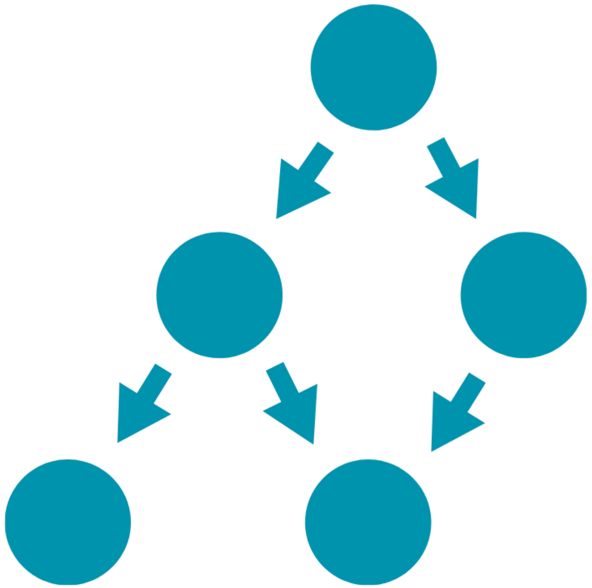


VERİ YAPILARILARI VE ALGORİTMALAR

Disjoint sets

Giriş

1. Ayrık Kümeler
2. Eşdeğerlik İlişkileri ve Eşdeğerlik Sınıfları
3. Ayrık Setler Soyut Veri Türü
4. Uygulamalar
5. Zaman Karmaşıklığı

Giriş

- Bu bölümde matematikte önemli bir kavramı işaret eden kümeler konusuna değineceğiz.
- Bu anlamı herhangi bir sıraya ihtiyaç duymayan bir gurup elemanın nasıl temsil edildiğidir.
- Ayrık küme soyut veri türü bu amaç için kullanılır.

Giriş

- Ayrık setler **(disjoint sets)** denklik **(equivalence)** problemini çözmek için kullanılır.

- Uygulaması **(implementation)** oldukça basittir.

- Uygulama için dizi ya da sözlük kullanılabilir ve her bir işlev birkaç satır kod ile tanımlanabilir.

- **Kruskal's minimum spanning tree** algoritması gibi pek çok farklı algoritma ayrık setleri yardımcı bir veri yapısı olarak kullanır.

Eşdeğerlik/Denklik İlişkileri ve Eşdeğerlik/Denklik Sınıfları

- Bir ayrık set S, kümelerin bir koleksiyonudur.

- $S_1, \ldots\ldots S_n$ burada $\forall_{i \neq j} S_i \cap S_j = \phi$

- Her setin, setin üyesi olan bir temsilcisi vardır (Öğeler karşılaştırılabilirse genellikle minimumdur)

Eşdeğerlik/Denklik İlişkileri ve Eşdeğerlik/Denklik Sınıfları

- S elemanları içeren bir küme ve R'de bir ilişki olarak bu küme üzerinde tanımlanasın.

- Bu şu anlama gelir:

$$a, b \in S, a\, R\, b$$

- Tanım ya doğrudur ya da yanlıştır.

Eşdeğerlik İlişkileri ve Eşdeğerlik Sınıfları

- Eğer $a\ R\ b$ doğru ise a'nın b ile ilişkilendirilir; aksi durumda a'nın b ile ilişkisi yoktur.

- Eğer aşağıdaki koşullar geçerli ise; A ilişki R eşdeğerlik ilişkisi olarak tanımlanır.
  - Dönüşlü (`Reflexive`):
    - Her öğe için $a \in S, a\ R\ a\ doğru\ ise.$
  - Simetrik (`Symmetric`):
    - İki öğe için $a, b \in S, eğer\ \boldsymbol{a\ R\ b}\ doğru\ ise\ \boldsymbol{b\ R\ a}\ doğrudur.$
  - Geçişli (`Transitive`):
    - Üç öğe için $a, b, c \in S, eğer\ \boldsymbol{a\ R\ b}\ ve\ \boldsymbol{b\ R\ c}\ doğru\ ise\ \boldsymbol{a\ R\ c}\ doğrudur.$

Eşdeğerlik İlişkileri ve Eşdeğerlik Sınıfları

- Bir örnek olarak, bir tam sayı kümesi üzerinde küçük eşit ($\leq$) ve büyük eşit ($\geq$) ilişkileri eşdeğer ilişkiler **değildir**.

- Bunlar dönüşlüdür.
  - (çünkü $a \leq a$)
- Geçişlidir.
  - ($a \leq b \; ve \; b \leq c, a \leq c \; anlamına \; gelir$)
- Simetrik değildir!
  - ($a \leq b, b \leq a$ anlamına gelmez).

Eşdeğerlik İlişkileri ve Eşdeğerlik Sınıfları

- Benzer şekilde demiryolu bağlantısı (`rail connectivity`) bir eşdeğer ilişkidir.

- Dönüşlüdür
  - Çünkü her bölge kendine bağlanır.
- Simetriktik
  - Eğer a şehrinden b şehrine bağlantı var ise; b şehrinden de a şehrine bağlantı vardır.
- Geçişlidir
  - Eğer a şehri b şehrine ve b şehri de c değerine bir bağlantıya sahipse; a şehri aynı zamanda c şehrine bağlıdır.

Eşdeğerlik İlişkileri ve Eşdeğerlik Sınıfları

- Bir $a \in S$'in eşdeğerlik sınıfı, a ile ilişkili olan tüm elemanları içeren S'nin bir altkümesidir.

- Eşdeğerlik sınıfı, S'nın bir parçasını oluşturur.

- S'nin tüm üyeleri tam olarak bir eşdeğerlik sınıfında görülür.

- $a\ R\ b$ ilişkisine karar vermek için, a ve b'nin aynı eşdeğerlik sınıfında olup olmadığının kontrol edilmesi gerekir.

Eşdeğerlik İlişkileri ve Eşdeğerlik Sınıfları

- Bir önceki demiryolu bağlantısı örneğinde, a ve b şehirleri eğer demiryolu ile birbirine bağlanmış ise aynı eşdeğerlik sınıfında yer alırlar. Eğer bağlantıları yok ise farklı eşdeğerlik sınıflarının parçalarıdır. Bu nedenle kesişimleri ϕ boştur.

- Eşdeğerlik sınıfları ayrık kümeler (disjoint sets) olarak da tanımlanır. Olası işlevler aşağıdaki gibi tanımlanabilir:
  - Eşdeğerlik sınıfı oluşturma (MAKESET)
  - Eşdeğerlik sınıf adı bulma (FIND)
  - Eşdeğerlik sınıflarını birleştirme (UNION)

Ayrık Setler Soyut Veri Türü

- MAKESET(X)
  - Tek X elemanından oluşan yeni bir küme oluşturma.
  - O(1)

- UNION(X,Y)
  - X ve Y elemanlarının birleşiminden oluşan yeni bir küme oluşturma ve X ve Y elemanlarını içeren kümeleri silme.

    $S_x, S_y \qquad S_x \cup S_y$

  - O(1)

- FIND(X)
  - X elemanını içeren kümenin adını dönme.
  - O(h)

Uygulamalar (Applications)

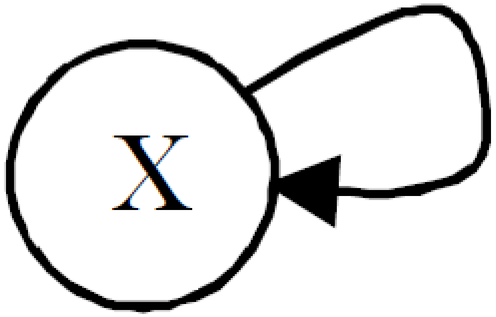
- Ağ bağlantılarını temsil etme
- İmge işleme
- En az ortak atayı bulma
- Sonlu durum otomataların (finite-state automata) eşdeğerliğini tanımlama
- Kruskal’s minimum spanning tree algoritması
- Oyun algoritmaları

Ayrık Kümeler Soyut Veri Yapısının Olası Uygulama Yaklaşımları

- Fast FIND (Quick FIND)
- Fast UNION (Quick Union)

MAKESET(X)

```
p(x)=x
rank(x)=0
```



Fast FIND (Quick Find)

- Bu metotta dizi kullanılır.
- Dizi her öğenin küme adını (set name) içerir.
- O(1) karmaşıklığa sahiptir, çünkü set name özelliğine dizi göz numarası verilerek erişim sağlanabilir.

Set Name

3	5	…	2	3
0	1	…	n-2	n-1

FIND(X)

- Find(x) – X değerini içeren kümenin adını döner.
  - {3,5,7,1,6}, {4,2,8}, {9},
  - Find(1) = 5
  - Find(4) = 8
  - Find(9) = ?

FIND(X)

S

{1,2,7,8,9,13,19}

{3}

{4}

{5}

{6}

{10}

{11,17}

{12}

{14,20,26,27}

{15,16,21}

.

.

{22,23,24,29,39,32
33,34,35,36}

Find(8) = 7

Find(14) = 20

Union(7,20)

S

{1,2,7,8,9,13,19,14,20 26,27}

{3}

{4}

{5}

{6}

{10}

{11,17}

{12}

{15,16,21}

.

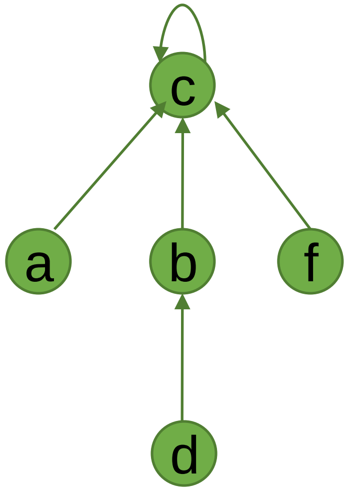
.

{22,23,24,29,39,32
33,34,35,36}

Fast FIND (Quick Find)

- **FIND_SET(d)**

```
  if d != p[d]
    p[d]= FIND_SET(p[d])
 return p[d]
```



Fast FIND (Quick FIND)

- Bu sunumda, UNION(a,b) (a küme i ve b küme j'de varsayımı ile) dizinin tamamının taranması gerekir i'ler j olarak değiştirilir.

- Maliyeti $O(n)$'dir. En kötü durumda $n-1$ birleşim $O(n^2)$'dir.

- Eğer $O(n^2)$ FIND işlevleri var ise, bu performans iyidir; her UNION ve FIND işlevi için ortalama zaman karmaşıklığı $O(1)$'dir.

UNION

- Bir dizi ikili ayrık set:
- {3,5,7} , {4,2,8}, {9}, {1,6}

- Her bir set benzersiz bir ada sahip:
- {3,<u>5</u>,7} , {4,2,<u>8</u>}, {<u>9</u>}, {<u>1</u>,6}

- UNION(x,y)
  - {3,<u>5</u>,7} , {4,2,<u>8</u>}, {<u>9</u>}, {<u>1</u>,6}
  - Union(5,1)

  {3,<u>5</u>,7,1,6}, {4,2,<u>8</u>}, {<u>9</u>},

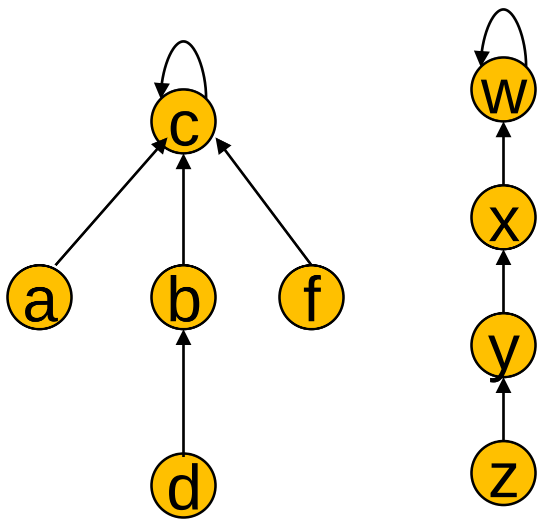
UNION

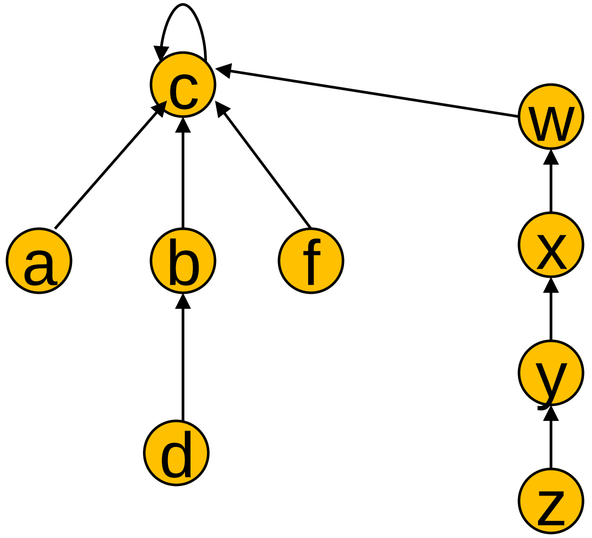
- **`UNION(x,y)`**

```
link(findSet(x),
        findSet(y))
```

- **`link(x,y)`**

```
if rank(x)>rank(y)
 then p(y)=x
 else
  p(x)=y
  if rank(x)=rank(y)
   then rank(y)++
```



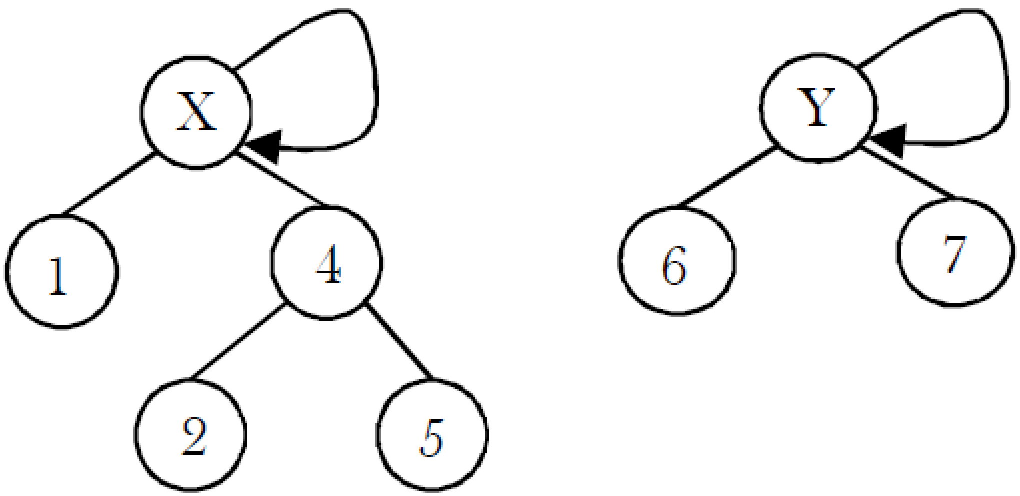


Fast UNION Uygulaması (Quick UNION)

- UNION işlevinin varyantları olabilir:
- Fast UNION (Slow FIND)
- Fast UNION (Quick FIND)
- Fast UNION (path compression)

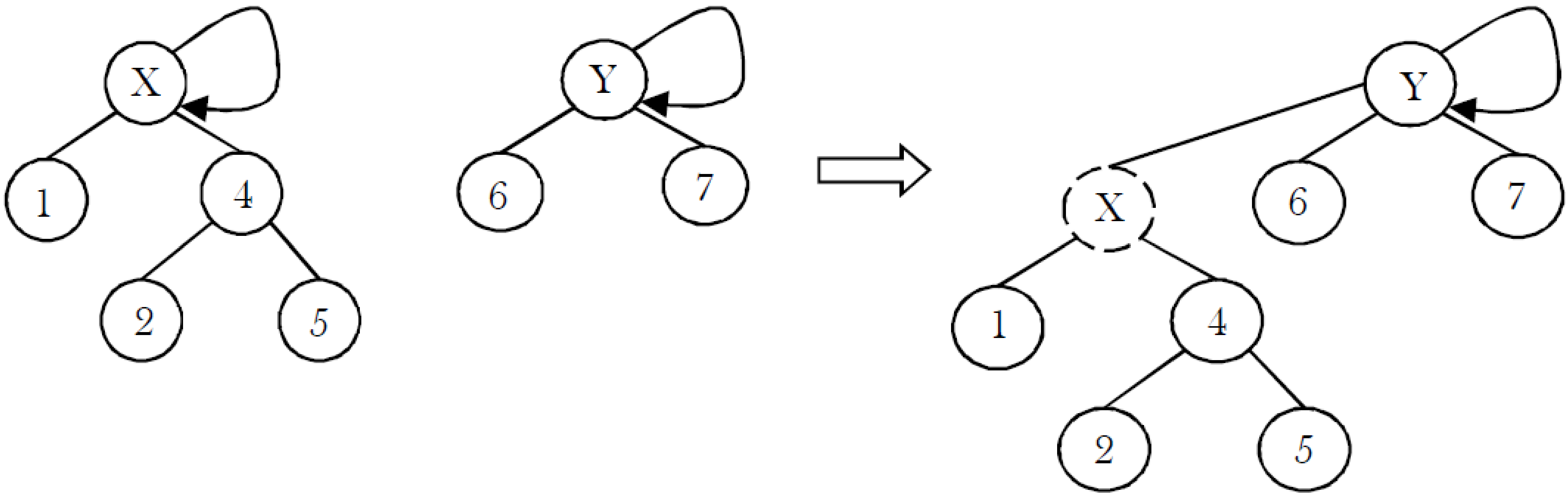
Fast UNION Uygulaması (Slow FIND)

- UNION(X,Y)



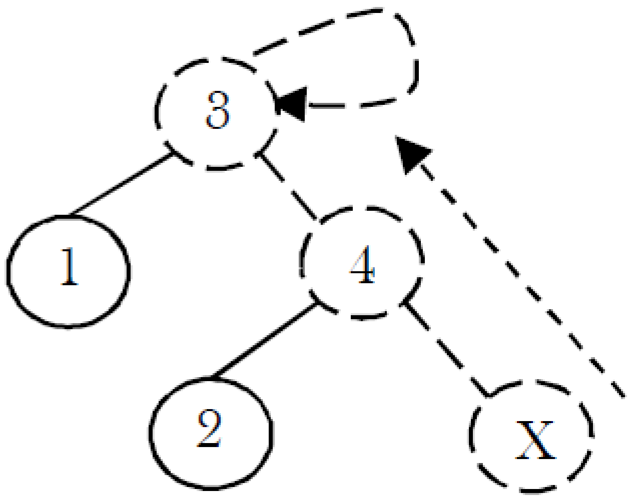
Fast UNION Uygulaması (Slow FIND)

- UNION(X,Y)

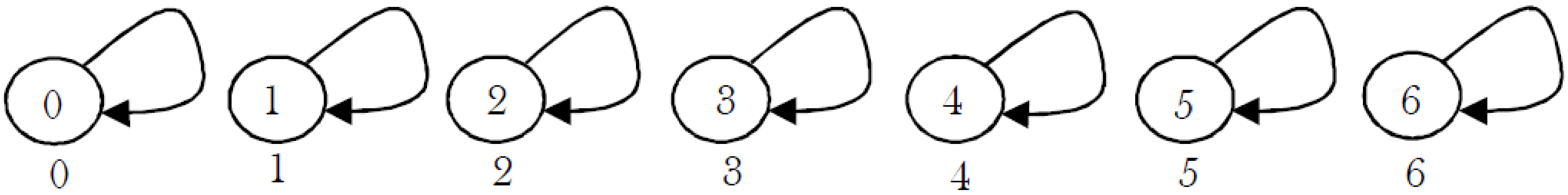


Fast UNION Uygulaması (Slow FIND)

- FIND(X): X elemanını içeren kümenin adını döner. Ağacın köküne gelinceye kadar X küme adı (set name) aranır.



Fast UNION Uygulaması (Slow FIND)



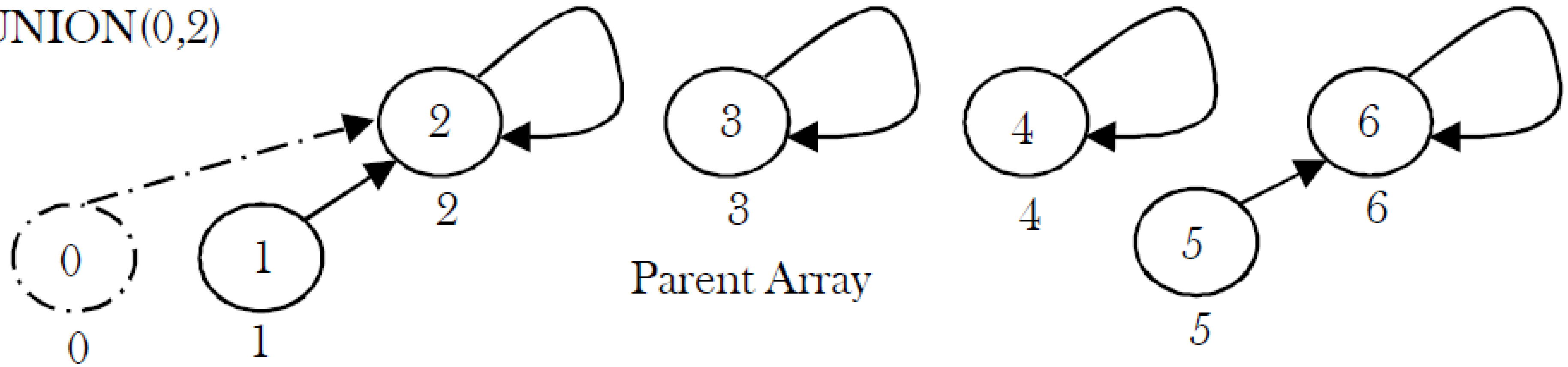
Parent Array

Fast UNION Uygulaması (Slow FIND)

UNION(5,6)

UNION(1,2)

UNION(0,2)

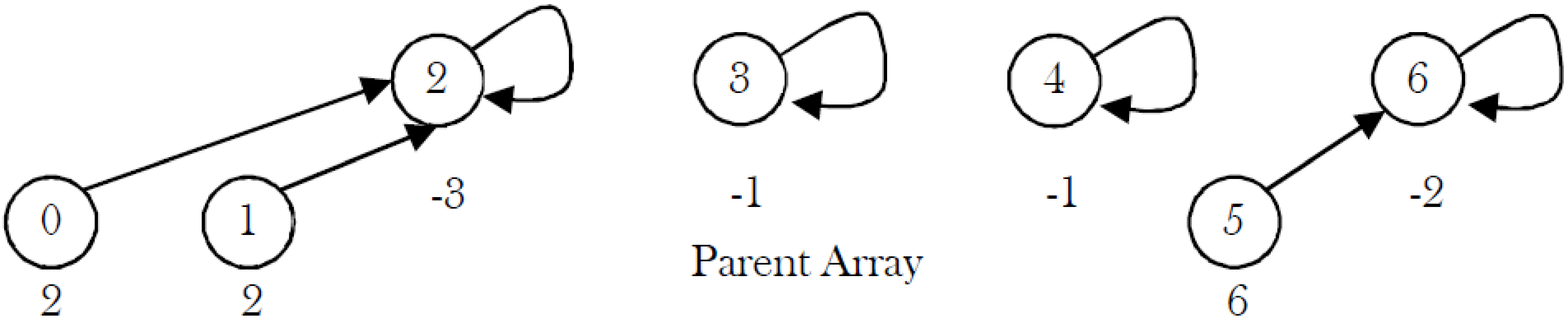


UNION işlevi sadece kökün ebeveynini değiştirir; kümedeki tüm elemanları değiştirmez. En kötü durumda; eğri ağaç için (skew tree) $O(n)$ karmaşıklığı söz konusu olabilir. Ortalama durumunda ağaç yüksekliği dikkate alınır.

Fast UNION Uygulaması (Quick FIND)

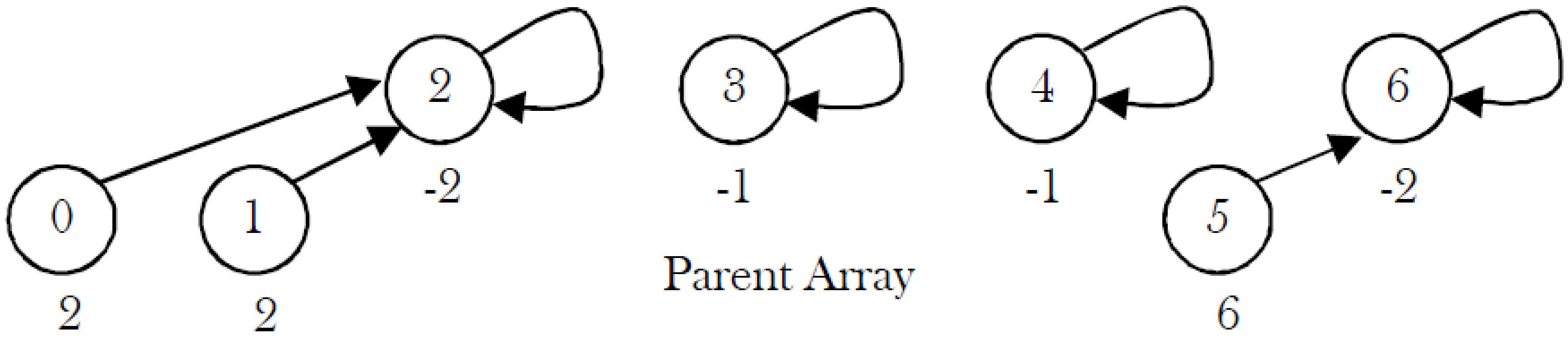
- Slow Find yaklaşımının temel problemi; eğri ağaçlar için arama maliyeti $\boldsymbol{O(n)}$ olarak elde edilir. Bunu geliştirmek için iki yol söz konusudur:
  - **UNION by Size (UNION by Weight)**
    - Daha küçük olan ağacı daha büyük ağacın alt-ağacı yapar.
  - **UNION by Height (UNION by Rank)**
    - Daha az yüksekliğe sahip olan ağacı, daha büyük yüksekliğe sahip olan ağacın alt-ağacı yapar.

Fast UNION Uygulaması (UNION by Size)



- 0 düğümü -> (parent) 2 (Atası 2)
- 1 düğümü -> (parent) 2 (Atası 2)
- 2 düğümü -> (size) -3 (Bu alt ağaç 3 düğüm içeriyor)
- 3 düğümü -> -1 (Bu alt ağaç 1 düğüm içeriyor)
- 4 düğümü -> -1 (Bu alt ağaç 1 düğüm içeriyor)
- 5 düğümü -> (parent) 6 (Atası 6)
- 6 düğümü -> (size) -2 (Bu alt ağaç 2 düğüm içeriyor)

Fast UNION Uygulaması (UNION by Height)

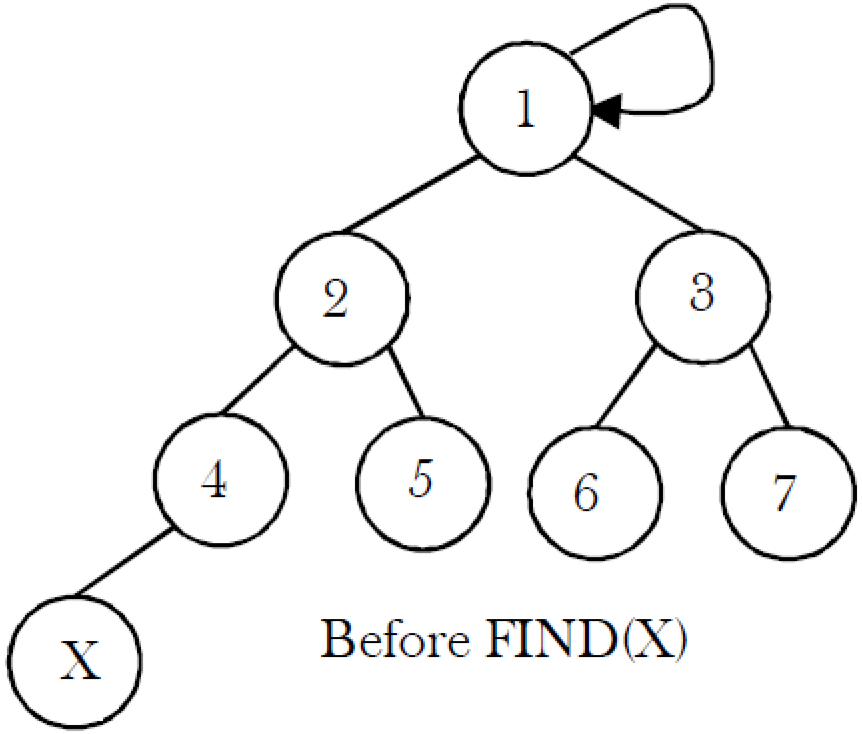


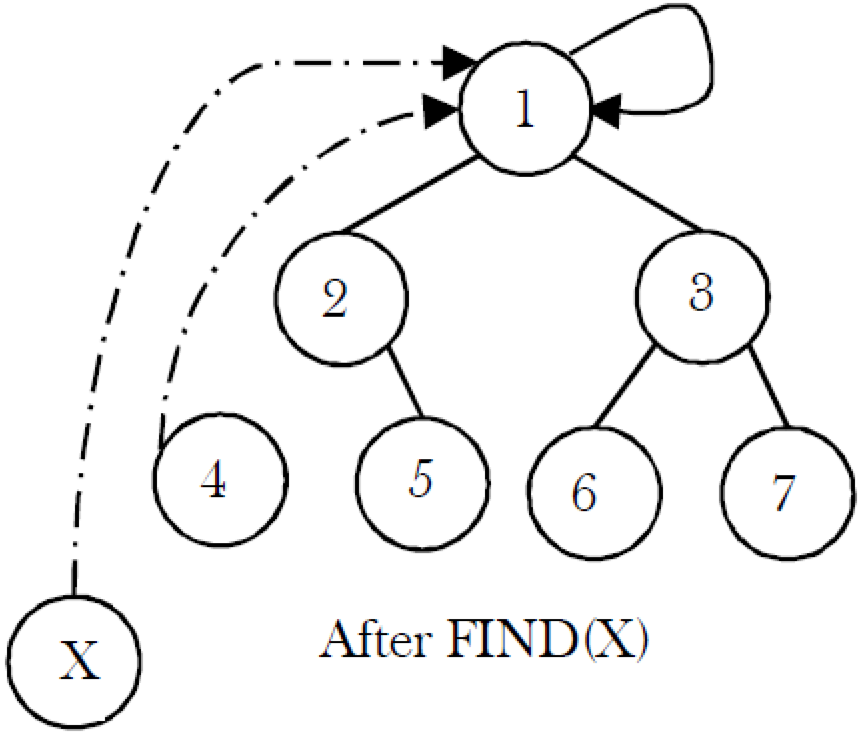
- 0 düğümü -> (parent) 2 (Atası 2)
- 1 düğümü -> (parent) 2 (Atası 2)
- 2 düğümü -> (size) -2 (Yüksekliğin negatifi depolanıyor: -2)
- 3 düğümü -> -1 (Yüksekliğin negatifi depolanıyor: -1)
- 4 düğümü -> -1 (Yüksekliğin negatifi depolanıyor: -1)
- 5 düğümü -> (parent) 6 (Atası 6)
- 6 düğümü -> (size) -2 (Yüksekliğin negatifi depolanıyor: -2)

Fast UNION Uygulaması (Path Compression)

- FIND işlevi kök yolundaki bir düğüm listesini gezer.
- FIND işlevi her düğümün kökü işaret etmesini sağlayarak iyileştirilir.
- Bu işlev **path compression** olarak ifade edilir.
- Path compression UNION by Size ile uygulanabilir ancak; UNION by Height ile verimli bir şekilde uygulanması mümkün değildir.

Fast UNION Uygulaması (Path Compression)



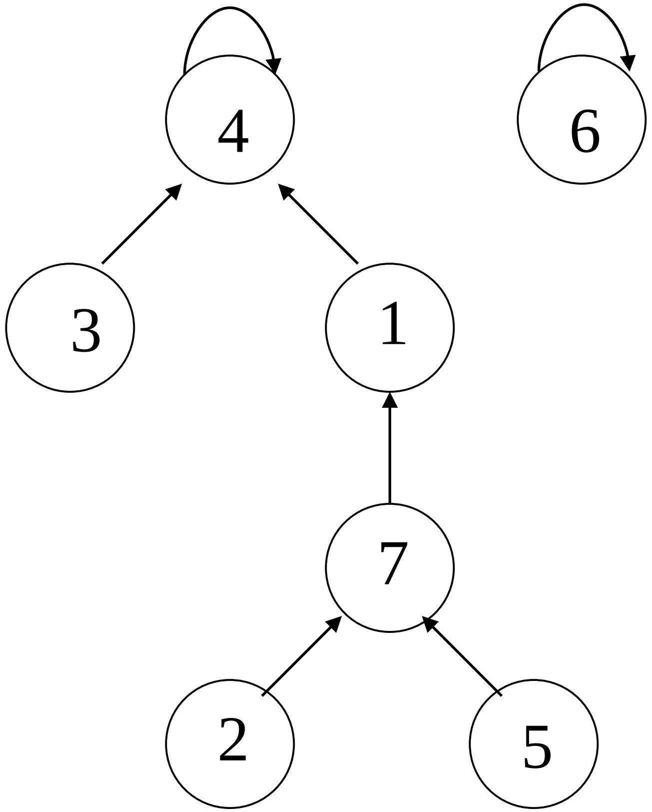


Örnek

- Executing find(5)

  7 →1→ 4→ 4

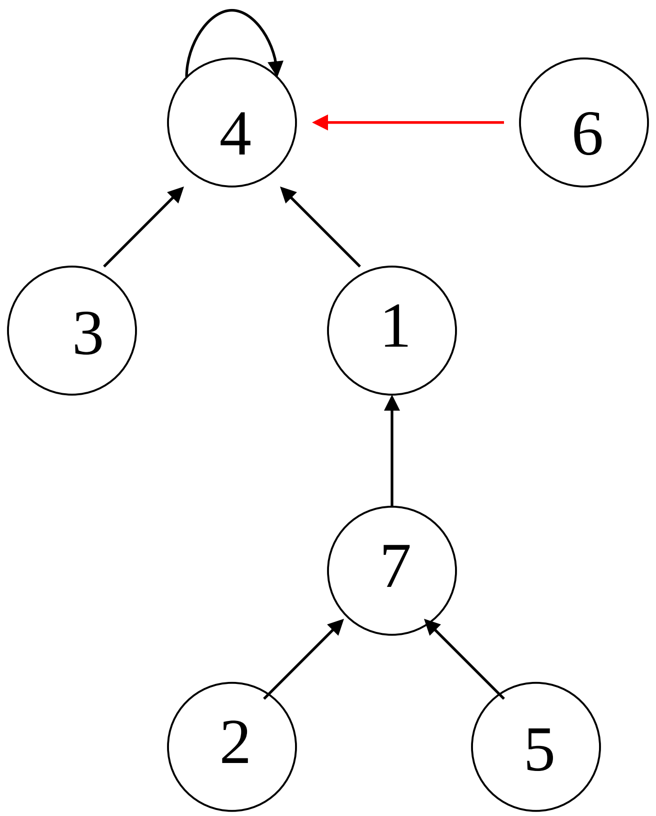
	1	2	3	4	5	6	..	N
Parent	4	7	4	4	7	6		
min				1		6		



Örnek

- Executing union(4,6)

	1	2	3	4	5	6	..	N
Parent	4	7	4	4	7	4		
min				1		1		



Örnek

- Dizi indisleri (Up[i] i'nin ebeveynini temsil eder.)

	1	2	3	4	5	6	7
up	0	1	0	7	7	5	0

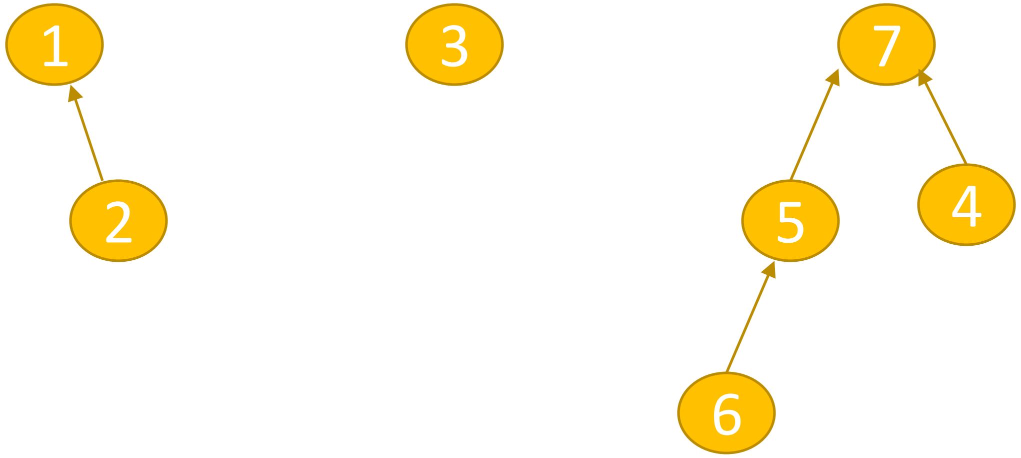
Up [x] = 0 ise x bir köktür.

Örnek

- Dizi indisleri (Up[i] i’nin ebeveynini temsil eder.)

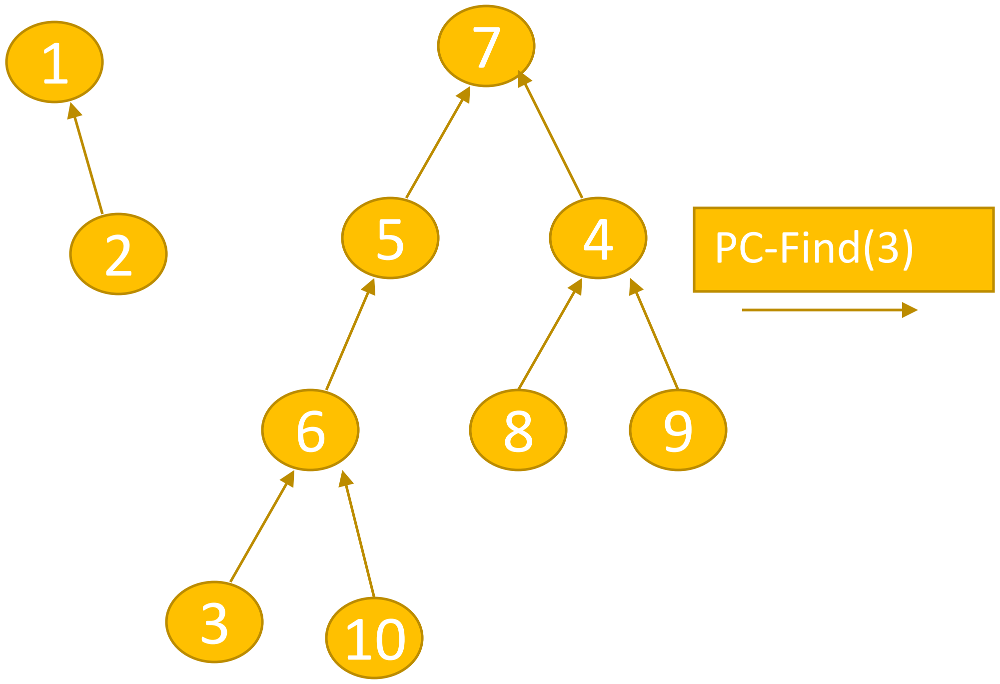
	1	2	3	4	5	6	7
up	0	1	0	7	7	5	0

Up [x] = 0 ise x bir köktür.



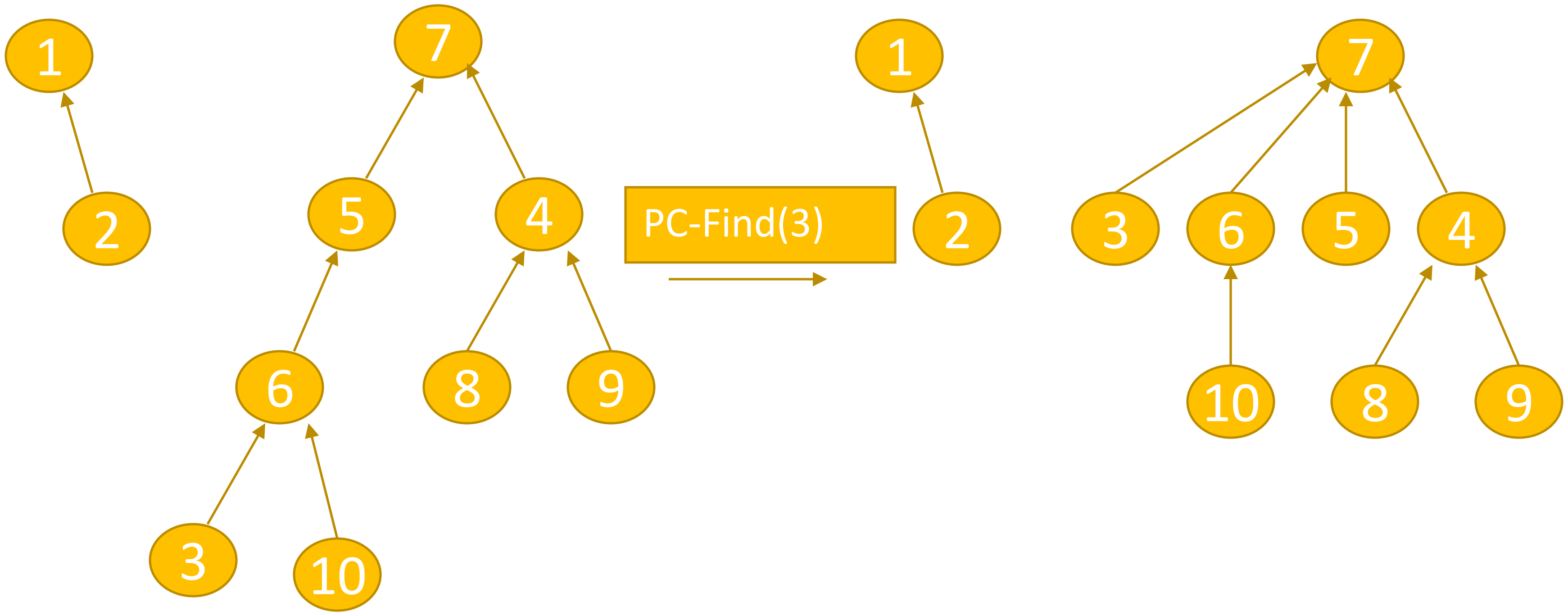
Örnek : Path Compression

- Bir FIND işlevinde, arama yolundaki tüm düğümleri doğrudan köke yönlendirin.



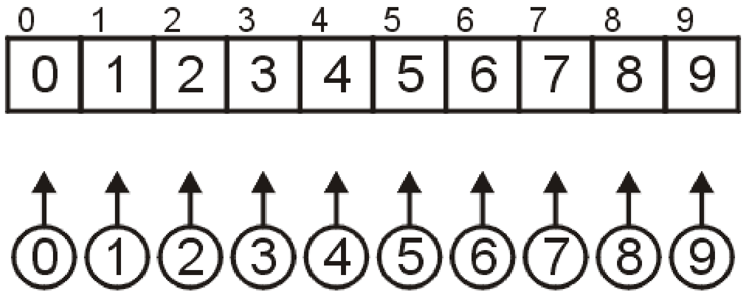
Örnek : Path Compression

- Bir FIND işlevinde, arama yolundaki tüm düğümleri doğrudan köke yönlendirin.



Örnek

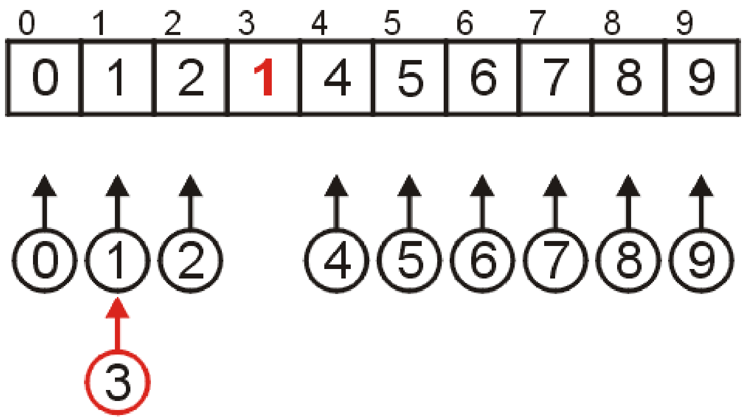
10 rakamdan oluşan ayrık bir set:



{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}

Örnek

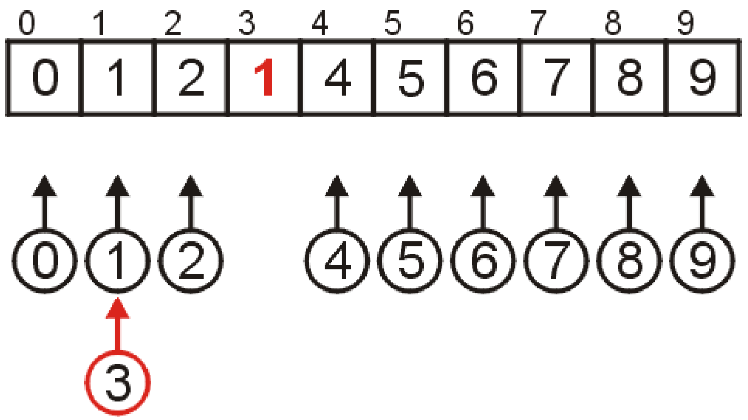
UNION(1,3)
Önce her iki set bulunur ve daha sonra güncelleme yapılır.



$\{0\}, \{1, 3\}, \{2\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}$

Örnek

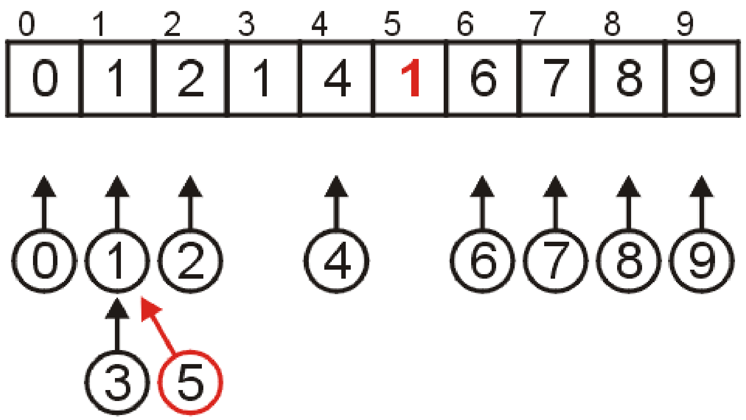
FIND(1) ve FIND(3)
Her ikisi de 1 dönecektir.



$\{0\}, \{1, 3\}, \{2\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}$

Örnek

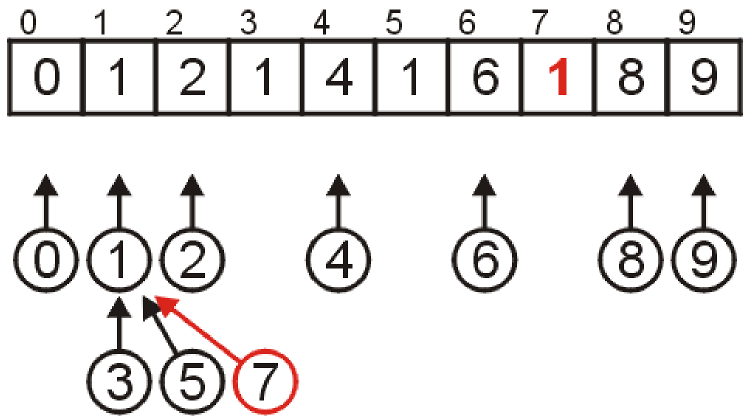
UNION(3,5)
Her iki varlık bulunacak ve daha sonra güncelleme yapılacaktır.



{0}, {1, 3, 5}, {2}, {4}, {6}, {7}, {8}, {9}

Örnek

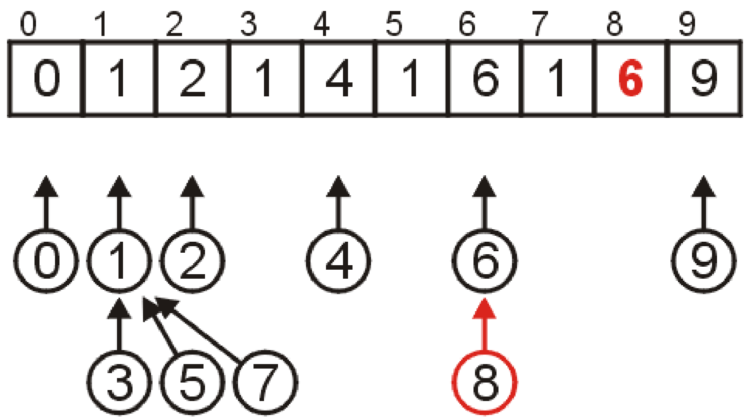
UNION(5,7)
Her iki varlık bulunacak ve daha sonra güncelleme yapılacaktır.



{0}, {1, 3, 5, 7}, {2}, {4}, {6}, {8}, {9}

Örnek

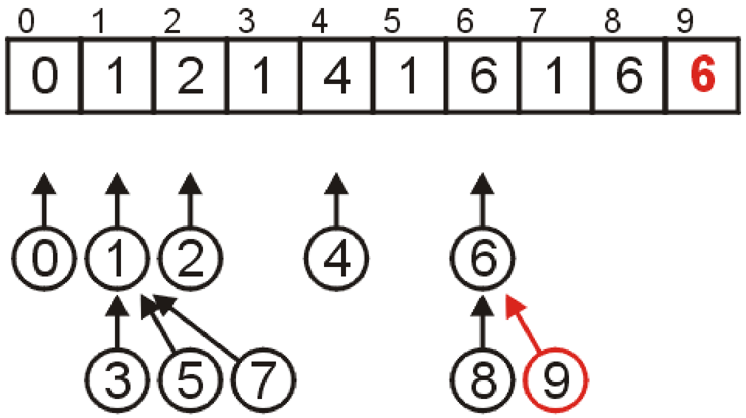
UNION(6,8)
Her iki varlık bulunacak ve daha sonra güncelleme yapılacaktır.



{0}, {1, 3, 5, 7}, {2}, {4}, {6, 8}, {9}

Örnek

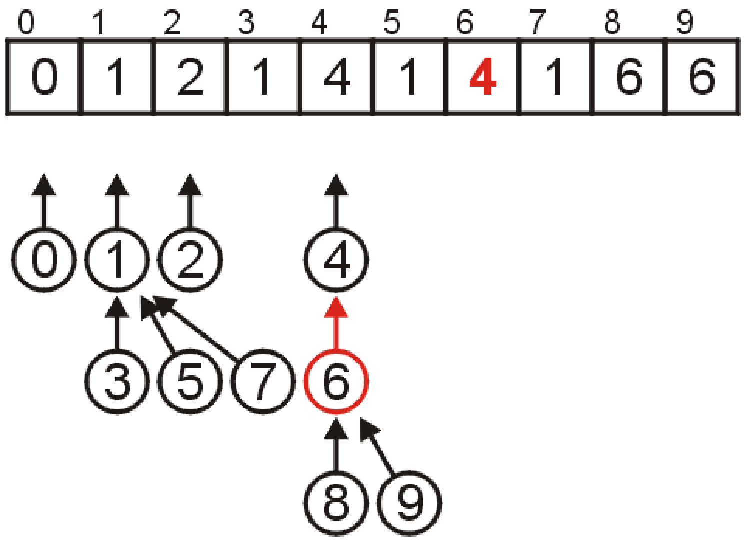
UNION(8,9)
Her iki varlık bulunacak ve daha sonra güncelleme yapılacaktır.



{0}, {1, 3, 5, 7}, {2}, {4}, {6, 8, 9}

Örnek

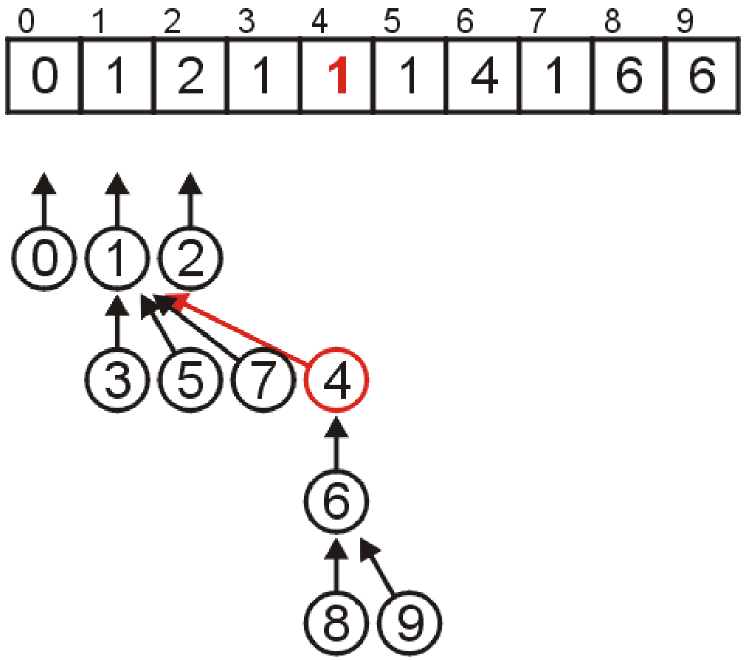
UNION(4,8)
Her iki varlık bulunacak ve daha sonra güncelleme yapılacaktır.



$\{0\}, \{1, 3, 5, 7\}, \{2\}, \{4, 6, 8, 9\}$

Örnek

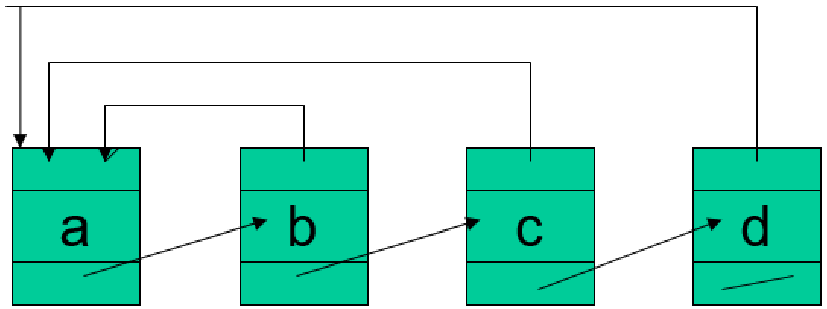
UNION(5,6)
Her iki varlık bulunacak ve daha sonra güncelleme yapılacaktır.



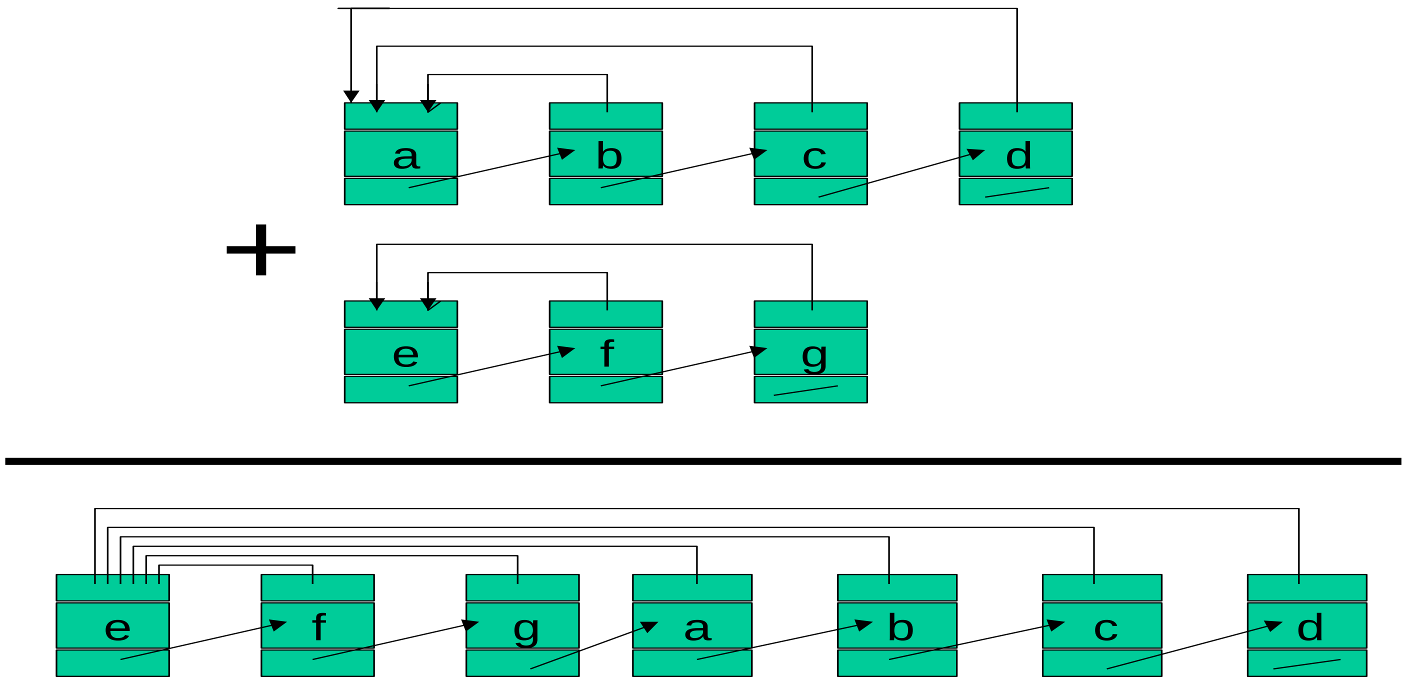
{0}, {1, 3, 4, 5, 6, 7, 8, 9}, {2}

Linked List Uygulaması

- Bir dizi bağlantılı liste tutuyoruz, her liste tek bir kümeye karşılık geliyor.
- Tüm elemenlar temsilci olan ilk elemana işaret ediyor.
- Kuyruk için bir işaretçi tutulur, böylece öğeler listenin sonuna eklenir.



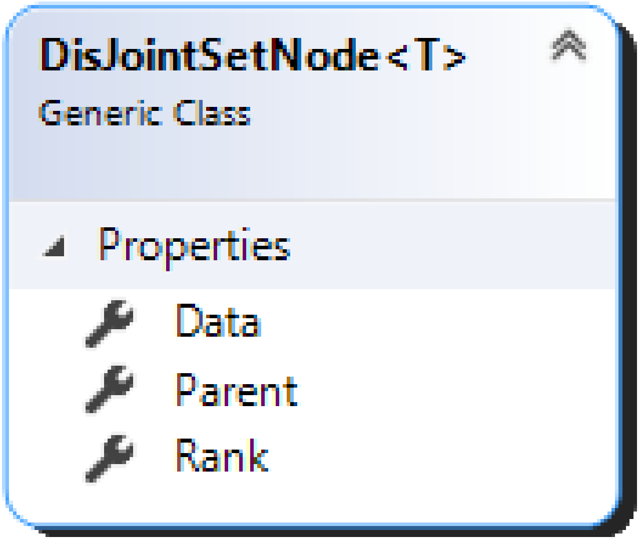
Bağlı Liste ile Birleşim (Union)



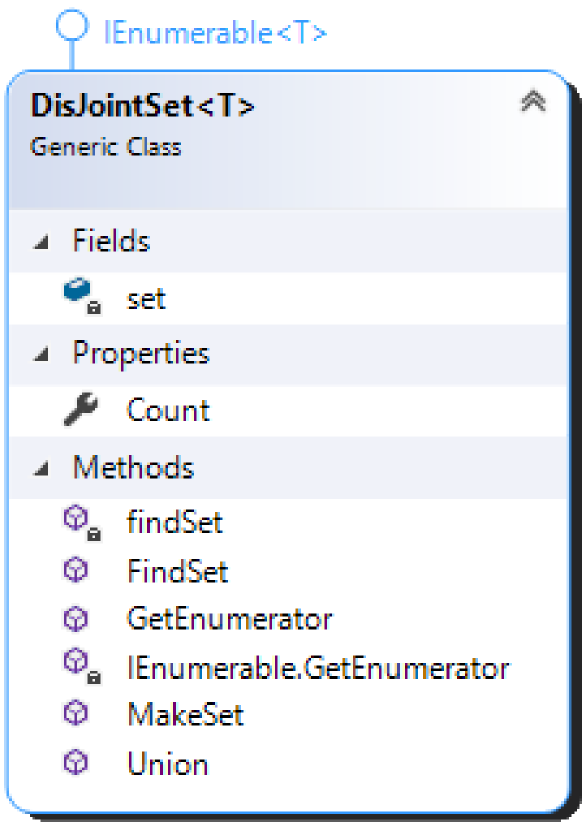
DisJointSetNode<T>

- Bir **`DisJointSetNode<T>`** tasarımı yapılırken:

- Veri (Data)
- Ebeveyn (Parent) * (işaretçi)
- Yükseklik (Height/Rank)



DisJointSet<T>



Zaman Karmaşıklığı

- m union-find işlevinin n boyutlu bir kümede uygulanmasının maliyeti:

Algorithm	Worst-case time
Quick-find	mn
Quick-union	mn
Quick-Union by Size/Height	$n + m \, logn$
Path compression	$n + m \, logn$
Quick-Union by Size/Height + Path Compression	$(m + n) \, logn$

Veri Yapıları ve Algoritmalar

ZAFER CÖMERT
Öğretim Üyesi