

VERİ YAPILARI · 3. HAFTA

Bağlı Liste

singly linked list

GEÇEN HAFTADAN ÜÇ SORU

Önce kısa bir hatırlama yapalım

1 Başa eklerken neden diğer elemanlar kaydı?

2 Python listesinde sona eklemek neden genelde hızlıydı?

3 class sözcüğü neyi tanımlar?

**Cevapları
bugün
kod üzerinde
göreceğiz**

Bağlı listeyi sıfırdan
kuracağız.

GEÇEN HAFTANIN BIRAKTIĞI SORU

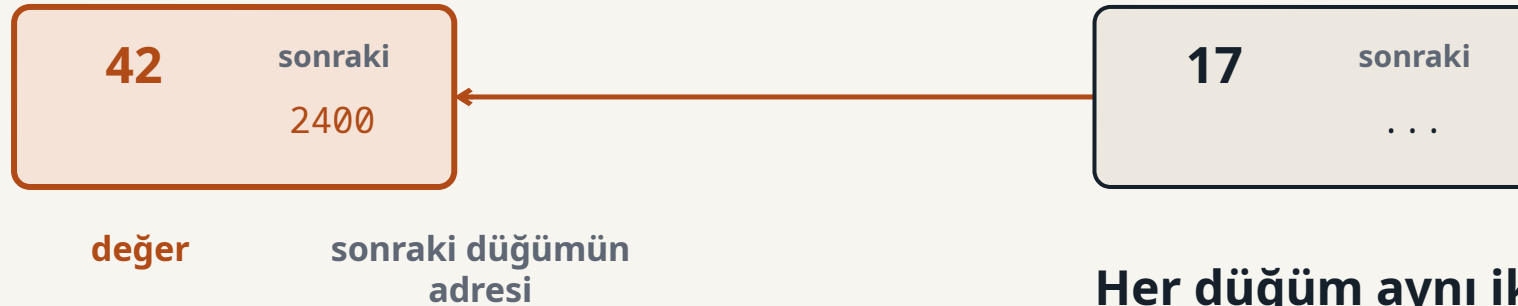
Başa eklerken kimseyi kaydırmamak olmaz mı?

Bugün bu soruya başka bir veri yapısıyla cevap vereceğiz.

TEMEL PARÇA

Düğüm nedir?

Düğüm (node), bir değeri ve bir sonraki düğümün adresini tutar.



Her düğüm aynı iki bilgiyi taşır.

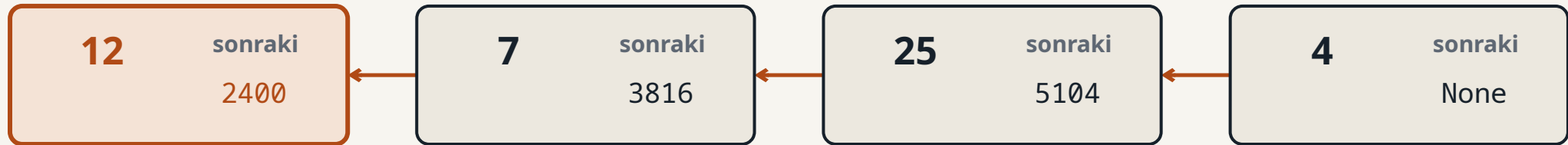
Düğümün bellekte nerede olduğu değerinden bağımsızdır.

SINGLY LINKED LIST

Tek yönlü bağlı liste

Düğümler, sonraki adreslerini göstererek bir sıra oluşturur.

baş listenin ilk düğümü



None listenin sonunu gösterir.

BUGÜNÜN ANA SORUSU

**Kimseyi kaydırmadan
başına ekleyebiliyorsak
dizi neden hâlâ var?**

Bir yapının kazancı başka bir işlemde bedel çıkarabilir.

PYTHON

class ile bir düğüm

class, aynı yapıya sahip nesnelerin tarifini yazar.

```
class Dugum:  
    def __init__(self, veri):  
        self.veri = veri  
        self.sonraki = None
```

```
ilk = Dugum(42)  
print(ilk.veri)
```

Turuncu satır işlemin kaynağını gösterir

Düğümün iki alanı

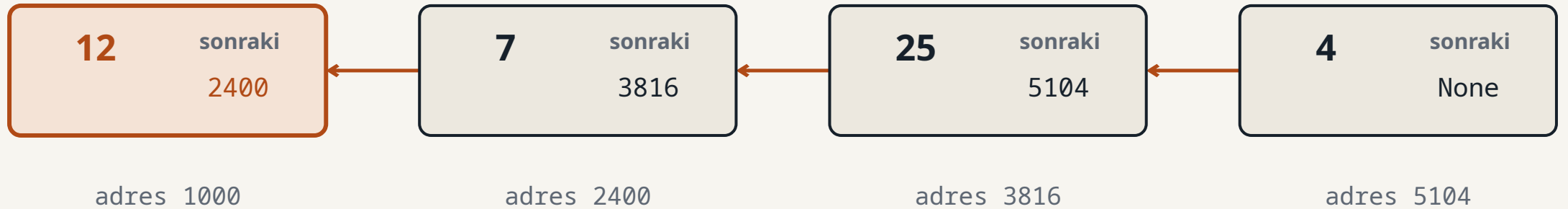
veri saklanan veri

sonraki sonraki düğüm

self nesnenin kendisini gösterir.

Düğüm bellekte tek kutu değildir

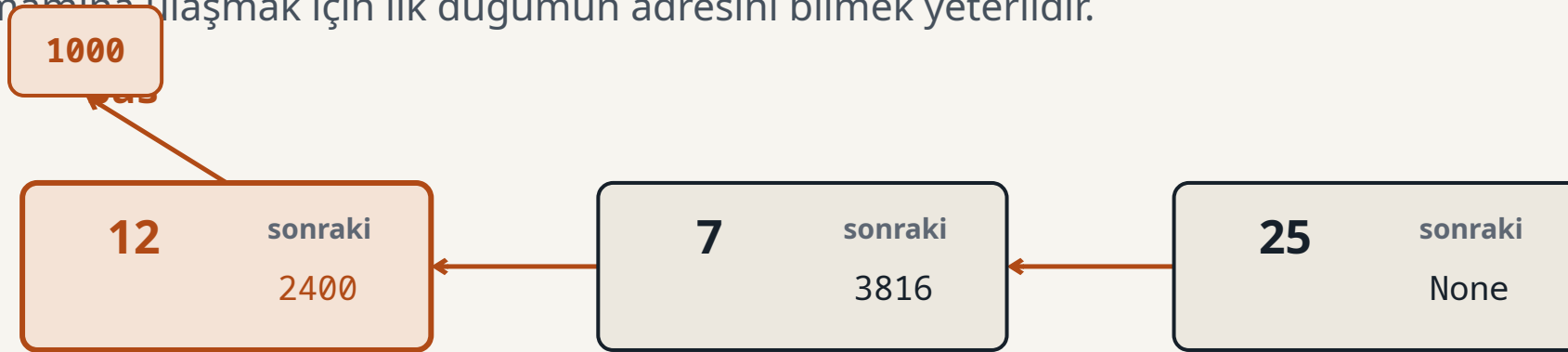
Her düğüm farklı bir adreste durabilir.



Dizideki gibi adres hesabı yapmayacağız. Bir sonraki adrese bakacağız.

Bağlantı adresi taşır

Listenin tamamına ulaşmak için ilk düğümün adresini bilmek yeterlidir.



bas değişirse listenin ilk düğümü değişir.

Başlangıç adresini saklayan alana bas diyeceğiz.

Baş'a ekleme

Yeni düğüm listenin başına bağlanır.



Başa ekleme kodla

Yeni düğüm, eski bas değerini devralır.

```
def basa_ekle(self, veri):  
    yeni = Dugum(veri)  
    yeni.sonraki = self.bas  
    self.bas = yeni  
    self.boyut += 1
```

Turuncu satır işlemin kaynağını gösterir

Üç bağlantı adımı

- 1 Düğümü oluştur
- 2 Eski başı göster
- 3 Başı güncelle

Başa eklemenin maliyeti

Listenin uzunluğu ne olursa olsun birkaç bağlantı değişir.

$O(1)$

sabit zaman

Veri 2 katına çıkınca

iş aynı kalır

Yeni düğüm yalnızca ilk düğümü gösterir.

Sona ekleme

Bu derste son düğümün adresini ayrıca saklamıyoruz.

bas → 1 → 2 → 3 → None

Son düğümü bulmak için baştan başlayıp bağlantıları tek tek izleriz.

Başa ekleme $O(1)$ kalır. Sona ekleme için listeyi gezeriz.

$O(n)$

Sona ekleme kodla

son düğümü bulmak için gezici ile dolaşırız.

```
def sona_ekle(self, veri):
    yeni = Dugum(veri)
    if self.bas is None:
        self.bas = yeni
        self.boyut += 1
        return
    gezici = self.bas
    while gezici.sonraki != None:
        gezici = gezici.sonraki
    gezici.sonraki = yeni
    self.boyut += 1
```

Boş liste

bas yeni düğümü gösterir

Dolu liste

gezici son düğüme ulaşır

Sona eklemenin maliyeti

Bu sınıfta sona eklemek için son düğümü baştan buluruz.

Başa ekleme

bas değişir

$O(1)$

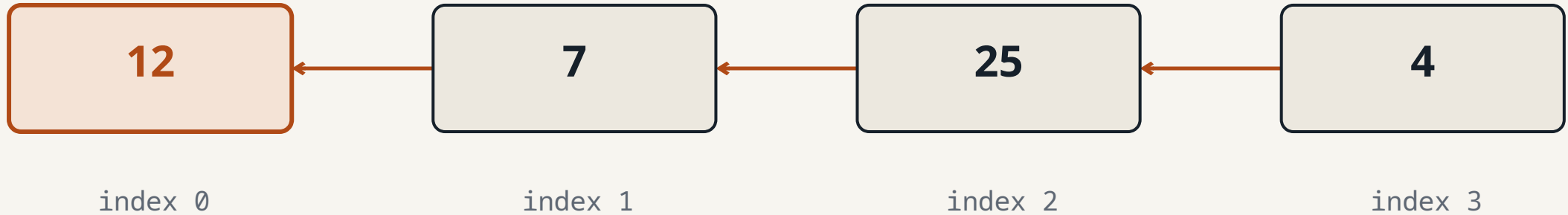
Sona ekleme

son düğümü bulur

$O(n)$

Sıra numarası yok, bağlantı var

Bağlı listede istenen sıra numarasına gitmek için bağlantıları izleriz.



Sıra numarasını bulmak için baştan ilerleriz.

500 numaralı sıraya gitmek

Index 500'deki elemana ulaşmak için 500 bağlantıyı izlemek gerekir.



$O(n)$

Index 500'e ulaşmak için baştan başlayıp 500 kez ilerleriz.

Dizi burada $O(1)$ avantajını korur.

Arama

Arama da düğümleri baştan sona kontrol eder.

```
def ara(self, aranan):  
    gezici = self.bas  
    while gezici is not None:  
        if gezici.veri == aranan:  
            return gezici  
        gezici = gezici.sonraki  
    return None
```

Turuncu satır işlemin kaynağını gösterir

Aranan başta olabilir

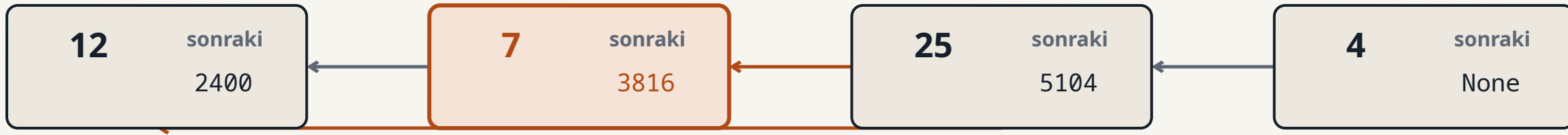
en iyi durum $O(1)$

Aranan sonda ya da yoksa

en kötü durum $O(n)$

Silme için önce öncekini buluruz

Ortadaki düğümü çıkarırken bağlantıyı üzerinden geçiririz.



silinecek

12 → 25

önceki.sonraki

Silme kodla

Düğümü bulurken bağlantıyı kaybetmemeliyiz.

```
def sil(self, silinecek):
    if self.bas is None: return False
    if self.bas.veri == silinecek:
        self.bas = self.bas.sonraki
        self.boyut -= 1
        return True
    onceki = self.bas
    while onceki.sonraki is not None:
        if onceki.sonraki.veri == silinecek:
            onceki.sonraki = onceki.sonraki.sonraki
            self.boyut -= 1
            return True
        onceki = onceki.sonraki
    return False
```

Üç durum

- 1 boş liste
- 2 ilk düğüm
- 3 ortadaki ya da son düğüm

Bağlantıyı değiştirmeden düğümü koparmıyoruz.

Boş liste ve tek eleman

Bağlantı güncellerken sınır durumlarını ayrıca düşünürüz.

Boş liste

```
bas = None  
boyut = 0
```

Tek eleman

```
bas var  
sonraki = None
```

Tek eleman sil

```
bas = None  
boyut = 0
```

Bu kontroller kodun güvenli çalışmasını sağlar.

İşlem maliyetleri

Bu haftaki sınıfta yalnızca bas ve boyut alanlarını tutuyoruz

İşlem	Bağlı liste	Neden
Baş ekleme	$O(1)$	bas değişir
Sona ekleme	$O(n)$	sonu bulmak için gezer
Index ile okuma	$O(n)$	baştan ilerler
Arama	$O(n)$	bağlantıları izler
Silme	$O(n)$	öncekini bulur

Dizi mi bağlı liste mi?

Seçim, sık yaptığımız işlere göre değişir.

Dizi

index ile okuma $O(1)$

başa ekleme $O(n)$

belirli bellekte bitişik alan

Bağlı liste

başa ekleme $O(1)$

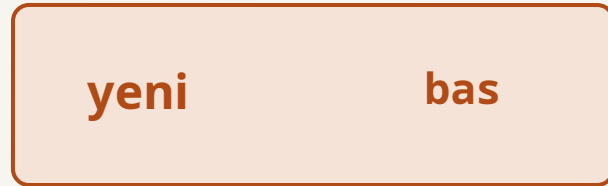
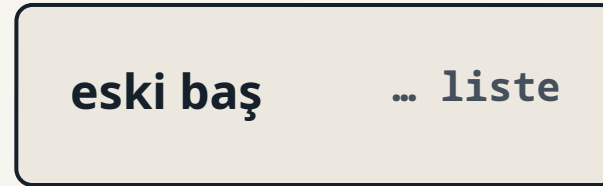
index ile okuma $O(n)$

düğüm adresleri farklı adreslerde

Hangi işlem sık yapılıyorsa ona göre seçeriz.

En sık yapılan hata

Yeni düğümü bağlamadan başı değiştirirsek listenin geri kalanını kaybederiz.



yanlış sıra

Önce `yeni.sonraki = bas`

Sonra `bas = yeni`

Bağlantıyı koru, sonra başlangıcı değiştir.

ANA SORUNUN CEVABI

Başa ekleme ucuzladı.

Karşılığında sıra numarasıyla erişim kayboldu.

Bağlı listeyi, sık yaptığımız işlemler başa ekleme ve bağlantı üzerinden ilerleme olduğunda seçeriz.

Kodda kullandığımız yapı

Notebook'ta düğüm ve listeyi iki sınıfla kuracağız.

```
class Dugum:
    def __init__(self, veri):
        self.veri = veri
        self.sonraki = None

class BagliListe:
    def __init__(self):
        self.bas = None
        self.boyut = 0
```

Turuncu satır işlemin kaynağını gösterir

Ders sırası

- 1 Düğümü oluştur
- 2 basa_ekle ve sona_ekle
- 3 Ara ve sil
- 4 Çıktıyı birlikte oku

TAHMİN ALIŞTIRMASI

İlk üç fonksiyonun Big O değerini tahmin edin.

`ilk_degeri_al(liste)`

`O( )`

`basa_ekle(liste, deger)`

`O( )`

`sona_ekle(liste, deger)`

`O( )`

Cevabı kağıda yazın.
Kod maliyeti söylemez.

İpucu

**İşlem kaç düğüm
gezebilir?**

TAHMİN ALIŞTIRMASI

Son üç fonksiyonun Big O değerini tahmin edin.

`deger_ara(liste, aranan)`

`O( )`

`deger_getir(liste, indeks)`

`O( )`

`deger_sil(liste, silinecek)`

`O( )`

Cevapları notebook'ta
beraber kontrol edeceğiz.

Cevap için ipucu

**İşlem kaç düğümü
gezmek zorunda?**

DERSİN TEK CÜMLESİ

**Bağlı liste bağlantıları
değiştirerek eklemeyi hızlandırır.**

Bunun bedeli, sıraya göre erişimin yavaşlamasıdır.

GELECEK HAFTA

**Geri al tuşu en son
ne yaptığımızı nereden biliyor?**

yığın · stack