shortener-analysis

November 19, 2022

An analysis of different approaches for implementing link-shortener

The first solution is to get the hash of the URL :

[7]:

```
import hashlib
url = 'https://www.pinterest.dk/pin/996422104878931347?
  ↪amp_client_id=CLIENT_ID(_)&mweb_unauth_id=%7B%7Bdefault.
  ↪session%7D%7D&simplified=true'
hashed_url = hashlib.sha256(url.encode('utf-8')).hexdigest()
print(hashed_url)
```

```
348f0fa71cd97914f48aa226ae3a2fee9ab86dfd3f34ca268c5e0b5719f67357
```

[5]:

```
print(len(hashed_url))
```

```
64
```

It is too long, so just getting hash is not enough for us. it is not suitable for URLs longer than 64 characters because there is a chance of collision, we have to avoid this, so let's take a look for a better solution. let me check out what is happening behind the scenes in the hash function :

$$h(x_0, x_1, ..., x_{r-1}) = z\sum_{i=0}^{r-1} z_i x_i$$

assume that every characters of URL is correspond to an integer (in this case we use ascii) like X(i) then take some random odd prime numbers for Z(i)

it is proved that :

$$Pr(h(x0, x1, ..., xr - 1) = h(y0, y1, ..., yr - 1)) \leq 3/2^w$$

[58]:

```
w = 64
a = 2 ** w
pr = 3 / a

print("the probability of having collision: ", pr)
```

```
the probability of having collision:  1.6263032587282567e-19
```

We see that the probability is very low and close to zero so I decided to use this hash function. first of all, we need a function to map every char to int, let's leave it to the python internal function which is Ord. Here for each Zi I use prime ** i, i increasing from 0 to 2000 (2000 is safe margin)

[60]:
```
import string
prime = 96527
modbase = 64 ** 8
max_len = 2000
primes = [prime ** i for i in range(max_len)]

def ordsum(text: str) -> int:
    return sum([(ord(c) * primes[i]) % modbase  for i, c in enumerate(text)]) %␣
  ↪modbase



print("resault: ", ordsum(url))
```

```
resault:  73106310689270
```

we have to change the output base so we use this function (I found it on the internet)

[62]:
```
digs = string.ascii_letters + string.digits + '-_'
```

A URL character can be one of the following

- A lower case alphabet ['a' to 'z'], total 26 characters
- An upper case alphabet ['A' to 'Z'], total 26 characters
- A digit ['0' to '9'], total 10 characters

There are total 26 + 26 + 10 = 62 possible characters.

[63]:
```
def int2base(x, base=64):
    if x < 0:
        sign = -1
    elif x == 0:
        return digs[0]
    else:
        sign = 1

    x *= sign
    digits = []

    while x:
        digits.append(digs[int(x % base)])
        x = int(x / base)

    if sign < 0:
        digits.append('-')
    digits.reverse()

    return ''.join(digits)
```

```
[76]: shortener_10_base = ordsum(url)
      print("\nshort link with base 10 : ", shortener_10_base, "\n")
```

```

short link with base 10 :  73106310689270

```

```
[80]: shortener_64_base = int2base(shortener_10_base)
      print("\nshort link with base 64 : ", shortener_64_base, "\n")
```

```

short link with base 64 :  qN1JRFh2

```

Ok now we have a solution that long url map to 8 characters and the probability of collision is almost zero, but for more safety we can use several prime number to make sure everything goes right.

```
[81]: prime_numbers = [65423, 66721, 73517, 78697, 86249, 95923, 50591, 20663, 22739,␣
       ↪101141]
```

If we have 10,000 link how much time does it take to generate short links ?

```
[82]: def x():
          for i in range(10000):
              s = ordsum(url)
              r = int2base(s)
          return r

      get_ipython().run_line_magic('timeit', 'x()')
```

```
662 ms ± 1.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

preprocessing primes values is a way to optimize the solution :

```
[85]: primes = [prime ** i for i in range(max_len)]
```

matrix multiplication of Zi & Xi is the best practice for optimization the run time.

I've implemented all of them in service.py in my project