159.172 Computational Thinking

Tutorial 8: Hashing

In the lectures, we discussed several ways of how to resolve a collision during hashing. In this tutorial, you are supposed to compare linear probing with quadratic probing to see which one results in fewer collisions, given different load factors of the hash table.

To keep things simple, we assume that the hash table is a Python list with a fixed size of 1009 entries (smallest table with more than 1000 entries where the number of entries is prime). To start with, entries in the table will be set to **`None`**. For hashing, we will use the division method: **`h(key) = key % M`**.

You need to implement an insert function that takes a key, a hash table, and a probe function as arguments and inserts the key into the hash table, using the probe function whenever a collision occurs. The insert function returns the number of collisions that occurred during insertion. The probe function takes two arguments, the home position and the index of the probe. For example, **`linear_probe(5, 3)`** means that we are performing the third probe on hash position 5, resulting in position 8.

To see how many collisions occur for different load factors, you can use the Python **`random`** library. In this library, you can find a function **`random.sample(p, k)`**, which returns a list of length **`k`** of unique elements chosen from the population **`p`**. For the purpose of this tutorial, you need to generate a list of 1000 elements in the range between 0 and 9999:

```
random.sample(range(10000), 1000)
```

Insert the elements into the hash table and record after every 10 insertions how many collisions occurred so far in total. There is no need to be fancy; just produce a list with the numbers of collisions for each of the two probe functions. Compare the lists to see which probe is better at which load factor. The following function might be helpful when comparing the two lists:

```
def comp_lists(list1, list2):
    def comp_elem(elem1, elem2):
        return 1 if elem1 < elem2 else 2
    return map(comp_elem, list1, list2)
```

Repeat the exercise for two other tables of different sizes.

Submit your work via Stream by the end of Sunday 4 October.