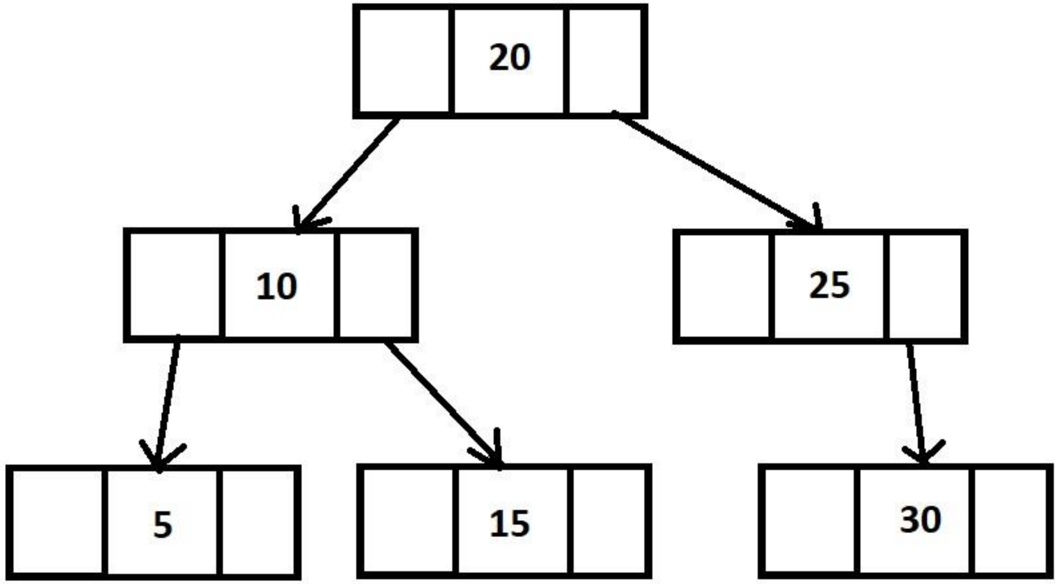
Compression of Trees & it's Space Optimization using intersection of lists

Date – 1/06/2021
Made in India

Suppose there is a Binary Search tree

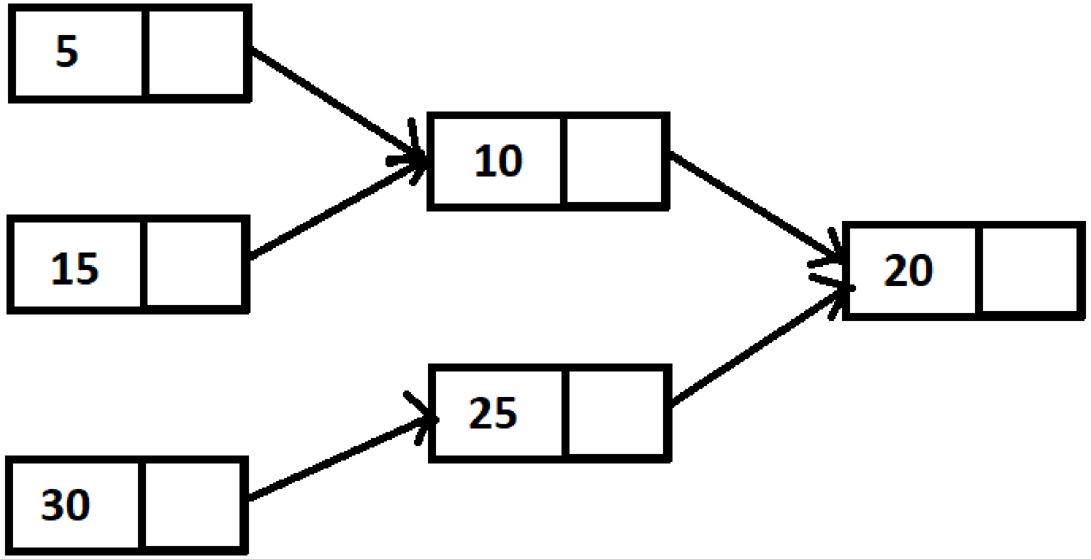


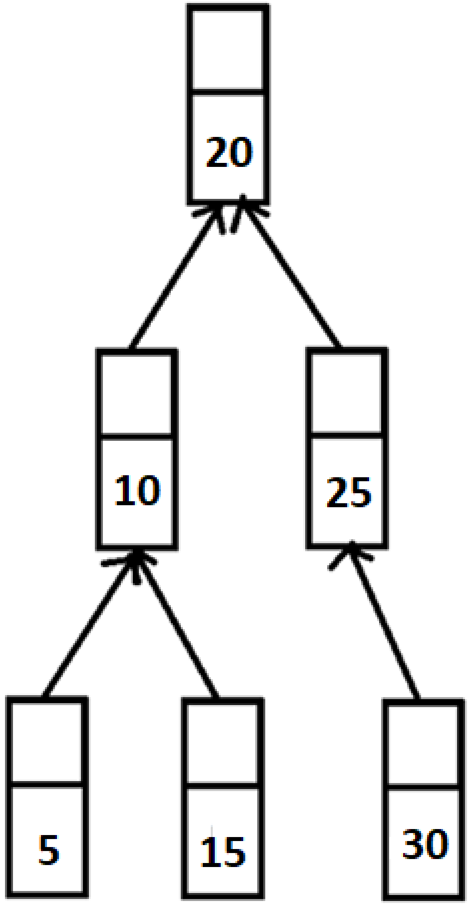
Space compression using linked list

Now we will try to remove the extra space for both **left** and **right** & replace it with **Next** field.

```
struct node
{
  struct node *left;
  int data;
  struct node *right;
};

struct node
{
  int data;
  struct node *next;
};
```

Intersection of Linked list(same as tree)

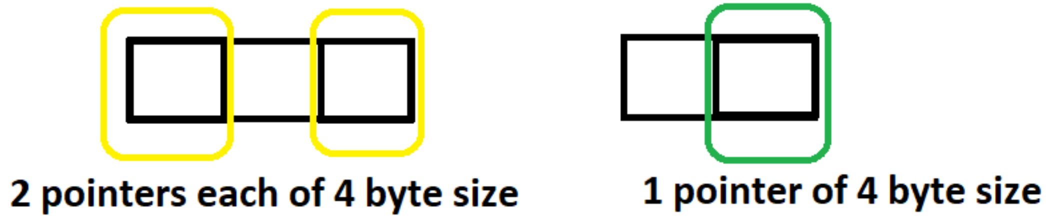




Merge point of 2 list is the parent of both the left and right node

Date – 1/06/2021

How compression will occur – part 1

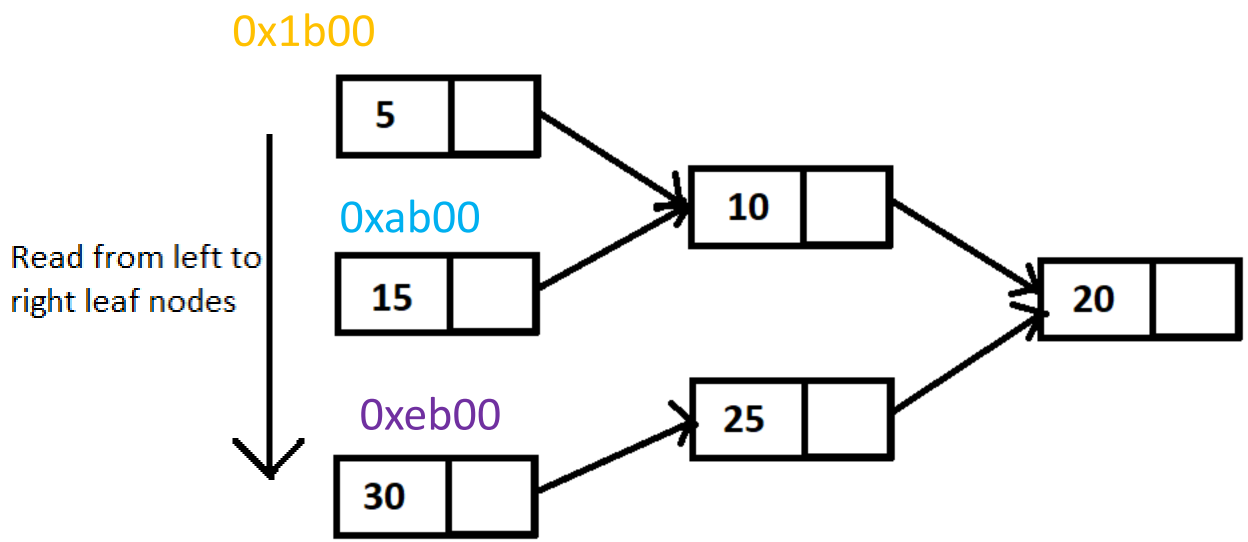



```
struct node
{
  struct node *left;
  int data;
  struct node *right;
};

struct node
{
  int data;
  struct node *next;
};
```

How compression will occur – part 2

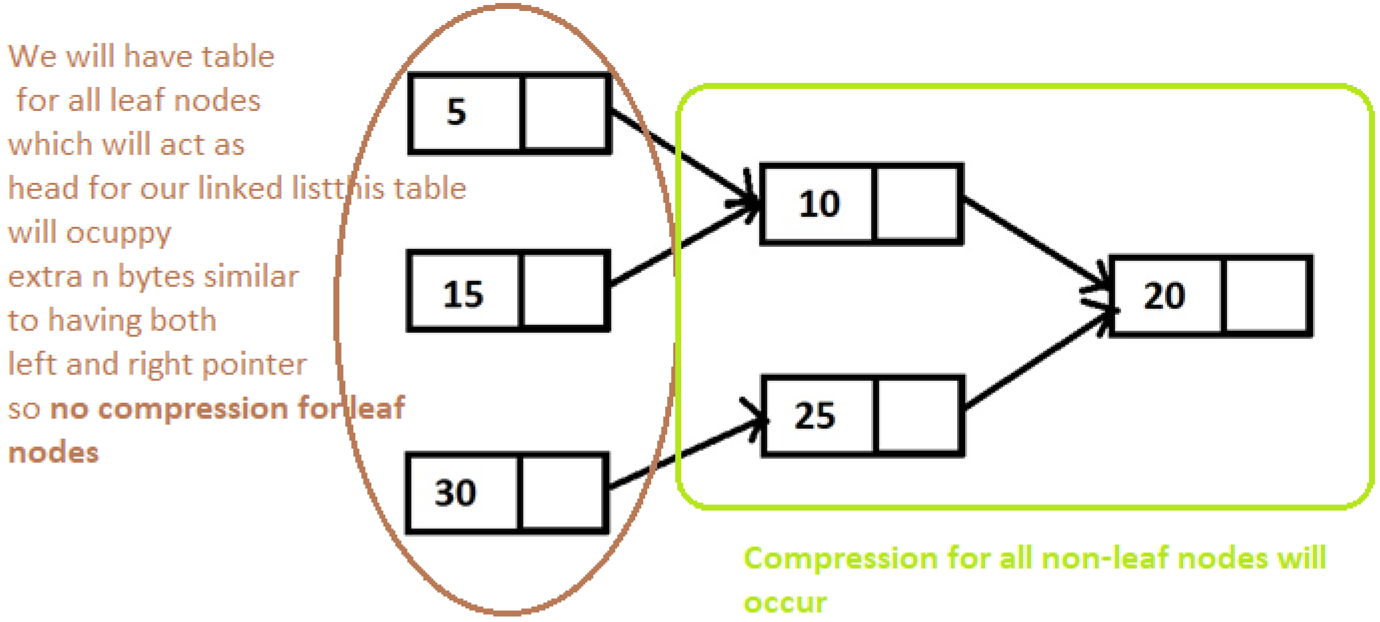
Read from left to right leaf nodes

Index	Leafnode_ptr
1	0x1b00
2	0xab00
3	0xeb00



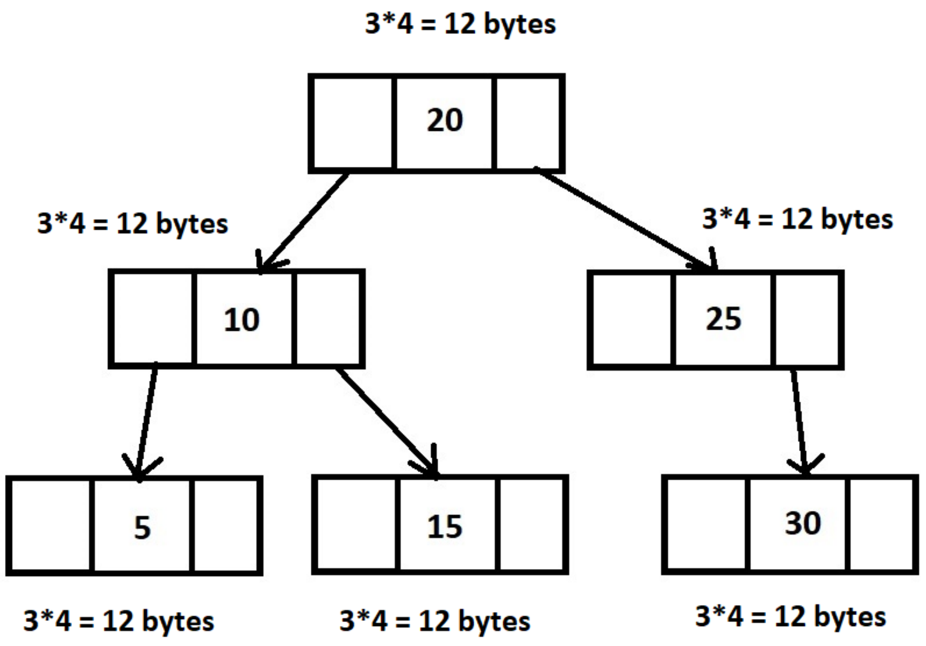
The leafnode_ptr will be read from the Leafnode table to traverse from child to parent linearly.
If any 2 leafnode_ptr's node->next is same then node->next is the parent for both left and right node

Compression will occur for non-leaf nodes

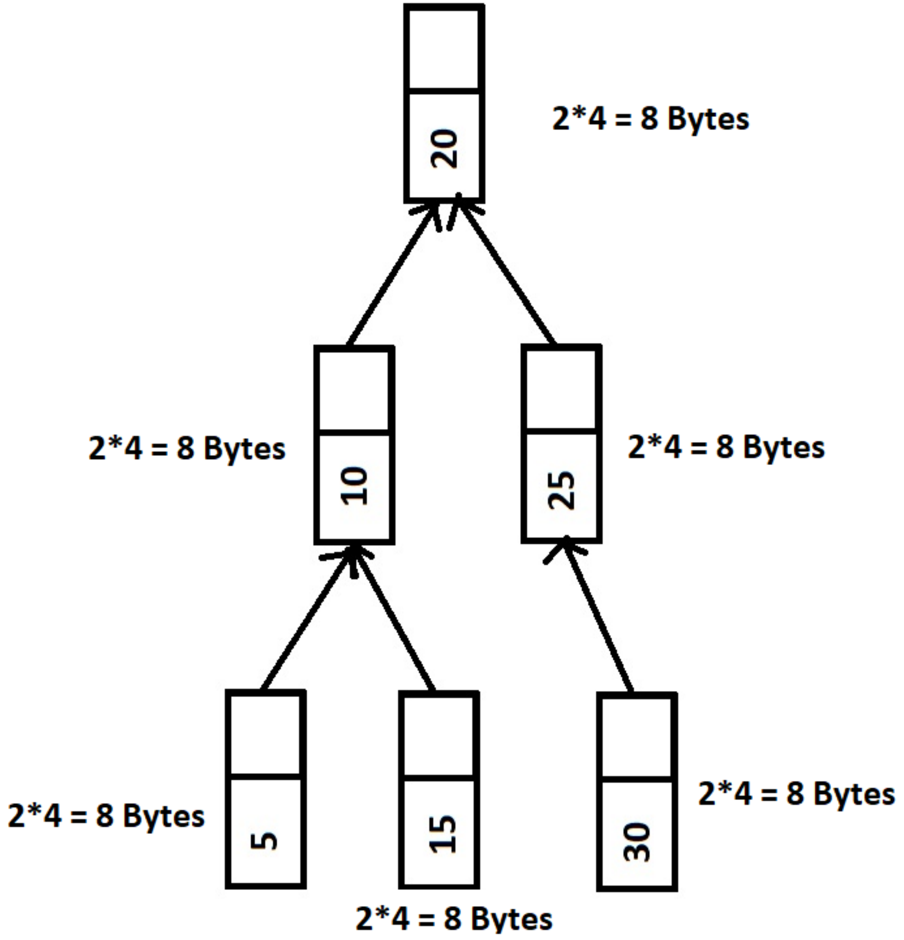


We can save 3*4 = 12 bytes for nodes (10,20,25)

Size difference calculations



Total Size = 12*6 = 72 bytes



Total Size = 8*6 = 48 bytes

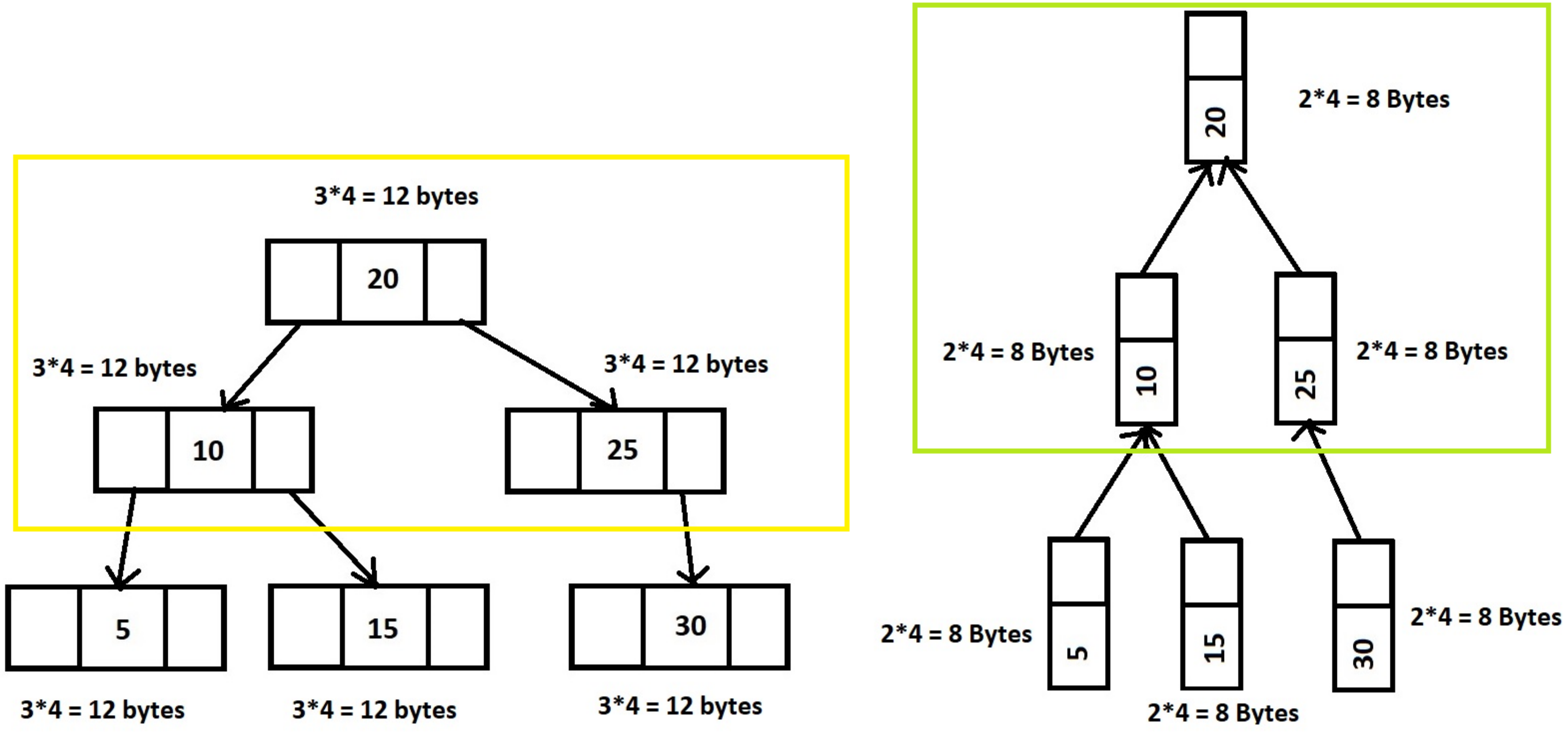
Index	Leafnode_ptr
1	0x1b00
2	0xab00
3	0xeb00

3*4 bytes = 12 bytes (indexing is not required, it's for mere presentation here)

Total Memory Saved = 72 - 48 - 12(for table) = 12 bytes

Copyright@ Daipayan Bhowal
Date – 1/06/2021

Formula for compression



Total saved memory = sizeof ptr * num of non leaf nodes * number of pointer in a node = 4*3*1 = 12 bytes

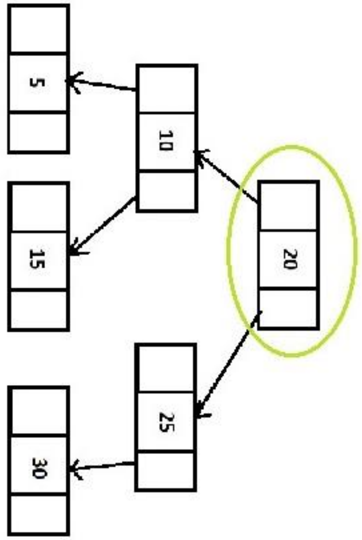
Copyright@ Daipayan Bhowal
Date – 1/06/2021

Compression algorithm

```
void compdp_trav(struct node* ptr, struct node* prev,struct node_c* previousnode)
{
  struct node* left = ptr->left;
  struct node* right = ptr->right;

 /* Creating a duplicate node with different linked list structure node_c by daipayan*/
  struct node_c* new_node = (struct node_c*) malloc(sizeof(struct node_c));
  new_node->data = ptr->data;
  /* For first node set next as NULL by daipayan */
  if(prev = NULL)
    new_node->next = NULL;
  else
    new_node->next = previousnode;

  /* If its a leafnode then store it in a table, increment the static counter and terminate the algorithm by daipayan*/
  if(ptr->left == NULL && ptr->right == NULL)
  {
    leafnode_ptr[i] = new_node;
    i++;
    return;
  }
  /* Traverse both left and right by daipayan */
  if(ptr->left)
  compdp_trav(left,ptr,new_node);

  if(ptr->right)
  compdp_trav(right,ptr,new_node);

}
```

Copyright@ Daipayan Bhowal
Date – 1/06/2021

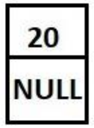
Compression algorithm explanation step by step part 1



Step 1 - We first traverse from top of the BST

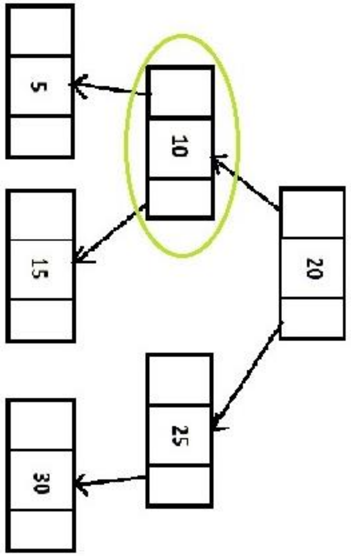
Step 2 - Create a new node with same data, as it is the first node so set next field as **NULL**

Step 3 - Now traverse in preorder way by calling the left node(10) first and then right node(25) recursively before calling both check left and right are **NULL** or not



Copyright@ Daipayan Bhowal
Date – 1/06/2021

Compression algorithm explanation step by step part 2



Step 4,5 & 6 are repitation of Step 1,2 & 3 except in Step 2 , next field is set as parent

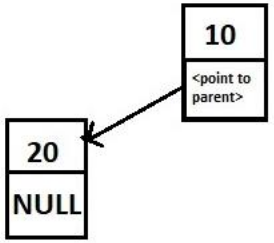
Step 4 - Now we are in left node, prevoiusnode(20) is passed to our function

Step 5 - Create a new node with same data(10) but set the next field as parent or prevoiusnode(20)

Step 6 - Now again do preorder traversal

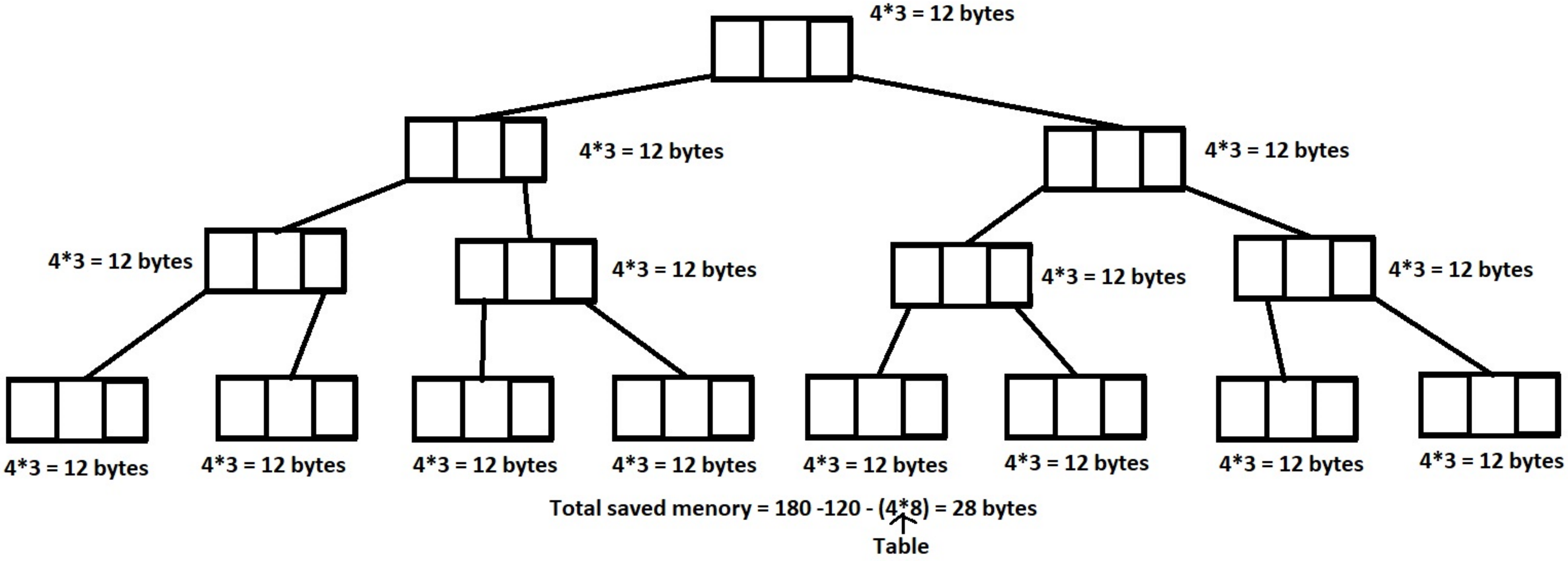
Repeat all the above steps 4,5 & 6 until we reach leafnodes

Step 7 - Now if leafnode is detected then fill the leafnode's ptr in the table(leafnode_ptr[]) and terminate the program.



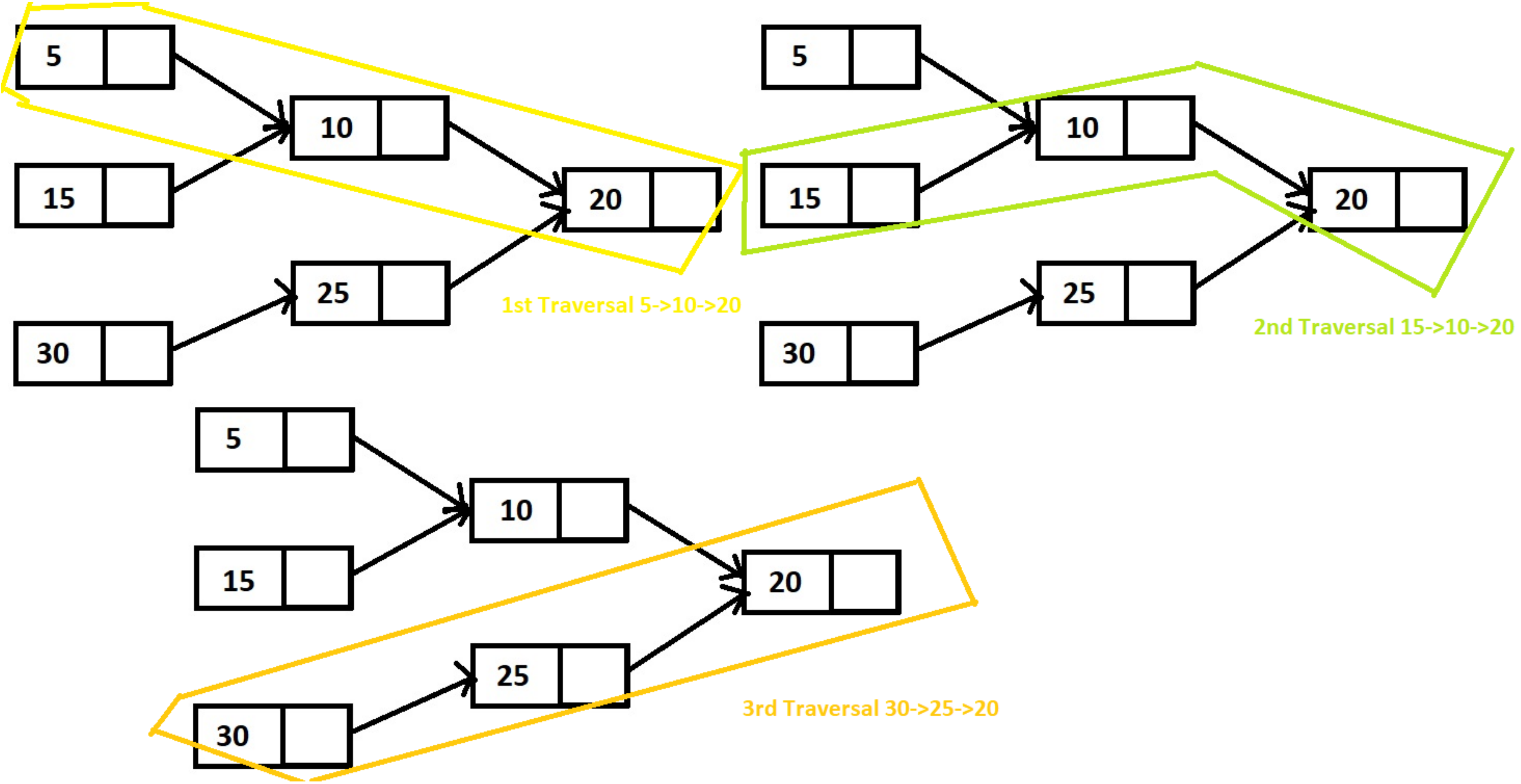
Copyright@ Daipayan Bhowal
Date – 1/06/2021

Now applying compression on larger tree …



Date – 1/06/2021

Traversal of the lists



Traversal algorithm

```
void linklist_dptr(struct node_c** ptr, int count)
{
   int k=0;
  for(k=0; k<count; k++)
  {
   /* fetch from leafnode_ptr  table */
   struct node_c* nd = ptr[k];
   printf("\nlinkedlist traversal!\n");
   /* traverse individual node*/
   while(nd)
   {
      printf("%d\n", nd->data);
      nd = nd->next;
   }
  }
}
```

Copyright@ Daipayan Bhowal
Date – 1/06/2021

How to resolve left and right nodes ?

Index	Leafnode_ptr
1	0x1b00
2	0xab00
3	0xeb00

Leafnode_ptr is an dynamic array here
Higher index will be more right most node, just need to compare array index with left and right node to distinguish the left and right.

Ex:- left->next == right->next then parent = left->next

In case of Binary Search Tree , we can compare the values of the data part , greater value will be on Right side of a parent node and lesser value will be on left side of a parent node.

Copyright@ Daipayan Bhowal
Date – 1/06/2021

More compression technique using tuple

Read from left to right leaf nodes

Index	Leafnode_ptrleft	Leafnode_ptrright
1	0x1b00	0xab00
2	NULL	0xeb00
3		

Copyright@ Daipayan Bhowal
Date – 1/06/2021