Module02_Day04_Searching

December 16, 2022

0.0.1 Set Operations

[ ]:
```
s = {1,2,3,4,5,7,6}
s
```

[ ]:
```
{1, 2, 3, 4, 5, 6, 7}
```

[ ]:
```
s.discard(8) # Does not give error
```

[ ]:
```
s.remove(8)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19988\2473678413.py in <module>
----> 1 s.remove(8)

KeyError: 8
```

[ ]:
```
for ele in s:
    print(ele)
```

```
1
2
3
4
5
6
7
```

[ ]:
```
# Only Set & Dictionaries are unordered in nature, Even it is sorted this␣
 ↪doesn'tt mean it is ordered
```

[ ]:
```
# Dict Keys -> only immutable
# Same with Sets -> only immutable
```

Hashable

[ ]:
```
hash("Sccaler")
```

[ ]:
```
8778957955003078891
```

[ ]:
```
hash([1,2,3]) # This means list is mutable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19988\2019702830.py in <module>
----> 1 hash([1,2,3]) # This means list is mutable

TypeError: unhashable type: 'list'
```

[ ]:
```
hash ({1,2,4,8,8})
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19988\815524556.py in <module>
----> 1 hash ({1,2,4,8,8})

TypeError: unhashable type: 'set'
```

[ ]:
```
hash(3)
```

[ ]:
```
3
```

0.0.2 Set operations

[ ]:
```
s1 = {1,2,4,4,1}
s2 = {5,4,2,6,7,7}

s1, s2
```

[ ]:
```
({1, 2, 4}, {2, 4, 5, 6, 7})
```

[ ]:
```
# Common elements
# s1 & s2
s1.intersection(s2)
```

[ ]:
```
{2, 4}
```

[ ]:
```
# Take all the elements and make 1 set
# s1 | s2
s1.union(s2)
```

[ ]:
```
{1, 2, 4, 5, 6, 7}
```

[ ]:
```
# only only which is not common
# s1 -s2, s2 - s1

s1.difference(s2), s2.difference(s1)
```

[ ]:
```
({1}, {5, 6, 7})
```

[ ]:
```
s1 == s2
```

[ ]:
```
False
```

[ ]:
```
s1 = {1,2,3}
s2 = {3,3,2,1,2,3,2,1} # will remove other elements

s1 == s2
```

[ ]:
```
True
```

[ ]:
```
s1 is s2
```

[ ]:
```
False
```

[ ]:
```
s1.issubset(s2)
```

[ ]:
```
True
```

[ ]:
```
# Returns element which are not common in any of it
s1 = {1,2,3,4}
s2 = {1,2,3,4,5,6}
s1.symmetric_difference(s2)
```

[ ]:
```
{5, 6}
```

0.0.3 DSA

Algorithm

- Set of instructions in a specific order
  - To solve problem efficiently
  - increase revenue as a Data Scientist -> Evaluate by Buisness Matrix

0.1 Searching

0.1.1 Linear Search

- Order not important
- Best case O(1), Worst Case O(n)
- Also called as Brute Force

[ ]:
```
def linearSearch(ids, target, start=0, end=len(ids)):
    count = ids.count(target)
    for i in range(start,end):
        if ids[i] == target:
            return f'Found at index: {i} & Count of {target}: {count}'
    return "Not found!"

ids = [4,7,8,2,1,7,9,3,7]
linearSearch(ids, 7, 2)
```

[ ]:
```
'Found at index: 5 & Count of 7: 3'
```

0.1.2 Binary Search

- Bi means 2
- One assumption is It works only on sorted/monotonic space
- Hence, elements are arranged in order
- Monotonic: Mono means one, All the elements are in increasing or decreasing order
- Best case O(1), worst Case O(log(n))

[ ]:
```
lst1 = [1,5,7,9,1] # Binary search possible because it is monotonic increasing
lst2 = [5,3,1,-2] # Possible because it is monotonic decreasing
```

[ ]:
```
ids = [2,3,5,7,9,11,13,15]
ids = [12, 23, 35, 42, 56]

ids = [15,13,11,9,7,5,3,2]

def binarySearch(ids, target):
    start=0
    end=len(ids)-1

    while(start <= end):

        mid = (start + end)//2
        if ids[mid] == target:
            return mid

        elif target < ids[mid]:
            # Target will be on left, Discard right elements
            end = mid-1

        else:
            # Target will be on right, Discard left elements
            start = mid +1

    return -1
```

```
binarySearch(ids,15)
```

[ ]: -1

[ ]:
```
ids = [2,3,5,7,9,11,13,15]
ids = [12, 23, 35, 42, 56]

ids = [15,13,11,9,7,5,3,2]

def binarySearch(ids, target):
    start=0
    end=len(ids)-1
    order = 1

    if ids[0] > ids[-1]:
        start,end = end,start
        order = -1

    while(start != end+order):

        mid = (start + end)//2
        if ids[mid] == target:
            return mid

        elif target < ids[mid]:
            end = mid - order

        else:
            start = mid + order

    return -1


binarySearch(ids,15)
```

[ ]: 0

0.1.3 SQRT

[ ]:
```
def sqrt(n):
    for i in range(n//2 + 1):
        if i * i ==n:
            return i
sqrt(50)
```

[ ]:
```python
def binaryPerfSqrt(n):
    if n == 1: return 1

    start = 0
    end = n//2

    while(start <= end):
        mid = (start+end)//2
        square = mid*mid
        if square == n:
            return mid
        elif square < n:
            start = mid + 1
        else:
            end = mid - 1
    return "Not Found"

binaryPerfSqrt(1)
```

[ ]:
```
1
```

[ ]:
```python
import numpy as np
```

[ ]:
```python
def eucDistSqrt(n):
    if n == 1: return 1.0
    if n == 0: return 0.0

    precision = np.float128(0.00000000000000001)
    start = np.float128(0)
    end = np.float128(n)

    if n > 0 and n < 1:
        start = n
        end = n+1

    while(abs(start-end) >= precision):
        mid = np.float128((start+end)/2)
        square = np.float128(mid*mid)

        if abs(square-n) <= precision:
            return mid

        elif square < n:
            start = np.float128(mid)

        else:
            end = np.float128(mid)
```

```
n = 7
eucDistSqrt(n), (n)**0.5
```

[ ]: (2.6457513110645905895, 2.6457513110645907)

0.1.4 HW

[ ]:
```
string='hello'
char='o'
for i in range(0,len(string)):
    if string[i] == char:
        print('Found the required character at position: ', i)
```

Found the required character at position: 4

[ ]:
```
def sentinelSearch(ar,target,n):
    last = ar[n-1]
    ar[n-1] = target
    i = 0
    while ar[i]!=target:
        i+=1
    ar[n-1] = last
    if (i<n-1) or target==ar[n-1]:
        return i
    else:
        return -1

sentinelSearch([1,2,3,4,5],5,5)
```

[ ]: 4

[ ]:
```
s = [1,2,3] + [1,3,4]
```

[ ]:
```
s.pop(0)
```

[ ]: 1

[ ]:
```
s
```

[ ]: [2, 3, 1, 3, 4]