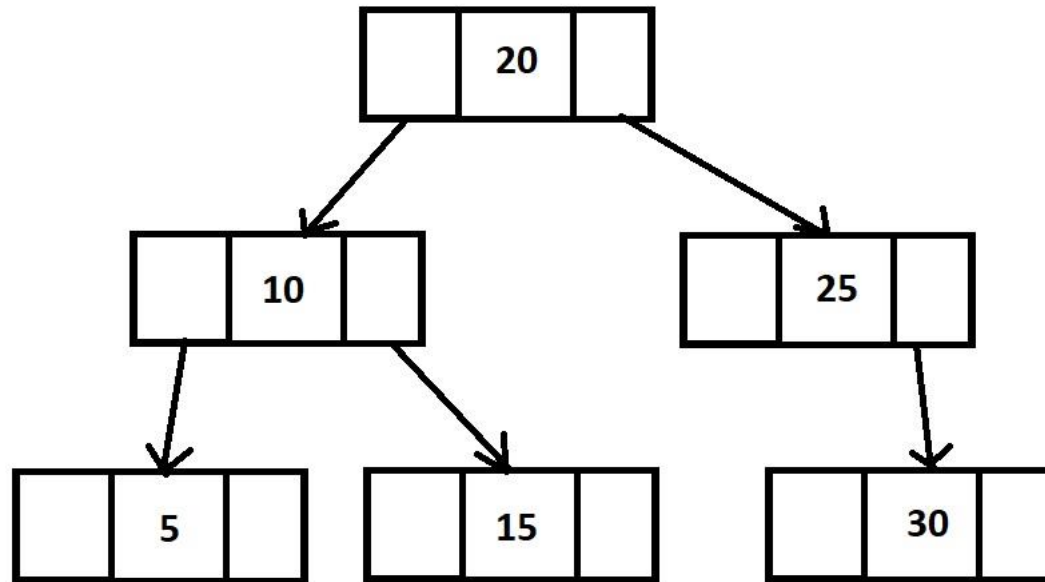


Compression of Trees & it's Space Optimization using intersection of lists

Copyright@ Daipayan Bhowal
Date – 1/06/2021
Made in India

Suppose there is a tree



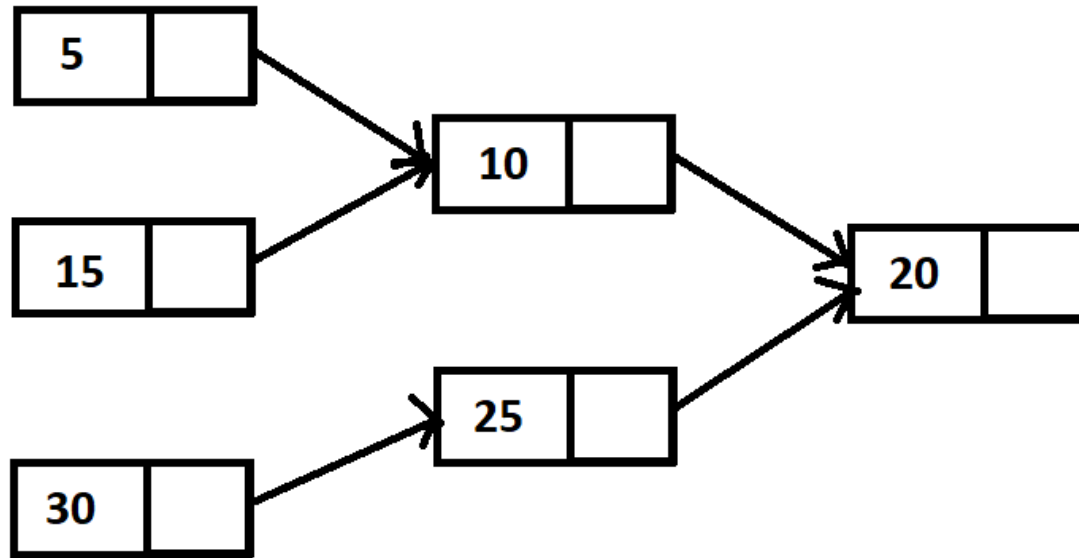
Space compression using linked list

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

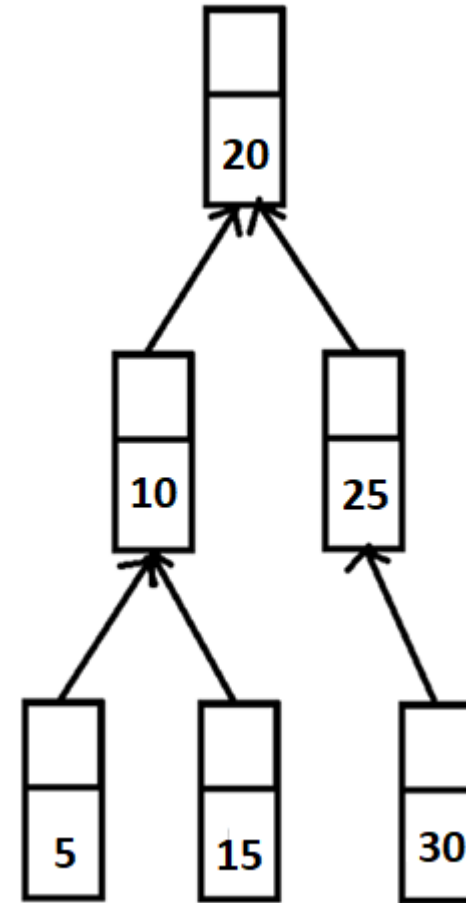
```
struct node
{
    struct node *left;
    int data;
    struct node *right;
};
```

```
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



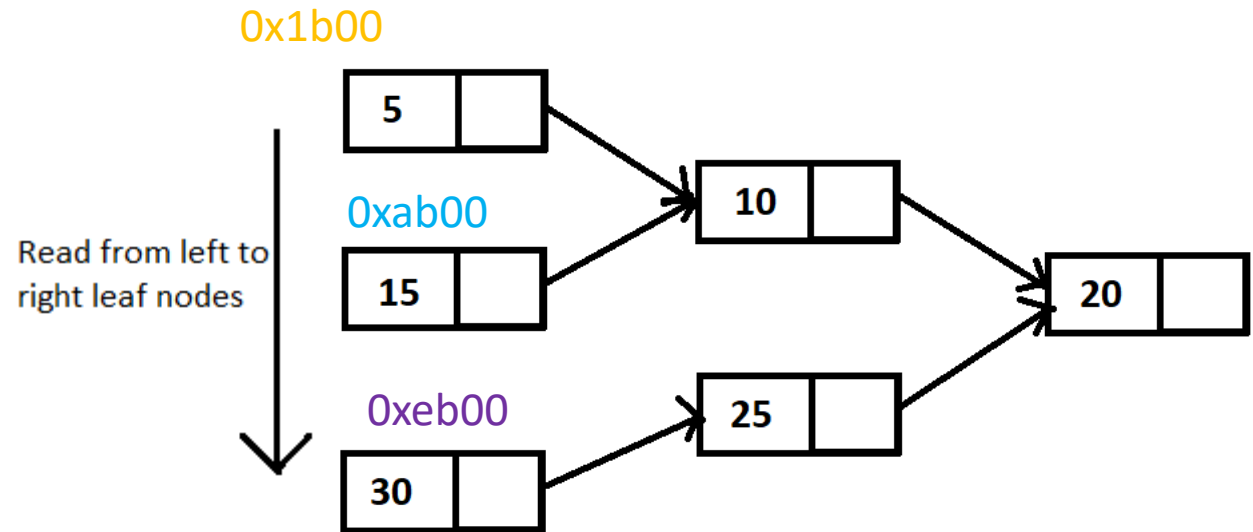
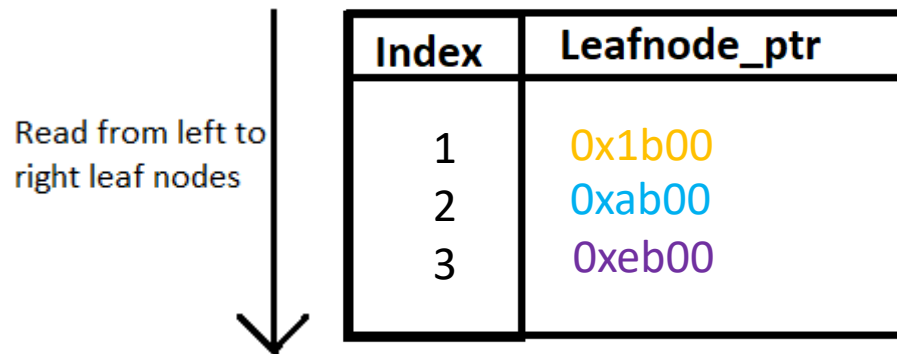
How compression will occur – part 1



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

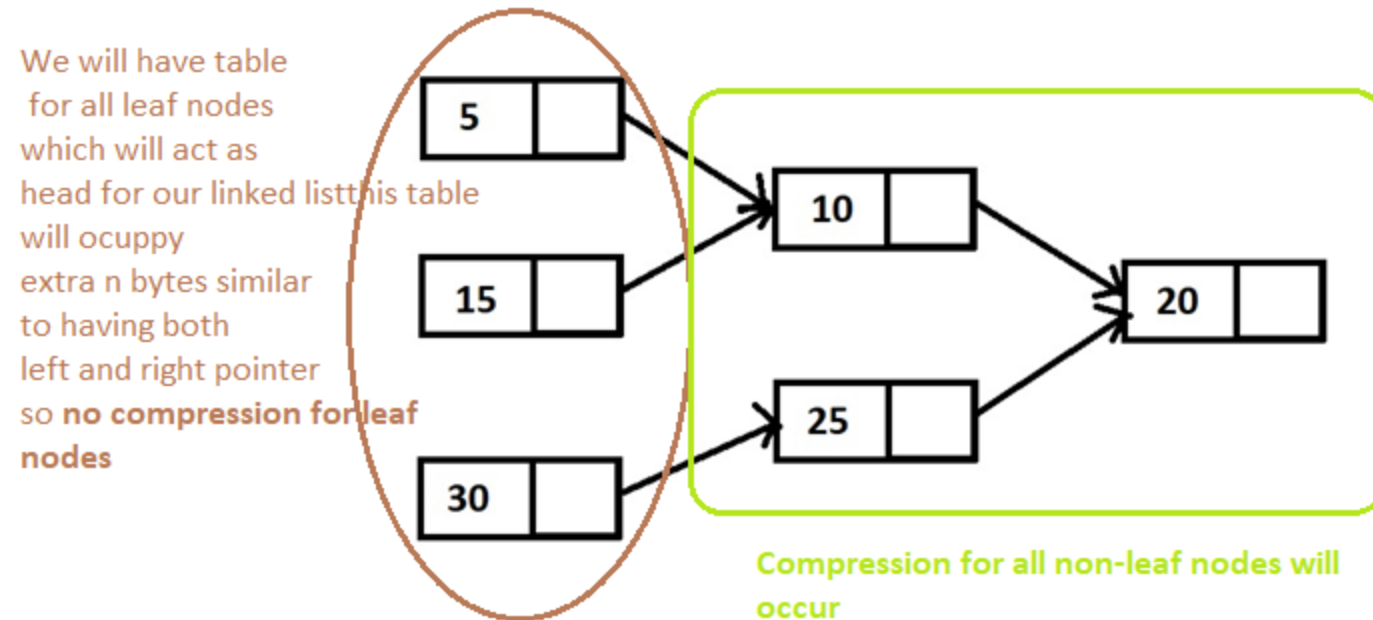
```
struct node
{
    int data;
    struct node *next;
};
```

How compression will occur – part 2



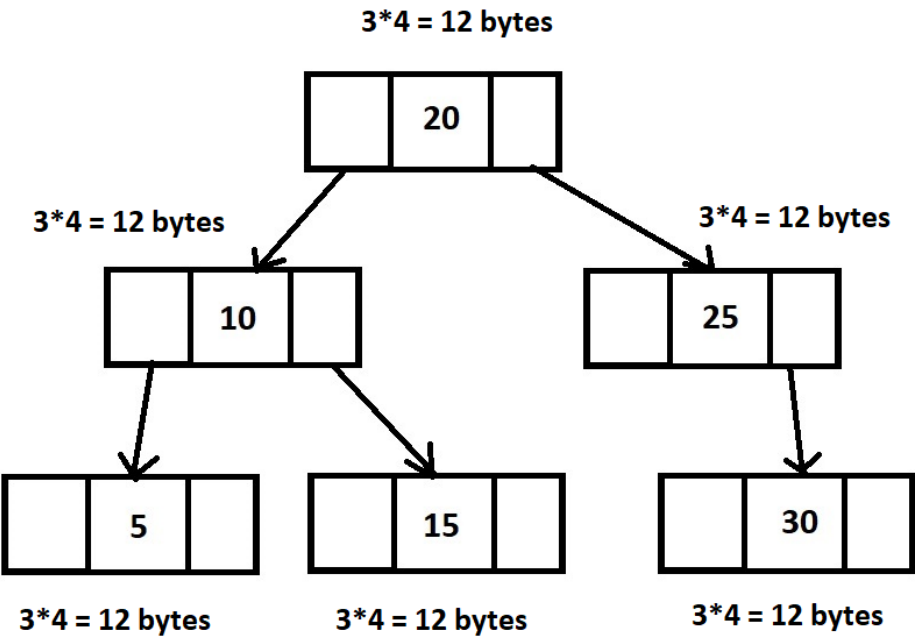
The leafnode_ptr will be read from the Leafnode table to traverse from child to parent linearly.
If any 2 leafnode_ptr's node->next is same then node->next is the parent for both left and right node

Compression will occur for non-leaf nodes



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

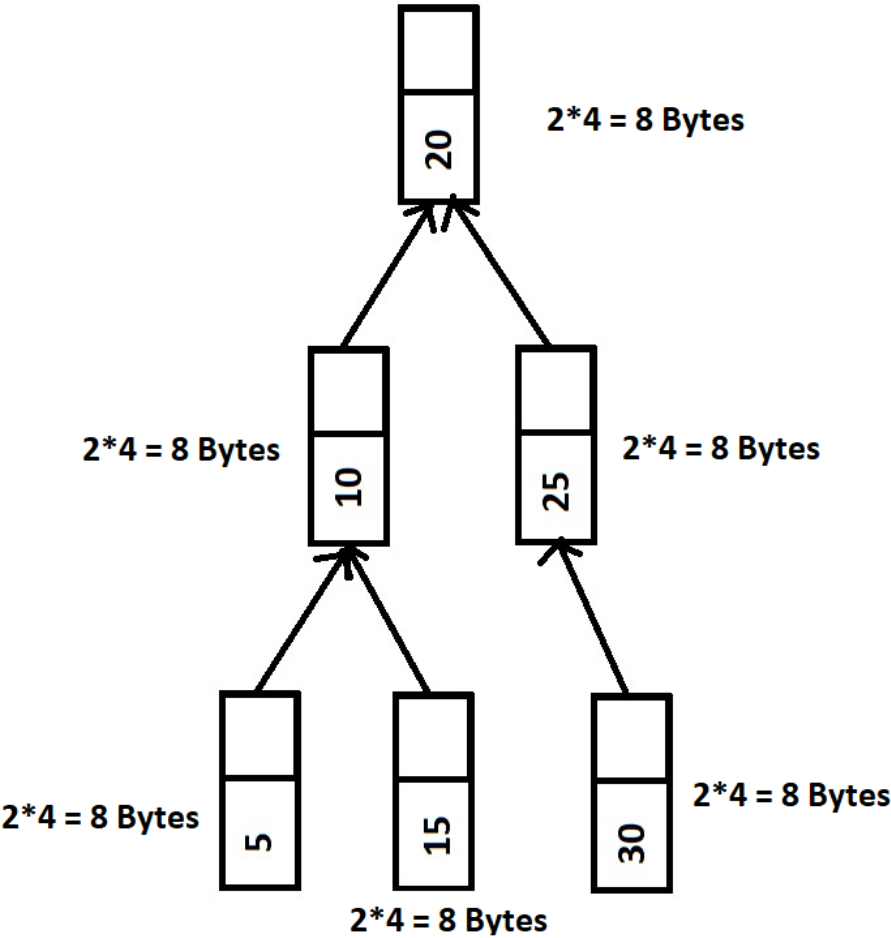
Size difference calculations



Total Size = 12*6 = 72 bytes

Index	Leafnode_ptr
1	0x1b00
2	0xab00
3	0xeb00

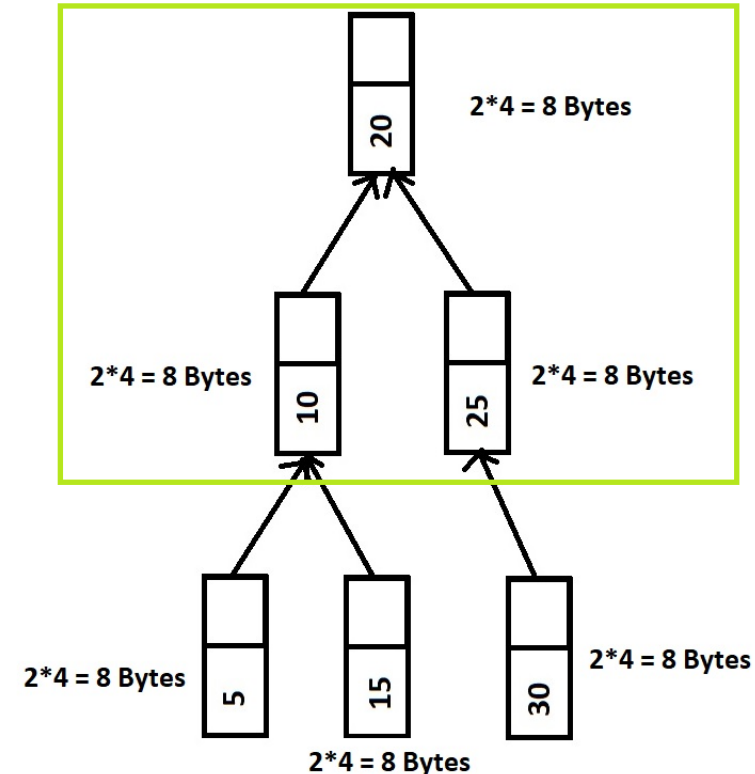
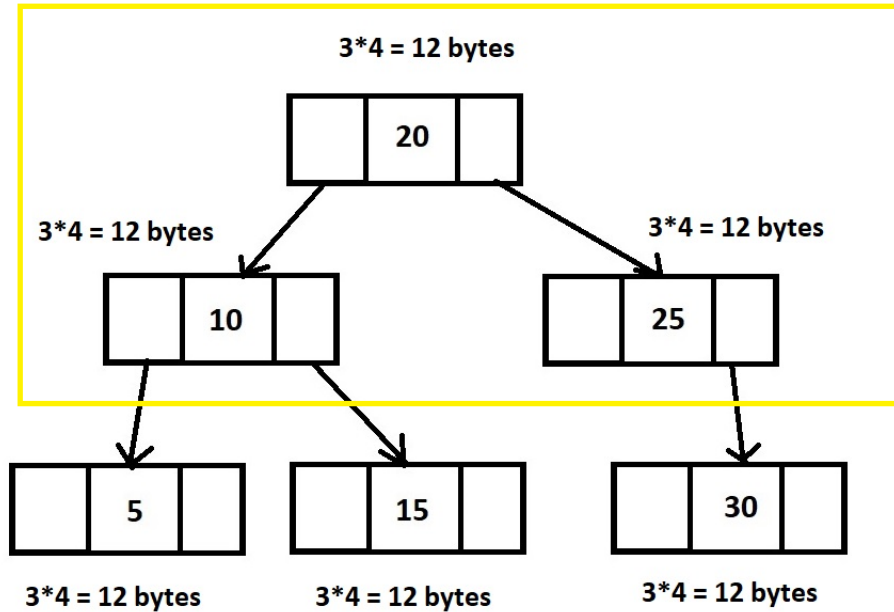
3*4 bytes = 12 bytes (indexing is not required, it's for mere presentation here)



Total Size = 8*6 = 48 bytes

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

Formula for compression



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

Compression algorithm

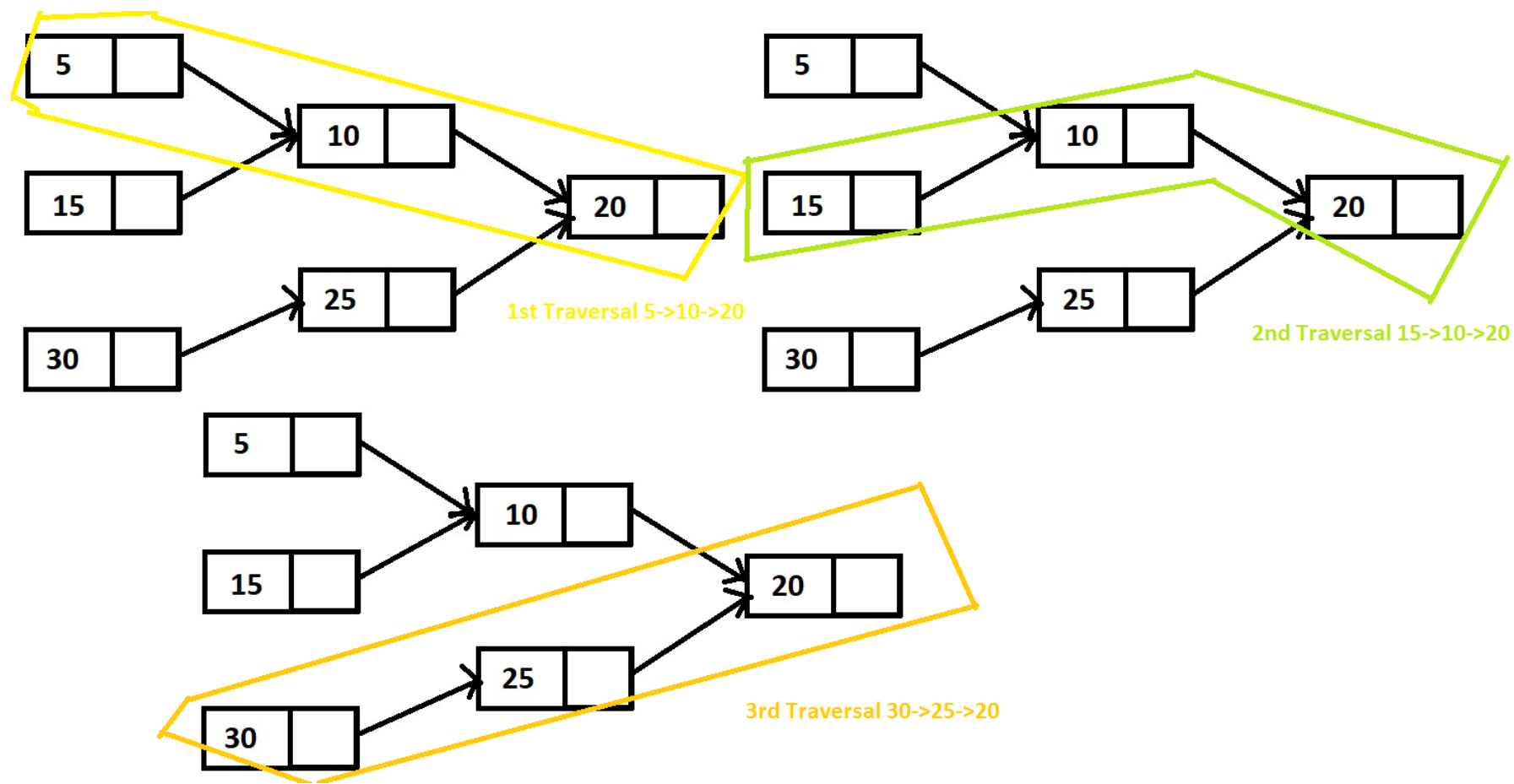
```
void compdp_trav(struct node* ptr, struct node* prev, struct node_c* previousnode)
{
    struct node* left = ptr->left;
    struct node* right = ptr->right;

    /* 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);
}
```

Traversal of the lists



Traversal algorithm

```
void linklist_dptra(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;
        }
    }
}
```

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.

More compression technique using tuple

Read from left to
right leaf nodes

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