Review of Data Structures

Virendra Singh

Professor, Indian Institute of Technology Bombay

And

Adjunct Professor, Indian Institute of Technology Jammu

http://www.ee.iitb.ac.in/~viren/

E-mail: viren@ee.iitb.ac.in, virendra.singh@iitjammu.ac.in

CSPL201: Data Organization & Retrieval

Lecture 5 (27 September 2020)

भारतीय प्रौद्योगिकी संस्थान जम्मू
INDIAN INSTITUTE OF TECHNOLOGY JAMMU
विद्याधनं सर्वधन प्रधानम्

Acknowledgement

- Prof. Sartaj Sahni, Uni. of Florida
- Prof. Hideo Fujiwara, NAIST
- Late Prof. A. Bhattacharya, SERC, IISc

Data Structures

- Data structure is a way to store and organize data in order to facilitate access and modification
- No single data structure works well for all purposes

- data object
- set or collection of instances
  - integer = {0, +1, -1, +2, -2, +3, -3, …}
  - daysOfWeek = {S,M,T,W,Th,F,Sa}
- instances may or may not be related
  - myDataObject = {apple, chair, 2, 5.2, red, green, Jack}

Dictionaries

- Collection of pairs.
  - (key, element)
  - Pairs have different keys.
- Operations.
  - get (theKey)
  - put (theKey, theElement)
  - remove (theKey)

Application

- Collection of student records in this class.
  - (key, element) = (student name, linear list of assignment and exam scores)
  - All keys are distinct.
- Get the element whose key is Ajay.
- Update the element whose key is Sarika.
  - put() implemented as update when there is already a pair with the given key.
  - remove() followed by put().

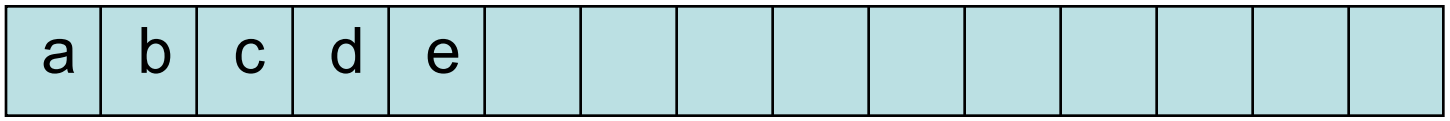
Dictionary With Duplicates

- Keys are not required to be distinct.
- Word dictionary.
  - Pairs are of the form (word, meaning).
  - May have two or more entries for the same word.
    - (bolt, a threaded pin)
    - (bolt, a crash of thunder)
    - (bolt, to shoot forth suddenly)
    - (bolt, a gulp)
    - (bolt, a standard roll of cloth)
    - etc.

Represent As A Linear List

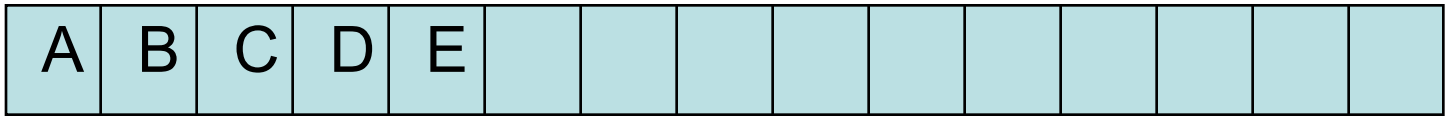
- $L = (e_0, e_1, e_2, e_3, \ldots, e_{n-1})$
- Each e_i is a pair (key, element).
- 5-pair dictionary D = (a, b, c, d, e).
  - a = (aKey, aElement), b = (bKey, bElement), etc.
- Array or linked representation.

Array Representation



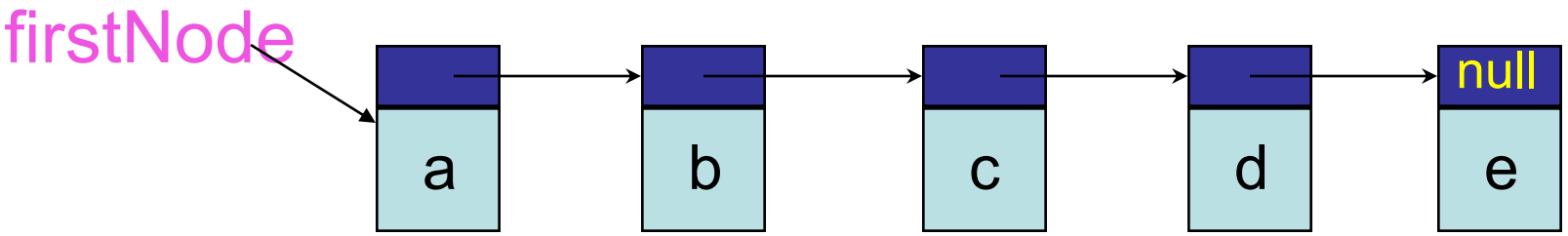
- get(theKey)
  - O(size) time
- put(theKey, theElement)
  - O(size) time to verify duplicate, O(1) to add at right end.
- remove(theKey)
  - O(size) time.

Sorted Array



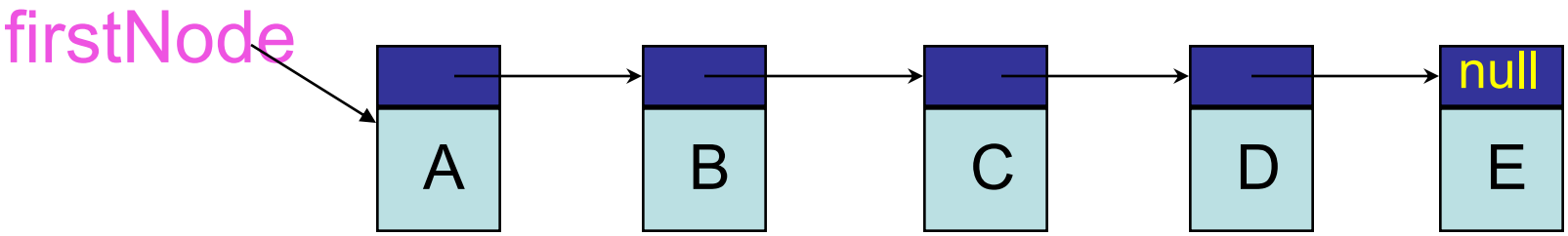
- elements are in ascending order of key.
- get(theKey)
  - O(log size) time
- put(theKey, theElement)
  - O(log size) time to verify duplicate, O(size) to add.
- remove(theKey)
  - O(size) time.

Unsorted Chain



- get(theKey)
  - O(size) time
- put(theKey, theElement)
  - O(size) time to verify duplicate, O(1) to add at left end.
- remove(theKey)
  - O(size) time.

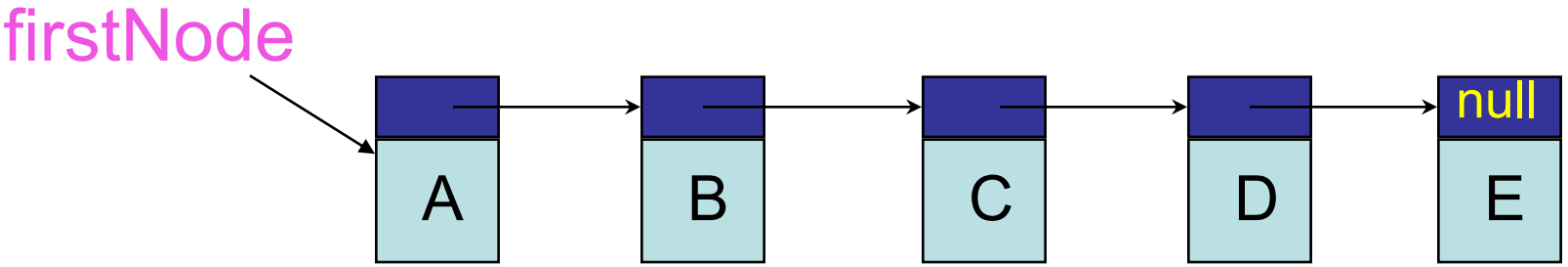
Sorted Chain



- Elements are in ascending order of Key.

- get(theKey)
  - O(size) time
- put(theKey, theElement)
  - O(size) time to verify duplicate, O(1) to put at proper place.

Sorted Chain



- Elements are in ascending order of Key.

- remove(theKey)
  - O(size) time.

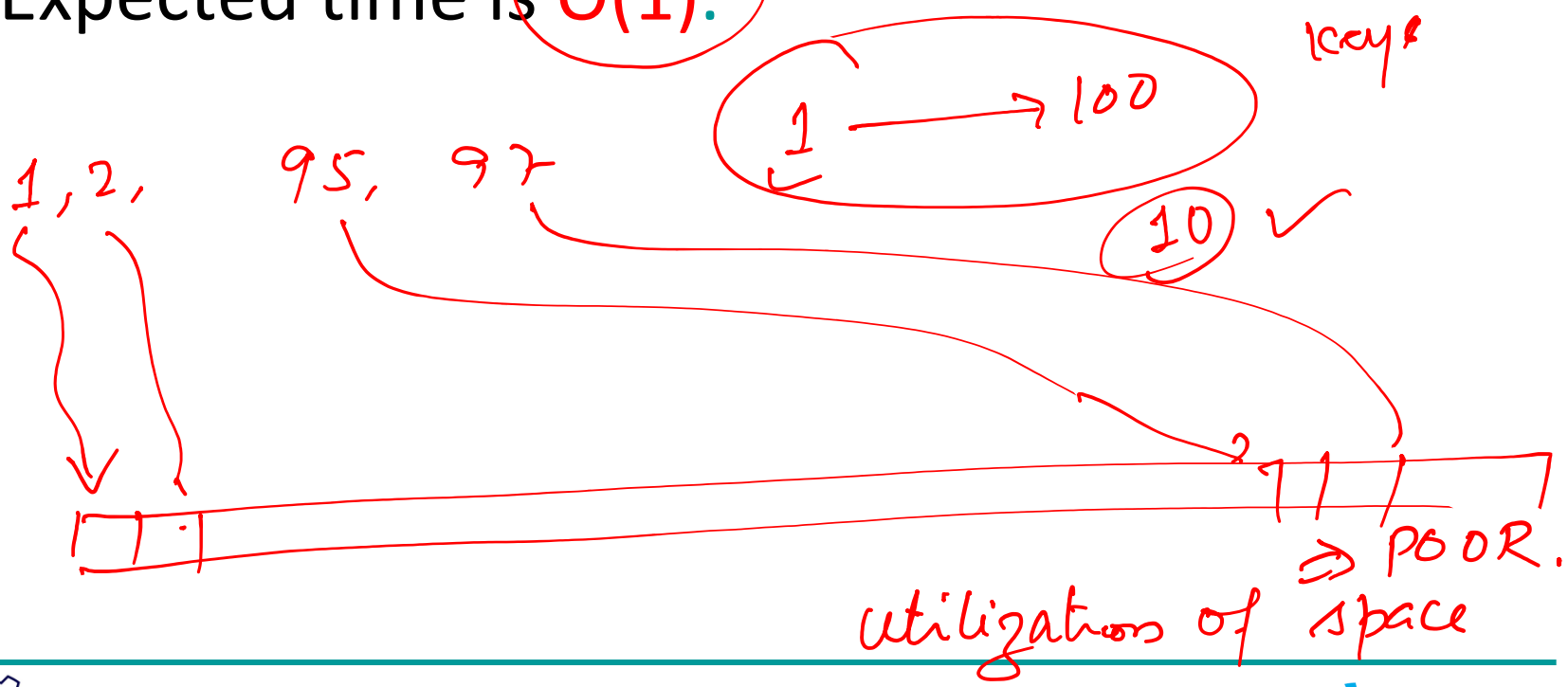
Skip Lists

- Worst-case time for get, put, and remove is O(size).
- Expected time is O(log size).

HASHING

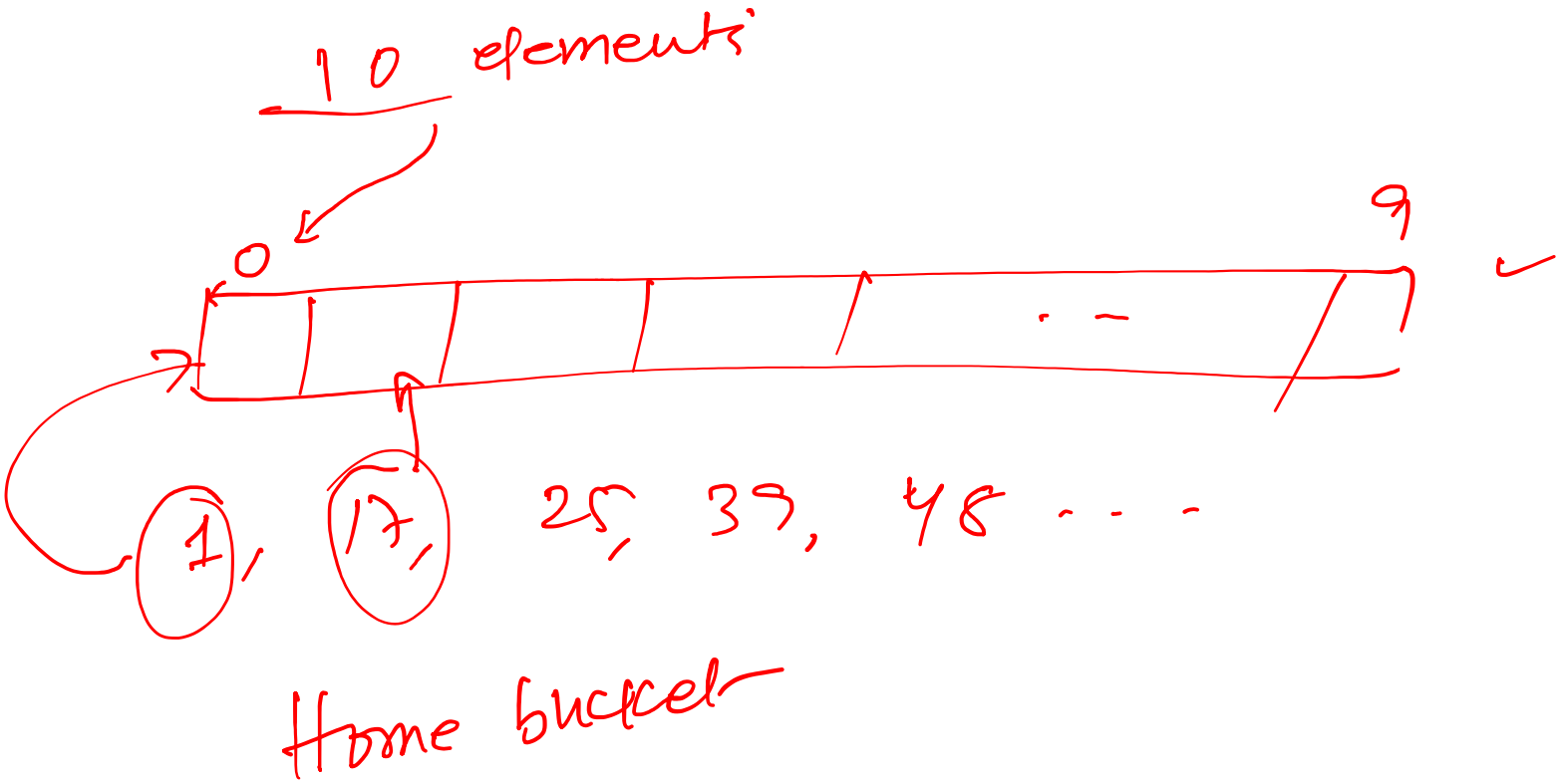
Hash Tables

- Worst-case time for get, put, and remove is O(size).
- Expected time is O(1).



Ideal Hashing

- Uses a 1D array (or table) table[0:b-1].
  - Each position of this array is a bucket.
  - A bucket can normally hold only one dictionary pair.
- Uses a hash function f that converts each key k into an index in the range [0, b-1].
  - f(k) is the home bucket for key k.
- Every dictionary pair (key, element) is stored in its home bucket table[f[key]].


10 elements
0
9
1, 17, 25, 39, 48 . . .
Home bucket

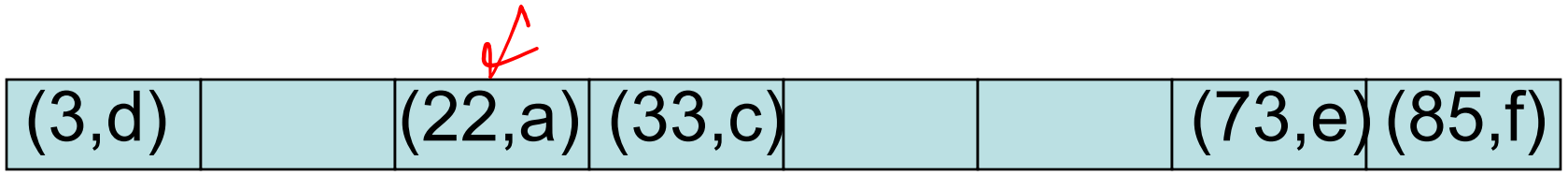
Ideal Hashing Example

- Pairs are: (22,a), (33,c), (3,d), (73,e), (85,f).
- Hash table is table[0:7], b = 8.
- Hash function is key/11.
- Pairs are stored in table as below:

0	1	2	3	4	5	6	7
(3,d)		(22,a)	(33,c)			(73,e)	(85,f)

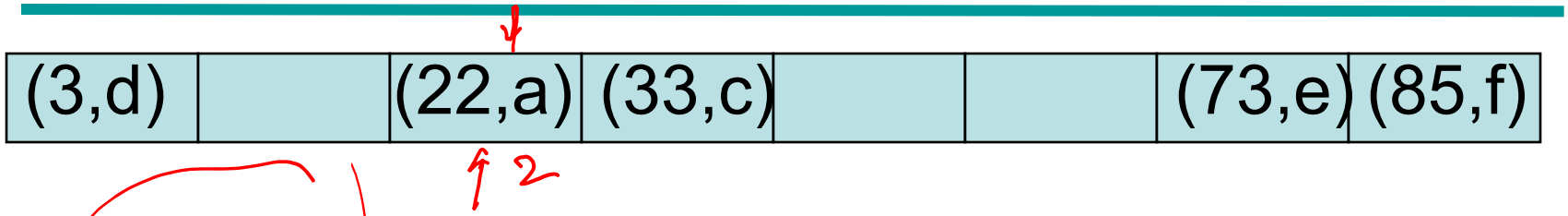
- get, put, and remove take O(1) time.

What Can Go Wrong?



- Where does (26,g) go?
- Keys that have the same home bucket are .
  - 22 and 26 are synonyms with respect to the hash function that is in use.
- The home bucket for (26,g) is already occupied.

What Can Go Wrong?



- A collision occurs when the home bucket for a new pair is occupied by a pair with a different key.
- An overflow occurs when there is no space in the home bucket for the new pair.
- When a bucket can hold only one pair, collisions and overflows occur together.
- Need a method to handle overflows.

O(n) ✓

Hash Table Issues

- Choice of hash function.
- Overflow handling method.
- Size (number of buckets) of hash table.

Hash Functions

- Two parts:
  - Convert key into an integer in case the key is not an integer.
- Map an integer into a home bucket.
  - f(k) is an integer in the range [0, b-1], where b is the number of buckets in the table.

Map Into A Home Bucket

(3,d)		(22,a)	(33,c)			(73,e)	(85,f)

- Most common method is by division.

  homeBucket = theKey % divisor

- divisor equals number of buckets b.
- 0 <= homeBucket < divisor = b

Uniform Hash Function

(3,d)		(22,a)	(33,c)			(73,e)	(85,f)

- Let keySpace be the set of all possible keys.
- A uniform hash function maps the keys in keySpace into buckets such that approximately the same number of keys get mapped into each bucket.

Uniform Hash Function

(3,d)		(22,a)	(33,c)			(73,e)	(85,f)

- Equivalently, the probability that a randomly selected key has bucket i as its home bucket is 1/b, 0 <= i < b.
- A uniform hash function minimizes the likelihood of an overflow when keys are selected at random.

Hashing By Division

- keySpace = all ints.
- For every b, the number of ints that get mapped (hashed) into bucket i is approximately $2^{32}/b$.
- Therefore, the division method results in a uniform hash function when keySpace = all ints.
- In practice, keys tend to be correlated.
- So, the choice of the divisor b affects the distribution of home buckets.

Selecting The Divisor

- Because of this correlation, applications tend to have a bias towards keys that map into odd integers (or into even ones).
- When the divisor is an even number, odd integers hash into odd home buckets and even integers into even home buckets.
  - 20%14 = 6, 30%14 = 2, 8%14 = 8
  - 15%14 = 1, 3%14 = 3, 23%14 = 9
- The bias in the keys results in a bias toward either the odd or even home buckets.

Selecting The Divisor

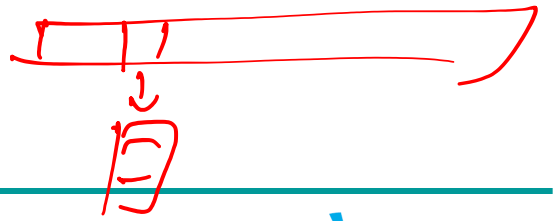
- When the divisor is an odd number, odd (even) integers may hash into any home.
  - 20%15 = 5, 30%15 = 0, 8%15 = 8
  - 15%15 = 0, 3%15 = 3, 23%15 = 8
- The bias in the keys does not result in a bias toward either the odd or even home buckets.
- Better chance of uniformly distributed home buckets.
- So do not use an even divisor.

Selecting The Divisor

- Similar biased distribution of home buckets is seen, in practice, when the divisor is a multiple of prime numbers such as 3, 5, 7, …
- The effect of each prime divisor p of b decreases as p gets larger.
- Ideally, choose b so that it is a prime number.
- Alternatively, choose b so that it has no prime factor smaller than 20.

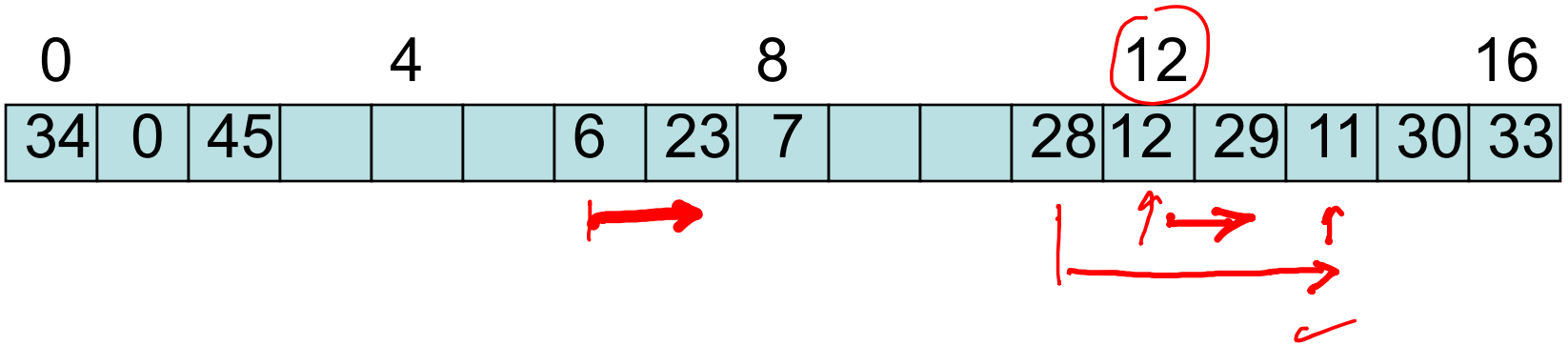
Overflow Handling

- An overflow occurs when the home bucket for a new pair (key, element) is full.
- We may handle overflows by:
  - Search the hash table in some systematic fashion for a bucket that is not full.
    - Linear probing (linear open addressing).
    - Quadratic probing.
    - Random probing.
  - Eliminate overflows by permitting each bucket to keep a list of all pairs for which it is the home bucket.
    - Array linear list.
    - Chain.



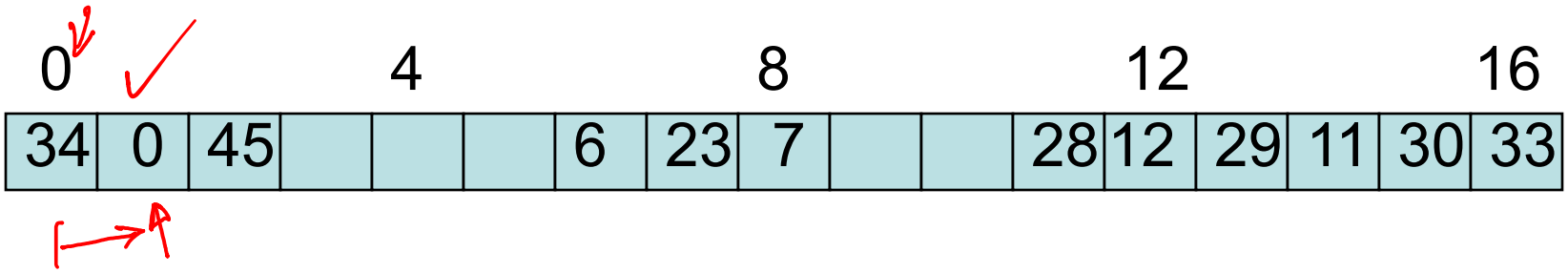
Linear Probing – Get And Put

- divisor = b (number of buckets) = 17.
- Home bucket = key % 17.

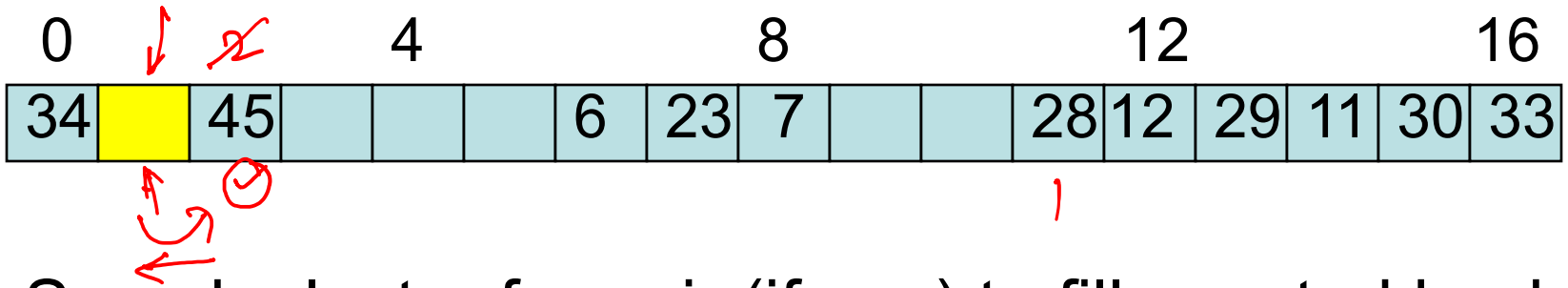


- Put in pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45

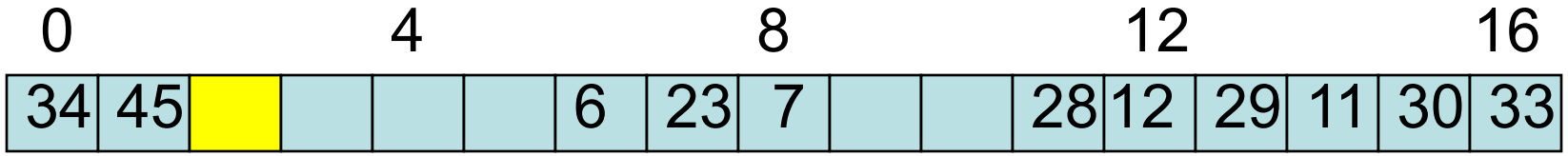
Linear Probing – Remove



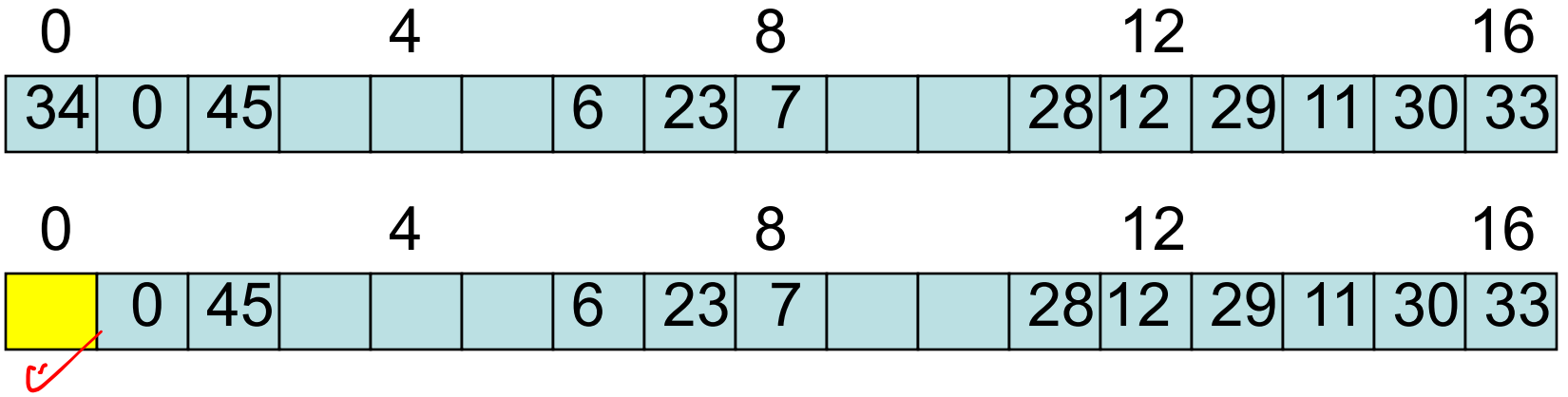
- remove(0)



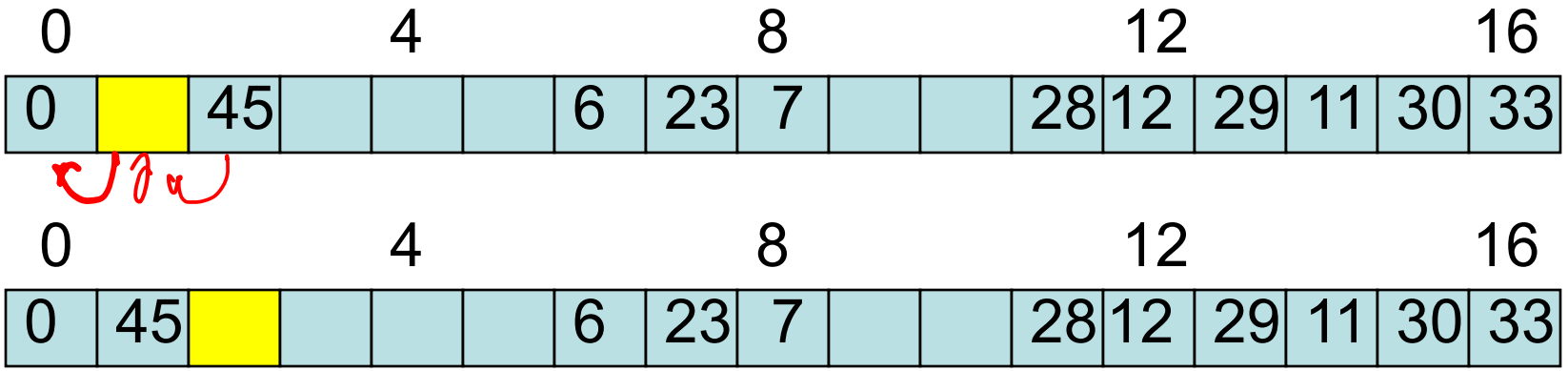
- Search cluster for pair (if any) to fill vacated bucket.



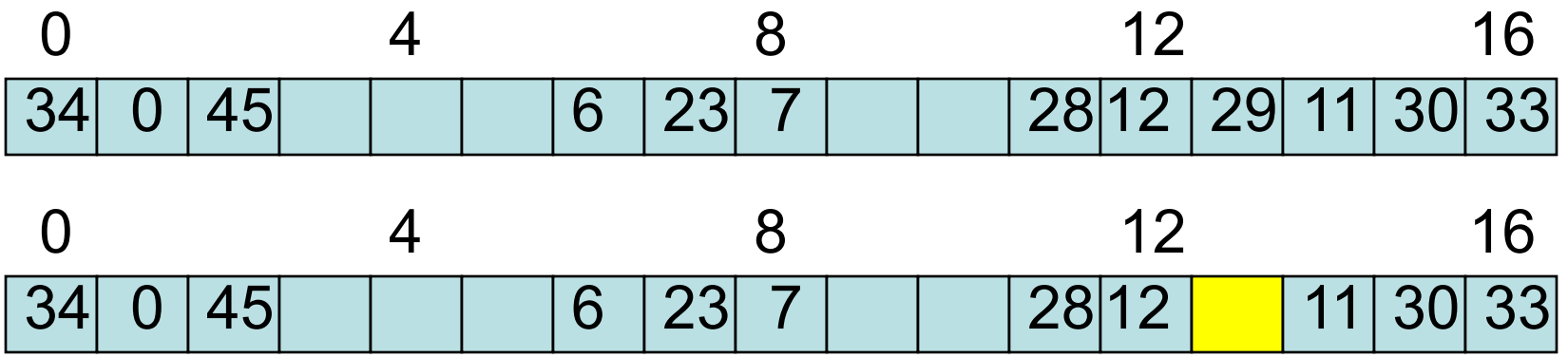
Linear Probing – remove(34)



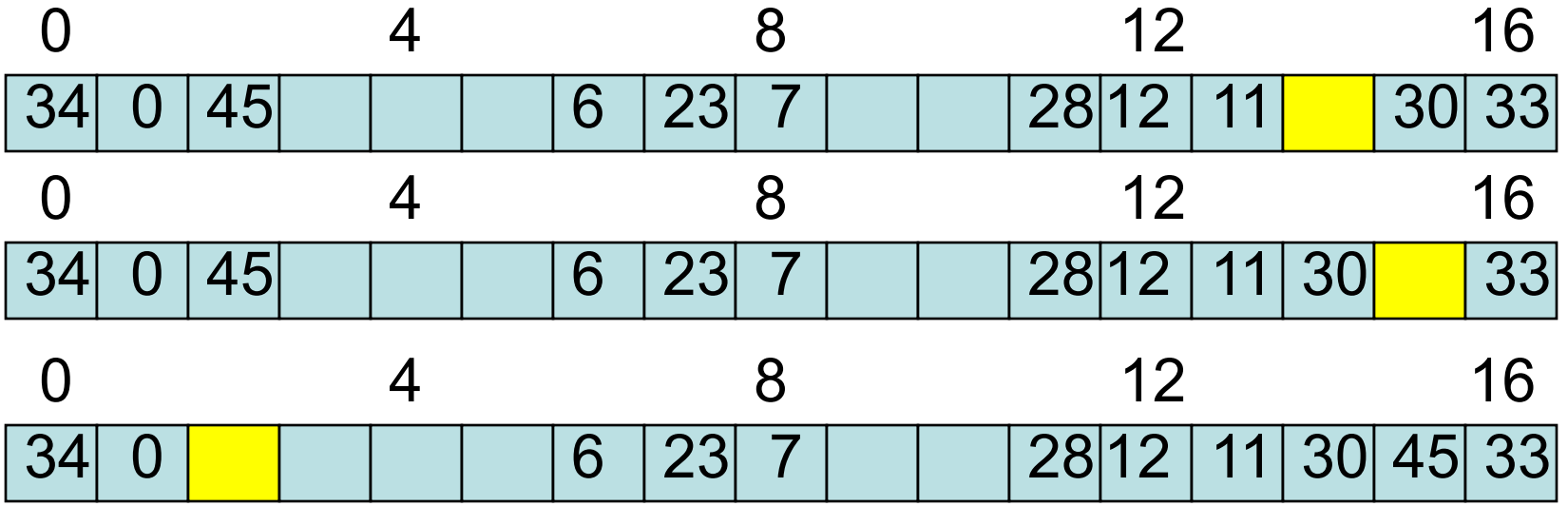
- Search cluster for pair (if any) to fill vacated bucket.



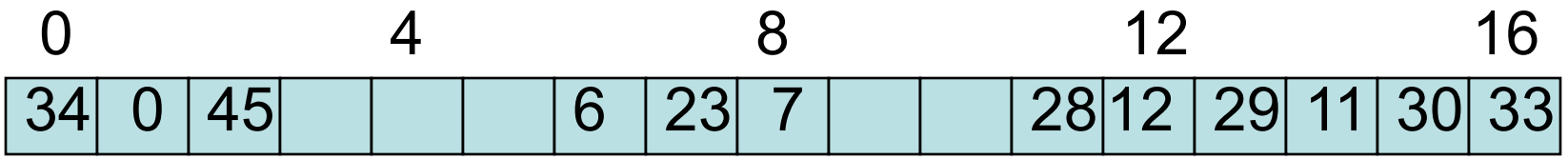
Linear Probing – remove(29)



- Search cluster for pair (if any) to fill vacated bucket.

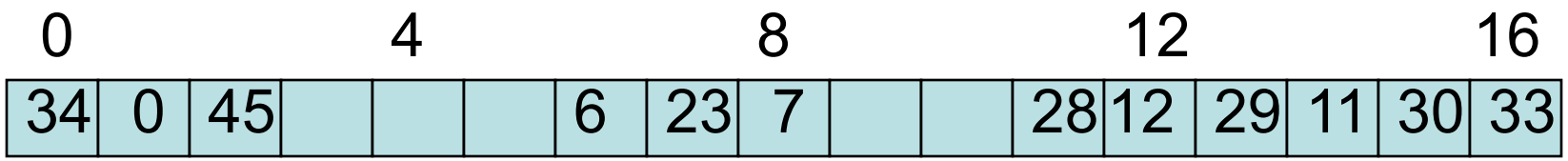


Performance Of Linear Probing



- Worst-case get/put/remove time is O(n), where n is the number of pairs in the table.
- This happens when all pairs are in the same cluster.

Expected Performance



- alpha = loading density = (number of pairs)/b.
  - alpha = 12/17.
- S_n = expected number of buckets examined in a successful search when n is large
- U_n = expected number of buckets examined in a unsuccessful search when n is large
- Time to put and remove governed by U_n.

Expected Performance

- S_n ~ ½(1 + 1/(1 – alpha))
- U_n ~ ½(1 + 1/(1 – alpha)^2)
- Note that 0 <= alpha <= 1

alpha	S_n	U_n
0.50	1.5	2.5
0.75	2.5	8.5
0.90	5.5	50.5

Alpha <= 0.75 is recommended.

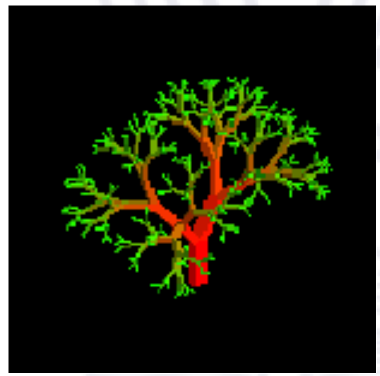
Hash Table Design

- Performance requirements are given, determine maximum permissible loading density.
- We want a successful search to make no more than 10 compares (expected).
  - $S_n \sim \frac{1}{2}(1 + 1/(1 - alpha))$
  - alpha <= 18/19
- We want an unsuccessful search to make no more than 13 compares (expected).
  - $U_n \sim \frac{1}{2}(1 + 1/(1 - alpha)^2)$
  - alpha <= 4/5
- So alpha <= min{18/19, 4/5} = 4/5.

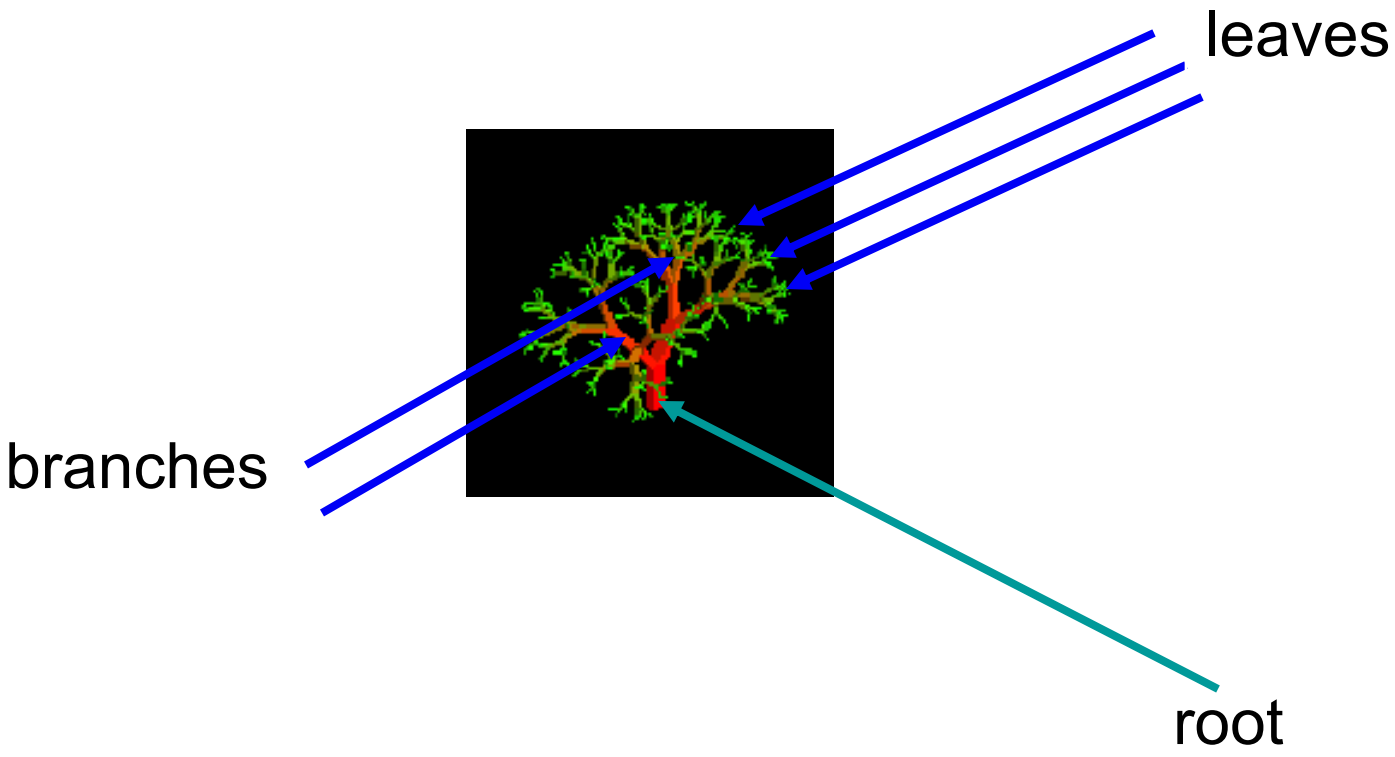
Hash Table Design

- Dynamic resizing of table.
  - Whenever loading density exceeds threshold (4/5 in our example), rehash into a table of approximately twice the current size.
- Fixed table size.
  - Know maximum number of pairs.
  - No more than 1000 pairs.
  - Loading density <= 4/5 b >= 5/4*1000 = 1250.
  - Pick b (equal to divisor) to be a prime number or an odd number with no prime divisors smaller than 20.

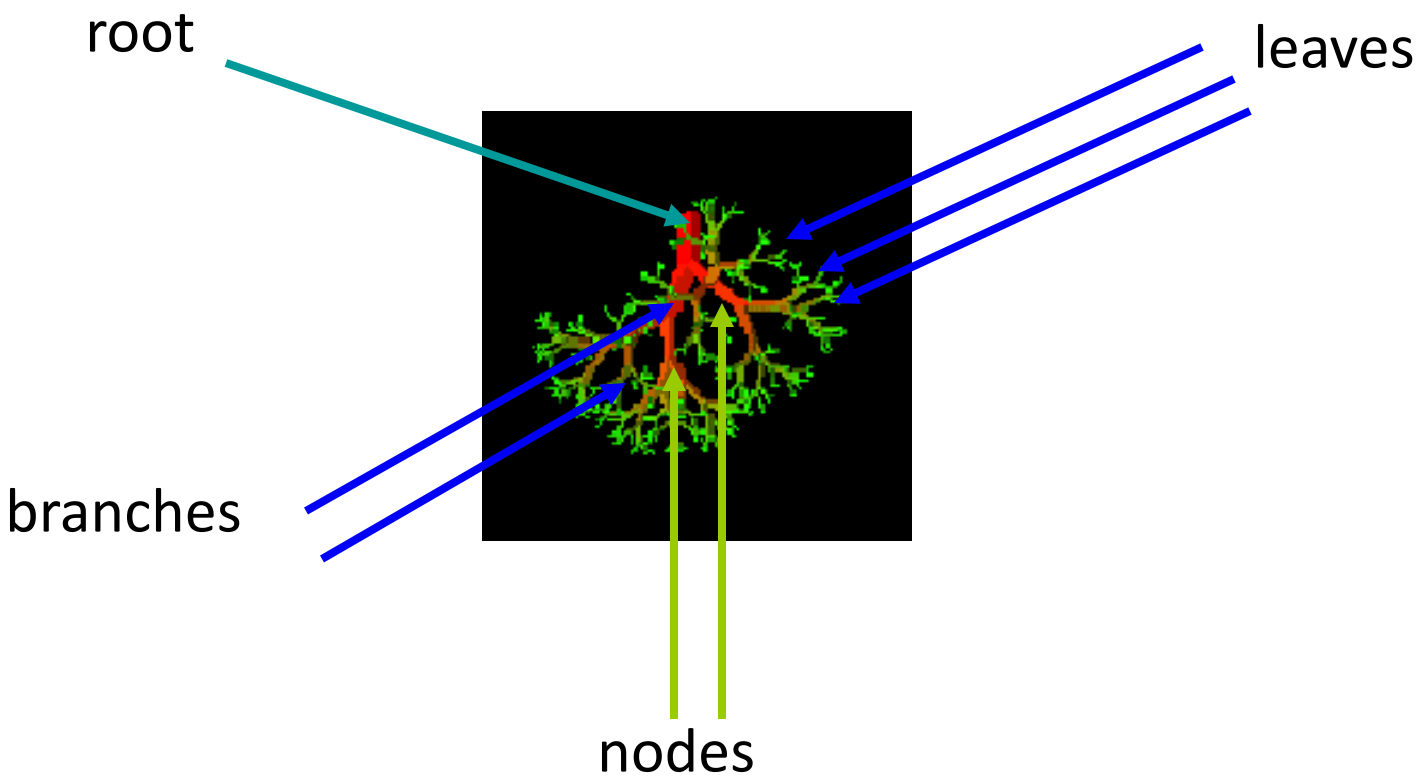
Trees



Nature Lover's View of A Tree



Computer Scientist's View



Linear Lists And Trees

- Linear lists are useful for serially ordered data.
  - $(e_0, e_1, e_2, \ldots, e_{n-1})$
  - Days of week.
  - Months in a year.
  - Students in this class.
- Trees are useful for hierarchically ordered data.
  - Employees of a corporation.
    - President, vice presidents, managers, and so on.
  - Java's classes.
    - Object is at the top of the hierarchy.
    - Subclasses of Object are next, and so on.

Hierarchical Data And Trees

- The element at the top of the hierarchy is the root.
- Elements next in the hierarchy are the children of the root.
- Elements next in the hierarchy are the and children of the root, and so on.
- Elements that have no children are .

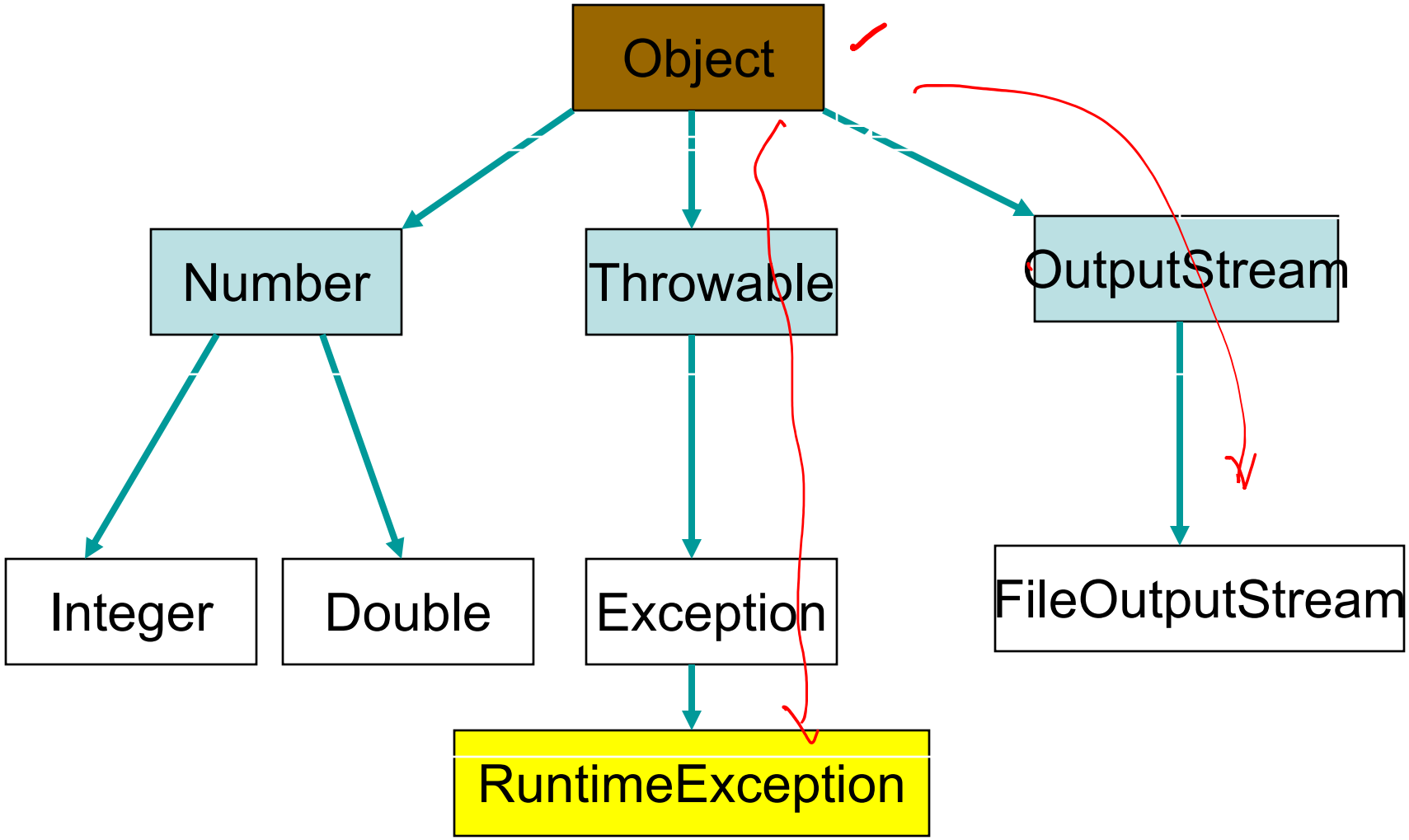
Definition

- A tree is a finite nonempty set of elements.
- One of these elements is called the root.
- The remaining elements, if any, are partitioned into trees, which are called the subtrees of t.

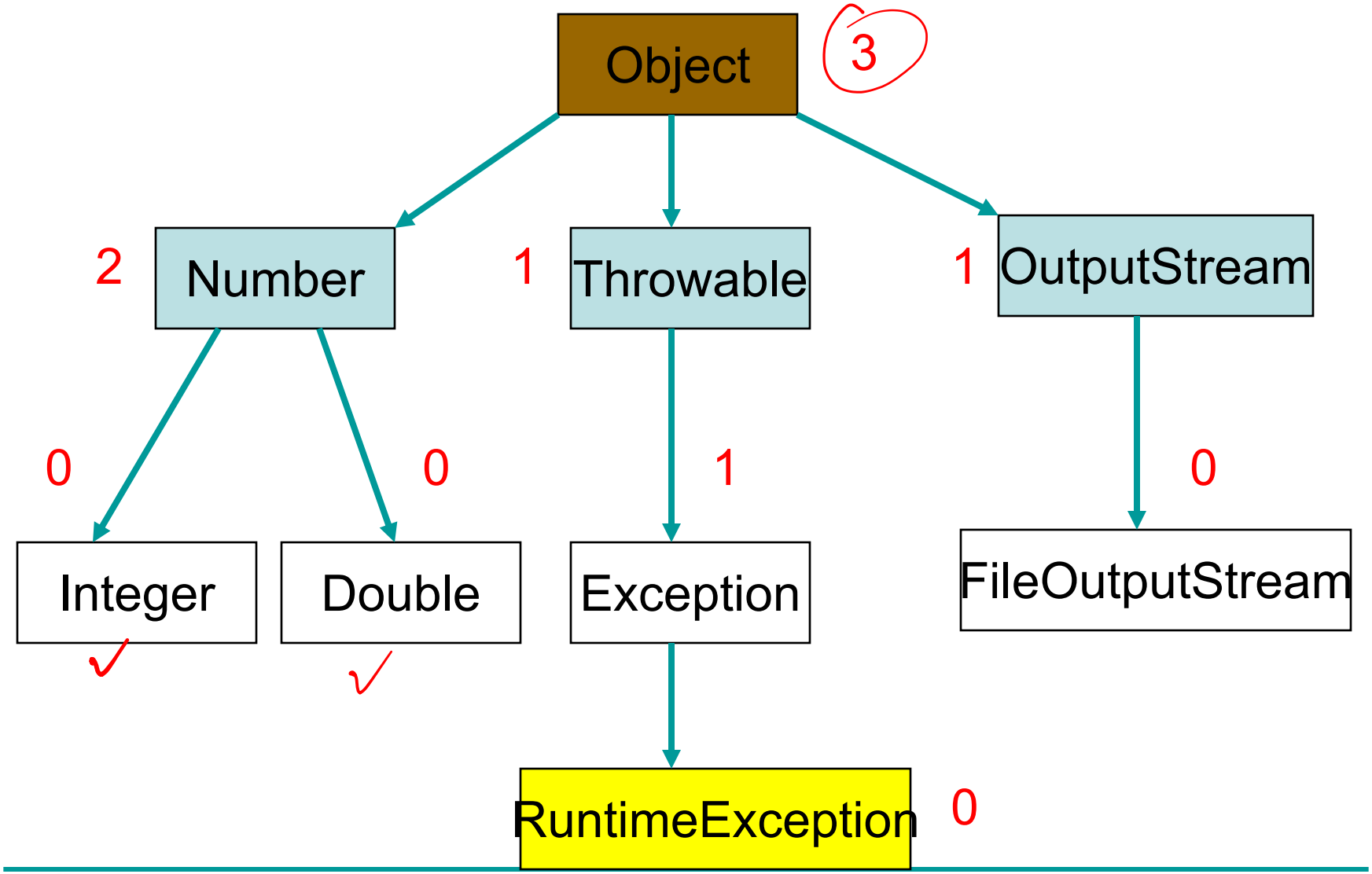
Caution

- Some texts start level numbers at 0 rather than at 1.
- Root is at level 0.
- Its children are at level 1.
- The grand children of the root are at level 2.
- And so on.
- We shall number levels with the root at level 1.

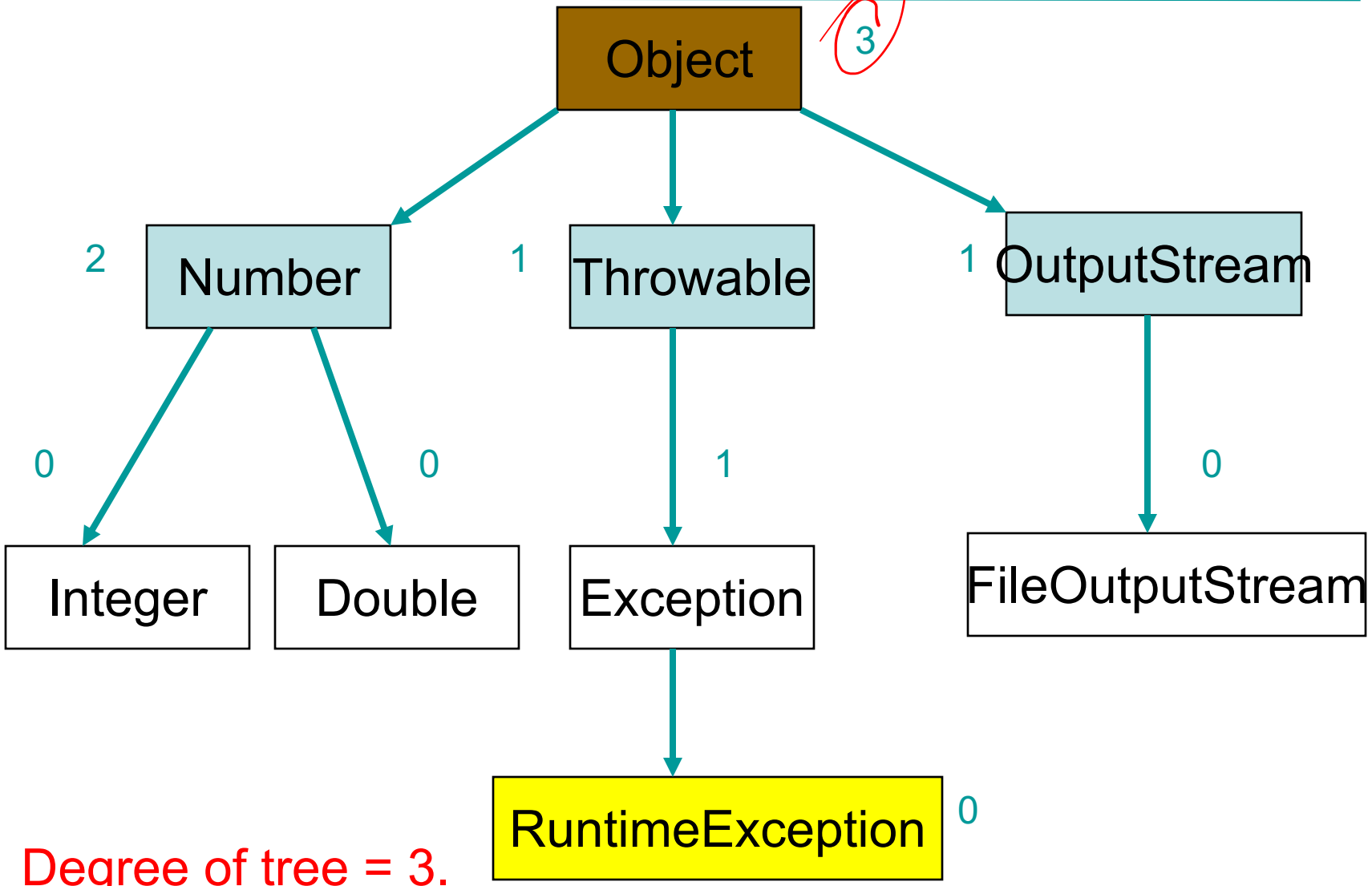
height = depth = number of levels



Node Degree = Number Of Children



Tree Degree = Max Node Degree



Degree of tree = 3.

Binary Tree

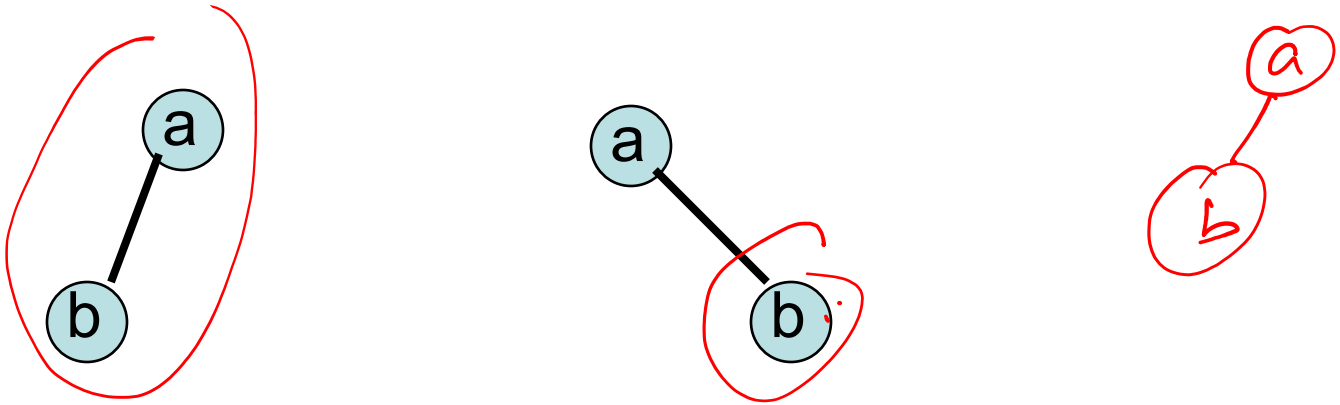
- Finite (possibly empty) collection of elements.
- A nonempty binary tree has a root element.
- The remaining elements (if any) are partitioned into two binary trees.
- These are called the left and right subtrees of the binary tree.

Differences Between A Tree & A Binary Tree

- No node in a binary tree may have a degree more than 2, whereas there is no limit on the degree of a node in a tree.
- A binary tree may be empty; a tree cannot be empty.

Differences Between A Tree & A Binary Tree

- The subtrees of a binary tree are ordered; those of a tree are not ordered.

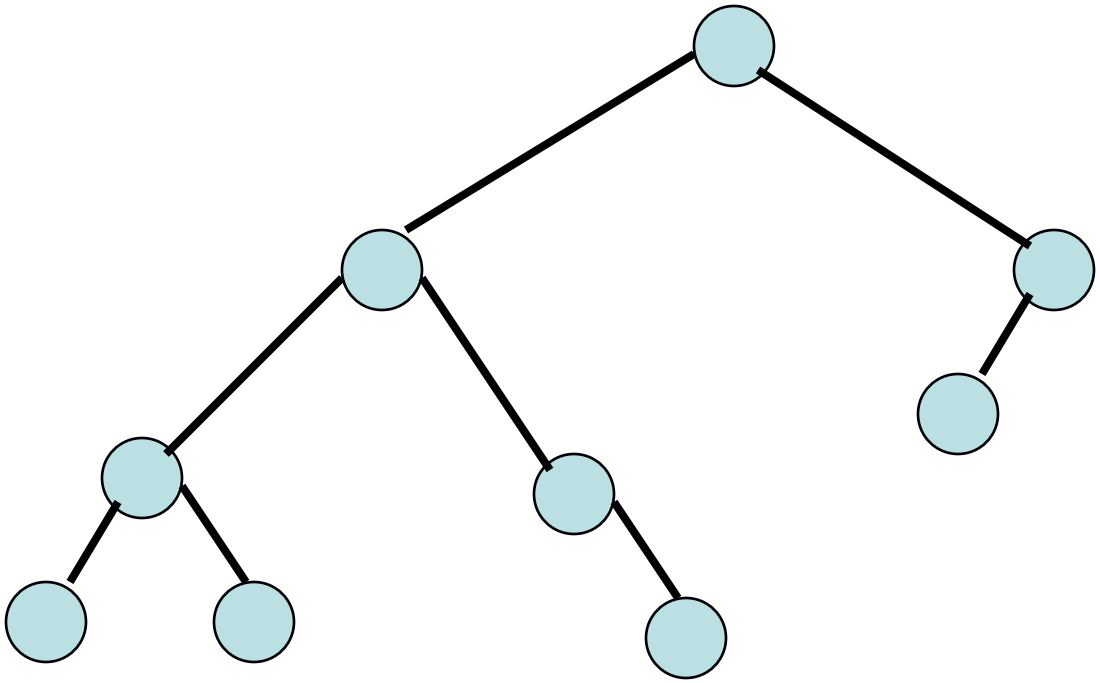


- Are different when viewed as binary trees.
- Are the same when viewed as trees.

Arithmetic Expressions

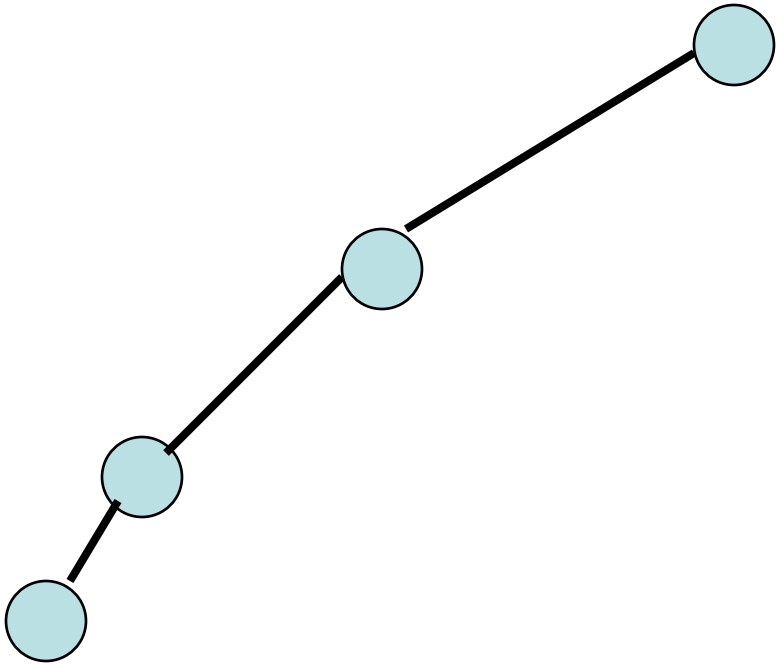
- (a + b) * (c + d) + e – f/g*h + 3.25
- Expressions comprise three kinds of entities.
  - Operators (+, -, /, *).
  - Operands (a, b, c, d, e, f, g, h, 3.25, (a + b), (c + d), etc.).
  - Delimiters ((,)).

Binary Tree Properties & Representation



Minimum Number Of Nodes

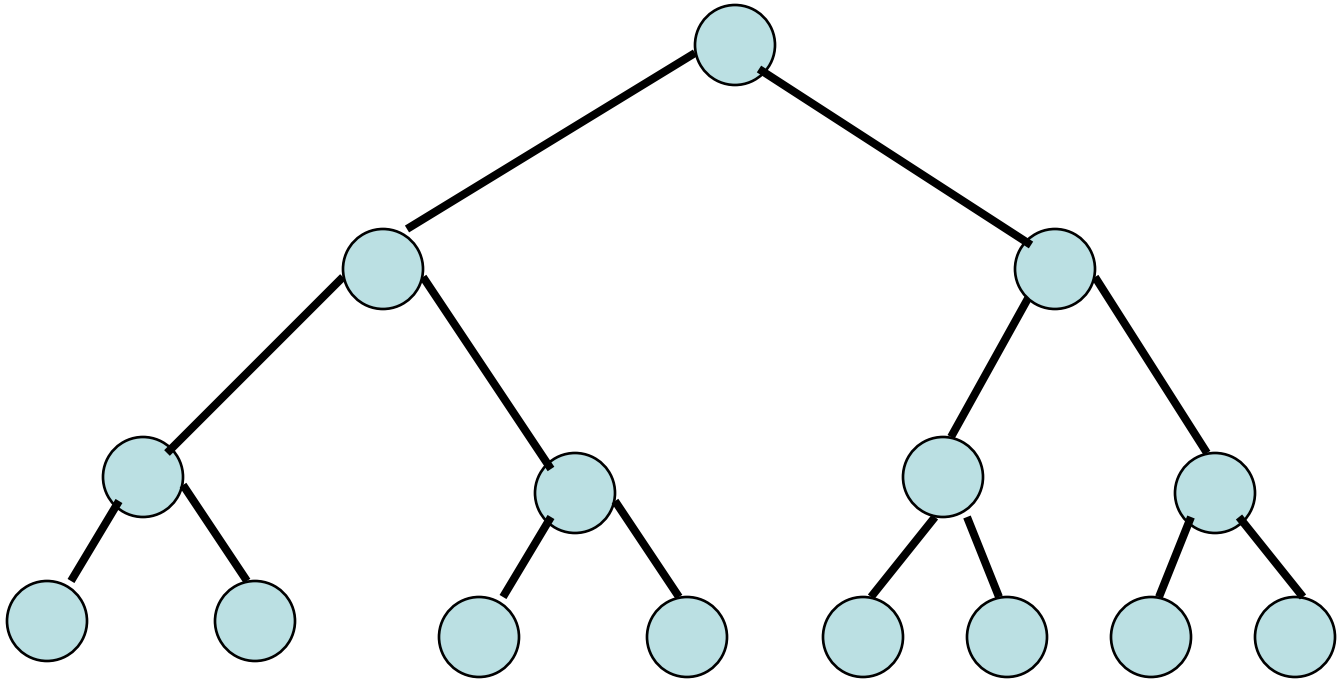
- Minimum number of nodes in a binary tree whose height is h.
- At least one node at each of first h levels.



minimum number of nodes is h

Maximum Number Of Nodes

- All possible nodes at first h levels are present.



Maximum number of nodes

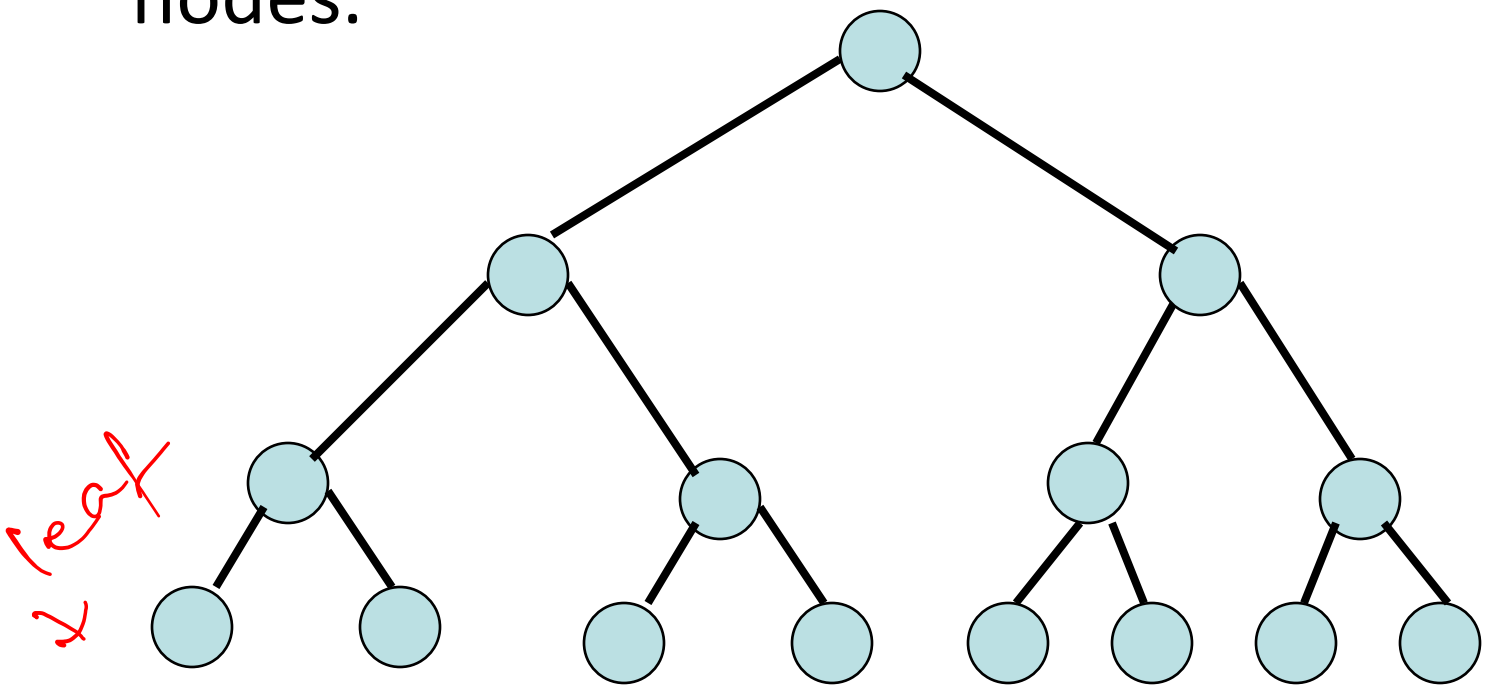
$= 1 + 2 + 4 + 8 + \ldots + 2^{h-1} = 2^h - 1$

Number Of Nodes & Height

- Let n be the number of nodes in a binary tree whose height is h.
- $h <= n <= 2^h - 1$
- $\log_2(n+1) <= h <= n$

Full Binary Tree

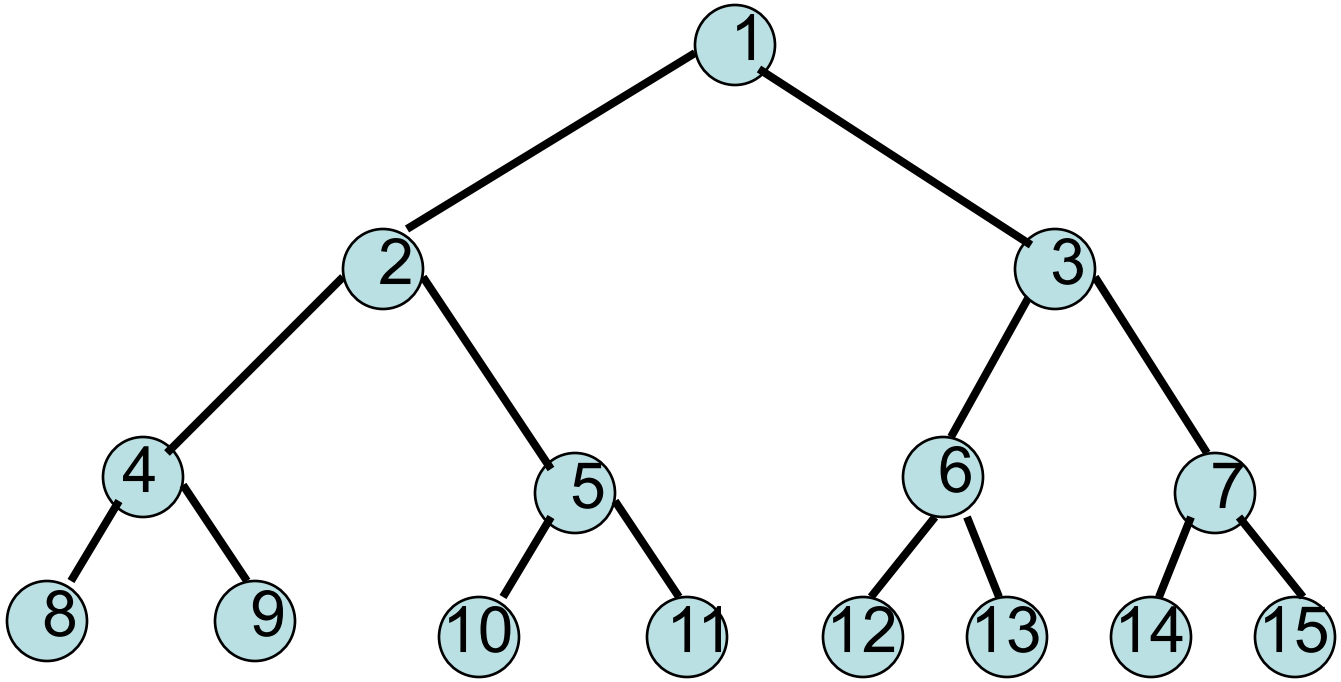
- A full binary tree of a given height h has $2^h - 1$ nodes.



Height 4 full binary tree.

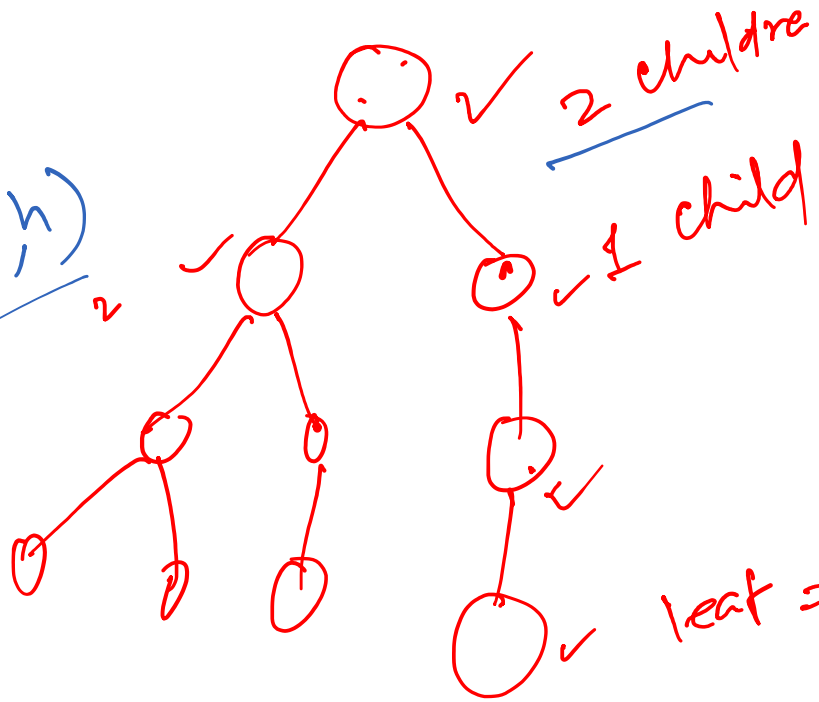
Numbering Nodes In A Full Binary Tree

- Number the nodes 1 through $2^h - 1$.
- Number by levels from top to bottom.
- Within a level number from left to right.



BINARY TREE

$m_0 = f(m_2, m_1, h)$



degree

height = 4

1 child = 3

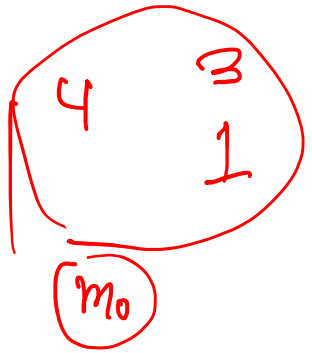
2 children = 3

m_0 =

h, m_2 → # nodes with 2 children

m_1 : # nodes with 1 child

m_0 : leaf nodes

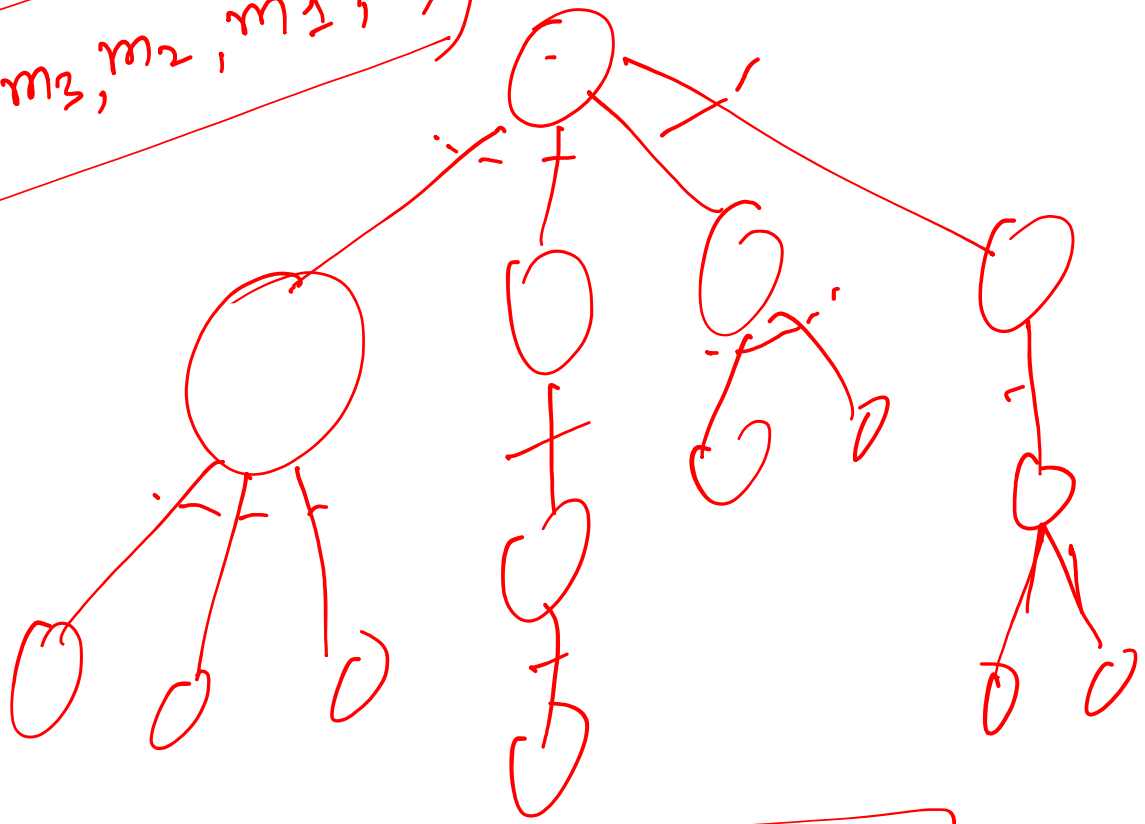


$2m_2 + m_1 + 1 =$

$m_0 = f(m_2, m_1, h)$

$2m_2 + m_1 \times 1 - 1$

$m_0 = f(m_4, m_3, m_2, m_1, h)$



h

Generalized Tree.

Assignment: Hashing with linear probing & Quadratic probing

Thank You