# Homework 6

due March 7, 2025

## Problem 1:

Given an LH file with 6000 buckets, find the addresses of the records with the following keys:

| Key | Bucket |
|---|---|
| 570046 | |
| 821365 | |
| 56847 | |
| 421651 | |
| 564040 | |
| 956869 | |
| 321872 | |
| 744050 | |
| 892987 | |
| 764908 | |

## Problem 2:

Linear Hashing does not distribute records evenly over records. Create a program that implements LH, storing records in a list. For the purpose of this exercise, we assume that we only store numbers. Have the program insert random numbers until the LH-file has 4000 or 6000 buckets. Use matplotlib.pyplot in order to graph the number of elements in the bucket based on the bucket number. The nominal capacity is 10.

```
import matplotlib.pyplot as plt
import numpy as np

class FileState:
    """Defines the file state of a Linear Hashing structure."""

    def __init__(self):
        self.level = 0
        self.split_pointer = 0

    def __str__(self):
        nb = self.number_of_buckets()
        return f"(l={self.level}, s={self.split_pointer}, n={nb})"
```

```
    def number_of_buckets(self):
        return 2**self.level+self.split_pointer

    def increment(self):
        self.split_pointer += 1
        if self.split_pointer == 2**self.level:
            self.level += 1
            self.split_pointer = 0

    def address(self, key):
        addr = key&(2**self.level-1)
        if addr < self.split_pointer:
            addr = key&(2**(self.level+1)-1)
        return addr

class LH:
    """Defines a LH Structure.
The keys are supposed to be strings or anything on which hash is defined.
The insert method insert(self, key, value) inserts a value for a given key.
The lookup method lookup(self, key) returns the inserted value. All
variable types qualify for a variable, but keys need to be
immutable."""

    def __init__(self, cap):
        self.buckets = [ [] ]
        self.file_state = FileState()
        self.capacity = cap
        self.nr_records = 0
        self.moved = []
        self.number_of_splits = 0

    def __str__(self):
        lengths = [str(len(bucket)) for bucket in self.buckets]
        return (f"LH with file state {self.file_state}\n" +  ' '.join(lengths) +
                '\n' + '\n'.join([str(bucket) for bucket in self.buckets]))

    def split(self):
        self.number_of_splits += 1
        to_split = self.file_state.split_pointer
        to_add = len(self.buckets)
        ret_val = len(self.buckets[to_split])
        self.file_state.increment()
        new_bucket = []
        new_old_bucket = []
        for key, value in self.buckets[to_split]:
            new_addr = self.file_state.address(abs(hash(key)))
            if new_addr == to_add:
                new_bucket.append((key, value))
            elif new_addr == to_split:
                new_old_bucket.append((key, value))
            else:
                print('trouble')
        self.buckets.append(new_bucket)
        self.buckets[to_split] = new_old_bucket
        return ret_val

    def insert(self, key, value):
        address = self.file_state.address(abs(hash(key)))
        self.buckets[address].append((key, value))
        self.nr_records += 1
        if self.nr_records / len(self.buckets) > self.capacity:
            self.split()
```

```
def lookup(self, key):
    address = self.file_state.address(abs(hash(key)))
    for index in range(len(self.buckets[address])):
        if self.buckets[address][index][0] == key:
            return self.buckets[address][index]
    return None
```