hashing

hashing = doğramak, kıymak

average time for insert, delete and find O(1)
- worst O(n) -

* inefficient in operations that require any ordering
  ↳ such as finding min or max, sorting

applications:
- dictionary
- spelling correction
- ctrl+f (find command)
- authentication (doğrulama)

hash table
[0]
[1]
[2] 22
[3]
[4]
[5] 35
[6]
[7] 27
[8]
[9]

insertion of data with its key

hash function: key of the data % table size = hash value

data 1: 22 → 22 % 10 = 2
data 2: 35 → 35 % 10 = 5
data 3: 27 → 27 % 10 = 7
data 4: 32 → 32 % 10 = 2 → collision

collision handling
1- seperate chaining (linked lists)
2- linear probing (araştırma, search)
3- quadratic probing
4- double hashing
} open adressing

! important to have prime table size (for uniform distribution)

① seperate chaning:
- the array elements are pointers to the first node of the lists
- new item is inserted to the front of the list
- load factor should be around 1.

[0] 3 → 9 → 6
[1] 7 → 4
[2] 2 → 14

load factor (λ): n(elements) / table size

- cost of searching = constant time + expected chain length = O(1+λ) - average $(1+\frac{\lambda}{2})$ -

open adressing:
- all the data go inside the table, so bigger table is needed
- the load factor should be below 0.5.
- if a collision occurs, alternative cells are tried until an empty cell is found

② linear probing

hash function

hash table

$$\left(\text{key of the data} \;\%\; \text{table size} \;+\; i\right) \;\%\; \text{table size} = \text{hash value}$$

data 1 : 22 → 2 (i=0)
data 2: 33 → 3 (i=0)
data 3: 24 → 4 (i=0)
data 4: 52 → 2 + i → i = 1 [3] full
i = 2 [4] full
i = 3 [5] ✓

[0]
[1]
[2] 22
[3] 33
[4] 24
[5] 52
[6]
[7]
[8]
[9]

b) use lazy deletion (if you remove item, algorithm fails. mark it in a special way)

* primary clustering = when the blocks of occupied cells start forming

insertion = $\left(1 + \left(\frac{1}{1-\lambda}\right)^2\right) \cdot \frac{1}{2}$ (the average cells that are examined)

find = successful is same as insertion, unsuccessful average $\left(1 + \frac{1}{1-\lambda}\right) \cdot \frac{1}{2}$

example of linear probing analysis

[0] 9
[1]
[2] 2
[3] 13
[4] 25
[5] 24
[6]
[7]
[8] 30
[9] 20
[10] 10

successful

* 20 — 20%11 → [9] ✓ — 1
* 30 — 30%11 → [8] ✓ — 1
* 2 — 2%11 → [2] ✓ — 1
* 13 — 13%11 → [2], [3] ✓ — 2
* 25 — 25%11 → [3], [4] ✓ — 2
* 24 — 24%11 → [2], [3], [4], [5] ✓ — 4
* 10 — 10%11 → [10] ✓ — 1
* 9 — 9%11 → [9], [10], [0] ✓ — 3

+ 15

avg = $\frac{15}{8}$

unsuccessful

* 0 — [0], [1] null ✓ — 2
* 1 — [1] null ✓ — 1
* 2 — [2][3][4][5][6] null ✓ — 5
* 3 — [3][4][5][6] null ✓ — 4
* 4 — [4][5][6] null ✓ — 3
* 5 — [5][6] null ✓ — 2
* 6 — [6] null ✓ — 1
* 7 — [7] null ✓ — 1
* 8 — [8][9][10][0][1] ✓ — 5
* 9 — [9][10][0][1] ✓ — 4
* 10 — [10][0][1] ✓ — 3

+ 31

avg = $\frac{31}{8}$

c) quadratic probing

hash function

$$\left(\text{key of the data} \;\%\; \text{table size} \;+\; i^2\right) \;\%\; \text{table size} = \text{hash value}$$

$H_i = H_{i-1} + 2i - 1 \pmod M$

* eliminates primary clustering problem of linear probing
* there is no guarantee to find an empty cell (problem)

(if size is prime and $\lambda \leq 0{,}5$, it is okay)

* secondary clustering = when the same alternative cells are occupied.

rehashing: expending the table, when the load factor reaches 0.5, dynamically.

- double to a prime
- reinsert the new table by using new hash function!

④ double hashing

- eliminates secondary clustering

hash function

$$hash1(k) + j \cdot hash2(k) = \text{hash value}$$