avut două *heap*-uri binomiale care (conform definiției) conțineau un singur arbore de un anumit grad dat.

Mai mult, operația de interclasare a arborilor ne asigură că cei doi arbori se află pe poziții consecutive în lista obținută după interclasare.

Totuși, în timpul execuției putem avea până la trei arbori care au un anumit grad dat. Vom vedea la momentul oportun cum poate apărea o astfel de situație.

La fiecare iterație a structurii repetitive vom decide dacă vom uni arborele curent și succesorul său în funcție de gradele lor și, eventual, în funcție de gradul arborelui care se află la două poziții după succesorul arborelui curent.

Pe parcursul executării acestui ciclu vom fi siguri întotdeauna că referințele spre arborele curent și spre succesorul acestuia nu sunt nule.

În continuare vom identifica cele patru cazuri care pot apărea în timpul executării ciclului.

Primul caz tratează situația în care arborele curent are gradul diferit de gradul succesorului său. În această situație cei doi arbori nu sunt uniți și se trece la următorul arbore din listă. Acest caz este ilustrat în figura 6.

Cel de-al doilea caz tratează situația în care avem trei arbori de același grad aflați pe poziții consecutive. Și în acest caz vom trece pur și simplu la arborele următor, urmând ca la următorul pas cei doi arbori care urmează după arborele curent să fie uniți. O reprezentare a acestui caz este prezentată în figura 7.

Celelalte două cazuri apar în situația în care arborele curent și succesorul său au același grad. Este demn de observat faptul că după cazul al doilea va apărea cu siguranță unul dintre aceste cazuri. În aceste cazuri cei doi arbori trebuie uniți, diferența dintre ele fiind dată de relația de ordine dintre cheile din rădăcinile celor doi arbori.

În cazul al treilea, cheia rădăcinii arborelui curent este mai mică sau egală cu cheia rădăcinii succesorului. În această situație, după unirea arborilor, succesorul este eliminat din listă. Figura 8 descrie acest al treilea caz.

În cazul al patrulea, cheia rădăcinii arborelui curent este mai mare decât cheia rădăcinii succesorului. În această situație, după unirea arborilor, arborele curent este eliminat din listă. Putem avea două situații pentru această eliminare: arborele curent poate sau nu fi primul în lista arborilor. Cazul este prezentat în figura 9.

Dacă ne-am aflat în al treilea sau al patrulea caz, am obținut un arbore binomial nou care este acum arborele curent. Înainte de efectuarea acestei operații, lista mai putea conține până la doi arbori de acest grad.

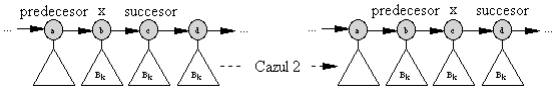

Figura 7: Al doilea caz pentru reuniunea a două heap-uri binomiale

Așadar, în acest moment ar putea urma nici unul, unul sau doi arbori cu același grad ca și arborele curent.

Dacă nu urmează nici unu alt arbore de același grad, atunci la pasul următor ne vom afla în primul caz; dacă urmează un singur arbore de același grad, atunci la pasul următor ne vom afla în al treilea sau al patrulea caz; în sfârșit, dacă urmează doi arbori de același grad, atunci la pasul următor ne vom afla în al doilea caz.

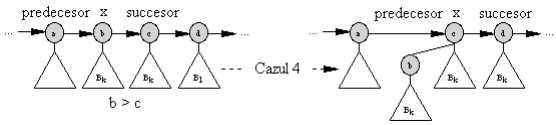

Figura 9: Al patrulea caz pentru reuniunea a două heap-uri binomiale

Ordinul de complexitate al acestui algoritm de reuniune a două *heap*-uri binomiale este $O(\log n)$, unde n este numărul total de noduri din cele două *heap*-uri.

Operația de reuniune este cea mai importantă dintre operațiile cu *heap*-uri binomiale, deoarece atât inserarea, cât și ștergerea, se bazează pe ea.

Inserarea

Pentru a adăuga un element într-un *heap* binomial *H* este suficient să creăm un nou *heap* binomial H' și să reunim *heap*-urile *H* și *H'*.

Crearea *heap*-ului binomial necesită un timp constant, iar interclasarea un timp logaritmic.

Versiunea în pseudocod a algoritmului de inserare a unui element într-un heap binomial este următoarea:

```
algoritm Inserare(H, x)
  x.părinte ← nil
  x.primul_fiu ← nil
  x.următor ← nil
  x.grad ← 0
  H'.vârf ← x
  H ← Reuniune(H, H')
sfârșit algoritm
```

Ștergerea minimului

Pentru a șterge elementul minim al unui *heap* binomial, vom identifica mai întâi arborele în a cărui rădăcină este păstrat acest element.

Vom elimina din lista arborilor *heap*-ului binomial arborele identificat și apoi vom construi un alt *heap* binomial care va conține arborii ai căror rădăcini au fost fiii rădăcinii care trebuie eliminată.

După interclasarea celor două *heap*-uri vom obține un *heap* binomial din care am eliminat elementul minim.

Versiunea în pseudocod a algoritmului descris este prezentată în continuare:

```
algoritm EliminăMinim(H)
  x ← DeterminăMinim(H)
  elimină x din H
  inversează ordinea arborilor
                din lista fiilor lui x
  H'.vârf ← x.primul_fiu
  H ← Reuniune(H, H')
  returnează x
sfârșit algoritm
```

Pașii acestui algoritm sunt prezentați în figura 10. Pentru exemplificare am folosit un *heap* binomial format trei arbori ale căror grade sunt 1, 2 și 4.

La primul pas vom identifica arborele a cărui rădăcină conține cea mai mică cheie. Cheia minimă este 1, deci vom elimina din *heap* arborele de grad 4.

La al doilea pas vom inversa ordinea fiilor rădăcinii arborelui eliminat pentru a putea crea un *heap* binomial ai cărui arbori să fie în ordinea corectă. Noua listă reprezintă un *heap* binomial care conține arbori binomiali de gradele 0, 1, 2 și 3.

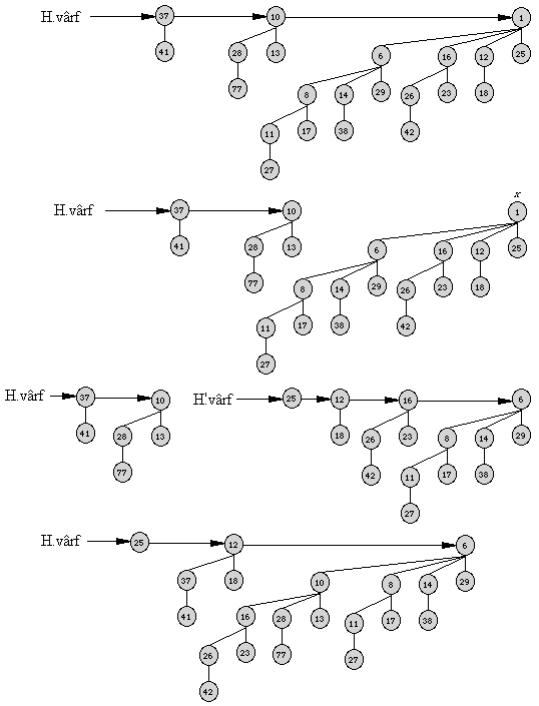

Figura 10: Exemplu pentru eliminarea elementului minim dintr-un heap binomial

La cel de-al treilea pas vom realiza reuniunea celor două *heap*-uri și vom obține un *heap* binomial cu trei arbori ale căror grade sunt 0, 2 și 4.

Se poate observa în imagine că structura *heap*-ului s-a schimbat destul de mult după ce am eliminat un element al acestuia.

Concluzii

Am prezentat în cadrul acestui articol o structură de date care poate fi folositoare în diferite cazuri.

Cea mai frecventă situație în care sunt utilizate aceste *heap*-uri binomiale apare atunci când avem mai multe cozi de priorități pe despre care știm că, la un moment dat, vor trebui să formeze o singură coadă de priorități (evident, cu respectarea priorităților).

Am văzut că un *heap* binomial este o coadă de priorități care nu este reprezentată sub forma unui arbore, ci sub forma unei colecții de arbori, iar fiecare dintre acești arbori trebuie să respecte structura de *heap*. O colecție de arbori este numită în majoritatea cazurilor ***pădure***.

Fiecare dintre arborii care formează un *heap* binomial are o structură specială; acești arbori sunt denumiți arbori binomiali, iar această denumire provine de la faptul că pe un nivel al unui astfel de arbore se află un număr de noduri egal cu coeficientul binomial corespunzător nivelului respectiv.

Arborii binomiali sunt arbori generali pentru care nu se impune nici o restricție cu privire la gradul maxim al unui nod (gradul unui nod nu este fixat la o anumită valoare sau limitat la un interval de valori).

O caracteristică remarcabilă a heap-urilor binomiale este reprezentată de faptul că operația de reuniune este similară adunării binare. Pădurea de arbori binomiali care formează un heap binomial poate fi asemuită cu mulțimea de biți care formează reprezentarea binară a unui număr natural.

Mai mult, reuniunea a două heap-uri binomiale se realizează într-o manieră foarte asemănătoare adunării binare. Arborii binomiali sunt uniți într-o manieră similară combinării biților atunci când sunt adunate două numere binare.