CS532 Homework 6
Archana Machireddy

Question 1

```
import math
from timeit import default_timer as timer
import random

class ht_element:
    def __init__(self, key):
        self.key = key
        self.next = None
        self.prev = None

def next_power_of_2(x):
    return 1 if x == 0 else 2**math.ceil(math.log2(x))

class chained_hash:

    def __init__(self,t_size):
        self.size = t_size
        self.new_size = next_power_of_2(self.size)
        self.T = [None] * self.new_size
        self.power = int(math.log2(self.new_size))
        knuth_A = (math.sqrt(5)-1)/2
        #         self.p=14
        self.p = self.power
        m = 2**self.p
        self.w = 32
        self.s = int(knuth_A * 2**(self.w))

    def insert(self, x):
        hash_val = self.hash_function(x.key)
        if self.T[hash_val] is None:
            self.T[hash_val] = x
        else:
            node = self.T[hash_val]
            x.next = node
            node.prev = x
            self.T[hash_val] = x

    def search(self,k):
        hash_val = self.hash_function(k)
        ans = None
```

```
        node = self.T[hash_val]
        while node is not None:
            if node.key == k:
                ans = node
                break
            else:
                node = node.next
        return ans

    def delete(self,x):
        hash_val = self.hash_function(x.key)
        if x.prev == None:
            self.T[hash_val] = x.next
        else:
            x.prev.next = x.next
            if x.next is not None:
                x.next.prev = x.prev

    def hash_function(self, k):
        return k % self.new_size

    def print_node(self,k):
        hash_val = self.hash_function(k)
        x = self.T[hash_val]
        list_contents=[]
        while x is not None:
            list_contents.append(str(x.key))
            x = x.next
        return " -> ".join(list_contents)

    def hash_function_mul(self,k):
        res= k * self.s
        #        print('k: ',k)
        #        print('k*s: ',res)
        r1 = res/(2**self.w)
        r0 = res%(2**self.w)
        #        print('r1: ',r1)
        #        print('r0: ',r0)
        final = r0 >> (self.w-self.p)
        return final


def test1():
    h = chained_hash(10)
    print('Hash table Initialization:')
    print(h.T)
    h.insert(ht_element(7))
    print('Hash table after inserting element 7:')
    print(h.T)
    print('Printing linked list at slot that key 7 was hashed to:')
```

```
    print(h.print_node(7))
    h.insert(ht_element(8))
    h.insert(ht_element(24))
    print('Printing linked list at slot that keys 8 and 24 were hashed
to:')
    print(h.print_node(8))
    print('Printing linked list at slot that keys 8 and 24 were hashed
to after deleting key 24:')
    h.delete(h.search(24))
    print(h.print_node(8))
```

```
Archanas-MBP:HW6 archana$ python3 hw6.py
Hash table Initialization:
[None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None]
Hash table after inserting element 7:
[None, None, None, None, None, None, None, <__main__.ht_element object at 0x10189ec18>, None, None, None, None
, None, None, None, None]
Printing linked list at slot that key 7 was hashed to:
7
Printing linked list at slot that keys 8 and 24 were hashed to:
24 -> 8
Printing linked list at slot that keys 8 and 24 were hashed to after deleting key 24:
8
```

Question 2

```
class ht_element2(ht_element):
    def __init__(self, key, value):
        super().__init__(key)
        self.value = value


def test2():
    h = chained_hash(10)
    h.insert(ht_element2(7,5))
    h.insert(ht_element2(8,6))
    h.insert(ht_element2(24,4))
    print('Inserted keys=(7,8,24) with values=(5,6,4)')
    print('Retriving value of key 24: ', h.search(24).value)
    print('Retriving value of key 8: ',h.search(8).value)
    print('Retriving value of key 7: ',h.search(7).value)
```

```
Inserted keys=(7,8,24) with values=(5,6,4)
Retriving value of key 24:  4
Retriving value of key 8:  6
Retriving value of key 7:  5
```

Question 3

```
def mul_hash(k):
    knuth_A = (math.sqrt(5)-1)/2
    p=14
    m = 2**p
    w = 32
    s = int(knuth_A * 2**(w))
    res= k * s
    print('k: ',k)
    print('k*s: ',res)
    r1 = res/(2**w)
    r0 = res%(2**w)
    print('r1: ',r1)
    print('r0: ',r0)
    final = r0 >> (w-p)
    return final


def test_mul_hash():
    k=123456
    check = mul_hash(k)
    print('h(k): ', check)
```

```
k:  123456
k*s:  327706022297664
r1:  76300.00410081446
r0:  17612864
h(k):  67
```

Integrating it in the chained hash class is shown in solution to question 1.

Question 4

```
class Node(object):
    def __init__(self, value):
        self.parent = self
        self.value = value
        self.rank = 0

    def __str__(self):
        return self.value


class forests():
    def __init__(self,values=[]):
        self.set = [Node(value) for value in values]

    def make_set(self,x):
        #        x.parent = x
        #        x.rank = 0
        self.set.append(Node(x))

    def union(self,x,y):
        self.link(self.find_set(x),self.find_set(y))

    def link(self,x,y):
        if x.rank > y.rank:
            y.parent = x
        else:
            x.parent = y
            if x.rank == y.rank:
                y.rank = y.rank + 1

    def find_set(self,x):
        if x != x.parent:
            x.parent = self.find_set(x.parent)
        return x.parent


def test_forests():
    a= forests(['a','b','c','d','e'])
    print(len(a.set))
    sets =  [str(a.find_set(x)) for x in a.set]
    print("set representatives:\t\t", sets)
    print("number of disjoint sets:\t", len(set(sets)))
    a.union(a.set[0],a.set[2])
    sets =  [str(a.find_set(x)) for x in a.set]
    print("set representatives:\t\t", sets)
    print("number of disjoint sets:\t", len(set(sets)))
    a.union(a.set[0],a.set[1])
```

```
    sets =  [str(a.find_set(x)) for x in a.set]
    print("set representatives:\t\t", sets)
    print("number of disjoint sets:\t", len(set(sets)))
```

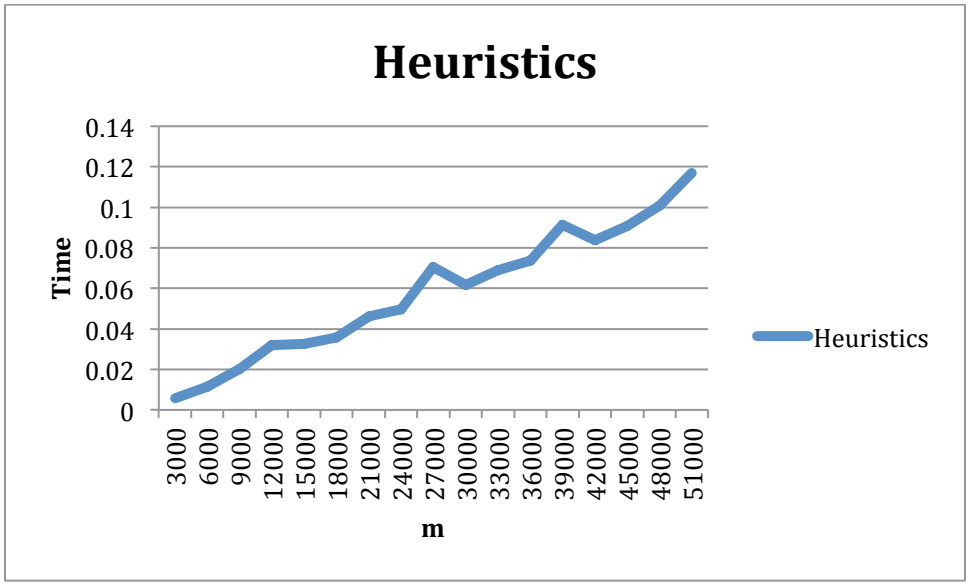
```
Number of disjoin sets:  5
Initial set representatives:     ['a', 'b', 'c', 'd', 'e']
Number of disjoin sets:  5
Set representatives after union of set[0] and set[2]:    ['c', 'b', 'c', 'd', 'e']
Number of disjoin sets:  4
Set representatives after union of set[0] and set[1]:    ['c', 'c', 'c', 'd', 'e']
Number of disjoin sets:  3
```

The class initializer is doing the function of make-set during initialization (setting the parent, value and rank). So, it can just be included in the initialization.

Question 5

```
def time_forests():
    time = []
    m  = range(300,6000,300)
    for i in m:
        n = int(i/3)
        start = timer()
        f = forests()
        for j in range(n):
            f.make_set(j)
        left = i - n
        while left > 0:
            a = random.randint(0,n-1)
            r1 = f.find_set(f.set[a])
            left = left-1
            r2 = r1
            while r1 == r2:
                b = random.randint(0,n-1)
                r2 = f.find_set(f.set[b])
                left = left-1
            f.union(r1,r2)
            left = left-1
        end = timer()
        print(i, end-start)
```

m	Time
3000	0.005694153
6000	0.01159682
9000	0.020357534
12000	0.032103187
15000	0.032596103
18000	0.035646319
21000	0.046231532
24000	0.049675917
27000	0.070627222
30000	0.06153777
33000	0.069074448
36000	0.073570288
39000	0.091271058
42000	0.083930505
45000	0.090872624
48000	0.100716254
51000	0.116978867
54000	0.107622541
57000	0.134907157
60000	0.13334815



The increase is almost linear.

Question 6

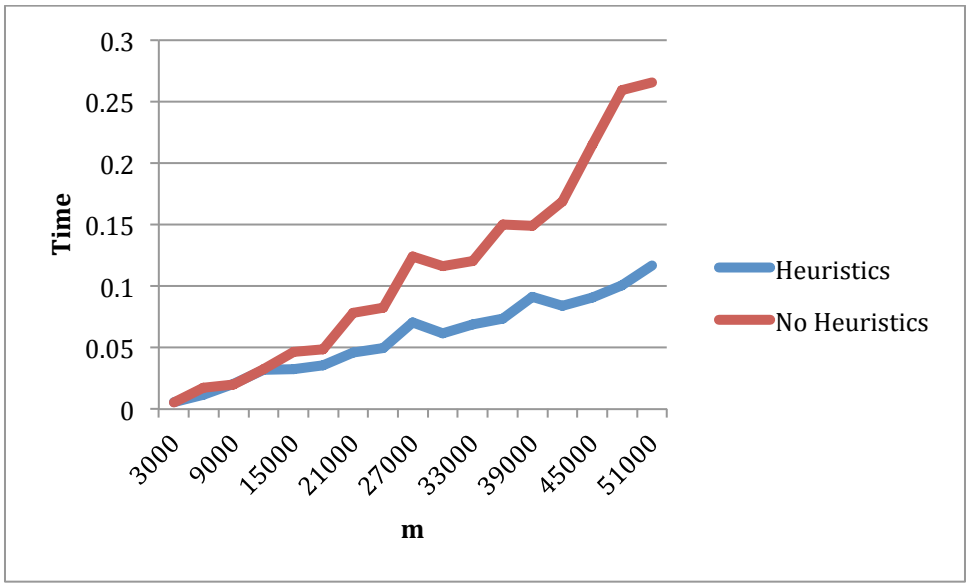
```
class forests_2():
    def __init__(self,values=[]):
        self.set = [Node(value) for value in values]

    def make_set(self,x):
        #        x.parent = x
        #        x.rank = 0
        self.set.append(Node(x))

    def union(self,x,y):
        x_n = self.find_set(x)
        y_n = self.find_set(y)
        x_n.parent = y_n

    def find_set(self,x):
        if x == x.parent:
            return x
        else:
            return self.find_set(x.parent)
```

Question 7

```
def time_forests(ver):
    time = []
    m  = range(3000,63000,3000)
    for i in m:
        n = int(i/3)
        start = timer()
        if ver == 1:
            f = forests()
        else:
            f = forests_2()
        for j in range(n):
            f.make_set(j)
        left = i - n
        while left > 0:
            a = random.randint(0,n-1)
            r1 = f.find_set(f.set[a])
            left = left-1
            r2 = r1
            while r1 == r2:
                b = random.randint(0,n-1)
                r2 = f.find_set(f.set[b])
                left = left-1
```

```
    f.union(r1,r2)
    left = left-1
end = timer()
print(i, end-start)
```

m	Time
3000	0.005218191
6000	0.017321978
9000	0.020208567
12000	0.032624985
15000	0.046544657
18000	0.048623987
21000	0.078163001
24000	0.082651261
27000	0.12400717
30000	0.11630151
33000	0.120294916
36000	0.149968778
39000	0.149292141
42000	0.169068866
45000	0.215476719
48000	0.259552196
51000	0.265630249
54000	0.254370726
57000	0.303942229
60000	0.306161941



The time taken for m = 60000 in the previous case is only 0.13 seconds, while here it is 0.30 seconds. Union by rank and path compression heuristics make sure the trees are not long. This makes the find operation faster. Without these heuristics, find has an average performance of O(log n) and worst case performance of O(n). So the time increases at a greater rate without the heuristics. The other two operations union and make set take constant time.