|  |  |  |
| --- | --- | --- |
| Sukkur_IBA_New_Logo | **Sukkur IBA University**  **Department of Computer Science** | **C:\Users\Saif Hassan\Downloads\CS logo (3).jpg** |

**DATA STRUCTURES**

**Lab01 – Arrays, LinkesLists**

**Instructor: Saif Hassan**

**READ IT FIRST**

Prior to start solving the problems in this assignments, please give full concentration on following points.

1. WORKING – This is individual lab. If you are stuck in a problem contact your teacher, but, in mean time start doing next question (don’t waste time).
2. DEADLINE – 11th March, 2022
3. SUBMISSION – This assignment needs to be submitted in a soft copy.
4. WHERE TO SUBMIT – Please visit your LMS.
5. WHAT TO SUBMIT – Submit this docx and pdf file.

**KEEP IT WITH YOU!**

1. Indent your code inside the classes and functions. It’s a good practice!
2. It is not bad if you keep your code indented inside the loops, if and else blocks as well.
3. Comment your code, where it is necessary.
4. Read the entire question. Don’t jump to the formula directly.

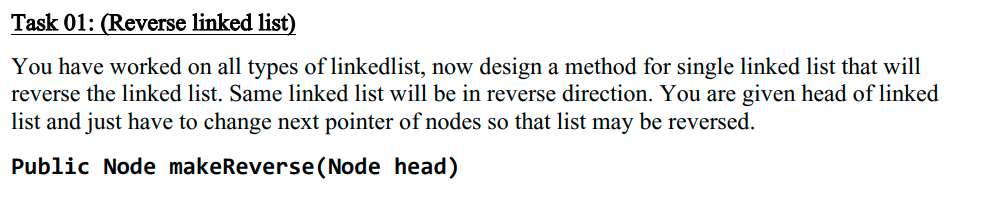
I, **Amjad Ali** with student ID \_**191-21-0001**\_

Section \_**A**\_hereby declare that I do understand the instructions above and follow them. This is

my own work.

**Exercises**

**Task1 Description**



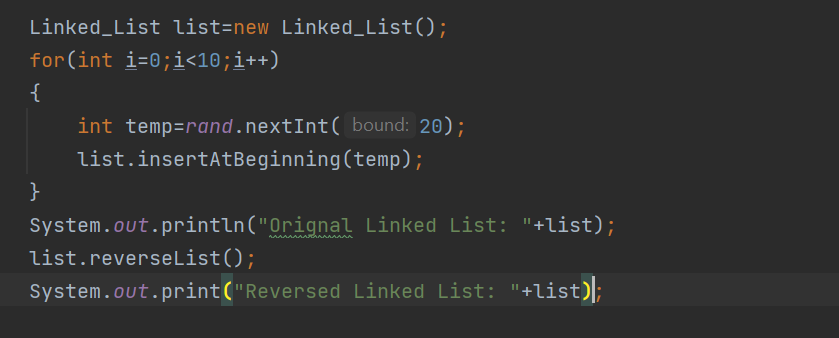
Solution:

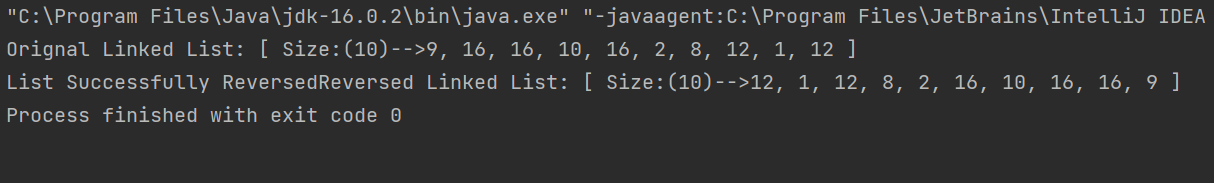
**Code**

1. public void reverseList(){
2. if(Head==null)
3. System.out.println("List is Empty");
4. else{
5. Node current=Head;
6. Node previous=null;
7. while(current.next!=null)
8. {
9. Node temp=current.next;
10. current.next=previous;
11. previous=current;
12. current=temp;
13. }
14. current.next=previous;
15. Head=current;
16. System.out.print("List Successfully Reversed");

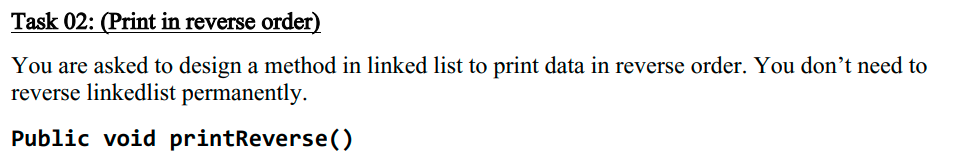
19. }
21. }

**Sample Input:**

**Sample Output**



**Task2 Description**

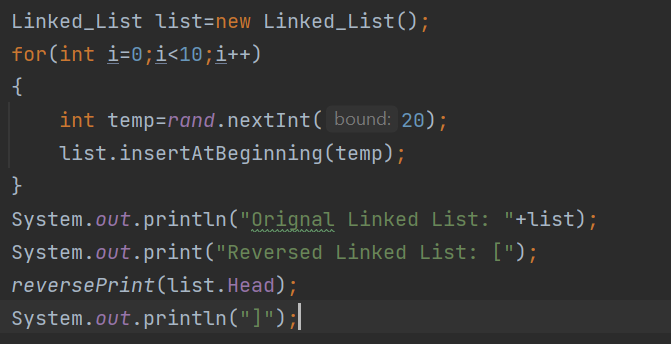


Solution:

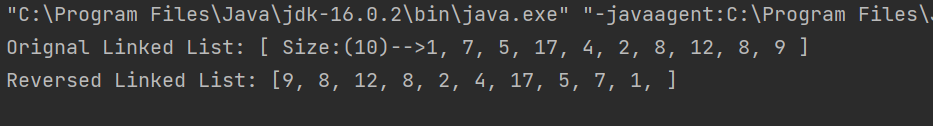
Code

1. public static void reversePrint(Node current)
2. {
3. if(current.next!=null)
4. {
5. reversePrint(current.next);
6. }
7. System.out.print(current.data+", ");
8. }

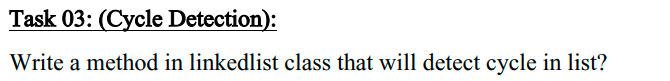
**Sample Input:**



**Sample Output**



**Task3 Description**

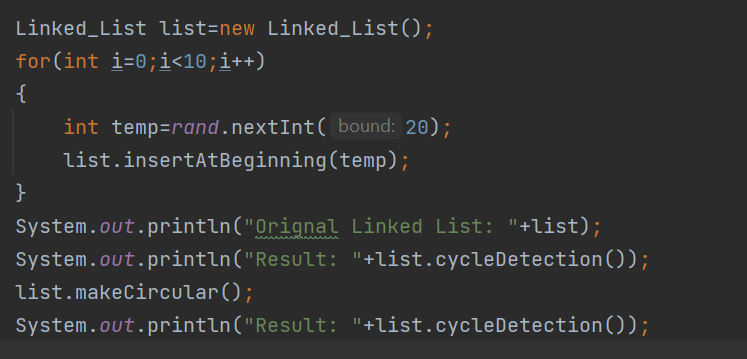


Solution:

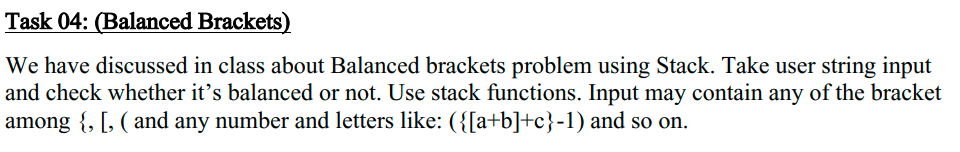
**Code**

1. public boolean cycleDetection()
2. {
3. Set<Node> set=new HashSet<>();
4. int size=size();
5. Node current=Head;
6. while(current!=null)
7. {
8. if(!set.add(current))
9. {
10. return true;
11. }
12. current=current.next;
13. }
15. return false;
16. }

**Sample Input:**

**Sample Output**

**Task4 Description**



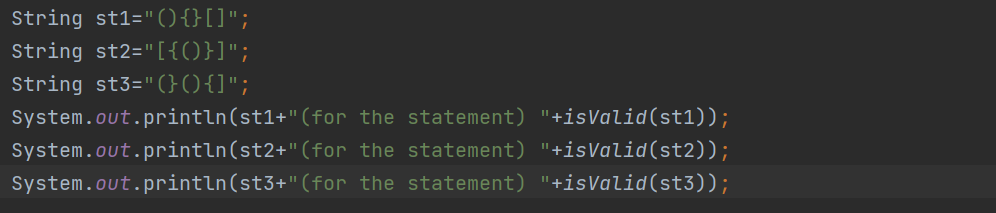
Solution:

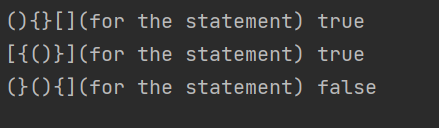
Code

1. public boolean isValid(String s) {
2. Stack<Character> arr=new Stack<>();
3. int length=s.length();
4. for(int i=0;i<length;i++)
5. {
6. if(s.charAt(i)=='[' || s.charAt(i)=='(' ||s.charAt(i)=='{' ||s.charAt(i)=='<')
7. arr.push(s.charAt(i));
8. if(s.charAt(i)==']')
9. {
10. if(arr.size()==0 || arr.pop()!='[')
11. return false;
12. }
13. if(s.charAt(i)==')' )
14. {
15. if(arr.size()==0 || arr.pop()!='(')
16. return false;
18. }
19. if(s.charAt(i)=='}')
20. {
21. if(arr.size()==0 ||arr.pop()!='{')
22. return false;
23. }
24. if(s.charAt(i)=='>')
25. {
26. if(arr.size()==0 ||arr.pop()!='<')
27. return false;
28. }

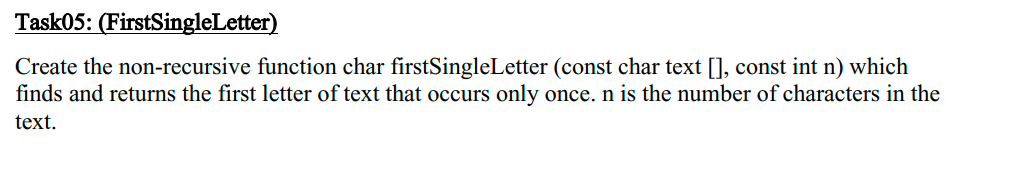
31. }
32. if(arr.size()==0)
33. return true;
34. else
35. return false;
37. }

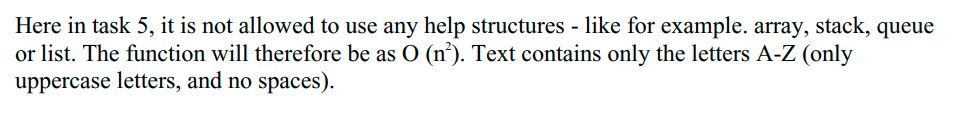
**Sample Input:**

**Sample Output**



**Task5 Description**

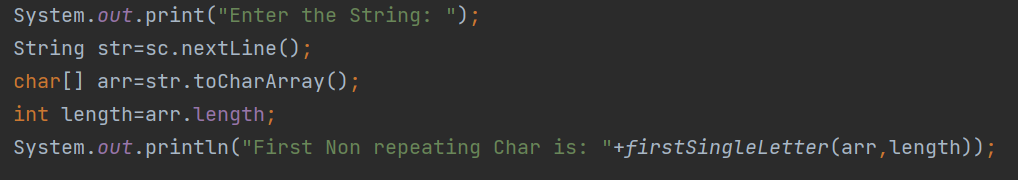


Solution:

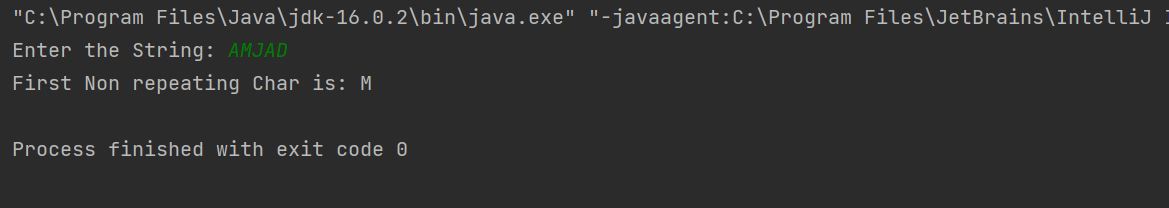
**Code:**

1. public static char firstSingleLetter(char text[], int n) {
2. boolean cond = true;
3. for (int i = 0; i < n; i++) {
4. for (int j = i + 1; j < n; j++) {
5. if (text[i] == text[j]) {
6. cond = false;
7. }
8. }
10. if (cond) {
11. return text[i];
12. }
13. cond = true;
14. }
15. System.out.println("No such Character in Text");
16. return '1';
17. }

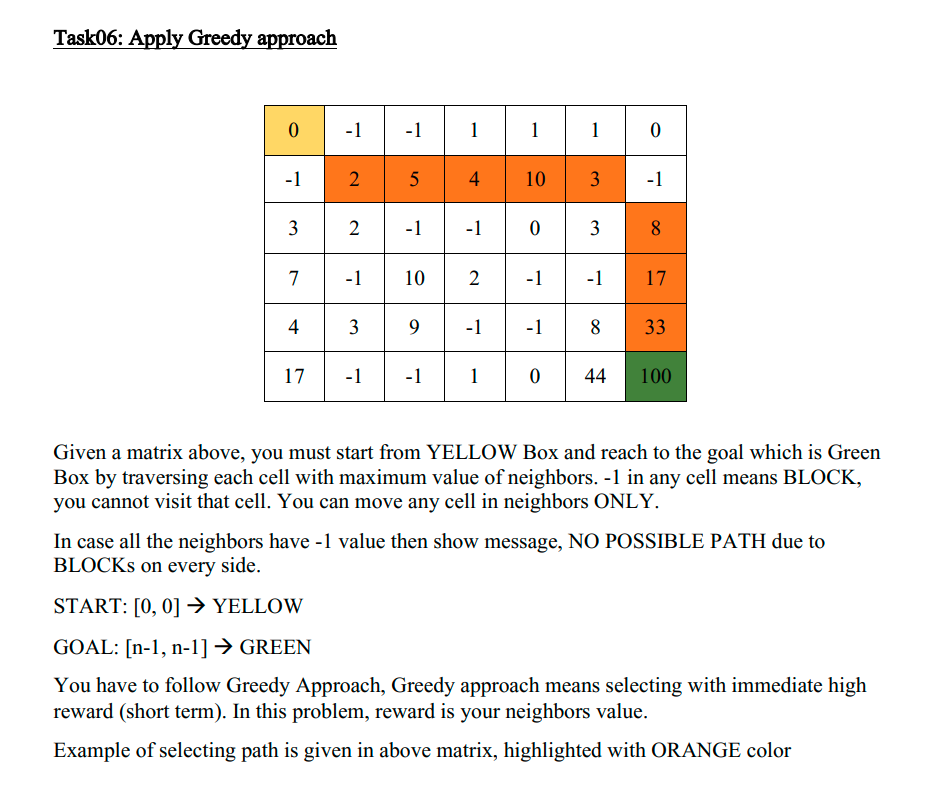
**Sample Input:**

**Sample Output**



**Task06 Description**



**Solution:**

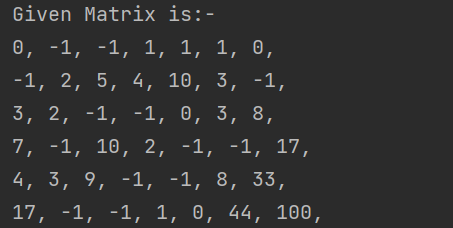
**Code**:

1. package com.company;
3. import java.util.\*;
4. import java.util.List;
6. public class Greedy {
7. static Random rand = new Random();
9. public static int[][] makeMatrix(int row, int col) {
10. int[][] matrix = new int[row][col];
11. for (int i = 0; i < row; i++) {
12. for (int j = 0; j < col; j++) {
13. int random = rand.nextInt(90);
14. if (i == 0 && j == 0) {
15. matrix[i][j] = 0;
16. } else if (i == (row - 1) && j == (col - 1)) {
17. matrix[i][j] = 100;
18. } else {
19. matrix[i][j] = random;
20. }
21. }
22. }
24. for (int i = 0; i < 18; i++) {
25. int r = rand.nextInt(row);
26. int c = rand.nextInt(col);
27. if ((r != 0 && r != (row - 1)) && (c != 0 && c != (col - 1))) {
28. matrix[r][c] = -1;
29. }
30. }
32. return matrix;
33. }
35. public static void printMatrix(int[][] matrix) {
36. for (int i = 0; i < matrix.length; i++) {
37. for (int j = 0; j < matrix[0].length; j++) {
38. System.out.print(matrix[i][j] + ", ");
39. }
40. System.out.println();
41. }
43. }
45. public static int[] findGreatest(int[][] matrix, int[] ind) {
46. int[][] point = new int[8][3];
47. for (int i = 0; i < 8; i++)
48. point[i][0] = -1;
50. try {
51. //1st
52. point[0][0] = matrix[ind[0] + 1][ind[1] + 1];
53. point[0][1] = ind[0] + 1;
54. point[0][2] = ind[1] + 1;
55. } catch (Exception e) {
57. }
58. try {
59. //2nd
60. point[1][0] = matrix[ind[0]][ind[1] + 1];
61. point[1][1] = ind[0];
62. point[1][2] = ind[1] + 1;
63. } catch (Exception e) {
65. }
66. try {
67. //3rd
68. point[2][0] = matrix[ind[0] + 1][ind[1]];
69. point[2][1] = ind[0] + 1;
70. point[2][2] = ind[1];
71. } catch (Exception e) {
73. }
74. try {
75. //4th
76. point[3][0] = matrix[ind[0] + 1][ind[1] - 1];
77. point[3][1] = ind[0] + 1;
78. point[3][2] = ind[1] - 1;
79. } catch (Exception e) {
81. }
82. try {
83. //5th
84. point[4][0] = matrix[ind[0] - 1][ind[1] + 1];
85. point[4][1] = ind[0] - 1;
86. point[4][2] = ind[1] + 1;
87. } catch (Exception e) {
89. }
90. try {
91. //6th
92. point[5][0] = matrix[ind[0] - 1][ind[1]];
93. point[5][1] = ind[0] - 1;
94. point[5][2] = ind[1];
95. } catch (Exception e) {
97. }
98. try {
99. //7th
100. point[6][0] = matrix[ind[0]][ind[1] - 1];
101. point[6][1] = ind[0];
102. point[6][2] = ind[1] - 1;
103. } catch (Exception e) {
105. }
106. try {
107. //8th
108. point[7][0] = matrix[ind[0] - 1][ind[1] - 1];
109. point[7][1] = ind[0] - 1;
110. point[7][2] = ind[1] - 1;
111. } catch (Exception e) {
113. }
115. int bigind = 0;
116. int biggest = point[0][0];
117. for (int i = 0; i < 8; i++) {
119. if (point[i][0] > biggest) {
120. bigind = i;
121. biggest = point[i][0];
123. }
125. }
126. matrix[point[bigind][1]][point[bigind][2]] = -1;
127. int[] arr = point[bigind];
128. return arr;
130. }
132. public static List<int[]> findPath(int[][] matrix) {
134. int result = 0;
135. List<int[]> path = new ArrayList<>();
136. int[] ind = {0, 0};
137. while (result != 100) {
138. int[] values = findGreatest(matrix, ind);
139. result = values[0];
140. ind[0] = values[1];
141. ind[1] = values[2];
142. if (result == -1) {
144. break;
145. }
147. path.add(values);
149. }
151. return path;
152. }
154. public static boolean pathcheck(List<int[]> list) {
155. int size = list.size();
156. int[] check = list.get((size - 1));
157. if (check[0] != 100)
158. return false;
159. else
160. return true;
161. }
163. public static void printPath(int[][] matrix) {
164. List<int[]> path = findPath(matrix);
165. boolean result = pathcheck(path);
166. if (result) {
167. int size = path.size();
168. System.out.println("POSSIBLE PATH");
169. for (int i = 0; i < size; i++) {
170. int arr[] = path.get(i);
171. System.out.println((i + 1) + " : Point value =" + arr[0] + " Index( " + arr[1] + ", " + arr[2] + " )");
172. }
174. } else {
175. System.out.println(" NO POSSIBLE PATH due to BLOCKs");
176. }

179. }
181. public static void main(String[] args) {
182. // Scanner sc=new Scanner(System.in);
183. // System.out.print("Enter the rows: ");
184. // int row=sc.nextInt();
185. // System.out.print("Enter the colomns: ");
186. // int col=sc.nextInt();
187. // int[][] matrix=makeMatrix(row,col);
188. int[][] matrix = {{0, -1, -1, 1, 1, 1, 0},
189. {-1, 2, 5, 4, 10, 3, -1},
190. {3, 2, -1, -1, 0, 3, 8},
191. {7, -1, 10, 2, -1, -1, 17},
192. {4, 3, 9, -1, -1, 8, 33},
193. {17, -1, -1, 1, 0, 44, 100}};
194. System.out.println("Given Matrix is:-");
195. printMatrix(matrix);
196. printPath(matrix);

199. }
200. }

**Sample Input:**



**Sample Output**

