Bölüm 18: En Kısa Yol ve Aktivite Ağları

C ile Veri Yapıları ve Algoritmalar

Kaba kuvvet · Dijkstra · Öncüller dizisi · Bellman-Ford · Floyd-Warshall · Topolojik sıra · Kritik yol

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Modellemek

Şehir/demiryolu ve ağ gecikmesi problemlerini ağırlıklı yönlü çizge olarak kurmak.

Dijkstra

Tek kaynaktan tüm hedeflere en kısa yolu açgözlü yöntemle bulmak; $O(n^2)$ kodu izlemek ve yolu öncüller dizisiyle yazdırmak.

Genişletmek

Negatif kenarda Bellman-Ford'u, tüm çiftler için Floyd-Warshall'ı uygulamak.

Topolojik sıra

AoV ağlarında öncül ilişkisine uyan doğrusal sırayı sayaç + yığın ile bulmak, döngüyü yakalamak.

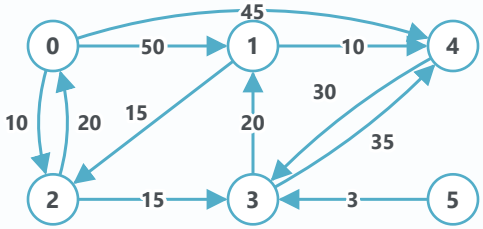
Zaman hesabı

AoE ağlarında en erken/en geç başlama zamanlarını ileri ve geri çözümlemeyle hesaplamak.

Kritik yol

Bolluğu (slack) bulup kritik aktiviteleri ve kritik yolu belirlemek.

En Kısa Yol Problemi (Shortest Path)



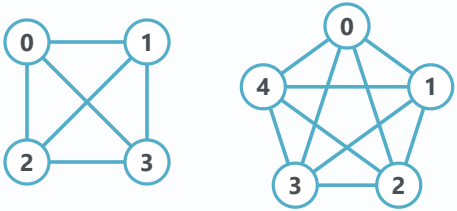
Horowitz–Sahni örneği

Kaynak	Hedef	Maliyet
0	5	Yol yok!
0	2	10
0	3	25
0	1	45
0	4	45

- Düğümler şehir, kenarlar iki şehir arasındaki demiryolu uzaklığı: bir şehirden diğerine **en kısa yol** hangisi?
- Ya da kenarlar bir ağda paket gecikmesi (trafik sıkışıklığı): bu verilere göre bir **router** nasıl yazılmalı?
- İki soru: 1) x'ten y'ye **yol var mı?** 2) Birden çok yol varsa **en kısası** hangisi?
- Tüm düğümlere birer kez uğrayıp başa dönmek (**gezgin satıcı**) farklı ve çok daha zor bir problemdir.

EN KISA YOL

Tek Kaynaktan Tek Hedefe: Kaba Kuvvet



K4 düzlemsel; K5 düzlemsel değil

K4 (0 → 3), n = 4 için 5 yol: 0-1-2-3, 0-1-3, 0-2-1-3, 0-2-3, 0-3

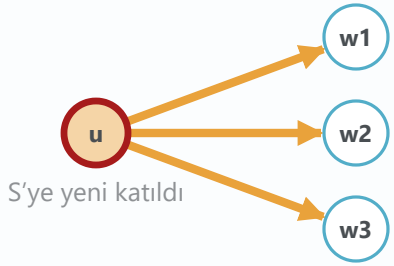
K5 (0 → 4), n = 5 için 1 + 3 + 6 + 6 = 16 yol: 0-1-2-3-4, 0-1-2-4, 0-1-3-2-4, 0-1-3-4, 0-1-4, 0-2-1-3-4, 0-2-1-4, 0-2-3-1-4, 0-2-3-4, 0-2-4, 0-3-1-2-4, 0-3-1-4, 0-3-2-1-4, 0-3-2-4, 0-3-4, 0-4

Tam çizgede yol sayısı: **1 + (n−2) + (n−2)(n−3) + ... + (n−2)!** ≈ 1 + O(n) + O(n^2) + ... ≈ **O(n^n)** (kaba ama doğru modelleme). Tek bir hedef için bile!

Hocadan: "Görüldüğü gibi kaba kuvvet yöntemiyle tek bir kaynaktan tek bir hedefe en kısa yolu bulurken bile karmaşıklık O(n^n) oldu... Elbette mantık doğru, ama böyle çözmeyeceğiz."

Etkin Çözüme Doğru: Açgözlü (Greedy) Fikir

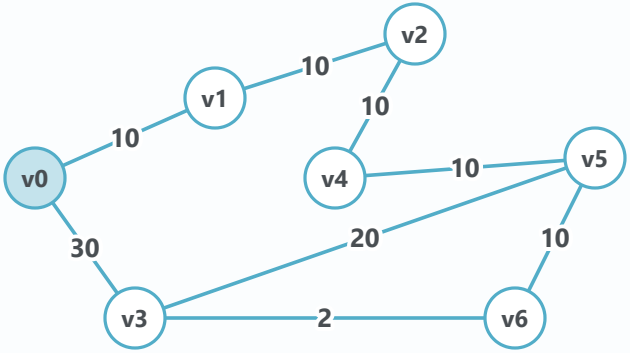
- **S**: en kısa yolu kesinleşmiş düğümler (başta yalnız kaynak 0).
- w ∉ S için `uzaklik[w]` : 0'dan başlayıp yalnızca **S'deki** düğümlerden geçerek w'de biten en kısa yolun uzunluğu.
- `uzaklik[u]` en küçük olan u seçilir ve S'ye eklenir.
- u'nun eklenmesi, u üzerinden geçen **yeni yollar** açar: `uzaklik[u] + maliyet(<u,w>) < uzaklik[w]` ise uzaklik[w] günlenir (u–w arasında ara düğüm yok).



u'nun komşularına u üzerinden yeni yollar

Bu, Edsger Dijkstra'nın (1959) algoritmasıdır. Kenar ağırlıkları **negatif olmamalıdır** (bkz. Bellman-Ford).

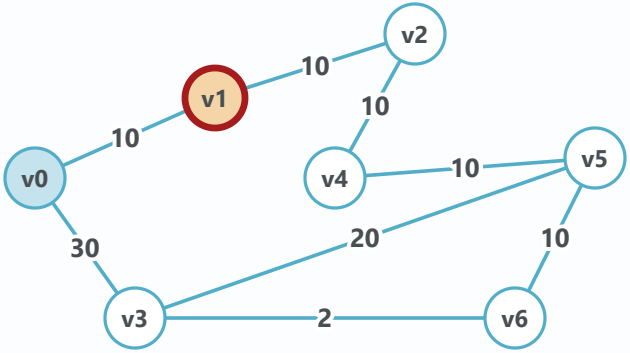
Adım Adım: v0’dan Tüm Düğümlere



	gidildi	uzaklık
v0	T	0
v1	F	10
v2	F	∞
v3	F	30
v4	F	∞
v5	F	∞
v6	F	∞

S = {v0}. Komşuların uzaklıkları doğrudan kenar maliyetleri: v1 = 10, v3 = 30; diğerleri ∞.

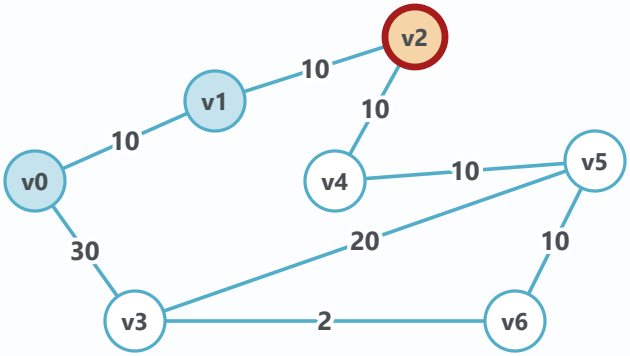
Adım Adım: v0'dan Tüm Düğümlere



	gidildi	uzaklık
v0	T	0
v1	T	10
v2	F	20
v3	F	30
v4	F	∞
v5	F	∞
v6	F	∞

En küçük uzaklıklı **v1** (10) S'ye katılır. Güncellenen: v2 = 20.

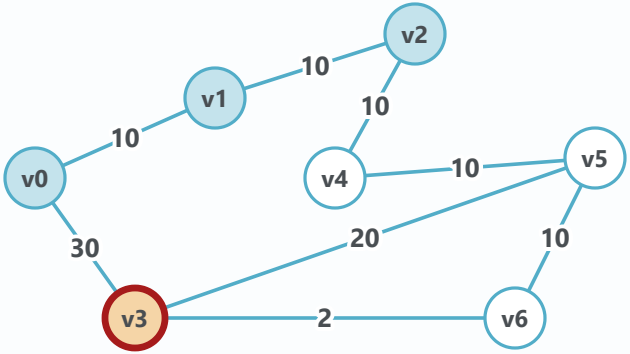
Adım Adım: v0’dan Tüm Düğümlere



	gidildi	uzaklık
v0	T	0
v1	T	10
v2	T	20
v3	F	30
v4	F	30
v5	F	∞
v6	F	∞

En küçük uzaklıklı **v2** (20) S’ye katılır. Güncellenen: v4 = 30.

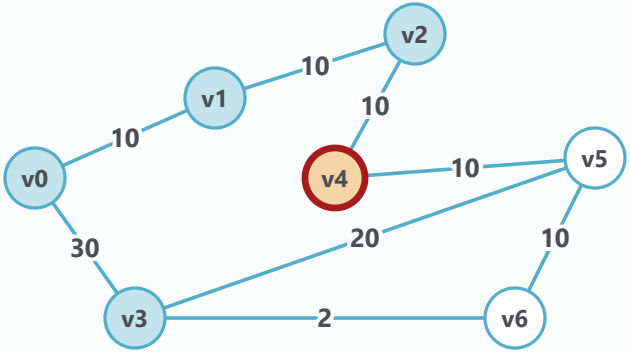
Adım Adım: v0'dan Tüm Düğümlere



	gidildi	uzaklık
v0	T	0
v1	T	10
v2	T	20
v3	T	30
v4	F	30
v5	F	50
v6	F	32

En küçük uzaklıklı **v3** (30) S'ye katılır. Güncellenen: v5 = 50, v6 = 32.

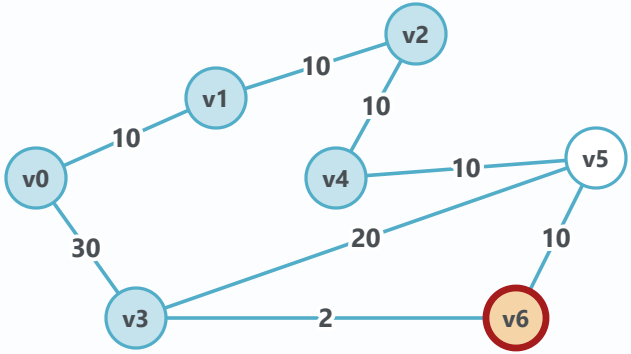
Adım Adım: v0'dan Tüm Düğümlere



	gidildi	uzaklık
v0	T	0
v1	T	10
v2	T	20
v3	T	30
v4	T	30
v5	F	40
v6	F	32

En küçük uzaklıklı **v4** (30) S'ye katılır. Güncellenen: v5 = 40. v5'e v4 üzerinden daha kısa yol bulundu (50 → 40)!

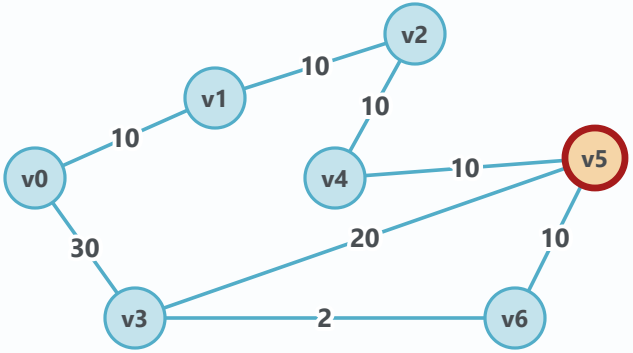
Adım Adım: v0'dan Tüm Düğümlere



	gidildi	uzaklık
v0	T	0
v1	T	10
v2	T	20
v3	T	30
v4	T	30
v5	F	40
v6	T	32

En küçük uzaklıklı **v6** (32) S'ye katılır. Güncelleme yok. Sıra numaraya göre değil uzaklığa göre belirlenir; 32 < 40 olduğu için önce **v6** seçilir.

Adım Adım: v0'dan Tüm Düğümlere



	gidildi	uzaklık
v0	T	0
v1	T	10
v2	T	20
v3	T	30
v4	T	30
v5	T	40
v6	T	32

En küçük uzaklıklı **v5** (40) S'ye katılır. Güncelleme yok.

C Kodu: Maliyet Matrisi ve minimumBul

```c
#include <limits.h>
#define MAX_DUGUM 6
#define SONSUZ 1000      /* baglanti yok */
int maliyet[][MAX_DUGUM] = {
    {   0,   50,  10, 1000,   45, 1000},
    {1000,    0,  15, 1000,   10, 1000},
    {  20, 1000,   0,   15, 1000, 1000},
    {1000,   20, 1000,   0,   35, 1000},
    {1000, 1000, 1000,  30,    0, 1000},   /* eksik satir */
    {1000, 1000, 1000,   3, 1000,    0}};  /* 100 -> 1000 */
int minimumBul(int uzaklik[], int n, short gidildi[])
{
    int i, min = INT_MAX, pos = -1;
    for (i = 0; i < n; i++)
        if (uzaklik[i] < min && !gidildi[i]) {
            min = uzaklik[i];  pos = i;
        }
    return pos;
}
```

Ayrıntılar:

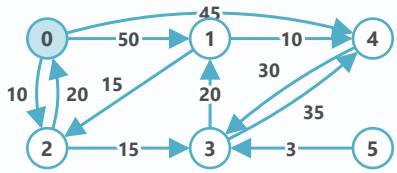
- `!gidildi[w]` yazılmalı; `!gidildi(w)` diziyi fonksiyon gibi çağırır.
- `INT_MAX` için `<limits.h>`; FALSE/TRUE tanımları.
- `minimumBul` −1 dönebilir (işaretsiz düğüm kalmamışsa); çağıran denetlemeli. Erişilemeyen düğüm de seçilebilir: `SONSUZ` (1000) < `INT_MAX`.

EnKisaYolBul(): Ana Yordam

```
void EnKisaYolBul(int v, int maliyet[][MAX_DUGUM],
                  int uzaklik[], int onceki[],
                  int n, short gidildi[])
{
    int i, u, w;
    for (i = 0; i < n; i++) {                  /* A1 */
        gidildi[i] = FALSE;
        uzaklik[i] = maliyet[v][i];
        onceki[i] = v;              /* yol icin tutulur */
    }
    gidildi[v] = TRUE;  uzaklik[v] = 0;
    for (i = 2; i < n; i++) {         /* n-2 tur */
        u = minimumBul(uzaklik, n, gidildi);
        if (u < 0) break;             /* kalan yok */
        gidildi[u] = TRUE;
        for (w = 0; w < n; w++)
            if (!gidildi[w] &&
                uzaklik[u] + maliyet[u][w] < uzaklik[w]) {
                uzaklik[w] = uzaklik[u] + maliyet[u][w];
                onceki[w] = u;
            }
    }
}
```

- **A1:** başlangıç değerleri — uzaklıklar doğrudan kenar maliyetleri.
- **A2:** dış döngü n − 2 tur: son kalan düğümün uzaklığı zaten kesindir.
- **A3:** `minimumBul` O(n).
- **A4:** tüm w'ler için gevşetme (*relaxation*) O(n).
- `onceki[]` : yolu geri izlemek için tutulur.

Kod İzleme: `EnKisaYolBul(0, …)`



	0	1	2	3	4	5
uzaklik	0	50	10	∞	45	∞
gidildi	T	F	F	F	F	F
onceki	0	0	0	0	0	0

A1 sonrası: uzaklik = 0'ın satırı (1000 = ∞); onceki[] = 0. Kaynak işaretlenir.

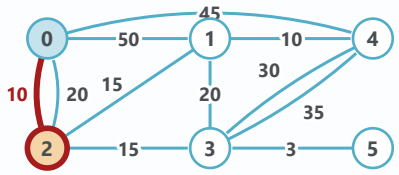
Adım 1 / 10

```
▶ gidildi[v] = TRUE;  uzaklik[v] = 0;
2 for (i = 2; i < n; i++) {            /* n-2 tur */
3     u = minimumBul(uzaklik, n, gidildi);
4     if (u < 0) break;                /* kalan yok */
5     gidildi[u] = TRUE;
6     for (w = 0; w < n; w++)
7         if (!gidildi[w] &&
8             uzaklik[u] + maliyet[u][w] < uzaklik[w]) {
9             uzaklik[w] = uzaklik[u] + maliyet[u][w];
10            onceki[w] = u;
11        }
12 }
```

DEĞİŞKENLER

`v = 0` `u = –`

Kod İzleme: `EnKisaYolBul(0, …)`



	0	1	2	3	4	5
uzaklik	0	50	10	∞	45	∞
gidildi	T	F	T	F	F	F
onceki	0	0	0	0	0	0

`minimumBul` : işaretsizler arasında en küçük uzaklık **2** (10). Kesinleşti.

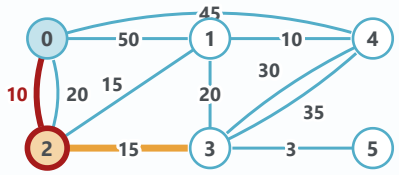
Adım 2 / 10

```
1  gidildi[v] = TRUE;  uzaklik[v] = 0;
2  for (i = 2; i < n; i++) {            /* n-2 tur */
▶      u = minimumBul(uzaklik, n, gidildi);
4      if (u < 0) break;                /* kalan yok */
▶      gidildi[u] = TRUE;
6      for (w = 0; w < n; w++)
7          if (!gidildi[w] &&
8              uzaklik[u] + maliyet[u][w] < uzaklik[w]) {
9              uzaklik[w] = uzaklik[u] + maliyet[u][w];
10             onceki[w] = u;
11         }
12 }
```

DEĞİŞKENLER

`v = 0` `u = 2` `uzaklik[u] = 10`

Kod İzleme: `EnKisaYolBul(0, …)`



	0	1	2	3	4	5
uzaklik	0	50	10	25	45	∞
gidildi	T	F	T	F	F	F
onceki	0	0	0	2	0	0

w = 3: 10 + 15 = 25 < eski değer → `uzaklik[3] = 25` , `onceki[3] = 2` .

Adım 3 / 10

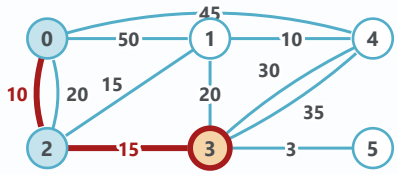
```
1  gidildi[v] = TRUE;  uzaklik[v] = 0;
2  for (i = 2; i < n; i++) {            /* n-2 tur */
3      u = minimumBul(uzaklik, n, gidildi);
4      if (u < 0) break;                /* kalan yok */
5      gidildi[u] = TRUE;
6      for (w = 0; w < n; w++)
▶          if (!gidildi[w] &&
▶              uzaklik[u] + maliyet[u][w] < uzaklik[w]) {
▶              uzaklik[w] = uzaklik[u] + maliyet[u][w];
▶              onceki[w] = u;
11         }
12 }
```

DEĞİŞKENLER

`v = 0` `u = 2` `w = 3`

`uzaklik[u] = 10`

Kod İzleme: `EnKisaYolBul(0, …)`



	0	1	2	3	4	5
uzaklik	0	50	10	25	45	∞
gidildi	T	F	T	T	F	F
onceki	0	0	0	2	0	0

`minimumBul` : işaretsizler arasında en küçük uzaklık **3** (25). Kesinleşti.

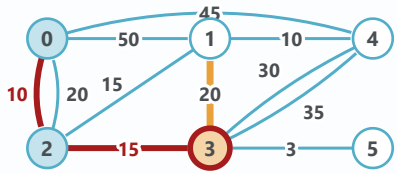
Adım 4 / 10

```
1  gidildi[v] = TRUE;  uzaklik[v] = 0;
2  for (i = 2; i < n; i++) {            /* n-2 tur */
▶      u = minimumBul(uzaklik, n, gidildi);
4      if (u < 0) break;                /* kalan yok */
▶      gidildi[u] = TRUE;
6      for (w = 0; w < n; w++)
7          if (!gidildi[w] &&
8              uzaklik[u] + maliyet[u][w] < uzaklik[w]) {
9              uzaklik[w] = uzaklik[u] + maliyet[u][w];
10             onceki[w] = u;
11         }
12 }
```

DEĞİŞKENLER

`v = 0` `u = 3` `uzaklik[u] = 25`

Kod İzleme: `EnKisaYolBul(0, …)`



	0	1	2	3	4	5
uzaklik	0	45	10	25	45	∞
gidildi	T	F	T	T	F	F
onceki	0	3	0	2	0	0

w = 1: 25 + 20 = 45 < eski değer → `uzaklik[1] = 45`, `onceki[1] = 3`.

Adım 5 / 10

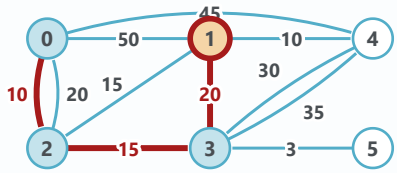
```
1  gidildi[v] = TRUE;  uzaklik[v] = 0;
2  for (i = 2; i < n; i++) {            /* n-2 tur */
3      u = minimumBul(uzaklik, n, gidildi);
4      if (u < 0) break;                /* kalan yok */
5      gidildi[u] = TRUE;
6      for (w = 0; w < n; w++)
▶          if (!gidildi[w] &&
▶              uzaklik[u] + maliyet[u][w] < uzaklik[w]) {
▶              uzaklik[w] = uzaklik[u] + maliyet[u][w];
▶              onceki[w] = u;
11         }
12 }
```

DEĞİŞKENLER

`v = 0` `u = 3` `w = 1`

`uzaklik[u] = 25`

Kod İzleme: `EnKisaYolBul(0, …)`



	0	1	2	3	4	5
uzaklik	0	45	10	25	45	∞
gidildi	T	T	T	T	F	F
onceki	0	3	0	2	0	0

`minimumBul` : işaretsizler arasında en küçük uzaklık **1** (45) — 1 ile 4 eşit (45); `<` olduğu için küçük indis 1 seçilir. Kesinleşti.

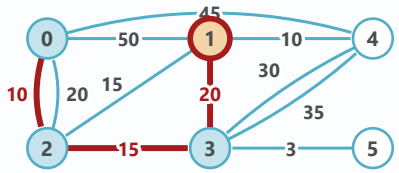
Adım 6 / 10

```
1  gidildi[v] = TRUE;  uzaklik[v] = 0;
2  for (i = 2; i < n; i++) {            /* n-2 tur */
▶      u = minimumBul(uzaklik, n, gidildi);
4      if (u < 0) break;                /* kalan yok */
▶      gidildi[u] = TRUE;
6      for (w = 0; w < n; w++)
7          if (!gidildi[w] &&
8              uzaklik[u] + maliyet[u][w] < uzaklik[w]) {
9              uzaklik[w] = uzaklik[u] + maliyet[u][w];
10             onceki[w] = u;
11         }
12 }
```

DEĞİŞKENLER

`v = 0` `u = 1` `uzaklik[u] = 45`

Kod İzleme: `EnKisaYolBul(0, …)`



	0	1	2	3	4	5
uzaklik	0	45	10	25	45	∞
gidildi	T	T	T	T	F	F
onceki	0	3	0	2	0	0

1 üzerinden hiçbir uzaklık kısalmadı (ör. 45 + 10 = 55 > 45).

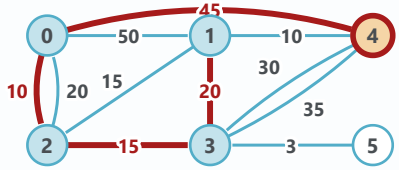
Adım 7 / 10

```
1  gidildi[v] = TRUE;  uzaklik[v] = 0;
2  for (i = 2; i < n; i++) {            /* n-2 tur */
3      u = minimumBul(uzaklik, n, gidildi);
4      if (u < 0) break;                /* kalan yok */
5      gidildi[u] = TRUE;
▶      for (w = 0; w < n; w++)
▶          if (!gidildi[w] &&
8              uzaklik[u] + maliyet[u][w] < uzaklik[w]) {
9              uzaklik[w] = uzaklik[u] + maliyet[u][w];
10             onceki[w] = u;
11         }
12 }
```

DEĞİŞKENLER

`v = 0` `u = 1` `uzaklik[u] = 45`

Kod İzleme: `EnKisaYolBul(0, …)`



	0	1	2	3	4	5
uzaklik	0	45	10	25	45	∞
gidildi	T	T	T	T	T	F
onceki	0	3	0	2	0	0

`minimumBul` : işaretsizler arasında en küçük uzaklık **4** (45). Kesinleşti.

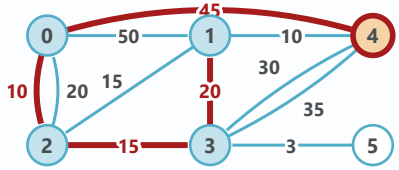
Adım 8 / 10

```
1  gidildi[v] = TRUE;  uzaklik[v] = 0;
2  for (i = 2; i < n; i++) {            /* n-2 tur */
▶      u = minimumBul(uzaklik, n, gidildi);
4      if (u < 0) break;                /* kalan yok */
▶      gidildi[u] = TRUE;
6      for (w = 0; w < n; w++)
7          if (!gidildi[w] &&
8              uzaklik[u] + maliyet[u][w] < uzaklik[w]) {
9              uzaklik[w] = uzaklik[u] + maliyet[u][w];
10             onceki[w] = u;
11         }
12 }
```

DEĞİŞKENLER

`v = 0` `u = 4` `uzaklik[u] = 45`

Kod İzleme: `EnKisaYolBul(0, …)`



	0	1	2	3	4	5
uzaklik	0	45	10	25	45	∞
gidildi	T	T	T	T	T	F
onceki	0	3	0	2	0	0

4 üzerinden hiçbir uzaklık kısalmadı (4'ten tek çıkış <4,3>, 3 zaten işaretli).

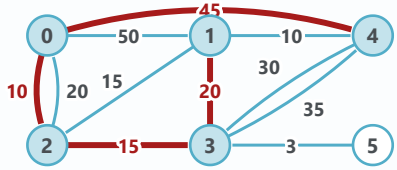
Adım 9 / 10

```
1  gidildi[v] = TRUE;  uzaklik[v] = 0;
2  for (i = 2; i < n; i++) {            /* n-2 tur */
3      u = minimumBul(uzaklik, n, gidildi);
4      if (u < 0) break;                /* kalan yok */
5      gidildi[u] = TRUE;
▶      for (w = 0; w < n; w++)
▶          if (!gidildi[w] &&
8              uzaklik[u] + maliyet[u][w] < uzaklik[w]) {
9              uzaklik[w] = uzaklik[u] + maliyet[u][w];
10             onceki[w] = u;
11         }
12 }
```

DEĞİŞKENLER

`v = 0` `u = 4` `uzaklik[u] = 45`

Kod İzleme: `EnKisaYolBul(0, …)`



	0	1	2	3	4	5
uzaklik	0	45	10	25	45	∞
gidildi	T	T	T	T	T	F
onceki	0	3	0	2	0	0

n − 2 = 4 tur bitti. Sonuç: 2 → 10, 3 → 25, 1 → 45, 4 → 45; 5'e **yol yok** (1000 kaldı).

Adım 10 / 10

```
1  gidildi[v] = TRUE;  uzaklik[v] = 0;
▶ for (i = 2; i < n; i++) {            /* n-2 tur */
3      u = minimumBul(uzaklik, n, gidildi);
4      if (u < 0) break;               /* kalan yok */
5      gidildi[u] = TRUE;
6      for (w = 0; w < n; w++)
7          if (!gidildi[w] &&
8              uzaklik[u] + maliyet[u][w] < uzaklik[w]) {
9              uzaklik[w] = uzaklik[u] + maliyet[u][w];
10             onceki[w] = u;
11         }
12 }
```

DEĞİŞKENLER

`v = 0` `u = −`

Analiz ve Yapıları İyileştirmek

Kitaptaki algoritma

$n + n[n + n] = n + 2n^2 =$ **$O(n^2)$** — sırasıyla A1, A2, A3, A4'ten.

Min öncelikli kuyruk

Uzaklıklar min-heap'te: A3'teki n yerine **log n**. Komşuluk listesiyle A4'teki n yerine **derece(v)**.

İyileştirilmiş toplam

O(n lg n) sanmak sık yapılan bir hatadır; her gevşetme de heap'i güncellediği için doğrusu **O((n + e) log n)** (Fibonacci heap: O(e + n log n)).

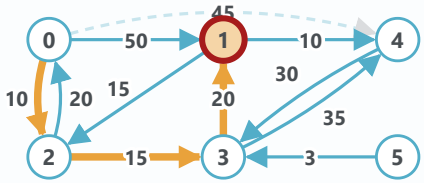
Üç algoritma

Kaba kuvvet $O(n^n)$ · matris $O(n^2)$ · heap + liste O((n + e) log n). Yoğun çizgede $O(n^2)$ sürüm hâlâ iyidir.

Hocadan: "Minimum öncelikli kuyruğu öğrendikten sonra dizide en küçük öğeyi aramak ne kadar hantal kalıyor değil mi? Ne kadar abes! Ne kadar ayıp!"

"Adam matristeki en büyük öğeyi bulana kadar, ben v0'dan tüm düğümlere olan en kısa yolları buluyorum."

Kod İzleme: `yol_yaz(onceki, 0, 1)`



onceki[]

0	1	2	3	4	5
0	3	0	2	0	0

1 ← 3 ← 2 ← 0 (tersten izlenir)

h = 1 ≠ 0: önce öncülün yolu yazılmalı → `yol_yaz(…, 3)`.

Adım 1 / 8

```
1 void yol_yaz(int onceki[], int s, int h)
2 {
▶       if (h == s) {
4           printf("%d", s);
5           return;
6       }
▶       yol_yaz(onceki, s, onceki[h]);   /* once oncul */
8       printf(" -> %d", h);
9 }
```

DEĞİŞKENLER

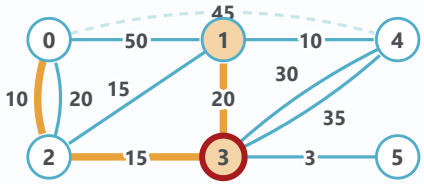
`s = 0` `h = 1` `onceki[h] = 3`

ÇAĞRI YIĞINI

`yol_yaz(1)` `main()`

ÇIKTI

Kod İzleme: `yol_yaz(onceki, 0, 1)`



onceki[]

0	1	2	3	4	5
0	3	0	2	0	0

1 ← 3 ← 2 ← 0 (tersten izlenir)

h = 3 ≠ 0: önce öncülün yolu yazılmalı → `yol_yaz(…, 2)`.

Adım 2 / 8

```
1 void yol_yaz(int onceki[], int s, int h)
2 {
▶       if (h == s) {
4           printf("%d", s);
5           return;
6       }
▶       yol_yaz(onceki, s, onceki[h]);   /* once oncul */
8       printf(" -> %d", h);
9 }
```

DEĞİŞKENLER

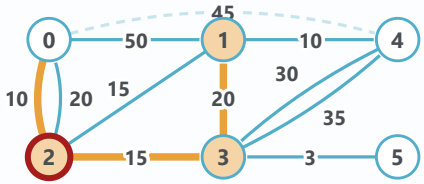
`s = 0` `h = 3` `onceki[h] = 2`

ÇAĞRI YIĞINI

`yol_yaz(3)` `yol_yaz(1)` `main()`

ÇIKTI

Kod İzleme: `yol_yaz(onceki, 0, 1)`



onceki[]

0	1	2	3	4	5
0	3	0	2	0	0

1 ← 3 ← 2 ← 0 (tersten izlenir)

h = 2 ≠ 0: önce öncülün yolu yazılmalı → `yol_yaz(…, 0)`.

Adım 3 / 8

```
1 void yol_yaz(int onceki[], int s, int h)
2 {
▶       if (h == s) {
4           printf("%d", s);
5           return;
6       }
▶       yol_yaz(onceki, s, onceki[h]);   /* once oncul */
8       printf(" -> %d", h);
9 }
```

DEĞİŞKENLER

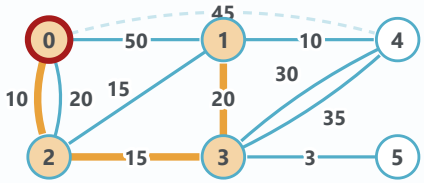
`s = 0` `h = 2` `onceki[h] = 0`

ÇAĞRI YIĞINI

`yol_yaz(2)` `yol_yaz(3)` `yol_yaz(1)` `main()`

ÇIKTI

Kod İzleme: `yol_yaz(onceki, 0, 1)`



onceki[]

0	1	2	3	4	5
0	3	0	2	0	0

1 ← 3 ← 2 ← 0 (tersten izlenir)

h = s = 0: kaynağa ulaşıldı → **0** yazılır, dönülür.

Adım 4 / 8

```
1 void yol_yaz(int onceki[], int s, int h)
2 {
▶     if (h == s) {
▶         printf("%d", s);
5         return;
6     }
7     yol_yaz(onceki, s, onceki[h]);   /* once oncul */
8     printf(" -> %d", h);
9 }
```

DEĞİŞKENLER

`s = 0` `h = 0`

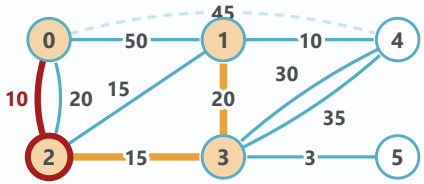
ÇAĞRI YIĞINI

`yol_yaz(0)` `yol_yaz(2)` `yol_yaz(3)` `yol_yaz(1)` `main()`

ÇIKTI

```
0
```

Kod İzleme: `yol_yaz(onceki, 0, 1)`



onceki[]

0	1	2	3	4	5
0	3	0	2	0	0

1 ← 3 ← 2 ← 0 (tersten izlenir)

`yol_yaz(2)` 'e dönüldü: " -> 2" yazılır.

Adım 5 / 8

```
1 void yol_yaz(int onceki[], int s, int h)
2 {
3     if (h == s) {
4         printf("%d", s);
5         return;
6     }
7     yol_yaz(onceki, s, onceki[h]);   /* once oncul */
▶     printf(" -> %d", h);
9 }
```

DEĞİŞKENLER

`s = 0` `h = 2`

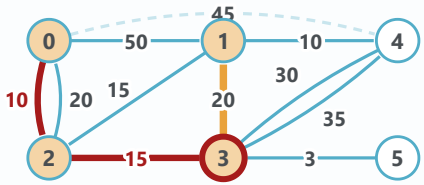
ÇAĞRI YIĞINI

`yol_yaz(2)` `yol_yaz(3)` `yol_yaz(1)` `main()`

ÇIKTI

```
0 -> 2
```

Kod İzleme: `yol_yaz(onceki, 0, 1)`



onceki[]

0	1	2	3	4	5
0	3	0	2	0	0

1 ← 3 ← 2 ← 0 (tersten izlenir)

`yol_yaz(3)` 'e dönüldü: " -> 3" yazılır.

Adım 6 / 8

```
1 void yol_yaz(int onceki[], int s, int h)
2 {
3     if (h == s) {
4         printf("%d", s);
5         return;
6     }
7     yol_yaz(onceki, s, onceki[h]);   /* once oncul */
▶     printf(" -> %d", h);
9 }
```

DEĞİŞKENLER

`s = 0` `h = 3`

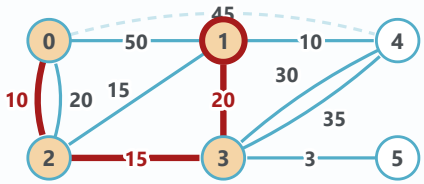
ÇAĞRI YIĞINI

`yol_yaz(3)` `yol_yaz(1)` `main()`

ÇIKTI

```
0 -> 2 -> 3
```

Kod İzleme: `yol_yaz(onceki, 0, 1)`



onceki[]

0	1	2	3	4	5
0	3	0	2	0	0

1 ← 3 ← 2 ← 0 (tersten izlenir)

`yol_yaz(1)` 'e dönüldü: " -> 1" yazılır.

Adım 7 / 8

```
1 void yol_yaz(int onceki[], int s, int h)
2 {
3     if (h == s) {
4         printf("%d", s);
5         return;
6     }
7     yol_yaz(onceki, s, onceki[h]);   /* once oncul */
▶     printf(" -> %d", h);
9 }
```

DEĞİŞKENLER

`s = 0` `h = 1`

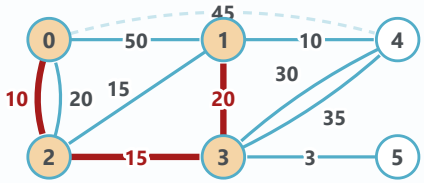
ÇAĞRI YIĞINI

`yol_yaz(1)` `main()`

ÇIKTI

```
0 -> 2 -> 3 -> 1
```

Kod İzleme: `yol_yaz(onceki, 0, 1)`



onceki[]

0	1	2	3	4	5
0	3	0	2	0	0

1 ← 3 ← 2 ← 0 (tersten izlenir)

Yol: **0 -> 2 -> 3 -> 1**, maliyet 10 + 15 + 20 = 45. Önce hedeften geriye yürüdük, yazarken çağrı yığını sırayı düzeltti.

Adım 8 / 8

```
1 void yol_yaz(int onceki[], int s, int h)
2 {
3     if (h == s) {
4         printf("%d", s);
5         return;
6     }
7     yol_yaz(onceki, s, onceki[h]);   /* once oncul */
▶     printf(" -> %d", h);
9 }
```

DEĞİŞKENLER

`s = 0` `h = 1`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
0 -> 2 -> 3 -> 1
```

Yolu Bulmak: Öncüller Dizisi

> **Hocadan:** “Şimdi maliyeti bulduk da yolu nasıl bulacağız? Maliyet bulundu, sıra yolda! … Hangi yoldan gidileceği belli olmadığından ileri doğru (forward) çözülemez. Tek bir diziyle çözülebilir: öncüller dizisi… Tersten kullanılacak.”

Neden tek dizi yeter?

Her düğümün en kısa yolu, öncülünün en kısa yolu + bir kenardır (optimal alt yapı). Yollar bir **ağaç** oluşturur: en kısa yol ağacı.

Güncelleme

Gevşetme başarılıysa `onceki[w] = u`. Maliyet değiştikçe öncül de değişir.

Tersten kullanmak

onceki[] ile hedeften kaynağa yürünür; özyineleme ya da yığın ile ters çevrilip yazılır.

Erişilemeyen düğüm

`uzaklik[h] ≥ SONSUZ` ise “yol yok” yazılmalı; onceki[] anlamsızdır.

DİJKSTRA

Yazılım Mühendisliği Kavramları

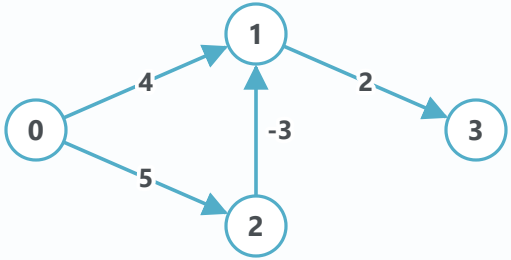
> **Hocadan:** "Vizedeki soruyu tek bir özyinelemeli yordamla çözmeye çalışanlar olmuş... Bazıları da yordamı `AgirlikBul(…);` diye çağırmış. Bu ne? "Hocam ne var ki orda?" Bu ne döner? "int döner." Hani nereye dönüyor? Dönen değer havada duruyor. "Ah, ben onu unutmuşum.""

Çözümler **modüler, basit, okunabilir, tekrar kullanılabilir, bakımı kolay** olmalıdır. El altında bir Yazılım Mühendisliği kitabı bulunmalı:

Reliability	**Usability**	**Maintainability**	**Portability**	**Integrity**
Flexibility	**Testability**	**Efficiency**	**Interoperability**	**Understandability**
Modularity	**Traceability**	**Modifiability**	**Correctness**	**Expandability**

Bu bölümdeki kodlar da bu yüzden küçük yordamlara bölündü: `minimumBul`, `yol_yaz`, `kume_bul` ...

Negatif Kenarda Dijkstra Neden Yanılır? (ek konu)



Düğüm	Dijkstra	Doğru
0	0	0
1	4	2
2	5	5
3	6	4

Dijkstra 1'i 4 ile kesinleştirir (ondan küçük aday yok). Sonra 2 → 1 kenarı (−3) 1'e 2 maliyetle ulaşır; ama 1 artık işaretlidir. 3 de yanlış kalır.

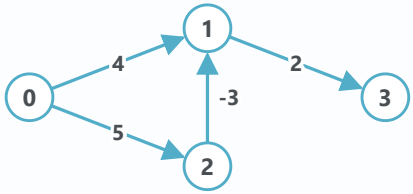
Varsayım

Dijkstra, seçilen düğüme daha sonra **daha ucuz** bir yoldan gelinemeyeceğini varsayar; bu yalnızca $w \geq 0$ iken doğrudur.

Çözüm

Bellman-Ford: tüm kenarları $n - 1$ tur gevşet. Negatif **döngü** varsa en kısa yol tanımsızdır; n. turda hâlâ gevşeme olursa bunu yakalar.

Kod İzleme: Bellman-Ford (kaynak 0)



d[]
0	1	2	3
0	∞	∞	∞

K[] (kenar listesi, bu sırayla)

0→1 (4)	0→2 (5)	1→3 (2)	2→1 (-3)

d[] = ∞, kaynak d[0] = 0.

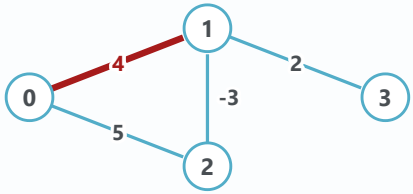
Adım 1 / 11

```
1  for (i = 0; i < n; i++) d[i] = SONSUZ;
▶  d[s] = 0;
3  for (t = 1; t <= n - 1; t++)          /* n-1 tur */
4      for (j = 0; j < e; j++) {
5          u = K[j].u;  v = K[j].v;
6          if (d[u] != SONSUZ && d[u] + K[j].w < d[v])
7              d[v] = d[u] + K[j].w;
8      }
9  for (j = 0; j < e; j++)               /* n. tur */
10     if (d[K[j].u] != SONSUZ &&
11         d[K[j].u] + K[j].w < d[K[j].v]) return -1;
12 return 0;              /* -1: negatif dongu var */
```

DEĞİŞKENLER

t = – j = –

Kod İzleme: Bellman-Ford (kaynak 0)



d[]

0	1	2	3
0	4	∞	∞

K[] (kenar listesi, bu sırayla)

0→1 (4)	0→2 (5)	1→3 (2)	2→1 (-3)

Tur 1, 0→1 (4): d[0] + 4 = 4 daha küçük → **d[1] = 4**.

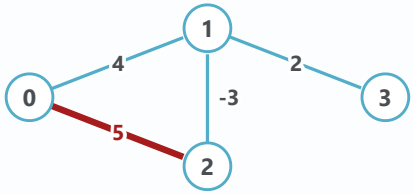
Adım 2 / 11

```
1  for (i = 0; i < n; i++) d[i] = SONSUZ;
2  d[s] = 0;
3  for (t = 1; t <= n - 1; t++)          /* n-1 tur */
4      for (j = 0; j < e; j++) {
5          u = K[j].u;  v = K[j].v;
▶          if (d[u] != SONSUZ && d[u] + K[j].w < d[v])
▶              d[v] = d[u] + K[j].w;
8      }
9  for (j = 0; j < e; j++)               /* n. tur */
10     if (d[K[j].u] != SONSUZ &&
11         d[K[j].u] + K[j].w < d[K[j].v]) return -1;
12 return 0;              /* -1: negatif dongu var */
```

DEĞİŞKENLER

t = 1 | j = 0 | u = 0 | v = 1

Kod İzleme: Bellman-Ford (kaynak 0)



d[]

0	1	2	3
0	4	5	∞

K[] (kenar listesi, bu sırayla)

0→1 (4)	0→2 (5)	1→3 (2)	2→1 (-3)

Tur 1, 0→2 (5): d[0] + 5 = 5 daha küçük → **d[2] = 5**.

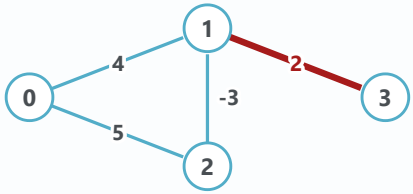
Adım 3 / 11

```
1  for (i = 0; i < n; i++) d[i] = SONSUZ;
2  d[s] = 0;
3  for (t = 1; t <= n - 1; t++)          /* n-1 tur */
4      for (j = 0; j < e; j++) {
5          u = K[j].u;  v = K[j].v;
▶          if (d[u] != SONSUZ && d[u] + K[j].w < d[v])
▶              d[v] = d[u] + K[j].w;
8      }
9  for (j = 0; j < e; j++)               /* n. tur */
10     if (d[K[j].u] != SONSUZ &&
11         d[K[j].u] + K[j].w < d[K[j].v]) return -1;
12 return 0;              /* -1: negatif dongu var */
```

DEĞİŞKENLER

t = 1 | j = 1 | u = 0 | v = 2

Kod İzleme: Bellman-Ford (kaynak 0)



d[]

0	1	2	3
0	4	5	6

K[] (kenar listesi, bu sırayla)

0→1 (4)	0→2 (5)	1→3 (2)	2→1 (-3)

Tur 1, 1→3 (2): d[1] + 2 = 6 daha küçük → **d[3] = 6**.

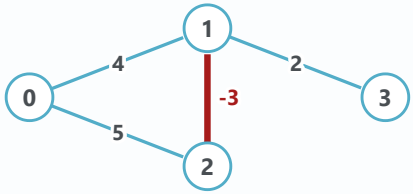
Adım 4 / 11

```
1  for (i = 0; i < n; i++) d[i] = SONSUZ;
2  d[s] = 0;
3  for (t = 1; t <= n - 1; t++)          /* n-1 tur */
4      for (j = 0; j < e; j++) {
5          u = K[j].u;  v = K[j].v;
▶          if (d[u] != SONSUZ && d[u] + K[j].w < d[v])
▶              d[v] = d[u] + K[j].w;
8      }
9  for (j = 0; j < e; j++)               /* n. tur */
10     if (d[K[j].u] != SONSUZ &&
11         d[K[j].u] + K[j].w < d[K[j].v]) return -1;
12 return 0;              /* -1: negatif dongu var */
```

DEĞİŞKENLER

t = 1 | j = 2 | u = 1 | v = 3

Kod İzleme: Bellman-Ford (kaynak 0)



d[]

0	1	2	3
0	2	5	6

K[] (kenar listesi, bu sırayla)

0→1 (4)	0→2 (5)	1→3 (2)	2→1 (-3)

Tur 1, 2→1 (-3): d[2] + (−3) = 2 daha küçük → **d[1] = 2**.

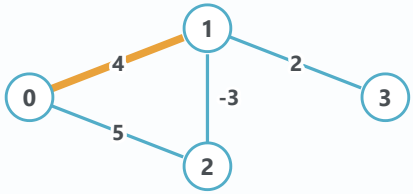
Adım 5 / 11

```
1  for (i = 0; i < n; i++) d[i] = SONSUZ;
2  d[s] = 0;
3  for (t = 1; t <= n - 1; t++)          /* n-1 tur */
4      for (j = 0; j < e; j++) {
5          u = K[j].u;  v = K[j].v;
▶          if (d[u] != SONSUZ && d[u] + K[j].w < d[v])
▶              d[v] = d[u] + K[j].w;
8      }
9  for (j = 0; j < e; j++)               /* n. tur */
10     if (d[K[j].u] != SONSUZ &&
11         d[K[j].u] + K[j].w < d[K[j].v]) return -1;
12 return 0;              /* -1: negatif dongu var */
```

DEĞİŞKENLER

`t = 1` `j = 3` `u = 2` `v = 1`

Kod İzleme: Bellman-Ford (kaynak 0)



d[]

0	1	2	3
0	2	5	6

K[] (kenar listesi, bu sırayla)

0→1 (4)	0→2 (5)	1→3 (2)	2→1 (-3)

Tur 2, 0→1 (4): 0 + 4 = 4 ≥ d[1] = 2, değişmez.

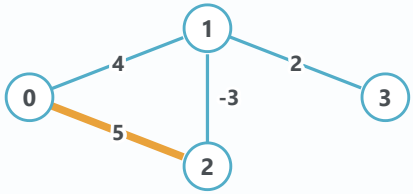
Adım 6 / 11

```
1  for (i = 0; i < n; i++) d[i] = SONSUZ;
2  d[s] = 0;
3  for (t = 1; t <= n - 1; t++)          /* n-1 tur */
4      for (j = 0; j < e; j++) {
5          u = K[j].u;  v = K[j].v;
▶          if (d[u] != SONSUZ && d[u] + K[j].w < d[v])
7              d[v] = d[u] + K[j].w;
8      }
9  for (j = 0; j < e; j++)               /* n. tur */
10     if (d[K[j].u] != SONSUZ &&
11         d[K[j].u] + K[j].w < d[K[j].v]) return -1;
12 return 0;              /* -1: negatif dongu var */
```

DEĞİŞKENLER

t = 2 | j = 0 | u = 0 | v = 1

Kod İzleme: Bellman-Ford (kaynak 0)



d[]

0	1	2	3
0	2	5	6

K[] (kenar listesi, bu sırayla)

0→1 (4)	0→2 (5)	1→3 (2)	2→1 (-3)

Tur 2, 0→2 (5): 0 + 5 = 5 ≥ d[2] = 5, değişmez.

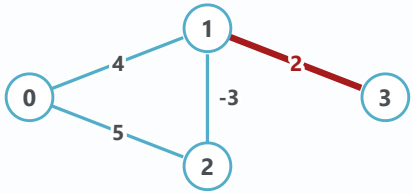
Adım 7 / 11

```
1  for (i = 0; i < n; i++) d[i] = SONSUZ;
2  d[s] = 0;
3  for (t = 1; t <= n - 1; t++)          /* n-1 tur */
4      for (j = 0; j < e; j++) {
5          u = K[j].u;  v = K[j].v;
▶          if (d[u] != SONSUZ && d[u] + K[j].w < d[v])
7              d[v] = d[u] + K[j].w;
8      }
9  for (j = 0; j < e; j++)               /* n. tur */
10     if (d[K[j].u] != SONSUZ &&
11         d[K[j].u] + K[j].w < d[K[j].v]) return -1;
12 return 0;              /* -1: negatif dongu var */
```

DEĞİŞKENLER

t = 2 | j = 1 | u = 0 | v = 2

Kod İzleme: Bellman-Ford (kaynak 0)



d[]

0	1	2	3
0	2	5	4

K[] (kenar listesi, bu sırayla)

0→1 (4)	0→2 (5)	1→3 (2)	2→1 (-3)

Tur 2, 1→3 (2): d[1] + 2 = 4 daha küçük → **d[3] = 4**.

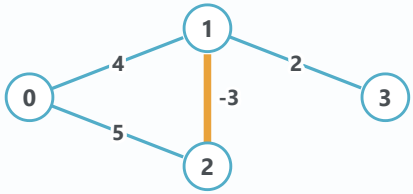
Adım 8 / 11

```
1  for (i = 0; i < n; i++) d[i] = SONSUZ;
2  d[s] = 0;
3  for (t = 1; t <= n - 1; t++)          /* n-1 tur */
4      for (j = 0; j < e; j++) {
5          u = K[j].u;  v = K[j].v;
▶          if (d[u] != SONSUZ && d[u] + K[j].w < d[v])
▶              d[v] = d[u] + K[j].w;
8      }
9  for (j = 0; j < e; j++)               /* n. tur */
10     if (d[K[j].u] != SONSUZ &&
11         d[K[j].u] + K[j].w < d[K[j].v]) return -1;
12 return 0;              /* -1: negatif dongu var */
```

DEĞİŞKENLER

t = 2 | j = 2 | u = 1 | v = 3

Kod İzleme: Bellman-Ford (kaynak 0)



d[]

0	1	2	3
0	2	5	4

K[] (kenar listesi, bu sırayla)

0→1 (4)	0→2 (5)	1→3 (2)	2→1 (-3)

Tur 2, 2→1 (-3): 5 + (−3) = 2 ≥ d[1] = 2, değişmez.

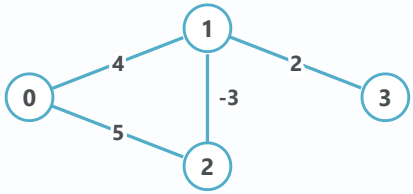
Adım 9 / 11

```
1  for (i = 0; i < n; i++) d[i] = SONSUZ;
2  d[s] = 0;
3  for (t = 1; t <= n - 1; t++)          /* n-1 tur */
4      for (j = 0; j < e; j++) {
5          u = K[j].u;  v = K[j].v;
▶          if (d[u] != SONSUZ && d[u] + K[j].w < d[v])
7              d[v] = d[u] + K[j].w;
8      }
9  for (j = 0; j < e; j++)               /* n. tur */
10     if (d[K[j].u] != SONSUZ &&
11         d[K[j].u] + K[j].w < d[K[j].v]) return -1;
12 return 0;              /* -1: negatif dongu var */
```

DEĞİŞKENLER

t = 2 | j = 3 | u = 2 | v = 1

Kod İzleme: Bellman-Ford (kaynak 0)



d[]

0	1	2	3
0	2	5	4

K[] (kenar listesi, bu sırayla)

0→1 (4)	0→2 (5)	1→3 (2)	2→1 (-3)

Tur 3: hiçbir kenar gevşemez (d zaten kesin). Genelde n − 1 tur gerekir: en kısa yol en çok n − 1 kenarlıdır.

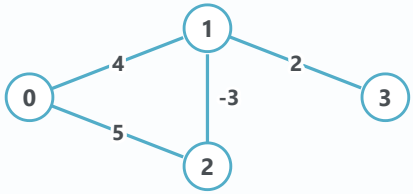
Adım 10 / 11

```
1  for (i = 0; i < n; i++) d[i] = SONSUZ;
2  d[s] = 0;
▶ for (t = 1; t <= n - 1; t++)          /* n-1 tur */
4      for (j = 0; j < e; j++) {
5          u = K[j].u;  v = K[j].v;
6          if (d[u] != SONSUZ && d[u] + K[j].w < d[v])
7              d[v] = d[u] + K[j].w;
8      }
9  for (j = 0; j < e; j++)              /* n. tur */
10     if (d[K[j].u] != SONSUZ &&
11         d[K[j].u] + K[j].w < d[K[j].v]) return -1;
12 return 0;              /* -1: negatif dongu var */
```

DEĞİŞKENLER

t = 3 j = –

Kod İzleme: Bellman-Ford (kaynak 0)



d[]

0	1	2	3
0	2	5	4

K[] (kenar listesi, bu sırayla)

0→1 (4)	0→2 (5)	1→3 (2)	2→1 (-3)

n. turda da gevşeme yok → negatif döngü yok, return 0. Sonuç d = [0, 2, 5, 4]. Maliyet **O(n·e)**.

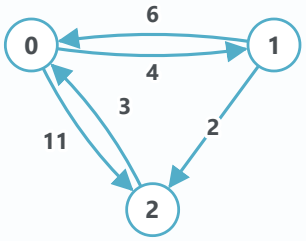
Adım 11 / 11

```
1  for (i = 0; i < n; i++) d[i] = SONSUZ;
2  d[s] = 0;
3  for (t = 1; t <= n - 1; t++)          /* n-1 tur */
4      for (j = 0; j < e; j++) {
5          u = K[j].u;  v = K[j].v;
6          if (d[u] != SONSUZ && d[u] + K[j].w < d[v])
7              d[v] = d[u] + K[j].w;
8      }
▶ for (j = 0; j < e; j++)                /* n. tur */
10     if (d[K[j].u] != SONSUZ &&
11         d[K[j].u] + K[j].w < d[K[j].v]) return -1;
▶ return 0;                 /* -1: negatif dongu var */
```

DEĞİŞKENLER

t = – j = –

Kod İzleme: Floyd-Warshall



A (k = −1)

	0	1	2
0	0	4	11
1	6	0	2
2	3	∞	0

A^{-1} = maliyet matrisi (Horowitz örneği; 2 → 1 doğrudan kenar yok).

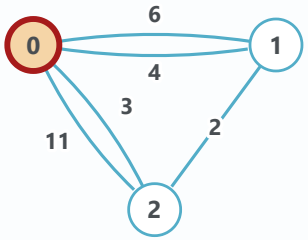
Adım 1 / 8

```
1 for (i = 0; i < n; i++)
2     for (j = 0; j < n; j++)
▶         A[i][j] = maliyet[i][j];
4 for (k = 0; k < n; k++)                /* ara dugum k */
5     for (i = 0; i < n; i++)
6         for (j = 0; j < n; j++)
7             if (A[i][k] + A[k][j] < A[i][j])
8                 A[i][j] = A[i][k] + A[k][j];
```

DEĞİŞKENLER

k = – i = – j = –

Kod İzleme: Floyd-Warshall



A (k = 0)

	0	1	2
0	0	4	11
1	6	0	2
2	3	∞	0

k = 0: artık yollar **0** düğümünden (ve daha küçük numaralılardan) geçebilir.

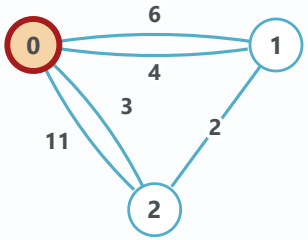
Adım 2 / 8

```
1 for (i = 0; i < n; i++)
2     for (j = 0; j < n; j++)
3         A[i][j] = maliyet[i][j];
▶ for (k = 0; k < n; k++)              /* ara dugum k */
5     for (i = 0; i < n; i++)
6         for (j = 0; j < n; j++)
7             if (A[i][k] + A[k][j] < A[i][j])
8                 A[i][j] = A[i][k] + A[k][j];
```

DEĞİŞKENLER

k = 0 | i = – | j = –

Kod İzleme: Floyd-Warshall



A (k = 0)

	0	1	2
0	0	4	11
1	6	0	2
2	3	7	0

A[2][0] + A[0][1] = 3 + 4 = 7 < ∞ → **A[2][1] = 7**.

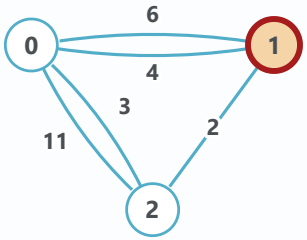
Adım 3 / 8

```
1 for (i = 0; i < n; i++)
2     for (j = 0; j < n; j++)
3         A[i][j] = maliyet[i][j];
4 for (k = 0; k < n; k++)              /* ara dugum k */
5     for (i = 0; i < n; i++)
6         for (j = 0; j < n; j++)
▶             if (A[i][k] + A[k][j] < A[i][j])
▶                 A[i][j] = A[i][k] + A[k][j];
```

DEĞİŞKENLER

k = 0 | i = 2 | j = 1

Kod İzleme: Floyd-Warshall



A (k = 1)

	0	1	2
0	0	4	11
1	6	0	2
2	3	7	0

k = 1: artık yollar **1** düğümünden (ve daha küçük numaralılardan) geçebilir.

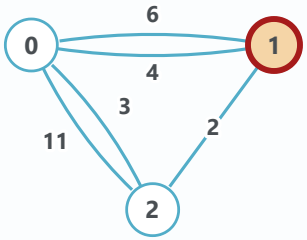
Adım 4 / 8

```
1 for (i = 0; i < n; i++)
2     for (j = 0; j < n; j++)
3         A[i][j] = maliyet[i][j];
▶ for (k = 0; k < n; k++)                /* ara dugum k */
5     for (i = 0; i < n; i++)
6         for (j = 0; j < n; j++)
7             if (A[i][k] + A[k][j] < A[i][j])
8                 A[i][j] = A[i][k] + A[k][j];
```

DEĞİŞKENLER

k = 1 | i = – | j = –

Kod İzleme: Floyd-Warshall



A (k = 1)

	0	1	2
0	0	4	6
1	6	0	2
2	3	7	0

A[0][1] + A[1][2] = 4 + 2 = 6 < 11 → **A[0][2] = 6**.

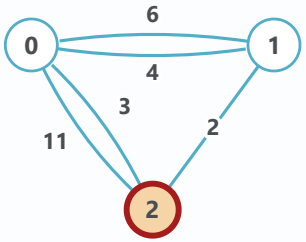
Adım 5 / 8

```
1 for (i = 0; i < n; i++)
2     for (j = 0; j < n; j++)
3         A[i][j] = maliyet[i][j];
4 for (k = 0; k < n; k++)              /* ara dugum k */
5     for (i = 0; i < n; i++)
6         for (j = 0; j < n; j++)
▶             if (A[i][k] + A[k][j] < A[i][j])
▶                 A[i][j] = A[i][k] + A[k][j];
```

DEĞİŞKENLER

`k = 1` `i = 0` `j = 2`

Kod İzleme: Floyd-Warshall



A (k = 2)

	0	1	2
0	0	4	6
1	6	0	2
2	3	7	0

k = 2: artık yollar **2** düğümünden (ve daha küçük numaralılardan) geçebilir.

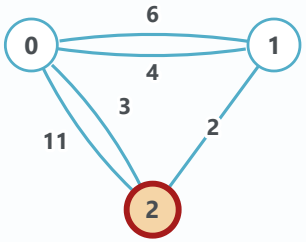
Adım 6 / 8

```
1 for (i = 0; i < n; i++)
2     for (j = 0; j < n; j++)
3         A[i][j] = maliyet[i][j];
▶ for (k = 0; k < n; k++)                /* ara dugum k */
5     for (i = 0; i < n; i++)
6         for (j = 0; j < n; j++)
7             if (A[i][k] + A[k][j] < A[i][j])
8                 A[i][j] = A[i][k] + A[k][j];
```

DEĞİŞKENLER

k = 2 | i = – | j = –

Kod İzleme: Floyd-Warshall



A (k = 2)

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

A[1][2] + A[2][0] = 2 + 3 = 5 < 6 → **A[1][0] = 5**.

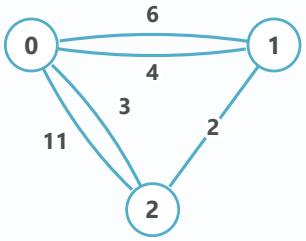
Adım 7 / 8

```
1 for (i = 0; i < n; i++)
2     for (j = 0; j < n; j++)
3         A[i][j] = maliyet[i][j];
4 for (k = 0; k < n; k++)              /* ara dugum k */
5     for (i = 0; i < n; i++)
6         for (j = 0; j < n; j++)
▶             if (A[i][k] + A[k][j] < A[i][j])
▶                 A[i][j] = A[i][k] + A[k][j];
```

DEĞİŞKENLER

k = 2 | i = 1 | j = 0

Kod İzleme: Floyd-Warshall



A (k = 2), sonuç

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

Bitti: FA = tüm çiftler arası en kısa uzaklıklar. Üç iç içe döngü → **O(n³)**; negatif kenar olabilir, negatif döngü olamaz.

Adım 8 / 8

```
1 for (i = 0; i < n; i++)
2     for (j = 0; j < n; j++)
3         A[i][j] = maliyet[i][j];
▶ for (k = 0; k < n; k++)              /* ara dugum k */
5     for (i = 0; i < n; i++)
6         for (j = 0; j < n; j++)
7             if (A[i][k] + A[k][j] < A[i][j])
8                 A[i][j] = A[i][k] + A[k][j];
```

DEĞİŞKENLER

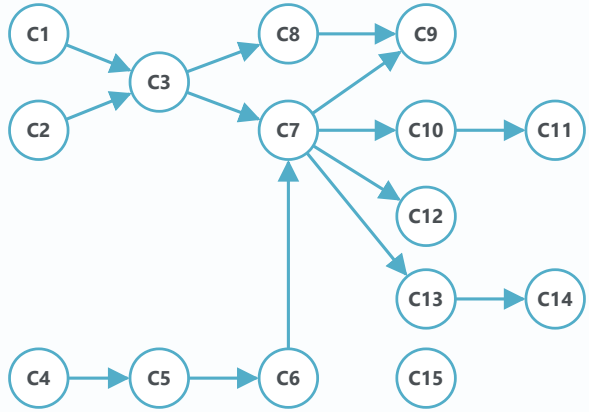
k = 3 | i = – | j = –

En Kısa Yol: Hangi Algoritma, Ne Zaman?

Problem	Algoritma	Zaman	Koşul
Ağırlıksız, tek kaynak	BFS	$O(n + e)$	tüm kenarlar eşit
Tek kaynak, $w \geq 0$	Dijkstra (matris)	$O(n^2)$	yoğun çizge
Tek kaynak, $w \geq 0$	Dijkstra (heap + liste)	$O((n + e) \log n)$	seyrek çizge
Tek kaynak, negatif kenar	Bellman-Ford	$O(n \cdot e)$	negatif döngüyü yakalar
DAG, tek kaynak	Topolojik sırada gevşetme	$O(n + e)$	döngü yok (aktivite ağları!)
Tüm çiftler	Floyd-Warshall	$O(n^3)$	negatif döngü yok
Tüm çiftler, seyrek	$n \times$ Dijkstra	$O(n (n + e) \log n)$	$w \geq 0$

Aktivite Ağları: Proje = Aktiviteler

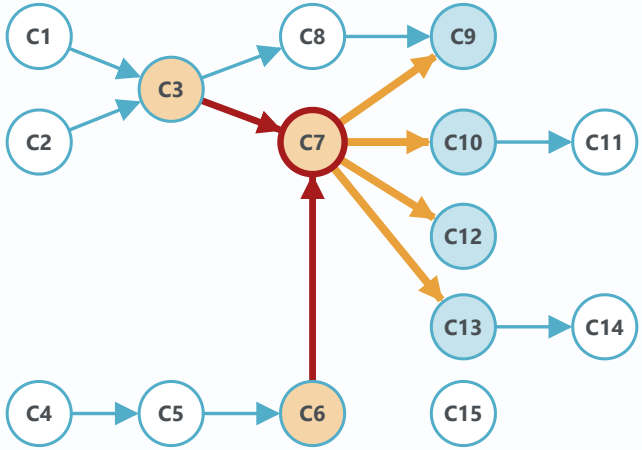
Kod	Ders	Ön gereksinim
C1	Programming I	–
C2	Discrete Math.	–
C3	Data Structures	C1, C2
C4	Calculus I	–
C5	Calculus II	C4
C6	Linear Algebra	C5
C7	Analysis of Algorithms	C3, C6
C8	Assembly Lang.	C3
C9	Operating Systems	C7, C8
C10	Programming Langs.	C7
C11	Compiler Design	C10
C12	Artificial Intelligence	C7
C13	Computational Theory	C7
C14	Parallel Algorithms	C13
C15	Numerical Analysis	–



Proje, alt işler (**aktiviteler**) bitince biter: mezuniyet = dersler. İnşaatta: temel atılır, 21 gün sonra 2. kat… Proje **ne zaman biter**? Hangi işler **kritik**? Hangilerinde **bolluk** var?

Düğümde Aktivite Ağları (AoV) ve Öncül İlişkisi

- **AoV** (*activity on vertex/node*): aktiviteler **düğüm**, öncül ilişkileri (*precedence relation*) **kenar**: i dersi j'nin öncülüyse <i, j>.
- C3 ve C6, C7'nin **yakın öncülleridir** (*immediate predecessors*).
- C9, C10, C12, C13, C7'nin **yakın ardıllarıdır** (*immediate successors*).
- C14, C3'ün bir ardılıdır ama **yakın** ardılı değildir.
- Öncül ilişkisi geçişlidir ve döngüsüz olmalıdır: A, B'nin; B de A'nın öncülü olamaz.



Hocadan: "Satır arası: Yöneylem, sayısal analiz gibi derslerin kalkmış olması öğrencilerin matematiksel olarak zayıf kalmasına yol açıyor."

Tanım: Topolojik Sıra (Topological Order)

Düğümlerin öyle bir **doğrusal sırası** ki, i, j'nin öncülüyse sırada i, j'den **önce** gelir. Topolojik sıra **birden çok** olabilir; çizgede döngü varsa hiç yoktur.

Soğuk Savaş yıllarında füze (Polaris) üretimini hızlandırmak için geliştirilen planlama çalışmalarına (PERT, 1958) dayanır. Derslerin iki geçerli topolojik sırası (programla sınandı):

```
C1, C2, C4, C5, C3, C6, C8, C7, C10, C13, C12, C14, C15, C11, C9
C4, C5, C2, C1, C6, C3, C8, C15, C7, C9, C10, C11, C12, C13, C14
```

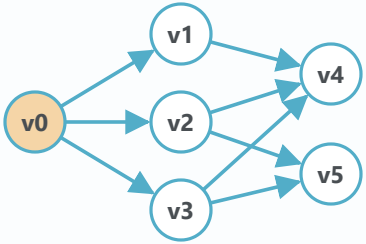
Sağlama

Her kenar <i, j> için i'nin sıradaki yeri j'ninkinden küçük mü? 14 kenarın hepsi sağlanıyor.

Kullanım

Ders planı, derleme sırası (make), görev zamanlama, hesap tablosu hücreleri, en erken zamanlar.

Topolojik Sıra Bulmak: Öncülsüz Düğümü Sil



sıra: —

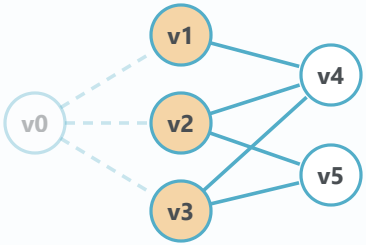
öncülü kalmayan: v0

a) v0'ın öncülü yok → ilk seçilecek düğüm.

Adım 1 / 7

Her seferinde öncülü olmayan düğümlerden biri alınır, yazılır, çıkan kenarları silinir; sona kadar böyle devam edilir. Öncülsüz düğüm kalmadıysa ama düğüm varsa: **döngü**.

Topolojik Sıra Bulmak: Öncülsüz Düğümü Sil



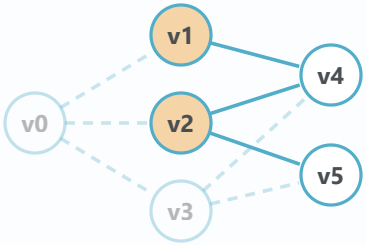
sıra: v0

öncülü kalmayan: v1, v2, v3

b) v0 yazıldı, çıkan kenarları silindi. Öncülü kalmayanlardan (turuncu) **herhangi biri** seçilebilir. Ör. v3 seçilirse <v3,v4> ve <v3,v5> silinir.

Her seferinde öncülü olmayan düğümlerden biri alınır, yazılır, çıkan kenarları silinir; sona kadar böyle devam edilir. Öncülsüz düğüm kalmadıysa ama düğüm varsa: **döngü**.

Topolojik Sıra Bulmak: Öncülsüz Düğümü Sil



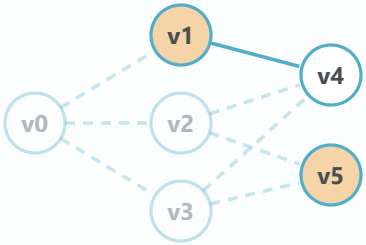
sıra: v0, v3

öncülü kalmayan: v1, v2

c) v3 yazıldı, çıkan kenarları silindi. Öncülü kalmayanlardan (turuncu) **herhangi biri** seçilebilir.

Her seferinde öncülü olmayan düğümlerden biri alınır, yazılır, çıkan kenarları silinir; sona kadar böyle devam edilir. Öncülsüz düğüm kalmadıysa ama düğüm varsa: **döngü**.

Topolojik Sıra Bulmak: Öncülsüz Düğümü Sil



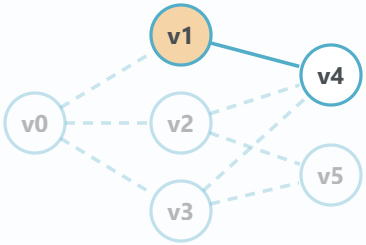
sıra: v0, v3, v2

öncülü kalmayan: v1, v5

d) v2 yazıldı, çıkan kenarları silindi. Öncülü kalmayanlardan (turuncu) **herhangi biri** seçilebilir.

Her seferinde öncülü olmayan düğümlerden biri alınır, yazılır, çıkan kenarları silinir; sona kadar böyle devam edilir. Öncülsüz düğüm kalmadıysa ama düğüm varsa: **döngü**.

Topolojik Sıra Bulmak: Öncülsüz Düğümü Sil



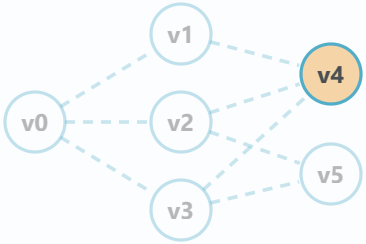
sıra: v0, v3, v2, v5

öncülü kalmayan: v1

e) v5 yazıldı, çıkan kenarları silindi. Öncülü kalmayanlardan (turuncu) **herhangi biri** seçilebilir.

Her seferinde öncülü olmayan düğümlerden biri alınır, yazılır, çıkan kenarları silinir; sona kadar böyle devam edilir. Öncülsüz düğüm kalmadıysa ama düğüm varsa: **döngü**.

Topolojik Sıra Bulmak: Öncülsüz Düğümü Sil



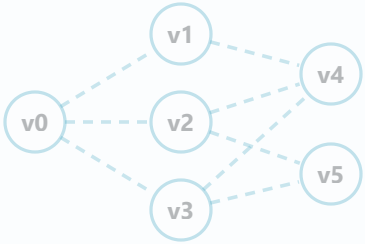
sıra: v0, v3, v2, v5, v1

öncülü kalmayan: v4

f) v1 yazıldı, çıkan kenarları silindi. Öncülü kalmayanlardan (turuncu) **herhangi biri** seçilebilir.

Her seferinde öncülü olmayan düğümlerden biri alınır, yazılır, çıkan kenarları silinir; sona kadar böyle devam edilir. Öncülsüz düğüm kalmadıysa ama düğüm varsa: **döngü**.

Topolojik Sıra Bulmak: Öncülsüz Düğümü Sil



sıra: v0, v3, v2, v5, v1, v4

öncülü kalmayan: —

g) Olası bir topolojik sıra: **v0, v3, v2, v5, v1, v4**.

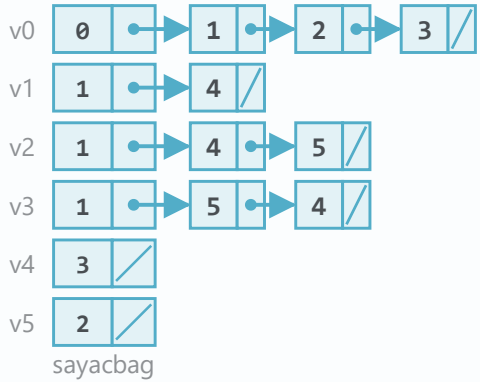
Adım 7 / 7

Her seferinde öncülü olmayan düğümlerden biri alınır, yazılır, çıkan kenarları silinir; sona kadar böyle devam edilir. Öncülsüz düğüm kalmadıysa ama düğüm varsa: **döngü**.

Veri Yapısı: Başlıkta Öncül Sayacı

```
typedef struct oge *oge_gost;
typedef struct oge {
    int dugum;
    oge_gost bag;
} oge;
typedef struct {
    int sayac;      /* <i,j> icin: basi j olan kenar sayisi */
    oge_gost bag;
} basoge;
basoge baslik[MAX_DUGUM];
```

- Düğümler 0..n−1 numaralı.
- `sayac` : düğümün öncül (giriş kenarı) sayısı; 0 ise yazılabilir.
- Kenarlar okunurken `baslik[j].sayac++` .



v0..v5 örneğinin sayaçlı listesi

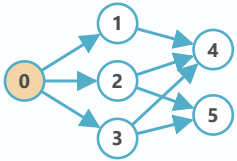
Topolojik Sıra Bulma Yordamı

```
void TopolojikSira(basoge baslik[], int n)
{
    int i, j, k, ust = -1;
    oge_gost p;
    for (i = 0; i < n; i++)      /* sayac 0: yigina */
        if (!baslik[i].sayac) {
            baslik[i].sayac = ust;  ust = i; }
    for (i = 0; i < n; i++) {
        if (ust == -1) { printf("dongu"); break; }
        j = ust;  ust = baslik[ust].sayac;
        printf("%d ", j);
        for (p = baslik[j].bag; p; p = p->bag) {
            k = p->dugum;
            if (--baslik[k].sayac == 0) {
                baslik[k].sayac = ust;  ust = k; }
        }
    }
}
```

> **Hocadan:** "Ortada yığın falan yok, ama "yığına ekle", "yığından al" diye açıklama satırları yazdık. Açıklama satırları işte bu yüzden önemlidir. Yığın kullanıyor da nasıl kullanıyor?"

- Öncülü kalmayan düğümler `sayac` alanları üzerinden **peş peşe bağlanır**: sayac artık "yığında altımdaki düğüm". Kümelerdeki **ebeveyn dizisine** benzer.
- `sayac--; if (!sayac)` ikilisi burada `--sayac == 0` olarak yazıldı (aynı anlam).

Kod İzleme: `TopolojikSira(baslik, 6)`



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	1	1	1	3	2

`ust = 0`

yığın (üst solda): v0

v0'ın sayacı 0 → yığına: `sayac = ust` (−1: yığının dibi), `ust = 0`.

Adım 1 / 11

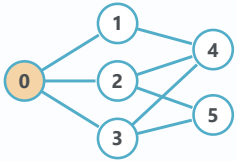
```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
▶      if (!baslik[i].sayac) {
▶          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
6      j = ust;  ust = baslik[ust].sayac;
7      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
10         if (--baslik[k].sayac == 0) {
11             baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

DEĞİŞKENLER

`i = 0` `ust = 0`

ÇIKTI

Kod İzleme: `TopolojikSira(baslik, 6)`



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	1	1	1	3	2

ust = 0

yığın (üst solda): v0

İlk döngü bitti: yalnız v0 öncülsüz. Diğer sayaçlar öncül sayısını tutuyor.

Adım 2 / 11

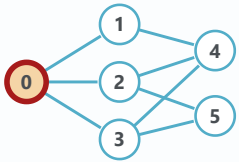
```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
▶      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
6      j = ust;  ust = baslik[ust].sayac;
7      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
10         if (--baslik[k].sayac == 0) {
11             baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

DEĞİŞKENLER

`i = 6` `ust = 0`

ÇIKTI

Kod İzleme: `TopolojikSira(baslik, 6)`



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	1	1	1	3	2

`ust = -1` yığın (üst solda): boş

Yığından al: **j = 0**, `ust = sayac[0] = −1`. v0 yazılır.

Adım 3 / 11

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
▶      j = ust;  ust = baslik[ust].sayac;
▶      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
10         if (--baslik[k].sayac == 0) {
11             baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

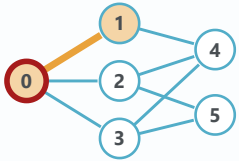
DEĞİŞKENLER

`i = 0` `j = 0` `ust = -1`

ÇIKTI

```
0
```

Kod İzleme: `TopolojikSira(baslik, 6)`



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	1	3	2

`ust = 1` yığın (üst solda): v1

<0,1>: v1'in sayacı 0'a düştü → yığına: `sayac[1] = -1` , `ust = 1` .

Adım 4 / 11

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
6      j = ust;  ust = baslik[ust].sayac;
7      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
▶          if (--baslik[k].sayac == 0) {
▶              baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

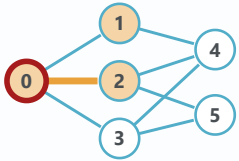
DEĞİŞKENLER

`i = 0` `j = 0` `k = 1` `ust = 1`

ÇIKTI

```
0
```

Kod İzleme: `TopolojikSira(baslik, 6)`



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	1	3	2

`ust = 2`

yığın (üst solda): v2, v1

<0,2>: v2'nin sayacı 0'a düştü → yığına: `sayac[2] = 1` , `ust = 2` .

Adım 5 / 11

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
6      j = ust;  ust = baslik[ust].sayac;
7      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
▶          if (--baslik[k].sayac == 0) {
▶              baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

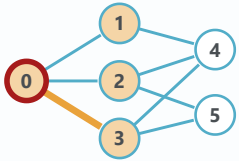
DEĞİŞKENLER

`i = 0` `j = 0` `k = 2` `ust = 2`

ÇIKTI

```
0
```

Kod İzleme: `TopolojikSira(baslik, 6)`



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	2	3	2

`ust = 3`

yığın (üst solda): v3, v2, v1

<0,3>: v3'ün sayacı 0'a düştü → yığına: `sayac[3] = 2` , `ust = 3` .

Adım 6 / 11

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
6      j = ust;  ust = baslik[ust].sayac;
7      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
▶          if (--baslik[k].sayac == 0) {
▶              baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

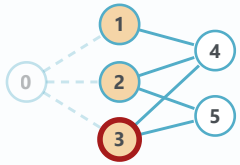
DEĞİŞKENLER

`i = 0` `j = 0` `k = 3` `ust = 3`

ÇIKTI

```
0
```

Kod İzleme: `TopolojikSira(baslik, 6)`



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	2	3	2

`ust = 2`

yığın (üst solda): v2, v1

Yığından al: **j = 3**, `ust = sayac[3] = 2`. v3 yazılır.

Adım 7 / 11

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
▶      j = ust;  ust = baslik[ust].sayac;
▶      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
10         if (--baslik[k].sayac == 0) {
11             baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

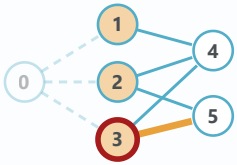
DEĞİŞKENLER

`i = 1` `j = 3` `ust = 2`

ÇIKTI

```
0 3
```

Kod İzleme: `TopolojikSira(baslik, 6)`



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	2	3	1

ust = 2 yığın (üst solda): v2, v1

<3,5>: v5'in öncül sayısı 1 azaldı → 1; henüz yazılamaz.

Adım 8 / 11

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
6      j = ust;  ust = baslik[ust].sayac;
7      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
▶          k = p->dugum;
▶          if (--baslik[k].sayac == 0) {
11             baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

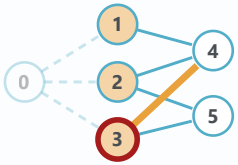
DEĞİŞKENLER

`i = 1` `j = 3` `k = 5` `ust = 2`

ÇIKTI

```
0 3
```

Kod İzleme: `TopolojikSira(baslik, 6)`



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	2	2	1

ust = 2 yığın (üst solda): v2, v1

<3,4>: v4'ün öncül sayısı 1 azaldı → 2; henüz yazılamaz.

Adım 9 / 11

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
6      j = ust;  ust = baslik[ust].sayac;
7      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
▶          k = p->dugum;
▶          if (--baslik[k].sayac == 0) {
11             baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

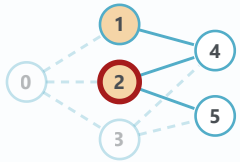
DEĞİŞKENLER

`i = 1` `j = 3` `k = 4` `ust = 2`

ÇIKTI

```
0 3
```

Kod İzleme: `TopolojikSira(baslik, 6)`



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	2	2	1

`ust = 1`

yığın (üst solda): v1

Yığından al: **j = 2**, `ust = sayac[2] = 1`. v2 yazılır.

Adım 10 / 11

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
▶      j = ust;  ust = baslik[ust].sayac;
▶      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
10         if (--baslik[k].sayac == 0) {
11             baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

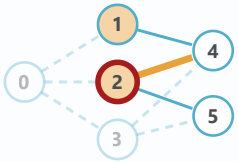
DEĞİŞKENLER

`i = 2` `j = 2` `ust = 1`

ÇIKTI

```
0 3 2
```

Kod İzleme: `TopolojikSira(baslik, 6)`



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	2	1	1

ust = 1

yığın (üst solda): v1

<2,4>: v4'ün öncül sayısı 1 azaldı → 1; henüz yazılamaz.

Adım 11 / 11

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
6      j = ust;  ust = baslik[ust].sayac;
7      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
▶          k = p->dugum;
▶          if (--baslik[k].sayac == 0) {
11             baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

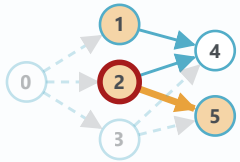
DEĞİŞKENLER

`i = 2` `j = 2` `k = 4` `ust = 1`

ÇIKTI

```
0 3 2
```

Kod İzleme: `TopolojikSira()` (devam)



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	2	1	1

`ust = 5`

yığın (üst solda): v5, v1

<2,5>: v5'in sayacı 0'a düştü → yığına: `sayac[5] = 1`, `ust = 5`.

Adım 1 / 6

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
6      j = ust;  ust = baslik[ust].sayac;
7      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
▶          if (--baslik[k].sayac == 0) {
▶              baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

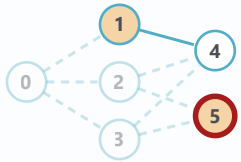
DEĞİŞKENLER

`i = 2` `j = 2` `k = 5` `ust = 5`

ÇIKTI

```
0 3 2
```

Kod İzleme: `TopolojikSira()` (devam)



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	2	1	1

`ust = 1`

yığın (üst solda): v1

Yığından al: **j = 5**, `ust = sayac[5] = 1`. v5 yazılır.

Adım 2 / 6

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
▶      j = ust;  ust = baslik[ust].sayac;
▶      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
10         if (--baslik[k].sayac == 0) {
11             baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

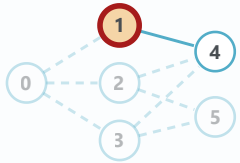
DEĞİŞKENLER

`i = 3` `j = 5` `ust = 1`

ÇIKTI

```
0 3 2 5
```

Kod İzleme: `TopolojikSira()` (devam)



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	2	1	1

`ust = -1` yığın (üst solda): boş

Yığından al: **j = 1**, `ust = sayac[1] = −1`. v1 yazılır.

Adım 3 / 6

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
▶      j = ust;  ust = baslik[ust].sayac;
▶      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
10         if (--baslik[k].sayac == 0) {
11             baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

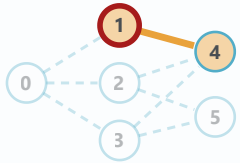
DEĞİŞKENLER

`i = 4` `j = 1` `ust = -1`

ÇIKTI

```
0 3 2 5 1
```

Kod İzleme: `TopolojikSira()` (devam)



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	2	-1	1

ust = 4 yığın (üst solda): v4

<1,4>: v4'ün sayacı 0'a düştü → yığına: `sayac[4] = -1`, `ust = 4`.

Adım 4 / 6

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
6      j = ust;  ust = baslik[ust].sayac;
7      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
▶          if (--baslik[k].sayac == 0) {
▶              baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

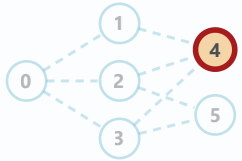
DEĞİŞKENLER

`i = 4` `j = 1` `k = 4` `ust = 4`

ÇIKTI

```
0 3 2 5 1
```

Kod İzleme: `TopolojikSira()` (devam)



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	2	-1	1

`ust = -1` yığın (üst solda): boş

Yığından al: **j = 4**, `ust = sayac[4] = −1`. v4 yazılır.

Adım 5 / 6

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
5      if (ust == -1) { printf("dongu"); break; }
▶      j = ust;  ust = baslik[ust].sayac;
▶      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
10         if (--baslik[k].sayac == 0) {
11             baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

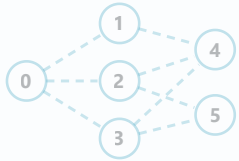
DEĞİŞKENLER

`i = 5` `j = 4` `ust = −1`

ÇIKTI

```
0 3 2 5 1 4
```

Kod İzleme: `TopolojikSira()` (devam)



baslik[].sayac

v0	v1	v2	v3	v4	v5
-1	-1	1	2	-1	1

`ust = -1` yığın (üst solda): boş

Tüm düğümler yazıldı: **0 3 2 5 1 4**. Döngü olsaydı bir turda `ust == -1` görülür, "döngü var" yazılırdı.

Adım 6 / 6

```
1  for (i = 0; i < n; i++)          /* sayac 0: yigina */
2      if (!baslik[i].sayac) {
3          baslik[i].sayac = ust;  ust = i; }
4  for (i = 0; i < n; i++) {
▶      if (ust == -1) { printf("dongu"); break; }
6      j = ust;  ust = baslik[ust].sayac;
7      printf("%d ", j);
8      for (p = baslik[j].bag; p; p = p->bag) {
9          k = p->dugum;
10         if (--baslik[k].sayac == 0) {
11             baslik[k].sayac = ust;  ust = k; }
12     }
13 }
```

DEĞİŞKENLER

`i = 6` `ust = -1`

ÇIKTI

```
0 3 2 5 1 4
```

Döngü Tespiti ve Analiz

Döngüyü de bulur

Döngü, A ve B'nin karşılıklı olarak birbirinin öncülü olmasıdır — bu bir hatadır. "Döngü varsa proje hiç bitmeyecektir." Bir turda yığın boşsa ama düğüm kalmışsa döngü vardır.

Analiz

İlk for O(n), ikinci for O(n); iç for d_j kez döner (çıkış derecesi). Toplam $O(\Sigma d_i + n)$ = **$O(n + e)$**.

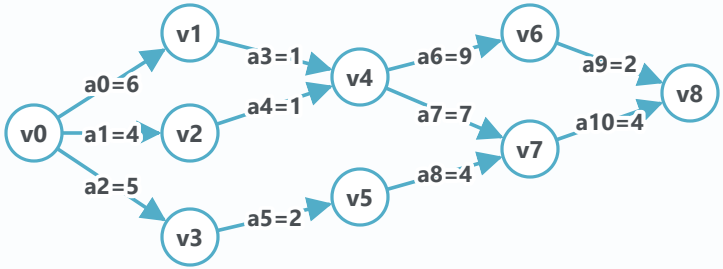
Sıra neye bağlı?

Yığına eklenme sırası komşuluk listesindeki sıraya bağlıdır; kuyruk kullanılsaydı (Kahn) başka ama yine geçerli bir sıra çıkardı.

Çizgenin yapısı

e, çizgenin bağlanışına göre O(n) ile $O(n^2)$ arasında değişir.

Kenarda Aktivite Ağları (AoE)



v0 başlangıç, v8 bitiş: 11 aktivite (kenar), 9 olay (düğüm)

AoE (activity on edge/arc)

AoV'nin tersine **yönlü kenarlar aktiviteleri** (süreleriyle), **düğümler olayları** temsil eder.

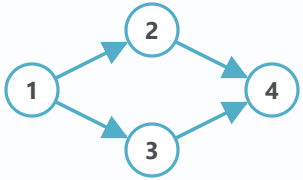
Olay (event)

Projenin tamamlanması için ulaşılması gereken ara aşama: "bitmesi gereken iş bir düğümle ifade ediliyor".

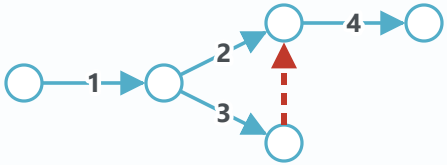
Paralellik

a0, a1, a2 paralel başlar; a3 ve a4 bitince a6, a7 başlar; a10 için a7 ve a8 bitmiş olmalıdır.

Graph Drawing: AoV'den AoE'ye, Yapay Aktivite



AoV: 4, hem 2'yi hem 3'ü bekler



yapay kenar (dummy arc), süre 0

AoV'de 4 aktivitesi 2 ve 3'ü bekler; AoE'de bu, maliyeti 0 olan bir yapay kenarla gösterilir.

Yapay aktivite (dummy activity)

Maliyeti 0 olan kenar. Ör. yukarıdaki projede a6 ve a7'nin a3, a4 **ve a5** bitince başlaması istenseydi <v5, v4> yapay aktivitesi gerekirdi.

Minimum arc problem

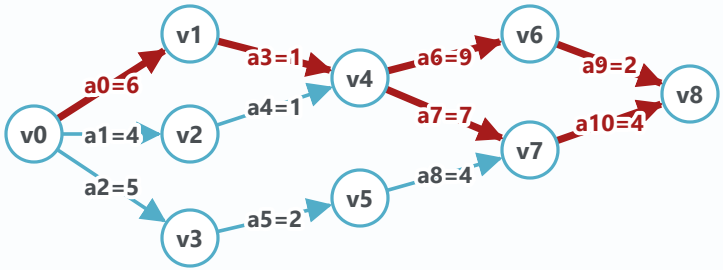
Çizgeyi **en az sayıda** yapay aktiviteyle çizme problemi: NP-zor. Üzerinde tez düzeyinde çalışmalar yapılmış bir araştırma konusudur.

Yöntemler

PERT (*Program Evaluation and Review Technique*, 1958), **CPM** (*Critical Path Method*, 1950'lerin sonu), RAMPS (*Resource Allocation and Multi-Project Scheduling*).

ERKEN/GEÇ ZAMANLAR

Kritik Yol (Critical Path)



Kritik yollar: v0-v1-v4-v6-v8 ve v0-v1-v4-v7-v8 (uzunluk 18)

- Aktivite süreleri toplandığında **en uzun** v0 → v8 yolu: **kritik yol**. Birden fazla olabilir.
- Burada 6 + 1 + 9 + 2 = 18 ve 6 + 1 + 7 + 4 = 18 → proje **en erken 18** günde biter.
- Bir aktivite, ön koşulları en kısa sürede biterse **ne zaman başlar**? Proje uzamasın diye **en geç ne zaman** başlamalı?
- Kritik işlerin ustalarına: "Sakın 4 günde bitirin, 5 güne sarkmasın. 3 günde biterse memnun olurum." — yöneylem problemi.

Hocadan: "2. vizede en erken ve en geç başlama zamanı hesaplamayı sordum, yine sorarım. Bunu veririm, "İşletin." derim. Anlaşıldığı zannedilir, ama %80'i işletemez."

En Erken Başlama Zamanları (Earliest Times)

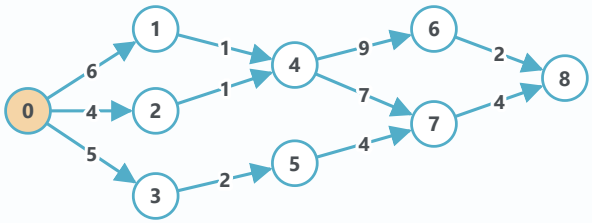
- a_i = <k, m> ise `EnErken(i) = EnErkenBas[k]` : aktivite, kuyruğundaki olay gerçekleşince başlar.
- Ör. a1: k = v0 → 0; a5: k = v3 → 5 (a2); a8: k = v5 → 5 + 2 = 7.
- v4'e iki yol var: a3 en erken 6 + 1 = 7'de, a4 4 + 1 = 5'te biter. İkisi de bitmeli → **en uzunu** (7) beklenir.
- `EnErkenBas[m] = max { EnErkenBas[k] + süre(<k,m>) }` , k ∈ P(m) (yakın öncüller).
- Başta dizi **0**: projenin başlangıcı 0. günü (ör. "8 Mayıs"); 10 konsaydı hepsi 10 kayardı.
- Hesap **topolojik sıraya** göre yapılmalıdır: öncüllerin değerleri önce bilinmeli.

```
/* TopolojikSira icinde sayac azaltilmadan once: */
if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
    EnErkenBas[k] = EnErkenBas[j] + p->sure;
```



Liste öğesine `sure` alanı eklenir; **daha uzun bir yol** bulununca değer günlenir.

Kod İzleme: EnErkenBas[] (topolojik sırada)



EnErkenBas[]

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

yığın (üst solda): 0

Başlangıç: EnErkenBas[] = 0; ilk döngü öncülsüz v0'ı yığına koydu.

Adım 1 / 11

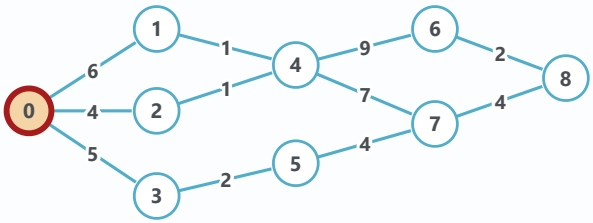
```
▶ j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
4     k = p->dugum;
5     if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
6         EnErkenBas[k] = EnErkenBas[j] + p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;  /* yigina */
9     }
10 }
```

DEĞİŞKENLER

`j = –` `ust = 0`

ÇIKTI

Kod İzleme: EnErkenBas[] (topolojik sırada)



EnErkenBas[]

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

yığın (üst solda): boş

Yığından **v0** alınır ve yazılır (EnErkenBas[0] = 0 artık kesin).

Adım 2 / 11

```
▶ j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
4     k = p->dugum;
5     if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
6         EnErkenBas[k] = EnErkenBas[j] + p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;  /* yigina */
9     }
10 }
```

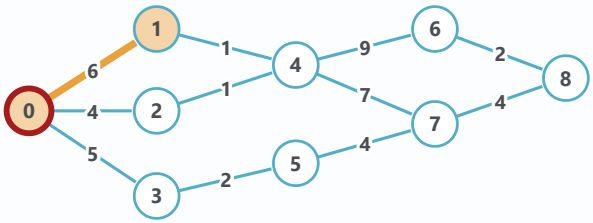
DEĞİŞKENLER

`j = 0` `ust = -1`

ÇIKTI

```
0
```

Kod İzleme: EnErkenBas[] (topolojik sırada)



EnErkenBas[]

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

yığın (üst solda): 1

<0,1> (süre 6): 0 + 6 = 6 > 0 → **EnErkenBas[1] = 6**; v1 yığına girdi.

Adım 3 / 11

```
1  j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2  printf("%d ", j);
3  for (p = baslik[j].bag; p; p = p->bag) {
▶      k = p->dugum;
▶      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
▶          EnErkenBas[k] = EnErkenBas[j] + p->sure;
7      if (--baslik[k].sayac == 0) {
▶          baslik[k].sayac = ust;  ust = k;  /* yigina */
9      }
10 }
```

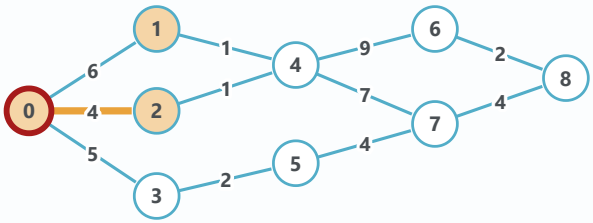
DEĞİŞKENLER

`j = 0` `k = 1` `ust = 1`

ÇIKTI

```
0
```

Kod İzleme: EnErkenBas[] (topolojik sırada)



EnErkenBas[]

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

yığın (üst solda): 2, 1

<0,2> (süre 4): 0 + 4 = 4 > 0 → **EnErkenBas[2] = 4**; v2 yığına girdi.

Adım 4 / 11

```
1  j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2  printf("%d ", j);
3  for (p = baslik[j].bag; p; p = p->bag) {
▶      k = p->dugum;
▶      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
▶          EnErkenBas[k] = EnErkenBas[j] + p->sure;
7      if (--baslik[k].sayac == 0) {
▶          baslik[k].sayac = ust;  ust = k;  /* yigina */
9      }
10 }
```

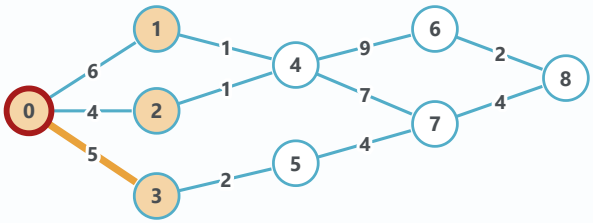
DEĞİŞKENLER

j = 0 | k = 2 | ust = 2

ÇIKTI

```
0
```

Kod İzleme: EnErkenBas[] (topolojik sırada)



EnErkenBas[]

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

yığın (üst solda): 3, 2, 1

<0,3> (süre 5): 0 + 5 = 5 > 0 → **EnErkenBas[3] = 5**; v3 yığına girdi.

Adım 5 / 11

```
1  j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2  printf("%d ", j);
3  for (p = baslik[j].bag; p; p = p->bag) {
▶      k = p->dugum;
▶      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
▶          EnErkenBas[k] = EnErkenBas[j] + p->sure;
7      if (--baslik[k].sayac == 0) {
▶          baslik[k].sayac = ust;  ust = k;  /* yigina */
9      }
10 }
```

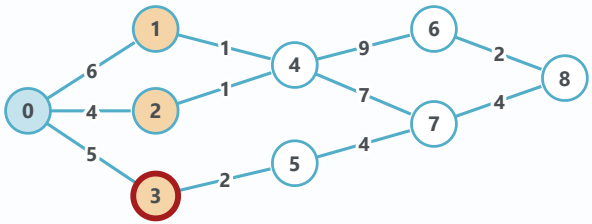
DEĞİŞKENLER

`j = 0` `k = 3` `ust = 3`

ÇIKTI

```
0
```

Kod İzleme: EnErkenBas[] (topolojik sırada)



EnErkenBas[]

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

yığın (üst solda): 2, 1

Yığından **v3** alınır ve yazılır (EnErkenBas[3] = 5 artık kesin).

Adım 6 / 11

```
▶ j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
  printf("%d ", j);
  for (p = baslik[j].bag; p; p = p->bag) {
      k = p->dugum;
      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
          EnErkenBas[k] = EnErkenBas[j] + p->sure;
      if (--baslik[k].sayac == 0) {
          baslik[k].sayac = ust;  ust = k;  /* yigina */
      }
  }
```

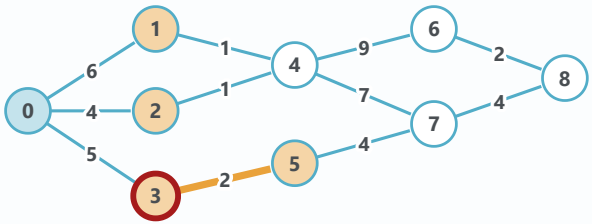
DEĞİŞKENLER

`j = 3` `ust = 2`

ÇIKTI

```
0 3
```

Kod İzleme: EnErkenBas[] (topolojik sırada)



EnErkenBas[]

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

yığın (üst solda): 5, 2, 1

<3,5> (süre 2): 5 + 2 = 7 > 0 → **EnErkenBas[5] = 7**; v5 yığına girdi.

Adım 7 / 11

```
1  j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2  printf("%d ", j);
3  for (p = baslik[j].bag; p; p = p->bag) {
▶      k = p->dugum;
▶      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
▶          EnErkenBas[k] = EnErkenBas[j] + p->sure;
7      if (--baslik[k].sayac == 0) {
▶          baslik[k].sayac = ust;  ust = k;  /* yigina */
9      }
10 }
```

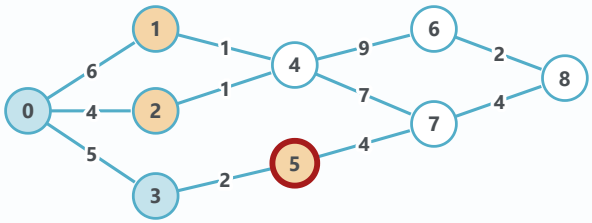
DEĞİŞKENLER

j = 3 | k = 5 | ust = 5

ÇIKTI

```
0 3
```

Kod İzleme: EnErkenBas[] (topolojik sırada)



EnErkenBas[]

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

yığın (üst solda): 2, 1

Yığından **v5** alınır ve yazılır (EnErkenBas[5] = 7 artık kesin).

Adım 8 / 11

```
▶ j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
4     k = p->dugum;
5     if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
6         EnErkenBas[k] = EnErkenBas[j] + p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;  /* yigina */
9     }
10 }
```

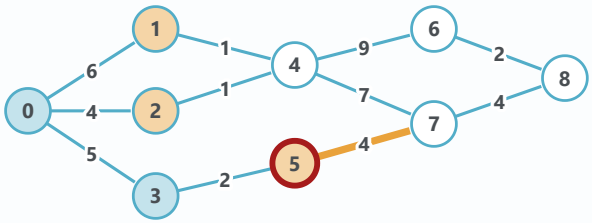
DEĞİŞKENLER

`j = 5` `ust = 2`

ÇIKTI

```
0 3 5
```

Kod İzleme: EnErkenBas[] (topolojik sırada)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	0	7	0	11	0

yığın (üst solda): 2, 1

<5,7> (süre 4): 7 + 4 = 11 > 0 → **EnErkenBas[7] = 11**.

Adım 9 / 11

```
1  j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2  printf("%d ", j);
3  for (p = baslik[j].bag; p; p = p->bag) {
▶      k = p->dugum;
▶      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
▶          EnErkenBas[k] = EnErkenBas[j] + p->sure;
7      if (--baslik[k].sayac == 0) {
8          baslik[k].sayac = ust;  ust = k;  /* yigina */
9      }
10 }
```

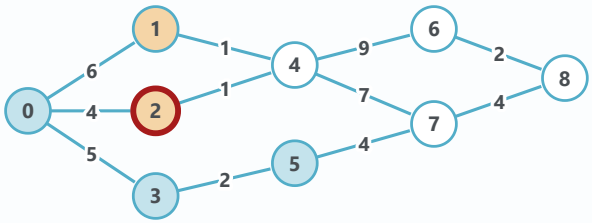
DEĞİŞKENLER

j = **5** | k = **7** | ust = **2**

ÇIKTI

```
0 3 5
```

Kod İzleme: EnErkenBas[] (topolojik sırada)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	0	7	0	11	0

yığın (üst solda): 1

Yığından **v2** alınır ve yazılır (EnErkenBas[2] = 4 artık kesin).

Adım 10 / 11

```
▶ j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
4     k = p->dugum;
5     if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
6         EnErkenBas[k] = EnErkenBas[j] + p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;  /* yigina */
9     }
10 }
```

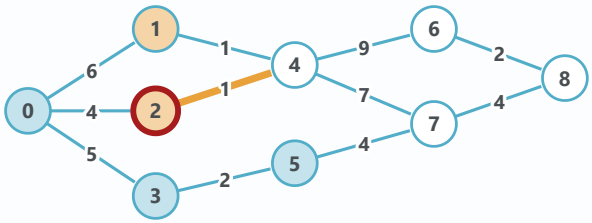
DEĞİŞKENLER

`j = 2` `ust = 1`

ÇIKTI

```
0 3 5 2
```

Kod İzleme: EnErkenBas[] (topolojik sırada)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	5	7	0	11	0

yığın (üst solda): 1

<2,4> (süre 1): 4 + 1 = 5 > 0 → **EnErkenBas[4] = 5**.

Adım 11 / 11

```
1  j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2  printf("%d ", j);
3  for (p = baslik[j].bag; p; p = p->bag) {
▶      k = p->dugum;
▶      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
▶          EnErkenBas[k] = EnErkenBas[j] + p->sure;
7      if (--baslik[k].sayac == 0) {
8          baslik[k].sayac = ust;  ust = k;  /* yigina */
9      }
10 }
```

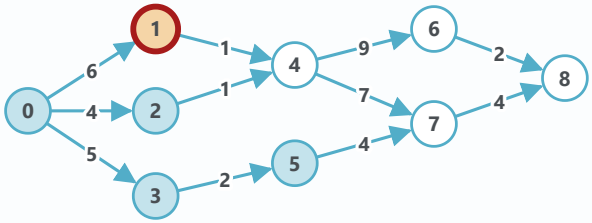
DEĞİŞKENLER

j = 2 | k = 4 | ust = 1

ÇIKTI

```
0 3 5 2
```

Kod İzleme: EnErkenBas[] (devam)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	5	7	0	11	0

yığın (üst solda): boş

Yığından **v1** alınır ve yazılır (EnErkenBas[1] = 6 artık kesin).

Adım 1 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
  printf("%d ", j);
  for (p = baslik[j].bag; p; p = p->bag) {
      k = p->dugum;
      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
          EnErkenBas[k] = EnErkenBas[j] + p->sure;
      if (--baslik[k].sayac == 0) {
          baslik[k].sayac = ust;  ust = k;  /* yigina */
      }
  }
```

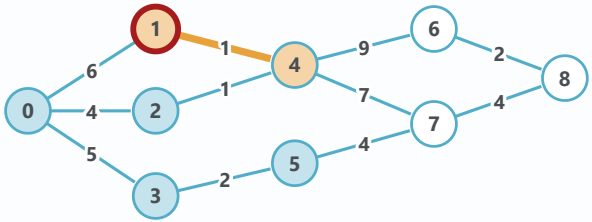
DEĞİŞKENLER

`j = 1` `ust = -1`

ÇIKTI

```
0 3 5 2 1
```

Kod İzleme: EnErkenBas[] (devam)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	7	7	0	11	0

yığın (üst solda): 4

<1,4> (süre 1): 6 + 1 = 7 > 5 → **EnErkenBas[4] = 7**; v4 yığına girdi.

Adım 2 / 10

```
1  j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2  printf("%d ", j);
3  for (p = baslik[j].bag; p; p = p->bag) {
▶      k = p->dugum;
▶      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
▶          EnErkenBas[k] = EnErkenBas[j] + p->sure;
7      if (--baslik[k].sayac == 0) {
▶          baslik[k].sayac = ust;  ust = k;  /* yigina */
9      }
10 }
```

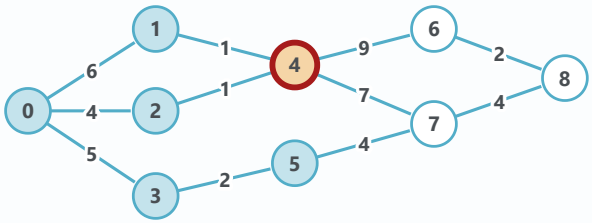
DEĞİŞKENLER

`j = 1` `k = 4` `ust = 4`

ÇIKTI

```
0 3 5 2 1
```

Kod İzleme: EnErkenBas[] (devam)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	7	7	0	11	0

yığın (üst solda): boş

Yığından **v4** alınır ve yazılır (EnErkenBas[4] = 7 artık kesin).

Adım 3 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
4     k = p->dugum;
5     if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
6         EnErkenBas[k] = EnErkenBas[j] + p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;  /* yigina */
9     }
10 }
```

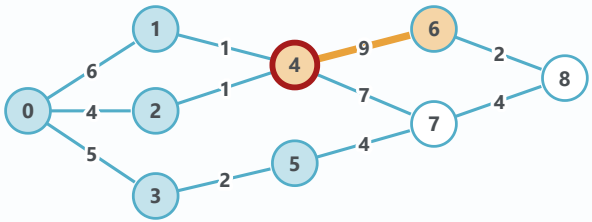
DEĞİŞKENLER

`j = 4` `ust = -1`

ÇIKTI

```
0 3 5 2 1 4
```

Kod İzleme: EnErkenBas[] (devam)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	7	7	16	11	0

yığın (üst solda): 6

<4,6> (süre 9): 7 + 9 = 16 > 0 → **EnErkenBas[6] = 16**; v6 yığına girdi.

Adım 4 / 10

```
1  j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2  printf("%d ", j);
3  for (p = baslik[j].bag; p; p = p->bag) {
▶      k = p->dugum;
▶      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
▶          EnErkenBas[k] = EnErkenBas[j] + p->sure;
7      if (--baslik[k].sayac == 0) {
▶          baslik[k].sayac = ust;  ust = k;  /* yigina */
9      }
10 }
```

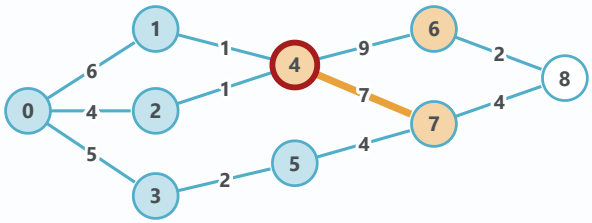
DEĞİŞKENLER

`j = 4` `k = 6` `ust = 6`

ÇIKTI

```
0 3 5 2 1 4
```

Kod İzleme: EnErkenBas[] (devam)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	7	7	16	14	0

yığın (üst solda): 7, 6

<4,7> (süre 7): 7 + 7 = 14 > 11 → **EnErkenBas[7] = 14**; v7 yığına girdi.

Adım 5 / 10

```
1  j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2  printf("%d ", j);
3  for (p = baslik[j].bag; p; p = p->bag) {
▶      k = p->dugum;
▶      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
▶          EnErkenBas[k] = EnErkenBas[j] + p->sure;
7      if (--baslik[k].sayac == 0) {
▶          baslik[k].sayac = ust;  ust = k;  /* yigina */
9      }
10 }
```

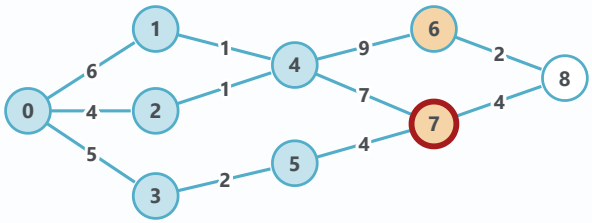
DEĞİŞKENLER

`j = 4` `k = 7` `ust = 7`

ÇIKTI

```
0 3 5 2 1 4
```

Kod İzleme: EnErkenBas[] (devam)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	7	7	16	14	0

yığın (üst solda): 6

Yığından **v7** alınır ve yazılır (EnErkenBas[7] = 14 artık kesin).

Adım 6 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
4     k = p->dugum;
5     if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
6         EnErkenBas[k] = EnErkenBas[j] + p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;  /* yigina */
9     }
10 }
```

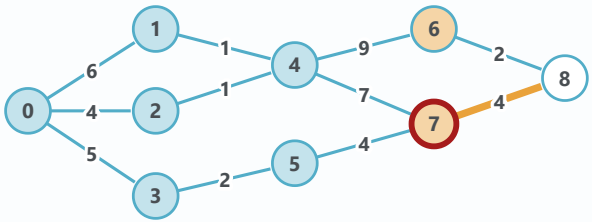
DEĞİŞKENLER

`j = 7` `ust = 6`

ÇIKTI

```
0 3 5 2 1 4 7
```

Kod İzleme: EnErkenBas[] (devam)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	7	7	16	14	18

yığın (üst solda): 6

<7,8> (süre 4): 14 + 4 = 18 > 0 → **EnErkenBas[8] = 18**.

Adım 7 / 10

```
1  j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2  printf("%d ", j);
3  for (p = baslik[j].bag; p; p = p->bag) {
▶      k = p->dugum;
▶      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
▶          EnErkenBas[k] = EnErkenBas[j] + p->sure;
7      if (--baslik[k].sayac == 0) {
8          baslik[k].sayac = ust;  ust = k;  /* yigina */
9      }
10 }
```

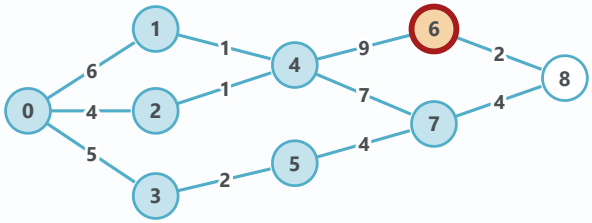
DEĞİŞKENLER

`j = 7` `k = 8` `ust = 6`

ÇIKTI

```
0 3 5 2 1 4 7
```

Kod İzleme: EnErkenBas[] (devam)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	7	7	16	14	18

yığın (üst solda): boş

Yığından **v6** alınır ve yazılır (EnErkenBas[6] = 16 artık kesin).

Adım 8 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
4     k = p->dugum;
5     if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
6         EnErkenBas[k] = EnErkenBas[j] + p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;  /* yigina */
9     }
10 }
```

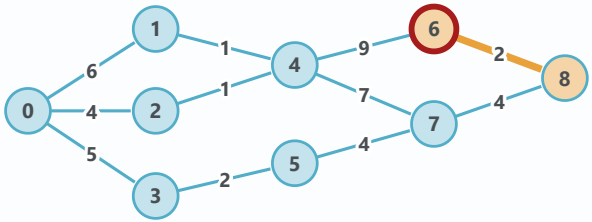
DEĞİŞKENLER

`j = 6` `ust = -1`

ÇIKTI

```
0 3 5 2 1 4 7 6
```

Kod İzleme: EnErkenBas[] (devam)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	7	7	16	14	18

yığın (üst solda): 8

<6,8> (süre 2): 16 + 2 = 18 ≤ 18, değişmez; v8 yığına girdi.

Adım 9 / 10

```
1  j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
2  printf("%d ", j);
3  for (p = baslik[j].bag; p; p = p->bag) {
▶      k = p->dugum;
▶      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
6          EnErkenBas[k] = EnErkenBas[j] + p->sure;
7      if (--baslik[k].sayac == 0) {
▶          baslik[k].sayac = ust;  ust = k;  /* yigina */
9      }
10 }
```

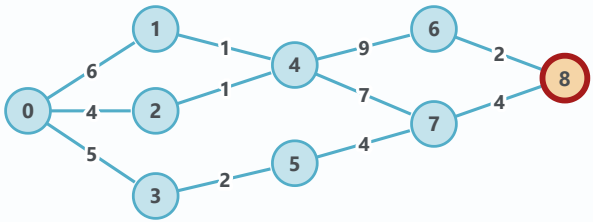
DEĞİŞKENLER

`j = 6` `k = 8` `ust = 8`

ÇIKTI

```
0 3 5 2 1 4 7 6
```

Kod İzleme: EnErkenBas[] (devam)



EnErkenBas[]

0	1	2	3	4	5	6	7	8
0	6	4	5	7	7	16	14	18

yığın (üst solda): boş

Yığından **v8** alınır ve yazılır (EnErkenBas[8] = 18 artık kesin).

Adım 10 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* yigindan al */
  printf("%d ", j);
  for (p = baslik[j].bag; p; p = p->bag) {
      k = p->dugum;
      if (EnErkenBas[k] < EnErkenBas[j] + p->sure)
          EnErkenBas[k] = EnErkenBas[j] + p->sure;
      if (--baslik[k].sayac == 0) {
          baslik[k].sayac = ust;  ust = k;  /* yigina */
      }
  }
```

DEĞİŞKENLER

`j = 8` `ust = -1`

ÇIKTI

```
0 3 5 2 1 4 7 6 8
```

ERKEN/GEÇ ZAMANLAR

En Erken Başlama Zamanı Hesaplama Tablosu

EnErkenBas	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	Yığın
Başlangıç	0	0	0	0	0	0	0	0	0	0
Çıktı v0	0	6	4	5	0	0	0	0	0	3, 2, 1
Çıktı v3	0	6	4	5	0	7	0	0	0	5, 2, 1
Çıktı v5	0	6	4	5	0	7	0	11	0	2, 1
Çıktı v2	0	6	4	5	5	7	0	11	0	1
Çıktı v1	0	6	4	5	7	7	0	11	0	4
Çıktı v4	0	6	4	5	7	7	16	14	0	7, 6
Çıktı v7	0	6	4	5	7	7	16	14	18	6
Çıktı v6	0	6	4	5	7	7	16	14	18	8
Çıktı v8	0	6	4	5	7	7	16	14	18	–

Hangi veriler kalıcı? “Ancak çıktıya yazılan düğümlerin en erken başlama zamanları kesinleşmiştir, o andan itibaren bir daha değişmez.” Ör. v5 işlenince [7] = 11 yazıldı; v4’ten gelen daha uzun yol onu 14 yaptı. Proje en erken **18** günde biter.

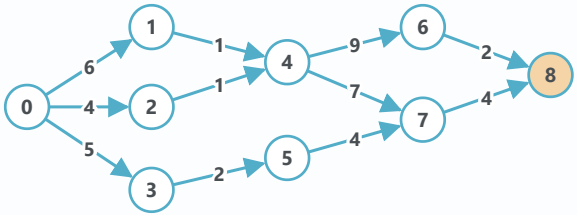
En Geç Başlama Zamanları (Latest Times)

- **Geri doğru** çözümleme: tüm oklar ters çevrilir; bu çizgenin topolojik sırasında çalışılır (v8 ilk, v0 son). Yapı: **komşuluk listesinin tersi**, O(n + e)'de kurulur.
- Son düğüm: `EnGecBas[n-1] = EnErkenBas[n-1]` = 18. Başta dizi 18 ile doldurulur.
- a_i = <k, m>: `EnGec(i) = EnGecBas[m] - süre`. v7 en geç 18 − 4 = 14, v6 en geç 18 − 2 = 16.
- Birden çok ardıl varsa **en küçüğü**: `EnGecBas[k] = min { EnGecBas[m] - süre }`. Ör. v0: v1 için 6 − 6 = 0, v2 için 6 − 4 = 2, v3 için 8 − 5 = 3 → **0**. Büyüğünü (3) alsaydık v1 gecikir, proje 21'e sarkardı.

```
if (EnGecBas[k] > EnGecBas[j] - p->sure)
    EnGecBas[k] = EnGecBas[j] - p->sure;
```

a9 = 2 olduğundan v6 en geç **16**. gündür (tablo da 16 der) ve v4 için iki yol da 7 verir; burada min kuralı fark yaratmaz. Min kuralını göstermek için v0 örneği daha uygundur.

Kod İzleme: EnGecBas[] (ters çizgede)



EnGecBas[]

0	1	2	3	4	5	6	7	8
18	18	18	18	18	18	18	18	18

yığın (üst solda): 8

Başlangıç: EnGecBas[] = 18; ters çizgede öncülsüz tek düğüm v8 yığında.

Adım 1 / 10

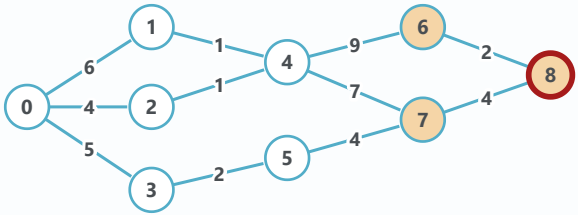
```
▶ j = ust;  ust = baslik[ust].sayac;   /* ters liste */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
4     k = p->dugum;                        /* k: oncul */
5     if (EnGecBas[k] > EnGecBas[j] - p->sure)
6         EnGecBas[k] = EnGecBas[j] - p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;
9     }
10 }
```

DEĞİŞKENLER

`j = –` `ust = 8`

ÇIKTI

Kod İzleme: EnGecBas[] (ters çizgede)



EnGecBas[]

0	1	2	3	4	5	6	7	8
18	18	18	18	18	18	16	14	18

yığın (üst solda): 7, 6

v8 alınır (EnGecBas[8] = 18 kesin). Öncüller: EnGecBas[6] = 16, EnGecBas[7] = 14. Yığına: v6, v7.

Adım 2 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* ters liste */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
▶     k = p->dugum;                        /* k: oncul */
▶     if (EnGecBas[k] > EnGecBas[j] - p->sure)
▶         EnGecBas[k] = EnGecBas[j] - p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;
9     }
10 }
```

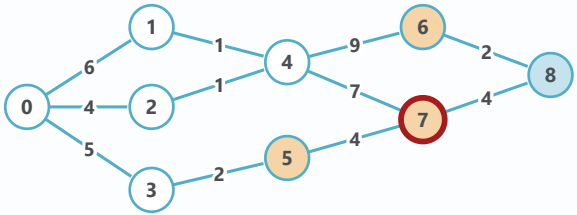
DEĞİŞKENLER

`j = 8` `ust = 7`

ÇIKTI

```
8
```

Kod İzleme: EnGecBas[] (ters çizgede)



EnGecBas[]

0	1	2	3	4	5	6	7	8
18	18	18	18	7	10	16	14	18

yığın (üst solda): 5, 6

v7 alınır (EnGecBas[7] = 14 kesin). Öncüller: EnGecBas[4] = 7, EnGecBas[5] = 10. Yığına: v5.

Adım 3 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* ters liste */
  printf("%d ", j);
  for (p = baslik[j].bag; p; p = p->bag) {
▶     k = p->dugum;                    /* k: oncul */
▶     if (EnGecBas[k] > EnGecBas[j] - p->sure)
▶         EnGecBas[k] = EnGecBas[j] - p->sure;
      if (--baslik[k].sayac == 0) {
          baslik[k].sayac = ust;  ust = k;
      }
  }
```

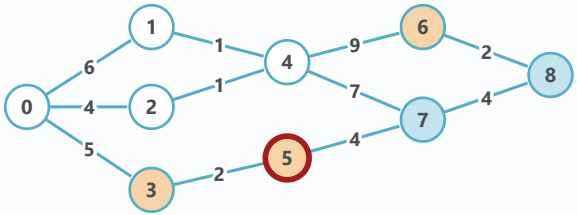
DEĞİŞKENLER

`j = 7` `ust = 5`

ÇIKTI

```
8 7
```

Kod İzleme: EnGecBas[] (ters çizgede)



EnGecBas[]

0	1	2	3	4	5	6	7	8
18	18	18	8	7	10	16	14	18

yığın (üst solda): 3, 6

v5 alınır (EnGecBas[5] = 10 kesin). Öncüller: EnGecBas[3] = 8. Yığına: v3.

Adım 4 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* ters liste */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
▶     k = p->dugum;                        /* k: oncul */
▶     if (EnGecBas[k] > EnGecBas[j] - p->sure)
▶         EnGecBas[k] = EnGecBas[j] - p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;
9     }
10 }
```

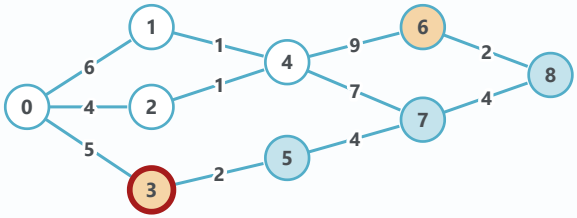
DEĞİŞKENLER

`j = 5` `ust = 3`

ÇIKTI

```
8 7 5
```

Kod İzleme: EnGecBas[] (ters çizgede)



EnGecBas[]

0	1	2	3	4	5	6	7	8
3	18	18	8	7	10	16	14	18

yığın (üst solda): 6

v3 alınır (EnGecBas[3] = 8 kesin). Öncüller: EnGecBas[0] = 3.

Adım 5 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* ters liste */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
▶     k = p->dugum;                    /* k: oncul */
▶     if (EnGecBas[k] > EnGecBas[j] - p->sure)
▶         EnGecBas[k] = EnGecBas[j] - p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;
9     }
10 }
```

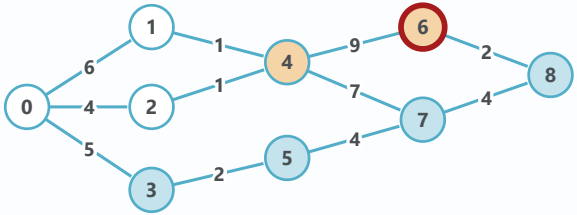
DEĞİŞKENLER

`j = 3` `ust = 6`

ÇIKTI

```
8 7 5 3
```

Kod İzleme: EnGecBas[] (ters çizgede)



EnGecBas[]

0	1	2	3	4	5	6	7	8
3	18	18	8	7	10	16	14	18

yığın (üst solda): 4

v6 alınır (EnGecBas[6] = 16 kesin). [4] değişmez. Yığına: v4.

Adım 6 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* ters liste */
  printf("%d ", j);
  for (p = baslik[j].bag; p; p = p->bag) {
▶     k = p->dugum;                    /* k: oncul */
▶     if (EnGecBas[k] > EnGecBas[j] - p->sure)
          EnGecBas[k] = EnGecBas[j] - p->sure;
      if (--baslik[k].sayac == 0) {
          baslik[k].sayac = ust;  ust = k;
      }
  }
```

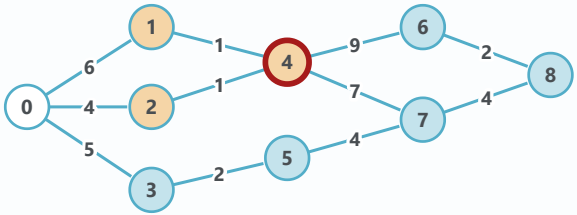
DEĞİŞKENLER

`j = 6` `ust = 4`

ÇIKTI

```
8 7 5 3 6
```

Kod İzleme: EnGecBas[] (ters çizgede)



EnGecBas[]

0	1	2	3	4	5	6	7	8
3	6	6	8	7	10	16	14	18

yığın (üst solda): 2, 1

v4 alınır (EnGecBas[4] = 7 kesin). Öncüller: EnGecBas[1] = 6, EnGecBas[2] = 6. Yığına: v1, v2.

Adım 7 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* ters liste */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
▶     k = p->dugum;                    /* k: oncul */
▶     if (EnGecBas[k] > EnGecBas[j] - p->sure)
▶         EnGecBas[k] = EnGecBas[j] - p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;
9     }
10 }
```

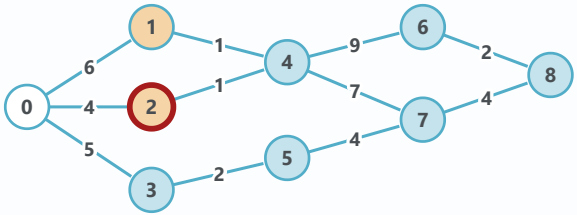
DEĞİŞKENLER

`j = 4` `ust = 2`

ÇIKTI

```
8 7 5 3 6 4
```

Kod İzleme: EnGecBas[] (ters çizgede)



EnGecBas[]

0	1	2	3	4	5	6	7	8
2	6	6	8	7	10	16	14	18

yığın (üst solda): 1

v2 alınır (EnGecBas[2] = 6 kesin). Öncüller: EnGecBas[0] = 2 (değişti).

Adım 8 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* ters liste */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
▶     k = p->dugum;                    /* k: oncul */
▶     if (EnGecBas[k] > EnGecBas[j] - p->sure)
▶         EnGecBas[k] = EnGecBas[j] - p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;
9     }
10 }
```

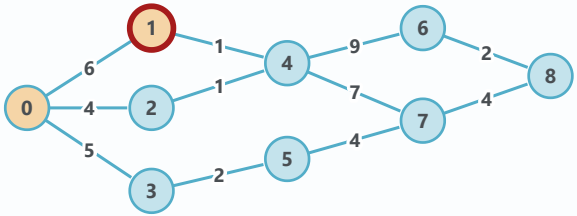
DEĞİŞKENLER

`j = 2` `ust = 1`

ÇIKTI

```
8 7 5 3 6 4 2
```

Kod İzleme: EnGecBas[] (ters çizgede)



EnGecBas[]

0	1	2	3	4	5	6	7	8
0	6	6	8	7	10	16	14	18

yığın (üst solda): 0

v1 alınır (EnGecBas[1] = 6 kesin). Öncüller: EnGecBas[0] = 0 (değişti). Yığına: v0.

Adım 9 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* ters liste */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
▶     k = p->dugum;                        /* k: oncul */
▶     if (EnGecBas[k] > EnGecBas[j] - p->sure)
▶         EnGecBas[k] = EnGecBas[j] - p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;
9     }
10 }
```

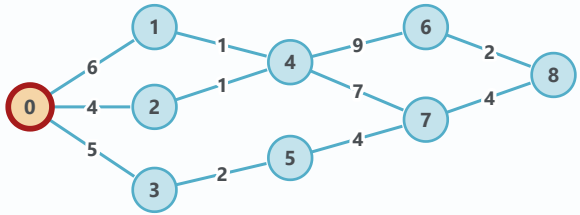
DEĞİŞKENLER

`j = 1` `ust = 0`

ÇIKTI

```
8 7 5 3 6 4 2 1
```

Kod İzleme: EnGecBas[] (ters çizgede)



EnGecBas[]

0	1	2	3	4	5	6	7	8
0	6	6	8	7	10	16	14	18

yığın (üst solda): boş

v0 alınır (EnGecBas[0] = 0 kesin). Öncülü yok (başlangıç olayı).

Adım 10 / 10

```
▶ j = ust;  ust = baslik[ust].sayac;   /* ters liste */
2 printf("%d ", j);
3 for (p = baslik[j].bag; p; p = p->bag) {
4     k = p->dugum;                        /* k: oncul */
5     if (EnGecBas[k] > EnGecBas[j] - p->sure)
6         EnGecBas[k] = EnGecBas[j] - p->sure;
7     if (--baslik[k].sayac == 0) {
8         baslik[k].sayac = ust;  ust = k;
9     }
10 }
```

DEĞİŞKENLER

`j = 0` `ust = -1`

ÇIKTI

```
8 7 5 3 6 4 2 1 0
```

ERKEN/GEÇ ZAMANLAR

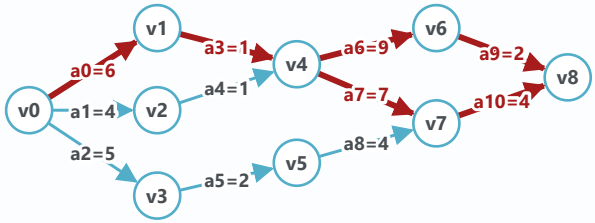
En Geç Başlama Zamanı Hesaplama Tablosu

EnGecBas	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	Yığın
Başlangıç	18	18	18	18	18	18	18	18	18	8
Çıktı v8	18	18	18	18	18	18	16	14	18	7, 6
Çıktı v7	18	18	18	18	7	10	16	14	18	5, 6
Çıktı v5	18	18	18	8	7	10	16	14	18	3, 6
Çıktı v3	3	18	18	8	7	10	16	14	18	6
Çıktı v6	3	18	18	8	7	10	16	14	18	4
Çıktı v4	3	6	6	8	7	10	16	14	18	2, 1
Çıktı v2	2	6	6	8	7	10	16	14	18	1
Çıktı v1	0	6	6	8	7	10	16	14	18	0
Çıktı v0	0	6	6	8	7	10	16	14	18	–

[0] üç kez günlendi: v3'ten 8 − 5 = 3, v2'den 6 − 4 = 2, v1'den 6 − 6 = **0**.

Bolluklar ve Kritik Aktiviteler

Akt.	Kenar	Süre	En erken	En geç	Bolluk
a0	(0,1)	6	0	0	**0: kritik**
a1	(0,2)	4	0	2	2
a2	(0,3)	5	0	3	3
a3	(1,4)	1	6	6	**0: kritik**
a4	(2,4)	1	4	6	2
a5	(3,5)	2	5	8	3
a6	(4,6)	9	7	7	**0: kritik**
a7	(4,7)	7	7	7	**0: kritik**
a8	(5,7)	4	7	10	3
a9	(6,8)	2	16	16	**0: kritik**
a10	(7,8)	4	14	14	**0: kritik**



kalın kırmızı kenar: bolluk 0 (kritik)

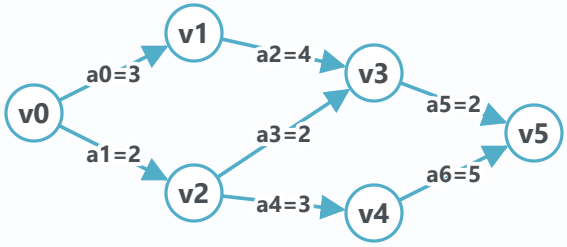
- Bolluk = en geç − en erken (a_i = <k,m>: EnGecBas[m] − süre − EnErkenBas[k]). Ör. a4: en erken 4, en geç 6 → **2 gün** gecikebilir.
- Bolluğu 0 olan **kritik aktivite**: en erken vaktinde başlamak zorunda. Project yazılımları bolluk dağılımını grafikle gösterir.
- Bu **toplam bolluktur** (*total float*): a2-a5-a8 yolundaki 3 günlük bolluk paylaşılır; a2 veya a5 kullanırsa a8 kritik olur.

Kritik Yolu Program Bulsun (ek konu)

```
/* EnErkenBas[] ve EnGecBas[] hesaplandiktan sonra */
for (i = 0; i < n; i++)
    for (p = baslik[i].bag; p; p = p->bag) {
        int erken = EnErkenBas[i];
        int gec   = EnGecBas[p->dugum] - p->sure;
        printf("<%d,%d> erken=%d gec=%d bolluk=%d%s\n",
               i, p->dugum, erken, gec, gec - erken,
               gec == erken ? " KRITIK" : "");
    }
```

- Kritik aktiviteler v0'dan v8'e uzanan bir (ya da birkaç) yol oluşturur: **kritik yol**.
- Tüm hesap: ileri çözümleme O(n + e) + transpoz liste O(n + e) + geri çözümleme O(n + e) = **O(n + e)**.
- DAG'de "en uzun yol" topolojik sırada kolaydır; genel çizgede en uzun basit yol NP-zordur.
- **Serbest bolluk** (*free float*): ardılların en erken zamanını geciktirmeden kullanılabilen bolluk; a2 ve a5 için 0, a8 için 3.

Sınav Sorusu: En Erken / En Geç Zamanları İşletin



1. Topolojik sırayı ve her adımda yığını yazın.
2. EnErkenBas[] ve EnGecBas[] tablolarını doldurun.
3. Her aktivitenin bolluğunu bulun; kritik yolu belirtin.

> **Hocadan:** "'Hocam nerden nereye kadar çıkacak?' İşlediğimiz bütün konulardan gelebilir. Veri Yapıları 1'den de gelebilir."

▶ **Çözüm**

Laboratuvar: Dijkstra

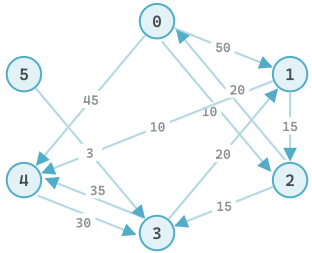
EnKisaYolBul(0, …): 6 düğüm

Girdi (yönlü kenarlar, ör. 0>1:50) `0>1:50, 0>2:10, 0>4:45, 1>2:15, 1>4:10, 2>0:20, 2>3:15, 3>1`

Uygula

KOD

```
 1 for (i = 0; i < n; i++) {        /* A1 */
 2     gidildi[i] = FALSE;
 3     uzaklik[i] = maliyet[v][i];
 4     onceki[i] = v;
 5 }
 6 gidildi[v] = TRUE;  uzaklik[v] = 0;
 7 for (i = 2; i < n; i++) {     /* n-2 tur */
 8   u = minimumBul(uzaklik, n, gidildi);
 9   if (u < 0) break; /* kalan yok */
10   gidildi[u] = TRUE;
11   for (w = 0; w < n; w++)
12     if (!gidildi[w] && uzaklik[u] +
13         maliyet[u][w] < uzaklik[w]) {
14       uzaklik[w] = uzaklik[u] +
```



uzaklik

onceki

ÇIKTI (henüz yok)

`EnKisaYolBul(0, …)` : kaynak 0. Kenar yoksa maliyet `SONSUZ = 1000` (∞). Düğüm yanındaki sayı o anki `uzaklik` değeridir.

ÖZET

Karmaşıklık Özeti

Algoritma	Yapı	Zaman
Kaba kuvvet (tek hedef)	tüm yollar	$O(n^n)$
Dijkstra (temel)	maliyet matrisi + diziler	$O(n^2)$
Dijkstra (iyileştirilmiş)	min-heap + komşuluk listesi	$O((n + e) \log n)$
Yolu yazdırma	öncüller dizisi	O(yol uzunluğu)
Bellman-Ford	kenar listesi	$O(n \cdot e)$
Floyd-Warshall	n × n matris	$O(n^3)$
Topolojik sıra	sayaçlı liste + gömülü yığın	$O(n + e)$
En erken / en geç zaman, bolluk	topolojik sıra + ters liste	$O(n + e)$

ÖZET

Sık Yapılan Hatalar

1. Negatif kenarlı çizgede Dijkstra kullanmak.
2. ∞ için INT_MAX seçip `uzaklik[u] + maliyet` toplamında taşma yaşamak.
3. En erken zamanda **min**, en geç zamanda **max** almak (tersi doğru!).
4. En geç zamanı ardıllar kesinleşmeden hesaplamak: sıra ters çizgenin topolojik sırası (ya da ileri sıranın tersi) olmalıdır.
5. Topolojik sıraya rastgele bir düğümden başlamak; öncülleri işlenmemiş düğümün değerini kesin sanmak.
6. Yolu ileri doğru kurmaya çalışmak; öncüller dizisini tersten kullanmamak.

Alıştırmalar

1. Yönlü örnek çizgede kaynak 2 olsaydı EnKisaYolBul hangi uzaklıkları bulurdu?
2. EnKisaYolBul'un dış döngüsü neden n − 2 tur döner? n − 1 tur dönse ne olurdu?
3. Adım adım örnekteki (v0…v6) çizgede v3–v6 kenarı 2 yerine 15 olsaydı S'ye katılma sırası ne olurdu?
4. Topolojik sıra yordamında yığın yerine kuyruk kullanılsaydı v0..v5 örneğinde çıktı ne olurdu?
5. AoE örneğinde a8'in süresi 4 yerine 8 olsaydı proje süresi ve kritik yol ne olurdu?
6. Floyd-Warshall örneğinde A[1][0] neden k = 2 turunda değişti?

▶ **Cevaplar / ipuçları**

ÖZET

Bölüm Özeti

En kısa yol

Kaba kuvvet $O(n^n)$; Dijkstra açgözlüdür, $w \geq 0$ ister: $O(n^2)$ ya da $O((n + e) \log n)$.

Yol

Öncüller dizisi tek başına yeter; tersten (özyineleme/yığın) yazdırılır.

Genellemeler

Negatif kenar: Bellman-Ford $O(ne)$. Tüm çiftler: Floyd-Warshall $O(n^3)$.

AoV ve topolojik sıra

Sayaç + gömülü yığın; $O(n + e)$; döngüyü de yakalar.

AoE ve zamanlar

İleri: max ile en erken; geri (ters çizge): min ile en geç. Bolluk = geç − erken.

Kritik yol

Bolluğu 0 olan aktiviteler; projenin süresini belirler.

ÖZET

Kaynaklar

- E. Horowitz, S. Sahni, S. Anderson-Freed, *Fundamentals of Data Structures in C*, 2. baskı, §6.4 Shortest Paths, §6.5 Activity Networks.
- T. H. Cormen ve ark., *Introduction to Algorithms*, 4. baskı, Bölüm 20.4 (Topological Sort), 22 (Single-Source Shortest Paths), 23 (All-Pairs).
- R. Sedgewick, *Algorithms in C, Part 5*, Bölüm 21 (Shortest Paths).

← Bölüm 17: Minimum Kapsayan Ağaçlar

Ders sayfası

Bölüm 19: Temel Sıralama Algoritmaları

Son

Bölüm 18: En Kısa Yol ve Aktivite Ağları

Sonraki bölüm: Bölüm 19: Temel Sıralama Algoritmaları · Ders sayfası