STUDENT PORTFOLIO

Name: U.Sai Krishna
Register Number:RA2111030010060
Mail ID:sk3660@srmist.edu.in
Department: NWS
Specialization: Cyber Security
Semester:3

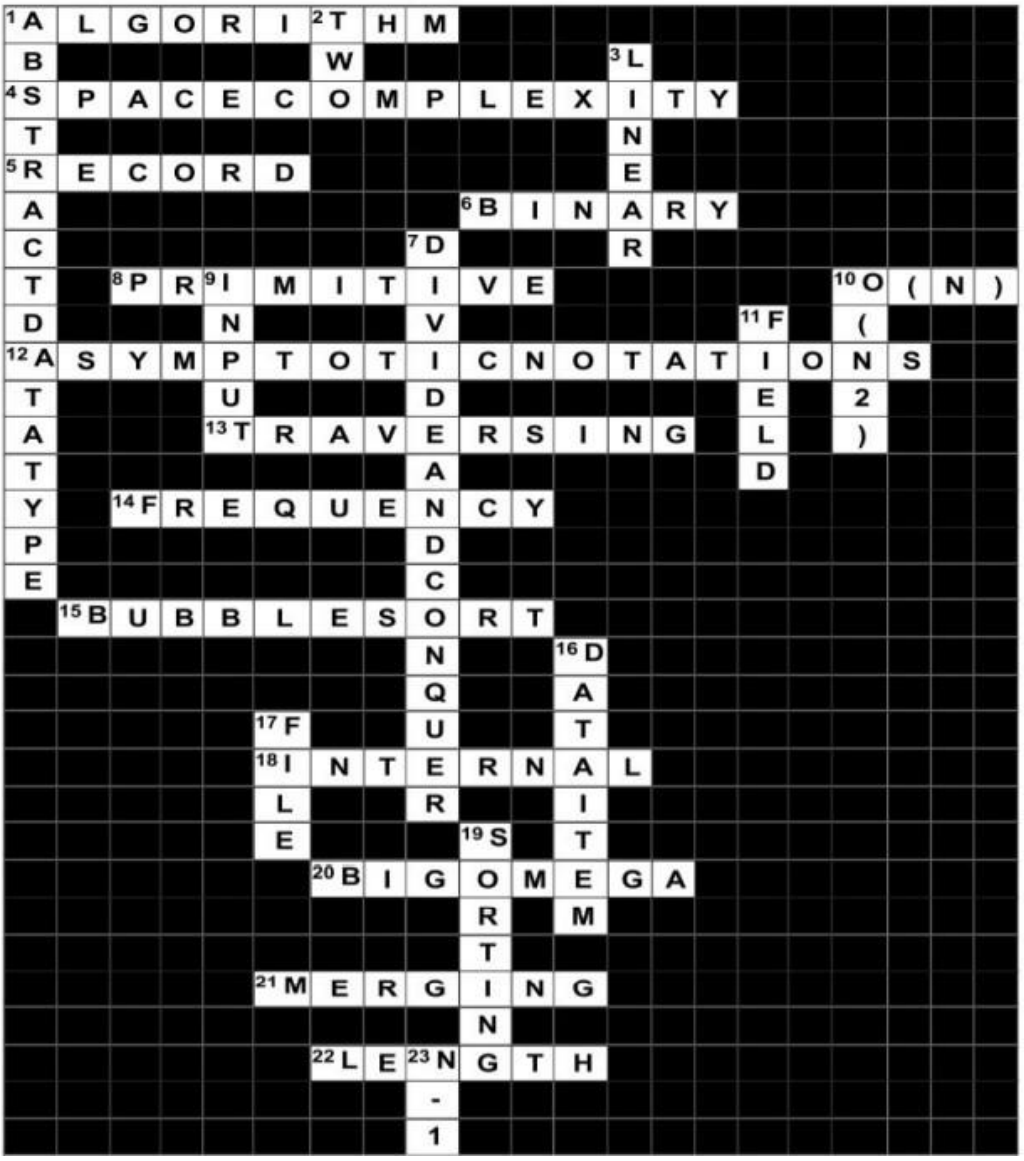
Subject Title: 18CSC201J Data Structures and Algorithms

Handled By: Dr.M.Jeyaselvi

Assignment – CrossWord Puzzle (Unit 1,2,3, & 4)
(Write about the assignment questions and how u solved differently)

UNIT-1 DATA STRUCTURES

Prepared by Dr. D.SHINY IRENE
AP/CSE/SRMIST



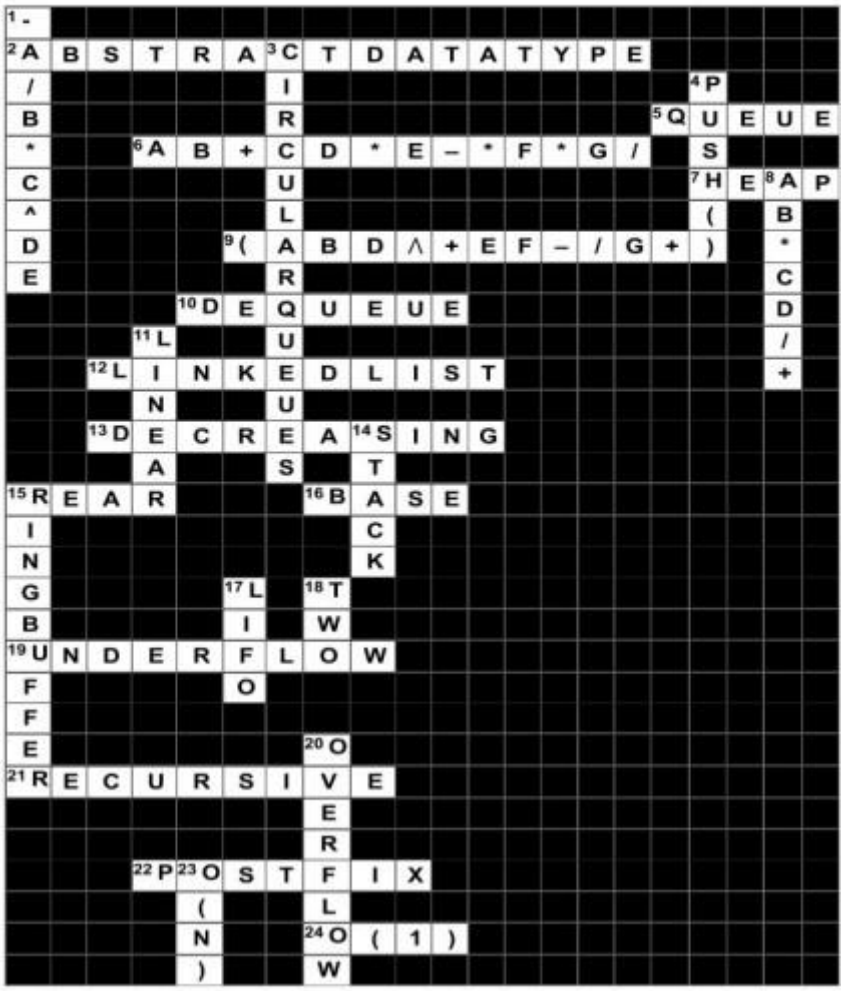
UNIT-2 DATA STRUCTURES & ALGORITHMS

Prepared by Dr. D.SHINY IRENE
AP\CSE/SRMIST

Assignment –CrossWord Puzzle (Unit 1,2,3, & 4)
(UNIT – 3)

UNIT-3 DATA STRUCTURES & ALGORITHMS

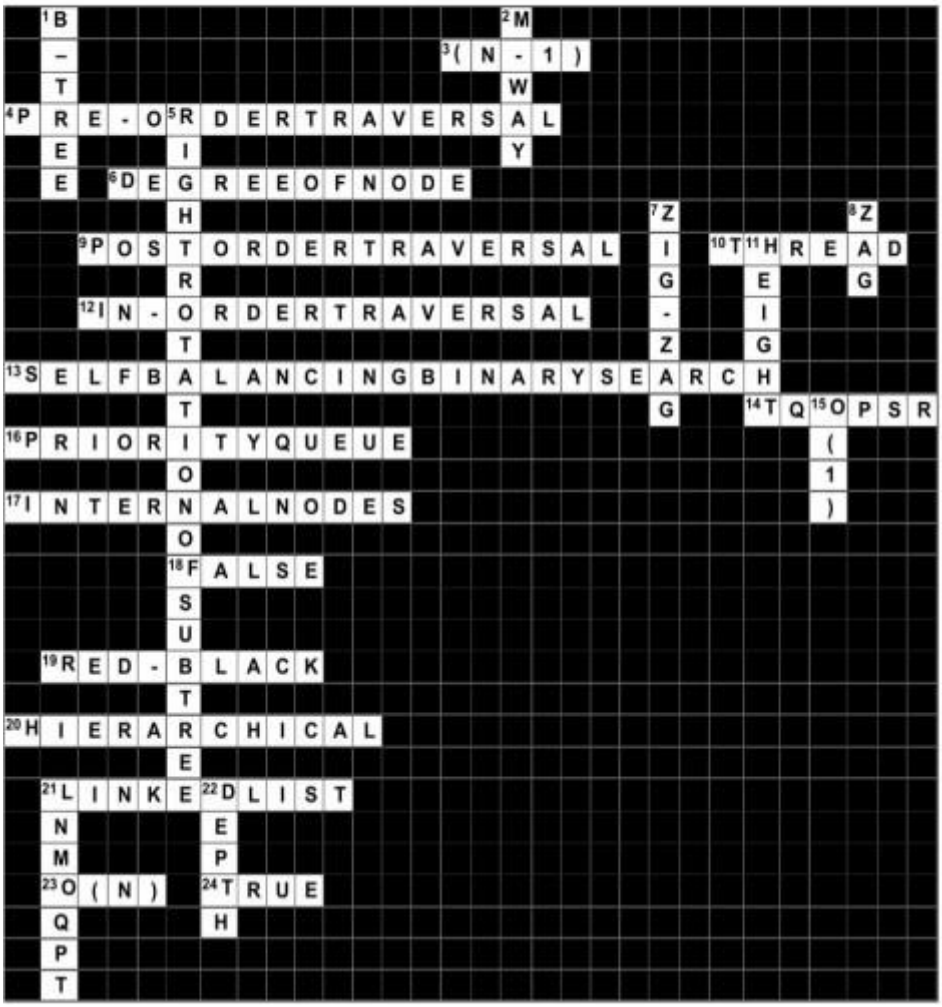
Prepared by Dr. D.SHINY IRENE
AP/CSE/SRMIST



Assignment – CrossWord Puzzle (Unit 1,2,3, & 4)
(UNIT – 4)

UNIT-4 DATA STRUCTURES & ALGORITHMS

Prepared by Dr. D.SHINY IRENE
AP/CSE/SRMIST



Assignment

(what is the most interesting part in the assignment)

Solving the puzzle was quite good.I was able to recall all the topics and at the same time

I was not feeling bored.

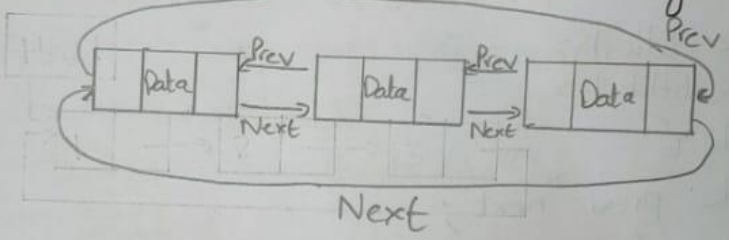
Assignment - 1

U. Sai Krishna
RA2111030010060

1. Definition of Circular Doubly Linked List.

Circular Doubly Linked List has properties of both circular and doubly linked lists, in which two consecutive elements are linked or connected by the previous and next pointer and the last node points to the first node by the next pointer and also the first node points to the last node by the previous pointer.

2. Graphical Representation of Circular Doubly Linked List.



3. Algorithm for Circular Doubly Linked List.

Step 1 => IF PTR NULL
Write Overflow
Go to step 13

Step 2 => SET NEW_NODE = PTR

Step 3 => SET PTR = PTR -> NEXT

Step 4 => SET NEW_NODE -> DATA = VAL

Step 5 => SET TEMP = HEAD

Step 6 => Repeat step 7 while TEMP->NEXT != HEAD

Step 7 => SET TEMP = TEMP -> NEXT
(End of loop)

Step 8 => SET TEMP —>NEXT = NEW_NODE
Step 9 => SET NEW_NODE -> PREV = TEMP
Step 10 => SET ~~HEAD~~ NEW_NODE —>NEXT = HEAD
Step 11 => SET HEAD -> PREV = NEW_NODE
Step 12 => SET HEAD = NEW_NODE
Step 13 => Exit

4. Code for Insertion and Deletion in Circular Doubly Linked list.

```
#include <stdio.h>
#include <stdlib.h>
Struct node {
struct node *prew *next;
int data;
};
struct node* head = NULL;
struct node* create (int);
Void insert_begin (int);
void insert_ end(int);
void insert_mid(int,int);
void delete_begin();
void delete_ end ();
void delete_mid ();
void display ();
int get data ();
int get_ position ();
```

```
delete_end();
break;
case 6:
printf("\n Delete a node from given position \n");
position = get_position();
delete_mid(position);
break;
default:
printf("\n Invalid choice \n"); }
printf("\n Do you want to continue?"); }

struct node *create(int data)
{ struct node *new_node = (struct node*) malloc(sizeof(struct node));
if (newNode == NULL){
printf("\n cant be allocated \n");
return NULL; }
new_node ->data = data;
new_node ->next = NULL;
new_node -> prev = NULL;
return new_node;
}
void insert_begin(int data){
struct node *new_node = create(data);
if(new_node)
{ if(head == NULL)
  { new_node -> next = new_node;
    new_node -> prev = new_node;
     head = new_node;
     return;
  }
 head ->prev ->next = new_node;
```

```
int main ()
{ int choice, data, position;
printf ("\n Enter your choice : ");
scanf ("%d", &choice);
switch(choice){
case 1:
printf("\n Inserting a node at beginning");
data = get_data();
insert_begin(data);
break;
case 2:
printf("\n Inserting a node at end");
data = get_data();
insert_end(data);
break;
case 3:
printf("\n Inserting a node at the given position");
data = get_data();
position = get_position();
insert_mid(position, data);
break;
case 4:
printf("\n Deleting a node from beginning\n");
delete_begin();
break;
```

```
case 5:
 printf("\n Deleting a node from end \n"); delete_end();
 break;
case 6:
 printf("\n Delete a node from given position\n");
 position = get_position();
 delete_mid(position);
 break;
default:
 printf("\n Delete a node from given position \n");
 printf("\n Invalid choice \n");
 }
 printf("\n Do you want to continue ?");
 }
 struct node * create (int data)
 {
struct node* new_node =(struct node*)malloc(sizeof(struct node));
if(new node == NULL){
printf("\n can't be allowed \n"); return NULL; }
new_node -> data = data;
new_node -> next = NULL;
new_node -> prev = NULL; return new_node; }
```

```
Void insert_begin(int data){
Struct node* new_node = create(data);
if(new_node)
{ struct if(head == NULL){
new_node -> next = new_node;
new_node -> prev = new_node;
 head = new_node;
 return;
}
head -> prev -> next = new,node;
 new,node -> prev = head -> prev;
 new_node -> next = head;
 head -> prev = new_node; head = new_node; }
}

Void insert_end(int data){
Struct node* new.node = create(data);
if(new_node){
if(head==NULL){
new_node -> next = new.node;
new_node -> prev = new.node; head = new_node;
return; }
head -> prev -> next = new_node.
new_node -> prev = head -> prev.
new_node -> next = head.
head -> prev = new_node; } }

void insert_mid(int position, data){
if(position <= 0){
printf("\n Invalid position \n"); }
else if(head == NULL && position > 1){
 printf("\n Invalid position \n"); }
```

```
else if(position ==1){ insert_begin(data); }
else { struct node*new_node = create(data);
if(new_node != NULL){
struct node *temp=head, *prev =NULL; int i =1;
while(i<=position){
prev = temp; temp=temp->next; }
prev->next = new_node;
new_node ->next = temp; }
}}
void delete_begin (){
if(head==NULL){ printf("\n List is empty \n"); return; }
else if(head->next==head){
free(head); head; NULL; return; }
struct node *last_node = head->prev;
head-> prev=last_node->prev; free(last_node); last_node=NULL;}
Void delete_mid(int position){
if(position <=0 && position >list=size()){
  printf("\n Invalid position \n"); }
else if(position ==1){ delete_begin(); }
else if(position == list_size()){ delete_end(); }
else { struct node* temp =head;
      struct node* prev = NULL; int i =1;
while(i<position){
     prev= temp; temp = temp->next; i++;}
   prev-> next = temp->next;
   temp-> next ->prev = prev;
   free(temp);
   temp = NULL; }
```

```
Void display () {
if (head == NULL) {
printf ("\n List is empty : \n");
return ; }
struct node * temp = head;
do { printf ("%d", temp->data);
     temp = temp->next; }
while (temp != head); }
int get_data () {
int data;
printf ("\n Enter data : \n");
scanf ("%d", &data);
return data ; }
int get_position () {
int position;
printf ("Enter position :");
scanf ("%d", &position);
return position;
}
```

5) Advantages and disadvantages of circular doubly linked list.

Advantages ->

1) If we are at a node, then we can go to any node. But in linear linked list it is not possible to go to previous node.

2) It saves time when we have to go to the first node. It can be done in single step because there is no need to traverse in between nodes.

Disadvantage ->

1) It is not easy to reverse the linklist.

2) If proper case is not taken, then the problem of infinite loop can occur.

3) If we at a node & go back to the previous node, then we can not do it in single step. Instead we have to complete the entire circle by going through in between nodes.

Assignment - 2

U. Sai Krishna
RA2111030010060

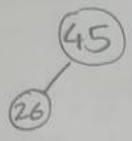
Create a binary search tree for the following numbers start from an empty binary search tree.

45, 26, 10, 60, 70, 30, 40 & delete keys 10, 60 & 45.

=> 45, 26, 10, 60, 70, 30, 40

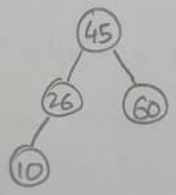
Step 1 =>

Step 2 =>

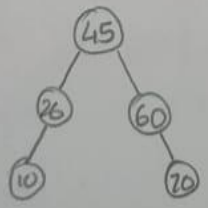


Step 3 =>

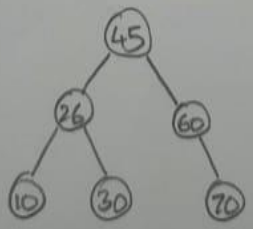
Step 4 =>



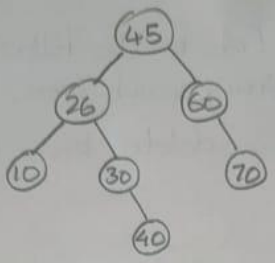
Step 5 =>



Step 6 =>

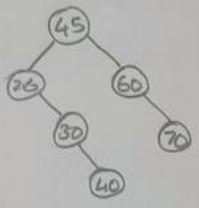


Step 7 =>

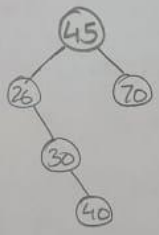


Deleting elements 10, 60, 45

Step 1 =>



Step 2 =>



Step 3 =>

Final Tree =>

Binary Search Tree

1. Definition of Binary Search Tree.

Binary search tree is a binary tree data structure with following properties.

1) The left subtree of a node contains only nodes with keys lesser than the root's key.
2) The right subtree of a node contains only nodes with keys greater than the root's key.
3) The left and right subtree each must also be a binary search tree.

Algorithm =>

Step-1 => Start

Step-2 => Create a new node

Step-3 => If tree is empty new node as root

Step-4 => Else compare new node with root element and the next nodes if it is greater put it in the right part else put it in the left part.

Step-5 => Repeat 2 until user enters 0.

Step-6 => End

Code =>

```
#include <stdio.h>
#include <stdlib.h>
struct node {
 int key;
 struct node *left *right};
struct newNode(int item){
 struct node *temp = (struct node*)malloc(sizeof(struct node));
 temp -> key = item;
 temp -> left = NULL;
 temp -> right = NULL;
 return temp; }
void inorder (struct node* root){
  if (root != NULL)
  { inorder(root -> left)
    printf("%d", root->key);
    inorder(root -> right);
   }
}
struct node *insert(struct node *node, int key){
if(node == NULL)
return newNode(key);
if(key<node -> key)
node -> left = insert(node->left,key);
```

```
else
    node->right = insert(node->right, key);
    return node;
}
struct node *deleteNode(struct node *root, int key) {
if (root == NULL)
return root;
if (key < root->key)
root->left = deleteNode(root->left, key)
else if(key > root->key)
root->right = deleteNode(root->right, key);
else {
        if(root->left = NULL) {
        struct node* temp = root->right;
        free(root);
        return temp;
        }
else if(root->right == NULL) {
    struct node *temp = root->left;
    free(root);
    return temp;
  }
}
```

```
int main () {
struct node *root = NULL;
int n=1, data, x;
while (n!=0)
scanf("%d", &data);
{root = insert(root,data)
  printf("Enter 0 to exit\n");
  scanf ("%d", &n);
}
printf("Enter node you want to delet");
scanf ("%d", &x);
inorder(root);
root = deleteNode(root,x);
inorder(root);
return 0;
}
```

Advantages of BST =>

i) Fast in insertion and deletion when balanced.

ii) We can also do range-queries. Find keys b/w N and M.

iii) Binary Search Tree is simple compared to other data structures.

Disadvantages of BST =>

i) The main disadvantage is that a BST should always be balanced.

ii) Accessing elements is slightly harder than in arrays.

iii) A imbalanced or degenerated BST can increase the complexity.

Assignment - 3

V. Sai Krishna
RA2110300100060

Consider a hash table of size seven, with starting index zero, and a hash function (3x+4) mod 7. Assuming the hash table is initially empty which of the following is the contents of the table when the sequence 1,3,8,10 is inserted into the table using closed hashing?
Note that '_' denotes an empty location in the table.

Given keys =>1, 3, 8, 10

Key	Location
1	[3(1)+4]%7 = 0
3	[3(3)+4]%7 = 6
8	[3(8)+4]%7 = 0 [To be put in next available space] => 1
10	[3(10)+4]%7 = 6 [To be put in next available space] => 2

0	1
1	8
2	10
3	
4	
5	
6	3

=> 1, 8, 10, _, _, _, 3

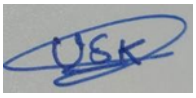
Therefore, option (B)

Codechef Achievements

https://www.codechef.com/users/srmcse_160

Any other
(Write if you registered or practise apart from Codechef(ex. Hackerrank, Leetcode etc.)

Hackerrank profile https://www.hackerrank.com/sk3660



Signature

[redacted] [redacted]

Note: Enclose the assignment and relevant certificates along with the profile