CIS 210

THIS WEEK:

• Binary Search
• A very short introduction to recursion

The process of preparing programs for a digital computer is especially attractive, not only because it can be economically and scientifically rewarding, but also because it can be an aesthetic experience much like composing poetry or music.

*Donald Knuth, The Art of Computer Programming, vol. 1*
Which of the following operations is out of order in the k-means cluster algorithm:

1. Decide how many clusters (k)
2. Measure the distance from each item to each cluster
3. Determine the centroids
4. Assign each data item to a cluster

(a) 1.  (b) 2.  (c) 3.  (d) 4.  (e) none of the above.
Which of line in the following results in an error when the function is called?

1. `def doSomethingFile(f):
2.   """(str) -> dict
3.   ""

4. `with open(f, 'r') as eqf:
5.   my_dict = {}

6.   key = 0
7.   `for line in eqf:
8.     line = line.strip().split(' ')
9.     data_point = float(line[1])
10.    key += 1
11.   my_dict[key] = data_point
12.   return my_dict

13. doSomethingFile('my_txt_file.txt')

(a) 7.  (b) 8.  (c) 9.  (d) 11.  (e) none of the above.
Finding an item in a sequence

Search for item n in a sequence, seq.
Return True if n is a member, else False.

(like Python in operator)
Finding an item in a sequence
Search for item n in a sequence, seq.
Return True if n is a member, else False.

Examples

'hello', 'i'
False

(4, 8, 2, 6), 8
True
Finding an item in a sequence
Search for item n in a sequence, seq.
Return True if n is a member, else False.

Examples

'hello', 'i'            (4, (8, 2), 6), (8, 2)            (), 99
False                   True                                False

(4, 8, 2, 6), 8        (4, (8, 2), 6), 2                (99), 99
True                   ?? False                             True

Test-driven design
Finding an item in a sequence

Search for item n in a sequence, seq.
Return True if n is a member, else False.

Examples

'hello', 'i' (4, 8, 2, 6), 8 (4, (8, 2), 6), (8, 2) () 99
False True True False

check each item in the sequence, compare to target (t)
if item is the target, done (True)
if item is not the target, keep looking
at end of sequence, done (False)
Finding an item in a sequence

def isIn(seq, t):
    """ (??, ??) -> ??
    
    Search for item n in a sequence, seq.
    Return True if n is a member, else False.
    
    >>> isIn('hello', 'i')
    False
    >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    True
    ""

    check each item in the sequence, compare to target (t)
    if item is the target, done (True)
    if item is not the target, keep looking
    at end of sequence, done (False)
    return #Boolean value
Finding an item in a sequence

def isIn(seq, t):
    """ (sequence, item) -> boolean ""
    Search for item n in a sequence, seq.
    Return True if n is a member, else False.

    >>> isIn('hello', 'i')
    False
    >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    True

    check each item in the sequence, compare to target (t) - for
    if item is the target, done (True) – conditional, equality, boolean
    if item is not the target, keep looking – for takes care of this
    at end of sequence, done (False) – for takes care of this, boolean
    return #Boolean value
Finding an item in a sequence

def isIn(seq, t):
    """ (sequence, item) -> boolean
    Search for item n in a sequence, seq.
    Return True if n is a member, else False.
    >>> isIn('hello', 'i')
    False
    >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    True
    """
    for item in seq:
        if item == t:
            found = True  # developing code
        else:
            found = False
    return found
Finding an item in a sequence

def isIn(seq, t):
    ''' (sequence, item) -> boolean

    Search for item n in a sequence, seq.
    Return True if n is a member, else False.
    
    >>> isIn('hello', 'i')
    False
    >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    True
    '''

    for item in seq:
        if item == t:
            found = True
        else:
            found = False  # developing code

    return found
Finding an item in a sequence

def isIn(seq, t):
    ''' (sequence, item) -> boolean

    Search for item n in a sequence, seq.
    Return True if n is a member, else False.
    
    >>> isIn('hello', 'i')
    False
    >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    True
    '''

    found = False
    for item in seq:
        if item == t:
            # developing code
            found = True
    return found
Finding an item in a sequence

def isIn(seq, t):
    """(sequence, item) -> boolean

    Search for item n in a sequence, seq.
    Return True if n is a member, else False.

    >>> isIn('hello', 'i')
    False
    >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    True
    """

    for item in seq:
        if item == t:  # developing code
            return True

    return False
Finding an item in a sequence

def isIn(seq, t):
    """ (sequence, item) -> boolean

    Search for item n in a sequence, seq.
    Return True if n is a member, else False.
"

>>> isIn('hello', 'i')
False
>>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
True
'`

for item in seq:  # exercise: what is needed to
    if item == t:   # return index of found item?
        found = True  # or all indices? or last index?
    return False
Finding an item in a sequence

def isIn(seq, t):
    """(sequence, item) -> boolean

    Search for item n in a sorted sequence, seq.
    Return True if n is a member, else False.

    >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    True
    """

    for item in seq:
        if item == t:
            return True

    return False
Finding an item in a sequence

def isIn(seq, t):
    """(sequence, item) -> boolean

    Search for item n in a sorted sequence, seq.
    Return True if n is a member, else False.

>>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
False
""

    for item in seq:
        if item == t:
            return True  # developing code
        if item > t:
            return False

    return False
Binary Search: efficient search technique, as long as the list is already sorted.

Each step divides the remaining data into equal parts and discards one part:

If remaining part is empty, then done (not found).

Go to mid-point of remaining part and compare to target.

If mid-point is the target, then done (found).

Otherwise, keep the part of the list where n could be, and search that. (Discard the rest.)
Each step divides the remaining data into equal parts and discards one part:

If remaining part is empty, then done (not found).

Go to mid-point of remaining part and compare to target.

If mid-point is the target, then done (found).

Otherwise, keep the part of the list where n could be, and search that. (Discard the rest.)

For example,

nums = (1, 3, 4, 6, 8, 9, 11)
target value: 4
For example,

```
nums = (1, 3, 4, 6, 8, 9, 11)
target value: 4
```

compare x to 6; x is smaller, so repeat with (1, 3, 4)

compare x to 3; x is larger, so repeat with (4)

compare x to 4; x == 4, so True is returned
For example,

`nums = (1, 3, 4, 6, 8, 9, 11, 13, 15, 25, 99, 100, 102)`
target value: 42
CIS 210

For example,

```
nums = (1, 3, 4, 6, 8, 9, 11, 13, 15, 25, 99, 100, 102)
target value: 42
```

compare x to 11; x is larger, so repeat with (13, 15, 25, 99, 100, 102) (is 25 or 99 the midpoint?)

compare x to 99; x is smaller, so repeat with (13, 15, 25)

compare x to 15; x is larger, so repeat with (25)

compare x to 25; not equal; done; False is returned
CIS 210

If \( n \) is number of items in sequence:

Sequential search \( \mathcal{O}(n) \)

Binary search \( \mathcal{O}(\log n) \)

When \( n \) is 150 ...

When \( n \) is around 1000 ...

When \( n \) is around 1,000,000 ... 1,000,000,000 ...
CIS 210

Sequential search \(O(n)\)

Binary search \(O(\log n)\)

When \(n\) is 150 ... \(150\) v. 8

When \(n\) is around 1000 ... \(1000\) v. 10

When \(n\) is around 1,000,000 ... \(1,000,000,000\) ... \(1,000,000\) v. 20 \(1,000,000,000,000\) v. 30
Should we sort the argument to the isIn function so we can always use binary search?
Should we sort the argument to the isIn function so we can always use binary search?

No. Good sort algorithms are $O(n \log n) > O(n)$ linear search.
CIS 210

✓ Binary Search
  • A very short introduction to recursion
Recall: Factorials

0! = 1
n! = n * (n-1)!
Factorials have a **recursive (inductive)** definition – elements in a set are defined in terms of other elements in the set

\[ 0! = 1 \]
\[ n! = n \times (n-1)! \]
\[ = n \times (n-1) \times (n-2)! \]
\[ = n \times (n-1) \times (n-2) \times (n-3)! \]
\[ = n \times (n-1) \times (n-2) \times (n-3)! \ldots \times 0! \]
Factorials have a **recursive (inductive)** definition — elements in a set are defined in terms of other (more basic) elements in the set.

\[
0! = 1 \\
n! = n \cdot (n-1)! \\
4! = ?? \\
4! = 4 \cdot 3! \\
4! = 4 \cdot (3 \cdot 2!) \\
4! = 4 \cdot (3 \cdot (2 \cdot 1!)) \\
4! = 4 \cdot (3 \cdot (2 \cdot (1 \cdot 1!))) \\
4! = 4 \cdot (3 \cdot (2 \cdot (1 \cdot 1)))
\]
Factorials have a recursive (inductive) definition – elements in a set are defined in terms of other (more basic) elements in the set

0! = 1
n! = n * (n-1)!

4! = ??
4! = 4 * 3!
4! = 4 * (3 * 2!)
4! = 4 * (3 * (2 * 1!))
4! = 4 * (3 * (2 * (1 * 0!)))
4! = 4 * (3 * (2 * (1 * 1)))

4! = 24
4! = 4 * 6
4! = 4 * (3 * 2)
4! = 4 * (3 * (2 * 1))
4! = 4 * (3 * (2 * (1)))
CIS 210

Factorials have a recursive (inductive) definition – elements in a set are defined in terms of other (more basic) elements in the set

\[
0! = 1 \quad \# \text{ base case} \\
n! = n \times (n-1)! \quad \# \text{ recursive (inductive) rule}
\]

\[
4! = 4 \times 3! = 4 \times 6 = 24 \\
3! = 3 \times 2! = 3 \times 2 = 6 \\
2! = 2 \times 1! = 2 \times 1 = 2 \\
1! = 1 \times 0! = 1 \times 1 = 1
\]