Bölüm 5: Polinomlar ve Seyrek Matrisler

C ile Veri Yapıları ve Algoritmalar

Polinom ADT · Terim dizisi · padd · Bağlı liste ile çarpım · Üçlü gösterim · transpose · fast_transpose · Seyrek çarpım · CSR

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Gösterim seçmek

Polinom için katsayı dizisi ile terim dizisini bellek ve zaman açısından karşılaştırmak.

Birleştirme (merge) mantığı

Sıralı iki terim dizisini padd ile O(n + m)'de toplamak ve analiz etmek.

Bağlı liste ile çarpım

Polinomları bağlı listeyle çarpıp aynı üslü terimleri birleştirmek.

Seyrek matris

Matrisi ⟨row, col, value⟩ üçlüleriyle tutup ne zaman kazandırdığını söylemek.

Transpoz almak

transpose ve fast_transpose'u izleyip O(cols·terms) ile O(cols + terms)'ü karşılaştırmak.

Çarpmak

Seyrek matris çarpımını transpoz yardımıyla kurmak; CSR gösterimini tanımak.

Polinom: ⟨üs, katsayı⟩ çiftlerinden oluşan küme

$A(x) = a_n x^n + \ldots + a_1 x + a_0$; her terim $a_i x^i$ bir **katsayı** ve bir **üs**tür. Derece: sıfır olmayan en büyük üs.

İşlem	Anlamı
`Zero()`, `IsZero(p)`	sıfır polinomu kur / sına
`Coef(p, e)`	e üslü terimin katsayısı (yoksa 0)
`LeadExp(p)`	en büyük üs
`Attach(p, c, e)`, `Remove(p, e)`	terim ekle / çıkar
`SingleMult(p, c, e)`	$p \cdot c\, x^e$
`Add(p1, p2)`, `Mult(p1, p2)`	toplam, çarpım

Toplama

$A(x) + B(x) = \Sigma\, (a_i + b_i) \cdot x^i$: aynı üslü katsayılar toplanır.

Çarpma

$A(x) \cdot B(x) = \Sigma\, (a_i x^i \cdot \Sigma\, b_j x^j)$: üsler toplanır, katsayılar çarpılır.

1. yöntem (kötü): tüm katsayıları tut

```c
#define MAX_DEGREE 101   /* azami derece + 1 */
typedef struct {
  int degree;               /* en büyük üs */
  float coef[MAX_DEGREE];   /* coef[i]: x^i katsayısı */
} polynomial;
polynomial P;
```

Artısı

`degree` saklanınca derece her seferinde hesaplanmaz; Coef ve toplama çok basittir.

Eksisi

Katsayısı 0 olan terimler de tutulur: seyrek polinomda büyük **bellek israfı**.



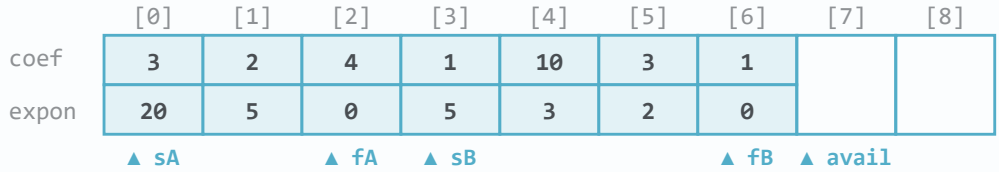
P(x) = $5x^{100}$ için coef[0..99] boşuna sıfır tutar: 101 float, 1 anlamlı

2. yöntem (daha iyi): yalnız sıfır olmayan terimler

```
#define MAX_TERMS 101   /* azami terim sayısı */
typedef struct {
  float coef;           /* katsayı */
  int expon;            /* üs */
} polynomial;
polynomial P[MAX_TERMS]; /* tüm polinomlar burada */
int avail = 0;          /* ilk boş konum */
```

Terimler **büyük üsten küçüğe** sıralı tutulur; her polinom P içinde bir `[start, finish]` aralığıdır.

> **Hocadan** *"Merge mantığı, sıralı yapılar üzerinde daha etkin algoritmalar üretmemize olanak sağlar."*



$$A(x) = 3x^{20} + 2x^{5} + 4 \cdot B(x) = x^{5} + 10x^{3} + 3x^{2} + 1$$

İki işaretçiyle birleştirerek toplamak

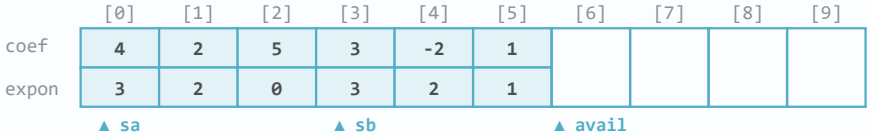
- **Üsler eşit**: katsayılar toplanır; toplam ≠ 0 ise sonuca eklenir; **iki işaretçi** de ilerler.
- **A'nınki büyük**: "A'daki üsse sahip bir terim B'de yoktur" — B'nin kalan terimleri daha küçüktür. A'nın terimi eklenir, yalnız A ilerler.
- **B'ninki büyük**: simetrik; B'nin terimi eklenir.
- Birinde terim kalmayınca diğerinin **kalanı olduğu gibi** kopyalanır.

```
#define COMPARE(x, y) \
  ((x) < (y) ? -1 : (y) < (x) ? 1 : 0)

void attach(float coef, int expon) {
  if (avail >= MAX_TERMS) {
    fprintf(stderr, "Dizi tasmasi!\n"); exit(1);
  }
  P[avail].coef = coef;
  P[avail++].expon = expon;
}
#define EK(k) attach(P[k].coef, P[k].expon)
```

`attach(int coef, …)` yazılırsa float katsayı **int'e kırpılır** (2.5 → 2). Parametre `float` olmalı.

padd: A + B terim terim



A = $4x^3 + 2x^2 + 5$ → P[0..2], B = $3x^3 - 2x^2 + x$ → P[3..5]. Sonuç D, **avail = 6**'dan başlar: *sd = 6.

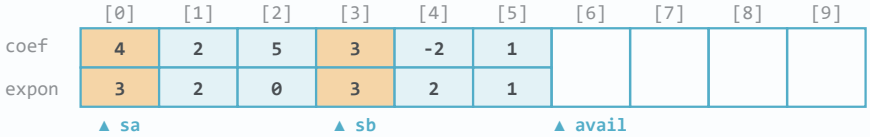
Adım 1 / 14

```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
▶    float c; *sd = avail;                /* D'nin başı */
3    while (sa <= fa && sb <= fb)
4      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
7      case  0: c = P[sa].coef + P[sb].coef;
8        if (c) attach(c, P[sa].expon); /* 0: ekleme */
9        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 0` `sb = 3` `avail = 6`

padd: A + B terim terim



COMPARE(3, 3) = **0**: üsler eşit.

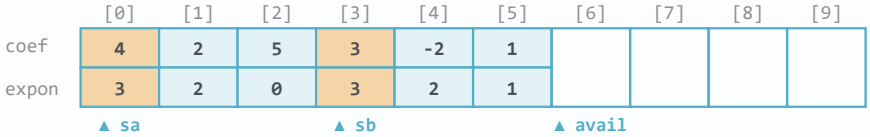
Adım 2 / 14

```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
▶    while (sa <= fa && sb <= fb)
▶      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
7      case  0: c = P[sa].coef + P[sb].coef;
8        if (c) attach(c, P[sa].expon); /* 0: ekleme */
9        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 0` `sb = 3` `avail = 6`

padd: A + B terim terim



c = 4 + (3) = **7**.

Adım 3 / 14

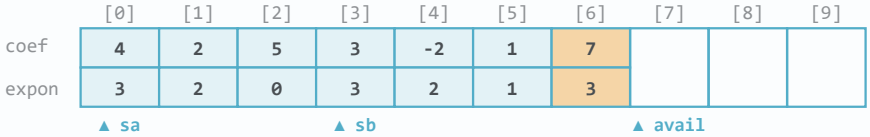
```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
3    while (sa <= fa && sb <= fb)
4      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
▶      case  0: c = P[sa].coef + P[sb].coef;
8        if (c) attach(c, P[sa].expon); /* 0: ekleme */
9        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 0` `sb = 3` `avail = 6`

`c = 7`

padd: A + B terim terim



c ≠ 0 → `attach(7, 3)` : P[6] = $7x^3$.

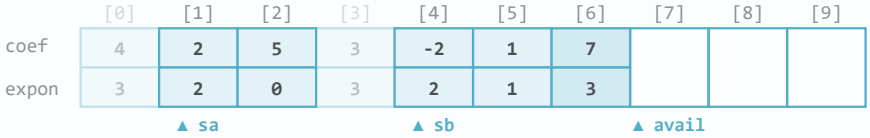
Adım 4 / 14

```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
3    while (sa <= fa && sb <= fb)
4      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
7      case  0: c = P[sa].coef + P[sb].coef;
▶        if (c) attach(c, P[sa].expon); /* 0: ekleme */
9        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 0` `sb = 3` `avail = 7` `c = 7`

padd: A + B terim terim



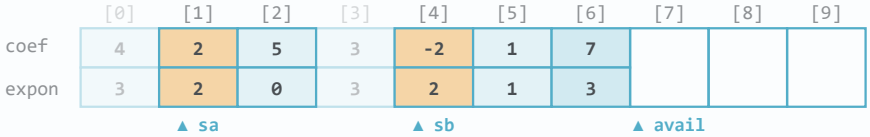
İki işaretçi de ilerler.


```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
3    while (sa <= fa && sb <= fb)
4      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
7      case  0: c = P[sa].coef + P[sb].coef;
8        if (c) attach(c, P[sa].expon); /* 0: ekleme */
▶        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 1` `sb = 4` `avail = 7` `c = 7`

padd: A + B terim terim



COMPARE(2, 2) = **0**: üsler eşit.

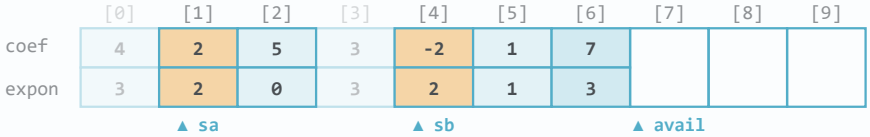
Adım 6 / 14

```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
▶    while (sa <= fa && sb <= fb)
▶      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
7      case  0: c = P[sa].coef + P[sb].coef;
8        if (c) attach(c, P[sa].expon); /* 0: ekleme */
9        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 1` `sb = 4` `avail = 7`

padd: A + B terim terim



c = 2 + (-2) = **0**.

Adım 7 / 14

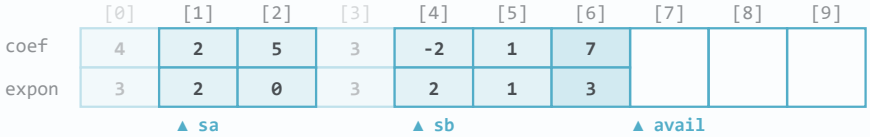
```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
3    while (sa <= fa && sb <= fb)
4      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
▶      case  0: c = P[sa].coef + P[sb].coef;
8        if (c) attach(c, P[sa].expon); /* 0: ekleme */
9        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 1` `sb = 4` `avail = 7`

`c = 0`

padd: A + B terim terim



c = 0 → terim sonuca **eklenmez** (x^2 birbirini götürdü).

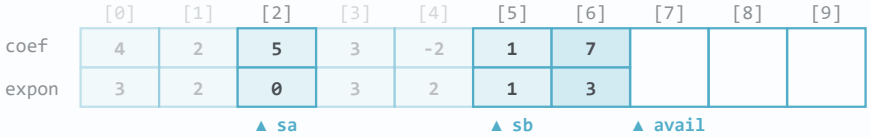
Adım 8 / 14

```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
3    while (sa <= fa && sb <= fb)
4      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
7      case  0: c = P[sa].coef + P[sb].coef;
▶        if (c) attach(c, P[sa].expon); /* 0: ekleme */
9        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 1` `sb = 4` `avail = 7` `c = 0`

padd: A + B terim terim



İki işaretçi de ilerler.

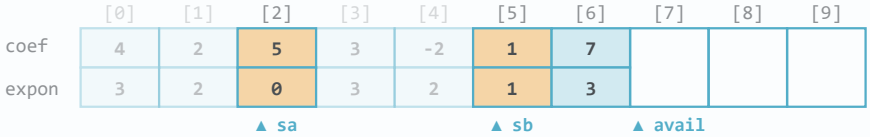
Adım 9 / 14

```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
3    while (sa <= fa && sb <= fb)
4      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
7      case  0: c = P[sa].coef + P[sb].coef;
8        if (c) attach(c, P[sa].expon); /* 0: ekleme */
▶        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 2` `sb = 5` `avail = 7` `c = 0`

padd: A + B terim terim



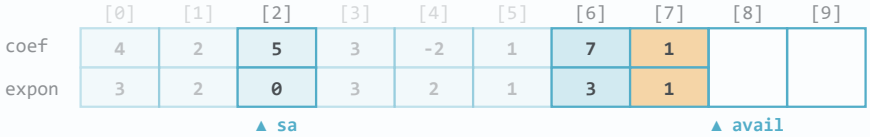
COMPARE(0, 1) = **-1**: B'nin üssü büyük.


```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
▶    while (sa <= fa && sb <= fb)
▶      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
7      case  0: c = P[sa].coef + P[sb].coef;
8        if (c) attach(c, P[sa].expon); /* 0: ekleme */
9        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 2` `sb = 5` `avail = 7`

padd: A + B terim terim



B'nin terimi (x) D'ye eklenir; **yalnız sb** ilerler.

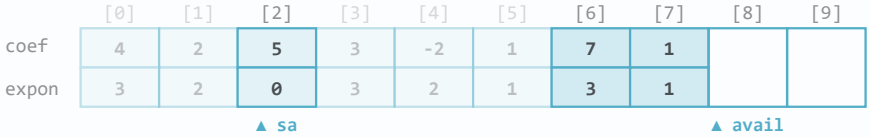
Adım 11 / 14

```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
3    while (sa <= fa && sb <= fb)
4      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
7      case  0: c = P[sa].coef + P[sb].coef;
8        if (c) attach(c, P[sa].expon); /* 0: ekleme */
9        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 2` `sb = 6` `avail = 8`

padd: A + B terim terim



sb = 6 > fb = 5 → B bitti, döngüden çıkılır.

Adım 12 / 14

```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
▶    while (sa <= fa && sb <= fb)
4      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
7      case  0: c = P[sa].coef + P[sb].coef;
8        if (c) attach(c, P[sa].expon); /* 0: ekleme */
9        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 2` `sb = 6` `avail = 8`

padd: A + B terim terim

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
coef	4	2	5	3	-2	1	7	1	5	
expon	3	2	0	3	2	1	3	1	0	

▲ avail

A'da kalan 5 olduğu gibi kopyalanır.

Adım 13 / 14

```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
3    while (sa <= fa && sb <= fb)
4      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
7      case  0: c = P[sa].coef + P[sb].coef;
8        if (c) attach(c, P[sa].expon); /* 0: ekleme */
9        sa++; sb++;
10     }
▶    for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
13   *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 3` `sb = 6` `avail = 9`

padd: A + B terim terim

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
coef	4	2	5	3	-2	1	7	1	5	
expon	3	2	0	3	2	1	3	1	0	

▲ avail

***fd = 8**. D(x) = P[6..8] = **$7x^3 + x + 5$**.

Adım 14 / 14

```
1  void padd(int sa,int fa,int sb,int fb,int*sd,int*fd){
2    float c; *sd = avail;                /* D'nin başı */
3    while (sa <= fa && sb <= fb)
4      switch (COMPARE(P[sa].expon, P[sb].expon)) {
5      case -1: EK(sb); sb++; break;      /* B büyük */
6      case  1: EK(sa); sa++; break;      /* A büyük */
7      case  0: c = P[sa].coef + P[sb].coef;
8        if (c) attach(c, P[sa].expon); /* 0: ekleme */
9        sa++; sb++;
10     }
11   for (; sa <= fa; sa++) EK(sa);       /* A'da kalan */
12   for (; sb <= fb; sb++) EK(sb);       /* B'de kalan */
▶    *fd = avail - 1;
14 }
```

DEĞİŞKENLER

`sa = 3` `sb = 6` `avail = 9` `sd = 6` `fd = 8`

padd'in karmaşıklığı

- A'da n, B'de m terim: her döngü turunda **en az bir** işaretçi ilerler → en çok n + m tur.
- Kalan kopyalama döngüleri de toplamda n + m'yi aşmaz → **O(n + m)**.
- En iyi durum: tüm üsler ortak (n = m); her tur case 0, iki işaretçi birlikte ilerler → Ω(n).
- Sıralı olmasaydı: her terim için sonuçta arama/araya ekleme (kaydırma) → **O(n^2)** düzeyine çıkardı.

Hocadan *"İstenen O(n+m) ise, daha iyisi yapılabiliyorsa ne âlâ!"*

Ortak P dizisinin bedeli

Sonuçlar hep `avail`'dan eklenir; artık polinomların yeri geri kazanılmaz. P dolunca **sıkıştırma** (compaction) gerekir — ya da bağlı liste.

ÖDEV 2: pmult, n polinomu toplamak ve çarpmak

1. İki polinomu (P dizisi gösterimiyle) çarpan `pmult` yordamını yazın ve performans analizini yapın.
2. n tane polinomu nasıl toplardınız? Veri yapısını kurun.
3. n tane polinomu nasıl çarpardınız? Veri yapısını kurun.

▶ **Çözüm**

Polinomu bağlı listeyle tutmak

```
typedef struct pol_oge *pol_gosterge;
typedef struct pol_oge {
  int katsayi;
  int us;
  pol_gosterge bag;
} pol_oge;
```



$$a = 3x^{14} + 2x^{8} + 1$$

- Yalnız sıfır olmayan terimler, üsse göre **azalan** sırada.
- Terim sayısı önceden bilinmez; ortak dizi, `avail` ve sıkıştırma derdi yok.
- Araya ekleme/silme işaretçi değiştirerek **O(1)** (yeri bulunduktan sonra).

Toplama yine birleştirme

Liste padd'i de iki işaretçiyle yürür: **O(n + m)**; sonuç düğümleri `malloc` ile gelir, iş bitince `free` edilmeli (Horowitz 4.4).

Yardımcılar: YeniTerimEkle ve SonrakiTerimiSil

```
/* terim oluştur; Onceki varsa peşine ekle */
pol_gosterge YeniTerimEkle(int Us, int Katsayi,
                           pol_gosterge Onceki) {
  pol_gosterge p = malloc(sizeof *p);
  if (!p) { printf("Bellek yetmiyor!\n"); exit(1); }
  p->us = Us;  p->katsayi = Katsayi;
  if (Onceki) {              /* araya ya da sona */
    p->bag = Onceki->bag;    /* önce yeni düğüm */
    Onceki->bag = p;         /* sonra önceki */
  } else p->bag = NULL;      /* tek başına */
  return p;
}
```

```
/* Onceki'den sonrakini sil */
void SonrakiTerimiSil(
        pol_gosterge Onceki) {
  pol_gosterge Silinecek =
        Onceki->bag;
  if (!Silinecek) return;
  Onceki->bag =
        Silinecek->bag;
  free(Silinecek);
}
```

Adres önce `Silinecek`'de tutulur: bağ güncellendikten sonra `free` edilebilsin; "bellekteki kalıntı temizlenmelidir".

Liste olarak verilen iki polinomu çarpmak

```
pol_gosterge yerlestir(pol_gosterge r,
                       int us, int kat) {
  while (r->bag && r->bag->us > us) r = r->bag;
  if (!r->bag || r->bag->us < us)
    YeniTerimEkle(us, kat, r);          /* yeni üs */
  else if ((r->bag->katsayi += kat) == 0)
    SonrakiTerimiSil(r);                /* 0 oldu */
  return r;
}
```

- A'nın her terimi B'nin her terimiyle çarpılır: **üsler toplanır, katsayılar çarpılır**.
- Sonuç listesi sıralı kalmalı; aynı üslü terim varsa katsayı **güncellenir**, 0 olursa terim **silinir**.
- `bas` : p'nin bir sonraki terimiyle çarpımlar, bir öncekinin `qBas` ile çarpımının **önünden** başlayamaz → arama oradan başlar.
- Okunaklılık için iç döngü gövdesi `yerlestir` yordamına ayrıldı.

PolCarp: (x + 1)·(x − 1)



A = x + 1, B = x − 1 çarpılacak (beklenen $x^2 - 1$). Sonuç listesi boş.

Adım 1 / 14

```
1  pol_gosterge PolCarp(pol_gosterge pBas,
2                       pol_gosterge qBas) {
▶    pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
4    for (p = pBas; p; p = p->bag) {
5      r = bas;               /* aramaya buradan başla */
6      for (q = qBas; q; q = q->bag) {
7        int us = p->us + q->us;
8        int kat = p->katsayi * q->katsayi;
9        if (rBas) r = yerlestir(r, us, kat);
10       else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
11       if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
14   return rBas;
15 }
```

DEĞİŞKENLER

`p = ?` `q = ?` `r = NULL` `bas = NULL` `rBas = NULL`

PolCarp: (x + 1)·(x − 1)

katsayi us bag

rBas = NULL (boş liste)

p: x q: —

p = x. **r = bas** = NULL (henüz liste yok).

Adım 2 / 14

```
1  pol_gosterge PolCarp(pol_gosterge pBas,
2                       pol_gosterge qBas) {
3    pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
▶    for (p = pBas; p; p = p->bag) {
▶      r = bas;               /* aramaya buradan başla */
6      for (q = qBas; q; q = q->bag) {
7        int us = p->us + q->us;
8        int kat = p->katsayi * q->katsayi;
9        if (rBas) r = yerlestir(r, us, kat);
10       else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
11       if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
14   return rBas;
15 }
```

DEĞİŞKENLER

p = &(1,1) | r = NULL | bas = NULL | rBas = NULL

PolCarp: (x + 1)·(x − 1)

katsayi us bag

rBas = NULL (boş liste)

p: x q: x

us = 1 + 1 = **2**, kat = 1·(1) = **1**.

Adım 3 / 14

```
1  pol_gosterge PolCarp(pol_gosterge pBas,
2                       pol_gosterge qBas) {
3    pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
4    for (p = pBas; p; p = p->bag) {
5      r = bas;               /* aramaya buradan başla */
6      for (q = qBas; q; q = q->bag) {
▶        int us = p->us + q->us;
▶        int kat = p->katsayi * q->katsayi;
9        if (rBas) r = yerlestir(r, us, kat);
10       else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
11       if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
14   return rBas;
15 }
```

DEĞİŞKENLER

p = &(1,1) | q = &(1,1) | us = 2 | kat = 1 | r = NULL | bas = NULL | rBas = NULL

PolCarp: (x + 1)·(x − 1)



p: x q: x

Liste boş → ilk düğüm x^2; **rBas = r = bas**.


```
1  pol_gosterge PolCarp(pol_gosterge pBas,
2                       pol_gosterge qBas) {
3    pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
4    for (p = pBas; p; p = p->bag) {
5      r = bas;                /* aramaya buradan başla */
6      for (q = qBas; q; q = q->bag) {
7        int us = p->us + q->us;
8        int kat = p->katsayi * q->katsayi;
9        if (rBas) r = yerlestir(r, us, kat);
▶        else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
11       if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
14   return rBas;
15 }
```

DEĞİŞKENLER

- p = &(1,1)
- q = &(1,1)
- us = 2
- kat = 1
- r = &(1,2)
- bas = &(1,2)
- rBas = &(1,2)

PolCarp: (x + 1)·(x − 1)



p: x q: x

q ilk terimdi → **bas = r** = &(1,2): p'nin sonraki terimi buradan aramaya başlayacak.

Adım 5 / 14

```
 1 pol_gosterge PolCarp(pol_gosterge pBas,
 2                      pol_gosterge qBas) {
 3   pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
 4   for (p = pBas; p; p = p->bag) {
 5     r = bas;               /* aramaya buradan başla */
 6     for (q = qBas; q; q = q->bag) {
 7       int us = p->us + q->us;
 8       int kat = p->katsayi * q->katsayi;
 9       if (rBas) r = yerlestir(r, us, kat);
10       else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
▶        if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
14   return rBas;
15 }
```

DEĞİŞKENLER

`p = &(1,1)` `q = &(1,1)` `us = 2` `kat = 1` `r = &(1,2)` `bas = &(1,2)` `rBas = &(1,2)`

PolCarp: (x + 1)·(x − 1)



p: x q: −1

us = 1 + 0 = **1**, kat = 1·(-1) = **-1**.

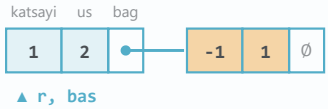
Adım 6 / 14

```
1  pol_gosterge PolCarp(pol_gosterge pBas,
2                       pol_gosterge qBas) {
3    pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
4    for (p = pBas; p; p = p->bag) {
5      r = bas;                /* aramaya buradan başla */
6      for (q = qBas; q; q = q->bag) {
▶        int us = p->us + q->us;
▶        int kat = p->katsayi * q->katsayi;
9        if (rBas) r = yerlestir(r, us, kat);
10       else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
11       if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
14   return rBas;
15 }
```

DEĞİŞKENLER

p = &(1,1) | q = &(-1,0) | us = 1 | kat = -1 | r = &(1,2) | bas = &(1,2) | rBas = &(1,2)

PolCarp: (x + 1)·(x − 1)



p: x q: −1

`yerlestir` : r'den sonra 1 üslü terim yok → **yeni düğüm** −x r'nin peşine eklenir.

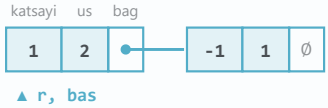
Adım 7 / 14

```
1  pol_gosterge PolCarp(pol_gosterge pBas,
2                       pol_gosterge qBas) {
3    pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
4    for (p = pBas; p; p = p->bag) {
5      r = bas;                /* aramaya buradan başla */
6      for (q = qBas; q; q = q->bag) {
7        int us = p->us + q->us;
8        int kat = p->katsayi * q->katsayi;
▶        if (rBas) r = yerlestir(r, us, kat);
10       else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
11       if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
14   return rBas;
15 }
```

DEĞİŞKENLER

`p = &(1,1)` `q = &(−1,0)` `us = 1` `kat = −1` `r = &(1,2)` `bas = &(1,2)` `rBas = &(1,2)`

PolCarp: (x + 1)·(x − 1)



p: 1 q: —

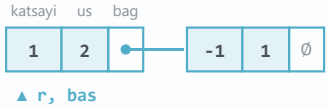
p = 1. **r = bas**: arama baştan değil, &(1,2) düğümünden başlar.


```
 1 pol_gosterge PolCarp(pol_gosterge pBas,
 2                      pol_gosterge qBas) {
 3   pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
▶  for (p = pBas; p; p = p->bag) {
▶    r = bas;               /* aramaya buradan başla */
 6    for (q = qBas; q; q = q->bag) {
 7      int us = p->us + q->us;
 8      int kat = p->katsayi * q->katsayi;
 9      if (rBas) r = yerlestir(r, us, kat);
10      else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
11      if (q == qBas) bas = r;  /* sonraki p için */
12    }
13  }
14  return rBas;
15 }
```

DEĞİŞKENLER

p = &(1,0) | r = &(1,2) | bas = &(1,2) | rBas = &(1,2)

PolCarp: (x + 1)·(x − 1)



p: 1 q: x

us = 0 + 1 = **1**, kat = 1·(1) = **1**.


```
1  pol_gosterge PolCarp(pol_gosterge pBas,
2                       pol_gosterge qBas) {
3    pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
4    for (p = pBas; p; p = p->bag) {
5      r = bas;                /* aramaya buradan başla */
6      for (q = qBas; q; q = q->bag) {
▶        int us = p->us + q->us;
▶        int kat = p->katsayi * q->katsayi;
9        if (rBas) r = yerlestir(r, us, kat);
10       else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
11       if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
14   return rBas;
15 }
```

DEĞİŞKENLER

p = &(1,0) | q = &(1,1) | us = 1 | kat = 1 | r = &(1,2) | bas = &(1,2) | rBas = &(1,2)

PolCarp: (x + 1)·(x − 1)



p: 1 q: x

`yerlestir` : 1 üslü terim zaten var; katsayı −1 + 1 = **0** → `SonrakiTerimiSil(r)` ile silinir.

Adım 10 / 14

```
1  pol_gosterge PolCarp(pol_gosterge pBas,
2                       pol_gosterge qBas) {
3    pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
4    for (p = pBas; p; p = p->bag) {
5      r = bas;               /* aramaya buradan başla */
6      for (q = qBas; q; q = q->bag) {
7        int us = p->us + q->us;
8        int kat = p->katsayi * q->katsayi;
▶        if (rBas) r = yerlestir(r, us, kat);
10       else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
11       if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
14   return rBas;
15 }
```

DEĞİŞKENLER

`p = &(1,0)` `q = &(1,1)` `us = 1` `kat = 1` `r = &(1,2)` `bas = &(1,2)` `rBas = &(1,2)`

PolCarp: (x + 1)·(x − 1)



p: 1 q: x

q ilk terimdi → **bas = r** = &(1,2): p'nin sonraki terimi buradan aramaya başlayacak.

Adım 11 / 14

```
1  pol_gosterge PolCarp(pol_gosterge pBas,
2                       pol_gosterge qBas) {
3    pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
4    for (p = pBas; p; p = p->bag) {
5      r = bas;               /* aramaya buradan başla */
6      for (q = qBas; q; q = q->bag) {
7        int us = p->us + q->us;
8        int kat = p->katsayi * q->katsayi;
9        if (rBas) r = yerlestir(r, us, kat);
10       else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
▶        if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
14   return rBas;
15 }
```

DEĞİŞKENLER

- p = &(1,0)
- q = &(1,1)
- us = 1
- kat = 1
- r = &(1,2)
- bas = &(1,2)
- rBas = &(1,2)

PolCarp: (x + 1)·(x − 1)



p: 1 q: −1

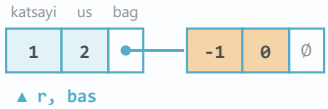
us = 0 + 0 = **0**, kat = 1·(-1) = **-1**.


```
1  pol_gosterge PolCarp(pol_gosterge pBas,
2                       pol_gosterge qBas) {
3    pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
4    for (p = pBas; p; p = p->bag) {
5      r = bas;                /* aramaya buradan başla */
6      for (q = qBas; q; q = q->bag) {
▶        int us = p->us + q->us;
▶        int kat = p->katsayi * q->katsayi;
9        if (rBas) r = yerlestir(r, us, kat);
10       else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
11       if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
14   return rBas;
15 }
```

DEĞİŞKENLER

p = &(1,0) | q = &(-1,0) | us = 0 | kat = -1 | r = &(1,2) | bas = &(1,2) | rBas = &(1,2)

PolCarp: (x + 1)·(x − 1)



p: 1 q: −1

`yerlestir` : r'den sonra 0 üslü terim yok → **yeni düğüm** −1 r'nin peşine eklenir.

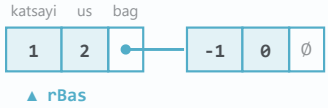
Adım 13 / 14

```
1  pol_gosterge PolCarp(pol_gosterge pBas,
2                       pol_gosterge qBas) {
3    pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
4    for (p = pBas; p; p = p->bag) {
5      r = bas;                /* aramaya buradan başla */
6      for (q = qBas; q; q = q->bag) {
7        int us = p->us + q->us;
8        int kat = p->katsayi * q->katsayi;
▶        if (rBas) r = yerlestir(r, us, kat);
10       else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
11       if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
14   return rBas;
15 }
```

DEĞİŞKENLER

`p = &(1,0)` `q = &(-1,0)` `us = 0` `kat = -1` `r = &(1,2)` `bas = &(1,2)` `rBas = &(1,2)`

PolCarp: (x + 1)·(x − 1)



$(x + 1)(x - 1) = x^2 - 1$

Sonuç: **$x^2 - 1$**. n·m = 4 çarpım yapıldı; aynı üslü −x ve +x birleşip silindi.

Adım 14 / 14

```
1  pol_gosterge PolCarp(pol_gosterge pBas,
2                       pol_gosterge qBas) {
3    pol_gosterge p, q, r = NULL, bas = NULL, rBas=NULL;
4    for (p = pBas; p; p = p->bag) {
5      r = bas;               /* aramaya buradan başla */
6      for (q = qBas; q; q = q->bag) {
7        int us = p->us + q->us;
8        int kat = p->katsayi * q->katsayi;
9        if (rBas) r = yerlestir(r, us, kat);
10       else rBas=r=bas=YeniTerimEkle(us, kat, NULL);
11       if (q == qBas) bas = r;  /* sonraki p için */
12     }
13   }
▶    return rBas;
15 }
```

DEĞİŞKENLER

`rBas = &(1,2)`

Problemin diğer çözümleri (A: n, B: m terim)

	yöntem	karmaşıklık
1	Hepsi çarpılır, her gelen sona eklenir; sonra sıralanır ve birleştirilir	$O(n^2m^2)$
2	Hepsi çarpılır; her seferinde baştan taranıp uygun konuma eklenir	$O(n^2m^2)$
3	A'nın her terimiyle çarpım için bir liste kurulur, her adımda sonuçla birleştirilir (merge)	$O(n^2m)$
4	3. yöntem, küçük polinom dışta	$O(\min(n,m)^2 \cdot \max(n,m))$
5	A'nın her terimi için ayrı liste; en sonda hepsi (ikişer ikişer) birleştirilir	$O(m \cdot n \cdot \lg n)$
6	5. yöntem, küçük polinom dışta	$O(m \cdot n \cdot \lg \min(m,n))$
7	Dizi açılır, n. indis n. üssü tutar	—

Hocadan *"n. indisin n. üssü tuttuğu bir dizi mi? ASLA!"*

Seyrek matris: öğelerin çoğu sıfır

	0	1	2	3	4	5
0	15	0	0	22	0	-15
1	0	11	3	0	0	0
2	0	0	0	-6	0	0
3	0	0	0	0	0	0
4	91	0	0	0	0	0
5	0	0	28	0	0	0

36 hücreden yalnız 8'i dolu

Kesin tanımı zordur; 2B dizide tutmak çok sayıda sıfır için **bellek kaybıdır**. Çözüm: her dolu öğeyi ⟨**row, col, value**⟩ üçlüsüyle tutmak.

```
#define MAX_TERMS 101
#define MAX_COL 50   /* azami sütun */
typedef struct {
  int row, col, value;   /* satır, sütun, değer */
} term;
term M[MAX_TERMS];
```

M[0] başlıktır: satır sayısı, sütun sayısı, dolu öğe sayısı. Üçlüler **önce satıra, sonra sütuna** göre sıralıdır.

Üçlü tablo ve ne zaman kazandırır?

	row	col	value
M[0]	6	6	8
M[1]	0	0	15
M[2]	0	3	22
M[3]	0	5	-15
M[4]	1	1	11
M[5]	1	2	3
M[6]	2	3	-6
M[7]	4	0	91
M[8]	5	2	28

Bellek

rows·cols tam sayı yerine 3·(t + 1). **t ≈ rows·cols/3** olunca eşitlenir; yani matrisin 3'te 1'i dolu olduğunda bellek maliyeti aynıdır. Daha doluysa bu yapı tercih edilmez.

Bu örnekte

36 tam sayı yerine 3·9 = 27 tam sayı.

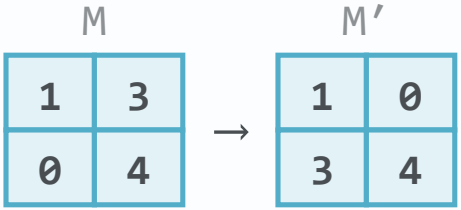
ADT işlemleri

Create(max_row, max_col), Transpose(a), Add(a, b), Multiply(a, b) — Horowitz ADT 2.3. Başka gösterimler de var (bağlı listeli, CSR…).

TRANSPOZ ALMA

Transpoz: M[i][j] → M'[j][i]

Satırları sütun olarak yazarız:

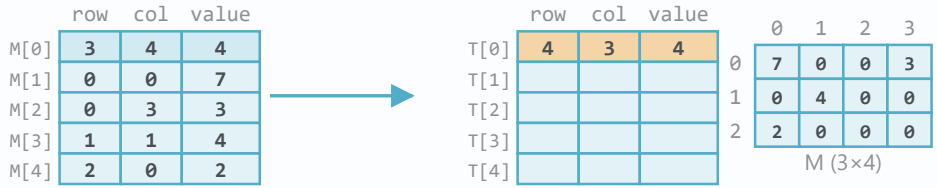


- Üçlülerde basitçe row ile col'u değiştirmek yetmez: sonuç **satır sırasını bozar**.
- Değiştirip sonra sıralamak ya da her öğeyi yerine **kaydırarak** eklemek pahalıdır.
- Çözüm: transpozun satırları = M'nin sütunları; **sütun sütun** taranıp sırayla üretilir.

Hocadan *"Kaydırma, taşıma kullanılıyorsa veri yapısının hakkı verilmiyor demektir!"*

`transpose` iç döngüsünü `j = 0`'dan başlatmak başlık M[0]'ı da tarar. (M[0].col = sütun sayısı hiçbir i'ye eşit olmadığından sonuç bozulmuyor, ama gereksiz.) Doğrusu `j = 1`.

transpose: her sütun için tüm terimleri tara



Başlık: T 4×3 olur (satır ↔ sütun), yine **4 terim**.

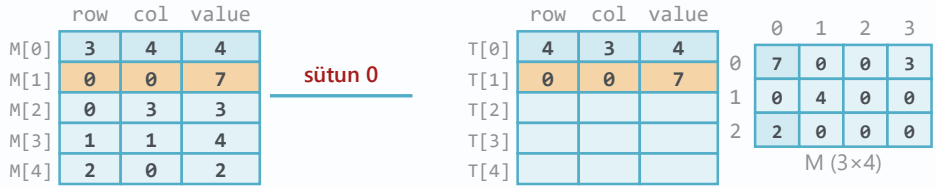
Adım 1 / 18

```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
▶    T[0].row = M[0].col;  T[0].col = M[0].row;
▶    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
9          if (M[j].col == i) {
10           T[cur].row = M[j].col; T[cur].col = M[j].row;
11           T[cur].value = M[j].value;  cur++;
12         }
13   }
14 }
```

DEĞİŞKENLER

n = **4**

transpose: her sütun için tüm terimleri tara



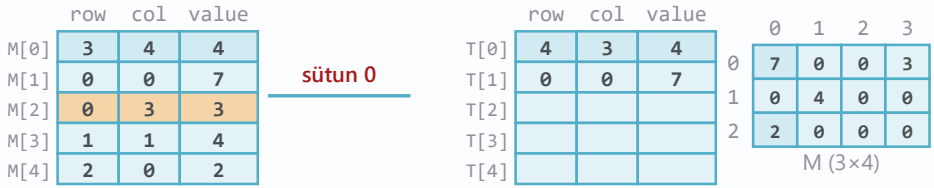
Sütun 0: M[1].col = 0 **eşit** → T[1] = (0, 0, 7), cur = 2.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶          if (M[j].col == i) {
▶            T[cur].row = M[j].col; T[cur].col = M[j].row;
▶            T[cur].value = M[j].value;  cur++;
12         }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 2` `i = 0` `j = 1` `M[j].col = 0`

transpose: her sütun için tüm terimleri tara



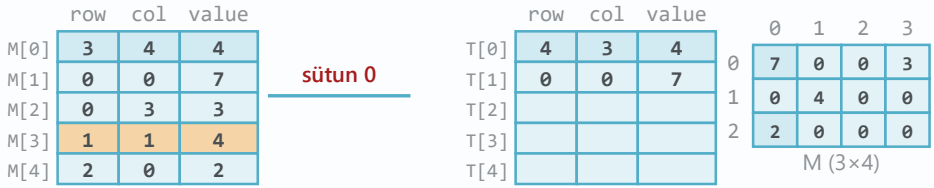
Sütun 0: M[2].col = 3 ≠ 0 → atla.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶         if (M[j].col == i) {
10          T[cur].row = M[j].col; T[cur].col = M[j].row;
11          T[cur].value = M[j].value;  cur++;
12        }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 2` `i = 0` `j = 2` `M[j].col = 3`

transpose: her sütun için tüm terimleri tara



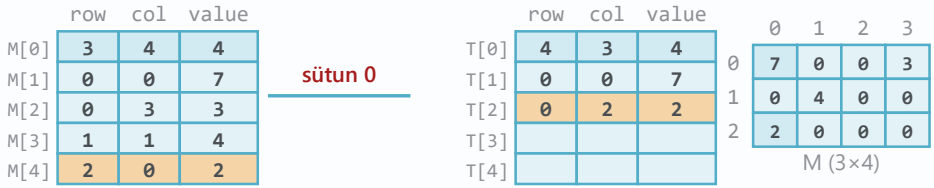
Sütun 0: M[3].col = 1 ≠ 0 → atla.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶         if (M[j].col == i) {
10          T[cur].row = M[j].col; T[cur].col = M[j].row;
11          T[cur].value = M[j].value;  cur++;
12        }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 2` `i = 0` `j = 3` `M[j].col = 1`

transpose: her sütun için tüm terimleri tara



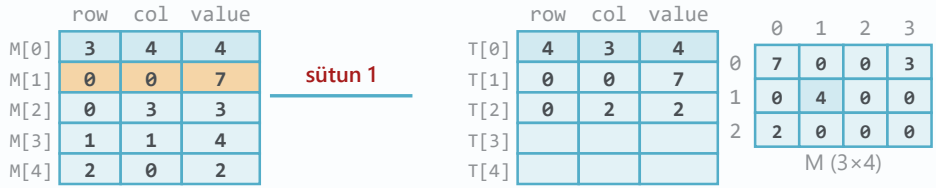
Sütun 0: M[4].col = 0 **eşit** → T[2] = (0, 2, 2), cur = 3.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶          if (M[j].col == i) {
▶            T[cur].row = M[j].col; T[cur].col = M[j].row;
▶            T[cur].value = M[j].value;  cur++;
12         }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 3` `i = 0` `j = 4` `M[j].col = 0`

transpose: her sütun için tüm terimleri tara



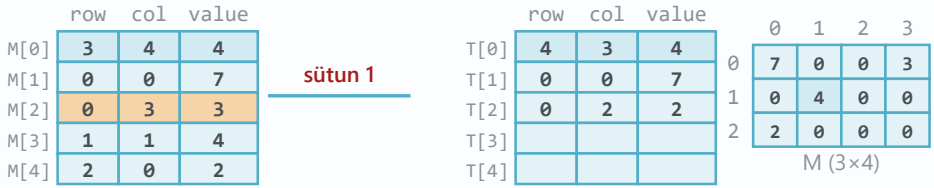
Sütun 1: M[1].col = 0 ≠ 1 → atla.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶         if (M[j].col == i) {
10          T[cur].row = M[j].col; T[cur].col = M[j].row;
11          T[cur].value = M[j].value;  cur++;
12        }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 3` `i = 1` `j = 1` `M[j].col = 0`

transpose: her sütun için tüm terimleri tara



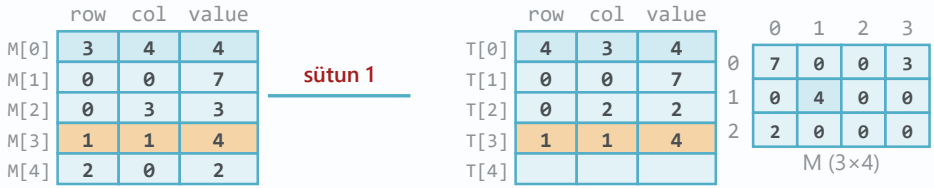
Sütun 1: M[2].col = 3 ≠ 1 → atla.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶         if (M[j].col == i) {
10          T[cur].row = M[j].col; T[cur].col = M[j].row;
11          T[cur].value = M[j].value;  cur++;
12        }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 3` `i = 1` `j = 2` `M[j].col = 3`

transpose: her sütun için tüm terimleri tara



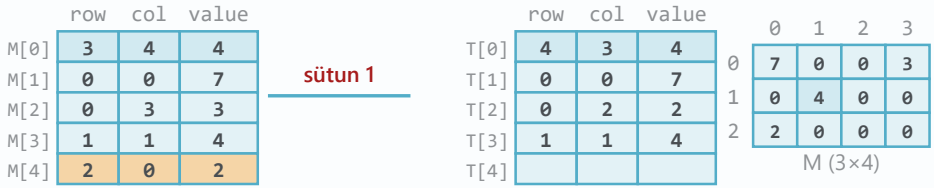
Sütun 1: M[3].col = 1 **eşit** → T[3] = (1, 1, 4), cur = 4.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶          if (M[j].col == i) {
▶            T[cur].row = M[j].col; T[cur].col = M[j].row;
▶            T[cur].value = M[j].value;  cur++;
12         }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 4` `i = 1` `j = 3` `M[j].col = 1`

transpose: her sütun için tüm terimleri tara



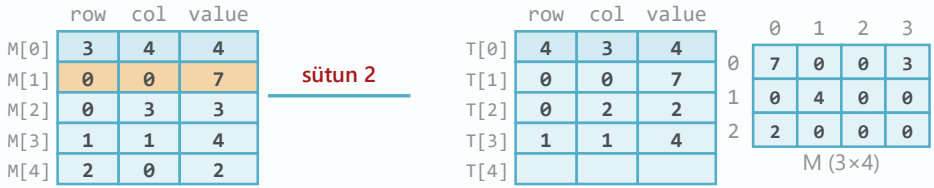
Sütun 1: M[4].col = 0 ≠ 1 → atla.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶         if (M[j].col == i) {
10          T[cur].row = M[j].col; T[cur].col = M[j].row;
11          T[cur].value = M[j].value;  cur++;
12        }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 4` `i = 1` `j = 4` `M[j].col = 0`

transpose: her sütun için tüm terimleri tara



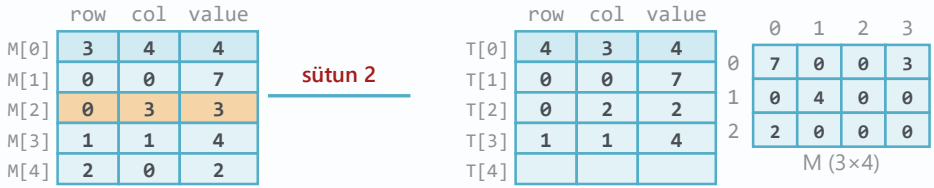
Sütun 2: M[1].col = 0 ≠ 2 → atla.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶         if (M[j].col == i) {
10          T[cur].row = M[j].col; T[cur].col = M[j].row;
11          T[cur].value = M[j].value;  cur++;
12        }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 4` `i = 2` `j = 1` `M[j].col = 0`

transpose: her sütun için tüm terimleri tara



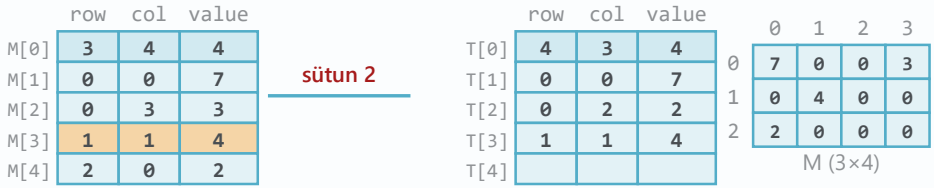
Sütun 2: M[2].col = 3 ≠ 2 → atla.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶          if (M[j].col == i) {
10           T[cur].row = M[j].col; T[cur].col = M[j].row;
11           T[cur].value = M[j].value;  cur++;
12         }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 4` `i = 2` `j = 2` `M[j].col = 3`

transpose: her sütun için tüm terimleri tara



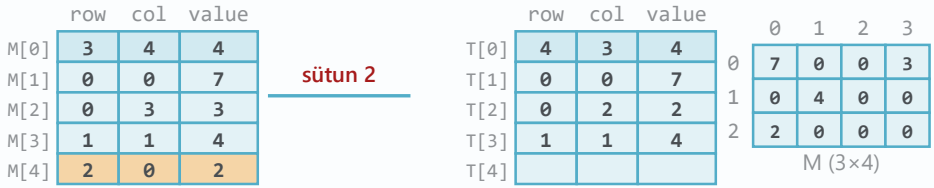
Sütun 2: M[3].col = 1 ≠ 2 → atla.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶         if (M[j].col == i) {
10          T[cur].row = M[j].col; T[cur].col = M[j].row;
11          T[cur].value = M[j].value;  cur++;
12        }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 4` `i = 2` `j = 3` `M[j].col = 1`

transpose: her sütun için tüm terimleri tara



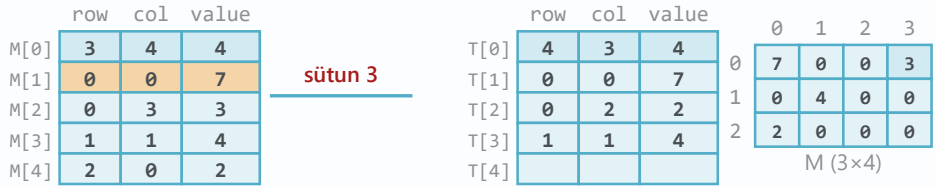
Sütun 2: M[4].col = 0 ≠ 2 → atla.


```
 1 void transpose(term M[], term T[]) {
 2   int n = M[0].value, i, j, cur;
 3   T[0].row = M[0].col;  T[0].col = M[0].row;
 4   T[0].value = n;
 5   if (n > 0) {
 6     cur = 1;                          /* T'de sıradaki */
 7     for (i = 0; i < M[0].col; i++)     /* her sütun */
 8       for (j = 1; j <= n; j++)         /* her terim */
 ▶       if (M[j].col == i) {
10         T[cur].row = M[j].col; T[cur].col = M[j].row;
11         T[cur].value = M[j].value;  cur++;
12       }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 4` `i = 2` `j = 4` `M[j].col = 0`

transpose: her sütun için tüm terimleri tara



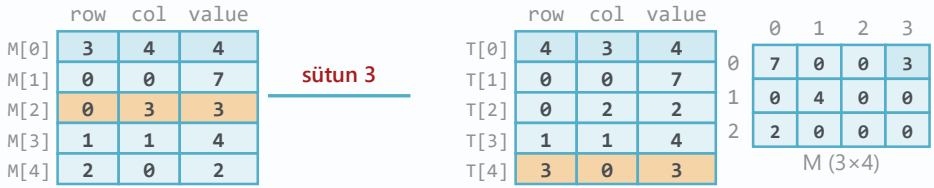
Sütun 3: M[1].col = 0 ≠ 3 → atla.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶         if (M[j].col == i) {
10          T[cur].row = M[j].col; T[cur].col = M[j].row;
11          T[cur].value = M[j].value;  cur++;
12        }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 4` `i = 3` `j = 1` `M[j].col = 0`

transpose: her sütun için tüm terimleri tara



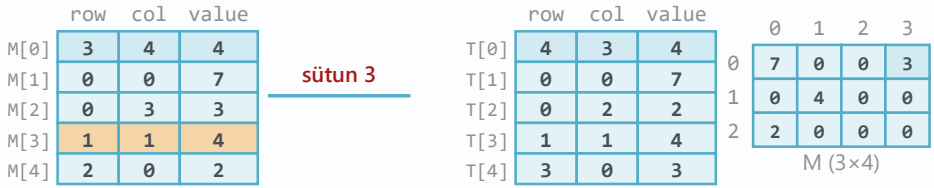
Sütun 3: M[2].col = 3 **eşit** → T[4] = (3, 0, 3), cur = 5.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶          if (M[j].col == i) {
▶            T[cur].row = M[j].col; T[cur].col = M[j].row;
▶            T[cur].value = M[j].value;  cur++;
12         }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 5` `i = 3` `j = 2` `M[j].col = 3`

transpose: her sütun için tüm terimleri tara



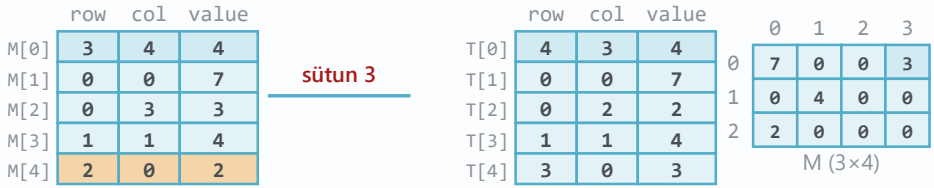
Sütun 3: M[3].col = 1 ≠ 3 → atla.


```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)      /* her sütun */
8        for (j = 1; j <= n; j++)          /* her terim */
▶         if (M[j].col == i) {
10          T[cur].row = M[j].col; T[cur].col = M[j].row;
11          T[cur].value = M[j].value;  cur++;
12        }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 5` `i = 3` `j = 3` `M[j].col = 1`

transpose: her sütun için tüm terimleri tara



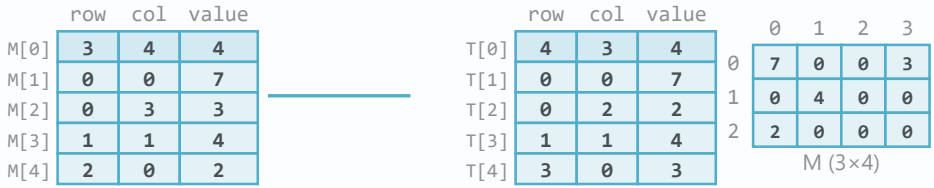
Sütun 3: M[4].col = 0 ≠ 3 → atla.


```
 1 void transpose(term M[], term T[]) {
 2   int n = M[0].value, i, j, cur;
 3   T[0].row = M[0].col;  T[0].col = M[0].row;
 4   T[0].value = n;
 5   if (n > 0) {
 6     cur = 1;                          /* T'de sıradaki */
 7     for (i = 0; i < M[0].col; i++)     /* her sütun */
 8       for (j = 1; j <= n; j++)         /* her terim */
 ▶       if (M[j].col == i) {
10         T[cur].row = M[j].col; T[cur].col = M[j].row;
11         T[cur].value = M[j].value;  cur++;
12       }
13   }
14 }
```

DEĞİŞKENLER

`n = 4` `cur = 5` `i = 3` `j = 4` `M[j].col = 0`

transpose: her sütun için tüm terimleri tara



Bitti: T satır sırasına göre **kendiliğinden sıralı** (M'de satırlar sıralıydı). 4 sütun × 4 terim = **16 karşılaştırma**.

Adım 18 / 18

```
1  void transpose(term M[], term T[]) {
2    int n = M[0].value, i, j, cur;
3    T[0].row = M[0].col;  T[0].col = M[0].row;
4    T[0].value = n;
5    if (n > 0) {
6      cur = 1;                          /* T'de sıradaki */
7      for (i = 0; i < M[0].col; i++)     /* her sütun */
8        for (j = 1; j <= n; j++)         /* her terim */
9          if (M[j].col == i) {
10           T[cur].row = M[j].col; T[cur].col = M[j].row;
11           T[cur].value = M[j].value;  cur++;
12         }
13   }
▶ }
```

DEĞİŞKENLER

`n = 4` `cur = 5`

transpose'un karmaşıklığı

- Dış döngü **cols** kez, iç döngü **terms** kez → **O(cols · terms)**.
- Matris doluysa terms = rows·cols → **O(cols² · rows)**: 2B dizide transpoz O(rows·cols) iken daha kötü!
- Yani bellekten kazanırken zamandan kaybettik — terms ≪ rows·cols olduğunda anlamlı.
- Sorun: aynı M dizisini **her sütun için yeniden** taramak.

Çıkış yolu

Her terimin T'de nereye gideceğini **önceden bilseydik**, M'yi bir kez tarayıp her terimi doğrudan yerine koyardık.

Örnek

Transpozun 0. satırında 2, 1. satırında 3 öğe olacaksa: 0. satır T[1]'den, 1. satır T[3]'ten, 2. satır T[6]'dan başlar.

fast_transpose: önce sayma, sonra yerleştirme

```
void fast_transpose(term M[], term T[]) {
  int i, j, nc = M[0].col, nt = M[0].value;
  int row_terms[MAX_COL], start_pos[MAX_COL];
  T[0].row = nc;  T[0].col = M[0].row;  T[0].value = nt;
  if (nt <= 0) return;                    /* boş matris */
  for (i = 0; i < nc; i++) row_terms[i] = 0;
  for (i = 1; i <= nt; i++) row_terms[M[i].col]++;
  start_pos[0] = 1;
  for (i = 1; i < nc; i++)
    start_pos[i] = start_pos[i-1] + row_terms[i-1];
  for (i = 1; i <= nt; i++) {
    j = start_pos[M[i].col]++;       /* yerini al, artır */
    T[j].row = M[i].col;  T[j].col = M[i].row;
    T[j].value = M[i].value;
  }
}
```

	row	col	value
T[0]	6	6	8
T[1]	0	0	15
T[2]	0	4	91
T[3]	1	1	11
T[4]	2	1	3
T[5]	2	5	28
T[6]	3	0	22
T[7]	3	2	-6
T[8]	5	0	-15

6×6 örnekte **row_terms = [2, 1, 2, 2, 0, 1]**, **start_pos = [1, 3, 4, 6, 8, 8]**.

fast_transpose (1): say ve başlangıçları bul

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2



Başlık kopyalanır: T 4×3, 4 terim. nt > 0 → devam.

Adım 1 / 10

```
1  void fast_transpose(term M[], term T[]) {
2    int i, j, nc = M[0].col, nt = M[0].value;
3    int row_terms[MAX_COL], start_pos[MAX_COL];
▶    T[0].row = nc;  T[0].col = M[0].row;
▶    T[0].value = nt;
6    if (nt <= 0) return;                  /* boş */
7    for (i = 0; i < nc; i++) row_terms[i] = 0;
8    for (i = 1; i <= nt; i++) row_terms[M[i].col]++;
9    start_pos[0] = 1;
10   for (i = 1; i < nc; i++)
11     start_pos[i] = start_pos[i-1] + row_terms[i-1];
12   /* 2. aşama: terimleri yerine koy */
```

DEĞİŞKENLER

`nc = 4` `nt = 4`

fast_transpose (1): say ve başlangıçları bul

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

row_terms	0	0	0	0
start_pos				

sütun 0 sütun 1 sütun 2 sütun 3

`row_terms` sıfırlanır: T'nin her satırı (= M'nin her sütunu) için sayaç.


```
1  void fast_transpose(term M[], term T[]) {
2    int i, j, nc = M[0].col, nt = M[0].value;
3    int row_terms[MAX_COL], start_pos[MAX_COL];
4    T[0].row = nc;  T[0].col = M[0].row;
5    T[0].value = nt;
6    if (nt <= 0) return;                    /* boş */
▶    for (i = 0; i < nc; i++) row_terms[i] = 0;
8    for (i = 1; i <= nt; i++) row_terms[M[i].col]++;
9    start_pos[0] = 1;
10   for (i = 1; i < nc; i++)
11     start_pos[i] = start_pos[i-1] + row_terms[i-1];
12   /* 2. aşama: terimleri yerine koy */
```

DEĞİŞKENLER

`nc = 4` `nt = 4`

fast_transpose (1): say ve başlangıçları bul

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	1	0	0	0
start_pos				

M[1] sütun 0'da → **row_terms[0]++** = 1.


```
1  void fast_transpose(term M[], term T[]) {
2    int i, j, nc = M[0].col, nt = M[0].value;
3    int row_terms[MAX_COL], start_pos[MAX_COL];
4    T[0].row = nc;  T[0].col = M[0].row;
5    T[0].value = nt;
6    if (nt <= 0) return;                    /* boş */
7    for (i = 0; i < nc; i++) row_terms[i] = 0;
▶    for (i = 1; i <= nt; i++) row_terms[M[i].col]++;
9    start_pos[0] = 1;
10   for (i = 1; i < nc; i++)
11     start_pos[i] = start_pos[i-1] + row_terms[i-1];
12   /* 2. aşama: terimleri yerine koy */
```

DEĞİŞKENLER

`nc = 4` `nt = 4` `i = 1` `M[i].col = 0`

fast_transpose (1): say ve başlangıçları bul

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	1	0	0	1
start_pos				

M[2] sütun 3'te → **row_terms[3]++** = 1.


```
1  void fast_transpose(term M[], term T[]) {
2    int i, j, nc = M[0].col, nt = M[0].value;
3    int row_terms[MAX_COL], start_pos[MAX_COL];
4    T[0].row = nc;  T[0].col = M[0].row;
5    T[0].value = nt;
6    if (nt <= 0) return;                    /* boş */
7    for (i = 0; i < nc; i++) row_terms[i] = 0;
▶    for (i = 1; i <= nt; i++) row_terms[M[i].col]++;
9    start_pos[0] = 1;
10   for (i = 1; i < nc; i++)
11     start_pos[i] = start_pos[i-1] + row_terms[i-1];
12   /* 2. aşama: terimleri yerine koy */
```

DEĞİŞKENLER

`nc = 4` `nt = 4` `i = 2` `M[i].col = 3`

fast_transpose (1): say ve başlangıçları bul

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	1	1	0	1
start_pos				

M[3] sütun 1'de → **row_terms[1]++** = 1.


```
1  void fast_transpose(term M[], term T[]) {
2    int i, j, nc = M[0].col, nt = M[0].value;
3    int row_terms[MAX_COL], start_pos[MAX_COL];
4    T[0].row = nc;  T[0].col = M[0].row;
5    T[0].value = nt;
6    if (nt <= 0) return;                    /* boş */
7    for (i = 0; i < nc; i++) row_terms[i] = 0;
▶    for (i = 1; i <= nt; i++) row_terms[M[i].col]++;
9    start_pos[0] = 1;
10   for (i = 1; i < nc; i++)
11     start_pos[i] = start_pos[i-1] + row_terms[i-1];
12   /* 2. aşama: terimleri yerine koy */
```

DEĞİŞKENLER

`nc = 4` `nt = 4` `i = 3` `M[i].col = 1`

fast_transpose (1): say ve başlangıçları bul

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos				

M[4] sütun 0'da → **row_terms[0]++** = 2.


```
1  void fast_transpose(term M[], term T[]) {
2    int i, j, nc = M[0].col, nt = M[0].value;
3    int row_terms[MAX_COL], start_pos[MAX_COL];
4    T[0].row = nc;  T[0].col = M[0].row;
5    T[0].value = nt;
6    if (nt <= 0) return;                  /* boş */
7    for (i = 0; i < nc; i++) row_terms[i] = 0;
▶    for (i = 1; i <= nt; i++) row_terms[M[i].col]++;
9    start_pos[0] = 1;
10   for (i = 1; i < nc; i++)
11     start_pos[i] = start_pos[i-1] + row_terms[i-1];
12   /* 2. aşama: terimleri yerine koy */
```

DEĞİŞKENLER

`nc = 4` `nt = 4` `i = 4` `M[i].col = 0`

fast_transpose (1): say ve başlangıçları bul

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos	1			

0. sütunun terimleri T[1]'den başlayacak: **start_pos[0] = 1**.

Adım 7 / 10

```
1  void fast_transpose(term M[], term T[]) {
2    int i, j, nc = M[0].col, nt = M[0].value;
3    int row_terms[MAX_COL], start_pos[MAX_COL];
4    T[0].row = nc;  T[0].col = M[0].row;
5    T[0].value = nt;
6    if (nt <= 0) return;                  /* boş */
7    for (i = 0; i < nc; i++) row_terms[i] = 0;
8    for (i = 1; i <= nt; i++) row_terms[M[i].col]++;
▶    start_pos[0] = 1;
10   for (i = 1; i < nc; i++)
11     start_pos[i] = start_pos[i-1] + row_terms[i-1];
12   /* 2. aşama: terimleri yerine koy */
```

DEĞİŞKENLER

`nc = 4` `nt = 4`

fast_transpose (1): say ve başlangıçları bul

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos	1	3		

start_pos[1] = 1 + 2 = **3**.


```
1  void fast_transpose(term M[], term T[]) {
2    int i, j, nc = M[0].col, nt = M[0].value;
3    int row_terms[MAX_COL], start_pos[MAX_COL];
4    T[0].row = nc;  T[0].col = M[0].row;
5    T[0].value = nt;
6    if (nt <= 0) return;                  /* boş */
7    for (i = 0; i < nc; i++) row_terms[i] = 0;
8    for (i = 1; i <= nt; i++) row_terms[M[i].col]++;
9    start_pos[0] = 1;
▶    for (i = 1; i < nc; i++)
▶      start_pos[i] = start_pos[i-1] + row_terms[i-1];
12   /* 2. aşama: terimleri yerine koy */
```

DEĞİŞKENLER

`nc = 4` `nt = 4` `i = 1`

fast_transpose (1): say ve başlangıçları bul

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos	1	3	4	

start_pos[2] = 3 + 1 = **4**.


```
1  void fast_transpose(term M[], term T[]) {
2    int i, j, nc = M[0].col, nt = M[0].value;
3    int row_terms[MAX_COL], start_pos[MAX_COL];
4    T[0].row = nc;  T[0].col = M[0].row;
5    T[0].value = nt;
6    if (nt <= 0) return;                  /* boş */
7    for (i = 0; i < nc; i++) row_terms[i] = 0;
8    for (i = 1; i <= nt; i++) row_terms[M[i].col]++;
9    start_pos[0] = 1;
▶    for (i = 1; i < nc; i++)
▶      start_pos[i] = start_pos[i-1] + row_terms[i-1];
12   /* 2. aşama: terimleri yerine koy */
```

DEĞİŞKENLER

`nc = 4` `nt = 4` `i = 2`

fast_transpose (1): say ve başlangıçları bul

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos	1	3	4	4

start_pos[3] = 4 + 0 = **4** (2. sütun boş: 3. sütun da 4'ten başlar).

Adım 10 / 10

```
1  void fast_transpose(term M[], term T[]) {
2    int i, j, nc = M[0].col, nt = M[0].value;
3    int row_terms[MAX_COL], start_pos[MAX_COL];
4    T[0].row = nc;  T[0].col = M[0].row;
5    T[0].value = nt;
6    if (nt <= 0) return;                  /* boş */
7    for (i = 0; i < nc; i++) row_terms[i] = 0;
8    for (i = 1; i <= nt; i++) row_terms[M[i].col]++;
9    start_pos[0] = 1;
▶    for (i = 1; i < nc; i++)
▶      start_pos[i] = start_pos[i-1] + row_terms[i-1];
12   /* 2. aşama: terimleri yerine koy */
```

DEĞİŞKENLER

`nc = 4` `nt = 4` `i = 3`

fast_transpose (2): terimleri yerine koy

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos	1	3	4	4

	row	col	value
T[0]	4	3	4
T[1]			
T[2]			
T[3]			
T[4]			

M bir kez, baştan sona taranır; her terim doğrudan yerine gider.

Adım 1 / 10

```
1  /* 1. aşama: row_terms ve start_pos hazır */
2  for (i = 1; i <= nt; i++) {
3    j = start_pos[M[i].col]++;  /* oku, artır */
4    T[j].row = M[i].col;
5    T[j].col = M[i].row;
6    T[j].value = M[i].value;
7  }
8 }
```

DEĞİŞKENLER

`nt = 4`

`start_pos = [1, 3, 4, 4]`

Hocadan *"Programcı, program yazdığı dilin işleç önceliklerini iyi bilmelidir."*

fast_transpose (2): terimleri yerine koy

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos	2	3	4	4

	row	col	value
T[0]	4	3	4
T[1]	0	0	7
T[2]			
T[3]			
T[4]			

M[1] = (0, 0, 7): **j = start_pos[0]++** → j = 1, start_pos[0] = 2.

Adım 2 / 10

```
1  /* 1. aşama: row_terms ve start_pos hazır */
2  for (i = 1; i <= nt; i++) {
▶    j = start_pos[M[i].col]++;  /* oku, artır */
4    T[j].row = M[i].col;
5    T[j].col = M[i].row;
6    T[j].value = M[i].value;
7  }
8 }
```

DEĞİŞKENLER

`nt = 4` `i = 1` `M[i].col = 0` `j = 1` `start_pos = [2, 3, 4, 4]`

Hocadan *"Programcı, program yazdığı dilin işleç önceliklerini iyi bilmelidir."*

fast_transpose (2): terimleri yerine koy

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos	2	3	4	4

	row	col	value
T[0]	4	3	4
T[1]	0	0	7
T[2]			
T[3]			
T[4]			

T[1] = (0, 0, 7): satır/sütun yer değiştirdi.

Adım 3 / 10

```
1  /* 1. aşama: row_terms ve start_pos hazır */
2  for (i = 1; i <= nt; i++) {
3    j = start_pos[M[i].col]++;  /* oku, artır */
▶    T[j].row = M[i].col;
▶    T[j].col = M[i].row;
▶    T[j].value = M[i].value;
7  }
8 }
```

DEĞİŞKENLER

`nt = 4` `i = 1` `M[i].col = 0` `j = 1` `start_pos = [2, 3, 4, 4]`

Hocadan *"Programcı, program yazdığı dilin işleç önceliklerini iyi bilmelidir."*

fast_transpose (2): terimleri yerine koy

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos	2	3	4	5

	row	col	value
T[0]	4	3	4
T[1]	0	0	7
T[2]			
T[3]			
T[4]	3	0	3

M[2] = (0, 3, 3): **j = start_pos[3]++** → j = 4, start_pos[3] = 5.

Adım 4 / 10

```
1  /* 1. aşama: row_terms ve start_pos hazır */
2  for (i = 1; i <= nt; i++) {
▶    j = start_pos[M[i].col]++;  /* oku, artır */
4    T[j].row = M[i].col;
5    T[j].col = M[i].row;
6    T[j].value = M[i].value;
7  }
8 }
```

DEĞİŞKENLER

`nt = 4` `i = 2` `M[i].col = 3` `j = 4` `start_pos = [2, 3, 4, 5]`

Hocadan *"Programcı, program yazdığı dilin işleç önceliklerini iyi bilmelidir."*

fast_transpose (2): terimleri yerine koy

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

row_terms	2	1	0	1
start_pos	2	3	4	5

sütun 0 sütun 1 sütun 2 sütun 3

	row	col	value
T[0]	4	3	4
T[1]	0	0	7
T[2]			
T[3]			
T[4]	3	0	3

T[4] = (3, 0, 3): satır/sütun yer değiştirdi.

Adım 5 / 10

```
1  /* 1. aşama: row_terms ve start_pos hazır */
2  for (i = 1; i <= nt; i++) {
3    j = start_pos[M[i].col]++;  /* oku, artır */
▶    T[j].row = M[i].col;
▶    T[j].col = M[i].row;
▶    T[j].value = M[i].value;
7  }
8 }
```

DEĞİŞKENLER

`nt = 4` `i = 2` `M[i].col = 3` `j = 4` `start_pos = [2, 3, 4, 5]`

Hocadan *"Programcı, program yazdığı dilin işleç önceliklerini iyi bilmelidir."*

fast_transpose (2): terimleri yerine koy

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos	2	4	4	5

	row	col	value
T[0]	4	3	4
T[1]	0	0	7
T[2]			
T[3]	1	1	4
T[4]	3	0	3

M[3] = (1, 1, 4): **j = start_pos[1]++** → j = 3, start_pos[1] = 4.

Adım 6 / 10

```
1   /* 1. aşama: row_terms ve start_pos hazır */
2   for (i = 1; i <= nt; i++) {
▶     j = start_pos[M[i].col]++;  /* oku, artır */
4     T[j].row = M[i].col;
5     T[j].col = M[i].row;
6     T[j].value = M[i].value;
7   }
8 }
```

DEĞİŞKENLER

`nt = 4` `i = 3` `M[i].col = 1` `j = 3` `start_pos = [2, 4, 4, 5]`

Hocadan *"Programcı, program yazdığı dilin işleç önceliklerini iyi bilmelidir."*

fast_transpose (2): terimleri yerine koy

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

row_terms	2	1	0	1
start_pos	2	4	4	5

sütun 0 sütun 1 sütun 2 sütun 3

	row	col	value
T[0]	4	3	4
T[1]	0	0	7
T[2]			
T[3]	1	1	4
T[4]	3	0	3

T[3] = (1, 1, 4): satır/sütun yer değiştirdi.

Adım 7 / 10

```
1  /* 1. aşama: row_terms ve start_pos hazır */
2  for (i = 1; i <= nt; i++) {
3    j = start_pos[M[i].col]++;  /* oku, artır */
▶    T[j].row = M[i].col;
▶    T[j].col = M[i].row;
▶    T[j].value = M[i].value;
7  }
8 }
```

DEĞİŞKENLER

`nt = 4` `i = 3` `M[i].col = 1` `j = 3` `start_pos = [2, 4, 4, 5]`

Hocadan *"Programcı, program yazdığı dilin işleç önceliklerini iyi bilmelidir."*

fast_transpose (2): terimleri yerine koy

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos	3	4	4	5

	row	col	value
T[0]	4	3	4
T[1]	0	0	7
T[2]	0	2	2
T[3]	1	1	4
T[4]	3	0	3

M[4] = (2, 0, 2): **j = start_pos[0]++** → j = 2, start_pos[0] = 3.

Adım 8 / 10

```
1  /* 1. aşama: row_terms ve start_pos hazır */
2  for (i = 1; i <= nt; i++) {
▶    j = start_pos[M[i].col]++;  /* oku, artır */
4    T[j].row = M[i].col;
5    T[j].col = M[i].row;
6    T[j].value = M[i].value;
7  }
8 }
```

DEĞİŞKENLER

`nt = 4` `i = 4` `M[i].col = 0` `j = 2` `start_pos = [3, 4, 4, 5]`

Hocadan *"Programcı, program yazdığı dilin işleç önceliklerini iyi bilmelidir."*

fast_transpose (2): terimleri yerine koy

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos	3	4	4	5

	row	col	value
T[0]	4	3	4
T[1]	0	0	7
T[2]	0	2	2
T[3]	1	1	4
T[4]	3	0	3

T[2] = (0, 2, 2): satır/sütun yer değiştirdi.

Adım 9 / 10

```
1  /* 1. aşama: row_terms ve start_pos hazır */
2  for (i = 1; i <= nt; i++) {
3    j = start_pos[M[i].col]++;  /* oku, artır */
▶    T[j].row = M[i].col;
▶    T[j].col = M[i].row;
▶    T[j].value = M[i].value;
7  }
8 }
```

DEĞİŞKENLER

`nt = 4` `i = 4` `M[i].col = 0` `j = 2` `start_pos = [3, 4, 4, 5]`

Hocadan *"Programcı, program yazdığı dilin işleç önceliklerini iyi bilmelidir."*

fast_transpose (2): terimleri yerine koy

	row	col	value
M[0]	3	4	4
M[1]	0	0	7
M[2]	0	3	3
M[3]	1	1	4
M[4]	2	0	2

	sütun 0	sütun 1	sütun 2	sütun 3
row_terms	2	1	0	1
start_pos	3	4	4	5

	row	col	value
T[0]	4	3	4
T[1]	0	0	7
T[2]	0	2	2
T[3]	1	1	4
T[4]	3	0	3

Bitti — sonuç transpose ile aynı. İki doğrusal tarama: **O(cols + terms)**.

Adım 10 / 10

```
1  /* 1. aşama: row_terms ve start_pos hazır */
2  for (i = 1; i <= nt; i++) {
3    j = start_pos[M[i].col]++;  /* oku, artır */
4    T[j].row = M[i].col;
5    T[j].col = M[i].row;
6    T[j].value = M[i].value;
7  }
▶ }
```

DEĞİŞKENLER

`nt = 4`

`start_pos = [3, 4, 4, 5]`

Hocadan *"Programcı, program yazdığı dilin işleç önceliklerini iyi bilmelidir."*

Analiz ve tek dizili sürüm

Döngüler: num_cols + num_terms + num_cols + num_terms (+ sabitler):

$$2 \cdot terms + 2 \cdot cols + 3 = O(cols + terms)$$

Dolu matriste O(rows·cols): 2B dizideki transpozla aynı derece. Bedeli: iki yardımcı dizi (O(cols) bellek).

> **Hocadan** *"Hızlı transpoz tek bellekle olur denmişti ya, işte bu o..."*

```
/* row_terms'ü yerinde start_pos'a çevir */
for (i = 0, pos = 1; i < nc; i++) {
  int c = row_terms[i];  /* bu sütundaki terim sayısı */
  row_terms[i] = pos;  /* artık başlangıç konumu */
  pos += c;
}
for (i = 1; i <= nt; i++) {
  j = row_terms[M[i].col]++;
  T[j].row = M[i].col;  T[j].col = M[i].row;
  T[j].value = M[i].value;
}
```

Hızlı transpoz tek bir dizi kullanılarak da yazılabilir. Bu, **saymalı sıralama**nın (counting sort) ta kendisi: anahtar = sütun no.

$D_{m\times p} = A_{m\times n} \cdot B_{n\times p}$

$$d_{ij} = \Sigma_{k=0}^{n-1} a_{ik} \cdot b_{kj}$$

A'nın i. satırı ile B'nin j. sütunu "çarpılıp toplanır". İki seyrek matrisin çarpımı seyrek **olmak zorunda değil**:

A

1	0	0
1	0	0
1	0	0

·

B

1	1	1
0	0	0
0	0	0

=

D

1	1	1
1	1	1
1	1	1

- Üçlülerde A'nın i. satırı **ardışık** durur, ama B'nin j. sütunu dağınıktır.
- Bu yüzden önce **B'nin transpozu** alınır (fast_transpose): B'nin sütunları B^T'nin ardışık satırları olur.
- Sonra A'nın her satırı ile B^T'nin her satırı, **col** alanına göre birleştirilerek (merge) çarpılır.

> "Önce **ilk** matrisin transpozu alınır" demek yanlıştır; Horowitz'in `mmult`'ında (ve mantık olarak) transpozu alınan **ikinci matris B**'dir.

Yardımcılar: satır sonu, nokta çarpım, sakla

```
/* s'deki terimin satırının bittiği yer */
int son(term t[], int s) {
  int e = s;
  while (e <= t[0].value &&
         t[e].row == t[s].row) e++;
  return e;
}
/* a[x..ex) satırı · b[y..ey) satırı */
int carp(term a[], int x, int ex,
         term b[], int y, int ey) {
  int sum = 0;
  while (x < ex && y < ey)
    if (a[x].col < b[y].col) x++;
    else if (a[x].col > b[y].col) y++;
    else sum += a[x++].value * b[y++].value;
  return sum;
}
```

```
void sakla(term d[], int *nd,
      int r, int c, int s) {
  if (s == 0) return;
  if (++*nd >= MAX_TERMS) {
    fprintf(stderr, "Cok terim\n");
    exit(1);
  }
  d[*nd].row = r;
  d[*nd].col = c;
  d[*nd].value = s;
}
```

`carp`, padd gibi iki işaretçili birleştirmedir: yalnız aynı **k** (col) değerli terimler çarpılır. Sıfır toplam saklanmaz.

Seyrek çarpım: A'nın satırı · Bᵀ'nin satırı

A (2×3)

	row	col	value
a[0]	2	3	3
a[1]	0	0	1
a[2]	0	2	2
a[3]	1	1	3

Bᵀ (B: 3×2)

	row	col	value
bt[0]	2	3	3
bt[1]	0	0	4
bt[2]	0	2	6
bt[3]	1	1	5

D = A·B (2×2)

	row	col	value
d[0]	2	2	0
d[1]			
d[2]			

A = [[1,0,2],[0,3,0]], B = [[4,0],[0,5],[6,0]]; B'nin transpozu `bt` önceden alındı.

Adım 1 / 12

```
1  void smult(term a[], term bt[], term d[]) {
▶    int ra, ea, rb, eb, nd = 0;
3    for (ra = 1; ra <= a[0].value; ra = ea) {
4      ea = son(a, ra);            /* A satırı [ra,ea) */
5      for (rb = 1; rb <= bt[0].value; rb = eb) {
6        eb = son(bt, rb);         /* B sütunu [rb,eb) */
7        sakla(d, &nd, a[ra].row, bt[rb].row,
8              carp(a, ra, ea, bt, rb, eb));
9      }
10   }
11   d[0].row = a[0].row;  d[0].col = bt[0].row;
12   d[0].value = nd;
13 }
```

DEĞİŞKENLER

nd = 0

Seyrek çarpım: A'nın satırı · Bᵀ'nin satırı

A (2×3)

	row	col	value
a[0]	2	3	3
a[1]	0	0	1
a[2]	0	2	2
a[3]	1	1	3

Bᵀ (B: 3×2)

	row	col	value
bt[0]	2	3	3
bt[1]	0	0	4
bt[2]	0	2	6
bt[3]	1	1	5

D = A·B (2×2)

	row	col	value
d[0]	2	2	0
d[1]			
d[2]			

A'nın 0. satırı: a[1..2].

Adım 2 / 12

```
1  void smult(term a[], term bt[], term d[]) {
2    int ra, ea, rb, eb, nd = 0;
3    for (ra = 1; ra <= a[0].value; ra = ea) {
▶      ea = son(a, ra);            /* A satırı [ra,ea) */
5      for (rb = 1; rb <= bt[0].value; rb = eb) {
6        eb = son(bt, rb);         /* B sütunu [rb,eb) */
7        sakla(d, &nd, a[ra].row, bt[rb].row,
8              carp(a, ra, ea, bt, rb, eb));
9      }
10   }
11   d[0].row = a[0].row;  d[0].col = bt[0].row;
12   d[0].value = nd;
13 }
```

DEĞİŞKENLER

ra = 1 | ea = 3 | nd = 0

Seyrek çarpım: A'nın satırı · Bᵀ'nin satırı

A (2×3)

	row	col	value
a[0]	2	3	3
a[1]	0	0	1
a[2]	0	2	2
a[3]	1	1	3

Bᵀ (B: 3×2)

	row	col	value
bt[0]	2	3	3
bt[1]	0	0	4
bt[2]	0	2	6
bt[3]	1	1	5

D = A·B (2×2)

	row	col	value
d[0]	2	2	0
d[1]			
d[2]			

B'nin 0. sütunu = bt[1..2].

Adım 3 / 12

```
1  void smult(term a[], term bt[], term d[]) {
2    int ra, ea, rb, eb, nd = 0;
3    for (ra = 1; ra <= a[0].value; ra = ea) {
4      ea = son(a, ra);            /* A satırı [ra,ea) */
5      for (rb = 1; rb <= bt[0].value; rb = eb) {
▶        eb = son(bt, rb);         /* B sütunu [rb,eb) */
7        sakla(d, &nd, a[ra].row, bt[rb].row,
8              carp(a, ra, ea, bt, rb, eb));
9      }
10   }
11   d[0].row = a[0].row;  d[0].col = bt[0].row;
12   d[0].value = nd;
13 }
```

DEĞİŞKENLER

`ra = 1` `ea = 3` `rb = 1` `eb = 3` `nd = 0`

Seyrek çarpım: A'nın satırı · Bᵀ'nin satırı

A (2×3)

	row	col	value
a[0]	2	3	3
a[1]	0	0	1
a[2]	0	2	2
a[3]	1	1	3

Bᵀ (B: 3×2)

	row	col	value
bt[0]	2	3	3
bt[1]	0	0	4
bt[2]	0	2	6
bt[3]	1	1	5

D = A·B (2×2)

	row	col	value
d[0]	2	2	1
d[1]	0	0	16
d[2]			

carp: 1·4 + 2·6 = **16** → d[1] = (0, 0, 16).

Adım 4 / 12

```
1  void smult(term a[], term bt[], term d[]) {
2    int ra, ea, rb, eb, nd = 0;
3    for (ra = 1; ra <= a[0].value; ra = ea) {
4      ea = son(a, ra);            /* A satırı [ra,ea) */
5      for (rb = 1; rb <= bt[0].value; rb = eb) {
6        eb = son(bt, rb);         /* B sütunu [rb,eb) */
▶        sakla(d, &nd, a[ra].row, bt[rb].row,
▶              carp(a, ra, ea, bt, rb, eb));
9      }
10   }
11   d[0].row = a[0].row;  d[0].col = bt[0].row;
12   d[0].value = nd;
13 }
```

DEĞİŞKENLER

`ra = 1` `ea = 3` `rb = 1` `eb = 3` `nd = 1`

Seyrek çarpım: A'nın satırı · Bᵀ'nin satırı

A (2×3)

	row	col	value
a[0]	2	3	3
a[1]	0	0	1
a[2]	0	2	2
a[3]	1	1	3

Bᵀ (B: 3×2)

	row	col	value
bt[0]	2	3	3
bt[1]	0	0	4
bt[2]	0	2	6
bt[3]	1	1	5

D = A·B (2×2)

	row	col	value
d[0]	2	2	1
d[1]	0	0	16
d[2]			

B'nin 1. sütunu = bt[3..3].

Adım 5 / 12

```
1  void smult(term a[], term bt[], term d[]) {
2    int ra, ea, rb, eb, nd = 0;
3    for (ra = 1; ra <= a[0].value; ra = ea) {
4      ea = son(a, ra);            /* A satırı [ra,ea) */
5      for (rb = 1; rb <= bt[0].value; rb = eb) {
▶        eb = son(bt, rb);         /* B sütunu [rb,eb) */
7        sakla(d, &nd, a[ra].row, bt[rb].row,
8              carp(a, ra, ea, bt, rb, eb));
9      }
10   }
11   d[0].row = a[0].row;  d[0].col = bt[0].row;
12   d[0].value = nd;
13 }
```

DEĞİŞKENLER

`ra = 1` `ea = 3` `rb = 3` `eb = 4` `nd = 1`

Seyrek çarpım: A'nın satırı · Bᵀ'nin satırı

A (2×3)

	row	col	value
a[0]	2	3	3
a[1]	0	0	1
a[2]	0	2	2
a[3]	1	1	3

Bᵀ (B: 3×2)

	row	col	value
bt[0]	2	3	3
bt[1]	0	0	4
bt[2]	0	2	6
bt[3]	1	1	5

D = A·B (2×2)

	row	col	value
d[0]	2	2	1
d[1]	0	0	16
d[2]			

carp: ortak col yok = **0** → sıfır, **saklanmaz**.


```
1  void smult(term a[], term bt[], term d[]) {
2    int ra, ea, rb, eb, nd = 0;
3    for (ra = 1; ra <= a[0].value; ra = ea) {
4      ea = son(a, ra);            /* A satırı [ra,ea) */
5      for (rb = 1; rb <= bt[0].value; rb = eb) {
6        eb = son(bt, rb);         /* B sütunu [rb,eb) */
▶        sakla(d, &nd, a[ra].row, bt[rb].row,
▶              carp(a, ra, ea, bt, rb, eb));
9      }
10   }
11   d[0].row = a[0].row;  d[0].col = bt[0].row;
12   d[0].value = nd;
13 }
```

DEĞİŞKENLER

`ra = 1` `ea = 3` `rb = 3` `eb = 4` `nd = 1`

Seyrek çarpım: A'nın satırı · Bᵀ'nin satırı

A (2×3)

	row	col	value
a[0]	2	3	3
a[1]	0	0	1
a[2]	0	2	2
a[3]	1	1	3

Bᵀ (B: 3×2)

	row	col	value
bt[0]	2	3	3
bt[1]	0	0	4
bt[2]	0	2	6
bt[3]	1	1	5

D = A·B (2×2)

	row	col	value
d[0]	2	2	1
d[1]	0	0	16
d[2]			

A'nın 1. satırı: a[3..3].

Adım 7 / 12

```
1  void smult(term a[], term bt[], term d[]) {
2    int ra, ea, rb, eb, nd = 0;
3    for (ra = 1; ra <= a[0].value; ra = ea) {
▶      ea = son(a, ra);            /* A satırı [ra,ea) */
5      for (rb = 1; rb <= bt[0].value; rb = eb) {
6        eb = son(bt, rb);         /* B sütunu [rb,eb) */
7        sakla(d, &nd, a[ra].row, bt[rb].row,
8              carp(a, ra, ea, bt, rb, eb));
9      }
10   }
11   d[0].row = a[0].row;  d[0].col = bt[0].row;
12   d[0].value = nd;
13 }
```

DEĞİŞKENLER

ra = 3 | ea = 4 | nd = 1

Seyrek çarpım: A'nın satırı · Bᵀ'nin satırı

A (2×3)

	row	col	value
a[0]	2	3	3
a[1]	0	0	1
a[2]	0	2	2
a[3]	1	1	3

Bᵀ (B: 3×2)

	row	col	value
bt[0]	2	3	3
bt[1]	0	0	4
bt[2]	0	2	6
bt[3]	1	1	5

D = A·B (2×2)

	row	col	value
d[0]	2	2	1
d[1]	0	0	16
d[2]			

B'nin 0. sütunu = bt[1..2].


```
1  void smult(term a[], term bt[], term d[]) {
2    int ra, ea, rb, eb, nd = 0;
3    for (ra = 1; ra <= a[0].value; ra = ea) {
4      ea = son(a, ra);            /* A satırı [ra,ea) */
5      for (rb = 1; rb <= bt[0].value; rb = eb) {
▶        eb = son(bt, rb);         /* B sütunu [rb,eb) */
7        sakla(d, &nd, a[ra].row, bt[rb].row,
8              carp(a, ra, ea, bt, rb, eb));
9      }
10   }
11   d[0].row = a[0].row;  d[0].col = bt[0].row;
12   d[0].value = nd;
13 }
```

DEĞİŞKENLER

ra = 3 · ea = 4 · rb = 1 · eb = 3 · nd = 1

Seyrek çarpım: A'nın satırı · Bᵀ'nin satırı

A (2×3)

	row	col	value
a[0]	2	3	3
a[1]	0	0	1
a[2]	0	2	2
a[3]	1	1	3

Bᵀ (B: 3×2)

	row	col	value
bt[0]	2	3	3
bt[1]	0	0	4
bt[2]	0	2	6
bt[3]	1	1	5

D = A·B (2×2)

	row	col	value
d[0]	2	2	1
d[1]	0	0	16
d[2]			

carp: ortak col yok = **0** → sıfır, **saklanmaz**.


```
1  void smult(term a[], term bt[], term d[]) {
2    int ra, ea, rb, eb, nd = 0;
3    for (ra = 1; ra <= a[0].value; ra = ea) {
4      ea = son(a, ra);            /* A satırı [ra,ea) */
5      for (rb = 1; rb <= bt[0].value; rb = eb) {
6        eb = son(bt, rb);         /* B sütunu [rb,eb) */
▶        sakla(d, &nd, a[ra].row, bt[rb].row,
▶              carp(a, ra, ea, bt, rb, eb));
9      }
10   }
11   d[0].row = a[0].row;  d[0].col = bt[0].row;
12   d[0].value = nd;
13 }
```

DEĞİŞKENLER

`ra = 3` `ea = 4` `rb = 1` `eb = 3` `nd = 1`

Seyrek çarpım: A'nın satırı · Bᵀ'nin satırı

A (2×3)

	row	col	value
a[0]	2	3	3
a[1]	0	0	1
a[2]	0	2	2
a[3]	1	1	3

Bᵀ (B: 3×2)

	row	col	value
bt[0]	2	3	3
bt[1]	0	0	4
bt[2]	0	2	6
bt[3]	1	1	5

D = A·B (2×2)

	row	col	value
d[0]	2	2	1
d[1]	0	0	16
d[2]			

B'nin 1. sütunu = bt[3..3].

Adım 10 / 12

```
1  void smult(term a[], term bt[], term d[]) {
2    int ra, ea, rb, eb, nd = 0;
3    for (ra = 1; ra <= a[0].value; ra = ea) {
4      ea = son(a, ra);            /* A satırı [ra,ea) */
5      for (rb = 1; rb <= bt[0].value; rb = eb) {
▶        eb = son(bt, rb);         /* B sütunu [rb,eb) */
7        sakla(d, &nd, a[ra].row, bt[rb].row,
8              carp(a, ra, ea, bt, rb, eb));
9      }
10   }
11   d[0].row = a[0].row;  d[0].col = bt[0].row;
12   d[0].value = nd;
13 }
```

DEĞİŞKENLER

`ra = 3` `ea = 4` `rb = 3` `eb = 4` `nd = 1`

Seyrek çarpım: A'nın satırı · Bᵀ'nin satırı

A (2×3)

	row	col	value
a[0]	2	3	3
a[1]	0	0	1
a[2]	0	2	2
a[3]	1	1	3

Bᵀ (B: 3×2)

	row	col	value
bt[0]	2	3	3
bt[1]	0	0	4
bt[2]	0	2	6
bt[3]	1	1	5

D = A·B (2×2)

	row	col	value
d[0]	2	2	2
d[1]	0	0	16
d[2]	1	1	15

carp: 3·5 = **15** → d[2] = (1, 1, 15).

Adım 11 / 12

```
1  void smult(term a[], term bt[], term d[]) {
2    int ra, ea, rb, eb, nd = 0;
3    for (ra = 1; ra <= a[0].value; ra = ea) {
4      ea = son(a, ra);            /* A satırı [ra,ea) */
5      for (rb = 1; rb <= bt[0].value; rb = eb) {
6        eb = son(bt, rb);         /* B sütunu [rb,eb) */
▶        sakla(d, &nd, a[ra].row, bt[rb].row,
▶              carp(a, ra, ea, bt, rb, eb));
9      }
10   }
11   d[0].row = a[0].row;  d[0].col = bt[0].row;
12   d[0].value = nd;
13 }
```

DEĞİŞKENLER

ra = 3 · ea = 4 · rb = 3 · eb = 4 · nd = 2

Seyrek çarpım: A'nın satırı · Bᵀ'nin satırı

A (2×3)

	row	col	value
a[0]	2	3	3
a[1]	0	0	1
a[2]	0	2	2
a[3]	1	1	3

Bᵀ (B: 3×2)

	row	col	value
bt[0]	2	3	3
bt[1]	0	0	4
bt[2]	0	2	6
bt[3]	1	1	5

D = A·B (2×2)

	row	col	value
d[0]	2	2	2
d[1]	0	0	16
d[2]	1	1	15

Başlık yazılır: D 2×2, 2 terim. D = [[16, 0], [0, 15]].

Adım 12 / 12

```
1  void smult(term a[], term bt[], term d[]) {
2    int ra, ea, rb, eb, nd = 0;
3    for (ra = 1; ra <= a[0].value; ra = ea) {
4      ea = son(a, ra);            /* A satırı [ra,ea) */
5      for (rb = 1; rb <= bt[0].value; rb = eb) {
6        eb = son(bt, rb);         /* B sütunu [rb,eb) */
7        sakla(d, &nd, a[ra].row, bt[rb].row,
8              carp(a, ra, ea, bt, rb, eb));
9      }
10   }
▶    d[0].row = a[0].row;  d[0].col = bt[0].row;
▶    d[0].value = nd;
13 }
```

DEĞİŞKENLER

nd = **2**

Seyrek çarpımın karmaşıklığı

- Transpoz: $O(cols_B + t_B)$.
- A'nın her satırı i için B^T baştan sona gezilir: $\Sigma_i\ t_B = \mathbf{rows_A \cdot t_B}$.
- Her (i, j) çifti için A satırının terimleri de gezilir: $\Sigma_j\ t_A = \mathbf{cols_B \cdot t_A}$.
- Toplam: $\mathbf{O(cols_B \cdot t_A + rows_A \cdot t_B)}$.

Klasik çarpım

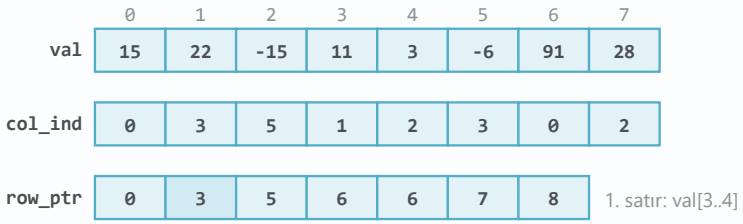
$O(rows_A \cdot cols_A \cdot cols_B)$. Matrisler doluysa ($t_A = rows_A \cdot cols_A$) seyrek sürüm de aynı dereceye çıkar, sabiti daha büyüktür.

Ne zaman değer?

$t_A \ll rows \cdot cols$ ve $t_B \ll rows \cdot cols$ iken. Sonuç D'nin seyrek olacağı garanti değildir: MAX_TERMS aşılabilir.

DİĞER GÖSTERİMLER

Sıkıştırılmış satır (CSR) gösterimi



Aynı 6×6 matris: 8 + 8 + 7 = 23 tam sayı (üçlülerde 27). i. satır: `val[row_ptr[i] .. row_ptr[i+1]-1]`; boş satır 3 → art arda eşit iki row_ptr.

```
/* y = M * x  (matris-vektör çarpımı) */
for (i = 0; i < rows; i++) {
  y[i] = 0;
  for (k = row_ptr[i]; k < row_ptr[i+1]; k++)
    y[i] += val[k] * x[col_ind[k]];
}
```

Kullanım

Bilimsel hesapta en yaygın biçimlerden; M·x O(rows + terms). Sütun sürümü: CSC.

Bağ

row_ptr, fast_transpose'taki start_pos ile aynı fikir (önek toplamları).

Seyrek matrisin bağlı listeyle gösterimi

- Üçlü dizide ekleme/silme kaydırma ister ve MAX_TERMS sınırı vardır.
- Bağlı gösterimde her dolu öğe bir düğümdür; aynı satırdakiler ve aynı sütundakiler **döngülü listelerle** bağlanır; satır/sütun başlık düğümleri vardır (Horowitz 4.7).
- Ekleme/silme O(1) bağ değişikliği (yer bulunduktan sonra); satıra ve sütuna göre gezinme kolay.
- Bedeli: öğe başına iki işaretçi ek bellek.

Hocadan *"Ödev kâğıdı benim kâğıdımla üst üste getirilip ışığa tutulduğunda... (fark görülmemeli)"*

Nerede?

Seyrek matrisin bağlı gerçeklemesi, bağlı listelerin doğrudan bir uygulamasıdır.

KAPANIŞ

Karmaşıklık Özeti

İşlem	Zaman	Not
padd (n, m terim)	$O(n + m)$	sıralı terimler + birleştirme
pmult (dizi, art arda padd)	$O(n^2 m)$	küçük polinom dışta: $O(\min^2 \cdot \max)$
PolCarp (bağlı liste)	$O(n^2 m)$ en kötü	bas ile arama kısalır
transpose	$O(\text{cols} \cdot \text{terms})$	dolu matriste $O(\text{cols}^2 \cdot \text{rows})$
fast_transpose	$O(\text{cols} + \text{terms})$	$O(\text{cols})$ ek bellek
Seyrek çarpım (smult / mmult)	$O(\text{cols}_B \cdot t_A + \text{rows}_A \cdot t_B)$	önce B^T
CSR ile M·x	$O(\text{rows} + \text{terms})$	satır başı dizisi

KAPANIŞ

Sık Yapılan Hatalar

1. Terimleri sırasız tutup padd'i yine de iki işaretçiyle yazmak.
2. Toplamı 0 olan terimi sonuca eklemek ya da listeden silmemek.
3. `attach`'ta float katsayıyı int parametreyle kırpmak; `avail` taşmasını denetlememek.
4. Transpozda yalnız row/col değiştirip sıralamayı unutmak; M[0] başlığını terim sanmak.
5. `start_pos[M[i].col]++` ifadesinde ön ek/son ek farkını karıştırmak.
6. Seyrek çarpımda A'nın transpozunu almak; sonucun seyrek kalacağını varsaymak.

KAPANIŞ

Alıştırmalar

1. $A = 2x^4 + 3x^2 - 1$ ve $B = -3x^2 + x + 1$ için padd'i izleyin. D'de kaç terim olur?
2. P dizisi gösteriminde 10 terimli iki polinomun toplamı en çok kaç attach çağrısı yapar?
3. 5×5 matriste en az kaç sıfır olmayan öğe varsa üçlü gösterim 2B diziden fazla yer kaplar?
4. Terimleri (0,1,5), (1,0,2), (1,3,4), (2,1,1) olan 3×4 matrisin row_terms ve start_pos dizileri nedir?
5. fast_transpose tek diziyle nasıl yazılır? Hangi sıralama algoritmasına benzer?
6. $(x^2 + 1)(x^2 - 1)$ çarpımında PolCarp hangi terimi siler?

▶ **İpucu / cevaplar**

KAPANIŞ

Bölüm Özeti

Gösterim önemli

Katsayı dizisi seyrek polinomda israf; terim dizisi yalnız anlamlı terimleri tutar.

Sıralılık + birleştirme

padd O(n + m): sıralı iki diziyi iki işaretçiyle yürümek.

Bağlı liste

Terim sayısı bilinmiyorsa esnek; PolCarp aynı üsleri birleştirir, sıfırları siler.

Üçlüler

⟨row, col, value⟩ + başlık M[0]; t < rows·cols/3 iken kazançlı.

Transpoz

transpose O(cols·t); fast_transpose say–yerleştir ile O(cols + t).

Çarpım

Önce B^T, sonra satır·satır birleştirme; CSR aynı "satır başı" fikrinin standart biçimi.

KAPANIŞ

Kaynaklar ve Devam

- E. Horowitz, S. Sahni, S. Anderson-Freed, *Fundamentals of Data Structures in C*, 2. bs., 2.4–2.5 (polinom, seyrek matris), 4.4 ve 4.7 (bağlı polinom ve seyrek matris).
- Y. Saad, *Iterative Methods for Sparse Linear Systems*, 3.4 (CSR/CSC biçimleri).

← Bölüm 4: Bağlı Listeler

Ders sayfası

Bölüm 6: Yığınlar

Son

Bölüm 5: Polinomlar ve Seyrek Matrisler

Sonraki bölüm: Bölüm 6: Yığınlar · Ders sayfası