# Week 1

1. You are provided with two sorted linked list. Merge two linked list in increasing order without any repeating values.

Input 1, 2, 4 and 3, 4, 5

1 2 4 and 3 4 5

Output [1, 2, 3, 4, 5]

***Answer:***

class Node

{

public int data;

public Node next;

}

public class GlobalMembers

{

public static Node getNode(int data)

{

// allocate space

Node temp = new Node();

// put in data

temp.data = data;

temp.next = null;

return temp;

}

// function to merge two sorted linked list

// in a sorted manner

public static Node sortedMerge(Node a, Node b)

{

Node result = null;

/\* Base cases \*/

if (a == null)

{

return (b);

}

else if (b == null)

{

return (a);

}

/\* Pick either a or b, and recur \*/

if (a.data <= b.data)

{

result = a;

result.next = sortedMerge(a.next, b);

}

else

{

result = b;

result.next = sortedMerge(a, b.next);

}

return (result);

}

/\* The function removes duplicates from a sorted list \*/

public static void removeDuplicates(Node head)

{

/\* Pointer to traverse the linked list \*/

Node current = head;

/\* Pointer to store the next pointer of a node to be deleted\*/

Node next\_next;

/\* do nothing if the list is empty \*/

if (current == null)

{

return;

}

/\* Traverse the list till last node \*/

while (current.next != null)

{

/\* Compare current node with next node \*/

if (current.data == current.next.data)

{

/\* The sequence of steps is important\*/

next\_next = current.next.next;

current.next = null;

current.next = next\_next;

}

else // This is tricky: only advance if no deletion

{

current = current.next;

}

}

}

// function to merge two sorted linked list

// without duplicates

public static Node sortedMergeWithoutDuplicates(Node head1, Node head2)

{

// merge two linked list in sorted manner

Node head = sortedMerge(head1, head2);

// remove duplicates from the list 'head'

removeDuplicates(head);

return head;

}

// function to print the linked list

public static void printList(Node head)

{

while (head != null)

{

System.out.print(head.data);

System.out.print(" ");

head = head.next;

}

}

// Driver program to test above

public static void main(String[] args)

{

// head1: 1->1->4->5->7

Node head1 = getNode(1);

head1.next = getNode(2);

head1.next.next = getNode(4);

// head2: 2->4->7->9

Node head2 = getNode(3);

head2.next = getNode(4);

head2.next.next = getNode(5);

Node head3;

head3 = sortedMergeWithoutDuplicates(head1, head2);

printList(head3);}}

Graphical user interface

Description automatically generated

Figure : Output of week 1A

# Week 2

# Week 3

# Week 4

Given 2D matrix of 1 and 0s. Find maximum area of square made by 0s.

INPUT: 1 0 1 0 0

0 1 1 1 1

0 0 0 0 1

0 0 0 1 1

0 1 0 1 1

OUTPUT: 4

***Answer:***

public class MAxSquare

{

public static int findMaxSquare(int [][] matrix)

{

int rows = matrix.length , cols=rows>0? matrix[0].length:0;

int[] dp = new int[cols+1];

int maxsqlen =0, prev=0;

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

{

for(int j = 1; j <= cols; j++)

{

int temp = dp [j];

if (matrix [i-1][j-1]==0)

{

dp [j]=Math.min(Math.min(dp[j-1], prev), dp[j])+1;

maxsqlen=Math.max(maxsqlen, dp[j]);

}

else

{

dp[j]=0;

}

prev=temp;

}

}

return maxsqlen \* maxsqlen;

}

// Driver program

public static void main(String[] args)

{

int mat[][] = {{1, 0, 1, 0, 0},

{0, 1, 1, 1, 1},

{0, 0, 0, 0, 1},

{0, 0, 0, 1, 1},

{0, 1, 0, 1, 1}};

System.out.println("answer is" + " " + findMaxSquare(mat));

}

}

Graphical user interface, text

Description automatically generated

Figure : Output and code of week 4

# Week 5

Using stack evaluate following expression.

Input: (5(3+2)-7)/2

Output: 9

***Answer:***

import java.util.Stack;

public class EvaluateString

{

public static int evaluate(String expression)

{

char[] tokens = expression.toCharArray();

Stack<Integer> values = new Stack<Integer>();

Stack<Character> ops = new Stack<Character>();

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

{

//skip it current token because nothing

if (tokens[i] == ' ')

continue;

// Current token is a number, push it to stack for numbers

if (tokens[i] >= '0' && tokens[i] <= '9')

{

StringBuffer sbuf = new StringBuffer();

// more than one digits in number

while (i < tokens.length && tokens[i] >= '0' && tokens[i] <= '9')

sbuf.append(tokens[i++]);

values.push(Integer.parseInt(sbuf.toString()));

}

// Current token is an opening brace, push it to 'ops'

else if (tokens[i] == '(')

ops.push(tokens[i]);

// Closing brace encountered, solve entire brace

else if (tokens[i] == ')')

{

while (ops.peek() != '(')

values.push(applyOp(ops.pop(), values.pop(), values.pop()));

ops.pop();

}

// Current token is an operator.

else if (tokens[i] == '+' || tokens[i] == '-' ||

tokens[i] == '\*' || tokens[i] == '/')

{

while (!ops.empty() && hasPrecedence(tokens[i], ops.peek()))

values.push(applyOp(ops.pop(), values.pop(), values.pop()));

ops.push(tokens[i]);

}

}

while (!ops.empty())

values.push(applyOp(ops.pop(), values.pop(), values.pop()));

return values.pop();

}

// Returns true if 'op2' has higher or same precedence as 'op1’ otherwise returns false.

public static boolean hasPrecedence(char op1, char op2)

{

if (op2 == '(' || op2 == ')')

return false;

if ((op1 == '\*' || op1 == '/') && (op2 == '+' || op2 == '-'))

return false;

else

return true;

}

//Return the result.

public static int applyOp(char op, int b, int a)

{

switch (op)

{

case '+':

return a + b;

case '-':

return a - b;

case '\*':

return a \* b;

case '/':

if (b == 0)

throw new

UnsupportedOperationException("Cannot divide by zero");

return a / b;

}

return 0;

}

// using above methods

public static void main(String[] args)

{

int a=(EvaluateString.evaluate("(5 \* ( 3 + 2 )-7 ) / 2"));

System.out.println("The result of (5(3+2)-7)/2 is:\t"+ a); }}

Graphical user interface, text

Description automatically generated

Figure : Output with code

A screenshot of a computer

Description automatically generated

Figure : solved code of week 5

# Week 6

Provided sorted linked list find the pair in the array whose product is closest to given integer n

Input: {2, 3, 5, 7, 100, 9, 18}

n=65

Output: 9 and 7 (the product of 9 and 7 is 63 and is closest to provided integer 65)

***Answer:***

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Main {

// Prints the pair with sum cloest to x

static void printClosest(int arr[], int n, int x)

{

int res\_l=0, res\_r=0; // To store indexes of result pair

// Initialize left and right indexes and difference between

// pair sum and x

int l = 0, r = n-1, diff = Integer.MAX\_VALUE;

// While there are elements between l and r

while (r > l)

{

// Check if this pair is closer than the closest pair so far

if (Math.abs(arr[l] \* arr[r] - x) < diff)

{

res\_l = l;

res\_r = r;

diff = Math.abs(arr[l] \* arr[r] - x);

}

// If this pair has more sum, move to smaller values.

if (arr[l] \* arr[r] > x)

r--;

else // Move to larger values

l++;

}

System.out.println(" The closest pair is "+arr[res\_l]+" and "+ arr[res\_r]);

}

// Driver program to test above function

public static void main(String[] args)

{

int arr[] = {2,3,5,7,100,9,18}, x = 63;

int n = arr.length;

printClosest(arr, n, x);

}

}

**Graphical user interface, text

Description automatically generated**

Figure : Result of code

Graphical user interface, text, application

Description automatically generated

Figure : code - week 6

# Week 7

You are given 2D matrix of 0 and 1 in sorted order, return index of the row which contains 2nd maximum no of 1s. (Note: index starting from 0).

Input:

0 0 0 1

1 1 1 1

0 0 1 1

0 1 1 1

Output: 3 (index)

***Answer:***

import java.io.\*;

public class GFG {

static int R = 4, C = 4;

static int first(int arr[], int low, int high)

{

if (high >= low) {

int mid = low + (high - low) / 2;

if ((mid == 0 || (arr[mid - 1] == 0)) && arr[mid] == 1)

return mid;

else if (arr[mid] == 0)

return first(arr, (mid + 1), high);

else

return first(arr, low, (mid - 1));

}

return -1;

}

// Function that returns index of row

// with maximum number of 1s.

static int rowWithMax1s(int mat[][])

{

// Initialize max values

int max\_row\_index = 0, max = -1;

int i, index;

for (i = 0; i < R; i++) {

index = first(mat[i], 0, C - 1);

if (index != 0 && C - index > max) {

max = C - index;

max\_row\_index = i;

}

}

return max\_row\_index;

}

// Driver Code

public static void main(String[] args)

{

int mat[][] = { { 0, 0, 0, 1 },

{ 1, 1, 1, 1 },

{ 0, 0, 1, 1 },

{ 0, 1, 1, 1 } };

System.out.println("Index of row with maximum 1s is "

+ rowWithMax1s(mat)); }}

Graphical user interface

Description automatically generated

Figure : Code and output of week7B