Kőrösi Gábor

Algoritmusok és adatszerkezetek a gyakorlatban

Jelen tananyag a Szegedi Tudományegyetemen készült az Európai Unió támogatásával.

Projekt azonosító: EFOP-3.4.3-16-2016-00014

Szegedi Tudományegyetem
Cím: 6720 Szeged, Dugonics tér 13.
www.u-szeged.hu
www.szechenyi2020.hu

Műveletek a gráfokkal

Összefoglaló

A gráfok alkalmazása nem áll meg a szélességi és mélységi bejárásnál, hiszen ahogyan azt már az előző tananyagban is említettük, ezzel az adatszerkezettel lehetőségünk nyílik arra, hogy útvonalakat keressünk az egyes elemek között. Ebben a fejezetben az ilyen „útkereső” algoritmusokkal fogunk megismerkedni.

Lecke fejezetei:

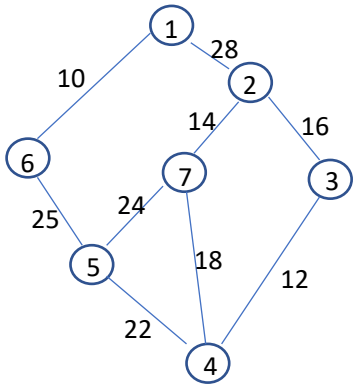
- Prím-algoritmus – Olvasó (15 perc)
- Kruskal-algoritmus – Olvasó (15 perc)
- Dijkstra-algoritmus – Olvasó (15 perc)
- Gyakorló feladatok – Gyakorlati (55 perc)

Téma típusa: Gyakorlati
Olvasási és gyakorlási idő: 90 perc

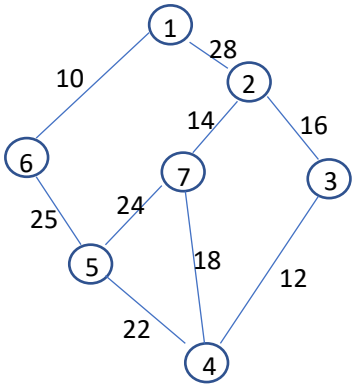
Prím-algoritmus *(15 perc)*

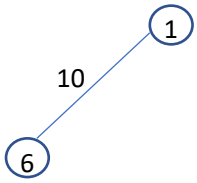
A Prím-algoritmus egy összefüggő súlyozott gráf minimális feszítőfáját határozza meg mohó stratégia segítségével. Működési elve, hogy csúcsonként haladva építi fel a fát egy tetszőleges csúcsból kiindulva, és minden egyes lépésben a lehető legolcsóbb élt keresi meg egy következő csúcshoz.

Az algoritmus lépései a következők: egyetlen fát növesztünk tetszőleges gyökérpontból indulva, majd minden lépésben új csúcsot kötünk be a fába, ahol legolcsóbb éllel elérhető csúcsot választjuk.

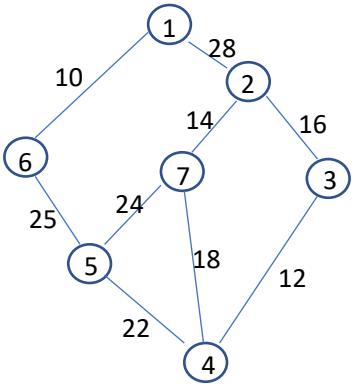


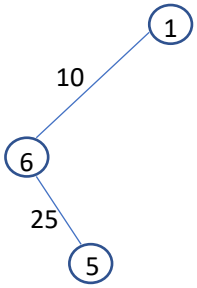
Induljunk el az 1 elemtől! Keressük meg a legkisebb súlyú útvonalat!



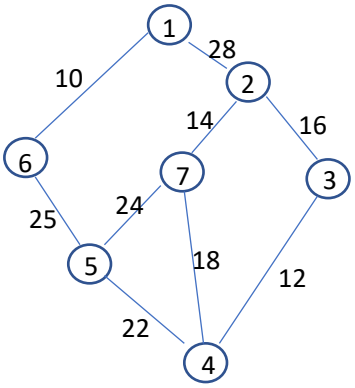


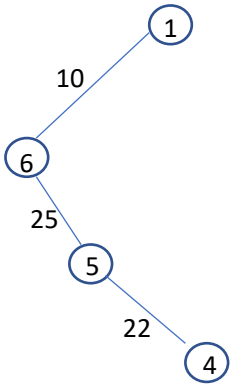
Keressük meg a következő legkisebb súlyú útvonalat, amely kapcsolódik a jelenlegi (új) gráfunkhoz!



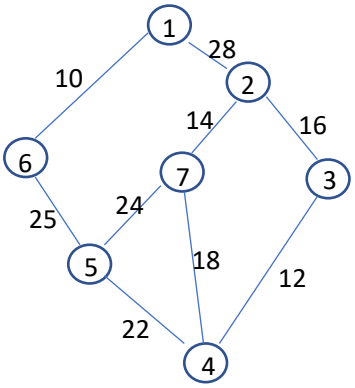


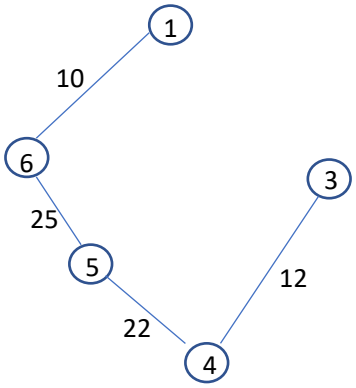
Keressük meg a következő legkisebb súlyú útvonalat, amely kapcsolódik a jelenlegi (új) gráfunkhoz!



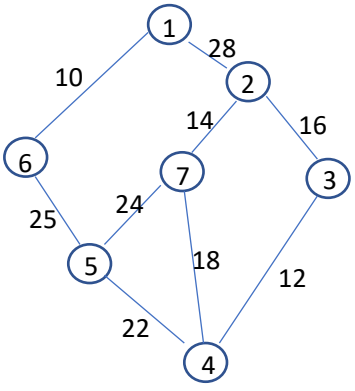


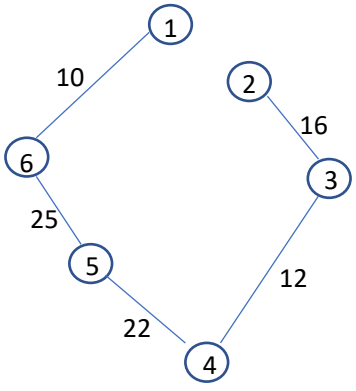
Keressük meg a következő legkisebb súlyú útvonalat, amely kapcsolódik a jelenlegi (új) gráfunkhoz!



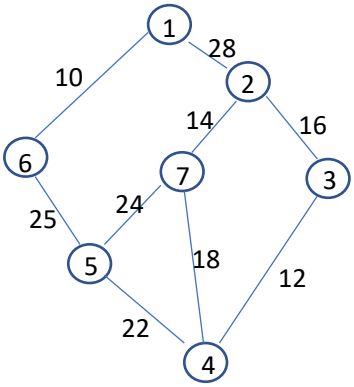


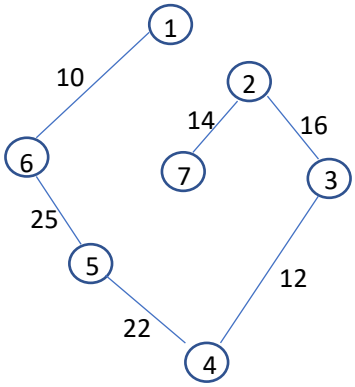
Keressük meg a következő legkisebb súlyú útvonalat, amely kapcsolódik a jelenlegi (új) gráfunkhoz!





Keressük meg a következő legkisebb súlyú útvonalat, amely kapcsolódik a jelenlegi (új) gráfunkhoz!





A megoldás 99.

```
import sys

class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    # nyomtassuk ki az eredmenyt
    def printMST(self, parent):
        print("El \tSuly")
        for i in range(1, self.V):
            print(parent[i], "-", i, "\t", self.graph[i][ parent[i] ])

    # keressuk meg a legrovidebb utat, amely meg nem szerepel a legrovidebb faban
    def minKey(self, key, mstSet):
        # adjunk kezdoerteket a min valtozonak
        min = sys.maxsize
        for v in range(self.V):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v
        return min_index

    def primMST(self):

        # keressuk meg a legkissebb kulcsot
        key = [sys.maxsize] * self.V
        parent = [None] * self.V
        # nezzuk meg az elso vertexet
        key[0] = 0
        mstSet = [False] * self.V
        parent[0] = -1 # az elso vertex a ROOT

        for cout in range(self.V):
            # keressuk meg a legkisebb sulyu vertexet
            # jelezzuk, hogy ez mar hasznalva van
            u = self.minKey(key, mstSet)
            # tegyuk be a vertexet a legrovidebb fa utvonalaba
            mstSet[u] = True
            # frissitsuk a koltseg valtozonkat
            for v in range(self.V):
                if self.graph[u][v] > 0 and mstSet[v] == False and key[v] > self.graph[u][v]:
                        key[v] = self.graph[u][v]
                        parent[v] = u
        self.printMST(parent)
```

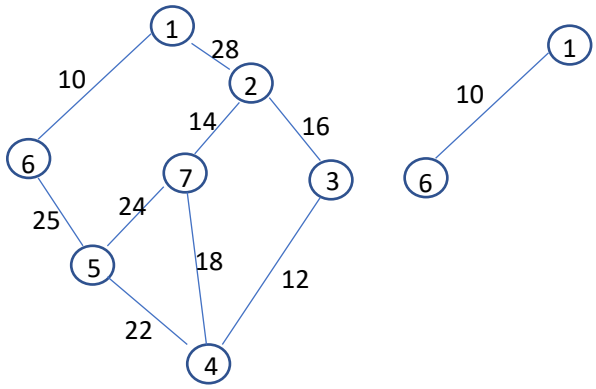
```
g = Graph(7)
g.graph = [ [ 0,28,  0,  0,  0,10,  0],
            [28,  0,16,  0,  0,  0,14],
            [ 0,16,  0,12,  0,  0,  0],
            [ 0,  0,12,  0,22,  0,18],
            [ 0,  0,  0,22,  0,25,24],
            [10,  0,  0,  0,25,  0,  0],
            [ 0,14,18,  0,24,  0,  0]]
g.primMST()
```

Kruskal-algoritmus (15 perc)

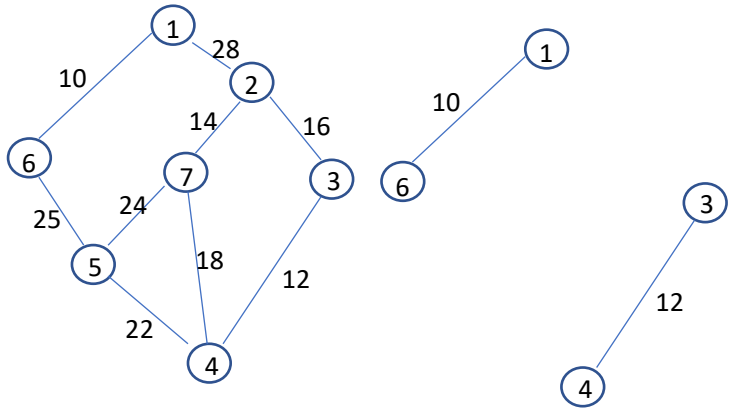
A Kruskal-algoritmus Joseph Kruskal (1928–2010) amerikai matematikustól és informatikustól származik, aki 1956-ban írta ezt le. Ez egy súlyozott gráfokat feldolgozó mohó algoritmus, melynek célja egy minimális feszítőfa megalkotása. Ha a gráf nem összefüggő, akkor pedig egy minimális feszítő erdőt hoz létre.

A Kruskal-algoritmus működése közben minimális fák erdejét tároljuk, ahol kezdetben minden pont egy külön fa, és ezt úgy változtatjuk, hogy minden lépésben a legkisebb, két fát összekötő élt húzzuk be (egyesítjük egyetlen fává a két fát)

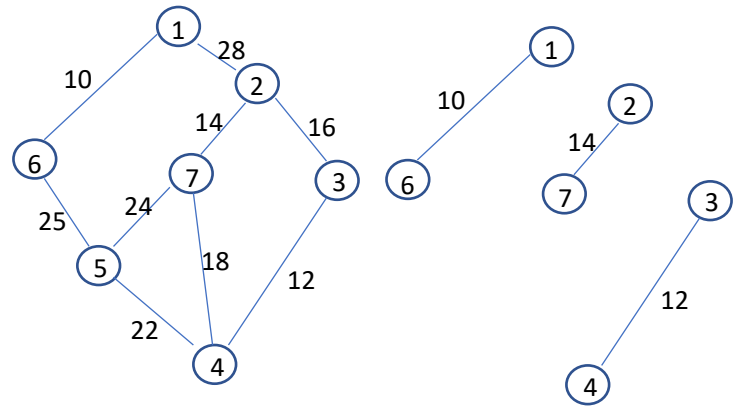
Kezdjük az első lépéssel, és keressük meg a legkisebb élet!



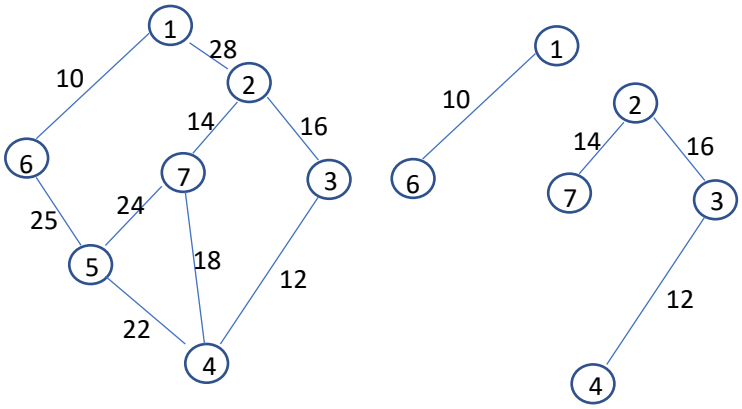
Keressük meg a következő legkisebb élet!



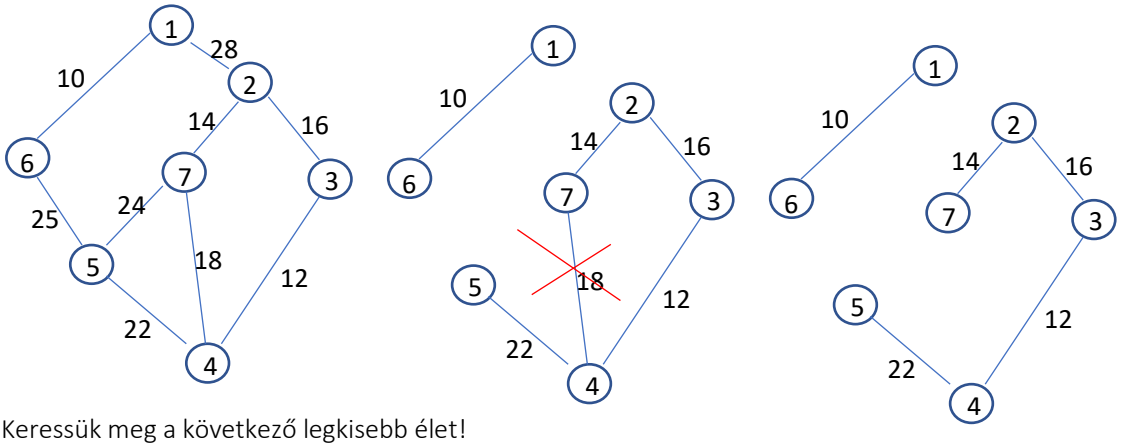
Keressük meg a következő legkisebb élet!



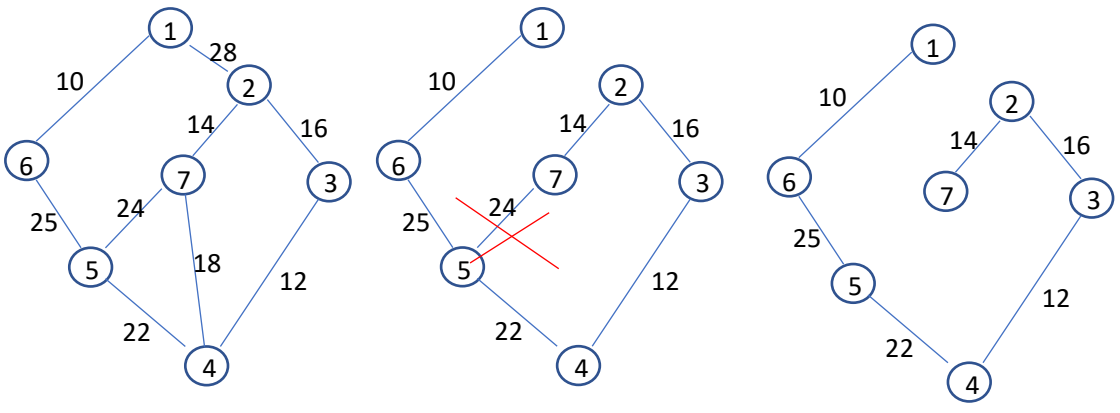
Keressük meg a következő legkisebb élet!



Keressük meg a következő legkisebb élet! Ez a 18-as lenne, viszont a Kruskal-algoritmus kimondja, hogy nem keletkezhet kör, ezért ez a megoldás nem jó.



Keressük meg a következő legkisebb élet!



```
from collections import defaultdict
class Graph:
    def __init__(self,vertices):
        self.V= vertices
        self.graph = []

    def addEdge(self,u,v,w):
        self.graph.append([u,v,w])

    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)

        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else :
            parent[yroot] = xroot
            rank[xroot] += 1

    def KruskalMST(self):

        result =[] #az eredmenyhalmaz
        i = 0 # index az eltarolt elekhez
        e = 0 # index a result vektorhoz

        # 1. lepes: rakjuk sorrendbe az eleket a sulyuk szerint
        self.graph =  sorted(self.graph,key=lambda item: item[2])
        parent = [] ; rank = []
        # Keszitsunk egy reszhalmazt egy elemmel
        for node in range(self.V):
            parent.append(node)
            rank.append(0)

        # addig ismeteljuk a lepeseket, amig V-1 elunk lesz a megoldashalmazban.
        while e < self.V -1 :

            # 2. lepes: Vegyuk ki a legkisebb elet, es leptessuk az indexet
            u,v,w =  self.graph[i]
            i = i + 1
            x = self.find(parent, u)
            y = self.find(parent ,v)

            # ha az uj el nem okoz kort, akkor adjuk a megoldashalmazhoz, es lepjunk tovabb
            if x != y:
                e = e + 1
                result.append([u,v,w])
                self.union(parent, rank, x, y)
            # kulonben ugorjuk at az elet

        # irjuk ki a kapott eredmenyt
        print (' a kovetkezo elek kepezik a megoldas halmazt' )
        for u,v,weight  in result:
            print ("%d -- %d == %d" % (u,v,weight))

g = Graph(4)
```

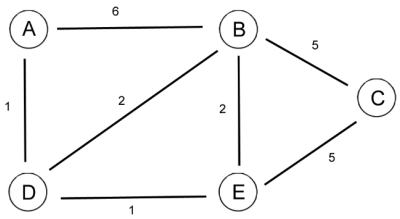
```
g.addEdge(0, 1, 10)
g.addEdge(0, 2, 6)
g.addEdge(0, 3, 5)
g.addEdge(1, 3, 15)
g.addEdge(2, 3, 4)

g.KruskalMST()

#This code is contributed by Neelam Yadav
```

Dijkstra-algoritmus (15 perc)

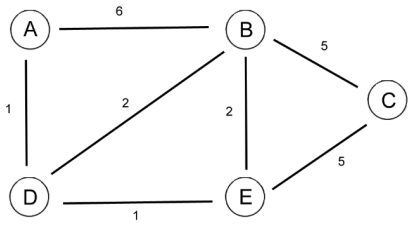
A Dijkstra-algoritmus egy mohó algoritmus, amivel irányított vagy irányítatlan gráfokban lehet megkeresni a legrövidebb utakat egy adott csúcspontból kiindulva. Az algoritmust Edsger Wybe Dijkstra holland informatikus fejlesztette ki.



VERTEX	Legr. A-ból	Előző elem.
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Az algoritmus működése z alábbi lépésekből áll:

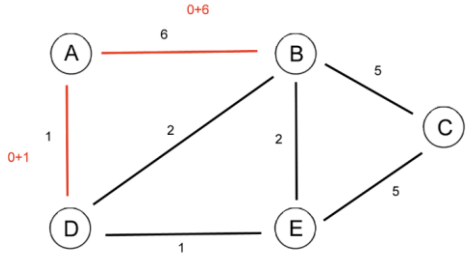
- Két tömböt rendelünk a gráfunkhoz.
- Megnézve = [] NemMegnézve = [A,B,C,D,E]
- Megnézzük, hogy az A-ból A-ba mekkora a távolság, a többit pedig inf.-re állítjuk



VERTEX	Legr. A-ból	Előző elem.
A	0	
B	inf	
C	inf	
D	inf	
E	inf	

Most megnézzük, hogy az A még meg nem látogatott szomszédjaihoz, hogyan tudunk eljutni. Amennyiben a kalkulált érték kisebb, mint az eddig ismert érték, úgy felülírjuk azt.

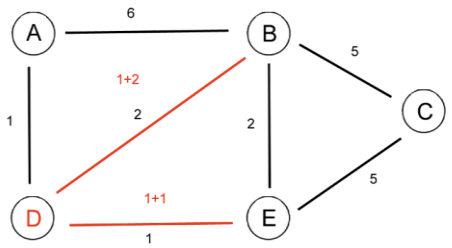
Megnézve = [A] NemMegnézve = [B,C,D,E]



VERTEX	Legr. A-ból	Előző elem.
A	0	
B	6	A
C	inf	
D	1	A
E	inf	

Következő lépésben kiválasztjuk azt a legkisebb VERTEXET a listából, amelyet még nem látogattunk meg. Esetünkben ez a D. Most a D szomszédjait kell megvizsgálnunk.

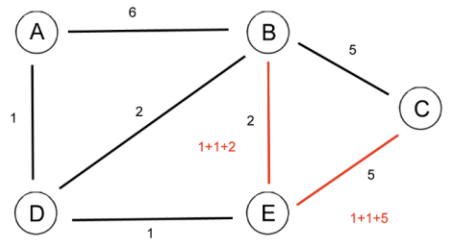
Megnézve = [A] NemMegnézve = [B,C,D,E]



VERTEX	Legr. A-ból	Előző elem.
A	0	
B	3	D
C	inf	
D	1	A
E	2	D

Következő lépésben kiválasztjuk azt a legkisebb VERTEXET a listából, amelyet még nem látogattunk meg. Esetünkben ez az E. Most az E szomszédjait kell megvizsgálnunk.

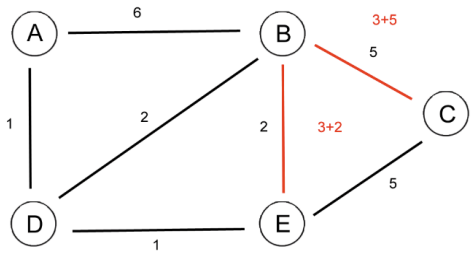
Megnézve = [A,D] NemMegnézve = [B,C,E]



VERTEX	Legr. A-ból	Előző elem.
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Következő lépésben kiválasztjuk azt a legkisebb VERTEXET a listából, amelyet még nem látogattunk meg. Esetünkben ez a B. Most a B szomszédjait kell megvizsgálnunk.

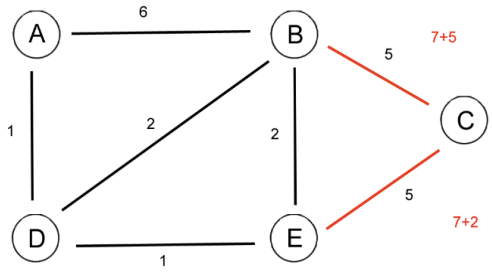
Megnézve = [A,D,E] NemMegnézve = [B,C]



VERTEX	Legr. A-ból	Előző elem.
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

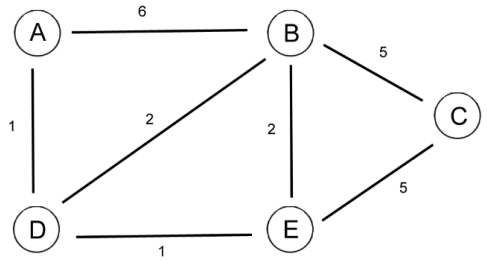
Következő lépésben kiválasztjuk azt a legkisebb VERTEXET a listából, amelyet még nem látogattunk meg. Esetünkben ez a C. Most a C szomszédjait kell megvizsgálnunk.

Megnézve = [A,D,E,B] NemMegnézve = [C]



VERTEX	Legr. A-ból	Előző elem.
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Megkaptuk a végeredményt.



VERTEX	Legr. A-ból	Előző elem.
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

```
import sys

class Graph():
        def __init__(self, vertices):
                self.V = vertices
                self.graph = [[0 for column in range(vertices)]
                                        for row in range(vertices)]
        def printSolution(self, dist):
                print ("Vertex \t tavolsag a forrastol")
                for node in range(self.V):
                        print (node, "\t", dist[node])

        # Keressuk meg a legrovidebb tavolsagot, ami meg nincs a megoldashalmazban
        def minDistance(self, dist, sptSet):

                # kezdoertek beallitasa
                min = sys.maxsize

                # legkissebb el keresese
                for v in range(self.V):
                        if dist[v] < min and sptSet[v] == False:
                                min = dist[v]
                                min_index = v

                return min_index

        def dijkstra(self, src):

                dist = [sys.maxsize] * self.V
                dist[src] = 0
                sptSet = [False] * self.V

                for cout in range(self.V):

                        # keresuk meg a legkisebb utat
                        u = self.minDistance(dist, sptSet)
                        # tegyuk be a megoldashalmazba az uj elet
                        sptSet[u] = True

                        # frissitsuk az ertekeket
                        for v in range(self.V):
                                if self.graph[u][v] > 0 and sptSet[v] == False and
                                dist[v] > dist[u] + self.graph[u][v]:
                                        dist[v] = dist[u] + self.graph[u][v]

                self.printSolution(dist)

g = Graph(9)
g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
                [4, 0, 8, 0, 0, 0, 0, 11, 0],
                [0, 8, 0, 7, 0, 4, 0, 0, 2],
```

```
                    [0, 0, 7, 0, 9, 14, 0, 0, 0],
                    [0, 0, 0, 9, 0, 10, 0, 0, 0],
                    [0, 0, 4, 14, 10, 0, 2, 0, 0],
                    [0, 0, 0, 0, 0, 2, 0, 1, 6],
                    [8, 11, 0, 0, 0, 0, 1, 0, 7],
                    [0, 0, 2, 0, 0, 0, 6, 7, 0]
                    ];

g.dijkstra(0);

# This code is contributed by Divyanshu Mehta
```

Gyakorló feladatok (45 perc)

1. Szemléltessük a fenti programokat lépésről, lépésre az *igraph* csomag segítségével!

Ajánlott kitekintő anyag (95 perc)

Gráfalgoritmusok **(5 perc)** Gelle Kitti - link

Prim-algoritmus **(2 perc)** – Youtube link angol nyelven

Prim and Kruskals algoritmus **(20 perc)** - Youtube link angol nyelven

Dijsktra-algoritmus **(18 perc)** – Youtube link angol nyelven

Minimális feszítőfák **(50 perc)** – MIT video link angol nyelven

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE