BLM267

Bölüm 12: Yığınlar

C Kullanarak Veri Yapıları, İkinci Baskı

- İkili Yığınlar
- Binom Yığınları

İkili Yığınlar

- İkili yığın, her düğümün şu yığın özelliğini karşıladığı tam bir ikili ağaçtır:
- **Eğer B, A'nın bir çocuğu ise, o zaman key(A) ≥ key(B)**
- Bu, her düğümdeki elemanların, sol ve sağ çocuğundaki elemandan büyük veya eşit olacağı anlamına gelir.
- Böylece, kök düğüm yığındaki en yüksek anahtar değerine sahiptir. Böyle bir yığın genellikle max-heap olarak bilinir.
- Alternatif olarak, her düğümdeki elemanlar, sol ve sağ çocuğundaki elemanlardan daha küçük veya eşit olacaktır.
- Bu nedenle, kök en düşük anahtar değerine sahiptir. Böyle bir yığına min-yığın denir.

İkili Yığınlar

- Şekil 12.1'de ikili bir minimum yığın ve ikili bir maksimum yığın gösterilmektedir.

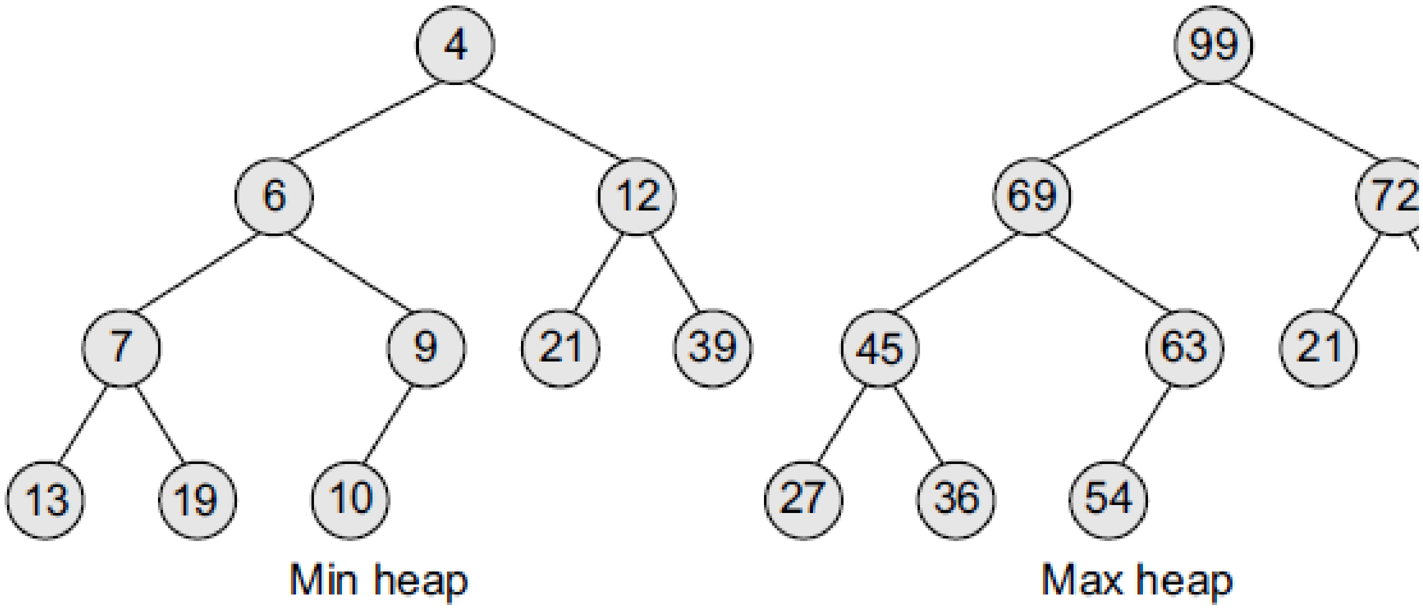


Figure 12.1 Binary heaps

İkili Yığınlar

- İkili yığınların özellikleri aşağıdaki gibi verilmiştir:
  - Bir yığın, tam bir ikili ağaç olarak tanımlandığından, tüm elemanlar bir dizide sıralı olarak depolanabilir. Tam bir ikili ağaçla aynı kuralları izler. Yani, bir eleman dizide i pozisyonundaysa, sol çocuğu 2i pozisyonunda ve sağ çocuğu 2i+1 pozisyonunda depolanır. Tersine, i pozisyonundaki bir elemanın ebeveyni i/2 pozisyonunda depolanır.
  - Tam bir ikili ağaç olduğundan ağacın son seviyesi hariç tüm seviyeleri tamamen doludur.
  - İkili ağacın yüksekliği log2n olarak verilir; burada n, eleman sayısıdır.
  - Yığınlar (kısmen sıralı ağaçlar olarak da bilinir) öncelikli kuyruklar uygulamak için oldukça popüler bir veri yapısıdır.
- İkili yığın, elemanların rastgele eklenebildiği ancak maksimum yığın durumunda yalnızca en yüksek değere sahip elemanın, minimum yığın durumunda ise yalnızca en düşük değere sahip elemanın çıkarıldığı kullanışlı bir veri yapısıdır.

İkili Yığınlar

İkili Yığına Yeni Bir Eleman Ekleme

- n elemanlı bir maksimum yığın H düşünün. Yığına yeni bir değe eklemek aşağıdaki iki adımda yapılır:

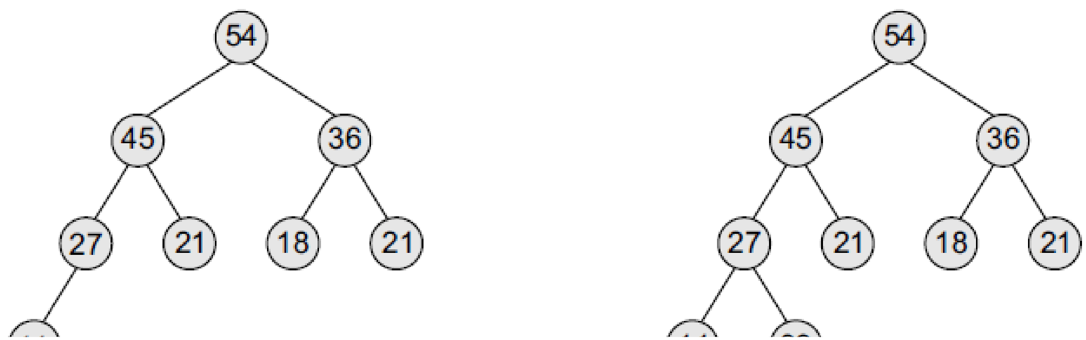
- 1. H'nin en altına yeni değeri ekleyin; böylece H hâlâ tam bir iki ağaç olur ama mutlaka bir yığın olmaz.
- 2. Yeni değerin H'deki uygun yerine yükselmesine izin verin, böylece H artık bir yığın haline gelir.

- Bunu yapmak için, yeni değeri ebeveyniyle karşılaştırarak doğr sırada olup olmadıklarını kontrol edin. Eğer doğru sıradalarsa, prosedür durur, aksi takdirde yeni değer ve ebeveyninin değeri değiştirilir ve Adım 2 tekrarlanır.

İkili Yığınlar

Example 12.1 Consider the max heap given in Fig. 12.2 and insert 99 in it.

Solution



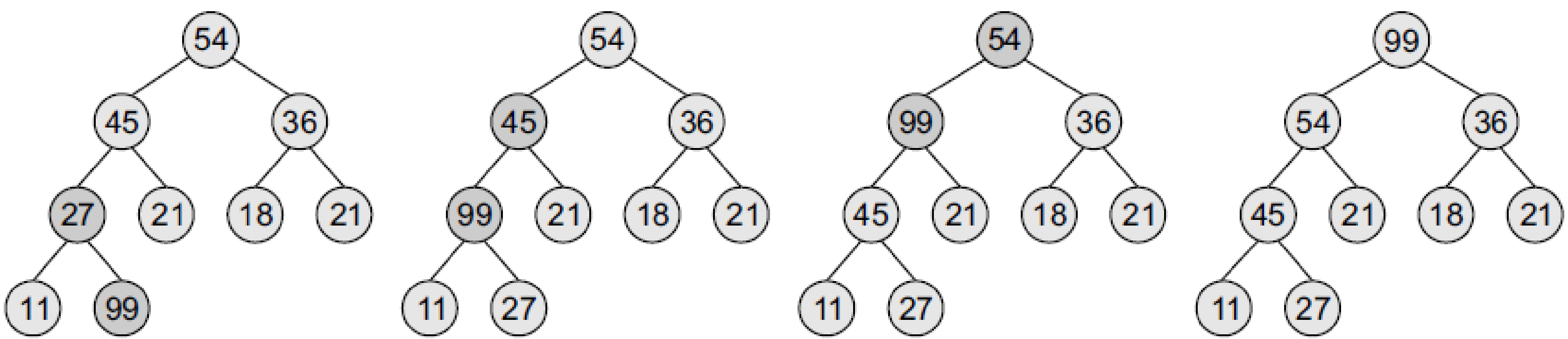


Figure 12.4 Heapify the binary heap

İkili Yığınlar

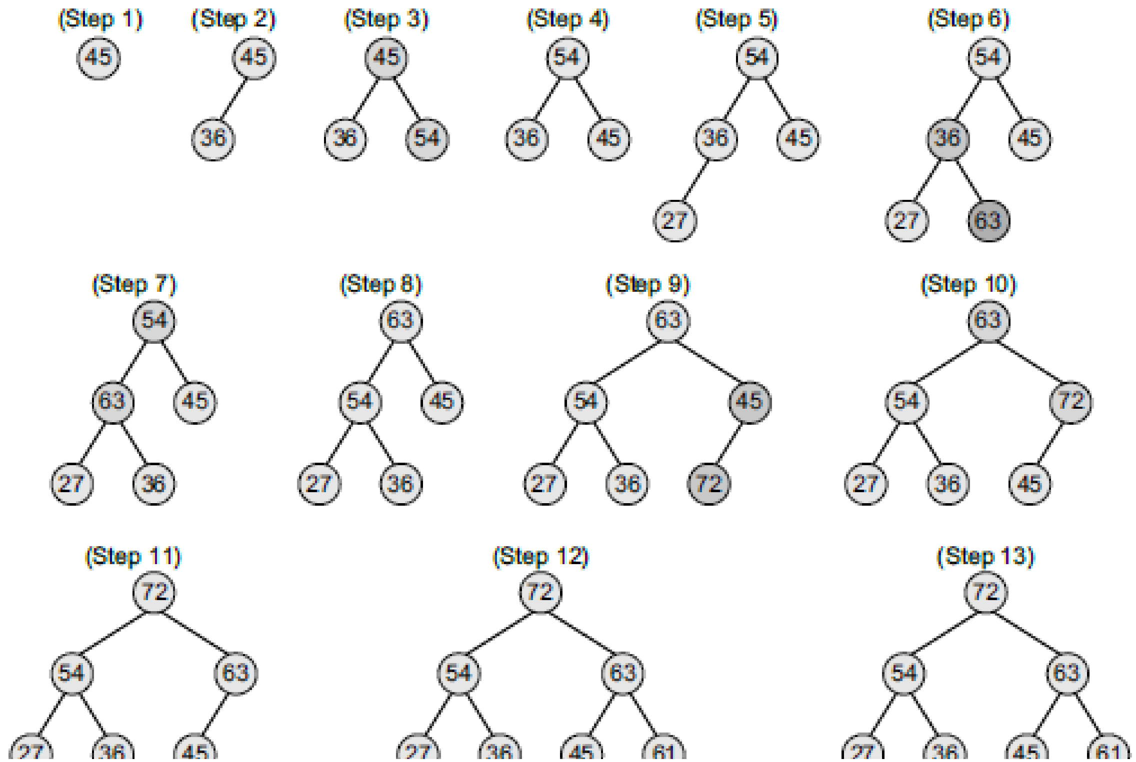
İkili Yığına Yeni Bir Eleman Ekleme

- İlk adım, ögeyi yığına yerleştirerek yığının tam bir ikili ağaç olmasını sağlamaktır.
- Yani, yeni değeri yığındaki 27 nolu düğümün sağ çocuğu olarak ekleyin. Bu Şekil 12.3'te gösterilmiştir.
- Şimdi ikinci adıma göre yeni değerin H'deki uygun yerine yükselmesini sağlayalım ki H de bir yığın olsun.
- 99'u ebeveyn düğüm değeriyle karşılaştırın. Ebeveyninin değerinden küçükse, yeni düğüm uygun yerindedir ve H bir yığındır.
- Eğer yeni değer, ebeveyninin düğümünün değerinden büyükse iki değeri birbiriyle değiştirin.
- H bir yığın haline gelene kadar tüm işlemi tekrarlayın. Bu Şekil 12.4'te gösterilmiştir.

İkili Yığınlar

Example 12.2 Build a max heap H from the given set of numbers: 45, 36, 54, 27, 63, 72, 61, and 18. Also draw the memory representation of the heap.

Solution



HEAP[1]	HEAP[2]	HEAP[3]	HEAP[4]	HEAP[5]	HEAP[6]	HEAP[7]	HEAP[8]	HEAP[9]	HEAP[10]
72	54	63	27	36	45	61	18		

Figure 12.6 Memory representation of binary heap H

İkili Yığınlar

İkili Yığına Yeni Bir Eleman Ekleme

- Yığına yeni bir değer eklemenin ardındaki kavramı tartıştıktan sonra, şimdi bunu yapmak için Şekil 12.7'de gösterilen algoritmaya bakalım.
- n elemanlı H'nin HEAP dizisinde saklandığını varsayıyoruz.
- VAL HEAP'e eklenmelidir. VAL'in yığında yükselirkenki konumu POS tarafından verilir ve PAR VAL'in ebeveyninin konumunu belirtir.

```
Step 1: [Add the new value and set its POS]
        SET N = N + 1, POS = N
Step 2: SET HEAP[N] = VAL
Step 3: [Find appropriate location of VAL]
        Repeat Steps 4 and 5 while POS > 1
Step 4:     SET PAR = POS/2
Step 5:     IF HEAP[POS] <= HEAP[PAR],
            then Goto Step 6.
            ELSE
                  SWAP HEAP[POS], HEAP[PAR]
                  POS = PAR
            [END OF IF]
      [END OF LOOP]
Step 6: RETURN
```

Figure 12.7 Algorithm to insert an element in a max heap

İkili Yığınlar

İkili Yığına Yeni Bir Eleman Ekleme

- Bu algoritmanın yığına tek bir değer eklediğini unutmayın. Bir yığın oluşturmak için bu algoritmayı bir döngüde kullanın.
- Örneğin, 9 elemanlı bir yığın oluşturmak için, 9 kez yürütülen v her geçişte tek bir değerin eklendiği bir for döngüsü kullanın.
- Bu algoritmanın ortalama durumdaki karmaşıklığı O(1)'dir.
- Bunun nedeni ikili yığının O(log n) yüksekliğe sahip olmasıdır.
- Elemanların yaklaşık %50'si yaprak ve %75'i alt iki seviyede olduğundan, eklenecek yeni eleman yığını korumak için sadece birkaç seviye yukarı doğru hareket edecektir.
- En kötü durumda, tek bir değerin eklenmesi O(log n) zaman alabilir ve benzer şekilde n elemanlı bir yığın oluşturmak için algoritma O(n log n) sürede çalışacaktır.

İkili Yığınlar

İkili Yığından Bir Elemanı Silme

- n elemana sahip bir H maksimum yığınını ele alalım. Yığının kökünden her zaman bir eleman silinir.
- Yani yığından bir elemanın silinmesi şu üç adımda gerçekleşir:

- 1. Kök düğümün değerini son düğümün değeriyle değiştirin, böylece H hala tam bir ikili ağaç olur ancak mutlaka bir yığın olmaz.
- 2. Son düğümü silin.
- 3. Yeni kök düğümün değerini, H yığın özelliğini karşılayacak şekilde aşağıya doğru batırın.
- Bu adımda, kök düğümün değerini, alt düğümün değeriyle (alt düğümler arasında hangisi en büyükse) değiştirin.

İkili Yığınlar

İkili Yığından Bir Elemanı Silme

- Burada kök düğümün değeri = 54 ve son düğümün değeri = 11. Dolayısıyla 54'ü 11 ile değiştirip son düğümü siliyoruz.

Example 12.3 Consider the max heap H shown in Fig. 12.8 and delete the root node's value.

Solution

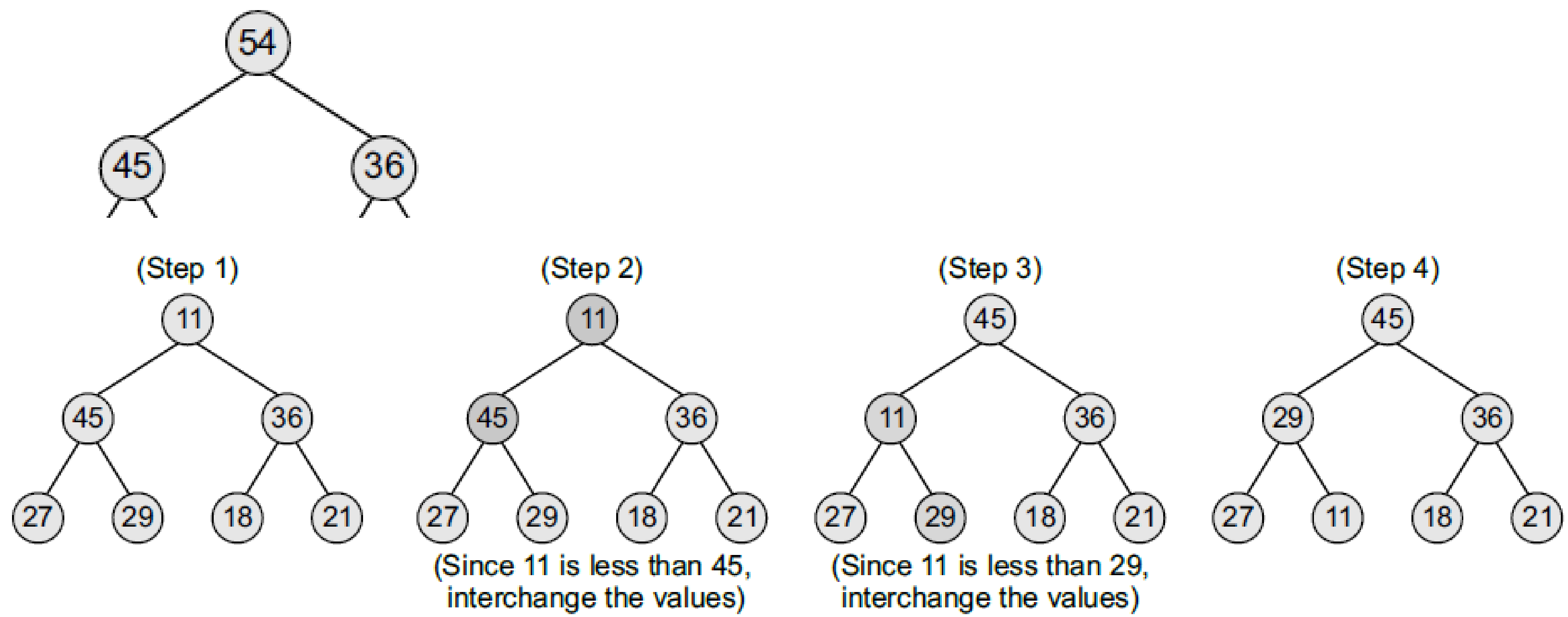


Figure 12.9 Binary heap

İkili Yığınlar

İkili Yığından Bir Elemanı Silme

- Kök elemanını yığından silmenin ardındaki kavramı tartıştıktan sonra, Şekil 12.10'da verilen algoritmaya bakalım.
- n elemanlı H yığınının HEAP adı verilen sıralı bir dizi kullanılarak depolandığını varsayıyoruz.
- LAST, yığın içindeki son elemandır ve PTR, LEFT ve RIGHT, LAST'ın ve onun sol ve sağ çocuklarının yığın içinde aşağı doğru hareket ettikçe sırasıyla konumlarını belirtir.

```
Step 1: [Remove the last node from the heap]
        SET LAST = HEAP[N], SET N = N - 1
Step 2: [Initialization]
        SET PTR = 1, LEFT = 2, RIGHT = 3
Step 3: SET HEAP[PTR] = LAST
Step 4: Repeat Steps 5 to 7 while LEFT <= N
Step 5: IF HEAP[PTR] >= HEAP[LEFT] AND
         HEAP[PTR] >= HEAP[RIGHT]
              Go to Step 8
        [END OF IF]
Step 6: IF HEAP[RIGHT] <= HEAP[LEFT]
              SWAP HEAP[PTR], HEAP[LEFT]
              SET PTR = LEFT
        ELSE
              SWAP HEAP[PTR], HEAP[RIGHT]
              SET PTR = RIGHT
        [END OF IF]
Step 7: SET LEFT = 2 * PTR and RIGHT = LEFT + 1
        [END OF LOOP]
Step 8: RETURN
```

Figure 12.10 Algorithm to delete the root element from a max heap

İkili Yığınlar

İkili Yığınların Uygulamaları

- İkili yığınlar esas olarak şunlar için uygulanır:
  1. Yığın sıralama algoritmasını kullanarak bir diziyi sıralama.
  2. Öncelikli kuyrukların uygulanması.

Öncelikli Kuyrukların İkili Yığın Uygulaması

- Öncelikli kuyruk, bir öğenin önden kuyruktan çıkarıldığı (veya kaldırıldığı) bir kuyruğa benzer.
- Ancak normal bir kuyruğun aksine, öncelikli bir kuyrukta öğelerin mantıksal sırası önceliklerine göre belirlenir.
- Önceliği yüksek olan elemanlar sıranın başına eklenirken, önceliği düşük olan elemanlar sıranın sonuna eklenir.
- Öncelik kuyruklarını doğrusal bir dizi kullanarak kolayca uygulayabiliriz, ancak öncelikle diziye bir eleman eklemek ve ardında sıralamak için gereken zamanı göz önünde bulundurmalıyız.
- Bir eleman eklemek için O(n) zamana ve diziyi sıralamak için en az O(n log n) zamana ihtiyacımız var.
- Bu nedenle, öncelikli bir kuyruğu uygulamanın daha iyi bir yolu, hem elemanların O(log n) sürede kuyruğa alınmasına hem de kuyruktan çıkarılmasına izin veren ikili bir yığın kullanmaktır.

Yığın Sıralaması

- Yığın sıralamasının çalışma zamanı karmaşıklığı O(nlogn)'dir.
- n elemanlı bir ARR dizisi verildiğinde, yığın sıralama algoritması ARR'y iki aşamada sıralamak için kullanılabilir:
- 1. aşamada ARR elemanlarını kullanarak bir yığın H oluşturun.
- 2. aşamada, 1. aşamada oluşturulan yığının kök elemanını tekrar tekra silin.
- Maksimum yığında, H'deki en büyük değerin her zaman kök düğümde bulunduğunu biliyoruz. Bu nedenle, 2. aşamada, kök eleman silindiğin aslında ARR'nin elemanlarını azalan düzende topluyoruz.

```
HEAPSORT(ARR, N)

Step 1: [Build Heap H]
        Repeat for I = 0 to N-1
                CALL Insert_Heap(ARR, N, ARR[I])
        [END OF LOOP]
Step 2: (Repeatedly delete the root element)
        Repeat while N>0
                CALL Delete_Heap(ARR, N, VAL)
                SET N = N + 1
        [END OF LOOP]
Step 3: END
```

Figure 14.12 Algorithm for heap sort

Yığın Sıralaması

- ***Yığın Sıralamanın Karmaşıklığı***
- Yığın sıralaması iki yığın işlemi kullanır: ekleme ve kök silme. Kökten çıkarılan her eleman dizinin son boş konumuna yerleştir
- 1. aşamada bir yığın oluşturduğumuzda, H'deki yeni elemanın doğru konumunu bulmak için yapılacak karşılaştırmaların sayısı H'nin derinliğini aşamaz.
- H tam bir ağaç olduğundan derinliği m'yi geçemez; burada m, H yığınındaki eleman sayısıdır.
- Böylece, H'ye ARR'nin n elemanını eklemek için toplam karşılaştırma sayısı g(n) şu şekilde sınırlandırılır:

  $g(n) <= n \log n$
- Bu nedenle, yığın sıralama algoritmasının ilk aşamasının çalışma süresi O(n log n)'dir.

Yığın Sıralaması

- ***Yığın Sıralamanın Karmaşıklığı***
- 2. aşamada, yığınlar halinde sol ve sağ alt ağaçlara sahip m elemanlı tam bir olan H'ye sahibiz.
- L'nin ağacın kökü olduğunu varsayarsak, ağacı yeniden yığmak L'yi ağaç H'd bir adım aşağı hareket ettirmek için 4 karşılaştırma gerektirir. H'nin derinliği O(log m)'yi aşamayacağından, ağacı yeniden yığmak L'nin H'deki doğru yerin bulmak için en fazla 4 log m karşılaştırma gerektirir.
- n eleman yığın H'den silineceğinden, yeniden yığınlama n kez yapılacaktır. B nedenle, n elemanı silmek için karşılaştırma sayısı şu şekilde sınırlandırılmış

  $h(n) <= 4n \log n$
- Bu nedenle, yığın sıralama algoritmasının ikinci aşamasının çalışma süresi O(log n)'dir.
- Her faz, O(n log n) ile orantılı zaman gerektirir. Bu nedenle, en kötü durumda elemanlı bir diziyi sıralamak için gereken çalışma süresi O(n log n) ile orantılıdır.
- Dolayısıyla yığın sıralamasının büyük veri kümelerini verimli bir şekilde sıralamak için kullanılabilen basit, hızlı ve kararlı bir sıralama algoritması olduğu sonucuna varabiliriz.

Binom Yığınları

- Binom yığını H, binom yığını özelliklerini sağlayan binom ağaçlarının kümesidir.
- Öncelikle iki terimli ağacın ne olduğunu tartışalım.
- Binom ağacı, aşağıdaki gibi yinelemeli olarak tanımlanabile sıralı bir ağaçtır:
  - 0. dereceden bir binom ağacının tek bir düğümü vardır.
  - i mertebesinden bir binom ağacının kök düğümünün çocukları 1, i–2, ..., 2, 1 ve 0 mertebesinden iki terimli ağaçların kök düğümleridir.
  - Bi iki terimli ağacın 2^I düğümü vardır.
  - Bi binom ağacının yüksekliği i'dir.

Binom Yığınları

- Farklı sıralarda birkaç binom ağacını gösteren Şekil 12.12'ye bakın. İki Bi-1 sırasındaki binom ağaçlarından, birinin kökü diğerinin kökünün en soldaki çocuğu olacak şekilde birbirine bağlayarak bir Bi binom ağacı oluşturabiliriz.

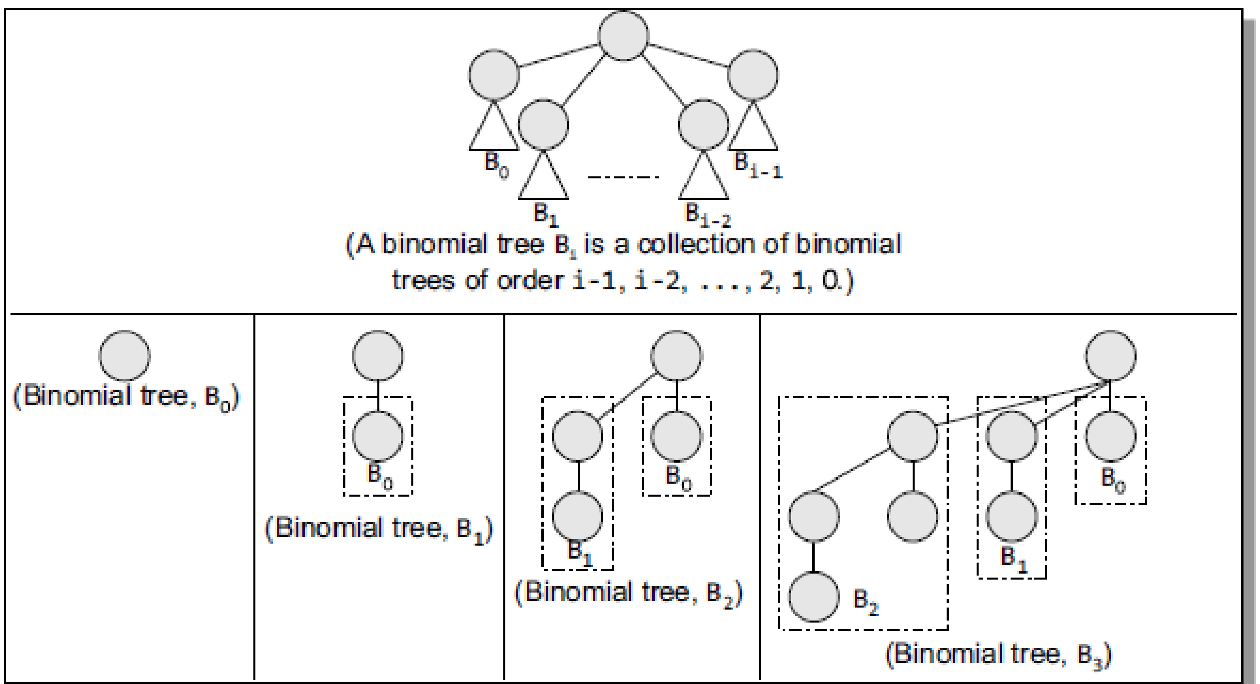


Figure 12.12 Binomial trees

Binom Yığınları

- Binom yığını H, aşağıdaki özellikleri sağlayan binom ağaçlarının bir koleksiyonudur:
  - H'deki her iki terimli ağaç, minimum yığın özelliğini karşılar (yani, bir düğümün anahtarı, üst düğümünün anahtarından büy veya ona eşittir).
  - Sıfırıncı derece de dahil olmak üzere her derece için bir veya s binom ağacı olabilir.
- İlk özelliğe göre, yığın sıralı bir ağacın kökü, ağaçtaki en küçük anahtarı içerir.
- İkinci özellik ise, N düğümü olan bir binom yığını H'nin en fazla log (N + 1) binom ağacı içerdiğini ima eder.

Binom Yığınları

- **Binom Yığınlarının Bağlantılı Gösterimi**
- Binom yığınındaki H her düğümün değerini depolayan bir val ala vardır. Ek olarak, her düğüm N'nin aşağıdaki işaretçileri vardır:
  - N'nin ebeveynini işaret eden P[N]
  - En soldaki çocuğu işaret eden Çocuk[N]
  - Kardeş[N], N'nin hemen sağında bulunan kardeşini gösterir
- Eğer N kök düğüm ise, o zaman P[N] = NULL. Eğer N'nin çocuğu yoksa, o zaman Child[N] = NULL ve eğer N ebeveyninin en sağdaki çocuğu ise, o zaman Sibling[N] = NULL.
- Buna ek olarak, her N düğümü, N'nin çocuklarının sayısını depolayan bir derece alanına sahiptir. Şekil 12.13'te gösterilen binom yığınına bakın.
- Şekil 12.14 buna karşılık gelen bağlantılı gösterimi göstermekte

Binom Yığınları

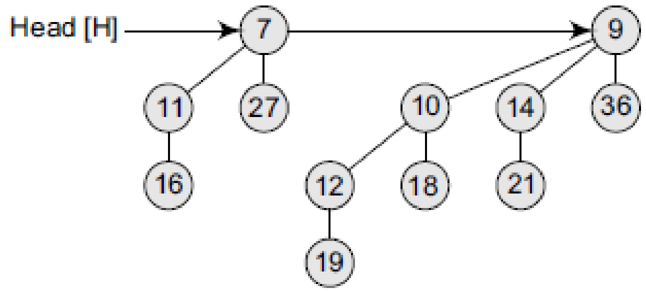


Figure 12.13 Binomial heap

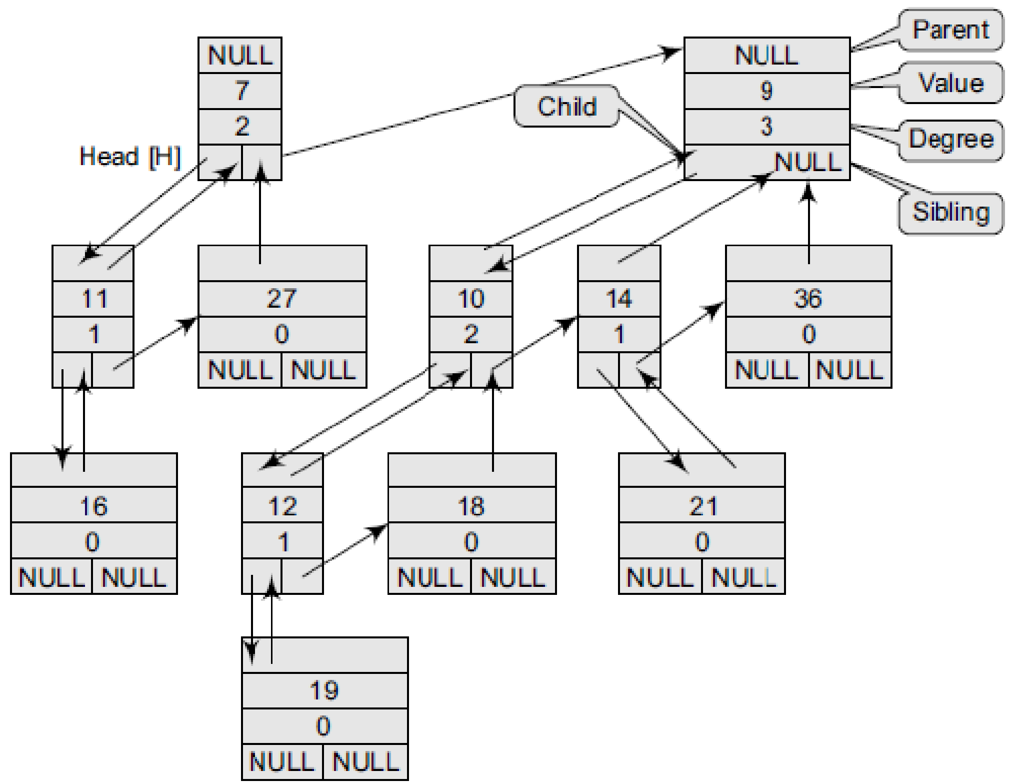


Figure 12.14 Linked representation of the binomial tree shown in Fig. 12.13

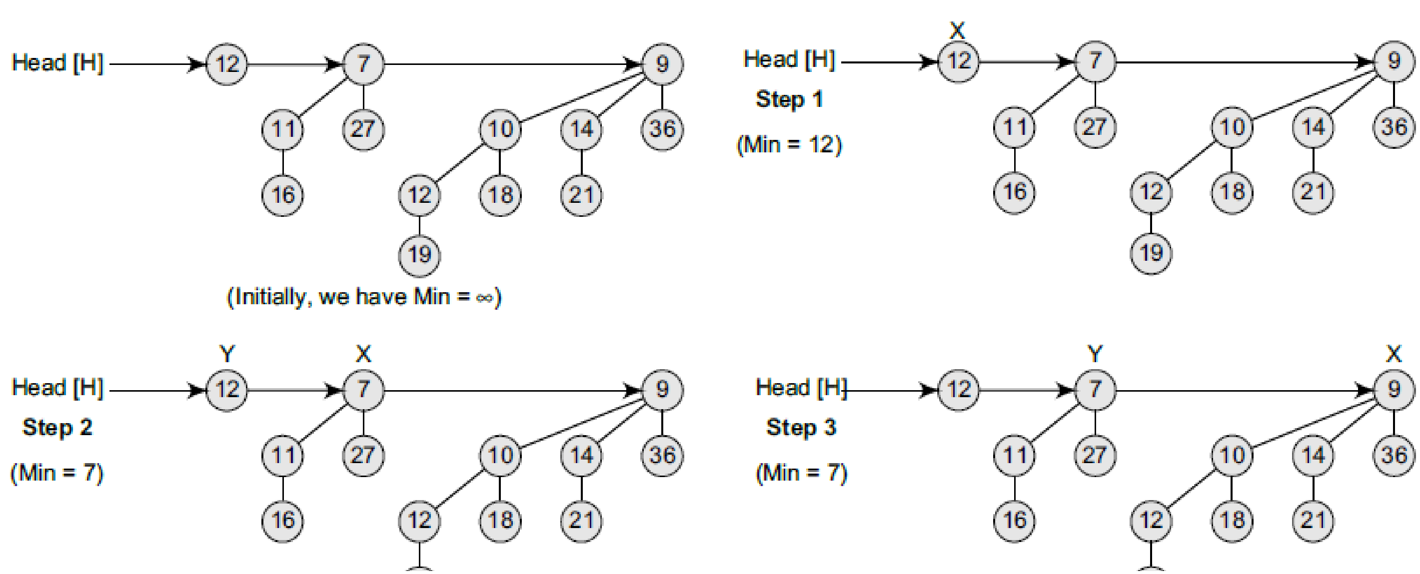
Binom Yığınları

- **Binom Yığınları Üzerindeki İşlemler**
- Bu bölümde, binom yığınları üzerinde gerçekleştirilebilecek farklı işlemleri tartışacağız.
- ***Yeni Bir Binom Yığını Oluşturma***
- Create_Binomial–Heap() prosedürü, Head[H]'nin NULL olarak ayarlandığı bir nesne H'yi tahsis eder ve döndürür. Bu prosedürün çalışma süresi O(1) olarak verilebilir.
- ***Minimum Anahtara Sahip Düğümü Bulma***
- Min_Binomial–Heap() prosedürü, binom yığını H'de minimum değere sahip düğüme bir işaretçi döndürür. Min_Binomial–Heap() için algoritm Şekil 12.15'te gösterilmiştir.
- Binom yığınının yığın-düzenli olduğunu daha önce tartışmıştık; bu nedenle, belirli bir binom ağacında en düşük değere sahip düğüm, bin yığınında bir kök düğüm olarak görünecektir.
- Bu nedenle, Min_ Binomial–Heap() prosedürü tüm kökleri kontrol eder Kontrol edilecek en fazla log (n + 1) kök olduğundan, bu prosedürün çalışma süresi O(log n)'dir.

Binom Yığınları

```
Min_Binomial-Heap(H)

Step 1: [INITIALIZATION] SET Y = NULL, X = Head[H] and Min = ∞
Step 2: REPEAT Steps 3 and 4 While X ≠ NULL
Step 3:      IF Val[X] < Min
                  SET Min = Val[X]
                  SET Y = X
             [END OF IF]
Step 4:      SET X = Sibling[X]
        [END OF LOOP]
Step 5: RETURN Y
```

Example 12.4 Consider the binomial heap given below and see how the procedure works in this case.



Binom Yığınları

- ***İki Binom Yığını Birleştirmek***
- İki binom yığınını birleştirme prosedürü diğer işlemler tarafından alt r olarak kullanılır.
- Link_Binomial–Tree() prosedürü, kökleri aynı dereceye sahip olan binom ağaçlarını birbirine bağlar.
- Y düğümünde kök salmış Bi-1 ağacını, Z düğümünde kök salmış Bi-1 ağacına bağlayan ve Z'yi Y'nin ebeveyni yapan algoritma Şekil 12.17'd gösterilmiştir.
- Link_Binomial–Tree() prosedürü, Y'yi O(1) sürede Z düğümünün çocuklarının bağlı listesinin yeni başı yapar.

```
Link_Binomial-Tree(Y, Z)

Step 1: SET Parent[Y] = Z
Step 2: SET Sibling[Y] = Child[Z]
Step 3: SET Child[Z] = Y
Step 4: Set Degree[Z] = Degree[Z]+ 1
Step 5: END
```

Figure 12.17 Algorithm to link two binomial trees

Binom Yığınları

- ***İki Binom Yığını Birleştirmek***
- İki binom yığını olan H1 ve H2'yi birleştiren algoritma Şekil 12.18'de verilmiştir.

```
Union_Binomial-Heap(H1, H2)

Step 1: SET H = Create_Binomial-Heap()
Step 2: SET Head[H] = Merge_Binomial-Heap(H1, H2)
Step 3: Free the memory occupied by H1 and H2
Step 4: IF Head[H] = NULL, then RETURN H
Step 5: SET PREV = NULL, PTR = Head[H] and NEXT =
        Sibling[PTR]
Step 6: Repeat Step 7 while NEXT ≠ NULL
Step 7:     IF Degree[PTR] ≠ Degree[NEXT] OR
            (Sibling[NEXT] ≠ NULL AND
            Degree[Sibling[NEXT]] = Degree[PTR]), then
                  SET PREV = PTR, PTR = NEXT
            ELSE IF Val[PTR] ≤ Val[NEXT], then
                  SET Sibling[PTR] = Sibling[NEXT]
                  Link_Binomial-Tree(NEXT, PTR)
                  ELSE
                        IF PREV = NULL, then
                           Head[H] = NEXT
                        ELSE
                           Sibling[PREV] = NEXT
                          Link_Binomial-Tree(PTR, NEXT)
                           SET PTR = NEXT
            SET NEXT = Sibling[PTR]
Step 8: RETURN H
```

Figure 12.18 Algorithm to unite two binomial heaps

Binom Yığınları

- ***İki Binom Yığını Birleştirmek***
- Algoritma H1 ve H2 yığınlarının orijinal gösterimlerini yok eder.
- Link_Binomial-Tree()'nin dışında, H1 ve H2'nin kök listelerini, derecey göre monotonik artan bir düzende sıralanmış tek bir bağlı listeye birleştirmek için kullanılan Merge_Binomial-Heap() adlı başka bir prosedürü kullanır.
- Algoritmada, 1. ve 3. Adımlar, H1 ve H2 binom yığınlarının kök listelerini, H1 ve H2'nin artan dereceye göre kesin olarak sıralanacağı şekilde tek bir kök listesi H'de birleştirir.
- Merge_Binomial-Heap(), monotonik artan dereceye göre sıralanmış b kök listesi H döndürür.
- H1 ve H2'nin kök listelerinde m kök varsa, Merge_Binomial-Heap() O(m) sürede çalışır.
- Bu prosedür, iki kök listesinin başlarındaki kökleri tekrar tekrar incele ve daha düşük dereceye sahip kökü çıkış kök listesine eklerken, giriş listesinden kaldırır.

Binom Yığınları

- ***İki Binom Yığını Birleştirmek***
- Algoritmanın 4. adımı, yığın H'de en az bir kök olup olmadığını kontrol eder.
- Algoritma ancak H'nin en az bir kökü varsa devam eder.
- Adım 5'te üç işaretçi başlatıyoruz: Şu anda incelenen kökü işaret eden PTR, kök listesinde PTR'den önceki kökü işaret eden PREV ve kök listesinde PTR'den sonraki kökü işaret eden NEXT.
- 6. Adımda, her yinelemede, derecelerine ve muhtemelen kardeş[NEXT] derecesine bağlı olarak PTR'yi NEXT'e mi yoksa NEXT'i PTR'ye mi bağlayacağımıza karar verdiğimiz bir while döngümüz var.

Binom Yığınları

- ***İki Binom Yığını Birleştirmek***
- 7. Adımda iki koşulu kontrol ediyoruz. İlk olarak, eğer degree[PTR] ≠ degree[NEXT] ise, yani PTR bir Bi ağacının kökü ve NEXT bir Bj ağacının kökü ise, j > i için, PTR ve NEXT birbirine bağlı değildir, ancak işaretçileri listede bir pozisyon daha aşağıya taşırız.
- İkinci olarak, PTR'nin eşit derecedeki üç kökün ilki olup olmadığını kontrol ediyoruz, yani,
  derece[PTR] = derece[SONRAKİ] = derece[Kardeş[SONRAKİ]]
- Bu durumda da PREV= PTR, PTR = NEXT yazarak işaretçileri listede bir pozisyon daha aşağıya taşımamız yeterli olacaktır.
- Ancak, yukarıdaki IF koşulları sağlanmıyorsa, ortaya çıkan durum PTR'nin eşit derecedeki iki kökün ilki olmasıdır, yani,
  derece[PTR] = derece[SONRAKİ] ≠ derece[Kardeş[SONRAKİ]]
- Bu durumda, hangisi daha küçük anahtara sahipse, PTR'yi NEXT'e vey NEXT'i PTR'ye bağlarız.
- Elbette iki düğüm birbirine bağlandıktan sonra daha küçük anahtara s olan düğüm kök olacaktır.

Binom Yığınları

- ***İki Binom Yığını Birleştirmek***
- Union_Binomial–Heap()'in çalışma süresi, n'nin H_1 ve H_2 binom yığınlarındaki toplam düğüm sayısı olduğu O(1og n) olarak verilebilir.
- H_1 n_1 düğümü ve H_2 n_2 düğümü içeriyorsa, H_1 en fazla 1og(n + 1) kök ve H_2 en fazla 1og(n_2 + 1) kök içerir, bu nedenle Merge_Binomial–Heap()'i çağırdığımızda H en fazla (1og n_2 + 1og n_1 + 2) ≤ (2 1og n + 2) = O(1og n) kök içerir.
- $n = n_1 + n_2$ olduğundan, Merge_Binomial–Heap()'in yürütülmesi O(log n) sürer.
- While döngüsünün her yinelemesi O(1) zaman alır ve en faz (1og n_1 + 1og n_2 + 2) yineleme olduğundan toplam zaman O(log n) olur.

Binom Yığınları

- ***İki Binom Yığını Birleştirmek***

Example 12.5 Unite the binomial heaps given below.

Solution

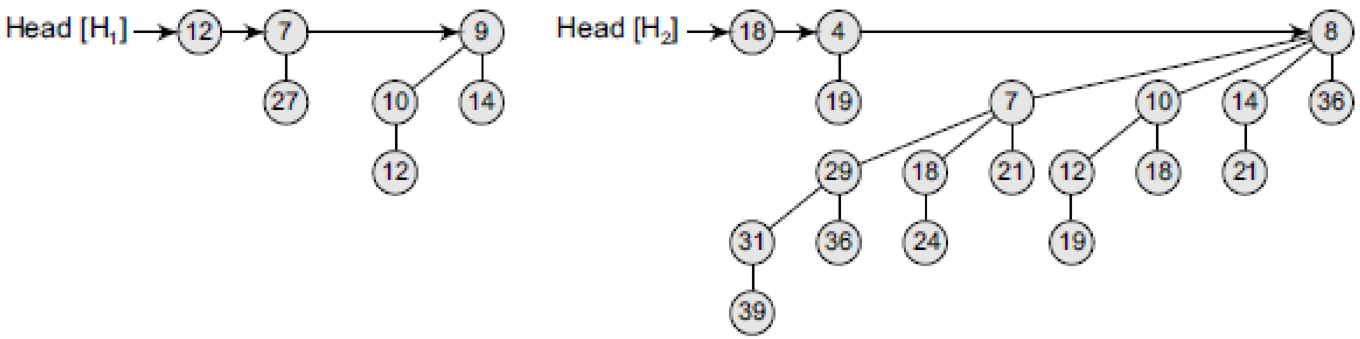
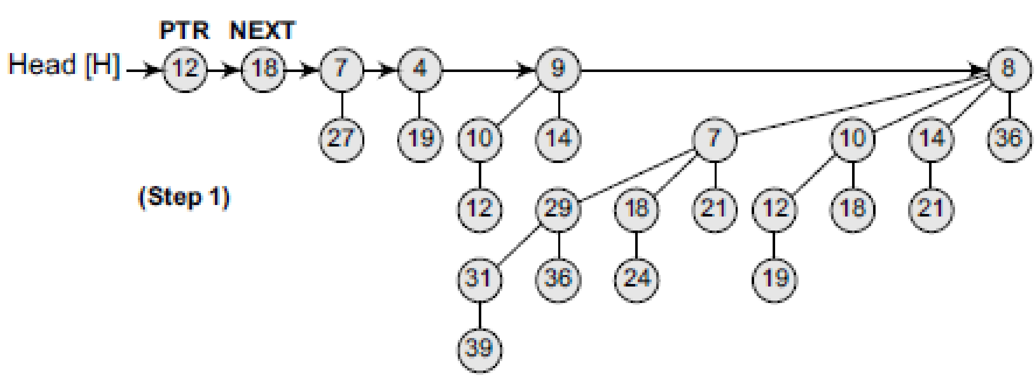


Figure 12.19(a)

After `Merge_Binomial-Heap()`, the resultant heap can be given as follows:



Binom Yığınları

- ***İki Binom Yığını Birleştirmek***

Link NEXT to PTR, making PTR the parent of the node pointed by NEXT.

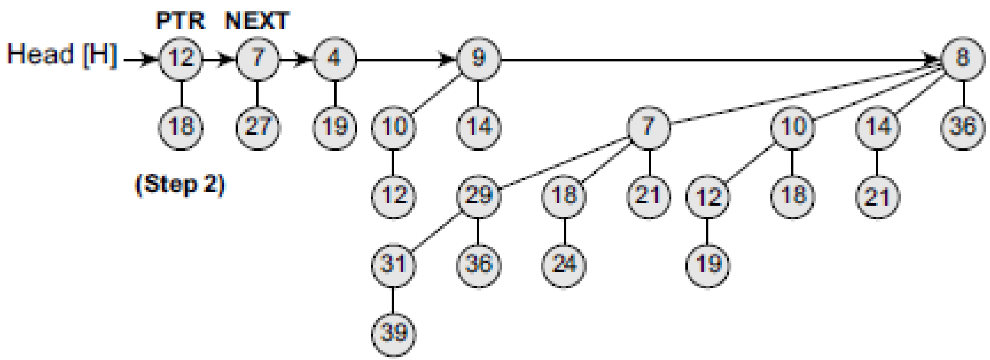


Figure 12.19(c)

Now PTR is the first of the three roots of equal degree, that is, degree[PTR] = degree[NEXT] = degree[sibling[NEXT]]. Therefore, move the pointers one position further down the list by writing PREV = PTR, PTR = NEXT, and NEXT = sibling[PTR].

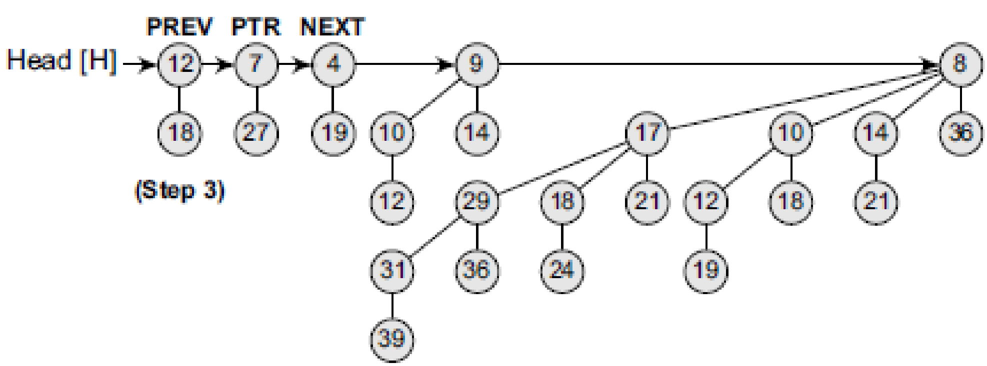


Figure 12.19(d)

Binom Yığınları

- ***İki Binom Yığını Birleştirmek***

Link PTR to NEXT, making NEXT the parent of the node pointed by PTR.

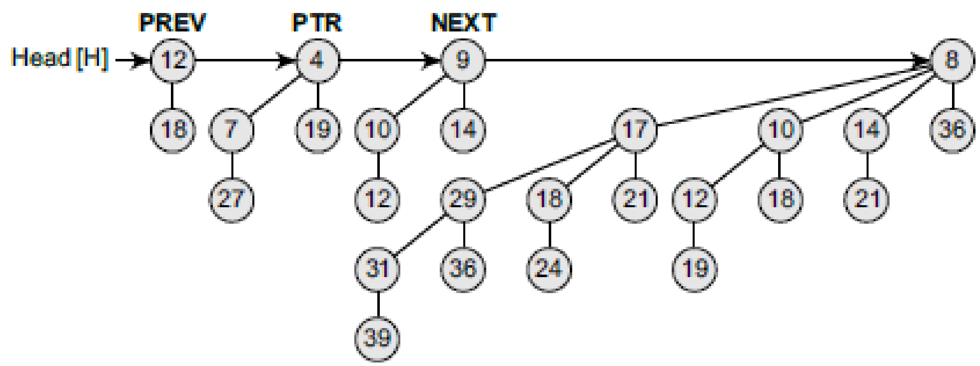


Figure 12.19(e)

Link NEXT to PTR, making PTR the parent of the node pointed by NEXT.

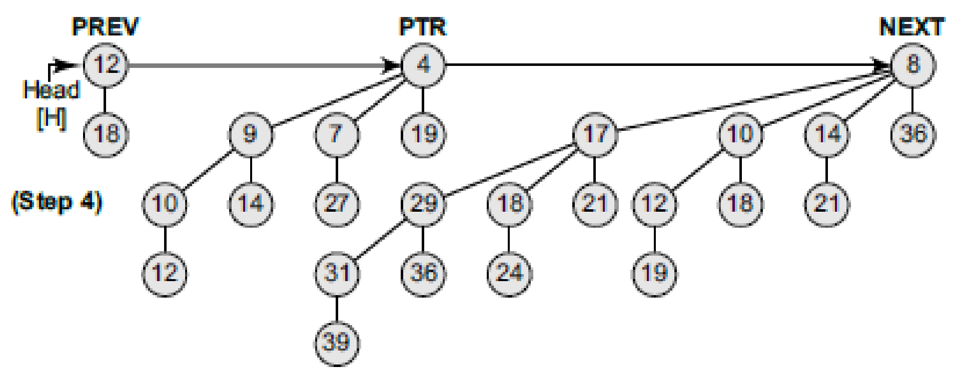


Figure 12.19(f) Binomial heap

Binom Yığınları

- ***Yeni Bir Düğüm Ekleme***
- Insert_Binomial–Heap() prosedürü, x düğümünü binom yığını H'ye eklemek için kullanılır.
- *Bu prosedürün ön koşulu, x'e yer ayrılmış olması ve val[x]'in zaten doldurulmuş olmasıdır.*
- Şekil 12.20'de gösterilen algoritma, basitçe O(1) sürede H' binom yığınını oluşturur.
- H’ yalnızca x adlı bir düğümü içerir.
- Son olarak algoritma, H'yi n-düğümlü binom yığını H ile O(log n) sürede birleştirir.
- H’ tarafından işgal edilen belleğin Union_Binomial–Heap(H, H’) prosedüründe serbest bırakıldığını unutmayın.

```
Insert_Binomial-Heap(H, x)

Step 1: SET H' = Create_Binomial-Heap()
Step 2: SET Parent[x] = NULL, Child[x] = NULL and
        Sibling[x] = NULL, Degree[x] = NULL
Step 3: SET Head[H'] = x
Step 4: SET Head[H] = Union_Binomial-Heap(H, H')
Step 5: END
```

Figure 12.20 Algorithm to insert a new element in a binomial heap

Binom Yığınları

- ***Minimum Anahtarlı Düğümün Çıkarılması***
- H ikili yığınından minimum anahtara sahip düğümü çıkarmak için algoritma Şekil 12.21'de gösterilmiştir.
- Min–Extract_Binomial–Heap prosedürü, parametre olarak bir yığın H kabul eder ve çıkarılan düğüme bir işaretçi döndürür.
- İlk adımda, minimum değere sahip bir kök düğümü olan R bulunur ve H'nin kök listesinden çıkar
- Daha sonra, R'nin çocuklarının sırası tersine çevrilir ve hepsi kök listesine eklenir.
- H'.
- Son olarak, iki yığını birleştirmek için Union_Binomial–Heap (H, H’) çağrılır ve R döndürülür.
- Min–Extract_Binomial–Heap() algoritması O(log n) sürede çalışır; burada n, H'deki düğüm sayıs

```
Min-Extract_Binomial Heap (H)

Step 1: Find the root R having minimum value in
        the root list of H
Step 2: Remove R from the root list of H
Step 3: SET H′ = Create_Binomial-Heap()
Step 4: Reverse the order of R's children thereby
        forming a linked list
Step 5: Set head[H′] to point to the head of the
        resulting list
Step 6: SET H = Union_Binomial-Heap(H, H′)
```

Figure 12.21 Algorithm to extract the node with minimum key from a binomial heap

Binom Yığınları

- ***Minimum Anahtarlı Düğümün Çıkarılması***

Example 12.6 Extract the node with the minimum value from the given binomial heap.

Solution

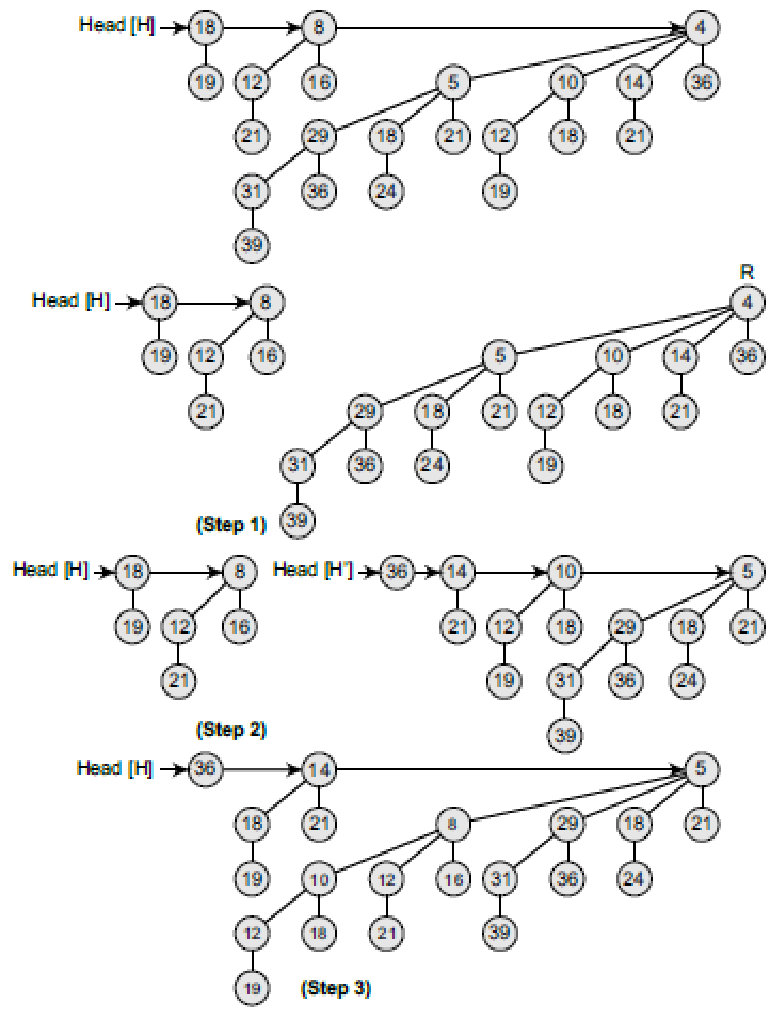


Figure 12.22 Binomial heap

Binom Yığınları

- ***Bir Düğümün Değerini Azaltma***
- Binom yığını H'deki x düğümünün değerini azaltmak için algoritma Şekil 12.23'te verilmiştir.
- Algoritmada düğümün değeri, düğümün geçerli değerinden küçük olan yeni değeri ile üzerine yazılır.
- Algoritmada öncelikle yeni değerin mevcut değerden büyük olmamasını kon ediyoruz ve ardından yeni değeri düğüme atıyoruz.
- Daha sonra PTR'nin başlangıçta x düğümünü işaret ettiği ağaca çıkarız. Whil döngüsünün her yinelemesinde, val[PTR], üst öğesi PAR'ın değeriyle karşılaştırılır.
- Ancak, PTR'lerden biri kök ise veya anahtar[PTR] ≥ anahtar[PAR] ise, o zama iki terimli ağaç yığın sıralıdır.
- Aksi takdirde, düğüm PTR yığın sıralamasını ihlal eder, bu nedenle anahtarı ebeveyninin anahtarıyla değiştirilir. Ağaçta bir seviye yukarı çıkmak ve işlem devam etmek için PTR = PAR ve PAR = Parent[PTR] ayarlarız.
- Binomial–Heap_Decrease_Val prosedürü, x düğümünün maksimum derinliği log n olduğundan O(log n) zaman alır, bu nedenle while döngüsü en fazla log kez yineleme yapacaktır.

Binom Yığınları

- ***Bir Düğümün Değerini Azaltma***

```
Binomial-Heap_Decrease_Val(H, x, k)

Step 1: IF Val[x] < k, then Print " ERROR"
Step 2: SET Val[x] = k
Step 3: SET PTR = x and PAR = Parent[PTR]
Step 4: Repeat while PAR ≠ NULL and Val[PTR] < Val[PAR
Step 5:             SWAP ( Val[PTR], Val[PAR] )
Step 6:             SET PTR = PAR and PAR = Parent [PTR]
        [END OF LOOP]
Step 7: END
```

Figure 12.23 Algorithm to decrease the value of a node x in a binoı heap H

Binom Yığınları

- ***Bir Düğümü Silme***
- Binomial–Heap_Decrease_Val prosedürünü anladığımızda, bir düğüm x'in değerini O(n) sürede binom yığını H'den silmek kolaylaşır.
- Algoritmaya başlamak için x değerini –∞ olarak ayarlıyoruz. Yığında –∞'dan küçük değere sahip düğüm olmadığını varsayarak, binom yığınından bir düğümü silmek için algoritma Şekil 12.24'te gösterildiği gibi verilebilir.
- Binom–Yığın_Silme–Düğüm prosedürü, x değerini –∞ değerine ayarlar; bu, tüm binom yığınındaki benzersiz bir minimum değerdir.
- Binomial–Heap_Decrease_Val algoritması bu anahtarı bir köke kadar kabarcıklandırı daha sonra bu kök, Min–Extract_Binomial–Heap prosedürüne bir çağrı yapılarak yığından kaldırılır.
- Binom–Yığın_Silme–Düğüm prosedürü O(log n) zaman alır.

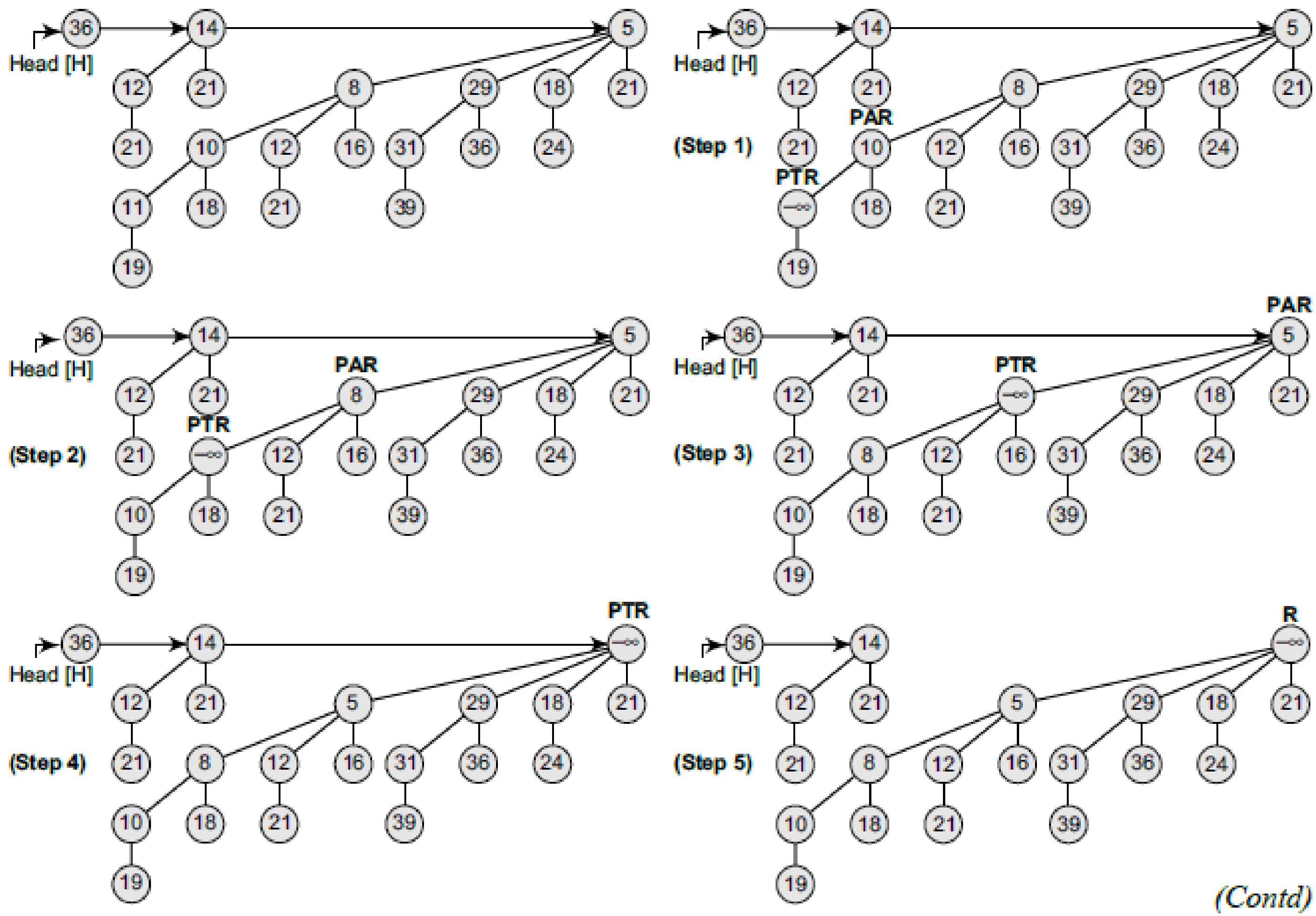
```
Binomial-Heap_Delete-Node(H, x)

Step 1: Binomial-Heap_Decrease_Val(H, x, -∞)
Step 2: Min-Extract_Binomial-Heap(H)
Step 3: END
```

Figure 12.24 Algorithm to delete a node from a bionomial heap

Binom Yığınları

Example 12.7 Delete the node with the value 11 from the binomial heap H.

Solution



(Contd)

Binom Yığınları

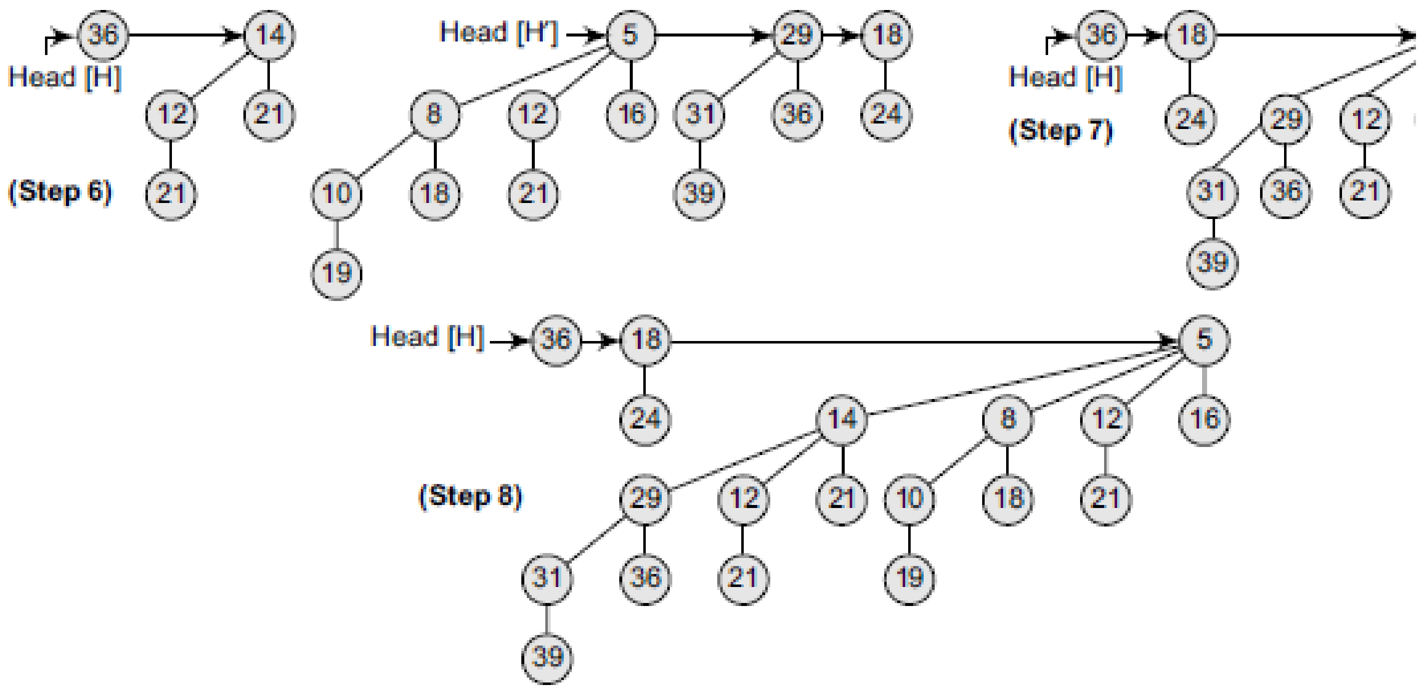


Figure 12.25 (Contd) Binomial heap