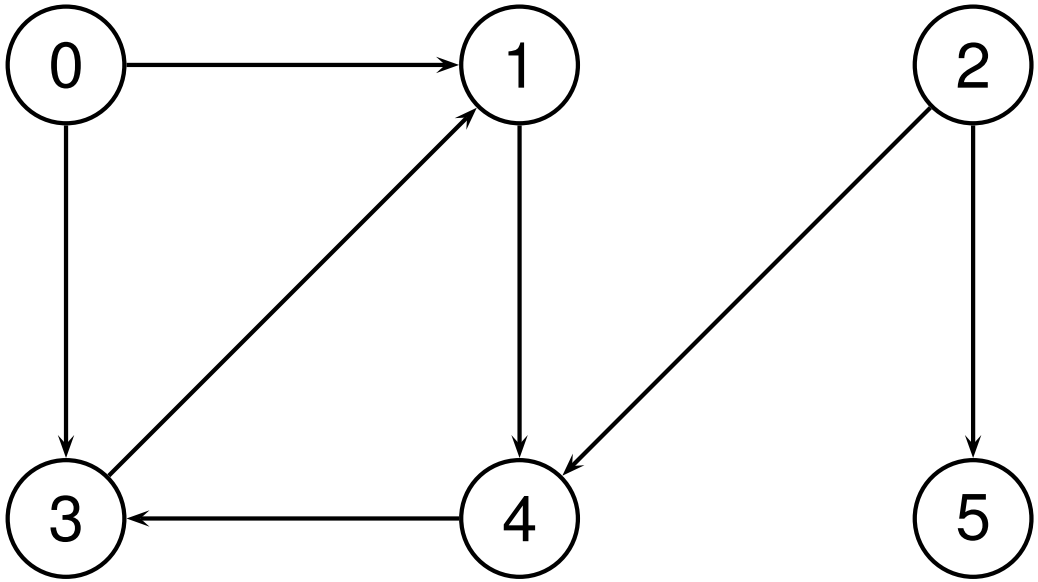
Bölüm 9. Çizge

Olcay Taner Yıldız

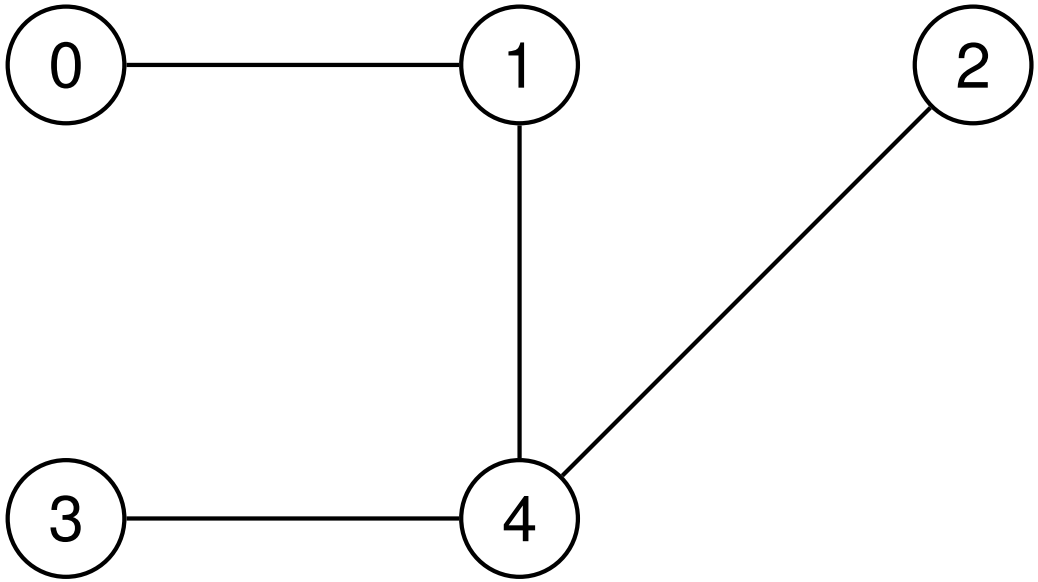
2014

Giriş

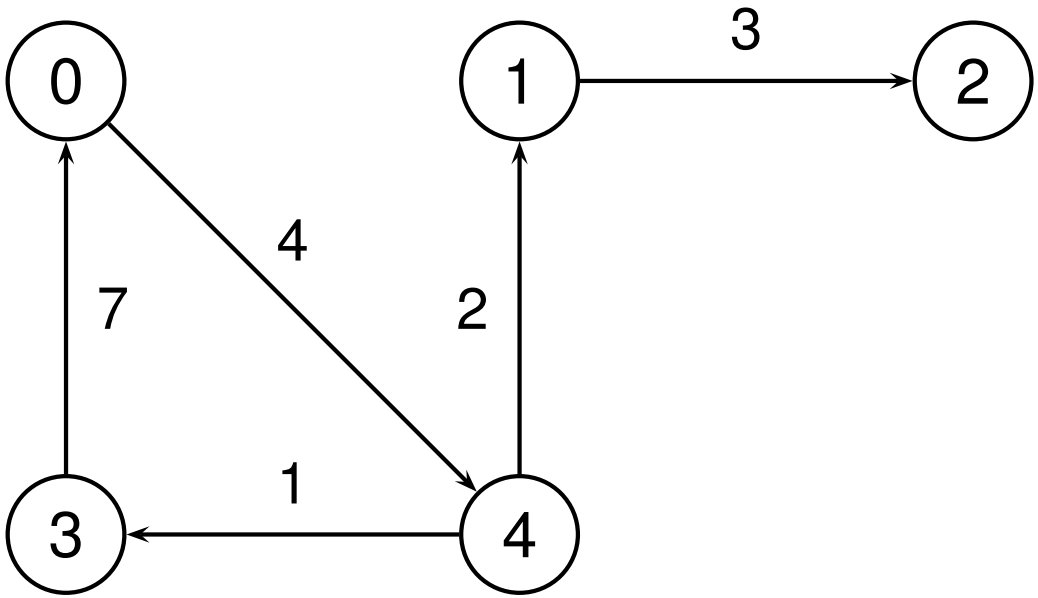
Altı nokta ve yedi doğrudan oluşan yönlü bir çizge



Beş nokta ve dört doğrudan oluşan yönsüz bir çizge



Beş nokta ve beş doğrudan oluşan ağırlıklı bir çizge



Çizge Tanımı

Örnek yönlü ve yönsüz çizgelerin komşu matris ile gösterimi

(a)

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

(b)

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

Örnek ağırlıkli çizgenin komşu matris ile gösterimi

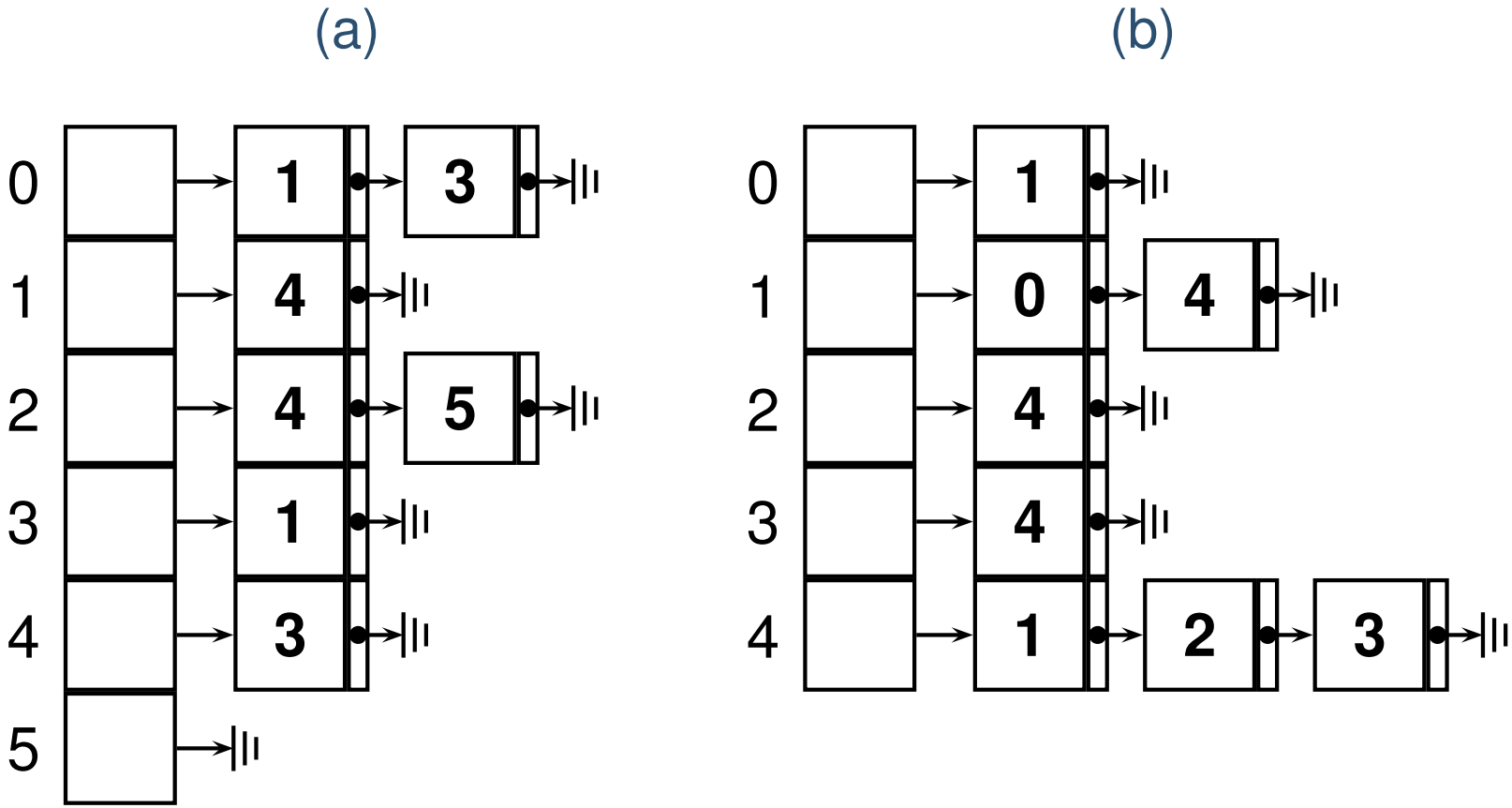
	0	1	2	3	4
0	**0**	**0**	**0**	**0**	**4**
1	**0**	**0**	**3**	**0**	**0**
2	**0**	**0**	**0**	**0**	**0**
3	**7**	**0**	**0**	**0**	**0**
4	**0**	**2**	**0**	**1**	**0**

Komşu matris ile çizge tanımı

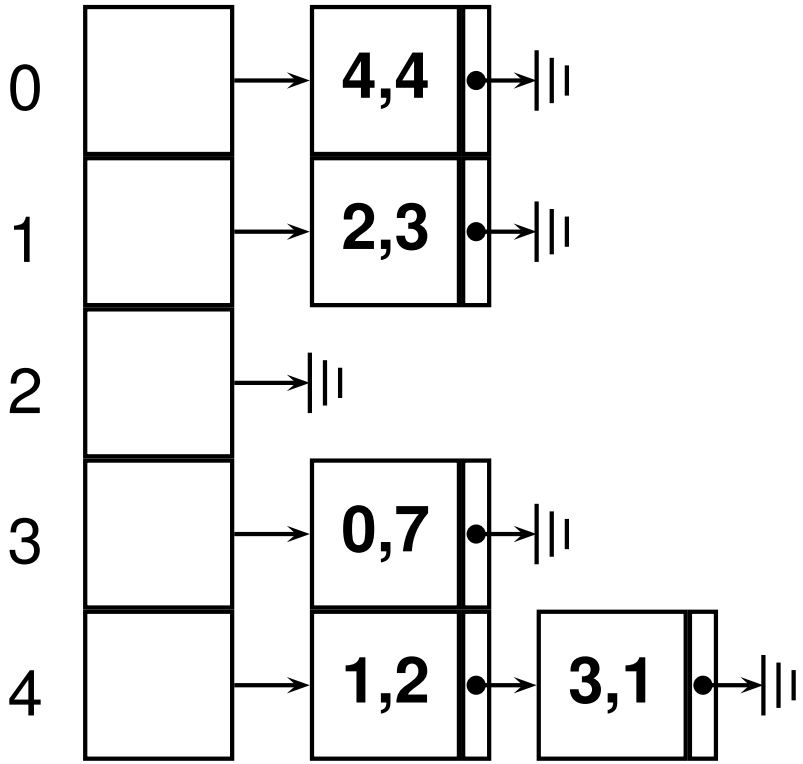

```
10
11
12
```

```
public class Cizge{
    int [][] kenar;
    int N;
    public Cizge(int N){
        int i, j;
        this.N = N;
        kenar = new int[N][N];
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                kenar[i][j] = 0;
    }
}
```

Örnek yönlü ve yönsüz çizgelerin komşu liste ile gösterimi



Örnek ağırlıklı çizgenin komşu liste ile gösterimi



Bir çizgedeki kenar bilgisini tutan eleman veri yapısı tanımı


```
1  public class Eleman{
2      int baslangic;
3      int bitis ;
4      int agirlik ;
5      Eleman ileri ;
6      public Eleman(int baslangic, int bitis, int agirlik){
7          this.baslangic = baslangic;
8          this.bitis  =  bitis ;
9          this.agirlik  =  agirlik ;
10          ileri   = null;
11     }
12 }
```

Komşu liste ile çizge tanımı


```
10
11
```

```
public class Cizge{
    Liste [] kenar;
    int N;
    public Cizge(int N){
        int i;
        this.N = N;
        kenar = new Liste[N];
        for (i = 0; i < N; i++)
            kenar[i] = new Liste();
    }
}
```

Kenar Ekleme

Komşu matris ile tanımlı bir çizgeye yeni bir kenar ekleme

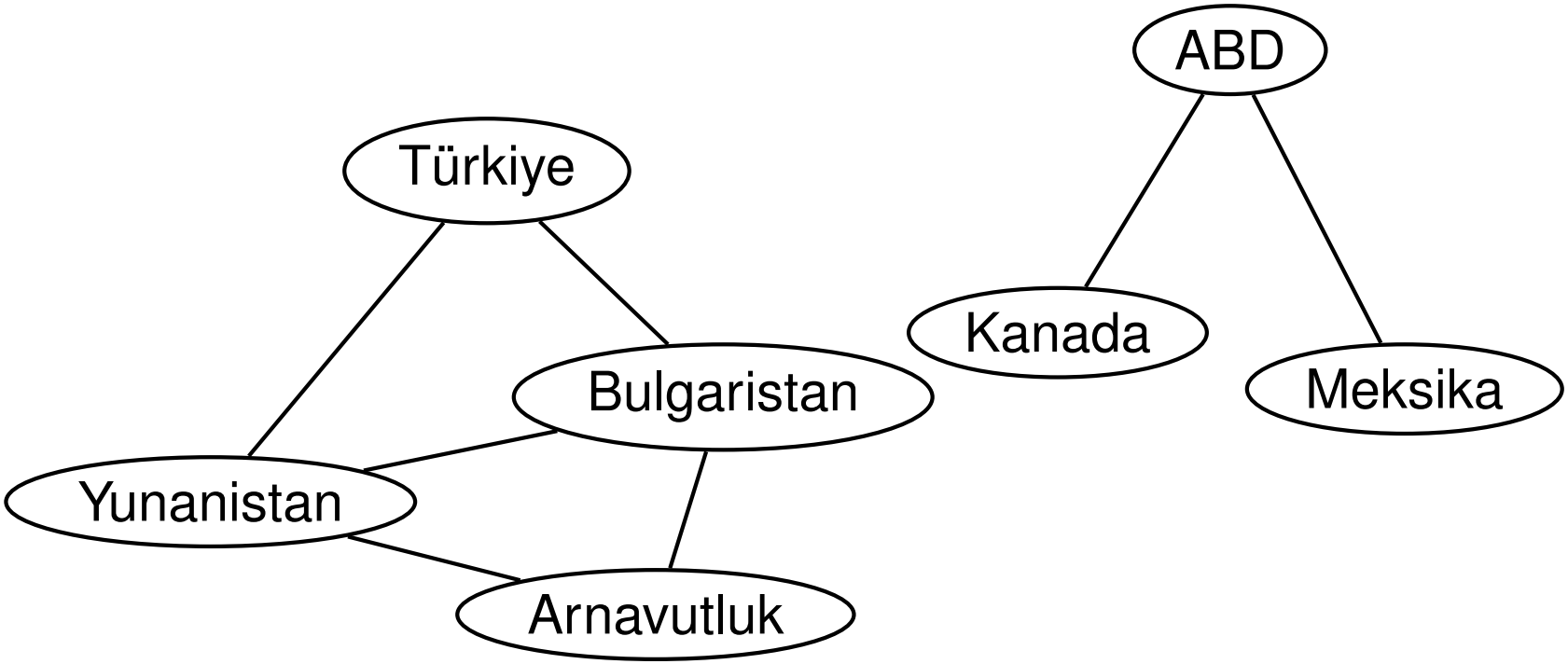
```
void kenarEkle(int baslangic, int bitis, int agirlik){
    kenar[baslangic][ bitis ] = agirlik ;
}
```

Komşu liste ile tanımlı bir çizgeye yeni bir kenar ekleme


```
void kenarEkle(int baslangic, int bitis, int agirlik){
    Eleman yeni;
    yeni = new Eleman(baslangic, bitis,  agirlik );
    kenar[baslangic].listeyeEkle(yeni);
}
```

Uygulama: Çizgede Bağlı Parçalar

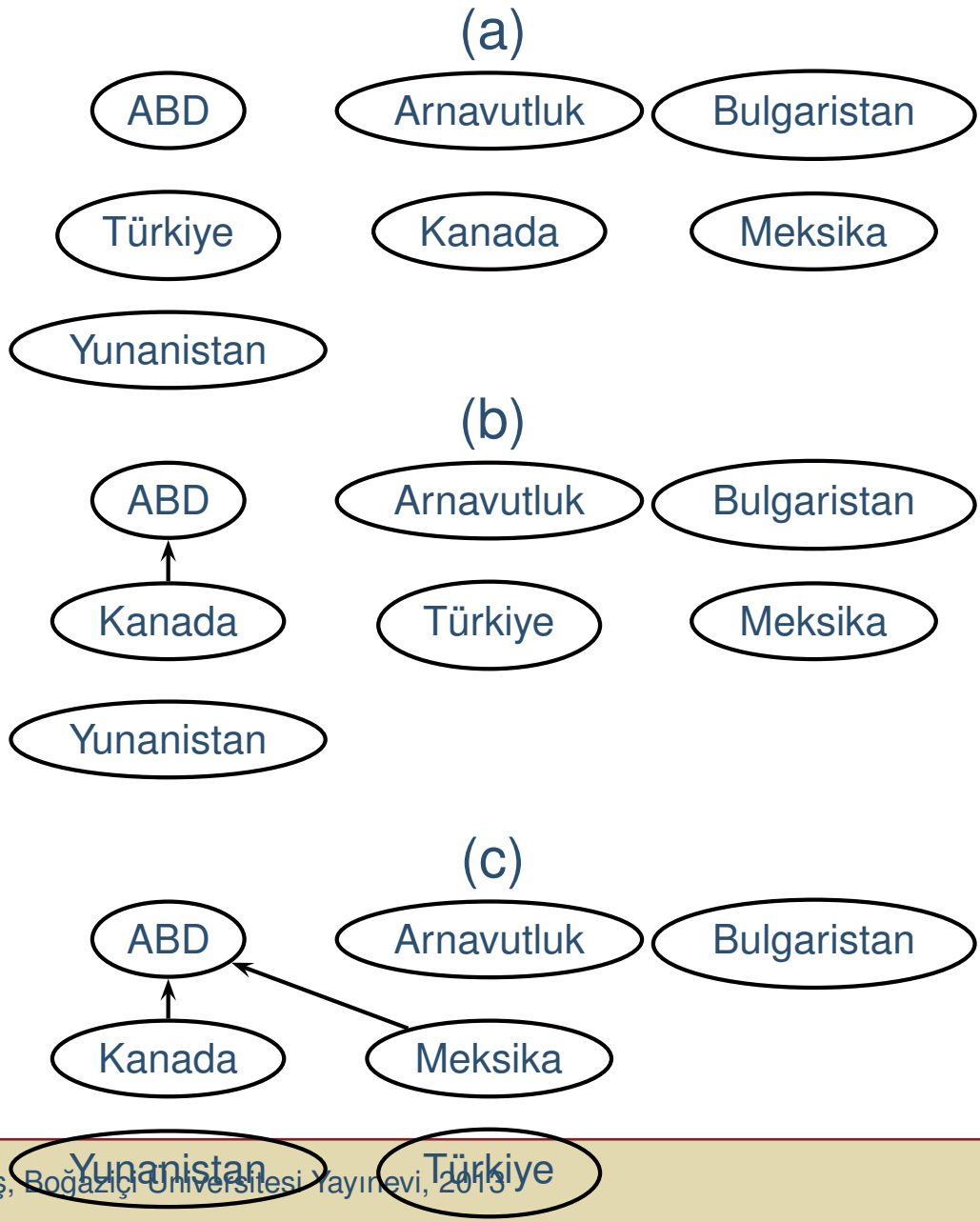
Örnek yedi ülkeli bir harita



Bir çizgedeki bağlı parçaları bulan ayrık küme algoritması


```
void bagliParcaBulAyrikKume(){
    int v, v1, v2;
    Eleman dugum;
    Ayrikkume a = new Ayrikkume(N);
    for (v1 = 0; v1 < N; v1++){
        dugum = kenar[v1].bas;
        while (dugum != null){
            v2 = dugum.bitis;
            if (a.kumeBul(v1) != a.kumeBul(v2))
                a.kumeBirlestir(v1, v2);
            dugum = dugum.ileri;
        }
    }
}
```

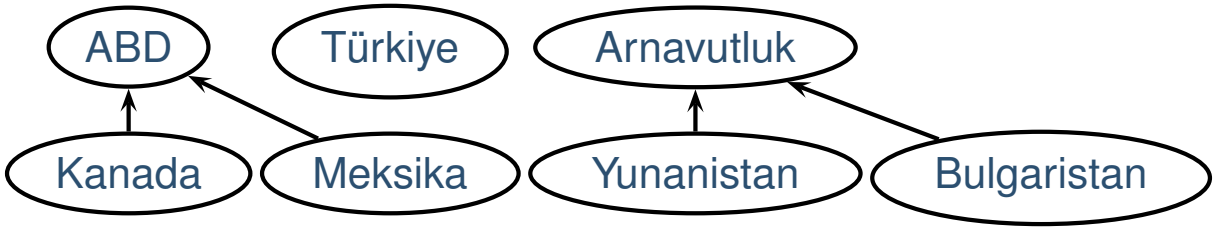
Önceki şekilde verilen çizgedeki bağlı parçaların ayrık küme metoduyla bulunması (1)



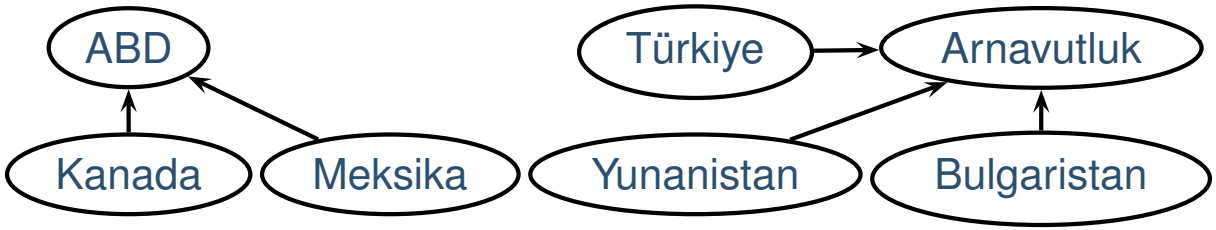
Önceki şekilde verilen çizgedeki bağlı parçaların ayrık küme metoduyla bulunması (2)

(d)

(e)



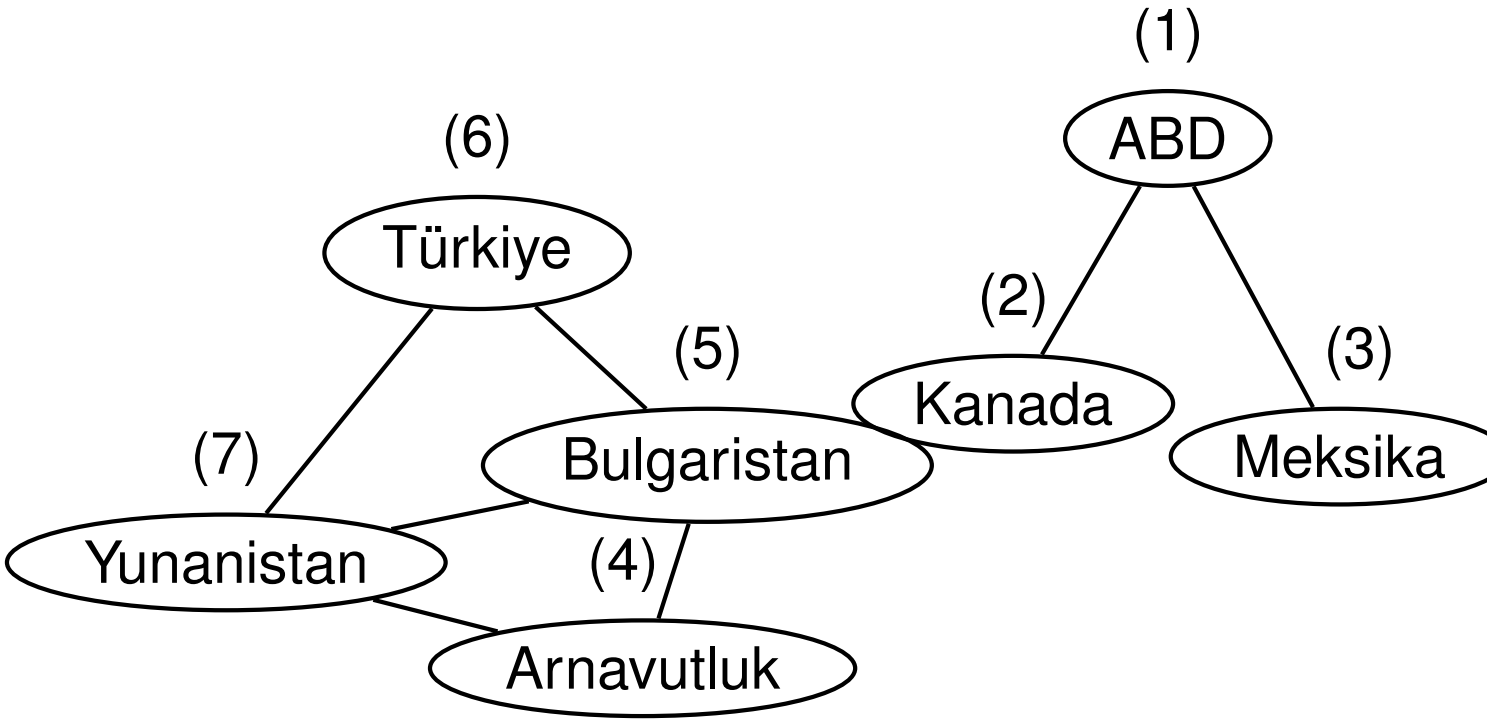
(f)



Verilen bir çizgedeki bağlı parçaları bulan derin arama algoritması


```
1  int bagliParcaBulDerinArama(){
2      int v, parcaSayisi = 0;
3      for (v = 0;  v < N; v++)
4          gezildi [v]  = false;
5      for (v = 0;  v < N; v++)
6          if (! gezildi [v ]){
7              gezildi [v]  = true;
8              derinArama(v);
9              parcaSayisi++;
10         }
11     return parcaSayisi;
12 }
13 void derinArama(int x){
14     Eleman dugum;
15     int y;
16     dugum = kenar[x].bas;
17     while (dugum != null){
18         y  = dugum.bitis;
19         if (! gezildi [y ]){
20             gezildi [y]  = true;
21             derinArama(y);
22         }
23         dugum = dugum.ileri;
24     }
25 }
```

Önceki şekilde verilen çizgedeki bağlı parçaların derin arama metoduyla bulunması



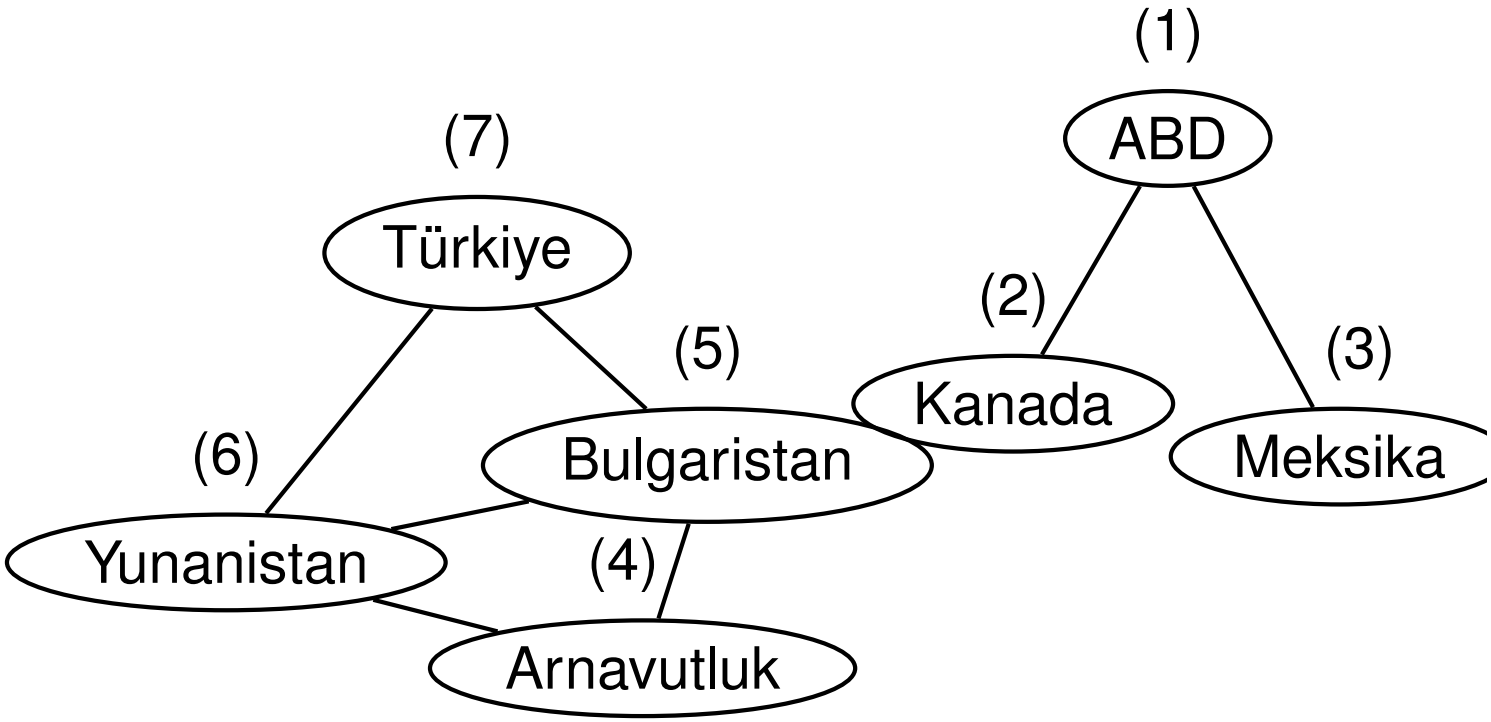
Verilen bir çizgedeki bağlı parçaları bulan geniş arama algoritması (1)


```
1  int bagliParcaBulGenisArama(){
2      int v, parcaSayisi = 0;
3      for (v = 0; v < N; v++)
4          gezildi [v] = false;
5      for (v = 0; v < N; v++)
6          if (! gezildi [v ]){
7              gezildi [v] = true;
8              genisArama(v);
9              parcaSayisi++;
10         }
11     return parcaSayisi;
12 }
```

Verilen bir çizgedeki bağlı parçaları bulan geniş arama algoritması (2)


```
13 void genisArama(int x){
14     Eleman dugum;
15     int y, v;
16     Ornek e;
17     Kuyruk k = new Kuyruk(100);
18     e = new Ornek(x);
19     k.kuyrugaEkle(e);
20     while (!k.kuyrukBos()){
21         e = k.kuyrukSil ();
22         v = e. icerik ;
23         dugum = kenar[v].bas;
24         while (dugum != null){
25             y = dugum.bitis;
26             if (! gezildi [y ]){
27                 gezildi [y] = true;
28                 e = new Ornek(y);
29                 k.kuyrugaEkle(e);
30             }
31             dugum = dugum.ileri;
32         }
33     }
34 }
```

Önceki çizgedeki bağlı parçaların geniş arama metoduyla bulunması



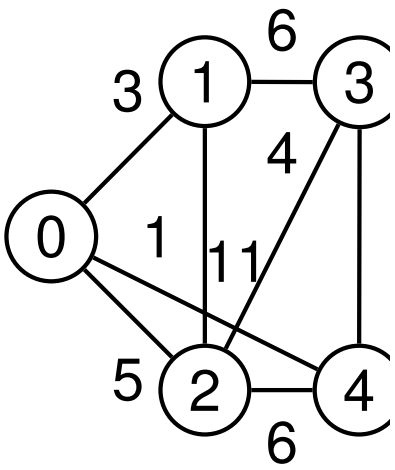
Uygulama: En Kısa Yol Problemi

Bellman-Ford algoritması


```
1  void bellmanFord(){
2      int i, u, v, dy, d[], once[];
3      d = new int[N];
4      once = new int[N];
5      for (i = 0; i < N; i++){
6          d[i] = Integer.MAX_VALUE;
7          once[i] = -1;
8      }
9      d[0] = 0;
10     for (i = 0; i < N - 1; i++)
11         for (u = 0; u < N; u++)
12             for (v = 0; v < N; v++){
13                 dy = kenar[u][v] + d[u];
14                 if (d[v] > dy){
15                     d[v] = dy;
16                     once[v] = u;
17                 }
18             }
19 }
```

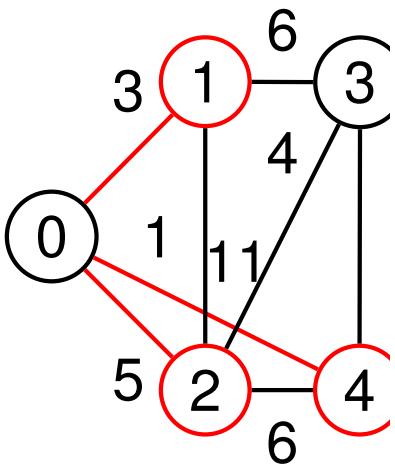
Bellman-Ford algoritmasının 5 noktalı bir çizge üstünde gösterimi (1)

(I)



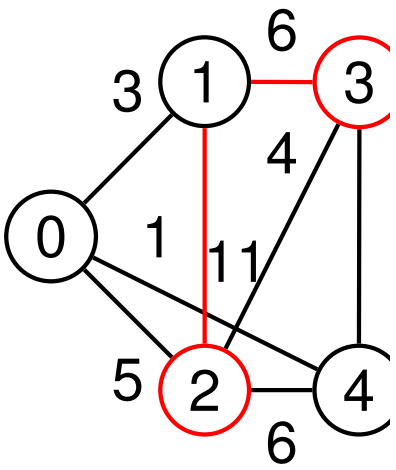
0	1	2	3	4
0	∞	∞	∞	∞
-1	-1	-1	-1	-1

(II)



0	1	2	3	4
0	3	5	∞	11
-1	0	0	-1	0

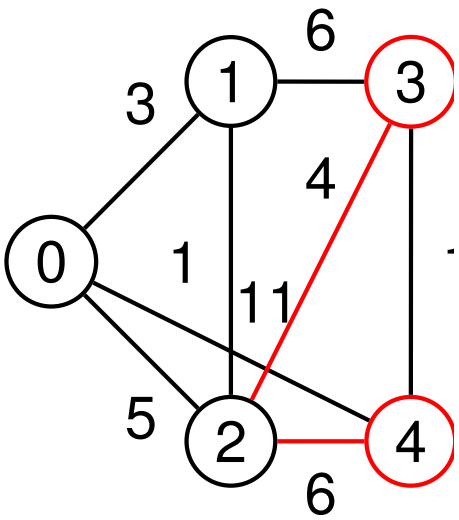
(III)



0	1	2	3	4
0	3	4	9	11
-1	0	1	1	0

Bellman-Ford algoritmasının 5 noktalı bir çizge üstünde gösterimi (2)

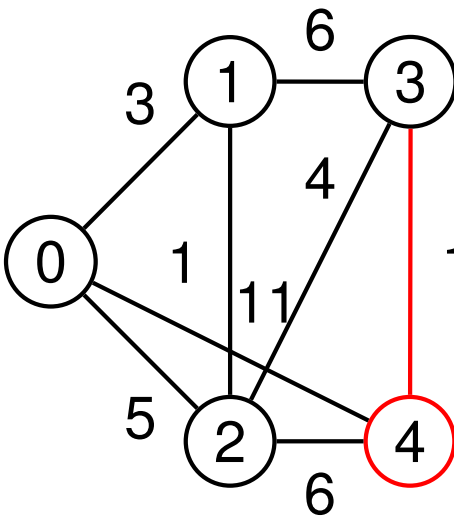
(IV)



0	1	2	3	4
0	3	4	8	10

-1	**0**	**1**	**2**	**2**

(V)



0	1	2	3	4
0	3	4	8	9

-1	**0**	**1**	**2**	**3**

once ***dizisi verildiğinde başlangıç noktasından bitiş noktasına giden en kısa yolu bulan algoritma***

```
1  void enKisaYol(int[] once, int bitis){
2      Liste  yol;
3      Eleman e;
4      int i;
5      yol  = new Liste();
6      i  =  bitis ;
7      while (once[i] !=  -1){
8          e = new Eleman(i);
9          yol.listeBasinaEkle(e);
10         i  = once[i];
11     }
12     e = new Eleman(i);
13     yol.listeBasinaEkle(e);
14     return yol;
15 }
```

Dijkstra algoritması (1)

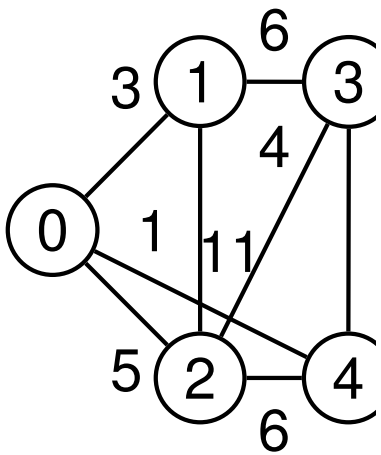

```
1  void dijkstra (){
2      int i, u, v, pos, dy, d [], once[];
3      Yigin T;
4      Nokta e;
5      d = new int[N];
6      once = new int[N];
7      for (i = 0; i < N; i++){
8          d[i] = Integer.MAX_VALUE;
9          once[i] = -1;
10     }
11     d[0] = 0;
12     T = new Yigin();
13     for (i = 0; i < N; i++){
14         e = new Nokta(d[i], i);
15         T.elemanEkle(e);
16     }
```

Dijkstra algoritması (2)


```
17      while (!T.yiginBos()){
18          e = T.asgariDondur();
19          u = e.ad;
20          for (v = 0; v < N; v++){
21              dy = kenar[u][v] + d[u];
22              if (d[v] > dy){
23                  pos = T.yiginArama(v);
24                  d[v] = dy;
25                  T.degerDegistir(pos, d[v]);
26                  once[v] = u;
27              }
28          }
29      }
30  }
```

Dijkstra algoritmasının 5 noktalı bir çizge üstünde gösterimi (1)

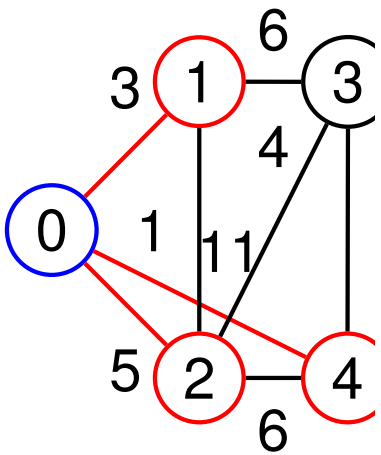
(I)



0	1	2	3	4
0	∞	∞	∞	∞

-1	-1	-1	-1	-1

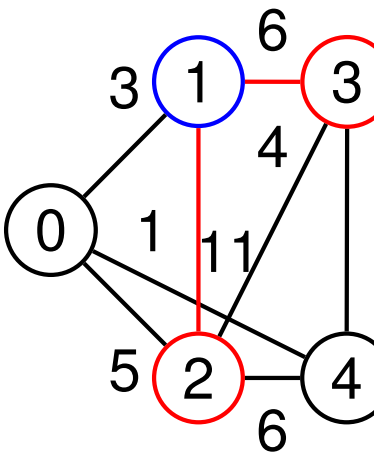
(II)



0	1	2	3	4
0	3	5	∞	11

-1	0	0	-1	0

(III)

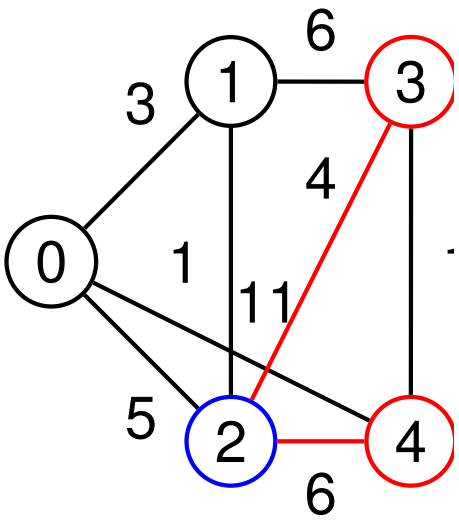


0	1	2	3	4
0	3	4	9	11

-1	0	1	1	0

Dijkstra algoritmasının 5 noktalı bir çizge üstünde gösterimi (2)

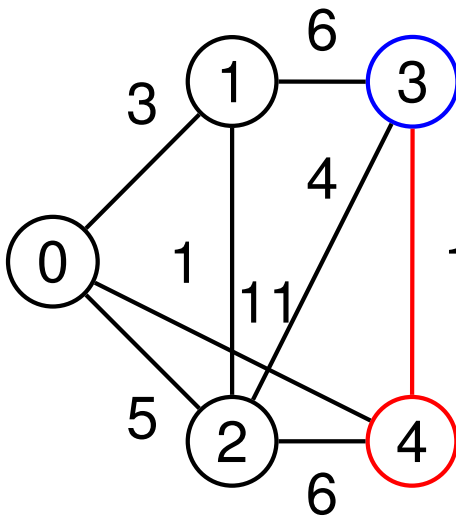
(IV)



0	1	2	3	4
0	3	4	8	10

-1	**0**	**1**	**2**	**2**

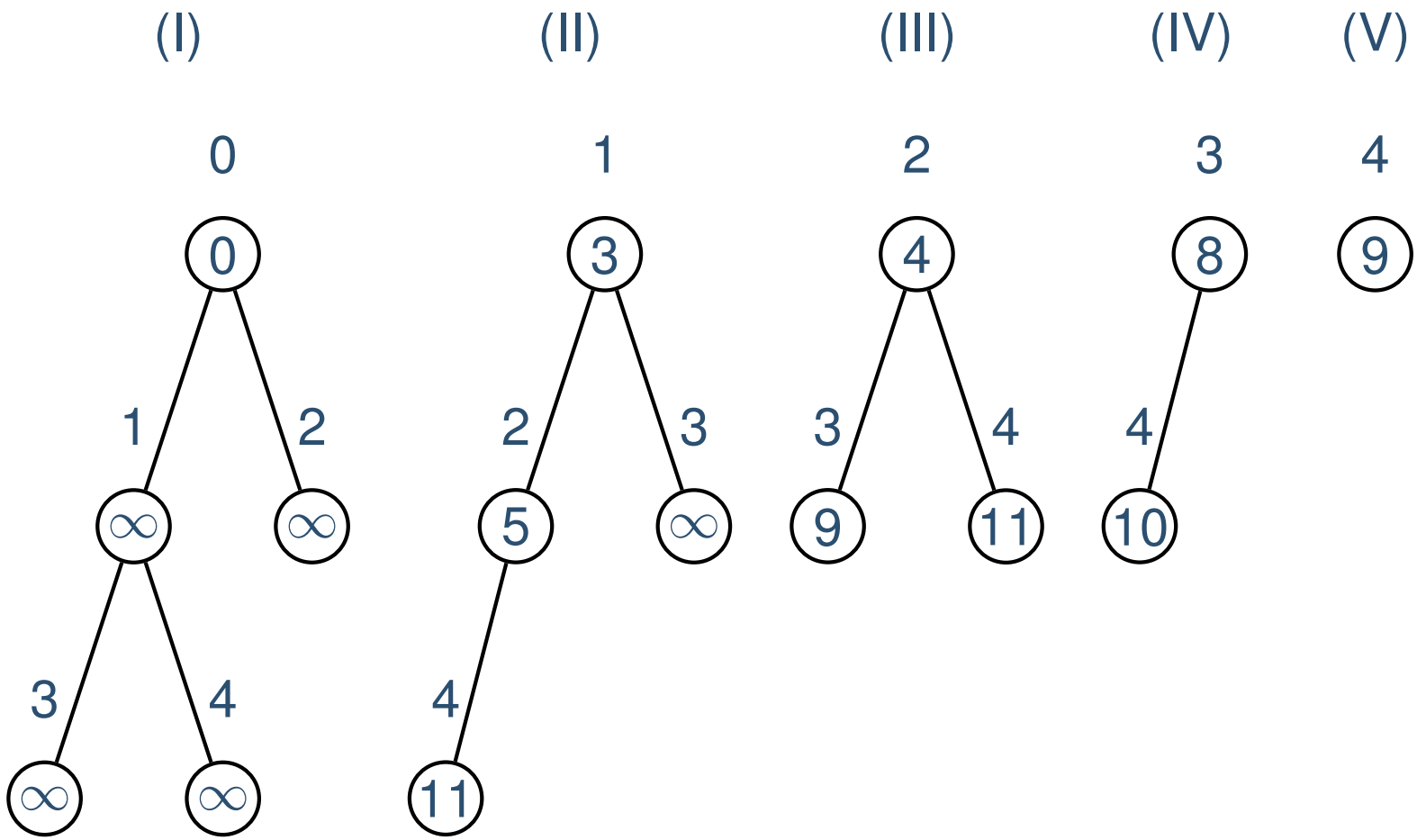
(V)



0	1	2	3	4
0	3	4	8	9

-1	**0**	**1**	**2**	**3**

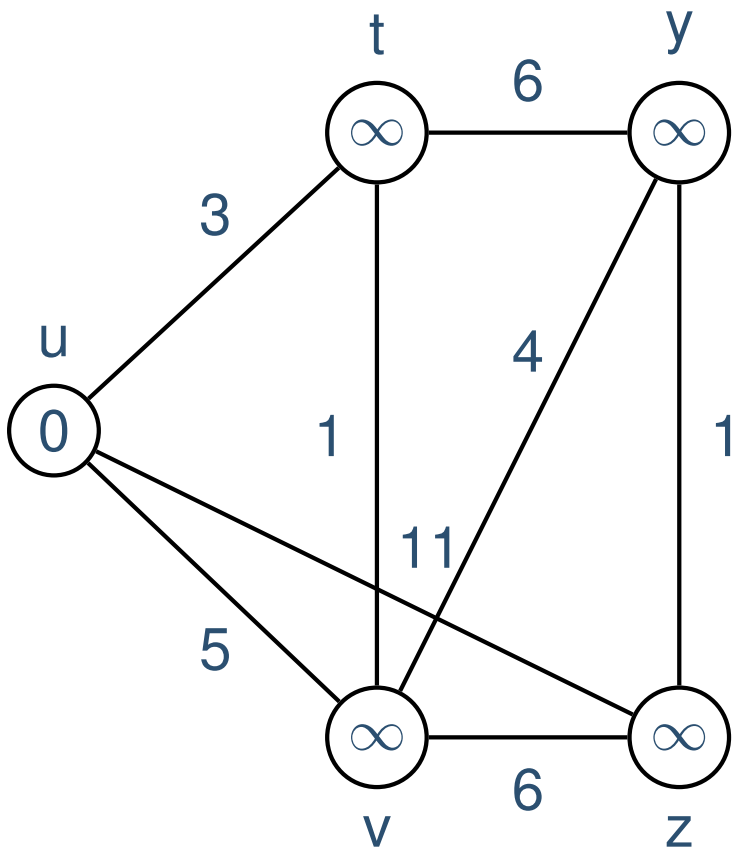
Dijkstra algoritma uygulamasındaki T yığını



Floyd-Warshall algoritması


```
10
11
12
13
14
```

```
void floydWarshall(){
   int i, j, k, dy, d [][];
   d = new int[N][N];
   for (i = 0; i < N; i++)
      for (j = 0; j < N; j++)
         d[i][j] = kenar[i][j];
   for (k = 0; k < N; k++)
      for (i = 0; i < N; i++)
         for (j = 0; j < N; j++){
            dy = d[i][k] + d[k][j];
            if (d[i][j] > dy)
               d[i][j] = dy;
         }
}
```



	u	t	v	y	z
u	0	3	5	∞	11
t	3	0	1	6	∞
v	5	1	0	4	6
y	∞	6	4	0	1
z	11	∞	6	1	0

Floyd-Warshall algoritmasının ilk 4 aşamasının gösterimi

(1)

	u	t	v	y	z
u	0	3	5	∞	11
t	3	0	1	6	**14**
v	5	1	0	4	6
y	∞	6	4	0	1
z	11	**14**	6	1	0

(2)

	u	t	v	y	z
u	0	3	**4**	**9**	11
t	3	0	1	6	14
v	**4**	1	0	4	6
y	**9**	6	4	0	1
z	11	14	6	1	0

(3)

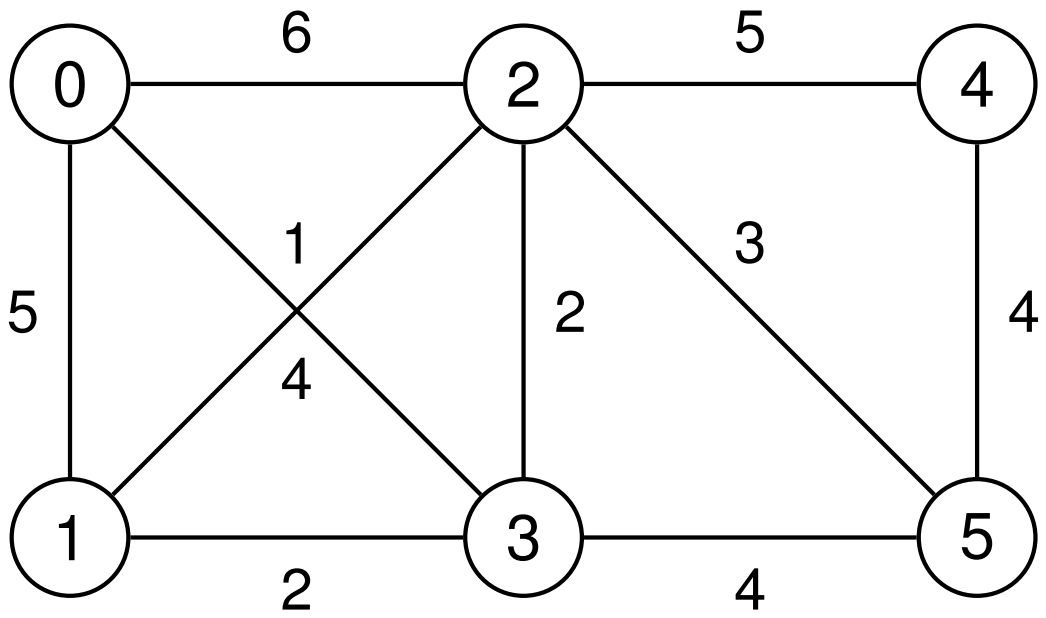
	u	t	v	y	z
u	0	3	4	**8**	**10**
t	3	0	1	**5**	**7**
v	4	1	0	4	6
y	**8**	**5**	4	0	1
z	**10**	**7**	6	1	0

(4)

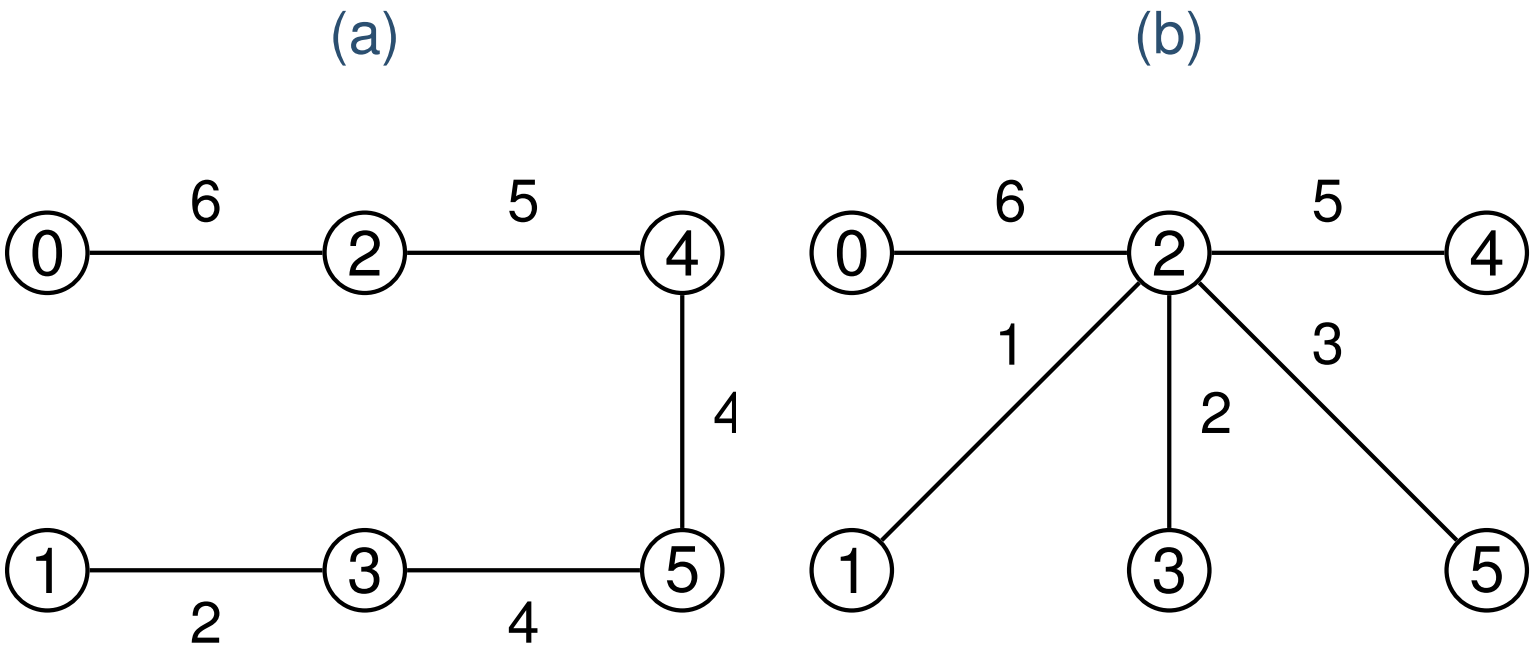
	u	t	v	y	z
u	0	3	4	8	**9**
t	3	0	1	5	**6**
v	4	1	0	4	**5**
y	8	5	4	0	1
z	**9**	**6**	**5**	1	0

Uygulama: En Küçük Kapsayan Ağaç

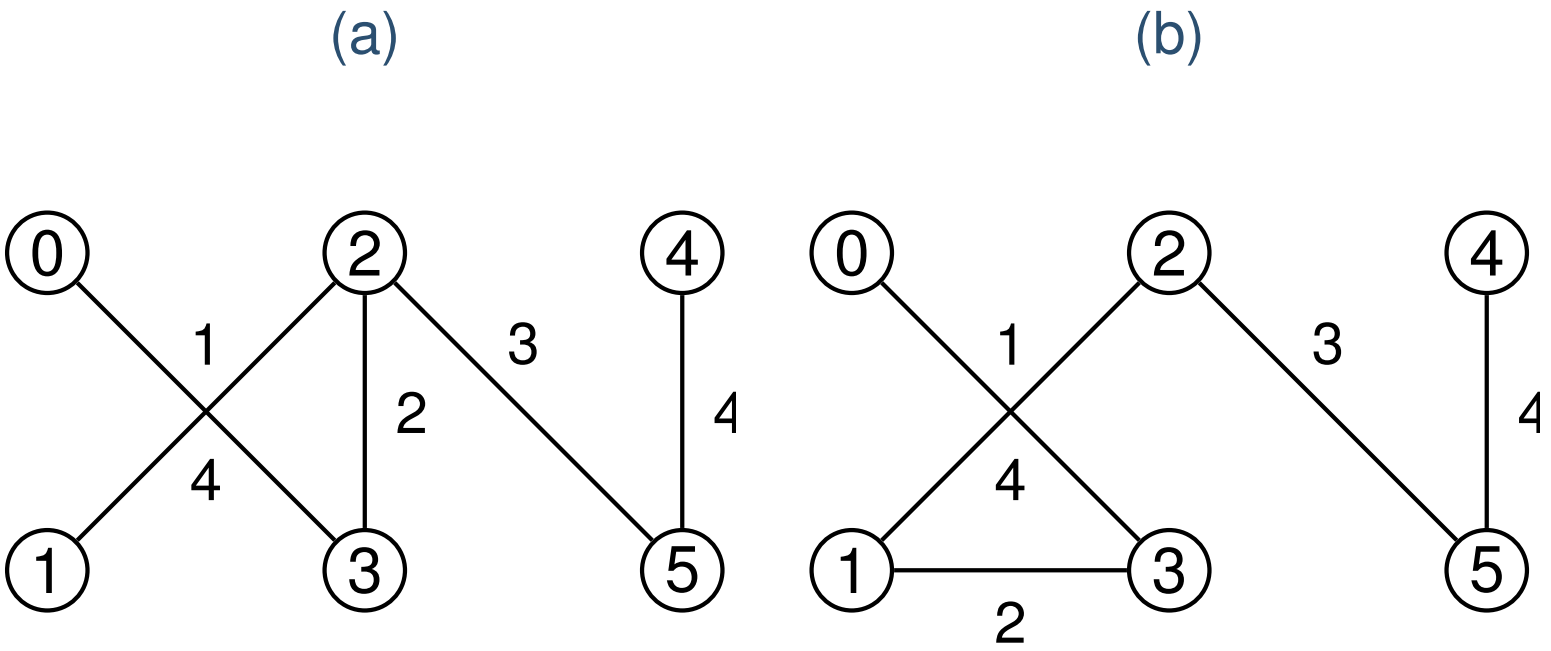
Altı noktadan oluşan ağırlıklı bir çizge



Önceki çizgeyi kapsayan iki ağaç



Önceki çizgeyi en küçük kapsayan iki ağaç



Bir çizgedeki tüm kenarları bir bağlı listede döndüren algoritma

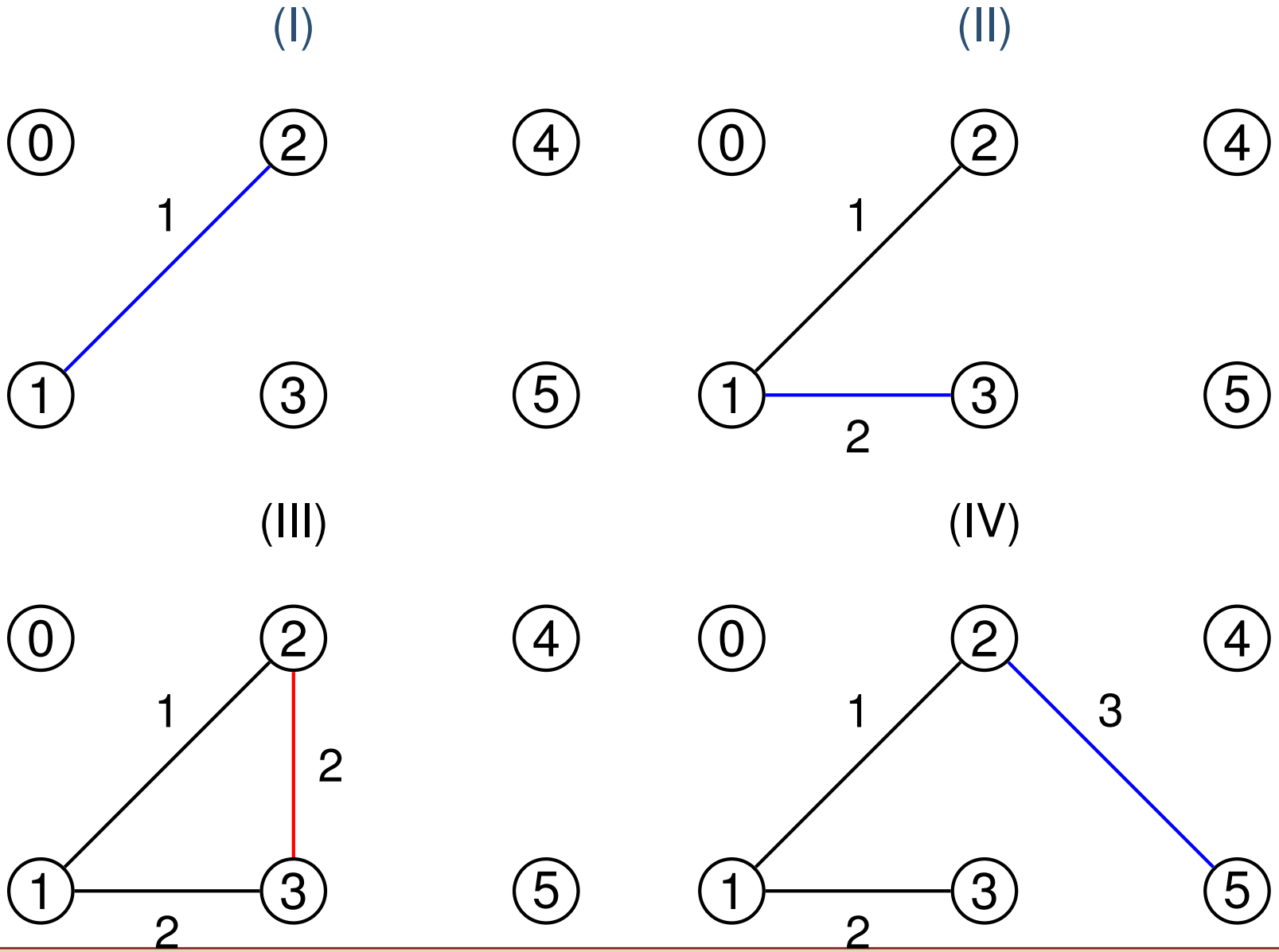

```
10
11
12
13
14
15
```

```
Liste  kenarListesi (){
    int v;
    Eleman dugum, yeni;
    Liste  kenarlar;
    kenarlar  = new Liste ();
    for (v = 0;  v < N; v++){
        dugum = kenar[v].bas;
        while (dugum != null){
            yeni  = new Eleman(dugum.baslangic, dugum.bitis, dugum.agirlik);
            kenarlar. listeyeEkle (yeni );
            dugum = dugum.ileri;
        }
    }
    return kenarlar;
}
```

Verilen bir çizgedeki en küçük kapsayan ağacı bulan Kruskal algoritması

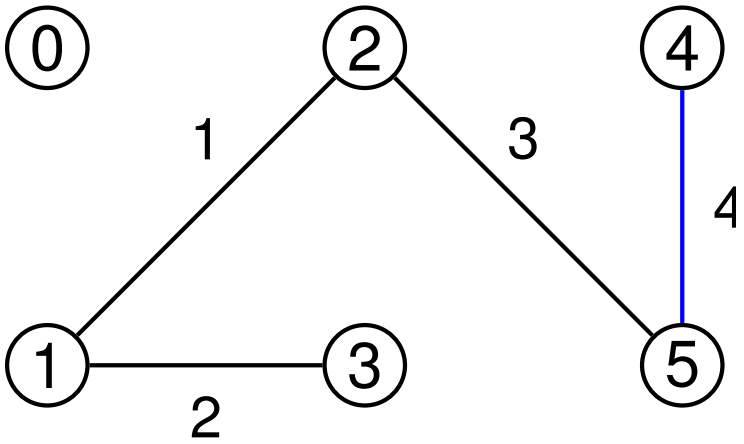

```
10  void kruskal(){
11      int v, u, kenarSayi;
12      Liste kenarlar;
13      Eleman dugum;
14      Ayrikkume a = new Ayrikkume(N);
15      kenarlar = kenarListesi ();
16      kenarlar. sirala ();
17      dugum = kenarlar.bas;
18      kenarSayi = 0;
19      while (kenarSayi < N - 1){
20          u = dugum.baslangic;
21          v = dugum.bitis;
22          if (a.kumeBul(u) != a.kumeBul(v)){
23              a.kumeBirlestir(u, v);
24              kenarSayi++;
25          }
26          dugum = dugum.ileri;
27      }
28  }
```

Kruskal algoritmasının önceki çizge üzerinde uygulanması (1)

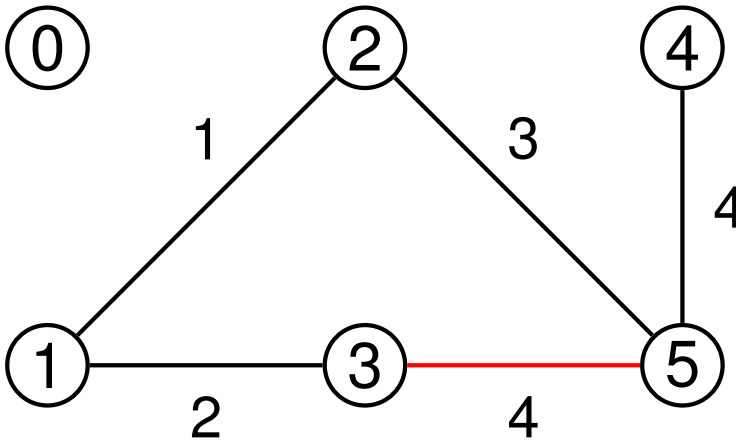


Kruskal algoritmasının önceki çizge üzerinde uygulanması (2)

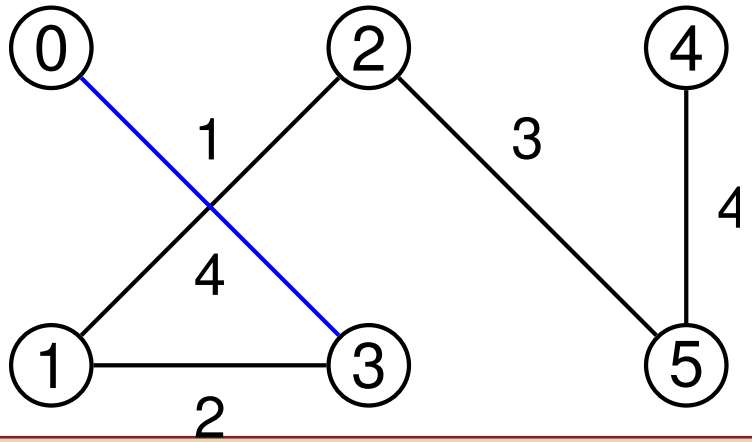
(V)



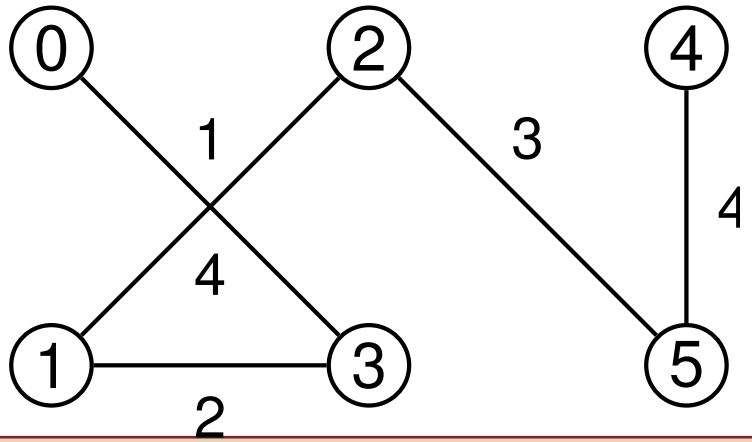
(VI)



(VII)



(VIII)



Verilen bir çizgedeki en küçük kapsayan ağacı bulan Prim algoritması (1)

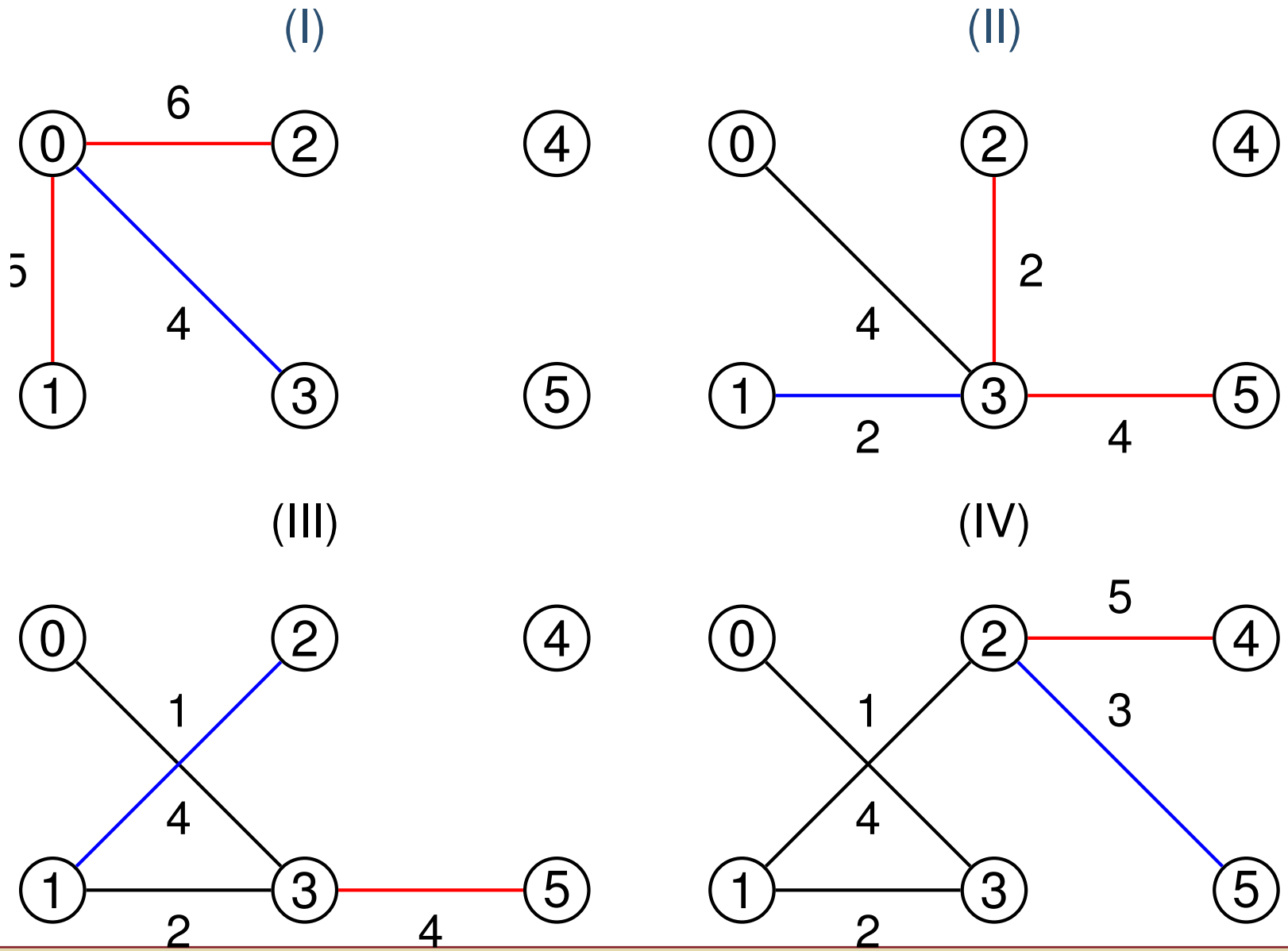

```
10
11
12
13
14
15
16
17
```

```
void prim(){
    int i, u, v, pos, d[], once[];
    Yigin T;
    Nokta e;
    Eleman k;
    d = new int[N];
    once = new int[N];
    for (i = 0; i < N; i++){
        d[i] = Integer.MAX_VALUE;
        once[i] = -1;
    }
    d[0] = 0;
    T = new Yigin();
    for (i = 0; i < N; i++){
        e = new Nokta(d[i], i);
        T.elemanEkle(e);
    }
```

Verilen bir çizgedeki en küçük kapsayan ağacı bulan Prim algoritması (2)

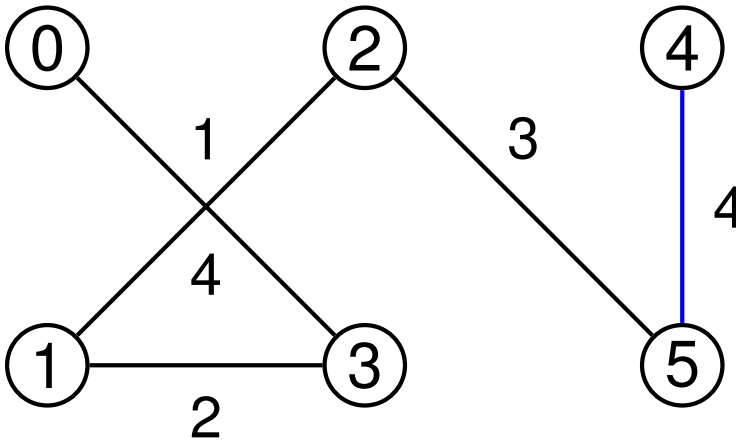

```
18      while (!T.yiginBos()){
19          e = T.asgariDondur();
20          u = e.ad;
21          k = kenar[u].bas;
22          while (k != NULL){
23              v = k. bitis ;
24              if (d[v] > k. agirlik ){
25                  pos = T.yiginArama(v);
26                  d[v] = k. agirlik ;
27                  T.degerDegistir(pos, d[v ]);
28                  once[v] = u;
29              }
30              k = k. ileri ;
31          }
32      }
33  }
```

Prim algoritmasının önceki çizge üzerinde uygulanması (1)

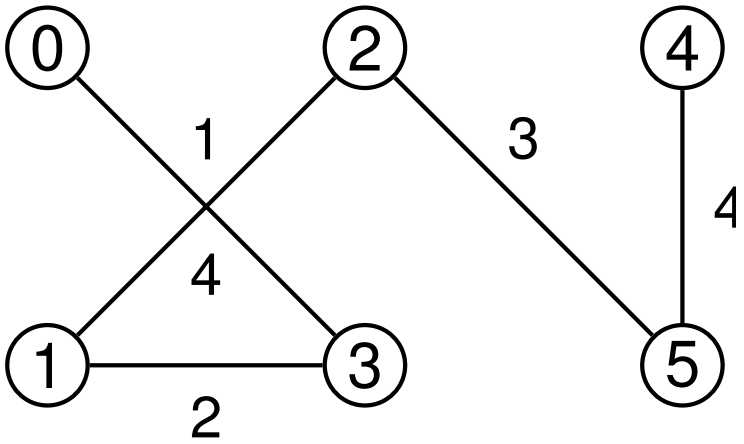


Prim algoritmasının önceki çizge üzerinde uygulanması (2)

(V)



(VI)



Prim algoritma uygulamasındaki T yığını