Muhammad Iqbal Alif Fadilla
140810180020
Tugas 6

Analisis Algoritma

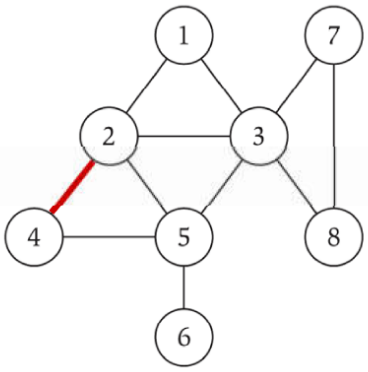
Tugas 5

Dibuat oleh:
Muhammad Iqbal Alif Fadilla
140810180020

Universitas Padjadjaran
Fakultas Matematika dan Ilmu Pengetahuan
2020

Muhammad Iqbal Alif Fadilla
140810180020
Tugas 6

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



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

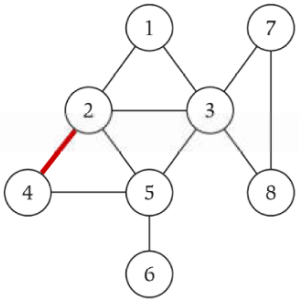
```
/*
    Nama        : Muhammad Iqbal Alif Fadilla
    Kelas       : B
    NPM         : 140810180020
    Deskripsi   : Adjacency Matrix with undirected graph
*/

#include <iostream>
#include <cstdlib>

using namespace std;

class AdjacencyMatrix
{
private:
    int n;
    int **adj;
    bool *visited;

public:
    AdjacencyMatrix(int n)
    {
        this->n = n;
        visited = new bool[n];
        adj = new int *[n];
        for (int i = 0; i < n; i++)
        {
            adj[i] = new int[n];
            for (int j = 0; j < n; j++)
            {
                adj[i][j] = 0;
            }
        }
    }
    /*
     * Menambahkan edge ke graf
     */
    void add_edge(int origin, int destin)
    {
        if (origin > n || destin > n || origin < 0 || destin < 0)
        {
            cout << "Invalid edge!\n";
        }
```

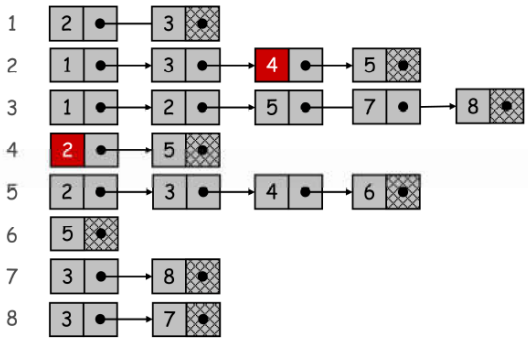
Muhammad Iqbal Alif Fadilla
140810180020
Tugas 6

```
        else
        {
            adj[origin - 1][destin - 1] = 1;
        }
    }
    /*
         * Mencetak graf
         */
    void display()
    {
        int i, j;
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < n; j++)
                cout << adj[i][j] << "  ";
            cout << endl;
        }
    }
};
/*
 * Main
 */
int main()
{
    int nodes, max_edges, origin, destin;
    cout << "Enter number of nodes: ";
    cin >> nodes;
    AdjacencyMatrix am(nodes);
    max_edges = nodes * (nodes - 1);
    for (int i = 0; i < max_edges; i++)
    {
        cout << "Enter edge (-1 -1 to exit): ";
        cin >> origin >> destin;
        if ((origin == -1) && (destin == -1))
            break;
        am.add_edge(origin, destin);
    }
    am.display();
    return 0;
}
```

Muhammad Iqbal Alif Fadilla
140810180020
Tugas 6

2. Dengan menggunakan undirected graph dan representasi adjacency list berikut, buatlah koding programnya menggunakan Bahasa C++





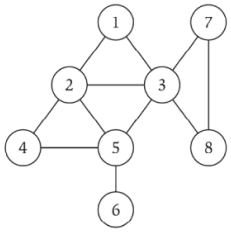

```
/*
    Nama        : Muhammad Iqbal Alif Fadilla
    Kelas       : B
    NPM         : 140810180020
    Deskripsi   : Adjacency List with undirected graph
*/

#include <iostream>
#include <cstdlib>

using namespace std;

struct AdjListNode
{
    int dest;
    struct AdjListNode *next;
};

struct AdjList
{
    struct AdjListNode *head;
};

/*
 * Class Graph
 */
class Graph
{
private:
    int V;
    struct AdjList *array;

public:
    Graph(int V)
    {
        this->V = V;
        array = new AdjList[V];
        for (int i = 1; i <= V; ++i)
            array[i].head = NULL;
    }
    /*
         * Creating New Adjacency List Node
         */
    AdjListNode *newAdjListNode(int dest)
```

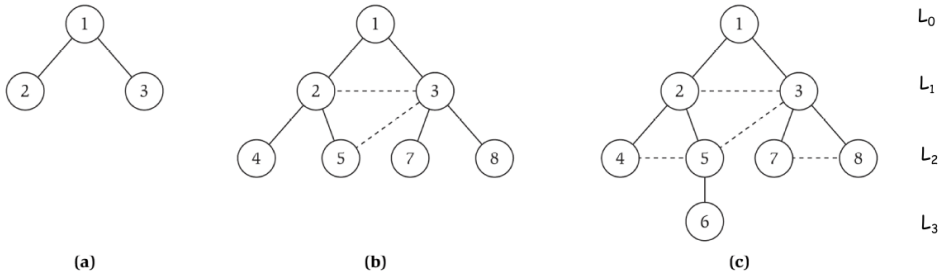
Muhammad Iqbal Alif Fadilla
140810180020
Tugas 6

```
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. };
82. 
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();
102. 
103.          return 0;
104.      }
```

Muhammad Iqbal Alif Fadilla
140810180020
Tugas 6

3. 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!






```
/*
    Nama        : Muhammad Iqbal Alif Fadilla
    Kelas       : B
    NPM         : 140810180020
    Deskripsi   : Breadth First Search
*/

#include <iostream>
#include <list>

using namespace std;

class Graph
{
    int V; // No. of vertices

    list<int> *adj;

public:
    Graph(int V); // Constructor

    // function to add an edge to graph
    void addEdge(int v, int w);

    // prints BFS traversal from a given source s
    void BFS(int s);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

```

Muhammad Iqbal Alif Fadilla
140810180020
Tugas 6

```
35. void Graph::addEdge(int v, int w)
36. {
37.     adj[v].push_back(w); // Add w to v’s list.
38. }
39.
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;
46.
47.     // Create a queue for BFS
48.     list<int> queue;
49.
50.     // Mark the current node as visited and enqueue it
51.     visited[s] = true;
52.     queue.push_back(s);
53.
54.     // 'i' will be used to get all adjacent
55.     // vertices of a vertex
56.     list<int>::iterator i;
57.
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();
64.
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. }
78.
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);
95.
```

Muhammad Iqbal Alif Fadilla
140810180020
Tugas 6

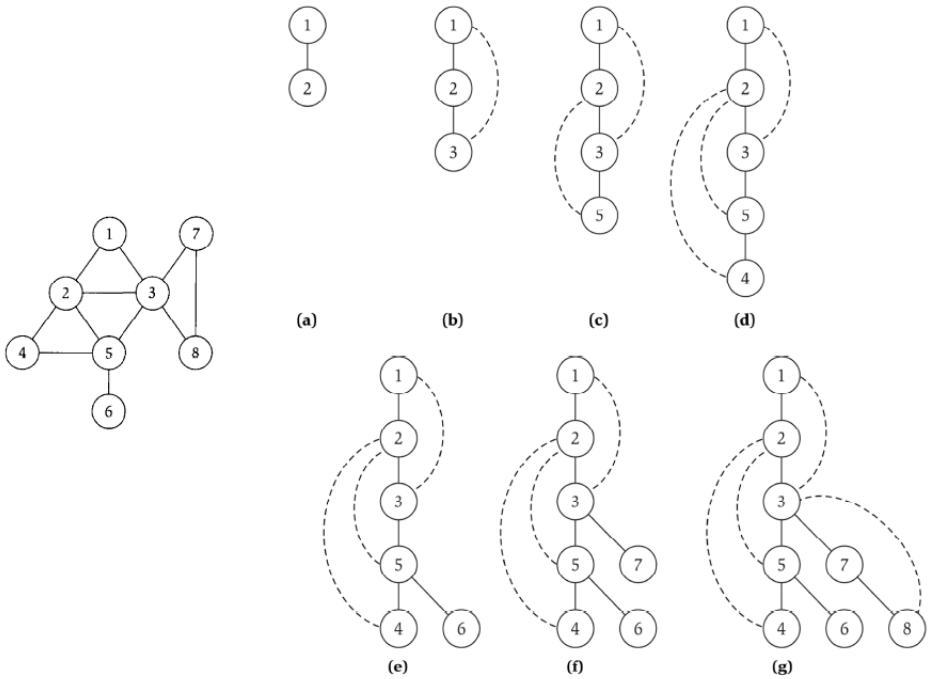
```
96.     cout << "Following is Breadth First Traversal "
97.          << "(starting from vertex 1) \n";
98.     g.BFS(1);
99.
100.          return 0;
101.      }
```

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-Θ nya adalah Θ(n).

Muhammad Iqbal Alif Fadilla
140810180020
Tugas 6

4. 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!




```
/*
    Nama        : Muhammad Iqbal Alif Fadilla
    Kelas       : B
    NPM         : 140810180020
    Deskripsi   : Depth First Search
*/

#include <iostream>
#include <list>
using namespace std;

class Graph
{
    int V; // No. simpul

    // Pointer ke array yang memiliki adjacency lists
    list<int> *adj;

    // Fungsi rekursif yang digunakan DFS
    void DFSUtil(int v, bool visited[]);

public:
    Graph(int V); // Constructor

    // fungsi untuk menambah tepian ke graf
    void addEdge(int v, int w);

    // DFS traversal dari simpul yang terjangkau dari v
    void DFS(int v);
};
```

Muhammad Iqbal Alif Fadilla
140810180020
Tugas 6

```
31.
32. Graph::Graph(int V)
33. {
34.     this->V = V;
35.     adj = new list<int>[V];
36. }
37.
38. void Graph::addEdge(int v, int w)
39. {
40.     adj[v].push_back(w); // Menambah w ke list v.
41. }
42.
43. void Graph::DFSUtil(int v, bool visited[])
44. {
45.     // Menandakan node bersangkutan sudah dikunjungi lalu cetak
46.     visited[v] = true;
47.     cout << v << " ";
48.
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. }
55.
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;
64.
65.     // Memanggil fungsi rekursif pembantu untuk mencetak DFS traversal
66.     DFSUtil(v, visited);
67. }
68.
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);
81.
82.     cout << "Depth First Traversal"
83.             " (dimulai dari node 1) \n";
84.     g.DFS(1);
85.
86.     return 0;
87. }
```

Muhammad Iqbal Alif Fadilla
140810180020
Tugas 6

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