BM40A1500 DATA STRUCTURES AND ALGORITHMS

HASHING

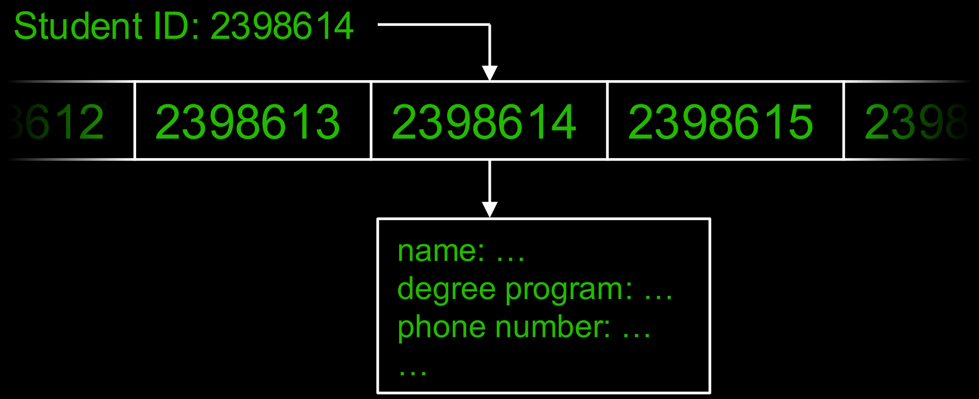
2024

INTRODUCTION

- Hashing is a method for storing and retrieving records from a database.
- Basic operations:
  - insert,
  - delete, and
  - search for records based on a search key value.
- When properly implemented, these operations can be performed in constant time.
  - All basic operations are Θ(1).
- A hash system stores records in an array called a hash table.
- The position (slot) of a record in the hash table is calculated using a hash function that takes the search key as input.
- Since hashing schemes place records in the table in whatever order satisfies the needs of the address calculation, records are not ordered by value.

EXAMPLE

- Database of student records.
  - 5000 students
- We want to access the records using the student ID.
  - Student ID: 0000000 - 9999999
- Solution 1: A list (array) with 10^7 elements.
  - One element for each possible student ID.



Student ID: 2398614

612	2398613	2398614	2398615	2398

name: …
degree program: …
phone number: …
…

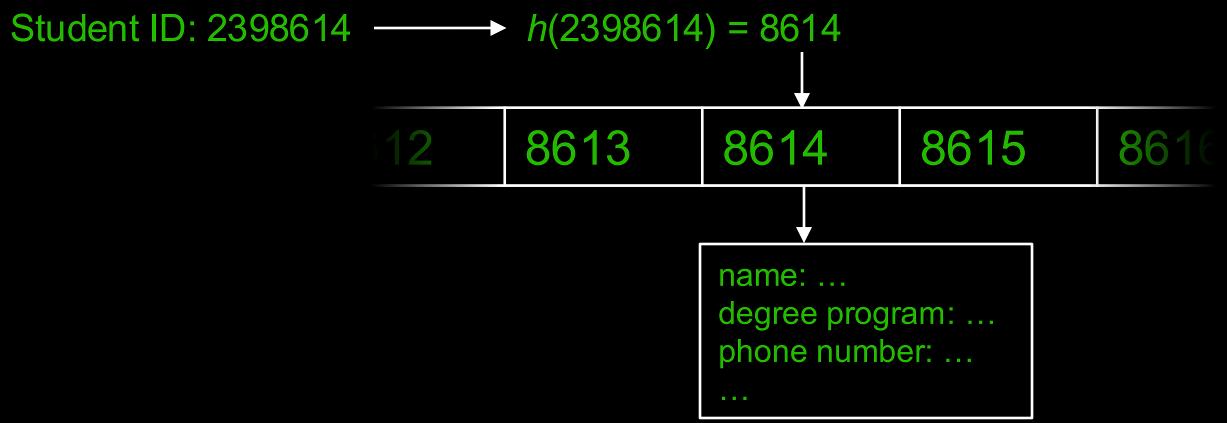
  - Waste of resources: only 0.05 % of the elements are in use.

EXAMPLE

- Solution 2: a hash table
  - A hash function (student ID as a key):

    $$h(x) = x \bmod 10^4$$

  - And a list (array) with 10^4 elements (slots).



Student ID: 2398614 → $h(2398614) = 8614$

12 | 8613 | 8614 | 8615 | 861

name: …
degree program: …
phone number: …
…

  - 50% of the elements in use → much more resource efficient.

EXAMPLE

- Solution 2: a hash table
  - A hash function (student ID as a key):

$$h(x) = x \bmod 10^4$$

  - And a list (array) with 10^4 elements (slots)
  - Collisions:

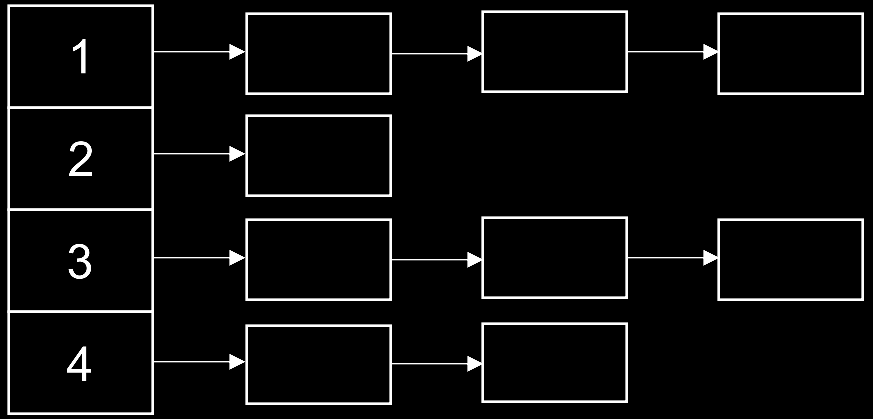
$$h(2814901) = 4901$$
$$h(6574901) = 4901$$

    - Cannot be fully avoided.

      → We need some solution to handle the collisions and allow storing the student records for all the students in the same hash table.

COLLISION RESOLUTION

- Open hashing
  - Collisions are stored outside the table
  - For example, a (linked) list in each slot → multiple records in each slot.

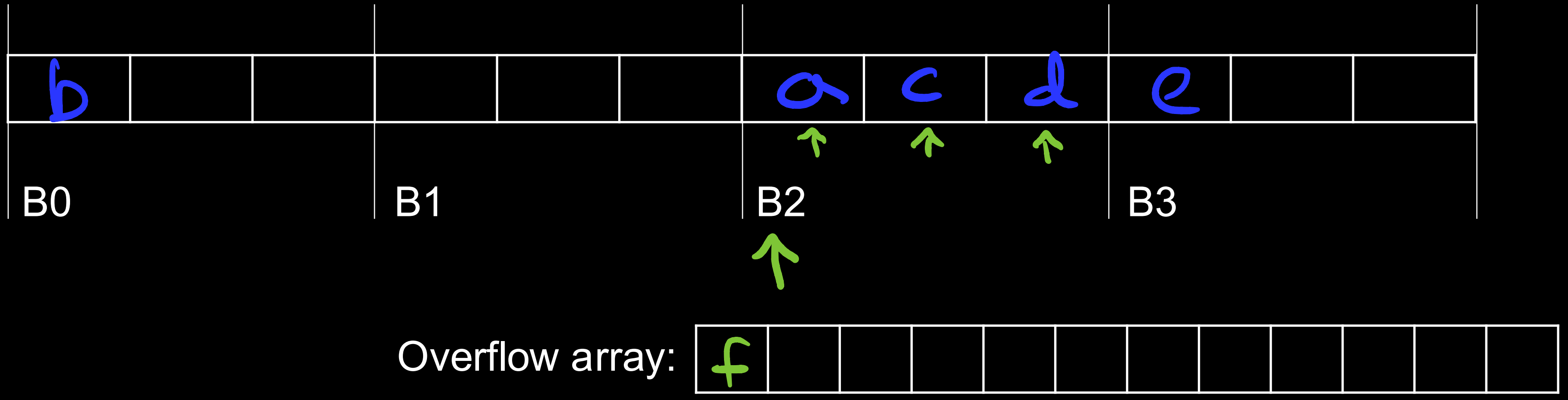


- Closed hashing
  - Each record is stored in the hash table.
  - If the "correct" (home) slot is occupied, a new slot is defined based on some collision resolution policy.

CLOSED HASHING: BUCKET HASHING

- The hash table is divided into **buckets** each containing multiple slots.
- The hash function points to the bucket and the record is stored in the first available slot in the bucket.
- New records that do not fit to the bucket are stored in the **overflow array**.

insert(f)
h(f)=2



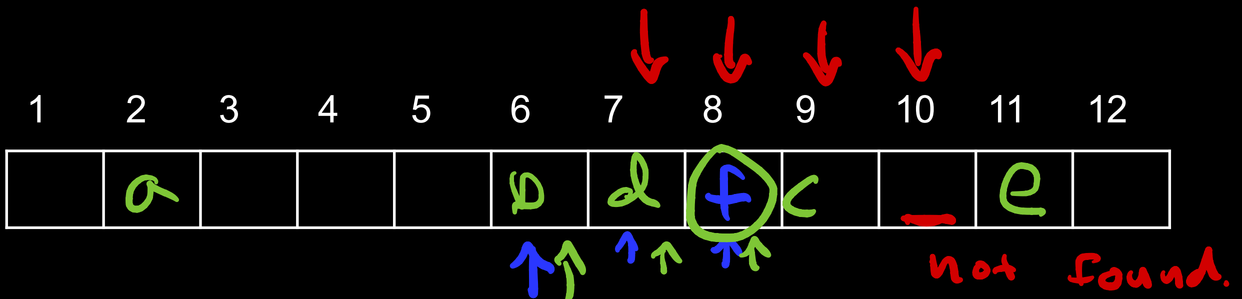
CLOSED HASHING

- Hashing with collision resolution policy
  - The most common way of implementing hashing.
  - The goal is to find a free slot in the hash table when the home position for the record is already occupied.
  - A sequence of hash table slots that can potentially hold the record.
    - The probe sequence (generated by some probe function).
    - The probe function returns an offset from the original home position.
    - When inserting, the probe sequence is followed until the key, or an empty slot is found.
  - Linear probing
    - collision resolution works by moving sequentially through the hash table from the home slot.
    - The probe function: *p(K, i) = i*

$P(K,i)$

$P(K,0)=0$ $P(K,1)=1$ $P(K,2)=2$

insert (f)
$h(f)=6$
search(f)



1	2	3	4	5	6	7	8	9	10	11	12
	a				b	d	f	c	_	e	

search(g)
$h(g)=7$

CLOSED HASHING: DELETION

- When deleting records from a hash table, there are two important considerations:
  1. Deleting a record must not hinder later searches.
     - The delete process cannot simply mark the slot as empty, because this will isolate records further down the probe sequence (search ends if empty slot is found).
  2. We do not want to make positions in the hash table unusable because of deletion.
     - The freed slot should be available to a future insertion.
- Solution: a special mark in place of the deleted record (tombstone):
  - When a tombstone is encountered when searching along a probe sequence, the search procedure continues with the search.
  - When a tombstone is encountered during insertion, that slot can be used to store the new record.
    - Note: we still need to verify that a duplicate is not in the table (can be after tombstone).

delete(c)
search(e)
h(e)=4

1	2	3	4	5	6	7	8	9	10	11	12
			a	b	T	d	e				
					f						

Insert(f)
h(f)=5

SELECTING HASH FUNCTION

- Collisions makes inserting, deleting, and searching for records slower.
  - → We want to use a hash function that minimize the number of collisions.
- The hash values produced by the hash function should be as evenly distributed as possible.
  - Understanding how our data (keys) are distributed.
  - For example:
    - Hash function: binning
      - All keys in the range 0 to 999 hash to slot 0, keys in the range 1000 to 1999 hash to slot 1, etc.
    - Keys: student IDs that are allocated in order.
    - First 1000 students have the student ID that hash to the same slot.
- See background material for the description of different hash functions.

LUT
University