CS566Lab4-2024

September 25, 2024

0.1 Lab 4. Introduction to algorithms

This is fourth Lab for CS 566. This problem was given in lecture.

0.1.1 Task 0. Implement Heap (4 points)

[147]:
```
import heapq
nums = [x for x in [5,4,3,2,10]]
heapq.heapify(nums)
print([x for x in nums])
```

```
[2, 4, 3, 5, 10]
```

[148]:
```
# Implement Data Structure called Heap
# Please implement the function create_heap() which takes as an input list of␣
 ↪numbers and returns the Heap
# Please implement the function heapify(), which takes as an input list of␣
 ↪numbers, length and index and performs heapify

def heapify(nums, n, i):
    # implement heapifyz
    left = 2*i+1
    right = 2*i+2
    largest = i
    if left < n and nums[left]>nums[largest]:
        largest = left
    if right < n and nums[right]>nums[largest]:
        largest = right
    if largest != i:
        nums[largest], nums[i] = nums[i], nums[largest]
        heapify(nums,n,largest)

def create_heap(ls):
    n = len(ls)
    for i in range(n//2,-1,-1):
        heapify(ls,n,i)

    return ls
```

[149]:
```
# testing code
# do NOT modify testing code below
def test_heap():
    nums = [5,4,3,2,10]
    expected = [10, 5, 3, 2, 4]
    actual = create_heap(nums)
    print(actual)
    assert expected == actual, "Mistake in implementation"
```

[150]:
```
# if your implementation is correct, you should see "OK"
# if your implementation is not correct, you will see "Mistake in␣
 ↪implementation"
test_heap()
print('OK')
```

```
[10, 5, 3, 2, 4]
OK
```

Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(logn) - due to recursive call stack (normally heap sort should be O(1))

Time Analysis: O(n)

0.1.2 Task 1. Implement HeapSort (4 points)

[151]:
```
# Implement HeapSort
# Please implement the function which takes as an input list of numbers and␣
 ↪sorts it using HeapSort
# You can use create_heap() function from above
def heap_sort(ls):
    ls = create_heap(ls)
    for i in range(len(ls)-1,0,-1):
        ls[i],ls[0] = ls[0],ls[i]
        heapify(ls,i,0)

    return ls
```

[152]:
```
def test_heap_sort():
    nums = [5,4,3,2,10]
    expected = [2, 3, 4, 5, 10]
    actual = heap_sort(nums)
    print(actual)
    assert expected == actual, "Mistake in implementation"
    print('OK')
```

[153]:
```
test_heap_sort()
```

```
[2, 3, 4, 5, 10]
OK
```

Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(logn) - due to recursive call stack (normally heap sort should be O(1))

Time Analysis: O(nlogn)

0.2 Task 2. Solve the problem "Height Checker" from https://leetcode.com/problems/height-checker/description/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

[154]:
```python
from typing import List
from collections import defaultdict

class Solution:
    def heightChecker(self, heights: List[int]) -> int:
        num_indices = 0
        # build counting array
        freq_arr = [0 for i in range(1,101)]
        for i in heights:
            freq_arr[i-1] +=1
        # sort the array
        sorted_heights = []
        for i in range(len(freq_arr)):
            if freq_arr[i] != 0:
                sorted_heights.extend([i+1]*freq_arr[i])
        # print(sorted_heights)
        # compare sorted with original
        for i in range(len(heights)):
            if sorted_heights[i] != heights[i]:
                num_indices += 1


        return num_indices
```

0.2.1 Do not modify the testing code below. If you get message "Mistake in test case #", it means that you algorithm is incorrect.

[155]:
```python
#test_case_1
expected, nums = 3, [1,1,4,2,1,3]
actual = Solution().heightChecker(nums)
assert expected==actual, "Mistake in test case 1"
```

```
#test_case_2
expected, nums = 5, [5,1,2,3,4]
actual = Solution().heightChecker(nums)
assert expected==actual, "Mistake in test case 2"
print('OK')
```

```
OK
```

0.2.2 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(n)

Time Analysis: O(n+k), where k is the largest value of the array

0.3 Task 3. Solve the problem "Top K Frequent Elements" from https://leetcode.com/problems/top-k-frequent-elements/description/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

[156]:
```
from typing import List
from collections import defaultdict

class Solution:
    def topKFrequent(self, nums: List[int], m: int) -> List[int]:
        # create dict with number: freq
        hash_map = {}
        for num in nums:
            if num in hash_map:
                hash_map[num] += 1
            else:
                hash_map[num] = 1
        # print(hash_map)
        # sort hashmap by value from largest to smallest, using sorted function
        hash_map = sorted(hash_map.items(), key=lambda x:x[1],reverse=True)
        # print(hash_map)
        # append top k elements from sorted to result
        result = []
        for i in range(k):
            result.append(hash_map[i][0])
        return result
```

[157]:
```
#test_case_1
expected, nums, k = [1, 2], [1,1,1,2,2,3], 2
actual = Solution().topKFrequent(nums, k)
assert expected==actual, "Mistake in test case 1"
```

```
#test_case_2
expected, nums, k = [1], [1], 1
actual = Solution().topKFrequent(nums, k)
assert expected==actual, "Mistake in test case 2"
print('OK')
```

```
OK
```

0.3.1 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(n) - used hashmap and array

Time Analysis: O(nlogn) - used sorting function