

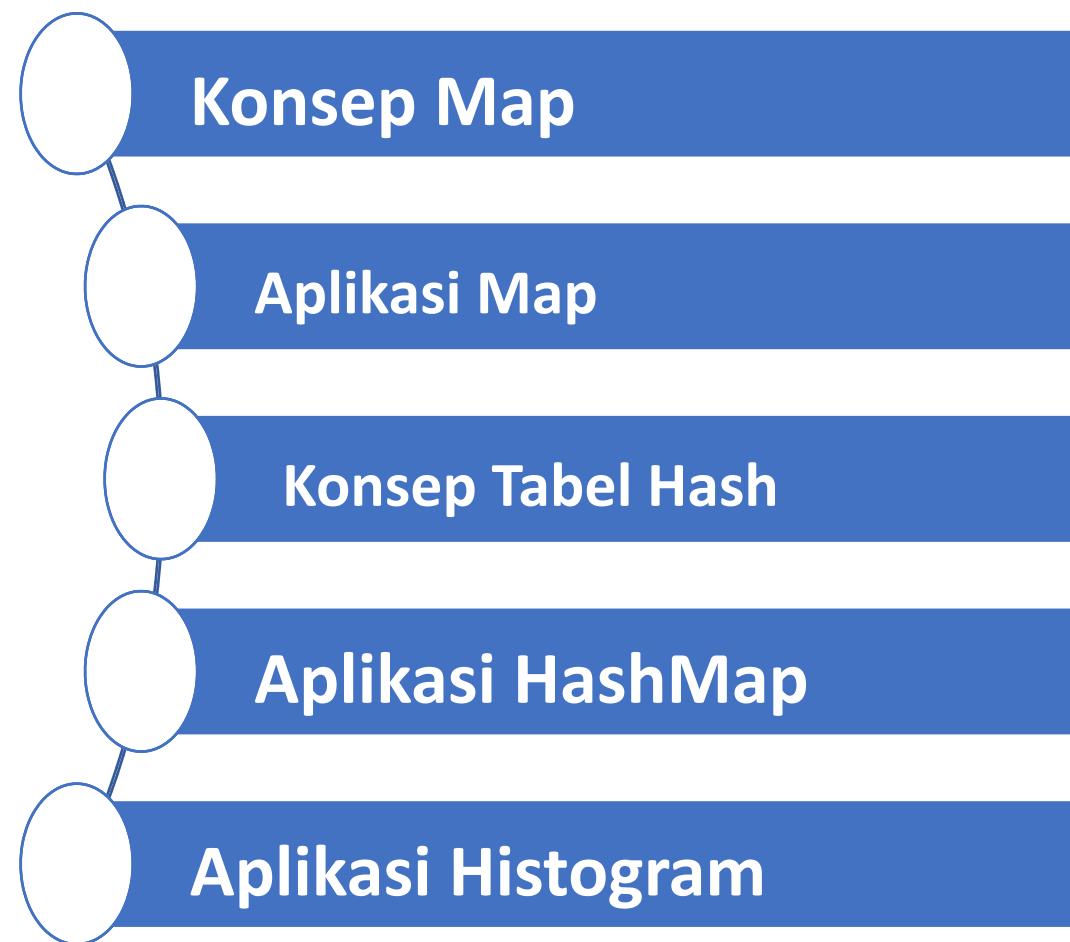


STRUKTUR DATA (PYTHON)

“Struktur Map dan Tabel Hash”

[@SUARGA] | [Pertemuan 15]

OutLine





Konsep Map

- Map adalah larik yang assosiatifi (associative array), suatu struktur yang memiliki fungsi pemetaan antara suatu nilai (value) dengan kunci-nya (key)
- Struktur Map dapat ditemukan pada berbagai aplikasi seperti: basis-data, domain name server (DNS), dan pada berbagai aplikasi lainnya.

ADT Map

- **M=MapSD()** : inisialisasi suatu map
- **M.add(k,v)** : menambah isi dari map
- **M.valueOf(k)** : mengembalikan nilai v dari kunci k
- **M.remove(k)** : menghapus item dengan kunci k
- **M.popitem()** : menghapus elemen terakhir
- **M.get(k, d=None)** : memberikan nilai pada kunci k bila k ada dalam map, bila tidak maka mengembalikan d
- **len(M)** : memberikan banyaknya item dalam map
- **iter(M)** : menyediakan iterasi menurut urutan kunci dari map M
- **M.popitem()** : mengeluarkan suatu elemen dari map
- **M.clear()** : mengosongkan map
- **M.keys()** : mengembalikan semua kunci yang ada dalam map
- **M.values()** : mengembalikan semua nilai yang ada dalam map
- **M.items()** : mengembalikan semua item (key,value) dari map

Implementasi Map

```
1 # Implementasi dari Map ADT menggunakan single list.
2 # MapSD.py : @Suarga
3 class MapSD :
4     # Creator objek Map
5     def __init__( self ):
6         self._entryList = list()
7
8     # mengembalikan jumlah anggota dalam Map.
9     def __len__( self ):
10        return len( self._entryList )
11
12    # menentukan posisi anggota dalam Map berdasarkan key.
13    def __contains__( self, key ):
14        ndx = self._findPosition( key )
15        return ndx is not None
16
17    # menambahkan entri baru bila key-nya baru. Bila key sudah ada
18    # maka nilainya adalah nilai baru (new value).
19    def add( self, key, value ):
20        ndx = self._findPosition( key )
21        if ndx is not None : # if the key was found
22            self._entryList[ndx].value = value
23            return False
24        else : # otherwise add a new entry
25            entry = _MapEntry( key, value )
26            self._entryList.append( entry )
27            return True
```

```
28
29     # memberikan value yang sesuai dengan key.
30     def valueOf( self, key ):
31         ndx = self._findPosition( key )
32         if ndx is not None:
33             val = self._entryList[ndx].value
34         else:
35             val = ' '
36         return val
37
38     # menghapus entri sesuai dengan key yang dimasukkan.
39     def remove( self, key ):
40         ndx = self._findPosition( key )
41         assert ndx is not None, "Invalid map key."
42         val = self._entryList[ndx].value
43         self._entryList.pop( ndx )
44         return val
45
46     # menghapus entri pertama
47     def popitem(self):
48         val = self._entryList[-1].value
49         self._entryList.pop(-1)
50         return val
```

```
51
52 # mengembalikan value entry k,
53 # atau menetapkan value = d apabila None
54 def get(self, key, d=None):
55     ndx = self._findPosition( key )
56     if ndx is not None:
57         val = self._entryList[ndx].value
58     else:
59         val = d
60         entry = _MapEntry(key, val)
61         self._entryList.append( entry )
62
63     return val
64
65 # memberikan daftar key yang ada
66 def keys(self):
67     allkey = []
68     m = len(self)
69     for i in range(m):
70         allkey.append(self._entryList[i].key)
71
72     return allkey
```

```
74 # memberikan daftar values
75 def values(self):
76     allval = []
77     m = len(self)
78     for i in range(m):
79         allval.append(self._entryList[i].value)
80
81     return allval
82
83 # menghapus semua items
84 def clear(self):
85     m = len(self)
86     for i in range(m):
87         self._entryList.pop(-1)
88
89
90 # method utk membantu pencarian index position dari kategori.
91 # bila key tidak ada, maka jawabannya None.
92 def _findPosition( self, key ):
93     # Iterate through each entry in the list.
94     for i in range( len(self) ) :
95         # Is the key stored in the ith entry?
96         if self._entryList[i].key == key :
97             return i
98     # When not found, return None.
99     return None
```

```
100  
101 # Storage class untuk menyediakan tempat pasangan key/value.  
102 class _MapEntry :  
103     def __init__( self, key, value ):  
104         self.key = key  
105         self.value = value  
106
```

Test Map

```
1 #MapSD_main.py
2 #program untuk mencoba struktur Map
3
4 import string
5 from MapSD import *
6
7 def main():
8     m = MapSD()
9     s = string.ascii_lowercase #kumpulan huruf kecil dari ascii
10    print("Memuat huruf ke dalam MAP")
11    for k, v in enumerate(s):
12        m.add(k, v)
13    print("Isi Map:")
14    for k in range(len(s)):
15        print (k, m.valueOf(k))
16
17    print("Mengambil elemen terakhir:")
18    x=m.popitem()
19    print(x, ' removed')
20    print("Mengapus isi kunci 5")
21    x=m.remove(5)
22    print(x, ' removed')
```

```
23  
24 print("Isi MAP saat ini:")  
25 for k in range(len(m)+1):  
26     print (k, m.valueOf(k))  
27  
28 x=m.get(10, ' ')  
29 print('value of 10 is ',x)  
30  
31 x=m.get(5, ' ')  
32 print('value of 5 is ',x)  
33  
34 x=m.get(24, ' ')  
35 print('value of 24 is ', x)  
36  
37 kk = m.keys()  
38 print('List of keys : ')  
39 print(kk)  
40  
41 vv=m.values()  
42 print('List of values : ')  
43 print(vv)
```

```
44  
45 print('Clear the map')  
46 m.clear()  
47 kk = m.keys()  
48 print(kk)  
49  
50 main()
```

Memuat huruf ke dalam MAP

Isi Map:

0 a

1 b

2 c

3 d

4 e

5 f

6 g

7 h

21 v

8 i

22 w

9 j

23 x

10 k

24 y

11 l

25 z

12 m

Mengambil elemen terakhir:

13 n

z removed

14 o

Mengapus isi kunci 5

15 p

f removed

16 q

17 r

18 s

19 t

20 u

Hasil Test

Mencoba memasukkan huruf kecil a .. z sebagai value dan kuncinya 0 .. 25

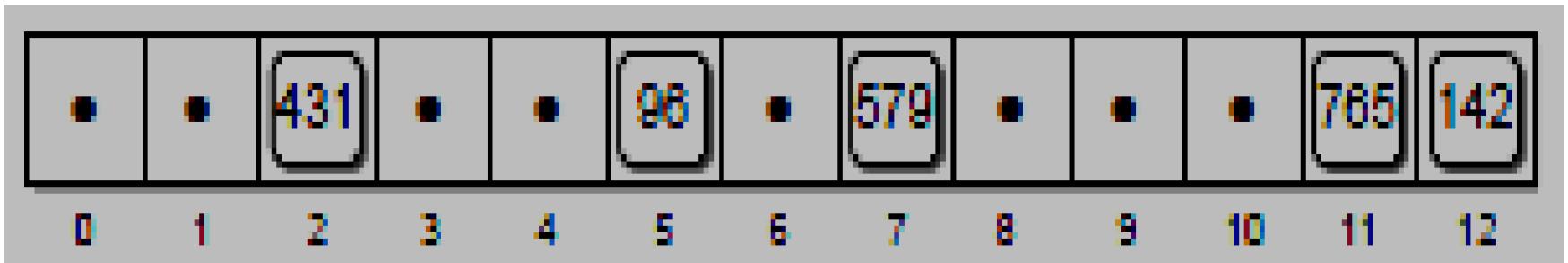
Isi MAP saat ini:

```
0 a
1 b
2 c
3 d
4 e
5
6 g
7 h
8 i
9 j
10 k
11 l
12 m    value of 10 is k
13 n    value of 5 is
14 o    value of 24 is y
15 p    List of keys :
16 q    [0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
17 r    23, 24, 5]
18 s    List of values :
19 t    ['a', 'b', 'c', 'd', 'e', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q',
20 u    'r', 's', 't', 'u', 'v', 'w', 'x', 'y', '']
21 v
22 w
23 x
24 y
```

Konsep Tabel Hash => HashMap

- Hashing adalah proses mapping suatu kunci dengan indeks, baik merupakan indeks array maupun merupakan indeks suatu lokasi yang lain.
- Kunci disimpan dalam tabel yang disebut hash-table dan indeks dari tabel dihitung melalui suatu fungsi, hash-function.

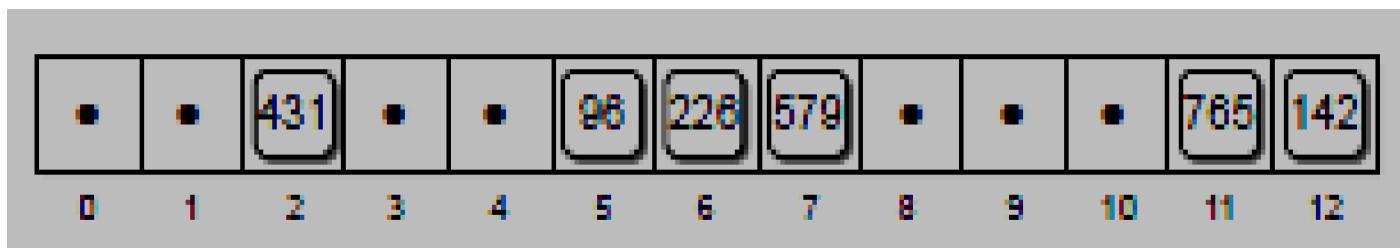
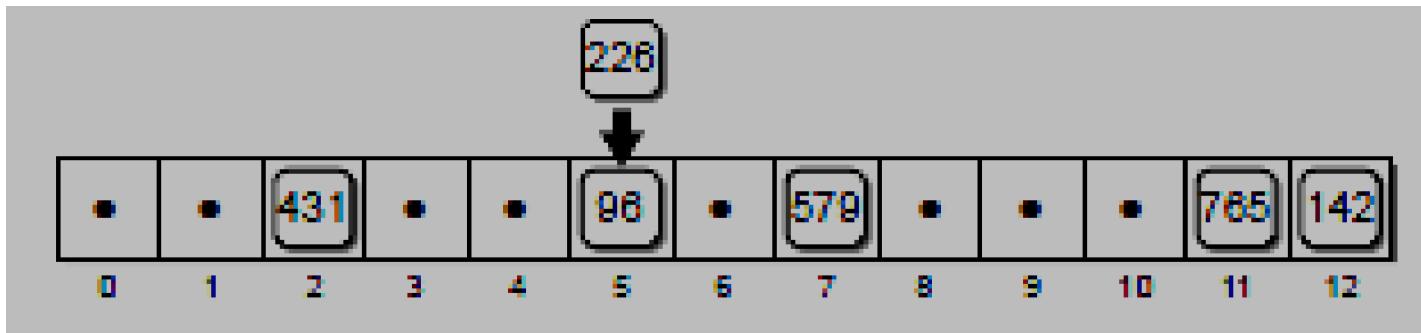
- Andaikan suatu hash bisa memuat M kunci, maka indeks dapat dibentuk misalnya melalui fungsi:
 - $h(key) = key \% M$
- **catatan key % M sama dengan key mod(M)**
- Contoh: tempatkan kunci : 765, 431, 96, 142, 579 ke dalam tabel dengan $M=13$, maka:
 - $h(765) = 765 \% 13 = 11,$
 - $h(431) = 431 \% 13 = 2,$
 - $h(96) = 96 \% 13 = 5$
 - $h(142) = 142 \% 13 = 12,$
 - $h(579) = 579 \% 13 = 7$



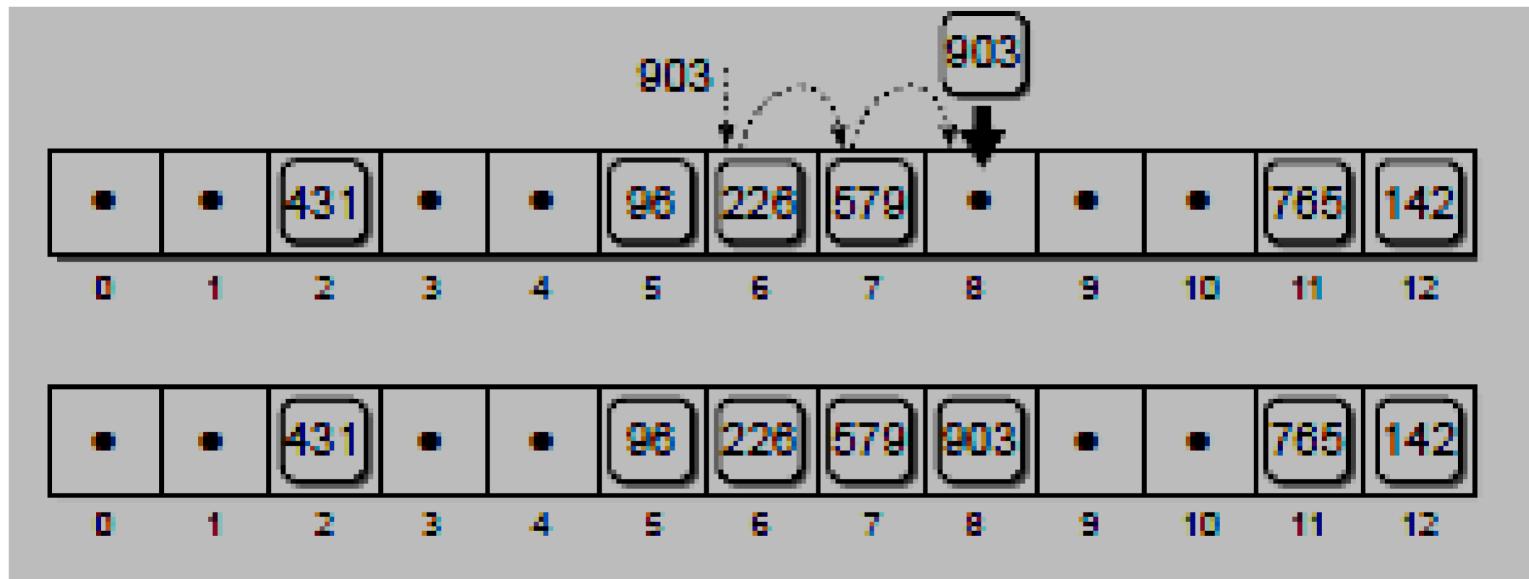
penempatan kunci sesuai hasil fungsi hash

- Persoalan yang bisa timbul pada fungsi hash sederhana ini adalah kemungkinan collision (tabrakan), dimana lebih dari satu kunci memberikan hasil hash yang sama.
- Salah satu solusi dari masalah collision adalah “linear probing”, yaitu bilamana terjadi collision maka key yang belakangan akan ditempatkan pada lokasi kosong yang ada disebelah kanan dari key pertama.

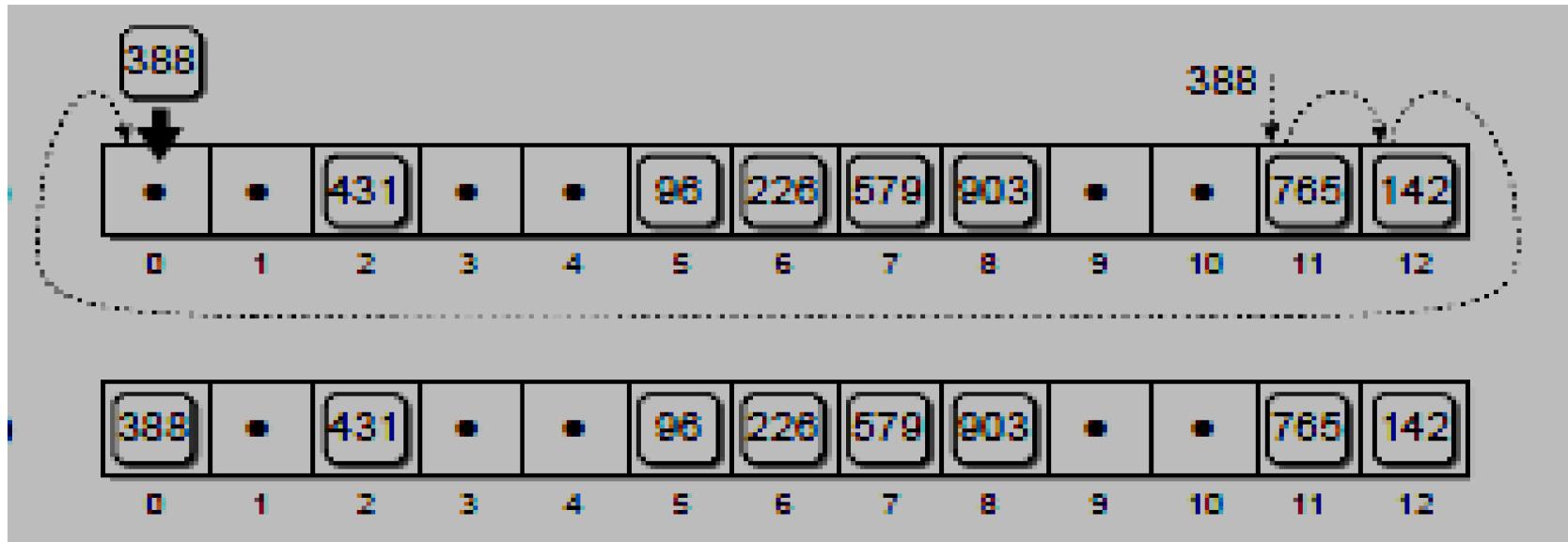
- Sebagai contoh andaikan key 226 akan dimasukkan, nilai $h(226) = 226 \% 13 = 5$ maka key 226 bisa ditempatkan pada posisi 6, seperti pada gambar berikut



- Apa yang terjadi apabila kunci 903 juga akan dimasukkan, dimana $h(903)$ adalah 6, dan di posisi 6 sudah ada 226, maka linear-probing akan mencari ke-kanan, ada 579, kekanan lagi kosong sehingga 903 masuk di posisi 8.

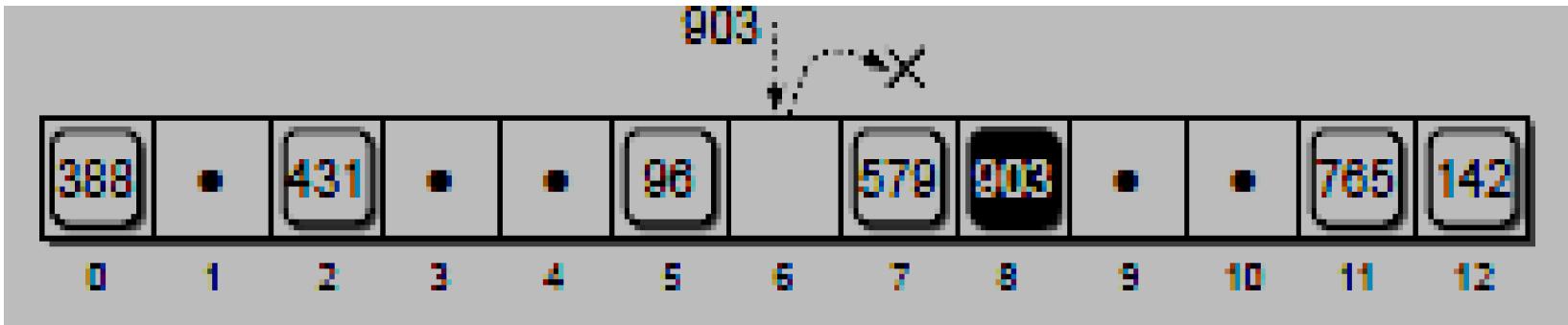


- Pencarian posisi ini dapat dilakukan seterusnya secara melingkar, selama jumlah key tidak > dari M.

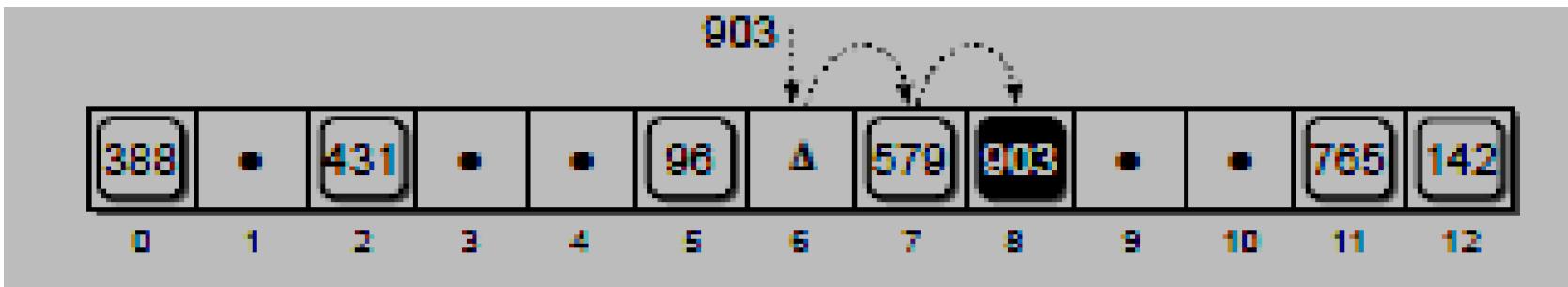


- Selanjutnya bagaimana proses penghapusan (delete) suatu kunci dari hash-table dilaksanakan.
- Pertama-tama kunci di-hash untuk menemukan lokasi, ketika lokasi ditemukan maka kunci dibandingkan, bila tidak sama maka coba periksa node dibelakangnya, kemungkinan merupakan hasil linear probing.
- Katakanlah node 226 mau di-delete, hash(226) menghasilkan 5, namun diposisi 5 ada node 96, maka periksa node berikutnya, ternyata sama dengan 226, jadi node ini apakah langsung dikosongkan?

- Pada saat 903 akan dimasukkan, ternyata hash(903) adalah 6, namun diposisi 6 ada node 226, sehingga harus dicarikan tempat yaitu diposisi 8.
- Apabila node 226 langsung dikosongkan, maka ketika 903 dicari akan terjadi error, karena posisi 6 kosong.
- Maka posisi yang ditinggalkan oleh 226 harus diberi tanda khusus yang merupakan indikator bahwa node ini sebenarnya memiliki data.
- Ketika indikator ditemukan maka elemen yang dicari bisa diperiksa pada node yang berada di sebelah kanan.



Posisi 6 tidak boleh dikosongkan



Posisi 6 di-isi indikator delta

- Apabila key ditambahkan terus ke dalam tabel, maka pada akhirnya akan terjadi cluster, karena probabilitas hasil hash memberi nilai yang sama untuk kunci berbeda semakin tinggi. Salah satu teknik untuk mengurangi terjadinya cluster adalah dengan mengubah fungsi hash dan teknik probing.
- Sebagai contoh fungsi hash dapat dimodifikasi menjadi:

$$\text{posisi} = (\text{home} + i) \% M$$

- dengan $i = 1, 2, \dots, (M-1)$, dan $\text{home} = \text{posisi awal}$.

- Maka hasil hash (765, 431, 96, 142, 579, 226, 903, 388) memberikan nilai sebagai berikut:

$$h(765) = 11$$

$$h(431) = 2$$

$$h(96) = 5$$

$$h(142) = 12$$

$$h(579) = 7$$

$$h(226) = 5 \rightarrow 6$$

$$h(903) = 6 \rightarrow 7 \rightarrow 8$$

$$h(388) = 11 \rightarrow 12 \rightarrow 0$$

- Masih terjadi collision tiga kali.

- Modifikasi selanjutnya sebagai berikut:

$$\text{posisi} = (\text{home} + i * c) \% M$$

- $M=13$, c adalah tetapan misalnya $c=3$, $i = 1, 2, \dots 12$, akan memberikan hasil hash yang lebih sedikit collision.
- $h(765) = 11 \quad h(579) = 7$
- $h(431) = 2 \quad h(226) = 5 \rightarrow 8$
- $h(96) = 5 \quad h(903) = 6$
- $h(142) = 12 \quad h(388) = 11 \rightarrow 1$



hasil modifikasi fungsi hash

- Suatu fungsi hash yang lain disebut “**quadratic probing**” bentuknya sebagai berikut:

$$\text{posisi} = (\text{home} + i^2) \% M$$

- Memakai dataset yang sama quadratic probing ini menghasilkan posisi sebagai berikut.
- $h(765) = 11 \quad h(579) = 7$
- $h(431) = 2 \quad h(226) = 5 \rightarrow 6$
- $h(96) = 5 \quad h(903) = 6 \rightarrow 7 \rightarrow 10$
- $h(142) = 12 \quad h(388) = 11 \rightarrow 12 \rightarrow 2 \rightarrow 7 \rightarrow 1$

- Fungsi hash yang cukup sering digunakan adalah “double hashing”, dengan bentuk sebagai berikut.

$$\text{posisi} = (\text{home} + i * \text{hp}(\text{key})) \% M$$

$$\text{hp}(\text{key}) = 1 + \text{key} \% P$$

$$P : \text{konstan} < M$$

- Sebagai contoh untuk $P=8$ maka hasil hash memberikan:

- $h(765) = 11 \quad h(579) = 7$
- $h(431) = 2 \quad h(226) = 5 \rightarrow 8$
- $h(96) = 5 \quad h(903) = 6$
- $h(142) = 12 \quad h(388) = 11 \rightarrow 3$

•	388	431	•	•	96	226	579	•	•	903	765	142
0	1	2	3	4	5	6	7	8	9	10	11	12

hasil dari quadratic probing

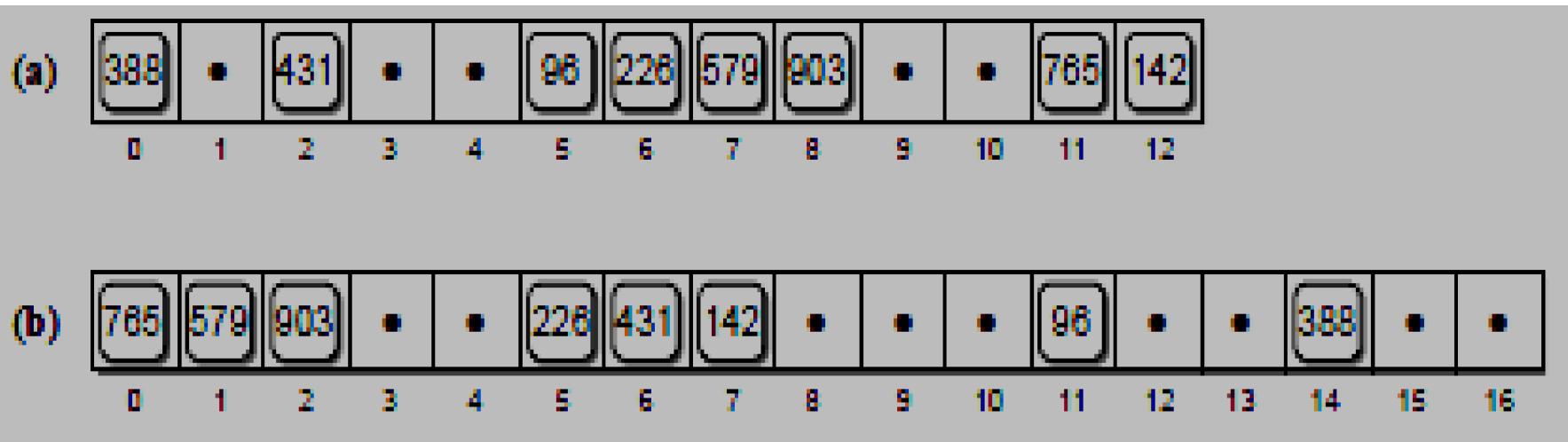
•	•	431	388	•	96	903	579	226	•	•	765	142
0	1	2	3	4	5	6	7	8	9	10	11	12

hasil dari double-hashing

- Persoalan utama dalam penempatan kunci ke suatu tabel hash adalah tidak ada informasi yang mendahului seberapa banyak data yang akan dimasukkan. Tindakan praktis yang sering dilakukan adalah pembuatan tabel hash dimulai dengan suatu ukuran tertentu, kemudian diekspansi ketika kebutuhan meningkat.
-
- Pemekaran tabel hash memberi konsekwensi penempatan kembali semua kunci yang sudah ada melalui tindakan rehash. Sebagai contoh ketika $M=13$ diubah menjadi $M=17$ maka akan diperoleh tabel yang lebih luas dan lebih sedikit collision.

$$\begin{aligned}
 h(765) &= 0 \\
 h(431) &= 6 \\
 h(96) &= 11 \\
 h(142) &= 6 \rightarrow 7
 \end{aligned}$$

$$\begin{aligned}
 h(579) &= 1 \\
 h(226) &= 5 \\
 h(903) &= 2 \\
 h(388) &= 14
 \end{aligned}$$



hasil dari proses pemekaran (rehash)

HashMap ADT

- **HashMap()** : mencipta suatu HashMap kosong
- **len()** : menhitung jumlah item dalam map
- **contains(k)** : memeriksa apakah k ada dalam map
- **add(k, v)** : menambahkan item baru (k,v) atau merubah value dari item menjadi v
- **valueOf(k)** : menampilkan value dari kunci k
- **remove(k)** : membuang nilai dari kunci k
- **findSlot(k, b)** : mencari index dari kunci k, bila b=True berarti mencari slot untuk data baru

Implementasi HashMap

```
# Implementasi dari HashMap ADT memakai closed hashing
# dan memeriksa ketersediaan slot memakai double hashing.
#HashMap.py : @Suarga

from Array import Array
from Empty import Empty

# Storage class for holding the key/value pairs.
class _MapEntry :
    def __init__( self, key, value ) :
        self.key = key
        self.value = value

global UNUSED, EMPTY
UNUSED = None

class HashMap :
    # menetapkan nilai awal status dari tabel Hash.
    #UNUSED = None
    EMPTY = _MapEntry( None, None )
```

```
# Creates an empty map instance.
def __init__( self ):
    self._table = Array( 9 )
    self._count = 0
    self._maxCount = len(self._table) - len(self._table) // 3
    EMPTY = _MapEntry( None, None )

# Returns the number of entries in the map.
def __len__( self ):
    return self._count

# Determines if the map contains the given key.
def __contains__( self, key ):
    slot = self._findSlot( key, False )
    return slot is not None
```

```
# Adds a new entry to the map if the key does not exist. Otherwise, the
# new value replaces the current value associated with the key.
def add( self, key, value ) :
    #print('inserting key : ', key, ' with value = ',value)
    slot = self._findSlot(key, False)
    if (self._count > 0) and (self._table[slot] is not None) :
        #if key in self :
        #    slot = self._findSlot( key, False )
        self._table[slot].value = value
        return False
    else :
        slot = self._findSlot( key, True )
        #print('slot = ',slot)
        self._table[slot] = _MapEntry( key, value )
        self._count += 1
        if self._count == self._maxCount :
            self._rehash()
    return True
```

```
# Returns the value associated with the key.
def valueOf( self, key ):
    slot = self._findSlot( key, False )
    assert slot is not None, "Invalid map key."
    if self._table[slot] is not None:
        x = self._table[slot].value
        #print('Display value of key ', key, ' is: ', x)
        return x

# Removes the entry associated with the key.
def remove( self, key ):
    print('Removing key : ', key)
    slot = self._findSlot( key, False )
    assert slot is not None, "Invalid map key."
    if self._table[slot] is not None:
        x = self._table[slot].value
        self._table[slot].value = None
        return x

# Returns an iterator for traversing the keys in the map.
def __iter__( self ):
    return iter( self._count )
```

```
# Finds the slot containing the key or where the key can be added.  
# forInsert indicates if the search is for an insertion, which locates  
# the slot into which the new key can be added.  
def _findSlot( self, key, forInsert ):  
    # Compute the home slot and the step size.  
    slot = self._hash1( key )  
    step = self._hash2( key )  
  
    # Probe for the key.  
    M = len(self._table)  
    while self._table[slot] is not None :  
        if forInsert and \  
            (self._table[slot] is None or self._table[slot] is None) :  
                #print('1. key ',key,' slot ',slot)  
                return slot  
        elif not forInsert and \  
            (self._table[slot] is not None and self._table[slot].key == key) :  
                #print('2. key ',key,' slot ',slot)  
                return slot  
        else :  
            slot = (slot + step) % M  
            #print('3. key ',key,' slot ',slot)  
    return slot
```

```
# The main hash function for mapping keys to table entries.  
def _hash1( self, key ):  
    return abs( hash(key) ) % len(self._table)  
  
# The second hash function used with double hashing probes.  
def _hash2( self, key ):  
    return 1 + abs( hash(key) ) % (len(self._table) - 2)
```

Test HashMap

```
#HashMap_main.py, @Suarga
#program menguji struktur HashMap
from HashMap import *
from HashMap import _MapEntry

def daftarIsi(HM):
    #mencetak isi H
    n = HM.__len__()
    for key in range(n):
        color = HM.valueOf(key)
        print("%d => %s" % (key, color))

def main():
    H=HashMap()
    print("Memasukkan 5 warna: Hijau,Kuning,Merah,Putih, Biru")
    warna = ['Hijau', 'Kuning', 'Merah', 'Putih', 'Biru']
    for key in range(5):
        H.add(key, warna[key])
```

```
daftarIsi(H)
```

```
print("menambah warna Coklat")
H.add(5, "Coklat")
```

```
daftarIsi(H)
```

```
print("menghapus isi slot 2")
v = H.remove(2)
print(v, ' telah dihapus')
```

```
daftarIsi(H)
```

```
main()
```

Hasil Test

```
===== RESTART: D:\USER\Python\HashMap.py =====
Masukkan 5 warna: Hijau,Kuning,Merah,Putih, Biru
0 => Hijau
1 => Kuning
2 => Merah
3 => Putih
4 => Biru
menambah warna Coklat
0 => Hijau
1 => Kuning
2 => Merah
3 => Putih
4 => Biru
5 => Coklat
menghapus isi slot 2
Removing key : 2
Merah telah dihapus
0 => Hijau
1 => Kuning
2 => None
3 => Putih
4 => Biru
5 => Coklat
```

Aplikasi HashMap: Histogram

- Pada bagian ini akan diuraikan suatu aplikasi dari HashMap dengan membangun aplikasi untuk histogram, misalnya untuk distribusi nilai. Adapun fungsi untuk kebutuhan ADT antara lain sebagai berikut:
 1. **Histogram()** : inisialisasi objek histogram, dengan cacah awal = 0
 2. **getCount()** : menghitung frekuensi dari data per-kategori
 3. **incCount()** : menambah cacahan data perkategori
 4. **totalCount()** : memberikan jumlah cacahan per-kategori
 5. **iterator()** : menciptakan iterator untuk kategori histogram

maphist.py

```
# Implementasi Histogram ADT memakai Hash Map.
# maphist.py
from HashMap import HashMap

class Histogram :
    # Creates a histogram containing the given categories.
    def __init__( self, catSeq ):
        self._freqCounts = HashMap()
        for cat in catSeq :
            self._freqCounts.add( cat, 0 )

    # Returns the frequency count for the given category.
    def getCount( self, category ) :
        assert category in self._freqCounts, "Invalid histogram category."
        return self._freqCounts.valueOf( category )
```

```
# Increments the counter for the given category.
def incCount( self, category ):
    assert category in self._freqCounts, "Invalid histogram category."
    value = self._freqCounts.valueOf( category )
    self._freqCounts.add( category, value + 1 )

# Returns the sum of the frequency counts.
def totalCount( self ):
    total = 0
    for cat in self._freqCounts :
        total += self._freqCounts.valueOf( cat )
    return total

# Returns an iterator for traversing the categories.
def __iter__( self ):
    return iter( self._freqCounts )
```

program histograf.py

```
# Prints a histogram for a distribution of letter grades computed
# from a collection of numeric grades extracted from a text file.
# Reference : Necaise.R.D. "Data Structures and Algorithms in Python"

from maphist import Histogram

# Determines the letter grade for the given numeric value.

def letterGrade( grade ):
    if grade >= 86 :
        return 'A'
    elif grade >= 76 :
        return 'B'
    elif grade >= 65 :
        return 'C'
    elif grade >= 55 :
        return 'D'
    else :
        return 'E'
```

```
# Prints the histogram as a horizontal bar chart.
def printChart( gradeHist ):
    print( "          Distribusi Nilai" )
    # Print the body of the chart.
    letterGrades = ( 'A', 'B', 'C', 'D', 'E' )
    for letter in letterGrades :
        print( " |" )
        print( letter + " +", end = "" )
        freq = gradeHist.getCount( letter )
        print( '*' * freq, ' ', freq )

    # Print the x-axis.
    print( " |" )
    print( " +---+---+---+---+---+---+---+---+---" )
    print( " 0   5   10   15   20   25   30   35" )
```

```
def main():
    # Create a Histogram instance for computing the frequencies.
    gradeHist = Histogram( "ABCDE" )

    # Open the text file containing the grades.
    gradeFile = open('fileNilai.txt', "r")

    # Extract the grades and increment the appropriate counter.
    for line in gradeFile :
        grade = int(line)
        gradeHist.incCount( letterGrade(grade) )

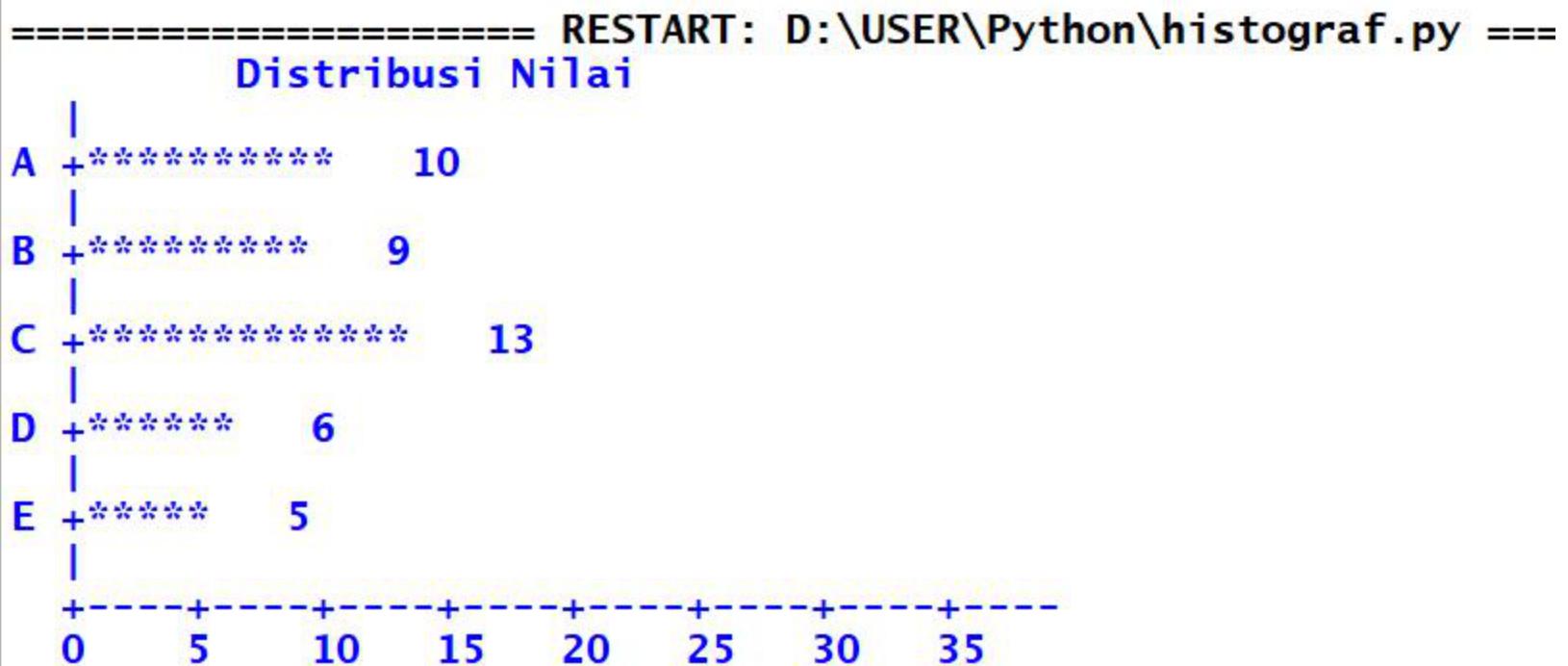
    # Print the histogram chart.
    printChart( gradeHist )

# Calls the main routine.
main()
```

fileNilai.txt

77	80	94	82
89	77	64	86
53	73	79	70
95	73	97	45
68	93	59	100
86	85	69	62
91	83	61	72
89	67	80	82
60	75	73	62
70	71	70	52
			42
			33
			67

Hasil run



>>>