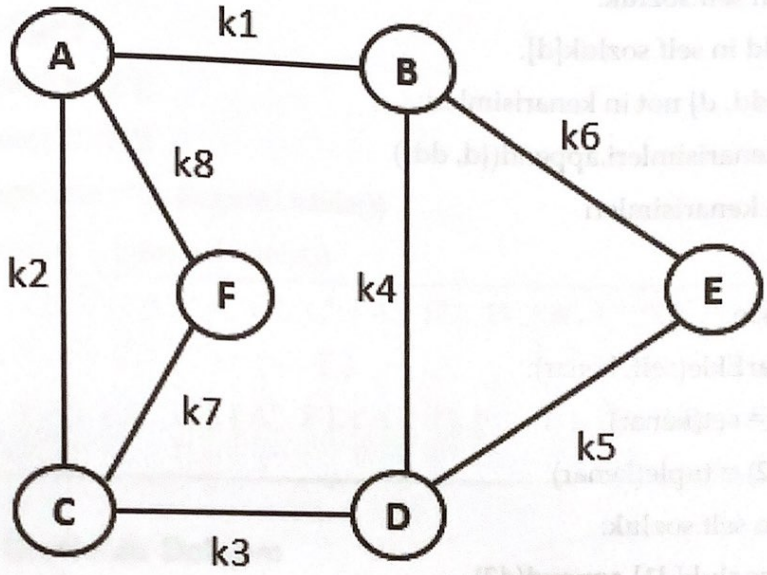
```
  def kenarListele(self):
    kenarisimleri = []
    for d in self.sozluk:
      for dd in self.sozluk[d]:
        if {dd, d} not in kenarisimleri:
          kenarisimleri.append({d, dd})
    return kenarisimleri

graf = {
  "A" : ["a","c"],
  "B" : ["a","d", "e"],
  "C" : ["a", "d"],
  "D" : ["b","c","e"],
  "E" : ["b","d"]
}

print(graf)
g = graph(graf)
print("Düğümler : ",g.dugumListele())
print('Kenarlar: ', g.kenarListele())
```

```
Düğümler :  ['A', 'B', 'C', 'D', 'E']
Kenarlar:  [{'a', 'A'}, {'c', 'A'}, {'a', 'B'}, {'d', 'B'}, {'e', 'B'}, {'a', 'C'}, {'C', 'd'}, {'D', 'b'}, {'D', 'c'}, {'e', 'D'}, {'E', 'b'}, {'E', 'd'}]
```

13.4. Düğüm ve Kenar Ekleme

Düğüm ve kenar eklemek sözlüğe bir değer eklemektir. Ancak eklenen düğümün bağlı olduğu düğüm sayısının birden fazla olması durumunda birden fazla bağlantı veya kenar eklenmiş olacaktır. Dolayısıyla eklenen kenarlar demet veya liste şeklinde verilmesi gerekir. Yukarıda verilen grafa E düğümü ve k7, k8 kenarların eklenmesini gerçekleştirelim. F düğümünün eklenmesi durumunda A ve C

düğümlerine yeni bağlantılarda eklenmiş olacaktır. A ve F düğümü arasında k8 (a,f), C ile F arasında k7 (c,f) eklenmiş olacaktır. Aşağıdaki graf elde edilmiş olunacaktır. Tüm kodlar aşağıdaki örnekte verilmiştir.




```
class graph:
  def __init__(self,sozluk=None):
    if sozluk is None:
      sozluk = {}
    self.sozluk = sozluk

# Düğümlerin Yazdırılması
  def dugumListele(self):
    return list(self.sozluk.keys())

# Kenarların Listelenmesi
  def kenarlar(self):
    return self.kenarListele()
```

```
  def kenarListele(self):
    kenarisimleri = []
    for d in self.sozluk:
      for dd in self.sozluk[d]:
        if {dd, d} not in kenarisimleri:
          kenarisimleri.append({d, dd})
    return kenarisimleri

#Kenar ekle
  def kenarEkle(self, kenar):
    kenar = set(kenar)
    (d1, d2) = tuple(kenar)
    if d1 in self.sozluk:
      self.sozluk[d1].append(d2)
    else:
      self.sozluk[d1] = [d2]

#düğüm Ekle
  def dugumEkle(self, dugum):
    if dugum not in self.sozluk:
      self.sozluk[dugum] = []

graf = {
  "A" : ["A","C"],
  "B" : ["A","D", "E"],
  "C" : ["A", "D"],
  "D" : ["B","C","E"],
  "E" : ["B","D"]
}
```

```
print(graf)
g = graph(graf)

g.dugumEkle('F')
g.kenarEkle({'A','F'})
g.kenarEkle({'C','F'})
print("Düğümler : ",g.dugumListele())
print('Kenarlar: ', g.kenarListele())
```

```
{'A': ['A', 'C'], 'B': ['A', 'D', 'E'], 'C': ['A', 'D'], 'D': ['B', 'C', 'E'], 'E': ['B', 'D']}
Düğümler:  ['A', 'B', 'C', 'D', 'E', 'F']
Kenarlar:  [{'A'}, {'A', 'C'}, {'A', 'F'}, {'A', 'B'}, {'D', 'B'}, {'E', 'B'}, {'D', 'C'}, {'E', 'D'}, {'F', 'C'}]
```

13.5. Graflarda Dolaşım

Graf düğümleri arasında dolaşım için genel olarak iki yaklaşım kullanılmaktadır.

- **Önce Derinlik (Depth first) dolaşımı:** Graf üzerindeki bir X düğümüne gidildikten sonra X düğümün bir komşusu seçilir ve ziyaret edilir. Ardından onun bir komşusu seçilir ve ard arda komşu seçimi yapılarak devam edilir. Komşu kalmayıncaya kadar devam edilir.
- **Önce Genişlik (Breadth first) dolaşımı:** Graf üzerindeki bir X düğümüne gidildikten sonra X düğümün sırasıyla tüm komşu düğümlerine gidilir ardından tüm komşu düğümlerin komşu düğümlerine gidilir.

13.5.1. Önce Derinlik (Depth First) Dolaşımı İşlem Adımları

- Önce bir başlangıç X düğümü seçilir ve ziyaret edilir.
- Seçilen X düğümün bir komşusu seçilir ve ziyaret edilir.
- 2. adım ziyaret edilecek komşu kalmayıncaya kadar tekrar edilir.
- Komşu kalmadığında tekrar geri dönülür ve önceki ziyaret edilmemiş düğümler için adım 2 ve 3 tekrar edilir.

Önce derinlik dolaşımı için bir örnek aşağıda verilmiştir. Örnek graf üzerinde başlangıçta "0" düğümü seçilmiştir. Daha sonra komşuları olan "3" düğümüne