Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

**08**

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| **1** | **Design & implement all methods of Simple Queue.** |
| **2** | **Design & implement all methods of Circular Queue.** |
| **3** | **Design and implement for Priority Queue.**   * + 1. **Method 1: Ordering in/ after Enqueue method**     2. **Method 2: Separate queues for different priorities.** |
|  |  |

Submitted On:

**26-1-2022**

(Date: DD/MM/YY)

**Task 1: Design & implement all methods of Simple Queue.**

**SOLUTION:** public class Queue {

private static int front, rear, capacity;

private static int[] queue;

public Queue(int c){

front = rear = 0;

capacity = c;

queue = new int[capacity];}

public void queueEnqueue(int data) {

if (capacity == rear) {

Console.Write("\nQueue is full\n");

return; }

else {

queue[rear] = data;

rear++; }

return; }

public void queueDequeue(){

// if queue is empty

if (front == rear) {

Console.Write("\nQueue is empty\n");

return; }

else {

for (int i = 0; i < rear - 1; i++) {

queue[i] = queue[i + 1]; }

if (rear < capacity)

queue[rear] = 0;

rear--; }

return; }

// print queue elements

public void queueDisplay(){

int i;

if (front == rear) {

Console.Write("\nQueue is Empty\n");

return; }

for (i = front; i < rear; i++) {

Console.Write(" {0} <-- ", queue[i]); }

return; }

// print front of queue

public void queueFront() {

if (front == rear) {

Console.Write("\nQueue is Empty\n");

return; }

Console.Write("\n\nFront Element is: {0}", queue[front]);

return;

class Program {

static void Main(string[] args){

Queue queue = new Queue(5);

queue.queueDisplay();

queue.queueEnqueue(20);

queue.queueEnqueue(30);

queue.queueEnqueue(40);

queue.queueEnqueue(50);

queue.queueEnqueue(60);

queue.queueDisplay();

queue.queueEnqueue(70);

queue.queueDisplay();

queue.queueDequeue();

queue.queueDequeue();

Console.Write("\n\nafter two node deletion\n\n");

queue.queueDisplay();

queue.queueFront();

Console.WriteLine("\n");

**OUTPUT:**

Text

Description automatically generated

**Task 2: Design & implement all methods of Circular Queue.**

**SOLUTION:**

class CircularQueue{

private int[] array;

private int length, rear, front;

public CircularQueue(int len) {

array = new int[len];

this.length = len;

this.front = -1;

this.rear = -1; }

public bool isFull() {

if ((front == rear + 1) || (front == 0 && rear == length - 1)){

return true; }

else {

return false; } }

public bool isEmpty(){

if (front == -1) {

return true; }

else {

return false; } }

public void enque(int val) {

if (isFull()){

Console.WriteLine("Que Full");

return; }

else {

if (front == -1) {

front = 0; }

rear = (rear + 1) % length;

array[rear] = val; } }

public int deque() {

int val;

if (isEmpty()) {

Console.WriteLine("Que Underflow");

return -1; }

else {

val = array[front];

if (front == rear) {

front = -1;

rear = -1; }

else {

array[front] = 0;

front = (front + 1) % length; }

return val; } }

public void Display() {

for (int i = 0; i < array.Length; i++) {

Console.WriteLine(array[i]); }} }

class Program {

static void Main(string[] args) {

CircularQueue q1 = new CircularQueue(5);

q1.enque(5);

q1.enque(10);

q1.enque(15);

q1.enque(20);

q1.enque(25);

q1.Display();

Console.WriteLine("\n------------\n");

q1.deque();

q1.deque();

q1.enque(100);

q1.Display();

**OUTPUT:**

Text

Description automatically generated

**Task 3: Design and implement for Priority Queue.**

* **Method 1: Ordering in/ after Enqueue method**
* **Method 2: Separate queues for different priorities.**

**SOLUTION:**

public class Node {

public int priority;

public int info;

public Node link;

public Node(int i, int pr) {

info = i;

priority = pr;

link = null; } }

public class PriorityQueueL {

private Node front;

public PriorityQueueL(){

front = null; }

public void Insert(int element, int elementPriority) {

Node temp, p;

temp = new Node(element, elementPriority);

/\* Queue is empty or element to be added has priority more than first element \*/

if (IsEmpty() || elementPriority < front.priority) {

temp.link = front;

front = temp; }

else {

p = front;

while (p.link != null && p.link.priority <= elementPriority)

p = p.link;

temp.link = p.link;

p.link = temp;

public int Delete() {

int element;

if (IsEmpty())

throw new System.InvalidOperationException("Queue Underflow");

else {

element = front.info;

front = front.link; }

return element; }

public bool IsEmpty() {

return (front == null); }

public void Display() {

Node p = front;

if (IsEmpty())

Console.WriteLine("Queue is empty\n");

else {

Console.WriteLine("Queue is :");

Console.WriteLine("Element Priority");

while (p != null) {

Console.WriteLine(p.info + " " + p.priority);

p = p.link; } }

Console.WriteLine(""); } }

class Program {

static void Main(string[] args) {

int choice, element, elementPriority;

PriorityQueueL pq = new PriorityQueueL();

while (true) {

Console.WriteLine("1.Insert a new element");

Console.WriteLine("2.Delete an element");

Console.WriteLine("3.Display the queue");

Console.WriteLine("4.Quit");

Console.Write("Enter your choice : ");

choice = Convert.ToInt32(Console.ReadLine());

if (choice == 4)

break;

switch (choice) {

case 1:

Console.WriteLine("Enter the element to be inserted: ");

element = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter its priority : ");

elementPriority = Convert.ToInt32(Console.ReadLine());

pq.Insert(element, elementPriority);

break;

case 2:

Console.WriteLine("Deleted element is: " + pq.Delete());

break;

case 3:

pq.Display();

break;

default:

Console.WriteLine("Wrong choice");

break; }

**OUTPUT:**

Text

Description automatically generated

Text

Description automatically generated

Bahria University,

Karachi Campus

A picture containing text, room

Description automatically generated

LAB EXPERIMENT NO.

**09**

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| **1** | **Implement bucket sort using linked list** |
| **2** | **Create static tree and perform in order, preorder, and post order traversal. Also search a required node in the tree.** |
|  |  |

Submitted On:

**26-1-2022**

(Date: DD/MM/YY)

**Task 1: Implement bucket sort using linked list**

**.**

**SOLUTION:**

public static void BucketSort(ref int[] arr) {

int min = int.MaxValue;

int max = 0;

for (int i = 0; i < arr.Length; i++){

if (arr[i] < min)

min = arr[i];

if (arr[i] > max)

max = arr[i]; }

List<int>[] b = new List<int>[max - min + 1];

for (int i = 0; i < b.Length; i++) {

b[i] = new List<int>();}

for (int i = 0; i < arr.Length; i++){

b[arr[i] - min].Add(arr[i]); }

int k = 0;

for (int i = 0; i < b.Length; i++) {

if (b[i].Count > 0) {

for (int j = 0; j < b[j].Count; j++){

arr[k] = b[i][j];

k++;

static void Main(string[] args){

int[] arr = new int[] { 9, 3, 6, 2, 8 };

Console.WriteLine("Elements are:");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("9");

Console.WriteLine("==============================");

Console.WriteLine("3");

Console.WriteLine("==============================");

Console.WriteLine("6");

Console.WriteLine("==============================");

Console.WriteLine("2");

Console.WriteLine("==============================");

Console.WriteLine("8");

Console.WriteLine("==============================");

Console.WriteLine("Elements after applying bucket sort are:");

BucketSort(ref arr);

arr.ToList().ForEach(x => Console.WriteLine(x));

**OUTPUT:**

Text

Description automatically generated

**Task 2: Create static tree and perform in order, preorder, and post order traversal. Also search a required node in the tree.**

**SOLUTION:**

class TreeNode {

int key;

public TreeNode left, Right;

public TreeNode(int key) {

this.key = key;

this.left = this.Right = null; }

public TreeNode getLeft(){

return this.left; }

public int GetKey(){

return this.key; }

public TreeNode getRight(){

return this.Right;}

public void setLeft(int key) {

this.left = new TreeNode(key); }

public void setRight(int key) {

this.Right = new TreeNode(key);}}

class BinaryTree{

public TreeNode Root;

public BinaryTree(){

this.Root = null;}

public BinaryTree(int key){

this.Root = new TreeNode(key);}

public void setRoot(int key){

this.Root = new TreeNode(key);}

public TreeNode getRoot(){

return this.Root;}

public void Inorder(TreeNode Node){

if (Node == null)

return;

else{

Inorder(Node.left);

Console.Write(Node.GetKey() + " ");

Inorder(Node.Right);}}

public void Preorder(TreeNode Node) {

if (Node == null)

return;

else{

Console.Write(Node.GetKey() + " ");

Preorder(Node.left);

Preorder(Node.Right);} }

public void PostOrder(TreeNode Node){

if (Node == null)

return;

else{

PostOrder(Node.left);

PostOrder(Node.Right);

Console.Write(Node.GetKey() + " ");}}}

class Program{

public static void Main(){

BinaryTree tree = new BinaryTree();

tree.setRoot(7);

tree.Root.setLeft(3);

tree.Root.setRight(8);

tree.Root.left.setRight(10);

tree.Root.left.setLeft(18);

tree.Root.Right.setRight(12);

tree.Root.Right.setLeft(19);

System.Console.WriteLine("--------InOrder Traversal--------");

tree.Inorder(tree.Root);

System.Console.WriteLine();

System.Console.WriteLine("--------PreOrder Traversal--------");

tree.Preorder(tree.Root);

System.Console.WriteLine();

System.Console.WriteLine("--------PostOrder Traversal--------");

tree.PostOrder(tree.Root);

**OUTPUT:**

![Graphical user interface, text

Description automatically generated]()

Bahria University,

Karachi Campus

A picture containing text, room

Description automatically generated

LAB EXPERIMENT NO.

**10**

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| **1** | **Write a program to implement concept of Binary Search Tree using dynamic trees** |
| **2** | **Implement the AVL Tree by performing searching.** |
|  |  |
|  |  |

Submitted On:

**26-1-2022**

(Date: DD/MM/YY)

**Task 1: Write a program to implement concept of Binary Search Tree using dynamic trees**

**SOLUTION:**

class Program{

public class Node{

public int data;

public Node left;

public Node right;

public Node(int data){

this.data = data;}

public Node()

{ }}

public class BST{

Node root;

public BST(){}

public BST(int data){

root = new Node(data);}

public void Insert(int data){

Node newElement = new Node(data);

if (root == null){

root = newElement;}

else{

root = Insert(root, newElement);}}

private Node Insert(Node node, Node nodeToAdd){

if (node == null){

node = nodeToAdd;

return node;}

else if (nodeToAdd.data < node.data){

node.left = Insert(node.left, nodeToAdd);}

else if (nodeToAdd.data >= node.data){

node.right = Insert(node.right, nodeToAdd);}

return node;}

public void display(){

if (root == null) {

Console.WriteLine("Binary Search Tree Is Empty!");}

inOrder(root);}

public void inOrder(Node node){

if (node != null){

inOrder(node.left);

Console.WriteLine("Value in the Node is : {0}", node.data);

inOrder(node.right);}}

public void search(int key){

if (search(key, root).data == key) {

Console.WriteLine("-------------------------");

Console.WriteLine(key + " Found ");}

else{

Console.WriteLine(key + " Not Found ");}}

public Node search(int value, Node currentNode){

if (value > currentNode.data){

if (value == currentNode.data)

return currentNode;

else{

return search(value, currentNode.right);}}

else{

if (value == currentNode.data)

return currentNode;

else{

return search(value, currentNode.left);}}

static void Main(string[] args{

Console.WriteLine("==============================");

Console.WriteLine("\*\*\*\*\*Binary Search Tree\*\*\*\*\*");

Console.WriteLine("==============================");

BST tree = new BST(0);

tree.Insert(5);

tree.Insert(14);

tree.Insert(1);

tree.Insert(9);

tree.display();

Console.WriteLine("==============================");

tree.search(5);

Console.WriteLine("==============================");

**OUTPUT:**

Text

Description automatically generated

**Task 2: Implement the AVL Tree by performing searching.**

**SOLUTION:**

class Program {

class Node{

public int data;

public Node left;

public Node right;

public Node(int data){

this.data = data;}

public Node()

{ }}

class AVL{

Node root;

public AVL(){

root = new Node();}

public void Insert(int data){

Node newItem = new Node(data);

if (root == null){

root = newItem;}

else{

root = Insert(root, newItem);}}

private Node Insert(Node r, Node n){

if (r == null){

r = n;

return r;}

else if (n.data < r.data){

r.left = Insert(r.left, n);

r = CheckBalance(r);

return r;}

else if (n.data >= r.data) {

r.right = Insert(r.right, n);

r = CheckBalance(r);

return r; }

return r; }

private Node CheckBalance(Node temp) {

int d = difference(temp);

if (d > 1) {

if (difference(temp.left) > 0) {

temp = RotateLL(temp); }

else {

temp = RotateLR(temp); }}

if (d < -1) {

if (difference(temp.right) > 0) {

temp = RotateRL(temp); }

else {

temp = RotateRR(temp); }}

return temp;}

private int depth(Node current){

int dep = 0;

if (current != null){

int l = depth(current.left);

int r = depth(current.right);

int m = max(l, r);

dep = m + 1; }

return dep;}

private int difference(Node current){

int l = depth(current.left);

int r = depth(current.right);

int diff = l - r;

return diff;}

private Node RotateRR(Node parent){

Node temp = parent.right;

parent.right = temp.left;

temp.left = parent;

return temp;}

private Node RotateLL(Node parent){

Node temp = parent.left;

parent.left = temp.right;

temp.right = parent;

return temp;}

private Node RotateLR(Node parent){

Node temp = parent.left;

parent.left = RotateRR(temp);

return RotateLL(parent);}

private Node RotateRL(Node parent){

Node temp = parent.right;

parent.right = RotateLL(temp);

return RotateRR(parent);}

private int max(int l, int r){

if (l > r)

return l;

else

return r;}

public void search(int key){

if (search(key, root).data == key){

Console.WriteLine("-----------------");

Console.WriteLine(key + " Found");}

else{

Console.WriteLine("-----------------");

Console.WriteLine(key + " Not Found");}}

private Node search(int value, Node current){

if (value < current.data){

if (value == current.data)

return current;

else

return search(value, current.left);}

else{

if (value == current.data)

return current;

else

return search(value, current.right);}}

public void display(){

if (root == null){

Console.WriteLine("AVL Tree Is Empty");

return;}

inOrder(root);}

public void inOrder(Node node){

if (node != null){

inOrder(node.left);

Console.WriteLine("Value in the Node is : {0}", node.data);

inOrder(node.right);}}}

static void Main(string[] args){

Console.WriteLine("======AVL TREE=======");

AVL tree = new AVL();

tree.Insert(5);

tree.Insert(3);

tree.Insert(1);

tree.Insert(9);

Console.WriteLine("==========================");

tree.display();

tree.search(5);

Console.WriteLine("==========================");}

**OUTPUT:**

Text

Description automatically generated

Bahria University,

Karachi Campus

A picture containing text, room

Description automatically generated

LAB EXPERIMENT NO.

**11**

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| **1** | **Create a program to implement Graphs with Adjacency matrix** |
| **2** | **Create a program to implement Graphs with Adjacency list** |
| **3** | **Create a program to implement BFS** |
| 4 | Create a program to implement DFS. |

Submitted On:

**26-1-2022**

(Date: DD/MM/YY)

**Task 1: Create a program to implement Graphs with Adjacency matrix**

**SOLUTION:**

public class AdjMatrixGraph{

int vertex;

int[,] matrix;

public AdjMatrixGraph(int vertex){

this.vertex = vertex;

matrix = new int[vertex, vertex];}

public void addEdge(int source, int destination){

matrix[source, destination] = 1;

matrix[destination, source] = 1;}

public void printGraph(){

Console.WriteLine("\nAdjacency Matrix\n");

for (int i = 0; i < vertex; i++){

for (int j = 0; j < vertex; j++){

Console.Write(matrix[i, j] + " ");}

Console.WriteLine();}}

class Program {

static void Main(String[] args){

AdjMatrixGraph graph = new AdjMatrixGraph(5);

graph.addEdge(0, 1);

graph.addEdge(1, 1);

graph.addEdge(2, 3);

graph.addEdge(3, 3);

graph.addEdge(3, 3);

graph.addEdge(4, 2);

graph.printGraph();

Console.WriteLine();

**OUTPUT:**

Text

Description automatically generated

**Task 2: Create a program to implement Graphs with Adjacency list**

**SOLUTION:**

class Program{

static void addEdges(LinkedList<int>[] adj, int u, int v){

adj[u].AddLast(v);

adj[v].AddLast(u); }

static void printGraph(LinkedList<int>[] adj){

Console.WriteLine("\nAdjacency List");

Console.WriteLine("--------------");

for (int i = 0; i < adj.Length; i++){

Console.Write("Head ({0}) : {1} ", i, i);

foreach (var item in adj[i]){

Console.Write(" --> " + item);}

Console.WriteLine();}}

static void Main(string[] args){

LinkedList<int>[] adjMatrix = new LinkedList<int>[5];

for (int i = 0; i < 5; i++){

adjMatrix[i] = new LinkedList<int>();}

addEdges(adjMatrix, 0, 1);

addEdges(adjMatrix, 1, 2);

addEdges(adjMatrix, 2, 3);

addEdges(adjMatrix, 3, 4);

addEdges(adjMatrix, 4, 0);

printGraph(adjMatrix);

Console.WriteLine();

**OUTPUT:**

A picture containing text

Description automatically generated

**Task 3: Create a program to implement BFS**

**SOLUTION:**

class Program{

class Graph{

private int \_V;

LinkedList<int>[] \_adj;

public Graph(int V){

\_adj = new LinkedList<int>[V];

for (int i = 0; i < \_adj.Length; i++){

\_adj[i] = new LinkedList<int>();}

\_V = V;}

public void AddEdge(int v, int w){

\_adj[v].AddLast(w);}

public void BFS(int s){

bool[] visited = new bool[\_V];

for (int i = 0; i < \_V; i++)

visited[i] = false;

LinkedList<int> queue = new LinkedList<int>();

visited[s] = true;

queue.AddLast(s);

while (queue.Any()){

s = queue.First();

Console.Write(s + " ");

queue.RemoveFirst();

LinkedList<int> list = \_adj[s];

foreach (var val in list){

if (!visited[val]){

visited[val] = true;

queue.AddLast(val);}}}}

static void Main(string[] args){

Graph g = new Graph(4);

g.AddEdge(0, 1);

g.AddEdge(0, 2);

g.AddEdge(1, 2);

g.AddEdge(2, 0);

g.AddEdge(2, 1);

g.AddEdge(2, 3);

g.AddEdge(3, 3);

Console.WriteLine("Following is Breadth First");

g.BFS(1);

**OUTPUT:**

Text

Description automatically generated

**Task 4: Create a program to implement DFS.**

**SOLUTION:**

class Program{

public class Graph{

LinkedList<int>[] linkedListArray;

public Graph(int v){

linkedListArray = new LinkedList<int>[v];}

public void AddEdge(int u, int v, bool blnBiDir = true){

if (linkedListArray[u] == null){

linkedListArray[u] = new LinkedList<int>();

linkedListArray[u].AddFirst(v);}

else{

var last = linkedListArray[u].Last;

linkedListArray[u].AddAfter(last, v);}

if (blnBiDir){

if (linkedListArray[v] == null){

linkedListArray[v] = new LinkedList<int>();

linkedListArray[v].AddFirst(u);}

else{

var last = linkedListArray[v].Last;

linkedListArray[v].AddAfter(last, u);}}}

internal void DFSHelper(int src, bool[] visited){

visited[src] = true;

Console.Write(src + " ");

if (linkedListArray[src] != null){

foreach (var item in linkedListArray[src]) {

if (!visited[item] == true){

DFSHelper(item, visited);}}}}

internal void DFS(){

Console.WriteLine("\n\t\tDFS");

Console.WriteLine("\t\t---\n");

bool[] visited = new bool[linkedListArray.Length + 1];

DFSHelper(1, visited);}}

static void Main(string[] args){

Graph graph = new Graph(11);

graph.AddEdge(1, 2, false);

graph.AddEdge(2, 3, false);

graph.AddEdge(3, 4, false);

graph.AddEdge(1, 5, false);

graph.AddEdge(5, 6, false);

graph.AddEdge(6, 7, false);

graph.AddEdge(5, 8, false);

graph.AddEdge(1, 9, false);

graph.AddEdge(9, 10, false);

graph.DFS();

**OUTPUT:**

A screenshot of a computer

Description automatically generated with medium confidence