Ülesanne Boonus: Koormustegur ja Rehashing (Double hashing vs Rehashing)

Selgita, mis on räsitabeli koormustegur ja miks see on oluline.
Räsitabeli koormustegur aitab näidata, kui palju koormust materjal või struktuur talub enne purunemist. See on oluline, sest aitab inseneridel hinnata materjalide vastupidavust erinevatele stressidele, tagades konstruktsioonide ohutuse.

Rakenda lihtsat Rehashingu protsessi ja aruta, kuidas see aitab säilitada efektiivset räsitabelit.

```
class Map:
    class MapNode:
        def __init__(self,key,value):
            self.key=key
            self.value=value
            self.next=None

    buckets=list()
    size=0
    numBuckets=0
    DEFAULT_LOAD_FACTOR = 0.75

    def __init__(self):
        Map.numBuckets = 5
        Map.buckets = [None]*Map.numBuckets

        print("HashMap created")
        print("Number of pairs in the Map: " + str(Map.size))
        print("Size of Map: " + str(Map.numBuckets))
        print("Default Load Factor : " + str(Map.DEFAULT_LOAD_FACTOR) + "\n")

    def getBucketInd(self,key):

        hashCode = hash(key)
        return (hashCode % Map.numBuckets)

    def insert(self,key,value):

        bucketInd = self.getBucketInd(key)
        head = Map.buckets[bucketInd]

        while (head != None):
            if (head.key==key):
                head.value = value
                return
            head = head.next

        newElementNode = Map.MapNode(key, value)
        head = Map.buckets[bucketInd]
        newElementNode.next = head
        Map.buckets[bucketInd]= newElementNode
        print("Pair(\" {} \", \" {} \") inserted successfully.".format(key,value))
        Map.size+=1

        loadFactor = (1* Map.size) / Map.numBuckets
        print("Current Load factor = " + str(loadFactor))

        if (loadFactor > Map.DEFAULT_LOAD_FACTOR):
            print(str(loadFactor) + " is greater than " +
str(Map.DEFAULT_LOAD_FACTOR) + "Rehashing will be done.")

            self.rehash()

            print("New Size of Map: " + str(Map.numBuckets))

        print("Number of pairs in the Map: " + str(Map.size))
        print("Size of Map: " + str(Map.numBuckets))

    def rehash(self):

        print("\n***Rehashing Started***\n")
        temp = Map.buckets
        buckets =(2 * Map.numBuckets)

        for i in range(2 * Map.numBuckets):
            Map.buckets.append(None)

        Map.size = 0
        Map.numBuckets *= 2

        for i in range(len(temp)):
            head = temp[i]

            while (head != None):
                key = head.key
                val = head.value

                self.insert(key, val)
                head = head.next

        print("\n***Rehashing Ended***")

    def printMap(self):

        temp = Map.buckets

        print("Current HashMap:")
        for i in range(len(temp)):

            head = temp[i]

            while (head != None):
                print("key = \" {} \", val = {}" .format(head.key,head.value))

                head = head.next


if __name__ == '__main__':
    map = Map()

    map.insert(1, "First")
    map.printMap()

    map.insert(2, "Second")
    map.printMap()

    map.insert(3, "Third")
    map.printMap()

    map.insert(4, "Fourth")
    map.printMap()

    map.insert(5, "Fifth")
    map.printMap()
```

See aitab säilitada efektiivset räsitabelit, sest iga kord, kui lisatakse element, toimub rehashing, mis aitab vältida kokkupõrkeid. Suurte andmekoguste puhul on see ressurssi ja aja kulukas, kuid on vajalik.

Analüüsi Rehashingu mõju räsitabeli jõudlusele.
Rehashing räsitabelis parandab jõudlust, vähendades võtmete kokkupõrkeid ja optimeerides otsimisprotsessi. See vähendab otsimise aega, kuna väheneb sama indeksi suunatud võtmete arv. Samas võib keerulisem rehashing suurendada mälu kasutust ja põhjustada jõudluse langust.