Data Structures

Hash Method

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Hash Method

- A hash function **converts** an object (number, array, string, object, etc...) to a single **integer** value
  - Usage? We need for dictionary, which is based on 'hash tables'
- There are different techniques (algorithms) to hash an object
- Let’s try to develop some of them to get better idea what is happening *behind the scenes*

Hashing an integer

- As a simple example, let's take an integer input, and build a hash function which converts any such input to a fixed-size value in the range [0, range-1]
  - **def** hash_integer(num, range)
- How can we code that?

```
def hash_integer(num, range):
    return num % range

if __name__ == '__main__':
    print(hash_integer(100, 1000))      # 100
    print(hash_integer(100, 19))        # 5
```

Hashing a string

- Given a string of lowercase letters, hash it to a number
  - Assume we also want this number **compressed** in the range p[0, range-1]
- **def** hash_string1(str, range)
- Give a trial

```
def hash_string1(str, range):
    sum = 0

    for ch in str:
        sum += ord(ch) - ord('a')    # idx from 0-25
    return sum % range
```

Hashing a string: Observe

```
print(hash_string1("abc", 7))        # 0+1+2 = 3 % 7 = 3
print(hash_string1("abcde", 70))     # 0+1+2+3+4 = 10 % 70 = 10
print(hash_string1("abcde", 7))      # 10 % 7 = 3
print(hash_string1("bcdea", 7))      # 10 % 7 = 3    SAME chars
print(hash_string1("abcwz", 7))      # 50 % 7 = 1
```

- Two strings have the same hash value: abc and abcde (both using %7)
- With a different range (7 vs 70), the same input can produce different outputs
- The permutations will always produce the same value
  - abcde and bcdea ⇒ same sorted string ⇒ same hashed value
- When the same values are output, we call this a **collision**
- We want to **reduce such collisions**. Can you find a better hash function?

Hashing a string with fewer collisions!

- To reduce collisions between the same permutations, let's consider the position
- We can think of a (lowercase) string as a **number of base** 26
  - What is 12345 in base 10? It's $1 * 10^4 + 2 * 10^3 + 3 * 10^2 + 4*10 + 5$
  - We use powers of 10 as we have 10 digits
  - If we have 26 letters (0-25), we use base 26
  - Google other examples: e.g famously, there's the **hexadecimal** number system, where we use base 16

```
def hash_string2(str, range):
    sum = 0

    for ch in str:
        idx = ord(ch) - ord('a')    # 0-25 range
        sum = sum * 26 + idx
    return sum % range
```

Hashing a string with fewer collisions!

```
print(hash_string2("abc", 7))       # 0+1*26+2 = 28 % 7 = 0

# 1 * 26 * 26 * 26 + 2 * 26 * 26 + 3 * 26 + 4 = 19010
print(hash_string2("abcde", 70))    # 19010 % 70 = 40

print(hash_string2("abcde", 7))     # 19010 % 7 = 5
print(hash_string2("bcdea", 7))     # 494260 % 7 = 4
print(hash_string2("abcwz", 7))     # 19525 % 7 = 2
```

- We are now seeing much bigger values
  - Observe: the % operator is really helpful in reducing values to a range
- Now, we have two DIFFERENT values for inputs of the same permutation (abcde, bcdea)
- Is it possible to have NO collisions at all? In general problems, NO
  - Many-to-1 mapping :(
- So, we just want to reduce collisions as much as possible
  - But Why does this matter?

Compression

- Hash functions don't always contain a %. Use of the % operator is called a **compression**
  - It compresses a big number to a specific range
- Using a **prime number** with % ⇒ **may** generate better distribution of values according to number theory, but collisions will still occur
  - Why does having a (mostly) random distribution matter? See upcoming lectures!
- We can do the hash function in any way, as long as it satisfies the request
  - **Same** input ⇒ Same output
  - E.g. return a positive integer for the input
- Bitwise operators can be used in hashing or **combining** several hash values
- (Optional) List of hash functions

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."