**LAPORAN PRAKTIKUM 6**

**ANALISIS ALGORITMA**



Disusun oleh :

**Alvin**

**140810180013**

**Kelas A**

Program Studi S-1 Teknik Informatika Departemen Ilmu Komputer

Fakultas Matematika dan Ilmu Pengetahuan Alam

Universitas Padjadjaran

2020

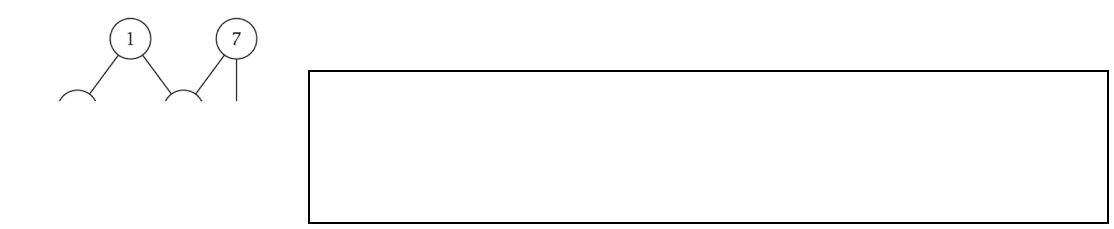
= 1 jika (u,v) adalah sebuah garis

Pendahuluan

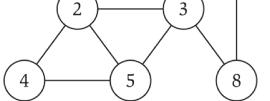
Graf Tak Berarah (Undirected Graf)

(Undirected) graph: G=(V,E)

* V = sekumpulan node (vertex, simpul, titik, sudut)
* E = sekumpulan edge (garis, tepi)
* Menangkap hubungan berpasangan antar objek.
* Parameter ukuran Graf: n = |V|, m=|E|



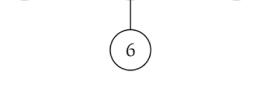
V = {1,2,3,4,5,6,7,8}



E = {(1,2), (1,3), (2,3), (2,4), (2,5), (3,5),(3,7), (3,8), (4,5), (5,6), (7,8)}

n = 8

M=11



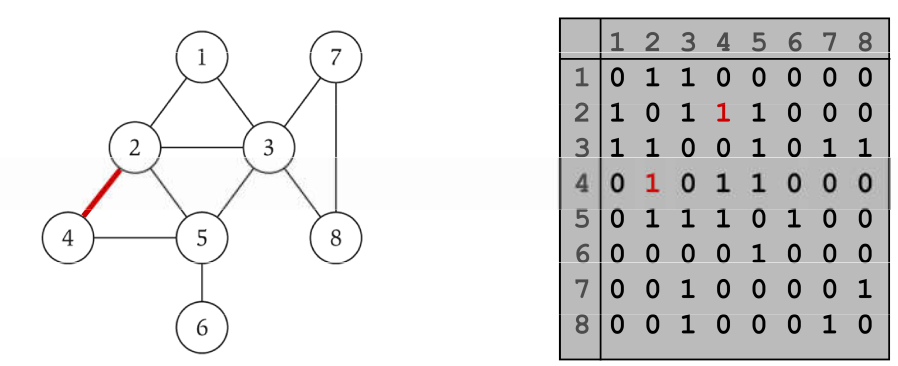
Dalam pemrograman, Graf dapat direpresentasikan dengan ***adjacency matrix*** dan ***adjacency list***

**Representasi Graf dengan Adjacency Matrix**

**Adjacency Matrix:** n-ke-n matriks dengan

* Dua representasi dari setiap sisi
* Ruang berukuran sebesar
* Memeriksa apakah (u, v) edge membutuhkan waktu Θ(1)

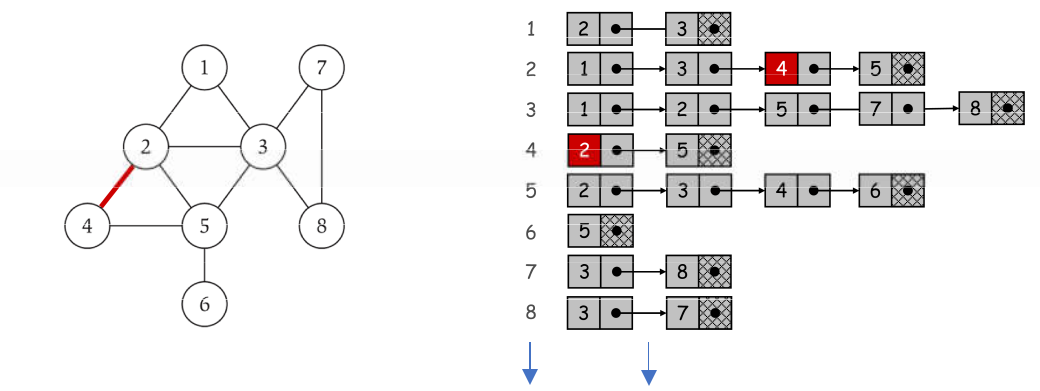
• Mengidentifikasi semua tepi membutuhkan Θ ( ) waktu



**Representasi Graf dengan Adjacency List**

**Adjacency List: node diindeks sebagai list**

* Dua representasi untuk setiap sisi
* Ukuran ruang m + n
* Memeriksa apakah (u, v) edge membutuhkan O (deg (u)). Degree = jumlah tetangga u.
* Mengidentifikasi semua tepi membutuhkan Θ(m + n).

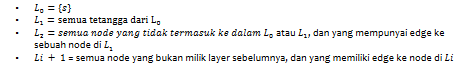


Array List

**Breadth First Search**

Intuisi BFS. Menjelajahi alur keluar dari s ke semua arah yang mungkin, tambahkan node satu “layer" sekaligus.

**Algoritma BFS**

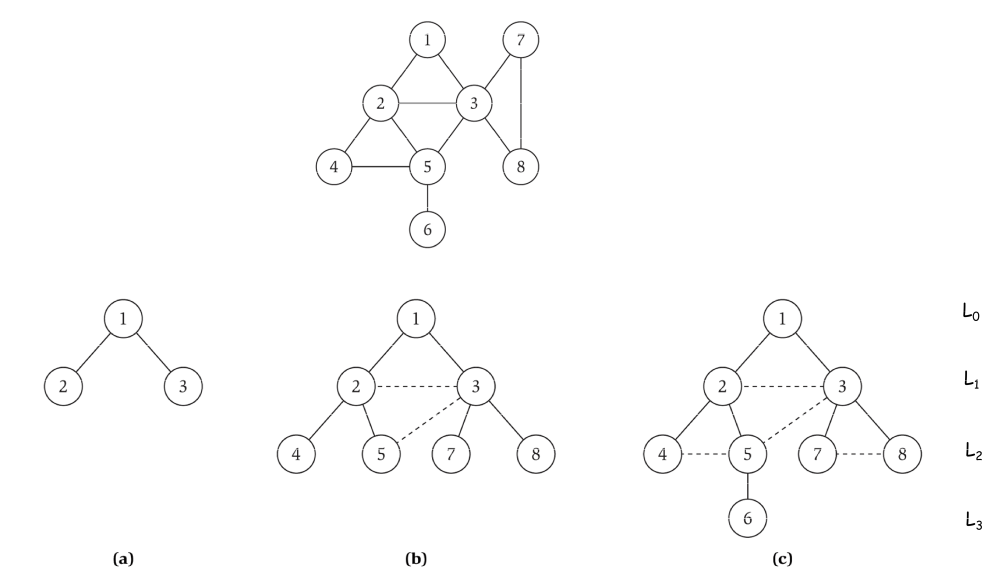


|  |
| --- |
|  |

Gambar 1. Ilustrasi algoritma BFS

**Teorema 1.0**

Untuk setiap , terdiri dari semua node pada jarak tepat ke dari . Ada path dari ke jika *t* muncul di beberapa layer.



Gambar 2. Ilustrasi pembentukan tree BFS dari undirected Graf

Implementasi BFS dalam Koding Program

* Adjacency list adalah representasi struktur data paling ideal untuk BFS
* Algoritma memeriksa setiap ujung yang meninggalkan node satu per satu. Ketika kita memindai edge yang meninggalkan u dan mencapai edge(u, v), kita perlu tahu apakah node v telah ditemukan sebelumnya oleh pencarian.
* Untuk menyederhanakan ini, kita maintain array yang ditemukan dengan panjang n dan mengatur

Discovered [v] = true segera setelah pencarian kita pertama kali melihat v. Algoritma BFS membangun

lapisan node L1, L2, ..., di mana Li adalah set node pada jarak dari sumber .

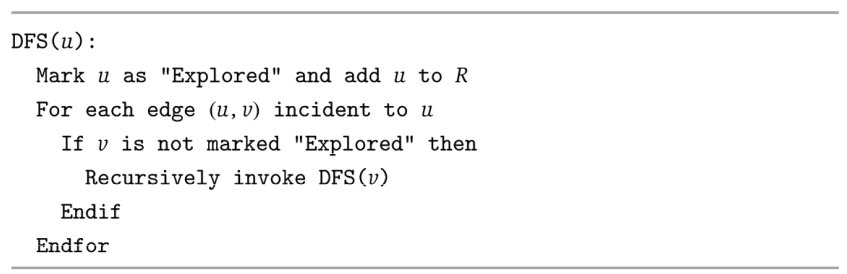
• Untuk mengelola node dalam layer , kami memiliki daftar [ ] untuk setiap = 0,1,2, ….

**Depth First Search**

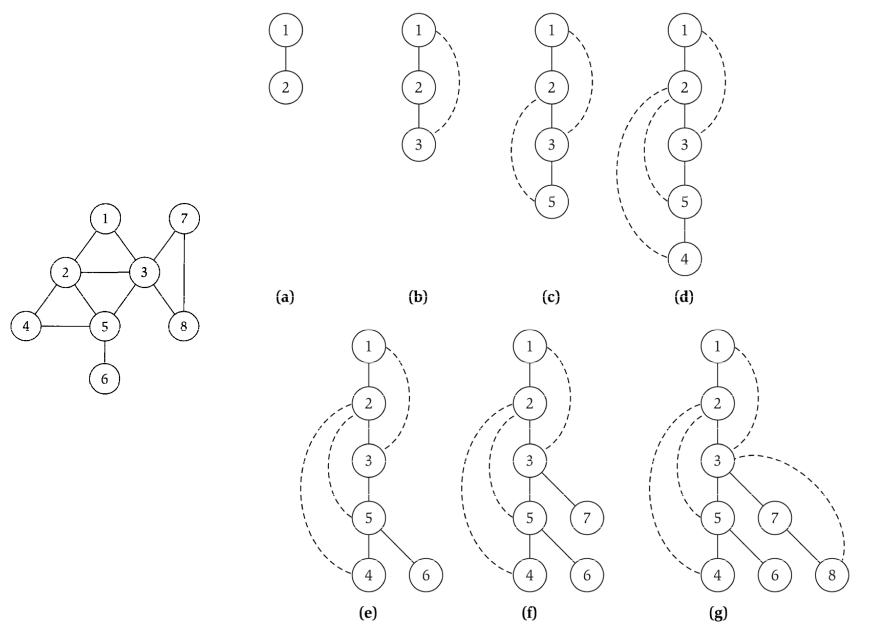
Algoritma BFS muncul, khususnya, sebagai cara tertentu mengurutkan node yang kita kunjungi — dalam lapisan berurutan, berdasarkan pada jarak node lain dari s. Metode alami lain untuk menemukan node yang dapat dijangkau dari s adalah pendekatan yang mungkin Anda dilakukan jika grafik G benar-benar sebuah labirin dari kamar yang saling berhubungan dan kita berjalan-jalan di dalamnya.

Kita akan mulai dari s dan mencoba edge pertama yang mengarah ke ke node v. Kita kemudian akan mengikuti edge pertama yang mengarah keluar dari v, dan melanjutkan dengan cara ini sampai kita mencapai "jalan buntu" —sebuah node di mana Anda sudah menjelajahi semua tetangganya. Kita kemudian akan mundur sampai kita mencapai node dengan tetangga yang belum dijelajahi, dan melanjutkan dari sana. Kita menyebutnya Depth-first search (DFS), karena ini mengeksplorasi G dengan masuk sedalam mungkin dan hanya mundur jika diperlukan.

DFS juga merupakan implementasi khusus dari algoritma component-growing generik yang dijelaskan sebelumnya. Kita dapat memulai DFS dari titik awal mana pun tetapi mempertahankan pengetahuan global tentang node yang telah dieksplorasi.



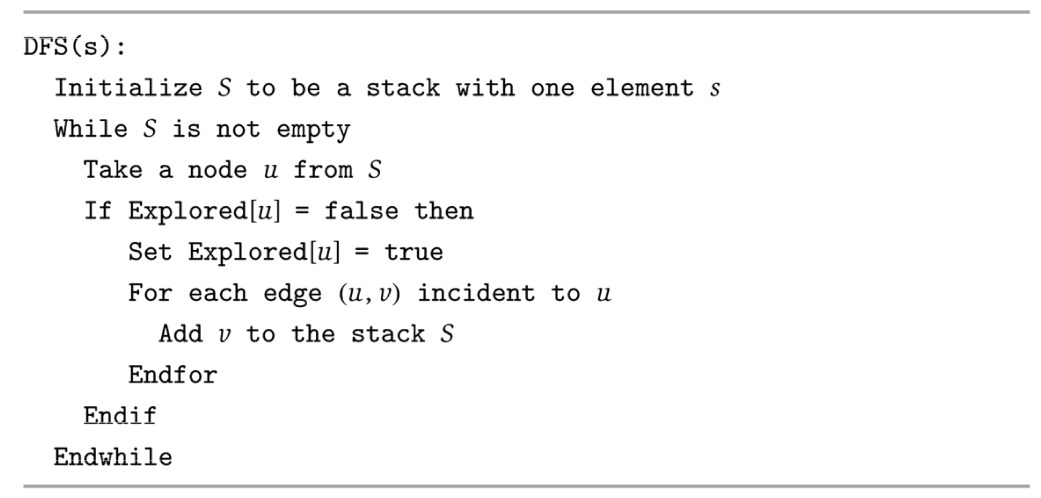
Untuk menerapkan ini pada problem konektivitas s-t, kita cukup mendeklarasikan semua node pada awalnya untuk tidak dieksplorasi, dan memanggil DFS (s).

Ada beberapa kesamaan dan beberapa perbedaan mendasar antara DFS dan BFS. Kesamaan didasarkan pada fakta bahwa mereka berdua membangun komponen terhubung yang mengandung s, dan bahwa mereka mencapai tingkat efisiensi yang serupa secara kualitatif. Sementara DFS akhirnya mengunjungi set node yang sama persis seperti BFS, ia biasanya melakukannya dalam urutan yang sangat berbeda; menyelidiki jalan panjang, berpotensi menjadi sangat jauh dari s, sebelum membuat cadangan untuk mencoba lebih dekat node yang belum dijelajahi.

Gambar 3. Ilustrasi pembentukan tree DFS dari undirected graph

**Implementasi BFS dalam Koding Program**

Implementasi DFS paling ideal adalah dengan menggunakan stack. Adapun algoritma DFS dengan stack adalah sebagai berikut:



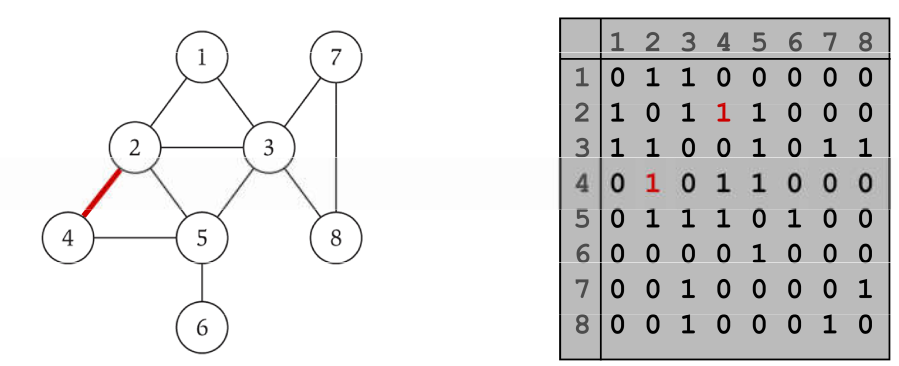
**Nama : Alvin**

**NPM : 140810180013**

**TUGAS-6**

**Tugas Anda**

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



* **Code**

*/\* Nama : Alvin*

*NPM : 140810180013*

*Kelas : A*

*Program : Representasi Graph dengan Matrix*

*\*/*

#include<iostream>

**using** **namespace** std**;**

**int** main**(){**

**int** m**[**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**,**1**,**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**}**

**};**

cout**<<"MATRIX-ADJACENCY\n"<<**endl**;**

**for(int** i**=**0**;**i**<**8**;**i**++){**

**for(int** j**=**0**;** j**<**8**;**j**++){**

cout**<<**m**[**i**][**j**]<<" ";**

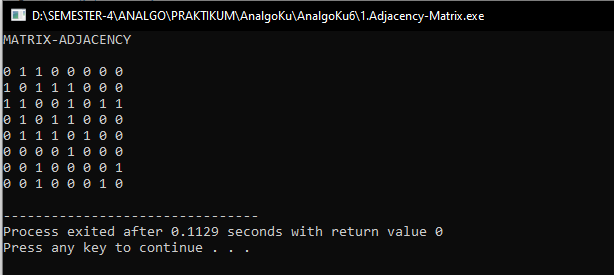
**}**

cout**<<**endl**;**

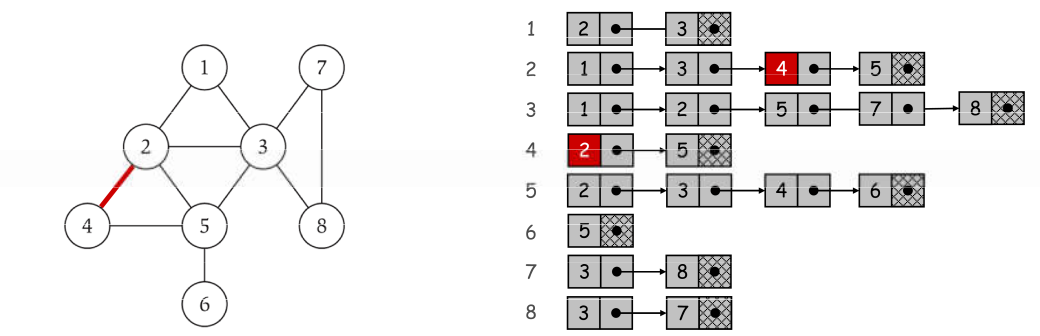
**}**

**}**

* **Screenshot**



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



* **Code**

*/\* Nama : Alvin*

*NPM : 140810180013*

*Kelas : A*

*Program : Representasi Graph dengan List*

*\*/*

#include<iostream>

#include<windows.h>

**using** **namespace** std**;**

**struct** adjacent**{**

**int** nodeAdj**;**

adjacent**\*** nextAdj**;**

**};**

**struct** elemen**{**

**int** node**;**

elemen**\*** next**;**

adjacent**\*** firstAdj**;**

**};**

**typedef** elemen**\*** pointerNode**;**

**typedef** adjacent**\*** pointerAdj**;**

**typedef** pointerNode list**;**

**void** createListNode**(**list**&** first**){**

first **=** NULL**;**

**}**

**void** createNode**(**pointerNode**&** pBaru**,int** vertex**){**

pBaru **=** **new** elemen**;**

pBaru**->**node **=** vertex**;**

pBaru**->**next **=** NULL**;**

pBaru**->**firstAdj **=** NULL**;**

**}**

**void** createAdjacent**(**pointerAdj**&** pBaru**,int** vertex**){**

pBaru **=** **new** adjacent**;**

pBaru**->**nodeAdj **=** vertex**;**

pBaru**->**nextAdj **=** NULL**;**

**}**

**void** insertAdjacent**(**pointerNode**&** curNode**,**pointerAdj pBaruAdj**){**

pointerAdj last**;**

**if(**curNode**->**firstAdj **==** NULL**){**

curNode**->**firstAdj **=** pBaruAdj**;**

**}else{**

last **=** curNode**->**firstAdj**;**

**while(**last**->**nextAdj **!=** NULL**){**

last **=** last**->**nextAdj**;**

**}**

last**->**nextAdj **=** pBaruAdj**;**

**}**

**}**

**void** insertElement**(**list**&** first**,** pointerNode pBaruNode**,** **int** size**){**

pointerNode last**;**

pointerAdj pBaruAdj**;**

**if(**first **==** NULL**){**

first **=** pBaruNode**;**

**}else{**

last **=** first**;**

**while(**last**->**next **!=** NULL**){**

last **=** last**->**next**;**

**}**

last**->**next **=** pBaruNode**;**

**}**

**if(**size**>**0**){**

cout**<<"Masukan node yang berhubungan dengan "<<**pBaruNode**->**node**<<" : "<<**endl**;**

**}**

**for(int** i **=** 0**;** i **<** size**;** i**++){**

**int** vertex**;**

cin**>>**vertex**;**

createAdjacent**(**pBaruAdj**,**vertex**);**

insertAdjacent**(**pBaruNode**,**pBaruAdj**);**

**}**

**}**

**void** output**(**list first**){**

pointerNode pOut**;**

pointerAdj pOutAdj**;**

**if(**first **==** NULL**){**

cout**<<"Tidak ada Node"<<**endl**;**

**}else{**

pOut **=** first**;**

**while(**pOut **!=** NULL**){**

cout**<<"Parent = "<<**pOut**->**node**<<**endl**;**

**if(**pOut**->**firstAdj **==** NULL**){**

cout**<<"Tidak ada adjacency"<<**endl**;**

**}else{**

pOutAdj **=** pOut**->**firstAdj**;**

cout**<<"Child = ";**

**while(**pOutAdj **!=** NULL**){**

cout**<<**pOutAdj**->**nodeAdj**<<" ";**

pOutAdj **=** pOutAdj**->**nextAdj**;**

**}**

**}**

cout**<<**endl**;**

pOut **=** pOut**->**next**;**

**}**

**}**

**}**

**int** main**(){**

list first**;**

pointerNode node**;**

cout**<<"===== MEMBUAT NODE =====\n"<<**endl**;**

createListNode**(**first**);**

createNode**(**node**,**1**);**

insertElement**(**first**,**node**,**2**);**

createNode**(**node**,**2**);**

insertElement**(**first**,**node**,**4**);**

createNode**(**node**,**3**);**

insertElement**(**first**,**node**,**5**);**

createNode**(**node**,**4**);**

insertElement**(**first**,**node**,**2**);**

createNode**(**node**,**5**);**

insertElement**(**first**,**node**,**4**);**

createNode**(**node**,**6**);**

insertElement**(**first**,**node**,**1**);**

createNode**(**node**,**7**);**

insertElement**(**first**,**node**,**2**);**

createNode**(**node**,**8**);**

insertElement**(**first**,**node**,**2**);**

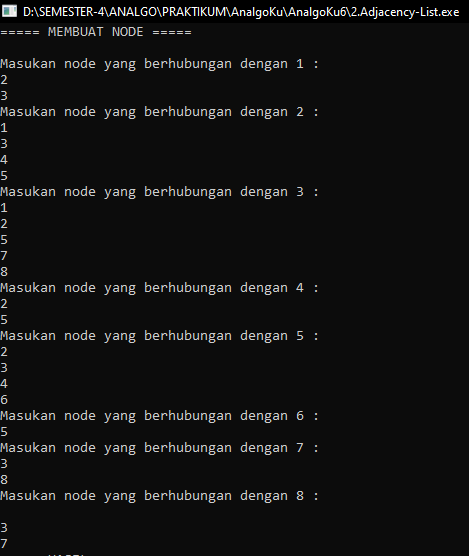
cout**<<"===== HASIL =====\n"<<**endl**;**

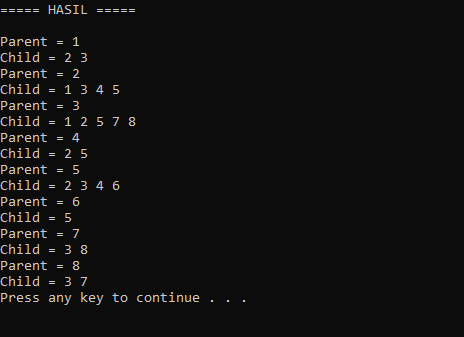
output**(**first**);**

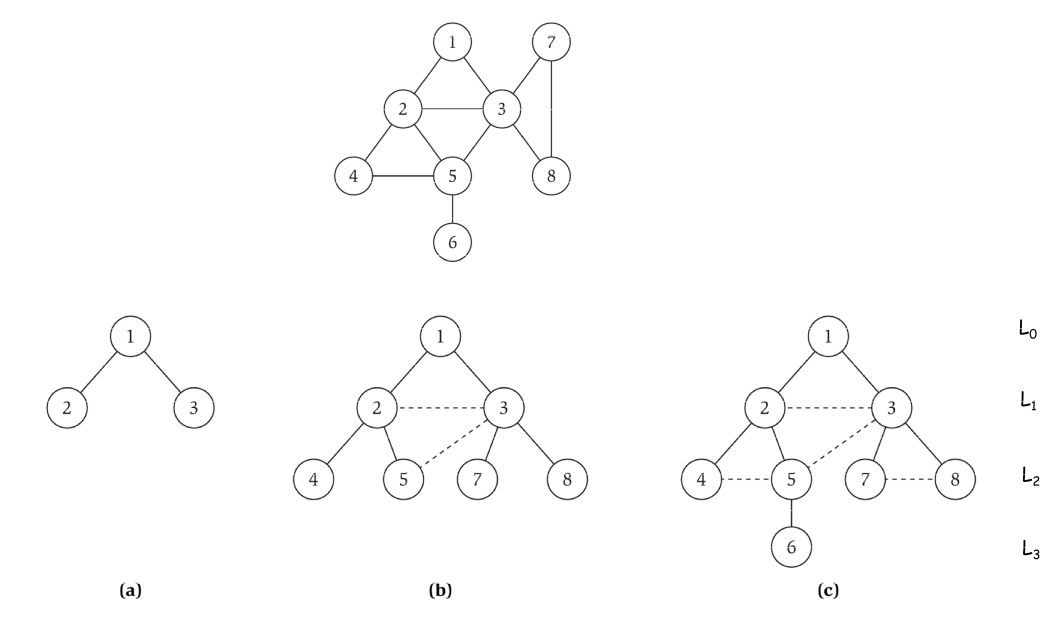
system**("pause");**

**}**

* **Screenshot**





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!

* **Code**

*/\* Nama : Alvin*

*NPM : 140810180013*

*Kelas : A*

*Program : Breadth First Search*

*\*/*

#include<iostream>

**using** **namespace** std**;**

**int** main**(){**

**int** vertexSize **=** 8**;**

**int** adjacency**[**8**][**8**]** **=** **{**

**{**0**,**1**,**1**,**0**,**0**,**0**,**0**,**0**},***//representasi graph dalam matrix adjacency*

**{**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**];**

*//inisialisasi start*

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**<<"=== HASIL DARI BREADTH FIRST SEARCH === "<<**endl**;**

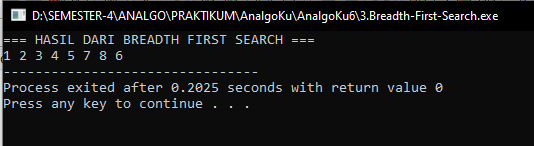
**for(int** i **=** 0**;** i **<** vertexSize**;** i**++){**

cout**<<**output**[**i**]<<" ";**

**}**

**}**

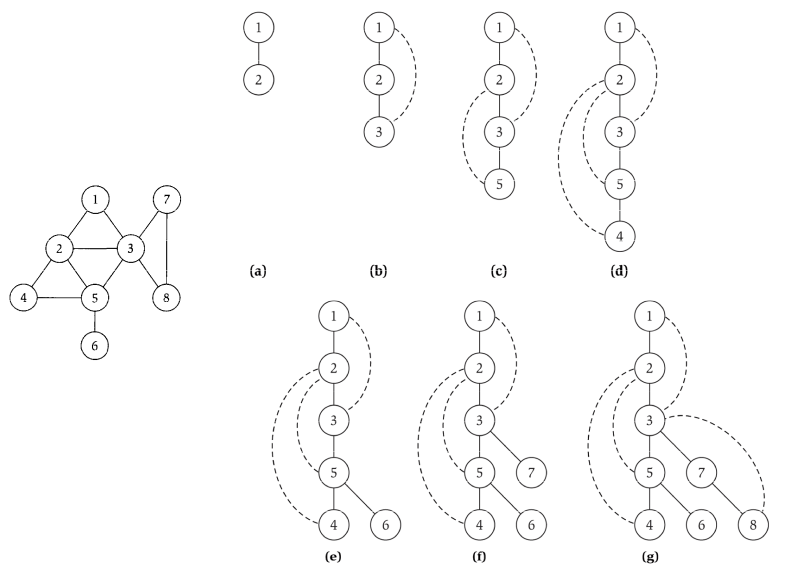
* **Screenshot**



* **Analisa**

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| ).

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!



* **Code**

*/\* Nama : Alvin*

*NPM : 140810180013*

*Kelas : A*

*Program : Depth First Search*

*\*/*

#include<bits/stdc++.h>

**using** **namespace** std**;**

**class** Graph

**{**

**int** V**;**

list**<int>** **\***adj**;**

**void** DFSUtil**(int** v**,** **bool** visited**[]);**

**public:**

Graph**(int** V**);**

**void** addEdge**(int** v**,** **int** w**);**

**void** DFS**(int** v**);**

**};**

Graph**::**Graph**(int** V**)**

**{**

**this->**V **=** V**;**

adj **=** **new** list**<int>[**V**];**

**}**

**void** Graph**::**addEdge**(int** v**,** **int** w**)**

**{**

adj**[**v**].**push\_back**(**w**);**

**}**

**void** Graph**::**DFSUtil**(int** v**,** **bool** visited**[])**

**{**

visited**[**v**]** **=** **true;**

cout **<<** v **<<** **" ";**

list**<int>::**iterator i**;**

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

**if** **(!**visited**[\***i**])**

DFSUtil**(\***i**,** visited**);**

**}**

**void** Graph**::**DFS**(int** v**)**

**{**

**bool** **\***visited **=** **new** **bool[**V**];**

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

visited**[**i**]** **=** **false;**

DFSUtil**(**v**,** visited**);**

**}**

**int** main**()**

**{**

Graph g**(**8**);** *//jumlah node*

g**.**addEdge**(**1**,** 2**);**

g**.**addEdge**(**1**,** 3**);**

g**.**addEdge**(**2**,** 3**);**

g**.**addEdge**(**2**,** 4**);**

g**.**addEdge**(**2**,** 5**);**

g**.**addEdge**(**3**,** 5**);**

g**.**addEdge**(**3**,** 7**);**

g**.**addEdge**(**3**,** 8**);**

g**.**addEdge**(**4**,** 5**);**

g**.**addEdge**(**5**,** 6**);**

g**.**addEdge**(**7**,** 8**);**

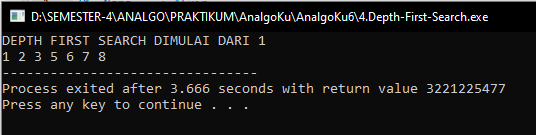
cout **<<** **"DEPTH FIRST SEARCH DIMULAI DARI 1 \n";**

g**.**DFS**(**1**);**

**return** 0**;**

**}**

* **Screenshot**



* **Analisa**

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.