**LAPORAN PRAKTIKUM**

**ANALISIS ALGORITMA**



**DISUSUN OLEH**

Gede Bagus Darmagita – 140810180068

**PROGRAM STUDI TEKNIK INFORMATIKA**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN**

**2020**

**Studi Kasus 6**

1. Dengan menggunakan undirected graph dan adjacency matrix berikut, buatlah koding programmnya menggunakan bahasa C++.

/\*

    Nama        : Gede Bagus Darmagita

    Kelas       : B

    NPM         : 140810180068

\*/

#include <iostream>

using namespace std;

int vertArr[20][20];

int count = 0;

void displayMatrix(int v);

void add\_edge(int u, int v);

main(int argc, char \*argv[])

{

    int v = 8;

    add\_edge(1, 2);

    add\_edge(1, 3);

    add\_edge(2, 4);

    add\_edge(2, 5);

    add\_edge(3, 2);

    add\_edge(3, 8);

    add\_edge(4, 5);

    add\_edge(5, 6);

    add\_edge(7, 3);

    add\_edge(8, 7);

    displayMatrix(v);

}

void displayMatrix(int v)

{

    int i, j;

    for (i = 0; i < v; i++)

    {

        for (j = 0; j < v; j++)

        {

            cout << vertArr[i][j] << " ";

        }

        cout << endl;

    }

}

void add\_edge(int u, int v)

{

    vertArr[u][v] = 1;

    vertArr[v][u] = 1;

}

1. Dengan menggunakan undirected graph dan representasi adjacency list, buatlah koding programmnya menggunakan bahasa C++.

/\*

    Nama        : Gede Bagus Darmagita

    Kelas       : B

    NPM         : 140810180068

\*/

#include <iostream>

#include <cstdlib>

using namespace std;

struct AdjListNode

{

    int dest;

    struct AdjListNode \*next;

};

struct AdjList

{

    struct AdjListNode \*head;

};

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;

    }

    AdjListNode \*newAdjListNode(int dest)

    {

        AdjListNode \*newNode = new AdjListNode;

        newNode->dest = dest;

        newNode->next = NULL;

        return newNode;

    }

    void addEdge(int src, int dest)

    {

        AdjListNode \*newNode = newAdjListNode(dest);

        newNode->next = array[src].head;

        array[src].head = newNode;

        newNode = newAdjListNode(src);

        newNode->next = array[dest].head;

        array[dest].head = newNode;

    }

    void printGraph()

    {

        int v;

        for (v = 1; v <= V; ++v)

        {

            AdjListNode \*pCrawl = array[v].head;

            cout << "\n Adjacency list of vertex " << v << "\n head ";

            while (pCrawl)

            {

                cout << "-> " << pCrawl->dest;

                pCrawl = pCrawl->next;

            }

            cout << endl;

        }

    }

};

int main()

{

    Graph Z(8);

    Z.addEdge(7, 8);

    Z.addEdge(5, 6);

    Z.addEdge(3, 8);

    Z.addEdge(3, 7);

    Z.addEdge(4, 5);

    Z.addEdge(5, 3);

    Z.addEdge(2, 5);

    Z.addEdge(2, 4);

    Z.addEdge(2, 3);

    Z.addEdge(1, 3);

    Z.addEdge(1, 2);

    Z.printGraph();

}

1. 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        : Gede Bagus Darmagita

    Kelas       : B

    NPM         : 140810180068

\*/

#include <iostream>

using namespace std;

int main()

{

    int vertexSize = 8;

    int adjacency[8][8] = {

        {0, 1, 1, 0, 0, 0, 0, 0},

        {1, 0, 1, 1, 1, 0, 0, 0},

        {1, 1, 0, 0, 1, 0, 1, 1},

        {0, 1, 0, 0, 1, 0, 0, 0},

        {0, 1, 1, 1, 0, 1, 0, 0},

        {0, 0, 0, 0, 1, 0, 0, 0},

        {0, 0, 1, 0, 0, 0, 0, 1},

        {0, 0, 1, 0, 0, 0, 1, 0}};

    bool discovered[vertexSize];

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

    {

        discovered[i] = false;

    }

    int output[vertexSize];

    discovered[0] = true;

    output[0] = 1;

    int counter = 1;

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

    {

        for (int j = 0; j < vertexSize; j++)

        {

            if ((adjacency[i][j] == 1) && (discovered[j] == false))

            {

                output[counter] = j + 1;

                discovered[j] = true;

                counter++;

            }

        }

    }

    cout << "BFS : " << endl;

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

    {

        cout << output[i] << " ";

    }

}

Kompleksitas Waktu Asimptotik :

V : Jumlah vertex E : Jumlah edge

- Menandai setiap vertex belum dikunjungi : O(V)

- Menandai vertex awal telah dikunjungi lalu masukkan ke queue : O(1)

- Keluarkan vertex dari queue kemudian cetak : O(V)

- Kunjungi setiap vertex yang belum dikunjungi kemudian masukkan ke queue : O(E)

Maka :

T(n) = O(V) + O(1) + O(V) + O(E)

= O(maks(V,1)) + O(V) + O(E)

= O(V) + O(V) + O(E)

= O(maks(V,V)) + O(E))

= O(V) + O(E)

= O(V+E)

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!

/\*

    Nama        : Gede Bagus Darmagita

    Kelas       : B

    NPM         : 140810180068

\*/

#include <iostream>

#include <list>

using namespace std;

class Graph

{

    int N;

    list<int> \*adj;

    void DFSUtil(int u, bool visited[])

    {

        visited[u] = true;

        cout << u << " ";

        list<int>::iterator i;

        for (i = adj[u].begin(); i != adj[u].end(); i++)

        {

            if (!visited[\*i])

            {

                DFSUtil(\*i, visited);

            }

        }

    }

public:

    Graph(int N)

    {

        this->N = N;

        adj = new list<int>[N];

    }

    void addEdge(int u, int v)

    {

        adj[u].push\_back(v);

    }

    void DFS(int u)

    {

        bool \*visited = new bool[N];

        for (int i = 0; i < N; i++)

        {

            visited[i] = false;

        }

        DFSUtil(u, visited);

    }

};

int main()

{

    Graph Z(8);

    Z.addEdge(1, 2);

    Z.addEdge(1, 3);

    Z.addEdge(2, 3);

    Z.addEdge(2, 4);

    Z.addEdge(2, 5);

    Z.addEdge(3, 7);

    Z.addEdge(3, 8);

    Z.addEdge(4, 5);

    Z.addEdge(5, 3);

    Z.addEdge(5, 6);

    Z.addEdge(7, 8);

    cout << "\nDFS Traversal Starts from Node 1" << endl;

    Z.DFS(1);

    return 0;

}

Kompleksitas Waktu Asimptotik :

V : Jumlah vertex E : Jumlah edge

- Menandai vertex awal telah dikunjungi kemudian cetak : O(1)

- Rekursif untuk semua vertex : T(E/1)

- Tandai semua vertex belum dikunjungi : O(V)

- Rekursif untuk mencetak DFS : T(V/1)

Maka :

T(n) = O(1) + T(V/1)+ O(V) + T(V/1)

= O(1) + O(E) + O(V) + O(V)

= O(maks(1,E)) + O(V) + O(V)

= O(E) + O(V) + O(V)

= O(maks(V,V)) + O(E))

= O(V) + O(E)

= O(V+E)