Dictionaries - Implementation

Perkuliahan Algoritma Pemrograman pada Semester Ganjil 2020

In [ ]:

```
import matrices
```

Concatenation pada List

Operasi '+' pada list merupakan operasi untuk menggabungkan dua buah list, bukan operasi penjumlahan. Berikut adalah contoh operasi concatenation pada list

In [ ]:

```
a=[1,2,3]
b=[4,5,6]
print('a=',a)
print('b=',b)
print('a+b=',a+b)
print('[a]+[b]=',[a]+[b])
```

In [ ]:

```
c=[]
c=c+a
print('c=',c)
d=[]
d=d+[a]
print('d+[a]=',d)
d=d+[b]
print('d+[b]=',d)
```

In [ ]:

```
row=int(input("jumlah baris Matriks-1="))
col=int(input("jumlah kolom Matriks-1="))
Mat1=matrices.createMat('Mat1',row,col)
```

In [ ]:

```
print(Mat1)
```

In [ ]:

```
matrices.displayMat(Mat1)
```

In [ ]:

```
print(Mat1)
matrices.displayMat(Mat1)
```

In [ ]:

```
matrices.displayMat(Mat2)
```

In [ ]:

```
mat3=matrices.addMat(Mat1,Mat2)
matrices.displayMat(mat3)
```

In [ ]:

```
row=int(input("jumlah baris Matriks-4="))
col=int(input("jumlah kolom Matriks-4="))
Mat4=matrices.createMat('Mat2',row,col)
```

In [ ]:

```
matrices.displayMat(Mat1)
matrices.displayMat(Mat4)
```

In [ ]:

```
mat5=matrices.multMat(Mat1,Mat4)
print(mat5)
```

In [ ]:

```
matrices.displayMat(mat5)
```

In [ ]:

```
with open('matrices.py', 'r') as f:
    print(f.read())
```

Penggunaan Dictionaries

Kelebihan struktur data dictionaries ini dibandingkan dengan list terletak pada **key** yang dimiliki oleh dictionaries, yang tidak harus berupa integer, akan tetapi dapat berupa string, character, atau yang lain. Sehingga jika suatu permasalahan membutuhkan key bukan berupa integer, maka struktur data ini sangatlah tepat.

Sparse Matrix

Pada contoh sebelumnya, operasi matriks menggunakan struktur data list. Setiap indeks berisi suatu nilai. Akan tetapi pada beberapa komputasi membutuhkan penyimpanan dalam bentuk **Sparse matrix**, yaitu suatu matrix yang sebagian besar anggotanya berupa nilai nol, hanya indeks tertentu saja yang memiliki anggota berupa nilai bukan nol, seperti contoh berikut.

$$\begin{bmatrix} 0 & 0 & 0 & 4 & 0 \\ 0 & 4 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Matriks tersebut adalah matriks yang berukuran 4 x 5, oleh karena itu terdapat 20 anggota yang dapat diisi atau disimpan dalam matriks tersebut. Hanya saja, hanya 4 indeks saja yang berisi nilai bukan nol, sedangkan yang lainnya bernilai nol. Oleh karena itu, jika disimpan dengan menggunakan struktur data list, banyak indeks yang berisi nilai nol, hal ini menjadi tidak efisien.

Sehingga dibutuhkanlah struktur data berupa dictionaries yang hampir menyerupai list, hanya saja key yang terdapat pada dictionaries tidak harus berupa integer dan berurutan. Dengan dictionaries ini, hanya anggota yang bernilai tidak nol saja yang akan disimpan.

Hanya saja karena matriks yang akan dibuat adalah matriks dua dimensi, maka dibutuhkanlah pasangan key yang menyatakan baris dan kolom suatu matriks.
Untuk itu, key dari matriks ini menggunakan **tuple**.

Tuple, sama halnya dengan list, tuple ini terdiri dari beberapa elemen, dan elemen tersebut dapat terdiri dari berbagai tipe. Jika pada list, representasi anggota menggunakan kurung siku atau [a,b,c,...,d], maka pada tuple menggunakan kurung atau (a, b, c, ..., d). Elemen pada tuple bersifat **immutable**, tidak dapat dirubah. Contoh tuple adalah sebagai berikut :

In [22]:

```
def createSparseMatrix():
    mat={}
    num=int(input('Jumlah elemen = '))
    for i in range(num):
        bar=int(input('baris ke - ?'))
        kol=int(input('kolom ke - ?'))
        data=int(input ('data ['+str(bar)+','+str(kol)+']='))
        mat[bar,kol]=data
    return mat

matriks=createSparseMatrix()
print(matriks)
```

```
Jumlah elemen = 4
 baris ke - ? 0
kolom ke - ? 3
 data [0,3]=4
 baris ke - ? 1
kolom ke - ? 1
 data [1,1]=4
 baris ke - ? 1
kolom ke - ? 4
 data [1,4]=7
 baris ke - ? 3
kolom ke - ? 0
 data [3,0]=1
{(0, 3): 4, (1, 1): 4, (1, 4): 7, (3, 0): 1}
```

In [1]:

```
a={1:1,7:2,9:3}
```

In [2]:

```
print(a)
```

```
{1: 1, 7: 2, 9: 3}
```

In [10]:

```
a.get((4),0)
```

Out[10]:

```
0
```

In [ ]:

```
aTuple=(1,2,4)
print(aTuple)
aList=[1,2,4]
print(aList)
```

In [ ]:

```
aTuple[1]=100
```

In [ ]:

```
aList[1]=100
print(aList)
```

Pada contoh diatas, maka dapat dilihat bahwa elemen-elemen yang terdapat pada struktur data tuple bersifat **imutable**, sedangkan pada list bersifat **mutable**.

Kelebihan dari tuple ini dibandingan list adalah :

- Data yang terdapat pada tuple akan aman, karena tidak dapat dirubah
- dengan sifat imutable ini, maka tuple dapat digunakan sebagai key pada **dictionary**

Misalkan terdapat matriks berikut :

$$\begin{bmatrix} 0 & 0 & 0 & 4 & 0 \\ 0 & 4 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Contoh penyimpanan matriks tersebut dengan menggunakan dictionaries dengan key berupa tuple, dapat dilihat sebagai berikut :

In [12]:

```
mat={}
mat[0,3]=1
print(mat)
```

```
{(0, 3): 1}
```

In [13]:

```
mat[(1,1)]=4
print(mat)
```

```
{(0, 3): 1, (1, 1): 4}
```

In [17]:

```
def createSparseMatrix():
    numOfElement=int(input('berapa jumlah elemen'))
    mat={}
    for i in range(numOfElement):
        r=int(input('Baris ke- ?'))
        c=int(input('Kolom ke- ?'))
        data=int(input('data ('+str(r)+','+str(c)+'= '))
        mat[r,c]=data
    return mat
```

In [21]:

```
def addSparseMatrix(mat1,mat2,bar,kol):
    mat={}
    for i in range(bar):
        for j in range(kol):
            if mat1.get((i,j),0)!=0 and mat2.get((i,j),0)==0:
                mat[i,j]=mat1[i,j]
            elif mat1.get((i,j),0)!=0 and mat2.get((i,j),0)!=0:
                mat[i,j]=mat1[i,j]+mat2[i,j]
            elif mat1.get((i,j),0)==0 and mat2.get((i,j),0)!=0:
                mat[i,j]=mat2[i,j]

    return mat

a={(0,0):1,(1,1):2,(2,3):4}
b={(1,1):3,(2,0):1,(2,3):2}
c=addSparseMatrix(a,b,3,4)
print(c)
```

```
{(0, 0): 1, (1, 1): 5, (2, 0): 1, (2, 3): 6}
```

In [18]:

```
mat1=createSparseMatrix()
```

```
berapa jumlah elemen3
Baris ke- ?0
Kolom ke- ?3
data (0,3= 4
Baris ke- ?1
Kolom ke- ?1
data (1,1= 4
Baris ke- ?1
Kolom ke- ?4
data (1,4= 7
```

In [19]:

```
print(mat1)
```

```
{(0, 3): 4, (1, 1): 4, (1, 4): 7}
```

In [ ]:

```
mat2={(0,3):1,(1,1):4,(1,4):7,(3,0):1}
print(mat2)
```

In [ ]:

```
mat2.get((0,5),0)
```

Untuk mengetahui apakah sebuah key terdapat pada suatu dictionary, dapat menggunakan method get pada dictionary. Jika key terdapat pada dictionary tersebut, maka return value akan berupa value yang terdapat pada key, jika key tidak terdapat dictionary tersebut, maka return value adalah nol.

Berikut contoh penggunaan method get pada dictionary untuk mengetahui keberadaan suatu key

In [ ]:

```
a=mat2.get((1,1),0)
print(a)
b=mat2.get((1,3),0)
print(b)
```

Berikut adalah contoh penjumlahan dua buah sparse matrix

In [ ]:

```
a={(0,2):1,(1,0):2}
b={(0,1):10,(2,0):7}
c={}
jmlBar=3
jmlKol=3
for i in range(0,jmlBar):
    for j in range(0,jmlKol):

        if (a.get((i,j),0)!=0) and (b.get((i,j),0)!=0):

            c[i,j]=a[i,j]+b[i,j]
        elif (a.get((i,j),0)==0) and (b.get((i,j),0)!=0):

            c[i,j]=b[i,j]
        elif (a.get((i,j),0)!=0) and (b.get((i,j),0)==0):

            c[i,j]=a[i,j]


print (c)
```

In [ ]:

```
import sparse
```

In [ ]:

```
(a,row_a,col_a)=sparse.inputData()
```

In [ ]:

```
sparse.displayData(a,row_a,col_a)
```

In [ ]:

```
print(a)
```

In [ ]:

```
sparse.displayData(a,row_a,col_a)
```

In [ ]:

```
(b,row_b,col_b)=sparse.inputData()
```

In [ ]:

```
sparse.displayData(b,row_b,col_b)
```

In [ ]:

```
c=sparse.multMatrix(a,b,3,2)
```

In [ ]:

```
print(c)
```

In [ ]:

```
sparse.displayData(c,3,2)
```

In [ ]:

```
with open('sparse.py', 'r') as f:
    print(f.read())
```