Session 09 - Hash Table

Instructor:

Dr. LE Thanh Tung

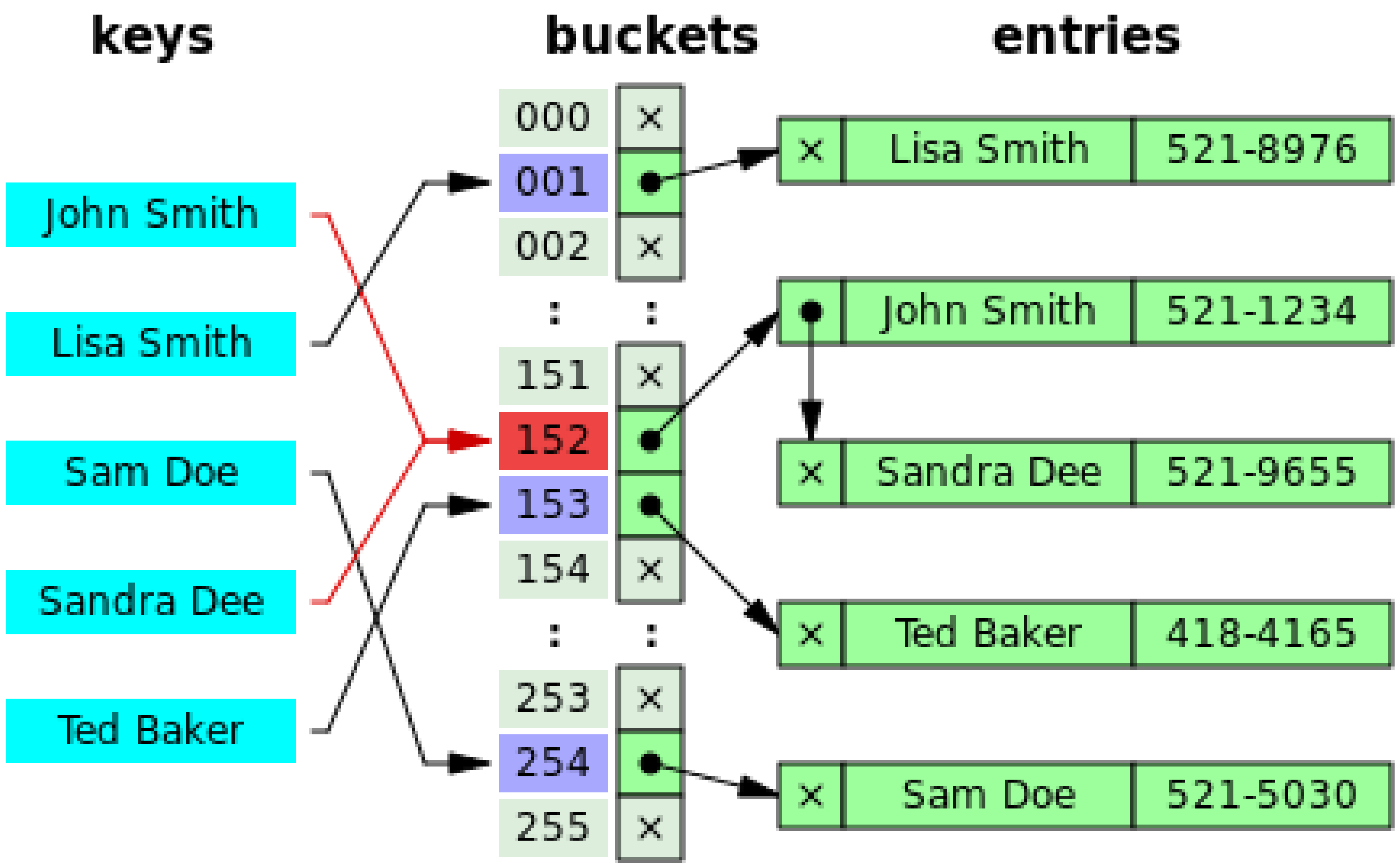
Content

1. Introduction
2. Hash Functions
3. Resolving Collisions
4. The Efficiency of Hashing

Introduction

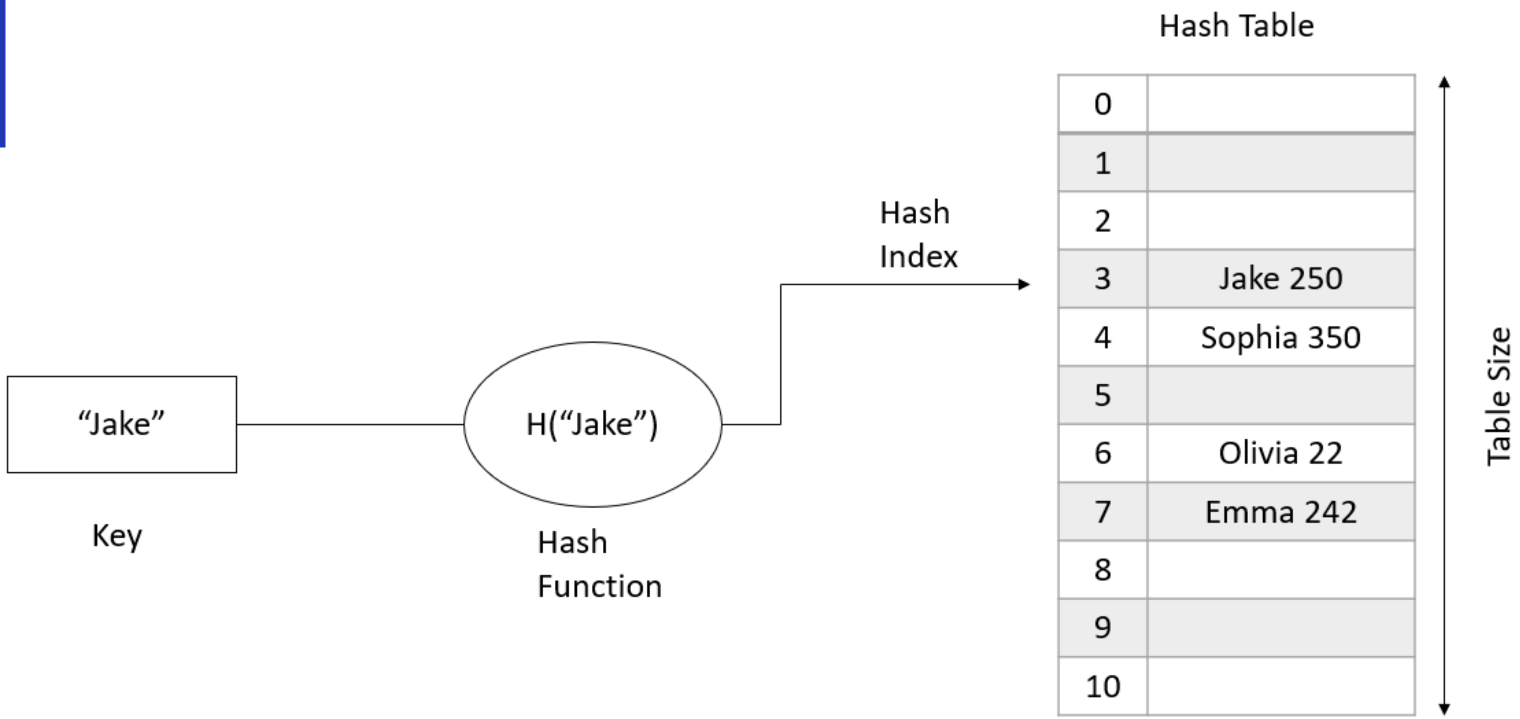
Introduction

- Binary search tree retrieval have order $O(\log_2 n)$
- Need a different strategy to locate an item
- Consider a "magic box" as an address calculator
  - Place/retrieve item from that address in an array
  - Ideally to a unique number for each key



Introduction

▪ Hashing is a technique to convert a range of key values into a range of indexes of an array.

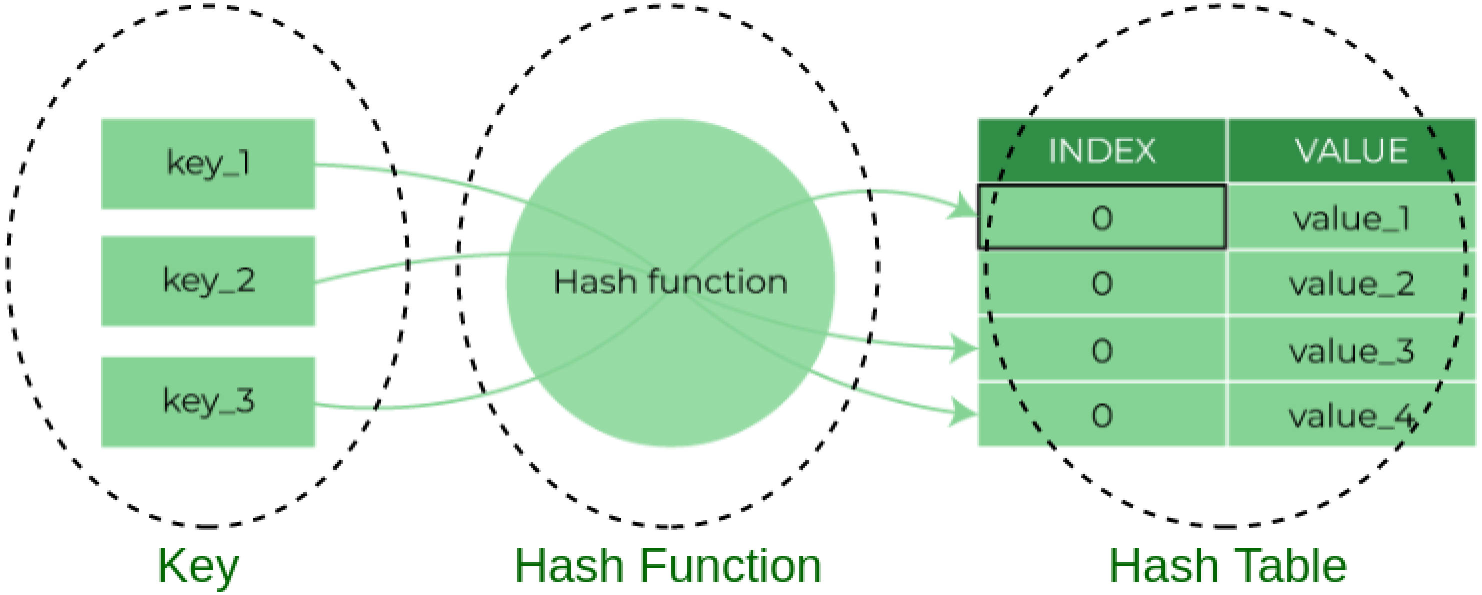


▪ Hashing is a technique to convert a range of key values into a range of indexes of an array.

▪ **Large keys** are converted into **small keys** by using **hash functions**

▪ The values are then stored in a data structure called **hash table**.

- Idea:
  - Distribute entries (key/value pairs) uniformly across an array.
  - Each element is assigned a key (converted key).

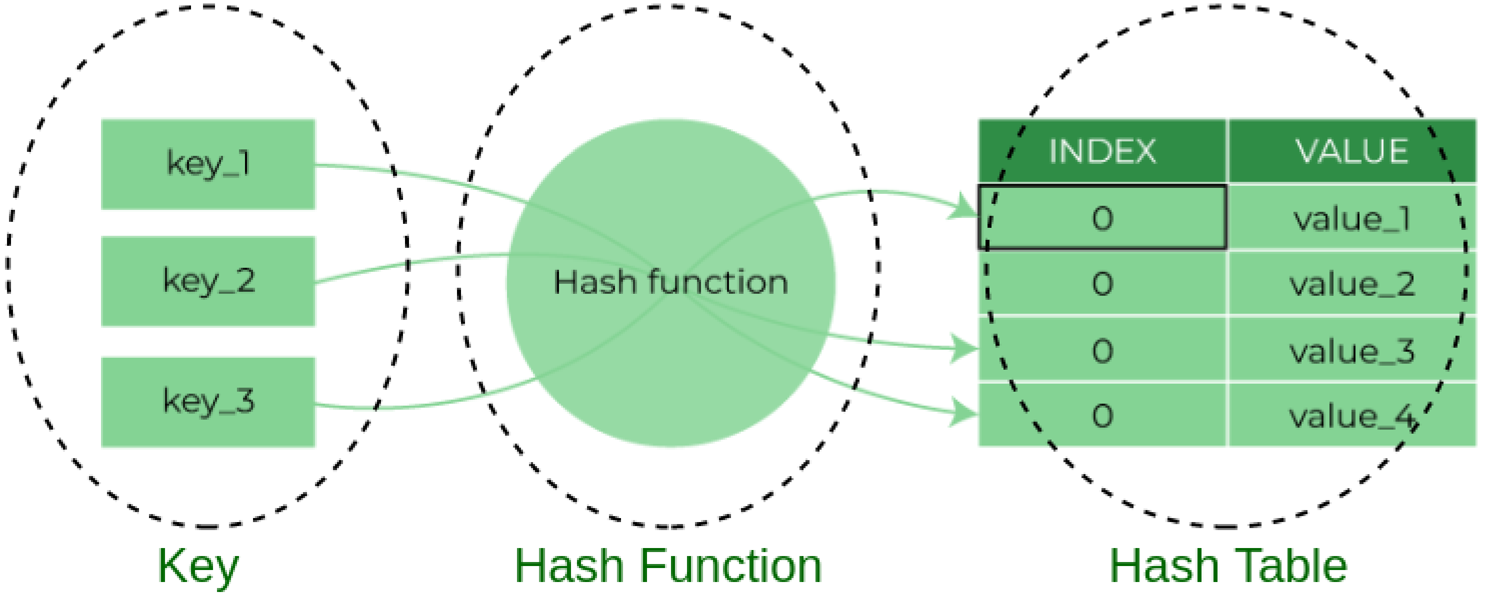


- Using that key to access the element in O(1) time. (The hash function computes an index suggesting where an entry can be found or inserted.)

Introduction

- A hash table is a data structure that is used to store keys/value pairs.

- It uses a hash function to compute an index into an array in which an element will be inserted or searched.



Hash Function

Hash Function

- Hash function is a mathematical function that can be used to map/converts a key to an integer value (an array index).

- The values returned by a hash function
  - hash values
  - hash codes
  - hash sums
  - digests
  - hashes.

Some Hash Functions

- Possible functions
  - Selecting digits
  - Folding
  - Modulo arithmetic
  - Converting a character string to an integer
    - Use ASCII values
    - Factor the results, Horner’s rule

Some Hash Functions

- Digit-selection:
  - Select some digits in the keys to create the hash value.
    - h(001**3**6482**5**) = 35
- Folding
  - h(001364825) = 0 + 0 + 1 + 3 + 6 + 4 + 8 + 2 + 5 = 29
  - h(**001**364**825**) = 001 + 364 + 825 = 1190
- Modulo arithmetic
  - h(Key) = Key mod 101
    - h(001364825) = 12

Some Hash Functions

- A string key hash function

$$h = \sum_{i=0}^{keylength} 128^i \times char(key[i])$$

Modulo Arithmetic

- Assume all keys are integers, and define

$$h(k) = k \; mod \; m$$

  - Where k is the key and m is the size of hash table
- Extreme deficiency: if $m = 2^r$, then the hash value doesn't even depend on all the bits of k
  - Ex: if k = 1011000111**011010** and $\mathrm{r} = 6$
  - then $h(k) = 011010$
- Size of hash table array should be a **prime** number

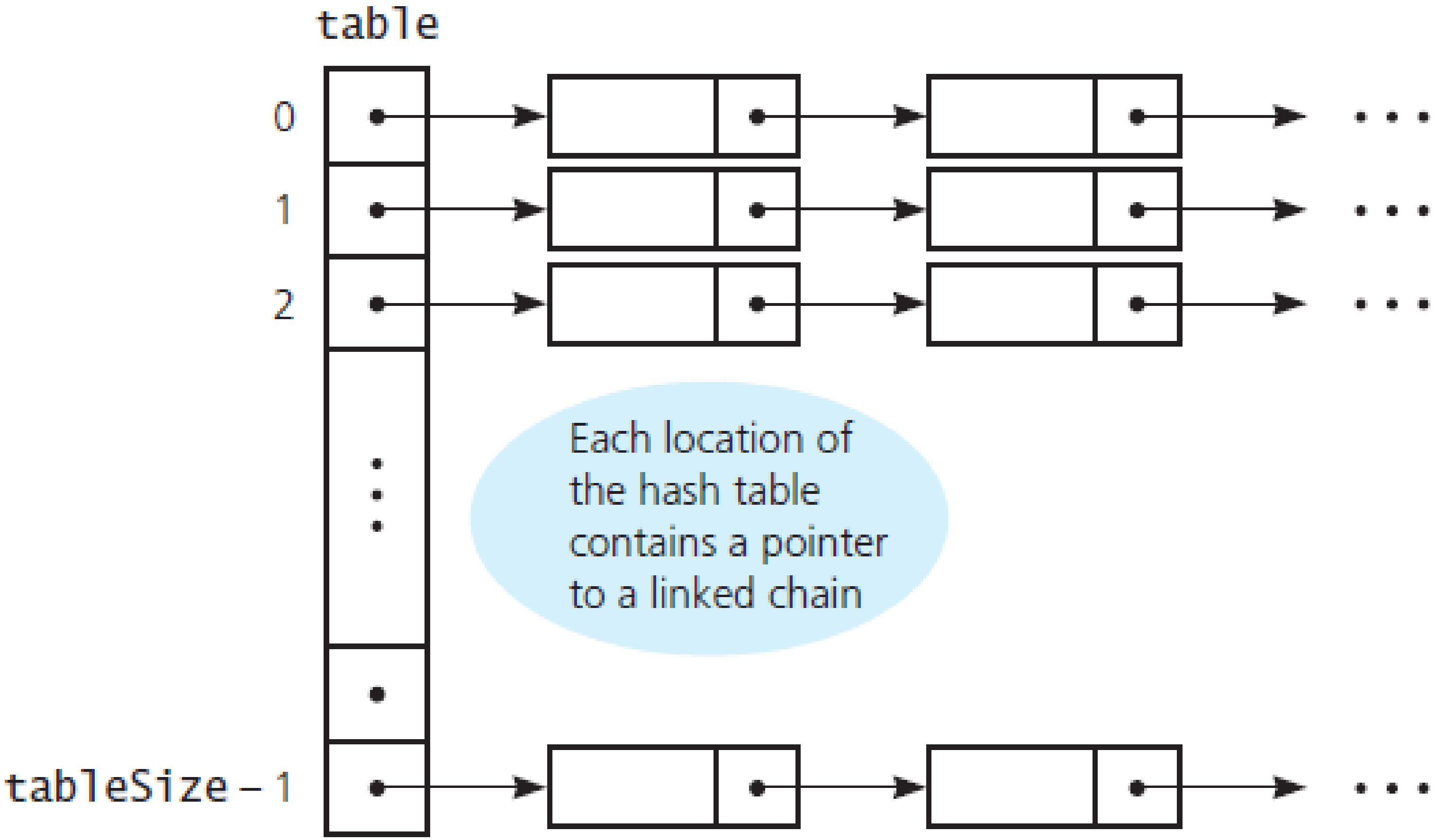
- Properties of good hash functions

Resolving Collisions

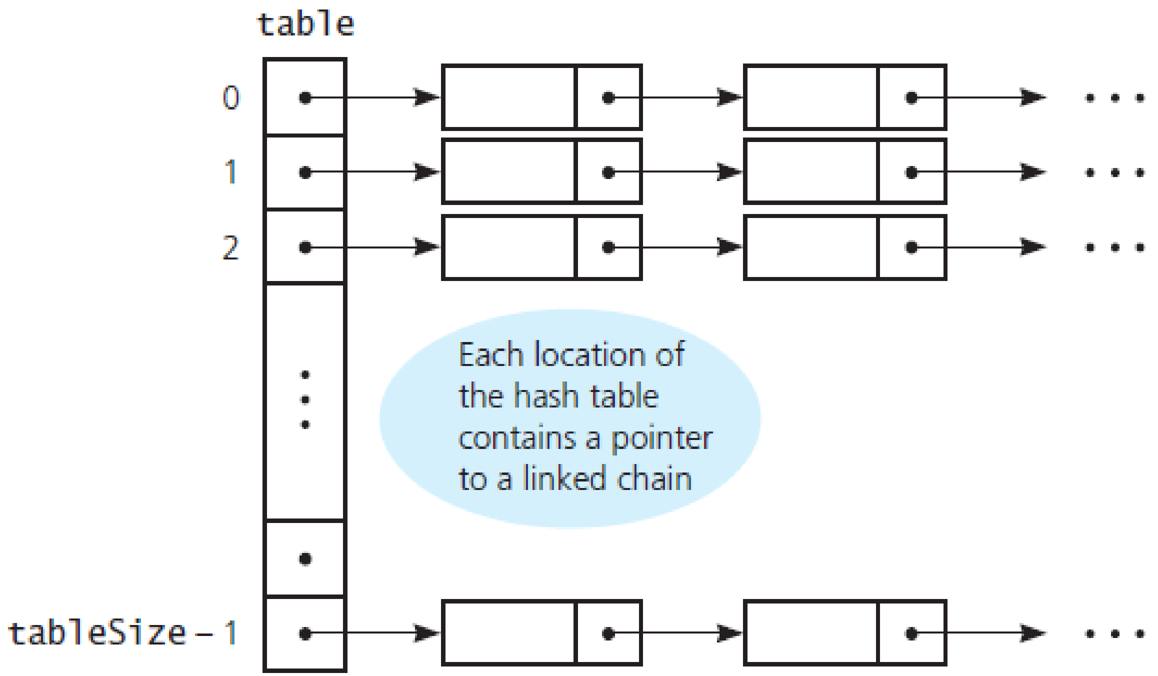
Collisions

- **Collision**: when two keys map to the **same location** in the hash table.
- Two ways to resolve collisions:
  - Separate Chaining – open hashing
  - Open Addressing – closed hashing
    - Linear probing
    - Quadratic probing
    - Double hashing

- Separate chaining: All keys that map to the same hash value are kept in a list (or "bucket")



- Each hash location can accommodate more than one item
- Each location is a "bucket" or an array itself
- Alternatively, design the hash table as an array of linked chains ("separate chaining").



Separate Chaining

- Give the hash function h(k) = k mod 10, distribute the list of integers

  10, 22, 107, 12, 42

  into the hash table

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Separate Chaining

- Give the hash function h(k) = k mod 10, distribute the list of integers

  10, 22, 107, 12, 42

  into the hash table

0	10
1	
2	22 → 12 → 42
3	
4	
5	
6	
7	107
8	
9	

Open Addressing

- Probe for another available location
- Some techniques:
  - Linear probing
  - Quadratic probing
  - Double hashing

Linear Probing

- In linear probing, the hash table is searched sequentially that starts from the original location of the hash.
- If in case the location that we get is **already occupied**, then we check for the next location
- Specifically,

$$\boldsymbol{H(k, step) = (h(k) + step) \bmod M}$$

Where:

- $step$ = 0, 1,...
- M : size of hash table

Linear Probing

fit@hcmus

- Given the hash function h(k) = k mod 10, we will distribute a list of integers

  10, 22, 107, 12, 42

  into a hash table. Additionally, we will employ linear probing to address any collisions that may arise during the process, using a step size of 1.

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

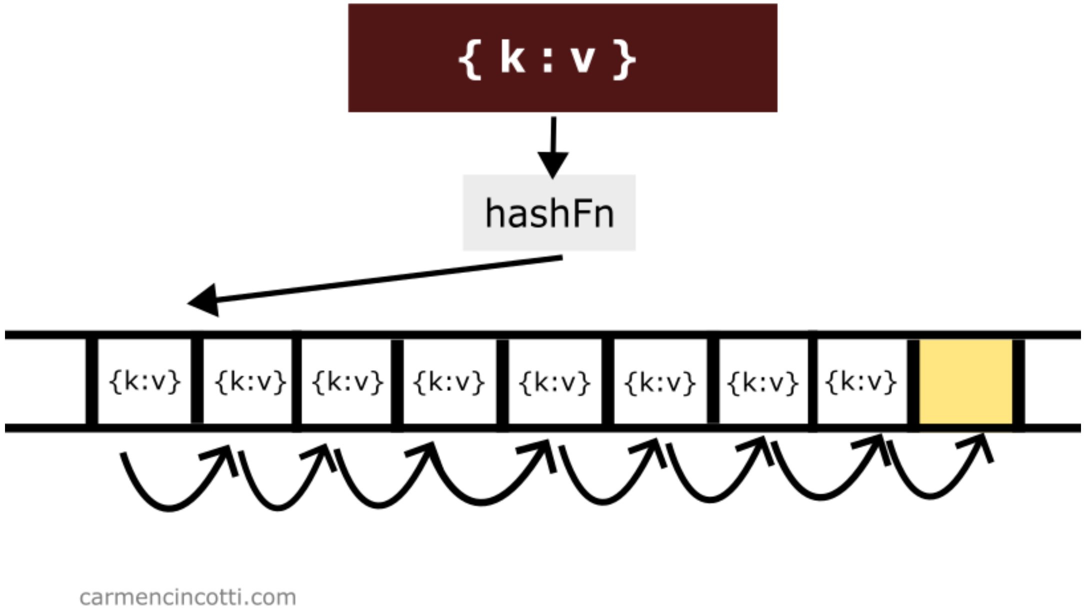
- Given the hash function h(k) = k mod 10, we will distribute a list of integers

  10, 22, 107, 12, 42

  into a hash table. Additionally, we will employ linear probing to address any collisions that may arise during the process, using a step size of 1.

0	10
1	
2	22
3	12
4	42
5	
6	
7	107
8	
9	

- Clustering is a phenomenon that occurs as elements are added to a hash table. Elements may have a tendency to clump together, forming clusters, which over time will significantly impact performance for searching and adding elements



Quadratic Probing

- Specifically,

$$H(k, step) = (h(k) + step^2) \bmod M$$

Where:

- $step$ = 0, 1,...
- M : size of hash table

Quadratic Probing

- Given the hash function h(k) = k mod 7, we will distribute a list of integers

  22, 30, 50, 37, 44

  into a hash table. Additionally, we will employ quadratic probing to address any collisions that may arise during the process

0	
1	
2	
3	
4	
5	
6	

Quadratic Probing

- Given the hash function h(k) = k mod 7, we will distribute a list of integers

  22, 30, 50, 37, 44

  into a hash table. Additionally, we will employ quadratic probing to address any collisions that may arise during the process

0	
1	22
2	30
3	37
4	
5	50
6	44

- Specifically,

$$H(k, step) = \left(h(k) + step * h_2(k)\right) mod\ M$$

Where:

  - $step$ = 0, 1,…
  - M : size of hash table
  - $h_2(.)$: the second hash function

Double Hashing

- Given the hash function $h(k) = k \ mod \ 7$, we will distribute a list of integers

  27, 43, 692, 72

  into a hash table. Additionally, we will employ double hashing to address any collisions that may arise during the process with the second hash function of $h_2(k) = 1 + (k \ mod \ 5)$

0	
1	
2	
3	
4	
5	
6	

Double Hashing

- Given the hash function $h(k) = k \; mod \; 7$, we will distribute a list of integers

  27, 43, 692, 72

  into a hash table. Additionally, we will employ double hashing to address any collisions that may arise during the process with the second hash function of $h_2(k) = 1 + (k \; mod \; 5)$

0	
1	43
2	692
3	
4	
5	72
6	27

- Some functions recommended in the literature:
  - $h_2(Key) = m - 2 - Key \, mod \, (m - 2)$
  - $h_2(Key) = 8 – (Key \, mod \, 8)$
  - $h_2(Key) = Key \, mod \, 97 + 1$

- **Advantages:**
  - Simple to implement.
  - Hash table never fills up, we can always add more elements to the chain.
  - Less sensitive to the hash function or load factors.
  - It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.

Separate Chaining

Disadvantages:

- Cache performance of chaining is not good as keys are stored using a linked list. Wastage of space (Some parts of hash table are never used)
- If the chain becomes long, then search time can become O(n) in the worst case.
- Uses extra space for links.

Open Addressing

- Removal requires specify state of an item
  - Occupied, emptied, removed → **Why?**
- Clustering is a problem
  - 2 keys have the same collision chain if their initial position is the same
- Double hashing can reduce clustering

Open Addressing

- Linear probing has the best cache performance but suffers from clustering. One more advantage of Linear probing is easy to compute.
- Quadratic probing lies between the two in terms of cache performance and clustering.
- Double hashing has poor cache performance but no clustering. Double hashing requires more computation time as two hash functions need to be computed.

The Efficiency of Hashing

The Efficiency of Hashing

- Efficiency of hashing involves the load factor alpha (α)

$$\alpha = \frac{\text{Current number of table items}}{tableSize}$$

The Efficiency of Hashing

- Efficiency of hashing involves the load factor alpha (α)
- The load factor is the average number of key-value pairs per bucket.

$$\alpha = \frac{\textit{Current number of table items}}{\textit{tableSize}}$$

- The higher the load factor, the slower the retrieval
- With open addressing, the load factor cannot exceed 1.
- With chaining, the load factor often exceeds 1

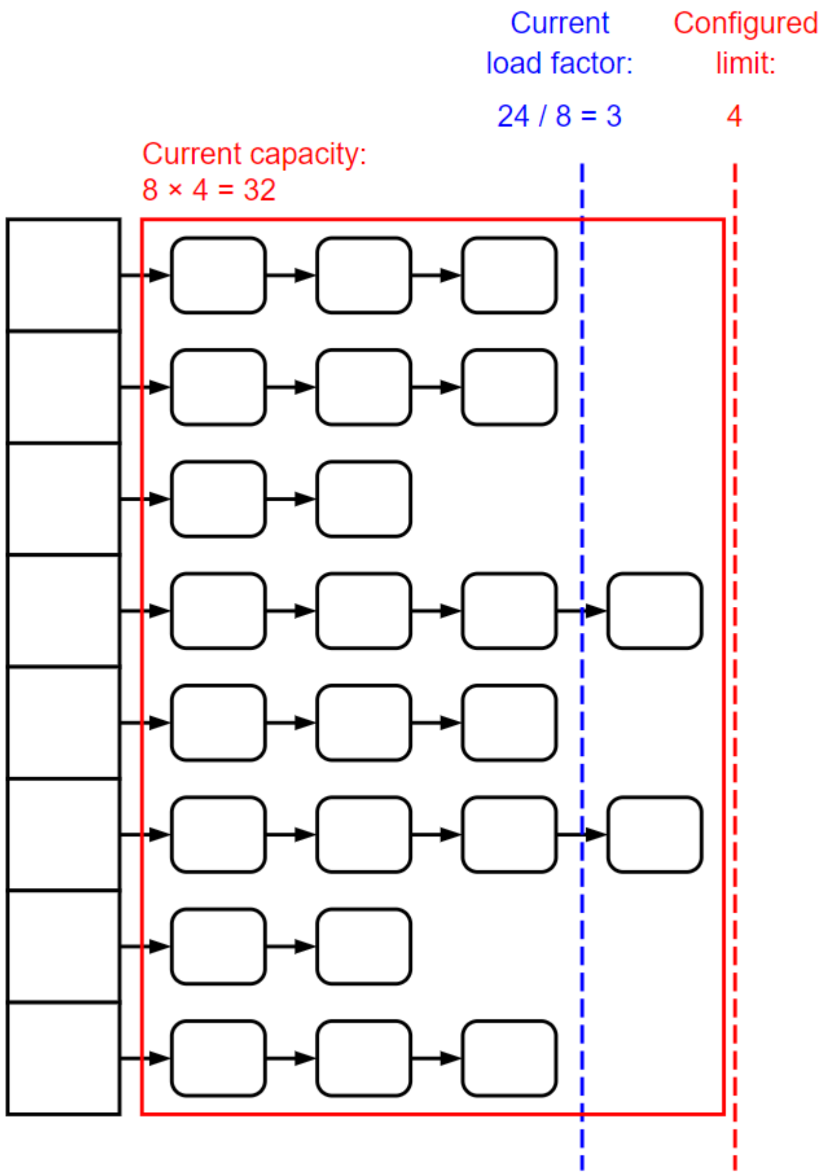
Capacity

- The capacity is the maximum number of key-value pairs for the given load factor limit and current bucket count

$$\text{capacity} = \text{number of buckets} \times \text{load factor limit}$$

The Efficiency of Hashing

- It is when the load factor reaches a given limit that rehashing kicks in.
- Since rehashing increases the number of buckets, it reduces the load factor
- The load factor limit is usually configurable and offers a **tradeoff** between **time** and **space** costs



The Efficiency of Hashing

- Linear probing – average value for α

$$\frac{1}{2}\left[1 + \frac{1}{1-\alpha}\right] \quad \text{for a successful search, and}$$

$$\frac{1}{2}\left[1 + \frac{1}{(1-\alpha)^2}\right] \quad \text{for an unsuccessful search}$$

- Quadratic probing and double hashing – efficiency for given α

$$\frac{-\log_e(1-\alpha)}{\alpha} \quad \text{for a successful search, and}$$

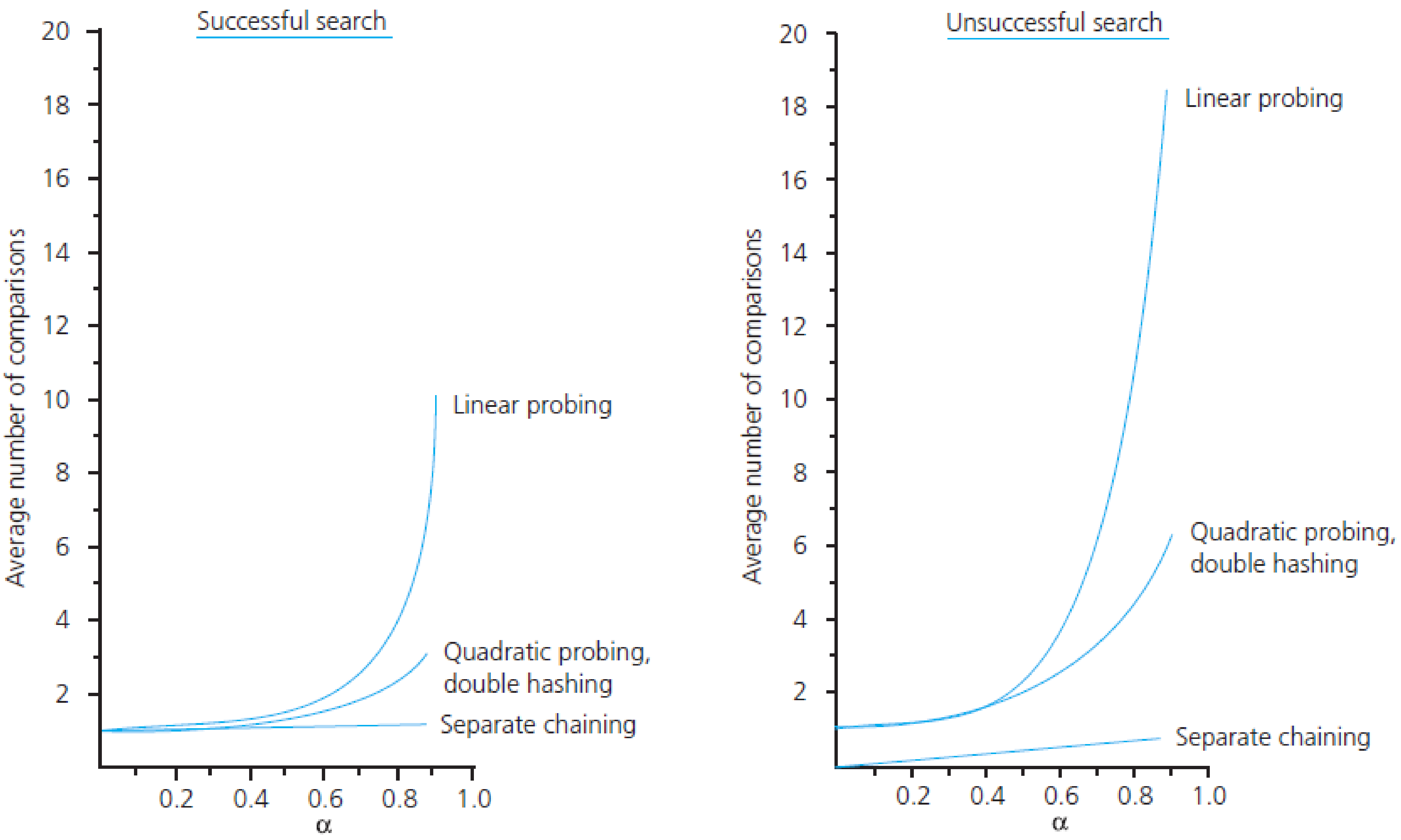
$$\frac{1}{1-\alpha} \quad \text{for an unsuccessful search}$$

- Separate chaining – efficiency for given α

$$1 + \frac{\alpha}{2} \qquad \text{for a successful search, and}$$

$$\alpha \qquad \text{for an unsuccessful search}$$

The Efficiency of Hashing



Maintaining Hashing Performance

- Collisions and their resolution typically cause the load factor α to increase
- To maintain efficiency, restrict the size of α
  - $\alpha \leq 0.5$ for open addressing
  - $\alpha \leq 1.0$ for separate chaining
- If load factor exceeds these limits
  - Increase size of hash table
  - Rehash with new hashing function

Exercise

- Given a hash table with m = 13 entries and the hash function

$$h(key) = key \ mod \ m$$

- Insert the keys {**10, 22, 31, 4, 15, 28, 17, 88, 59**} in the given order (from left to right) to the hash table. If there is a collision, use each of the following open addressing resolving methods:

  A. Linear probing

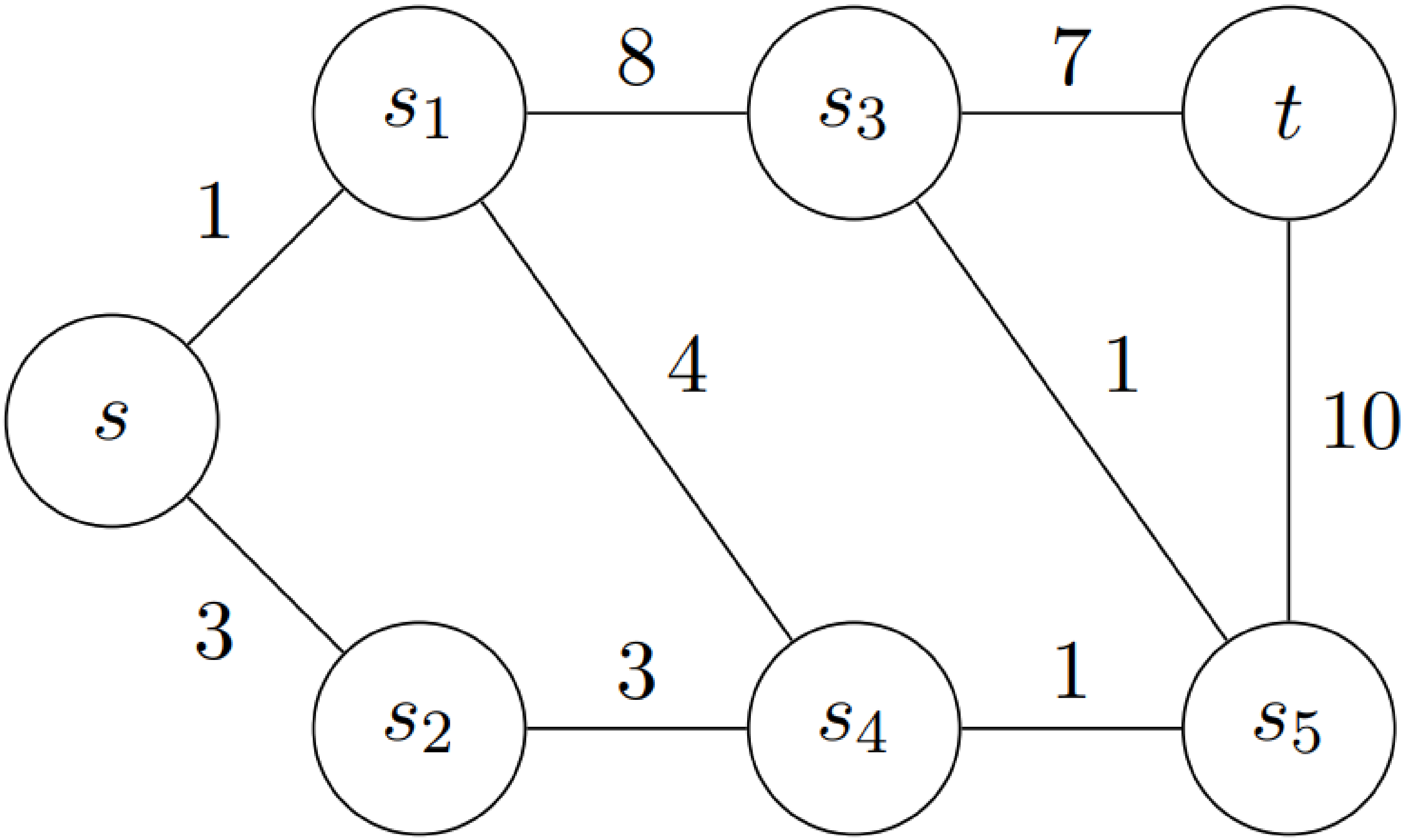
  B. Quadratic probing

  C. Double hashing with $h_2(key) = (key \ mod \ 7) + 1$

THANK YOU
for YOUR ATTENTION

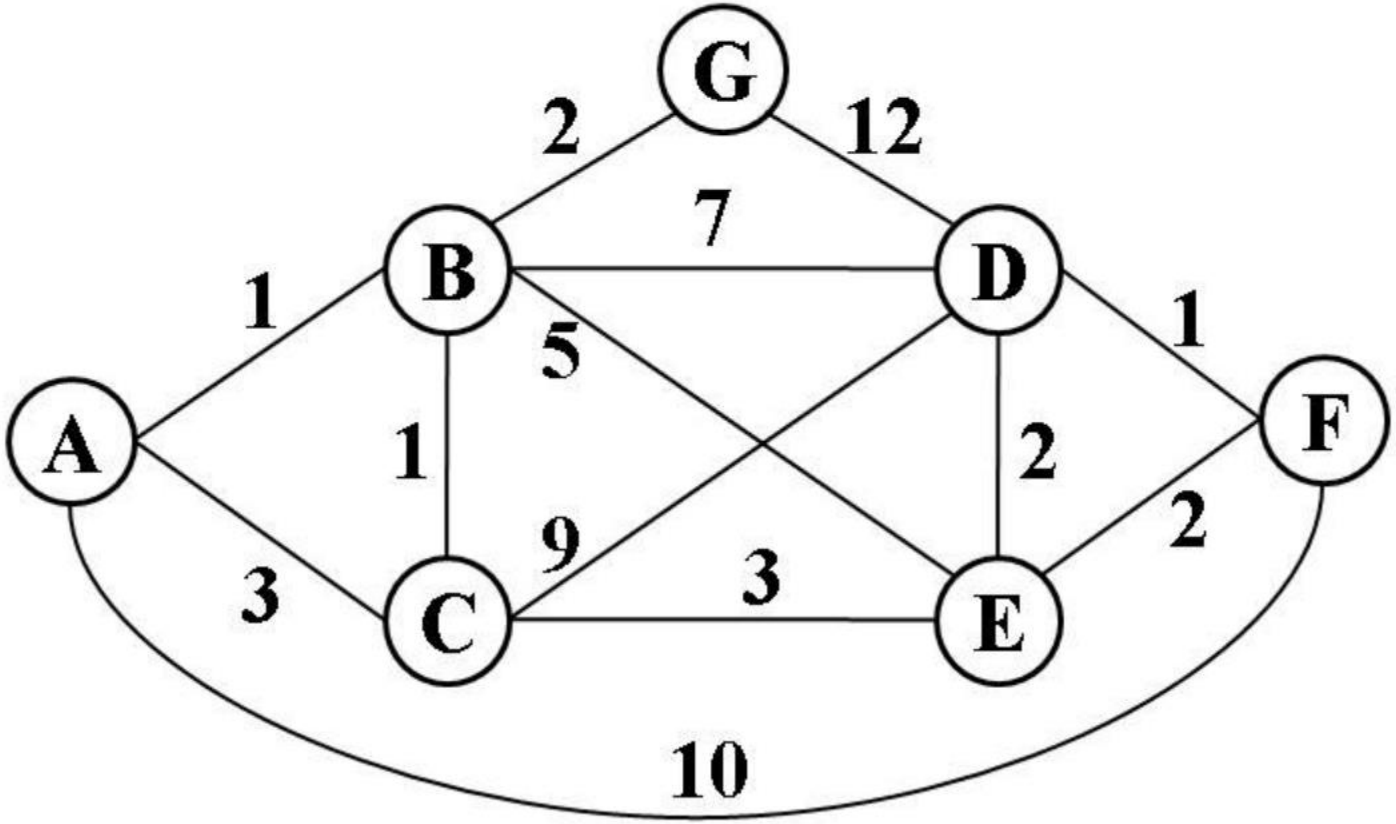
Exercise

- Apply Dijkstra's algorithm to the following graph to find the shortest path (and its cost) from s to the other vertices. Write down all intermediate steps



Exercise

- Apply Dijkstra's algorithm to the following graph to find the shortest path (and its cost) from **G** to the other vertices. Write down all intermediate steps



Exercise

- Apply Prim/ Kruskal's algorithm to find the minimum spanning tree of the graph