📏 Module 5

Hash Tables, Heaps, and Treaps

▼ What is a hash table?

- a form of list (an associative array) where elements are accessed by a keyword rather than an index number
- internally, a hash table will use a slightly modified version of a hashing function in order to find the index position in which the element should be inserted
- this allows fast look-ups, since it's using an index number, which corresponds to the hash value of the key
- Example: hash table
  - Set the size of the hash table to 256 elements to start with.
  - Initialize a list containing 256 elements. These elements are often referred to as slots or buckets.
  - Add a counter for the number of actual hash table elements.

```
# hash item
class HashItem:
    def __init__(self, key, value):
        self.key = key
        self.value = value

# hash table
class HashTable:
    def __init__(self):
        self.size = 256
        self.slots = [None for i in range(self.size)]
        self.count = 0
    # hashing function
    # prefixed by underscore to indicate internal use
    def _hash(self, key):
            mult = 1
            hv = 0
            for ch in key:
                hv += mult * ord(ch)
                mult += 1
            return hv % self.size
```

▼ What is open addressing?

- a method of collision resolution in hash tables
- a hash collision is resolved by probing, or searching through alternative locations in the array (the probe sequence) until either the target record is found, or an unused array slot is found, which indicates that there is no such key in the table

▼ What is linear probing?

When we want to insert a key, we check whether the slot already has an item or not. If it does, we look for the next available slot.

▼ What is quadratic probing?

- the method iteratively tries the buckets, until finding an empty bucket
- as with linear probing, the quadratic probing strategy complicates the removal operation, but it does avoid the kinds of clustering patterns that occur with linear probing
- quadratic probing creates its own kind of clustering, called secondary clustering, where the set of filled array cells still has a non-uniform pattern, even if we assume that the original hash codes are distributed uniformly
- when N is prime and the bucket array is less than half-full, the quadratic probing strategy is guaranteed to find an empty slot

▼ What is double hashing?

- With double hashing, we choose a secondary hash function, h⎵, and if h maps some key k to a bucket A[h(k)] that is already occupied, then we iteratively try the buckets A[(h(k)+ f(i)) mod N] next, for i = 1,2,3,..., where f(i) = i·h␣(k).

- An open addressing strategy that does not cause clustering like linear probing or quadratic probing is the double hashing strategy.

▼ What is hashing?

- Hashing is the concept of converting data of arbitrary size into data of fixed size.
- For example, we want to hash the expression "hello world"—this means we want to get a numeric value that we could say represents the string.

▼ Why not create a perfect hashing function?

- In practice, hashing functions normally need to be very fast, so trying to create a function that will give each string a unique hash value is normally not possible.
- Instead, we live with the fact that we sometimes get collisions (two or more strings having the same hash value), and when that happens, we come up with a strategy for resolving them.

▼ A hash function consists of two components. What are they?

- a hash code that maps a key k to an integer
- a compression function that maps the hash code to an integer within a range of indices, [0, N − 1], for a bucket array

▼ What is the benefit of splitting up the hash function into components?

- The advantage of separating the hash function into two such components is that the hash code portion of that computation is independent of a specific hash table size.
- This allows the development of a general hash code for each object that can be used for a hash table of any size; only the compression function depends upon the table size.
- This is particularly convenient, because the underlying bucket array for a hash table may be dynamically resized, depending on the number of items currently stored in the map.

▼ How can you compute hash codes in Python?

- using a built-in function with signature hash(x) that returns an integer value that serves as the hash code for object x
- ONLY immutable data types are deemed hashable in Python.
    - This restriction is meant to ensure that a particular object's hash code remains constant during that object's lifespan.
    - This is an important property for an object's use as a key in a hash table.
    - Among Python's built-in data types, the immutable int, float, str, tuple, and frozenset classes produce robust hash codes, via the hash function.

▼ What are heaps?

- a specialization of a tree in which the nodes are ordered in a specific way
- a heap is a binary tree *T* that stores a collection of items at its positions and that satisfies two additional properties:
    - a relational property defined in terms of the way keys are stored in *T*
    - a structural property defined in terms of the shape of *T* itself
- heaps are often used to implement priority queues

▼ What is the heap-order property?

In a heap T, for every position p other than the root, the key stored at p is greater than or equal to the key stored at p's parent.

▼ What is a max heap?

- Each parent node must always be greater than or equal to its children.
- The root node must be the greatest value in the tree.

▼ What is a min heap?

- Each parent node must be less than or equal to both its children.
- The root node holds the lowest value.

▼ What is a binary heap?

- A binary heap allows us to perform both insertions and removals in logarithmic time.
- The fundamental way the heap achieves this improvement is to use the structure of a binary tree to find a compromise between elements being entirely unsorted and perfectly sorted.

▼ What are treaps?

- a data structure which combines binary tree and binary heap (hence the name: tree + heap = Treap)
- it stores pairs (X, Y) in a binary tree in such a way that it is a binary search tree by X and a binary heap by Y