Ülesanne 5: Topelt Räsimise (Double hashing) Rakendamine

Rakenda topelt räsimise algoritm ja aruta, kuidas see aitab ületada ühekordse räsimise piiranguid.

```
class DoubleHashing:
    def __init__(self, size):
        self.size = size
        self.hash_table = [None] * self.size

    def hash_function_1(self, key):
        return key % self.size

    def hash_function_2(self, key):
        return 7 - (key % 7)
    def insert(self, key):
        index = self.hash_function_1(key)

        if self.hash_table[index] is None:
            self.hash_table[index] = key
        else:
            index2 = self.hash_function_2(key)
            i = 1
            while True:
                new_index = (index + i * index2) %
self.size       if self.hash_table[new_index] is None:
                    self.hash_table[new_index] = key
                    break
                i += 1

    def display(self):
        print("Hash Table:")
        for i in range(self.size):
            print(i, "->", self.hash_table[i])


hash_table = DoubleHashing(10)
keys = [12, 5, 23, 25, 42, 35]

for key in keys:
    hash_table.insert(key)

hash_table.display()
```

Topelt räsimine võimaldab ületada ühekordse räsimise piiranguid mitmel viisil:

1. Vähendatud kokkupõrgete tõenäosus: Kui esimene räsimine tekitab kokkupõrke, siis teine räsimine aitab leida uue vaba koha, vähendades seeläbi kokkupõrgete mõju ja võimaldades elementidel hajuda ühtlasemalt.
2. Suurem hajutatus: Kasutades kahte erinevat räsimisfunktsiooni, laieneb võimalike kohtade hulk, kuhu element võib paigutuda. See aitab vältida klastrite moodustumist ja parandab üldist jõudlust, eriti juhul, kui räsimisfunktsioonid on hästi valitud.

Analüüsi oma rakenduse aja- ja ruumikomplekssust.

Rakenduse aja- ja ruumikomplekssus on kõige halvemas olukorras O(n), sest kõige halvemas olukorras on kõik kohad hash tabelis kinni ja peame läbima kõik kohad, et leida vaba koht,

kuhu sisestada võti. While loop jookseb sellises olukorras n korda, kus n on hash tabeli suurus. Seetõttu on rakenduse aja- ja ruumikomplekssus O(n).

Paku välja stsenaarium, kus topelt räsimine oleks eriti efektiivne.

Olukord: Suur andmebaas, kus on vähe kokkupõrkeid ja soovitakse vältida klastrite moodustumist.

Kujutleme suurt online-kaubandusplatvormi, kus on miljoneid tooteid ning iga toote kohta on unikaalne identifikaator või võti. Andmebaas peab haldama neid tooteid kiiresti ja tõhusalt, et tagada kiire otsing ja pääs iga toote detailideni.

Sellises stsenaariumis võib topelt räsimine olla efektiivne, sest:

Vähe kokkupõrkeid: Toote-ID-d võivad olla hästi hajutatud ning väga vähe või peaaegu üldse mitte kokkupõrkeid. Topelträsimine aitab säilitada kiiret otsingut ja andmetele juurdepääsu, hajutades elemendid ühtlaselt.
Klastrite vältimine: Kaubandusplatvorm soovib vältida toodete klastrite moodustumist, kus paljud tooted võiksid asuda samades või lähedalasuvates räsiindeksites. Topelträsimine aitab laiendada võimalikke kohti, kuhu toote-ID-d võivad paigutuda, vähendades seeläbi klastrite tekkimise riski.

Seega võimaldab topelträsimine selles stsenaariumis paremat hajutatust, vähendades kokkupõrgete mõju ning tagades kiire ja tõhusa juurdepääsu suurele hulgale andmetele, mis on hajutatud ühtlaselt üle räsiindeksite.