Bölüm 12: Kılavuzlu İkili Ağaçlar

C ile Veri Yapıları ve Algoritmalar

n+1 NULL işaretçi | Kurma kuralları | Başlık düğümü | Ardıl bulma | Yığınsız gezinme | Sağa / sola ekleme | Morris gezinmesi

Kaynak: Horowitz–Sahni–Anderson-Freed 5.5

GİRİŞ

Öğrenme Çıktıları

Boş işaretçileri saymak

n düğümlü ikili ağaçta n+1 NULL işaretçi olduğunu kanıtlamak.

Kılavuz kurmak

Sol NULL → kök ortada öncül, sağ NULL → kök ortada ardıl kuralını uygulamak.

Ayırt etmek

solKilavuz / sagKilavuz bayrakları ve başlık düğümüyle yapıyı eksiksiz tanımlamak.

Yığınsız gezinmek

`kokortada_ardil_bul` ile O(1) ek bellekle kök ortada (ve kök önde) gezinmek.

Ekleme yapmak

`saga_ekle`'nin iki durumunu tek yordamda birleştirmek; `sola_ekle`'yi türetmek.

İyi kod yazmak

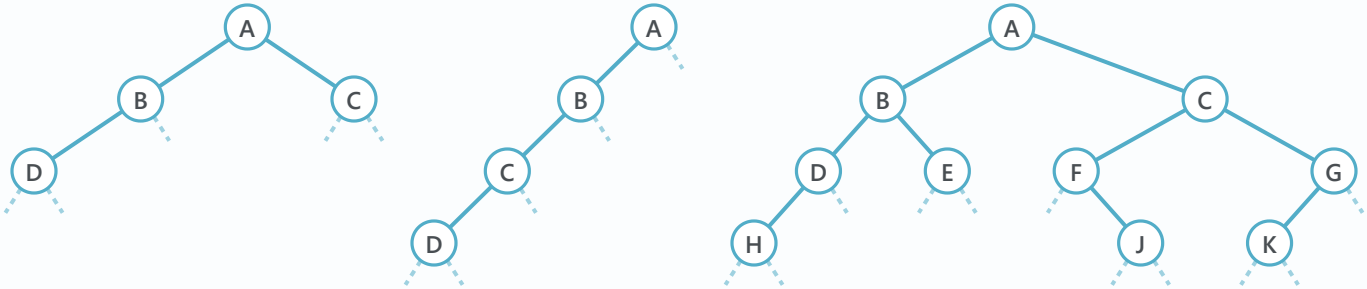
Açık tanımlı yordam, anlamlı açıklama satırı, NULL denetiminin yeri.

NEDEN KILAVUZ?

n düğümde n+1 NULL işaretçi

- Her düğümde 2 işaretçi alanı → ağaçta toplam **2n** işaretçi.
- Kök dışındaki her düğüme ebeveyninden bir işaretçi gelir → **n − 1** işaretçi dolu.
- Geri kalan **2n − (n−1) = n + 1** işaretçi NULL: işaretçilerin yarıdan fazlası boşa!

> **"A year spent in artificial intelligence is enough to make one believe in God."**
> — Alan J. Perlis



`n = 4 NULL = 5` `n = 4 NULL = 5` `n = 10 NULL = 11`

Kesikli kısa çizgiler NULL işaretçiler; üç örnekte de NULL sayısı n + 1.

NEDEN KILAVUZ?

Fikir: boş işaretçileri yol gösterici yapmak

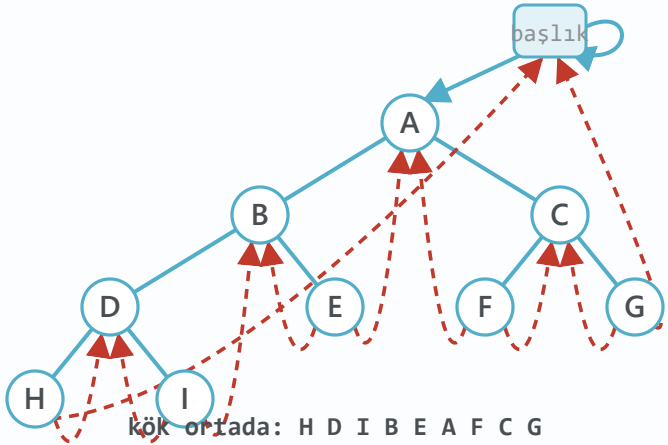
- A. J. Perlis ve C. Thornton, NULL işaretçilerin ağaçtaki **diğer düğümleri** göstermek için kullanılabileceğini gösterdiler; bu işaretçilere **kılavuz** (thread) dediler.
- Kazanç: kök ortada gezinme için **yığın ya da özyineleme gerekmez**; her düğümden ardılına doğrudan gidilir.
- Bedel: her işaretçinin çocuk mu kılavuz mu olduğunu söyleyen **birer bit**.

Özyinelemeli gezinme O(h) çağrı yığını, yinelemeli gezinme O(h) yığın kullanır. Kılavuzlu ağaçta ek bellek **O(1)**.

KURMA KURALLARI

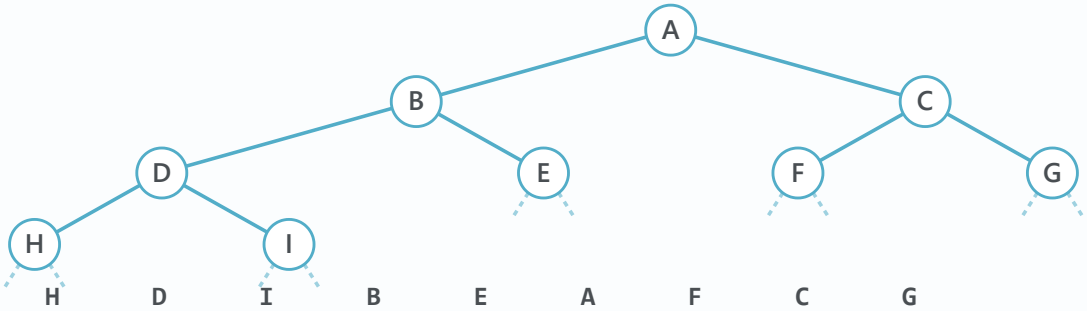
Kılavuzlu ikili ağaç kurma kuralları

1. `p->solcocuk == NULL` ise `p->solcocuk`, kök ortada gezinmede p'den **önce** ziyaret edilen öğeyi gösterir: NULL, p'nin **kök ortada öncülü** (inorder predecessor) ile değiştirilir.
2. `p->sagcocuk == NULL` ise `p->sagcocuk`, p'den **sonra** ziyaret edilen öğeyi gösterir: NULL, p'nin **kök ortada ardılı** (inorder successor) ile değiştirilir.



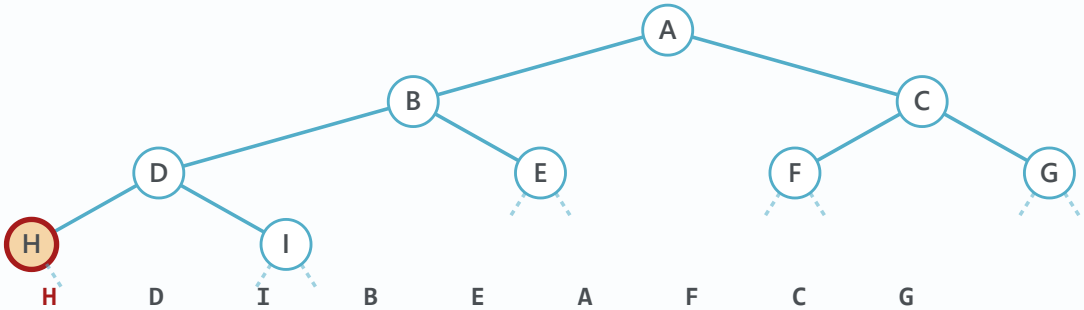
Kesikli kırmızı oklar kılavuzlardır. Kılavuzlar, ağacın kök ortada sırasını ağacın içine "örer".

Kılavuzları tek tek kurmak



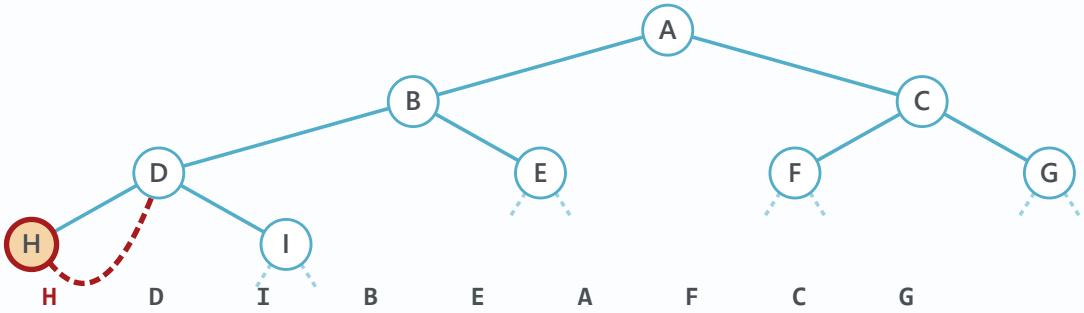
Sıradan ikili ağaç: 9 düğüm, 10 NULL işaretçi (kesikli). Kök ortada sıra: H D I B E A F C G.

Kılavuzları tek tek kurmak



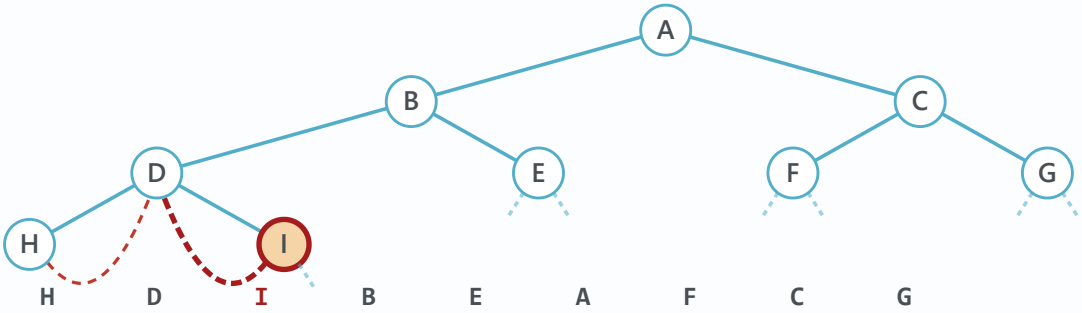
H'nin **sol** işaretçisi NULL → kök ortada **öncülü** başlığı gösterir (öncül/ardıl yok: şimdilik havada, başlık düğümüne bağlanacak).

Kılavuzları tek tek kurmak



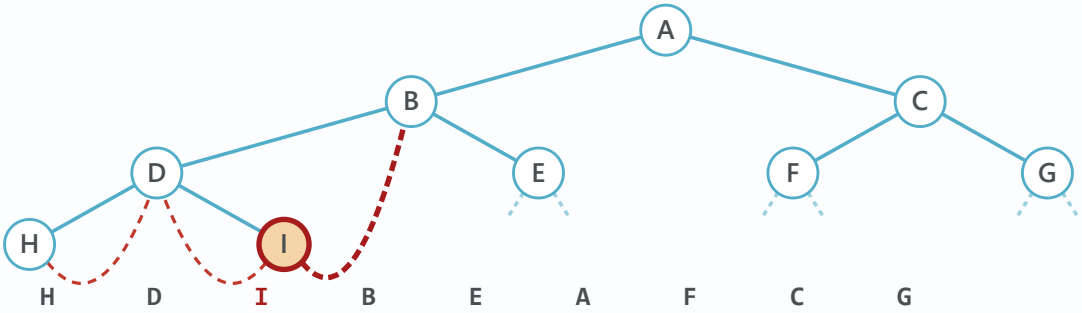
H'nin **sağ** işaretçisi NULL → kök ortada **ardılı** D'yi gösterir.

Kılavuzları tek tek kurmak



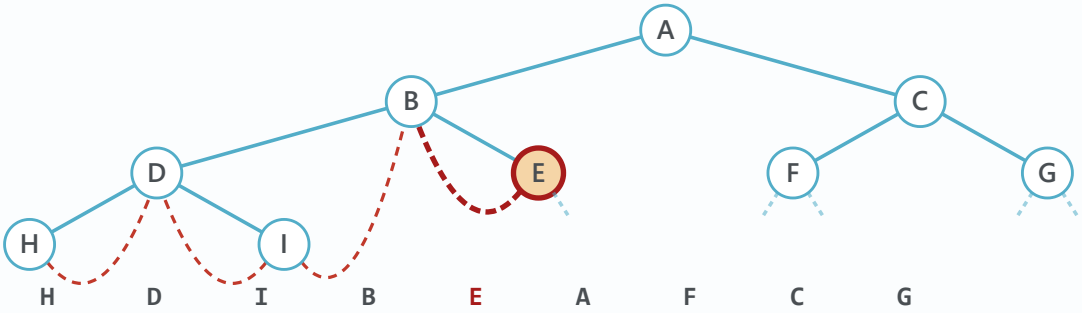
I'nın **sol** işaretçisi NULL → kök ortada **öncülü** D'yi gösterir.

Kılavuzları tek tek kurmak



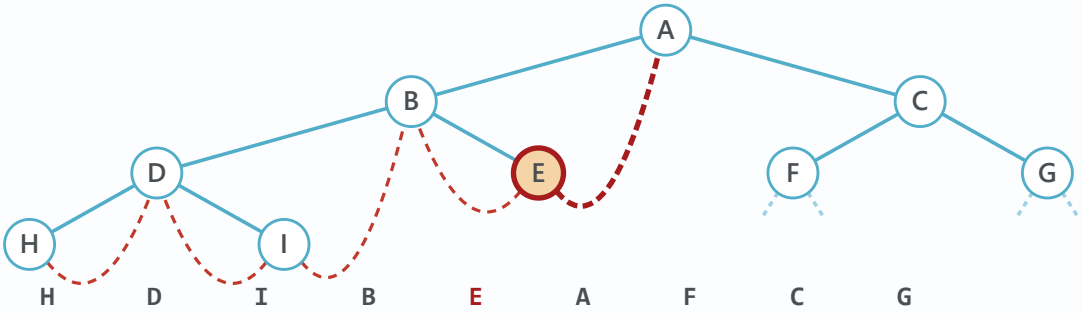
I'nın **sağ** işaretçisi NULL → kök ortada **ardılı** B'yi gösterir.

Kılavuzları tek tek kurmak



E'nin **sol** işaretçisi NULL → kök ortada **öncülü** B'yi gösterir.

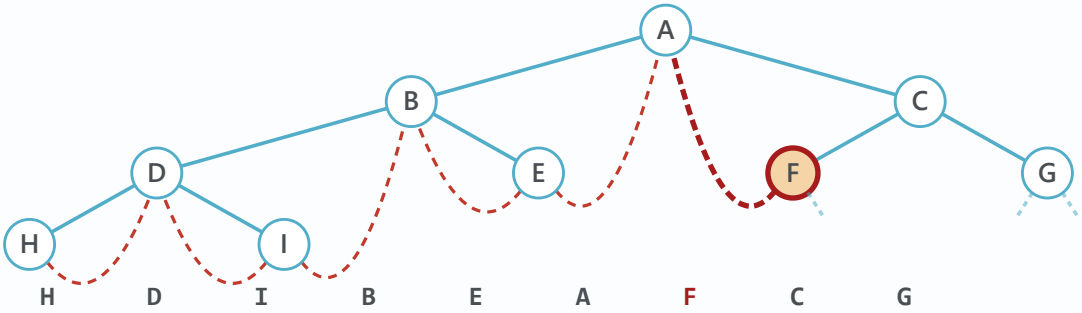
Kılavuzları tek tek kurmak



E'nin **sağ** işaretçisi NULL → kök ortada **ardılı** A'yı gösterir.

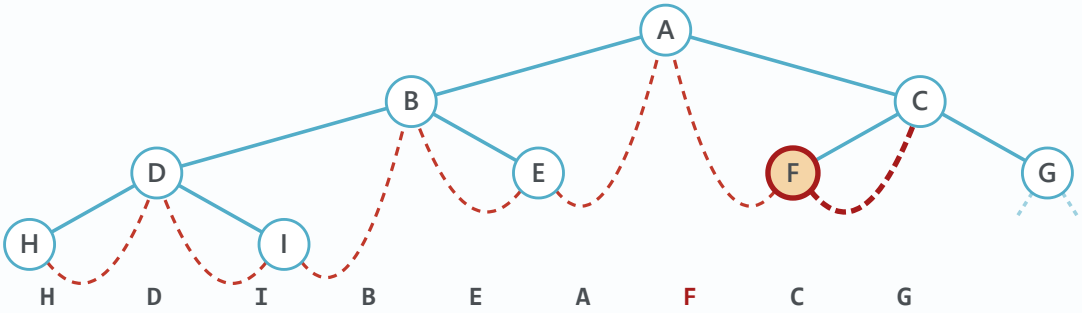
KURMA KURALLARI

Kılavuzları tek tek kurmak



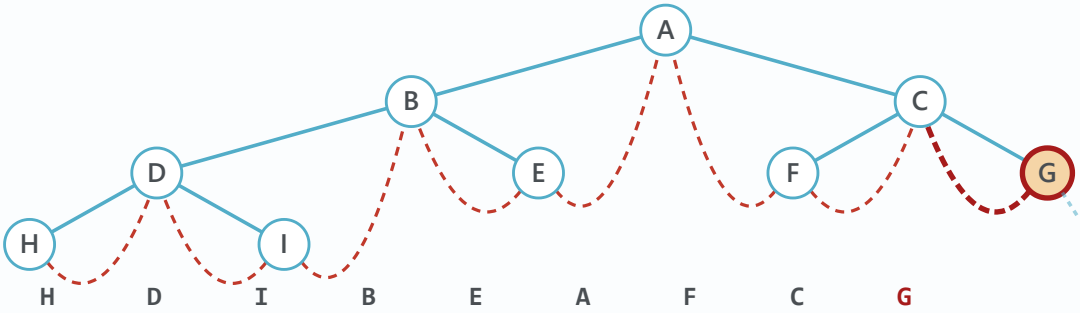
F'nin **sol** işaretçisi NULL → kök ortada **öncülü** A'yı gösterir.

Kılavuzları tek tek kurmak



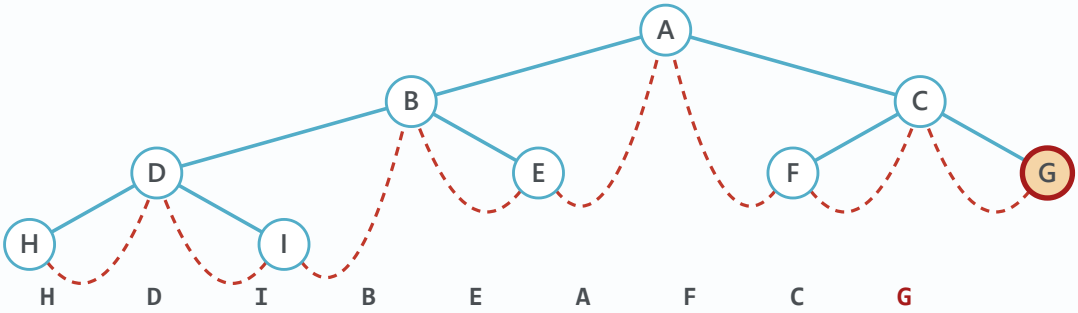
F'nin **sağ** işaretçisi NULL → kök ortada **ardılı** C'yi gösterir.

Kılavuzları tek tek kurmak



G'nin **sol** işaretçisi NULL → kök ortada **öncülü** C'yi gösterir.

Kılavuzları tek tek kurmak



G'nin **sağ** işaretçisi NULL → kök ortada **ardılı** başlığı gösterir (öncül/ardıl yok: şimdilik havada, başlık düğümüne bağlanacak).

KURMA KURALLARI

Kırmızı oku derleyiciye nasıl anlatırız?

```
typedef struct kilavuz_agac *kilavuz_gost;
struct kilavuz_agac {
    short int    solKilavuz; /* TRUE: solCocuk kılavuz */
    kilavuz_gost solCocuk;
    char         veri;
    kilavuz_gost sagCocuk;
    short int    sagKilavuz; /* TRUE: sagCocuk kılavuz */
};
```

Bir struct tanımı kapanıştaki `;` ile biter (satır 8); unutmak sık yapılan bir hatadır. `TRUE/FALSE` : `enum { FALSE, TRUE }` .

HOCADAN "Derleyicide kırmızı pointer var mı? Onların kırmızı olduğunu nereden anlayacağız? Fazladan iki bilgi tutmak gerekir."

HOCADAN "Alan karmaşıklığı artmaz mı? Artsın, zaman karmaşıklığı alan karmaşıklığından daha önemlidir. Kaldı ki, bunu 2 bitle bile halledebilirim."

Havada kalan iki kılavuz ve başlık düğümü

- H'nin sol ve G'nin sağ kılavuzunun gösterecek öğesi yok: **havada asılı**.
- Bir **başlık** öğesi ekleriz: gerçek ağacın kökü `L→solCocuk` ile erişilir.
- Asılı iki kılavuz başlığı gösterir; başlığın sağı hep kendisidir, sağ kılavuzu FALSE.
- Boş ağaçta sol işaretçi de başlığı gösterir, sol kılavuz TRUE; öğe eklenince FALSE yapılır.

solKilavuz	solCocuk	veri	sagCocuk	sagKilavuz
TRUE	→ başlık	–	→ başlık	FALSE

Boş kılavuzlu ağaç

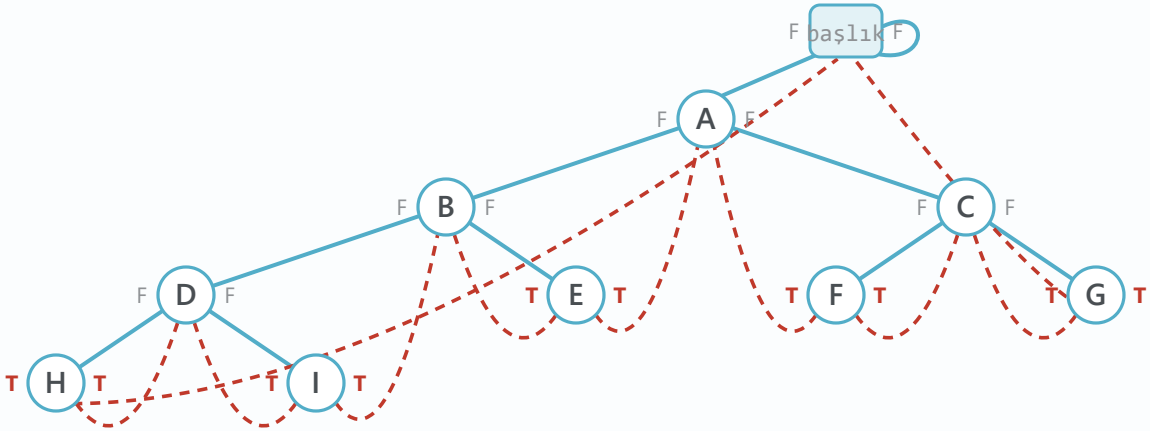
Başlık düğümü: boş ağaç ve dolu ağaç



solCocuk → kendisi (kılavuz, TRUE) sagCocuk → kendisi (FALSE)

Boş ağaç: yalnızca başlık; iki işaretçisi de kendini gösterir.

Başlık düğümü: boş ağaç ve dolu ağaç



Dolu ağaç: başlığın solu köke (A); H'nin sol ve G'nin sağ kılavuzu başlığa bağlandı. T/F: kılavuz bayrakları.

KURMA KURALLARI

Ağaç, kök ortada gezinmeye göre kurulur

HOCADAN "İkili ağacı inorder gezinmeye göre oluşturuyoruz. Preorder gezinmeye göre zaten kurulamaz."

gezinme	sıra
kök ortada (in-order)	H D I B E A F C G
kök önde (pre-order)	A B D H I E C F G
kök sonda (post-order)	H I D E B F G C A

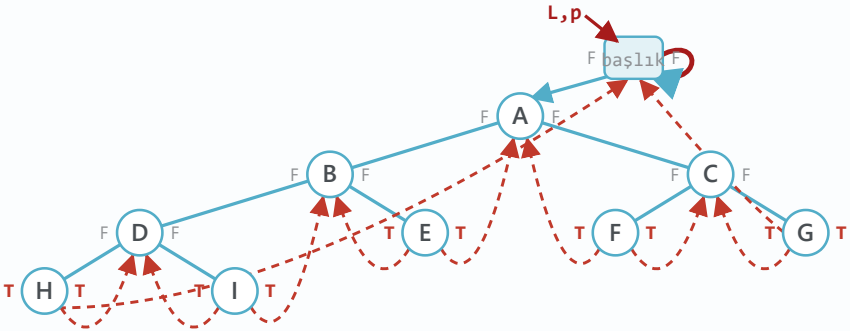
Kılavuzlar kök ortada öncül/ardıla göre kurulur; kök önde ve kök sonda gezinme de (daha dolambaçlı yoldan) bu kılavuzlarla yapılabilir.

Kök ortada ardılı bulmak

1. `p→sagKilavuz == TRUE` ise `p→sagCocuk` (kılavuz olduğu için) zaten p'nin ardılıdır.
2. `p→sagKilavuz == FALSE` ise sağ çocuktan başlanıp `solCocuk` bağları `solKilavuz` TRUE oluncaya dek izlenir: sağ alt ağacın **en soldaki** düğümü ardıldır.

```
kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
  // L ile işaret edilen (NULL olmayan) öğenin ardılı
  kilavuz_gost p = L->sagCocuk;
  if (!L->sagKilavuz)        // sağ alt ağaç var mı
    while (!p->solKilavuz)   // en soluna dek in
      p = p->solCocuk;
  return p;
}
```

kokortada_ardil_bul(): başlık, A ve I için



`kokortada_ardil_bul(başlık)` : p = başlık->sagCocuk = başlık.

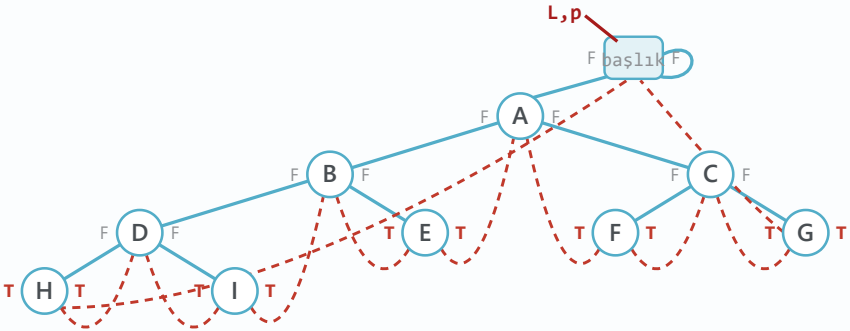

```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
▶   kilavuz_gost p = L->sagCocuk;
4   if (!L->sagKilavuz)        // sağ alt ağaç var mı
5     while (!p->solKilavuz)   // en soluna dek in
6       p = p->solCocuk;
7   return p;
8 }
```

DEĞİŞKENLER

`L = &başlık` `p = &başlık`

`L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



sagKilavuz FALSE: başlığın gerçek bir sağ alt ağacı var; ardıl oranın en solunda.

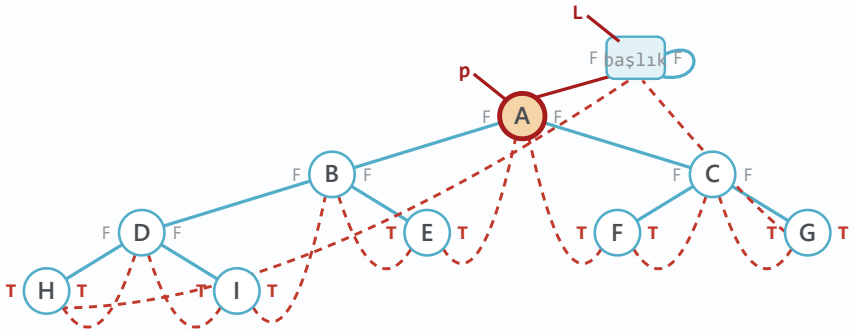

```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
3   kilavuz_gost p = L->sagCocuk;
▶   if (!L->sagKilavuz)        // sağ alt ağaç var mı
5     while (!p->solKilavuz)   // en soluna dek in
6       p = p->solCocuk;
7   return p;
8 }
```

DEĞİŞKENLER

`L = &başlık` `p = &başlık`

`L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



başlık->solKilavuz FALSE → sola: p = A.

Adım 3 / 16

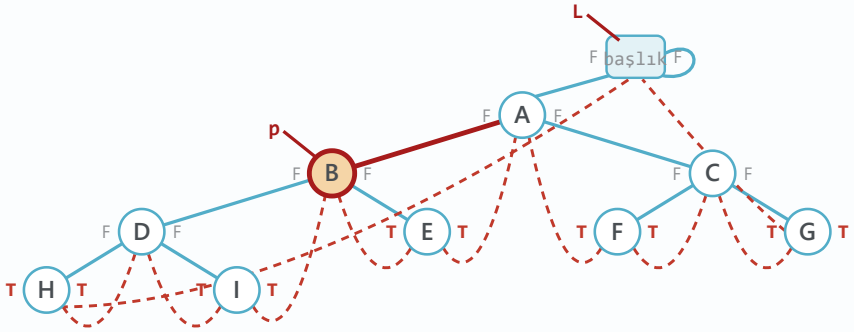
```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
3   kilavuz_gost p = L->sagCocuk;
4   if (!L->sagKilavuz)        // sağ alt ağaç var mı
▶     while (!p->solKilavuz)   // en soluna dek in
▶       p = p->solCocuk;
7   return p;
8 }
```

DEĞİŞKENLER

`L = &başlık` `p = &düğüm(A)`

`L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



A->solKilavuz FALSE → sola: p = B.

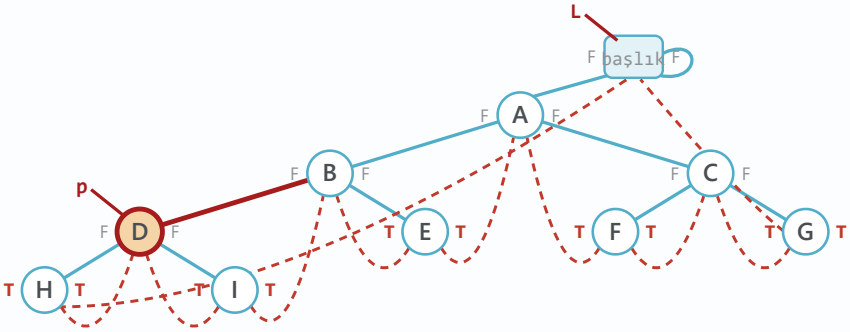
Adım 4 / 16

```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
3   kilavuz_gost p = L->sagCocuk;
4   if (!L->sagKilavuz)        // sağ alt ağaç var mı
▶     while (!p->solKilavuz)   // en soluna dek in
▶       p = p->solCocuk;
7   return p;
8 }
```

DEĞİŞKENLER

`L = &başlık` `p = &düğüm(B)` `L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



B->solKilavuz FALSE → sola: p = D.

Adım 5 / 16

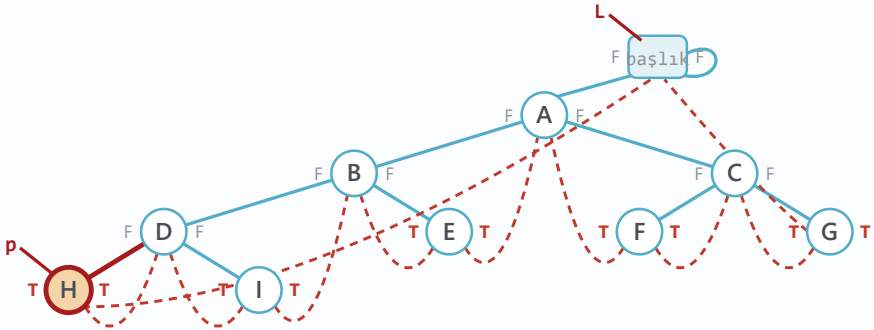
```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
3   kilavuz_gost p = L->sagCocuk;
4   if (!L->sagKilavuz)        // sağ alt ağaç var mı
▶     while (!p->solKilavuz)   // en soluna dek in
▶       p = p->solCocuk;
7   return p;
8 }
```

DEĞİŞKENLER

`L = &başlık` `p = &düğüm(D)`

`L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



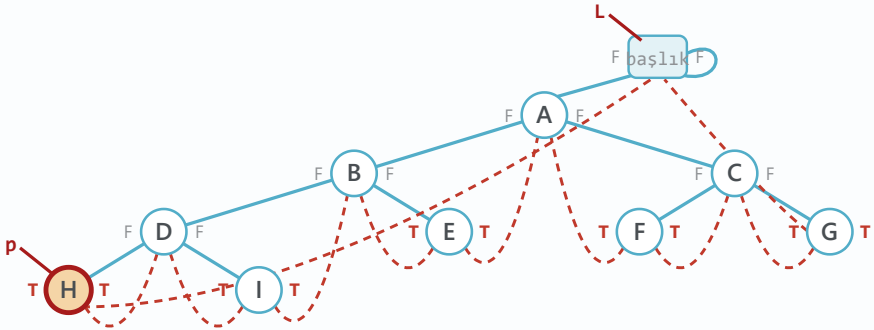
D->solKilavuz FALSE → sola: p = H.


```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
3   kilavuz_gost p = L->sagCocuk;
4   if (!L->sagKilavuz)        // sağ alt ağaç var mı
▶     while (!p->solKilavuz)   // en soluna dek in
▶       p = p->solCocuk;
7   return p;
8 }
```

DEĞİŞKENLER

`L = &başlık` `p = &düğüm(H)` `L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



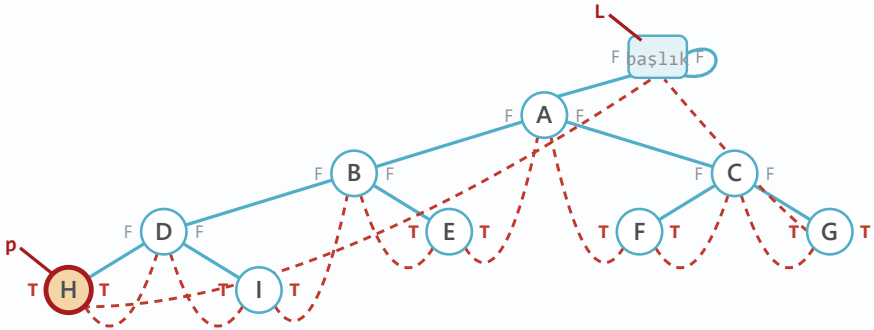
H->solKilavuz TRUE: daha sola gidilemez → en soldaki düğüm.


```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
3   kilavuz_gost p = L->sagCocuk;
4   if (!L->sagKilavuz)        // sağ alt ağaç var mı
▶     while (!p->solKilavuz)   // en soluna dek in
6       p = p->solCocuk;
7   return p;
8 }
```

DEĞİŞKENLER

`L = &başlık` `p = &düğüm(H)` `L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



H döndürülür: başlığın kök ortada ardılı. (Başlığın ardılı = ilk ziyaret edilen düğüm.)

Adım 8 / 16

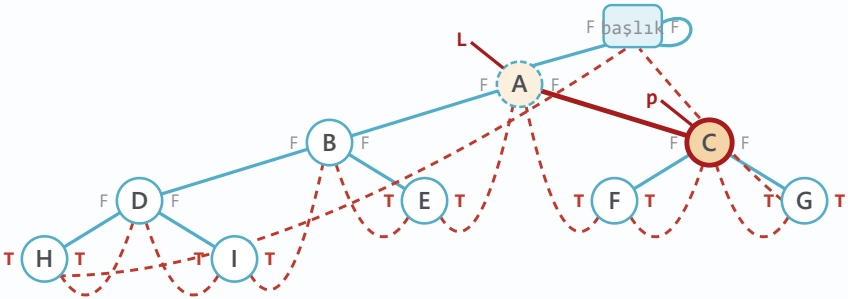
```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
3   kilavuz_gost p = L->sagCocuk;
4   if (!L->sagKilavuz)         // sağ alt ağaç var mı
5     while (!p->solKilavuz)    // en soluna dek in
6       p = p->solCocuk;
▶   return p;
8 }
```

DEĞİŞKENLER

`L = &başlık` `p = &düğüm(H)`

`L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



`kokortada_ardil_bul(A)` : p = A->sagCocuk = C.

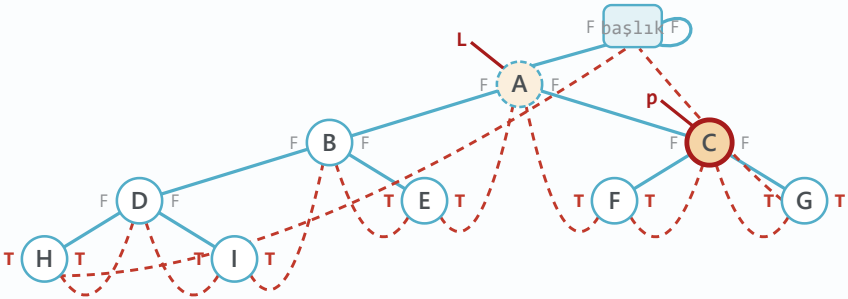

```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
▶   kilavuz_gost p = L->sagCocuk;
4   if (!L->sagKilavuz)        // sağ alt ağaç var mı
5     while (!p->solKilavuz)   // en soluna dek in
6       p = p->solCocuk;
7   return p;
8 }
```

DEĞİŞKENLER

`L = &düğüm(A)` `p = &düğüm(C)`

`L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



sagKilavuz FALSE: A'nın gerçek bir sağ alt ağacı var; ardıl oranın en solunda.

Adım 10 / 16

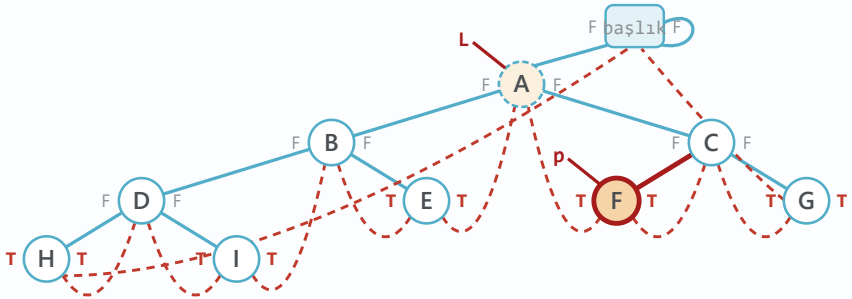
```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
3   kilavuz_gost p = L->sagCocuk;
▶   if (!L->sagKilavuz)          // sağ alt ağaç var mı
5     while (!p->solKilavuz)     // en soluna dek in
6       p = p->solCocuk;
7   return p;
8 }
```

DEĞİŞKENLER

`L = &düğüm(A)` `p = &düğüm(C)`

`L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



C->solKilavuz FALSE → sola: p = F.

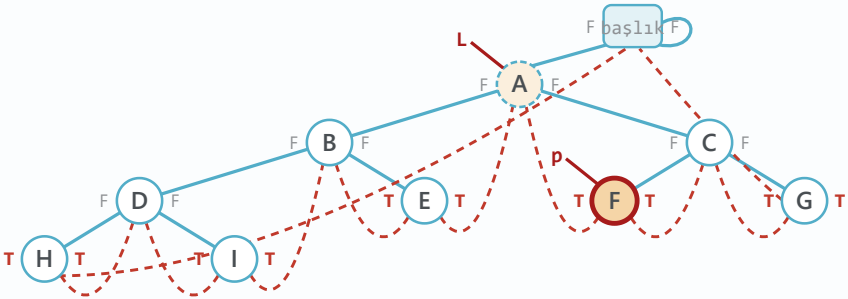

```
1  kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2    // L ile işaret edilen (NULL olmayan) öğenin ardılı
3    kilavuz_gost p = L->sagCocuk;
4    if (!L->sagKilavuz)        // sağ alt ağaç var mı
▶      while (!p->solKilavuz)   // en soluna dek in
▶        p = p->solCocuk;
7    return p;
8  }
```

DEĞİŞKENLER

`L = &düğüm(A)` `p = &düğüm(F)`

`L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



F->solKilavuz TRUE: daha sola gidilemez → en soldaki düğüm.

Adım 12 / 16

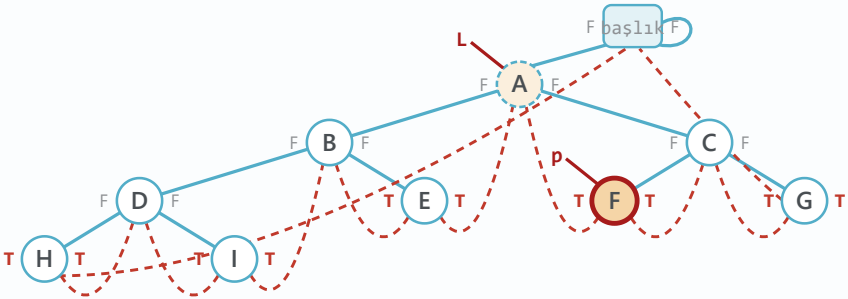
```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
3   kilavuz_gost p = L->sagCocuk;
4   if (!L->sagKilavuz)        // sağ alt ağaç var mı
▶     while (!p->solKilavuz)   // en soluna dek in
6       p = p->solCocuk;
7   return p;
8 }
```

DEĞİŞKENLER

`L = &düğüm(A)` `p = &düğüm(F)`

`L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



F döndürülür: A'nın kök ortada ardılı.

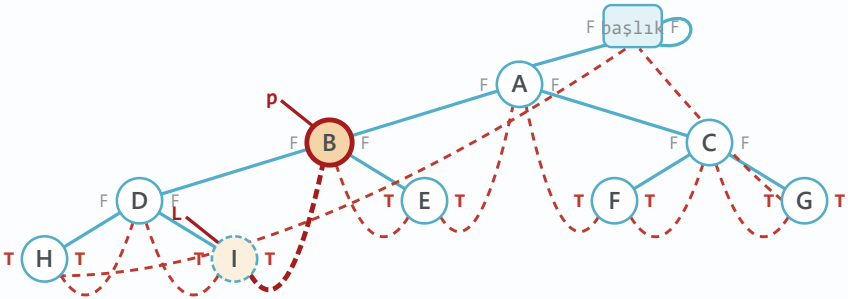
Adım 13 / 16

```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
3   kilavuz_gost p = L->sagCocuk;
4   if (!L->sagKilavuz)        // sağ alt ağaç var mı
5     while (!p->solKilavuz)   // en soluna dek in
6       p = p->solCocuk;
▶   return p;
8 }
```

DEĞİŞKENLER

`L = &düğüm(A)` `p = &düğüm(F)` `L→sagKilavuz = FALSE`

kokortada_ardil_bul(): başlık, A ve I için



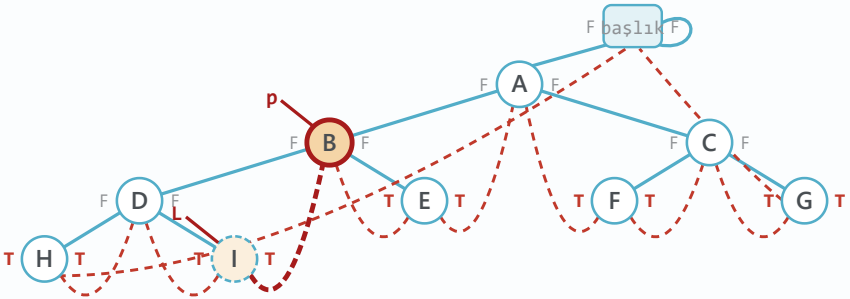
`kokortada_ardil_bul(I)` : p = I->sagCocuk = B.


```
1  kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2    // L ile işaret edilen (NULL olmayan) öğenin ardılı
▶    kilavuz_gost p = L->sagCocuk;
4    if (!L->sagKilavuz)        // sağ alt ağaç var mı
5      while (!p->solKilavuz)   // en soluna dek in
6        p = p->solCocuk;
7    return p;
8  }
```

DEĞİŞKENLER

`L = &düğüm(I)` `p = &düğüm(B)` `L→sagKilavuz = TRUE`

kokortada_ardil_bul(): başlık, A ve I için



I'nın sağ işaretçisi bir **kılavuz** (TRUE): ardıl doğrudan B. Döngüye girilmez.

Adım 15 / 16

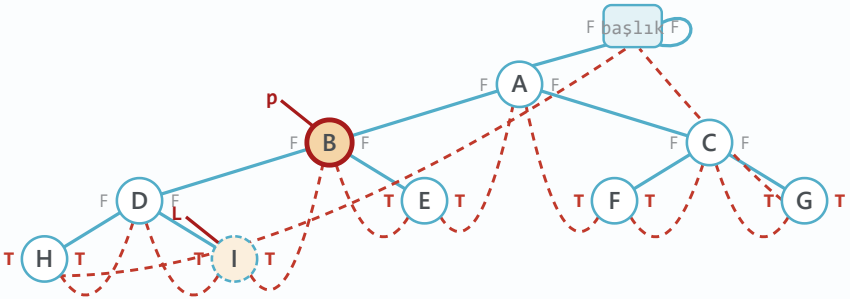
```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
3   kilavuz_gost p = L->sagCocuk;
▶   if (!L->sagKilavuz)          // sağ alt ağaç var mı
5     while (!p->solKilavuz)     // en soluna dek in
6       p = p->solCocuk;
7   return p;
8 }
```

DEĞİŞKENLER

`L = &düğüm(I)` `p = &düğüm(B)`

`L→sagKilavuz = TRUE`

kokortada_ardil_bul(): başlık, A ve I için



B döndürülür: I'nın kök ortada ardılı.

Adım 16 / 16

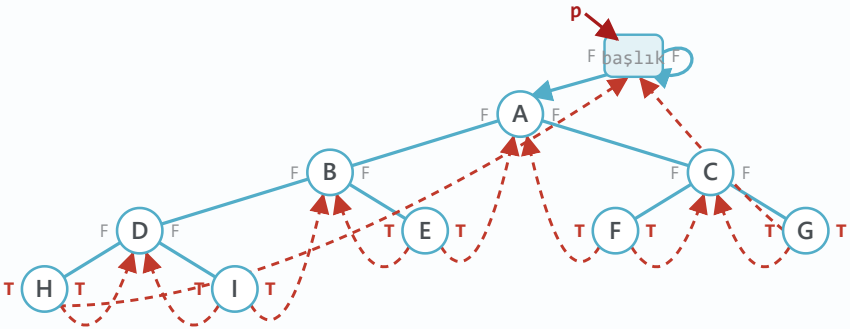
```
1 kilavuz_gost kokortada_ardil_bul(kilavuz_gost L) {
2   // L ile işaret edilen (NULL olmayan) öğenin ardılı
3   kilavuz_gost p = L->sagCocuk;
4   if (!L->sagKilavuz)        // sağ alt ağaç var mı
5     while (!p->solKilavuz)   // en soluna dek in
6       p = p->solCocuk;
▶   return p;
8 }
```

DEĞİŞKENLER

`L = &düğüm(I)` `p = &düğüm(B)` `L→sagKilavuz = TRUE`

KÖK ORTADA GEZİNME · KODU ADIM ADIM ÇALIŞTIR

kilavuz_kokortada_dolas(): – H D I B E A F C G –



p = L: başlıktan başlanır. Yığın yok, özyineleme yok; tek bir işaretçi yeter.

Adım 1 / 11

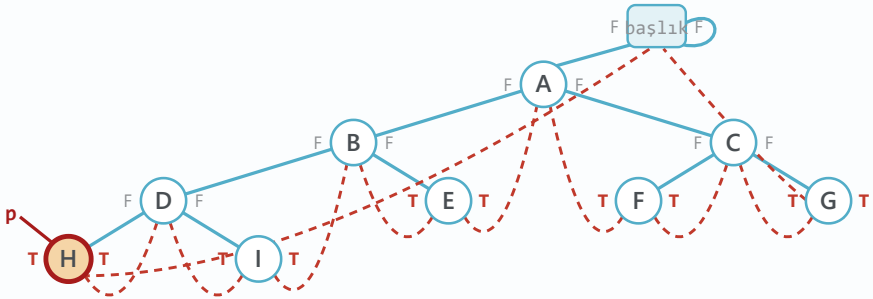
```
1 void kilavuz_kokortada_dolas(kilavuz_gost L) {
▶   kilavuz_gost p = L;        // L: başlık düğümü
3   for (;;) {
4     p = kokortada_ardil_bul(p);
5     if (p == L) break;       // başlığa dönüldü mü
6     printf("%c ", p->veri);
7   }
8 }
```

DEĞİŞKENLER

L = &başlık p = &başlık

ÇIKTI

kilavuz_kokortada_dolas(): – H D I B E A F C G –



p = ardıl(başlık) = **H** (sağ alt ağacın en solu); yazdırılır.

Adım 2 / 11

```
1 void kilavuz_kokortada_dolas(kilavuz_gost L) {
2   kilavuz_gost p = L;        // L: başlık düğümü
3   for (;;) {
▶     p = kokortada_ardil_bul(p);
5     if (p == L) break;       // başlığa dönüldü mü
▶     printf("%c ", p->veri);
7   }
8 }
```

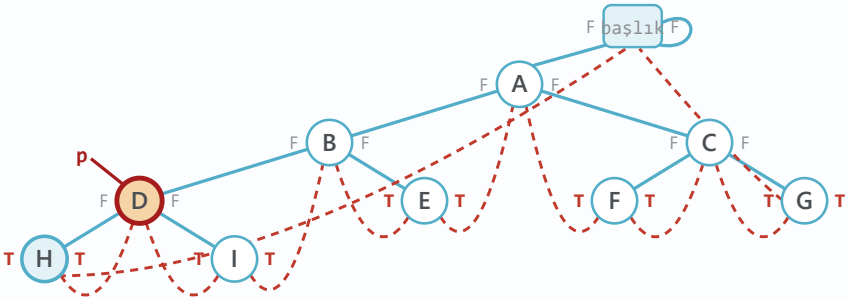
DEĞİŞKENLER

`L = &başlık` `p = &düğüm(H)`

ÇIKTI

```
H
```

kilavuz_kokortada_dolas(): – H D I B E A F C G –



p = ardıl(H) = **D** (kılavuzla doğrudan); yazdırılır.

Adım 3 / 11

```
1 void kilavuz_kokortada_dolas(kilavuz_gost L) {
2   kilavuz_gost p = L;        // L: başlık düğümü
3   for (;;) {
▶     p = kokortada_ardil_bul(p);
5     if (p == L) break;       // başlığa dönüldü mü
▶     printf("%c ", p->veri);
7   }
8 }
```

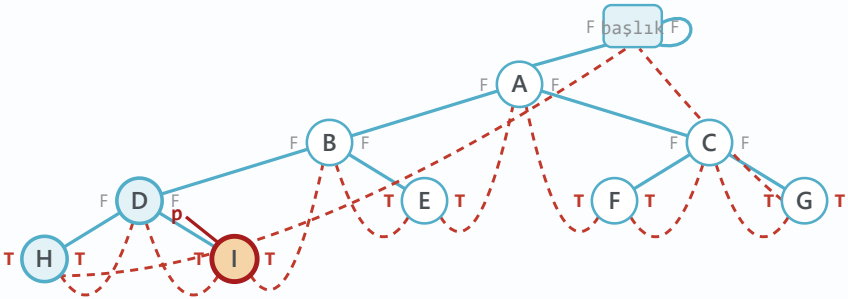
DEĞİŞKENLER

`L = &başlık` `p = &düğüm(D)`

ÇIKTI

```
H D
```

kilavuz_kokortada_dolas(): – H D I B E A F C G –



p = ardıl(D) = **I** (sağ alt ağacın en solu); yazdırılır.

Adım 4 / 11

```
1 void kilavuz_kokortada_dolas(kilavuz_gost L) {
2   kilavuz_gost p = L;        // L: başlık düğümü
3   for (;;) {
▶     p = kokortada_ardil_bul(p);
5     if (p == L) break;       // başlığa dönüldü mü
▶     printf("%c ", p->veri);
7   }
8 }
```

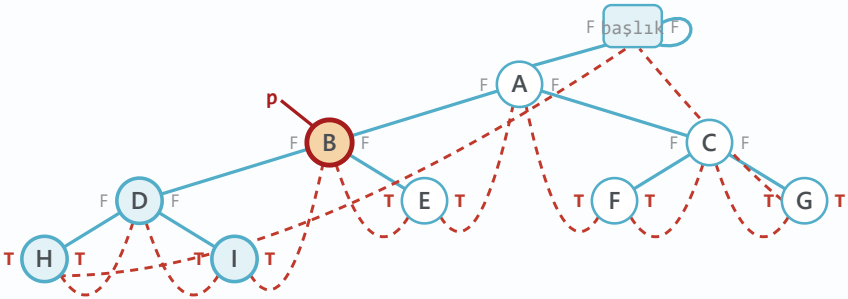
DEĞİŞKENLER

`L = &başlık` `p = &düğüm(I)`

ÇIKTI

```
H D I
```

kilavuz_kokortada_dolas(): – H D I B E A F C G –



p = ardıl(I) = **B** (kılavuzla doğrudan); yazdırılır.

Adım 5 / 11

```
1 void kilavuz_kokortada_dolas(kilavuz_gost L) {
2   kilavuz_gost p = L;         // L: başlık düğümü
3   for (;;) {
▶     p = kokortada_ardil_bul(p);
5     if (p == L) break;        // başlığa dönüldü mü
▶     printf("%c ", p->veri);
7   }
8 }
```

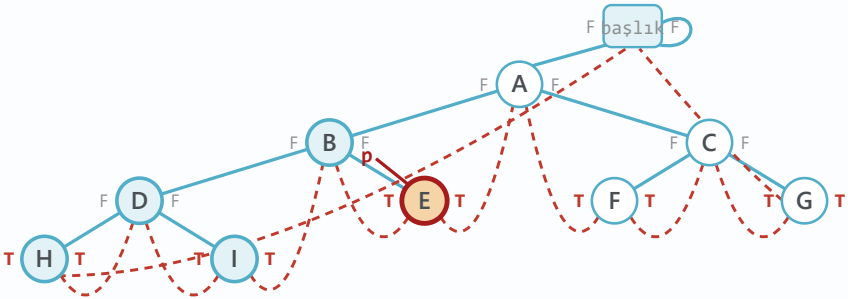
DEĞİŞKENLER

`L = &başlık` `p = &düğüm(B)`

ÇIKTI

```
H D I B
```

kilavuz_kokortada_dolas(): – H D I B E A F C G –



p = ardıl(B) = **E** (sağ alt ağacın en solu); yazdırılır.

Adım 6 / 11

```
1 void kilavuz_kokortada_dolas(kilavuz_gost L) {
2   kilavuz_gost p = L;         // L: başlık düğümü
3   for (;;) {
▶     p = kokortada_ardil_bul(p);
5     if (p == L) break;        // başlığa dönüldü mü
▶     printf("%c ", p->veri);
7   }
8 }
```

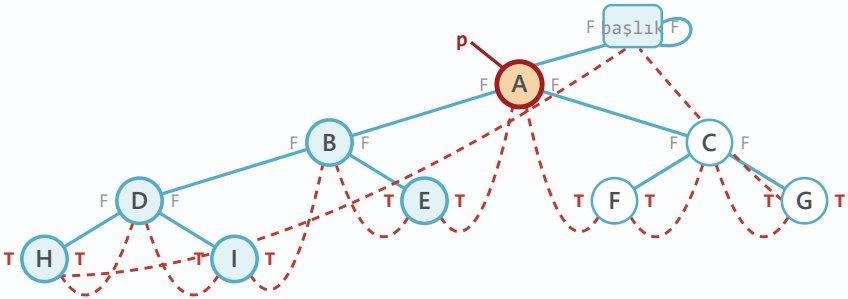
DEĞİŞKENLER

`L = &başlık` `p = &düğüm(E)`

ÇIKTI

```
H D I B E
```

kilavuz_kokortada_dolas(): – H D I B E A F C G –



p = ardıl(E) = **A** (kılavuzla doğrudan); yazdırılır.

Adım 7 / 11

```
1 void kilavuz_kokortada_dolas(kilavuz_gost L) {
2   kilavuz_gost p = L;         // L: başlık düğümü
3   for (;;) {
▶     p = kokortada_ardil_bul(p);
5     if (p == L) break;        // başlığa dönüldü mü
▶     printf("%c ", p->veri);
7   }
8 }
```

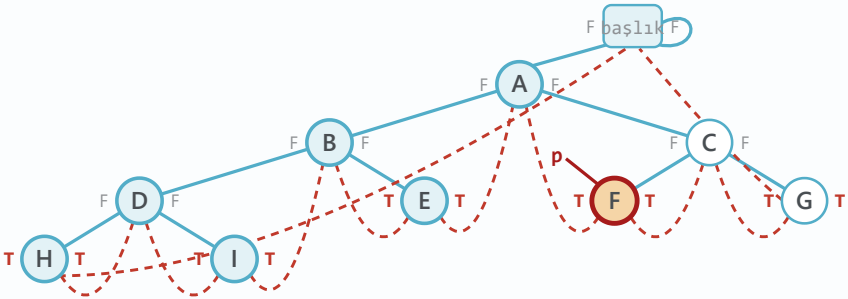
DEĞİŞKENLER

`L = &başlık` `p = &düğüm(A)`

ÇIKTI

```
H D I B E A
```

kilavuz_kokortada_dolas(): – H D I B E A F C G –



p = ardıl(A) = **F** (sağ alt ağacın en solu); yazdırılır.

Adım 8 / 11

```
1  void kilavuz_kokortada_dolas(kilavuz_gost L) {
2    kilavuz_gost p = L;        // L: başlık düğümü
3    for (;;) {
▶      p = kokortada_ardil_bul(p);
5      if (p == L) break;       // başlığa dönüldü mü
▶      printf("%c ", p->veri);
7    }
8  }
```

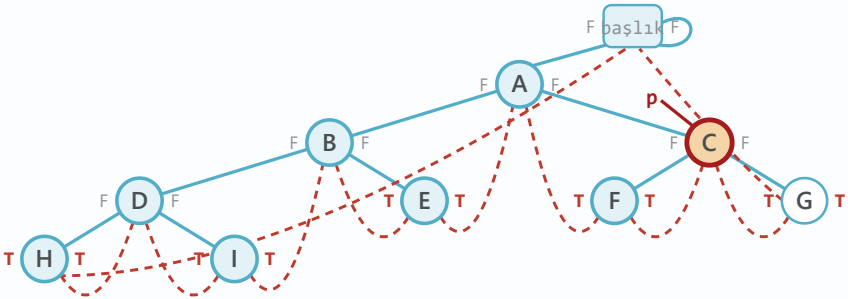
DEĞİŞKENLER

`L = &başlık` `p = &düğüm(F)`

ÇIKTI

```
H D I B E A F
```

kilavuz_kokortada_dolas(): – H D I B E A F C G –



p = ardıl(F) = **C** (kılavuzla doğrudan); yazdırılır.


```
1 void kilavuz_kokortada_dolas(kilavuz_gost L) {
2   kilavuz_gost p = L;         // L: başlık düğümü
3   for (;;) {
▶     p = kokortada_ardil_bul(p);
5     if (p == L) break;        // başlığa dönüldü mü
▶     printf("%c ", p->veri);
7   }
8 }
```

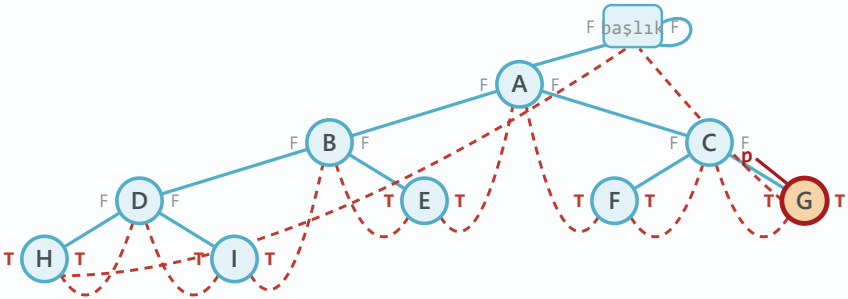
DEĞİŞKENLER

`L = &başlık` `p = &düğüm(C)`

ÇIKTI

```
H D I B E A F C
```

kilavuz_kokortada_dolas(): – H D I B E A F C G –



p = ardıl(C) = **G** (sağ alt ağacın en solu); yazdırılır.

Adım 10 / 11

```
1 void kilavuz_kokortada_dolas(kilavuz_gost L) {
2   kilavuz_gost p = L;         // L: başlık düğümü
3   for (;;) {
▶     p = kokortada_ardil_bul(p);
5     if (p == L) break;        // başlığa dönüldü mü
▶     printf("%c ", p->veri);
7   }
8 }
```

DEĞİŞKENLER

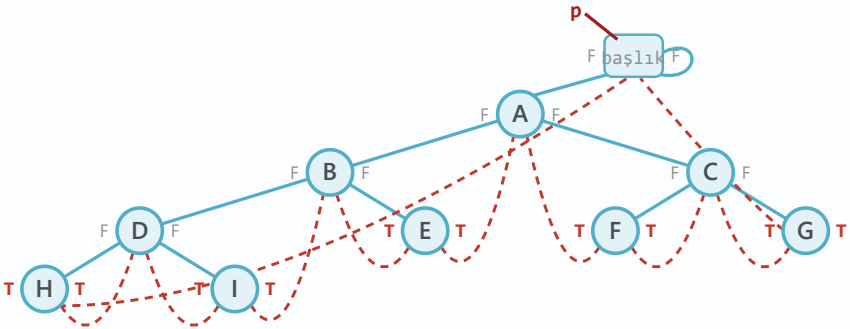
`L = &başlık` `p = &düğüm(G)`

ÇIKTI

```
H D I B E A F C G
```

KÖK ORTADA GEZİNME · KODU ADIM ADIM ÇALIŞTIR

kilavuz_kokortada_dolas(): – H D I B E A F C G –



G'nin ardılı (kılavuzla doğrudan) başlık: gezinme tamamlandı → `break`.

Adım 11 / 11

```
1 void kilavuz_kokortada_dolas(kilavuz_gost L) {
2   kilavuz_gost p = L;        // L: başlık düğümü
3   for (;;) {
▶     p = kokortada_ardil_bul(p);
▶     if (p == L) break;       // başlığa dönüldü mü
6     printf("%c ", p->veri);
7   }
8 }
```

DEĞİŞKENLER

`L = &başlık` `p = &başlık`

ÇIKTI

```
H D I B E A F C G
```

Kılavuzlu gezinmenin maliyeti

- Her kılavuz ve her gerçek sol/sağ bağ en çok bir kez izlenir → **O(n)** zaman.
- Ek bellek **O(1)**: yalnızca p işaretçisi (özyinelemeli gezinmede O(h) çağrı yığını, yinelemelide O(h) yığın).
- Herhangi bir düğümden ardılına, gezinmeyi baştan yapmadan geçilebilir.

HOCADAN "Bu iki zât diyor ki: "Kök ortada gezinmeye göre kurduğum bu ağaçta döngüsel olarak hem postorder hem de preorder gezinebilirim." Preorder kolayca yazılabilir; postorder'ı yazmak öncekinden zordur."

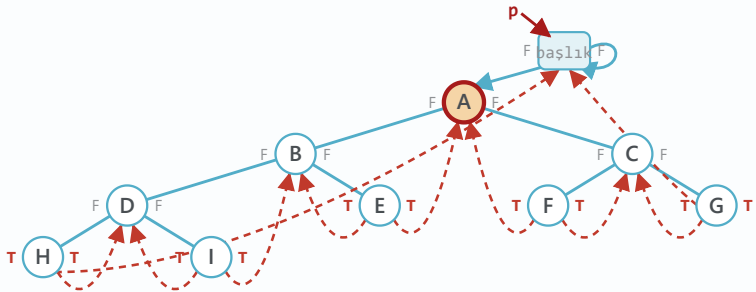
HOCADAN "Bir problemin çözümüne yönelik makaleler okunur; $O(n^2)$'de çözülebilmiş. Oturup çalışarak $O(n^{1.89})$'a düşüren bir algoritma yazmak büyük iştir! Literatüre o algoritma geçer."

Satır numaraları

Kod parçalarındaki satır numaraları, editörde ALT basılı tutularak sütun seçimiyle seçilip DEL ile silinebilir.

KÖK ORTADA GEZİNME · KODU ADIM ADIM ÇALIŞTIR

Kök önde gezinme aynı kılavuzlarla



ardıl(başlık) = **A** (sol çocuk).

Adım 1 / 10

```
1 kilavuz_gost kokonde_ardil_bul(kilavuz_gost p) {
▶   if (!p->solKilavuz)        // sol çocuk varsa o
▶     return p->solCocuk;
4   while (p->sagKilavuz)      // sağ kılavuzlarla çık
5     p = p->sagCocuk;
6   return p->sagCocuk;        // ilk gerçek sağ çocuk
7 }
```

DEĞİŞKENLER

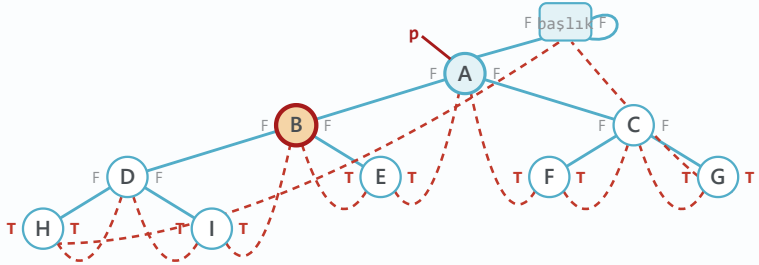
`p = &başlık` `sonuç = &düğüm(A)`

ÇIKTI

```
A
```

Sol çocuk varsa ardıl odur; yoksa sağ kılavuzlarla yukarı çıkılıp ilk gerçek sağ çocuğa geçilir. Kök sonda için ebeveyn bilgisi gerekir (daha zor).

Kök önde gezinme aynı kılavuzlarla



ardıl(A) = **B** (sol çocuk).

Adım 2 / 10

```
1 kilavuz_gost kokonde_ardil_bul(kilavuz_gost p) {
▶   if (!p->solKilavuz)        // sol çocuk varsa o
▶     return p->solCocuk;
4   while (p->sagKilavuz)      // sağ kılavuzlarla çık
5     p = p->sagCocuk;
6   return p->sagCocuk;        // ilk gerçek sağ çocuk
7 }
```

DEĞİŞKENLER

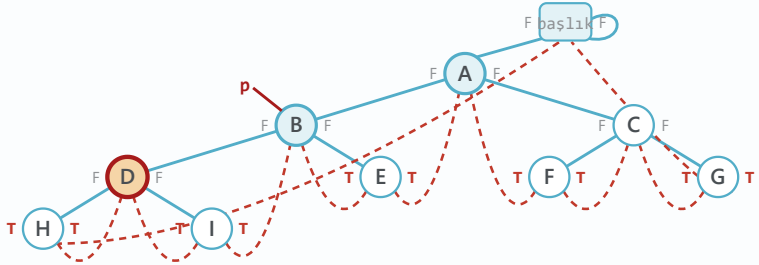
`p = &düğüm(A)` `sonuç = &düğüm(B)`

ÇIKTI

```
A B
```

Sol çocuk varsa ardıl odur; yoksa sağ kılavuzlarla yukarı çıkılıp ilk gerçek sağ çocuğa geçilir. Kök sonda için ebeveyn bilgisi gerekir (daha zor).

Kök önde gezinme aynı kılavuzlarla



ardıl(B) = **D** (sol çocuk).

Adım 3 / 10

```
1 kilavuz_gost kokonde_ardil_bul(kilavuz_gost p) {
▶   if (!p->solKilavuz)        // sol çocuk varsa o
▶     return p->solCocuk;
4   while (p->sagKilavuz)      // sağ kılavuzlarla çık
5     p = p->sagCocuk;
6   return p->sagCocuk;        // ilk gerçek sağ çocuk
7 }
```

DEĞİŞKENLER

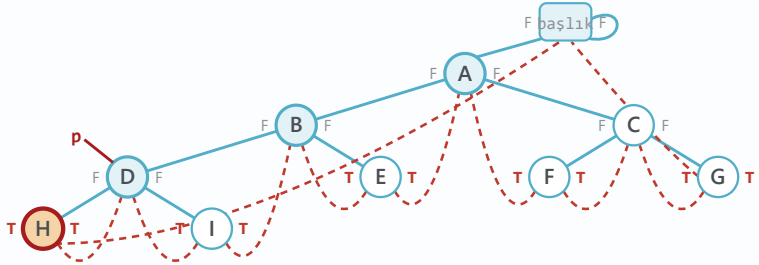
`p = &düğüm(B)` `sonuç = &düğüm(D)`

ÇIKTI

```
A B D
```

Sol çocuk varsa ardıl odur; yoksa sağ kılavuzlarla yukarı çıkılıp ilk gerçek sağ çocuğa geçilir. Kök sonda için ebeveyn bilgisi gerekir (daha zor).

Kök önde gezinme aynı kılavuzlarla



ardıl(D) = **H** (sol çocuk).

Adım 4 / 10

```
1 kilavuz_gost kokonde_ardil_bul(kilavuz_gost p) {
▶   if (!p->solKilavuz)        // sol çocuk varsa o
▶     return p->solCocuk;
4   while (p->sagKilavuz)      // sağ kılavuzlarla çık
5     p = p->sagCocuk;
6   return p->sagCocuk;        // ilk gerçek sağ çocuk
7 }
```

DEĞİŞKENLER

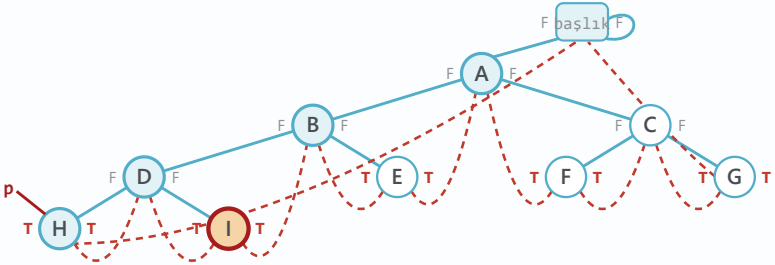
`p = &düğüm(D)` `sonuç = &düğüm(H)`

ÇIKTI

```
A B D H
```

Sol çocuk varsa ardıl odur; yoksa sağ kılavuzlarla yukarı çıkılıp ilk gerçek sağ çocuğa geçilir. Kök sonda için ebeveyn bilgisi gerekir (daha zor).

Kök önde gezinme aynı kılavuzlarla



ardıl(H) = **I** (sağ kılavuzlarla D üzerinden sağ çocuk).

Adım 5 / 10

```
1 kilavuz_gost kokonde_ardil_bul(kilavuz_gost p) {
2   if (!p->solKilavuz)        // sol çocuk varsa o
3     return p->solCocuk;
▶   while (p->sagKilavuz)      // sağ kılavuzlarla çık
▶     p = p->sagCocuk;
▶   return p->sagCocuk;        // ilk gerçek sağ çocuk
7 }
```

DEĞİŞKENLER

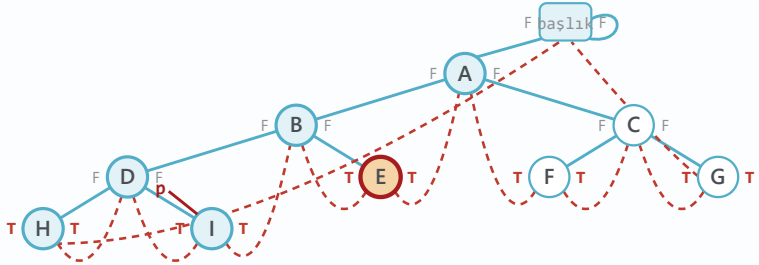
`p = &düğüm(H)` `sonuç = &düğüm(I)`

ÇIKTI

```
A B D H I
```

Sol çocuk varsa ardıl odur; yoksa sağ kılavuzlarla yukarı çıkılıp ilk gerçek sağ çocuğa geçilir. Kök sonda için ebeveyn bilgisi gerekir (daha zor).

Kök önde gezinme aynı kılavuzlarla



ardıl(I) = **E** (sağ kılavuzlarla B üzerinden sağ çocuk).

Adım 6 / 10

```
1 kilavuz_gost kokonde_ardil_bul(kilavuz_gost p) {
2   if (!p->solKilavuz)        // sol çocuk varsa o
3     return p->solCocuk;
▶   while (p->sagKilavuz)      // sağ kılavuzlarla çık
▶     p = p->sagCocuk;
▶   return p->sagCocuk;        // ilk gerçek sağ çocuk
7 }
```

DEĞİŞKENLER

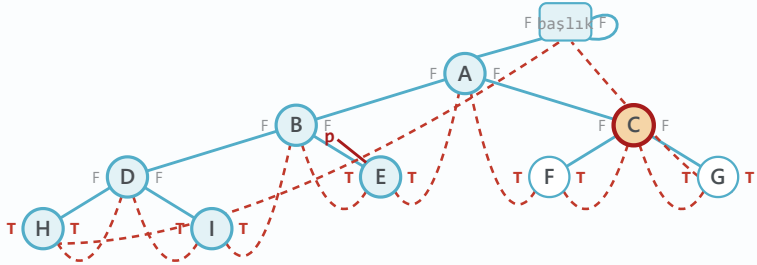
`p = &düğüm(I)` `sonuç = &düğüm(E)`

ÇIKTI

```
A B D H I E
```

Sol çocuk varsa ardıl odur; yoksa sağ kılavuzlarla yukarı çıkılıp ilk gerçek sağ çocuğa geçilir. Kök sonda için ebeveyn bilgisi gerekir (daha zor).

Kök önde gezinme aynı kılavuzlarla



ardıl(E) = **C** (sağ kılavuzlarla A üzerinden sağ çocuk).

Adım 7 / 10

```
1 kilavuz_gost kokonde_ardil_bul(kilavuz_gost p) {
2   if (!p->solKilavuz)        // sol çocuk varsa o
3     return p->solCocuk;
▶   while (p->sagKilavuz)      // sağ kılavuzlarla çık
▶     p = p->sagCocuk;
▶   return p->sagCocuk;        // ilk gerçek sağ çocuk
7 }
```

DEĞİŞKENLER

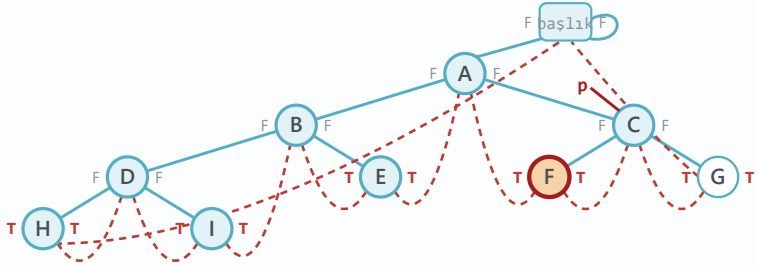
`p = &düğüm(E)` `sonuç = &düğüm(C)`

ÇIKTI

```
A B D H I E C
```

Sol çocuk varsa ardıl odur; yoksa sağ kılavuzlarla yukarı çıkılıp ilk gerçek sağ çocuğa geçilir. Kök sonda için ebeveyn bilgisi gerekir (daha zor).

Kök önde gezinme aynı kılavuzlarla



ardıl(C) = **F** (sol çocuk).

Adım 8 / 10

```
1 kilavuz_gost kokonde_ardil_bul(kilavuz_gost p) {
▶   if (!p->solKilavuz)        // sol çocuk varsa o
▶     return p->solCocuk;
4   while (p->sagKilavuz)      // sağ kılavuzlarla çık
5     p = p->sagCocuk;
6   return p->sagCocuk;        // ilk gerçek sağ çocuk
7 }
```

DEĞİŞKENLER

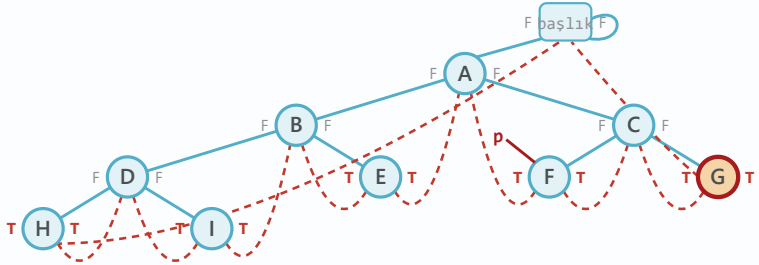
`p = &düğüm(C)` `sonuç = &düğüm(F)`

ÇIKTI

```
A B D H I E C F
```

Sol çocuk varsa ardıl odur; yoksa sağ kılavuzlarla yukarı çıkılıp ilk gerçek sağ çocuğa geçilir. Kök sonda için ebeveyn bilgisi gerekir (daha zor).

Kök önde gezinme aynı kılavuzlarla



ardıl(F) = **G** (sağ kılavuzlarla C üzerinden sağ çocuk).

Adım 9 / 10

```
1 kilavuz_gost kokonde_ardil_bul(kilavuz_gost p) {
2   if (!p->solKilavuz)        // sol çocuk varsa o
3     return p->solCocuk;
▶   while (p->sagKilavuz)      // sağ kılavuzlarla çık
▶     p = p->sagCocuk;
▶   return p->sagCocuk;        // ilk gerçek sağ çocuk
7 }
```

DEĞİŞKENLER

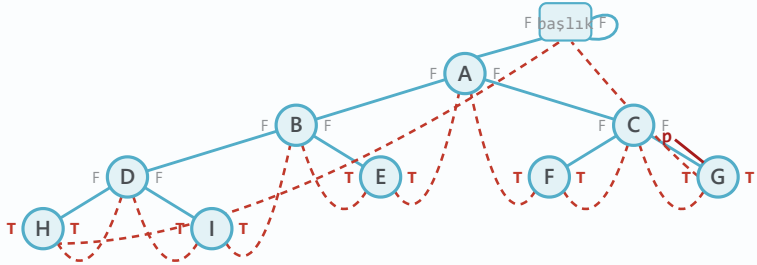
`p = &düğüm(F)` `sonuç = &düğüm(G)`

ÇIKTI

```
A B D H I E C F G
```

Sol çocuk varsa ardıl odur; yoksa sağ kılavuzlarla yukarı çıkılıp ilk gerçek sağ çocuğa geçilir. Kök sonda için ebeveyn bilgisi gerekir (daha zor).

Kök önde gezinme aynı kılavuzlarla



G: sağ kılavuzlarla başlık üzerinden sağ çocuk → başlık: gezinme biter.

Adım 10 / 10

```
1 kilavuz_gost kokonde_ardil_bul(kilavuz_gost p) {
2   if (!p->solKilavuz)        // sol çocuk varsa o
3     return p->solCocuk;
▶   while (p->sagKilavuz)      // sağ kılavuzlarla çık
▶     p = p->sagCocuk;
▶   return p->sagCocuk;        // ilk gerçek sağ çocuk
7 }
```

DEĞİŞKENLER

`p = &düğüm(G)` `sonuç = &başlık`

ÇIKTI

```
A B D H I E C F G
```

Sol çocuk varsa ardıl odur; yoksa sağ kılavuzlarla yukarı çıkılıp ilk gerçek sağ çocuğa geçilir. Kök sonda için ebeveyn bilgisi gerekir (daha zor).

KÖK ORTADA GEZİNME

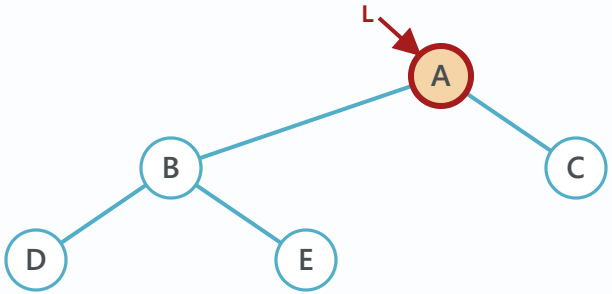
Morris gezinmesi: kılavuzları geçici kurmak

```
void morris_kokortada(agac_gost L) {
  while (L) {
    if (!L->solcocuk) {             /* sol yok: yaz, sağa */
      printf("%c", L->veri);  L = L->sagcocuk;
    } else {
      agac_gost p = L->solcocuk;    /* kök ortada öncülü */
      while (p->sagcocuk && p->sagcocuk != L)
        p = p->sagcocuk;
      if (!p->sagcocuk) {           /* geçici kılavuz kur */
        p->sagcocuk = L;  L = L->solcocuk;
      } else {                      /* kılavuzu sök */
        p->sagcocuk = NULL;
        printf("%c", L->veri);  L = L->sagcocuk;
      }
    }
  }
}
```

J. M. Morris (1979). Bayrak alanı gerekmez: kılavuz gezinme sırasında kurulup sökülür; ağaç o sırada değiştiği için eşzamanlı okumaya uygun değildir.

KÖK ORTADA GEZİNME

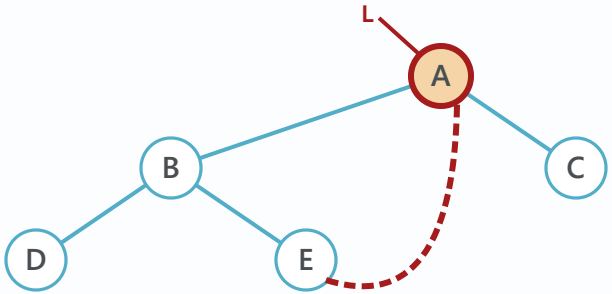
Morris gezinmesi adım adım: A(B(D,E),C)



çıktı:

Başlangıç: yığın yok. Solu olan her düğümde, öncülünün (boş) sağ işaretçisine **geçici kılavuz** kurulacak.

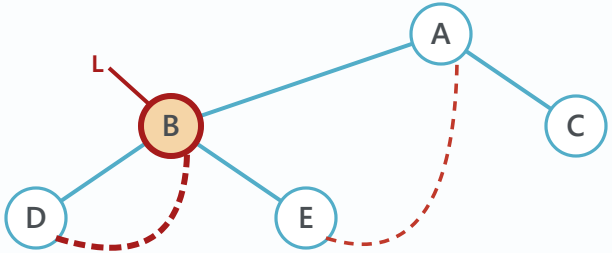
Morris gezinmesi adım adım: A(B(D,E),C)



çıktı:

A'nın öncülü E; sağı boş → kılavuz E → A kurulur, sola inilir.

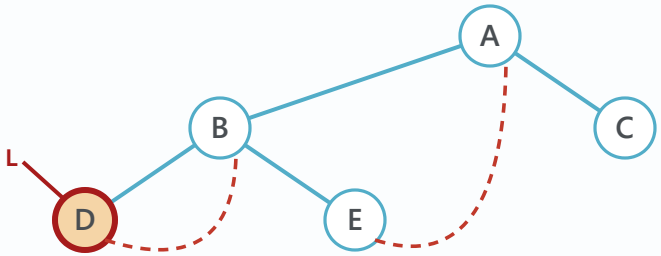
Morris gezinmesi adım adım: A(B(D,E),C)



çıktı:

B'nin öncülü D; sağı boş → kılavuz D → B kurulur, sola inilir.

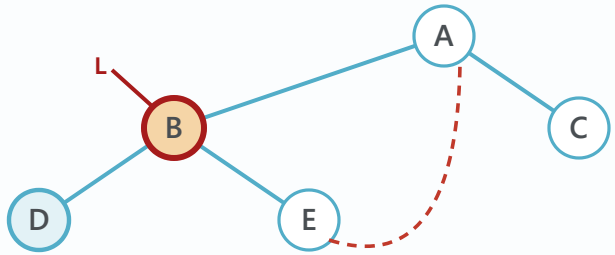
Morris gezinmesi adım adım: A(B(D,E),C)



çıktı: D

D'nin solu yok: yazılır, sağa (ya da geçici kılavuzla) geçilir → B.

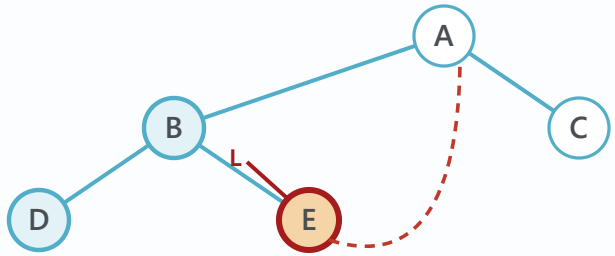
Morris gezinmesi adım adım: A(B(D,E),C)



çıktı: DB

Öncül D'den kılavuzla B'ye dönüldü: kılavuz sökülür, B yazılır, sağa.

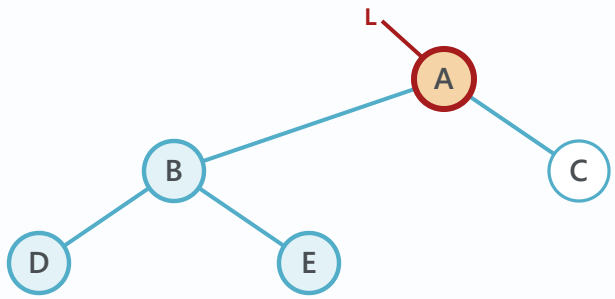
Morris gezinmesi adım adım: A(B(D,E),C)



çıktı: DBE

E'nin solu yok: yazılır, sağa (ya da geçici kılavuzla) geçilir → A.

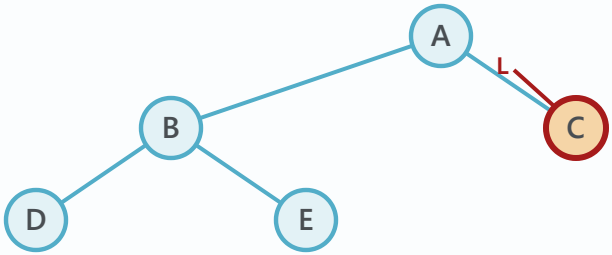
Morris gezinmesi adım adım: A(B(D,E),C)



çıktı: DBEA

Öncül E'den kılavuzla A'ya dönüldü: kılavuz sökülür, A yazılır, sağa.

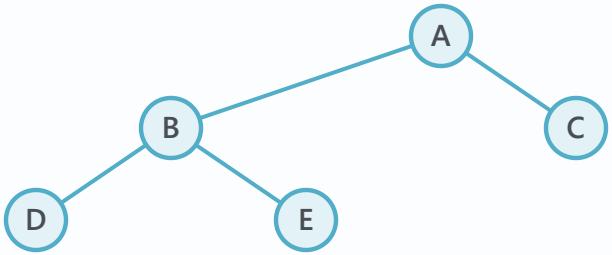
Morris gezinmesi adım adım: A(B(D,E),C)



çıktı: DBEAC

C'nin solu yok: yazılır, sağa (ya da geçici kılavuzla) geçilir → NULL.

Morris gezinmesi adım adım: A(B(D,E),C)



çıktı: DBEAC

Bitti: **DBEAC**. Ağaç eski hâline döndü; ek bellek O(1), zaman O(n).

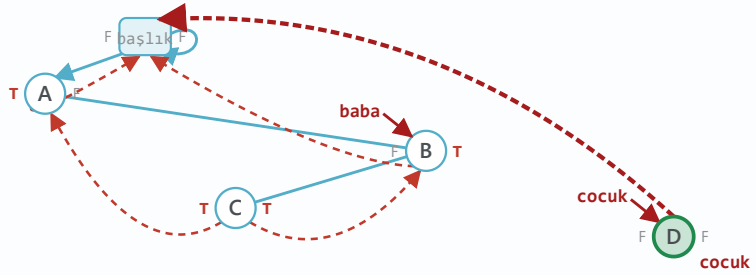
Kılavuzlu ağaca düğüm ekleme

- Ekleme tek başına kolay bir yordam değildir, **biraz girifttir**.
- Sağa ve sola ekleme biçimi farklı → iki ayrı yordam: `saga_ekle`, `sola_ekle`.
- Sağa eklemede iki durum: (1) ebeveynin sağ alt ağacı **yok**, (2) **var**.
- BST'de ekleme hep çocuğu olmayan yere tek atamaydı; burada kılavuzların (kırmızı okların) da doğru yeri göstermesi gerekir.

```
cocuk->sagCocuk  = baba->sagCocuk;
cocuk->sagKilavuz = TRUE;  // ebeveynin sağı yoktu
cocuk->solCocuk  = baba;
cocuk->solKilavuz = TRUE;
baba->sagCocuk   = cocuk;
baba->sagKilavuz = FALSE;  // TRUE idi
```

Durum 1: 6 aktarma komutuyla ekleme.

Durum 1: sağ alt ağacı olmayan ebeveyne ekleme



cocuk->sagCocuk = baba->sagCocuk: ebeveynin sağ kılavuzu (ardılı) çocuğa geçer. (Başlık olup olmadığı bizi ilgilendirmiyor.)

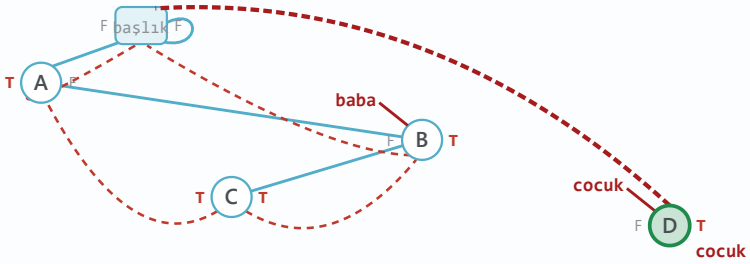
Adım 1 / 6

```
▶ cocuk->sagCocuk  = baba->sagCocuk;
2 cocuk->sagKilavuz = TRUE;  // ebeveynin sağı yoktu
3 cocuk->solCocuk  = baba;
4 cocuk->solKilavuz = TRUE;
5 baba->sagCocuk   = cocuk;
6 baba->sagKilavuz = FALSE;  // TRUE idi
```

DEĞİŞKENLER

- `baba = &düğüm(B)`
- `cocuk = &düğüm(D)`
- `baba→sagKilavuz = TRUE`
- `cocuk→sagKilavuz = FALSE`

Durum 1: sağ alt ağacı olmayan ebeveyne ekleme



cocuk->sagKilavuz = TRUE: ebeveynin alt ağacı yoktu, çocuğun da olamaz → kırmızı ok.

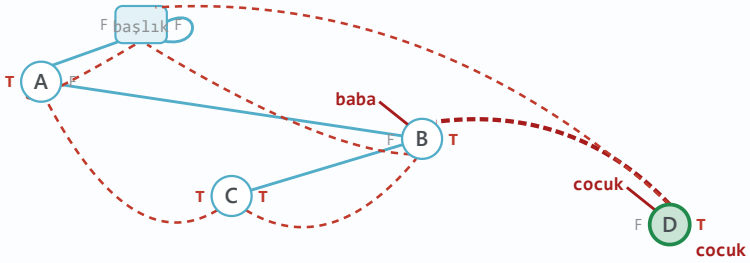
Adım 2 / 6

```
1 cocuk->sagCocuk   = baba->sagCocuk;
▶ cocuk->sagKilavuz = TRUE;  // ebeveynin sağı yoktu
3 cocuk->solCocuk   = baba;
4 cocuk->solKilavuz = TRUE;
5 baba->sagCocuk    = cocuk;
6 baba->sagKilavuz = FALSE;  // TRUE idi
```

DEĞİŞKENLER

- `baba = &düğüm(B)`
- `cocuk = &düğüm(D)`
- `baba→sagKilavuz = TRUE`
- `cocuk→sagKilavuz = TRUE`

Durum 1: sağ alt ağacı olmayan ebeveyne ekleme



cocuk->solCocuk = baba: çocuğun kök ortada öncülü ebeveyn.

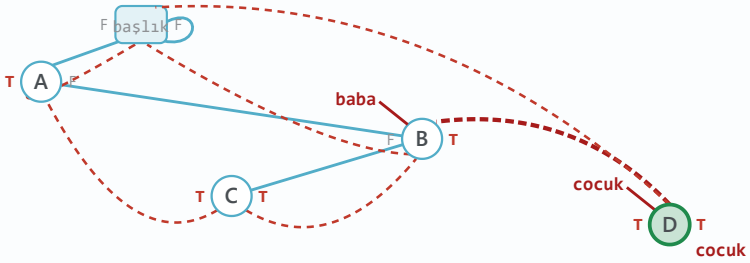
Adım 3 / 6

```
1 cocuk->sagCocuk   = baba->sagCocuk;
2 cocuk->sagKilavuz = TRUE;  // ebeveynin sağı yoktu
▶ cocuk->solCocuk   = baba;
4 cocuk->solKilavuz = TRUE;
5 baba->sagCocuk    = cocuk;
6 baba->sagKilavuz = FALSE;  // TRUE idi
```

DEĞİŞKENLER

- `baba = &düğüm(B)`
- `cocuk = &düğüm(D)`
- `baba→sagKilavuz = TRUE`
- `cocuk→sagKilavuz = TRUE`

Durum 1: sağ alt ağacı olmayan ebeveyne ekleme



cocuk->solKilavuz = TRUE: bu da bir kılavuz (kırmızı).

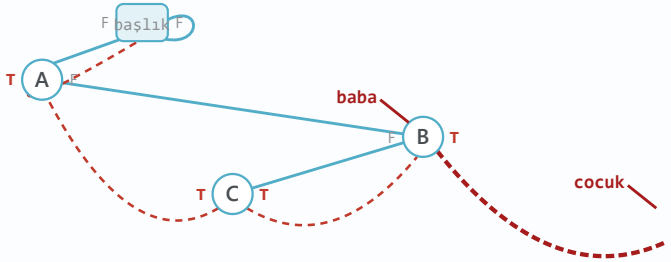
Adım 4 / 6

```
1 cocuk->sagCocuk   = baba->sagCocuk;
2 cocuk->sagKilavuz = TRUE;  // ebeveynin sağı yoktu
3 cocuk->solCocuk   = baba;
▶ cocuk->solKilavuz = TRUE;
5 baba->sagCocuk    = cocuk;
6 baba->sagKilavuz = FALSE;  // TRUE idi
```

DEĞİŞKENLER

- `baba = &düğüm(B)`
- `cocuk = &düğüm(D)`
- `baba→sagKilavuz = TRUE`
- `cocuk→sagKilavuz = TRUE`

Durum 1: sağ alt ağacı olmayan ebeveyne ekleme



baba->sagCocuk = cocuk: ebeveynden çocuğa gerçek (siyah) bağ.

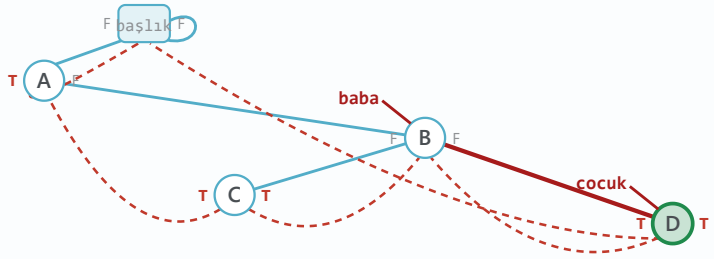
Adım 5 / 6

```
1 cocuk->sagCocuk  = baba->sagCocuk;
2 cocuk->sagKilavuz = TRUE;  // ebeveynin sağı yoktu
3 cocuk->solCocuk  = baba;
4 cocuk->solKilavuz = TRUE;
▶ baba->sagCocuk   = cocuk;
6 baba->sagKilavuz = FALSE;  // TRUE idi
```

DEĞİŞKENLER

- `baba = &düğüm(B)`
- `cocuk = &düğüm(D)`
- `baba→sagKilavuz = TRUE`
- `cocuk→sagKilavuz = TRUE`

Durum 1: sağ alt ağacı olmayan ebeveyne ekleme



baba->sagKilavuz = FALSE: ebeveynin sağı artık kılavuz değil, çocuk. Ekleme bitti: 6 aktarma.

Adım 6 / 6

```
1 cocuk->sagCocuk  = baba->sagCocuk;
2 cocuk->sagKilavuz = TRUE;  // ebeveynin sağı yoktu
3 cocuk->solCocuk  = baba;
4 cocuk->solKilavuz = TRUE;
5 baba->sagCocuk   = cocuk;
▶ baba->sagKilavuz = FALSE;  // TRUE idi
```

DEĞİŞKENLER

- `baba = &düğüm(B)`
- `cocuk = &düğüm(D)`
- `baba→sagKilavuz = FALSE`
- `cocuk→sagKilavuz = TRUE`

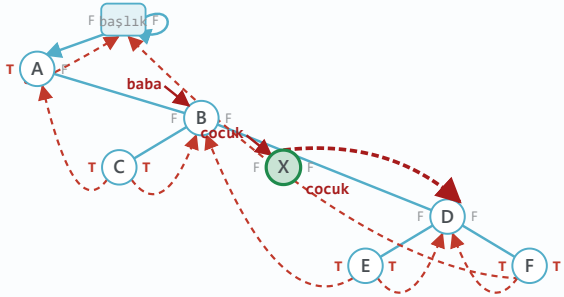
Durum 2: ebeveynin sağ alt ağacı var

- İstenen: D, B'nin çocuğu iken X'in B'nin çocuğu yapılması; ama D alt ağacı **kaybolmamalı**, kırmızı işaretçiler yine doğru yerleri göstermeli.
- X'in öncülü B olur. D alt ağacının en solundaki E'nin eski öncülü B idi; şimdi E, X'in **ardılı** oldu.
- Öyleyse: **çocuk, ebeveynin eski ardılının öncülü yapılmalıdır.**
- Durum 2 kodunda: 2. satır `cocuk->sagKilavuz = FALSE` (ebeveynin alt ağacı şimdi çocuğun alt ağacı); E'nin sol kılavuzu ayrıca düzeltilmelidir.

SONUÇ: 2. durum için ebeveynin eski ardılının öncül bilgisini günlemek ve 2 numaralı aktarmayı `cocuk→sagKilavuz = baba→sagKilavuz` yazmak suretiyle her iki durumun da üstesinden gelen **tek bir yordam** yazılabilir.

saga_ekle(B, X): tek yordam, 2. durum

```
1  void saga_ekle(kilavuz_gost baba,
2                 kilavuz_gost cocuk) {
3    // cocuk'u baba'nın sağ çocuğu yapar (kılavuzlu)
4    kilavuz_gost p;
▶   cocuk->sagCocuk    = baba->sagCocuk;
6    cocuk->sagKilavuz = baba->sagKilavuz;
7    cocuk->solCocuk   = baba;
8    cocuk->solKilavuz = TRUE;
9    baba->sagCocuk    = cocuk;
10   baba->sagKilavuz  = FALSE;
11   if (!cocuk->sagKilavuz) { // ebeveynin alt ağacı var
     mı
12     p = kokortada_ardil_bul(cocuk);
13     p->solCocuk = cocuk;     // ardılın öncülü çocuk
14   }
15 }
```



cocuk->sagCocuk = baba->sagCocuk: X'in sağına B'nin eski sağ alt ağacı (D) gelir.

Adım 1 / 9

DEĞİŞKENLER

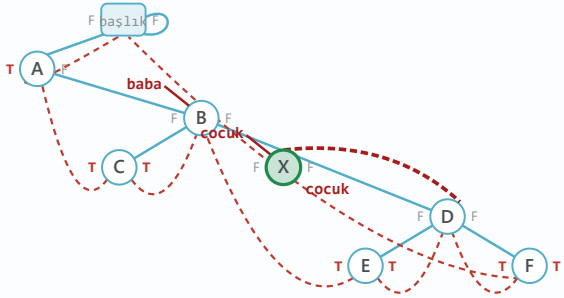
`baba = &düğüm(B)` `cocuk = &düğüm(X)`

`baba→sagKilavuz = FALSE`

`cocuk→sagKilavuz = FALSE`

saga_ekle(B, X): tek yordam, 2. durum

```
1  void saga_ekle(kilavuz_gost baba,
2                 kilavuz_gost cocuk) {
3    // cocuk'u baba'nın sağ çocuğu yapar (kılavuzlu)
4    kilavuz_gost p;
5    cocuk->sagCocuk   = baba->sagCocuk;
▶    cocuk->sagKilavuz = baba->sagKilavuz;
7    cocuk->solCocuk   = baba;
8    cocuk->solKilavuz = TRUE;
9    baba->sagCocuk    = cocuk;
10   baba->sagKilavuz  = FALSE;
11   if (!cocuk->sagKilavuz) { // ebeveynin alt ağacı var
     mı
12     p = kokortada_ardil_bul(cocuk);
13     p->solCocuk = cocuk;     // ardılın öncülü çocuk
14   }
15 }
```



cocuk->sagKilavuz = baba->sagKilavuz (FALSE): ebeveynin alt ağacı artık çocuğun alt ağacı; siyahlık korunur.

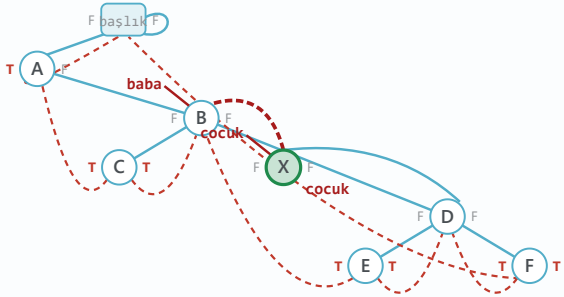
Adım 2 / 9

DEĞİŞKENLER

- `baba = &düğüm(B)`
- `cocuk = &düğüm(X)`
- `baba→sagKilavuz = FALSE`
- `cocuk→sagKilavuz = FALSE`

saga_ekle(B, X): tek yordam, 2. durum

```
1  void saga_ekle(kilavuz_gost baba,
2                 kilavuz_gost cocuk) {
3    // cocuk'u baba'nın sağ çocuğu yapar (kılavuzlu)
4    kilavuz_gost p;
5    cocuk->sagCocuk   = baba->sagCocuk;
6    cocuk->sagKilavuz = baba->sagKilavuz;
▶    cocuk->solCocuk   = baba;
8    cocuk->solKilavuz = TRUE;
9    baba->sagCocuk    = cocuk;
10   baba->sagKilavuz  = FALSE;
11   if (!cocuk->sagKilavuz) { // ebeveynin alt ağacı var
     mı
12     p = kokortada_ardil_bul(cocuk);
13     p->solCocuk = cocuk;     // ardılın öncülü çocuk
14   }
15 }
```



cocuk->solCocuk = baba: X'in öncülü B.

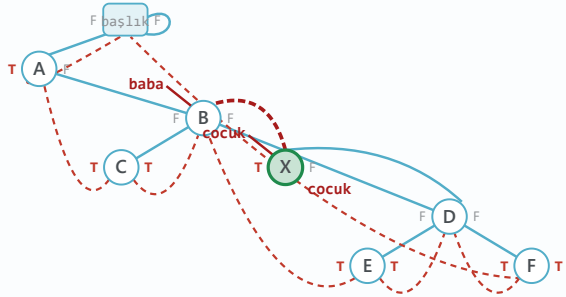
Adım 3 / 9

DEĞİŞKENLER

- baba = **&düğüm(B)**
- cocuk = **&düğüm(X)**
- baba→sagKilavuz = **FALSE**
- cocuk→sagKilavuz = **FALSE**

saga_ekle(B, X): tek yordam, 2. durum

```
1  void saga_ekle(kilavuz_gost baba,
2                 kilavuz_gost cocuk) {
3    // cocuk'u baba'nın sağ çocuğu yapar (kılavuzlu)
4    kilavuz_gost p;
5    cocuk->sagCocuk   = baba->sagCocuk;
6    cocuk->sagKilavuz = baba->sagKilavuz;
7    cocuk->solCocuk   = baba;
▶    cocuk->solKilavuz = TRUE;
9    baba->sagCocuk    = cocuk;
10   baba->sagKilavuz  = FALSE;
11   if (!cocuk->sagKilavuz) { // ebeveynin alt ağacı var
     mı
12     p = kokortada_ardil_bul(cocuk);
13     p->solCocuk = cocuk;     // ardılın öncülü çocuk
14   }
15 }
```



cocuk->solKilavuz = TRUE: kırmızı ok.

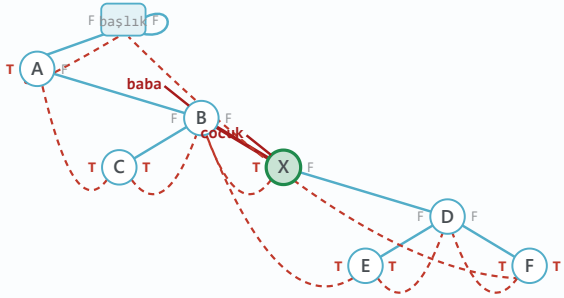
Adım 4 / 9

DEĞİŞKENLER

- baba = &düğüm(B)
- cocuk = &düğüm(X)
- baba→sagKilavuz = FALSE
- cocuk→sagKilavuz = FALSE

saga_ekle(B, X): tek yordam, 2. durum

```
1  void saga_ekle(kilavuz_gost baba,
2                 kilavuz_gost cocuk) {
3    // cocuk'u baba'nın sağ çocuğu yapar (kılavuzlu)
4    kilavuz_gost p;
5    cocuk->sagCocuk   = baba->sagCocuk;
6    cocuk->sagKilavuz = baba->sagKilavuz;
7    cocuk->solCocuk   = baba;
8    cocuk->solKilavuz = TRUE;
▶    baba->sagCocuk    = cocuk;
10   baba->sagKilavuz  = FALSE;
11   if (!cocuk->sagKilavuz) { // ebeveynin alt ağacı var
     mı
12     p = kokortada_ardil_bul(cocuk);
13     p->solCocuk = cocuk;     // ardılın öncülü çocuk
14   }
15 }
```



baba->sagCocuk = cocuk: B → X gerçek bağ.

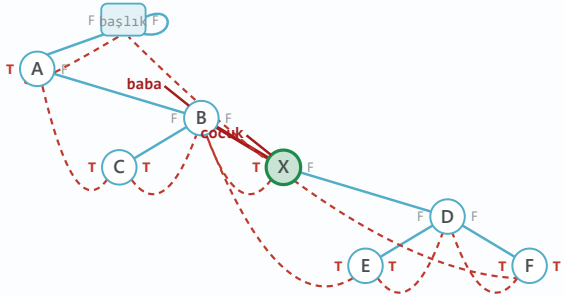
Adım 5 / 9

DEĞİŞKENLER

- baba = &düğüm(B)
- cocuk = &düğüm(X)
- baba→sagKilavuz = FALSE
- cocuk→sagKilavuz = FALSE

saga_ekle(B, X): tek yordam, 2. durum

```
1  void saga_ekle(kilavuz_gost baba,
2                 kilavuz_gost cocuk) {
3    // cocuk'u baba'nın sağ çocuğu yapar (kılavuzlu)
4    kilavuz_gost p;
5    cocuk->sagCocuk   = baba->sagCocuk;
6    cocuk->sagKilavuz = baba->sagKilavuz;
7    cocuk->solCocuk   = baba;
8    cocuk->solKilavuz = TRUE;
9    baba->sagCocuk    = cocuk;
▶    baba->sagKilavuz  = FALSE;
11   if (!cocuk->sagKilavuz) { // ebeveynin alt ağacı var
     mı
12     p = kokortada_ardil_bul(cocuk);
13     p->solCocuk = cocuk;     // ardılın öncülü çocuk
14   }
15 }
```



baba->sagKilavuz = FALSE: zaten FALSE idi; 1. durum için gerekli, burada zararsız. “Kodun doğru çalıştığına hemen karar verilmemeli!”

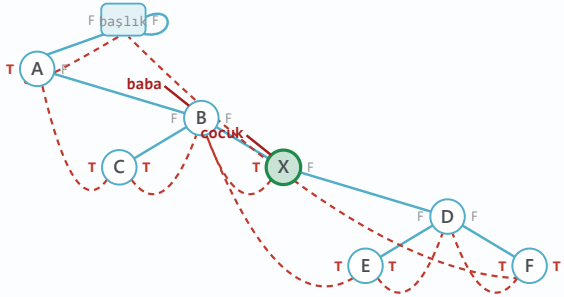
Adım 6 / 9

DEĞİŞKENLER

- `baba = &düğüm(B)`
- `cocuk = &düğüm(X)`
- `baba→sagKilavuz = FALSE`
- `cocuk→sagKilavuz = FALSE`

saga_ekle(B, X): tek yordam, 2. durum

```
1  void saga_ekle(kilavuz_gost baba,
2                 kilavuz_gost cocuk) {
3    // cocuk'u baba'nın sağ çocuğu yapar (kılavuzlu)
4    kilavuz_gost p;
5    cocuk->sagCocuk    = baba->sagCocuk;
6    cocuk->sagKilavuz  = baba->sagKilavuz;
7    cocuk->solCocuk    = baba;
8    cocuk->solKilavuz  = TRUE;
9    baba->sagCocuk     = cocuk;
10   baba->sagKilavuz   = FALSE;
▶    if (!cocuk->sagKilavuz) { // ebeveynin alt ağacı var
     mı
12     p = kokortada_ardil_bul(cocuk);
13     p->solCocuk = cocuk;     // ardılın öncülü çocuk
14   }
15 }
```



cocuk->sagKilavuz FALSE → X'in sağ alt ağacı var: X'in ardılının sol kılavuzu hâlâ **B**'yi gösteriyor, düzeltilmeli.

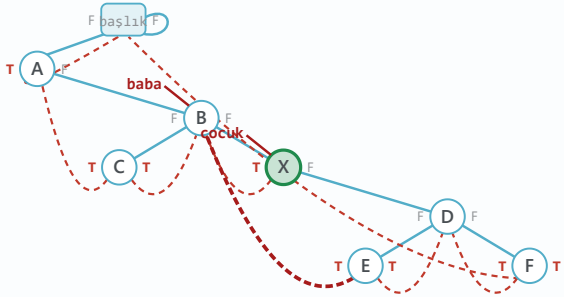
Adım 7 / 9

DEĞİŞKENLER

- `baba = &düğüm(B)`
- `cocuk = &düğüm(X)`
- `baba→sagKilavuz = FALSE`
- `cocuk→sagKilavuz = FALSE`

saga_ekle(B, X): tek yordam, 2. durum

```
1  void saga_ekle(kilavuz_gost baba,
2                 kilavuz_gost cocuk) {
3    // cocuk'u baba'nın sağ çocuğu yapar (kılavuzlu)
4    kilavuz_gost p;
5    cocuk->sagCocuk   = baba->sagCocuk;
6    cocuk->sagKilavuz = baba->sagKilavuz;
7    cocuk->solCocuk   = baba;
8    cocuk->solKilavuz = TRUE;
9    baba->sagCocuk    = cocuk;
10   baba->sagKilavuz  = FALSE;
11   if (!cocuk->sagKilavuz) { // ebeveynin alt ağacı var
     mı
▶      p = kokortada_ardil_bul(cocuk);
13     p->solCocuk = cocuk;     // ardılın öncülü çocuk
14   }
15 }
```



p = kokortada_ardil_bul(X) = **E** (D'nin en solu). E'nin eski öncülü B idi.

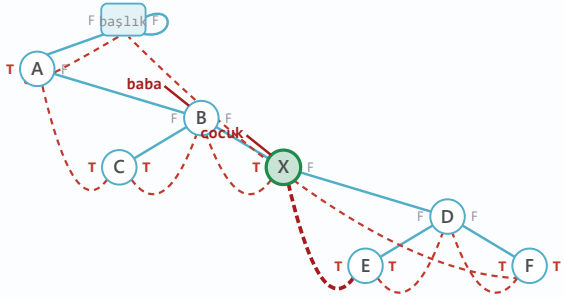
Adım 8 / 9

DEĞİŞKENLER

- `baba = &düğüm(B)`
- `cocuk = &düğüm(X)`
- `baba→sagKilavuz = FALSE`
- `cocuk→sagKilavuz = FALSE`
- `p = &düğüm(E)`

saga_ekle(B, X): tek yordam, 2. durum

```
1  void saga_ekle(kilavuz_gost baba,
2                 kilavuz_gost cocuk) {
3    // cocuk'u baba'nın sağ çocuğu yapar (kılavuzlu)
4    kilavuz_gost p;
5    cocuk->sagCocuk   = baba->sagCocuk;
6    cocuk->sagKilavuz = baba->sagKilavuz;
7    cocuk->solCocuk   = baba;
8    cocuk->solKilavuz = TRUE;
9    baba->sagCocuk    = cocuk;
10   baba->sagKilavuz  = FALSE;
11   if (!cocuk->sagKilavuz) { // ebeveynin alt ağacı var
     mı
12     p = kokortada_ardil_bul(cocuk);
▶      p->solCocuk = cocuk;     // ardılın öncülü çocuk
14   }
15 }
```



p->solCocuk = cocuk: E'nin sol kılavuzu artık X'i gösterir. **Çocuk, ebeveynin eski ardılının öncülü yapıldı.**

Adım 9 / 9

DEĞİŞKENLER

- `baba = &düğüm(B)`
- `cocuk = &düğüm(X)`
- `baba→sagKilavuz = FALSE`
- `cocuk→sagKilavuz = FALSE`
- `p = &düğüm(E)`

DÜĞÜM EKLEME

saga_ekle satır satır

[1–2] Açık tanım
Adı belli, giden parametreler belli, dönüş değeri belli.

[5–6]
Çocuğun sağ işaretçi ve sağ kılavuzu ebeveynden alınır: 1. durumda kırmızılık (TRUE), 2. durumda siyahlık (FALSE) korunur.

[7–8]
Çocuğun kök ortada öncülü ebeveyn: sol işaretçi ebeveyne, sol kılavuz TRUE (kırmızı).

[10]
Ebeveynin sağ kılavuzuna FALSE: 2. durumda gereksiz ama zararsız; 1. durumda gerekli.

[11–13]
Çocuğun sağ kılavuzu FALSE ise (2. durum) ardılın sol kılavuzu çocuğa çevrilir; 1. durumda yalnızca bir test yapılıp dönülür.

Büyük/küçük harf
C büyük/küçük harfe duyarlıdır: `baba->SagCocuk`, `baba->SagKilavuz` (büyük S) derlenmez. Denetlenen koşul ebeveynin **sağ** çocuğunun olup olmadığıdır.

NULL mı kontrolü nerede yapılmalı?

HOCADAN "saga_ekle yordamı, müstakil olarak kullanılmayıp, başka bir yordam içinde kullanılacağı için NULL mı kontrolünü burada yapmaya gerek yoktur. Bu kontrolü, saga_ekle yordamını kullanan yordamda yapmak gerekir. Ama saga_ekle yordamını müstakil olarak birisi kullanacaksa o zaman bunu da kontrol etmeliyiz."

- Örnek: kılavuzlu arama ağacına eklerken sağa mı sola mı ekleneceği bulununca, tek atama yerine `saga_ekle` ya da `sola_ekle` çağrılır; ebeveyn zaten NULL değildir.
- Kodun giriş kısmında denetim yapılmıyorsa bu **açıklamada** belirtilmelidir: "baba ve cocuk NULL olmamalı".
- Genel kütüphane yordamında savunmacı denetim (`assert` ya da hata dönüşü) eklenir.

Açıklama satırı ne söylemeli?

```
/* KÖTÜ: kodu tekrarlıyor */
if (!cocuk->sagKilavuz)  // sağ kılavuz FALSE mu
/* İYİ: niyeti söylüyor */
if (!cocuk->sagKilavuz)  // ebeveynin alt ağacı var mı
```

- Açıklama, kodun **ne yapmak istediğini** söylemeli; kodun kendisini tekrarlamamalı.
- Böylece okuyan "yapmak istediği doğru, ama kodu yanlış yazmış" diyebilir: hata ayıklama kolaylaşır.

HOCADAN "Buraya "ebeveynin sağ çocuğu FALSE mu" diye sakın yazılmasın! Sonradan gelen okuyucu "Ben onu zaten biliyorum." demesin. Açıklama satırları, kodda söylenenden farklı şeyler de söylesin."

Tek yordama indirmek ve yazılım mühendisliği

HOCADAN “Her duruma ayrı ayrı yordam yazsak, işin tadı kaçacak; cocukYok_sagaEkle, cocukVar_sagaEkle, cocukYok_solaEkle, cocukVar_solaEkle. Karmaşıklığı artırmadan bunları tek yordama indirebilirsek, iyi! … “İç içe if… if… deriz, tüm durumları tek yordamda yazabiliriz.” Ondan kaçınılmalı. Öyle olacağına ayrı ayrı yordamlar olsun.”

- Önce 3–5 farklı durum analiz edilir; sonra yordam adı, dönüş değeri, parametreler belirlenir.
- Ortak işler ortak biçimde tanımlanır (burada: kılavuz bayrağını ebeveynden kopyalamak).
- Sonuç: kod bölük pörçük değil; sade, okunabilir, bakımı kolay.

HOCADAN “Yazılım mühendisliği kavramlarını keşke 1. sınıfta öğretseler, ama çok erken olur. … Bu bölüm bana göre 6 yıl olmalı.”

SORU: sola ekleme yordamını düşününüz

Çocukla işaret edilen öğeyi, ebeveynin **sol** çocuğu olacak biçimde kılavuzlu ağaca ekleyen `sola_ekle` yordamını yazınız.

- Sol ↔ sağ, öncül ↔ ardıl yer değiştirir: ebeveynin eski **öncülü**, çocuğun öncülü olur; onun sağ kılavuzu çocuğa çevrilir.
- Kök ortada öncül bulma: sol alt ağaç varsa onun **en sağı**, yoksa sol kılavuz.

Çözüm: sola_ekle ve kokortada_oncel_bul

```
kilavuz_gost kokortada_oncel_bul(kilavuz_gost L) {
  kilavuz_gost p = L->solCocuk;
  if (!L->solKilavuz)
    while (!p->sagKilavuz) p = p->sagCocuk;
  return p;
}
void sola_ekle(kilavuz_gost baba, kilavuz_gost cocuk) {
  cocuk->solCocuk   = baba->solCocuk;
  cocuk->solKilavuz = baba->solKilavuz;
  cocuk->sagCocuk   = baba;
  cocuk->sagKilavuz = TRUE;      // ardılı ebeveyn
  baba->solCocuk    = cocuk;
  baba->solKilavuz  = FALSE;
  if (!cocuk->solKilavuz) {      // baba'nın sol alt ağacı vardı
    kilavuz_gost p = kokortada_oncel_bul(cocuk);
    p->sagCocuk = cocuk;         // öncülün ardılı çocuk
  }
}
```

Kod yazarken teyakkuz!

HOCADAN “Kod yazarken teyakkuz! Her an dinamite basılabilir. “Bunu yapıyorum, ama doğru mu yapıyorum?””

HOCADAN “Bu satırdan sonra kodun doğru çalıştığına hemen karar verilmemeli. Daha işletilecek bir if bloğu var.”

Kılavuzlu ağaç

+ Yığınsız O(1) bellekli gezinme
+ Her düğümden ardıl/öncül O(h)'de, gezinmede amortize O(1)
+ NULL alanlar boşa gitmez

Bedeli

– Her düğümde 2 bayrak
– Ekleme/silme daha girift
– Her yordam kılavuzları gözetmeli

Laboratuvar: Kılavuzlu Kök Ortada Gezinme

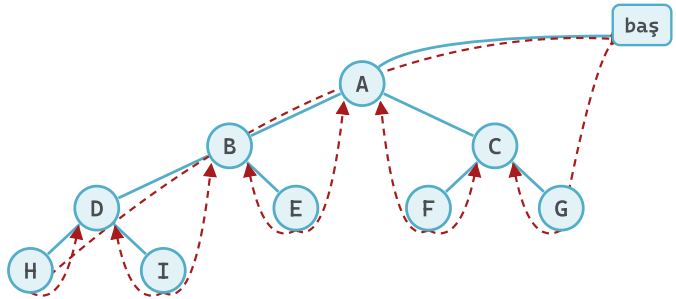
H D I B E A F C G

Girdi (düzey sırasıyla düğümler, boş için -) A, B, C, D, E, F, G, H, I

Uygula

KOD: kilavuz_kokortada_dolas

```
1 void kilavuz_kokortada_dolas(
2                       kilavuz_gost L) {
3   kilavuz_gost p = L;   // L: başlık düğümü
4   for (;;) {
5     p = kokortada_ardil_bul(p);
6     if (p == L) break; // başlığa dönüldü mü
7     printf("%c ", p->veri);
8   }
9 }
```



p – L –

ÇIKTI (henüz yok)

Kılavuzlu ağaç: boş sol bağ kök ortada öncülü, boş sağ bağ ardılı gösterir (kesikli oklar); uçtakiler başlık düğümünü gösterir. Gezinme yığınsız ve özyinelemesizdir.

ÖZET

Karmaşıklık özeti

işlem	zaman	ek bellek	not
kokortada_ardil_bul	O(h) en kötü	O(1)	kılavuz varsa O(1)
kilavuz_kokortada_dolas	O(n)	O(1)	yığın / özyineleme yok
kök önde kılavuzlu gezinme	O(n)	O(1)	aynı kılavuzlarla
saga_ekle / sola_ekle	O(h)	O(1)	ardıl/öncül araması
Morris gezinmesi (bayraksız)	O(n)	O(1)	geçici kılavuzlar
özyinelemeli kök ortada (karşılaştırma)	O(n)	O(h)	çağrı yığını

ÖZET

Sık yapılan hatalar

1. Kılavuz bayrağına bakmadan `solCocuk` / `sagCocuk`'u gerçek çocuk sanmak (sonsuz döngü).
2. Başlık düğümünü unutmak: H'nin sol ve G'nin sağ kılavuzu havada kalır.
3. Sağa eklemede 2. durumda ardılın sol kılavuzunu güncellememek.
4. `cocuk→sagKilavuz`'u sabit TRUE yazıp 2. durumu bozmak.
5. Boş ağaçta başlığın `solKilavuz = TRUE` olduğunu unutup kökü aramak.
6. Büyük/küçük harf: `SagCocuk` ≠ `sagCocuk` (C büyük/küçük harfe duyarlıdır).

ÖZET

Alıştırmalar

1. A(B(D, E), C(–, F)) ağacının kılavuzlarını çiziniz; kaç kılavuz vardır?
2. Kök ortada **öncül** bulan `kokortada_oncel_bul` yordamını yazınız.
3. Kılavuzlu ağacı kök ortada **tersten** (G C F A …) gezinen yordamı yazınız.
4. Boş ağaca (yalnız başlık) ilk düğümü `sola_ekle(baslik, x)` ile eklemek doğru mu?
5. Kök sonda gezinme kılavuzlarla neden zordur? Hangi ek bilgi işi kolaylaştırır?
6. Kılavuzlu ikili **arama** ağacına ekleme yordamını `saga_ekle` / `sola_ekle` kullanarak yazınız.

▶ **İpuçları / cevaplar**

ÖZET

Bölüm özeti

n + 1 NULL

2n işaretçinin n−1'i dolu; kalanlar kılavuz olur.

Kurallar

Sol NULL → öncül, sağ NULL → ardıl; bayraklar ayırt eder; başlık asılı kılavuzları toplar.

Ardıl

Sağ kılavuz TRUE ise doğrudan; değilse sağ alt ağacın en solu.

Gezinme

Yığınsız O(n) zaman, O(1) bellek; kök önde de mümkün.

Ekleme

saga_ekle: kılavuz bayrağı ebeveynden kopyalanır; 2. durumda ardılın öncülü çocuk.

Kod kalitesi

Açık tanım, anlamlı açıklama, NULL denetiminin yeri, tek yordam.

ÖZET

Kaynaklar ve devam

- Horowitz–Sahni–Anderson-Freed, *Fundamentals of Data Structures in C*, 5.5 Threaded Binary Trees.
- A. J. Perlis, C. Thornton, “Symbol manipulation by threaded lists”, *CACM* 3(4), 1960.
- J. M. Morris, “Traversing binary trees simply and cheaply”, *Information Processing Letters* 9(5), 1979; Knuth, TAOCP 1, 2.3.1.

← Bölüm 11: Arama Ağaçları

Ders sayfası

Bölüm 13: Öncelikli Kuyruk ve Heap

Son

Bölüm 12: Kılavuzlu İkili Ağaçlar

Sonraki bölüm: Bölüm 13: Öncelikli Kuyruk ve Heap · Ders sayfası