Software Engineering
UA COLLEGE OF ENGINEERING

Module 5. Searching

DR. DIANA SALDANA
dianasaldana@arizona.edu
(928) 317-6313

THE UNIVERSITY OF ARIZONA

Unit 1. Searching Basics

THE UNIVERSITY OF ARIZONA

Searching

Software Engineering
UA COLLEGE OF ENGINEERING

- Organizing and retrieving information is at the heart of most computer applications, and searching is surely the most frequently performed of all computing tasks.
- Search can be viewed abstractly as a process to determine if an element with a particular value is a member of a particular set.
- The more common view of searching is an attempt to find the record within a collection of records that has a particular key value, or those records in a collection whose key values meet some criterion such as falling within a range of values.

THE UNIVERSITY OF ARIZONA

Searching

Software Engineering
UA COLLEGE OF ENGINEERING

- We can categorize search algorithms into three general approaches:
  - 1. Sequential and list methods.
  - 2. Direct access by key value (hashing).
  - 3. Tree indexing methods.

THE UNIVERSITY OF ARIZONA

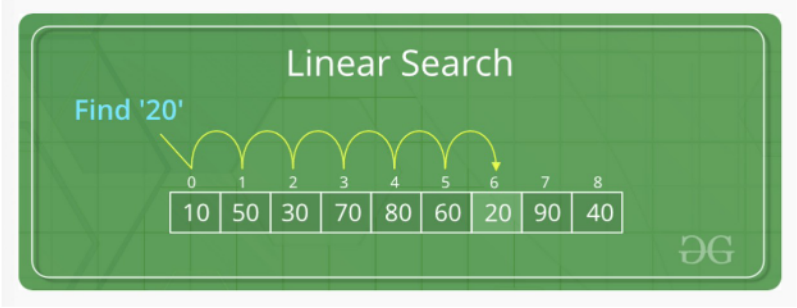
Searching

Software Engineering
UA COLLEGE OF ENGINEERING

- With all the vast amounts of data out there, it's important to be able to search for specific items efficiently.
- Search algorithms are often closely tied with data structures because how data is stored affects how it can be searched.

THE UNIVERSITY OF ARIZONA

Linear Search



Linear Search Algorithm

Linear Search is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set. It is the easiest searching algorithm.

Algorithm

Software Engineering
UA COLLEGE OF ENGINEERING

- Start from the leftmost element of *arr[]* and one by one compare *x* with each element of *arr[]*.
- If *x* matches with an element, return the index.
- If *x* doesn’t match with any of the elements, return -1.

THE UNIVERSITY OF ARIZONA

```
// Java code for linearly searching x in arr[]. If x
// is present then return its location, otherwise
// return -1

class GFG {
        public static int search(int arr[], int x)
        {
                int N = arr.length;
                for (int i = 0; i < N; i++) {
                        if (arr[i] == x)
                                return i;
                }
                return -1;
        }

        // Driver's code
        public static void main(String args[])
        {
                int arr[] = { 2, 3, 4, 10, 40 };
                int x = 10;

                // Function call
                int result = search(arr, x);
                if (result == -1)
                        System.out.print(
                                "Element is not present in array");
                else
                        System.out.print("Element is present at index "
                                         +
result);
        }
}
```

Linear Search Recursive Approach

Software Engineering
UA COLLEGE OF ENGINEERING

- If the size of the array is zero then, return -1, representing that the element is not found. This can also be treated as the base condition of a recursion call.
- Otherwise, check if the element at the current index in the array is equal to the key or not i.e., arr[size-1] == key.
  - If equal, then return the index of the found key.

THE UNIVERSITY OF ARIZONA

```java
// Java Recursive Code For Linear Search
import java.io.*;

class Test {
		static int arr[] = { 5, 15, 6, 9, 4 };

		// Recursive Method to search key in the array
		static int linearsearch(int arr[], int size, int key)
		{
			if (size == 0) {
				return -1;
			}
			else if (arr[size - 1] == key) {
				// Return the index of found key.
				return size - 1;
			}
			else {
				return linearsearch(arr, size - 1, key);
			}
		}

		// Driver method
		public static void main(String[] args)
		{
			int key = 4;

			// Function call to find key
			int index = linearsearch(arr, arr.length, key);
			if (index != -1)
				System.out.println(
					"The element " + key + " is found at "
					+ index + " index of the given array.");

			else
				System.out.println("The element " + key + " is not found.");
		}
}
```

Linear Search on an unsorted array

Time and Space Complexity

Software Engineering
UA COLLEGE OF ENGINEERING

- **Best Case:** In the best case, the key might be present at the first index. So, the best-case complexity is Ω(1).
- **Average Case:** Ø(n)
- **Worst Case:** In the worst case, the key might be present at the last index i.e., opposite to the end from which the search has started in the list. So, the worst-case complexity is O(n) where n is the size of the list.
- **Space Complexity:** O(1) as except for the variable to iterate through the list, no other variable is used.

THE UNIVERSITY OF ARIZONA

Java Implementation

```java
import java.util.Scanner;
public class LinearSearch {
   public static int linearSearch(int [] numbers, int key) {
      int i;

      for (i = 0; i < numbers.length; ++i) {
         if (numbers[i] == key) {
            return i;
         }
      }

      return -1; /* not found */
   }

   public static void main(String [] args) {
      Scanner scnr = new Scanner(System.in);
      int [] numbers = {2, 4, 7, 10, 11, 32, 45, 87};
      int i;
      int key;
      int keyIndex;

      System.out.print("NUMBERS: ");
      for (i = 0; i < numbers.length; ++i) {
         System.out.print(numbers[i] + " ");
      }
      System.out.println();

      System.out.print("Enter a value: ");
      key = scnr.nextInt();

      keyIndex = linearSearch(numbers, key);

      if (keyIndex == -1) {
         System.out.println(key + " was not found.");
      }
      else {
         System.out.println("Found " + key + " at index " + keyIndex + ".");
      }
   }
}
```

Binary Search

Binary Search

Software Engineering
UA COLLEGE OF ENGINEERING

- Binary Search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half.
- The idea of binary search is to use the information that the array is sorted and reduce the time complexity to ***O(log n)***.

THE UNIVERSITY OF ARIZONA

Algorithm

Software Engineering
UA COLLEGE OF ENGINEERING

- Begin with the mid element of the whole array as a search key.
- If the value of the search key is equal to the item then return an index of the search key.
- Or if the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
- Otherwise, narrow it to the upper half.
- Repeatedly check from the second point until the value is found or the interval is empty.

THE UNIVERSITY OF ARIZONA

Algorithm

Software Engineering
UA COLLEGE OF ENGINEERING

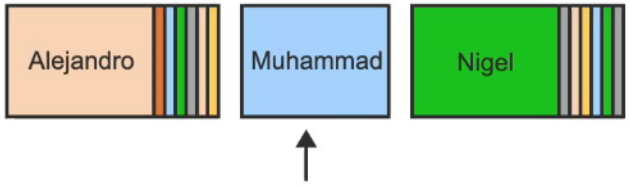
- 1. Compare *x* with the middle element.
- 2. If *x* matches with the middle element, we return the mid index.
- 3. Else if *x* is greater than the mid element, then *x* can only lie in the right half subarray after the mid element. So, we recur for the right half.
- 4. Else (*x* is smaller) recur for the left half.

THE UNIVERSITY OF ARIZONA

Binary Search

- Linear search may require searching all list elements, which can lead to long runtimes. For example, searching for a contact on a smartphone one-by-one from first to last can be time consuming.
- Because a contact list is sorted, a faster search, known as binary search, checks the middle contact first.
- If the desired contact come alphabetically before the middle contact, binary search will then search the first half and otherwise the last half.
- Each step reduces the contacts that need to be searched by half.

1 2 3 4 2x speed



A contact list stores contacts sorted by name. Searching for Pooja using a binary search starts by checking the middle contact.

Captions

1. A contact list stores contacts sorted by name. Searching for Pooja using a binary search starts by checking the middle contact.
2. The middle contact is Muhammad. Pooja is alphabetically after Muhammad, so the binary search only searches the contacts after Muhammad. Only half the contacts now need to be searched.
3. Binary search continues by checking the middle element between Muhammad and the last contact. Pooja is before Sharod, so the search continues with only those contacts between Muhammad and Sharod, which is one fourth of the contacts.
4. The middle element between Muhammad and Sharod is Pooja. Each step reduces the number of contacts to search by half.

Feedback?

Binary search

Software Engineering
UA COLLEGE OF ENGINEERING

- Binary search is a faster algorithm for searching a list if the list's elements are sorted and directly accessible (such as an array).
- Binary search first checks the middle element of the list.
- If the search key is found, the algorithm returns the matching location.
- If the search key is not found, the algorithm repeats the search on the remaining left sublist (if the search key was less than the middle element) or the remaining right sublist (if the search key was greater than the middle element).

THE UNIVERSITY OF ARIZONA

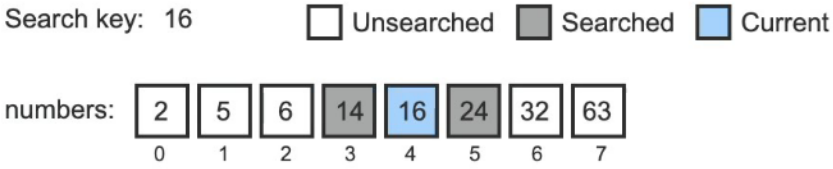
Binary search on a sorted array

Software Engineering
UA COLLEGE OF ENGINEERING

THE UNIVERSITY OF ARIZONA

PARTICIPATION ACTIVITY | 17.2.3: Binary search efficiently searches sorted list by reducing the search space by half each iteration.

Start 2x speed



Captions

1. Elements with indices between low and high remain to be searched.
2. Search starts by checking the middle element.
3. If search key is greater than element, then only elements in right sublist need to be searched.
4. Each iteration reduces search space by half. Search continues until key found or search space is empty.

Feedback?

Binary Search

Software Engineering
UA COLLEGE OF ENGINEERING

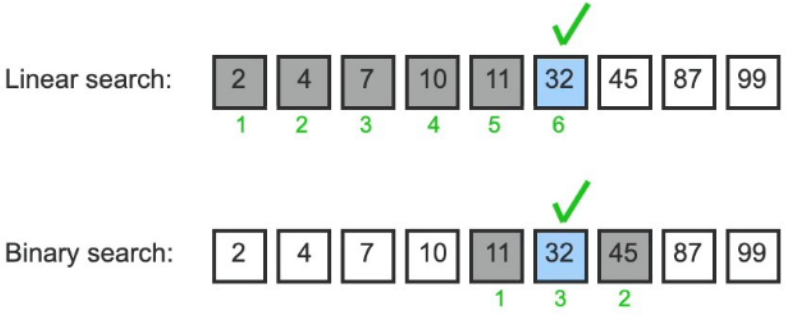
- Binary search is incredibly efficient in finding an element within a sorted list.
- During each iteration or step of the algorithm, binary search reduces the search space by half.
- The search terminates when the element is found or the search space is empty (element not found).

THE UNIVERSITY OF ARIZONA

PARTICIPATION ACTIVITY | 17.2.5: Speed of linear search versus binary search to find a number within a sorted list.

Start ☐ 2x speed

Search key: 32 ☐ Unsearched ☐ Searched ☐ Current

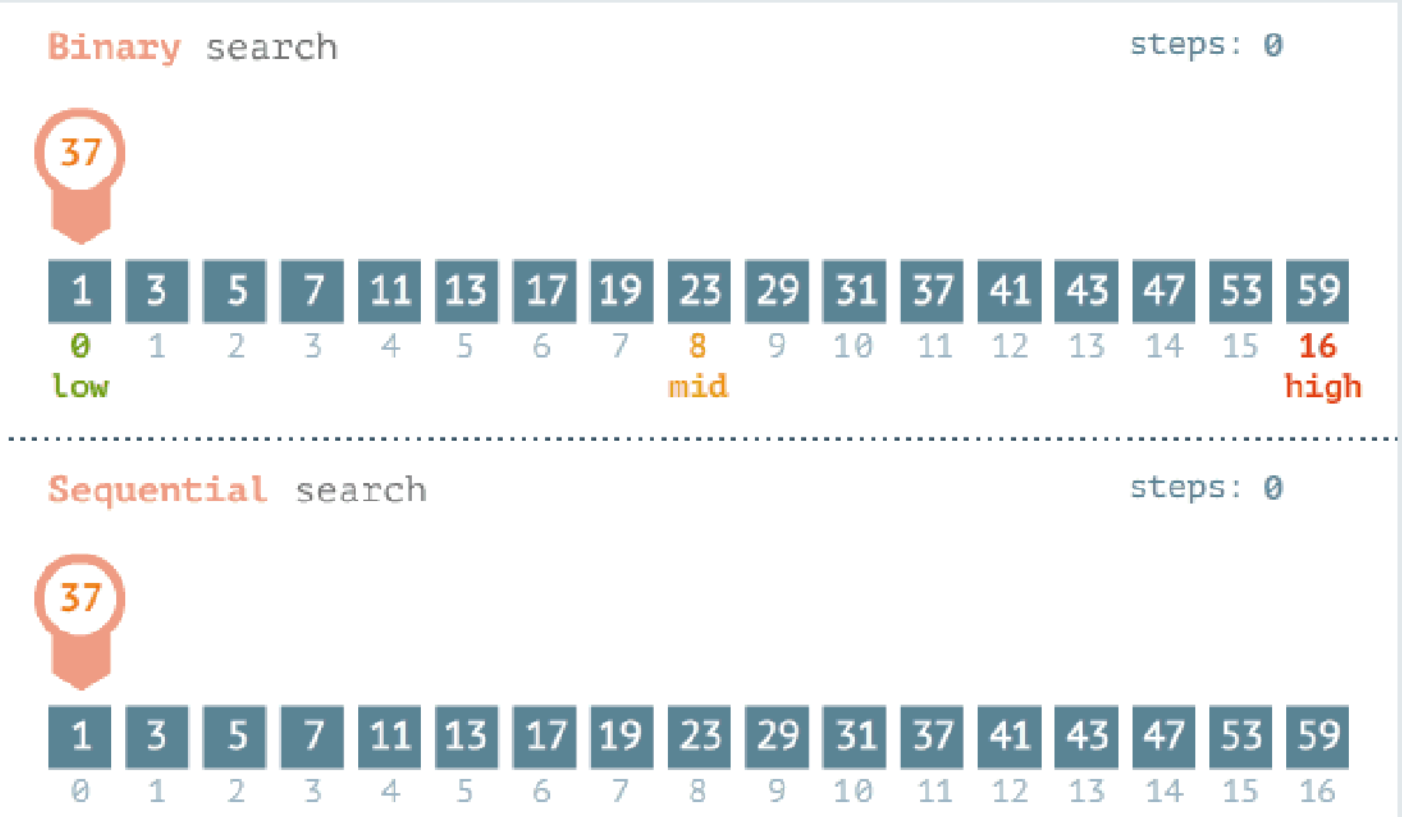


Captions

1. A binary search begins with the middle element of the list. Each subsequent search reduces the search space by half. Using binary search, a match was found with only 3 comparisons.
2. Using linear search, a match was found after 6 comparisons. Compared to a linear search, binary search is incredibly efficient in finding an element within a sorted list.

Feedback?

Linear Search vs Binary Search


Binary search
steps: 0
37
1 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
low
mid
high
Sequential search
steps: 0
37
1 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Java Implementation


```java
import java.util.Scanner;

public class BinarySearch {
   public static int binarySearch(int [] numbers, int key) {
      int mid;
      int low;
      int high;

      low = 0;
      high = numbers.length - 1;

      while (high >= low) {
         mid = (high + low) / 2;
         if (numbers[mid] < key) {
            low = mid + 1;
         }
         else if (numbers[mid] > key) {
            high = mid - 1;
         }
         else {
            return mid;
         }
      }

      return -1; // not found
   }

   public static void main(String [] args) {
      Scanner scnr = new Scanner(System.in);
      int [] numbers = {2, 4, 7, 10, 11, 32, 45, 87};
      int i;
      int key;
      int keyIndex;

      System.out.print("NUMBERS: ");
      for (i = 0; i < numbers.length; ++i) {
         System.out.print(numbers[i] + " ");
      }
      System.out.println();

      System.out.print("Enter a value: ");
      key = scnr.nextInt();

      keyIndex = binarySearch(numbers, key);

      if (keyIndex == -1) {
         System.out.println(key + " was not found.");
      }
      else {
         System.out.println("Found " + key + " at index " + keyIndex + ".");
      }
   }
}
```

Time and Space Complexity

- **Best Case:** Ω(1). Best case is when the element is at the middle index of the array. It take only one comparison to find the target element.
- **Average Case:** Ø(log n)
- **Worst Case:** O(log n). It is if the element is present in the first position.
- **Space Complexity:** O(1). Iterative algorithm that does not require any additional data structures.

*Consider array **arr[]** of length **N** and element **X** to be found. There can be two cases:*

- ***Case1:** Element is present in the array*
- ***Case2:** Element is not present in the array.*

*There are **N** Case1 and **1** Case2. So total number of cases = **N+1**. Now notice the following:*

- *An element at index N/2 can be found in **1** comparison*
- *Elements at index N/4 and 3N/4 can be found in **2** comparisons.*
- *Elements at indices N/8, 3N/8, 5N/8 and 7N/8 can be found in **3** comparisons and so on.*

- *1 comparison = 1*
- *2 comparisons = 2*
- *3 comparisons = 4*
- ***x** comparisons = 2^{x-1} where **x** belongs to the range **[1, logN]** because maximum comparisons = maximum time N can be halved = maximum comparisons to reach 1st element = logN.*

Searching Practice (working in-pairs)

THE UNIVERSITY OF ARIZONA

Java implementation

Software Engineering
UA COLLEGE OF ENGINEERING

- USING THE SORTING AND SEARCHING ALGORITHMS IN D2L…

- 1. Find the minimum element in an unsorted array using Linear Search.
- 2. Find the minimum element in an unsorted array using Binary Search.
- Test your code with the following arrays:
- Array 1: [100 775 146 483 789 990 844 926 89 416 540 470 747 398 608 356 84 315 971 834 591 447]
- Array 2: [915 550 369 666 100 435 638 342 440 780 902 901 329 115 29 361 930 505 439 75 322]

THE UNIVERSITY OF ARIZONA

Unit 2. Hashtables

Hashtables

- Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects. Some examples of how hashing is used in our lives include:
- In universities, each student is assigned a unique roll number that can be used to retrieve information about them.
- In libraries, each book is assigned a unique number that can be used to determine information about the book, such as its exact position in the library or the users it has been issued too.

Hashtables

Software Engineering
UA COLLEGE OF ENGINEERING

- In hashing, large keys are converted into small keys by using hash functions. The values are then stored in a data structure called hash table.
- The idea of hashing is to distribute entries (key, value pairs) uniformly across an array.
- By using that key, you can access the element in O(1) time.

THE UNIVERSITY OF ARIZONA

Hashtables

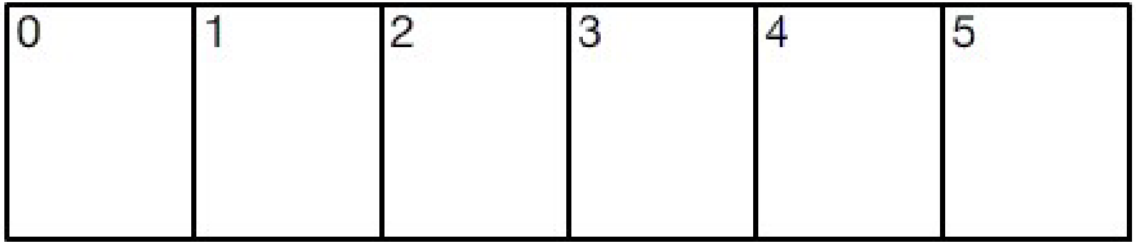
Software Engineering
UA COLLEGE OF ENGINEERING

- 1. An element is converted into an integer by using a hash function. This element can be used as an index to store the original element, which falls into the hash table.
- 2. The element is stored in the hash table where it can be quickly retrieve using hashed key
  - Hash = hashfunc(key)
  - Index = hash % array_size

THE UNIVERSITY OF ARIZONA

Insert:

Key: lion
Value: yellow



Features of Hashtable

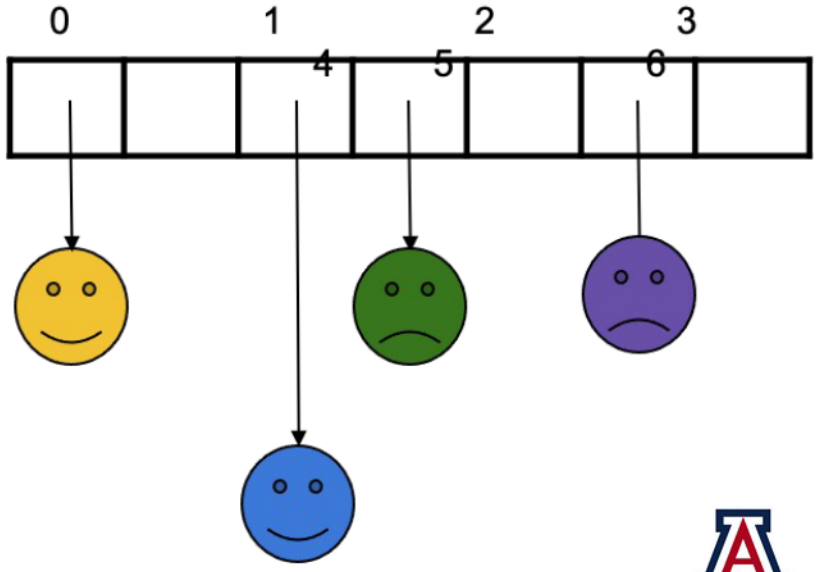
Software Engineering
UA COLLEGE OF ENGINEERING

- It is similar to HashMap, but is synchronized.
- Hashtable stores key/value pair in hash table.
- In Hashtable we specify an object that is used as a key, and the value we want to associate to that key.
- The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.

THE UNIVERSITY OF ARIZONA

Use of hashtable

- …we could just use an array to store the data
- …then finding an item would be EASY because the key would just be the index!
- If you wanted to add a new (key, value) pair, that would be EASY (as long as there isn't already an item with that key)!



Internal Working of Hashtable

Software Engineering
UA COLLEGE OF ENGINEERING

- It makes use of hashCode() method to determine which bucket the key/value pair should map.
- The hash function helps to determine the location for a given key in the bucket list.
- Generally, hashcode is a non-negative integer that is equal for equal Objects and may or may not be equal for unequal Objects. To determine whether two objects are equal or not, hashtable makes use of the equals() method.

THE UNIVERSITY OF ARIZONA

Internal Working of Hashtable

Software Engineering
UA COLLEGE OF ENGINEERING

- It is possible that two unequal Objects have the same hashcode. This is called a collision. To resolve collisions, hashtable uses an array of lists. The pairs mapped to a single bucket (array index) are stored in a list reference is stored in the array index.

- **K-** The type of the keys in the map.
- **V-** The type of values mapped in the map.

THE UNIVERSITY OF ARIZONA

Time-Space Tradeoff

Software Engineering
UA COLLEGE OF ENGINEERING

- If we had unlimited space, we could use a huge array and have **unique** integer keys for all possible data items (which may or may not be used!)
- If we had unlimited time, we could use an unordered array and just use sequential search (O(N)) to find an item by mapping each of the N keys to a **unique** index.

THE UNIVERSITY OF ARIZONA

A Compromise

- Since neither space nor time are unlimited, we can make a sort of compromise between the two and create a new structure called a *hashtable*.
- We will limit the space but we will use it wisely so that the searches are still possible in a reasonable amount of time.

Hashtable Overview

Software Engineering
UA COLLEGE OF ENGINEERING

- One way to implement an *unordered* symbol table
- Two parts:
  - Hash function **h(k)** that turns a key **k** into an integer (the result of *h(k)* is called the *hash value* of *k*.
  - An array (called a table) of size **M**
- Main idea: Store item **(k, v)** at index **i = h(k)** in the array
- That means we need a hash function that turns **k** into an integer in the interval [0, **M** - 1].

THE UNIVERSITY OF ARIZONA

Example

Software Engineering
UA COLLEGE OF ENGINEERING

Table: M = 11 (possible indices: [0, 10])

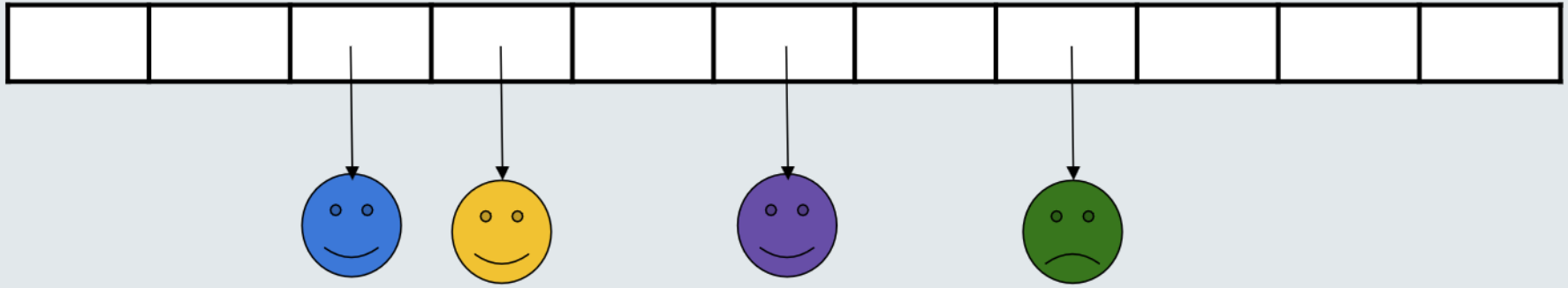
Values:

Keys: yellow, green, purple, blue

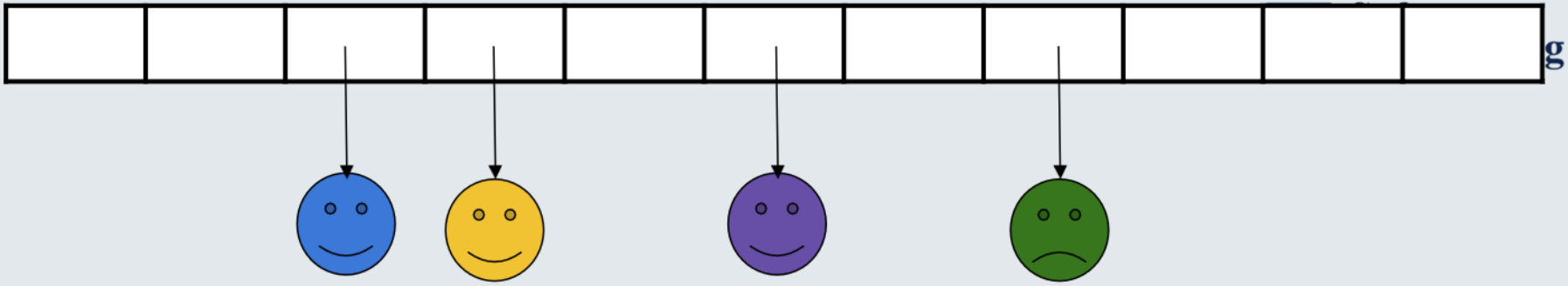
Hash Function: Let x be the numerical position in the alphabet of the first letter in the key. Return *x mod 11*.

h(yellow) = 25 mod 11 = 3
h(green) = 7 mod 11 = 7
h(purple) = 16 mod 11 = 5
h(blue) = 2 mod 11 = 2

THE UNIVERSITY OF ARIZONA



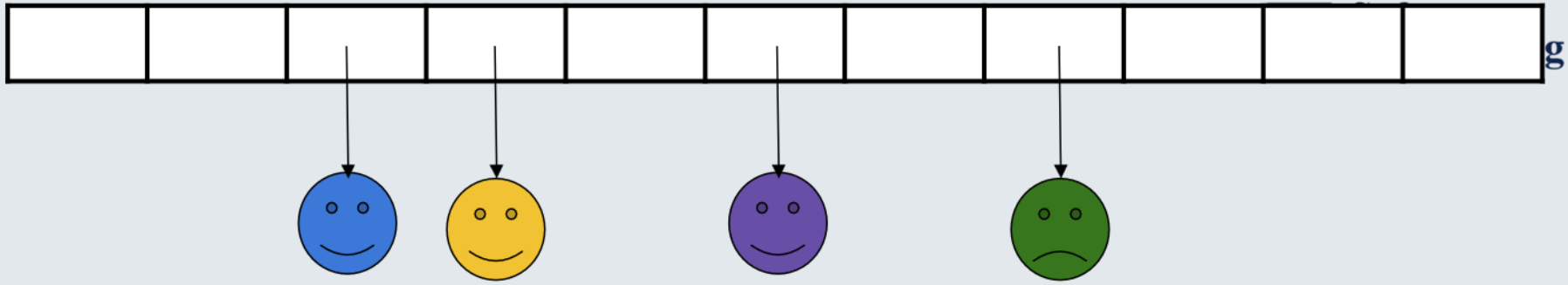
- **put(yellow,):**
  - calculate h(yellow) = 3
  - a[3] =
- **get(green):**
  - calculate h(green) = 7
  - return a[7]
- **delete(blue):**
  - calculate h(blue) = 2
  - a[2] = null



- **put(yellow,):**
  - calculate h(yellow) = 3
  - a[3] =
- **get(green):**
  - calculate h(green) = 7
  - return a[7]
- **delete(blue):**
  - calculate h(blue) = 2
  - a[2] = null

All
O(1)

THE UNIVERSITY OF ARIZONA



put(pink,)

h(pink) = 16 mod 11 = 5

Collision!!!

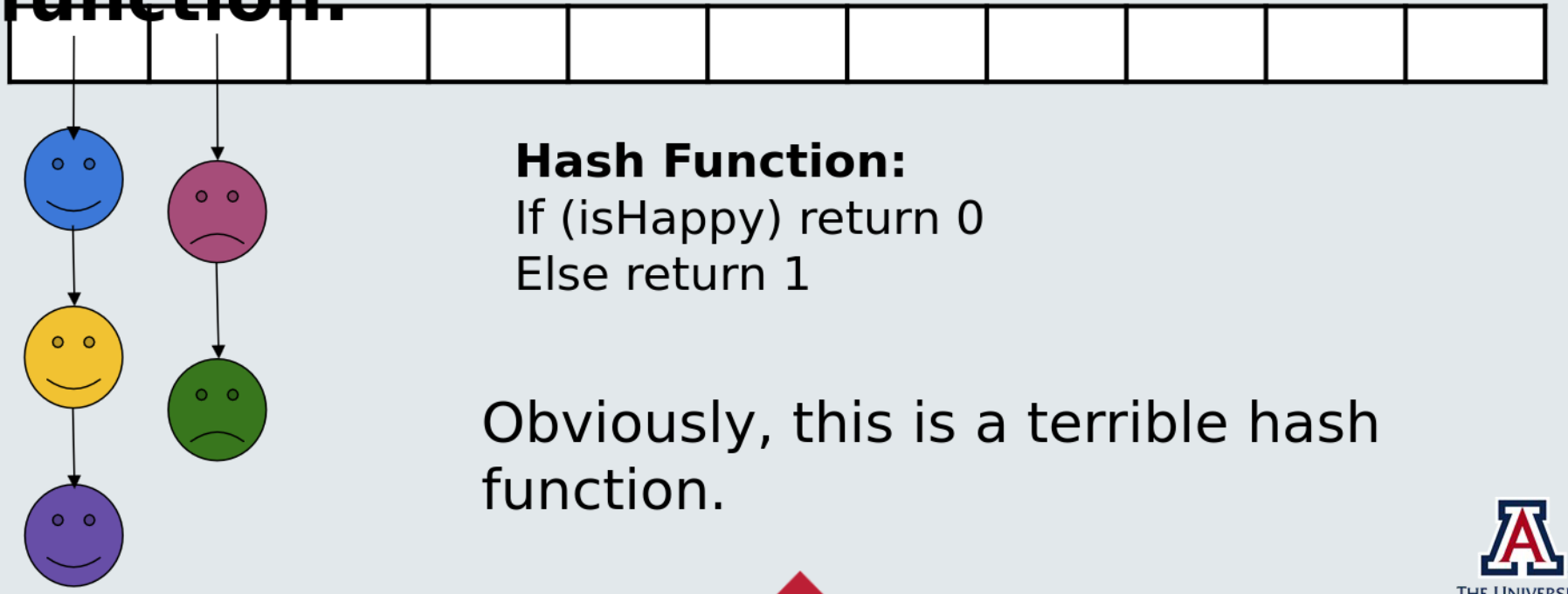
Collision Avoidance

Software Engineering
UA COLLEGE OF ENGINEERING

- If possible, we would like to avoid collisions.
- To do that, we need
  - a good hash function
  - a well-managed table (particularly the ratio of items to the size of the table, which is called the *load factor*)
- Even so, we will not likely succeed in complete collision avoidance, so we will still have to deal with collision *management*.
- But in the meantime, let's consider what makes a good hash function.

THE UNIVERSITY OF ARIZONA

A good hash function should minimize collisions. This is not a good hash function.



Hash Function:
If (isHappy) return 0
Else return 1

Obviously, this is a terrible hash function.

So how can we design a hash function that minimizes collisions?

Key Idea: Use as much of the key as possible so as to make each key hash to something that is more likely to be unique.

Hash Functions

What is it?

Software Engineering
UA COLLEGE OF ENGINEERING

- A hash function is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size, which falls into the hash table.
- The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes.

THE UNIVERSITY OF ARIZONA

To achieve a good hashing mechanism…

Software Engineering
UA COLLEGE OF ENGINEERING

- WE NEED A GOOD HASH FUNCTION THAT FOLLOWS THE NEXT:

- **1. Easy to compute:** It should be easy to compute and must not become an algorithm in itself.
- **2. Uniform distribution:** It should provide a uniform distribution across the hash table and should not result in clustering.
- **3. Less collisions:** Collisions occur when pairs of elements are mapped to the same hash value. These should be avoided.

THE UNIVERSITY OF ARIZONA

Modular Hashing

- A modular hash function is a hash function that uses the modulo operator to map a key to a slot in a hash table.
- The hash function is $h(k) = k \bmod m$, where k is an integer hash code and m is usually the number of buckets.
- For example: if the hash table has 12 slots and the key is 100, then $h(k) = 4$.

There are several types of hash functions, including:

Software Engineering
UA COLLEGE OF ENGINEERING

- **Universal hash function:** This function returns a unique hash number (random number generation).
- **Division Method.**
- **Mid Square Method.**
- **Folding Method.**
- **Multiplication Method.**

THE UNIVERSITY OF ARIZONA

Collision Management

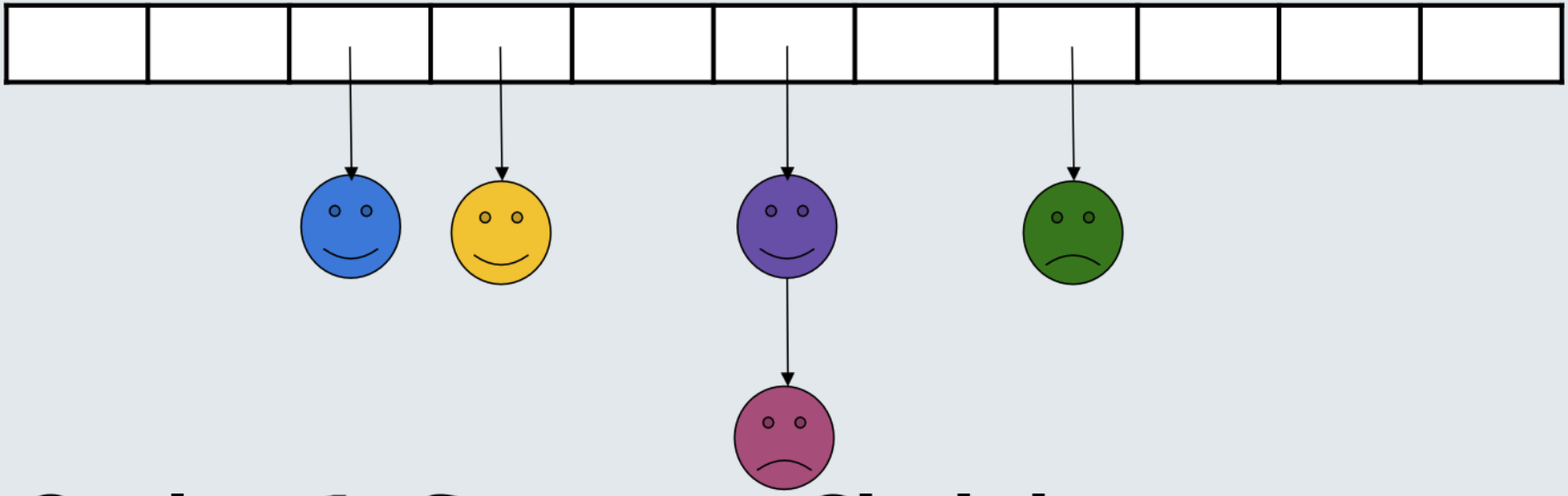
THE UNIVERSITY OF ARIZONA

Collision Management

- Separate Chaining
- Linear Probing
- Quadratic Probing

Image from
https://clipartart.com/wallpaper/getimg.html

Separate Chaining



Option 1: Separate Chaining

put(pink,)

h(pink) = 16 mod 11 = 5

Collisions and How to handle them

Software Engineering
UA COLLEGE OF ENGINEERING

SEPARATE CHAINING TECHNIQUE

- The idea is to make each cell of the hash table point to a linked list of records that have the same hash function values. It is simple but requires additional memory outside the table.
- In this technique, the worst case occurs when all the values are in the same index or linked list, making the search complexity linear (n= length of the linked list).

THE UNIVERSITY OF ARIZONA

Separate Chaining

- The hashtable is an array of lists.
- Collisions are managed by adding to lists when items hash to an existing list.
- This means that for *put* and *get*, the runtime is proportional to the length of the list in question.

Separate Chaining

- The hashtable is an array of linked lists.
- Collisions are managed by adding to lists when items hash to an existing list.
- This means that for *put* and *get*, the runtime is proportional to the length of the list in question, which could be…***N* in the worst case…BUT…**
- **…with a good hash function and a good table size, we expect the length of each list to be about *N/M*. This is called the load factor, indicated by α.**

How do you choose a good M (size of table)?

Software Engineering
UA COLLEGE OF ENGINEERING

- What is the relationship between **M** and the average length of the lists?

  Bigger M means shorter lists!

- It's a good idea to manage your table to maintain a reasonable load factor so that each of the operations can be done in a reasonable amount of time. The goal is to have to close to O(1).

THE UNIVERSITY OF ARIZONA

Analysis

Software Engineering
UA COLLEGE OF ENGINEERING

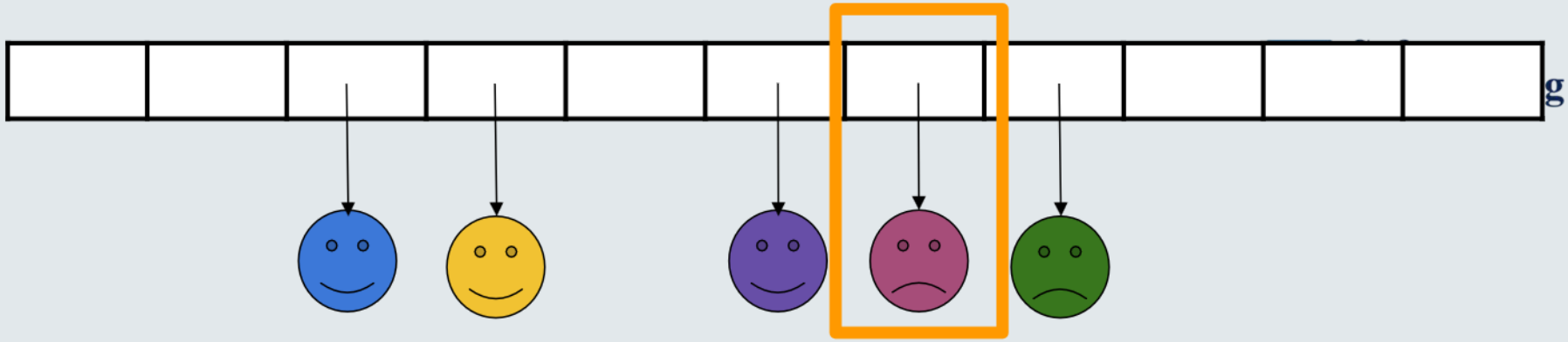
Analyze these operations where **M** is the size of the table and **N** is the number of key-value pairs:

- **put(k, v)**
  - best case: **O(1)**
  - worst-case: **O(N)**
  - expected case, assuming a good hash function that uniformly distributes keys, **O(N/M)**
- **delete(k):** same as **put**
- **get(k):** same as **put**

What if **M ≈ N**? **The expected runtime becomes O(1).**

THE UNIVERSITY OF ARIZONA

Linear Probing



Option 2: Linear Probing

put(pink,)

h(pink) = 16 mod 11 = 5

Linear Probing

Software Engineering
UA COLLEGE OF ENGINEERING

1. **Linear Probing:** We linearly probe/look for next slot. If slot [hash(x)%size] is full, we try [hash(x)%size+1]. If that is full too, we try [hash(x)%size+2]...until an available space is found. Linear Probing has the best cache performance, but downside includes primary and secondary clustering.

THE UNIVERSITY OF ARIZONA

Linear Probing

- The hashtable is an array.
- Collisions are managed by searching for an open spot as follows:
  - try *h(k)*
  - if that's taken, try *(h(k) + 1) % M*
  - if that's taken, try *(h(k) + 2) % M*
  - and so on…
- Typically, you want to maintain the hashtable to be between ⅛ and ½ full to get good average runtimes for the basic operations. **This is called the load factor, indicated by α. Note, although in this case α < 1, it can still be calculated by N/M.**

Load Factor α

Software Engineering
UA COLLEGE OF ENGINEERING

- The number of elements in a hash table divided by the number of slots.
- The higher the load factor, the slower the retrieval.
- Load factor N/M where n = number of entries in the structure; N = number of slots in the array.
  - If I have 10 elements and 10 slots in the array, then the load factor is 1.

THE UNIVERSITY OF ARIZONA

Properties of Linear Probing

Software Engineering
UA COLLEGE OF ENGINEERING

- **Null** used to mark division between clusters
- The longer a cluster, the longer the search time will be
- The longer a cluster, the more likely its length will increase

THE UNIVERSITY OF ARIZONA

Load Factor: α

Separate Chaining

- How long are the lists?
- α = N/M (the average number of keys per list), which is often larger than 1

Linear Probing

- How full is the array?
- α = N/M = fraction of table entries that are occupied (cannot be greater than 1)

Quadrating Probing

Quadrating Probing

Software Engineering
UA COLLEGE OF ENGINEERING

2. Quadratic Probing: We look for i^2th iteration. If slot [hash(x)%size] is full, we try [(hash(x)+1*1)%size]. If that is also full, we try [(hash(x)+2*2)%size]...until an available space is found. Secondary clustering might arise here and there is no guarantee of finding a slot in this approach.

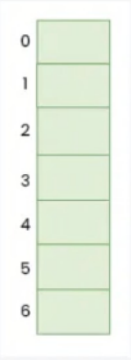
THE UNIVERSITY OF ARIZONA

Quadratic Probing

- Using linear probing, collisions are resolved by searching for an open spot in the following way: h(k), h(k)+1, h(k) + 2…
- Quadratic probing, the following spots are searched: h(k), h(k) + 1^2, h(k) + 2^2, h(k) + 3^2…
- Introduces the possibility for infinite (or at least very long) loops in searching for an open spot, so you may need a limit on how many probes it will try before giving up and re-sizing the array.

Quadratic Probing (Open Addressing)

Slot

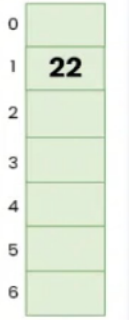


Step 01

Empty hash table with range of hash values from **0 to 6** according to the hash function provided.

Quadratic Probing (Open Addressing)

Slot

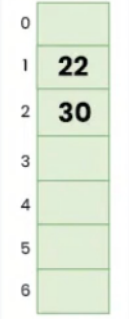


Step 02

The first key to be inserted is **22** which is mapped to **slot 1 (22%7=1)**

Quadratic Probing (Open Addressing)

Slot

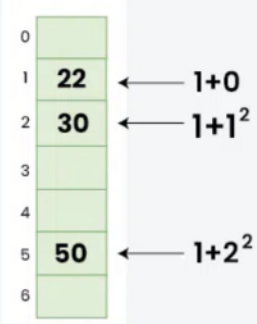


Step 03

The next key is **30** which is mapped to **slot 2 (30%7=2)**

Quadratic Probing (Open Addressing)

Slot



Step 04

The next key is **50** which is mapped to **slot 1 (50%7=1)** but slot 1 is already occupied. So, we will search **slot 1+1^2**, i.e. **1+1 = 2**. Again slot 2 is occupied, so we will search cell **1+2^2**, i.e.**1+4 = 5**,

Hashtables Applications

Applications

Software Engineering
UA COLLEGE OF ENGINEERING

- **Associative arrays:** Hash tables are commonly used to implement many types of in-memory tables. They are used to implement associative arrays (arrays whose indices are arbitrary strings or other complicated objects).
- **Database indexing:** Hash tables may also be used as disk – based data structures and database indices.

THE UNIVERSITY OF ARIZONA

Applications

Software Engineering
UA COLLEGE OF ENGINEERING

- **Caches:** Hash tables can be used to implement caches i.e. auxiliary data tables that are used to speed up the access to data, which is primarily stored in slower media.
- **Object representation:** Several dynamic languages, such as Perl, Python, JavaScript, and Ruby, use hash tables to implement objects.
- **Hash functions** are used in various algorithms to make their computing faster.

THE UNIVERSITY OF ARIZONA