ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

ALGORITMUSOK ÉS ADATSZERKEZETEK

Python 9-10.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FÉLÉVES TEMATIKA

Oktatási hét	Témakör - Gyakorlat
1	Bevezetés a Python programozási nyelvbe, alapvető szintaxis és kifejezések, változók, típusok és alapvető műveletek, alapvető be- és kimenet, feltételes elágazások (if, elif, else)
2	Ciklusok (for, while), listák és egyszerű iterációk
3	Függvények és modulok, függvények definiálása és hívása, paraméterek és visszatérési értékek, beépített modulok használata, fájlkezelés, kivételkezelés
4	Algoritmusok és optimalizálási stratégiák rekurziós példán keresztül (brute force, dinamikus programozás, mohó algoritmusok, visszalépéses keresés)
5	Adatszerkezetek I. - Lista, tömb, sor, verem, listaműveletek, queue és stack implementációja statikusan (tömb) és dinamikusan (lista), gyakorlati feladatok
6	Adatszerkezetek II. – Halmaz és szótár, halmazműveletek és szótárműveletek, gyakorlati feladatok
7	**Zárthelyi dolgozat 1.**
8	Adatszerkezetek III. – Láncolt lista implementálása
9	Adatszerkezetek IV. – Bináris keresőfa implementálása
10	Gráfok alapjai, reprezentációik, gráfalgoritmusok implementálása
11	*Rektori szünet*
12	***Féléves beadandó feladat leadása és bemutatása***
13	**Zárthelyi dolgozat 2.**
14	**Félév zárása, javító/pótló zárthelyi dolgozat**

2 fős csapatban - videó
Max. 5 perc
2025.11.25.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

BEADANDÓ FELADATOK I.

- **1. Nyolckirálynő probléma** (*visszalépéses keresés*)
  - Helyezd el a 8 királynőt a sakktáblán úgy, hogy egyik se támadhassa a másikat. A megoldást rekurzív visszalépéses kereséssel valósítsd meg.
- **2. Keresztrejtvény** (*visszalépéses keresés*)
  - Adott szavakat helyezz el egy NxN-es rácsban úgy, hogy vízszintesen és függőlegesen értelmes keresztrejtvény jöjjön létre. A szavak elhelyezése visszalépéses kereséssel történjen.
- **3. Sudoku** (*visszalépéses keresés*)
  - Oldj meg egy 9×9-es Sudoku rejtvényt visszalépéses keresés segítségével. A bemenet lehet előre kitöltött részleges tábla.
- **4. Utazó ügynök probléma** (*mohó algoritmus*)
  - Készíts programot, amely a városok közötti távolságokat ismerve mohó módon választja mindig a legközelebbi következő várost. Számítsd ki a bejárt útvonal teljes hosszát.
- **5. Kruskal-algoritmus** (*mohó algoritmus*)
  - Valósítsd meg a Kruskal-algoritmust, amely egy gráf minimális feszítőfáját állítja elő az élek súlyai alapján.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

BEADANDÓ FELADATOK II.

- **6. Dijkstra algoritmus** (*mohó algoritmus*)
  - Készíts programot, amely egy súlyozott, irányított gráfban meghatározza a kezdőpontból a legrövidebb utat minden más csúcshoz Dijkstra algoritmusával.
- **7. Huffman kódolás** (*mohó algoritmus*)
  - Készíts programot, amely egy szöveg karaktereinek előfordulási gyakorisága alapján létrehozza a Huffman-fát és az optimális bináris kódokat.
- **8. Hanoi tornyai** (*rekurzió*)
  - Mozgasd át a korongokat az egyik toronyról a másikra a játékszabályok betartásával. A megoldást rekurzívan valósítsd meg.
- **9. Összefésülő rendezés** (*rekurzió*)
  - Valósítsd meg az összefésülő (merge sort) rendezést rekurzív módon, majd teszteld a működését különböző méretű listákon.
- **10. Gyorsrendezés** (*rekurzió*)
  - Készíts programot, amely a gyorsrendezés (quick sort) algoritmussal rendezi a bemeneti listát rekurzívan.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

BEADANDÓ FELADATOK

- **11. Láncolt lista kapcsolódó műveletek megvalósítása (min. 5 szabadon választott)** *(láncolt lista)*
  - Valósíts meg legalább 5 alapműveletet (ami nem volt az órán) egy egyszerű láncolt lista adatszerkezeten.
- **12. Bináris fa "aktuális állapotának" szemléletes megjelenítése képernyőn** *(bináris fa)*
  - Készíts olyan programot, amely egy bináris fa szerkezetét építi fel és a fa aktuális állapotát szövegesen vagy grafikus formában megjeleníti a képernyőn.
- **13. Gráf műveletek megvalósítása szomszédsági mátrix segítségével** (*gráf*)
  - Készíts programot, amely egy gráfot tárol szomszédsági mátrix formában, és megvalósítja az alapműveleteket (csúcs/él hozzáadása, törlése, bejárások).
- **14. Labirintus megoldása** (*visszalépéses keresés*)
  - Adott egy NxN-es mátrix (0: szabad, 1: fal). Keresd meg az utat a kezdőpontból a célpontba visszalépéses kereséssel.
- **15. Kifejezés kiértékelése bináris fával** *(bináris fa)*
  - Írj programot, amely egy matematikai kifejezést (pl. (3 + 5) * 2) bináris fába épít, majd a fa bejárásával kiszámítja az eredményt. A program legyen képes több szintű zárójelezés és műveleti prioritás kezelésére is.

Bináris fa és keresőfa implementálása

BINÁRIS FA

A **bináris fa** (*binary tree*) egy **hierarchikus adatszerkezet**, amelyben **minden csomópontnak legfeljebb két gyermeke** (*gyermekcsomópont*ja) van, ezeket röviden *balgyerek*nek és *jobbgyerek*nek is nevezhetjük.

Másképpen azt is mondhatjuk, hogy minden csomóponthoz (csúcsponthoz) legfeljebb két *részfa* csatlakozik.

KAPCSOLÓDÓ FOGALMAK

Élek: Az élek a csomópontok közötti kapcsolatok. Egy él köti össze a **szülőcsomópont**ot egy **gyermek csomópont**tal.

Gyökér: A gyökér a bináris fa legfelső csomópontja. Ez az egyetlen csomópont, amelynek nincs szülője. A teljes fa a gyökértől kezdve érhető el.

Levelek: A levelek vagy **levélcsomópont**ok olyan csomópontok, amelyeknek nincs gyermeke. Más néven ezek a fa terminális csomópontjai.

Ős és leszármazott: Ha a C csomópontból vezet út a K-ba, akkor C a K őse, K a C leszármazottja.

Részfa: Egy C csomópont az összes leszármazottjával együtt részfát alkot, amelynek gyökere C.

KAPCSOLÓDÓ FOGALMAK (FOLYT.)

Csomópont mélysége, szintje: a gyökértől a hozzá elvezető út hossza (az útban lévő élek száma).

Fa magassága: a fa szintjeinek száma. A gyökértől a levelekig vezető utak közül a leghosszabb út (utak) csomópontjainak száma (élek száma+1).

(Erre más definíció is elképzelhető: a leghosszabb út éleinek a száma.)

BINÁRIS FÁK ELŐNYEI

Előnyök:

- **hatékony keresés** (a bal részfában lévő elemek kisebbek, a jobb részfában lévő elemek nagyobbak, mint a szülő csomópont kulcsa – kiegyensúlyozott fa esetén)
- **rendezett bejárások** (rekurzív struktúra - preorder, inorder, postorder bejárás)
- **rugalmasság**: viszonylag gyors elemfelvétel és elemtörlés – dinamikus memóriahasználat
- **többféleképp ábrázolható**: például láncolt listákkal vagy tömbökkel, a csomópontok tartalma alkalmas lehet más adatszerkezetek indexelésére

BINÁRIS FÁK ALKALMAZÁSAI

Példák alkalmazásokra:

- **bináris keresési** algoritmusok: hatékony keresés.
- **rendezési algoritmusok**: pl. farendezés és kupacrendezés.
- **adatbázisrendszerek**: minden csomópont egy rekordot képvisel. Hatékony keresési műveleteket és nagy mennyiségű adat kezelését teszi lehetővé.
- **fájlrendszerek**: minden csomópont egy könyvtárat vagy fájlt képvisel.
- **tömörítési algoritmusok**: pl. Huffman-kódolás megvalósítása.
- **játék AI**: minden csomópont egy lehetséges lépést jelent a játékban. Az AI-algoritmus képes keresni a fát, hogy megtalálja a lehető legjobb lépést.

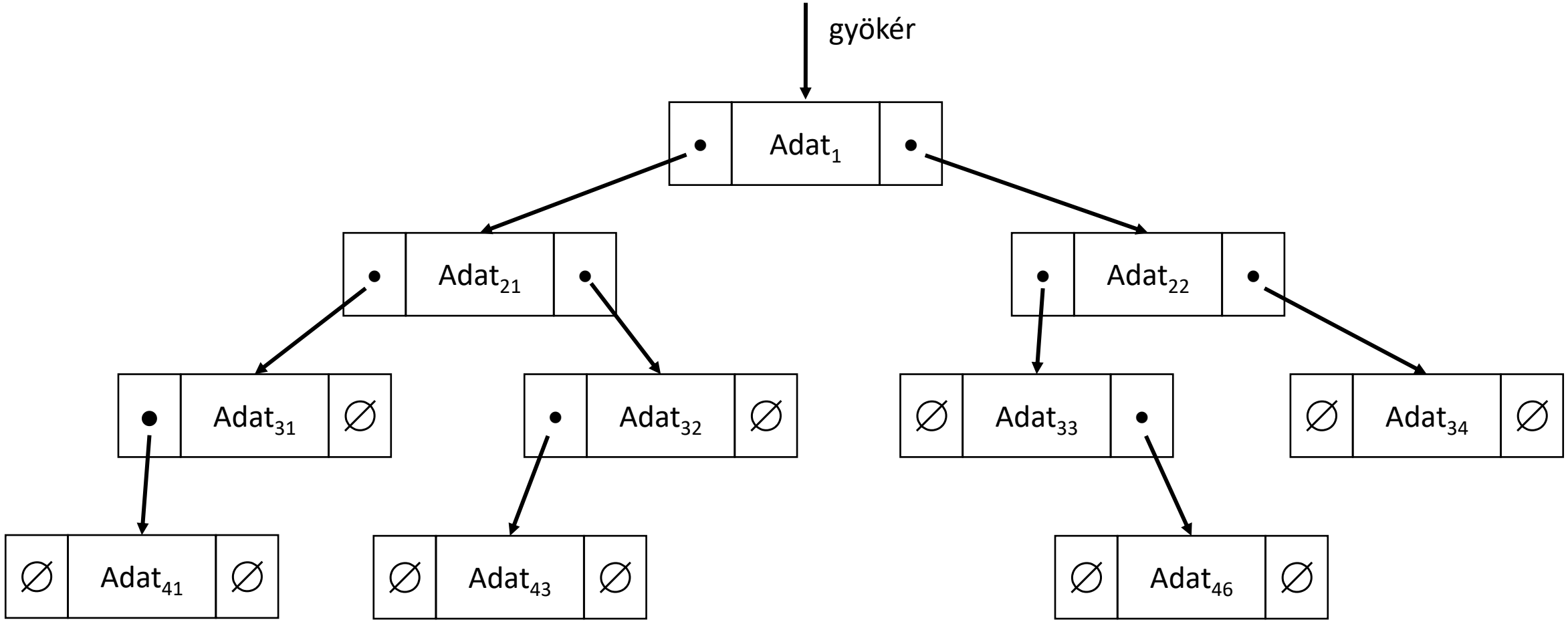
BINÁRIS FA LÁNCOLT ÁBRÁZOLÁSSAL

Matematikai szempontból egy bináris fa megfelel egy *irányított fagráf*nak, amelyben egy kitüntetett csúcsból (a gyökérből) pontosan egy út vezet minden más csúcshoz.

- A bináris fa felfogható **rekurzív adatszerkezetként**, és láncoltan ábrázolva **minden egyes csomópont tartalmaz egy adatrészt, valamint mutatókat (hivatkozásokat) a bal és jobb oldali gyermekeire (ha vannak), amelyek ugyanilyen típusú csomópontok.**

A hatékonyság növelése érdekében sok esetben a szülőcsomópontra vonatkozó hivatkozást (mutatót) is tárolják a gyerekcsomópontokban.

BINÁRIS FA LÁNCOLT ÁBRÁZOLÁSSAL (FOLYT.)



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NODE (CSOMÓPONT) OSZTÁLY

```
1  class Node:                          # Csomópont osztály
2      def __init__(self, data):        # konstruktor
3          self.data=data               # a csomóponthoz tartozó adat
4          self.left=None               # balgyerek üres, "nem mutat sehova"
5          self.right=None              # jobbgyerek üres, "nem mutat sehova"
6                                       # néha szoktak szülő mutatót is alkalmazni
7
8      def __str__(self):
9          return str(self.data)        # az adatrész sztringreprezenációját adja vissza
```

A hívásoknál a *self* paramétert nem kell szerepeltetni!

A *data* adattagban bármilyen típusú értéket tárolhatunk, és az át is adható az *str* függvénynek.

A bináris fa implementálásánál most nem foglalkozunk hibakezeléssel. Ha egy művelet értelmetlen, akkor egyszerűen vagy hatástalan lesz, vagy *None* értéket fog visszaadni.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

BINARYTREE (BINÁRIS FA) OSZTÁLY

```
11  class BinaryTree:                   # BinFa, azaz bináris fa
12      def __init__(self,data=None): # konstruktor
13          if data==None:              # üres fa
14              self.root=None          # a gyökérmutató (root) "nem mutat sehova"
15          else:
16              new_node=Node(data)     # a gyökércsomópont létrehozása (egyelemű fa)
17              self.root=new_node      # a gyökérmutató a gyökércsomópontra mutat
18          self.act=self.root          # kezdetben az aktuális (act) mutató a gyökérre mutat
```

Ha nem adunk meg paramétert a konstruktornak, akkor üres fa jön létre; míg, ha megadunk egy tárolandó adatot, akkor egy egyelemű fa (csak a gyökércsomópontból áll).

Az egyszerűség kedvéért használni fogunk egy aktuális (*act*) mutatót, amely a bináris fa valamely csomópontjára mutat. Kezdetben ugyanazt a címet tartalmazza, mint a gyökérmutató (*root*).

BINARYTREE OSZTÁLY

isEmpty, Top és Next függvények

```
20      def isEmpty(self):              # Üres_e művelet
21          return self.root==None      # ha a gyökér "nem mutat sehova", akkor a fa üres
22
23      def Top(self):                  # Elejére művelet
24          self.act=self.root          # az act mutató a gyökérre mutasson
25
26      def Next(self,direction): # Következőre műv.: iránytól függően léptetjük az act mutatót
27          if self.act!=None:      # ha az act nem "sehova", vagyis létező csomópontra mutat,
28              if direction=="balra":      # és ha az irány balra, akkor
29                  self.act=self.act.left  # lépjünk a balgyerekre;
30              if direction=="jobbra":     # és ha az irány jobbra, akkor
31                  self.act=self.act.right # lépjünk a jobbgyerekre
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT1 – PYTHON_07_01.PY

Nyisd meg és egészítsd ki a Python_07_01.py kódot a *#főprogram* megjegyzés után:

- hozzunk létre egy üres bináris fát uresfa néven
- hozzunk létre egy egyelemű fát fa néven, a gyökérelem az „alma” szót tartalmazza, majd feltételvizsgálat után írassuk ki, hogy üres vagy nem a fa
- írassuk ki a gyökérelemet, illetve az aktuális elemet
- a gyökérelem jobbgyerekét hozzuk létre, és tartalmazza a „banán” szót
- léptessük jobbra az aktuális mutatót
- írassuk ki most is az aktuális elemet

BINARYTREE OSZTÁLY

Bal-Közép-Jobb (BKJ) bejárás, másképpen inorder bejárás

A **bejárás** azt jelenti, hogy az adatszerkezet összes elemét egyszer elérjük, illetve feldolgozzuk.

A **Bal-Közép-Jobb bejárás (inorder)** szerint rekurzív módon először bejárjuk az aktuális csomópont bal oldali részfáját, majd feldolgozzuk az aktuális csomópontot, végül bejárjuk az aktuális csomópont jobb oldali részfáját. A BKJ-bejárás a **mélységi keresések** közé tartozik.

Alkalmazás:

- bináris keresőfáknál az elemeket rendezés szerinti sorrendben tudjuk elérni.

A megvalósításnál egy listába gyűjtjük ki a csomópontok adatait.

BINARYTREE OSZTÁLY

traversalInOrder és InOrder függvények

```
33  # BKJ-bejárás (bal-közép-jobb bejárás; inorder bejárás)
34  def traversalInOrder(self):         # az aktuális objektum rekurzív InOrder fv.-ét hívja
35      return self.InOrder(self.root) # meg a gyökércsomóponttal, amely listában adja vissza
36                                     # a csomópontokban tárolt adatokat
37  def InOrder(self,node):
38      result=[]                              # a lista üresre állítása
39      if node:                               # ha a csomópont nem üres, azaz van gyereke, akkor
40          result=self.InOrder(node.left) # rekurzívan bejárjuk a bal részfát
41          result.append(node.data)       # az eredménylistához fűzzük a csomópont adatát
42          result+=self.InOrder(node.right) # hozzáfűzzük a rekurzívan bejárt jobb részfát
43      return result                          # visszaadjuk a kész listát a bejárást indító
44                                             # függvénynek
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT2 – PYTHON_07_02.PY

Nyisd meg a Python_07_02.py forráskódot, figyeld meg a csomópontok felvételét a fába!

Futtasd is a programot, majd tanulmányozd a bejárás eredményét!

BINARYTREE OSZTÁLY

Közép-Bal-Jobb (KBJ) bejárás, másképpen preorder bejárás

A **Közép-Bal-Jobb bejárás (preorder)** szerint először feldolgozzuk az aktuális csomópontot, majd rekurzív módon bejárjuk az aktuális csomópont bal oldali részfáját, végül bejárjuk az aktuális csomópont jobb oldali részfáját. A KBJ-bejárás is a **mélységi keresések** közé tartozik.

Alkalmazás:

- kifejezésfa kiértékelése → prefix kifejezés (lengyel-forma)
- fa másolása, mentése

BINARYTREE OSZTÁLY

traversalPreOrder és PreOrder függvények

```
46      # KBJ-bejárás (közép-bal-jobb bejárás; preorder bejárás)
47      def traversalPreOrder(self):
48          return self.PreOrder(self.root)
49
50      def PreOrder(self,node):
51          result=[]
52          if node:
53              result.append(node.data)
54              result+=self.PreOrder(node.left)
55              result+=self.PreOrder(node.right)
56          return result
```

BINARYTREE OSZTÁLY

Bal-Jobb-Közép (BJK) bejárás, másképpen postorder bejárás

A **Bal-Jobb-Közép bejárás (postorder)** szerint először rekurzív módon bejárjuk az aktuális csomópont bal oldali részfáját, majd bejárjuk az aktuális csomópont jobb oldali részfáját, végül feldolgozzuk az aktuális csomópontot. A BJK-bejárás is a mélységi keresések közé tartozik.

Alkalmazás:

- kifejezésfa kiértékelése → postfix kifejezés (fordított lengyel-forma)
- fa törlése (biztosítva van, hogy a gyerekcsomópontok előbb törlődnek, mint a szülők)

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT3 – PYTHON_07_03.PY

Nyisd meg a Python_07_03.py forráskódot!

Futtasd és tanulmányozd a KBJ-bejárás (preorder) eredményét!

Készítsd el a BJK-bejáráshoz (postorder) szükséges két függvényt, majd próbáld is ki!

BINARYTREE OSZTÁLY

további egyszerű műveletek

```
70      def isEnd(self):                # Végén_e művelet
71          return self.act==None
72
73      def rootElement(self):          # Gyökérelem művelet
74          if not self.isEmpty():
75              return self.root.data
76          else:
77              return None
78
79      def actualElement(self):        # Aktelem művelet
80          if self.act!=None:
81              return self.act.data
82          else:
83              return None
84
85      def updateRootElement(self,data): # GyökérMódosít művelet
86          if not self.isEmpty():
87              self.root.data=data
88
89      def updateActualElement(self,data): # AktElemMódosít művelet
90          if self.act!=None:
91              self.act.data=data
```

BINARYTREE OSZTÁLY

insertBinTree függvény

```
93      # binfa aktuális csomópontjához binfa részfát illeszt a megadott irányba
94      def insertBinTree(self,bintree,direction):
95          if self.isEmpty():                  # ha a binfa üres volt, akkor ez lesz a binfa
96              self.root=bintree.root
97              self.act=self.root
98          else:
99              if direction=="balra" and self.act.left==None:  # ha a balgyerek "üres"
100                 self.act.left=bintree.root
101             if direction=="jobbra" and self.act.right==None: # ha a jobbgyerek "üres"
102                 self.act.right=bintree.root
```

BINARYTREE OSZTÁLY

removeBinTree függvény

```
104     # binfa aktuális csomópontja "alól" leválaszt egy binfát
105     def removeBinTree(self,bintree,direction):   # a bintree lesz a leválasztott binfa
106         if bintree.root==None and self.act!=None:
107                 if direction=="balra":
108                     bintree.root=self.act.left
109                     self.act.left=None
110                 if direction=="jobbra":
111                     bintree.root=self.act.right
112                     self.act.right=None
113                 bintree.act=bintree.root
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT4 – PYTHON_07_04.PY

Nyisd meg a Python_07_04.py kódot és készítsd el a *partOfBinTree* (rész) függvényt, amely az aktuális csomópont bal vagy jobb oldali részfáját kimásolja! (Tehát az eredeti bináris fában is megmarad, csak ennyiben különbözik a *removeBinTree* függvénytől.)

Futtasd a programot!

Csak az egyszerűbb kiíratás miatt készült az alábbi függvény:

```
124     def printBinTree(self):
125         for item in self.traversalInOrder():
126             print(item,end=' ')
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT5 – PYTHON_07_05.PY

Nyisd meg a Python_07_05.py kódot és a #Főprogram komment utáni sorokat írd be! Futtasd a programot!

BINÁRIS KERESŐFA

A **bináris keresőfa** (*binary search tree, BST*) egy olyan **speciális bináris fa**, amely rendezett sorrendet tart fenn.

- Egy bináris keresőfában bármely adott csomópont esetében a bal oldali részfa összes csomópontjának értéke kisebb, mint az adott csomópont értéke, a jobb oldali részfa összes csomópontjának értéke pedig nagyobb.
- Ez a tulajdonság lehetővé teszi a hatékony keresési, beszúrási és törlési műveleteket.

BINÁRIS KERESŐFA

Használat:

- útvonalkeresési algoritmusok (pl. videójátékoknál)
- fájltömörítés Huffman-kódolással
- renderelési számítások (pl. a játékok között a Doom volt az első 1993-ban)
- szintaktikai elemzések (alacsony szintű nyelvek fordítóprogramjai)
- adatbázisoknál kulcsok keresése

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

BINÁRIS KERESŐFA IMPLEMENTÁLÁSA

Egy egyszerűsített bináris keresőfát készítünk.

Ez nem kiegyensúlyozott fa lesz, ezért a mélysége naggyá válhat; illetve a fa vagy részei láncolt listává "degenerálódhatnak", amelyben csak lineárisan kereshetünk (nem lesz hatékony a sok összehasonlítás miatt)!

Az egyszerűség kedvéért maga **a tárolt adatelem lesz egyúttal a kulcs** is, vagyis ez alapján történik a rendezés (összehasonlítás).

Feltételezzük, hogy a kulcsok mind különbözők.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NODE (CSOMÓPONT) OSZTÁLY

```
1  class Node:                          # Csomópont osztály
2      def __init__(self, data):        # konstruktor
3          self.data=data               # a csomóponthoz tartozó adat
4          self.left=None               # balgyerek üres, "nem mutat sehova"
5          self.right=None              # jobbgyerek üres, "nem mutat sehova"
6
7      def __str__(self):
8          return str(self.data)        # az adatrész sztringreprezenációját adja vissza
```

A hívásoknál a *self* paramétert nem kell szerepeltetni!

A *data* adattagban bármilyen típusú értéket tárolhatunk, és az át is adható az *str* függvénynek.

Lehetne használni szülőcsomópontra hivatkozó mutatót (*parent*) is.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

BINARYSEARCHTREE (BINÁRIS KERESŐFA) OSZTÁLY

```
11  class BinarySearchTree:          # BinKerFa, azaz bináris keresőfa
12      def __init__(self):          # konstruktor
13          self.root=None           # a gyökérmutató (root) "nem mutat sehova"
```

A (paraméter nélküli) konstruktor üres keresőfát hoz létre. Most nem fogunk használni aktuális csomópontra hivatkozó mutatót.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

BINARYSEARCHTREE OSZTÁLY
traversalInOrder és InOrder függvények

A csomópontokban tárolt adatelemek rendezett kiíratása:

```
15    # BKJ-bejárás (bal-közép-jobb bejárás; inorder bejárás):
16    # ennek hatására a kulcs szerinti rendezettséggel járjuk be a fát, vagyis
17    # rendezetten írathatjuk ki az adatelemeket.
18    def traversalInOrder(self):          # az aktuális objektum rekurzív InOrder fv.-ét hívja
19        return self.InOrder(self.root) # meg a gyökércsomóponttal, amely listában adja vissza
20                                         # a csomópontokban tárolt adatokat
21    def InOrder(self,node):
22        result=[]                              # a lista üresre állítása
23        if node:                               # ha a csomópont nem üres, azaz van gyereke, akkor
24            result=self.InOrder(node.left) # rekurzívan bejárjuk a bal részfát
25            result.append(node.data)       # az eredménylistához fűzzük a csomópont adatát
26            result+=self.InOrder(node.right) # hozzáfűzzük a rekurzívan bejárt jobb részfát
27        return result                          # visszaadjuk a kész listát a bejárást indító
28                                               # függvénynek
```

BINARYSEARCHTREE OSZTÁLY

Insert függvény

```
46    def Insert(self,data):    # Beilleszt művelet; a rendezettség szerinti helyére illeszti az elemet
47        if self.root==None:       # ha üres volt a fa, akkor
48            self.root=Node(data)  #  az új elemet gyökérként adjuk hozzá
49        else:                                       # különben:
50            self.recursiveInsert(self.root,data)    # a rekurzív fv. hívása a gyökércsomóponttal és a
51                                                    # a beillesztendő adatelemmel (ami most kulcs is)
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

BINARYSEARCHTREE OSZTÁLY

recursiveInsert függvény

```
53  def recursiveInsert(self,node,data):
54      if data<node.data: # ha a beillesztendő elem kisebb, mint az aktuális csomópontbeli elem, akkor
55          if node.left==None:         # ha az aktuális balgyereke üres, akkor
56              node.left=Node(data)    # ide tesszük az új elemet (és a rekurzió leáll)
57
58          else:                       # különben (ekkor a balgyerek nem üres):
59              self.recursiveInsert(node.left,data) # rekurzív hívással a bal oldali részfában folytatjuk
60      else:                         # különben:
61          if data>node.data: # ha a beillesztendő elem nagyobb, mint az aktuális csomópontbeli, akkor
62              if node.right==None:    # ha az aktuális jobbgyereke üres, akkor
63                  node.right=Node(data) # ide tesszük az új elemet (és a rekurzió leáll)
64              else:                       # különben (ekkor a jobbgyerek nem üres):
65                  self.recursiveInsert(node.right,data) # rekurzív hívással a jobb oldali részfában
66                                                        # folytatjuk
67          # ha az "alulról második" if feltétele sem teljesül, akkor már van ilyen kulcsú elem
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT1 – PYTHON_08_01.PY

Nyisd meg és egészítsd ki a Python_08_01.py kódot a *#főprogram* megjegyzés után:

- hozzunk létre egy üres bináris keresőfát kerfa néven
- illesszük be egymás után a ”körte”, ”alma”, ”barack” és ”cseresznye” elemeket
- írassuk ki az elemeket (a bináris fáknál már megismert *printBinTree* függvény segítségével): ábécé sorrend szerinti eredményt kapunk
- próbáljuk meg ismét beilleszteni a ”cseresznye” elemet
- ismét írassuk ki az elemeket

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT2 – PYTHON_08_02.PY

Nyisd meg a Python_08_02.py kódot, és az inorder bejárás mintájára írj két függvényt (*traversalReverseOrder* és *ReverseOrder* néven), amelyekkel fordított sorrendű bejárást tudunk megvalósítani! Ezáltal csökkenő sorrendben tudjuk kiíratni az elemeket.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

BINARYSEARCHTREE OSZTÁLY

Search és recursiveSearch függvények

```
33      def Search(self,data):          # Keres művelet
34          return self.recursiveSearch(self.root,data)  # a rekurzív függvény hívása
35
36      def recursiveSearch(self,node,data):
37          if node==None or node.data==data: # leállási feltétel (alapeset): a csomópont üres,
38                                            # vagyis "leléptünk a fáról", azaz nincs a keresett elem;
39                                            # vagy megtaláltuk
40              return node
41          if data<node.data:      # ha a keresett elem kisebb, mint a csomópontbeli, akkor
42              return self.recursiveSearch(node.left,data) # a bal részfában folytatjuk a keresést
43          return self.recursiveSearch(node.right,data) # a jobb részfában folytatjuk a keresést,
44                                                       # mert itt csak data>node.data lehet már
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT3 – PYTHON_08_03.PY

Nyisd meg és egészítsd ki a Python_08_03.py kódot a *#főprogram* megjegyzés után:

- kerestessük a ”barack” elemet, és írassuk ki, hogy benne van-e a fában
- ezután kerestessük a ”szilva” elemet, és írassuk ki, hogy benne van-e a fában
- vegyük fel a "banán", "szilva" és "ananász” elemeket
- írassuk ki a fa elemeit
- ezután megint kerestessük a ”szilva” elemet (kicsit másképpen, mint fent), és írassuk ki, hogy benne van-e a fában

BINARYSEARCHTREE OSZTÁLY

A törlésnél 3 eset lehet:

1. **a csomópont levél (nincsenek gyermekei)**: egyszerűen "töröljük".

2. **csak egy részfája (gyermeke) van**: ennek a részfának a gyökerét kell a helyére tenni.

3. **két részfája (gyermeke) van**: ekkor a jobb oldali részfájának legbaloldalibb csomópontjával (tehát a sorrendi utódjával) kicseréljük (a benne tárolt értékkel felülírjuk a törlésre kijelölt csúcs tartalmát), ezzel a törlendő csomópontunk vagy levél lett, vagy csak egy részfája van (bal oldali részfája biztosan nem lehet). Így utána eljárhatunk a fenti 2 szabály valamelyike alapján. (Lehetne a bal oldali részfa legjobboldalibb csomópontjával, vagyis a sorrendi elődjével is dolgozni.)

BINARYSEARCHTREE OSZTÁLY

Delete függvény

A *Delete* függvény hívja meg a rekurzív függvényt.

```
81      def Delete(self,data):                  # Törlés művelet
82          self.root=self.recursiveDelete(self.root,data) # a visszaadott csomópont mutatja, hogy
83                                                  # éppen hol járunk a fában
```

BINARYSEARCHTREE OSZTÁLY

recursiveDelete függvény

```
85      def recursiveDelete(self,node,data):
86          if node==None:                  # ha a fa üres; vagy "leléptünk" a fáról, vagyis
87              return node                 # a törlendő elem nincs a fában;
88          if data<node.data:                                  # a bal részfában folytatjuk;
89              node.left=self.recursiveDelete(node.left,data)
90          elif data>node.data:                                # a jobb részfában folytatjuk;
91              node.right=self.recursiveDelete(node.right,data)
92          else:                                           # megvan a törlendő elem
93              if node.left==None and node.right==None:    # ha levélcsomópont
94                  return None
95              elif node.left==None:           # ha csak jobbgyereke van
96                  return node.right
97              elif node.right==None:          # ha csak balgyereke van
98                  return node.left
99              current=node.right              # ha idejut a végrehajtás, akkor két gyereke van,
100             while current.left:             # megkeressük a jobb oldali részfájának legbaloldalibb
101                 current=current.left        # csomópontját (a ciklus végén a current az)
102             tmp=current                     # ez kerül a tmp-be
103             node.data=tmp.data              # ennek az adatrészét megkapja a törlendő csomópont
104             node.right=self.recursiveDelete(node.right,tmp.data) # most már törölhető az eredetileg
105                                             # törlendő csomópont jobb oldali részfájából a tmp
106                                             # csomópont (ami vagy levél, vagy csak egy gyereke van)
107         return node
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT4 – PYTHON_08_04.PY

Nyisd meg és egészítsd ki a Python_08_04.py kódot a *#főprogram* megjegyzés után:

- töröljük a ”cseresznye” elemet
- töröljük a ”barack” és a ”szilva” elemet is
- hozzunk létre egy üres fát kfa néven
- próbáljuk törölni a kfa fából az ”alma” elemet

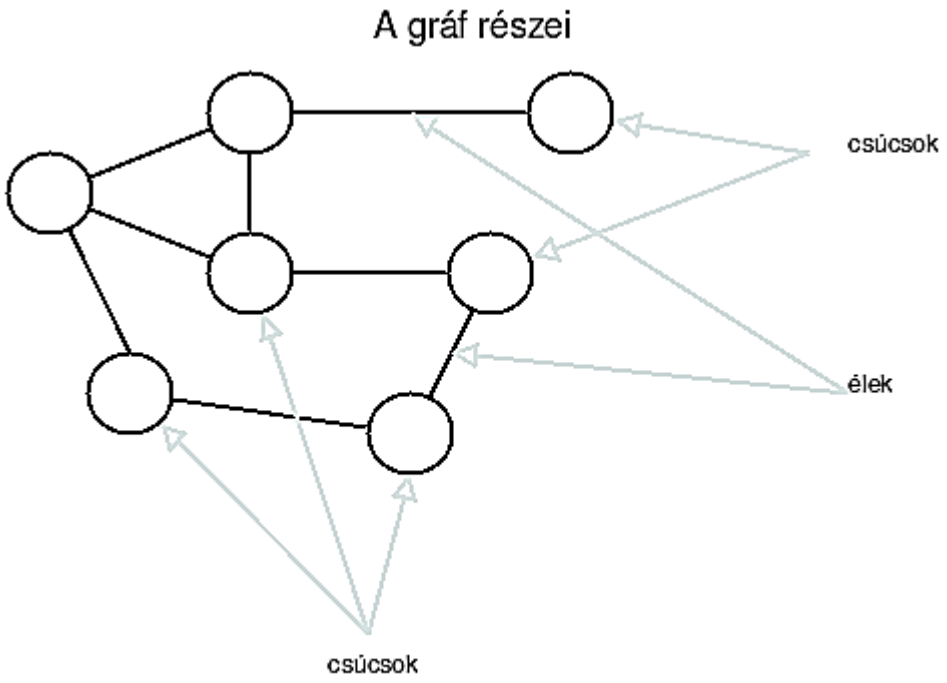
Gráf implementálása

GRÁFOK HASZNÁLATA

A gráfok alapvető adatszerkezetek az informatikában, hiszen megkönnyítik a különböző egyedek közötti bonyolult kapcsolatok ábrázolását és kezelését.

Nagyon hasznosak:

- a különféle hálózatok (pl. társadalmi, közlekedési, számítógépes stb.) modellezésére,
- molekulaszerkezetek vizsgálatára,
- vagy például a bioinformatika területén is
- (és még hosszasan lehetne sorolni).



GRÁF IMPLEMENTÁLÁSA

A gráf ábrázolásához **szomszédsági listás tárolást** fogunk alkalmazni, ami általában láncolt listák tömbje szokott lenni: az i. elem, vagyis az i. láncolt lista tartalmazza az i. csúcs szomszédjait (vagyis azon csúcsokat, amelyekbe vezet él az i. csúcsból).

Ezt most annyiban módosítjuk, hogy nem tömböt, hanem **szótárt** fogunk használni. A megfeleltetés tehát a következő:

gráf csúcspontjai ⟷ szótár kulcsai

csúcsok szomszédjainak listái ⟷ szótár értékei

Irányítatlan gráf esetén az egyes éleket mindkét csúcs listájában szerepeltetni kell, és vigyázni kell például az *addEdge* (*HozzáadÉl*) és a *removeEdge* (*EltávolítÉl*) metódusok használatával: (u,v) él esetén (v,u) is szerepel!

GRÁFOK (FOLYT.)

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

- A megvalósított gráf nem rendelkezik súlyokkal; illetve, az egyes csomópontok most nem rendelkeznek külön adatrésszel (csak a nevükkel vagy a sorszámukkal). Ha szükséges, akkor a szótár értékeiként szereplő listákat lehet bővíteni a kívánt adatokkal.
- A megvalósításunk irányított és irányítatlan gráfokkal is használható.
- Ha az adott problémakörben biztosan tudnánk, hogy a gráfunkban pl. nincs többszörös él, akkor az éllisták helyett halmazokat is lehetne használni.
- A megvalósításunkban lesznek olyan függvények, amelyek működése előfeltételhez kötött, vagyis a hívásukkor nekünk kell biztosítani, hogy ne kapjunk futási hibát, illetve kivételt.

GRAPH (GRÁF) OSZTÁLY

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

```
1  class Graph:
2      def __init__(self,dict=None): # konstruktor
3          if dict==None:            # ha nem adtunk meg dict, azaz szótár paramétert, akkor
4              dict={}               # legyen üres, és így a gráf is üres;
5          self.dict=dict            # a dict adattagba kerül a dict paraméter értéke
```

GRÁFOK
getVertices és getEdges függvények

```
8     def getVertices(self):                  # Csúcsok művelet:
9         return list(self.dict.keys())       # egy listában adja vissza a gráf összes csúcsát
10
11    def getEdges(self, directed=True): # Élek művelet: ha a gráf nem irányított, akkor
12        edges=[]                            # a directed paramétert False-nak kell megadni;
13        for u in self.dict:                 # egy listában adja vissza a gráf összes élét;
14            for v in self.dict[u]:
15                if directed:                # ha a gráf irányított:
16                    edges.append((u,v)) # akkor az (u,v) él mindenképpen bekerül a listába;
17                elif (v,u) not in edges:# különben (irányítatlan) ha fordított sorrendben
18                    edges.append((u,v)) # még nincs az éllistában, akkor felvesszük az
19        return edges                        # (u,v) élt; végül visszaadjuk az élek listáját
```

GRÁFOK
addEdge függvény

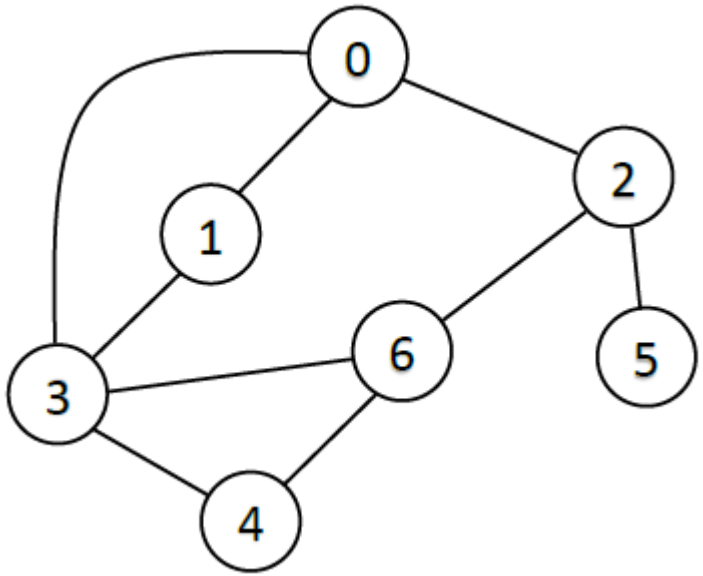
Az *addEdge* (*HozzáadÉl*) függvénynél előfeltétel az, hogy mindkét csúcspont létezik. Egyébként erősen hiányos, hiszen futási hibát kapunk, ha az *u* nem csúcspontja a gráfnak (vagyis a szótárban nincs ilyen kulcs). Ha *u* csúcspont, de a *v* nem, akkor is futási hibát kapunk, ha a gráf irányítatlan (a (*v*,*u*) él hozzáadásánál). A *get* függvény használata sem jelentene önmagában megoldást. Bár ekkor a hiányzó kulcs nem okozna problémát, de ha *u* létezik és *v* nem (és a gráf irányított), akkor úgy csatolnánk az (*u*,*v*) élt a gráfhoz, hogy a *v* csúcspont nem is létezik.

```
21      def addEdge(self,u,v,directed=True): # HozzáadÉl művelet:
22          self.dict[u].append(v)             # az (u,v) él hozzáadása;
23          if not directed:                   # ha irányítatlan a gráf, akkor a (v,u) élt is kell
24              self.dict[v].append(u)
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

AZ IRÁNYÍTATLAN TESZTGRÁFUNK

```
27  szotar={
28              "0" : ["1","2","3"],
29              "1" : ["0","3"],
30              "2" : ["0","5","6"],
31              "3" : ["0","1","4","6"],
32              "4" : ["3","6"],
33              "5" : ["2"],
34              "6" : ["2","3","4"]
35          }
36  graf=Graph(szotar)
```



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT1 – PYTHON_09_01.PY

Nyisd meg a Python_09_01.py kódot, majd írd meg az *addEdge* függvényt úgy, hogy biztonságosan működjön! Végül teszteld!

Segítség az (*u*,*v*) él beszúrásához:

- ha *u* nincs a gráfban, akkor *u* új csúcs lesz, és a *v*-ből az *u*-hoz tartozó egyelemű szomszédsági lista lesz (vagyis új kulcs-érték párként fel kell venni a szótárba (*u*,*v*)-t)
- egyébként *u* benne van a gráfban, és akkor csak bővítjük a szomszédsági listáját *v*-vel

Ezután

- ha *v* nincs a gráfban, akkor
    - ha a gráf irányított, akkor csak fel kell venni *v*-t egy olyan csúcsnak, amelyből nem vezet ki él (vagyis a szótárban a *v* kulcshoz üres szomszédsági lista tartozik)
    - különben új kulcs-érték párként fel kell venni a szótárba a (*v*,*u*)-t is
- különben *v* benne van a gráfban és
    - ha a gráf nem irányított, akkor *v* szomszédsági listáját bővítjük *u*-val

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

removeEdge és Neighbors függvények

A ***removeEdge*** (***EltávolítÉl***) és a ***Neighbors*** (***Szomszédok***) függvény használatánál szintén feltételezve van, hogy az eltávolítandó él létezik; illetve létezik az a csúcs, amelynek a szomszédjait szeretnénk megkapni.

A javításhoz a *removeEdge*-nél felhasználható a következő oldalon szereplő *isEdge* függvény; a *Neighbors*-nél pedig a szótárak *get* függvénye (amely nem létező kulcsnál *None* értéket ad vissza).

```
26      def removeEdge(self,u,v,directed=True): # EltávolítÉl művelet:
27          self.dict[u].remove(v)                # törli az (u,v) élt;
28          if not directed:                      # ha irányítatlan a gráf:
29              self.dict[v].remove(u)            # akkor a (v,u) élt is töröljük
30
31      def Neighbors(self,u):              # Szomszédok művelet: az összes olyan csúcs listáját
32          return self.dict.get(u)         # adja, amelyhez vezet él u-ból
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

addVertex, Size és isEdge függvények

```
34      def addVertex(self,u):                  # HozzáadCsúcs művelet:
35          if u not in self.dict:              # ha még nincs a gráfban ez a csúcs:
36              self.dict[u]=[]                 # akkor felvesszük üres szomszédsági listával
37                                              # (vagyis izolált pontként)
38
39      def Size(self):                     # Méret művelet: hány csúcspont van
40          return len(self.dict)
41
42      def isEdge(self,u,v):               # VanÉl művelet: visszaadott logikai értékkel jelzi,
43          return v in self.dict[u]        # hogy van-e (u,v) él a gráfban
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT2 – PYTHON_09_02.PY

Nyisd meg a Python_09_02.py kódot, majd egészítsd ki a kommentek segítségével a # *főprogram* komment utáni részt!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

BREADTH-FIRST SEARCH (SZÉLESSÉGI BEJÁRÁS)

Breadth-first search (*BFS*, *„Szélesség-első keresés”*), másképpen ***level-order traversal*** (*„szintenkénti sorrend bejárás”*), magyarul szokásosan ***szélességi bejárás*** lényege röviden: meglátogatja a kezdő csomópontot, majd a kezdő csomópont összes szomszédját, majd ezeknek a csomópontoknak a szomszédjait, és így tovább. A meglátogatott csomópontokat a „meglátogatott” (*visited*) halmaz segítségével követi nyomon. A sor a meglátogatandó csomópontok tárolására szolgál a felfedezésük sorrendjében. *(Precízebben a kódnál lesz.)*

Néhány alkalmazása:

- útkereső algoritmusok (pl. GPS-navigációnál legrövidebb út)
- ciklusok felderítése irányítatlan gráfokban
- minimális feszítőfa ellőállítása
- maximális áramlás megtalálása egy hálózatban (Ford-Fulkerson algoritmus) stb.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

SZÉLESSÉGI BEJÁRÁS IMPLEMENTÁLÁSA ITERATÍV MEGOLDÁS

```
46      def BreadthFirstTraversal(self,start): # SzélességiBejárás; nem rekurzív megvalósítás;
47                                             # egy összefüggő gráfot tud csak bejárni
48          queue=[start]    # a kezdőpontot betesszük egy üres sorba (Python-lista sorként kezelve)
49          visited=set()    # üreshalmaz létrehozása
50          visited.add(start) # a kezdőpontot a halmazba is betesszük
51                             # A sorba azon pontokat tesszük, amelyeket elért a bejárásunk, de még
52                             # nem kerültek feldolgozásra.
53                             # A halmazt arra használjuk, hogy egy elért ("meglátogatott") pontot,
54                             # azaz vagy már feldolgozott, vagy feldolgozásra "kijelölt"
55                             # (sorba bekerült) pontot ne vegyünk fel újból a sorba.
56          while queue:       # egyenértékű ezzel: len(queue)!= 0, vagyis amíg nem üres a sor, addig:
57              node=queue.pop(0)     # kivesszük a sor elejéről a következő pontot a node-ba,
58              print(node,end=' ')   # feldolgozzuk (azaz most kiírjuk), majd
59              for tmp in self.Neighbors(node):# az összes szomszédját megvizsgáljuk:
60                  if tmp not in visited:      # ha a tmp szomszéd még nincs a halmazban:
61                      queue.append(tmp)       # akkor betesszük a sor végére és
62                      visited.add(tmp)        # betesszük a halmazba is
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

SZÉLESSÉGI BEJÁRÁS IMPLEMENTÁLÁSA REKURZÍV MEGOLDÁS

Ez egy kicsit erőltetett megoldás, javasolt inkább az iteratív algoritmuson alapuló kód. Egy nagyobb gráfnál amúgy is gondot okozhat a sok rekurzív hívás.

```
65      def rec_BreadthFirst(self,start,visited=None,queue=None):
66          if visited is None:
67              visited=set()
68          if queue is None:
69              queue=[start]
70          if not queue:                   # ha a sor üres, akkor visszatérés
71              return
72          node=queue.pop(0)
73          if node not in visited:
74              print(node,end=' ')
75              visited.add(node)
76              for tmp in self.Neighbors(node):
77                  if tmp not in visited:
78                      queue.append(tmp)
79          self.rec_BreadthFirst(start,visited,queue)
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT3 – PYTHON_09_03.PY

Nyisd meg a Python_09_03.py kódot, teszted a programot (az iteratív és a rekurzív szélességi bejárást)!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

DEPTH-FIRST SEARCH (MÉLYSÉGI BEJÁRÁS)

Depth-first search (*DFS*, *„Mélység-első keresés”*), magyarul szokásosan ***mélységi bejárás*** lényege röviden: meglátogatja a kezdő csomópontot, majd onnan továbblép egy szomszédos, még be nem járt csomópontba. Ha egy adott csomópontból nem tud bejáratlan csomópontba jutni, akkor visszalép a járt úton addig, amíg nem ér olyan csomópontba, ahonnan tovább lehetne lépni. Ha pedig visszajut a kezdőpontba és már ott sincs bejárható csomópont, akkor az azt jelenti, hogy teljesen bejárta a gráfot. Az elágazási pontokhoz való visszatéréshez használ vermet. *(Precízebben a kódnál lesz.)*

Néhány alkalmazása:

- egy gráf erősen összefüggő komponenseinek megtalálása
- útvonal megtalálása, feltérképezése
- ciklusok felderítése gráfban
- topológiai rendezés
- rejtvények megoldása
- egy probléma ütemezése stb.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

MÉLYSÉGI BEJÁRÁS IMPLEMENTÁLÁSA ITERATÍV MEGOLDÁS

```
def DepthFirstTraversal(self,start):    # MélységiBejárás; nem rekurzív megvalósítás;
    node=start                          # egy összefüggő gráfot tud csak bejárni;
    edge=0
    node_stack = [node]  # a kezdőpontot betesszük egy üres verembe (Python-lista veremként)
    edge_stack=[edge]    # a 0 élsorszámot is betesszük egy másik verembe
    visited = set()  # üreshalmaz létrehozása
                     # A halmaz itt is arra szolgál, hogy ne dolgozzunk fel egy pontot
                     # többször.
                     # A veremre azért van szükség, hogy az egyes bejárt utakról vissza
                     # tudjunk térni az egyes elágazási pontokhoz, de az is kell, hogy
                     # melyik volt az utolsó él, amely mentén továbbhaladtunk, ezért
                     # az élt is vermelni kell. Az egyszerűség kedvéért két külön verem van
                     # (az egyik a pontoknak, a másik a hozzájuk tartozó éleknek).
                     # Amíg nem üres a verem, addig még van bejáratlan pont.
```

MÉLYSÉGI BEJÁRÁS IMPLEMENTÁLÁSA ITERATÍV MEGOLDÁS (FOLYT.)

```
while node_stack:                   # amíg nem üres a pontverem, addig:
    if node not in visited:         # ha a pont nincs benne a halmazban:
        visited.add(node)           # akkor betesszük a halmazba, és
        print(node, end=' ')        # feldolgozzuk (azaz most kiírjuk);
    while (edge<=(len(self.Neighbors(node))-1) and  # az aktuális pont olyan szomszédját
          (self.Neighbors(node)[edge] in visited)): # keressük, amelyik nincs benne a
        edge+=1                             # halmazban: ehhez az él sorszámát növelgetjük 1-gyel
    if (edge<=(len(self.Neighbors(node))-1)): # ha van ilyen szomszéd,
        node_stack.append(node)             # akkor a pontot és az él sorszámát
        edge_stack.append(edge)             # (amelyen "továbbmegyünk") verembe tesszük,
        node=self.Neighbors(node)[edge] # majd az él másik végpontja (a talált szomszéd)
        edge=0                              # lesz az aktuális (bekerül a node-ba),
                                          # és a legelső (vagyis a 0.) szóba jöhető élt vesszük
    else:                   # ha nem találtunk ilyen szomszédot (mind benne van a halmazban),
        node=node_stack.pop() # akkor kivesszük a veremből a tetején lévő pontot és a
        edge=edge_stack.pop() # megfelelő élsorszámot,
        edge+=1               # majd növeljük az élsorszámot
        # (vagyis visszaléptünk egyet és próbáljuk másik "útvonalon" folytatni).
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT4 – PYTHON_09_04.PY

Nyisd meg a Python_09_04.py kódot, majd írd meg a kommentek segítségével a mélységi keresést megvalósító rekurzív függvényt!

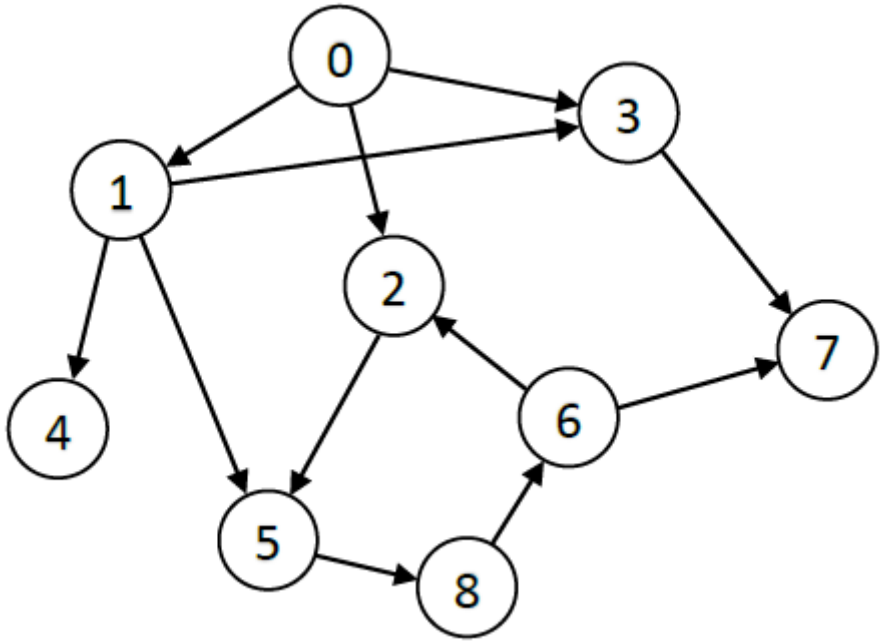
Próbáld is ki (hasonlítsd össze az iteratív és a rekurzív mélységi bejárás eredményét)!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT5 – PYTHON_09_05.PY

Nyisd meg a Python_09_05.py kódot, majd add meg az alábbi ábrán látható irányított gráfhoz tartozó szomszédsági listás ábrázolást (vagyis a szótárat)!

Teszteld a programot!



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KÖSZÖNÖM

A MEGTISZTELŐ FIGYELMET!

Módné T. Judit ✉ *modne.t.judit@amk.uni-obuda.hu*