Search Algorithms - Hash and BST

Course: Algorithms

Faculty: Dr. Rajendra Prasath

Autumn 2018

Search Algorithms – Hash Based and BST

This lecture covers two more search algorithms: Hash Based and Binary Search Tree based algorithms for searching an element in a given collection. We provide illustrations and the complexity analysis of these two search algorithms

Recap: Search Algorithms

- Given: A collection C of elements
- Task: Search for the element x

- **Existence:**
  - Does C contain a target element x?
  - Response: Yes (exists) / No (does not exist)
- **Retrieval:**
  - Give your roll number and get all details?
  - Search anything in Google – Scalable searches
- **Associate Look up:**
  - Information Associated with the key x

Two More Search Algorithms

- In this lecture, we will focus on two more search algorithms:
- Hash Based Search Algorithm
  - Handling large collection that may not be necessarily ordered
- Binary Search Tree Algorithm
  - Tree based search algorithm
- Complexity analysis

Hash Based Search Algorithm

- Given:
  - A collection C of n elements
  - Search the collection for an element x

- Basic Idea:
  - Organize the elements into a hash table A that has k bins

Hash-Based Search

Two Steps:

- Load hash Table consisting of n elements
  - Design of the hash function
  - Handle collisions (two keys map to the same bin)
    - How to avoid collisions
- Search for an element in the collection
  - Collection is organized in bins

Hash-Based Search

- Each element e in C can be mapped to a key k = key(e) such that

$$\text{if } e_i = e_j \text{ then } key(e_i) = key(e_j)$$

- A hash function h = hash(e) uses the key value key(e) to determine the bin in which e will be inserted.

- Once the hash table is constructed then search for an item x is transformed into a search for x within C[h] where h = hash(x).

Hash Based Search

```
Procedure loadTable(size, C)

begin
    A = new array of given size
    For i = 0 to n-1 do
        h = hash(C[i])
        if (A[h] is empty) then
            A[h] = new linked list
        add C[i] to A[h]
    return A
end
```

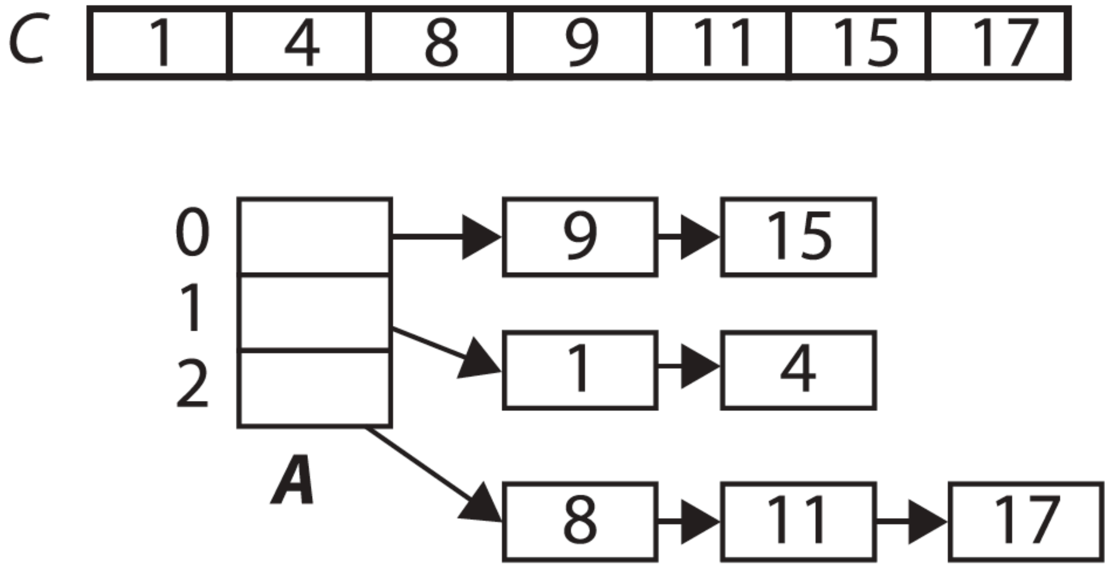
Hash Based Search

```
Procedure Search(A, x)

begin
    h = hash(x)
    list = A[h]
    if (list is empty) then
        return false
    if (list contains x) then
        return true
    return false
end
```

Hash-Based Search - Example

Loading the hash table:



A handles collisions with lists

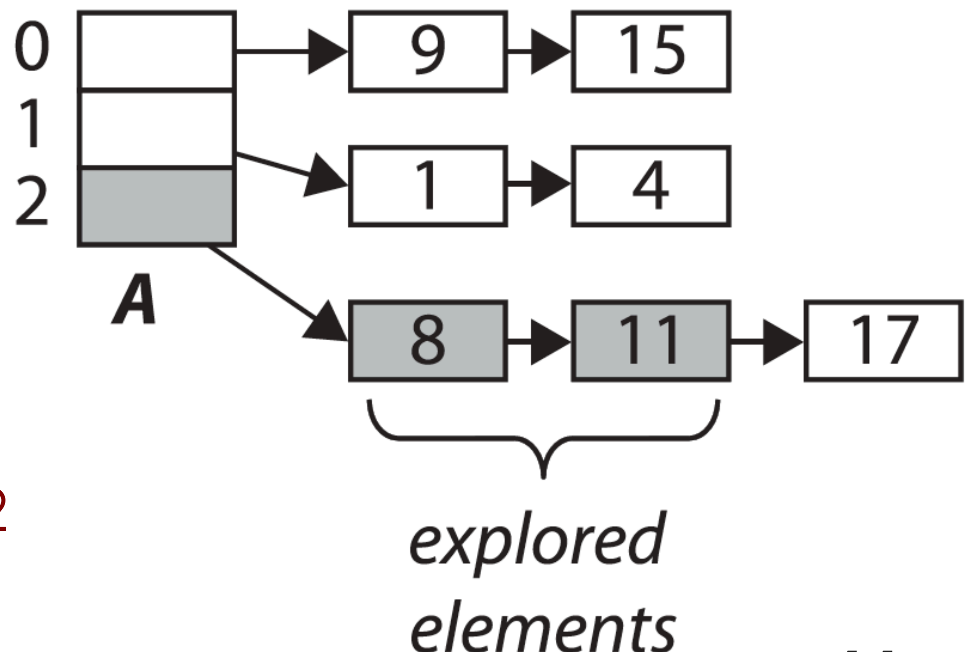
Has(e) = remainder of e ÷ 3

Hash-Based Search - Example

Searching the hash table:

Task:

Search (A, 11)

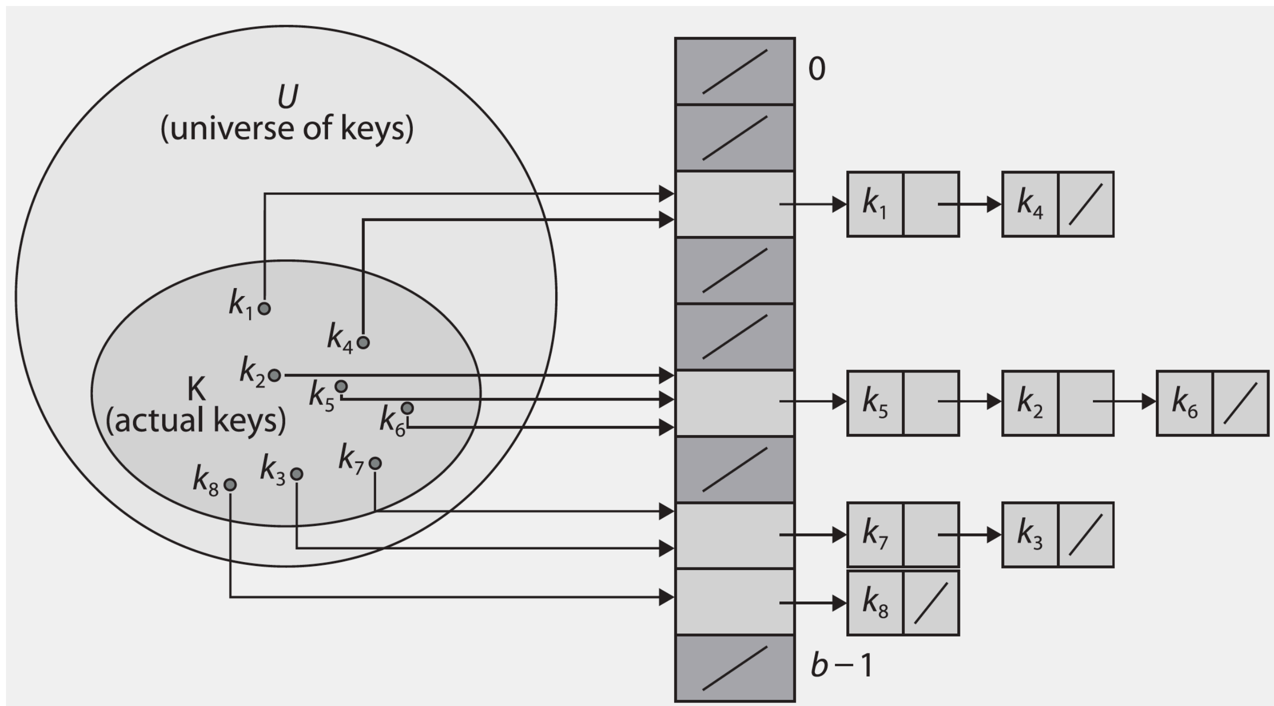


Note:

Remainder of 11 ÷ 3 = 2

Hash-based Search - Facts

- Handling the set of possible keys
- How to define an Hash function that is robust
- How to avoid collisions with lists



Hash-based Search - Analysis

- Input size: n elements in the collection C and x has to be search for
- Best Case:
  - O(1)
- Average Case:
  - O(1)
- Worst Case:
  - O(n)
- Data Structures:
  - Arrays, Hash

Binary Search Tree

- Elements are arranged in a tree like structure
  - Based on Divide and conquer approach

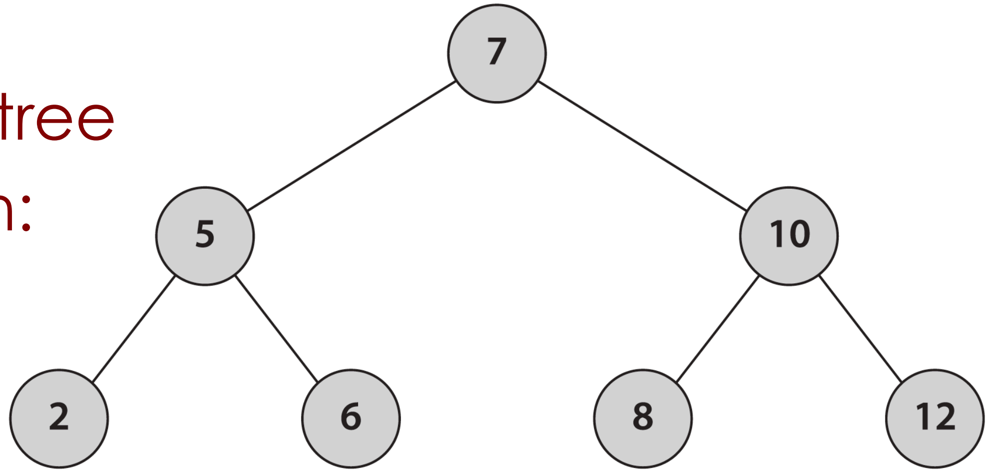
  - Partial Order is imposed in arranging the elements

  - How to arrange elements?
    - Root

    - Based on the partial order, organize elements in
      - Left Subtree
      - Right Subtree

Binary Search Tree

- Look at the following example:
- The node in the tree has two children:
  - Left
  - Right



- Binary Search tree property:
  - Let k be the key at root.
  - All keys in the left subtree are ≤ k
  - All keys in the right subtree are > k (vice versa)

BST - Properties

- A binary tree is either empty or consists of a node called the root together with two binary trees called the left subtree and the right subtree

- It is a tree with two children
  - Left Child
  - Right Child

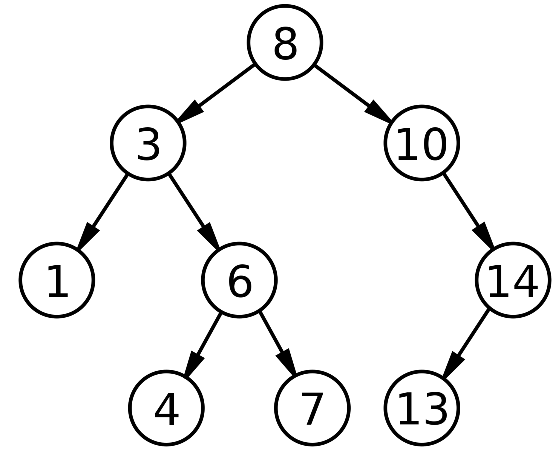
- Let h = height of a binary tree
  - max # of leaves = 2h
  - max # of nodes = 2h + 1 – 1

- Full binary tree:
  - A binary tree with height h and 2h + 1 - 1 nodes (or 2h leaves)

Binary Tree

- Create Binary Tree
  - Create an empty queue Q
  - Add Q the root node
  - while (Q != NULL)
    - New Node = dequeue Q
    - Update data of the New Node
    - Add New Node's children (first left then right) to Q

Binary Search Tree



- Create BST
  - Create an empty queue Q
  - Add Q the root node
  - while (Q != NULL)
    - New Node = dequeue Q
    - Update data of the New Node **by checking the value of the key field**
    - Add New Node to either the left or right subtree of Q

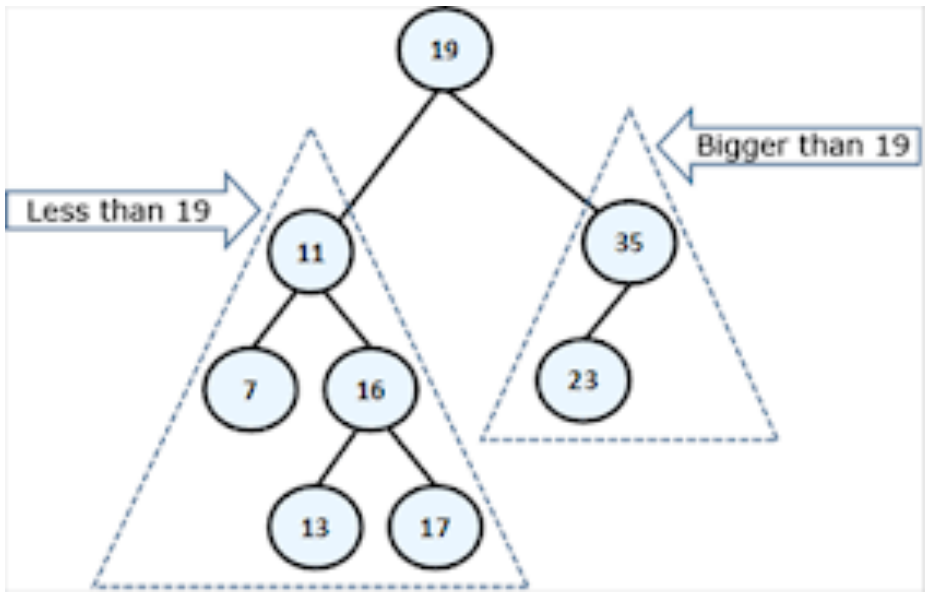
BST - Algorithm

The following recursive procedure searches for the **key** in a binary search tree rooted at **node:**

```
Procedure binarySearchTree(key, node)
begin
    if node is NULL then return false
    if node.key = key then
        return true
  if key < node.key
     return search_recursively(key, node.left)
  return search_recursively(key, node.right)
end
```

Binary Search – An Example

Look at the following Example:



BST - Operations

Operations that can be applied on BST:

- Searching for an element
- Finding Minimum Value
- Finding the Maximum Value
- Finding 2nd Max number

. . .

and so on

Binary Search Tree - Analysis

- Input size: n elements in the collection C and x has to be search for
- Average Case:
  - O(log n)
- Worst Case:
  - O(log n)
- Data Structures:
  - Arrays, Trees and its variations

Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)
- **Promising Students** (having CGPA above 6.5 and less than 8.5)

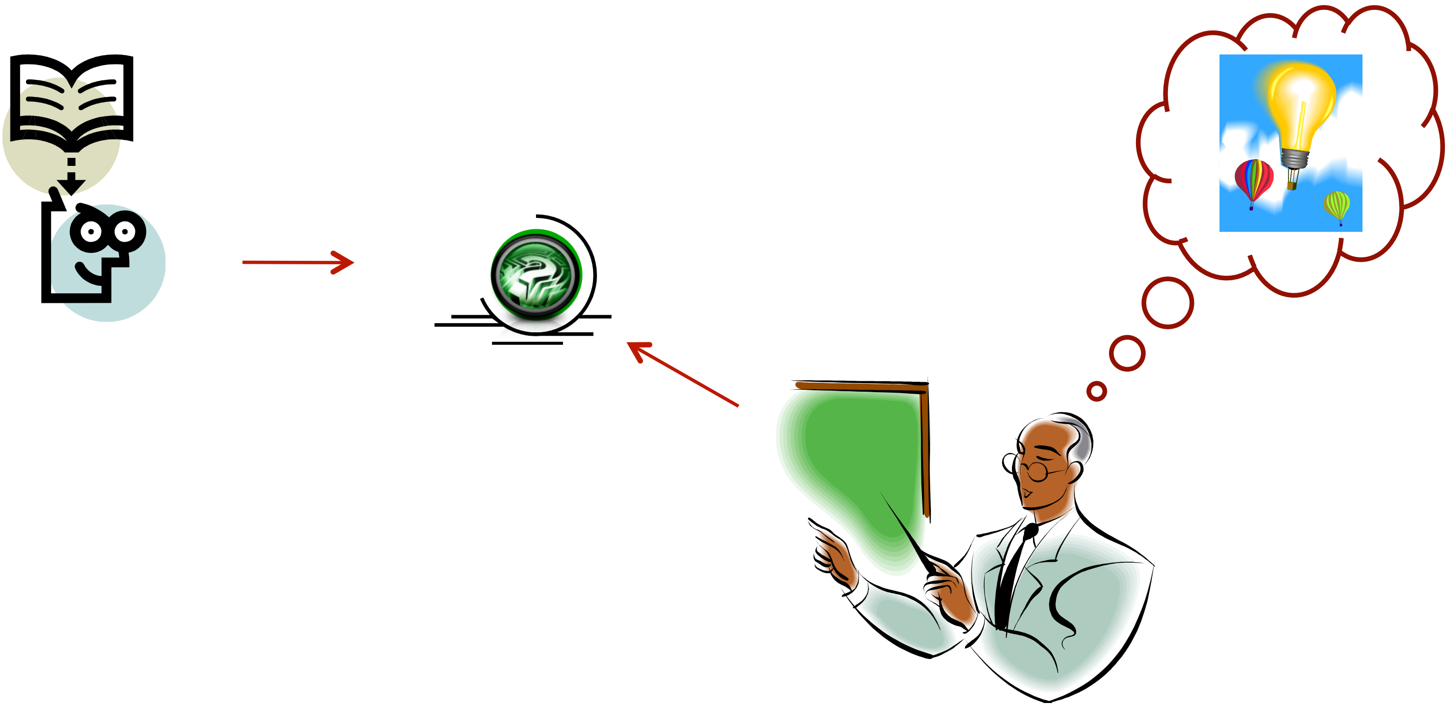
- **Needy Students** (having CGPA less than 6.5)
  - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

Assistance

- You may post your questions to me at any time
- You may meet me in person on available time or with an appointment
- TA s would assist you to clear your doubts.
- You may leave me an email any time (email is the best way to reach me faster)

Thanks …



… Questions ???