Bölüm 8: Çevrim İçi Algoritmalar

İleri Algoritmalar

Çevrim içi ve çevrim dışı | Rekabetçi analiz | Kayak kiralama | Sayfalama: FIFO, LRU, OPT | Liste güncelleme | Veri akışı algoritmaları | Rezervuar örnekleme | Artımlı öğrenme

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü:
https://sercankulcu.github.io/files/advanced_algorithms/slides/Bolum_08_Cevrim_Ici_Algoritmalar.html

GİRİŞ

Öğrenme Çıktıları

Model

Çevrim içi (online) ve çevrim dışı (offline) problem modelini ayırt etmek; geri alınamaz karar kavramını açıklamak.

Rekabetçi analiz

Rekabet oranını (competitive ratio) tanımlamak; kayak kiralama problemi için 2 − 1/B oranını kanıtlamak.

Sayfalama ve listeler

FIFO, LRU ve OPT ile sayfa hatalarını adım adım hesaplamak; LRU'nun k-rekabetçi, MTF'nin 2-rekabetçi olduğunu yorumlamak.

Veri akışları

Hareketli ortalama, çevrim içi ortanca ve rezervuar örneklemeyi Java ile yazmak; öğe başına zaman ve belleği analiz etmek.

GİRİŞ

Bilgi Olsun Yeter

- Bazen kararlar **tüm bilgiye sahip olunmadan** alınır.
- **Çevrim içi algoritmalar** (online algorithms):
  - bilginin **adım adım** geldiği durumlarda
  - o ana kadarki bilgiyle **en iyi kararı** vermeye çalışır.
- Verilen karar **geri alınamaz**: sonradan gelen bilgi geçmiş kararı değiştirmez.

Satranç oyuncusu rakibin sonraki hamlesini bilmeden hamle yapar; hamle bir kez yapıldıktan sonra geri alınamaz.



GİRİŞ

Çevrim İçi Algoritmalar

- Bir **labirentte** yolculuk yapmaya benzer.
  - Labirentin tamamı görünmez; yalnızca öndeki yol görülür.
- Her adımda **sınırlı bilgi** ile en iyi yol seçilmeye çalışılır.
- Bilgi akışının **sürekli** olmadığı durumlarda **hızlı karar** vermeyi sağlar.
- Örnekler:
  - Trafik sıkışıklığında hangi yol kullanılacak?
  - Satranç oynarken bir sonraki hamle ne olacak?
  - Anlık trafik yoğunluğuna göre ışıkların ayarlanması.



GİRİŞ

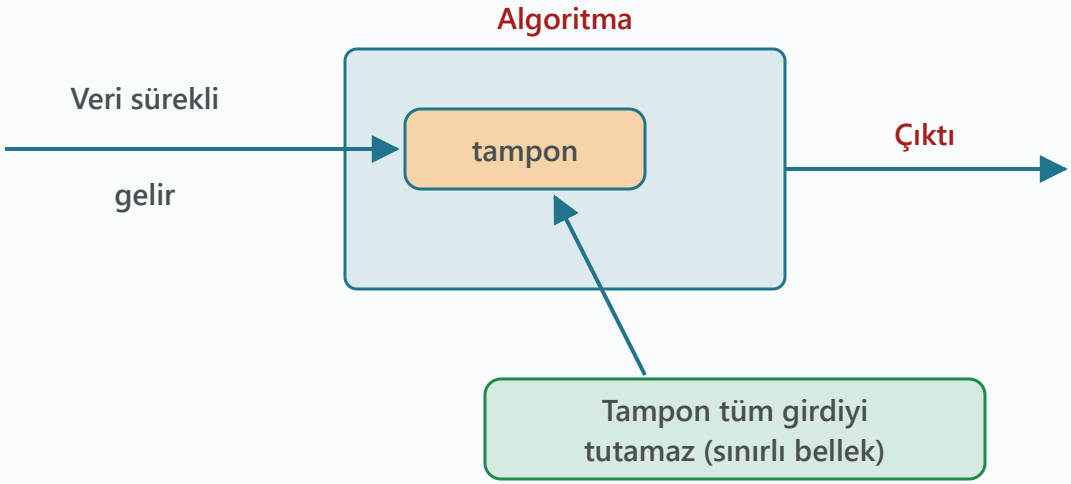
Çevrim İçi ve Çevrim Dışı Problem Modeli

Girdi bir **istek dizisidir**: $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_n$. Fark, algoritmanın bu diziyi **ne zaman** gördüğündedir.

	Çevrim dışı (offline)	**Çevrim içi (online)**
Girdi	Tamamı baştan bilinir	Parça parça gelir; σ_t gelince σ_{t+1} bilinmez
Karar	Tüm girdiye bakılarak verilir	σ_t geldiği anda verilmeli, **geri alınamaz**
Örnek	Sıralama, Dijkstra, sırt çantası (DP)	Önbellek yönetimi, kayak kiralama, borsa
En iyi çözüm	OPT hesaplanabilir	OPT ancak geriye dönüp bakınca bilinir
Ölçüt	Zaman / alan karmaşıklığı	**Rekabet oranı**: ALG / OPT

"Çevrim içi" burada *internet* anlamına gelmez; **girdinin zaman içinde açığa çıkması** demektir. Eklemeli sıralama (insertion sort) çevrim içi çalışabilir, seçimli sıralama (selection sort) çalışamaz.

Çevrim İçi Algoritma Modeli (Online Algorithms)



- Veri **sürekli** gelir; algoritma her parçayı geldiğinde işler ve hemen çıktı üretir.
- Bellek (tampon, buffer) **tüm girdiyi tutamaz**: geçmiş ancak özet olarak saklanabilir (toplam, heap, örnek…).
- Bu yüzden her öğe için **az iş** (çoğunlukla O(1) ya da O(log n)) ve **az bellek** hedeflenir.

Uygulama Alanları

Borsa
Anlık fiyata göre al-sat kararı; gelecek fiyat bilinmez.

Önbellek
İşlemci, işletim sistemi, web: hangi sayfa bellekte kalsın?

Yönlendirme
Bağlantıları sürekli değişen ağda en kısa yol.

Kaynak tahsisi
Bant genişliği ve CPU zamanının dinamik paylaşımı.

Zamanlama
İşler geldikçe makinelere atanır (iş çizelgeleme).

Reklam ve öneri
Kullanıcı geldiğinde hangi reklam / ürün gösterilsin?

Makine öğrenmesi
Model her yeni örnekle güncellenir (artımlı öğrenme).

Güvenlik
Dolandırıcılık ve DDoS saldırılarını akış üzerinde yakalama.

Otonom araçlar
Algılayıcı verisi geldikçe gerçek zamanlı karar.

Ortak nokta: ortam **dinamik**, karar **gerçek zamanlı**, geleceğe dair bilgi **eksik**.

GİRİŞ

Avantajlar ve Dezavantajlar

Avantajlar

+ **Dinamik** ortamlarda çalışır; girdinin bitmesini beklemez.
+ **Gerçek zamanlı** karar verir; gecikme düşüktür.
+ Az bellekle çok büyük, hatta **sonsuz** akışları işleyebilir.
+ Rekabet oranı ile **en kötü durum garantisi** verilebilir.

Dezavantajlar

– İlk adımlarda **yanlış kararlar** alınabilir ve sonraki adımları etkiler.
– Satrançta yanlış bir hamle oyunu kaybettirebilir.
– Her zaman **en iyi global çözümü** bulmayı garanti etmez.
– Başlangıçtaki girdiye duyarlıdır; tasarımı ve analizi daha karmaşıktır.

GİRİŞ

Çevrim İçi Algoritma Tasarım Yaklaşımları

Açgözlü (greedy)
Geleni o an en iyi görünen biçimde karşılar; küresel etkiyi düşünmez. Örnek: istek geldiğinde en yakın boş sunucuya atama.

Özetleme
Geçmişi tümüyle saklamak yerine küçük bir özet tutar: toplam, iki heap, örneklem. Örnek: hareketli ortalama, çevrim içi ortanca.

Uyarlanır (adaptive)
Hatalarından öğrenir, davranışını zamanla değiştirir. Örnek: LRU, öne taşı (MTF), artımlı öğrenme.

Deterministik
Aynı girdiye hep aynı kararı verir; kötü niyetli bir girdi dizisi (düşman) onu kolayca tuzağa düşürebilir.

Rastgele (randomized)
Kararlarına rastlantı katar; düşman ne yapacağını önceden bilemez. Örnek: rezervuar örnekleme, rastgele işaretleme.

Rekabetçi Analiz (Competitive Analysis)

- Çevrim içi algoritma ALG, her girdide **her şeyi önceden bilen** en iyi çevrim dışı algoritma OPT ile karşılaştırılır.
- ALG, **c-rekabetçidir** (c-competitive), eğer her istek dizisi σ için

$$\mathrm{ALG}(\sigma) \le c \cdot \mathrm{OPT}(\sigma) + \alpha$$

  koşulu sağlanır (α: σ'dan bağımsız sabit; α = 0 ise **kesin** rekabetçi).
- En küçük böyle c değerine **rekabet oranı** denir: $c = \sup_{\sigma} \mathrm{ALG}(\sigma) / \mathrm{OPT}(\sigma)$.
- Oran ne kadar **1'e yakınsa**, gelecek bilgisinin eksikliği o kadar az zarar verir.

Kötümser bakış

Analiz **en kötü girdi** üzerinedir: ortalama ya da olasılık varsayımı yoktur.

Maliyet problemi

Burada maliyet en aza indirilir ($c \ge 1$). Kazanç problemlerinde oran OPT / ALG olarak tanımlanır.

REKABETÇİ ANALİZ

Düşman (Adversary) Modeli

Rekabet oranını **alttan sınırlamak** için algoritmaya karşı oynayan bir düşman düşünülür: girdiyi o üretir ve OPT'yi o çalıştırır.

Deterministik ALG

Düşman algoritmanın kodunu bilir; her adımda ALG'nin **en çok zarar göreceği** isteği üretir.

Habersiz düşman

(oblivious) Rastgele algoritmanın yazı-tura sonuçlarını göremez; diziyi baştan seçer. Beklenen ALG maliyeti ölçülür.

Uyarlanır düşman

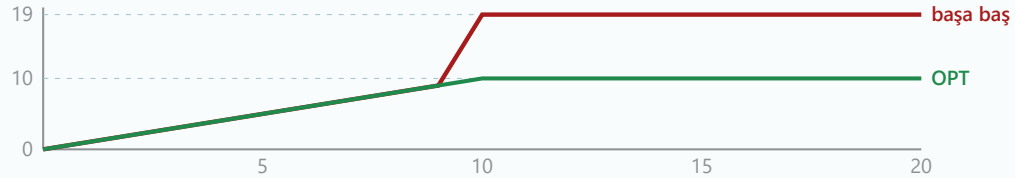
(adaptive) ALG'nin rastgele kararlarını görerek sonraki isteği seçer; rastgeleliğin faydası azalır.

Kanıt kalıbı: "Hiçbir deterministik algoritma c'den iyi olamaz" demek için, her ALG'ye karşı $ALG(\sigma) / OPT(\sigma) \geq c$ yapan bir σ dizisi kurulur.

KAYAK KİRALAMA

Kayak Kiralama Problemi (Ski Rental)

- Kayak takımı **günlüğü 1 TL'ye kiralanabilir** ya da **B TL'ye satın alınabilir**.
- Kaç gün kayak yapılacağı (d) **bilinmez**: her sabah "bugün de kayacak mıyım?" sorusu yanıtlanır. Hava bozarsa sezon biter.
- Her gün karar: **bugün de kirala** mı, yoksa **satın al** mı?
- Geleceği bilen OPT kolaydır: **OPT(d) = min(d, B)**.



B = 10; yatay eksen gerçek gün sayısı d. En kötü oran d = B'de: 19/10 = 2 − 1/B.

Hep kirala
d → ∞ iken maliyet d, OPT = B: oran **sınırsız**.

İlk gün satın al
d = 1 iken maliyet B, OPT = 1: oran **B**.

Başa baş (break-even)
B − 1 gün kiralanır, B. gün satın alınır: oran **2 − 1/B**.

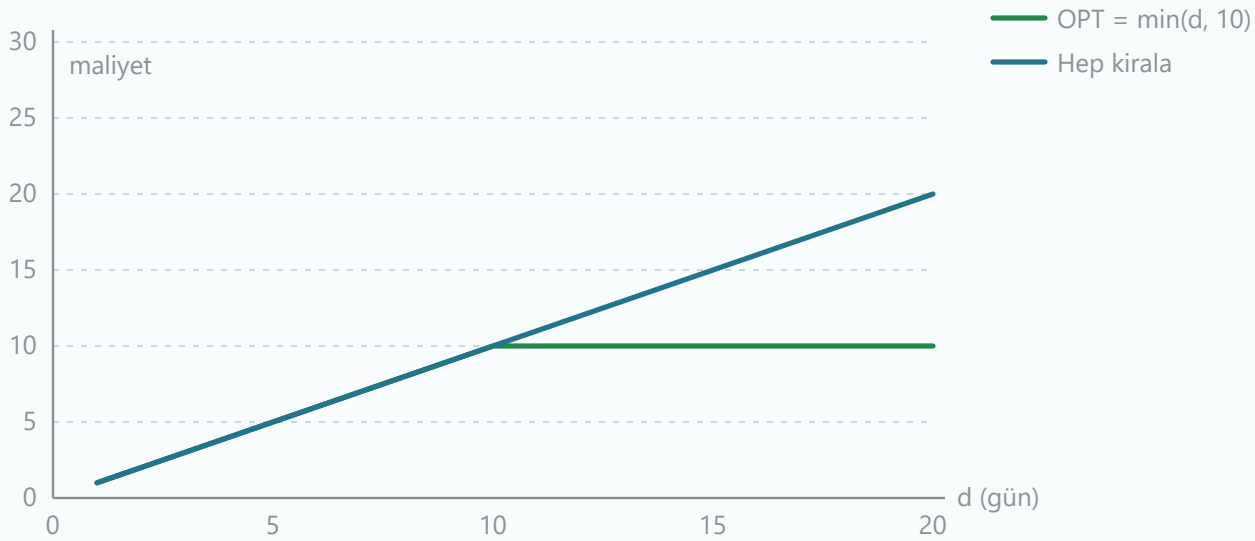
Aynı yapı: bulut sunucusunu saatlik mi kiralayalım, yıllık mı alalım; TCP bağlantısını açık mı tutalım, kapatalım mı.

Stratejilerin Maliyeti (B = 10)



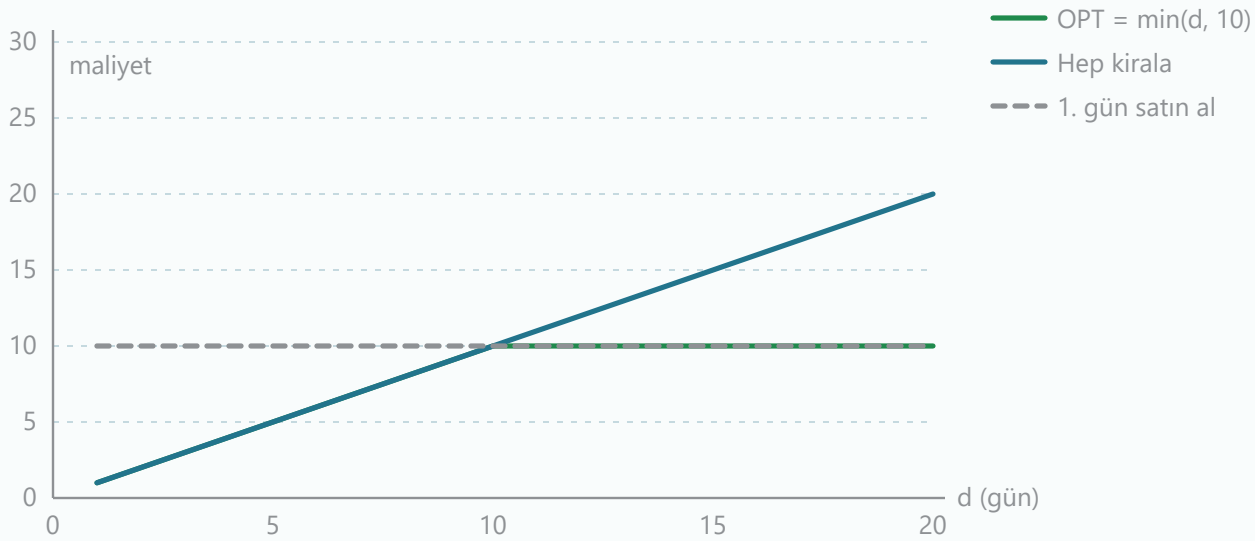
Geleceği bilen **OPT**: d < 10 ise hep kiralar (maliyet d), d ≥ 10 ise ilk gün satın alır (maliyet 10).

Stratejilerin Maliyeti (B = 10)



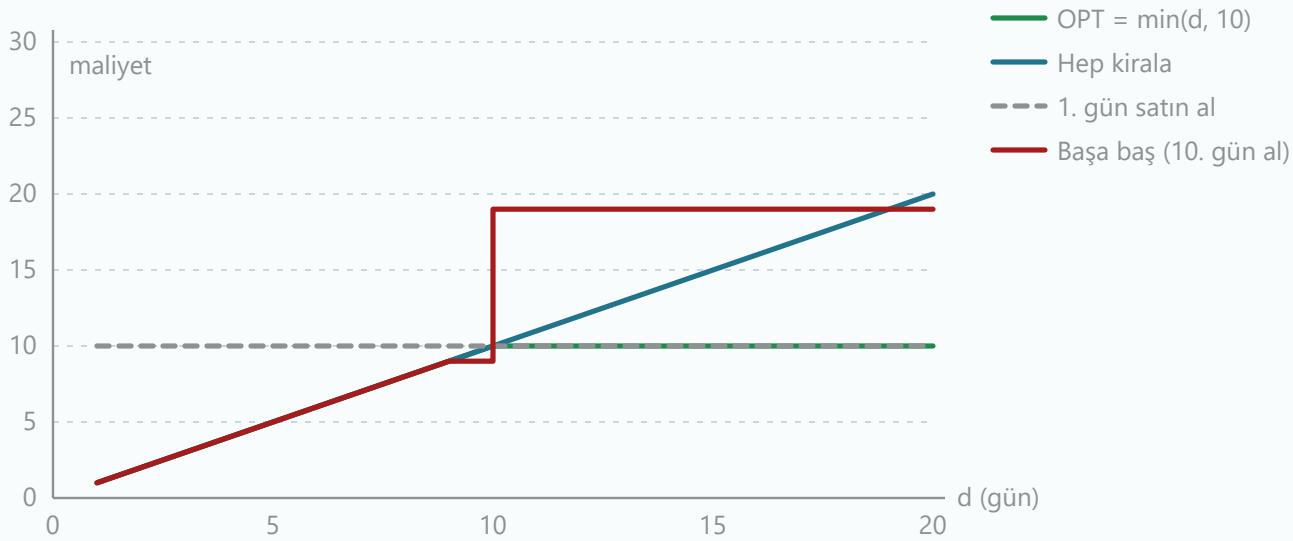
Hep kirala: maliyet d. d büyüdükçe OPT = 10 sabitken maliyet sınırsız artar → rekabetçi **değil**.

Stratejilerin Maliyeti (B = 10)



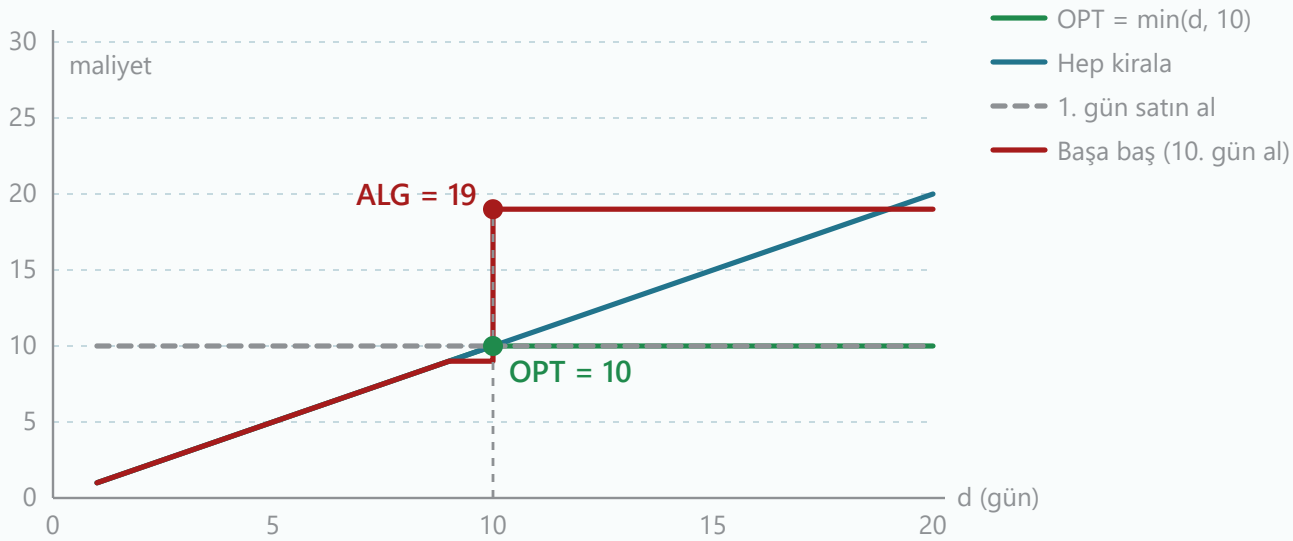
İlk gün satın al: maliyet hep 10. Ama d = 1 ise OPT = 1 → oran 10 = B. Kötü.

Stratejilerin Maliyeti (B = 10)



Başa baş: 9 gün kiralanır, 10. gün satın alınır. $d < 10$ iken OPT ile aynı; $d \geq 10$ iken maliyet $9 + 10 = 19$.

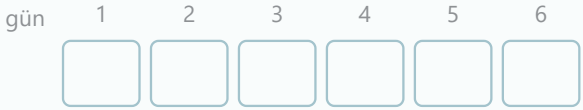
Stratejilerin Maliyeti (B = 10)



En kötü durum d = 10: ALG = 19, OPT = 10 → oran **1,9 = 2 − 1/B**. Hiçbir d bundan kötü değildir.

Başa Baş Stratejisi: Kodu Adım Adım İzle

```
1  int basaBas(int B, int d) {  // d bilinmez
▶    int maliyet = 0;
3    for (int gun = 1; gun <= d; gun++) {
4      if (gun < B) {
5        maliyet += 1;              // kirala
6      } else {
7        maliyet += B;              // satın al
8        break;                     // ücret biter
9      }
10   }
11   return maliyet;
12 }
```



ALG = 0

B = 4 TL (satın alma), kira günlüğü 1 TL. Sezon aslında d = 6 gün sürecek; algoritma bunu bilmez, her sabah yalnızca "bugün de kayıyor muyum?" sorusunu görür.

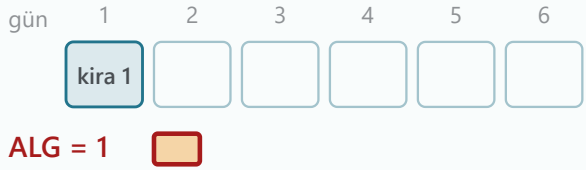
Adım 1 / 7

DEĞİŞKENLER

`B = 4` `gun = –` `maliyet = 0`

Başa Baş Stratejisi: Kodu Adım Adım İzle

```
1  int basaBas(int B, int d) {  // d bilinmez
2    int maliyet = 0;
▶    for (int gun = 1; gun <= d; gun++) {
▶      if (gun < B) {
▶        maliyet += 1;              // kirala
6      } else {
7        maliyet += B;              // satın al
8        break;                     // ücret biter
9      }
10   }
11   return maliyet;
12 }
```



Gün 1: 1 < B → kiralanır. maliyet = 1.

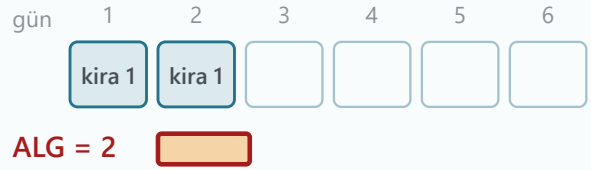
Adım 2 / 7

DEĞİŞKENLER

B = 4 | gun = 1 | maliyet = 1

Başa Baş Stratejisi: Kodu Adım Adım İzle

```
1  int basaBas(int B, int d) {  // d bilinmez
2    int maliyet = 0;
▶    for (int gun = 1; gun <= d; gun++) {
▶      if (gun < B) {
▶        maliyet += 1;              // kirala
6      } else {
7        maliyet += B;              // satın al
8        break;                     // ücret biter
9      }
10   }
11   return maliyet;
12 }
```



Gün 2: 2 < B → kiralanır. maliyet = 2.

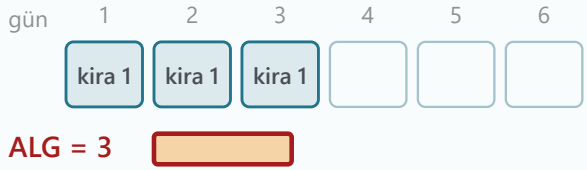
Adım 3 / 7

DEĞİŞKENLER

`B = 4` `gun = 2` `maliyet = 2`

Başa Baş Stratejisi: Kodu Adım Adım İzle

```
1  int basaBas(int B, int d) {  // d bilinmez
2    int maliyet = 0;
▶    for (int gun = 1; gun <= d; gun++) {
▶      if (gun < B) {
▶        maliyet += 1;              // kirala
6      } else {
7        maliyet += B;              // satın al
8        break;                     // ücret biter
9      }
10   }
11   return maliyet;
12 }
```



Gün 3: 3 < B → kiralanır. maliyet = 3.

Adım 4 / 7

DEĞİŞKENLER

B = 4 | gun = 3 | maliyet = 3

Başa Baş Stratejisi: Kodu Adım Adım İzle

```
1  int basaBas(int B, int d) {  // d bilinmez
2    int maliyet = 0;
3    for (int gun = 1; gun <= d; gun++) {
4      if (gun < B) {
5        maliyet += 1;              // kirala
▶      } else {
▶        maliyet += B;              // satın al
8        break;                     // ücret biter
9      }
10   }
11   return maliyet;
12 }
```



Gün 4 = B: kiraya verilen toplam B − 1 = 3 TL oldu; şimdi **satın alınır**: maliyet = 3 + 4 = 7.

Adım 5 / 7

DEĞİŞKENLER

`B = 4` `gun = 4` `maliyet = 7`

Başa Baş Stratejisi: Kodu Adım Adım İzle

```
1  int basaBas(int B, int d) {  // d bilinmez
2    int maliyet = 0;
3    for (int gun = 1; gun <= d; gun++) {
4      if (gun < B) {
5        maliyet += 1;              // kirala
6      } else {
7        maliyet += B;              // satın al
▶        break;                    // ücret biter
9      }
10   }
11   return maliyet;
12 }
```



`break` : kayak artık bizim; 5. ve 6. günler bedava.

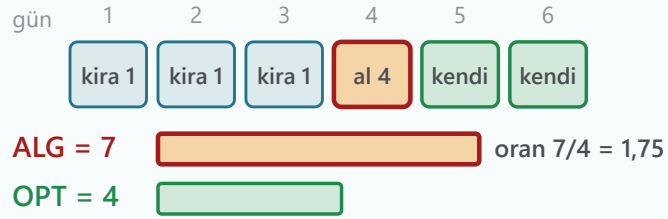
Adım 6 / 7

DEĞİŞKENLER

`B = 4` `gun = 4` `maliyet = 7`

Başa Baş Stratejisi: Kodu Adım Adım İzle

```
1  int basaBas(int B, int d) {  // d bilinmez
2    int maliyet = 0;
3    for (int gun = 1; gun <= d; gun++) {
4      if (gun < B) {
5        maliyet += 1;              // kirala
6      } else {
7        maliyet += B;              // satın al
8        break;                     // ücret biter
9      }
10   }
▶    return maliyet;
12 }
```



ALG = 7, OPT = min(6, 4) = 4 (ilk gün satın almak). Oran 7/4 = 1,75 = 2 − 1/B. d ne olursa olsun oran bunu aşmaz.

Adım 7 / 7

DEĞİŞKENLER

`maliyet = 7` `OPT = min(d, B)=4`

KAYAK KİRALAMA

Neden Daha İyisi Yok? Alt Sınır

Her deterministik strateji tek bir sayı ile tanımlanır: **k** = satın alma günü.

- **Düşman**: satın alma gününden sonra hava bozar, yani d = k.
- ALG = (k − 1) kira + B, OPT = min(k, B).
- k < B: oran (k − 1 + B) / k ≥ (2B − 1) / B.
- k ≥ B: oran (k − 1 + B) / B ≥ (2B − 1) / B.
- En iyisi **k = B**: oran **2 − 1/B**.

Rastgele seçilen k ile beklenen oran **e / (e − 1) ≈ 1,58**'e iner (Karlin vd., 1994).

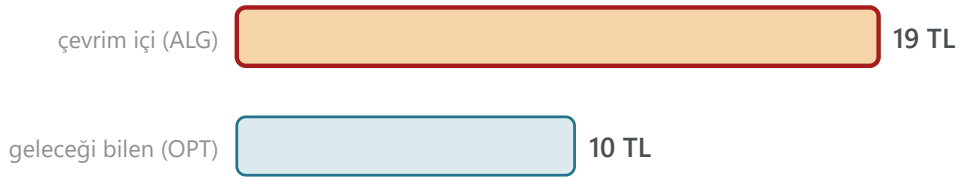
k	kira	ALG	OPT	oran
1	0	10	1	10
2	1	11	2	5,5
5	4	14	5	2,8
9	8	18	9	2
10	9	19	10	**1,9**
11	10	20	10	2
15	14	24	10	2,4
20	19	29	10	2,9

B = 10; d = k (düşmanın seçimi). Tablo, en kötü oranın k = B'de en küçük olduğunu gösterir.

KAYAK KİRALAMA

Deneyin: Kayak Kiralama

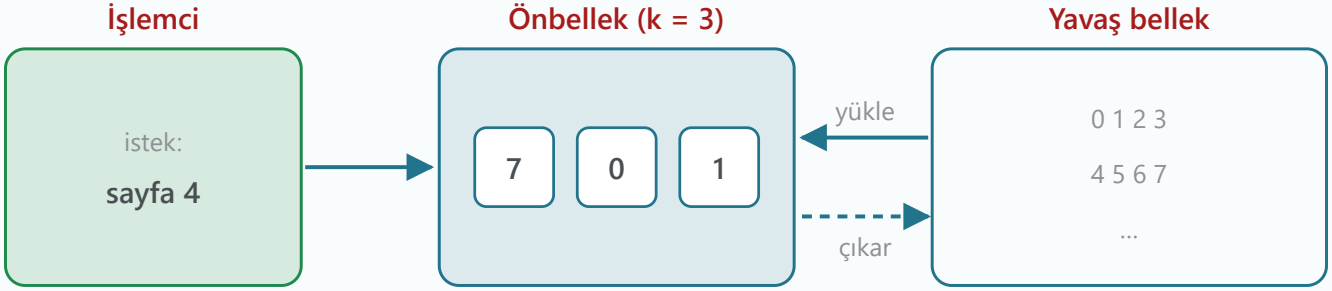
B **10** satın alma günü k **10** gerçek gün d **12**



9 gün kira + 10 TL satın alma. Oran ALG / OPT = **1,90** · bu k için en kötü oran (d = k) = **1,90** · en iyi deterministik: 2 − 1/B = **1,90**

Strateji: k − 1 gün kiralanır, k. gün hâlâ kayılıyorsa satın alınır. En kötü oran, düşmanın d = k seçtiği durumdur.

Sayfalama Problemi (Paging)



- Hızlı bellek (önbellek, cache) **k sayfa** tutar; yavaş bellek sınırsızdır.
- İstenen sayfa önbellekteyse **isabet** (hit, maliyet 0); değilse **sayfa hatası** (page fault, maliyet 1): sayfa yüklenir.
- Önbellek doluysa **hangi sayfa çıkarılacak**? Gelecek istekler bilinmez: tipik bir çevrim içi problem.
- Amaç: toplam **sayfa hatası sayısını** en aza indirmek.

SAYFALAMA

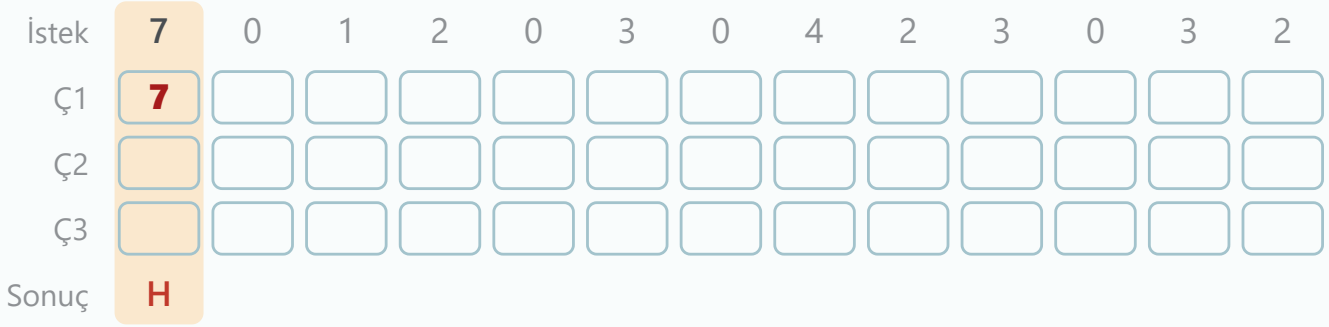
Sayfa Çıkarma Politikaları

Politika	Hangi sayfayı çıkarır?	Çevrim içi?	Rekabet oranı
FIFO	Önbelleğe **en önce yüklenen** sayfayı (kuyruk)	evet	k
LRU	**En uzun süredir kullanılmayan** sayfayı (least recently used)	evet	k
FWF	Hata olunca ve önbellek doluysa **hepsini** boşaltır (flush when full)	evet	k
LFU	**En az sıklıkla kullanılan** sayfayı (least frequently used)	evet	sınırsız
LIFO	**En son yüklenen** sayfayı	evet	sınırsız
OPT (MIN)	Bir sonraki kullanımı **en uzak gelecekte** olanı (Belady, 1966)	hayır	1

OPT (longest forward distance) geleceği bilmeyi gerektirir; uygulanamaz ama **karşılaştırma ölçütü** olarak kullanılır. Kanıtlanabilir biçimde en az sayfa hatasını verir.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

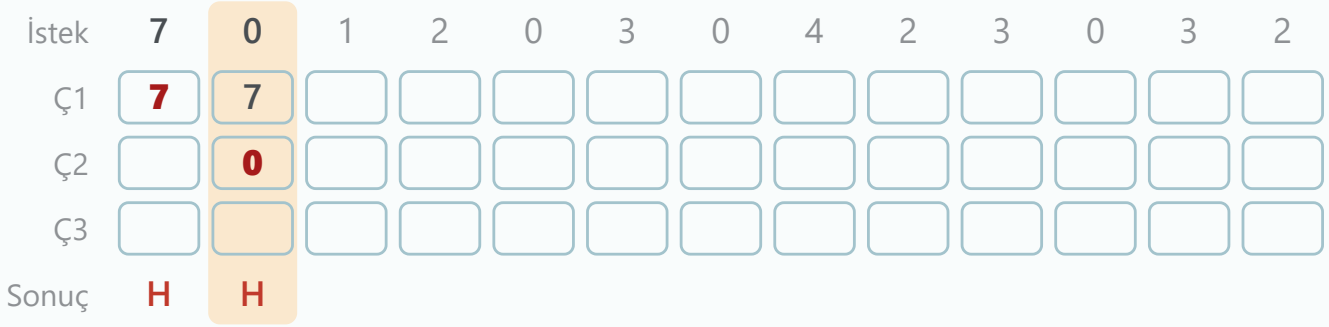


FIFO · k = 3 · sayfa hatası: 1

İstek 1: sayfa **7** yok → **sayfa hatası**; boş çerçeve Ç1'e yüklenir. Toplam hata: 1.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

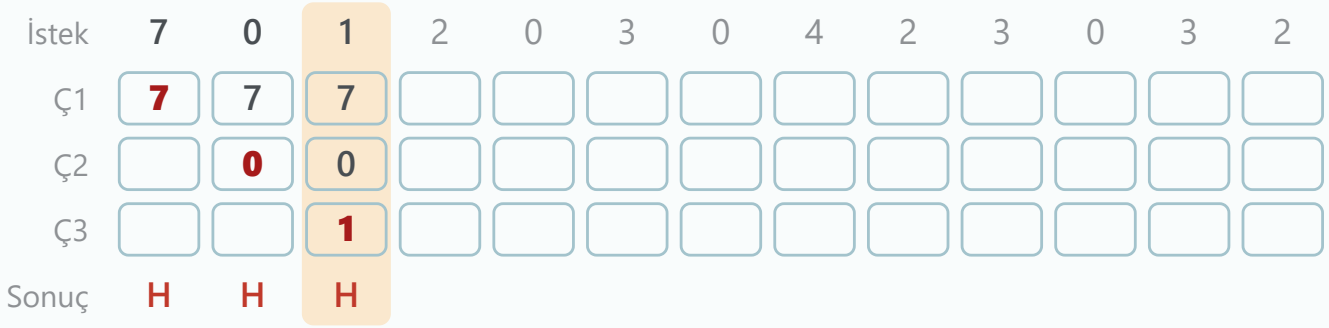


FIFO · k = 3 · sayfa hatası: 2

İstek 2: sayfa **0** yok → **sayfa hatası**; boş çerçeve Ç2'ye yüklenir. Toplam hata: 2.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.



FIFO · k = 3 · sayfa hatası: 3

İstek 3: sayfa **1** yok → **sayfa hatası**; boş çerçeve Ç3'e yüklenir. Toplam hata: 3.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**									
Ç2		**0**	0	0									
Ç3			**1**	1									
Sonuç	H	H	H	H									

FIFO · k = 3 · sayfa hatası: 4

İstek 4: sayfa **2** yok → **sayfa hatası**; en önce yüklenen **7** çıkarılır. Toplam hata: 4.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2								
Ç2		**0**	0	0	**0**								
Ç3			**1**	1	1								
Sonuç	H	H	H	H	✓								

FIFO · k = 3 · sayfa hatası: 4

İstek 5: sayfa **0** önbellekte → **isabet**. FIFO sırası **değişmez**. Toplam hata: 4.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2							
Ç2		**0**	0	0	**0**	**3**							
Ç3			**1**	1	1	1							
Sonuç	H	H	H	H	✓	H							

FIFO · k = 3 · sayfa hatası: 5

İstek 6: sayfa **3** yok → **sayfa hatası**; en önce yüklenen **0** çıkarılır. Toplam hata: 5.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2						
Ç2		**0**	0	0	**0**	**3**	3						
Ç3			**1**	1	1	1	**0**						
Sonuç	H	H	H	H	✓	H	H						

FIFO · k = 3 · sayfa hatası: 6

İstek 7: sayfa **0** yok → **sayfa hatası**; en önce yüklenen **1** çıkarılır. Toplam hata: 6.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	**4**					
Ç2		**0**	0	0	**0**	**3**	3	3					
Ç3			**1**	1	1	1	**0**	0					
Sonuç	H	H	H	H	✓	H	H	H					

FIFO · k = 3 · sayfa hatası: 7

İstek 8: sayfa **4** yok → **sayfa hatası**; en önce yüklenen **2** çıkarılır. Toplam hata: 7.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	**4**	4				
Ç2		**0**	0	0	**0**	**3**	3	3	**2**				
Ç3			**1**	1	1	1	**0**	0	0				
Sonuç	H	H	H	H	✓	H	H	H	H				

FIFO · k = 3 · sayfa hatası: 8

İstek 9: sayfa **2** yok → **sayfa hatası**; en önce yüklenen **3** çıkarılır. Toplam hata: 8.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	**4**	4	4			
Ç2		**0**	0	0	**0**	**3**	3	3	**2**	2			
Ç3			**1**	1	1	1	**0**	0	0	**3**			
Sonuç	H	H	H	H	✓	H	H	H	H	H			

FIFO · k = 3 · sayfa hatası: 9

İstek 10: sayfa **3** yok → **sayfa hatası**; en önce yüklenen **0** çıkarılır. Toplam hata: 9.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**		
Ç2		**0**	0	0	**0**	**3**	3	3	**2**	2	2		
Ç3			**1**	1	1	1	**0**	0	0	**3**	3		
Sonuç	H	H	H	H	✓	H	H	H	H	H	H		

FIFO · k = 3 · sayfa hatası: 10

İstek 11: sayfa **0** yok → **sayfa hatası**; en önce yüklenen **4** çıkarılır. Toplam hata: 10.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**	0	
Ç2		**0**	0	0	**0**	**3**	3	3	**2**	2	2	2	
Ç3			**1**	1	1	1	**0**	0	0	**3**	3	**3**	
Sonuç	H	H	H	H	✓	H	H	H	H	H	H	✓	

FIFO · k = 3 · sayfa hatası: 10

İstek 12: sayfa **3** önbellekte → **isabet**. FIFO sırası **değişmez**. Toplam hata: 10.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

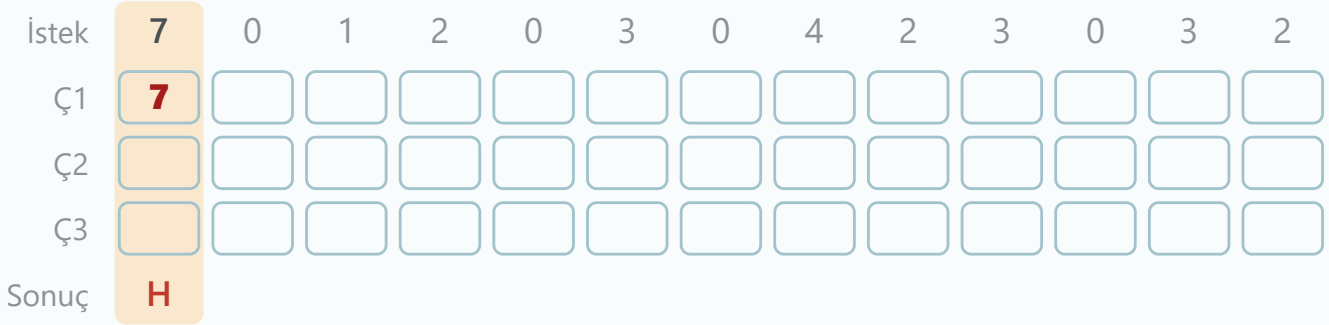
İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**	0	0
Ç2		**0**	0	0	**0**	**3**	3	3	**2**	2	2	2	**2**
Ç3			**1**	1	1	1	**0**	0	0	**3**	3	**3**	3
Sonuç	H	H	H	H	✓	H	H	H	H	H	H	✓	✓

FIFO · k = 3 · sayfa hatası: 10

İstek 13: sayfa **2** önbellekte → **isabet**. FIFO sırası **değişmez**. Toplam hata: 10.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

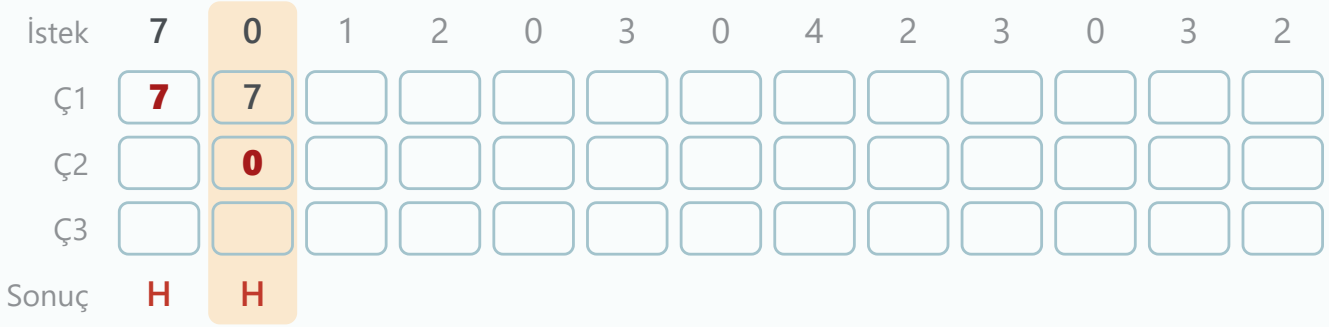


LRU · k = 3 · sayfa hatası: 1

İstek 1: sayfa **7** yok → **sayfa hatası**; boş çerçeve Ç1'e yüklenir. Toplam hata: 1.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

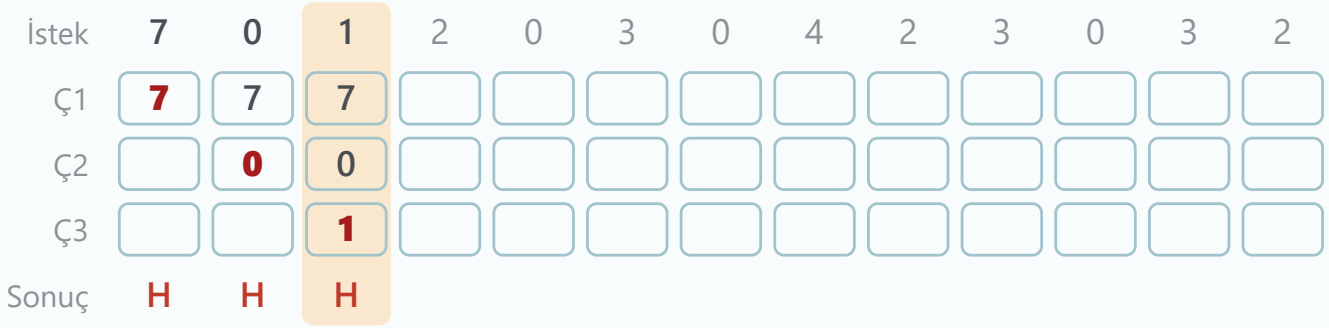


LRU · k = 3 · sayfa hatası: 2

İstek 2: sayfa **0** yok → **sayfa hatası**; boş çerçeve Ç2'ye yüklenir. Toplam hata: 2.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

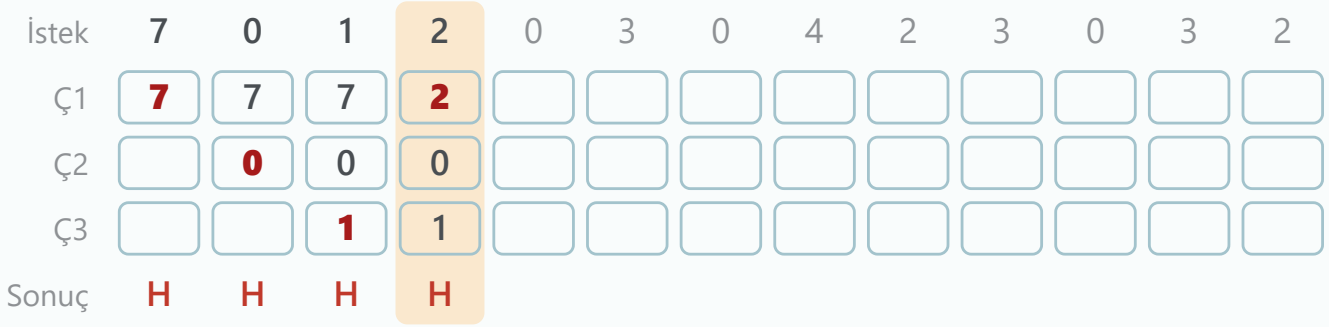


LRU · k = 3 · sayfa hatası: 3

İstek 3: sayfa **1** yok → **sayfa hatası**; boş çerçeve Ç3'e yüklenir. Toplam hata: 3.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.



LRU · k = 3 · sayfa hatası: 4

İstek 4: sayfa **2** yok → **sayfa hatası**; en uzun süredir kullanılmayan **7** çıkarılır. Toplam hata: 4.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

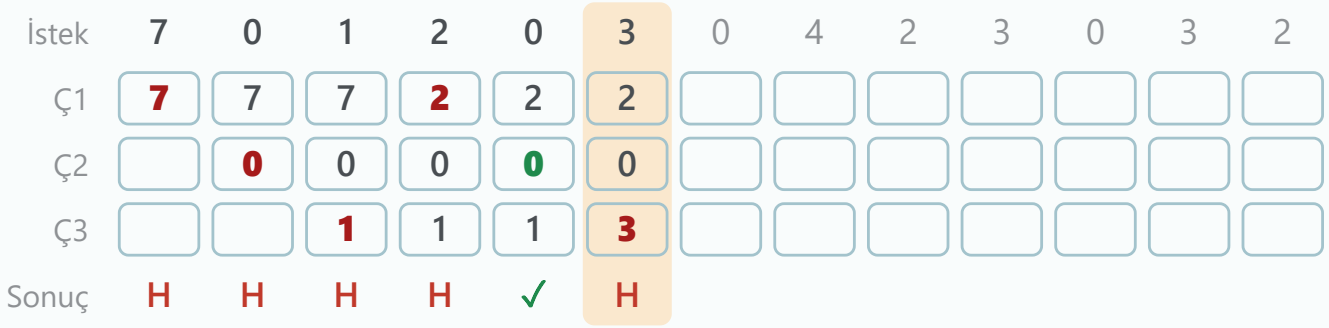
İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2								
Ç2		**0**	0	0	**0**								
Ç3			**1**	1	1								
Sonuç	H	H	H	H	✓								

LRU · k = 3 · sayfa hatası: 4

İstek 5: sayfa **0** önbellekte → **isabet**. 0 artık en yeni kullanılan sayfa. Toplam hata: 4.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.



LRU · k = 3 · sayfa hatası: 5

İstek 6: sayfa **3** yok → **sayfa hatası**; en uzun süredir kullanılmayan **1** çıkarılır. Toplam hata: 5.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2						
Ç2		**0**	0	0	**0**	0	**0**						
Ç3			**1**	1	1	**3**	3						
Sonuç	H	H	H	H	✓	H	✓						

LRU · k = 3 · sayfa hatası: 5

İstek 7: sayfa **0** önbellekte → **isabet**. 0 artık en yeni kullanılan sayfa. Toplam hata: 5.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	**4**					
Ç2		**0**	0	0	**0**	0	**0**	0					
Ç3			**1**	1	1	**3**	3	3					
Sonuç	H	H	H	H	✓	H	✓	H					

LRU · k = 3 · sayfa hatası: 6

İstek 8: sayfa **4** yok → **sayfa hatası**; en uzun süredir kullanılmayan **2** çıkarılır. Toplam hata: 6.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	**4**	4				
Ç2		**0**	0	0	**0**	0	**0**	0	0				
Ç3			**1**	1	1	**3**	3	3	**2**				
Sonuç	H	H	H	H	✓	H	✓	H	H				

LRU · k = 3 · sayfa hatası: 7

İstek 9: sayfa **2** yok → **sayfa hatası**; en uzun süredir kullanılmayan **3** çıkarılır. Toplam hata: 7.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	**4**	4	4			
Ç2		**0**	0	0	**0**	0	**0**	0	0	**3**			
Ç3			**1**	1	1	**3**	3	3	**2**	2			
Sonuç	H	H	H	H	✓	H	✓	H	H	H			

LRU · k = 3 · sayfa hatası: 8

İstek 10: sayfa **3** yok → **sayfa hatası**; en uzun süredir kullanılmayan **0** çıkarılır. Toplam hata: 8.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**		
Ç2		**0**	0	0	**0**	0	**0**	0	0	**3**	3		
Ç3			**1**	1	1	**3**	3	3	**2**	2	2		
Sonuç	H	H	H	H	✓	H	✓	H	H	H	H		

LRU · k = 3 · sayfa hatası: 9

İstek 11: sayfa **0** yok → **sayfa hatası**; en uzun süredir kullanılmayan **4** çıkarılır. Toplam hata: 9.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**	0	
Ç2		**0**	0	0	**0**	0	**0**	0	0	**3**	3	**3**	
Ç3			**1**	1	1	**3**	3	3	**2**	2	2	2	
Sonuç	H	H	H	H	✓	H	✓	H	H	H	H	✓	

LRU · k = 3 · sayfa hatası: 9

İstek 12: sayfa **3** önbellekte → **isabet**. 3 artık en yeni kullanılan sayfa. Toplam hata: 9.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

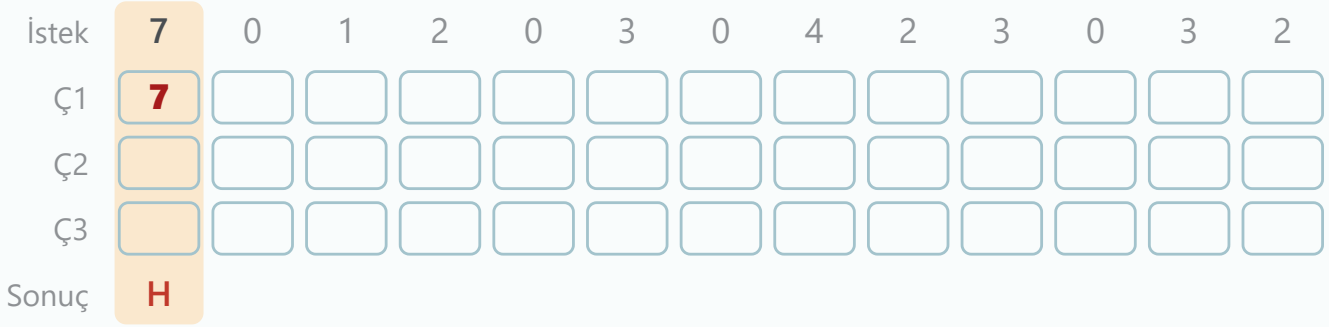
İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**	0	0
Ç2		**0**	0	0	**0**	0	**0**	0	0	**3**	3	**3**	3
Ç3			**1**	1	1	**3**	3	3	**2**	2	2	2	**2**
Sonuç	H	H	H	H	✓	H	✓	H	H	H	H	✓	✓

LRU · k = 3 · sayfa hatası: 9

İstek 13: sayfa **2** önbellekte → **isabet**. 2 artık en yeni kullanılan sayfa. Toplam hata: 9.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.



OPT · k = 3 · sayfa hatası: 1

İstek 1: sayfa **7** yok → **sayfa hatası**; boş çerçeve Ç1'e yüklenir. Toplam hata: 1.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7											
Ç2		0											
Ç3													
Sonuç	H	H											

OPT · k = 3 · sayfa hatası: 2

İstek 2: sayfa **0** yok → **sayfa hatası**; boş çerçeve Ç2'ye yüklenir. Toplam hata: 2.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7										
Ç2		**0**	0										
Ç3			**1**										
Sonuç	H	H	H										

OPT · k = 3 · sayfa hatası: 3

İstek 3: sayfa **1** yok → **sayfa hatası**; boş çerçeve Ç3'e yüklenir. Toplam hata: 3.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**									
Ç2		**0**	0	0									
Ç3			**1**	1									
Sonuç	H	H	H	H									

OPT · k = 3 · sayfa hatası: 4

İstek 4: sayfa **2** yok → **sayfa hatası**; **7** bir daha hiç istenmiyor → çıkarılır. Toplam hata: 4.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2								
Ç2		**0**	0	0	**0**								
Ç3			**1**	1	1								
Sonuç	H	H	H	H	✓								

OPT · k = 3 · sayfa hatası: 4

İstek 5: sayfa **0** önbellekte → **isabet**. Toplam hata: 4.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2							
Ç2		**0**	0	0	**0**	0							
Ç3			**1**	1	1	**3**							
Sonuç	H	H	H	H	✓	H							

OPT · k = 3 · sayfa hatası: 5

İstek 6: sayfa **3** yok → **sayfa hatası**; **1** bir daha hiç istenmiyor → çıkarılır. Toplam hata: 5.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2						
Ç2		**0**	0	0	**0**	0	**0**						
Ç3			**1**	1	1	**3**	3						
Sonuç	H	H	H	H	✓	H	✓						

OPT · k = 3 · sayfa hatası: 5

İstek 7: sayfa **0** önbellekte → **isabet**. Toplam hata: 5.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	2					
Ç2		**0**	0	0	**0**	0	**0**	**4**					
Ç3			**1**	1	1	**3**	3	3					
Sonuç	H	H	H	H	✓	H	✓	H					

OPT · k = 3 · sayfa hatası: 6

İstek 8: sayfa **4** yok → **sayfa hatası**; bir sonraki kullanımı en uzak olan **0** (11. istek) çıkarılır. Toplam hata: 6.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	2	**2**				
Ç2		**0**	0	0	**0**	0	**0**	**4**	4				
Ç3			**1**	1	1	**3**	3	3	3				
Sonuç	H	H	H	H	✓	H	✓	H	✓				

OPT · k = 3 · sayfa hatası: 6

İstek 9: sayfa **2** önbellekte → **isabet**. Toplam hata: 6.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	2	**2**	2			
Ç2		**0**	0	0	**0**	0	**0**	**4**	4	4			
Ç3			**1**	1	1	**3**	3	3	3	**3**			
Sonuç	H	H	H	H	✓	H	✓	H	✓	✓			

OPT · k = 3 · sayfa hatası: 6

İstek 10: sayfa **3** önbellekte → **isabet**. Toplam hata: 6.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	2	**2**	2	2		
Ç2		**0**	0	0	**0**	0	**0**	**4**	4	4	**0**		
Ç3			**1**	1	1	**3**	3	3	3	**3**	3		
Sonuç	H	H	H	H	✓	H	✓	H	✓	✓	H		

OPT · k = 3 · sayfa hatası: 7

İstek 11: sayfa **0** yok → **sayfa hatası**; **4** bir daha hiç istenmiyor → çıkarılır. Toplam hata: 7.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	2	**2**	2	2	2	
Ç2		**0**	0	0	**0**	0	**0**	**4**	4	4	**0**	0	
Ç3			**1**	1	1	**3**	3	3	3	**3**	3	**3**	
Sonuç	H	H	H	H	✓	H	✓	H	✓	✓	H	✓	

OPT · k = 3 · sayfa hatası: 7

İstek 12: sayfa **3** önbellekte → **isabet**. Toplam hata: 7.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. **H**: hata, ✓: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	**7**	7	7	**2**	2	2	2	2	**2**	2	2	2	**2**
Ç2		**0**	0	0	**0**	0	**0**	**4**	4	4	**0**	0	0
Ç3			**1**	1	1	**3**	3	3	3	**3**	3	**3**	3
Sonuç	H	H	H	H	✓	H	✓	H	✓	✓	H	✓	✓

OPT · k = 3 · sayfa hatası: 7

İstek 13: sayfa **2** önbellekte → **isabet**. Toplam hata: 7.

Üç Politikanın Karşılaştırması

Politika	Sayfa hatası	İsabet	ALG / OPT	Gerekli bilgi
FIFO	10	3	1,43	Yükleme sırası (kuyruk)
LRU	9	4	1,29	Son kullanım zamanı
OPT	7	6	1	Tüm gelecek istekler

- İlk k = 3 hata **kaçınılmazdır** (zorunlu / soğuk hatalar, compulsory misses): her politika öder.
- LRU, **yerellik** (locality of reference) olan dizilerde iyidir: yakın geçmişte kullanılan sayfa yakında yine kullanılır. Burada 0 sayfası sık istendiği için FIFO onu gereksiz yere çıkarır.
- Gerçek sistemlerde LRU'ya yakın, ucuz yaklaşımlar kullanılır: **saat algoritması** (clock / ikinci şans).

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
▶    var onb = new LinkedList<Integer>();
▶    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```

istek: 1 2 3 1 4 2

onb: [ ] [ ] [ ]

en eski — en yeni

Önbellek boş, `hata = 0`. Liste başı en eski, sonu en yeni kullanılan sayfa.

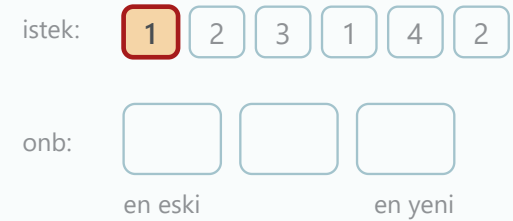
Adım 1 / 19

DEĞİŞKENLER

`k = 3` `sayfa = –` `hata = 0` `onb = []`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
▶    for (int sayfa : istekler) {
▶      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



Sayfa 1 listede yok: `remove` false → else dalı.

Adım 2 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 1` `hata = 0` `onb = []`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
▶        hata++;
▶        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



`hata = 1`. Önbellekte yer var (boyut 0 < 3) → çıkarma yok.

Adım 3 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 1` `hata = 1` `onb = []`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
▶        onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



Sayfa 1 sona eklenir.

Adım 4 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 1` `hata = 1`

`onb = [1]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
▶    for (int sayfa : istekler) {
▶      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



Sayfa 2 listede yok: `remove` false → else dalı.

Adım 5 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 2` `hata = 1` `onb = [1]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
▶        hata++;
▶        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



`hata = 2`. Önbellekte yer var (boyut 1 < 3) → çıkarma yok.

Adım 6 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 2` `hata = 2` `onb = [1]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
▶        onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



Sayfa 2 sona eklenir.

Adım 7 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 2` `hata = 2`

`onb = [1, 2]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
▶    for (int sayfa : istekler) {
▶      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



Sayfa 3 listede yok: `remove` false → else dalı.

Adım 8 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 3` `hata = 2` `onb = [1, 2]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
▶        hata++;
▶        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



`hata = 3`. Önbellekte yer var (boyut 2 < 3) → çıkarma yok.

Adım 9 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 3` `hata = 3` `onb = [1, 2]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
▶       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



Sayfa 3 sona eklenir.

Adım 10 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 3` `hata = 3`

`onb = [1, 2, 3]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
▶    for (int sayfa : istekler) {
▶      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



Sayfa 1 listede: `remove` true döner ve onu listeden çıkarır → **isabet**.

Adım 11 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 1` `hata = 3` `onb = [2, 3]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
▶        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



Sayfa 1 sona eklenir: artık **en yeni** kullanılan.

Adım 12 / 19

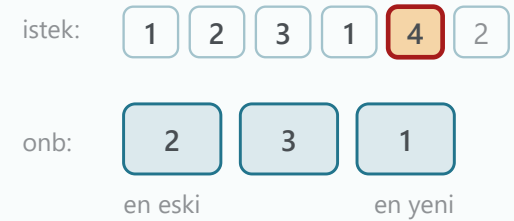
DEĞİŞKENLER

`k = 3` `sayfa = 1` `hata = 3`

`onb = [2, 3, 1]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
▶    for (int sayfa : istekler) {
▶      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



Sayfa 4 listede yok: `remove` false → else dalı.

Adım 13 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 4` `hata = 3` `onb = [2, 3, 1]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
▶        hata++;
▶        if (onb.size() == k)
▶          onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



`hata = 4`. Önbellek dolu → baştaki (en eski kullanılan) **2** çıkarılır.

Adım 14 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 4` `hata = 4` `onb = [3, 1]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
▶        onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



Sayfa 4 sona eklenir.

Adım 15 / 19

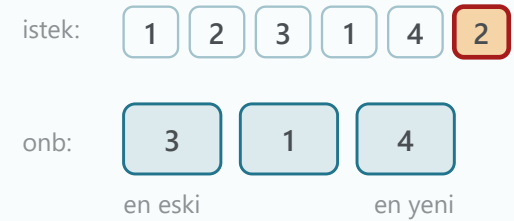
DEĞİŞKENLER

`k = 3` `sayfa = 4` `hata = 4`

`onb = [3, 1, 4]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
▶    for (int sayfa : istekler) {
▶      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



Sayfa 2 listede yok: `remove` false → else dalı.

Adım 16 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 2` `hata = 4` `onb = [3, 1, 4]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
▶        hata++;
▶        if (onb.size() == k)
▶          onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



`hata = 5`. Önbellek dolu → baştaki (en eski kullanılan) **3** çıkarılır.

Adım 17 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 2` `hata = 5` `onb = [1, 4]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
▶        onb.addLast(sayfa);
12     }
13   }
14   return hata;
15 }
```



Sayfa 2 sona eklenir.

Adım 18 / 19

DEĞİŞKENLER

`k = 3` `sayfa = 2` `hata = 5`

`onb = [1, 4, 2]`

LRU: Kodu Adım Adım İzle

```
1  int lruHata(int[] istekler, int k) {
2    var onb = new LinkedList<Integer>();
3    int hata = 0;
4    for (int sayfa : istekler) {
5      if (onb.remove((Integer) sayfa)) {
6        onb.addLast(sayfa); // en yeni yap
7      } else {
8        hata++;
9        if (onb.size() == k)
10         onb.removeFirst(); // en eskiyi at
11       onb.addLast(sayfa);
12     }
13   }
▶    return hata;
15 }
```

istek: 1 2 3 1 4 2

onb: 1 4 2

en eski — en yeni

Dizi bitti: 6 istekte **5 sayfa hatası**, 1 isabet. `remove` listeyi taradığı için her istek O(k); HashMap + çift bağlı liste ile O(1) olur.

Adım 19 / 19

DEĞİŞKENLER

`k = 3` `sayfa = –` `hata = 5` `onb = [1, 4, 2]`

SAYFALAMA

LRU Önbelleği: O(1) Uygulama

```
class LruOnbellek<A, D>
    extends LinkedHashMap<A, D> {
  private final int k;

  LruOnbellek(int k) {
    super(16, 0.75f, true); // erişim sırası
    this.k = k;
  }

  @Override
  protected boolean removeEldestEntry(
      Map.Entry<A, D> enEski) {
    return size() > k;      // k aşılınca at
  }
}
```

- `LinkedHashMap` = **hash tablosu + çift bağlı liste**.
- `true` (accessOrder): her `get` / `put` öğeyi listenin **sonuna** taşır.
- `removeEldestEntry` : boyut k'yi aşınca **baştaki** (en eski kullanılan) silinir.
- Bul, taşı, sil: hepsi **O(1)**; bellek O(k).

FIFO için aynı sınıf `false` (insertion order) ile kurulur: erişim sırayı değiştirmez.

Teorem: Deterministik Alt Sınır k

- k + 1 farklı sayfa yeter. Düşman her adımda ALG'nin önbelleğinde **olmayan** tek sayfayı ister → ALG **her istekte** hata yapar.
- OPT (en uzağı çıkar) bir hata yaptığında, çıkardığı sayfa en az k − 1 istek boyunca istenmez → OPT en çok **k istekte bir** hata yapar.
- Sonuç: ALG / OPT → **k** (Sleator ve Tarjan, 1985).

Örnekte (k = 3, LRU'ya karşı): LRU 12 hata, OPT 6 hata; ilk 3 zorunlu hata dışarıda tutulursa 9'a 3.

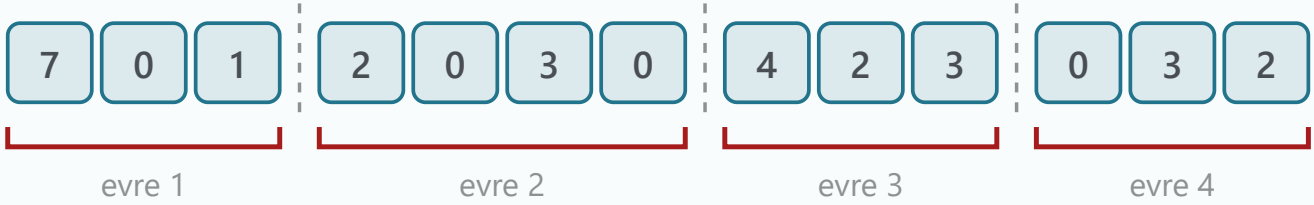
İstek	1	2	3	4	1	2	3	4	1	2	3	4
Ç1	1	1	1	4	4	4	3	3	3	2	2	2
Ç2		2	2	2	1	1	1	4	4	4	3	3
Ç3			3	3	3	2	2	2	1	1	1	4
Sonuç	H	H	H	H	H	H	H	H	H	H	H	H

LRU · k = 3 · sayfa hatası: 12

İstek	1	2	3	4	1	2	3	4	1	2	3	4
Ç1	1	1	1	1	1	1	1	1	1	2	2	2
Ç2		2	2	2	2	2	3	3	3	3	3	3
Ç3			3	4	4	4	4	4	4	4	4	4
Sonuç	H	H	H	H	✓	✓	H	✓	✓	H	✓	✓

OPT · k = 3 · sayfa hatası: 6

Teorem: LRU (ve FIFO) k-Rekabetçidir



- Diziyi **evrelere** (phase) böl: her evre, **k farklı sayfa** içeren en uzun parça; (k + 1). farklı sayfa yeni evreyi başlatır.
- **LRU, bir evrede en çok k hata yapar:** evrede istenen bir sayfa, evre bitene kadar çıkarılmaz (ondan daha eski kullanılmış en az bir sayfa hep vardır). Her farklı sayfa en çok bir kez hata verir.
- **OPT, ardışık iki evre başına en az 1 hata yapar:** bir evrenin 2. isteğinden sonraki evrenin 1. isteğine kadar k + 1 farklı sayfa istenir; k çerçeveye sığmaz.
- m evre için LRU ≤ k · m, OPT ≥ m − 1 → **LRU ≤ k · OPT + k**.

Örnek dizide 4 evre: LRU = 9 ≤ 3 · 4, OPT = 7 ≥ 3. Teori en kötü durumu sınırlar; gerçek dizilerde yerellik sayesinde LRU, OPT'ye çok daha yakındır.

Belady Anomalisi: Daha Çok Bellek, Daha Çok Hata

İstek	1	2	3	4	1	2	5	1	2	3	4	5
Ç1	**1**	1	1	**4**	4	4	**5**	5	5	5	5	**5**
Ç2		**2**	2	2	**1**	1	1	**1**	1	**3**	3	3
Ç3			**3**	3	3	**2**	2	2	**2**	2	**4**	4
Sonuç	H	H	H	H	H	H	H	✓	✓	H	H	✓

FIFO · k = 3 · sayfa hatası: 9

İstek	1	2	3	4	1	2	5	1	2	3	4	5
Ç1	**1**	1	1	1	**1**	1	**5**	5	5	5	**4**	4
Ç2		**2**	2	2	2	**2**	2	**1**	1	1	1	**5**
Ç3			**3**	3	3	3	3	3	**2**	2	2	2
Ç4				**4**	4	4	4	4	4	**3**	3	3
Sonuç	H	H	H	H	✓	✓	H	H	H	H	H	H

FIFO · k = 4 · sayfa hatası: 10

- Dizi: 1 2 3 4 1 2 5 1 2 3 4 5.
- FIFO, k = 3: **9 hata**; k = 4: **10 hata**. Önbellek büyüdü, hata arttı!
- LRU aynı dizide: k = 3 → 10, k = 4 → 8. LRU ve OPT bir **yığın algoritmasıdır** (stack algorithm): k çerçevedeki sayfalar, k + 1 çerçevedekilerin alt kümesidir → anomali olmaz.

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
▶    Queue<Integer> kuyruk = new LinkedList<>();
▶    Set<Integer> kume = new HashSet<>();
▶    int hata = 0;
5    for (int p : istekler) {
6      if (kume.contains(p)) continue;   // isabet
7      hata++;
8      if (kume.size() == k)
9        kume.remove(kuyruk.poll()); // en eski
10     kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



kuyruk

hata = 0

en eski

Belady dizisi 1 2 3 4 1 2 5 1 2 3 4 5, k = 3. Kuyruk yükleme sırasını, küme hızlı "var mı?" sorgusunu tutar.

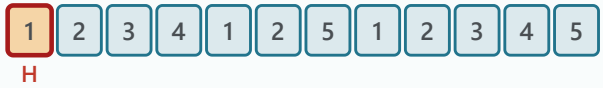
Adım 1 / 14

DEĞİŞKENLER

k = 3 hata = 0

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
6      if (kume.contains(p)) continue;   // isabet
       hata++;
       if (kume.size() == k)
9        kume.remove(kuyruk.poll()); // en eski
       kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



kuyruk [1] (en eski)

hata = 1

p = 1 yok → hata 1 (zorunlu hata); yer var, kuyruğa eklenir.

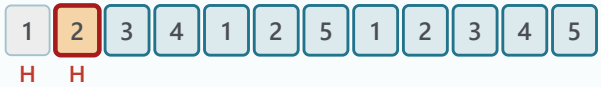
Adım 2 / 14

DEĞİŞKENLER

`p = 1` `hata = 1` `kuyruk = [1]`

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
6      if (kume.contains(p)) continue;   // isabet
▶      hata++;
▶      if (kume.size() == k)
9        kume.remove(kuyruk.poll()); // en eski
▶      kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



kuyruk



hata = 2

p = 2 yok → hata 2 (zorunlu hata); yer var, kuyruğa eklenir.

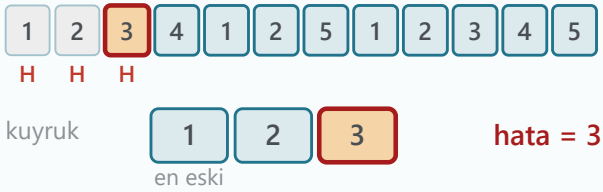
Adım 3 / 14

DEĞİŞKENLER

`p = 2` `hata = 2` `kuyruk = [1, 2]`

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
6      if (kume.contains(p)) continue;   // isabet
▶      hata++;
▶      if (kume.size() == k)
9        kume.remove(kuyruk.poll()); // en eski
▶      kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



p = 3 yok → hata 3 (zorunlu hata); yer var, kuyruğa eklenir.

Adım 4 / 14

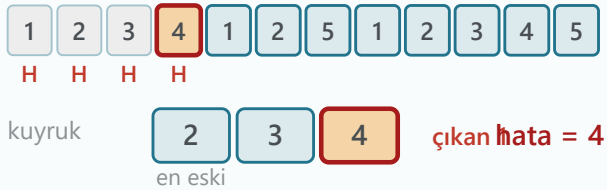
DEĞİŞKENLER

`p = 3` `hata = 3`

`kuyruk = [1, 2, 3]`

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
6      if (kume.contains(p)) continue;   // isabet
▶      hata++;
▶      if (kume.size() == k)
▶        kume.remove(kuyruk.poll()); // en eski
▶      kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



p = 4 yok → hata 4. Önbellek dolu: en önce yüklenen **1** çıkar, 4 kuyruğun sonuna girer.

Adım 5 / 14

DEĞİŞKENLER

`p = 4` `hata = 4`

`kuyruk = [2, 3, 4]`

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
6      if (kume.contains(p)) continue;   // isabet
▶      hata++;
▶      if (kume.size() == k)
▶        kume.remove(kuyruk.poll()); // en eski
▶      kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



p = 1 yok → hata 5. Önbellek dolu: en önce yüklenen **2** çıkar, 1 kuyruğun sonuna girer.

Adım 6 / 14

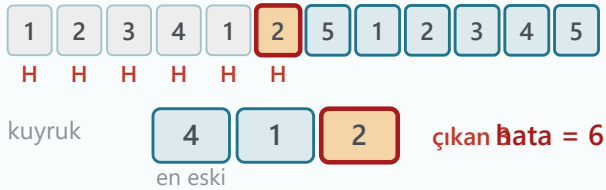
DEĞİŞKENLER

`p = 1` `hata = 5`

`kuyruk = [3, 4, 1]`

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
6      if (kume.contains(p)) continue;   // isabet
▶      hata++;
▶      if (kume.size() == k)
▶        kume.remove(kuyruk.poll()); // en eski
▶      kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



p = 2 yok → hata 6. Önbellek dolu: en önce yüklenen **3** çıkar, 2 kuyruğun sonuna girer.

Adım 7 / 14

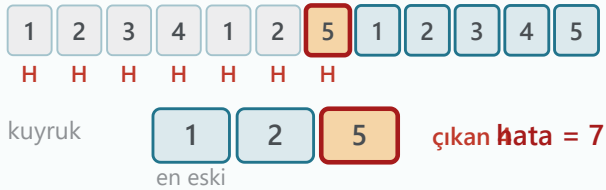
DEĞİŞKENLER

`p = 2` `hata = 6`

`kuyruk = [4, 1, 2]`

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
6      if (kume.contains(p)) continue;   // isabet
▶      hata++;
▶      if (kume.size() == k)
▶        kume.remove(kuyruk.poll()); // en eski
▶      kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



p = 5 yok → hata 7. Önbellek dolu: en önce yüklenen **4** çıkar, 5 kuyruğun sonuna girer.

Adım 8 / 14

DEĞİŞKENLER

`p = 5` `hata = 7`

`kuyruk = [1, 2, 5]`

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
▶      if (kume.contains(p)) continue;   // isabet
7      hata++;
8      if (kume.size() == k)
9        kume.remove(kuyruk.poll()); // en eski
10     kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



p = 1 önbellekte → **isabet**. FIFO'da isabet kuyruk sırasını değiştirmez.

Adım 9 / 14

DEĞİŞKENLER

`p = 1` `hata = 7`

`kuyruk = [1, 2, 5]`

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
▶      if (kume.contains(p)) continue;   // isabet
7      hata++;
8      if (kume.size() == k)
9        kume.remove(kuyruk.poll()); // en eski
10     kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



p = 2 önbellekte → **isabet**. FIFO'da isabet kuyruk sırasını değiştirmez.

Adım 10 / 14

DEĞİŞKENLER

`p = 2` `hata = 7`

`kuyruk = [1, 2, 5]`

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
6      if (kume.contains(p)) continue;   // isabet
▶      hata++;
▶      if (kume.size() == k)
▶        kume.remove(kuyruk.poll()); // en eski
▶      kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



p = 3 yok → hata 8. Önbellek dolu: en önce yüklenen **1** çıkar, 3 kuyruğun sonuna girer.

Adım 11 / 14

DEĞİŞKENLER

`p = 3` `hata = 8`

`kuyruk = [2, 5, 3]`

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
6      if (kume.contains(p)) continue;   // isabet
▶      hata++;
▶      if (kume.size() == k)
▶        kume.remove(kuyruk.poll()); // en eski
▶      kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



p = 4 yok → hata 9. Önbellek dolu: en önce yüklenen **2** çıkar, 4 kuyruğun sonuna girer.

Adım 12 / 14

DEĞİŞKENLER

`p = 4` `hata = 9`

`kuyruk = [5, 3, 4]`

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
▶      if (kume.contains(p)) continue;   // isabet
7      hata++;
8      if (kume.size() == k)
9        kume.remove(kuyruk.poll()); // en eski
10     kuyruk.offer(p); kume.add(p);
11   }
12   return hata;
13 }
```



p = 5 önbellekte → **isabet**. FIFO'da isabet kuyruk sırasını değiştirmez.

Adım 13 / 14

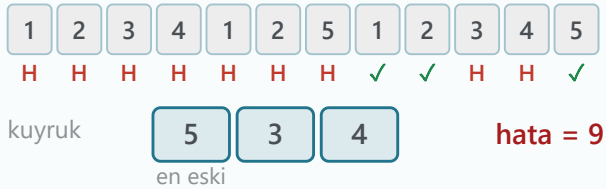
DEĞİŞKENLER

`p = 5` `hata = 9`

`kuyruk = [5, 3, 4]`

Belady Dizisinde FIFO: Kodu Adım Adım İzle

```
1  int fifoHata(int[] istekler, int k) {
2    Queue<Integer> kuyruk = new LinkedList<>();
3    Set<Integer> kume = new HashSet<>();
4    int hata = 0;
5    for (int p : istekler) {
6      if (kume.contains(p)) continue;   // isabet
7      hata++;
8      if (kume.size() == k)
9        kume.remove(kuyruk.poll()); // en eski
10     kuyruk.offer(p); kume.add(p);
11   }
▶    return hata;
13 }
```



`return 9`. Aynı kod k = 4 ile 10 hata verir: çerçeve artınca hata da arttı (Belady anomalisi).

Adım 14 / 14

DEĞİŞKENLER

`hata = 9` `dönüş = 9`

Rastgele Sayfalama: İşaretleme Algoritması

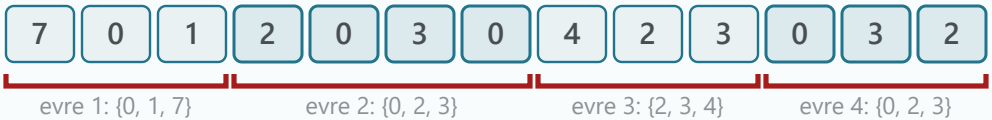
İşaretleme (Marking)

1. Evre başında tüm sayfaların işareti kaldırılır.
2. İstenen sayfa **işaretlenir** (isabet ya da hata).
3. Hata olursa **işaretsiz** sayfalardan biri **rastgele** çıkarılır.
4. Tüm sayfalar işaretliyse yeni evre başlar (adım 1).

Sonuçlar

- Habersiz düşmana karşı beklenen oran $\mathbf{2H_k}$ (Fiat vd., 1991).
- $H_k = 1 + 1/2 + \ldots + 1/k \approx \ln k$. k = 8 için $H_8 \approx 2{,}72$ → oran ≈ 5,4 (deterministikte 8).
- Hiçbir rastgele algoritma $\mathbf{H_k}$'den iyi olamaz.
- LRU da bir işaretleme algoritmasıdır (en eski işaretsizi seçer).

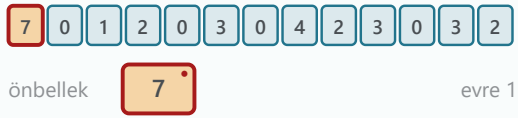
k = 3: her evre en çok k farklı sayfa içerir; (k + 1). farklı sayfa yeni evreyi başlatır



Rastgelelik, düşmanın "ALG'nin önbelleğinde olmayan sayfayı iste" stratejisini bozar: hangi sayfanın çıktığını bilemez.

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
▶      if (!onb.contains(p)) {
▶        hata++;
▶        if (onb.size() == k) {
9          if (isr.size() == k) isr.clear(); // yeni evre
10         List<Integer> aday = new ArrayList<>(onb);
11         aday.removeAll(isr);          // işaretsizler
12         int i = zar.nextInt(aday.size());
13         onb.remove(aday.get(i));
14       }
▶        onb.add(p);
16     }
▶      isr.add(p);
18   }
19   return hata;
20 }
```



hata = 1

İstekler 7 0 1 2 0 3 0 4 2 3 0 3 2, k = 3. p = 7 yok → hata 1; önbelleğe eklenir ve işaretlenir (•).

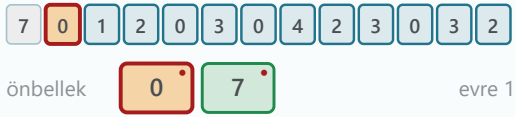
Adım 1 / 14

DEĞİŞKENLER

`p = 7` `hata = 1` `onb = [7]` `isr = [7]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
▶      if (!onb.contains(p)) {
▶        hata++;
▶        if (onb.size() == k) {
9          if (isr.size() == k) isr.clear(); // yeni evre
10         List<Integer> aday = new ArrayList<>(onb);
11         aday.removeAll(isr);          // işaretsizler
12         int i = zar.nextInt(aday.size());
13         onb.remove(aday.get(i));
14       }
▶        onb.add(p);
16     }
▶      isr.add(p);
18   }
19   return hata;
20 }
```



hata = 2

p = 0 yok → hata 2 (zorunlu); yer var, eklenip işaretlenir.

Adım 2 / 14

DEĞİŞKENLER

`p = 0` `hata = 2`

`onb = [0, 7]` `isr = [0, 7]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
▶      if (!onb.contains(p)) {
▶        hata++;
▶        if (onb.size() == k) {
9          if (isr.size() == k) isr.clear(); // yeni evre
10         List<Integer> aday = new ArrayList<>(onb);
11         aday.removeAll(isr);          // işaretsizler
12         int i = zar.nextInt(aday.size());
13         onb.remove(aday.get(i));
14       }
▶        onb.add(p);
16     }
▶      isr.add(p);
18   }
19   return hata;
20 }
```



hata = 3

p = 1 yok → hata 3 (zorunlu); yer var, eklenip işaretlenir.

Adım 3 / 14

DEĞİŞKENLER

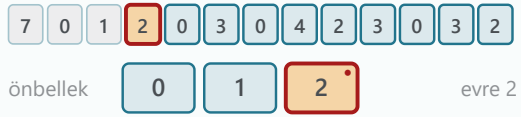
`p = 1` `hata = 3`

`onb = [0, 1, 7]`

`isr = [0, 1, 7]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
       if (!onb.contains(p)) {
         hata++;
         if (onb.size() == k) {
           if (isr.size() == k) isr.clear(); // yeni evre
           List<Integer> aday = new ArrayList<>(onb);
           aday.removeAll(isr);          // işaretsizler
           int i = zar.nextInt(aday.size());
           onb.remove(aday.get(i));
14       }
         onb.add(p);
16     }
       isr.add(p);
18   }
19   return hata;
20 }
```



adaylar [0, 1, 7] → çıkan 7

hata = 4

Tüm sayfalar işaretliydi → **yeni evre**: işaretler silinir. p = 2 yok → hata 4. İşaretsiz adaylar [0, 1, 7] arasından rastgele **7** çıkar.

Adım 4 / 14

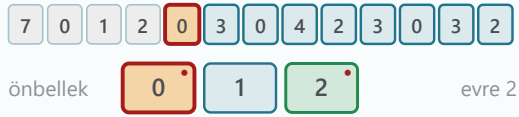
DEĞİŞKENLER

`p = 2` `hata = 4`

`onb = [0, 1, 2]` `isr = [2]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
▶      if (!onb.contains(p)) {
7        hata++;
8        if (onb.size() == k) {
9          if (isr.size() == k) isr.clear(); // yeni evre
10         List<Integer> aday = new ArrayList<>(onb);
11         aday.removeAll(isr);          // işaretsizler
12         int i = zar.nextInt(aday.size());
13         onb.remove(aday.get(i));
14       }
15       onb.add(p);
16     }
▶      isr.add(p);
18   }
19   return hata;
20 }
```



hata = 4

p = 0 önbellekte → **isabet**; yalnızca işaretlenir (•).

Adım 5 / 14

DEĞİŞKENLER

`p = 0` `hata = 4`

`onb = [0, 1, 2]`

`isr = [0, 2]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
       if (!onb.contains(p)) {
         hata++;
         if (onb.size() == k) {
           if (isr.size() == k) isr.clear(); // yeni evre
           List<Integer> aday = new ArrayList<>(onb);
           aday.removeAll(isr);          // işaretsizler
           int i = zar.nextInt(aday.size());
           onb.remove(aday.get(i));
14       }
         onb.add(p);
16     }
       isr.add(p);
18   }
19   return hata;
20 }
```



adaylar [1] → çıkan 1 — hata = 5

p = 3 yok → hata 5. İşaretsiz adaylar [1] arasından rastgele **1** çıkar.

Adım 6 / 14

DEĞİŞKENLER

`p = 3` `hata = 5`

`onb = [0, 2, 3]`

`isr = [0, 2, 3]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
▶      if (!onb.contains(p)) {
7        hata++;
8        if (onb.size() == k) {
9          if (isr.size() == k) isr.clear(); // yeni evre
10         List<Integer> aday = new ArrayList<>(onb);
11         aday.removeAll(isr);          // işaretsizler
12         int i = zar.nextInt(aday.size());
13         onb.remove(aday.get(i));
14       }
15       onb.add(p);
16     }
▶      isr.add(p);
18   }
19   return hata;
20 }
```



hata = 5

p = 0 önbellekte → **isabet**; yalnızca işaretlenir (•).

Adım 7 / 14

DEĞİŞKENLER

`p = 0` `hata = 5`

`onb = [0, 2, 3]`

`isr = [0, 2, 3]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
       if (!onb.contains(p)) {
         hata++;
         if (onb.size() == k) {
           if (isr.size() == k) isr.clear(); // yeni evre
           List<Integer> aday = new ArrayList<>(onb);
           aday.removeAll(isr);          // işaretsizler
           int i = zar.nextInt(aday.size());
           onb.remove(aday.get(i));
14       }
         onb.add(p);
16     }
       isr.add(p);
18   }
19   return hata;
20 }
```



adaylar [0, 2, 3] → çıkan 3 — hata = 6

Tüm sayfalar işaretliydi → **yeni evre**: işaretler silinir. p = 4 yok → hata 6. İşaretsiz adaylar [0, 2, 3] arasından rastgele **3** çıkar.

Adım 8 / 14

DEĞİŞKENLER

`p = 4` `hata = 6`

`onb = [0, 2, 4]` `isr = [4]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
▶      if (!onb.contains(p)) {
7        hata++;
8        if (onb.size() == k) {
9          if (isr.size() == k) isr.clear(); // yeni evre
10         List<Integer> aday = new ArrayList<>(onb);
11         aday.removeAll(isr);          // işaretsizler
12         int i = zar.nextInt(aday.size());
13         onb.remove(aday.get(i));
14       }
15       onb.add(p);
16     }
▶      isr.add(p);
18   }
19   return hata;
20 }
```



hata = 6

p = 2 önbellekte → **isabet**; yalnızca işaretlenir (•).

Adım 9 / 14

DEĞİŞKENLER

`p = 2` `hata = 6`

`onb = [0, 2, 4]`

`isr = [2, 4]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
       if (!onb.contains(p)) {
         hata++;
         if (onb.size() == k) {
           if (isr.size() == k) isr.clear(); // yeni evre
           List<Integer> aday = new ArrayList<>(onb);
           aday.removeAll(isr);          // işaretsizler
           int i = zar.nextInt(aday.size());
           onb.remove(aday.get(i));
14       }
         onb.add(p);
16     }
       isr.add(p);
18   }
19   return hata;
20 }
```



adaylar [0] → çıkan 0 — hata = 7

p = 3 yok → hata 7. İşaretsiz adaylar [0] arasından rastgele **0** çıkar.

Adım 10 / 14

DEĞİŞKENLER

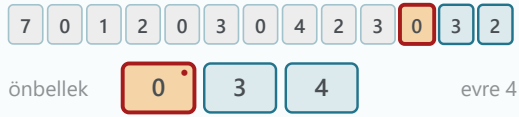
`p = 3` `hata = 7`

`onb = [2, 3, 4]`

`isr = [2, 3, 4]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
       if (!onb.contains(p)) {
         hata++;
         if (onb.size() == k) {
           if (isr.size() == k) isr.clear(); // yeni evre
           List<Integer> aday = new ArrayList<>(onb);
           aday.removeAll(isr);          // işaretsizler
           int i = zar.nextInt(aday.size());
           onb.remove(aday.get(i));
14       }
         onb.add(p);
16     }
       isr.add(p);
18   }
19   return hata;
20 }
```



adaylar [2, 3, 4] → çıkan 2 — hata = 8

Tüm sayfalar işaretliydi → **yeni evre**: işaretler silinir. p = 0 yok → hata 8. İşaretsiz adaylar [2, 3, 4] arasından rastgele **2** çıkar.

Adım 11 / 14

DEĞİŞKENLER

`p = 0` `hata = 8`

`onb = [0, 3, 4]` `isr = [0]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
▶      if (!onb.contains(p)) {
7        hata++;
8        if (onb.size() == k) {
9          if (isr.size() == k) isr.clear(); // yeni evre
10         List<Integer> aday = new ArrayList<>(onb);
11         aday.removeAll(isr);          // işaretsizler
12         int i = zar.nextInt(aday.size());
13         onb.remove(aday.get(i));
14       }
15       onb.add(p);
16     }
▶      isr.add(p);
18   }
19   return hata;
20 }
```



hata = 8

p = 3 önbellekte → **isabet**; yalnızca işaretlenir (•).

Adım 12 / 14

DEĞİŞKENLER

`p = 3` `hata = 8`

`onb = [0, 3, 4]`

`isr = [0, 3]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
       if (!onb.contains(p)) {
         hata++;
         if (onb.size() == k) {
           if (isr.size() == k) isr.clear(); // yeni evre
           List<Integer> aday = new ArrayList<>(onb);
           aday.removeAll(isr);          // işaretsizler
           int i = zar.nextInt(aday.size());
           onb.remove(aday.get(i));
14       }
         onb.add(p);
16     }
       isr.add(p);
18   }
19   return hata;
20 }
```



adaylar [4] → çıkan 4 — hata = 9

p = 2 yok → hata 9. İşaretsiz adaylar [4] arasından rastgele **4** çıkar.

Adım 13 / 14

DEĞİŞKENLER

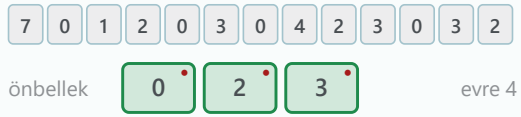
`p = 2` `hata = 9`

`onb = [0, 2, 3]`

`isr = [0, 2, 3]`

İşaretleme Algoritması: Kodu Adım Adım İzle

```
1  int isaretleme(int[] istekler, int k) {
2    Set<Integer> onb = new HashSet<>();
3    Set<Integer> isr = new HashSet<>();   // işaretli
4    int hata = 0;
5    for (int p : istekler) {
6      if (!onb.contains(p)) {
7        hata++;
8        if (onb.size() == k) {
9          if (isr.size() == k) isr.clear(); // yeni evre
10         List<Integer> aday = new ArrayList<>(onb);
11         aday.removeAll(isr);          // işaretsizler
12         int i = zar.nextInt(aday.size());
13         onb.remove(aday.get(i));
14       }
15       onb.add(p);
16     }
17     isr.add(p);
18   }
▶    return hata;
20 }
```



hata = 9

`return 9`. Bu çekilişlerle işaretleme 9 hata yaptı (LRU 9, FIFO 10, OPT 7); başka tohumla sonuç değişebilir.

Adım 14 / 14

DEĞİŞKENLER

`hata = 9` `dönüş = 9`

Deneyin: Sayfalama Politikaları

istekler `7 0 1 2 0 3 0 4 2 3 0 3 2` k 3 **FIFO** **LRU** **OPT** **Belady örneği**

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4	4	4	0	0	0
Ç2		0	0	0	0	0	0	0	0	3	3	3	3
Ç3			1	1	1	3	3	3	2	2	2	2	2
Sonuç	H	H	H	H	✓	H	✓	H	H	H	H	✓	✓

LRU gösteriliyor · k = 3 · sayfa hatası: FIFO **10**, LRU **9**, OPT **7**

0–9 arası rakamlar, boşlukla ayrılmış (en çok 16 istek). Turuncu hücre: yeni yüklenen sayfa · yeşil: isabet. Enter = yeniden çalıştır.

LİSTE GÜNCELLEME

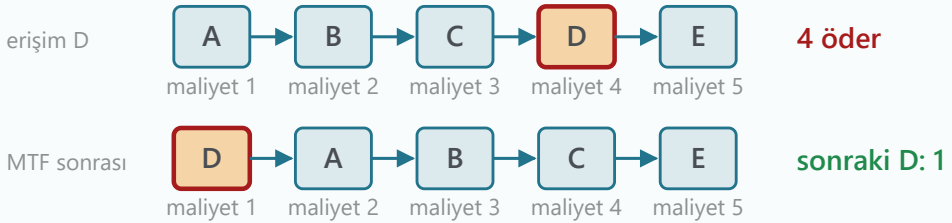
Liste Güncelleme Problemi (List Update)

- n öğeli bir **bağlı liste**; i. konumdaki öğeye erişmenin maliyeti **i**'dir (baştan taranır).
- Erişilen öğe, erişimden sonra **ücretsiz** olarak öne doğru taşınabilir.
- İstekler çevrim içi gelir: hangi öğenin sık isteneceği bilinmez.

Öne taşı (MTF)
Move-to-front: erişilen öğe listenin **başına** alınır.

Yer değiştir
Transpose: erişilen öğe **bir önceki** ile yer değiştirir.

Sıklık sayacı
Frequency count: öğeler erişim sayısına göre **azalan** sırada tutulur.

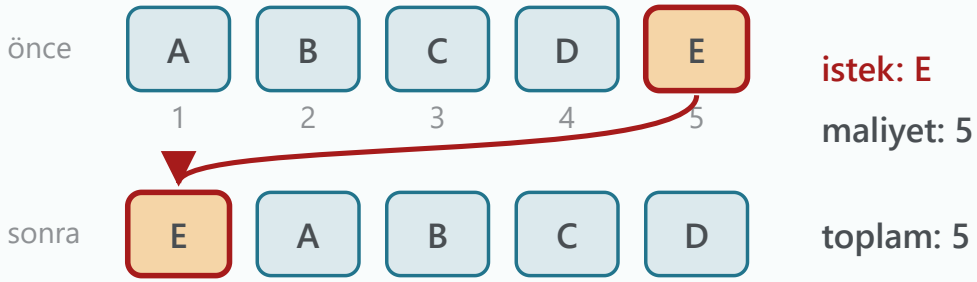


Uygulamalar: kendini düzenleyen listeler (self-organizing lists), hash tablosu kovalarındaki zincirler, **MTF dönüşümü** ile veri sıkıştırma (bzip2).

LİSTE GÜNCELLEME · ADIM ADIM

Öne Taşı (MTF) Adım Adım

Başlangıç listesi A B C D E; istekler **E D E A D E**.



İstek **E**: 5. konumda → maliyet 5; **başa taşınır**. Toplam: 5.

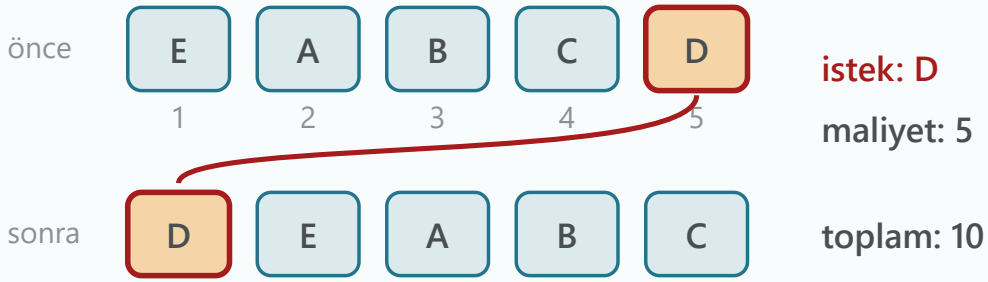
Adım 1 / 6

Strateji	E	D	E	A	D	E	Toplam
Öne taşı (MTF)	5	5	2	3	3	3	**21**
Taşıma yok	5	4	5	1	4	5	**24**
Yer değiştir	5	5	5	1	5	5	**26**

LİSTE GÜNCELLEME · ADIM ADIM

Öne Taşı (MTF) Adım Adım

Başlangıç listesi A B C D E; istekler **E D E A D E**.



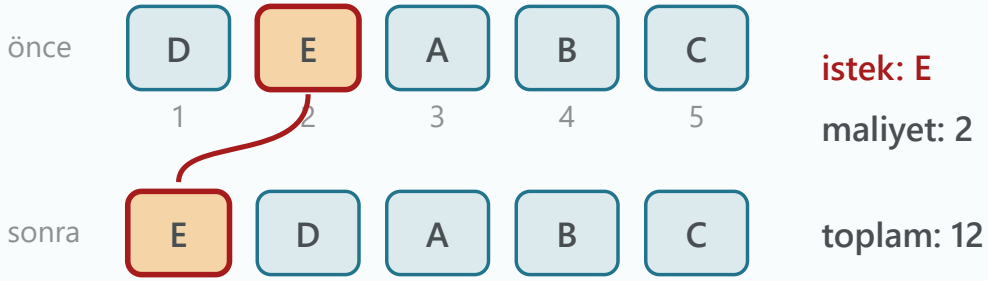
İstek **D**: 5. konumda → maliyet 5; **başa taşınır**. Toplam: 10.

Adım 2 / 6

Strateji	E	D	E	A	D	E	Toplam
Öne taşı (MTF)	5	5	2	3	3	3	**21**
Taşıma yok	5	4	5	1	4	5	**24**
Yer değiştir	5	5	5	1	5	5	**26**

Öne Taşı (MTF) Adım Adım

Başlangıç listesi A B C D E; istekler **E D E A D E**.

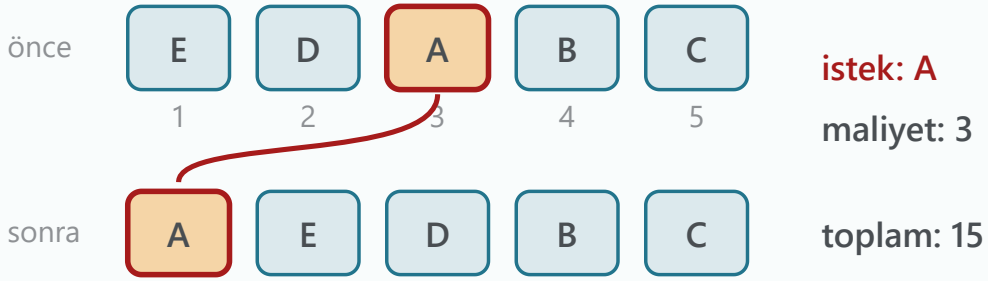


İstek **E**: 2. konumda → maliyet 2; **başa taşınır**. Toplam: 12.

Strateji	E	D	E	A	D	E	Toplam
Öne taşı (MTF)	5	5	2	3	3	3	**21**
Taşıma yok	5	4	5	1	4	5	**24**
Yer değiştir	5	5	5	1	5	5	**26**

Öne Taşı (MTF) Adım Adım

Başlangıç listesi A B C D E; istekler **E D E A D E**.

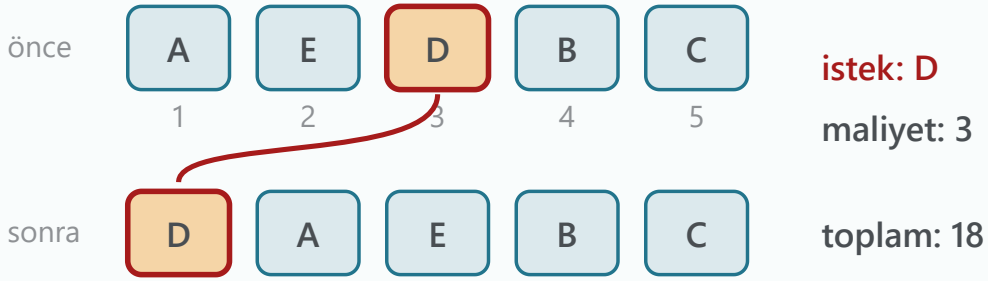


İstek **A**: 3. konumda → maliyet 3; **başa taşınır**. Toplam: 15.

Strateji	E	D	E	A	D	E	Toplam
Öne taşı (MTF)	5	5	2	3	3	3	**21**
Taşıma yok	5	4	5	1	4	5	**24**
Yer değiştir	5	5	5	1	5	5	**26**

Öne Taşı (MTF) Adım Adım

Başlangıç listesi A B C D E; istekler **E D E A D E**.

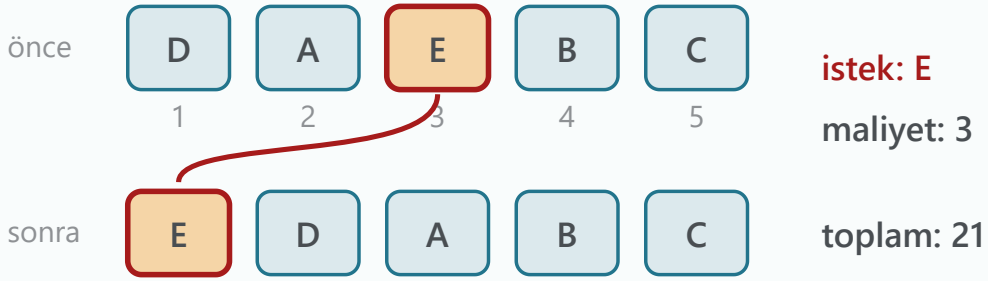


İstek **D**: 3. konumda → maliyet 3; **başa taşınır**. Toplam: 18.

Strateji	E	D	E	A	D	E	Toplam
Öne taşı (MTF)	5	5	2	3	3	3	**21**
Taşıma yok	5	4	5	1	4	5	**24**
Yer değiştir	5	5	5	1	5	5	**26**

Öne Taşı (MTF) Adım Adım

Başlangıç listesi A B C D E; istekler **E D E A D E**.



İstek **E**: 3. konumda → maliyet 3; **başa taşınır**. Toplam: 21. Aynı dizide hiç taşımayan liste 24, transpoz (transpose) 26 öder.

Strateji	E	D	E	A	D	E	Toplam
Öne taşı (MTF)	5	5	2	3	3	3	**21**
Taşıma yok	5	4	5	1	4	5	**24**
Yer değiştir	5	5	5	1	5	5	**26**

Teorem: MTF 2-Rekabetçidir

- **Sleator ve Tarjan (1985):** her istek dizisi σ için $MTF(\sigma) \le 2 \cdot OPT(\sigma) - m$ (m: istek sayısı; OPT'nin ücretli takasları da maliyetine katılır).
- Kanıt fikri: **potansiyel fonksiyonu** Φ = MTF listesi ile OPT listesi arasındaki **ters sıralı çift** (inversion) sayısı. Her erişimde MTF'nin gerçek maliyeti + $\Delta\Phi \le 2 \cdot$ (OPT maliyeti) − 1. Bu, amortize analizdir.

Yer değiştir rekabetçi değil

- Son iki öğe sırayla istenir: $x_n, x_{n-1}, x_n, \ldots$ Her erişim yerini değiştirir ama hep n. ve (n−1). konumda kalır → istek başına ≈ n. MTF ise ilk ikisinden sonra istek başına 2 öder.

Sıklık sayacı da değil

- Önce x_1 n kez, x_2 n − 1 kez, … istenir; sonra ters sırada. Sayaç eski sırayı uzun süre korur; maliyet $\Theta(n)$ kat kötüleşebilir.

Veri Akışı (Data Stream) Modeli

- Öğeler x_1, x_2, … tek tek gelir; akış çok uzun ya da **sonsuz** olabilir.
- Her öğeye **bir kez** bakılır (tek geçiş); geçmiş tümüyle saklanamaz.
- Her an bir sorunun cevabı hazır olmalı: "şu ana kadarki ortalama / ortanca / bir örneklem nedir?"

Problem	Saklanan özet	Öğe başına	Bellek
Hareketli ortalama (son k)	Kuyruk + toplam	O(1)	O(k)
Çevrim içi ortanca	Max-heap + min-heap	O(log n)	O(n)
Rezervuar örnekleme	k öğelik örneklem	O(1)	O(k)
Akışta örüntü arama	Son m karakter + kayan hash	O(1)	O(m)
Çalışan ortalama	n ve μ	O(1)	O(1)

Hareketli Ortalama (Moving Average)

- Akıştaki **son k öğenin** ortalaması: sensör verisini yumuşatma, borsada 50 günlük ortalama.
- Saf yöntem her seferinde k öğeyi toplar: **O(k)**.
- Akıllı yöntem: **kayan pencere** (sliding window). Toplamı tut; yeni öğe girerken ekle, en eski çıkarken çıkar: **O(1)**.

$$\text{toplam} \leftarrow \text{toplam} - x_{t-k} + x_t$$
$$\text{ortalama} = \text{toplam} / \min(t, k)$$

```
class HareketliOrtalama {
  private final int k;
  private final Queue<Integer> pencere
      = new LinkedList<>();
  private int toplam = 0;

  HareketliOrtalama(int k) { this.k = k; }

  double sonraki(int deger) {
    if (pencere.size() == k)
      toplam -= pencere.poll();
    pencere.offer(deger);
    toplam += deger;
    return toplam / (double) pencere.size();
  }
}
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
▶   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
4   pencere.offer(deger);
5   toplam += deger;
6   return toplam / (double) pencere.size();
7 }
```

akış: **4** 8 6 2 10

pencere: [ ] [ ] [ ]

toplam = 0

Yeni değer 4. Pencerede 0 öğe var (< 3) → çıkarma yok.

Adım 1 / 17

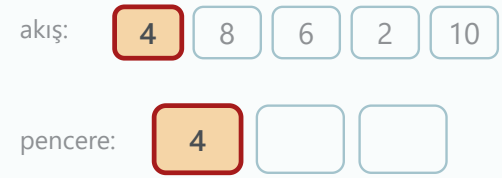
DEĞİŞKENLER

`k = 3` `deger = 4` `toplam = 0` `pencere = []`

ÇIKTI

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
2   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
▶   pencere.offer(deger);
▶   toplam += deger;
6   return toplam / (double) pencere.size();
7 }
```



toplam = 4

4 kuyruğa eklenir, toplama katılır: toplam = 4.

Adım 2 / 17

DEĞİŞKENLER

`k = 3` `deger = 4` `toplam = 4` `pencere = [4]`

ÇIKTI

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
2   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
4   pencere.offer(deger);
5   toplam += deger;
▶   return toplam / (double) pencere.size();
7 }
```

akış: **4** 8 6 2 10

pencere: **4**

toplam = 4

Ortalama = 4 / 1 = **4**.

Adım 3 / 17

DEĞİŞKENLER

`k = 3` `deger = 4` `toplam = 4` `pencere = [4]`

ÇIKTI

```
4.00
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
▶   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
4   pencere.offer(deger);
5   toplam += deger;
6   return toplam / (double) pencere.size();
7 }
```



toplam = 4

Yeni değer 8. Pencerede 1 öğe var (< 3) → çıkarma yok.

Adım 4 / 17

DEĞİŞKENLER

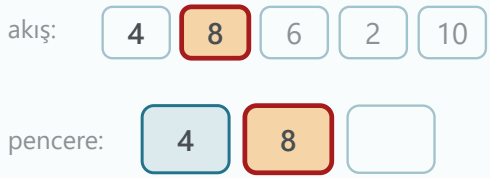
`k = 3` `deger = 8` `toplam = 4` `pencere = [4]`

ÇIKTI

```
4.00
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
2   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
▶   pencere.offer(deger);
▶   toplam += deger;
6   return toplam / (double) pencere.size();
7 }
```



toplam = 12

8 kuyruğa eklenir, toplama katılır: toplam = 12.

Adım 5 / 17

DEĞİŞKENLER

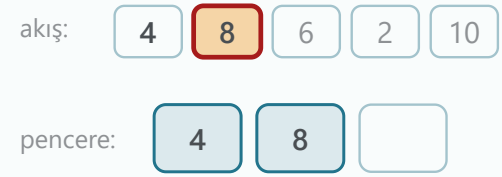
`k = 3` `deger = 8` `toplam = 12` `pencere = [4, 8]`

ÇIKTI

```
4.00
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
2   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
4   pencere.offer(deger);
5   toplam += deger;
▶   return toplam / (double) pencere.size();
7 }
```



toplam = 12

Ortalama = 12 / 2 = **6**.

Adım 6 / 17

DEĞİŞKENLER

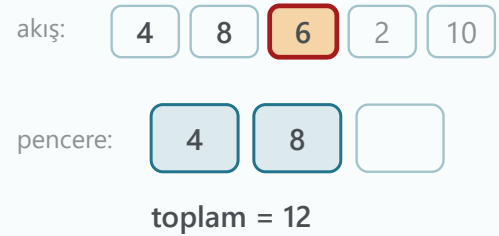
`k = 3` `deger = 8` `toplam = 12` `pencere = [4, 8]`

ÇIKTI

```
4.00 6.00
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
▶   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
4   pencere.offer(deger);
5   toplam += deger;
6   return toplam / (double) pencere.size();
7 }
```



toplam = 12

Yeni değer 6. Pencerede 2 öğe var (< 3) → çıkarma yok.

Adım 7 / 17

DEĞİŞKENLER

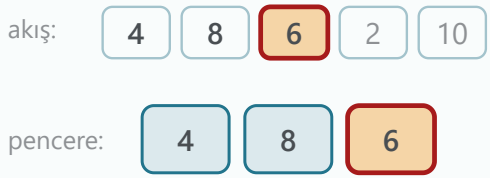
`k = 3` `deger = 6` `toplam = 12` `pencere = [4, 8]`

ÇIKTI

```
4.00 6.00
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
2   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
▶   pencere.offer(deger);
▶   toplam += deger;
6   return toplam / (double) pencere.size();
7 }
```



toplam = 18

6 kuyruğa eklenir, toplama katılır: toplam = 18.

Adım 8 / 17

DEĞİŞKENLER

`k = 3` `deger = 6` `toplam = 18`

`pencere = [4, 8, 6]`

ÇIKTI

```
4.00 6.00
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
2   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
4   pencere.offer(deger);
5   toplam += deger;
▶   return toplam / (double) pencere.size();
7 }
```

akış: 4 | 8 | 6 | 2 | 10

pencere: 4 | 8 | 6

toplam = 18

Ortalama = 18 / 3 = **6**.

Adım 9 / 17

DEĞİŞKENLER

`k = 3` `deger = 6` `toplam = 18` `pencere = [4, 8, 6]`

ÇIKTI

```
4.00 6.00 6.00
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
▶   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
4   pencere.offer(deger);
5   toplam += deger;
6   return toplam / (double) pencere.size();
7 }
```



Yeni değer 2. Pencere dolu (boyut 3 = k) → en eski çıkarılacak.

Adım 10 / 17

DEĞİŞKENLER

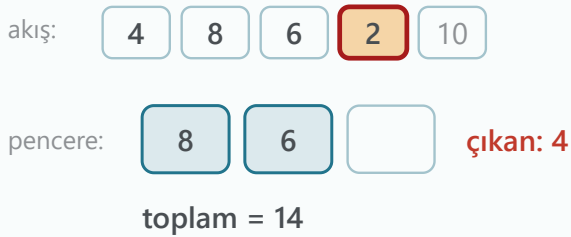
`k = 3` `deger = 2` `toplam = 18` `pencere = [4, 8, 6]`

ÇIKTI

```
4.00 6.00 6.00
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
2   if (pencere.size() == k)
▶     toplam -= pencere.poll(); // en eski
4   pencere.offer(deger);
5   toplam += deger;
6   return toplam / (double) pencere.size();
7 }
```



En eski öğe **4** kuyruktan alınır, toplamdan çıkarılır: toplam = 14.

Adım 11 / 17

DEĞİŞKENLER

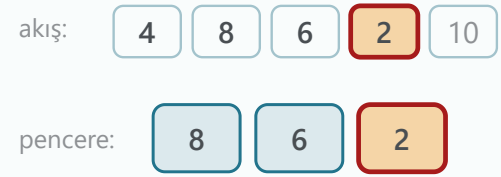
`k = 3` `deger = 2` `toplam = 14` `pencere = [8, 6]`

ÇIKTI

```
4.00 6.00 6.00
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
2   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
▶   pencere.offer(deger);
▶   toplam += deger;
6   return toplam / (double) pencere.size();
7 }
```



2 kuyruğa eklenir, toplama katılır: toplam = 16.

Adım 12 / 17

DEĞİŞKENLER

`k = 3` `deger = 2` `toplam = 16` `pencere = [8, 6, 2]`

ÇIKTI

```
4.00 6.00 6.00
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
2   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
4   pencere.offer(deger);
5   toplam += deger;
▶   return toplam / (double) pencere.size();
7 }
```

akış: 4 8 6 **2** 10

pencere: 8 6 2

toplam = 16

Ortalama = 16 / 3 = **5,33**.

Adım 13 / 17

DEĞİŞKENLER

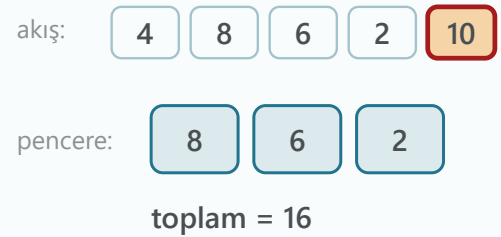
`k = 3` `deger = 2` `toplam = 16` `pencere = [8, 6, 2]`

ÇIKTI

```
4.00 6.00 6.00 5.33
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
▶   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
4   pencere.offer(deger);
5   toplam += deger;
6   return toplam / (double) pencere.size();
7 }
```



Yeni değer 10. Pencere dolu (boyut 3 = k) → en eski çıkarılacak.

Adım 14 / 17

DEĞİŞKENLER

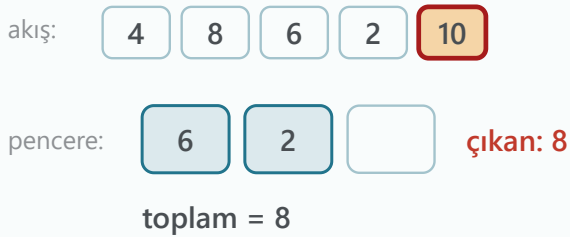
`k = 3` `deger = 10` `toplam = 16` `pencere = [8, 6, 2]`

ÇIKTI

```
4.00 6.00 6.00 5.33
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
2   if (pencere.size() == k)
▶     toplam -= pencere.poll(); // en eski
4   pencere.offer(deger);
5   toplam += deger;
6   return toplam / (double) pencere.size();
7 }
```



En eski öğe **8** kuyruktan alınır, toplamdan çıkarılır: toplam = 8.

Adım 15 / 17

DEĞİŞKENLER

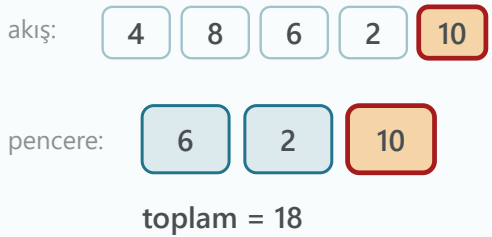
`k = 3` `deger = 10` `toplam = 8` `pencere = [6, 2]`

ÇIKTI

```
4.00 6.00 6.00 5.33
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
2   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
▶   pencere.offer(deger);
▶   toplam += deger;
6   return toplam / (double) pencere.size();
7 }
```



10 kuyruğa eklenir, toplama katılır: toplam = 18.

Adım 16 / 17

DEĞİŞKENLER

`k = 3` `deger = 10` `toplam = 18` `pencere = [6, 2, 10]`

ÇIKTI

```
4.00 6.00 6.00 5.33
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {
2   if (pencere.size() == k)
3     toplam -= pencere.poll(); // en eski
4   pencere.offer(deger);
5   toplam += deger;
▶   return toplam / (double) pencere.size();
7 }
```

akış: 4 8 6 2 10

pencere: 6 2 10

toplam = 18

Ortalama = 18 / 3 = **6**.

Adım 17 / 17

DEĞİŞKENLER

`k = 3` `deger = 10` `toplam = 18` `pencere = [6, 2, 10]`

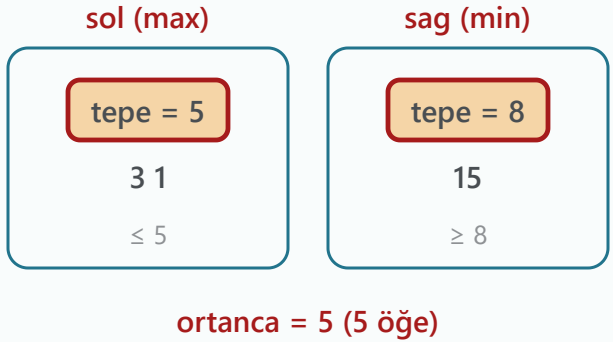
ÇIKTI

```
4.00 6.00 6.00 5.33 6.00
```

VERİ AKIŞLARI

Çevrim İçi Ortanca (Running Median)

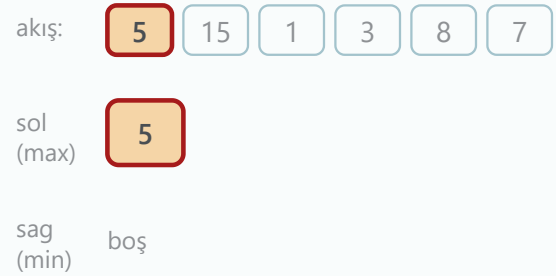
- Her yeni öğeden sonra **o ana kadarki ortanca** (median) istenir.
- Her seferinde sıralamak: O(n log n). Sıralı diziye eklemek: O(n).
- **İki heap**: küçük yarı bir **max-heap'te** (sol), büyük yarı bir **min-heap'te** (sag).
- Değişmezler: sol'daki her öğe ≤ sag'daki her öğe; |sol| = |sag| ya da |sol| = |sag| + 1.
- Ortanca heap tepelerinden **O(1)**'de okunur; ekleme **O(log n)**.



Sayı adedi tekse ortanca sol'un tepesi; çiftse iki tepenin ortalamasıdır. Java'da max-heap: `new PriorityQueue<>((a, b) -> b - a)`.

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
▶    if (sol.isEmpty() || x <= sol.peek())
▶      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
6    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
8    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



x = 5: sol boş → **sol** heap'ine eklenir.

Adım 1 / 18

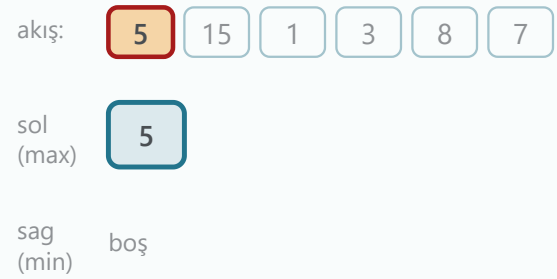
DEĞİŞKENLER

`x = 5` `sol = [5]` `sag = []` `ortanca = –`

ÇIKTI

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
2    if (sol.isEmpty() || x <= sol.peek())
3      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
▶    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
▶    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



Boyutlar 1 ve 0: denge tamam, taşıma yok.

Adım 2 / 18

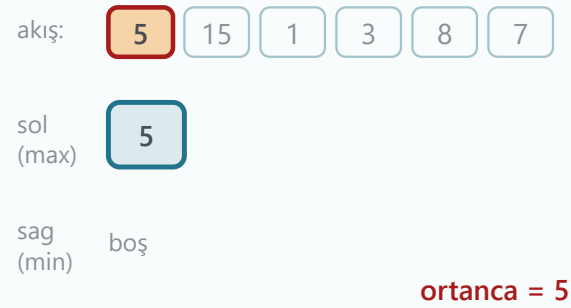
DEĞİŞKENLER

`x = 5` `sol = [5]` `sag = []` `ortanca = –`

ÇIKTI

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
2    if (sol.isEmpty() || x <= sol.peek())
3      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
6    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
8    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
▶    if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
▶    return sol.peek();
15 }
```



Öğe sayısı tek: ortanca = sol tepesi = **5**.

Adım 3 / 18

DEĞİŞKENLER

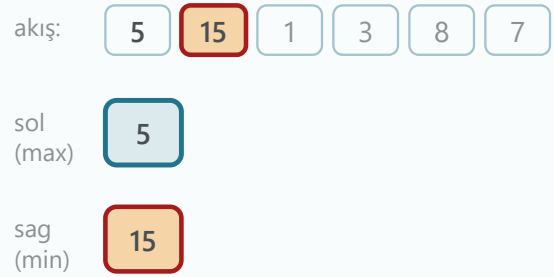
`x = 5` `sol = [5]` `sag = []` `ortanca = 5`

ÇIKTI

```
5.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
 1 void ekle(int x) {
 2   if (sol.isEmpty() || x <= sol.peek())
 3     sol.offer(x);          // küçük yarı
 4   else
 5     sag.offer(x);          // büyük yarı
 6   if (sol.size() > sag.size() + 1)
 7     sag.offer(sol.poll());
 8   else if (sag.size() > sol.size())
 9     sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



x = 15: 15 > 5 → **sag** heap'ine eklenir.

Adım 4 / 18

DEĞİŞKENLER

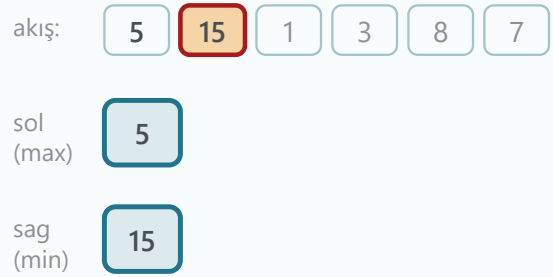
`x = 15` `sol = [5]` `sag = [15]` `ortanca = –`

ÇIKTI

```
5.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
2    if (sol.isEmpty() || x <= sol.peek())
3      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
▶    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
▶    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



Boyutlar 1 ve 1: denge tamam, taşıma yok.

Adım 5 / 18

DEĞİŞKENLER

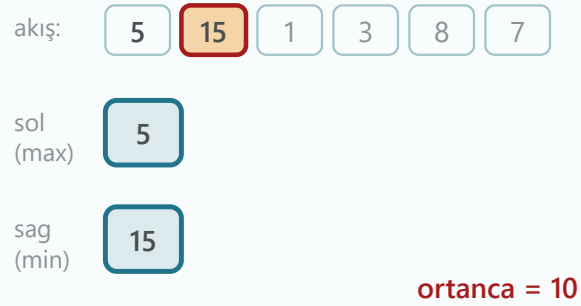
`x = 15` `sol = [5]` `sag = [15]` `ortanca = –`

ÇIKTI

```
5.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
2    if (sol.isEmpty() || x <= sol.peek())
3      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
6    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
8    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
▶      return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



Öğe sayısı çift: ortanca = (5 + 15) / 2 = **10**.

Adım 6 / 18

DEĞİŞKENLER

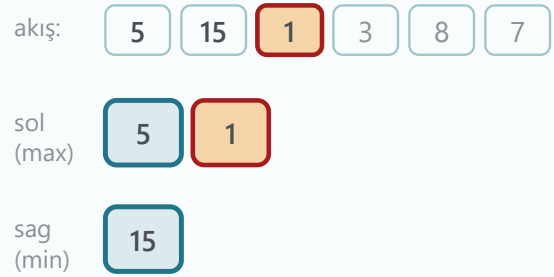
`x = 15` `sol = [5]` `sag = [15]` `ortanca = 10`

ÇIKTI

```
5.0 10.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
▶    if (sol.isEmpty() || x <= sol.peek())
▶      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
6    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
8    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



x = 1: 1 ≤ 5 → **sol** heap'ine eklenir.

Adım 7 / 18

DEĞİŞKENLER

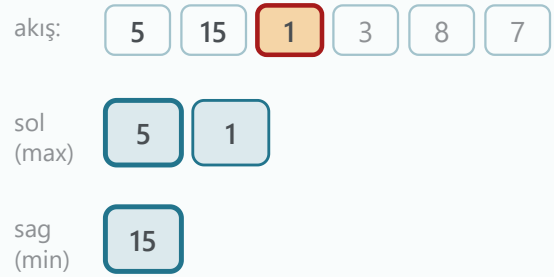
`x = 1` `sol = [5, 1]` `sag = [15]` `ortanca = –`

ÇIKTI

```
5.0 10.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
2    if (sol.isEmpty() || x <= sol.peek())
3      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
6    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
8    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



Boyutlar 2 ve 1: denge tamam, taşıma yok.

Adım 8 / 18

DEĞİŞKENLER

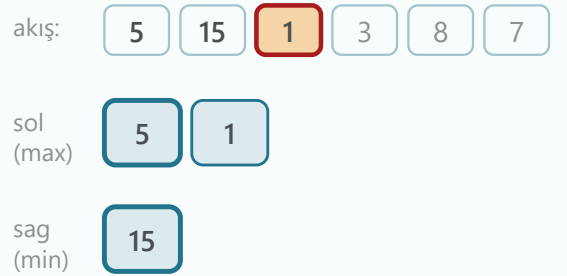
`x = 1` `sol = [5, 1]` `sag = [15]` `ortanca = –`

ÇIKTI

```
5.0 10.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
2    if (sol.isEmpty() || x <= sol.peek())
3      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
6    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
8    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
▶    if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
▶    return sol.peek();
15 }
```



Öğe sayısı tek: ortanca = sol tepesi = **5**.

Adım 9 / 18

DEĞİŞKENLER

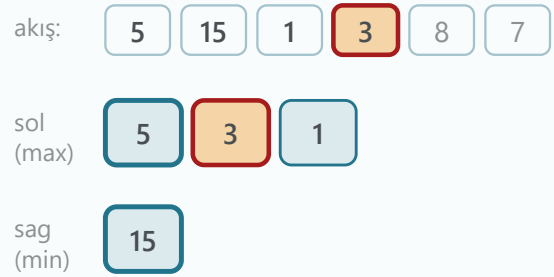
`x = 1` `sol = [5, 1]` `sag = [15]` `ortanca = 5`

ÇIKTI

```
5.0 10.0 5.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
▶    if (sol.isEmpty() || x <= sol.peek())
▶      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
6    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
8    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



x = 3: 3 ≤ 5 → **sol** heap'ine eklenir.

Adım 10 / 18

DEĞİŞKENLER

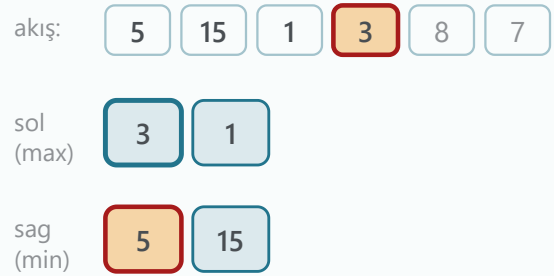
`x = 3` `sol = [5, 3, 1]` `sag = [15]` `ortanca = –`

ÇIKTI

```
5.0 10.0 5.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
2    if (sol.isEmpty() || x <= sol.peek())
3      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
▶    if (sol.size() > sag.size() + 1)
▶      sag.offer(sol.poll());
8    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



Denge bozuldu (|sol| = 3, |sag| = 1): sol tepesi **5** sag heap'ine taşınır.

Adım 11 / 18

DEĞİŞKENLER

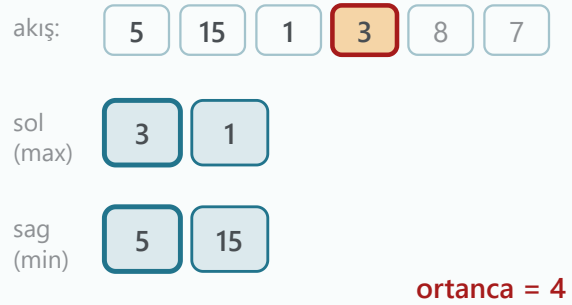
`x = 3` `sol = [3, 1]` `sag = [5, 15]` `ortanca = –`

ÇIKTI

```
5.0 10.0 5.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
2    if (sol.isEmpty() || x <= sol.peek())
3      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
6    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
8    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
▶      return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



Öğe sayısı çift: ortanca = (3 + 5) / 2 = **4**.

Adım 12 / 18

DEĞİŞKENLER

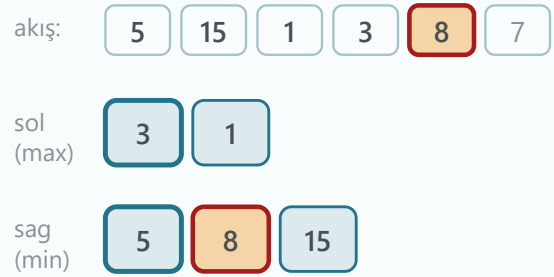
`x = 3` `sol = [3, 1]` `sag = [5, 15]` `ortanca = 4`

ÇIKTI

```
5.0 10.0 5.0 4.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
 1 void ekle(int x) {
 2   if (sol.isEmpty() || x <= sol.peek())
 3     sol.offer(x);          // küçük yarı
 4   else
 5     sag.offer(x);          // büyük yarı
 6   if (sol.size() > sag.size() + 1)
 7     sag.offer(sol.poll());
 8   else if (sag.size() > sol.size())
 9     sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



x = 8: 8 > 3 → **sag** heap'ine eklenir.

Adım 13 / 18

DEĞİŞKENLER

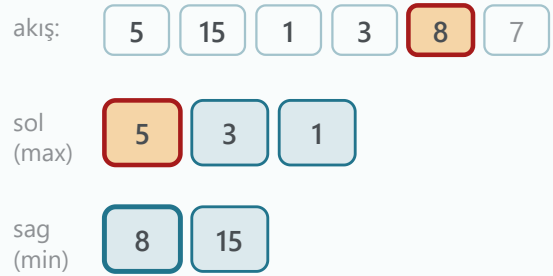
`x = 8` `sol = [3, 1]` `sag = [5, 8, 15]` `ortanca = –`

ÇIKTI

```
5.0 10.0 5.0 4.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
2    if (sol.isEmpty() || x <= sol.peek())
3      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
▶    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
▶    else if (sag.size() > sol.size())
▶      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



Denge bozuldu (|sol| = 2, |sag| = 3): sag tepesi **5** sol heap'ine taşınır.

Adım 14 / 18

DEĞİŞKENLER

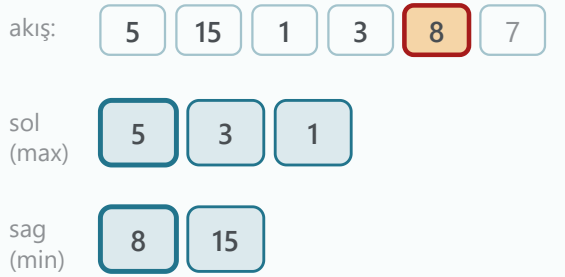
`x = 8` `sol = [5, 3, 1]` `sag = [8, 15]` `ortanca = –`

ÇIKTI

```
5.0 10.0 5.0 4.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
 1 void ekle(int x) {
 2   if (sol.isEmpty() || x <= sol.peek())
 3     sol.offer(x);          // küçük yarı
 4   else
 5     sag.offer(x);          // büyük yarı
 6   if (sol.size() > sag.size() + 1)
 7     sag.offer(sol.poll());
 8   else if (sag.size() > sol.size())
 9     sol.offer(sag.poll());
10 }
11 double ortanca() {
▶  if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
▶  return sol.peek();
15 }
```



Öğe sayısı tek: ortanca = sol tepesi = **5**.

Adım 15 / 18

DEĞİŞKENLER

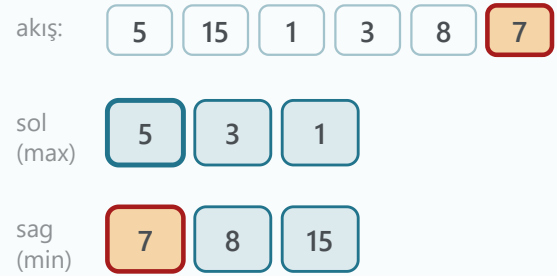
`x = 8` `sol = [5, 3, 1]` `sag = [8, 15]` `ortanca = 5`

ÇIKTI

```
5.0 10.0 5.0 4.0 5.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
▶    if (sol.isEmpty() || x <= sol.peek())
3      sol.offer(x);          // küçük yarı
▶    else
▶      sag.offer(x);          // büyük yarı
6    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
8    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



x = 7: 7 > 5 → **sag** heap'ine eklenir.

Adım 16 / 18

DEĞİŞKENLER

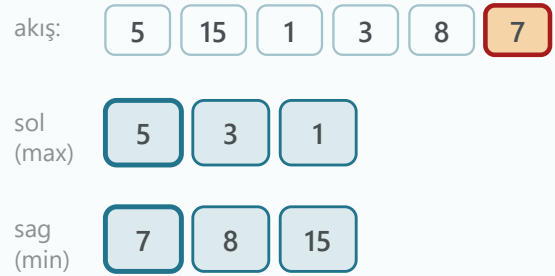
`x = 7` `sol = [5, 3, 1]` `sag = [7, 8, 15]` `ortanca = –`

ÇIKTI

```
5.0 10.0 5.0 4.0 5.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
2    if (sol.isEmpty() || x <= sol.peek())
3      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
▶    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
▶    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
13     return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



Boyutlar 3 ve 3: denge tamam, taşıma yok.

Adım 17 / 18

DEĞİŞKENLER

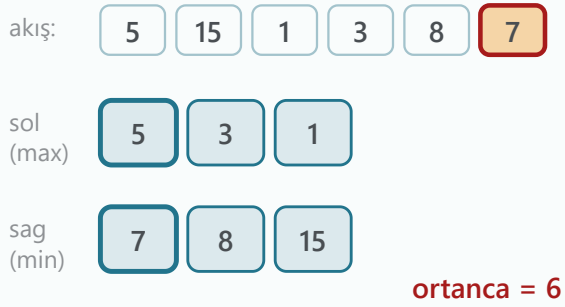
`x = 7` `sol = [5, 3, 1]` `sag = [7, 8, 15]` `ortanca = –`

ÇIKTI

```
5.0 10.0 5.0 4.0 5.0
```

Çevrim İçi Ortanca: Kodu Adım Adım İzle

```
1  void ekle(int x) {
2    if (sol.isEmpty() || x <= sol.peek())
3      sol.offer(x);          // küçük yarı
4    else
5      sag.offer(x);          // büyük yarı
6    if (sol.size() > sag.size() + 1)
7      sag.offer(sol.poll());
8    else if (sag.size() > sol.size())
9      sol.offer(sag.poll());
10 }
11 double ortanca() {
12   if (sol.size() == sag.size())
▶      return (sol.peek() + sag.peek()) / 2.0;
14   return sol.peek();
15 }
```



Öğe sayısı çift: ortanca = (5 + 7) / 2 = **6**.

Adım 18 / 18

DEĞİŞKENLER

`x = 7` `sol = [5, 3, 1]` `sag = [7, 8, 15]` `ortanca = 6`

ÇIKTI

```
5.0 10.0 5.0 4.0 5.0 6.0
```

VERİ AKIŞLARI

Rezervuar Örnekleme (Reservoir Sampling)

- Uzunluğu **bilinmeyen** bir akıştan, her öğenin **eşit olasılıkla** seçildiği k öğeli bir örneklem tutmak.
- **Algoritma R** (Vitter, 1985): ilk k öğe rezervuara konur. Sonra 0 tabanlı i. öğe için ($i \geq k$) $j = \text{rastgele}[0, i]$ seçilir; **$j < k$** ise $\text{rezervuar}[j] \leftarrow x_i$.
- i. öğe **k / (i + 1)** olasılıkla girer; girerse rastgele bir yerdeki öğeyi çıkarır.

Kanıt (n öğe sonunda her öğe k / n olasılıkla rezervuardadır): x_i girer: $k/(i+1)$. Sonraki her $t > i$ adımında çıkarılmama olasılığı $1 - (k/(t+1)) \cdot (1/k) = t/(t+1)$. Çarpım:

$$k/(i+1) \cdot (i+1)/(i+2) \cdot \ldots \cdot (n-1)/n = \mathbf{k / n}$$

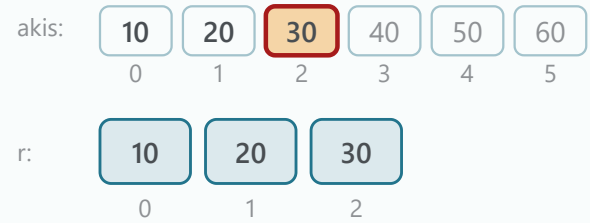


Örnek: $k = 2$, $n = 5$, x_2 öğesi.

İlk k öğe için de aynı: girme olasılığı 1, sonraki çarpım $k/(k+1) \cdot \ldots \cdot (n-1)/n = k/n$. Bellek $O(k)$, öğe başına $O(1)$.

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1  List<Integer> rezervuar(int[] akis, int k) {
2    List<Integer> r = new ArrayList<>();
3    for (int i = 0; i < k; i++) r.add(akis[i]);
4    for (int i = k; i < akis.length; i++) {
5      int j = rastgele.nextInt(i + 1); // 0..i
6      if (j < k)
7        r.set(j, akis[i]);
8    }
9    return r;
10 }
```



İlk k = 3 öğe (10, 20, 30) doğrudan rezervuara konur.

Adım 1 / 8

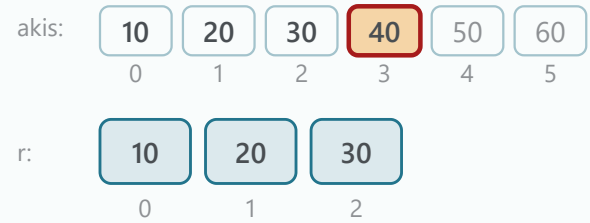
DEĞİŞKENLER

`k = 3` `i = –` `j = –`

`olasılık = –` `r = [10, 20, 30]`

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1  List<Integer> rezervuar(int[] akis, int k) {
2    List<Integer> r = new ArrayList<>();
3    for (int i = 0; i < k; i++) r.add(akis[i]);
▶    for (int i = k; i < akis.length; i++) {
▶      int j = rastgele.nextInt(i + 1); // 0..i
6      if (j < k)
7        r.set(j, akis[i]);
8    }
9    return r;
10 }
```



i = 3 (değer 40): j = rastgele[0, 3] → **j = 1** çıktı (bu izlemede sabitlendi).

Adım 2 / 8

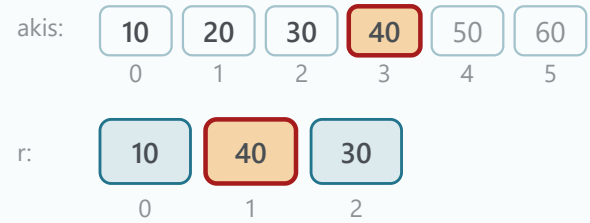
DEĞİŞKENLER

`k = 3` `i = 3` `j = 1`

`olasılık = 3/4` `r = [10, 20, 30]`

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1  List<Integer> rezervuar(int[] akis, int k) {
2    List<Integer> r = new ArrayList<>();
3    for (int i = 0; i < k; i++) r.add(akis[i]);
4    for (int i = k; i < akis.length; i++) {
5      int j = rastgele.nextInt(i + 1); // 0..i
▶      if (j < k)
▶        r.set(j, akis[i]);
8    }
9    return r;
10 }
```



j = 1 < 3 → r[1] = 40 (20 çıkar). 40'ın girme olasılığı 3/4.

Adım 3 / 8

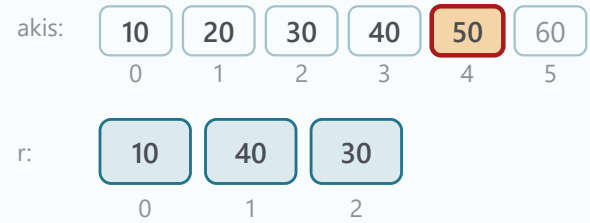
DEĞİŞKENLER

`k = 3` `i = 3` `j = 1`

`olasılık = 3/4` `r = [10, 40, 30]`

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1  List<Integer> rezervuar(int[] akis, int k) {
2    List<Integer> r = new ArrayList<>();
3    for (int i = 0; i < k; i++) r.add(akis[i]);
▶    for (int i = k; i < akis.length; i++) {
▶      int j = rastgele.nextInt(i + 1); // 0..i
6      if (j < k)
7        r.set(j, akis[i]);
8    }
9    return r;
10 }
```



i = 4 (değer 50): j = rastgele[0, 4] → **j = 4** çıktı (bu izlemede sabitlendi).

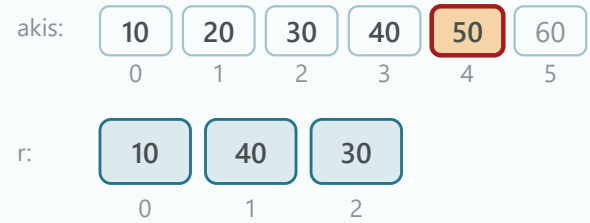
Adım 4 / 8

DEĞİŞKENLER

`k = 3` `i = 4` `j = 4` `olasılık = 3/5` `r = [10, 40, 30]`

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1  List<Integer> rezervuar(int[] akis, int k) {
2    List<Integer> r = new ArrayList<>();
3    for (int i = 0; i < k; i++) r.add(akis[i]);
4    for (int i = k; i < akis.length; i++) {
5      int j = rastgele.nextInt(i + 1); // 0..i
▶      if (j < k)
7        r.set(j, akis[i]);
8    }
9    return r;
10 }
```



j = 4 ≥ 3 → 50 rezervuara **girmez**; rezervuar değişmez.

Adım 5 / 8

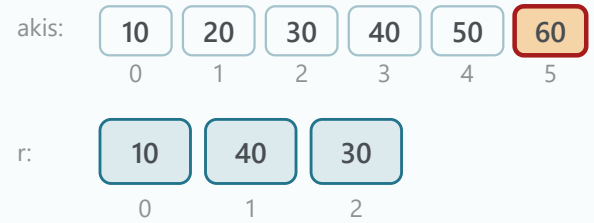
DEĞİŞKENLER

`k = 3` `i = 4` `j = 4`

`olasılık = 3/5` `r = [10, 40, 30]`

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1  List<Integer> rezervuar(int[] akis, int k) {
2    List<Integer> r = new ArrayList<>();
3    for (int i = 0; i < k; i++) r.add(akis[i]);
▶    for (int i = k; i < akis.length; i++) {
▶      int j = rastgele.nextInt(i + 1); // 0..i
6      if (j < k)
7        r.set(j, akis[i]);
8    }
9    return r;
10 }
```



i = 5 (değer 60): j = rastgele[0, 5] → **j = 0** çıktı (bu izlemede sabitlendi).

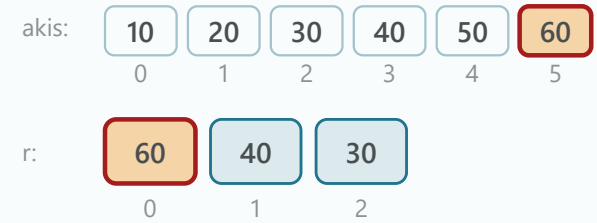
Adım 6 / 8

DEĞİŞKENLER

`k = 3` `i = 5` `j = 0` `olasılık = 3/6` `r = [10, 40, 30]`

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1  List<Integer> rezervuar(int[] akis, int k) {
2    List<Integer> r = new ArrayList<>();
3    for (int i = 0; i < k; i++) r.add(akis[i]);
4    for (int i = k; i < akis.length; i++) {
5      int j = rastgele.nextInt(i + 1); // 0..i
▶      if (j < k)
▶        r.set(j, akis[i]);
8    }
9    return r;
10 }
```



j = 0 < 3 → r[0] = 60 (10 çıkar). 60'ın girme olasılığı 3/6.

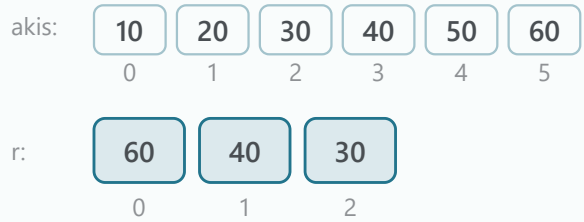
Adım 7 / 8

DEĞİŞKENLER

`k = 3` `i = 5` `j = 0` `olasılık = 3/6` `r = [60, 40, 30]`

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1  List<Integer> rezervuar(int[] akis, int k) {
2    List<Integer> r = new ArrayList<>();
3    for (int i = 0; i < k; i++) r.add(akis[i]);
4    for (int i = k; i < akis.length; i++) {
5      int j = rastgele.nextInt(i + 1); // 0..i
6      if (j < k)
7        r.set(j, akis[i]);
8    }
▶    return r;
10 }
```



Akış bitti: örneklem [60, 40, 30]. Başka j değerleriyle başka sonuç çıkar; her öğe 3/6 = 1/2 olasılıkla seçilir.

Adım 8 / 8

DEĞİŞKENLER

`k = 3` `i = 6` `j = –` `olasılık = –` `r = [60, 40, 30]`

Akışta Örüntü Arama: Kayan Hash

```
1  void ara(String akis, String oruntu) {
▶    int m = oruntu.length(), B = 101, h = 0, hp = 0;
▶    int gucu = 1;                          // B^(m-1)
▶    for (int i = 0; i < m - 1; i++) gucu *= B;
▶    for (int i = 0; i < m; i++)
▶      hp = hp * B + oruntu.charAt(i);
7    for (int t = 0; t < akis.length(); t++) {
8      char c = akis.charAt(t);
9      if (t >= m) h -= akis.charAt(t - m) * gucu;
10     h = h * B + c;           // giren eklenir
11     if (t >= m - 1 && h == hp
12         && akis.startsWith(oruntu, t - m + 1))
13       System.out.println("eşleşme: " + (t - m + 1));
14   }
15 }
```



h = 0 hp = 6631 ("AB")

Örüntü "AB", m = 2, B = 101. gucu = 101^{m-1} = 101, hp = 65 · 101 + 66 = **6631**. Akış "CABDAB" karakter karakter gelir; her yeni karakterde h_{yeni} = (h_{eski} − $c_{çıkan}$ · B^{m-1}) · B + c_{giren}: baştan O(m) yerine **O(1)**.

Adım 1 / 7

DEĞİŞKENLER

`m = 2` `gucu = 101` `hp = 6631` `h = 0`

ÇIKTI

Akışta Örüntü Arama: Kayan Hash

```
1  void ara(String akis, String oruntu) {
2    int m = oruntu.length(), B = 101, h = 0, hp = 0;
3    int gucu = 1;                          // B^(m-1)
4    for (int i = 0; i < m - 1; i++) gucu *= B;
5    for (int i = 0; i < m; i++)
6      hp = hp * B + oruntu.charAt(i);
7    for (int t = 0; t < akis.length(); t++) {
▶      char c = akis.charAt(t);
9      if (t >= m) h -= akis.charAt(t - m) * gucu;
▶      h = h * B + c;           // giren eklenir
11     if (t >= m - 1 && h == hp
12         && akis.startsWith(oruntu, t - m + 1))
13       System.out.println("eşleşme: " + (t - m + 1));
14   }
15 }
```



h = 67 **hp = 6631 ("AB")**

0·101 + 67

t = 0, c = C (67): pencere henüz dolmadı; h = 67.

Adım 2 / 7

DEĞİŞKENLER

`t = 0` `c = C` `h = 67` `hp = 6631`

ÇIKTI

Akışta Örüntü Arama: Kayan Hash

```
1  void ara(String akis, String oruntu) {
2    int m = oruntu.length(), B = 101, h = 0, hp = 0;
3    int gucu = 1;                          // B^(m-1)
4    for (int i = 0; i < m - 1; i++) gucu *= B;
5    for (int i = 0; i < m; i++)
6      hp = hp * B + oruntu.charAt(i);
7    for (int t = 0; t < akis.length(); t++) {
▶      char c = akis.charAt(t);
9      if (t >= m) h -= akis.charAt(t - m) * gucu;
▶      h = h * B + c;           // giren eklenir
11     if (t >= m - 1 && h == hp
12         && akis.startsWith(oruntu, t - m + 1))
13       System.out.println("eşleşme: " + (t - m + 1));
14   }
15 }
```



h = 6832 hp = 6631 ("AB")

67·101 + 65

t = 1: Pencere doldu: "CA". h = 6832 ≠ hp → eşleşme yok.

Adım 3 / 7

DEĞİŞKENLER

t = 1 | c = A | h = 6832 | hp = 6631

ÇIKTI

Akışta Örüntü Arama: Kayan Hash

```
1  void ara(String akis, String oruntu) {
2    int m = oruntu.length(), B = 101, h = 0, hp = 0;
3    int gucu = 1;                          // B^(m-1)
4    for (int i = 0; i < m - 1; i++) gucu *= B;
5    for (int i = 0; i < m; i++)
6      hp = hp * B + oruntu.charAt(i);
7    for (int t = 0; t < akis.length(); t++) {
▶      char c = akis.charAt(t);
▶      if (t >= m) h -= akis.charAt(t - m) * gucu;
▶      h = h * B + c;          // giren eklenir
▶      if (t >= m - 1 && h == hp
▶          && akis.startsWith(oruntu, t - m + 1))
▶        System.out.println("eşleşme: " + (t - m + 1));
14   }
15 }
```



h = 6631 **hp = 6631 ("AB")**

(6832 − 67·101)·101 + 66

t = 2: Çıkan C (67 · 101 çıkarılır), giren B. h = 6631 = hp ve karakterler de tutuyor → **eşleşme: 1**.

Adım 4 / 7

DEĞİŞKENLER

`t = 2` `c = B` `h = 6631` `hp = 6631`

ÇIKTI

```
eşleşme: 1
```

Akışta Örüntü Arama: Kayan Hash

```
1  void ara(String akis, String oruntu) {
2    int m = oruntu.length(), B = 101, h = 0, hp = 0;
3    int gucu = 1;                          // B^(m-1)
4    for (int i = 0; i < m - 1; i++) gucu *= B;
5    for (int i = 0; i < m; i++)
6      hp = hp * B + oruntu.charAt(i);
7    for (int t = 0; t < akis.length(); t++) {
▶      char c = akis.charAt(t);
▶      if (t >= m) h -= akis.charAt(t - m) * gucu;
▶      h = h * B + c;           // giren eklenir
11     if (t >= m - 1 && h == hp
12         && akis.startsWith(oruntu, t - m + 1))
13       System.out.println("eşleşme: " + (t - m + 1));
14   }
15 }
```



h = 6734 **hp = 6631 ("AB")**

(6631 − 65·101)·101 + 68

t = 3: Çıkan A (65 · 101 çıkarılır), giren D. h = 6734 ≠ hp → eşleşme yok.

Adım 5 / 7

DEĞİŞKENLER

`t = 3` `c = D` `h = 6734` `hp = 6631`

ÇIKTI

```
eşleşme: 1
```

Akışta Örüntü Arama: Kayan Hash

```
1  void ara(String akis, String oruntu) {
2    int m = oruntu.length(), B = 101, h = 0, hp = 0;
3    int gucu = 1;                          // B^(m-1)
4    for (int i = 0; i < m - 1; i++) gucu *= B;
5    for (int i = 0; i < m; i++)
6      hp = hp * B + oruntu.charAt(i);
7    for (int t = 0; t < akis.length(); t++) {
▶      char c = akis.charAt(t);
▶      if (t >= m) h -= akis.charAt(t - m) * gucu;
▶      h = h * B + c;           // giren eklenir
11     if (t >= m - 1 && h == hp
12         && akis.startsWith(oruntu, t - m + 1))
13       System.out.println("eşleşme: " + (t - m + 1));
14   }
15 }
```



h = 6933 **hp = 6631 ("AB")**

(6734 − 66·101)·101 + 65

t = 4: Çıkan B (66 · 101 çıkarılır), giren A. h = 6933 ≠ hp → eşleşme yok.

Adım 6 / 7

DEĞİŞKENLER

`t = 4` `c = A` `h = 6933` `hp = 6631`

ÇIKTI

```
eşleşme: 1
```

Akışta Örüntü Arama: Kayan Hash

```
1  void ara(String akis, String oruntu) {
2    int m = oruntu.length(), B = 101, h = 0, hp = 0;
3    int gucu = 1;                          // B^(m-1)
4    for (int i = 0; i < m - 1; i++) gucu *= B;
5    for (int i = 0; i < m; i++)
6      hp = hp * B + oruntu.charAt(i);
7    for (int t = 0; t < akis.length(); t++) {
▶      char c = akis.charAt(t);
▶      if (t >= m) h -= akis.charAt(t - m) * gucu;
▶      h = h * B + c;          // giren eklenir
▶      if (t >= m - 1 && h == hp
▶          && akis.startsWith(oruntu, t - m + 1))
▶        System.out.println("eşleşme: " + (t - m + 1));
14   }
15 }
```



h = 6631 **hp = 6631 ("AB")**

(6933 − 68·101)·101 + 66

t = 5: Çıkan D (68 · 101 çıkarılır), giren B. h = 6631 = hp ve karakterler de tutuyor → **eşleşme: 4**. Hash eşitliği yetmez: çakışma olabileceği için karakterler de karşılaştırılır.

Adım 7 / 7

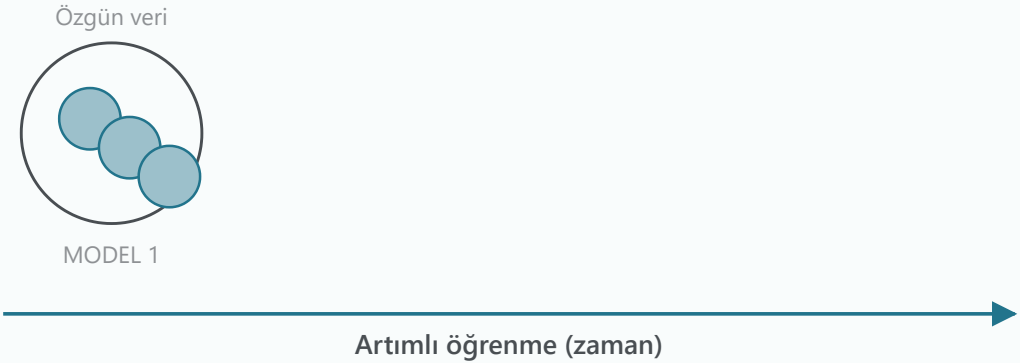
DEĞİŞKENLER

`t = 5` `c = B` `h = 6631` `hp = 6631`

ÇIKTI

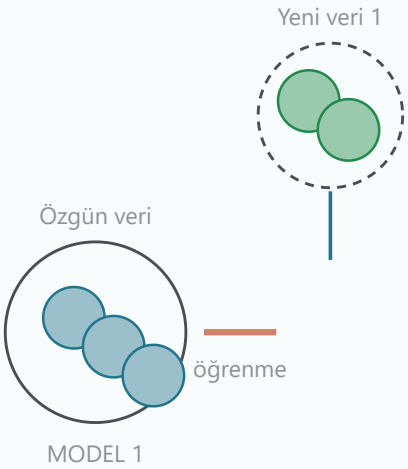
```
eşleşme: 1
eşleşme: 4
```

Artımlı Öğrenme (Incremental Learning)



Başlangıçta model **özgün veri kümesi** ile eğitilir: MODEL 1.

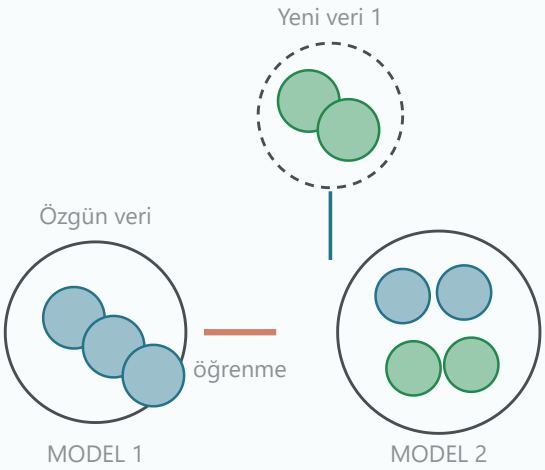
Artımlı Öğrenme (Incremental Learning)



Artımlı öğrenme (zaman)

Yeni veri 1 gelir. Tüm veriyle baştan eğitmek yerine model yalnızca yeni veriyle **güncellenir**.

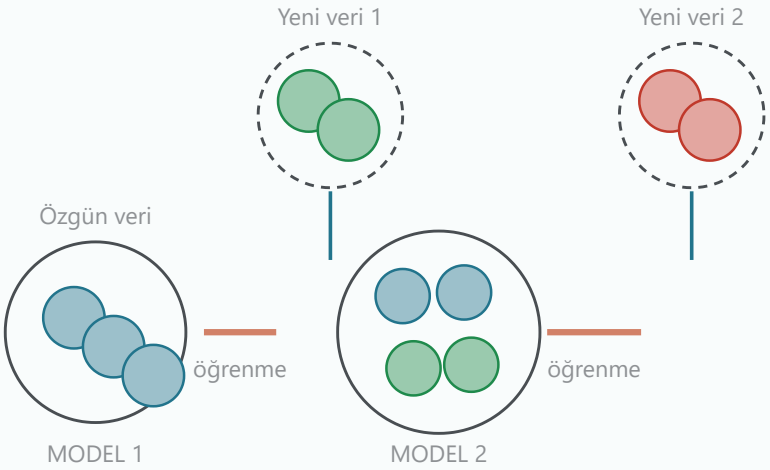
Artımlı Öğrenme (Incremental Learning)



Artımlı öğrenme (zaman)

MODEL 2 hem eski hem yeni verideki bilgiyi temsil eder; eski veri **yeniden işlenmez**.

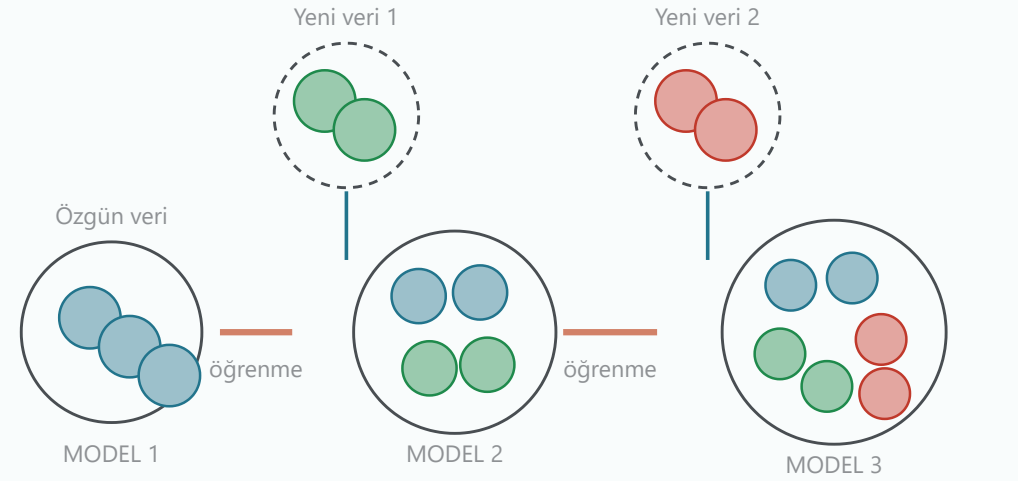
Artımlı Öğrenme (Incremental Learning)



Artımlı öğrenme (zaman)

Yeni veri 2 gelir; aynı güncelleme tekrarlanır.

Artımlı Öğrenme (Incremental Learning)



MODEL 3 üç veri kümesinin bilgisini taşır. Model hep **güncel**, maliyet yalnızca yeni veri kadardır.

Artımlı Güncelleme ve Zorlukları

Çalışan ortalama

Ortalama yeni öğeyle güncellenir; geçmişi saklamaya gerek yoktur:

$$\mu_n = \mu_{n-1} + (x_n - \mu_{n-1}) / n$$

Akış 4, 8, 6: $\mu = 4 \rightarrow 4 + (8 - 4)/2 = 6 \rightarrow 6 + (6 - 6)/3 =$ **6**.

Çevrim içi gradyan inişi

$$w \leftarrow w - \eta \cdot \nabla \ell(w; x_t, y_t)$$

Her örnek bir kez kullanılır; η: öğrenme oranı.

Zorluklar

- **Felaket boyutunda unutma** (catastrophic forgetting): yeni veriye uyum sağlarken eski bilgi kaybolur.
- **Kavram kayması** (concept drift): verinin dağılımı zamanla değişir; eski veri yanıltıcı olabilir.
- **Sıra etkisi**: veri hangi sırayla gelirse model ona göre şekillenir.
- **Gürültü**: tek bir aykırı örnek modeli bozabilir; küçük η ve kayan pencere yardımcı olur.

ÖZET

Problemler ve Sonuçlar

Problem	Çevrim içi algoritma	Sonuç	Not
Kayak kiralama	B − 1 gün kiralanır, sonra alınır	2 − 1/B	deterministikte en iyi; rastgele e/(e−1)
Sayfalama	LRU, FIFO, FWF	k	deterministikte en iyi
Sayfalama	LFU, LIFO	sınırsız	rekabetçi değil
Sayfalama (rastgele)	İşaretleme	$2H_k$	alt sınır H_k
Liste güncelleme	Öne taşı (MTF)	2	transpoz ve sıklık sayacı rekabetçi değil
Hareketli ortalama	Kuyruk + toplam	O(1)	bellek O(k)
Çevrim içi ortanca	İki heap	O(log n)	ortanca okuma O(1)
Rezervuar örnekleme	Algoritma R	O(1)	her öğe k/n olasılıkla

Üst beş satırdaki sayılar rekabet oranı; alttakiler öğe başına zamandır.

Özet

Konu	Hatırlanacaklar
Model	girdi parça parça gelir; karar hemen verilir ve geri alınamaz
Rekabet oranı	$ALG(\sigma) \le c \cdot OPT(\sigma) + \alpha$, her σ için; en kötü durum analizi, düşman modeli
Kayak kiralama	$OPT = \min(d, B)$; başa baş stratejisi $2 - 1/B$; daha iyi deterministik strateji yok
Sayfalama	FIFO, LRU k-rekabetçi; OPT en uzak geleceği çıkarır; FIFO'da Belady anomalisi
Liste güncelleme	maliyet = konum; MTF 2-rekabetçi (potansiyel fonksiyonu: ters çiftler)
Veri akışları	tek geçiş, az bellek: kayan pencere, iki heap, rezervuar, kayan hash
Artımlı öğrenme	model yeni veriyle güncellenir; unutma ve kavram kayması zorlukları

ÖZET

Sık Yapılan Hatalar

"Çevrim içi = internet" sanmak

Çevrim içi algoritma, girdinin **zamanla açığa çıktığı** modeldir; ağ bağlantısıyla ilgisi yoktur.

Oranı ortalama durumla karıştırmak

Rekabet oranı **en kötü** girdiye göredir; "genelde iyi çalışır" bir rekabet oranı değildir.

FIFO ile LRU'yu karıştırmak

FIFO'da isabet sırayı **değiştirmez**; LRU'da isabet sayfayı en yeni yapar.

Zorunlu hataları unutmak

İlk k farklı sayfa her politikada hata verir; OPT bile 0 hata yapamaz.

Rezervuarda yanlış aralık

j = rastgele[0, i] olmalı (i dahil, 0 tabanlı). `nextInt(i)` yazmak olasılıkları bozar.

Ortancada dengeyi unutmak

Yalnızca eklemek yetmez; boyut farkı 1'i aşınca tepe öbür heap'e taşınmalıdır.

ÖZET

Örnek Problemler

1. İstek dizisi **1 2 3 4 1 2 5 1 2 3 4 5**, k = 3 için OPT kaç sayfa hatası yapar?
2. Kayak kiralamada B = 20. Başa baş stratejisinin oranı nedir? 5. gün satın alan stratejinin en kötü oranı nedir?
3. A B C D listesinde MTF ile **D C D A** isteklerinin toplam maliyeti nedir?
4. Akış 2, 9, 4, 7 için çevrim içi ortancaları sırasıyla yazın.
5. Rezervuar örneklemede k = 2, akışın 0 tabanlı 9. öğesi rezervuara hangi olasılıkla girer? Akış 10 öğeyle biterse bu öğe sonda hangi olasılıkla rezervuardadır?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. FIFO sayfalamayı bir `Queue` ve bir `HashSet` ile O(1) zamanda yazın.
2. OPT'yi (Belady) Java ile yazın. Zaman karmaşıklığı nedir?
3. Kayak kiralamada "B/2. gün satın al" stratejisinin oranını bulun.
4. Çevrim içi ortancaya `sil(x)` eklemek neden zordur?
5. k = 1 rezervuarda 3 öğeli akışın her öğesinin 1/3 olasılıkla seçildiğini gösterin.
6. Son k öğenin **en büyüğünü** amortize O(1) zamanda bulun.

▶ **Cevaplar / ipuçları**

Kaynaklar

- A. Borodin ve R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.
- D. D. Sleator ve R. E. Tarjan, "Amortized Efficiency of List Update and Paging Rules", *Communications of the ACM* 28(2), 1985.
- L. A. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer", *IBM Systems Journal* 5(2), 1966.
- A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator ve N. E. Young, "Competitive Paging Algorithms", *Journal of Algorithms* 12(4), 1991.
- J. S. Vitter, "Random Sampling with a Reservoir", *ACM Transactions on Mathematical Software* 11(1), 1985.
- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, 2022 (Bölüm 27: Online Algorithms).

Son

Bölüm 8: Çevrim İçi Algoritmalar

Sonraki bölüm: Bölüm 9: Paralel Algoritmalar · Terim sözlüğü