**LAPORAN PRAKTIKUM**

**SESI X**

**PRAKTIKUM COMP6362 – DATA STRUCTURES**

**KELAS BC20**



Oleh :

2440059495 – Bismo Agung Tri Achmad Bramantyo

**SEMESTER GENAP 2020/2021  
BINA NUSANTARA UNIVERSITY**

**MALANG**

## **Kode Program**

|  |
| --- |
| Source Code |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #define V 5  //case 1  void init(int arr[][V]) {    int i, j;    for (i = 0; i < V; i++)      for (j = 0; j < V; j++)        arr[i][j] = 0;  }  void addEdge(int arr[][V], int i, int j) {    arr[i][j] = 1;    arr[j][i] = 1;  }  void print\_AdjMatrix(int arr[][V]) {    int i, j;    printf("\nAdjacency Matrix For This Graph\n");    printf("-----------------------------------\n\n");      for (i =0; i<V ; i++){      printf("    %d", i);    }    printf("\n");    printf("--------------------------\n");    for (i = 0; i < V; i++) {      printf("%d:  ", i);      for (j = 0; j < V; j++) {        printf("%d    ", arr[i][j]);      }      printf("\n");    }    printf("--------------------------\n");  }  //case 2  struct node {      int vertex;    struct node\* next;    };  struct node\* createNode(int);  struct Graph {    int numVertices;    struct node\*\* adjLists;  };  // membuat node  struct node\* create\_Node(int v) {    struct node\* new\_Node = malloc(sizeof(struct node));    new\_Node->vertex = v;    new\_Node->next = NULL;    return new\_Node;  }  // membuat graph  struct Graph\* create\_A\_Graph(int vertices) {      struct Graph\* graph = malloc(sizeof(struct Graph));    graph->numVertices = vertices;    graph->adjLists = malloc(vertices \* sizeof(struct node\*));    int i;    for (i = 0; i < vertices; i++)      graph->adjLists[i] = NULL;    return graph;  }  // Add edge  void add\_Edge(struct Graph\* graph, int a, int b) {      // Add edge dari a ke b    struct node\* new\_Node = create\_Node(a);    new\_Node->next = graph->adjLists[b];    graph->adjLists[b] = new\_Node;    // Add edge dari b ke a    new\_Node = create\_Node(b);    new\_Node->next = graph->adjLists[a];    graph->adjLists[a] = new\_Node;  }  // Print graph  void print\_Graph(struct Graph\* graph) {    int v; //vertices    printf("\n Adjacency List of this Graph\n");    printf(" -------------------------------\n");    for (v = 0; v < graph->numVertices; v++)    {      struct node\* temp = graph->adjLists[v];      printf("\n Vertex %d: ", v);      while (temp != NULL) {        printf("%d -> ", temp->vertex);        temp = temp->next;        if (temp == NULL){          printf("NULL");      }      }      printf("\n");    }  }  //case 3;  int count[V];  void count\_degree(int a[][V]){      int i, j;      for(i=0; i<V ; i++){      for(j=0 ; j<V ; j++){        if(j < V){        if(a[i][j] == 1 && a[j][i] == 1){ //jika di matrix bernilai 1          count[i]++;        }        else{          count[i] = count[i] + 0;        }         }      }    }      printf("\nDegree of all vertices\n");    printf("-----------------------\n");      for(i=0 ; i<V ; i++){      printf("Degree of vertex %d: ", i);      printf("%d\n", count[i]);    }      for(i=0 ; i<V ; i++){      count[i]=0; //variabel count di nol kan disetiap index    }    }  //case 4  struct queue{    int items[V];    int front;    int rear;  };  struct queue\* create\_Queue();  void enqueue(struct queue\* q, int);  int dequeue(struct queue\* q);  void display(struct queue\* q);  int isEmpty(struct queue\* q);  void print\_Queue(struct queue\* q);  struct node\_BFS{    int vertex;    struct node\_BFS\* next;  };  struct node\_BFS\* createNode\_BFS(int);  struct Graph\_BFS{    int numVertices;    struct node\_BFS\*\* adjLists;    int\* visited;  };  // BFS algorithm  void bfs(struct Graph\_BFS\* graph, int startVertex) {    struct queue\* q = create\_Queue();    graph->visited[startVertex] = 1;    enqueue(q, startVertex);    while (!isEmpty(q)) {      print\_Queue(q);      int currentVertex = dequeue(q);      printf("\n                            Visited %d\n", currentVertex);      struct node\_BFS\* temp = graph->adjLists[currentVertex];      while (temp) {        int adjVertex = temp->vertex;        if (graph->visited[adjVertex] == 0) {          graph->visited[adjVertex] = 1;          enqueue(q, adjVertex);        }        temp = temp->next;      }    }  }  // Creating a node  struct node\_BFS\* createNode\_BFS(int v) {    struct node\_BFS\* new\_Node = malloc(sizeof(struct node\_BFS));    new\_Node->vertex = v;    new\_Node->next = NULL;    return new\_Node;  }  // Creating a graph  struct Graph\_BFS\* createGraph(int vertices) {    struct Graph\_BFS\* graph = malloc(sizeof(struct Graph));    graph->numVertices = vertices;    graph->adjLists = malloc(vertices \* sizeof(struct node\*));    graph->visited = malloc(vertices \* sizeof(int));    int i;    for (i = 0; i < vertices; i++) {      graph->adjLists[i] = NULL;      graph->visited[i] = 0;    }    return graph;  }  // Add edge  void addEdge\_BFS(struct Graph\_BFS\* graph, int src, int dest) {    // Add edge from src to dest    struct node\_BFS\* new\_Node = createNode\_BFS(dest);    new\_Node->next = graph->adjLists[src];    graph->adjLists[src] = new\_Node;    // Add edge from dest to src    new\_Node = createNode\_BFS(src);    new\_Node->next = graph->adjLists[dest];    graph->adjLists[dest] = new\_Node;  }  // Create a queue  struct queue\* create\_Queue() {    struct queue\* q = malloc(sizeof(struct queue));    q->front = -1;    q->rear = -1;    return q;  }  // Check if the queue is empty  int isEmpty(struct queue\* q) {    if (q->rear == -1)      return 1;    else      return 0;  }  // Adding elements into queue  void enqueue(struct queue\* q, int value) {    if (q->rear == V - 1)      printf("\nQueue is Full\n");    else {      if (q->front == -1)        q->front = 0;      q->rear++;      q->items[q->rear] = value;    }  }  // Removing elements from queue  int dequeue(struct queue\* q) {    int item;    if (isEmpty(q)) {      printf("Queue is empty");      item = -1;    } else {      item = q->items[q->front];      q->front++;      if (q->front > q->rear) {        q->front = q->rear = -1;      }    }    return item;  }  // Print the queue  void print\_Queue(struct queue\* q) {      int i = q->front;    if (isEmpty(q)) {      printf("Queue is empty");    } else {      printf("\nQueue contains ");      for (i = q->front; i < q->rear + 1; i++) {        printf("%d ", q->items[i]);      }    }  }  //case 5  struct node\_DFS{    int vertex;    struct node\_DFS\* next;  };  struct node\_DFS\* createNode\_DFS(int v);  struct Graph\_DFS{    int numVertices;    int\* visited;    struct node\_DFS\*\* adjLists;  };  // DFS algo  void DFS(struct Graph\_DFS\* graph, int vertex) {    struct node\_DFS\* adjList = graph->adjLists[vertex];    struct node\_DFS\* temp = adjList;    graph->visited[vertex] = 1;    printf("Visited %d \n", vertex);    while (temp != NULL) {      int connectedVertex = temp->vertex;      if (graph->visited[connectedVertex] == 0) {        DFS(graph, connectedVertex);      }      temp = temp->next;    }  }  // Create a node  struct node\_DFS\* createNode\_DFS(int v) {    struct node\_DFS\* new\_Node = malloc(sizeof(struct node));    new\_Node->vertex = v;    new\_Node->next = NULL;    return new\_Node;  }  // Create graph  struct Graph\_DFS\* createGraph\_DFS(int vertices) {    struct Graph\_DFS\* graph = malloc(sizeof(struct Graph\_DFS));    graph->numVertices = vertices;    graph->adjLists = malloc(vertices \* sizeof(struct node\_DFS\*));    graph->visited = malloc(vertices \* sizeof(int));    int i;    for (i = 0; i < vertices; i++) {      graph->adjLists[i] = NULL;      graph->visited[i] = 0;    }    return graph;  }  // Add edge  void addEdge\_DFS(struct Graph\_DFS\* graph, int src, int dest) {    // Add edge from src to dest    struct node\_DFS\* new\_Node = createNode\_DFS(dest);    new\_Node->next = graph->adjLists[src];    graph->adjLists[src] = new\_Node;    // Add edge from dest to src    new\_Node = createNode\_DFS(src);    new\_Node->next = graph->adjLists[dest];    graph->adjLists[dest] = new\_Node;  }  int main() {      int parameter = 1;    int menu = 0;    int adjMatrix[V][V]; //matrix untuk case 1    struct Graph\* graph = create\_A\_Graph(V); //inisiasi case 2    struct Graph\_BFS\* graph\_BFS = createGraph(V); //inisiasi graph untuk case 4    struct Graph\_DFS\* graph\_DFS = createGraph\_DFS(V);      while (parameter == 1){      printf("GRAPH REPRESENTATION AND TRASVERSAL\n");    printf("-------------------------------------\n\n\n");    printf("        1   ---   2\n");    printf("         \\       / \\   \n");    printf("          4 --- 3 -- 0 \n\n");    printf("1. Show Adjancency Matrix\n");    printf("2. Show Adjacency List\n");    printf("3. Show Degree of all vertices\n");    printf("4. Show BFS Traversal from vertex 0\n");    printf("5. show DFS Traversal from vertex 0\n");    printf("6. Exit\n\n");    printf(">> Input Choice: ");    scanf("%d", & menu);      switch(menu){        case 1:            init(adjMatrix);              addEdge(adjMatrix, 0, 3);              addEdge(adjMatrix, 1, 2);              addEdge(adjMatrix, 1, 4);              addEdge(adjMatrix, 2, 3);              addEdge(adjMatrix, 2, 4);              addEdge(adjMatrix, 3, 4);              print\_AdjMatrix(adjMatrix);              getchar();              getchar();              system("cls");        break;        case 2:          //input edge di soal (konstan)                add\_Edge(graph, 4, 2);        add\_Edge(graph, 4, 3);              add\_Edge(graph, 4, 1);        add\_Edge(graph, 3, 2);        add\_Edge(graph, 2, 1);              add\_Edge(graph, 0, 3);                print\_Graph(graph);              getchar();              getchar();              system("cls");        break;        case 3:        init(adjMatrix);              addEdge(adjMatrix, 0, 3);              addEdge(adjMatrix, 1, 2);              addEdge(adjMatrix, 1, 4);              addEdge(adjMatrix, 2, 3);              addEdge(adjMatrix, 2, 4);              addEdge(adjMatrix, 3, 4);                count\_degree(adjMatrix);              getchar();              getchar();              system("cls");        break;        case 4:                addEdge\_BFS(graph\_BFS, 4, 2);        addEdge\_BFS(graph\_BFS, 4, 3);              addEdge\_BFS(graph\_BFS, 4, 1);        addEdge\_BFS(graph\_BFS, 3, 2);        addEdge\_BFS(graph\_BFS, 2, 1);              addEdge\_BFS(graph\_BFS, 0, 3);                printf("\n\n  BFS Traversal\n");              printf("  --------------\n");              bfs(graph\_BFS, 0);              getchar();              getchar();              system("cls");        break;        case 5:          addEdge\_DFS(graph\_DFS, 0, 3);        addEdge\_DFS(graph\_DFS, 1, 2);        addEdge\_DFS(graph\_DFS, 2, 3);        addEdge\_DFS(graph\_DFS, 4, 2);        addEdge\_DFS(graph\_DFS, 4, 3);        addEdge\_DFS(graph\_DFS, 4, 1);                printf("\nDFS Traversal\n");              printf("--------------\n\n");              DFS(graph\_DFS, 0);              getchar();              getchar();              system("cls");        break;      case 6:        printf("\n-----Programm Closed Thank you-----\n");        parameter = 0;        break;      default:        system("cls");        break;    }    }  } |

## **Deskripsi Program**

1. Library

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

1. Variabel Universal

#define V 5

V adalah sebuah integer untuk menampung nilai yang mengandung jumlah vertex, karena di soal konstan jadi saya gunakan define.

int count[V];

Variabel count bertipe integer dan memiliki indeks, berfungsi sebagai variabel penampung nilai degree dari vertex digunakan pada case 3 pada fungsi main.

1. Struct dan fungsi setiap case.
2. Case 1

void init(int arr[][V]) {

  int i, j;

  for (i = 0; i < V; i++)

    for (j = 0; j < V; j++)

      arr[i][j] = 0;

}

Fungsi init digunakan untuk inisiasi matrix dalam menentukan adjacency matrix, dengan parameter int arr[][V] , yang nantinya pada setiap kolom dan baris akan diisi angka 0 berukuran V x V.

void addEdge(int arr[][V], int i, int j) {

  arr[i][j] = 1;

  arr[j][i] = 1;

}

Addedge berfungsi untuk memberikan nilai 1 kepada setiap vertex yang berhubungan (berdasarkan edge).matrix yang tadinya hanya berisi 0 akan diubah nilai nya menjadi 1, apabila vertex nya berhubungan.

void print\_AdjMatrix(int arr[][V]) {

  int i, j;

  printf("\nAdjacency Matrix For This Graph\n");

  printf("-----------------------------------\n\n");

  for (i =0; i<V ; i++){

    printf("    %d", i);

  }

  printf("\n");

  printf("--------------------------\n");

  for (i = 0; i < V; i++) {

    printf("%d:  ", i);

    for (j = 0; j < V; j++) {

      printf("%d    ", arr[i][j]);

    }

    printf("\n");

  }

  printf("--------------------------\n");

}

Fungsi print\_AdjMatrix berguna untuk mencetak hasil akhir dari adjacency matrix, jadi program akan mencetak index baik secara vertikal maupun horizontal, baru setelah itu akan dicetak matrix yang sudah ada nilai dari masing masing vertex nya, matriks akan berukuran V x V yang sudah ditentukan.

1. Case 2

struct node {

  int vertex;

  struct node\* next;

};

struct node\* createNode(int);

struct node untuk mendefinisikan apa saja yang ada di dalam node tersebut, ada variabel vertex bertipe integer dan juga pointer next, selain itu juga pendefinisian struct untuk membuat node baru.

struct Graph {

  int numVertices;

  struct node\*\* adjLists;

};

Struct graph untuk mendefinisikan graph yang akan dipakai saat menghitung Adjacency list. Ada variabel numvertices bertipe integer dan juga pointer matrix 2 dimensi adjlist.

struct node\* create\_Node(int v) {

  struct node\* new\_Node = malloc(sizeof(struct node));

  new\_Node->vertex = v;

  new\_Node->next = NULL;

  return new\_Node;

}

Create node berfungsi untuk membuat node baru dengan passign parameter int v, alokasi memori akan dilakukan lalu data vertex newnode akan diisi oleh v, lalu pointer next nya di null kan.

struct Graph\* create\_A\_Graph(int vertices) {

  struct Graph\* graph = malloc(sizeof(struct Graph));

  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices \* sizeof(struct node\*));

  int i;

  for (i = 0; i < vertices; i++)

    graph->adjLists[i] = NULL;

  return graph;

}

Create\_A\_Graph berfungsi untuk membuat graph yang menampung Adj list, pengalokasian memori akan dilakukan kemudian pada data numVertices atau jumlah vertices akan di subtitusi dati int vertices(parameter) lalu pengalokasian memori akan dilakuakn untuk data adj list, kemudian data adj list di setiap index akan di null kan terlebih dahulu.

void add\_Edge(struct Graph\* graph, int a, int b) {

  // Add edge dari a ke b

  struct node\* new\_Node = create\_Node(a);

  new\_Node->next = graph->adjLists[b];

  graph->adjLists[b] = new\_Node;

  // Add edge dari b ke a

  new\_Node = create\_Node(b);

  new\_Node->next = graph->adjLists[a];

  graph->adjLists[a] = new\_Node;

}

Add\_edge berfungsi untuk membuat edge secara bolak balik dari a ke b maupun b ke a.

void print\_Graph(struct Graph\* graph) {

  int v; //vertices

  printf("\n Adjacency List of this Graph\n");

  printf(" -------------------------------\n");

  for (v = 0; v < graph->numVertices; v++)

  {

    struct node\* temp = graph->adjLists[v];

    printf("\n Vertex %d: ", v);

    while (temp != NULL) {

      printf("%d -> ", temp->vertex);

      temp = temp->next;

      if (temp == NULL){

        printf("NULL");

    }

    }

    printf("\n");

  }

}

Print\_graph berfungsi untuk menampilkan graph (adj list) yang sudah dibuat tadi, program akan mencetak adj list sampai temp NULL dan apabila temp NULL maka akan dicetak NULL di akhir.

1. Case 3

void count\_degree(int a[][V]){

  int i, j;

  for(i=0; i<V ; i++){

    for(j=0 ; j<V ; j++){

      if(j < V){

      if(a[i][j] == 1 && a[j][i] == 1){ //jika di matrix bernilai 1

        count[i]++;

      }

      else{

        count[i] = count[i] + 0;

      }

       }

    }

  }

  printf("\nDegree of all vertices\n");

  printf("-----------------------\n");

  for(i=0 ; i<V ; i++){

    printf("Degree of vertex %d: ", i);

    printf("%d\n", count[i]);

  }

  for(i=0 ; i<V ; i++){

    count[i]=0; //variabel count di nol kan disetiap index

  }

}

Count degree berfungsi untuk menghitung nilai degree dari setiap vertex, program akan mengecek keseluruhan jumlah vertex lalu pada setiap index akan dicek apabial a[i][j] dan a[j][i] sama sama bernilai 1, maka count pada indeks tersebut akan ditambahkan 1 nilai nya, lalu program akan mencetak keseluruhan nilai tersebut. Dan setelah itu program akan kembali megkosong kan int count di semua index agar tidak terjadi penambahan berulang pada saat program dijalankan untuk kedua kalinya.

1. Case 4

struct queue{

  int items[V];

  int front;

  int rear;

};

struct queue\* create\_Queue();

void enqueue(struct queue\* q, int);

int dequeue(struct queue\* q);

void display(struct queue\* q);

int isEmpty(struct queue\* q);

void print\_Queue(struct queue\* q);

struct queue adalah struct untuk queue yang digunakan di BFS

lalu ada pendefinisian beberapa fungsi.

struct node\_BFS{

  int vertex;

  struct node\_BFS\* next;

};

struct node\_BFS\* createNode\_BFS(int);

struct Graph\_BFS{

  int numVertices;

  struct node\_BFS\*\* adjLists;

  int\* visited;

};

Node\_BFS adalah struct yang mendefinisikan node di dalam proses BFS, memiliki int vertex lalu pointer next dan memiliki struct lain yaitu create node. Graph\_BFS adalah struct untuk mendefinisikan graph dari BFS, memiliki int numvertices(jumlah vertex) dan memiliki pinter untuk matrix 2 dimensi dan pointer visited.

void bfs(struct Graph\_BFS\* graph, int startVertex) {

  struct queue\* q = create\_Queue();

  graph->visited[startVertex] = 1;

  enqueue(q, startVertex);

  while (!isEmpty(q)) {

    print\_Queue(q);

    int currentVertex = dequeue(q);

    printf("\n                            Visited %d\n", currentVertex);

    struct node\_BFS\* temp = graph->adjLists[currentVertex];

    while (temp) {

      int adjVertex = temp->vertex;

      if (graph->visited[adjVertex] == 0) {

        graph->visited[adjVertex] = 1;

        enqueue(q, adjVertex);

      }

      temp = temp->next;

    }

  }

}

// Creating a node

struct node\_BFS\* createNode\_BFS(int v) {

  struct node\_BFS\* new\_Node = malloc(sizeof(struct node\_BFS));

  new\_Node->vertex = v;

  new\_Node->next = NULL;

  return new\_Node;

}

// Creating a graph

struct Graph\_BFS\* createGraph(int vertices) {

  struct Graph\_BFS\* graph = malloc(sizeof(struct Graph));

  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices \* sizeof(struct node\*));

  graph->visited = malloc(vertices \* sizeof(int));

  int i;

  for (i = 0; i < vertices; i++) {

    graph->adjLists[i] = NULL;

    graph->visited[i] = 0;

  }

  return graph;

}

// Add edge

void addEdge\_BFS(struct Graph\_BFS\* graph, int src, int dest) {

  // Add edge from src to dest

  struct node\_BFS\* new\_Node = createNode\_BFS(dest);

  new\_Node->next = graph->adjLists[src];

  graph->adjLists[src] = new\_Node;

  // Add edge from dest to src

  new\_Node = createNode\_BFS(src);

  new\_Node->next = graph->adjLists[dest];

  graph->adjLists[dest] = new\_Node;

}

// Create a queue

struct queue\* create\_Queue() {

  struct queue\* q = malloc(sizeof(struct queue));

  q->front = -1;

  q->rear = -1;

  return q;

}

// Check if the queue is empty

int isEmpty(struct queue\* q) {

  if (q->rear == -1)

    return 1;

  else

    return 0;

}

// Adding elements into queue

void enqueue(struct queue\* q, int value) {

  if (q->rear == V - 1)

    printf("\nQueue is Full\n");

  else {

    if (q->front == -1)

      q->front = 0;

    q->rear++;

    q->items[q->rear] = value;

  }

}

// Removing elements from queue

int dequeue(struct queue\* q) {

  int item;

  if (isEmpty(q)) {

    printf("Queue is empty");

    item = -1;

  } else {

    item = q->items[q->front];

    q->front++;

    if (q->front > q->rear) {

      q->front = q->rear = -1;

    }

  }

  return item;

}

// Print the queue

void print\_Queue(struct queue\* q) {

  int i = q->front;

  if (isEmpty(q)) {

    printf("Queue is empty");

  } else {

    printf("\nQueue contains ");

    for (i = q->front; i < q->rear + 1; i++) {

      printf("%d ", q->items[i]);

    }

  }

}

1. Case 5

struct node\_DFS{

  int vertex;

  struct node\_DFS\* next;

};

struct node\_DFS\* createNode\_DFS(int v);

struct Graph\_DFS{

  int numVertices;

  int\* visited;

  struct node\_DFS\*\* adjLists;

};

// DFS algo

void DFS(struct Graph\_DFS\* graph, int vertex) {

  struct node\_DFS\* adjList = graph->adjLists[vertex];

  struct node\_DFS\* temp = adjList;

  graph->visited[vertex] = 1;

  printf("Visited %d \n", vertex);

  while (temp != NULL) {

    int connectedVertex = temp->vertex;

    if (graph->visited[connectedVertex] == 0) {

      DFS(graph, connectedVertex);

    }

    temp = temp->next;

  }

}

// Create a node

struct node\_DFS\* createNode\_DFS(int v) {

  struct node\_DFS\* new\_Node = malloc(sizeof(struct node));

  new\_Node->vertex = v;

  new\_Node->next = NULL;

  return new\_Node;

}

// Create graph

struct Graph\_DFS\* createGraph\_DFS(int vertices) {

  struct Graph\_DFS\* graph = malloc(sizeof(struct Graph\_DFS));

  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices \* sizeof(struct node\_DFS\*));

  graph->visited = malloc(vertices \* sizeof(int));

  int i;

  for (i = 0; i < vertices; i++) {

    graph->adjLists[i] = NULL;

    graph->visited[i] = 0;

  }

  return graph;

}

// Add edge

void addEdge\_DFS(struct Graph\_DFS\* graph, int src, int dest) {

  // Add edge from src to dest

  struct node\_DFS\* new\_Node = createNode\_DFS(dest);

  new\_Node->next = graph->adjLists[src];

  graph->adjLists[src] = new\_Node;

  // Add edge from dest to src

  new\_Node = createNode\_DFS(src);

  new\_Node->next = graph->adjLists[dest];

  graph->adjLists[dest] = new\_Node;

}

1. Main()

int main() {

  int parameter = 1;

  int menu = 0;

  int adjMatrix[V][V]; //matrix untuk case 1

  struct Graph\* graph = create\_A\_Graph(V); //inisiasi case 2

  struct Graph\_BFS\* graph\_BFS = createGraph(V); //inisiasi graph untuk case 4

  struct Graph\_DFS\* graph\_DFS = createGraph\_DFS(V);

  while (parameter == 1){

  printf("GRAPH REPRESENTATION AND TRASVERSAL\n");

  printf("-------------------------------------\n\n\n");

  printf("        1   ---   2\n");

  printf("         \\       / \\   \n");

  printf("          4 --- 3 -- 0 \n\n");

  printf("1. Show Adjancency Matrix\n");

  printf("2. Show Adjacency List\n");

  printf("3. Show Degree of all vertices\n");

  printf("4. Show BFS Traversal from vertex 0\n");

  printf("5. show DFS Traversal from vertex 0\n");

  printf("6. Exit\n\n");

  printf(">> Input Choice: ");

  scanf("%d", & menu);

  switch(menu){

    case 1:

        init(adjMatrix);

            addEdge(adjMatrix, 0, 3);

            addEdge(adjMatrix, 1, 2);

            addEdge(adjMatrix, 1, 4);

            addEdge(adjMatrix, 2, 3);

            addEdge(adjMatrix, 2, 4);

            addEdge(adjMatrix, 3, 4);

            print\_AdjMatrix(adjMatrix);

            getchar();

            getchar();

            system("cls");

    break;

    case 2:

        //input edge di soal (konstan)

            add\_Edge(graph, 4, 2);

      add\_Edge(graph, 4, 3);

            add\_Edge(graph, 4, 1);

      add\_Edge(graph, 3, 2);

      add\_Edge(graph, 2, 1);

            add\_Edge(graph, 0, 3);

            print\_Graph(graph);

            getchar();

            getchar();

            system("cls");

    break;

    case 3:

      init(adjMatrix);

            addEdge(adjMatrix, 0, 3);

            addEdge(adjMatrix, 1, 2);

            addEdge(adjMatrix, 1, 4);

            addEdge(adjMatrix, 2, 3);

            addEdge(adjMatrix, 2, 4);

            addEdge(adjMatrix, 3, 4);

            count\_degree(adjMatrix);

            getchar();

            getchar();

            system("cls");

      break;

    case 4:

            addEdge\_BFS(graph\_BFS, 4, 2);

      addEdge\_BFS(graph\_BFS, 4, 3);

            addEdge\_BFS(graph\_BFS, 4, 1);

      addEdge\_BFS(graph\_BFS, 3, 2);

      addEdge\_BFS(graph\_BFS, 2, 1);

            addEdge\_BFS(graph\_BFS, 0, 3);

            printf("\n\n  BFS Traversal\n");

            printf("  --------------\n");

            bfs(graph\_BFS, 0);

            getchar();

            getchar();

            system("cls");

      break;

    case 5:

      addEdge\_DFS(graph\_DFS, 0, 3);

      addEdge\_DFS(graph\_DFS, 1, 2);

      addEdge\_DFS(graph\_DFS, 2, 3);

      addEdge\_DFS(graph\_DFS, 4, 2);

      addEdge\_DFS(graph\_DFS, 4, 3);

      addEdge\_DFS(graph\_DFS, 4, 1);

            printf("\nDFS Traversal\n");

            printf("--------------\n\n");

            DFS(graph\_DFS, 0);

            getchar();

            getchar();

            system("cls");

      break;

    case 6:

      printf("\n-----Programm Closed Thank you-----\n");

      parameter = 0;

      break;

    default:

      system("cls");

      break;

  }

}

}

## **Bukti Presentasi**