Remaining Module-1

B Trees, Binomial Heap, Matrix Multiplication using DP

BCSC0012

Designing Techniques of Algorithms

Design and Analysis of Algorithms

Dr. Gaurav Kumar
Asst. Prof, CEA, GLAU, Mathura

Designing Techniques of Algorithms

01

B Trees

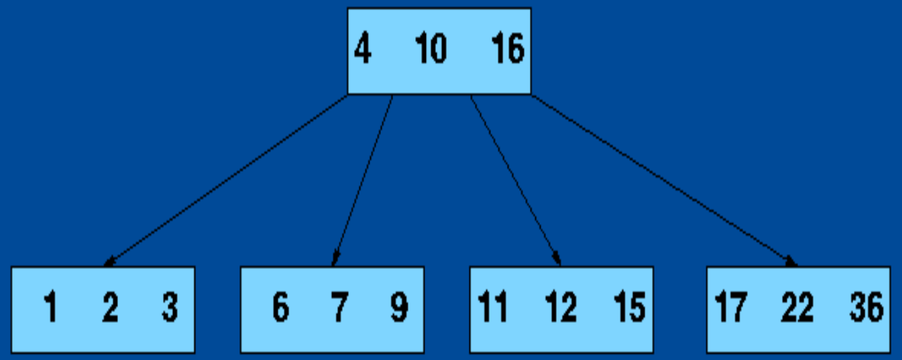
02

Binomial Heap

03

Matrix Chain Multiplication using Dynamic Programming

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU



01

B Trees

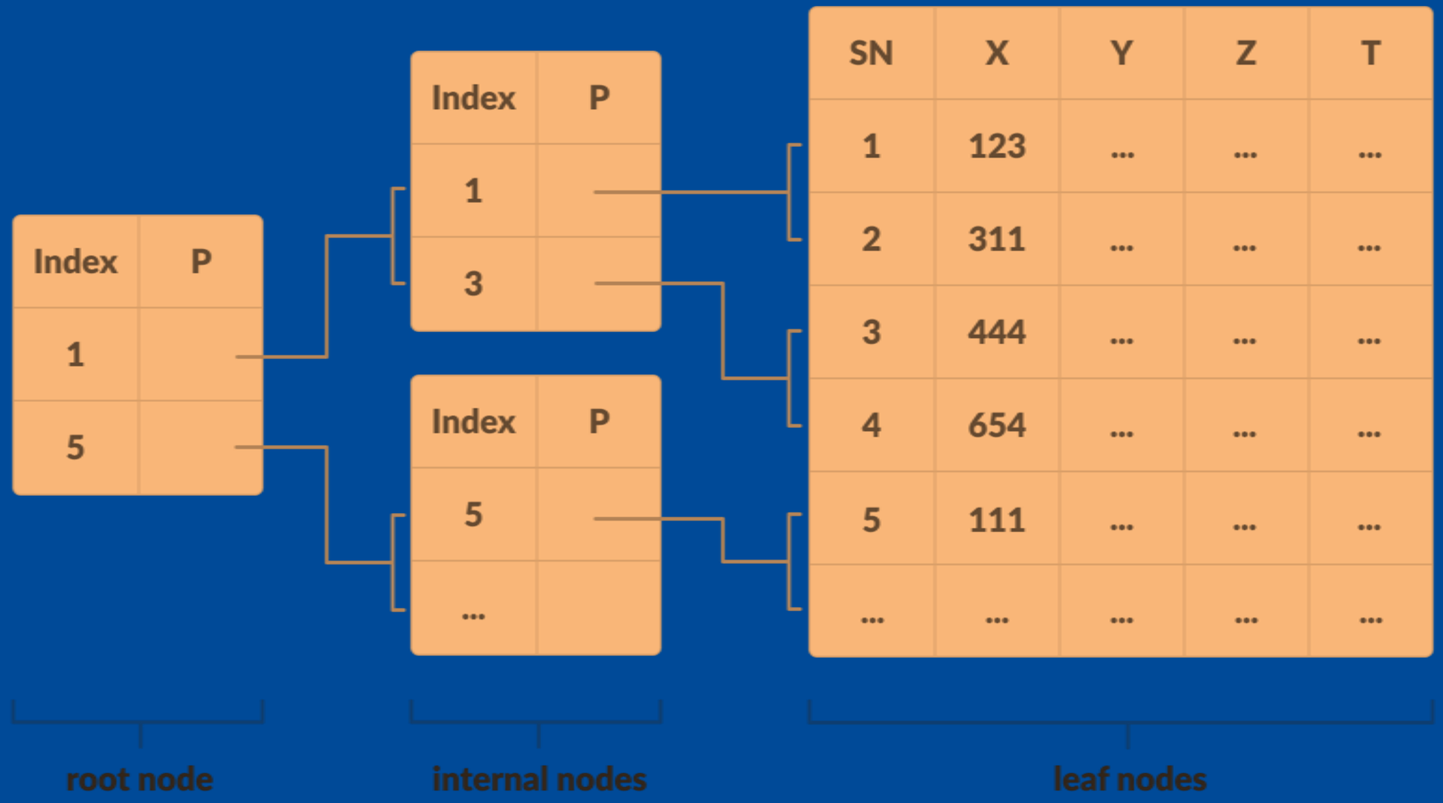
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

B Trees

- **B-Tree is a self-balancing search tree**, It is a specialized form of m-way search tree that can be widely used for disk access in the form of **multilevel indexing**.
- In most of the other self-balancing search trees (like AVL and Red-Black Trees), it is assumed that everything is in main memory.
- To understand the use of B-Trees,
  - we must think of the huge amount of data that cannot fit in main memory.
  - Disk access time is very high compared to the main memory access time.
- The **main idea of using B-Trees is to reduce the number of disk accesses**.
- Another **main reason of using B tree is its capability to store large number of keys in a single node** and large key values by keeping the height of the tree relatively small. It is **also called Fat Tree**.

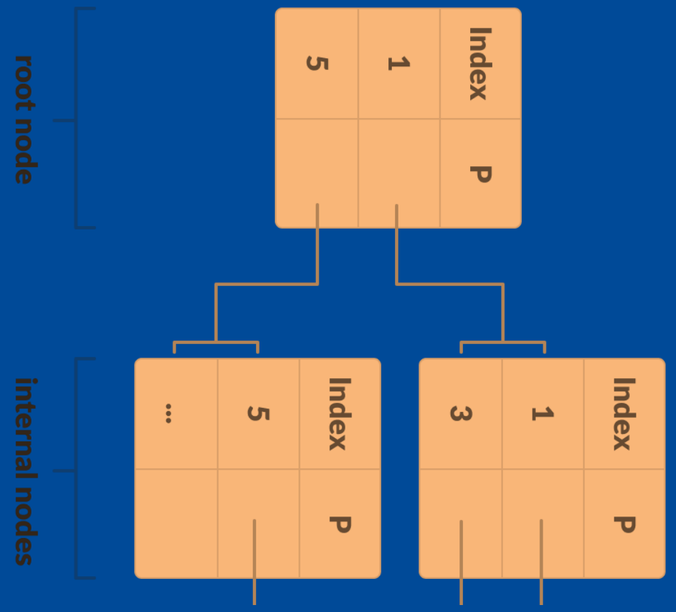
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

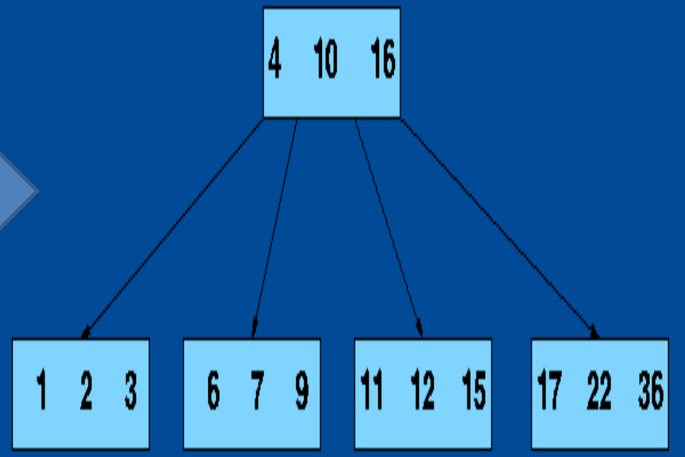
B Trees Multilevel Indexing



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Visualization of B Trees Multilevel Indexing





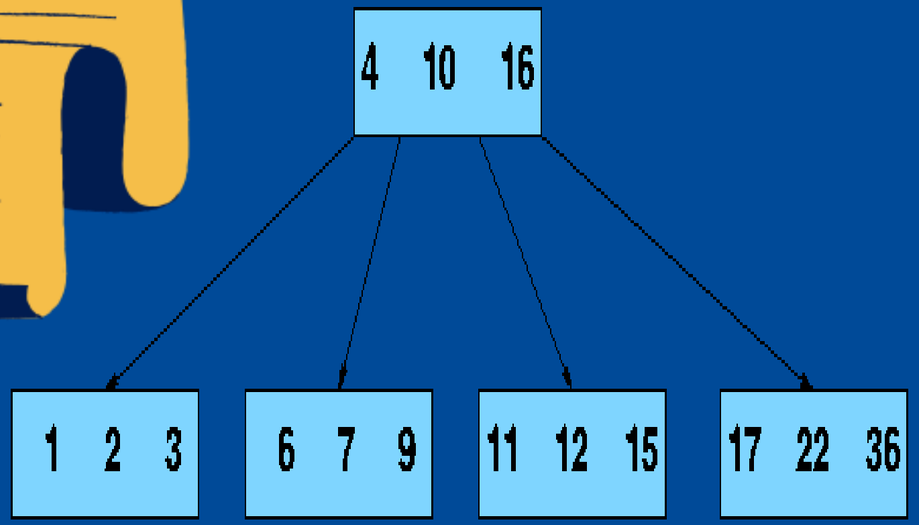
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Visualization of B Trees & Binary Search Tree

A B-Tree of order m can have at most **m-1 keys and m children**.

It is B Tree of order = 4

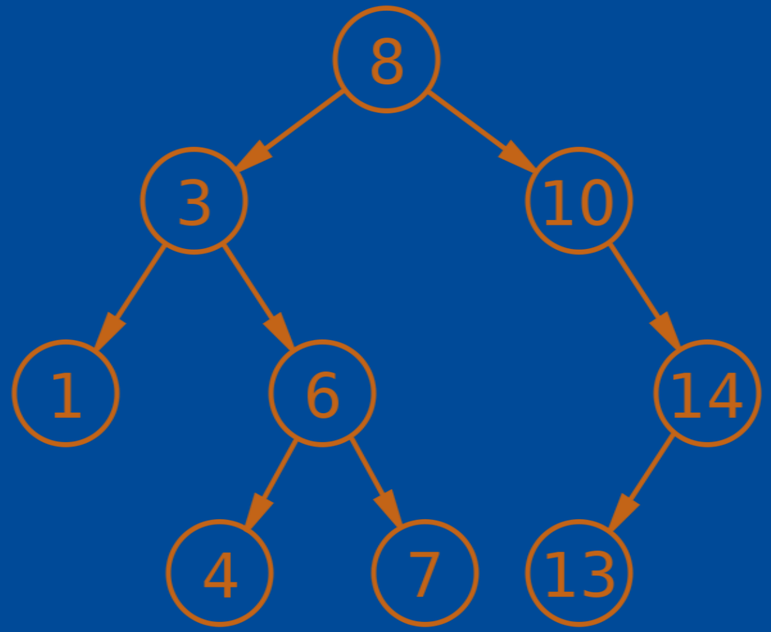
Means it can have maximum 4 children and each node can contain maximum 3 keys



B Tree
(Node contains max 3 Keys)

It is Binary Search Tree of order = 2

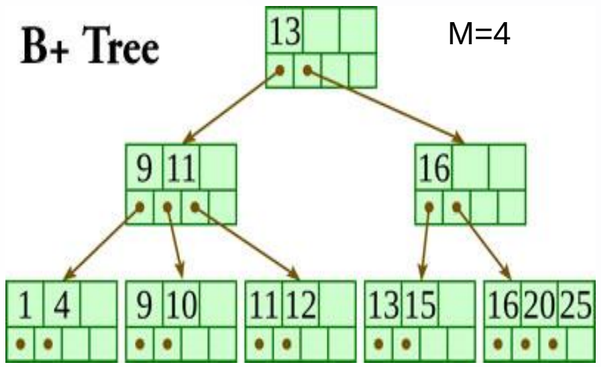
Means it can have maximum 2 children and each node can contain maximum 1 key



Binary Search Tree
(Node contains only one Key)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

B Trees Properties

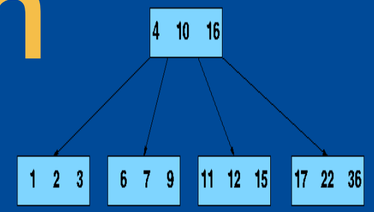


It contains the following properties.

1) Every node in a B-Tree of **order m contains at most m children and m-1 keys**.
2) **Every non-leaf node** (except root node) contain at least **[m/2] children**.
3) The **root nodes must have at least 2 children**.
4) A node (except root node) should contain a **minimum of [m/2] - 1 keys.**
5) **All keys of a node are sorted in increasing order**. If an internal node has 3 child nodes (subtrees), then it must have 2 keys: a_1 and a_2. All values in the leftmost subtree will be less than a_1, all values in the middle subtree will be between a_1 and a_2, and all values in the rightmost subtree will be greater than a_2.
6) Insertion of a Node in B-Tree happens only at Leaf Node.
7) All leaf nodes must be at the same level.
8) Like other balanced Binary Search Trees, time complexity to search, insert and delete is O(log n).

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

B Trees Construction



Insertion Operation

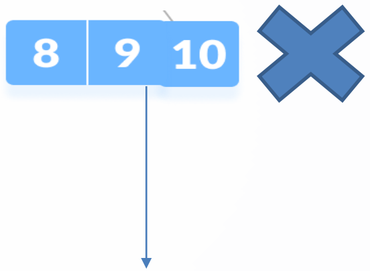
1. If the tree is empty, allocate a root node and insert the key.
2. Update the allowed number of keys in the node.
3. Search the appropriate node for insertion.
4. If the node is full, follow the steps below.
    a) Insert the elements in increasing order.
    b) Now, there are elements greater than its limit. So, split at the median.
    c) Push the median key upwards and make the left keys as a left child and the right keys as a right child.
5. If the node is not full, follow the steps 2.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Example of B Trees Constructions

The keys are 8, 9, 10, 11, 15, 20, 17. Order = 3

A node of order 3 can contain max 2 keys

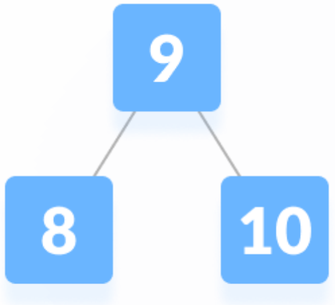


Step 4 : If the node is full, follow the steps below.

a) Insert the elements in increasing order.

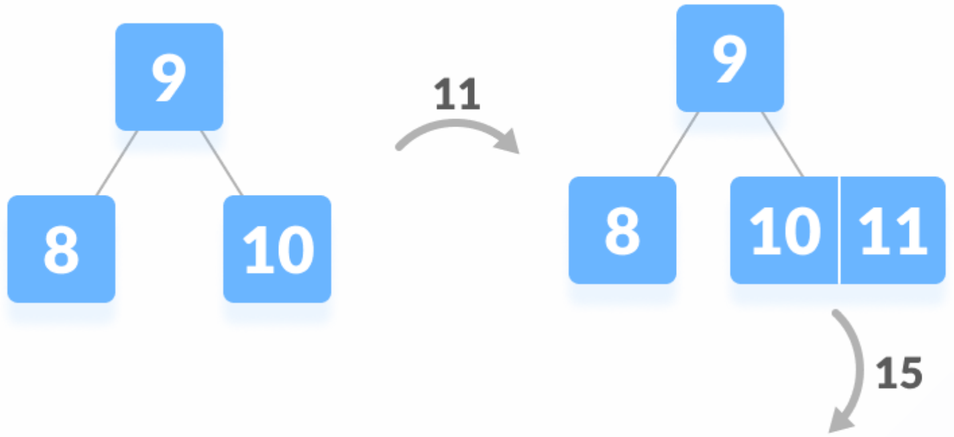
b) Now, there are elements greater than its limit. So, split at the median.

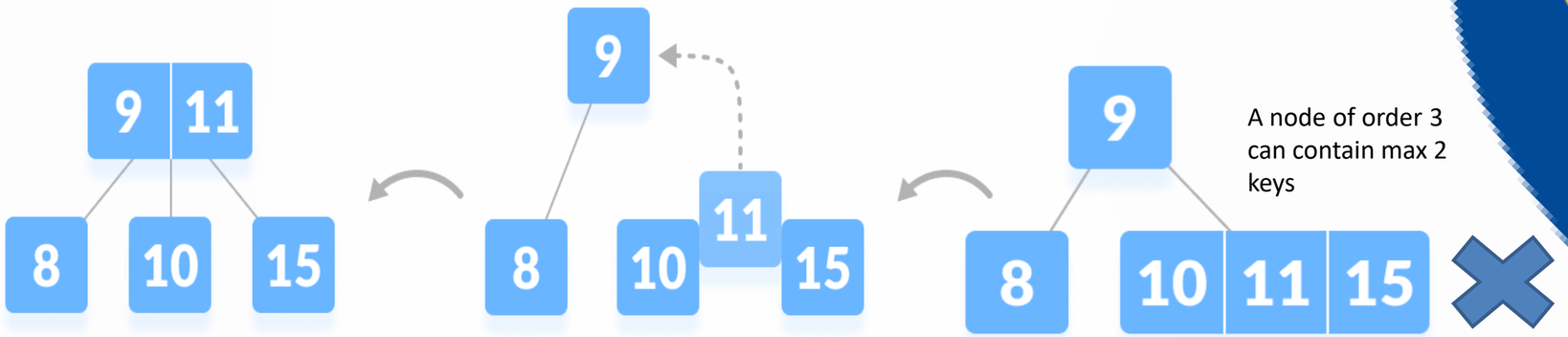
c) Push the median key upwards and make the left keys as a left child and the right keys as a right child.



B Trees Constructions

The keys are 8, 9, 10, 11, 15, 20, 17. Order = 3



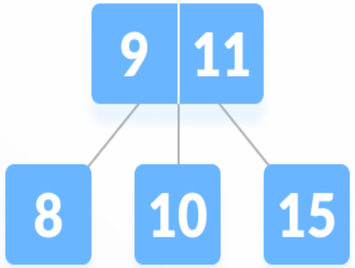


A node of order 3 can contain max 2 keys

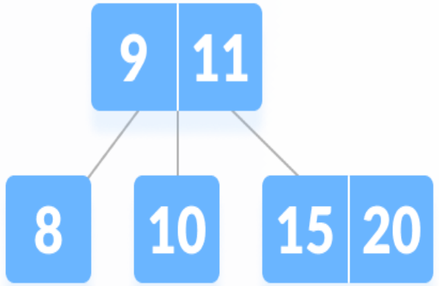
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

B Trees Constructions

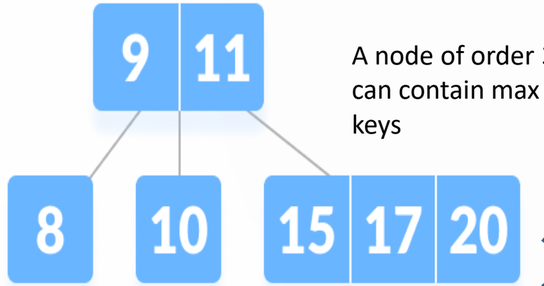
The keys are 8, 9, 10, 11, 15, 20, 17. Order = 3



20



17

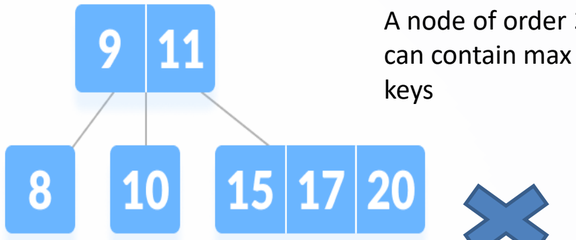


A node of order 3 can contain max 2 keys

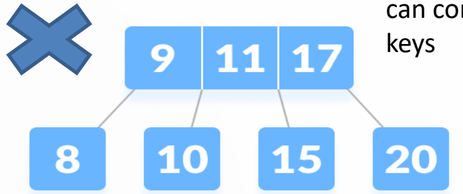
B Trees Constructions

The keys are 8, 9, 10, 11, 15, 20, 17.

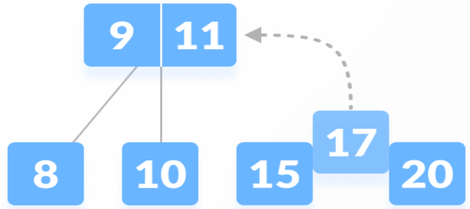
Order = 3

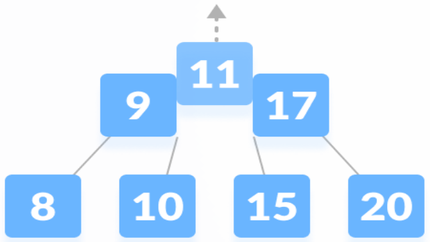


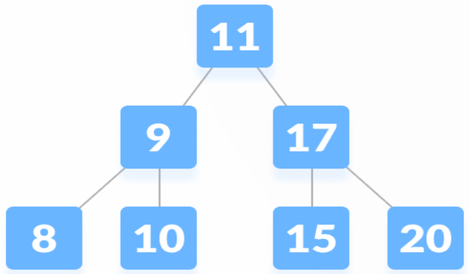
A node of order 3 can contain max 2 keys



A node of order 3 can contain max 2 keys







~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Deletion from B Trees

- Deleting an element on a B-tree consists of three main events:
  I. searching the node where the key to be deleted exists,
  II. deleting the key and
  III. balancing the tree if required.

- While deleting a tree, a condition called **underflow** may occur.

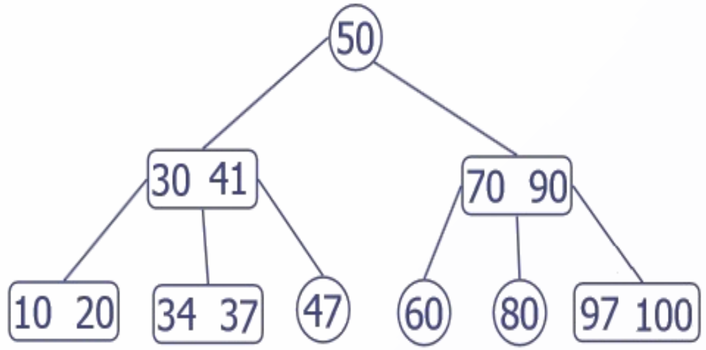
- Underflow occurs when a node contains less than the minimum number of keys it should hold. A node (except root node) should contain a **minimum of [m/2] - 1 keys.**

Deletion from B Trees

The terms to be understood before studying deletion operation are:

1. Inorder Predecessor: The largest key on the left child of a node is called its inorder predecessor.

Ex- inorder predecessor of 50 is 41.



2. Inorder Successor: The smallest key on the right child of a node is called its inorder successor.

Ex- inorder successor of 50 is 70.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

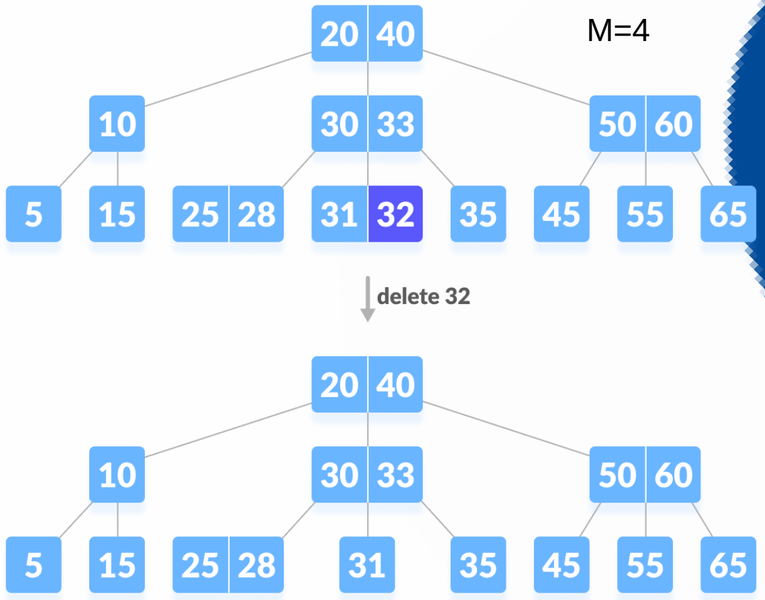
Deletion from B Trees

There are three main cases for deletion operation in a B tree.

Case I

The key to be deleted lies in the leaf. There are two cases for it.

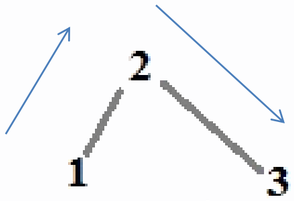
I. **The deletion of the key does not violate the property of the minimum number of keys a node should hold.**



In the given tree, deleting 32 does not violate the above properties.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Deletion from B Trees



Example

There are three main cases for deletion operation in a B tree.

Case I The key to be deleted lies in the leaf.

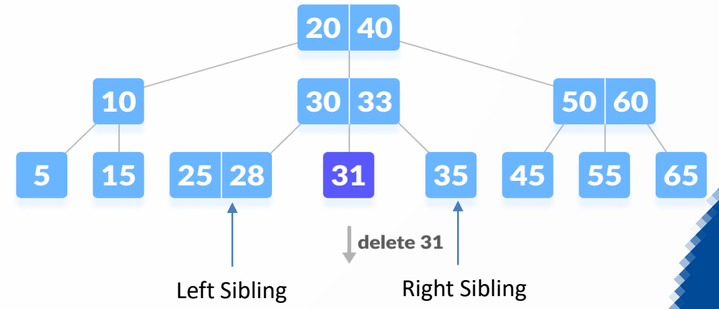
ii. The deletion of the key violates the property of the minimum number of keys a node should hold. In this case, We borrow a key from its immediate neighboring sibling node in the order of left to right.

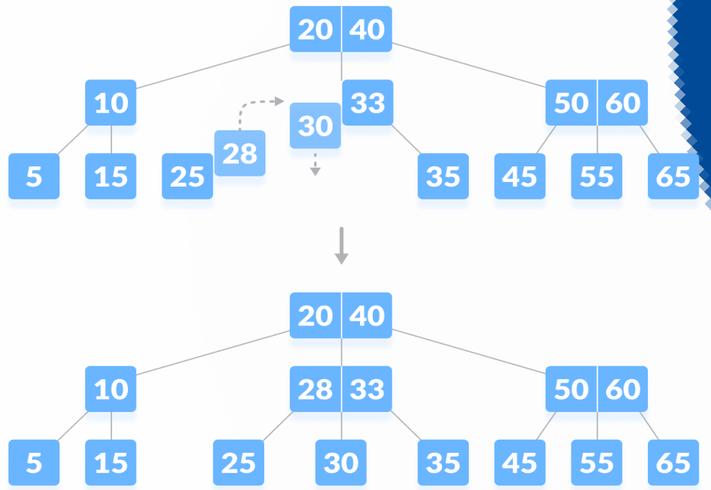
a) First, visit the immediate left sibling. If the left sibling node has more than a minimum number of keys, then borrow a key from this node.

b) Else, check to borrow from the immediate right sibling node.

Deleting 31 results in the second condition.

Let us borrow a key from the left sibling node.





~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Deletion from B Trees

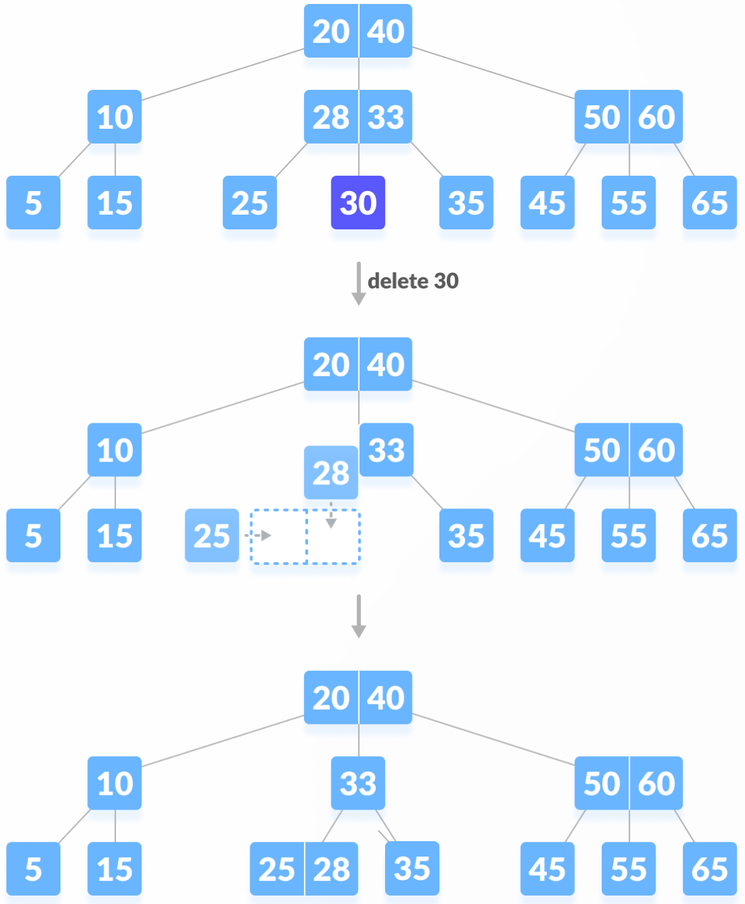
There are three main cases for deletion operation in a B tree.

Case I

The key to be deleted lies in the leaf.

c) If both the **immediate sibling nodes already have a minimum number of keys**, then merge the node with either the left sibling node or the right sibling node. **This merging is done through the parent node.**

Deleting 30 results of (C) case.



Deletion from B Trees

There are three main cases for deletion operation in a B tree.

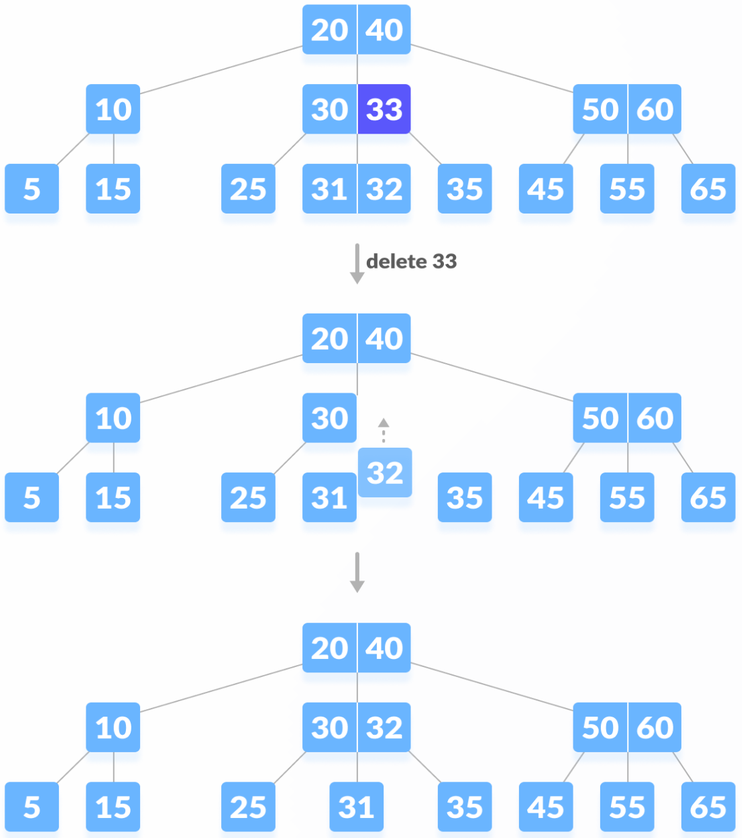
Case II

If the key to be deleted lies in the internal node, the following cases occur.

1) The internal node, which is deleted, is replaced by an inorder predecessor if the left child has more than the minimum number of keys.
2) The internal node, which is deleted, is replaced by an inorder successor if the right child has more than the minimum number of keys.

* The largest key on the left child of a node is called its inorder predecessor.

Deleting 33 results of Case 2



Deletion from B Trees

There are three main cases for deletion operation in a B tree.

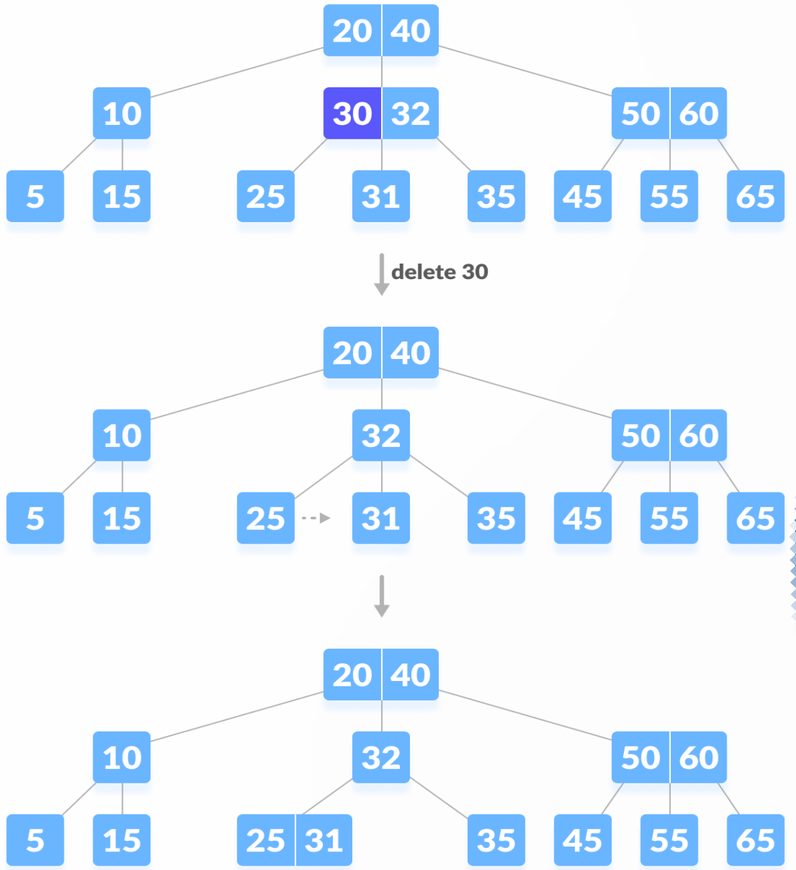
Case II

If the key to be deleted lies in the internal node, the following cases occur.

3) If either child has exactly a minimum number of keys then, merge the left and the right children.

Note: After merging if the parent node has less than the minimum number of keys then, look for the siblings as in Case I.

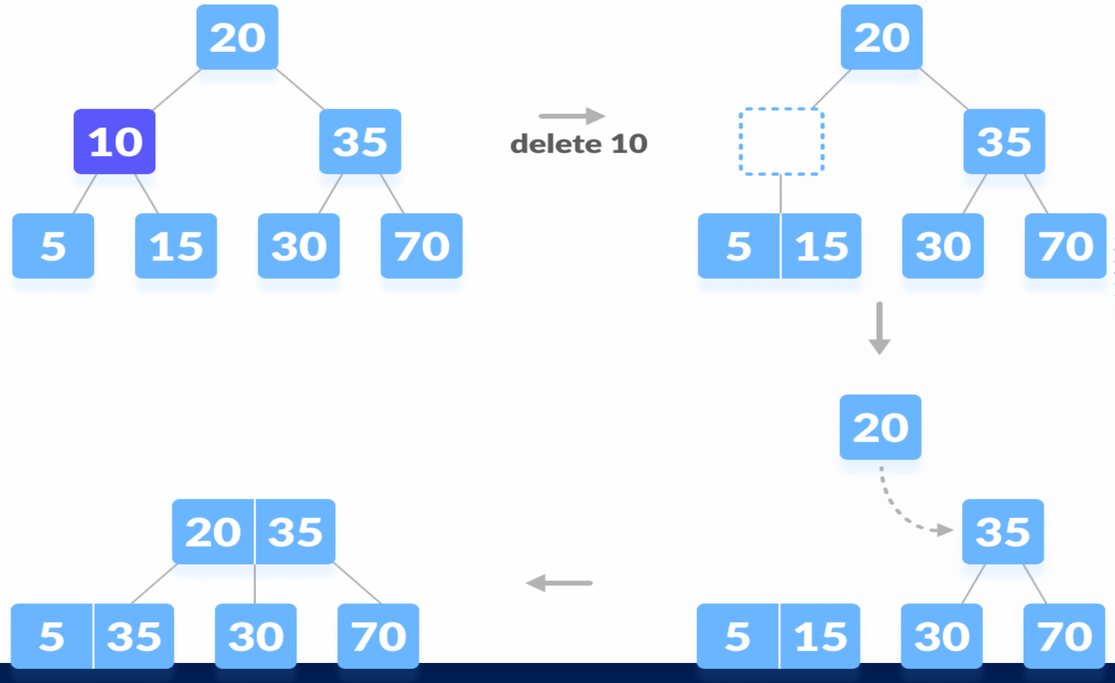
Deleting 30 results of third sub case of Case 2

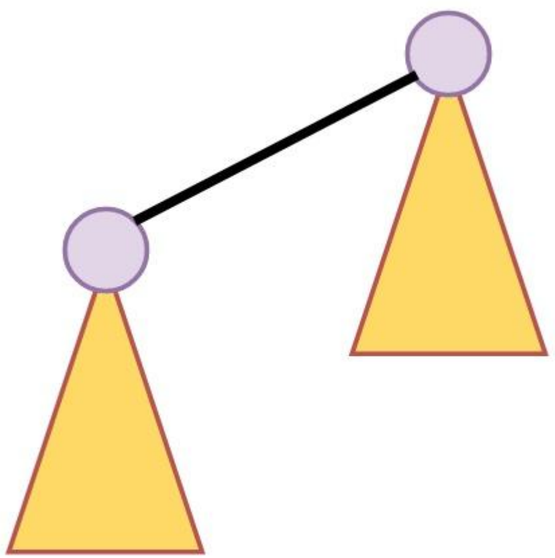


Deletion from B Trees

Case III In this case, the height of the tree shrinks.

- If the target key lies in an internal node, and the deletion of the key leads to a fewer number of keys in the node (i.e. less than the minimum required),
  - Then look for the inorder predecessor or the inorder successor for replacement (if child has more than one keys)
  - If both the children contain a minimum number of keys then, borrowing cannot take place. This leads to Case II(3) i.e. merging the children.
  - Again, look for the sibling to borrow a key. But, if the sibling also has only a minimum number of keys then, merge the node with the sibling along with the parent. Arrange the children accordingly (increasing order).



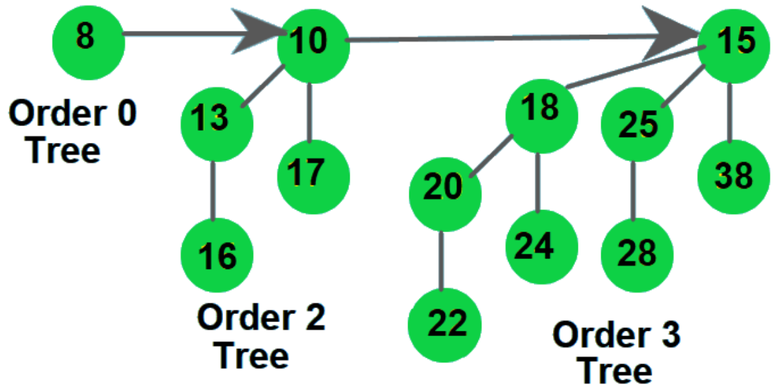


02. Binomial Heap

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Binomial Heap

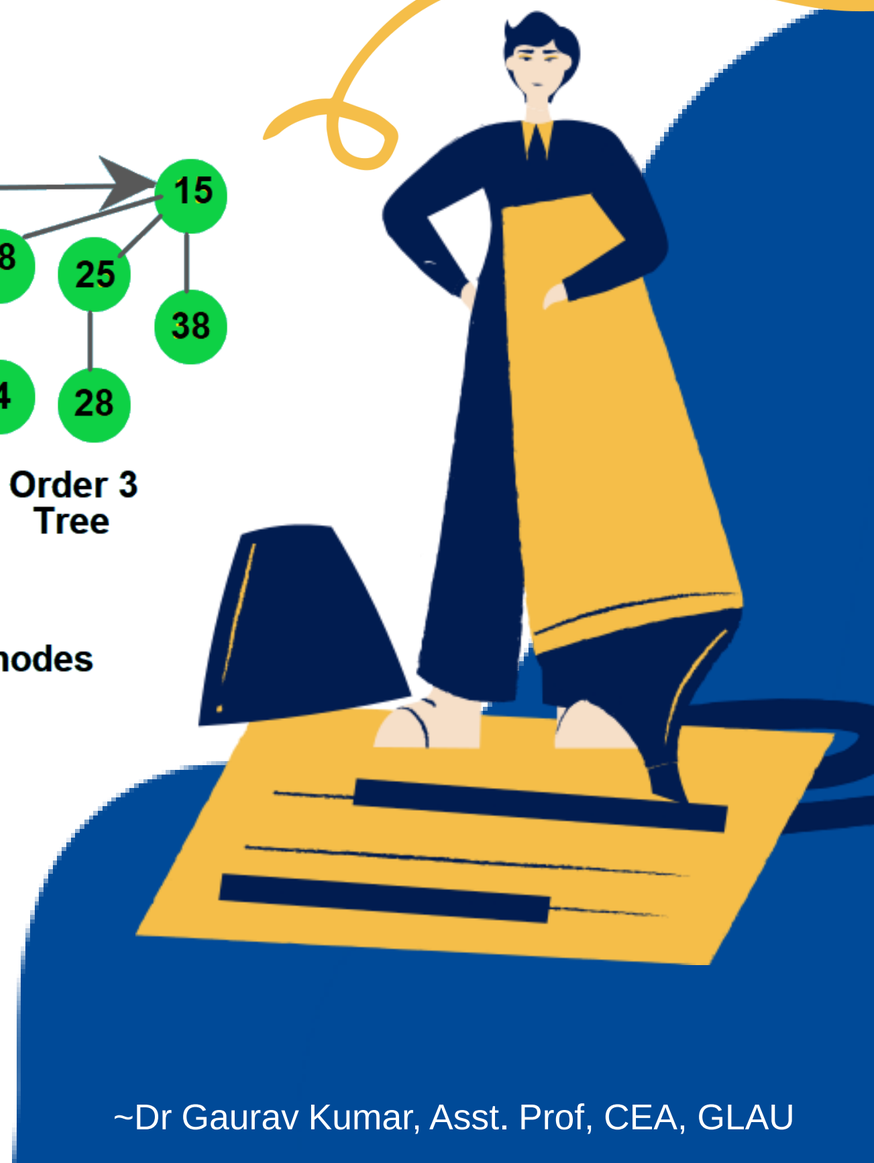
- A binomial heap can be defined as the collection of **binomial tree** that satisfies the min/max heap properties.
  - The min/max heap is a heap in which each node has a value lesser/greater than the value of its child nodes.



A Binomial Heap with 13 nodes

Properties

1) Each binomial heap obeys the min-heap properties.
2) For any non-negative integer k, there should be atmost one binomial tree in a heap where root has degree k.



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding Binomial Tree

- A Binomial Tree of **order k** can be constructed by **taking two binomial trees of order k-1** and making one as leftmost child of other.

Or

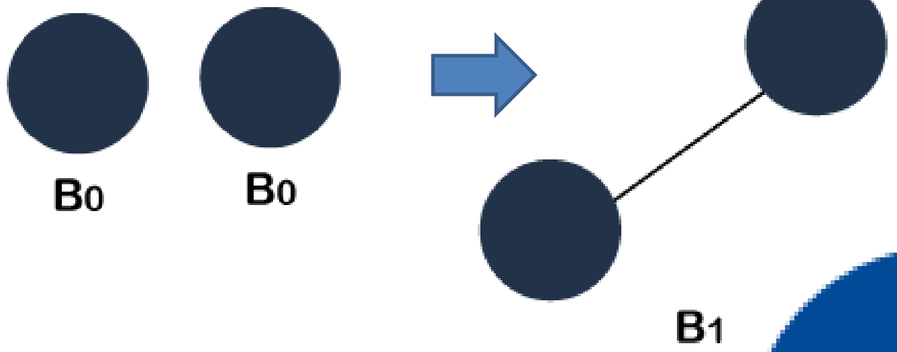
- **B_k** consists of two binomial trees, i.e., B_{k-1} and B_{k-1} are linked together in which one tree becomes the left sub- tree of another binomial tree.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Example of Binomial Tree

- If B_0 where k is 0, means that there would exist only one node in the tree as shown.

- If B_1, where k is 1, means k-1 equal to 0. Therefore, there would be two binomial trees of B_0 in which one B_0 becomes the left subtree of another B_0.

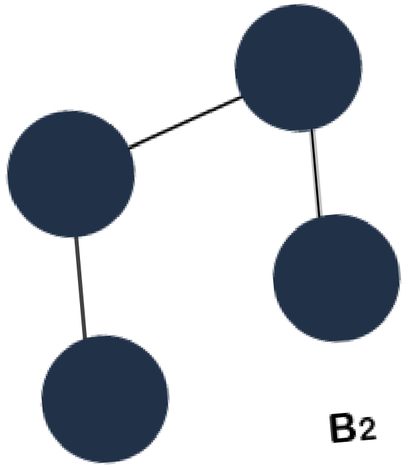


Note: A Binomial Tree of **order k** can be constructed by **taking two binomial trees of order k-1** and making one as leftmost child of other.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

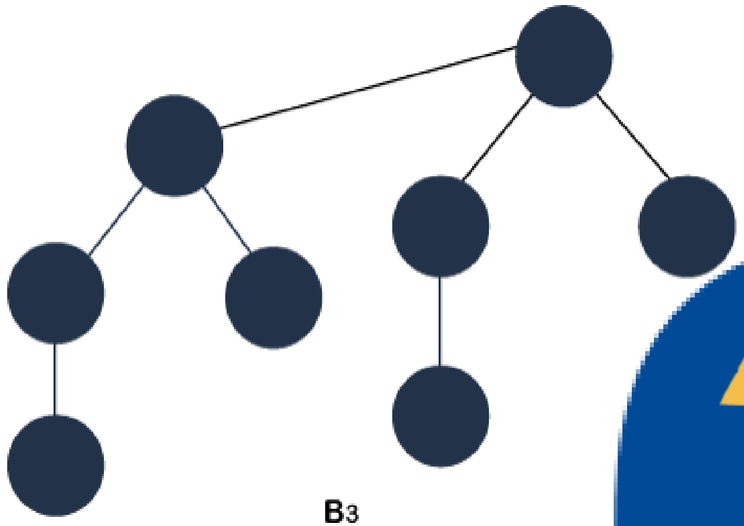
Example of Binomial Tree

- If B_2, where k is 2, means k-1 equal to 1. Therefore, there would be two binomial trees of B_1 in which one B_1 becomes the left subtree of another B_1.



B2

- If B_3 , where k is 3, means k-1 equal to 2. Therefore, there would be two binomial trees of B_2 in which one B_2 becomes the left subtree of another B_2.



B3

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Properties of Binomial Tree

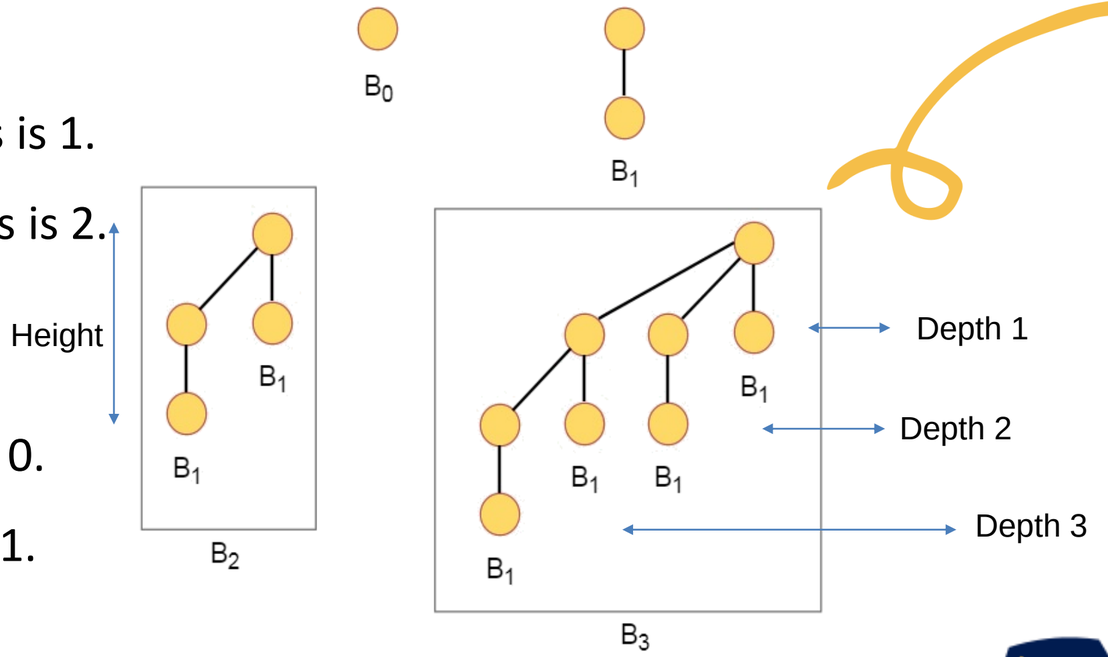
1) It has **exactly 2^k nodes**
   - If k=1 then $2^0 = 1$. The number of nodes is 1.

     If k = 2 then $2^1 = 2$. The number of nodes is 2.

2) The **height of the tree is k.**

   If k=0 then the height of the tree would be 0.

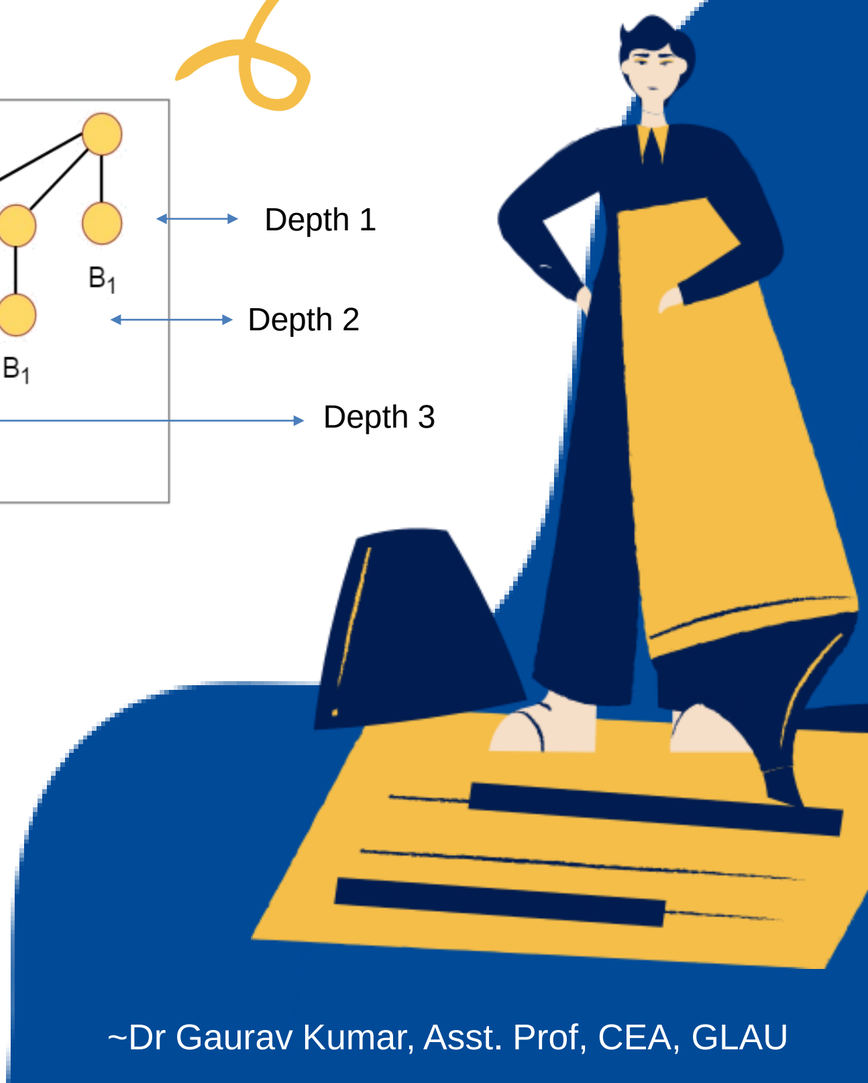
   If k=1 then the height of the tree would be 1.



3) There are **exactly ${}^kC_i$ nodes at depth i for i = 0, 1, . . . , k.**

   For example, if k is equal to 3 and at depth i=2, we have to determine the number of nodes. Here, ${}^3c_2 = 3$

4) The **root has degree k and children of root are themselves Binomial Trees with order k-1, k-2,.. 0 from left to right**.



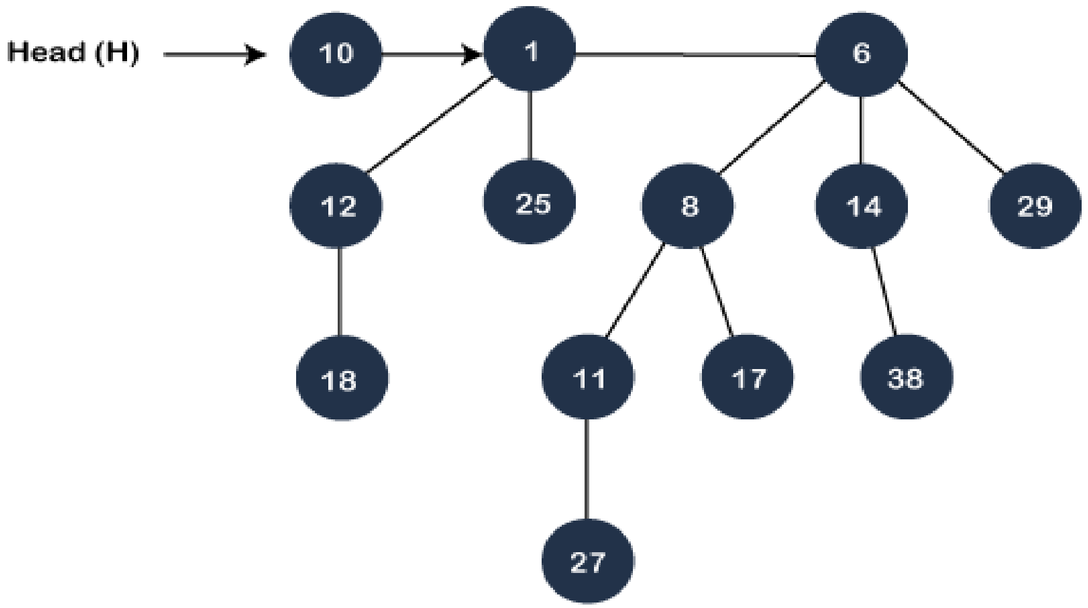
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding Binomial Heap

- A binomial heap can be defined as the collection of **binomial tree** that satisfies the min heap properties.

Properties

1) Each binomial heap obeys the min-heap properties.
2) For any non-negative integer k, there should be at-most one binomial tree in a heap where root has degree k.

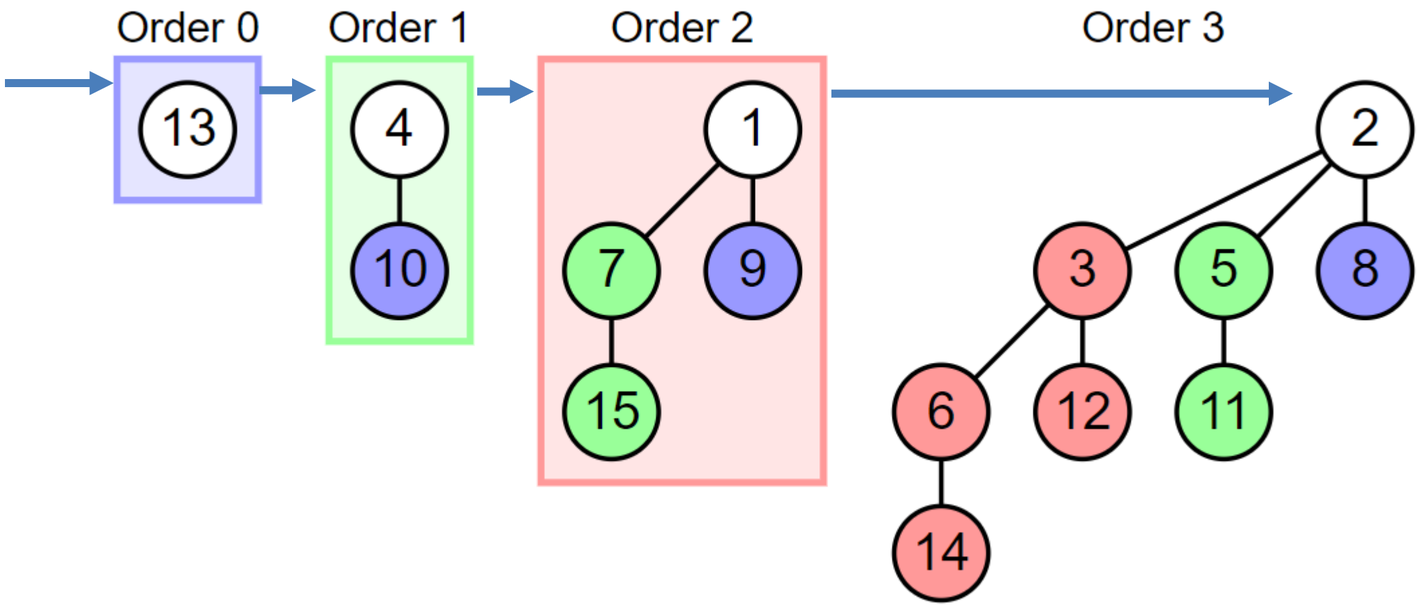


- Three binomial trees, i.e., B_0, B_2, B_3.
- The all three binomial trees satisfy the min heap's property as all the nodes have a smaller value than the child nodes.
- It also satisfies the second property of the binomial heap.

For example, if we consider the value of k as 3, then we can observe in the figure that the only one binomial tree of degree 3 exists.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

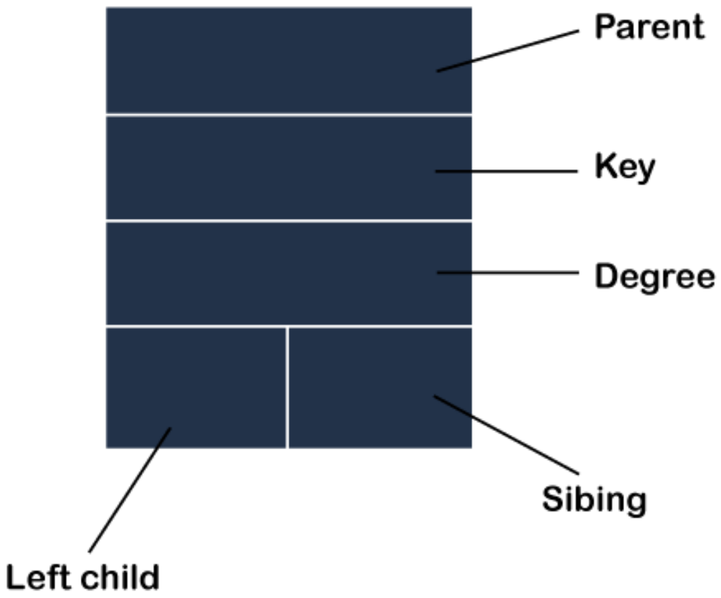
Understanding Binomial Heap



For **n number of nodes**, the binomial heap can contain **at-most Log(n) binomial tree.**

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Representation of Binomial Heap in Memory



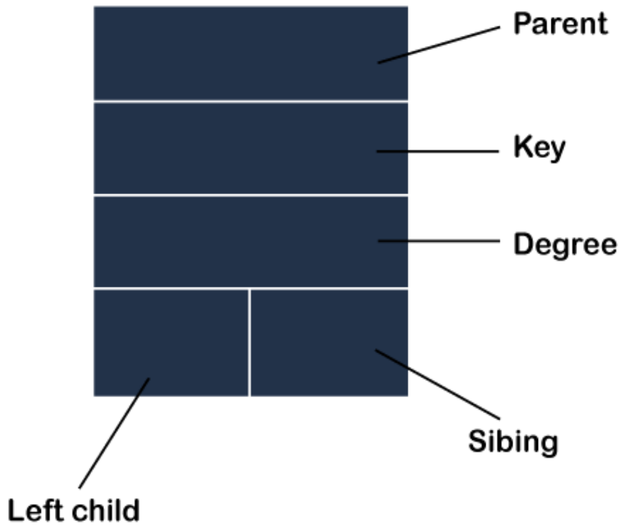
1) The first block in the memory contains the pointer that stores the address of the parent of the node.
2) The second block stores the key value of the node.
3) The third block contains the degree of the node.
4) The fourth block is divided into two parts, i.e., left child and sibling. The left child contains the address of the left child of a node, whereas the **sibling** contains the address of the sibling.

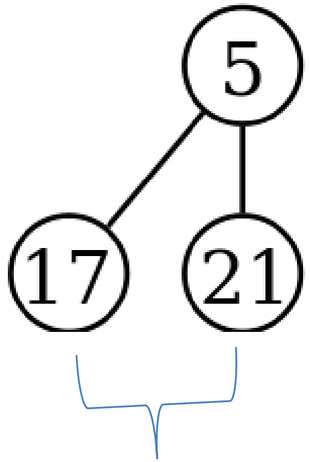
Pointers in each node: Each node has the following pointers:

1. A parent pointer pointing to the immediate parent of the node
2. A left pointer pointing to the first child of the node
3. A right pointer pointing to the next sibling of the node.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding the Binomial Heap in Memory



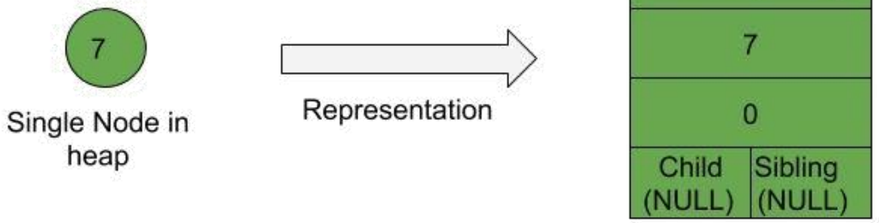


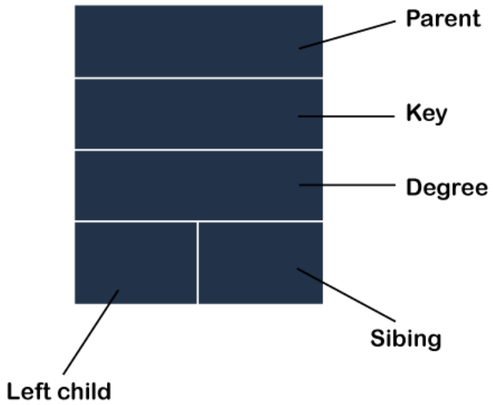
Left Child **Sibling (Same Level Children)**

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

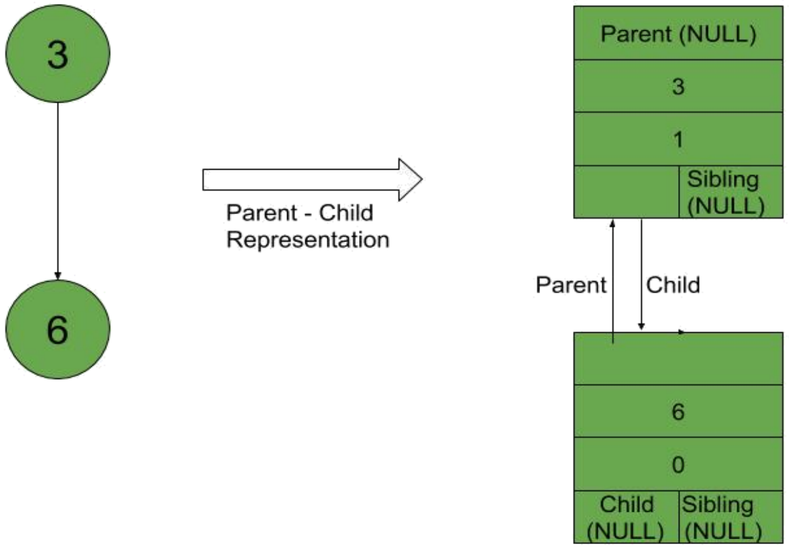
Understanding the Representation of Binomial Heap in Memory

Single node in the Heap



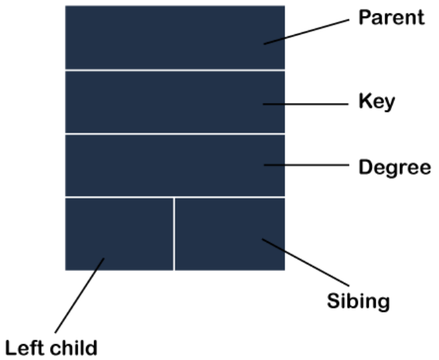


Parent – Child relationship between nodes

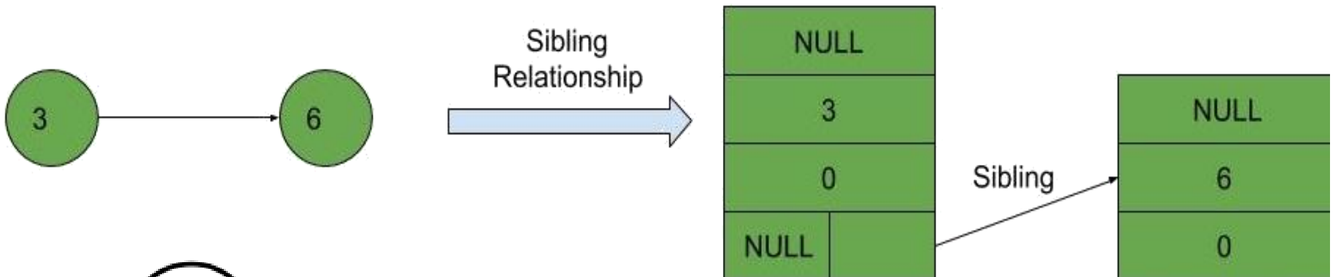


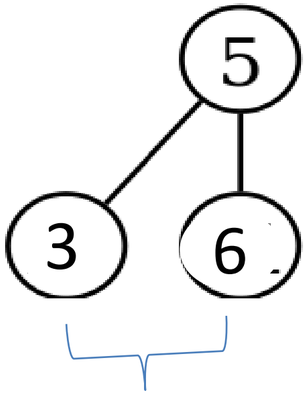
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding the Representation of Binomial Heap in Memory

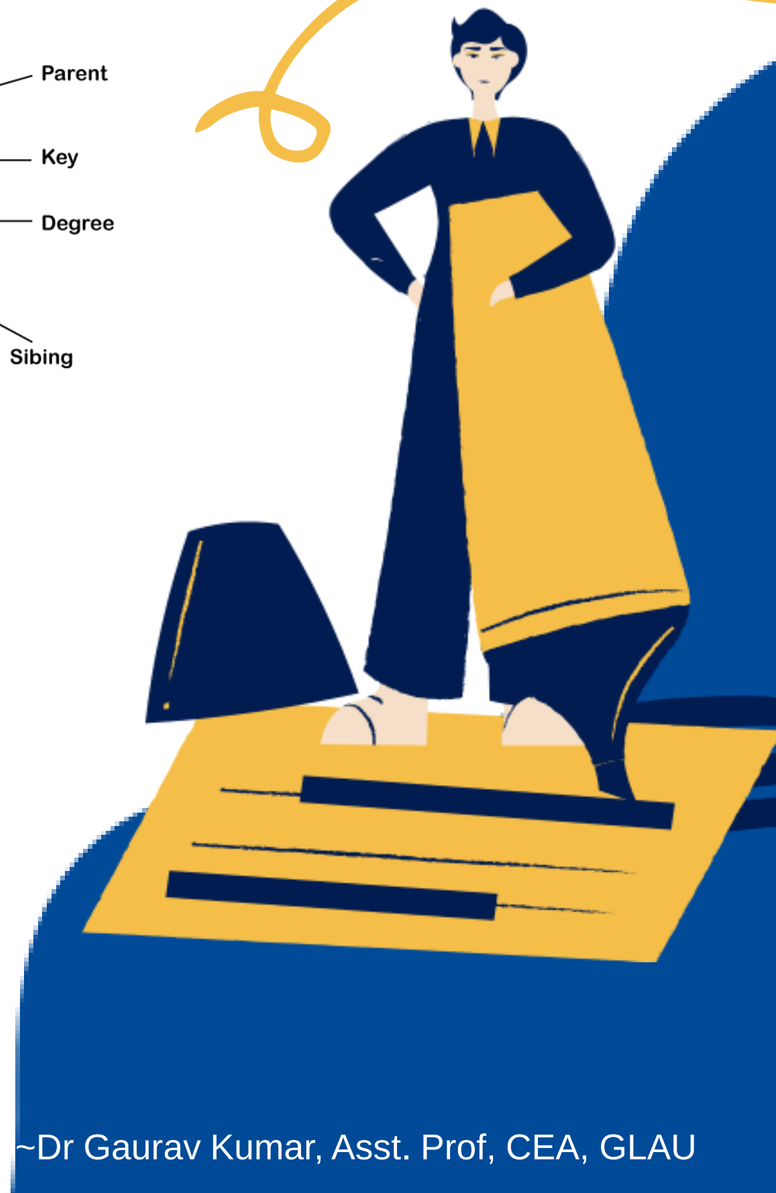


Sibling relationship between nodes



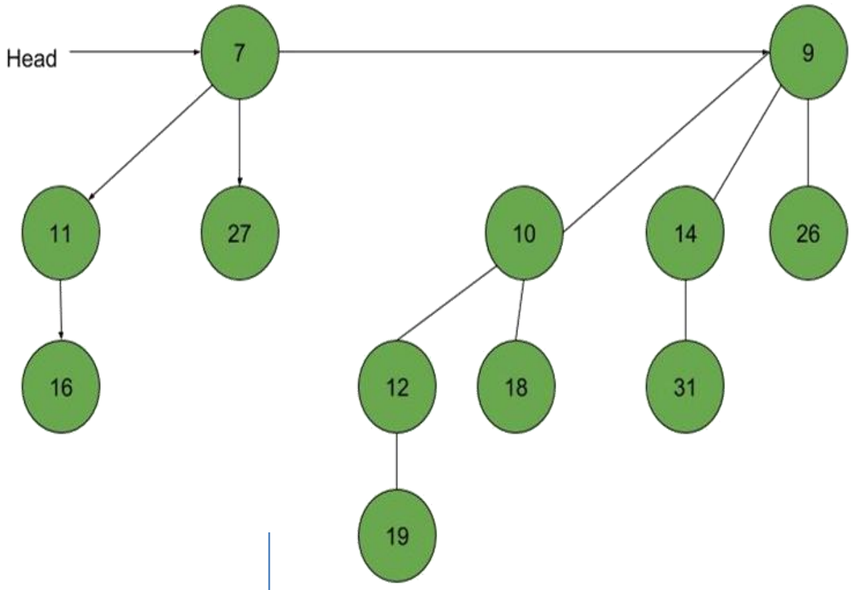


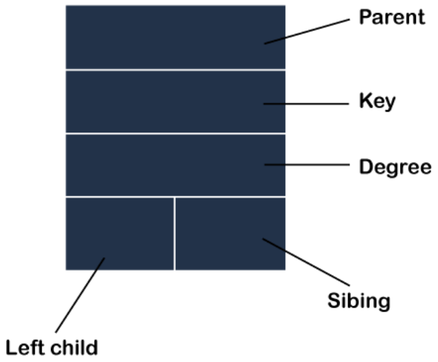
Sibling (Same Level Children)

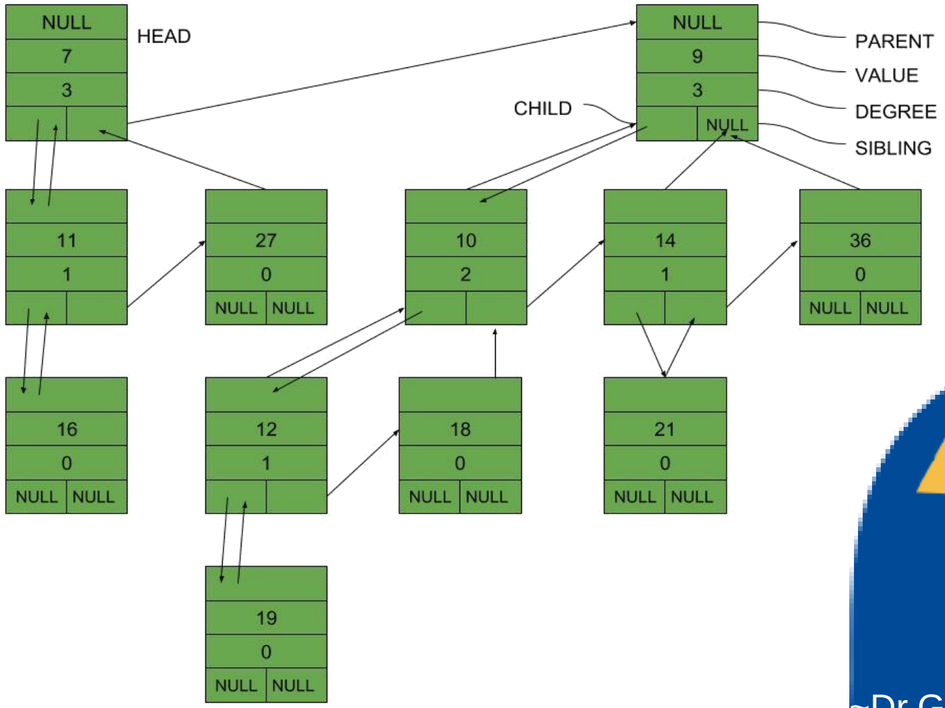


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

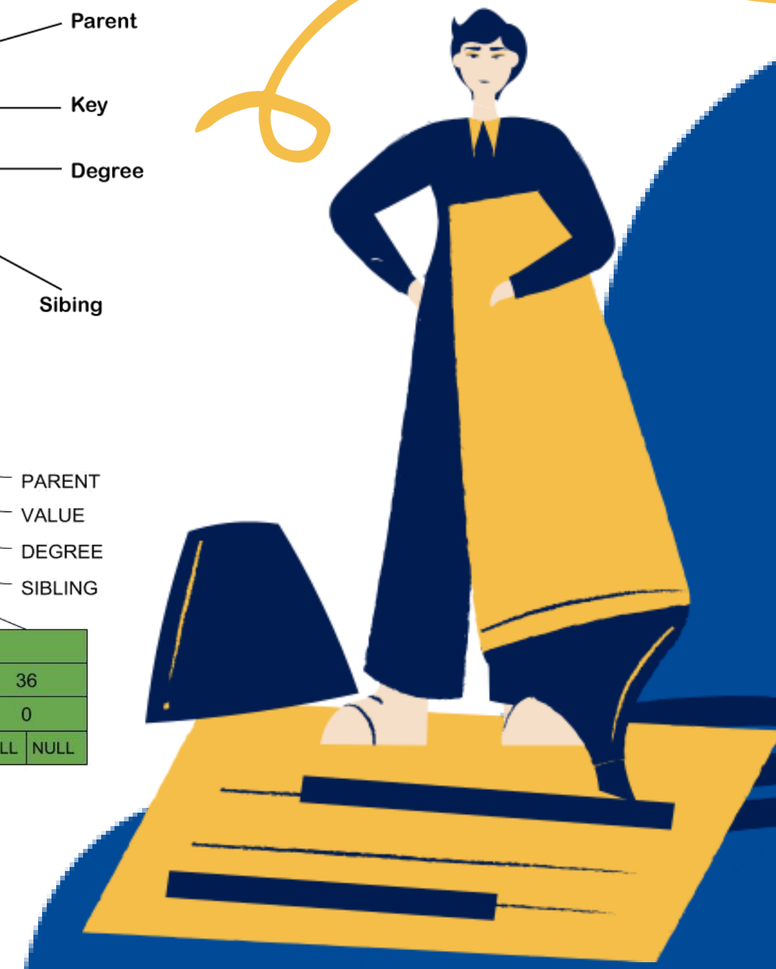
Understanding the Representation of Binomial Heap in Memory







Memory Representation

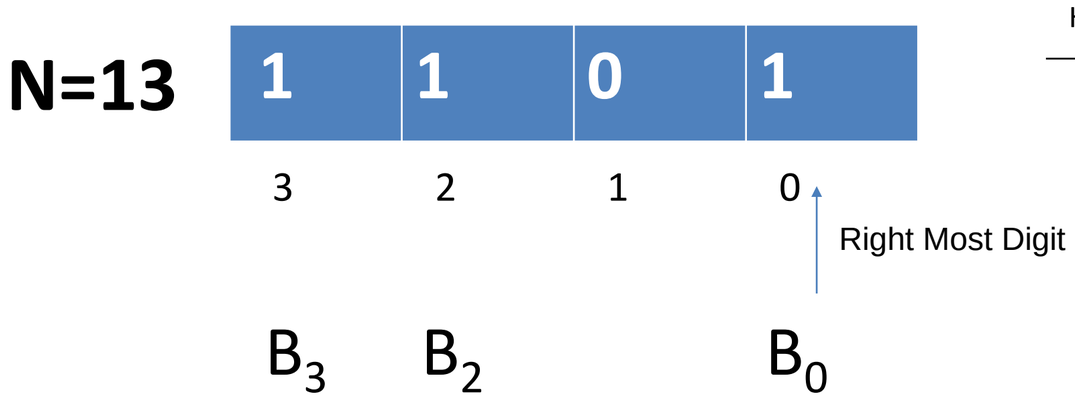


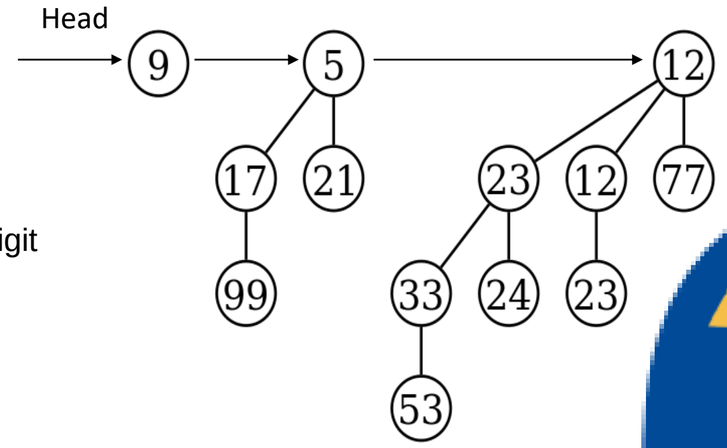
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

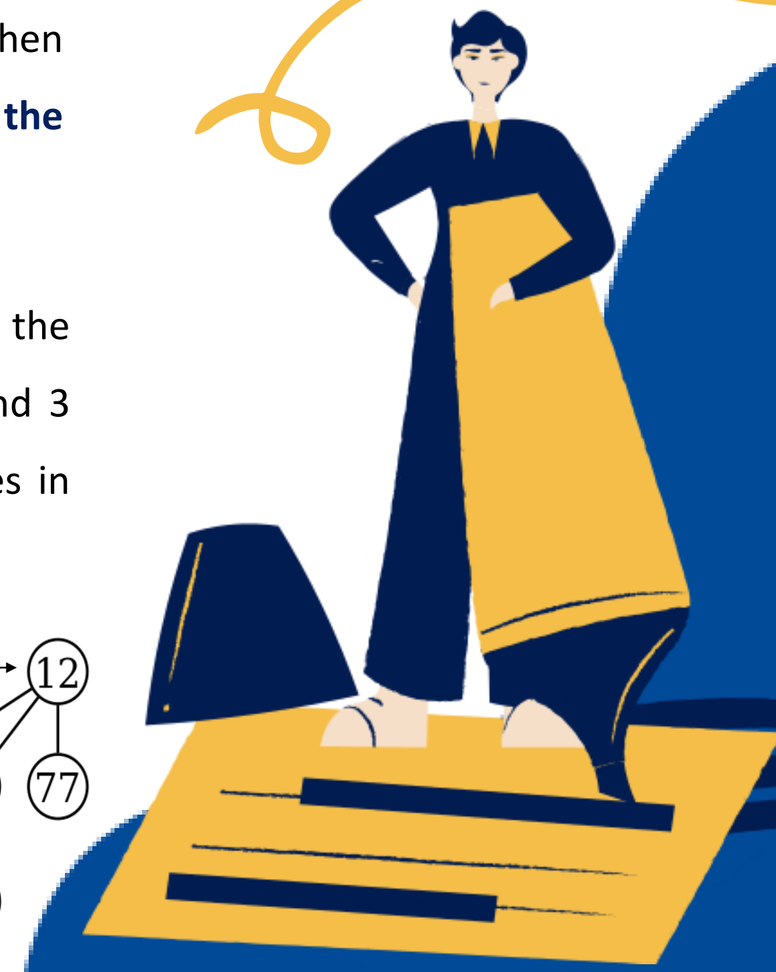
Construction of Binomial Heap

We want to create the **binomial heap of 'n' nodes**, **convert the number into binary form**, and then **start the numbering from right most digit**, and **the 1's position number will represents the binomial tree of that order**.

For example: binomial heap of 13 nodes; the **binary form of 13 is 1101**, so if we start the numbering from the rightmost digit, then we can observe that 1 is available at the 0, 2, and 3 positions; therefore, the binomial heap with 13 nodes will have B_0, B_2, and B_3 binomial trees in increasing order.





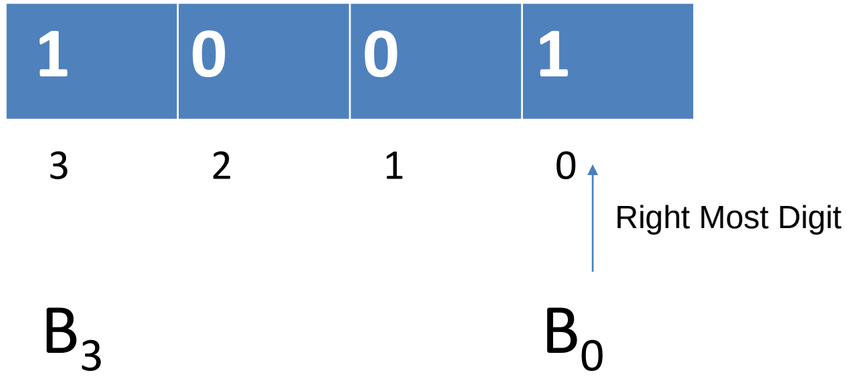


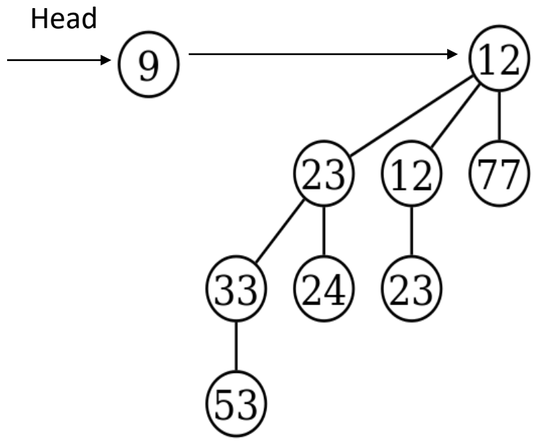
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Construction of Binomial Heap

For example: binomial heap of 9 nodes; the binary form of 9 is 1001, so if we start the numbering from the rightmost digit, then we can observe that 1 is available at the 0 and 3 positions; therefore, the binomial heap with 9 nodes will have B_0 and B_3 binomial trees in increasing order.

N=9



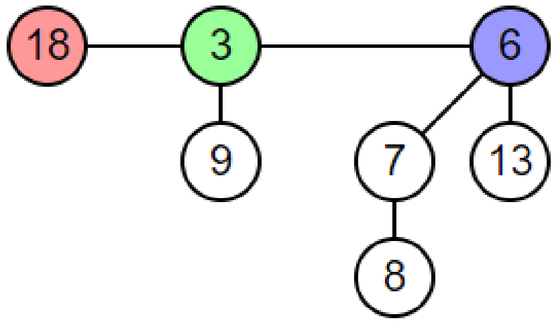


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Operations on Binomial Heap

1. Finding the minimum key

- As we know that binomial heap is a collection of binomial trees and each binomial tree satisfies the property of min heap means root node contains the minimum value.
- We need to compare only root node of all the binomial trees to find the minimum key.



In this case, maximum 3 comparisons is required to find the minimum key.

- There are at-most logn binomial tree for n nodes. The time complexity for finding a minimum key is O(logn).

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Operations on Binomial Heap

2. Union Operation

Given two Binomial Heaps H1 and H2, union(H1, H2) creates a single Binomial Heap.

First Step - The first step is to simply merge the two Heaps in non-decreasing order of degrees. After the simple merge, we need to make sure that there is at most one Binomial Tree of any order (i.e. Binomial heap should not contain same order binomial trees). To do this, we need to combine Binomial Trees of the same order.

Second Step- Define and keep track of three-pointers x, next-x and next-(next-x) or sibling (next-x) and follow the following cases:

Case 1: If degree[x] is not equal to degree[next-x] then move the pointer ahead.

Case 2: **if degree[x] = degree[next-x]**

Case 2.1 if degree[next-(next-x)] is also same (same with case 2) then Move the pointer ahead.

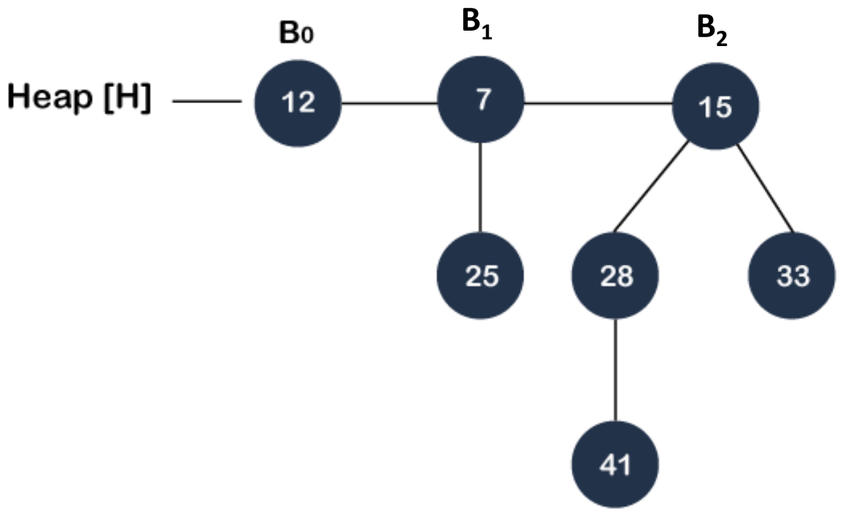
Case 2.2: If degree [next-(next-x)] is not same (not same with case 2) and key[x] is smaller than key[next-x] then remove [next-x] from root and attached to the child of x by linking to it.

Case 2.3 : If degree [next-(next-x)] is not same (not same with case 2) and key[x] is larger than key[next-x] then remove x from the root and attached to the child of next-x.

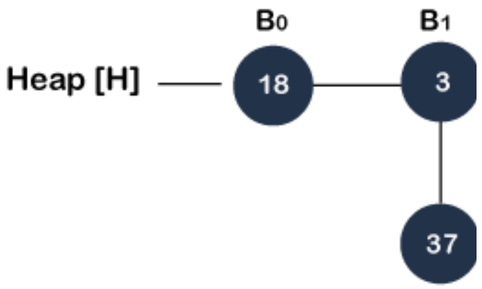
Operations on Binomial Heap

2. Union Operation

Binomial Heap 1



Binomial Heap 2

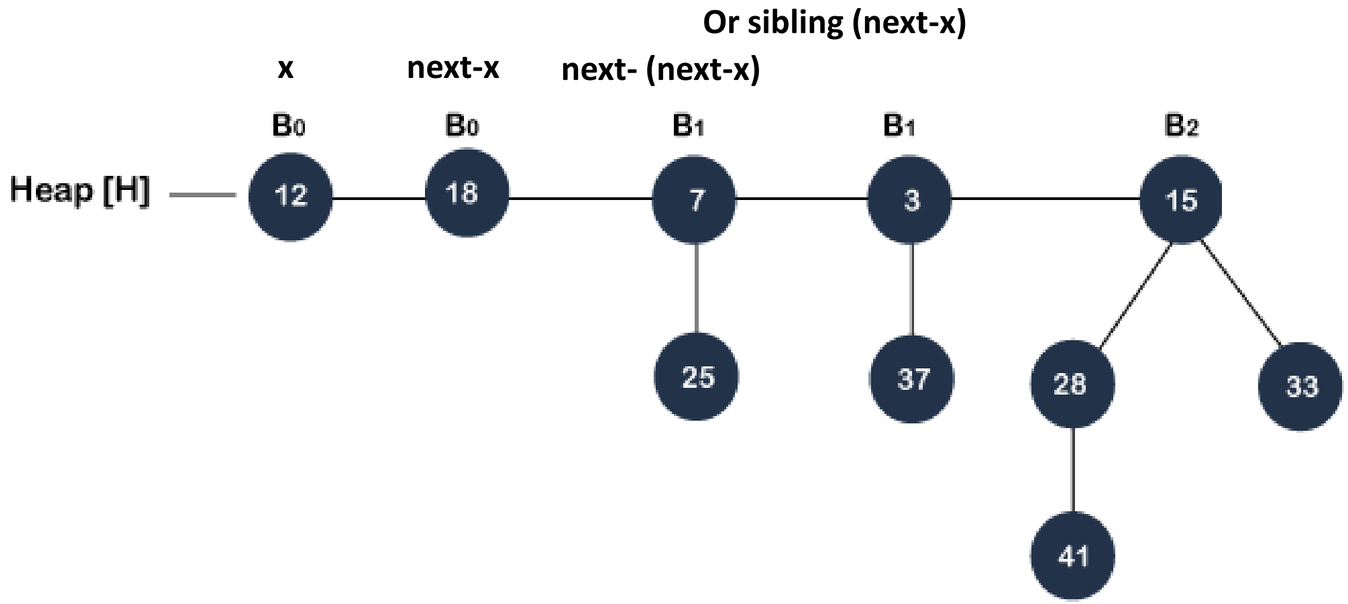


Step 1- The first step is to simply merge the two Heaps in non-decreasing order of degrees

Operations on Binomial Heap

2. Union Operation

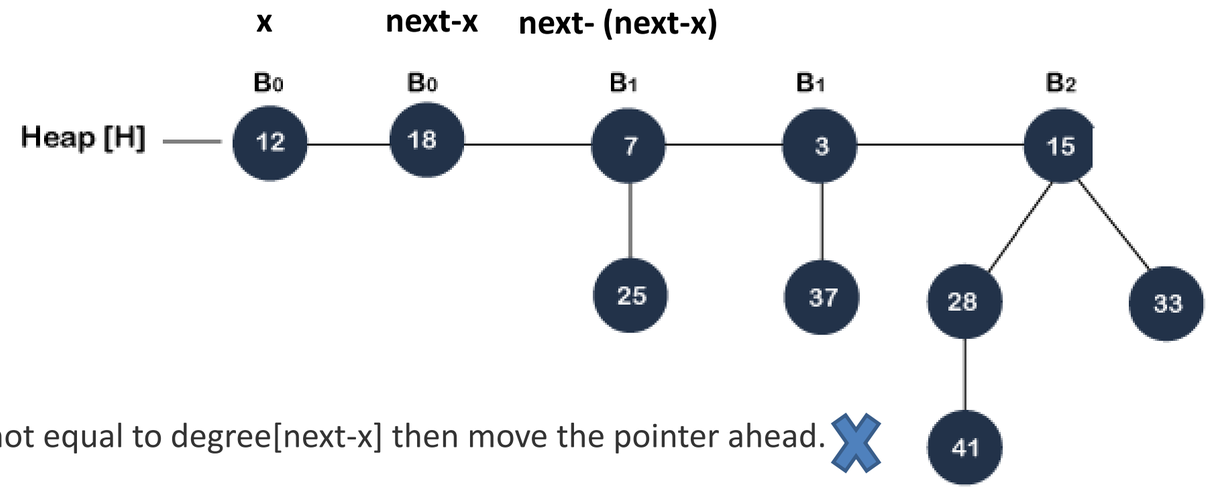
Step 1- The first step is to simply merge the two Heaps in non-decreasing order of degrees



Step 2- Define Initially, x points to the B_0 having value 12, and next-x points to B_0 having value 18. The B_1 is the sibling of B_0 having node value 18. Therefore, the sibling B_1 is represented as sibling[next x] or next- (next-x)

Operations on Binomial Heap

2. Union Operation



Step-3

Case 1: If degree[x] is not equal to degree[next-x] then move the pointer ahead.

Case 2: if degree[x] = degree[next-x]

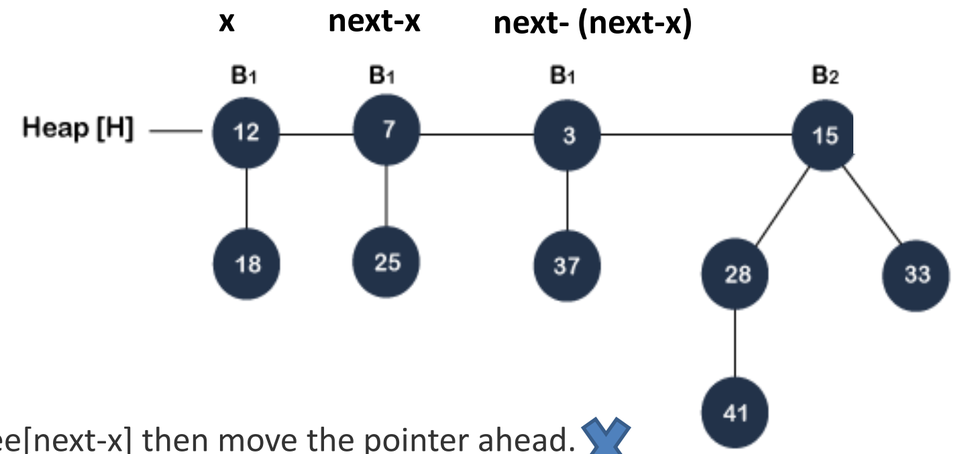
Case 2.1 if degree[next-(next-x)] is also same (same with case 2) then Move the pointer ahead.

Case 2.2: If degree [next-(next-x)] is not same (not same with case 2) and key[x] is smaller than key[next-x] then remove [next-x] from root and attached to the child of x by linking to it.

Case 2.3 : If degree [next-(next-x)] is not same (not same with case 2) and key[x] is larger than key[next-x] then remove x from the root and attached to the child of next-x.

Operations on Binomial Heap

2. Union Operation



Step-3

Case 1: If degree[x] is not equal to degree[next-x] then move the pointer ahead.

Case 2: if degree[x] = degree[next-x]

Case 2.1 if degree[next-(next-x)] is also same (same with case 2) then Move the pointer ahead.

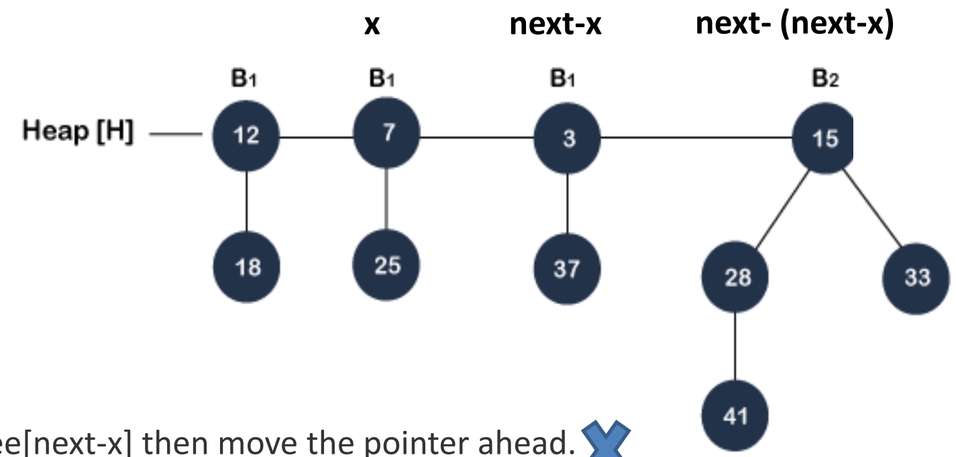
Case 2.2: If degree [next-(next-x)] is not same (not same with case 2) and key[x] is smaller than key[next-x] then remove [next-x] from root and attached to the child of x by linking to it.

Case 2.3 : If degree [next-(next-x)] is not same (not same with case 2) and key[x] is larger than key[next-x] then remove x from the root and attached to the child of next-x.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Operations on Binomial Heap

2. Union Operation



Step-3

Case 1: If degree[x] is not equal to degree[next-x] then move the pointer ahead.

Case 2: if degree[x] = degree[next-x]

Case 2.1 if degree[next-(next-x)] is also same (same with case 2) then Move the pointer ahead.

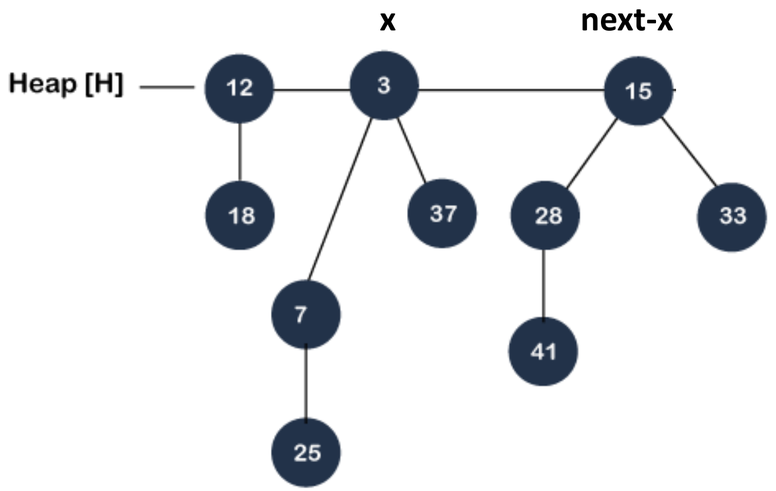
Case 2.2: If degree [next-(next-x)] is not same (not same with case 2) and key[x] is smaller than key[next-x] then remove [next-x] from root and attached to the child of x by linking to it.

Case 2.3 : If degree [next-(next-x)] is not same (not same with case 2) and key[x] is larger than key[next-x] then remove x from the root and attached to the child of next-x.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Operations on Binomial Heap

2. Union Operation



Step-3

Case 1: If degree[x] is not equal to degree[next-x] then move the pointer ahead.

Case 2: if degree[x] = degree[next-x]

Case 2.1 if degree[next-(next-x)] is also same (same with case 2) then Move the pointer ahead.

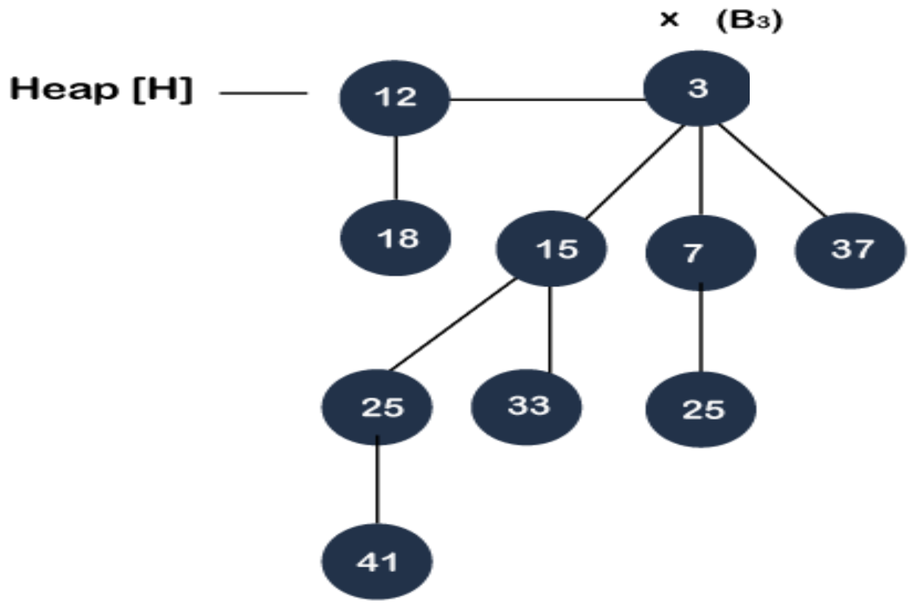
Case 2.2: If degree [next-(next-x)] is not same (not same with case 2) and key[x] is smaller than key[next-x] then remove [next-x] from root and attached to the child of x by linking to it.

Case 2.3 : If degree [next-(next-x)] is not same (not same with case 2) and key[x] is larger than key[next-x] then remove x from the root and attached to the child of next-x.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Operations on Binomial Heap

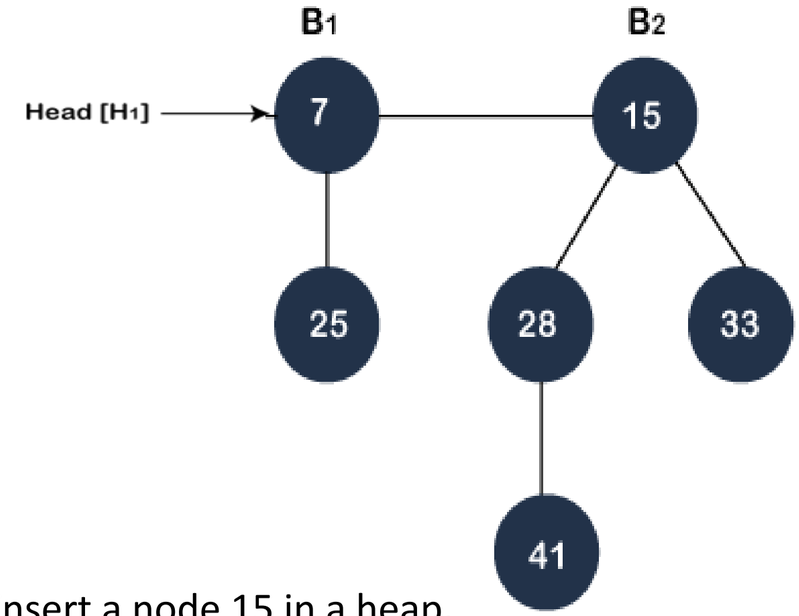
2. Union Operation



- The above tree is the final tree after the union of two binomial heaps.
- It is satisfying the properties of Binomial Heap. (For any non-negative integer k, there should be atmost one binomial tree in a heap where root has degree k.)
- The time complexity for finding a union is O(logn).

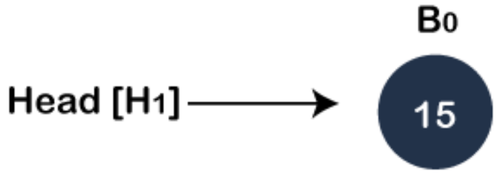
Operations on Binomial Heap

3. Insert Operation



Consider the above heap, and we want to insert a node 15 in a heap.

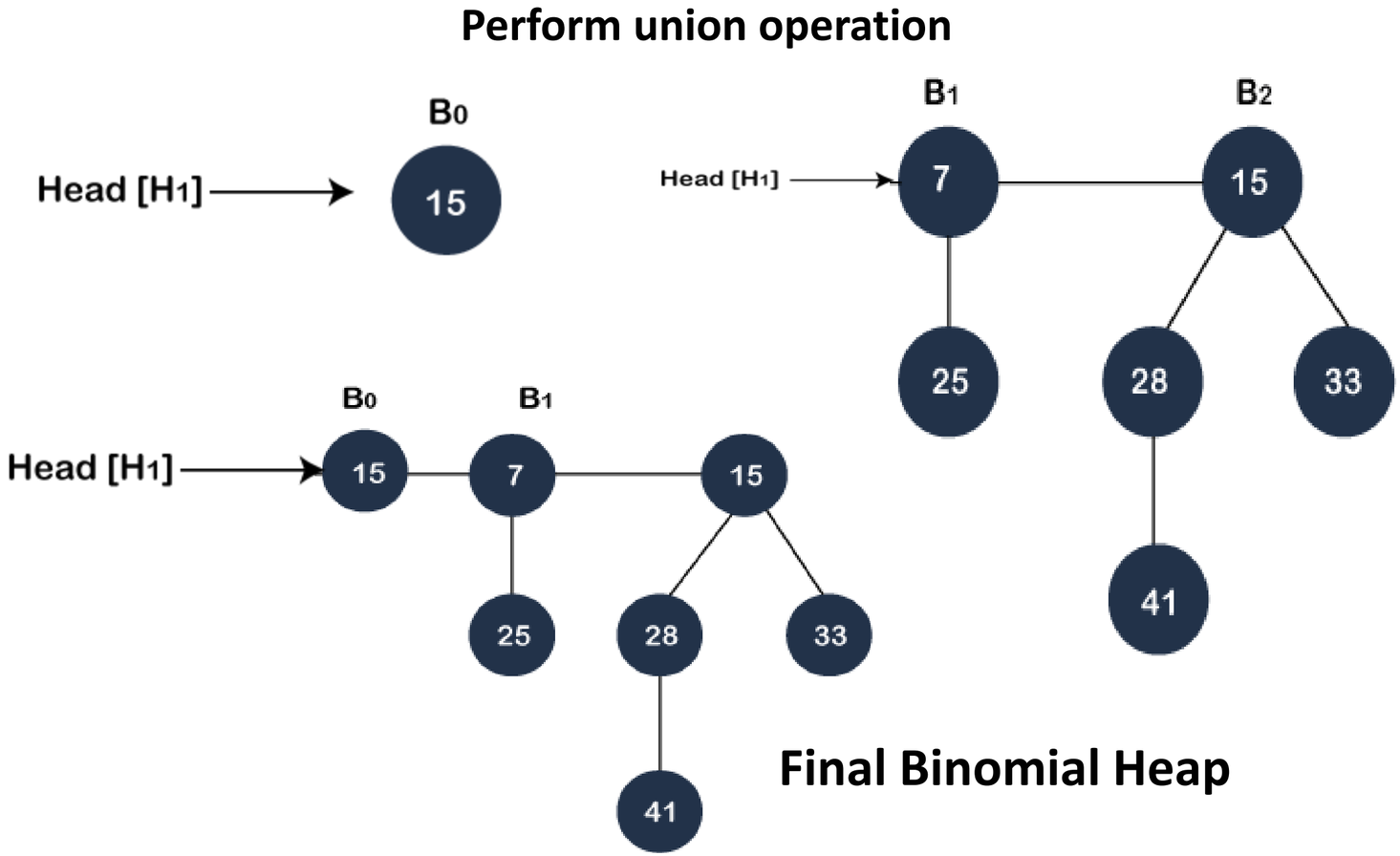
- Create a new binomial heap of 1 node (the node to be inserted) and make union with the other heap.



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Operations on Binomial Heap

3. Insert Operation



If after performing the union operation, final heap satisfies the Binomial heap properties, then this is the final heap other wise balance it using different union case.

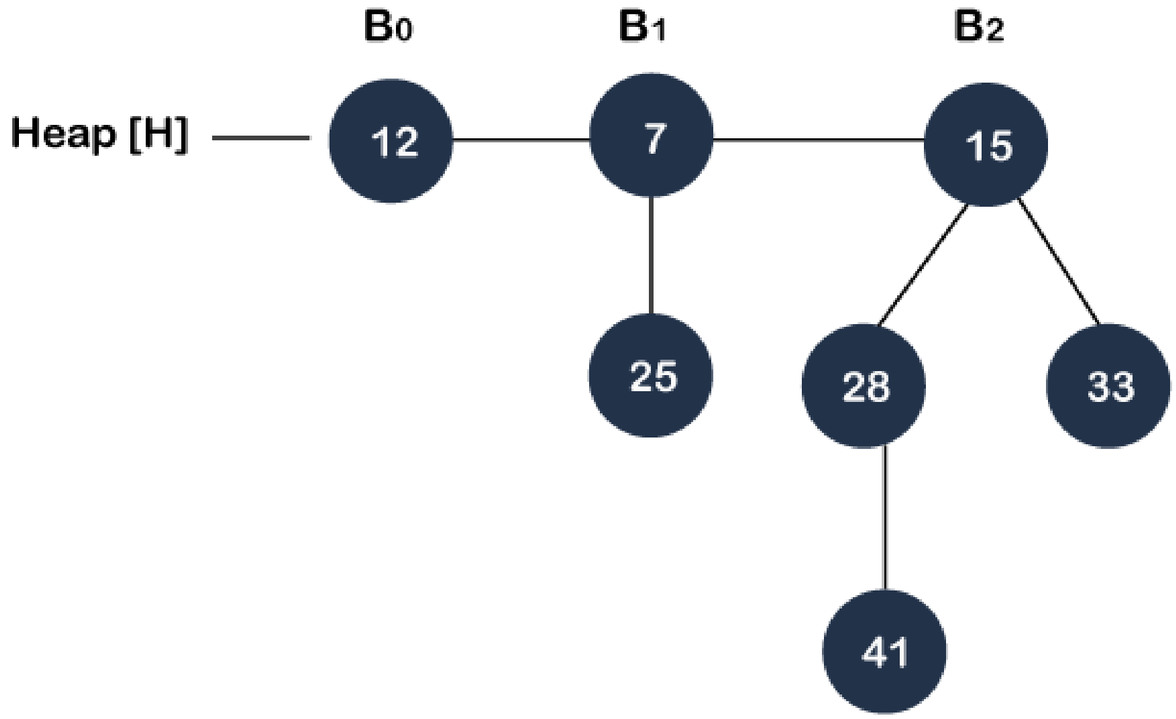
The time complexity for inserting a node is O(logn).

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Operations on Binomial Heap

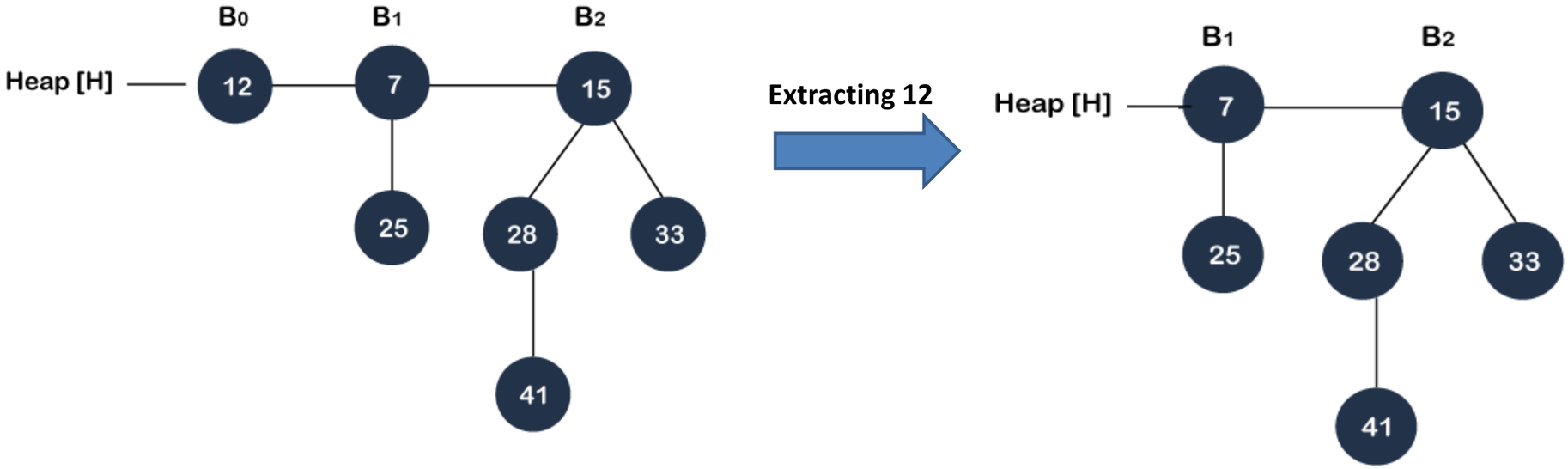
4. Extracting a minimum key

Extracting a minimum key means that we need to remove the element which has minimum key value. We already know that in min heap, the root element contains the minimum key value. Therefore, we have to compare the key value of root node of all the binomial trees.



Operations on Binomial Heap

4. Extracting a minimum key

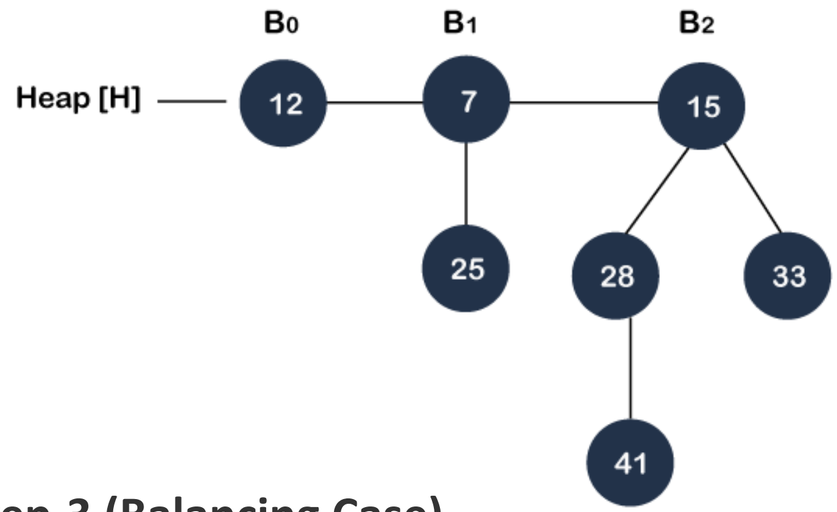


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

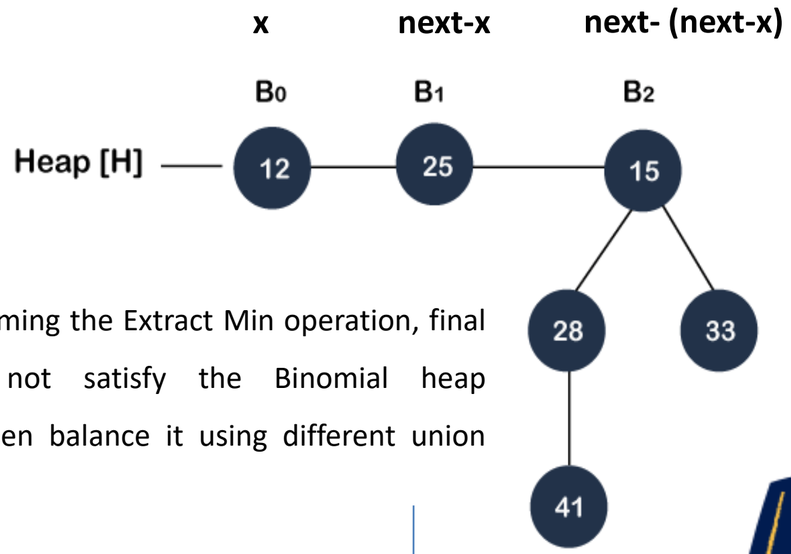
Operations on Binomial Heap

The time complexity for removing a minimum key is O(logn).

4. Extracting a minimum key



Extracting 7



If after performing the Extract Min operation, final heap does not satisfy the Binomial heap properties, then balance it using different union cases.

Step-3 (Balancing Case)

Case 1: If degree[x] is not equal to degree[next-x] then move the pointer ahead.

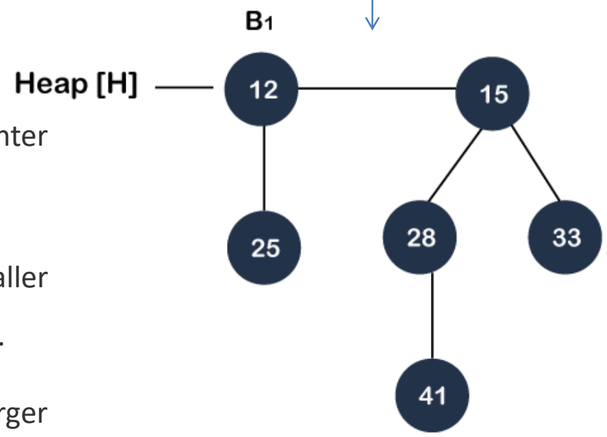
Case 2: if degree[x] = degree[next-x]

Case 2.1 if degree[next-(next-x)] is also same (same with case 2) then Move the pointer ahead.

Case 2.2: If degree [next-(next-x)] is not same (not same with case 2) and key[x] is smaller than key[next-x] then remove [next-x] from root and attached to the child of x by linking to it.

Case 2.3 : If degree [next-(next-x)] is not same (not same with case 2) and key[x] is larger than key[next-x] then remove next-x from the root and attached to the child of x.

Case 2.2 is applied.

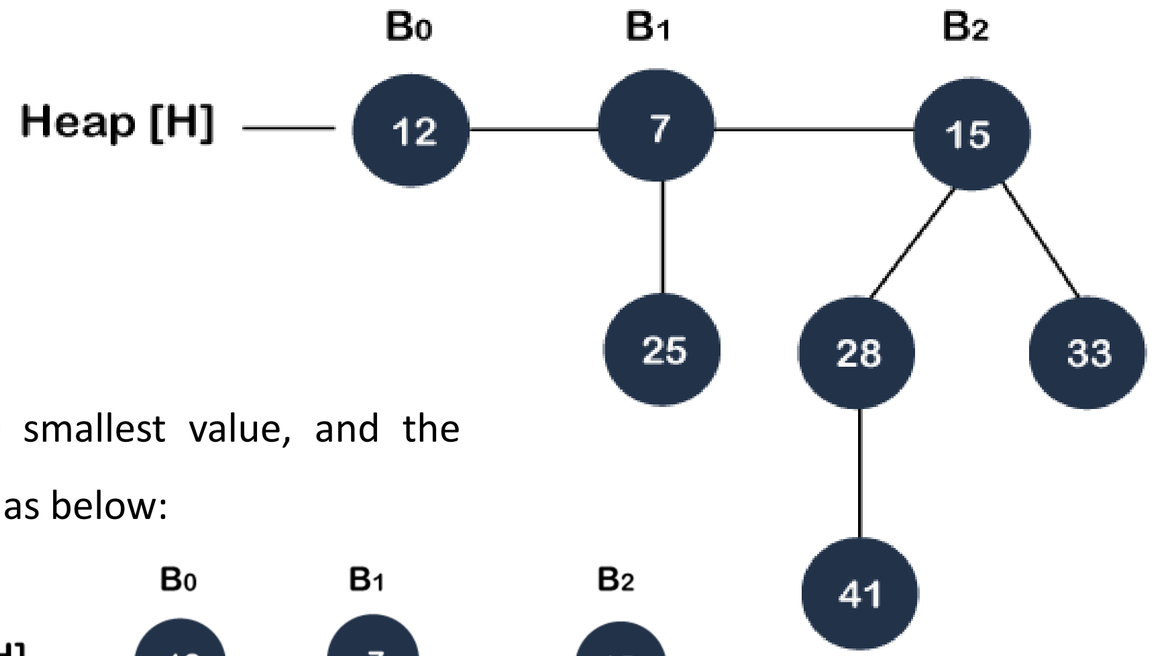


Operations on Binomial Heap

The time complexity for deleting a node is O(logn).

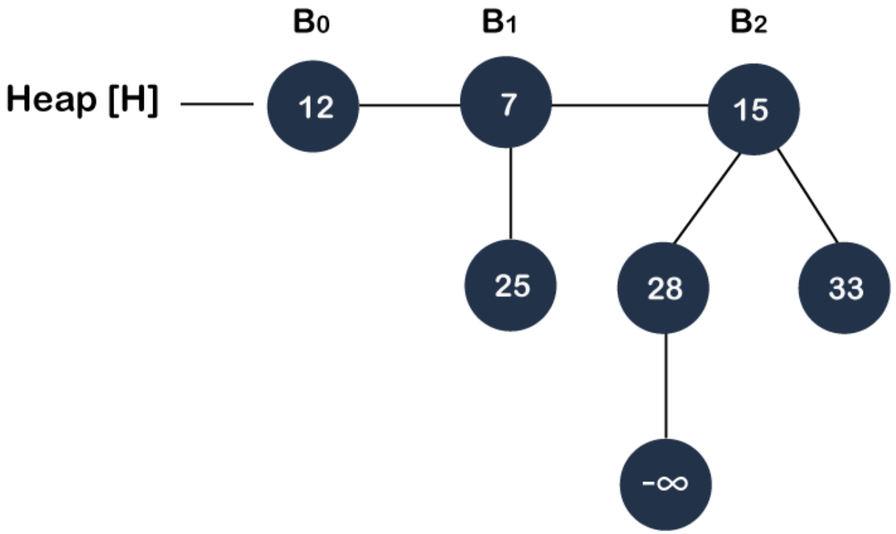
5. Deleting a node

We want to delete node 41



Step- 1

We replace node 41 with the smallest value, and the smallest value is -infinity, shown as below:



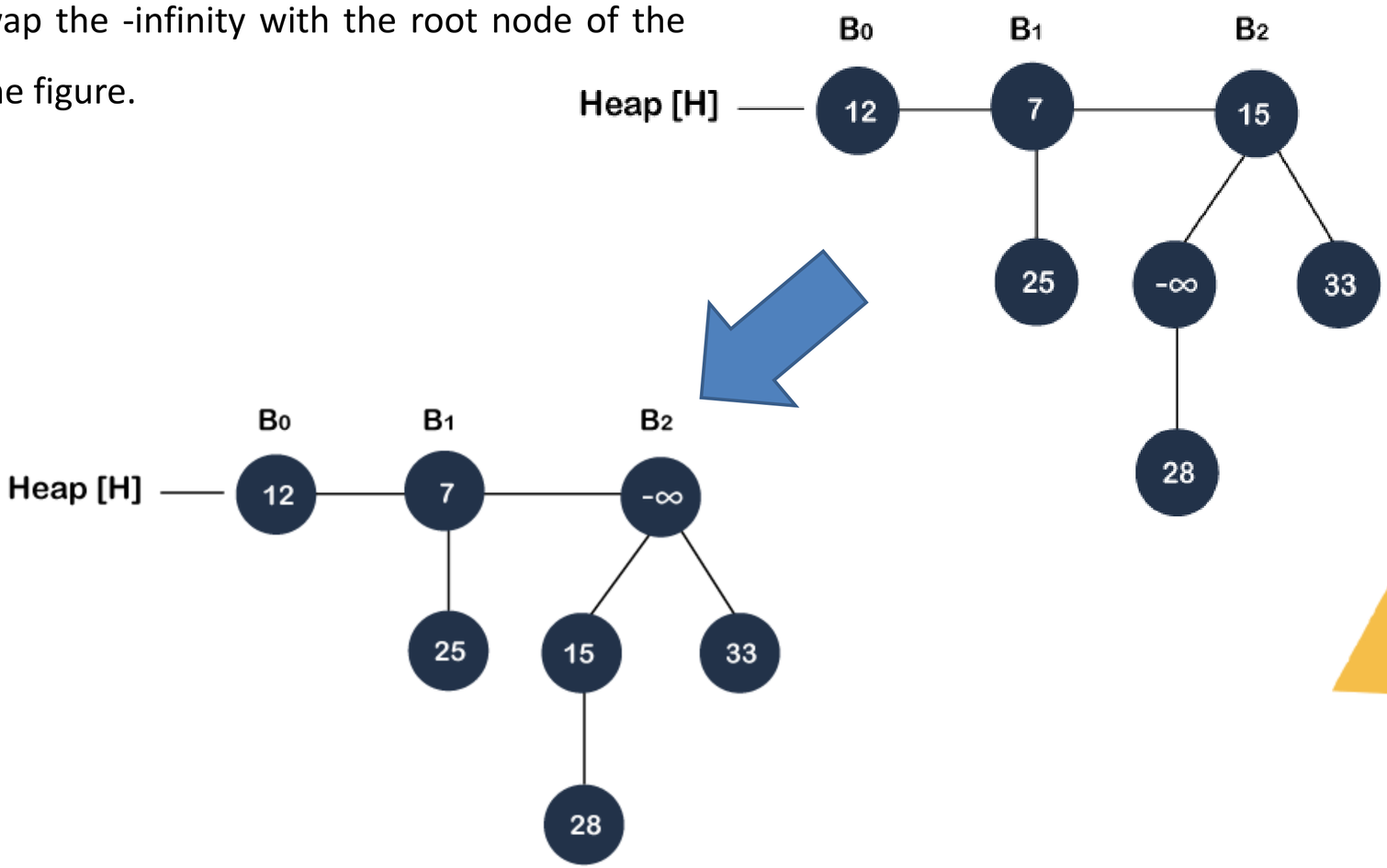
Operations on Binomial Heap

The time complexity for deleting a node is O(logn).

5. Deleting a node

Step- 2

Now we will swap the -infinity with the root node of the tree shown in the figure.



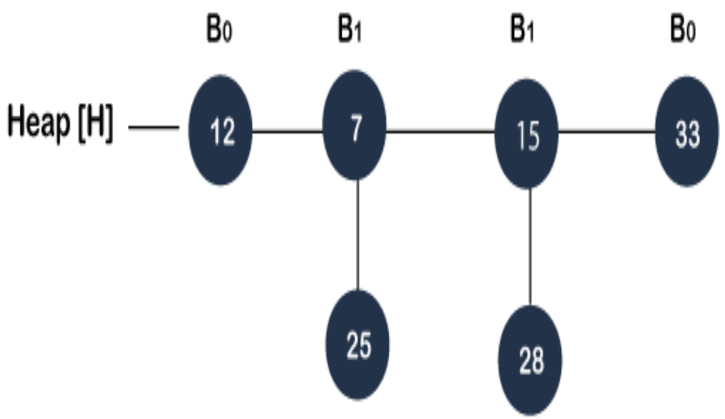
Operations on Binomial Heap

The time complexity for deleting a node is O(logn).

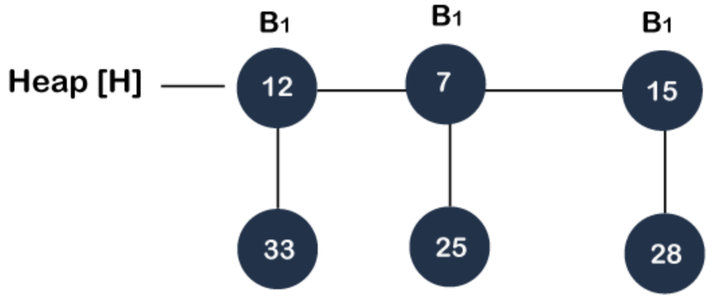
5. Deleting a node

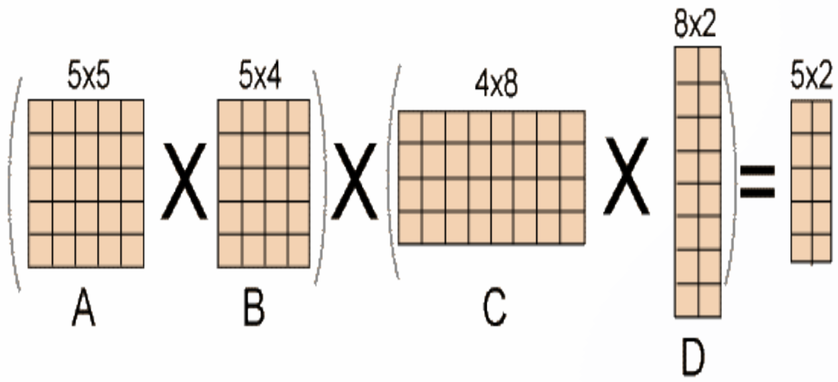
Step- 3

The next step is to extract the minimum key. Since the minimum key in a heap is -infinity so we will extract this key, and the heap would be:



Balance it





03. Matrix Chain Multiplication

Using Dynamic Programming

Matrix Chain Multiplication

Problem Statement

Given a chain $<M_1, M_2...M_n>$ of n two-dimensional matrices. Find the most efficient way **(parenthesize the product or order of product)** to multiply these matrices together $M_1 \times M_2 \times \cdots \times M_n$ that **minimizes the number of multiplications**. Also calculate the total number of multiplications.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding Matrix Chain Multiplication

Due to associative nature of matrix multiplication, we have many ways to multiply a chain of matrices.

In other words, no matter how we parenthesize the product, the result will be the same.

For example, if we had four matrices A, B, C, and D, we would have:

(ABC)D = (AB)(CD) = A(BCD) = ...(the result will be the same in all cases)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding Matrix Chain Multiplication

In order to Multiply two matrices, number of columns in first matrix should be equal to number of rows in second matrix.

For example- There are two matrices A and B with dimensions

A (2 x **3**) and B (**3** x 2).

$$\underset{A}{\begin{bmatrix} 5 & 2 & 3 \\ 2 & 0 & 4 \end{bmatrix}} \underset{B}{\begin{bmatrix} 1 & 7 \\ 2 & 5 \\ 1 & 9 \end{bmatrix}} = \begin{bmatrix} (5*1)+(2*2)+(3*1) & (5*7)+(2*5)+(3*9) \\ (2*1)+(0*2)+(4*1) & (2*7)+(0*5)+(4*9) \end{bmatrix} = \underset{C}{\begin{bmatrix} 12 & 72 \\ 6 & 50 \end{bmatrix}}$$

Total number of multiplication of A (2 x **3**) and B (**3** x 2) to obtain C (2 x2) matrix = **2 x 3 x 2 =12**

In generalized way matrices A (P x Q) and B(Q x R) will result matrix (P x R) which contains P * R elements. To calculate each element need “Q” number of multiplications.

Total multiplications needed are P * Q * R

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding Matrix Chain Multiplication

Let assume there are 3 matrices A (1 x 2), B (2 x 3), C (3 x 2).

We can find the final result in two ways **(AB)C** or **A(BC).** We get same result in any way since **matrix multiplication satisfies associativity property.** It is not commutative i.e. A * (B*C) not equal to A * (C * B)

If we follow first way, i.e. (AB)C way.

To calculate (AB) we need 1*2*3 = 6 multiplications.

Now resultant AB get dimensions 1 x 3 this multiplied with C need 1*3*2 = 6 multiplications.

Total 6+6 = 12 multiplications needed.

If we follow second way, i.e. A(BC) way.

To calculate (BC) we need 2*3 *2 = 12 multiplications.

Now resultant BC get dimensions 2 x 3. A multiplied with this result need 1*2*2 = 4.

Total 12+4 = 16 multiplications needed.

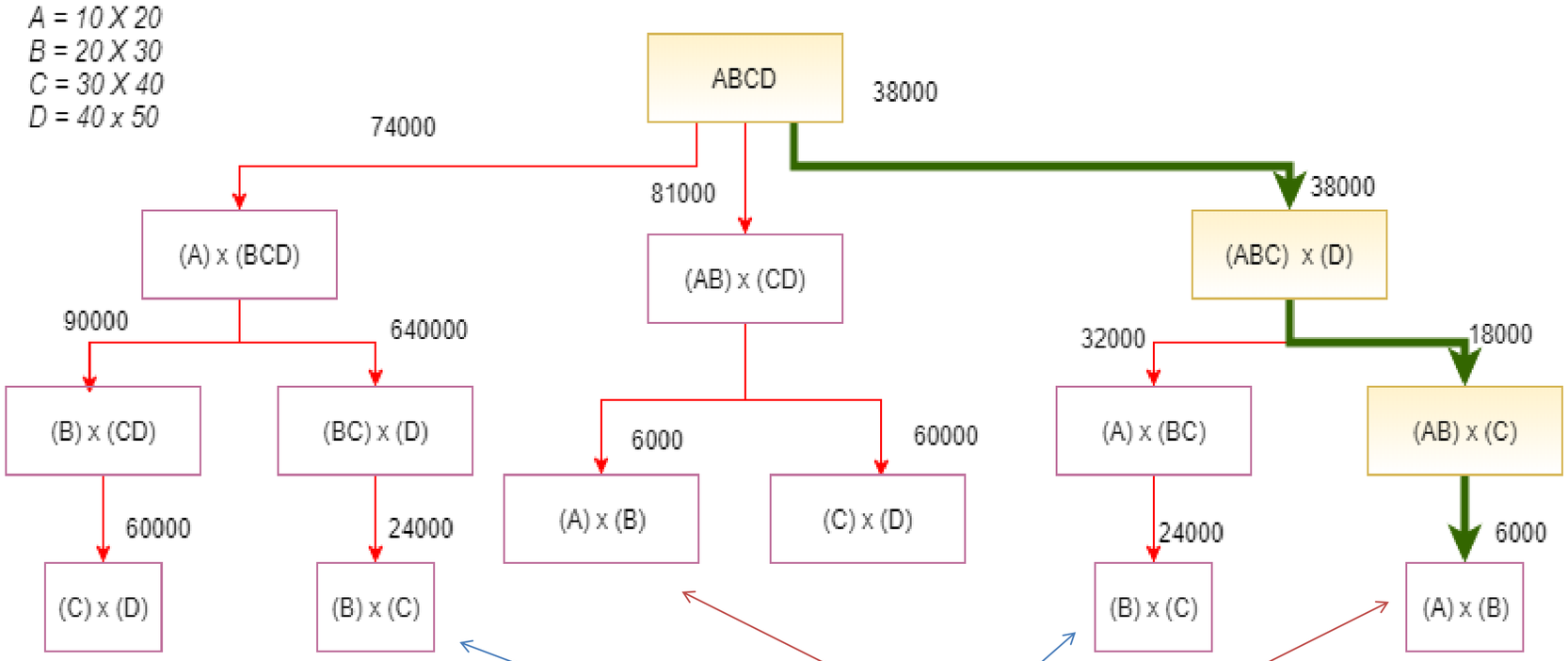
We can observe that based on the way we parenthesize the matrices total number of multiplications are changing.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding Matrix Chain Multiplication

Now, Lets assume there are 4 matrices A, B, C, D of size given below.

We can find the final result in different ways **as shown in the graph.**



Overlapping sub-problems

Will use Dynamic programming to solve the overlapping sub problems

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

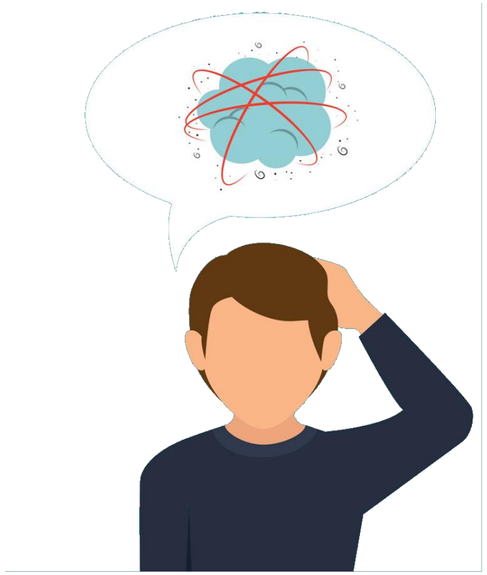
Matrix Chain Multiplication using Dynamic Programming

Let we have “n” number of matrices A1, A2, A3 …… An and

dimensions are d0 x d1, d1 x d2, d2 x d3 …… d_{n-1} x d_n i.e Dimension of Matrix $\mathbf{A_i}$ **is** $\mathbf{d_{i-1}}$ **x** $\mathbf{d_i}$

Solving a chain of matrix that,

$A_i\ A_{i+1}\ A_{i+2}\ A_{i+3}$ ……. $A_j = (A_i\ A_{i+1}\ A_{i+2}\ A_{i+3}$ ……. $A_k)\ (A_{k+1}\ A_{k+2}$ ……. $A_j) + d_{i-1}\ d_k\ d_j$ where **i <= k < j.**



Confused??

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding Matrix Chain Multiplication using Dynamic Programming

Solving a chain of matrix that,

$A_i\ A_{i+1}\ A_{i+2}\ A_{i+3}\ \ldots\ldots\ A_j = (A_i\ A_{i+1}\ A_{i+2}\ A_{i+3}\ \ldots\ldots\ A_k)\ (A_{k+1}\ A_{k+2}\ \ldots\ldots\ A_j) + d_{i-1}\ d_k\ d_j$ where **i <= k < j.**

(dimensions are d0 x d1, d1 x d2, d2 x d3 …… d_{n-1} x d_n)

Example

We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7.

We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.

To store results, in dynamic programming we create a table

1	2	3	4	5	
0					1
	0				2
		0			3
			0		4
				0	5

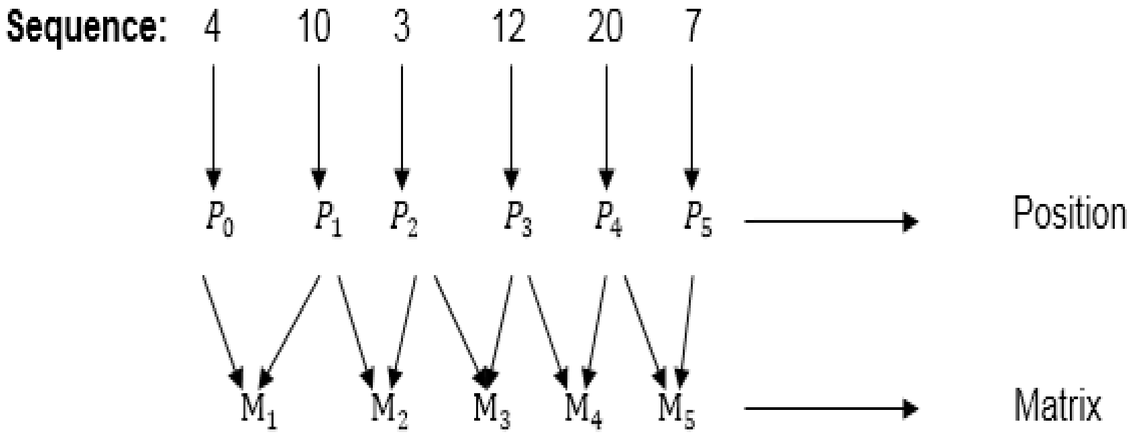
We initialize the diagonal element with equal i,j value with '0'. (If there is only one matrix no need to multiply with any other. So 0 (zero) multiplications required.)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding Matrix Chain Multiplication using Dynamic Programming

We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7.

We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.



M (row x column)

M_1 (4 x 10) → M_1 (P_0 x P_1)

M_2 (10 x 3) → M_2 (P_1 x P_2)

M_3 (3 x 12) → M_3 (P_2 x P_3)

M_4 (12 x 20) → M_4 (P_3 x P_4)

M_5 (20 x 7) → M_5 (P_4 x P_5)

For M_i ⟶ p [i] as column

p [i-1] as row

1	2	3	4	5	
0					1
	0				2
		0			3
			0		4
				0	5

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding Matrix Chain Multiplication using Dynamic Programming

We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7.

We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.

Step-1 We have to sort out all the combination, First Calculate Product of 2 matrices

1. m (1,2) = m_1 x m_2 = = 4 x 10 x 3 = 120
2. m (2, 3) = m_2 x m_3 = = 10 x 3 x 12 = 360
3. m (3, 4) = m_3 x m_4 = 3 x 12 x 20 = 720
4. m (4,5) = m_4 x m_5 = 12 x 20 x 7 = 1680

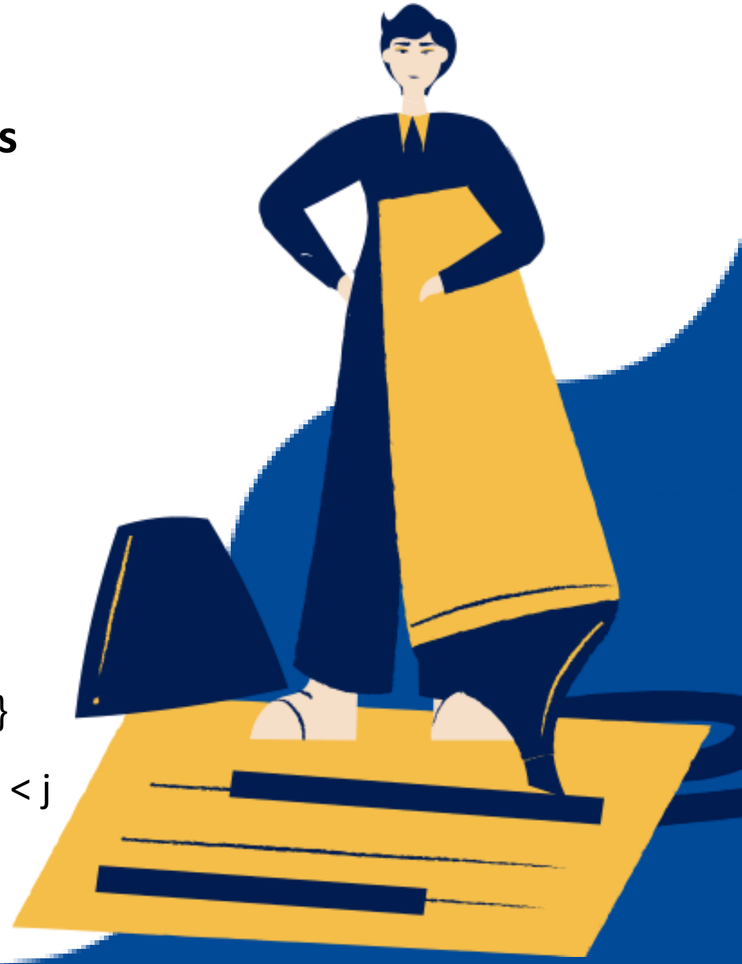
M (row x column)
M_1 (4 x 10) → M_1 (P_0 x P_1)
M_2 (10 x 3) → M_2 (P_1 x P_2)
M_3 (3 x 12) → M_3 (P_2 x P_3)
M_4 (12 x 20) → M_4 (P_3 x P_4)
M_5 (20 x 7) → M_5 (P_4 x P_5)

1	2	3	4	5	
0	120				1
	0	360			2
		0	720		3
			0	1680	4
				0	5

We can also use this Formula

$m[i , j] = \min \{ m[i , k] + m[i+k , j] + p_i\text{-}1*p_k*p_j \}$

if i < j…. where p is dimension of matrix , i ≤ k < j



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding Matrix Chain Multiplication using Dynamic Programming

Step-2 Calculate Product of 3 matrices

M (row x column)
M_1 (4 x 10) → M_1 (P_0 x P_1)
M_2 (10 x 3) → M_2 (P_1 x P_2)
M_3 (3 x 12) → M_3 (P_2 x P_3)
M_4 (12 x 20) → M_4 (P_3 x P_4)
M_5 (20 x 7) → M_5 (P_4 x P_5)

M [1, 3] = M_1 M_2 M_3

There are two cases by which we can solve this multiplication:

Case 1: (M_1 x M_2) + M_3

Case 2: M_1+ (M_2x M_3)

After solving both cases we choose the case in which minimum output is there.

$$M[1,3] = \min \begin{cases} M[1,2] + M[3,3] + P_0P_2P_3 = 120 + 0 + 4 \times 3 \times 12 = 264 \\ M[1,1] + M[2,3] + P_0P_1P_3 = 0 + 360 + 4 \times 10 \times 12 = 840 \end{cases} \rightarrow \textbf{M [1, 3] =264}$$

$$M[2,4] = \min \begin{cases} M[2,3] + M[4,4] + P_1P_3P_4 = 360 + 0 + 10 \times 12 \times 20 = 2760 \\ M[2,2] + M[3,4] + P_1P_2P_4 = 0 + 720 + 10 \times 3 \times 20 = 1320 \end{cases} \rightarrow \textbf{M [2, 4] =1320}$$

M [3, 5] = M_3 M_4 M_5 → 1140

1	2	3	4	5	
0	120				1
	0	360			2
		0	720		3
			0	1680	4
				0	5

→

1	2	3	4	5	
0	120	264			1
	0	360	1320		2
		0	720	1140	3
			0	1680	4
				0	5

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding Matrix Chain Multiplication using Dynamic Programming

Step-3 Now Calculate Product of 4 matrices

M [1, 4] = M_1 M_2 M_3 M_4

There are three cases by which we can solve this multiplication:

Case 1: (M_1 x M_2 x M_3) + M_4

Case 2: M_1 x (M_2x M_3 x M_4)

Case 3: (M_1 x M_2) x (M_3 x M_4)

After solving both cases we choose the case in which minimum output is there.

M (row x column)
M_1 (4 x 10) → M_1 (P_0 x P_1)
M_2 (10 x 3) → M_2 (P_1 x P_2)
M_3 (3 x 12) → M_3 (P_2 x P_3)
M_4 (12 x 20) → M_4 (P_3 x P_4)
M_5 (20 x 7) → M_5 (P_4 x P_5)

$$M[1,4] = \min \begin{cases} M[1,3] + M[4,4] + P_0P_3P_4 = 264 + 0 + 4 \times 12 \times 20 = 1224 \\ M[1,1] + M[2,4] + P_0P_1P_4 = 0 + 1320 + 4 \times 10 \times 20 = 2120 \\ M[1,2] + M[3,4] + P_0P_2P_4 = 120 + 720 + 4 \times 3 \times 20 = 1080 \end{cases}$$

→ **M [1, 4] =1080**

M [2, 5] = M_2 M_3 M_4 M_5 → 1350

	1	2	3	4	5
1	0	120	264		
2		0	360	1320	
3			0	720	1140
4				0	1680
5					0

→

	1	2	3	4	5
1	0	120	264	1080	
2		0	360	1320	1350
3			0	720	1140
4				0	1680
5					0

Understanding Matrix Chain Multiplication using Dynamic Programming

Step-4 Now Calculate Product of 5 matrices

$M[1, 5] = M_1 M_2 M_3 M_4 M_5$

There are four cases by which we can solve this multiplication:

Case 1: $(M_1 \times M_2 \times M_3 \times M_4) \times M_5$ Case 2: $(M_1 \times M_2 \times M_3) \times (M_4 \times M_5)$

Case 3: $(M_1 \times M_2) \times (M_3 \times M_4 \times M_5)$ Case 4: $(M_1 \times (M_2 \times M_3 \times M_4 \times M_5)$

After solving both cases we choose the case in which minimum output is there.

M (row x column)
M_1 (4 x 10) → M_1 (P_0 x P_1)
M_2 (10 x 3) → M_2 (P_1 x P_2)
M_3 (3 x 12) → M_3 (P_2 x P_3)
M_4 (12 x 20) → M_4 (P_3 x P_4)
M_5 (20 x 7) → M_5 (P_4 x P_5)

$$M[1,5] = \min \begin{cases} M[1,4] + M[5,5] + P_0P_4P_5 = 1080 + 0 + 4 \times 20 \times 7 = 1544 \\ M[1,3] + M[4,5] + P_0P_3P_5 = 264 + 1680 + 4 \times 12 \times 7 = 2016 \\ M[1,2] + M[3,5] + P_0P_2P_5 = 120 + 1140 + 4 \times 3 \times 7 = 1344 \\ M[1,1] + M[2,5] + P_0P_1P_5 = 0 + 1350 + 4 \times 10 \times 7 = 1630 \end{cases}$$

→ **M [1, 5] =1344**

1	2	3	4	5	
0	120	264	1080		1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

→ **Final Ouput**

1	2	3	4	5	
0	120	264	1080	1344	1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

Happy Learning!

If you have any doubts, or queries , can be discussed in the C-11, Room 310, AB-1.

or share it on WhatsApp 8586968801

if there is any suggestion or feedback about slides, please write it on gaurav.kumar@gla.ac.in

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU