**Nama : Abdillah Mufki Auzan Mubin**

**NPM : 40621100046**

**TUGAS ALGORITMA DAN PEMROGRAMAN II Pertemuan 9**

**1. favorites\_list.py**

|  |
| --- |
| **Python** |
| from .positional\_list import PositionalList  class FavoritesList:  """List of elements ordered from most frequently accessed to least."""  #------------------------------ nested \_Item class ------------------------------  class \_Item:  \_\_slots\_\_ = '\_value', '\_count' # streamline memory usage  def \_\_init\_\_(self, e):  self.\_value = e # the user's element  self.\_count = 0 # access count initially zero  #------------------------------- nonpublic utilities -------------------------------  def \_find\_position(self, e):  """Search for element e and return its Position (or None if not found)."""  walk = self.\_data.first()  while walk is not None and walk.element().\_value != e:  walk = self.\_data.after(walk)  return walk  def \_move\_up(self, p):  """Move item at Position p earlier in the list based on access count."""  if p != self.\_data.first():  cnt = p.element().\_count  walk = self.\_data.before(p)  if cnt > walk.element().\_count: # must shift forward  while (walk != self.\_data.first() and  cnt > self.\_data.before(walk).element().\_count):  walk = self.\_data.before(walk)  self.\_data.add\_before(walk, self.\_data.delete(p)) # delete/reinsert  #------------------------------- public methods -------------------------------  def \_\_init\_\_(self):  """Create an empty list of favorites."""  self.\_data = PositionalList() # will be list of \_Item instances  def \_\_len\_\_(self):  """Return number of entries on favorites list."""  return len(self.\_data)  def is\_empty(self):  """Return True if list is empty."""  return len(self.\_data) == 0  def access(self, e):  """Access element e, thereby increasing its access count."""  p = self.\_find\_position(e) # try to locate existing element  if p is None:  p = self.\_data.add\_last(self.\_Item(e)) # if new, place at end  p.element().\_count += 1 # always increment count  self.\_move\_up(p)  def remove(self, e):  """Remove element e from the list of favorites."""  p = self.\_find\_position(e) # try to locate existing element  if p is not None:  self.\_data.delete(p) # delete, if found  def top(self, k):  """Generate sequence of top k elements in terms of access count."""  if not 1 <= k <= len(self):  raise ValueError('Illegal value for k')  walk = self.\_data.first()  for j in range(k):  item = walk.element() # element of list is \_Item  yield item.\_value # report user's element  walk = self.\_data.after(walk)  def \_\_repr\_\_(self):  """Create string representation of the favorites list."""  return ', '.join('({0}:{1})'.format(i.\_value, i.\_count) for i in self.\_data)  if \_\_name\_\_ == '\_\_main\_\_':  fav = FavoritesList()  for c in 'hello. this is a test of mtf': # well, not the mtf part...  fav.access(c)  k = min(5, len(fav))  print('Top {0}) {1:25} {2}'.format(k, [x for x in fav.top(k)], fav)) |

|  |
| --- |
| **Python** |
| from .favorites\_list import FavoritesList  from .positional\_list import PositionalList  class FavoritesListMTF(FavoritesList):  """List of elements ordered with move-to-front heuristic."""  # we override \_move\_up to provide move-to-front semantics  def \_move\_up(self, p):  """Move accessed item at Position p to front of list."""  if p != self.\_data.first():  self.\_data.add\_first(self.\_data.delete(p)) # delete/reinsert  # we override top because list is no longer sorted  def top(self, k):  """Generate sequence of top k elements in terms of access count."""  if not 1 <= k <= len(self):  raise ValueError('Illegal value for k')  # we begin by making a copy of the original list  temp = PositionalList()  for item in self.\_data: # positional lists support iteration  temp.add\_last(item)  # we repeatedly find, report, and remove element with largest count  for j in range(k):  # find and report next highest from temp  highPos = temp.first()  walk = temp.after(highPos)  while walk is not None:  if walk.element().\_count > highPos.element().\_count:  highPos = walk  walk = temp.after(walk)  # we have found the element with highest count  yield highPos.element().\_value # report element to user  temp.delete(highPos) # remove from temp list  if \_\_name\_\_ == '\_\_main\_\_':  fav = FavoritesListMTF()  for c in 'hello. this is a test of mtf':  fav.access(c)  k = min(5, len(fav))  print('Top {0}) {1:25} {2}'.format(k, [x for x in fav.top(k)], fav)) |