Data Structures in Python

1. Hash Table
2. Collision Resolution
3. **Double Hashing & Rehashing**
4. HashMap Coding

Agenda & Readings

- Collision Resolution
  - Separate Chaining
  - Open Addressing
    - Linear Probing
    - Quadratic Probing
    - **Double Hashing**
- **Rehashing**

- Reference:
  - Problem Solving with Algorithms and Data Structures
  - Chapter 5 - Hashing

Collision Resolution by Open Addressing

1. Linear Probing (선형조사법)
2. Quadratic Probing (이차조사법)
3. Double Hashing (이중해싱법)

Collision Resolution - Double Hashing이중해싱법

- Keep two hash functions, h(x) and h'(x).
- Use **a second hash function** for all tries i other than 0
  $f(i) = i * \mathbf{h'(x)}$
- Good choices for h'(x)?

Hash function

$h_i(k) = (h(k) + f(i))$ % TableSize

Collision Resolution - Double Hashing 이중해싱법

- Keep two hash functions, h(x) and h'(x).
- Use **a second hash function** for all tries i other than 0
  f(i) = i * **h'(x)**
- Good choices for h'(x)?
  - Should never evaluate to 0.
  - h'(x) = R - (x % R)
    - R is prime number less than TableSize.

Hash function

```
hi(k) = (h(k) + f(i)) % TableSize
```

Collision Resolution - Double Hashing이중해싱법

- Keep two hash functions, h(x) and h'(x).
- Use **a second hash function** for all tries i other than 0
  f(i) = i * **h'(x)**
- Good choices for h'(x)?
  - Should never evaluate to 0.
  - h'(x) = R - (x % R)
    - R is prime number less than TableSize.
- For example, h(x) = k % 10 with R = 7

  $h_0(49)$ = (h(49) + f(0)) % 10 = 9 (collision)

  $h_1(49)$ = (h(49) + 1 * (7 - 49 % 7)) % 10 = 6

Hash function

$h_i(k)$ = (h(k) + f(i)) % TableSize

Collision Resolution - Double Hashing이중해싱법

- Keep two hash functions, h(x) and h'(x).
- Use **a second hash function** for all tries i other than 0
  $f(i) = i * \mathbf{h'(x)}$
- Good choices for h'(x)?
  - Should never evaluate to 0.
  - $h'(x) = R - (x \% R)$
    - R is prime number less than TableSize.
- For example, $h(x) = k \% 10$ with $R = 7$

  $h_0(49) = (h(49) + f(0)) \% 10 = 9$ (collision)

  $h_1(49) = (h(49) + \underbrace{\overbrace{1}^{i} * \overbrace{(7 - 49 \% 7)}^{h'(x)}}_{f(1)}) \% 10 = 6$

Hash function
$h_i(k) = (h(k) + f(i)) \% TableSize$

Collision Resolution - Double Hashing이중해싱법

- Keep two hash functions, h(x) and h'(x).
- Use **a second hash function** for all tries i other than 0
  $f(i) = i * \mathbf{h'(x)}$
- Good choices for h'(x)?
  - Should never evaluate to 0.
  - $h'(x) = R - (x \% R)$
    - R is prime number less than TableSize.

Hash function
`hi(k) = (h(k) + f(i)) % TableSize`

- For example, $h(x) = k \% 10$ with $R = 7$

  $h_0(49) = (h(49) + f(0)) \% 10 = 9$ (collision)

  $h_1(49) = (h(49) + 1 * (7 - 49 \% 7)) \% 10 = 6$

  f(1)   h'(x)
  i

If we assume that $h_1(49) = 6$ ends up a collision, the next probing is …

Collision Resolution - Double Hashing이중해싱법

- Keep two hash functions, h(x) and h'(x).
- Use **a second hash function** for all tries i other than 0
  $f(i) = i * \mathbf{h'(x)}$
- Good choices for h'(x)?
  - Should never evaluate to 0.
  - $h'(x) = R - (x \% R)$
    - R is prime number less than TableSize.
- For example, h(x) = k % 10 with R = 7

Hash function
$h_i(k) = (h(k) + f(i)) \% \text{TableSize}$

$h_0(49) = (h(49) + f(0)) \% 10 = 9$ (collision)

$h_1(49) = (h(49) + \underbrace{1 * \overbrace{(7 - 49 \% 7)}^{h'(x)}}_{f(1)}) \% 10 = 6$

i

If we assume that $h_1(49) = 6$ ends up a collision, the next probing is …

$h_2(49) = (h(49) + 2 * (7 - 49 \% 7)) \% 10 = 3$

Collision Resolution Example - Double Hashing이중해싱법

- Insert keys 43, 25 into the hash table below and find the probe sequence for each:
- Use h(k) = k % 13 with R = 7.

0	1	2	3	4	5	6	7	8	9	10	11	12
26	None	54	94	17	31	None	None	None	None	None	None	17

$h_0(43)$ = h(43) = 43 % 13 = 4 (collision)

$h_1(43)$ = (h(43) + 1 * (7 - 43 % 7)) % 13 = (4 + 6) % 13 = 10, Probe sequence: 4, 10

Collision Resolution Example - Double Hashing이중해싱법

- Insert keys 43, 25 into the hash table below and find the probe sequence for each:
- Use h(k) = k % 13 with R = 7.

0	1	2	3	4	5	6	7	8	9	10	11	12
26	None	54	94	17	31	None	None	None	None	None	None	17

$h_0(43) = h(43) = 43 \% 13 = 4$ (collision)

$h_1(43) = (h(43) + 1 * (7 - 43 \% 7)) \% 13 = (4 + 6) \% 13 = 10$, Probe sequence: 4, 10

$h_0(25) = h(25) = 25 \% 13 = 12$ (collision)

$h_1(25) = (h(25) + 1 * (7 - 25 \% 7)) \% 13 = (12 + 3) \% 13 = 2$

$h_2(25) = (h(25) + 2 * (7 - 25 \% 7)) \% 13 = (12 + 6) \% 13 = 5$

$h_1(25) = (h(25) + 3 * (7 - 25 \% 7)) \% 13 = (12 + 9) \% 13 = 8$, Probe sequence:12,2,5,8

Collision Resolution Example - Double Hashing이중해싱법

- Insert keys 43, 25 into the hash table below and find the probe sequence for each:
- Use h(k) = k % 13 with R = 7.

0	1	2	3	4	5	6	7	8	9	10	11	12
26	None	54	94	17	31	None	None	None	None	None	None	17

$h_0(43) = h(43) = 43 \% 13 = 4$ (collision)

$h_1(43) = (h(43) + 1 * (7 - 43 \% 7)) \% 13 = (4 + 6) \% 13 = 10$, Probe sequence: 4, 10

$h_0(25) = h(25) = 25 \% 13 = 12$ (collision)

$h_1(25) = (h(25) + 1 * (7 - 25 \% 7)) \% 13 = (12 + 3) \% 13 = 2$

$h_2(25) = (h(25) + 2 * (7 - 25 \% 7)) \% 13 = (12 + 6) \% 13 = 5$

$h_1(25) = (h(25) + 3 * (7 - 25 \% 7)) \% 13 = (12 + 9) \% 13 = 8$, Probe sequence:12,2,5,8

0	1	2	3	4	5	6	7	8	9	10	11	12
26	None	54	94	17	31	None	None	25	None	43	None	17

Collision Resolution Exercise - Double Hashing이중해싱법

Insert sequence: **89, 18, 49, 58, 69, 23**

$h(x) = x \% 10$

$h'(x) = R - (x \% R)$
R is prime number less than TableSize

	Empty Table	After 89	After 18	After 49	After 58	After 69	After 23
0							
1							
2							
3							
4							
5							
6				49	49	49	49
7							
8			18	18	18	18	18
9		89	89	89	89	89	89
Unsucessful no. of probes		**0**	**0**	**1**	**2**	**2**	

$h_0(49) = (h(49)+f(0)) \% 10 = 9$ (collision)
$h_1(49) = (h(49)+1*(7 - 49 \% 7)) \% 10 = 6$

$h_0(58) =$
$h_1(58) =$

$h_0(69) =$
$h_1(69) =$

$h_0(23) =$
$h_1(23) =$
:

Collision Resolution Analysis - Double Hashing

- Imperative that TableSize is prime
  - e.g., insert 23 into previous table

Is it good or bad?

- Empirical tests show **double hashing** close to random hashing.
- Extra hash function takes extra time to compute.

Rehashing

- Rehashing is the reconstruction of the hash table:

0	6
1	15
2	
3	24
4	
5	
6	13

0	
1	
2	
3	
4	
5	
6	6
7	23
8	24
9	
10	
11	
12	
13	13
14	
15	15
16	

Rehashing

- Rehashing is the reconstruction of the hash table:
  - All the elements in the container are rearranged according to their hash value into the new set of buckets. This may alter the order of iteration of elements within the container.

- Increases the size of the hash table when load factor becomes "too high" (defined by a cutoff)
  - Anticipating that collisions would become higher
- Typically expand the table to twice its size (but still prime)
  - $TableSize_{new} = nextprime(2 * TableSize_{old})$
  - e.g., 2 → 5, 5 → 11, 11 → 23
- Need to reinsert all existing elements into new hash table

Rehashing Example

h(x) = x % 7
$\lambda = 0.57$

0	6
1	15
2	
3	24
4	
5	
6	13

Insert 23

$\lambda = 0.71$

0	6
1	15
2	
3	24
4	
5	
6	13

TableSize = 7

$\lambda_{max} = 0.6$

Rehashing
since $\lambda > \lambda_{max}$

0	
1	
2	
3	
4	
5	
6	6
7	23
8	24
9	
10	
11	
12	
13	13
14	
15	15
16	

h(x) = x % 17
$\lambda = 0.29$

TableSize = 17
nextprime(TableSize * 2)

Hashing Analysis

- The **load factor** (λ) of the hash table is the number of items in the table divided by the size of the table.

- If λ is small then keys are more likely to be mapped to slots where they belong and searching will be O(1).

- If λ is large then collisions are more likely and more comparisons (is the slot available or not) are needed to find an empty slot.

Rehashing Analysis

- If the load factor goes high, the performance slows down significantly. In that case the easiest solution is to copy the entire hash table into a larger table.
  This process is called rehashing.

- When to rehash
  - For separate chaining, the load factor should not exceed **0.75.**
    For open addressing, the load factor should not exceed **0.5.**

- Rehashing a table is expensive (since elements must be inserted using the new hash function) - do only occasionally, e.g. double size of table each time, but make sure that the size is a prime number.

Rehashing - Exercise

- Rehash the following table into a new hash table below using the hash function: Use hash(key) = key % 13 and quadratic probing to resolve the collisions. Show your computation, collision and resolution. Compute the load factors before and after rehashing .

0	1	2	3	4	5	6
56	43	30	None	None	26	13

0	1	2	3	4	5	6	7	8	9	10	11	12

Hashing Applications

- Symbol table in compilers
- Accessing tree or graph nodes by name
  - e.g., city names in Google maps
- Maintaining a transposition table in games
  - Remember previous game situations and the move taken (avoid re-computation)
- Dictionary lookups
  - Spelling checkers
  - Natural language understanding (word sense)
- Heavily used in text processing languages
  - e.g., Perl, Python, etc.

Summary

- The hash table size uses a prime number in general.
  - The table size is larger than number of inputs (to maintain $\lambda \ll 1.0$)
  - It helps its performance and prevents it from rehashing.
- The collision cannot be avoided.
  - Collision resolution strategies are required.
  - There are some trade-offs between chaining vs. probing
  - Collision chances decrease in this order:
    linear probing → quadratic probing → double hashing
- Rehashing is recommended when the load factor λ exceeds 0.5 in general.