**Lab 10 Report**

**CS303L-L3 Algorithms and Data Structures**

**Sam Lazrak**

**Spring Semester 2018**

**Objectives:**

* Implement Breadth First Search algorithm on graphs.
* Compare the performances of BFS using adjacency lists and adjacency matrix

**In-class Assignments:**

1. The Graph class has been uploaded in Canvas. It implements adjacency lists to store the neighbors of a vertex. Extend the graph class to make two new subclasses DirectedGraph and UndirectedGraph. Hint: Override the addEdge() method.
2. Write a driver program, which reads input files mediumG.txt as an undirected graph and reads an input file tinyDG.txt as a directed graph. This driver program should display the graphs in the form of adjacency lists.
3. Implement BFS algorithm on an undirected graph following the pseudo-code given below. Read the file mediumG.txt as the input graph. Print the BFS paths from a source to all the other nodes in the graph.

The following source code was written:

GRAPH :

public class Graph {

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

BufferedReader read = new BufferedReader(new FileReader("tinyDG.txt"));

DirectedGraph directed = new DirectedGraph(read);

System.out.println(directed.tostring());

BufferedReader r = new BufferedReader(new FileReader("mediumG.txt"));

UndirectedGraph undirected = new UndirectedGraph(r);

System.out.println(undirected.tostring());

BreadthFirstSearch bfs = new BreadthFirstSearch();

System.out.println("BFS: " + "\n");

bfs.BFS(undirected, 0);

}

public int V;

public int E;

public LinkedList<Integer>[] adj;

public Graph()

{

V = 0;

E = 0;

}

public Graph(BufferedReader reader) throws IOException

{

String line;

line = reader.readLine();

V = Integer.parseInt(line);

line = reader.readLine();

E = Integer.parseInt(line);

adj = new LinkedList[V];

for (int v = 0; v < V; v++) {

adj[v] = new LinkedList<Integer>();

}

while ((line = reader.readLine()) != null) {

int tempV1, tempV2;

StringTokenizer st = new StringTokenizer(line, " ");

tempV1 = Integer.parseInt(st.nextToken());

tempV2 = Integer.parseInt(st.nextToken());

addEdge(tempV1, tempV2);

}

}

public void addEdge(int v, int w) {

//override

}

public String tostring()

{

String s = new String();

s = "There are "+V+" vertices and "+E+" edges\n";

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

{

s = s+i+": ";

for(int j = 0; j<adj[i].size();j++)

{

s = s+adj[i].get(j)+" ";

}

s = s+"\n";

}

return s;

}

}

UNDIRECTED GRAPH :

public class UndirectedGraph extends Graph {

public UndirectedGraph(){

super();

}

public UndirectedGraph(BufferedReader reader) throws IOException {

super(reader);

}

public void addEdge(int v, int w) {

super.adj[v].add(w);

super.adj[w].add(v);

}

}

DIRECTED GRAPH :

public class DirectedGraph extends Graph {

public DirectedGraph(){

super();

}

public DirectedGraph(BufferedReader reader) throws IOException {

super(reader);

}

public void addEdge(int v, int w) {

super.adj[v].add(w);

}

}

BREADTH FIRST SEARCH :

public class BreadthFirstSearch {

public void BFS(Graph G, int s){

// Mark all the vertices as not visited(By default

// set as false)

boolean visited[] = new boolean[G.V];

int counter = 0;

// Create a queue for BFS

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

// Mark the current node as visited and enqueue it

visited[s]=true;

queue.add(s);

while (queue.size() != 0)

{

counter++;

// Dequeue a vertex from queue and print it

s = queue.poll();

System.out.print(s+" --> ");

if(counter%10 == 0) {

System.out.println();

}

// Get all adjacent vertices of the dequeued vertex s

// If a adjacent has not been visited, then mark it

// visited and enqueue it

Iterator<Integer> i = G.adj[s].listIterator();

while (i.hasNext())

{

int n = i.next();

if (!visited[n])

{

visited[n] = true;

queue.add(n);

}

}

}

}

}

The output was:

There are 13 vertices and 22 edges

0: 1 5

1:

2: 3 0

3: 2 5

4: 2 3

5: 4

6: 0 8 4 9

7: 9 6

8: 6

9: 10 11

10: 12

11: 12 4

12: 9

There are 250 vertices and 1273 edges

0: 15 24 44 49 58 59 68 80 97 114 149 160 163 176 191 202 204 209 211 222 225

1: 72 107 130 150 164 189 194 200 203 220

2: 14 18 42 51 79 86 108 110 141

3: 37 45 67 76 115 153 228 241

4: 5 26 55 77 78 112 128 138 159 239 240

5: 26 32 55 67 77 102 104 217 226 4

6: 16 54 98 99 117 129 140 147 166 178 236

7: 42 57 65 71 101 125 148 157 181 184 188 197 230

8: 11 30 43 82 85 143 152 179 207 210 212 221 244 246

9: 23 33 58 68 114 142 195

10: 105 106 123 175 246

11: 30 43 82 85 143 152 175 207 212 244 246 8

12: 28 35 36 41 88 94 113 121 170 182 198 242

13: 19 100 103 129 133 162 174 192

14: 18 51 86 129 133 166 2

15: 24 39 49 58 66 80 114 149 163 202 204 209 211 222 225 0

16: 54 98 99 117 129 140 147 166 178 236 6

17: 41 81 121 134 158 170 182 223 229

18: 35 51 86 94 141 14 2

19: 70 79 84 100 103 174 179 192 243 13

20: 40 75 89 116 127 164 190 194 220 247

21: 27 62 65 71 138 184 188 230 233 240

22: 34 53 56 73 120 145

23: 33 58 68 114 176 195 222 9

24: 39 66 80 114 149 163 206 209 211 222 225 15 0

25: 60 63 96 111 199

26: 55 77 78 102 138 217 226 239 240 5 4

27: 62 65 71 138 184 188 230 233 240 21

28: 35 41 94 113 121 170 182 198 223 242 12

29: 47 64 91 109 137 146 167 218 224 227

30: 43 70 79 82 143 152 156 179 207 210 212 214 219 221 244 11 8

31: 37 115 153 228 241

32: 52 77 93 102 104 144 151 160 168 185 187 201 208 226 231 248 5

33: 58 114 163 222 23 9

34: 53 56 73 120 145 22

35: 36 41 88 94 141 198 28 18 12

36: 41 88 98 182 35 12

37: 76 95 115 153 228 241 31 3

38: 74 109 126 183 215

39: 66 80 149 206 209 211 24 15

40: 75 89 116 150 164 190 194 220 247 20

41: 81 88 121 170 182 198 36 35 28 17 12

42: 86 101 108 135 141 157 181 196 7 2

43: 82 152 156 207 210 212 219 221 244 30 11 8

44: 49 59 68 80 93 97 144 160 168 176 185 191 202 204 222 225 231 248 0

45: 48 67 76 83 95 104 217 232 3

46: 161 169 177 186

47: 64 91 109 137 146 167 218 224 29

48: 50 83 104 144 185 201 216 217 232 248 45

49: 59 80 93 97 144 160 176 185 191 202 204 222 225 248 44 15 0

50: 59 80 97 104 144 185 201 232 248 48

51: 70 79 86 110 133 214 18 14 2

52: 77 93 102 151 168 187 208 226 231 32

53: 56 73 81 119 120 134 145 229 34 22

54: 99 117 140 147 16 6

55: 67 78 112 128 136 159 217 239 26 5 4

56: 73 119 120 145 161 53 34 22

57: 65 118 125 148 151 157 172 181 184 188 197 208 230 7

58: 68 114 163 176 191 202 204 209 211 222 33 23 15 9 0

59: 80 97 144 185 204 225 248 50 49 44 0

60: 63 96 111 199 237 25

61: 87 89 111 130 194 234

62: 71 78 90 128 138 188 233 239 240 27 21

63: 96 199 237 60 25

64: 91 109 119 134 137 145 146 183 215 218 227 47 29

65: 71 125 138 148 151 157 181 184 188 197 208 230 240 57 27 21 7

66: 149 206 209 39 24 15

67: 83 112 217 55 45 5 3

68: 114 160 165 176 191 202 204 222 58 44 23 9 0

69: 107 128 173

70: 79 84 100 174 179 212 214 244 51 30 19

71: 135 148 157 181 184 188 230 233 240 65 62 27 21 7

72: 107 150 177 186 189 200 203 220 249 1

73: 120 145 56 53 34 22

74: 109 126 183 215 38

75: 89 116 164 190 194 220 247 40 20

76: 95 115 153 228 241 45 37 3

77: 78 102 138 151 187 208 226 240 52 32 26 5 4

78: 112 128 138 159 239 240 77 62 55 26 4

79: 84 110 174 179 212 214 70 51 30 19 2

80: 97 149 202 204 225 59 50 49 44 39 24 15 0

81: 119 134 146 227 229 53 41 17

82: 85 152 175 207 212 244 246 43 30 11 8

83: 95 104 201 217 232 67 48 45

84: 100 103 106 131 174 179 192 193 243 79 70 19

85: 152 175 246 82 11 8

86: 108 135 141 51 42 18 14 2

87: 111 130 136 194 234 61

88: 98 182 41 36 35 12

89: 116 127 130 164 194 75 61 40 20

90: 113 173 233 242 62

91: 109 119 134 137 145 146 218 224 227 64 47 29

92: 122 132 139 171 172 205 235

93: 97 144 160 168 176 185 187 191 202 204 226 231 248 52 49 44 32

94: 141 198 242 35 28 18 12

95: 115 153 216 83 76 45 37

96: 199 237 63 60 25

97: 144 160 168 176 185 191 202 204 225 231 248 93 80 59 50 49 44 0

98: 117 178 236 88 36 16 6

99: 129 140 147 162 54 16 6

100: 103 133 174 192 84 70 19 13

101: 108 110 122 125 139 156 157 181 196 205 214 219 42 7

102: 138 187 226 240 77 52 32 26 5

103: 174 192 243 100 84 19 13

104: 144 185 201 217 232 248 83 50 48 45 32 5

105: 106 123 131 143 193 243 246 10

106: 123 131 143 179 193 243 246 105 84 10

107: 130 173 200 203 72 69 1

108: 110 122 135 139 156 181 196 205 214 219 101 86 42 2

109: 126 137 146 183 215 218 91 74 64 47 38 29

110: 122 139 156 196 205 207 210 212 214 219 221 108 101 79 51 2

111: 87 61 60 25

112: 128 136 159 234 239 78 67 55 4

113: 121 158 170 182 198 223 242 90 28 12

114: 163 176 191 202 204 209 211 222 225 68 58 33 24 23 15 9 0

115: 153 228 241 95 76 37 31 3

116: 164 190 194 220 247 89 75 40 20

117: 140 147 167 178 236 98 54 16 6

118: 124 142 151 155 165 172 180 184 197 208 213 57

119: 120 134 137 145 146 227 91 81 64 56 53

120: 145 161 229 119 73 56 53 34 22

121: 158 170 182 198 223 242 113 41 28 17 12

122: 139 156 196 205 207 210 214 219 221 110 108 101 92

123: 175 246 106 105 10

124: 142 155 165 171 172 180 197 213 235 118

125: 148 157 172 181 184 197 230 101 65 57 7

126: 183 215 109 74 38

127: 89 20

128: 136 159 173 239 112 78 69 62 55 4

129: 133 147 166 178 236 99 16 14 13 6

130: 194 234 107 89 87 61 1

131: 143 179 193 243 106 105 84

132: 154 171 235 238 92

133: 166 129 100 51 14 13

134: 137 145 146 227 119 91 81 64 53 17

135: 141 181 230 108 86 71 42

136: 159 234 128 112 87 55

137: 145 146 183 215 218 224 227 134 119 109 91 64 47 29

138: 151 188 226 233 239 240 102 78 77 65 62 27 26 21 4

139: 156 196 205 210 214 219 221 122 110 108 101 92

140: 147 162 117 99 54 16 6

141: 135 94 86 42 35 18 2

142: 154 155 165 171 172 180 195 213 235 238 124 118 9

143: 152 175 179 193 212 244 246 131 106 105 30 11 8

144: 168 185 187 201 204 231 232 248 104 97 93 59 50 49 48 44 32

145: 146 137 134 120 119 91 73 64 56 53 34 22

146: 218 224 227 145 137 134 119 109 91 81 64 47 29

147: 162 166 140 129 117 99 54 16 6

148: 157 181 184 188 197 230 125 71 65 57 7

149: 163 206 209 211 222 225 80 66 39 24 15 0

150: 164 169 177 186 189 190 203 220 72 40 1

151: 168 187 208 226 231 138 118 77 65 57 52 32

152: 179 207 210 212 221 244 246 143 85 82 43 30 11 8

153: 228 241 115 95 76 37 31 3

154: 171 195 235 238 245 142 132

155: 165 171 172 180 197 213 235 238 142 124 118

156: 196 205 207 210 212 214 219 221 139 122 110 108 101 43 30

157: 181 184 188 197 230 148 125 101 71 65 57 42 7

158: 170 200 203 223 229 249 121 113 17

159: 234 239 136 128 112 78 55 4

160: 168 176 187 191 202 204 225 231 97 93 68 49 44 32 0

161: 169 177 186 189 229 249 120 56 46

162: 192 147 140 99 13

163: 202 209 211 222 225 149 114 58 33 24 15 0

164: 190 194 220 247 150 116 89 75 40 20 1

165: 171 172 180 191 213 155 142 124 118 68

166: 236 147 133 129 16 14 6

167: 224 117 47 29

168: 187 204 208 226 231 248 160 151 144 97 93 52 44 32

169: 177 186 189 190 220 161 150 46

170: 182 198 223 229 249 158 121 113 41 28 17 12

171: 172 180 213 235 238 245 165 155 154 142 132 124 92

172: 180 197 213 235 171 165 155 142 125 124 118 92 57

173: 128 107 90 69

174: 179 192 243 103 100 84 79 70 19 13

175: 246 143 123 85 82 11 10

176: 191 202 204 222 225 160 114 97 93 68 58 49 44 23 0

177: 186 189 203 249 169 161 150 72 46

178: 236 129 117 98 16 6

179: 193 212 244 174 152 143 131 106 84 79 70 30 19 8

180: 213 172 171 165 155 142 124 118

181: 184 188 196 230 157 148 135 125 108 101 71 65 57 42 7

182: 198 223 242 170 121 113 88 41 36 28 17 12

183: 215 137 126 109 74 64 38

184: 188 197 230 181 157 148 125 118 71 65 57 27 21 7

185: 201 232 248 144 104 97 93 59 50 49 48 44 32

186: 189 203 249 177 169 161 150 72 46

187: 208 226 231 248 168 160 151 144 102 93 77 52 32

188: 230 233 240 184 181 157 148 138 71 65 62 57 27 21 7

189: 200 203 220 249 186 177 169 161 150 72 1

190: 220 247 169 164 150 116 75 40 20

191: 202 204 222 225 231 176 165 160 114 97 93 68 58 49 44 0

192: 243 174 162 103 100 84 19 13

193: 243 179 143 131 106 105 84

194: 220 164 130 116 89 87 75 61 40 20 1

195: 238 245 154 142 23 9

196: 205 214 219 181 156 139 122 110 108 101 42

197: 230 184 172 157 155 148 125 124 118 65 57 7

198: 223 242 182 170 121 113 94 41 35 28 12

199: 237 96 63 60 25

200: 203 223 249 189 158 107 72 1

201: 216 217 232 248 185 144 104 83 50 48 32

202: 204 209 211 222 225 191 176 163 160 114 97 93 80 68 58 49 44 15 0

203: 249 200 189 186 177 158 150 107 72 1

204: 222 225 231 202 191 176 168 160 144 114 97 93 80 68 59 58 49 44 15 0

205: 207 210 214 219 221 196 156 139 122 110 108 101 92

206: 209 149 66 39 24

207: 210 212 214 219 221 244 205 156 152 122 110 82 43 30 11 8

208: 226 231 187 168 151 118 77 65 57 52 32

209: 211 222 225 206 202 163 149 114 66 58 39 24 15 0

210: 212 214 219 221 244 207 205 156 152 139 122 110 43 30 8

211: 222 225 209 202 163 149 114 58 39 24 15 0

212: 214 219 221 244 210 207 179 156 152 143 110 82 79 70 43 30 11 8

213: 235 238 180 172 171 165 155 142 124 118

214: 219 221 212 210 207 205 196 156 139 122 110 108 101 79 70 51 30

215: 183 137 126 109 74 64 38

216: 232 201 95 48

217: 232 201 104 83 67 55 48 45 26 5

218: 224 227 146 137 109 91 64 47 29

219: 221 214 212 210 207 205 196 156 139 122 110 108 101 43 30

220: 247 194 190 189 169 164 150 116 75 72 40 20 1

221: 219 214 212 210 207 205 156 152 139 122 110 43 30 8

222: 225 211 209 204 202 191 176 163 149 114 68 58 49 44 33 24 23 15 0

223: 242 249 200 198 182 170 158 121 113 28 17

224: 218 167 146 137 91 47 29

225: 222 211 209 204 202 191 176 163 160 149 114 97 80 59 49 44 24 15 0

226: 231 208 187 168 151 138 102 93 77 52 32 26 5

227: 218 146 137 134 119 91 81 64 29

228: 241 153 115 76 37 31 3

229: 249 170 161 158 120 81 53 17

230: 197 188 184 181 157 148 135 125 71 65 57 27 21 7

231: 248 226 208 204 191 187 168 160 151 144 97 93 52 44 32

232: 248 217 216 201 185 144 104 83 50 48 45

233: 240 188 138 90 71 62 27 21

234: 159 136 130 112 87 61

235: 238 213 172 171 155 154 142 132 124 92

236: 178 166 129 117 98 16 6

237: 199 96 63 60

238: 245 235 213 195 171 155 154 142 132

239: 240 159 138 128 112 78 62 55 26 4

240: 239 233 188 138 102 78 77 71 65 62 27 26 21 4

241: 228 153 115 76 37 31 3

242: 223 198 182 121 113 94 90 28 12

243: 193 192 174 131 106 105 103 84 19

244: 246 212 210 207 179 152 143 82 70 43 30 11 8

245: 238 195 171 154

246: 244 175 152 143 123 106 105 85 82 11 10 8

247: 220 190 164 116 75 40 20

248: 232 231 201 187 185 168 144 104 97 93 59 50 49 48 44 32

249: 229 223 203 200 189 186 177 170 161 158 72

BFS:

0 --> 15 --> 24 --> 44 --> 49 --> 58 --> 59 --> 68 --> 80 --> 97 -->

114 --> 149 --> 160 --> 163 --> 176 --> 191 --> 202 --> 204 --> 209 --> 211 --> 222 --> 225 --> 39 --> 66 --> 206 --> 93 --> 144 --> 168 --> 185 --> 231 --> 248 --> 33 --> 23 --> 9 --> 50 --> 165 --> 187 --> 32 --> 226 --> 52 -->

201 --> 232 --> 104 --> 48 --> 208 --> 151 --> 195 --> 142 --> 171 --> 172 --> 180 --> 213 --> 155 --> 124 --> 118 --> 102 --> 77 --> 5 --> 138 --> 26 -->

216 --> 217 --> 83 --> 45 --> 65 --> 57 --> 238 --> 245 --> 154 --> 235 -->

132 --> 92 --> 197 --> 125 --> 184 --> 240 --> 78 --> 4 --> 55 --> 67 -->

188 --> 233 --> 239 --> 62 --> 27 --> 21 --> 95 --> 76 --> 3 --> 71 -->

148 --> 157 --> 181 --> 230 --> 7 --> 122 --> 139 --> 205 --> 101 --> 112 -->

128 --> 159 --> 136 --> 90 --> 115 --> 153 --> 37 --> 228 --> 241 --> 135 -->

42 --> 196 --> 108 --> 156 --> 207 --> 210 --> 214 --> 219 --> 221 --> 110 --> 234 --> 173 --> 69 --> 87 --> 113 --> 242 --> 31 --> 141 --> 86 --> 2 -->

212 --> 43 --> 30 --> 244 --> 152 --> 82 --> 11 --> 8 --> 79 --> 70 -->

51 --> 130 --> 61 --> 107 --> 111 --> 194 --> 121 --> 158 --> 170 --> 182 -->

198 --> 223 --> 28 --> 12 --> 94 --> 35 --> 18 --> 14 --> 179 --> 143 -->

246 --> 85 --> 175 --> 84 --> 174 --> 19 --> 100 --> 133 --> 89 --> 1 -->

200 --> 203 --> 72 --> 60 --> 25 --> 220 --> 164 --> 116 --> 75 --> 40 -->

20 --> 41 --> 17 --> 229 --> 249 --> 88 --> 36 --> 129 --> 166 --> 193 -->

131 --> 106 --> 105 --> 123 --> 10 --> 103 --> 192 --> 243 --> 13 --> 127 -->

150 --> 189 --> 186 --> 177 --> 63 --> 96 --> 199 --> 237 --> 247 --> 190 -->

169 --> 81 --> 134 --> 161 --> 120 --> 53 --> 98 --> 147 --> 178 --> 236 -->

99 --> 16 --> 6 --> 162 --> 46 --> 119 --> 146 --> 227 --> 137 --> 145 -->

91 --> 64 --> 56 --> 73 --> 34 --> 22 --> 117 --> 140 --> 54 --> 218 -->

224 --> 109 --> 47 --> 29 --> 183 --> 215 --> 167 --> 126 --> 74 --> 38

**Homework Assignment:**

1. Change the Graph implementation to have an adjacency matrix in place of adjacency lists and then compare the time to execute BFS from a source to all the nodes for both adjacency lists and adjacency matrix.
2. Write a detailed report on the performance and the time complexities of the two variations of graph implementation for BFS. Report the performance for each of the three input graphs: largeG.txt, mediumG.txt, tinyDG.txt.

The following source code:

ADJACENT MATRIX GRAPH :

package Lab10;

public class AdjacentMatrixGraph {

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

int x = 0;

BufferedReader scan = new BufferedReader(new FileReader("tinyDG.txt"));

UndirectedGraph undirected = new UndirectedGraph(scan);

long usTime = System.nanoTime();

undirected.BFS(undirected, x);

System.out.println("BFS on an undirected graph with tiny input took: " + (System.nanoTime() - usTime) + " nanoseconds.");

BufferedReader scan2 = new BufferedReader(new FileReader("tinyDG.txt"));

Undirected gm = new Undirected(scan2);

long gsTime = System.nanoTime();

gm.bfs(gm, x);

System.out.println("BFS on an adjacency matrix with tiny input took: " + (System.nanoTime() - gsTime) + " nanoseconds.");

System.out.println();

BufferedReader scan3 = new BufferedReader(new FileReader("mediumG.txt"));

UndirectedGraph undirectedmed = new UndirectedGraph(scan3);

long umTime = System.nanoTime();

undirectedmed.BFS(undirectedmed, x);

System.out.println("BFS on an undirected graph with medium input took: " + (System.nanoTime() - umTime) + " nanoseconds.");

BufferedReader scan4 = new BufferedReader(new FileReader("mediumG.txt"));

Undirected gmMed = new Undirected(scan4);

long gmTime = System.nanoTime();

gmMed.bfs(gmMed, x);

System.out.println("BFS on an adjacency matrix with medium input took: " + (System.nanoTime() - gmTime) + " nanoseconds.");

System.out.println();

BufferedReader scan5 = new BufferedReader(new FileReader("largeG.txt"));

UndirectedGraph undirectedLarge = new UndirectedGraph(scan5);

long ulTime = System.nanoTime();

undirectedLarge.BFS(undirectedLarge, x);

System.out.println("BFS on an undirected graph with large input took: " + (System.nanoTime() - ulTime) + " nanoseconds.");

BufferedReader scan6 = new BufferedReader(new FileReader("largeG.txt"));

Undirected gmLarge = new Undirected(scan6);

long glTime = System.nanoTime();

gmLarge.bfs(gmLarge, x);

System.out.println("BFS on an adjacency matrix with large input took: " + (System.nanoTime() - glTime) + " nanoseconds.");

}

public int V;

public int E;

public int[][] adj;

public AdjacentMatrixGraph()

{

V = 0;

E = 0;

}

public AdjacentMatrixGraph(BufferedReader reader) throws IOException

{

String line;

line = reader.readLine();

V = Integer.parseInt(line);

line = reader.readLine();

E = Integer.parseInt(line);

adj = new int[V][V];

while ((line = reader.readLine()) != null) {

int tempV1, tempV2;

StringTokenizer st = new StringTokenizer(line, " ");

tempV1 = Integer.parseInt(st.nextToken());

tempV2 = Integer.parseInt(st.nextToken());

addE(tempV1, tempV2, 1);

}

}

public void addE(int v, int w, int edge) {

//override

}

public String tostring()

{ int i, j;

String s = new String();

s = "There are "+V+" vertices and "+E+" edges\n";

for(i = 0;i < V; i++)

{

System.out.println();

s = s+i+": ";

for(j = 0; j< V -1; j++)

{

s = s+ adj[i][j] +" ";

}

s = s+ adj[i][j] + "\n";

}

return s;

}

public void bfs(AdjacentMatrixGraph G, int s){

// Mark all the vertices as not visited(By default

// set as false)

boolean visited[] = new boolean[G.V];

// Create a queue for BFS

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

// Mark the current node as visited and enqueue it

queue.add(s);

while (queue.size() != 0)

{

// Dequeue a vertex from queue and print it

s = queue.poll();

// Get all adjacent vertices of the dequeued vertex s

// If a adjacent has not been visited, then mark it

// visited and enqueue it

for(int j = 0; j < G.V; j++) {

if( !visited[j] && G.adj[s][j] > 0 && G.adj[j][s] > 0)

{

visited[j] = true;

queue.add(j);

}

}

}

}

public void printPath(AdjacentMatrixGraph G, int s){

boolean visited[] = new boolean[G.V];

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

queue.add(s);

while (queue.size() != 0)

{

s = queue.poll();

for(int j = 0; j < G.V; j++) {

if( !visited[j] && G.adj[s][j] > 0 && G.adj[j][s] > 0)

{

visited[j] = true;

queue.add(j);

}

}

}

}

}

UNDIRECTED GRAPH :

public class Undirected extends AdjacentMatrixGraph {

public Undirected(){

super();

}

public Undirected(BufferedReader reader) throws IOException {

super(reader);

}

public void addE(int v, int w, int edge) {

super.adj[v][w] = edge;

super.adj[w][v] = edge;

}

}

DIRECTED GRAPH :

public class Directed extends AdjacentMatrixGraph {

public Directed(){

super();

}

public Directed(BufferedReader reader) throws IOException {

super(reader);

}

public void addE(int v, int w, int edge) {

super.adj[v][w] = edge;

}

}

The output was:

BFS on an undirected graph with tiny input took: 478613 nanoseconds.

BFS on an adjacency matrix with tiny input took: 110740 nanoseconds.

BFS on an undirected graph with medium input took: 2532972 nanoseconds.

BFS on an adjacency matrix with medium input took: 3676564 nanoseconds.

Exception in thread "main" java.lang.OutOfMemoryError: Java heap space

at Lab10.Graph.<init>(Graph.java:59)

at Lab10.UndirectedGraph.<init>(UndirectedGraph.java:13)

at Lab10.AdjacentMatrixGraph.main(AdjacentMatrixGraph.java:48)

Large input does not work because there is not enough memory for it on my laptop.