Lab 1 -Stack

import java.io.FileWriter;

import java.io.IOException;

public class FileTest {

public static void main(String args[]) throws IOException //Throwing exception so calling class will deal w it.

{

try(FileWriter locFile =new FileWriter("Test.dat"))//Better because neater and closes stream automatically.

{

locFile.write("2021A7PS001 AAAA 1/1/2000 7.50"+"\n");

locFile.write("2021A7PS002 BBBB 2/1/2000 9.20"+"\n");

locFile.write("2021A7PS003 CCCC 3/1/2000 9.60"+"\n");

locFile.write("2021A7PS004 DDDD 4/1/2000 8.75"+"\n");

locFile.write("2021A7PS005 EEEE 5/1/2000 9.25"+"\n");

}

}

/\*FileWriter locFile =null;//Should be outside.

try

{

locFile = new FileWriter("Test.dat");

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

locFile.write(i+"" + "\n");

}

}finally

{

if(locFile!=null) {

System.out.println("Closing.");

locFile.close();

}

}\*/

}

import java.io.FileReader;

import java.io.FileWriter;

import java.util.Scanner;

import java.util.Stack;

public class DataReader {

static String arr[]=new String[5];

static{

try {

Scanner scanner = new Scanner(new FileReader("Test.dat"));

scanner.useDelimiter("\\n");

int i=0;

while(scanner.hasNextLine())

{

String let=scanner.nextLine();

arr[i]=let;

i++;

}

}catch(Exception e)

{

e.printStackTrace();

}

}

public static void main(String args[])

{

Stack stk=new Stack();

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

{

System.out.println(arr[i]);

stk.push(arr[i]);

}

System.out.println("");

try(FileWriter outFile =new FileWriter("Out.dat"))//Better because neater and closes stream automatically.

{

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

{

String temp=stk.pop().toString();

outFile.write(temp+"\n");

System.out.println(temp);

}

}catch (Exception e)

{

e.printStackTrace();

}

}

}

Lab 2 - Queue

import java.util.\*;

import java.io.\*;

class student{

public String id, name, dob;

public float gpa;

student(String id, String name, String dob, float gpa){

this.id = id;

this.name = name;

this.dob = dob;

this.gpa = gpa;

}

}

class Queue{

private int size, front, rear;

private ArrayList<student> circular\_queue = new ArrayList<student>();

Queue(int queue\_size)

{

this.size = queue\_size;

this.front = this.rear = -1;

}

public void enQueue(student queue\_data)

{

if((front == 0 && rear == size - 1) ||

(rear == (front - 1) % (size - 1)))

{

System.out.print("Queue Full!");

}

else if(front == -1)

{

front = 0;

rear = 0;

circular\_queue.add(rear, queue\_data);

}

else if(rear == size - 1 && front != 0)

{

rear = 0;

circular\_queue.set(rear, queue\_data);

}

else

{

rear = (rear + 1);

if(front <= rear)

{

circular\_queue.add(rear, queue\_data);

}

else

{

circular\_queue.set(rear, queue\_data);

}

}

}

public student deQueue()

{

student temp;

if(front == -1)

{

System.out.print("Queue Empty!");

throw new EmptyStackException();

}

temp = circular\_queue.get(front);

if(front == rear)

{

front = -1;

rear = -1;

}

else if(front == size - 1)

{

front = 0;

}

else

{

front = front + 1;

}

return temp;

}

}

class Main

{

public static void main (String args[]) throws FileNotFoundException

{

Scanner readMyFile = new Scanner(new File(args[0]));

PrintWriter writeToMyFile = new PrintWriter(new File(args[1]));

Queue obj = new Queue(5);

while(readMyFile.hasNext()){

String id = readMyFile.next();

String name = readMyFile.next();

String dob = readMyFile.next();

Float gpa = readMyFile.nextFloat();

student stu = new student(id, name, dob, gpa);

obj.enQueue(stu);

}

for (int k =1; k<=5; k++)

{

student st = obj.deQueue();

if(st.gpa < 9.0f){

System.out.println(st.name);

}

writeToMyFile.format("%s %s %s %f%n", st.id,st.name,st.dob,st.gpa);

}

readMyFile.close();

writeToMyFile.close();

}

}

Lab 3 - DLL

import java.util.\*;

import java.io.\*;

class student{

public String id, name, dob;

public float gpa;

student(String id, String name, String dob, float gpa){

this.id = id;

this.name = name;

this.dob = dob;

this.gpa = gpa;

}

}

class Node

{

student data;

Node next, prev;

}

class LinkedList

{

static Node head = null;

static Node deleteNode(Node del)

{

if (head == null || del == null)

return null;

if (head == del)

head = del.next;

if (del.next != null)

del.next.prev = del.prev;

if (del.prev != null)

del.prev.next = del.next;

del = null;

return head;

}

static void deleteNodeAtGivenPos(int n)

{

if (head == null || n <= 0)

return;

Node current = head;

int i;

for (i = 1; current != null && i < n; i++)

{

current = current.next;

}

if (current == null)

return;

deleteNode(current);

}

static void push(student new\_data)

{

Node new\_node = new Node();

new\_node.data = new\_data;

new\_node.prev = null;

new\_node.next = head;

if (head != null)

head.prev = new\_node;

head = new\_node;

}

static void printList()

{

Node temp = head;

if (temp == null)

System.out.print("Doubly Linked list empty");

while (temp != null)

{

student st = temp.data;

System.out.format("%s %s %s %f%n", st.id,st.name,st.dob,st.gpa);

temp = temp.next;

}

System.out.println();

}

static void reversePrint( )

{

Node head\_ref = head;

Node tail = head\_ref;

while (tail.next != null)

{

tail = tail.next;

}

while (tail != head\_ref)

{

student st = tail.data;

System.out.format("%s %s %s %f%n", st.id,st.name,st.dob,st.gpa);

tail = tail.prev;

}

student st = tail.data;

System.out.format("%s %s %s %f%n", st.id,st.name,st.dob,st.gpa);

}

public static void main (String args[]) throws FileNotFoundException

{

Scanner readMyFile = new Scanner(new File(args[0]));

while(readMyFile.hasNext()){

String id = readMyFile.next();

String name = readMyFile.next();

String dob = readMyFile.next();

Float gpa = readMyFile.nextFloat();

student stu = new student(id, name, dob, gpa);

push(stu);

}

System.out.println("Doubly linked "

+"list before deletion:");

printList();

System.out.println("Doubly linked "

+"list before deletion(In reverse order):");

reversePrint();

Scanner s = new Scanner(System.in);

System.out.println();

System.out.print("Enter node position to delete: ");

int n = s.nextInt();

deleteNodeAtGivenPos(n);

System.out.println("Doubly linked "

+"list after deletion:");

printList();

readMyFile.close();

}

}

Lab 4 - Linear Search, Binary Search

public class Search

{

public static int linearSearch(int[] arr, int key){

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

if(arr[i] == key){

return i;

}

}

return -1;

}

public static int binarySearch(int arr[], int l, int r, int x)

{

if (r >= l) {

int mid = l + (r - l) / 2;

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

return binarySearch(arr, mid + 1, r, x);

}

return -1;

}

public static void main(String[] args) {

int[] arr = new int[10000];

for(int i = 0; i<10000;i++){

arr[i] = i;

}

System.out.println("Carrying out linear search");

System.out.println("\nInput: 5000");

long startTime = System.nanoTime();

int x = linearSearch(arr, 5000);

long endTime = System.nanoTime() - startTime;

if(x == -1){

System.out.println("Not Found");

System.out.println("Execution Time(nano seconds): "+endTime);

}

else{

System.out.println("Found at position: "+x);

System.out.println("Execution Time(nano seconds): "+endTime);

}

System.out.println("\nInput: 9997");

startTime = System.nanoTime();

x = linearSearch(arr, 9997);

endTime = System.nanoTime() - startTime;

if(x == -1){

System.out.println("Not Found");

System.out.println("Execution Time(nano seconds): "+endTime);

}

else{

System.out.println("Found at position: "+x);

System.out.println("Execution Time(nano seconds): "+endTime);

}

System.out.println("\nInput: 50000");

startTime = System.nanoTime();

x = linearSearch(arr, 50000);

endTime = System.nanoTime() - startTime;

if(x == -1){

System.out.println("Not Found");

System.out.println("Execution Time(nano seconds): "+endTime);

}

else{

System.out.println("Found at position: "+x);

System.out.println("Execution Time(nano seconds): "+endTime);

}

System.out.println("\n\nCarrying out binary search");

startTime = System.nanoTime();

x = binarySearch(arr, 0, 9999, 5000);

endTime = System.nanoTime() - startTime;

if(x == -1){

System.out.println("Not Found");

System.out.println("Execution Time(nano seconds): "+endTime);

}

else{

System.out.println("Found at position: "+x);

System.out.println("Execution Time(nano seconds): "+endTime);

}

System.out.println("\nInput: 9997");

startTime = System.nanoTime();

x = binarySearch(arr, 0, 9999, 9997);

endTime = System.nanoTime() - startTime;

if(x == -1){

System.out.println("Not Found");

System.out.println("Execution Time(nano seconds): "+endTime);

}

else{

System.out.println("Found at position: "+x);

System.out.println("Execution Time(nano seconds): "+endTime);

}

System.out.println("\nInput: 50000");

startTime = System.nanoTime();

x = binarySearch(arr, 0, 9999, 50000);

endTime = System.nanoTime() - startTime;

if(x == -1){

System.out.println("Not Found");

System.out.println("Execution Time(nano seconds): "+endTime);

}

else{

System.out.println("Found at position: "+x);

System.out.println("Execution Time(nano seconds): "+endTime);

}

}

}

Lab 5 - Merge Sort

import java.util.\*;

import java.lang.Math;

import java.io.\*;

public class Main

{

private static void mergeSort(int[] array, int low, int high)

{

if (high <= low)

return;

int mid = (low + high) / 2;

mergeSort(array, low, mid);

mergeSort(array, mid + 1, high);

merge(array, low, mid, high);

}

private static void merge(int[] array, int low, int mid, int high)

{

int leftArray[] = new int[mid - low + 1];

int rightArray[] = new int[high - mid];

for (int i = 0; i < leftArray.length; i++)

leftArray[i] = array[low + i];

for (int i = 0; i < rightArray.length; i++)

rightArray[i] = array[mid + i + 1];

int leftIndex = 0;

int rightIndex = 0;

for (int i = low; i < high + 1; i++)

{

if (leftIndex < leftArray.length && rightIndex < rightArray.length)

{

if (leftArray[leftIndex] >= rightArray[rightIndex])

{

array[i] = leftArray[leftIndex];

leftIndex++;

}

else

{

array[i] = rightArray[rightIndex];

rightIndex++;

}

}

else if (leftIndex < leftArray.length)

{

array[i] = leftArray[leftIndex];

leftIndex++;

}

else if (rightIndex < rightArray.length)

{

array[i] = rightArray[rightIndex];

rightIndex++;

}

}

}

public static void main (String args[]) throws FileNotFoundException{

PrintWriter writeToMyFile = new PrintWriter(new File("mergeout.txt"));

int max = 2147483647;

int min = 1;

int[] arr = new int[10000];

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

arr[i] = (int)(Math.random() \* (max - min + 1) + min);

}

long startTime = System.nanoTime();

mergeSort(arr, 0, arr.length - 1);

long endTime = System.nanoTime() - startTime;

System.out.println("Execution Time(nano seconds): "+endTime);

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

writeToMyFile.format("%d%n", arr[i]);

}

writeToMyFile.close();

}

}

Lab 6 - Hashing

import java.util.\*;

import java.io.\*;

public class Main

{

public static void main(String[] args) throws FileNotFoundException{

Hashtable<Integer, ArrayList<String>> hash = new Hashtable<Integer, ArrayList<String>>();

ArrayList<String> mod0 = new ArrayList<String>();

ArrayList<String> mod1 = new ArrayList<String>();

ArrayList<String> mod2 = new ArrayList<String>();

ArrayList<String> mod3 = new ArrayList<String>();

ArrayList<String> mod4 = new ArrayList<String>();

ArrayList<String> mod5 = new ArrayList<String>();

Scanner readMyFile = new Scanner(new File("source.txt"));

while(readMyFile.hasNext()){

String data = readMyFile.next();

int digitsum = 0;

int alphasum = 0;

for(int it = 0; it < data.length(); it++){

char ch = data.charAt(it);

if (Character.isAlphabetic(ch)) {

alphasum += (int) ch;

}

else if (Character.isDigit(ch)){

digitsum += (int) ch;

}

}

int hash\_value = ((alphasum + 2 \* digitsum) \* 17 +5)%6;

System.out.println("The Hash value of "+data+" is "+hash\_value);

if (hash\_value == 0){mod0.add(data);}

else if (hash\_value == 1){mod1.add(data);}

else if (hash\_value == 2){mod2.add(data);}

else if (hash\_value == 3){mod3.add(data);}

else if (hash\_value == 4){mod4.add(data);}

else if (hash\_value == 5){mod5.add(data);}

}

System.out.println("============================================================================================");

hash.put(0, mod0);

hash.put(1, mod1);

hash.put(2, mod2);

hash.put(3, mod3);

hash.put(4, mod4);

hash.put(5, mod5);

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

System.out.println("The subset of " + k +" : "+hash.get(k).toString());

}

readMyFile.close();

}

}

Lab 7 - BST

import java.util.\*;

import java.io.\*;

class BST\_class {

class Node {

String key;

String name;

float CGPA;

Node left, right;

public Node(String data, String n, float c){

key = data;

name = n;

CGPA = c;

left = right = null;

}

}

Node root;

BST\_class(){

root = null;

}

String minValue(Node root) {

String minval = root.key;

while (root.left != null) {

minval = root.left.key;

root = root.left;

}

return minval;

}

void insert(String key, String name, float c) {

root = insert\_Recursive(root, key, name, c);

}

Node insert\_Recursive(Node root, String key, String name, float c) {

if (root == null) {

root = new Node(key, name, c);

return root;

}

if (key.compareTo(root.key)<0)

root.left = insert\_Recursive(root.left, key, name, c);

else if (key.compareTo(root.key)>0)

root.right = insert\_Recursive(root.right, key, name , c);

return root;

}

void inorder() {

inorder\_Recursive(root);

}

void inorder\_Recursive(Node root) {

if (root != null) {

inorder\_Recursive(root.left);

System.out.print(root.key + " ");

System.out.print(root.name + " ");

System.out.println(root.CGPA + " ");

inorder\_Recursive(root.right);

}

}

void postorder() {

postorder\_Recursive(root);

}

void postorder\_Recursive(Node root) {

if (root != null) {

postorder\_Recursive(root.left);

postorder\_Recursive(root.right);

System.out.print(root.key + " ");

System.out.print(root.name + " ");

System.out.println(root.CGPA + " ");

}

}

void preorder() {

preorder\_Recursive(root);

}

void preorder\_Recursive(Node root) {

if (root != null) {

System.out.print(root.key + " ");

System.out.print(root.name + " ");

System.out.println(root.CGPA + " ");

preorder\_Recursive(root.left);

preorder\_Recursive(root.right);

}

}

}

class Main{

public static void main(String[] args) throws FileNotFoundException{

Scanner readMyFile = new Scanner(new File("source.txt"));

BST\_class bst = new BST\_class();

while(readMyFile.hasNext()){

String key = readMyFile.next();

String name = readMyFile.next();

float CGPA = readMyFile.nextFloat();

bst.insert(key, name, CGPA);

}

System.out.println("\n\nIn order traversal:");

bst.inorder();

System.out.println("\n\nPre order traversal:");

bst.preorder();

System.out.println("\n\nPost order traversal:");

bst.postorder();

}

}

Lab 8 - Min Heap

import java.util.\*;

import java.io.\*;

class Main {

private String[] Heap;

private int size;

private int maxsize;

private static final int FRONT = 1;

public Main(int maxsize)

{

this.maxsize = maxsize;

this.size = 0;

Heap = new String[this.maxsize + 1];

Heap[0] ="";

}

private int parent(int pos) { return pos / 2; }

private int leftChild(int pos) { return (2 \* pos); }

private int rightChild(int pos)

{

return (2 \* pos) + 1;

}

private boolean isLeaf(int pos)

{

if (pos > (size / 2) && pos <= size) {

return true;

}

return false;

}

private void swap(int fpos, int spos)

{

String tmp;

tmp = Heap[fpos];

Heap[fpos] = Heap[spos];

Heap[spos] = tmp;

}

private void minHeapify(int pos)

{

if (!isLeaf(pos)) {

if (Heap[pos].compareTo(Heap[leftChild(pos)]) > 0

|| Heap[pos].compareTo(Heap[rightChild(pos)]) > 0)

{

if (Heap[leftChild(pos)].compareTo(Heap[rightChild(pos)]) < 0) {

swap(pos, leftChild(pos));

minHeapify(leftChild(pos));

}

else {

swap(pos, rightChild(pos));

minHeapify(rightChild(pos));

}

}

}

}

public void insert(String element)

{

if (size >= maxsize) {

return;

}

Heap[++size] = element;

int current = size;

while (Heap[current].compareTo(Heap[parent(current)]) < 0) {

swap(current, parent(current));

current = parent(current);

}

}

public void printHeap()

{

for (int i = 1; i <= size; ++i)

System.out.print(Heap[i] + " ");

System.out.println();

}

public String giveTop()

{

return Heap[FRONT];

}

public String remove()

{

String popped = Heap[FRONT];

Heap[FRONT] = Heap[size--];

minHeapify(FRONT);

return popped;

}

public static void main(String[] args) throws FileNotFoundException

{

System.out.println("Step by step formation of min heap:");

Main minHeap = new Main(100);

Scanner readMyFile = new Scanner(new File("heapin.txt"));

String data = readMyFile.nextLine();

String[] nodes = data.split(", ");

for (String s: nodes) {

minHeap.insert(s);

minHeap.printHeap();

}

System.out.println("Heap sort implementation:");

for(int i = 0; i < nodes.length - 1 ; i++)

{

System.out.print(minHeap.remove() +" ");

}

System.out.println(minHeap.giveTop());

}

}

Lab 9 - Heap Sort

import java.util.\*;

import java.io.\*;

public class Main {

public void sort(String arr[])

{

int n = arr.length;

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (int i = n - 1; i > 0; i--) {

String temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

}

void heapify(String arr[], int n, int i)

{

int largest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && arr[l].compareTo(arr[largest]) > 0)

largest = l;

if (r < n && arr[r].compareTo(arr[largest]) > 0)

largest = r;

if (largest != i) {

String swap = arr[i];

arr[i] = arr[largest];

arr[largest] = swap;

printArray(arr);

heapify(arr, n, largest);

}

}

static void printArray(String arr[])

{

int n = arr.length;

for (int i = 0; i < n; ++i)

System.out.print(arr[i] + " ");

System.out.println();

}

public static void main(String[] args) throws FileNotFoundException

{

Scanner readMyFile = new Scanner(new File("hsortin.txt"));

String data = readMyFile.nextLine();

String[] arr = data.split(", ");

int n = arr.length;

Main ob = new Main();

System.out.println("Array before sorting:");

printArray(arr);

System.out.println("Steps of sorting:");

ob.sort(arr);

System.out.println("Array after sorting:");

printArray(arr);

}

}

Lab 10 - DFS (Directed)

import java.io.\*;

import java.util.\*;

class Main

{

static int N = 1000;

static ArrayList<ArrayList<Integer>> adj = new ArrayList<ArrayList<Integer>>();

static void dfsUtil(int u, int node, boolean visited[],

ArrayList<ArrayList<Integer>> road\_used,

int parent, int it)

{

int c = 0;

for (int i = 0; i < node; i++)

if (visited[i])

c++;

if (c == node)

return;

visited[u] = true;

road\_used.add(new ArrayList<Integer>(Arrays.asList( parent, u )));

System.out.print((char)(u+65) + " -> ");

for (int x : adj.get(u))

{

if (!visited[x])

{

dfsUtil(x, node, visited, road\_used, u, it + 1);

}

}

System.out.print("Backtracking\n");

if(u == 0){

System.out.print("\n\nNodes visited: ");

for (int ui = 0; ui<visited.length; ui++)

{

if ((visited[ui]) == true)

{

System.out.print((char)(ui+65) + " ");

}

}

System.exit(0);

}

for(int y = 0; y < road\_used.size(); y++)

{

if(road\_used.get(y).get(1) == u)

{

dfsUtil(road\_used.get(y).get(0), node,

visited,road\_used, u, it + 1);

}

}

}

static void dfs(int node)

{

boolean[] visited = new boolean[node];

ArrayList<ArrayList<Integer>> road\_used = new ArrayList<ArrayList<Integer>>();

for (int i = 0; i < node; i++)

{

visited[i] = false;

}

dfsUtil(0, node, visited, road\_used, -1, 0);

}

static void insertEdge(int u, int v)

{

adj.get(u).add(v);

}

public static void main(String[] args) throws FileNotFoundException

{

Scanner readMyFile = new Scanner(new File("graphin.txt"));

int node = readMyFile.nextInt();

int edge = readMyFile.nextInt();

for(int i = 0; i < N; i++)

{

adj.add(new ArrayList<Integer>());

}

while(readMyFile.hasNext()){

insertEdge((int)readMyFile.next().charAt(0) - 65, (int)readMyFile.next().charAt(0) - 65);

}

System.out.println("Nodes reachable from A(Steps with backtracing): ");

dfs(node);

readMyFile.close();

}

}

Lab 10 - DFS(Undirected)

import java.io.\*;

import java.util.\*;

class Main

{

static int N = 1000;

static ArrayList<ArrayList<Integer>> adj = new ArrayList<ArrayList<Integer>>();

static void dfsUtil(int u, int node, boolean visited[],

ArrayList<ArrayList<Integer>> road\_used,

int parent, int it)

{

int c = 0;

for (int i = 0; i < node; i++)

if (visited[i])

c++;

if (c == node)

return;

visited[u] = true;

road\_used.add(new ArrayList<Integer>(Arrays.asList( parent, u )));

System.out.print((char)(u+65) + " -> ");

for (int x : adj.get(u))

{

if (!visited[x])

{

dfsUtil(x, node, visited, road\_used, u, it + 1);

}

}

System.out.print("Backtracking\n");

for(int y = 0; y < road\_used.size(); y++)

{

if(road\_used.get(y).get(1) == u)

{

dfsUtil(road\_used.get(y).get(0), node,

visited,road\_used, u, it + 1);

}

}

}

static void dfs(int node)

{

boolean[] visited = new boolean[node];

ArrayList<ArrayList<Integer>> road\_used = new ArrayList<ArrayList<Integer>>();

for (int i = 0; i < node; i++)

{

visited[i] = false;

}

dfsUtil(0, node, visited, road\_used, -1, 0);

}

static void insertEdge(int u, int v)

{

adj.get(u).add(v);

adj.get(v).add(u);

}

public static void main(String[] args) throws FileNotFoundException

{

Scanner readMyFile = new Scanner(new File("graphin.txt"));

int node = readMyFile.nextInt();

int edge = readMyFile.nextInt();

for(int i = 0; i < N; i++)

{

adj.add(new ArrayList<Integer>());

}

while(readMyFile.hasNext()){

insertEdge((int)readMyFile.next().charAt(0) - 65, (int)readMyFile.next().charAt(0) - 65);

}

System.out.println("Nodes reachable from A(Steps with backtracing): ");

dfs(node);

}

}