Nama : Andyan Yogawardhana

NIM : 21/482180/PA/21030

Kelas : KOMB1

Tugas 5 – Graph

Source Code

1. package Adjacency;
2. import java.util.LinkedList;
3. import java.util.Queue;
4. import java.util.Stack;
5. public class AdjacencyMatrix {
6. public static void main(String[] args) {
7. AdjMatrix graph = new AdjMatrix(6);
8. graph.addEdge(0, 1);
9. graph.addEdge(0, 2);
10. graph.addEdge(1, 3);
11. graph.addEdge(2, 3);
12. graph.addEdge(2, 4);
13. graph.addEdge(3, 5);
14. graph.addEdge(4, 5);
15. graph.printGraph();
16. graph.bfs(1);
17. graph.dfs(1);
18. graph.bfs(2);
19. graph.dfs(2);
20. graph.bfs(3);
21. graph.dfs(3);
22. graph.bfs(4);
23. graph.dfs(4);
24. graph.bfs(5);
25. graph.dfs(5);
26. }
27. }
28. *// 1. Implementasi Adjacency Matrix*
29. class AdjMatrix {
30. private int vertices;
31. private int[][] matrix;
32. *// constructor dengan parameter jumlah vertex dalam graph*
33. public AdjMatrix(int v) {
34. this.vertices = v;
35. *// matrix 2 dimensi untuk menampung edge (0 jika tidak berhubungan, 1 jika berhubungan)*
36. matrix = new int[v][v];
37. }
38. public void addEdge(int from, int to) {
39. *// mengatur edge antara vertex "from" dan vertex "to" menjadi 1 karena telah terhubung*
40. matrix[from][to] = 1;
41. matrix[to][from] = 1;
42. }
43. public void printGraph() {
44. System.out.println("    Matrices Adjacency");   *// judul tabel*
45. System.out.print("  | ");
46. for(int j = 0; j < vertices; j++) {
47. System.out.print(j + "   ");    *// judul kolom*
48. }
49. System.out.println();
50. System.out.print("--+");
51. for(int j = 0; j < vertices; j++) {
52. System.out.print("----");   *// batas tabel*
53. }
54. System.out.println();
55. for(int i = 0; i < vertices; i++) {
56. System.out.print(i + " | ");    *// judul baris*
57. for(int k = 0; k < vertices; k++) {
58. System.out.print(matrix[i][k] + "   ");     *// nilai adjacency antara vertex kolom dan vertex baris*
59. }
60. System.out.println();
61. }
62. System.out.println();
63. }
65. *// 2. BFS versi Adjacency Matrix*
66. public void bfs(int start) {
67. boolean visited[] = new boolean[vertices];  *// menampung visited value suatu vertex*
68. Queue<Integer> queue = new LinkedList<>();
69. *// mengubah value pada vertex awal menjadi true karena pertama kali dikunjungi*
70. visited[start] = true;
71. *// memasukkan vertex awal ke queue untuk dijadikan head bfs*
72. queue.add(start);
73. System.out.print("BFS (Head = " + start + ") : ");
74. while(queue.size() != 0) {
75. *// mengubah nilai variabel start menjadi nilai index pertama queue lalu melakukan dequeue*
76. start = queue.poll();
77. *// print vertex dari variabel start*
78. System.out.print(start + " ");
79. *// melakukan perulangan sebanyak jumlah vertex kali untuk mencari rute bfs*
80. for(int i = 0; i < vertices; i++) {
81. *// jika vertex ke-i belum dikunjungi dan terdapat edge antara matrix "start" dan "i"*
82. if(!visited[i] && matrix[start][i] == 1) {
83. visited[i] = true;  *// kunjungi dan ubah nilai visited matrix i*
84. queue.add(i);       *// masukkan vertex i ke queue*
85. }
86. }   *// perulangan dilanjutkan hingga semua vertex selesai dikunjungi*
87. }
88. System.out.println();
89. }
90. *// 3. Method DFS*
91. public void dfs(int start) {
92. boolean visited[] = new boolean[vertices];  *// menampung visited value suatu vertex*
93. Stack<Integer> stack = new Stack<>();
94. *// mengubah value pada vertex awal menjadi true karena pertama kali dikunjungi*
95. visited[start] = true;
96. *// memasukkan vertex awal ke stack untuk dijadikan head bfs*
97. stack.push(start);
98. System.out.print("DFS (Head = " + start + ") : ");
99. while(stack.size() != 0) {
100. *// mengubah nilai variabel start menjadi nilai index teratas dari stack lalu mengeluarkan nilai tersebut dari stack*
101. start = stack.pop();
102. *// print vertex dari variabel start*
103. System.out.print(start + " ");
104. *// melakukan perulangan sebanyak jumlah vertex kali untuk mencari rute dfs*
105. *// perulangan dilakukan dari index terbesar agar urutan dfs sesuai dengan besar valuenya*
106. for(int i = vertices - 1; i >= 0; i--) {
107. *// jika vertex ke-i belum dikunjungi dan terdapat edge antara matrix "start" dan "i"*
108. if(!visited[i] && matrix[start][i] == 1) {
109. visited[i] = true;  *// kunjungi dan ubah nilai visited matrix i*
110. stack.add(i);       *// masukkan vertex i ke stack*
111. }
112. }   *// perulangan dilanjutkan hingga semua vertex selesai dikunjungi*
113. }
114. System.out.println();
115. }
116. }

Output Terminal

