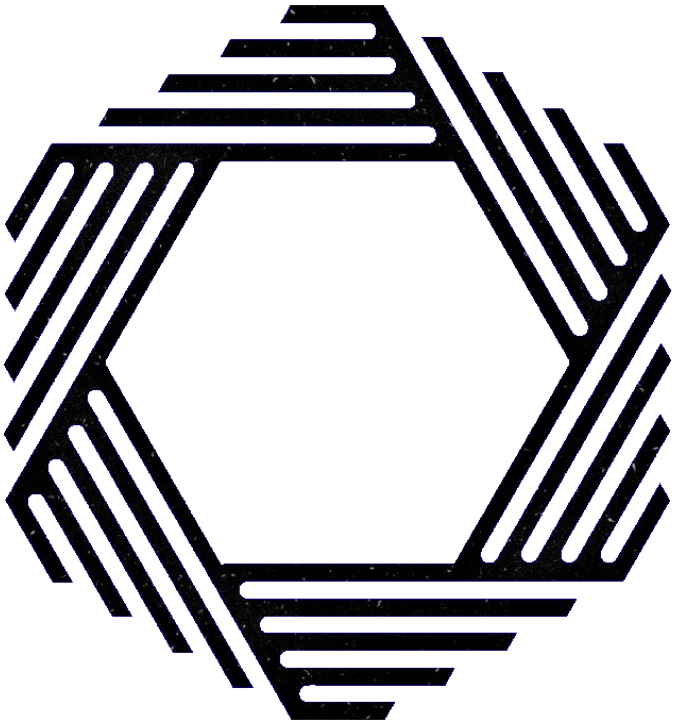
Dijkstra Algorithması

Python



Süha Arslan
19630355@stu.omu.edu.tr

programmer-666/DijkstraWithPython

Dijkstra Algoritması | Python

Dijkstra
Sezgisel arama algoritmalarından biridir. En az maliyetle en kısa yolu bulmaya çalışır.

Python
Veri bilimi ve çeşitli işler için kullanılan programlama dili. Dijkstra algoritmasının hazırlanmasında Python3.9 ile birlikte şu modüller de kullanılmıştır:

- Pandas
- Numpy
- Matplotlib
- Os
- Threading
- Random
- Hashlib
- Subprocess
- Timeit

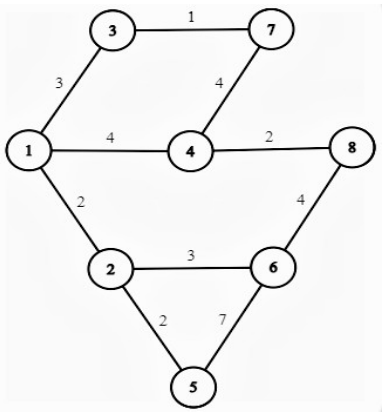
Comnetpr Modülü
Comnetpr graf oluşturmak ve içerisinde kayıtlı olan yol bulma algoritmalarını oluşturulan graflar üzerinde denemek için hazırlanmış özel bir modüldür.

Graph ve Algorithms olmak üzere iki alt sınıfı vardır.

Graph
Bu sınıf ile graflar oluşturulmaktadır. CSV üzerinden okuma yapılarak ya da elle girilerek graflar hazırlanabilir. Hazırlanan grafların vertex ve edge tabloları (matrix) dataframe'e dönüştürülebilmektedir.

Algorithms
Dijkstra algoritmasının bulunduğu sınıf. İçerisine çeşitli yol bulma algoritmaları sınıf olarak tanımlanarak kullanılabilir.

Üzerinde Çalışılacak Graf



Dijkstra'nın Çalışması

Girdiler

Algoritmanın çalışabilmesi için list olarak vertex ve edge matrislerine sahip olması gerekmektedir. Graph modülünden gelen dataframe halindeki vertex-edge verileri list'e dönüştürülerek kullanılabilir.

Kabaca Mantık

1. Ziyaret edilmeyen düğümlerin her birinden en az mesafeye sahip düğümü bul ve git.
2. Mevcut mesafesi toplamından ve aralarındaki kenarın maliyetinden daha büyük olana kadar ziyaret edilen düğümün her bir komşu düğümü için mesafeyi güncelle.
3. Tüm düğümler ziyaret edilene kadar 1 ve 2 numaralı adımları tekrar et.

Kodlar

```
def __init__(self):
    self.vad = [[0 for i in range(2)]] # [0,0]
```

self.vad değişkeni içerisinde iki adet değer tutan bir listeyi barındırır. Bu listedeki birinci değer ziyaret edilmiş düğümleri, ikinci değer ise kaynaktan itibaren olan maliyeti tutar.

```
def next(self):
    self.m = -32 # minimum
    for i in range(self.vexlen): # düğüm matrisinin boyutu kadar
        # kısa yola sahip düğümü seç
        # vad içinden i.listenin ziyaret edilmiş düğümü 0 ise ve m 0 dan küçükse
        # veya vad içinden i.listenin maliyeti vad içinden m.listenin maliyetinden
        # küçük veya eşit ise
        # m'yi i'ye eşitle
        # m'yi döndür
        if (self.vad[i][0] == 0) and (self.m < 0 or self.vad[i][1] <= self.vad[self.m][1]):
            self.m = i
    return self.m
```

next fonksiyonu ziyaret edilmeyen düğümleri bulmakta kullanılır.

Nxtt listesinin içeriği.

```
[0, 1, 2, 6, 4, 3, 5, 7]
```

```
def findWay(self):
        vexs, nxtt, ng, nds, self.dEdges = [], [], [], [], []
        i = 0
        while i < self.vexlen-1: # Vertex sayısı kadar sonsuz eleman ekler
            self.vad.append([0, 10**10])
            i = i + 1
        for vertex in range(self.vexlen):
            nxt = self.next() # Sıradaki düğümü bul
            nxtt.append(nxt) # sıradaki düğümü nxtt listesine ekle
            vexs.append(vertex) # düğümü vexs listsine ekle
            for nghi in range(self.vexlen): # vertex uzunluğu kadar döngü
                # ziyaret edilmeyen düğümler için mesafenin hesaplanması
                if (self.vad[nghi][0] == 0) and (self.vextable[nxt][nghi] == 1):
                    # eğer mevcut mesafe komşunun mesafesi ise güncelle
                    ng.append(nghi) # komşunun ng listesine eklenmesi
                    nt = self.vad[nxt][1] + self.edgtable[nxt][nghi]
                    nds.append(nt) # yeni değer nds listesine eklenir
                    #print(nxt,ndist)
                    # yeni hesaplanan değer büyükse
                    if self.vad[nghi][1] > nt:
                        self.vad[nghi][1] = nt # maliyeti güncelle
                        self.dEdges.append([nxt,nghi,nt]) # dijkstra kenarları
            self.vad[nxt][0] = 1 # düğüm ziyaret edildi
        self.nxtt = nxtt
        self.ng = ng
        self.nds = nds
        self.vexs = vexs
        return self.vad
        self.clearData() # Yolar bulunduktan sonra kullanılan değişkenleri temizler
```

findWays fonksiyonu grafta aramayı yapar. Bulduğu kenarları vad içerisine yerleştirir ve döndürür.

```
   s   d   c   s   d   c
0  0.0  1.0  2.0  NaN  NaN  NaN
1  0.0  1.0  2.0  NaN  NaN  NaN
2  0.0  1.0  2.0  1.0  4.0  4.0
3  0.0  1.0  2.0  1.0  5.0  5.0
4  0.0  2.0  3.0  NaN  NaN  NaN
5  0.0  2.0  3.0  2.0  6.0  4.0
6  0.0  3.0  4.0  NaN  NaN  NaN
7  0.0  3.0  4.0  3.0  7.0  6.0
```

Grafikler

Veriler üzerinde yorum yapmayı kolaylaştırmak için çeşitli grafikler ile düğüm-kenar ilişkileri gösterilmiştir.

Scatter

Nokta grafiği ile düğümler arası maliyet yorumlanabilmektedir. Kaynak ve hedef arası olan maliyet noktaların rengiyle ifade edilmektedir. Noktalar mordan sarıya doğru giden bir skalada renk alır, sarıya ne kadar yakın o kadar maliyetli.

Histogram

Bu grafikte ise düğümler üzerindeki bağlantı sayıları görüntülenebilmektedir. Y ekseni bağlantı sayısı, X ekseni ise düğümün numarasını ifade eder.

Line

Çizgi grafiğinde başlangıç noktasından (1 numaralı düğüm) diğer düğümlere olan maliyetler gösterilmektedir. Y ekseni maliyet, X ekseni düğüm numarasını ifade eder.

3DScatter

Yine düğümler arası maliyetin yorumlanmasında kullanılabilecek bir grafik.