Nama : Naufal Ariful Amri

NPM : 140810180009

1. Dengan menggunakan *undirected graph* dan *adjency matrix* buatlah koding programnya menggunakan bahasa c++
2. Dengan menggunakan *undirected graph* dan *adjency list* buatlah koding programnya menggunakan bahasa c++
3. Buatlah program Breadth First Search dari algoritma BFS yang telah diberikan. Kemudian uji coba program dengan menginputjan undirected graph. Hitung secara asimptotik berapa kompleksitas waktunya
4. Buatlah program Depth First Search dari algoritma DFS yang telah diberikan. Kemudian uji coba program dengan menginputjan undirected graph. Hitung secara asimptotik berapa kompleksitas waktunya

Jawab :

1. Code

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

int vertArr[20][20] ;
int count = 0;
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;
}
main(int argc, char* argv[]) {
   int v = 6;
   add_edge(0, 4);
   add_edge(0, 3);
   add_edge(1, 2);
   add_edge(1, 4);
   add_edge(1, 5);
   add_edge(2, 3);
   add_edge(2, 5);
   add_edge(5, 3);
   add_edge(5, 4);
   displayMatrix(v);
}
```

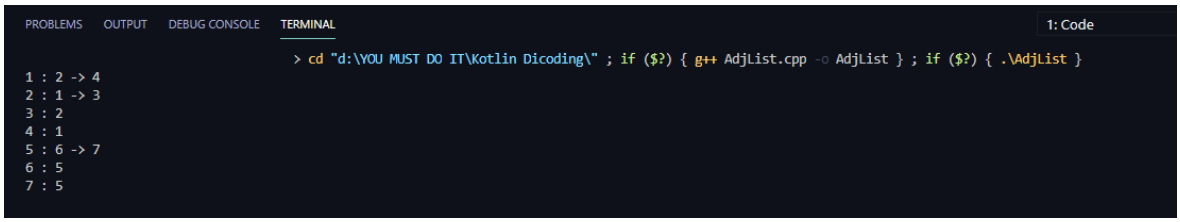
2. Code

```cpp
#include <iostream>
using namespace std ;
int array[10][10] ;
int count [10] ;

void add_graph(int a , int b, int array[10][10], int count[10]) {
     array[a][b] = 1 ;
     array[b][a] = 1 ;
     count[a] = 1 ;
     count[b] = 1 ;
}

void show_graph(int array[10][10]){
     for (int i = 1; i < 10; i++) {
          int temp = 0 ;
          if (count[i] == 1){
               cout << i << " : ";
          }
          for (int j = 1; j < 10; j++) {
               if(array[i][j] ==  1 ) {
                    if (temp > 0) {
                         cout << " -> " ;
                    }
                    cout << j  ;
                    temp++ ;
               }
          }
          cout << endl ;
     }
}

int main(int argc, char const *argv[]) {
     add_graph(1,2,array,count) ;
     add_graph(4,1,array,count) ;
     add_graph(3,2,array,count) ;
     add_graph(6,5,array,count) ;
     add_graph(7,5,array,count) ;
     show_graph(array) ;
     return 0;
}
```



3. Code

```
#include<stdio.h>
#include<stdlib.h>
#include <iostream>

using namespace std ;
#define MAX 100

#define initial 1
#define waiting 2
#define visited 3

int n;
int adj[MAX][MAX];
int state[MAX];
int queue[MAX], front = -1,rear = -1;



void BF_Traversal() {
     int v;
     for(v = 0; v < n; v++)
          state[v] = initial;

     cout << "Enter Start Vertex for BFS: \n" ;
     scanf("%d", &v);
     BFS(v);
}

void BFS(int v)
{
     int i;

     insert_queue(v);
     state[v] = waiting;

     while(!isEmpty_queue()) {
          v = delete_queue( );
          printf("%d ",v);
          state[v] = visited;

          for(i=0; i<n; i++) {
               if(adj[v][i] == 1 && state[i] == initial) {
                    insert_queue(i);
                    state[i] = waiting;
               }
          }
```

```
     }
     cout << "\n";
}

void insert_queue(int vertex) {
     if(rear == MAX-1)
          printf("Queue Overflow\n");
     else
     {
          if(front == -1)
               front = 0;
          rear = rear+1;
          queue[rear] = vertex ;
     }
}

int isEmpty_queue()
{
     if(front == -1 || front > rear)
          return 1;
     else
          return 0;
}

int delete_queue()
{
     int delete_item;
     if(front == -1 || front > rear)
     {
          printf("Queue Underflow\n");
          exit(1);
     }

     delete_item = queue[front];
     front = front+1;
     return delete_item;
}

void create_graph()
{
     int count,max_edge,origin,destin;

     printf("Enter number of vertices : ");
     scanf("%d",&n);
     max_edge = n*(n-1);

     for(count=1; count<=max_edge; count++)
     {
          printf("Enter edge %d( -1 -1 to quit ) : ",count);
          scanf("%d %d",&origin,&destin);

          if((origin == -1) && (destin == -1))
               break;

          if(origin>=n || destin>=n || origin<0 || destin<0)
          {
               printf("Invalid edge!\n");
               count--;
```

```
            }
            else
            {
                 adj[origin][destin] = 1;
            }
       }
}

int main()
{
     create_graph();
     BF_Traversal();
     return 0;
}
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

                                > cd "d:\YOU MUST DO IT\Kotlin Dicoding\" ; if ($?) { g++ BFS.cpp -o BFS } ; if ($?) { .\BFS }
Enter number of vertices : 6
Enter edge 1( -1 -1 to quit ) : 1 4
Enter edge 2( -1 -1 to quit ) : 2 3
Enter edge 3( -1 -1 to quit ) : 5 1
Enter edge 4( -1 -1 to quit ) : 2 4
Enter edge 5( -1 -1 to quit ) : 3 5
Enter edge 6( -1 -1 to quit ) : 1 2
Enter edge 7( -1 -1 to quit ) : 1 3
Enter edge 8( -1 -1 to quit ) : -1 -1
Enter Start Vertex for BFS:
1
1 2 3 4 5
PS D:\YOU MUST DO IT\Kotlin Dicoding>
```

Di sini, V adalah jumlah simpul dan E adalah jumlah tepi. Kompleksitas waktu adalah O (V + E) karena kita melintasi setiap simpul grafik yang membutuhkan waktu O (V) dan untuk setiap simpul, kita menambahkan simpul anak-anaknya, jadi berapa banyak simpul anak yang dimiliki simpul? ia memiliki node anak-anak sebanyak itu memiliki tepi keluar dari itu. Misalnya, 'A' memiliki 3 simpul anak karena ada 3 tepi yang keluar dan 'B has memiliki 2 simpul anak karena ada 2 tepi yang keluar dan sebagainya. Jadi, ini membutuhkan waktu O (E). Itu membuat kompleksitas waktu O (V) + O (E) -> O (V + E)

4. Code

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std ;

int cost[10][10],i,j,k,n,qu[10],front,rare,v,visit[10],visited[10];
int main() {
    int m;
    //clrscr();
    cout <<"Enter no of vertices:";
    cin >> n;
    cout <<"Enter no of edges:";
    cin >> m;
    cout <<"\nEDGES \n";
```

```
    for(k=1; k<=m; k++)
    {
        cin >>i>>j;
        cost[i][j]=1;
    }
    cout <<"Enter initial vertex to traverse from:";
    cin >> v;
    cout <<"Visitied vertices:";
    cout << v <<" ";
    visited[v] = 1;
    k = 1;
    while(k < n)
    {
        for(j=1; j<=n; j++)
            if(cost[v][j] != 0 && visited[j] != 1 && visit[j] != 1){
                visit[j]=1;
                qu[rare++]=j;
            }
        v=qu[front++];
        cout << v << " ";
        k++;
        visit[v]=0;
        visited[v]=1;
    }
    getch();
    return 0;
}
```

```
> cd "d:\YOU MUST DO IT\Kotlin Dicoding\" ; if ($?) { g++ DFS.cpp -o DFS } ; if ($?) { .\DFS }
Enter no of vertices:4
Enter no of edges:4

EDGES
1 2
2 3
4 1
1 3
Enter initial vertex to traverse from:3
Visitied vertices:3 0 0 0
```

DFS dapat diimplementasikan dalam dua pendekatan, (i) rekursi atau (ii) while. Gagasannya sama: alih-alih menyimpan node perantara dalam antrian, kami mendorongnya ke dalam tumpukan. Ketika kami memunculkan tumpukan, kami selalu mendapatkan simpul yang datang terbaru.

Misal b = faktor percabangan rata-rata, m = kedalaman maksimum pohon pencarian.

Kompleksitas waktu = O (b^m).

Kompleksitas ruang = O (m.b) jika ketika kita mengunjungi sebuah node, kita push.stack semua tetangganya. O (m) jika kita hanya mendorong. Salah satu anak ketika kita memperluas perbatasan.