Bölüm 16: Kümeler ve Union-Find

C ile Veri Yapıları ve Algoritmalar

Ayrık kümeler | Birleşim ve küme bulma | ebeveyn dizisi | Kötü durum $O(n^2)$

Ağırlıklı kural | Azaltma kuralı | Ters Ackermann | Bileşen · Kruskal

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Modellemek

İkili ayrık kümeleri çocuktan ebeveyne bağlı ağaçlarla göstermek.

Gerçeklemek

Tek bir `ata` dizisiyle `Kume_Bul` ve `Birlesim` yordamlarını yazmak.

Kötü durumu bulmak

Hangi işlem dizisinin ağacı dejenere ettiğini ve $O(n^2)$ maliyeti göstermek.

İyileştirmek

Ağırlıklı kural ve azaltma (collapsing) kuralını uygulamak, lemmaları yorumlamak.

Analiz etmek

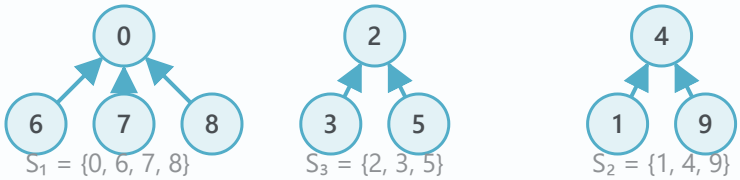
Tarjan–van Leeuwen sonucunu ve ters Ackermann fonksiyonunu açıklamak.

Kullanmak

Bağlantılı bileşen sayma ve Kruskal algoritmasında union-find'ı uygulamak.

Kümelerde birleşim (union) ve bulma (find)

Evren {0, 1, …, n−1}; kümeler **ikili ayrık** (pairwise disjoint): $S_1 \cap S_2 = S_2 \cap S_3 = S_1 \cap S_3 = \emptyset$. n = 10 için:



Her küme bir ağaçtır; bağlar **çocuklardan atalara** doğrudur (oklar yukarı).

Hocadan *"Çok çok temel bir veri yapısıdır, ama birçok problemde kullanırız. Öğrenilmeli ki kullanılabilsin."*

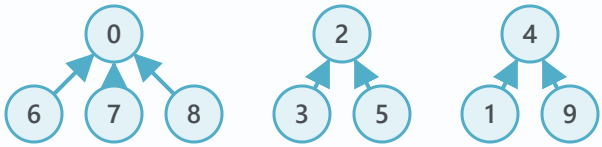
İki işlem ve birleşimin iki biçimi

1 · Ayrık küme birleşimi

$S_i \cup S_j = \{x : x \in S_i \vee x \in S_j\}$. Birleşme sonrası kümeler bağımsız olarak bulunmaz: $S_1 \cup S_2 = \{0, 6, 7, 8, 1, 4, 9\}$.

2 · Küme_bul(i)

i öğesini içeren kümeyi bulur (kümenin adı = ağacın kökü).



Kökler 0, 4, 2. Birleşim için köklerden birinin ebeveyn bilgisi diğer kökü gösterecek şekilde günlenir.

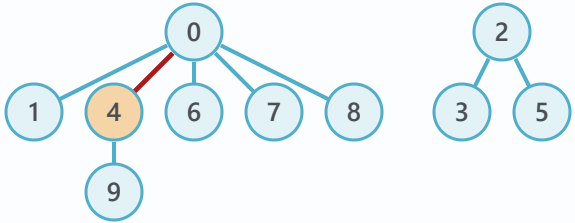
İki işlem ve birleşimin iki biçimi

1 · Ayrık küme birleşimi

$S_i \cup S_j = \{x : x \in S_i \vee x \in S_j\}$. Birleşme sonrası kümeler bağımsız olarak bulunmaz: $S_1 \cup S_2 = \{0, 6, 7, 8, 1, 4, 9\}$.

2 · Küme_bul(i)

i öğesini içeren kümeyi bulur (kümenin adı = ağacın kökü).



$S_1 \cup S_2$: 4'ün ebeveyni 0 yapılır; S_2 ağacı 0'ın alt ağacı olur.

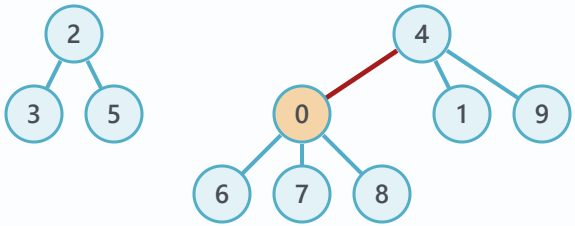
İki işlem ve birleşimin iki biçimi

1 · Ayrık küme birleşimi

$S_i \cup S_j = \{x : x \in S_i \vee x \in S_j\}$. Birleşme sonrası kümeler bağımsız olarak bulunmaz: $S_1 \cup S_2 = \{0, 6, 7, 8, 1, 4, 9\}$.

2 · Küme_bul(i)

i öğesini içeren kümeyi bulur (kümenin adı = ağacın kökü).



$S_2 \cup S_1$: bu kez 0'ın ebeveyni 4 yapılır. İkisi de doğrudur; hangisinin seçileceği maliyeti etkiler.

Kümenin adı: kök düğüm

Küme adı	bağ (kök)
S_1	→ 0
S_2	→ 4
S_3	→ 2

Adlar metin olarak gerekiyorsa ayrı bir tabloda kökü gösteren bir bağ tutulur.

Birleşim ve Küme_bul algoritmalarını incelerken her küme adını ağacın **kökü** ile ilişkilendireceğiz: "S_1 kümesi" yerine "**0 nolu küme**" diyeceğiz.

Böylece `Küme_bul(8) == Küme_bul(6)` sorusu "8 ile 6 aynı kümede mi?" demektir.

Hocadan *"Önce problemi bir matematikçi gibi, programlama bilmeyen birisi gibi ele alıyoruz; öncelikle problemi çizge teorisindeki terminoloji ile doğru ifade edebilmeliyiz."*

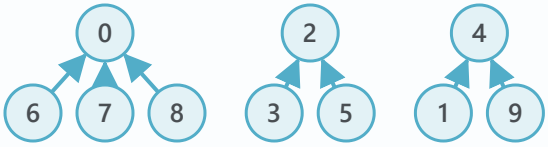
Nasıl gerçekleyelim?

Yöntem	Birleşim	Küme bul	Sorun
Bağlı liste	O(1)	O(n)	arama yavaş
Sıralı tutmak	—	—	sıralamak O(n lg n)
ebeveyn dizisi (ağaç)	O(1)	ağaç derinliği	derinliği küçük tutmalı

Hocadan *"En güvenilir, en sağlam, en hızlı veri yapısı olan diziyi kullanarak gerçeklemenin bir yolu var mı?"*

Düğümler 0'dan n−1'e numaralı olduğundan **numarayı indis** olarak kullanırız: her düğüm yalnızca bir alana (ebeveynine) gereksinim duyar.

`ata` dizisi: her indis ebeveynine bir bağ



ata[i] = i'nin ebeveyni; kökte −1

	0	1	2	3	4	5	6	7	8	9
ata	-1	4	-1	2	-1	2	0	0	0	4

- 1 ve 9'un ebeveyni **4**; 3 ve 5'in ebeveyni **2**; 6, 7 ve 8'in ebeveyni **0**.
- 0, 2 ve 4 kümelerin kökleridir: `ata[kök] = −1` (negatif = kök).

```
#define N 10
int ata[N] = { -1, 4, -1, 2, -1, 2, 0, 0, 0, 4 };
```

DİZİ İLE GERÇEKLEME

İlk sürüm: `Kume_Bul_1` ve `Birlesim_1`

```
int Kume_Bul_1(int i) {
  for (; ata[i] >= 0; i = ata[i])
    ;                /* negatife dek ilerle */
  return i;
}
void Birlesim_1(int i, int j) {
  ata[i] = j;        /* i kümesini j'ye bağla */
}
```

Küme_bul(i)

Ebeveyn bilgisi negatif bir değer olana (köke) dek ilerler; kökü döndürür.

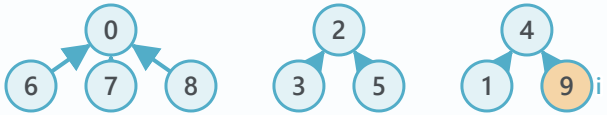
Birleşim(i, j)

i ve j kümeleri (kökleri) aktarıldıktan sonra biri diğerinin alt ağacı olacak şekilde günlenir: O(1).

Ön koşul

i ve j **kök** olmalıdır; değilse önce `Kume_Bul_1` ile kökler bulunur.

Kod izleme: `Kume_Bul_1(9)`, `Birlesim_1(4, 0)`



Kume_Bul_1(9)

	0	1	2	3	4	5	6	7	8	9
ata	-1	4	-1	2	-1	2	0	0	0	4

Kume_Bul_1(9): 9 hangi kümede?

Adım 1 / 9

```
▶ int Kume_Bul_1(int i) {
2   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

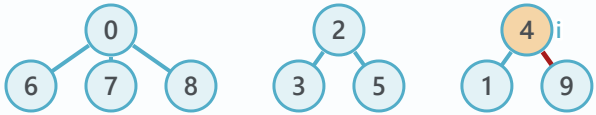
`i = 9`

ÇAĞRI YIĞINI

`Kume_Bul_1(9)`

`main()`

Kod izleme: `Kume_Bul_1(9)`, `Birlesim_1(4, 0)`



Kume_Bul_1(9)

	0	1	2	3	4	5	6	7	8	9
ata	-1	4	-1	2	-1	2	0	0	0	4

ata[9] = 4 ≥ 0 → i = ata[i] = **4**.

Adım 2 / 9

```
1 int Kume_Bul_1(int i) {
▶   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

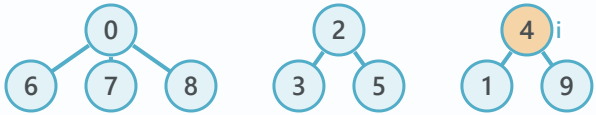
`i = 9` `ata[i] = 4`

ÇAĞRI YIĞINI

`Kume_Bul_1(9)`

`main()`

Kod izleme: `Kume_Bul_1(9)`, `Birlesim_1(4, 0)`



Kume_Bul_1(9)

	0	1	2	3	4	5	6	7	8	9
ata	-1	4	-1	2	-1	2	0	0	0	4

ata[4] = −1 < 0: kök. **return 4** → 9, 4 nolu kümede.

Adım 3 / 9

```
1 int Kume_Bul_1(int i) {
▶   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
▶   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

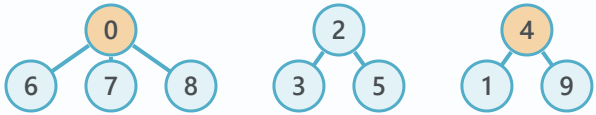
`i = 4` `ata[i] = -1`

ÇAĞRI YIĞINI

`Kume_Bul_1(9)`

`main()`

Kod izleme: `Kume_Bul_1(9)`, `Birlesim_1(4, 0)`



Birlesim_1(4, 0)

	0	1	2	3	4	5	6	7	8	9
ata	-1	4	-1	2	-1	2	0	0	0	4

Birlesim_1(4, 0): iki kök veriliyor.

Adım 4 / 9

```
1 int Kume_Bul_1(int i) {
2   for (; ata[i] >= 0; i = ata[i])
3     ;                 /* negatife dek ilerle */
4   return i;
5 }
▶ void Birlesim_1(int i, int j) {
7   ata[i] = j;         /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

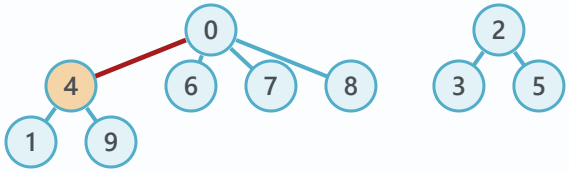
`i = 4` `j = 0`

ÇAĞRI YIĞINI

`Birlesim_1(4, 0)`

`main()`

Kod izleme: `Kume_Bul_1(9)`, `Birlesim_1(4, 0)`



Birlesim_1(4, 0)

	0	1	2	3	4	5	6	7	8	9
ata	-1	4	-1	2	0	2	0	0	0	4

ata[4] = 0: 4'ün ağacı 0'ın altına girdi. Tek atama: O(1).

Adım 5 / 9

```
1 int Kume_Bul_1(int i) {
2   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
▶   ata[i] = j;         /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

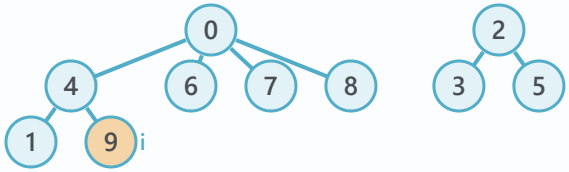
`i = 4` `j = 0` `ata[4] = 0`

ÇAĞRI YIĞINI

`Birlesim_1(4, 0)`

`main()`

Kod izleme: `Kume_Bul_1(9)`, `Birlesim_1(4, 0)`



Kume_Bul_1(9)

	0	1	2	3	4	5	6	7	8	9
ata	-1	4	-1	2	0	2	0	0	0	4

Yeniden **Kume_Bul_1(9)**: yol bir düğüm uzadı.

Adım 6 / 9

```
▶ int Kume_Bul_1(int i) {
2   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

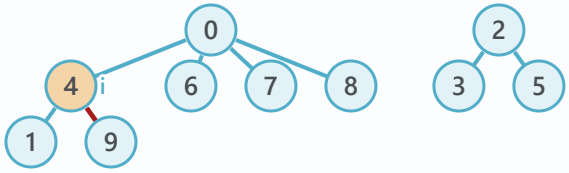
`i = 9`

ÇAĞRI YIĞINI

`Kume_Bul_1(9)`

`main()`

Kod izleme: `Kume_Bul_1(9)`, `Birlesim_1(4, 0)`



Kume_Bul_1(9)

	0	1	2	3	4	5	6	7	8	9
ata	-1	4	-1	2	0	2	0	0	0	4

ata[9] = 4 ≥ 0 → i = **4**.

Adım 7 / 9

```
1 int Kume_Bul_1(int i) {
▶   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

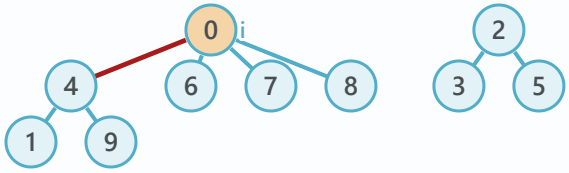
`i = 9` `ata[i] = 4`

ÇAĞRI YIĞINI

`Kume_Bul_1(9)`

`main()`

Kod izleme: `Kume_Bul_1(9)`, `Birlesim_1(4, 0)`



Kume_Bul_1(9)

	0	1	2	3	4	5	6	7	8	9
ata	-1	4	-1	2	0	2	0	0	0	4

ata[4] = 0 ≥ 0 → i = **0**.

Adım 8 / 9

```
1 int Kume_Bul_1(int i) {
▶   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

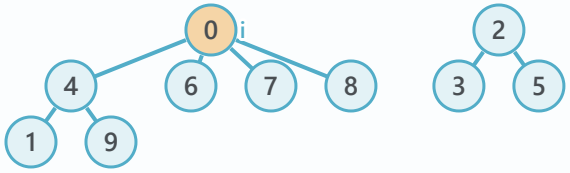
`i = 4` `ata[i] = 0`

ÇAĞRI YIĞINI

`Kume_Bul_1(9)`

`main()`

Kod izleme: `Kume_Bul_1(9)`, `Birlesim_1(4, 0)`



Kume_Bul_1(9)

	0	1	2	3	4	5	6	7	8	9
ata	-1	4	-1	2	0	2	0	0	0	4

return 0: artık 9, 0 nolu kümede. Maliyet = düğümün derinliği.

Adım 9 / 9

```
1 int Kume_Bul_1(int i) {
▶   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
▶   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

`i = 0` `ata[i] = -1`

ÇAĞRI YIĞINI

`Kume_Bul_1(9)`

`main()`

ANALİZ · KODU ADIM ADIM ÇALIŞTIR

Kod izleme: `Birlesim_1(0,1)` … `Birlesim_1(3,4)`, `Kume_Bul_1(0)`



başlangıç

	0	1	2	3	4
ata	-1	-1	-1	-1	-1

n = 5 tek öğeli küme: $S_i = \{i\}$, tümünde `ata[i] = -1`.

Adım 1 / 11

```
▶ int Kume_Bul_1(int i) {
2   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

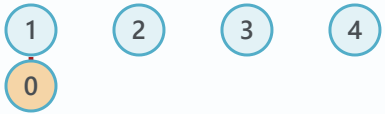
DEĞİŞKENLER

`n = 5`

ÇAĞRI YIĞINI

`main()`

Kod izleme: `Birlesim_1(0,1)` … `Birlesim_1(3,4)`, `Kume_Bul_1(0)`



Birlesim_1(0, 1)

	0	1	2	3	4
ata	1	-1	-1	-1	-1

Birlesim_1(0, 1): ata[0] = 1. Her çağrı O(1).

Adım 2 / 11

```
1 int Kume_Bul_1(int i) {
2   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
▶   ata[i] = j;         /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

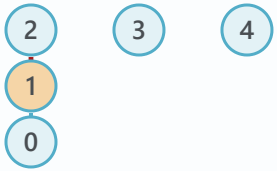
`i = 0` `j = 1`

ÇAĞRI YIĞINI

`Birlesim_1(0, 1)`

`main()`

Kod izleme: `Birlesim_1(0,1)` … `Birlesim_1(3,4)`, `Kume_Bul_1(0)`



Birlesim_1(1, 2)

	0	1	2	3	4
ata	1	2	-1	-1	-1

Birlesim_1(1, 2): ata[1] = 2. Her çağrı O(1).

Adım 3 / 11

```
1 int Kume_Bul_1(int i) {
2   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
▶   ata[i] = j;         /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

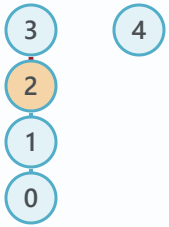
`i = 1` `j = 2`

ÇAĞRI YIĞINI

`Birlesim_1(1, 2)`

`main()`

Kod izleme: `Birlesim_1(0,1)` … `Birlesim_1(3,4)`, `Kume_Bul_1(0)`



Birlesim_1(2, 3)

	0	1	2	3	4
ata	1	2	3	-1	-1

Birlesim_1(2, 3): ata[2] = 3. Her çağrı O(1).

Adım 4 / 11

```
1 int Kume_Bul_1(int i) {
2   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
▶   ata[i] = j;         /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

`i = 2` `j = 3`

ÇAĞRI YIĞINI

`Birlesim_1(2, 3)`

`main()`

Kod izleme: `Birlesim_1(0,1)` … `Birlesim_1(3,4)`, `Kume_Bul_1(0)`



Birlesim_1(3, 4)

	0	1	2	3	4
ata	1	2	3	4	-1

Birlesim_1(3, 4): ata[3] = 4. Her çağrı O(1).

Adım 5 / 11

```
1 int Kume_Bul_1(int i) {
2   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
▶   ata[i] = j;         /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

`i = 3` `j = 4`

ÇAĞRI YIĞINI

`Birlesim_1(3, 4)`

`main()`

Kod izleme: `Birlesim_1(0,1)` … `Birlesim_1(3,4)`, `Kume_Bul_1(0)`



Kume_Bul_1(0)

	0	1	2	3	4
ata	1	2	3	4	-1

Ağaç bir **zincire** dönüştü; derinlik n. Şimdi **Kume_Bul_1(0)**.

Adım 6 / 11

```
▶ int Kume_Bul_1(int i) {
2   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

`i = 0` `adım = 0`

ÇAĞRI YIĞINI

`Kume_Bul_1(0)`

`main()`

Kod izleme: `Birlesim_1(0,1)` … `Birlesim_1(3,4)`, `Kume_Bul_1(0)`



Kume_Bul_1(0)

	0	1	2	3	4
ata	1	2	3	4	-1

i = ata[0] = **1** (1. ilerleme).

Adım 7 / 11

```
1 int Kume_Bul_1(int i) {
▶   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

`i = 1` `adım = 1`

ÇAĞRI YIĞINI

`Kume_Bul_1(0)`

`main()`

Kod izleme: `Birlesim_1(0,1)` … `Birlesim_1(3,4)`, `Kume_Bul_1(0)`



Kume_Bul_1(0)

	0	1	2	3	4
ata	1	2	3	4	-1

i = ata[1] = **2** (2. ilerleme).

Adım 8 / 11

```
1 int Kume_Bul_1(int i) {
▶   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

`i = 2` `adım = 2`

ÇAĞRI YIĞINI

`Kume_Bul_1(0)`

`main()`

Kod izleme: `Birlesim_1(0,1)` … `Birlesim_1(3,4)`, `Kume_Bul_1(0)`



Kume_Bul_1(0)

	0	1	2	3	4
ata	1	2	3	4	-1

i = ata[2] = **3** (3. ilerleme).

Adım 9 / 11

```
1 int Kume_Bul_1(int i) {
▶   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

`i = 3` `adım = 3`

ÇAĞRI YIĞINI

`Kume_Bul_1(0)`

`main()`

Kod izleme: `Birlesim_1(0,1)` … `Birlesim_1(3,4)`, `Kume_Bul_1(0)`



Kume_Bul_1(0)

	0	1	2	3	4
ata	1	2	3	4	-1

i = ata[3] = **4** (4. ilerleme).

Adım 10 / 11

```
1 int Kume_Bul_1(int i) {
▶   for (; ata[i] >= 0; i = ata[i])
3     ;                /* negatife dek ilerle */
4   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;        /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

`i = 4` `adım = 4`

ÇAĞRI YIĞINI

`Kume_Bul_1(0)`

`main()`

Kod izleme: `Birlesim_1(0,1)` … `Birlesim_1(3,4)`, `Kume_Bul_1(0)`



Kume_Bul_1(0)

	0	1	2	3	4
ata	1	2	3	4	-1

return 4 — 4 = n−1 ilerleme. Kume_Bul_1(1), (2)… sırayla n−2, n−3, … ilerler: toplam Σ = $O(n^2)$.

Adım 11 / 11

```
1 int Kume_Bul_1(int i) {
2   for (; ata[i] >= 0; i = ata[i])
3     ;                 /* negatife dek ilerle */
▶   return i;
5 }
6 void Birlesim_1(int i, int j) {
7   ata[i] = j;         /* i kümesini j'ye bağla */
8 }
```

DEĞİŞKENLER

`i = 4` `adım = 4`

ÇAĞRI YIĞINI

`Kume_Bul_1(0)`

`main()`

ANALİZ

`Kume_Bul_1` ve `Birlesim_1`'in analizi

```
/* n tek öğeli kümeyle başla */
Birlesim_1(0, 1);
Birlesim_1(1, 2);
...
Birlesim_1(n-2, n-1);
Kume_Bul_1(0);
Kume_Bul_1(1);
...
Kume_Bul_1(n-1);
```

- n−1 birleşim: her biri sabit → toplam **O(n)**, sorun yok.
- Ağaç zincire döner; derinlik **n** olur.
- (n−i). düzeydeki i için Kume_Bul_1 O(i) sürer: 0 için n−1, 1 için n−2… ilerleme.
- Tek bir Kume_Bul_1 O(n)'e, n tanesi toplamda **$\Sigma\, i = O(n^2)$**'ye çıkar.

Bu işlem dizisi ağaç yapısını bozar. Suçlu birleşimin kendisi değil, *hangi ağacın hangisine* bağlandığıdır. Çözüm: ağırlıklı kural.

ANALİZ

Bir parantez: en kötü durumu test etmek

En kötü durum girdisi

En kötü durum analizinde kod, çakılacağı ya da işletiminin en uzun süreceği verilerle test edilmelidir. Bunu saptamak zordur.

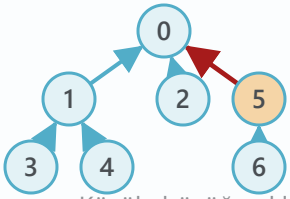
Burada

Zincir üreten birleşim sırası, `Kume_Bul_1` için tam da böyle bir girdidir: rastgele testler onu kolayca kaçırır.

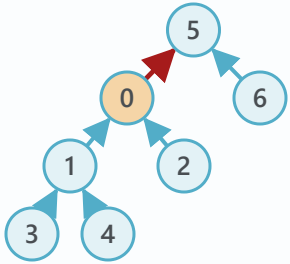
Hocadan *"Beta sürümü ne demektir? 'Biz kafasızız, gözden kaçırdığımız hatalar olabilir. Kafasızlığımızı affedin, hata bulursanız bizi bilgilendirin ki, onun üstüne gidelim.' Software Testing ayrı bir alandır; onun üzerine de makaleler okunmalı."*

Ağırlıklı kural (weighting rule)

Tanım: i ağacındaki düğüm sayısı j ağacındakinden azsa j, i'nin ebeveyni; aksi hâlde i, j'nin ebeveyni yapılır. (Düğüm sayısı çok olan ebeveyn olur.)



Küçük, büyüğe eklenirse: derinlik 3



Büyük, küçüğe eklenirse: derinlik 4

Hocadan *"Sual: Acaba her kümede (ağaçta) kaç düğüm olduğu bilgisi için ayrı bir dizi mi tutmamız gerekir? El-Cevap: Kökler dışındaki tüm düğümler için ebeveyn dizisinde pozitif değerler tutuluyordu. Öyleyse köklere ilişkin düğüm sayısı bilgisi de ebeveyn dizisinde negatif olarak tutulabilir."*

AĞIRLIKLI KURAL

`Birlesim_2`: kökte eksi düğüm sayısı

```
void Birlesim_2(int i, int j) {
  /* ağırlıklı kural; i ve j kök olmalı */
  int p = ata[i] + ata[j];  /* -(toplam düğüm) */
  if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
    ata[i] = j;             /* i'yi j'ye bağla */
    ata[j] = p;             /* j'nin sayısını günle */
  } else {                  /* j'de az ya da eşit */
    ata[j] = i;
    ata[i] = p;
  }
}
```

Kökte −sayı

Kökteki değer `-(düğüm sayısı)`: tek öğeli küme −1, 8 öğeli küme −8.

Neden `>`?

Değerler negatif: `ata[i] > ata[j]`, i'nin **daha az** düğümü olduğu anlamına gelir (−2 > −4).

Maliyet

Hâlâ O(1); ek dizi gerekmez.

AĞIRLIKLI KURAL · KODU ADIM ADIM ÇALIŞTIR

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(0, 1)

	0	1	2	3	4	5	6	7
ata	-1	-1	-1	-1	-1	-1	-1	-1

Birlesim_2(0, 1): 0 kümesinde 1, 1 kümesinde 1 düğüm.

Adım 1 / 16

```
▶ void Birlesim_2(int i, int j) {
2   /* ağırlıklı kural; i ve j kök olmalı */
3   int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4   if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5     ata[i] = j;             /* i'yi j'ye bağla */
6     ata[j] = p;             /* j'nin sayısını günle
  */
7   } else {                  /* j'de az ya da eşit */
8     ata[j] = i;
9     ata[i] = p;
10  }
11 }
```

DEĞİŞKENLER

`i = 0` `j = 1` `ata[i] = -1` `ata[j] = -1`

ÇAĞRI YIĞINI

- **`Birlesim_2(0, 1)`**
- `main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(0, 1)

	0	1	2	3	4	5	6	7
ata	-1	-1	-1	-1	-1	-1	-1	-1

p = -1 + (-1) = **-2**: birleşik kümede 2 düğüm.

Adım 2 / 16

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
▶    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

`i = 0` `j = 1` `p = -2`

ÇAĞRI YIĞINI

`Birlesim_2(0, 1)`

`main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(0, 1)

	0	1	2	3	4	5	6	7
ata	-1	-1	-1	-1	-1	-1	-1	-1

-1 > -1 yanlış (j'de az ya da eşit) → **else**: j, i'ye bağlanacak.

Adım 3 / 16

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
▶    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

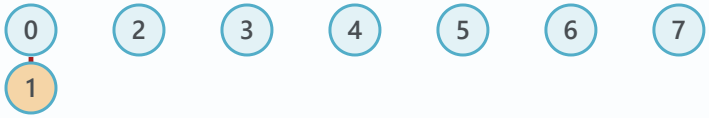
`i = 0` `j = 1` `p = -2`

ÇAĞRI YIĞINI

`Birlesim_2(0, 1)`

`main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(0, 1)

	0	1	2	3	4	5	6	7
ata	-2	0	-1	-1	-1	-1	-1	-1

ata[1] = 0, **ata[0] = -2**. En derin düğüm 2. düzeyde.

Adım 4 / 16

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
▶      ata[j] = i;
▶      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

`i = 0` `j = 1` `p = -2`

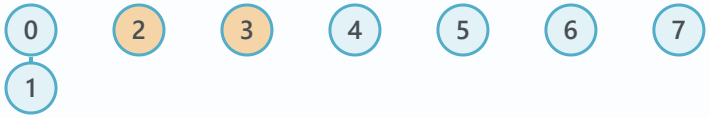
`ata[j] = 0` `ata[i] = -2`

ÇAĞRI YIĞINI

`Birlesim_2(0, 1)`

`main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(2, 3)

	0	1	2	3	4	5	6	7
ata	-2	0	-1	-1	-1	-1	-1	-1

Birlesim_2(2, 3): 2 kümesinde 1, 3 kümesinde 1 düğüm.

Adım 5 / 16

```
▶ void Birlesim_2(int i, int j) {
2   /* ağırlıklı kural; i ve j kök olmalı */
3   int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4   if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5     ata[i] = j;             /* i'yi j'ye bağla */
6     ata[j] = p;             /* j'nin sayısını günle
  */
7   } else {                  /* j'de az ya da eşit */
8     ata[j] = i;
9     ata[i] = p;
10  }
11 }
```

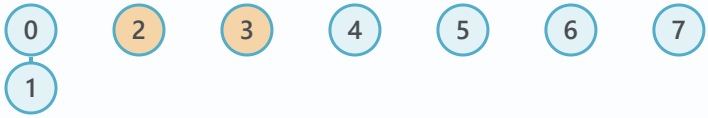
DEĞİŞKENLER

`i = 2` `j = 3` `ata[i] = -1` `ata[j] = -1`

ÇAĞRI YIĞINI

- `Birlesim_2(2, 3)`
- `main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(2, 3)

	0	1	2	3	4	5	6	7
ata	-2	0	-1	-1	-1	-1	-1	-1

p = -1 + (-1) = **-2**: birleşik kümede 2 düğüm.

Adım 6 / 16

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
▶    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

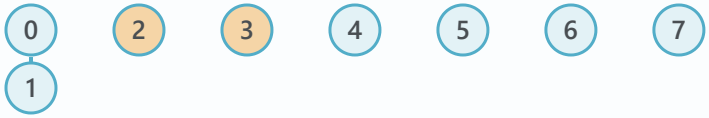
`i = 2` `j = 3` `p = −2`

ÇAĞRI YIĞINI

`Birlesim_2(2, 3)`

`main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(2, 3)

	0	1	2	3	4	5	6	7
ata	-2	0	-1	-1	-1	-1	-1	-1

-1 > -1 yanlış (j'de az ya da eşit) → **else**: j, i'ye bağlanacak.

Adım 7 / 16

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
▶    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

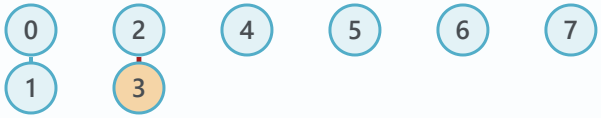
`i = 2` `j = 3` `p = -2`

ÇAĞRI YIĞINI

`Birlesim_2(2, 3)`

`main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(2, 3)

	0	1	2	3	4	5	6	7
ata	-2	0	-2	2	-1	-1	-1	-1

ata[3] = 2, **ata[2] = -2**. En derin düğüm 2. düzeyde.


```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];   /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {     /* i'de daha az düğüm */
5      ata[i] = j;              /* i'yi j'ye bağla */
6      ata[j] = p;              /* j'nin sayısını günle
   */
7    } else {                   /* j'de az ya da eşit */
▶      ata[j] = i;
▶      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

`i = 2` `j = 3` `p = −2`

`ata[j] = 2` `ata[i] = −2`

ÇAĞRI YIĞINI

`Birlesim_2(2, 3)`

`main()`

AĞIRLIKLI KURAL · KODU ADIM ADIM ÇALIŞTIR

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(4, 5)

	0	1	2	3	4	5	6	7
ata	-2	0	-2	2	-1	-1	-1	-1

Birlesim_2(4, 5): 4 kümesinde 1, 5 kümesinde 1 düğüm.

Adım 9 / 16

```
▶ void Birlesim_2(int i, int j) {
2   /* ağırlıklı kural; i ve j kök olmalı */
3   int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4   if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5     ata[i] = j;             /* i'yi j'ye bağla */
6     ata[j] = p;             /* j'nin sayısını günle
  */
7   } else {                  /* j'de az ya da eşit */
8     ata[j] = i;
9     ata[i] = p;
10  }
11 }
```

DEĞİŞKENLER

`i = 4` `j = 5` `ata[i] = -1` `ata[j] = -1`

ÇAĞRI YIĞINI

`Birlesim_2(4, 5)`

`main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(4, 5)

	0	1	2	3	4	5	6	7
ata	-2	0	-2	2	-1	-1	-1	-1

p = -1 + (-1) = **-2**: birleşik kümede 2 düğüm.

Adım 10 / 16

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
▶    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

`i = 4` `j = 5` `p = -2`

ÇAĞRI YIĞINI

`Birlesim_2(4, 5)`

`main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(4, 5)

	0	1	2	3	4	5	6	7
ata	-2	0	-2	2	-1	-1	-1	-1

-1 > -1 yanlış (j'de az ya da eşit) → **else**: j, i'ye bağlanacak.

Adım 11 / 16

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
▶    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

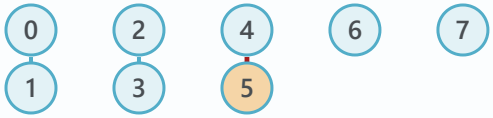
`i = 4` `j = 5` `p = -2`

ÇAĞRI YIĞINI

`Birlesim_2(4, 5)`

`main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(4, 5)

	0	1	2	3	4	5	6	7
ata	-2	0	-2	2	-2	4	-1	-1

ata[5] = 4, **ata[4] = -2**. En derin düğüm 2. düzeyde.

Adım 12 / 16

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
▶      ata[j] = i;
▶      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

`i = 4` `j = 5` `p = -2` `ata[j] = 4` `ata[i] = -2`

ÇAĞRI YIĞINI

`Birlesim_2(4, 5)`

`main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(6, 7)

	0	1	2	3	4	5	6	7
ata	-2	0	-2	2	-2	4	-1	-1

Birlesim_2(6, 7): 6 kümesinde 1, 7 kümesinde 1 düğüm.

Adım 13 / 16

```
▶ void Birlesim_2(int i, int j) {
2   /* ağırlıklı kural; i ve j kök olmalı */
3   int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4   if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5     ata[i] = j;             /* i'yi j'ye bağla */
6     ata[j] = p;             /* j'nin sayısını günle
  */
7   } else {                  /* j'de az ya da eşit */
8     ata[j] = i;
9     ata[i] = p;
10  }
11 }
```

DEĞİŞKENLER

`i = 6` `j = 7` `ata[i] = -1` `ata[j] = -1`

ÇAĞRI YIĞINI

- `Birlesim_2(6, 7)`
- `main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(6, 7)

	0	1	2	3	4	5	6	7
ata	-2	0	-2	2	-2	4	-1	-1

p = -1 + (-1) = **-2**: birleşik kümede 2 düğüm.

Adım 14 / 16

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
▶    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

`i = 6` `j = 7` `p = -2`

ÇAĞRI YIĞINI

`Birlesim_2(6, 7)`

`main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(6, 7)

	0	1	2	3	4	5	6	7
ata	-2	0	-2	2	-2	4	-1	-1

-1 > -1 yanlış (j'de az ya da eşit) → **else**: j, i'ye bağlanacak.

Adım 15 / 16

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
▶    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

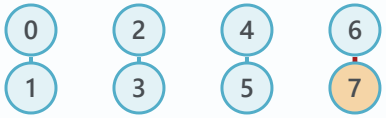
`i = 6` `j = 7` `p = -2`

ÇAĞRI YIĞINI

`Birlesim_2(6, 7)`

`main()`

Kod izleme: `Birlesim_2` (0,1) (2,3) (4,5) (6,7)



Birlesim_2(6, 7)

	0	1	2	3	4	5	6	7
ata	-2	0	-2	2	-2	4	-2	6

ata[7] = 6, **ata[6] = -2**. En derin düğüm 2. düzeyde.

Adım 16 / 16

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
▶      ata[j] = i;
▶      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

`i = 6` `j = 7` `p = -2`

`ata[j] = 6` `ata[i] = -2`

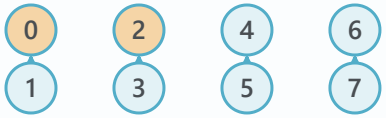
ÇAĞRI YIĞINI

`Birlesim_2(6, 7)`

`main()`

AĞIRLIKLI KURAL · KODU ADIM ADIM ÇALIŞTIR

Kod izleme: `Birlesim_2` (0,2) (4,6) (0,4)



Birlesim_2(0, 2)

	0	1	2	3	4	5	6	7
ata	-2	0	-2	2	-2	4	-2	6

Birlesim_2(0, 2): 0 kümesinde 2, 2 kümesinde 2 düğüm.

Adım 1 / 12

```
▶ void Birlesim_2(int i, int j) {
2   /* ağırlıklı kural; i ve j kök olmalı */
3   int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4   if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5     ata[i] = j;             /* i'yi j'ye bağla */
6     ata[j] = p;             /* j'nin sayısını günle
  */
7   } else {                  /* j'de az ya da eşit */
8     ata[j] = i;
9     ata[i] = p;
10  }
11 }
```

DEĞİŞKENLER

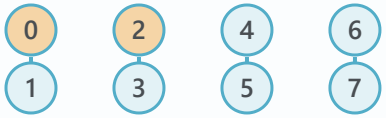
`i = 0` `j = 2` `ata[i] = -2` `ata[j] = -2`

ÇAĞRI YIĞINI

`Birlesim_2(0, 2)`

`main()`

Kod izleme: `Birlesim_2` (0,2) (4,6) (0,4)



Birlesim_2(0, 2)

	0	1	2	3	4	5	6	7
ata	-2	0	-2	2	-2	4	-2	6

p = -2 + (-2) = **-4**: birleşik kümede 4 düğüm.

Adım 2 / 12

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
▶    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

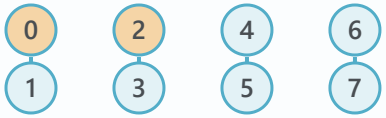
`i = 0` `j = 2` `p = -4`

ÇAĞRI YIĞINI

`Birlesim_2(0, 2)`

`main()`

Kod izleme: `Birlesim_2` (0,2) (4,6) (0,4)



Birlesim_2(0, 2)

	0	1	2	3	4	5	6	7
ata	-2	0	-2	2	-2	4	-2	6

-2 > -2 yanlış (j'de az ya da eşit) → **else**: j, i'ye bağlanacak.

Adım 3 / 12

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
▶    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

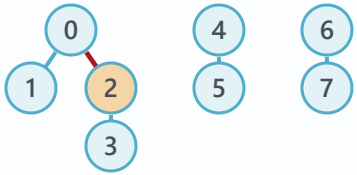
`i = 0` `j = 2` `p = -4`

ÇAĞRI YIĞINI

`Birlesim_2(0, 2)`

`main()`

Kod izleme: `Birlesim_2` (0,2) (4,6) (0,4)



Birlesim_2(0, 2)

	0	1	2	3	4	5	6	7
ata	-4	0	0	2	-2	4	-2	6

ata[2] = 0, **ata[0] = -4**. En derin düğüm 3. düzeyde.

Adım 4 / 12

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
▶      ata[j] = i;
▶      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

`i = 0` `j = 2` `p = -4`

`ata[j] = 0` `ata[i] = -4`

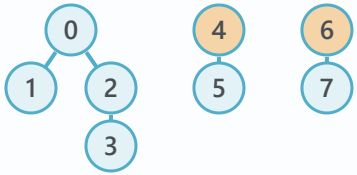
ÇAĞRI YIĞINI

`Birlesim_2(0, 2)`

`main()`

AĞIRLIKLI KURAL · KODU ADIM ADIM ÇALIŞTIR

Kod izleme: `Birlesim_2` (0,2) (4,6) (0,4)



Birlesim_2(4, 6)

	0	1	2	3	4	5	6	7
ata	-4	0	0	2	-2	4	-2	6

Birlesim_2(4, 6): 4 kümesinde 2, 6 kümesinde 2 düğüm.

Adım 5 / 12

```
▶ void Birlesim_2(int i, int j) {
2   /* ağırlıklı kural; i ve j kök olmalı */
3   int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4   if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5     ata[i] = j;             /* i'yi j'ye bağla */
6     ata[j] = p;             /* j'nin sayısını günle
  */
7   } else {                  /* j'de az ya da eşit */
8     ata[j] = i;
9     ata[i] = p;
10  }
11 }
```

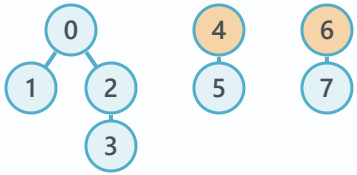
DEĞİŞKENLER

`i = 4` `j = 6` `ata[i] = -2` `ata[j] = -2`

ÇAĞRI YIĞINI

- `Birlesim_2(4, 6)`
- `main()`

Kod izleme: `Birlesim_2` (0,2) (4,6) (0,4)



Birlesim_2(4, 6)

	0	1	2	3	4	5	6	7
ata	-4	0	0	2	-2	4	-2	6

p = -2 + (-2) = **-4**: birleşik kümede 4 düğüm.

Adım 6 / 12

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
▶    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

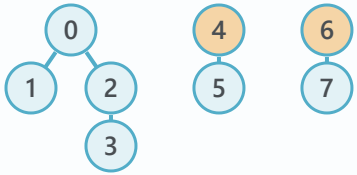
`i = 4` `j = 6` `p = -4`

ÇAĞRI YIĞINI

`Birlesim_2(4, 6)`

`main()`

Kod izleme: `Birlesim_2` (0,2) (4,6) (0,4)



Birlesim_2(4, 6)

	0	1	2	3	4	5	6	7
ata	-4	0	0	2	-2	4	-2	6

-2 > -2 yanlış (j'de az ya da eşit) → **else**: j, i'ye bağlanacak.

Adım 7 / 12

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
▶    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

`i = 4` `j = 6` `p = -4`

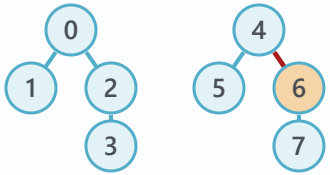
ÇAĞRI YIĞINI

`Birlesim_2(4, 6)`

`main()`

AĞIRLIKLI KURAL · KODU ADIM ADIM ÇALIŞTIR

Kod izleme: `Birlesim_2` (0,2) (4,6) (0,4)



Birlesim_2(4, 6)

	0	1	2	3	4	5	6	7
ata	-4	0	0	2	-4	4	4	6

ata[6] = 4, **ata[4] = -4**. En derin düğüm 3. düzeyde.

Adım 8 / 12

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
▶      ata[j] = i;
▶      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

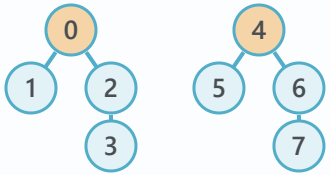
`i = 4` `j = 6` `p = -4` `ata[j] = 4` `ata[i] = -4`

ÇAĞRI YIĞINI

`Birlesim_2(4, 6)`

`main()`

Kod izleme: `Birlesim_2` (0,2) (4,6) (0,4)



Birlesim_2(0, 4)

	0	1	2	3	4	5	6	7
ata	-4	0	0	2	-4	4	4	6

Birlesim_2(0, 4): 0 kümesinde 4, 4 kümesinde 4 düğüm.

Adım 9 / 12

```
▶ void Birlesim_2(int i, int j) {
2   /* ağırlıklı kural; i ve j kök olmalı */
3   int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4   if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5     ata[i] = j;             /* i'yi j'ye bağla */
6     ata[j] = p;             /* j'nin sayısını günle
  */
7   } else {                  /* j'de az ya da eşit */
8     ata[j] = i;
9     ata[i] = p;
10  }
11 }
```

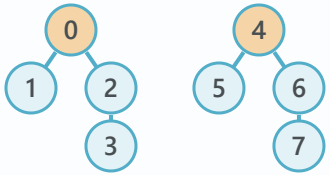
DEĞİŞKENLER

`i = 0` `j = 4` `ata[i] = -4` `ata[j] = -4`

ÇAĞRI YIĞINI

- `Birlesim_2(0, 4)`
- `main()`

Kod izleme: `Birlesim_2` (0,2) (4,6) (0,4)



Birlesim_2(0, 4)

	0	1	2	3	4	5	6	7
ata	-4	0	0	2	-4	4	4	6

p = -4 + (-4) = **-8**: birleşik kümede 8 düğüm.

Adım 10 / 12

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
▶    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

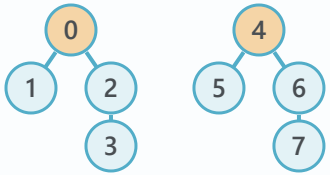
`i = 0` `j = 4` `p = -8`

ÇAĞRI YIĞINI

`Birlesim_2(0, 4)`

`main()`

Kod izleme: `Birlesim_2` (0,2) (4,6) (0,4)



Birlesim_2(0, 4)

	0	1	2	3	4	5	6	7
ata	-4	0	0	2	-4	4	4	6

-4 > -4 yanlış (j'de az ya da eşit) → **else**: j, i'ye bağlanacak.

Adım 11 / 12

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];   /* -(toplam düğüm) */
▶    if (ata[i] > ata[j]) {     /* i'de daha az düğüm */
5      ata[i] = j;              /* i'yi j'ye bağla */
6      ata[j] = p;              /* j'nin sayısını günle
   */
7    } else {                   /* j'de az ya da eşit */
8      ata[j] = i;
9      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

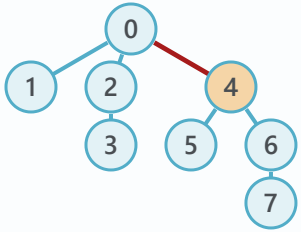
`i = 0` `j = 4` `p = -8`

ÇAĞRI YIĞINI

`Birlesim_2(0, 4)`

`main()`

Kod izleme: `Birlesim_2` (0,2) (4,6) (0,4)



Birlesim_2(0, 4)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	4	6

ata[4] = 0, **ata[0] = -8**. En derin düğüm 4. düzeyde.

Adım 12 / 12

```
1  void Birlesim_2(int i, int j) {
2    /* ağırlıklı kural; i ve j kök olmalı */
3    int p = ata[i] + ata[j];  /* -(toplam düğüm) */
4    if (ata[i] > ata[j]) {    /* i'de daha az düğüm */
5      ata[i] = j;             /* i'yi j'ye bağla */
6      ata[j] = p;             /* j'nin sayısını günle
   */
7    } else {                  /* j'de az ya da eşit */
▶      ata[j] = i;
▶      ata[i] = p;
10   }
11 }
```

DEĞİŞKENLER

`i = 0` `j = 4` `p = -8` `ata[j] = 0` `ata[i] = -8`

ÇAĞRI YIĞINI

`Birlesim_2(0, 4)`

`main()`

AĞIRLIKLI KURAL

Soru: Ağacı ne kadar dejenere eder, bozar?

Hocadan *"Algoritma doğru, etkili ve yazılım mühendisliği kurallarına göre yazılacak. Ama bitmiyor, en çok bozacak veriyi sorguluyoruz. Arka planda ne olursa, kodum en fazla çalışır? Bilgisayar bilimi, bilgisayar mühendisliği işte budur!"*

Hocadan *"'Hepiniz C, Java vs. biliyor musunuz?' Evet, biliyorlar. 'Tamam, hepiniz işe girin, aynı maaş!' Bilgisayar mühendisi der ki: 'Bir dakika! Ben bilgisayar mühendisiyim. Nasıl aynı maaş olur? Sen bir soru sor bakalım! Bir soru sor, nasıl çözdüğüme bak!'"*

Ağırlıklı kuralda en kötü durum: her seferinde **eşit büyüklükte** iki ağacı birleştirmek (`Birlesim_2` izlemelerindeki gibi). Her birleşimde küçük ağacın tüm düğümlerinin derinliği 1 artar, ama ağaç en az iki katına çıkar.

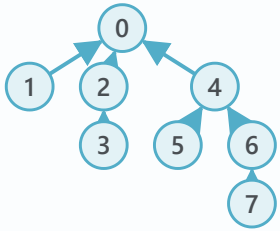
AĞIRLIKLI KURAL

Lemma: ağırlıklı birleşimle derinlik logaritmiktir

LEMMA (HOROWİTZ)

T, `Birlesim_2` ile çeşitli kümelerin birleşimi sonucu elde edilmiş **m düğümlü** bir ağaç ise T'nin derinliği (düzey sayısı) ⌊**lg m**⌋ **+ 1**'den büyük olamaz.

- **İspat fikri (tümevarım):** bir düğümün derinliği yalnızca onun ağacı *daha küçük ya da eşit* bir ağaca bağlanınca 1 artar.
- O zaman düğümün yer aldığı ağaç en az **iki katına** çıkar.
- m düğüme en çok lg m kez iki katına çıkılabilir → derinlik ≤ ⌊lg m⌋ + 1.



m = 8 → düzey sayısı 4 = ⌊lg 8⌋ + 1

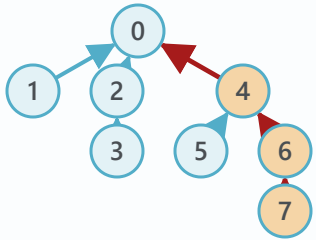
Hocadan *"Matematik bölümü olsaydık 'İspatlayınız.' diye soru devam edecekti. […] Karmaşıklığı n'den ⌊lg n⌋+1'e düşürdük. Algoritma geliştirme bu, işte! Ama yerimizde durmuyoruz, devam…"*

Azaltma kuralı (collapsing rule)

Son durumdaki yapıda `Küme_bul(7)` köke 3 ilerlemede ulaşır. Program `Küme_bul(7)`'yi tekrar tekrar işletirse maliyet katlanır. Düğüm sayısı az olanı çok olana eklesek bile derinlik artabiliyor; derinliği denetleyemediğimize göre bir adım ileri gideriz:

TANIM

Eğer j, i düğümünden köke olan yol üzerinde bir düğüm ise (i kök değil) j düğümü **doğrudan köke bağlanır**.



7 → 6 → 4 → 0 yolundaki her düğüm köke bağlanacak

Hocadan *"Bunun graf teorisindeki karşılığı 'kaplama' ya da 'kapama'. Graf teorisi ödevlerinden birinde, hangisi olduğu söylenmeyecek, bu kullanılmalı. Problem farklı, alt yapısı bu."*

AZALTMA KURALI

`Kume_bul_2`: önce kök bulunur, sonra yol köke bağlanır

```
int Kume_bul_2(int i) {
  int r, s;
  for (r = i; ata[r] >= 0; r = ata[r])
    ;                    /* kökü bul */
  while (i != r) {       /* yolu azalt */
    s = ata[i];          /* i'nin ebeveynini sakla */
    ata[i] = r;          /* i'yi doğrudan köke bağla */
    i = s;               /* ebeveynine geç */
  }
  return r;
}
```

1. geçiş (for)

`r` kökü bulur (ebeveyn bilgisi negatif).

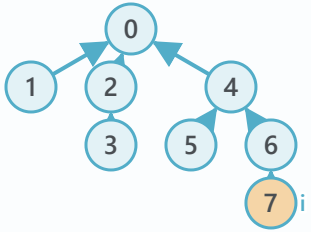
2. geçiş (while)

Yol üzerindeki her düğüm doğrudan `r`'ye bağlanır. Köke zaten bağlı bir düğüm için de (gereksiz ama zararsız) bir kez çalışır.

Kazanç

Aynı yoldaki sonraki aramalar bir çırpıda köke ulaşır.

Kod izleme: `Kume_bul_2(7)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	4	6

Kume_bul_2(7).

Adım 1 / 12

```
▶ int Kume_bul_2(int i) {
2   int r, s;
3   for (r = i; ata[r] >= 0; r = ata[r])
4     ;                    /* kökü bul */
5   while (i != r) {       /* yolu azalt */
6     s = ata[i];          /* i'nin ebeveynini sakla */
7     ata[i] = r;          /* i'yi doğrudan köke bağla
  */
8     i = s;               /* ebeveynine geç */
9   }
10  return r;
11 }
```

DEĞİŞKENLER

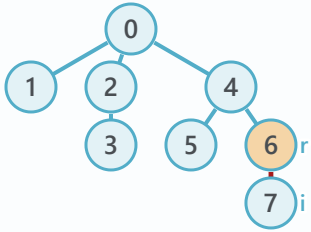
`i = 7`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: `Kume_bul_2(7)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	4	6

ata[7] = 6 ≥ 0 → r = **6**.

Adım 2 / 12

```
1  int Kume_bul_2(int i) {
2    int r, s;
▶    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

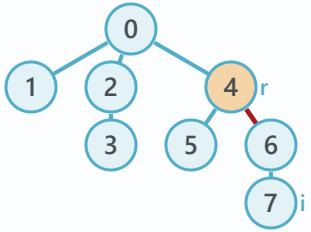
`i = 7` `r = 6`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: `Kume_bul_2(7)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	4	6

ata[6] = 4 ≥ 0 → r = **4**.

Adım 3 / 12

```
1  int Kume_bul_2(int i) {
2    int r, s;
▶    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

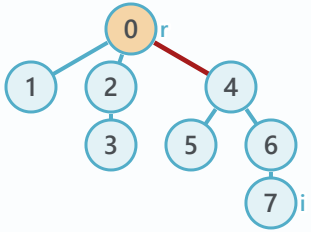
`i = 7` `r = 4`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: `Kume_bul_2(7)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	4	6

ata[4] = 0 ≥ 0 → r = **0**.

Adım 4 / 12

```
1  int Kume_bul_2(int i) {
2    int r, s;
▶    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

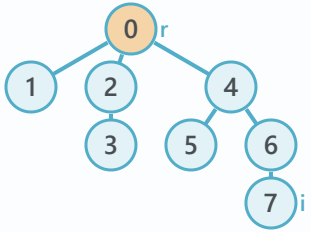
`i = 7` `r = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: `Kume_bul_2(7)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	4	6

ata[0] = -8 < 0: kök **r = 0**.

Adım 5 / 12

```
1  int Kume_bul_2(int i) {
2    int r, s;
▶    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

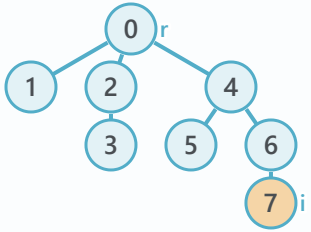
`i = 7` `r = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: `Kume_bul_2(7)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	4	6

i = 7 ≠ r: **s = ata[7] = 6** saklanır.

Adım 6 / 12

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
▶    while (i != r) {       /* yolu azalt */
▶      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

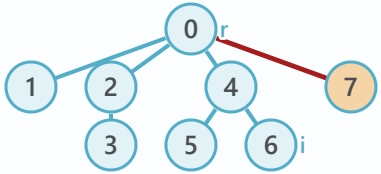
`i = 7` `r = 0` `s = 6`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: `Kume_bul_2(7)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	4	0

ata[7] = 0: 7 doğrudan köke bağlandı; i = s = 6.

Adım 7 / 12

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
▶      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
▶      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

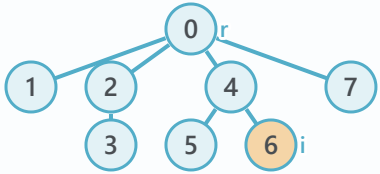
`i = 6` `r = 0` `s = 6`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: `Kume_bul_2(7)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	4	0

i = 6 ≠ r: **s = ata[6] = 4** saklanır.

Adım 8 / 12

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
▶    while (i != r) {       /* yolu azalt */
▶      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

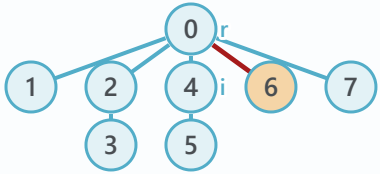
`i = 6` `r = 0` `s = 4`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: `Kume_bul_2(7)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

ata[6] = 0: 6 doğrudan köke bağlandı; i = s = 4.

Adım 9 / 12

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
▶      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
▶      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

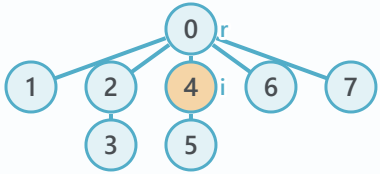
`i = 4` `r = 0` `s = 4`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: `Kume_bul_2(7)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

i = 4 ≠ r: **s = ata[4] = 0** saklanır.

Adım 10 / 12

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
▶    while (i != r) {       /* yolu azalt */
▶      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

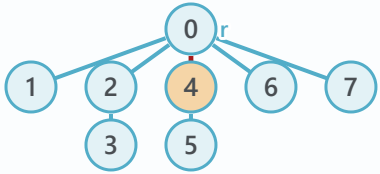
`i = 4` `r = 0` `s = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: `Kume_bul_2(7)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

ata[4] = 0 (zaten köke bağlıydı: fazladan bir tur); i = 0.

Adım 11 / 12

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
▶      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
▶      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

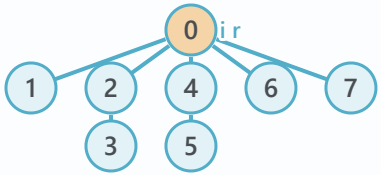
`i = 0` `r = 0` `s = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: `Kume_bul_2(7)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

i == r: döngü biter, **return 0**. ata = [-8, 0, 0, 2, 0, 4, 0, 0].

Adım 12 / 12

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
▶    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
▶    return r;
11 }
```

DEĞİŞKENLER

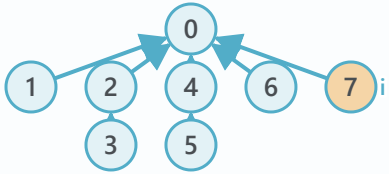
`i = 0` `r = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

Kume_bul_2(7).

Adım 1 / 15

```
▶ int Kume_bul_2(int i) {
2   int r, s;
3   for (r = i; ata[r] >= 0; r = ata[r])
4     ;                     /* kökü bul */
5   while (i != r) {        /* yolu azalt */
6     s = ata[i];           /* i'nin ebeveynini sakla */
7     ata[i] = r;           /* i'yi doğrudan köke bağla
  */
8     i = s;                /* ebeveynine geç */
9   }
10  return r;
11 }
```

DEĞİŞKENLER

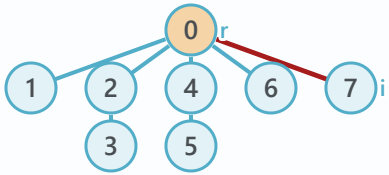
`i = 7`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

ata[7] = 0 ≥ 0 → r = **0**.

Adım 2 / 15

```
1  int Kume_bul_2(int i) {
2    int r, s;
▶    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

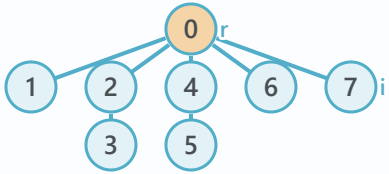
`i = 7` `r = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

ata[0] = -8 < 0: kök **r = 0**.

Adım 3 / 15

```
1  int Kume_bul_2(int i) {
2    int r, s;
▶    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

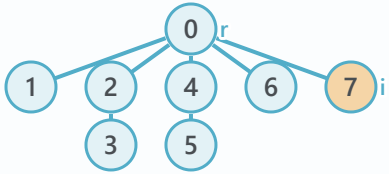
`i = 7` `r = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

i = 7 ≠ r: **s = ata[7] = 0** saklanır.

Adım 4 / 15

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
▶    while (i != r) {       /* yolu azalt */
▶      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

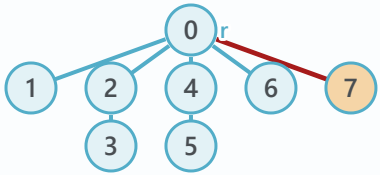
`i = 7` `r = 0` `s = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

ata[7] = 0 (zaten köke bağlıydı: fazladan bir tur); i = 0.


```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
▶      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
▶      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

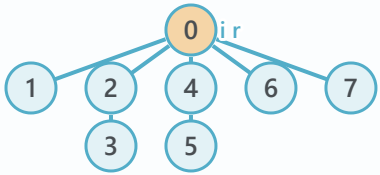
`i = 0` `r = 0` `s = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(7)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

i == r: döngü biter, **return 0**. ata = [-8, 0, 0, 2, 0, 4, 0, 0].

Adım 6 / 15

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
▶    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
▶    return r;
11 }
```

DEĞİŞKENLER

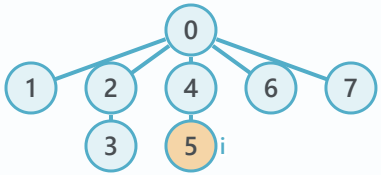
`i = 0` `r = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(7)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(5)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

Kume_bul_2(5).

Adım 7 / 15

```
▶ int Kume_bul_2(int i) {
2   int r, s;
3   for (r = i; ata[r] >= 0; r = ata[r])
4     ;                     /* kökü bul */
5   while (i != r) {        /* yolu azalt */
6     s = ata[i];           /* i'nin ebeveynini sakla */
7     ata[i] = r;           /* i'yi doğrudan köke bağla
  */
8     i = s;                /* ebeveynine geç */
9   }
10  return r;
11 }
```

DEĞİŞKENLER

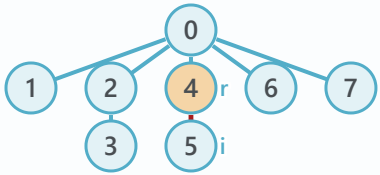
`i = 5`

ÇAĞRI YIĞINI

`Kume_bul_2(5)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(5)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

ata[5] = 4 ≥ 0 → r = **4**.

Adım 8 / 15

```
1  int Kume_bul_2(int i) {
2    int r, s;
▶    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

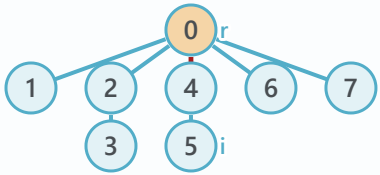
`i = 5` `r = 4`

ÇAĞRI YIĞINI

`Kume_bul_2(5)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(5)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

ata[4] = 0 ≥ 0 → r = **0**.

Adım 9 / 15

```
1  int Kume_bul_2(int i) {
2    int r, s;
▶    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

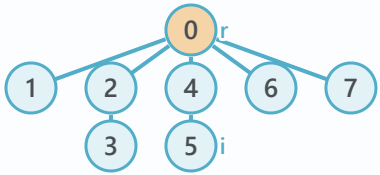
`i = 5` `r = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(5)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(5)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

ata[0] = -8 < 0: kök **r = 0**.

Adım 10 / 15

```
1  int Kume_bul_2(int i) {
2    int r, s;
▶    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

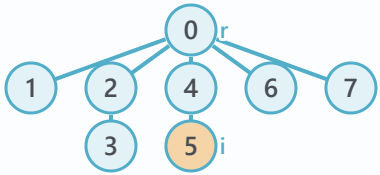
`i = 5` `r = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(5)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(5)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	4	0	0

i = 5 ≠ r: **s = ata[5] = 4** saklanır.

Adım 11 / 15

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
▶    while (i != r) {       /* yolu azalt */
▶      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

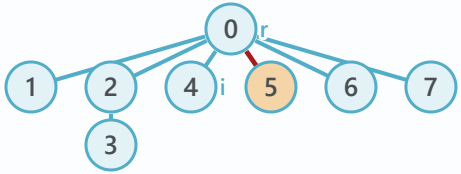
`i = 5` `r = 0` `s = 4`

ÇAĞRI YIĞINI

`Kume_bul_2(5)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(5)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	0	0	0

ata[5] = 0: 5 doğrudan köke bağlandı; i = s = 4.


```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
       ata[i] = r;          /* i'yi doğrudan köke bağla
   */
       i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

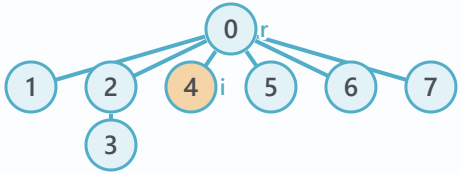
`i = 4` `r = 0` `s = 4`

ÇAĞRI YIĞINI

`Kume_bul_2(5)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(5)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	0	0	0

i = 4 ≠ r: **s = ata[4] = 0** saklanır.

Adım 13 / 15

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
▶    while (i != r) {       /* yolu azalt */
▶      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

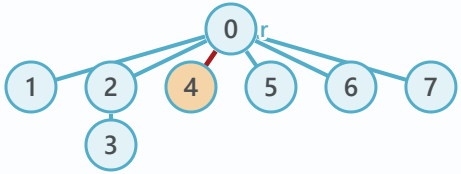
`i = 4` `r = 0` `s = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(5)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(5)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	0	0	0

ata[4] = 0 (zaten köke bağlıydı: fazladan bir tur); i = 0.

Adım 14 / 15

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
5    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
▶      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
▶      i = s;               /* ebeveynine geç */
9    }
10   return r;
11 }
```

DEĞİŞKENLER

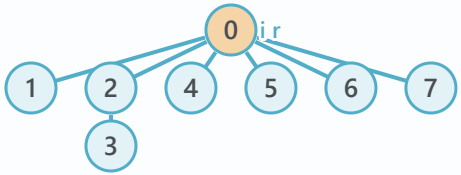
`i = 0` `r = 0` `s = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(5)`

`main()`

Kod izleme: yine `Kume_bul_2(7)`, sonra `Kume_bul_2(5)`



Kume_bul_2(5)

	0	1	2	3	4	5	6	7
ata	-8	0	0	2	0	0	0	0

i == r: döngü biter, **return 0**. ata = [-8, 0, 0, 2, 0, 0, 0, 0].

Adım 15 / 15

```
1  int Kume_bul_2(int i) {
2    int r, s;
3    for (r = i; ata[r] >= 0; r = ata[r])
4      ;                    /* kökü bul */
▶    while (i != r) {       /* yolu azalt */
6      s = ata[i];          /* i'nin ebeveynini sakla */
7      ata[i] = r;          /* i'yi doğrudan köke bağla
   */
8      i = s;               /* ebeveynine geç */
9    }
▶    return r;
11 }
```

DEĞİŞKENLER

`i = 0` `r = 0`

ÇAĞRI YIĞINI

`Kume_bul_2(5)`

`main()`

AZALTMA KURALI

“For vardı, bir de while ekledik!”

Hocadan *“‘For vardı, bir de while döngüsü ekledik. Bu nasıl iyileştirme, anlamadım!’ Ben de anlamadım, bakalım. Küme_bul_2(7) ikinci kez çağrılınca ve sonraki çağrılarda bir çırpıda r=0 bulacaktır.”*

Çağrı	Kume_Bul_1 (ilerleme)	Kume_bul_2 (ilerleme + bağlama)
1. Küme_bul(7)	3	3 + 3
2. Küme_bul(7)	3	1 + 1
10 kez Küme_bul(7)	30	6 + 9·2 = 24
Küme_bul(5) (sonra)	2	2 + 2, ardından 1 + 1

İlk çağrı yaklaşık iki kat pahalıdır, ama yolu kalıcı olarak kısaltır. Maliyet, birçok işlem üzerinden **amortize** edilince neredeyse sabite iner.

Lemma (Tarjan ve van Leeuwen): neredeyse sabit

LEMMA

Her biri tek düğümlü kümelerle başlanıp **f** küme bulma ve **u** birleştirme (ağırlıklı kural + azaltma kuralı) herhangi bir karışık sırada uygulandığında gereken en çok zaman T(f, u) olsun. u ≥ n/2 için:

$$k_1 \cdot (n + f \cdot \alpha(f+n, n)) \leq T(f, u) \leq k_2 \cdot (n + f \cdot \alpha(f+n, n))$$

- α: **ters Ackermann** fonksiyonu; çok yavaş artar.
- Evrendeki atom sayısı kadar öğe için bile uygulamada **α ≤ 4**.
- İşlem başına maliyet n'e bölünce **neredeyse O(1)**.

```
/* Ackermann: çok hızlı büyür */
A(m, n) = n + 1                    m = 0
        = A(m-1, 1)                m > 0, n = 0
        = A(m-1, A(m, n-1))        m > 0, n > 0
```

Hocadan *"1 milyon öğe var; biri O(1)'de buluyor, öteki hâlâ listede arıyor... Hani dedik ya, aynı işe girecek olan matematikçi ile mühendisin farkı budur işte!"*

AZALTMA KURALI

Ackermann fonksiyonu ve tersi

	n = 0	n = 1	n = 2	n = 3	n = 4
m = 0	1	2	3	4	5
m = 1	2	3	4	5	6
m = 2	3	5	7	9	11
m = 3	5	13	29	61	125
m = 4	13	65533	$2^{65536}-3$	…	…

- $A(5, 1) = 2^{2^{\cdot^{\cdot^{2}}}} - 3$ (65536 tane 2'nin üs kulesi): akıl almaz büyüklük.
- Ters: $\alpha(m, n) = \min\{ i \geq 1 : A(i, \lfloor m/n \rfloor) \geq \lg n \}$ — A çok hızlı büyüdüğü için α çok yavaş büyür.

Ackermann fonksiyonunun birden çok tanımı vardır. Yukarıdaki tanımla A(3,4) = 125'tir; Horowitz'in kullandığı tanımda ($A(1,j) = 2^j$ …) ise A(3,1) = 16'dır. Hangi tanım seçilirse seçilsin sonuç değişmez: uygulamada $\alpha \leq 4$.

Sınav sorusu: Ackermann fonksiyonunu özyinelemeli yazınız

SINAV SORUSU

Ackermann fonksiyonunu özyinelemeli olarak yazınız. Ardından A(1, 2) çağrısının çağrı yığınını adım adım gösteriniz.

▶ **Çözüm**

Kod izleme: `ackermann(1, 2)`

etkin çağrılar (soldan sağa derinleşir)

A(1,2)

`ackermann(1, 2)` çağrılır.

Adım 1 / 12

```
▶ unsigned long ackermann(unsigned long m,
2                         unsigned long n) {
3   if (m == 0) return n + 1;
4   if (n == 0) return ackermann(m - 1, 1);
5   return ackermann(m - 1, ackermann(m, n - 1));
6 }
```

DEĞİŞKENLER

`m = 1` `n = 2`

ÇAĞRI YIĞINI

`ackermann(1,2)`

`main()`

Kod izleme: `ackermann(1, 2)`

etkin çağrılar (soldan sağa derinleşir)

A(1,2)

A(1,2): önce içteki A(1,1) hesaplanmalı.

Adım 2 / 12

```
1 unsigned long ackermann(unsigned long m,
2                         unsigned long n) {
3   if (m == 0) return n + 1;
4   if (n == 0) return ackermann(m - 1, 1);
▶   return ackermann(m - 1, ackermann(m, n - 1));
6 }
```

DEĞİŞKENLER

`m = 1` `n = 2`

ÇAĞRI YIĞINI

`ackermann(1,2)`

`main()`

Kod izleme: `ackermann(1, 2)`

etkin çağrılar (soldan sağa derinleşir)



A(1,1): önce içteki A(1,0) hesaplanmalı.

Adım 3 / 12

```
1 unsigned long ackermann(unsigned long m,
2                         unsigned long n) {
3   if (m == 0) return n + 1;
4   if (n == 0) return ackermann(m - 1, 1);
▶   return ackermann(m - 1, ackermann(m, n - 1));
6 }
```

DEĞİŞKENLER

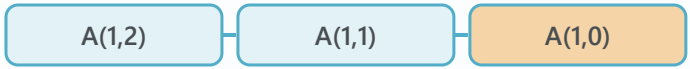
`m = 1` `n = 1`

ÇAĞRI YIĞINI

- `ackermann(1,1)`
- `ackermann(1,2)`
- `main()`

Kod izleme: `ackermann(1, 2)`

etkin çağrılar (soldan sağa derinleşir)



A(1,0): n = 0 → A(0, 1) çağrılır.

Adım 4 / 12

```
1 unsigned long ackermann(unsigned long m,
2                         unsigned long n) {
3   if (m == 0) return n + 1;
▶   if (n == 0) return ackermann(m - 1, 1);
5   return ackermann(m - 1, ackermann(m, n - 1));
6 }
```

DEĞİŞKENLER

`m = 1` `n = 0`

ÇAĞRI YIĞINI

- `ackermann(1,0)`
- `ackermann(1,1)`
- `ackermann(1,2)`
- `main()`

Kod izleme: `ackermann(1, 2)`

etkin çağrılar (soldan sağa derinleşir)



A(0,1) = 2

A(0,1): m = 0 → **return 2**.

Adım 5 / 12

```
1 unsigned long ackermann(unsigned long m,
2                         unsigned long n) {
▶   if (m == 0) return n + 1;
4   if (n == 0) return ackermann(m - 1, 1);
5   return ackermann(m - 1, ackermann(m, n - 1));
6 }
```

DEĞİŞKENLER

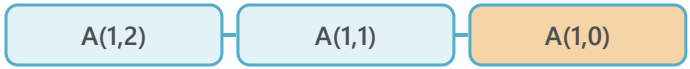
`m = 0` `n = 1`

ÇAĞRI YIĞINI

- **ackermann(0,1)**
- ackermann(1,0)
- ackermann(1,1)
- ackermann(1,2)
- main()

Kod izleme: `ackermann(1, 2)`

etkin çağrılar (soldan sağa derinleşir)



A(1,0) = 2

A(1,0) = A(0,1) = **2**, dönülür.

Adım 6 / 12

```
1 unsigned long ackermann(unsigned long m,
2                         unsigned long n) {
3   if (m == 0) return n + 1;
    if (n == 0) return ackermann(m - 1, 1);
5   return ackermann(m - 1, ackermann(m, n - 1));
6 }
```

DEĞİŞKENLER

`m = 1` `n = 0` `sonuç = 2`

ÇAĞRI YIĞINI

- `ackermann(1,0)`
- `ackermann(1,1)`
- `ackermann(1,2)`
- `main()`

Kod izleme: `ackermann(1, 2)`

etkin çağrılar (soldan sağa derinleşir)



İç çağrı 2 verdi → dış çağrı A(0,2).

Adım 7 / 12

```
1 unsigned long ackermann(unsigned long m,
2                         unsigned long n) {
3   if (m == 0) return n + 1;
4   if (n == 0) return ackermann(m - 1, 1);
▶   return ackermann(m - 1, ackermann(m, n - 1));
6 }
```

DEĞİŞKENLER

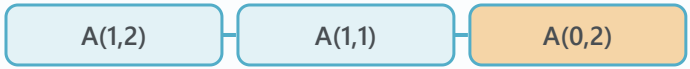
`m = 1` `n = 1` `iç = 2`

ÇAĞRI YIĞINI

- `ackermann(1,1)`
- `ackermann(1,2)`
- `main()`

Kod izleme: `ackermann(1, 2)`

etkin çağrılar (soldan sağa derinleşir)



A(0,2) = 3

A(0,2): m = 0 → **return 3**.

Adım 8 / 12

```
1 unsigned long ackermann(unsigned long m,
2                         unsigned long n) {
3   if (m == 0) return n + 1;
4   if (n == 0) return ackermann(m - 1, 1);
5   return ackermann(m - 1, ackermann(m, n - 1));
6 }
```

DEĞİŞKENLER

`m = 0` `n = 2`

ÇAĞRI YIĞINI

- `ackermann(0,2)`
- `ackermann(1,1)`
- `ackermann(1,2)`
- `main()`

Kod izleme: `ackermann(1, 2)`

etkin çağrılar (soldan sağa derinleşir)



A(1,1) = 3

A(1,1) = **3**.

Adım 9 / 12

```
1 unsigned long ackermann(unsigned long m,
2                         unsigned long n) {
3   if (m == 0) return n + 1;
4   if (n == 0) return ackermann(m - 1, 1);
▶   return ackermann(m - 1, ackermann(m, n - 1));
6 }
```

DEĞİŞKENLER

`m = 1` `n = 1` `sonuç = 3`

ÇAĞRI YIĞINI

- `ackermann(1,1)`
- `ackermann(1,2)`
- `main()`

Kod izleme: `ackermann(1, 2)`

etkin çağrılar (soldan sağa derinleşir)

A(1,2)

İç çağrı 3 verdi → dış çağrı A(0,3).

Adım 10 / 12

```
1 unsigned long ackermann(unsigned long m,
2                         unsigned long n) {
3   if (m == 0) return n + 1;
4   if (n == 0) return ackermann(m - 1, 1);
▶   return ackermann(m - 1, ackermann(m, n - 1));
6 }
```

DEĞİŞKENLER

`m = 1` `n = 2` `iç = 3`

ÇAĞRI YIĞINI

`ackermann(1,2)`

`main()`

Kod izleme: `ackermann(1, 2)`

etkin çağrılar (soldan sağa derinleşir)



A(0,3) = 4

A(0,3): m = 0 → **return 4**.

Adım 11 / 12

```
1 unsigned long ackermann(unsigned long m,
2                         unsigned long n) {
3   if (m == 0) return n + 1;
4   if (n == 0) return ackermann(m - 1, 1);
5   return ackermann(m - 1, ackermann(m, n - 1));
6 }
```

DEĞİŞKENLER

`m = 0` `n = 3`

ÇAĞRI YIĞINI

- `ackermann(0,3)`
- `ackermann(1,2)`
- `main()`

Kod izleme: `ackermann(1, 2)`

etkin çağrılar (soldan sağa derinleşir)

A(1,2)

A(1,2) = 4

A(1,2) = **4**.

Adım 12 / 12

```
1 unsigned long ackermann(unsigned long m,
2                         unsigned long n) {
3   if (m == 0) return n + 1;
4   if (n == 0) return ackermann(m - 1, 1);
▶   return ackermann(m - 1, ackermann(m, n - 1));
6 }
```

DEĞİŞKENLER

m = 1 | n = 2 | sonuç = 4

ÇAĞRI YIĞINI

ackermann(1,2)

main()

Bir parantez: bu lemmalar nereden geliyor?

Hocadan *"Tarjan ünlü bir graf teoricidir, birçok teorisi var. Bu lemma, Tarjan ile Jan van Leeuwen'in ortak çalışmasından çıktı (1984)."*

Hocadan *"Lisansta 4 yıl okunup bir meslek kazanılıyor, doktorada ise yalnızca bir problem üzerine 4 yıl çalışılıyor. Dünyada çözülememiş bir problem çözülüyor [...] O yüzden grup içinde çalışılırsa sürekli öğrenilir."*

Hocadan *"Bu konular bir haftada bitmez; hiç çalışılmazsa hiç bitmez. Her gün düzenli olarak 2–3 saat çalışmak gerekir. Haftanın konusu hemen o hafta tekrar edilmeli."*

UYGULAMA · KODU ADIM ADIM ÇALIŞTIR

Kod izleme: bağlantılı bileşen sayma



	0	1	2	3	4	5
ata	-1	-1	-1	-1	-1	-1

Kenarlar u–v: (0,1), (2,3), (1,4), (3,2). Kaç bağlantılı bileşen var?

Adım 1 / 11

```
▶ int bilesen_say(int n, int m, int u[], int v[]) {
2   for (int i = 0; i < n; i++) ata[i] = -1;
3   int sayi = n;               /* her düğüm ayrı */
4   for (int e = 0; e < m; e++) {
5     int a = Kume_bul_2(u[e]), b = Kume_bul_2(v[e]);
6     if (a != b) {             /* farklı bileşen */
7       Birlesim_2(a, b);
8       sayi--;
9     }
10  }
11  return sayi;
12 }
```

DEĞİŞKENLER

`n = 6` `m = 4`

ÇAĞRI YIĞINI

`bilesen_say(6, 4, u, v)`

`main()`

Kod izleme: bağlantılı bileşen sayma



sayi = 6

	0	1	2	3	4	5
ata	-1	-1	-1	-1	-1	-1

Her düğüm kendi kümesi: ata = −1, **sayi = 6**.

Adım 2 / 11

```
1  int bilesen_say(int n, int m, int u[], int v[]) {
▶    for (int i = 0; i < n; i++) ata[i] = -1;
▶    int sayi = n;                /* her düğüm ayrı */
4    for (int e = 0; e < m; e++) {
5      int a = Kume_bul_2(u[e]), b = Kume_bul_2(v[e]);
6      if (a != b) {              /* farklı bileşen */
7        Birlesim_2(a, b);
8        sayi--;
9      }
10   }
11   return sayi;
12 }
```

DEĞİŞKENLER

`n = 6` `m = 4` `sayi = 6`

ÇAĞRI YIĞINI

`bilesen_say(6, 4, u, v)`

`main()`

Kod izleme: bağlantılı bileşen sayma



sayi = 6

	0	1	2	3	4	5
ata	-1	-1	-1	-1	-1	-1

Kenar (0, 1): Kume_bul_2(0) = 0, Kume_bul_2(1) = 1.

Adım 3 / 11

```
1  int bilesen_say(int n, int m, int u[], int v[]) {
2    for (int i = 0; i < n; i++) ata[i] = -1;
3    int sayi = n;               /* her düğüm ayrı */
4    for (int e = 0; e < m; e++) {
▶      int a = Kume_bul_2(u[e]), b = Kume_bul_2(v[e]);
6      if (a != b) {             /* farklı bileşen */
7        Birlesim_2(a, b);
8        sayi--;
9      }
10   }
11   return sayi;
12 }
```

DEĞİŞKENLER

`e = 0` `u[e] = 0` `v[e] = 1`

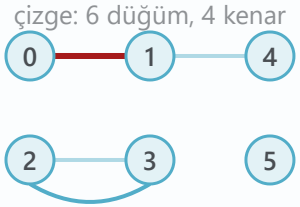
`a = 0` `b = 1` `sayi = 6`

ÇAĞRI YIĞINI

`bilesen_say(6, 4, u, v)`

`main()`

Kod izleme: bağlantılı bileşen sayma





sayi = 5

	0	1	2	3	4	5
ata	-2	0	-1	-1	-1	-1

0 ≠ 1: farklı bileşenler → **Birlesim_2(0, 1)**, sayi = 5.

Adım 4 / 11

```
1  int bilesen_say(int n, int m, int u[], int v[]) {
2    for (int i = 0; i < n; i++) ata[i] = -1;
3    int sayi = n;                /* her düğüm ayrı */
4    for (int e = 0; e < m; e++) {
5      int a = Kume_bul_2(u[e]), b = Kume_bul_2(v[e]);
▶      if (a != b) {              /* farklı bileşen */
▶        Birlesim_2(a, b);
▶        sayi--;
9      }
10   }
11   return sayi;
12 }
```

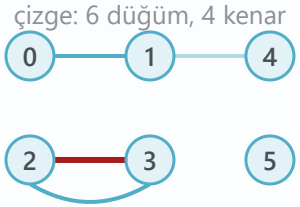
DEĞİŞKENLER

e = 0 | a = 0 | b = 1 | sayi = 5

ÇAĞRI YIĞINI

- bilesen_say(6, 4, u, v)
- main()

Kod izleme: bağlantılı bileşen sayma





sayi = 5

	0	1	2	3	4	5
ata	-2	0	-1	-1	-1	-1

Kenar (2, 3): Kume_bul_2(2) = 2, Kume_bul_2(3) = 3.

Adım 5 / 11

```
1  int bilesen_say(int n, int m, int u[], int v[]) {
2    for (int i = 0; i < n; i++) ata[i] = -1;
3    int sayi = n;                /* her düğüm ayrı */
4    for (int e = 0; e < m; e++) {
▶      int a = Kume_bul_2(u[e]), b = Kume_bul_2(v[e]);
6      if (a != b) {              /* farklı bileşen */
7        Birlesim_2(a, b);
8        sayi--;
9      }
10   }
11   return sayi;
12 }
```

DEĞİŞKENLER

`e = 1` `u[e] = 2` `v[e] = 3`

`a = 2` `b = 3` `sayi = 5`

ÇAĞRI YIĞINI

`bilesen_say(6, 4, u, v)`

`main()`

Kod izleme: bağlantılı bileşen sayma



sayi = 4

	0	1	2	3	4	5
ata	-2	0	-2	2	-1	-1

2 ≠ 3: farklı bileşenler → **Birlesim_2(2, 3)**, sayi = 4.

Adım 6 / 11

```
1  int bilesen_say(int n, int m, int u[], int v[]) {
2    for (int i = 0; i < n; i++) ata[i] = -1;
3    int sayi = n;                /* her düğüm ayrı */
4    for (int e = 0; e < m; e++) {
5      int a = Kume_bul_2(u[e]), b = Kume_bul_2(v[e]);
▶      if (a != b) {              /* farklı bileşen */
▶        Birlesim_2(a, b);
▶        sayi--;
9      }
10   }
11   return sayi;
12 }
```

DEĞİŞKENLER

e = 1 | a = 2 | b = 3 | sayi = 4

ÇAĞRI YIĞINI

bilesen_say(6, 4, u, v)

main()

Kod izleme: bağlantılı bileşen sayma



sayi = 4

	0	1	2	3	4	5
ata	-2	0	-2	2	-1	-1

Kenar (1, 4): Kume_bul_2(1) = 0, Kume_bul_2(4) = 4.

Adım 7 / 11

```
1  int bilesen_say(int n, int m, int u[], int v[]) {
2    for (int i = 0; i < n; i++) ata[i] = -1;
3    int sayi = n;                /* her düğüm ayrı */
4    for (int e = 0; e < m; e++) {
▶      int a = Kume_bul_2(u[e]), b = Kume_bul_2(v[e]);
6      if (a != b) {              /* farklı bileşen */
7        Birlesim_2(a, b);
8        sayi--;
9      }
10   }
11   return sayi;
12 }
```

DEĞİŞKENLER

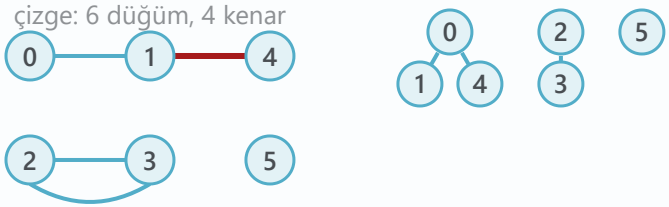
e = 2 | u[e] = 1 | v[e] = 4 | a = 0 | b = 4 | sayi = 4

ÇAĞRI YIĞINI

bilesen_say(6, 4, u, v)

main()

Kod izleme: bağlantılı bileşen sayma



sayi = 3

	0	1	2	3	4	5
ata	-3	0	-2	2	0	-1

0 ≠ 4: farklı bileşenler → **Birlesim_2(0, 4)**, sayi = 3.

Adım 8 / 11

```
1  int bilesen_say(int n, int m, int u[], int v[]) {
2    for (int i = 0; i < n; i++) ata[i] = -1;
3    int sayi = n;                /* her düğüm ayrı */
4    for (int e = 0; e < m; e++) {
5      int a = Kume_bul_2(u[e]), b = Kume_bul_2(v[e]);
▶      if (a != b) {              /* farklı bileşen */
▶        Birlesim_2(a, b);
▶        sayi--;
9      }
10   }
11   return sayi;
12 }
```

DEĞİŞKENLER

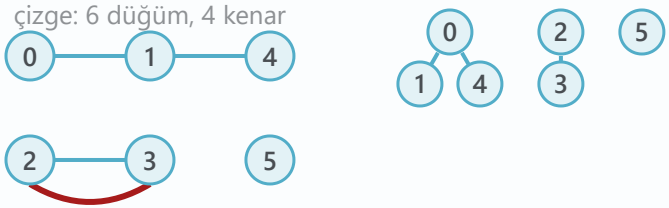
e = 2 | a = 0 | b = 4 | sayi = 3

ÇAĞRI YIĞINI

bilesen_say(6, 4, u, v)

main()

Kod izleme: bağlantılı bileşen sayma



sayi = 3

	0	1	2	3	4	5
ata	-3	0	-2	2	0	-1

Kenar (3, 2): Kume_bul_2(3) = 2, Kume_bul_2(2) = 2.

Adım 9 / 11

```
1  int bilesen_say(int n, int m, int u[], int v[]) {
2    for (int i = 0; i < n; i++) ata[i] = -1;
3    int sayi = n;                /* her düğüm ayrı */
4    for (int e = 0; e < m; e++) {
▶      int a = Kume_bul_2(u[e]), b = Kume_bul_2(v[e]);
6      if (a != b) {              /* farklı bileşen */
7        Birlesim_2(a, b);
8        sayi--;
9      }
10   }
11   return sayi;
12 }
```

DEĞİŞKENLER

`e = 3` `u[e] = 3` `v[e] = 2` `a = 2` `b = 2` `sayi = 3`

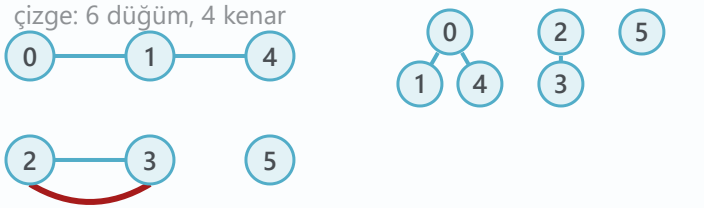
ÇAĞRI YIĞINI

`bilesen_say(6, 4, u, v)`

`main()`

UYGULAMA · KODU ADIM ADIM ÇALIŞTIR

Kod izleme: bağlantılı bileşen sayma



sayi = 3

	0	1	2	3	4	5
ata	-3	0	-2	2	0	-1

a == b = 2: 3 ile 2 zaten aynı bileşende; kenar bir **döngü** kapatır, sayı değişmez.

Adım 10 / 11

```
1  int bilesen_say(int n, int m, int u[], int v[]) {
2    for (int i = 0; i < n; i++) ata[i] = -1;
3    int sayi = n;                /* her düğüm ayrı */
4    for (int e = 0; e < m; e++) {
5      int a = Kume_bul_2(u[e]), b = Kume_bul_2(v[e]);
▶      if (a != b) {              /* farklı bileşen */
7        Birlesim_2(a, b);
8        sayi--;
9      }
10   }
11   return sayi;
12 }
```

DEĞİŞKENLER

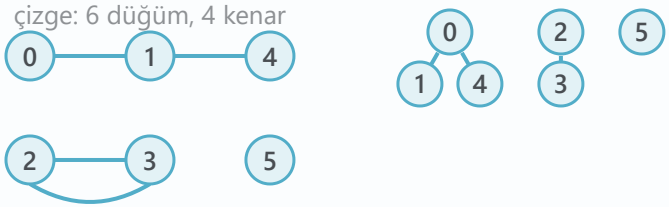
e = 3 | a = 2 | b = 2 | sayi = 3

ÇAĞRI YIĞINI

bilesen_say(6, 4, u, v)

main()

Kod izleme: bağlantılı bileşen sayma



sayi = 3

	0	1	2	3	4	5
ata	-3	0	-2	2	0	-1

return 3: bileşenler {0,1,4}, {2,3}, {5}. Toplam O(n + m·α).

Adım 11 / 11

```
1  int bilesen_say(int n, int m, int u[], int v[]) {
2    for (int i = 0; i < n; i++) ata[i] = -1;
3    int sayi = n;                /* her düğüm ayrı */
4    for (int e = 0; e < m; e++) {
5      int a = Kume_bul_2(u[e]), b = Kume_bul_2(v[e]);
6      if (a != b) {              /* farklı bileşen */
7        Birlesim_2(a, b);
8        sayi--;
9      }
10   }
▶    return sayi;
12 }
```

DEĞİŞKENLER

`sayi = 3`

ÇAĞRI YIĞINI

`bilesen_say(6, 4, u, v)`

`main()`

UYGULAMALAR

Union-find başka nerelerde?

Eşdeğerlik sınıfları
i ≡ j bildirimlerinden sınıfları çıkarmak (Horowitz'in eşdeğerlik problemi).

Ağ bağlantısı
"Bu iki bilgisayar bağlı mı?" sorularını çevrimiçi yanıtlamak (Sedgewick'in açılış örneği).

Görüntü işleme
Aynı renkteki komşu pikselleri bölgelere ayırmak (bağlantılı bileşen etiketleme).

Süzülme (percolation)
Izgarada üstten alta açık yol var mı? Monte Carlo deneylerinin çekirdeği.

Derleyiciler
Tip çıkarımında birleştirme (unification), takma ad (alias) analizi.

Çevrimdışı LCA
Tarjan'ın en yakın ortak ata algoritması.

Kruskal algoritması: kenarlar ağırlığa göre sıralanır; bir kenarın uçları için Küme_bul farklı kökler verirse kenar ağaca alınır ve iki küme birleştirilir, aynı kökü verirse kenar bir döngü kapatır ve reddedilir. Maliyet: sıralama $O(m \lg m)$ + union-find ≈ $O(m \cdot \alpha)$.

ÖZET

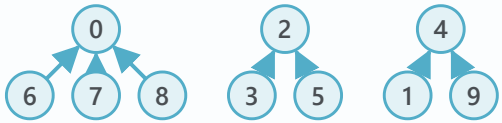
Sınav sorusu: ebeveyn dizisinin içeriği

SINAV SORUSU

Birlesim_2 ve Kume_bul_2 verilmiştir; kümeler $S_1 = \{0, 6, 7, 8\}$, $S_2 = \{1, 4, 9\}$, $S_3 = \{2, 3, 5\}$ (şekildeki ağaçlar).

(i) İlk başta `ata` dizisinin içeriği nedir?

(ii) `Birlesim_2(4, 2); Kume_bul_2(5); Birlesim_2(0, 4); Kume_bul_2(7); Kume_bul_2(3);` çağrılarından sonra `ata` ne olur? *Not: (i) hatalıysa (ii) değerlendirilmez.*



▶ **Çözüm**

ÖZET

Karmaşıklık özeti

Sürüm	Birleşim	Küme bul (en kötü)	n−1 birleşim + m bulma
Birlesim_1 + Kume_Bul_1	O(1)	O(n)	O(n + m·n)
Birlesim_2 (ağırlıklı)	O(1)	O(lg n)	O(n + m lg n)
Birlesim_2 + Kume_bul_2	O(1)	O(lg n) tek işlem	O(n + m·α(m+n, n))

Bellek: yalnızca n tamsayılık `ata` dizisi. $\alpha \leq 4$ olduğundan son satır uygulamada doğrusaldır.

Birleşim tek başına hep O(1)'dir; maliyeti belirleyen, ağacın derinliği yani Küme_bul'dur. Ağırlıklı kural derinliği sınırlar, azaltma kuralı yolları kısaltır.

ÖZET

Sık yapılan hatalar

1. Birleşime kök yerine rastgele düğüm vermek: `ata[i] = j` i'nin alt ağacını koparır.
2. Aynı kümedeki iki kökü birleştirmek (i == j): kök kendini gösterir, sonsuz döngü.
3. Ağırlıklı kuralda negatif değerlerin karşılaştırmasını ters yorumlamak (−2 > −4 “daha az düğüm”).
4. Birleşimde kökteki düğüm sayısını (p) günlemeyi unutmak.
5. Kume_bul_2'de ebeveyni saklamadan (s) bağlamak: yol kaybolur.
6. Başlangıçta `ata` dizisini −1 ile doldurmamak.

ÖZET

Alıştırmalar

1. n = 6, ata tümüyle −1. `Birlesim_2(0,1); Birlesim_2(2,3); Birlesim_2(0,2); Birlesim_2(4,5)` sonrası ata?
2. Ağırlıklı kural kullanılırken 16 düğümlü bir ağacın derinliği en çok kaçtır?
3. Ağırlık yerine **yükseklik** (rank) tutan bir birleşim kuralı yazın. Ne değişir?
4. Kume_bul_2'yi özyinelemeli yazın (tek satırlık yol sıkıştırma).
5. Kruskal'da bir kenarın reddedilmesi neden döngü demektir?
6. n düğüm, m kenarlı bir çizgede bileşen sayısını bulmanın maliyeti nedir?

▶ **Cevapları göster**

ÖZET

Bölüm özeti

Model

Ayrık kümeler = kökü küme adı olan, çocuktan ebeveyne bağlı ağaçlar.

Gösterim

Tek `ata` dizisi; kökte negatif değer.

Kötü durum

Rastgele bağlama zincir üretir: n bulma $O(n^2)$.

Ağırlıklı kural

Küçük ağaç büyüğe: derinlik ≤ ⌊lg m⌋ + 1.

Azaltma kuralı

Yol köke bağlanır; sonraki aramalar bir çırpıda.

Sonuç

İkisi birlikte: işlem başına $\alpha \leq 4$ — neredeyse $O(1)$.

ÖZET

Kaynaklar ve devam

- E. Horowitz, S. Sahni, S. Anderson-Freed, *Fundamentals of Data Structures in C*, Bölüm 5.10 (Set Representation).
- T. H. Cormen vd., *Introduction to Algorithms* (CLRS), "Data Structures for Disjoint Sets" bölümü.
- R. Sedgewick, *Algorithms in C*, Bölüm 1 (union-find ile bağlantılılık).
- R. E. Tarjan, J. van Leeuwen, "Worst-case analysis of set union algorithms", *J. ACM* 31(2), 1984.

← Bölüm 15: Çizgeler

Ders sayfası

Bölüm 17: Minimum Kapsayan Ağaçlar →

Son

Bölüm 16: Kümeler ve Union-Find

Sonraki bölüm: Bölüm 17: Minimum Kapsayan Ağaçlar · Ders sayfası