**Analisis Algoritma**

Tugas 5



­­

**Dibuat oleh:**

Muhammad Iqbal Alif Fadilla

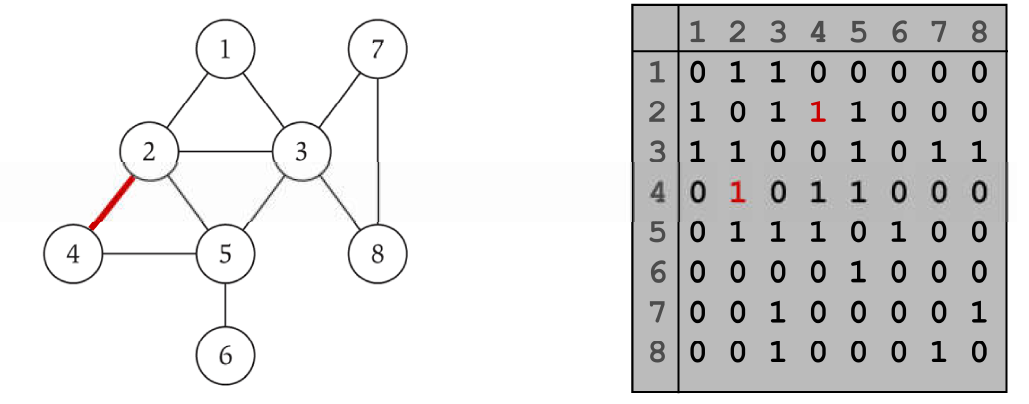
140810180020

**Universitas Padjadjaran**

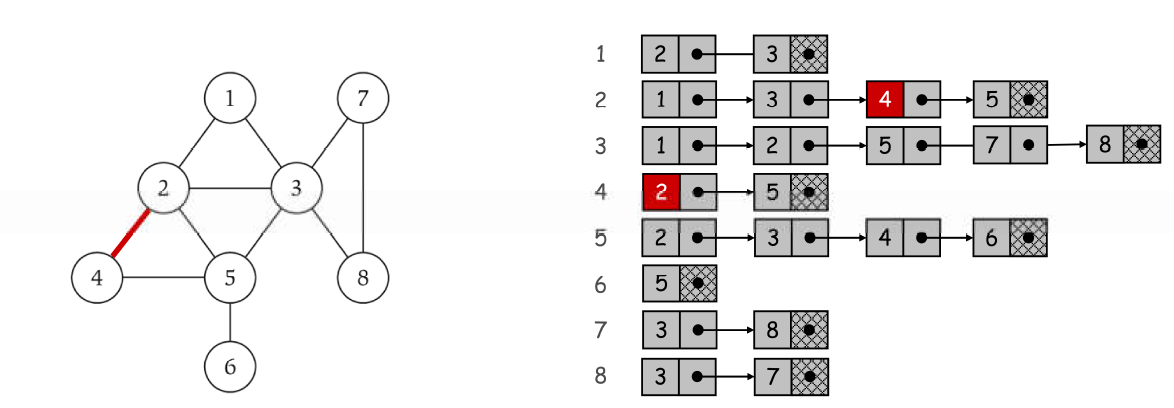
**Fakultas Matematika dan Ilmu Pengetahuan**

**2020**

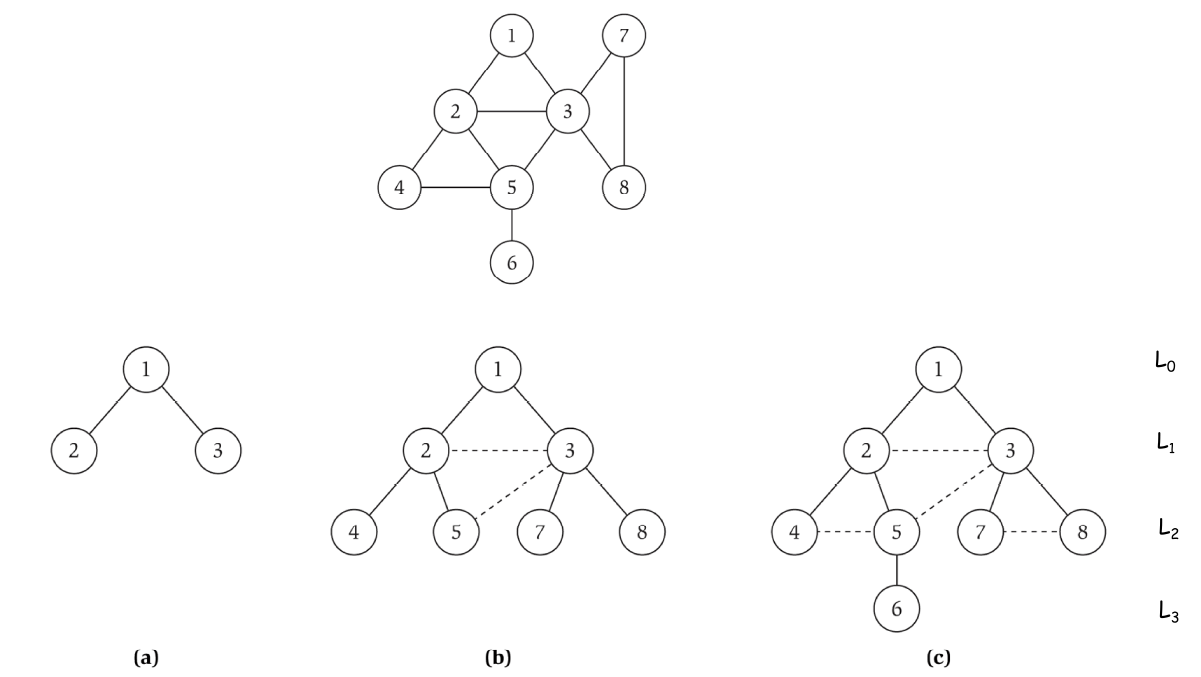
1. Dengan menggunakan undirected graph dan adjacency matrix berikut, buatlah koding programnya menggunakan Bahasa C++



1. /\*
2. Nama        : Muhammad Iqbal Alif Fadilla
3. Kelas       : B
4. NPM         : 140810180020
5. Deskripsi   : Adjacency Matrix with undirected graph
6. \*/
8. #include <iostream>
9. #include <cstdlib>
11. **using** **namespace** std;
13. **class** AdjacencyMatrix
14. {
15. **private**:
16. **int** n;
17. **int** \*\*adj;
18. **bool** \*visited;
20. **public**:
21. AdjacencyMatrix(**int** n)
22. {
23. **this**->n = n;
24. visited = **new** **bool**[n];
25. adj = **new** **int** \*[n];
26. **for** (**int** i = 0; i < n; i++)
27. {
28. adj[i] = **new** **int**[n];
29. **for** (**int** j = 0; j < n; j++)
30. {
31. adj[i][j] = 0;
32. }
33. }
34. }
35. /\*
36. \* Menambahkan edge ke graf
37. \*/
38. **void** add\_edge(**int** origin, **int** destin)
39. {
40. **if** (origin > n || destin > n || origin < 0 || destin < 0)
41. {
42. cout << "Invalid edge!\n";
43. }
44. **else**
45. {
46. adj[origin - 1][destin - 1] = 1;
47. }
48. }
49. /\*
50. \* Mencetak graf
51. \*/
52. **void** display()
53. {
54. **int** i, j;
55. **for** (i = 0; i < n; i++)
56. {
57. **for** (j = 0; j < n; j++)
58. cout << adj[i][j] << "  ";
59. cout << endl;
60. }
61. }
62. };
63. /\*
64. \* Main
65. \*/
66. **int** main()
67. {
68. **int** nodes, max\_edges, origin, destin;
69. cout << "Enter number of nodes: ";
70. cin >> nodes;
71. AdjacencyMatrix am(nodes);
72. max\_edges = nodes \* (nodes - 1);
73. **for** (**int** i = 0; i < max\_edges; i++)
74. {
75. cout << "Enter edge (-1 -1 to exit): ";
76. cin >> origin >> destin;
77. **if** ((origin == -1) && (destin == -1))
78. **break**;
79. am.add\_edge(origin, destin);
80. }
81. am.display();
82. **return** 0;
83. }
84. Dengan menggunakan undirected graph dan representasi adjacency list berikut, buatlah koding programnya menggunakan Bahasa C++



1. /\*
2. Nama        : Muhammad Iqbal Alif Fadilla
3. Kelas       : B
4. NPM         : 140810180020
5. Deskripsi   : Adjacency List with undirected graph
6. \*/
8. #include <iostream>
9. #include <cstdlib>
11. **using** **namespace** std;
13. **struct** AdjListNode
14. {
15. **int** dest;
16. **struct** AdjListNode \*next;
17. };
19. **struct** AdjList
20. {
21. **struct** AdjListNode \*head;
22. };
24. /\*
25. \* Class Graph
26. \*/
27. **class** Graph
28. {
29. **private**:
30. **int** V;
31. **struct** AdjList \*array;
33. **public**:
34. Graph(**int** V)
35. {
36. **this**->V = V;
37. array = **new** AdjList[V];
38. **for** (**int** i = 1; i <= V; ++i)
39. array[i].head = NULL;
40. }
41. /\*
42. \* Creating New Adjacency List Node
43. \*/
44. AdjListNode \*newAdjListNode(**int** dest)
45. {
46. AdjListNode \*newNode = **new** AdjListNode;
47. newNode->dest = dest;
48. newNode->next = NULL;
49. **return** newNode;
50. }
51. /\*
52. \* Adding Edge to Graph
53. \*/
54. **void** addEdge(**int** src, **int** dest)
55. {
56. AdjListNode \*newNode = newAdjListNode(dest);
57. newNode->next = array[src].head;
58. array[src].head = newNode;
59. newNode = newAdjListNode(src);
60. newNode->next = array[dest].head;
61. array[dest].head = newNode;
62. }
63. /\*
64. \* Print the graph
65. \*/
66. **void** printGraph()
67. {
68. **int** v;
69. **for** (v = 1; v <= V; ++v)
70. {
71. AdjListNode \*pCrawl = array[v].head;
72. cout << "\n Adjacency list of vertex " << v << "\n head ";
73. **while** (pCrawl)
74. {
75. cout << "-> " << pCrawl->dest;
76. pCrawl = pCrawl->next;
77. }
78. cout << endl;
79. }
80. }
81. };
83. /\*
84. \* Main
85. \*/
86. **int** main()
87. {
88. Graph gh(8);
89. gh.addEdge(1, 2);
90. gh.addEdge(1, 3);
91. gh.addEdge(2, 3);
92. gh.addEdge(2, 4);
93. gh.addEdge(2, 5);
94. gh.addEdge(3, 5);
95. gh.addEdge(3, 7);
96. gh.addEdge(3, 8);
97. gh.addEdge(4, 5);
98. gh.addEdge(5, 6);
99. gh.addEdge(7, 8);
100. // print the adjacency list representation of the above graph
101. gh.printGraph();
103. **return** 0;
104. }
105. Buatlah program Breadth First Search dari algoritma BFS yang telah diberikan. Kemudian uji coba program Anda dengan menginputkan undirected graph sehingga menghasilkan tree BFS. Hitung dan berikan secara asimptotik berapa kompleksitas waktunya dalam Big-Θ!



1. /\*
2. Nama        : Muhammad Iqbal Alif Fadilla
3. Kelas       : B
4. NPM         : 140810180020
5. Deskripsi   : Breadth First Search
6. \*/
8. #include <iostream>
9. #include <list>
11. **using** **namespace** std;
13. **class** Graph
14. {
15. **int** V; // No. of vertices
17. list<**int**> \*adj;
19. **public**:
20. Graph(**int** V); // Constructor
22. // function to add an edge to graph
23. **void** addEdge(**int** v, **int** w);
25. // prints BFS traversal from a given source s
26. **void** BFS(**int** s);
27. };
29. Graph::Graph(**int** V)
30. {
31. **this**->V = V;
32. adj = **new** list<**int**>[V];
33. }
35. **void** Graph::addEdge(**int** v, **int** w)
36. {
37. adj[v].push\_back(w); // Add w to v’s list.
38. }
40. **void** Graph::BFS(**int** s)
41. {
42. // Mark all the vertices as not visited
43. **bool** \*visited = **new** **bool**[V];
44. **for** (**int** i = 0; i < V; i++)
45. visited[i] = **false**;
47. // Create a queue for BFS
48. list<**int**> queue;
50. // Mark the current node as visited and enqueue it
51. visited[s] = **true**;
52. queue.push\_back(s);
54. // 'i' will be used to get all adjacent
55. // vertices of a vertex
56. list<**int**>::iterator i;
58. **while** (!queue.empty())
59. {
60. // Dequeue a vertex from queue and print it
61. s = queue.front();
62. cout << s << " ";
63. queue.pop\_front();
65. // Get all adjacent vertices of the dequeued
66. // vertex s. If a adjacent has not been visited,
67. // then mark it visited and enqueue it
68. **for** (i = adj[s].begin(); i != adj[s].end(); ++i)
69. {
70. **if** (!visited[\*i])
71. {
72. visited[\*i] = **true**;
73. queue.push\_back(\*i);
74. }
75. }
76. }
77. }
79. // Driver program to test methods of graph class
80. **int** main()
81. {
82. // Create a graph given in the above diagram
83. Graph g(8);
84. g.addEdge(1, 2);
85. g.addEdge(1, 3);
86. g.addEdge(2, 4);
87. g.addEdge(2, 5);
88. g.addEdge(2, 3);
89. g.addEdge(3, 7);
90. g.addEdge(3, 8);
91. g.addEdge(4, 5);
92. g.addEdge(5, 3);
93. g.addEdge(5, 6);
94. g.addEdge(7, 8);
96. cout << "Following is Breadth First Traversal "
97. << "(starting from vertex 1) \n";
98. g.BFS(1);
100. **return** 0;
101. }

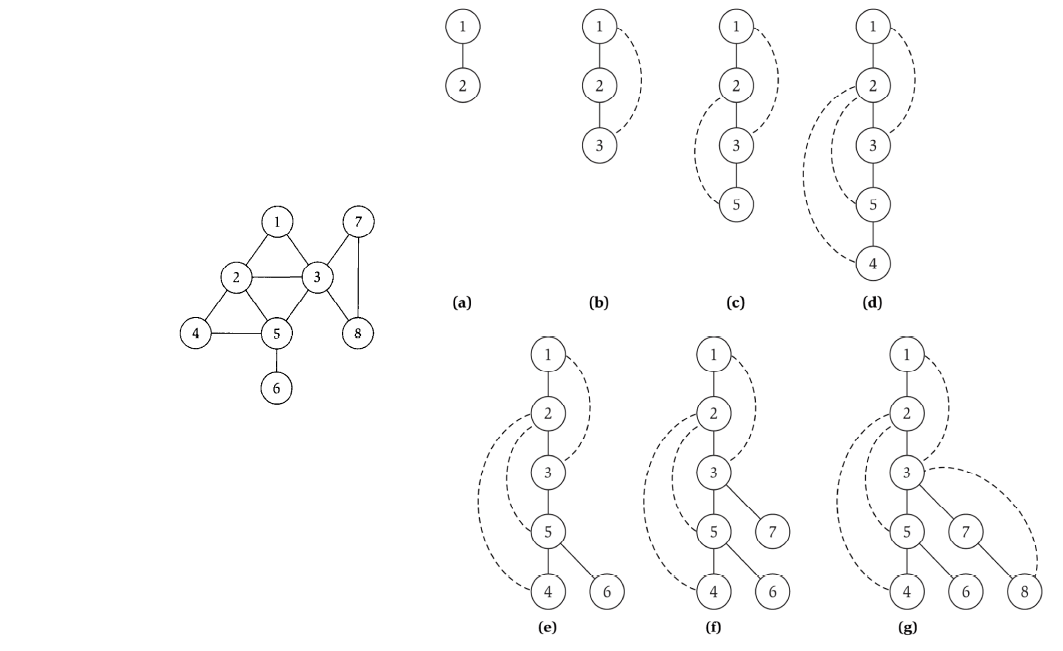
Analisis :

* BFS merupakan metode pencarian secara melebar sehingga mengunjungi node dari kiri ke kanan di level yang sama. Apabila semua node pada suatu level sudah dikunjungi semua, maka akan berpindah ke level selanjutnya. Dalam worst case BFS harus mempertimbangkan semua jalur (path) untuk semua node yang mungkin, maka nilai kompleksitas waktu dari BFS adalah O( |V| + |E| ).
* Karena Big-O dari BFS adalah O(V+E) dimana V itu jumlah vertex dan E itu adalah

jumlah edges maka Big-O = O(n) dimana n = v+e

Maka dari itu Big-Ө nya adalah Ө(n).

1. Buatlah program Depth First Search dari algoritma DFS yang telah diberikan. Kemudian uji coba program Anda dengan menginputkan undirected graph sehingga menghasilkan tree DFS. Hitung dan berikan secara asimptotik berapa kompleksitas waktunya dalam Big-Θ!



1. /\*
2. Nama        : Muhammad Iqbal Alif Fadilla
3. Kelas       : B
4. NPM         : 140810180020
5. Deskripsi   : Depth First Search
6. \*/
8. #include <iostream>
9. #include <list>
10. **using** **namespace** std;
12. **class** Graph
13. {
14. **int** V; // No. simpul
16. // Pointer ke array yang memiliki adjacency lists
17. list<**int**> \*adj;
19. // Fungsi rekursif yang digunakan DFS
20. **void** DFSUtil(**int** v, **bool** visited[]);
22. **public**:
23. Graph(**int** V); // Constructor
25. // fungsi untuk menambah tepian ke graf
26. **void** addEdge(**int** v, **int** w);
28. // DFS traversal dari simpul yang terjangkau dari v
29. **void** DFS(**int** v);
30. };
32. Graph::Graph(**int** V)
33. {
34. **this**->V = V;
35. adj = **new** list<**int**>[V];
36. }
38. **void** Graph::addEdge(**int** v, **int** w)
39. {
40. adj[v].push\_back(w); // Menambah w ke list v.
41. }
43. **void** Graph::DFSUtil(**int** v, **bool** visited[])
44. {
45. // Menandakan node bersangkutan sudah dikunjungi lalu cetak
46. visited[v] = **true**;
47. cout << v << " ";
49. // Ulang simpul berdekatan ke node ini
50. list<**int**>::iterator i;
51. **for** (i = adj[v].begin(); i != adj[v].end(); ++i)
52. **if** (!visited[\*i])
53. DFSUtil(\*i, visited);
54. }
56. // DFS traversal dari simpul terjangkau dari v.
57. // Menggunakan rekursif DFSUtil()
58. **void** Graph::DFS(**int** v)
59. {
60. // Menandakan semua simpul belum dikunjungi
61. **bool** \*visited = **new** **bool**[V];
62. **for** (**int** i = 0; i < V; i++)
63. visited[i] = **false**;
65. // Memanggil fungsi rekursif pembantu untuk mencetak DFS traversal
66. DFSUtil(v, visited);
67. }
69. **int** main()
70. {
71. // Membuat graf di diagram
72. Graph g(8);
73. g.addEdge(1, 2);
74. g.addEdge(1, 3);
75. g.addEdge(2, 5);
76. g.addEdge(2, 4);
77. g.addEdge(5, 6);
78. g.addEdge(3, 7);
79. g.addEdge(3, 8);
80. g.addEdge(7, 8);
82. cout << "Depth First Traversal"
83. " (dimulai dari node 1) \n";
84. g.DFS(1);
86. **return** 0;
87. }

Analisis :

* DFS merupakan metode pencarian mendalam, yang mengunjungi semua node dari yang terkiri lalu geser ke kanan hingga semua node dikunjungi. Kompleksitas ruang algoritma DFS adalah O(bm), karena kita hanya hanya perlu menyimpan satu buah lintasan tunggal dari akar sampai daun, ditambah dengan simpul-simpul saudara kandungnya yang belum dikembangkan.
* Big O

Kompleksitas total DFS () adalah (V+E).

O(n) dengan V = Jumlah Verteks dan E = Jumlah Edges