# JOBSHEET 13

## Graphs: Breadth First Search and Depth First Search

| Fakultas | Proram Studi | Kode MK | Waktu |
|---|---|---|---|
| Teknik | Teknik Informatika | INF1.62.2014 | 2 x 50 Menit |

### TUJUAN PRAKTIKUM

1. Mahasiswa memahami konsep pencarian DFS dan BFS dalam pemrograman C.
2. Mahasiswa memahami operasi yang ada dalam pencarian DFS dan BFS.
3. Mahasiswa mampu mengimplementasikan pencarian struktur data DFS dan BFS menggunakan pemrograman C dengan IDE.

### HARDWARE & SOFTWARE

1. Personal Computer
2. DevC++ IDE

### TEORI SINGKAT

1. Depth First Search/Traversal

Traversal artinya mengunjungi semua simpul/node dari sebuah graph. DFS atau DFT adalah sebuah algoritma pencarian rekursif untuk mencari semua titik yang ada pada sebuah graph atau pohon simpul. Pada materi kali ini akan dibahas contoh penggunaan algoritma DFS, DFS pseudocode dank ode yang digunakan dalam bahasa pemograman C.

Algoritma DFS

Implementasi sederhana dari sebuah DFS adalah membagi setiap titik pada sebuah graph atau tree menjadi 2 kategori :

1. Visited
2. Not Visited

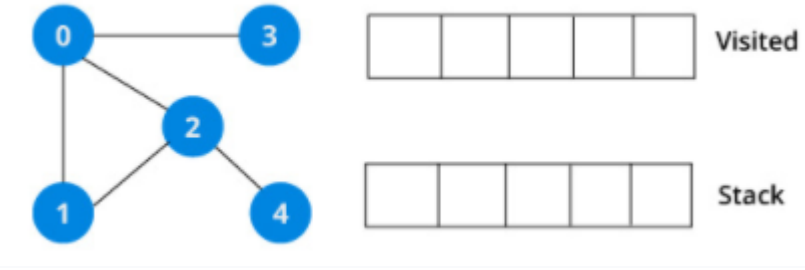
Tujuan dari algoritma ini adalah untuk menandai setiap titik/vertex yang telah dikunjungi sambil menghidari titik yang telah dikunjungi.

Cara kerja algoritma DFS :

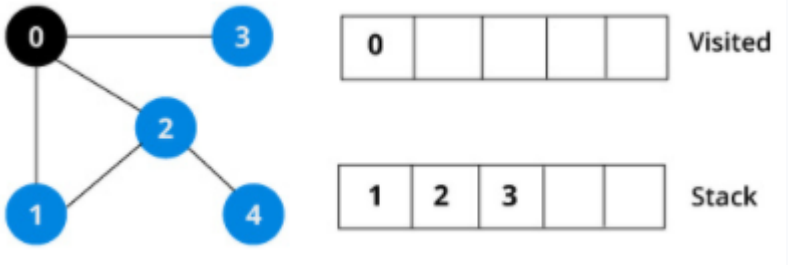
1. Dimulai dengan meletakkan salah satu dari titik graph di atas sebuah tumpukan.
2. Ambil item paling atas dari tumpukan dan tambahkan dalam daftar titik yang telah dikunjungi.
3. Buatlah sebuah daftar dari titik yang segaris lurus dengannya. Tambahkan satu yang mana belum dikunjungi pada tumpukan paling atas.
4. Ulangi langkah ke 2 dan 3 sampai dengan tumpukan menjadi kosong.

Contoh Implementasi DFS dalam sbuah Graph/Tree

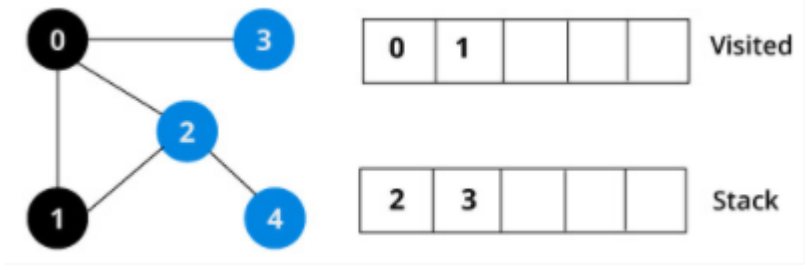
Sebagai contoh disini menggunakan graph tidak terarah dengan 5 simpul/vertices :



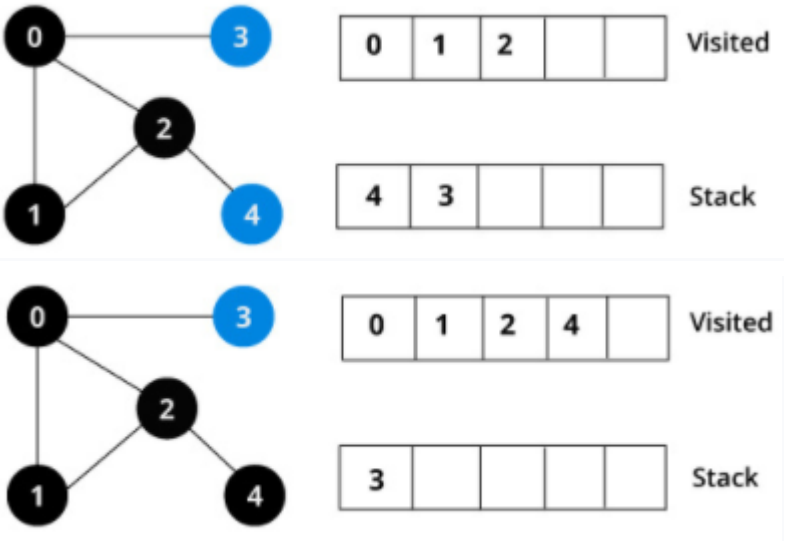

Sebagai awalan dimulai dari titik vertex 0, algoritma DFS dimulai dengan meletakkan titik pertama ke dalam daftar yang telah dikunjungi dan meletakkan semua titik yang berhubungan dengan titik 0 ke dalam sebuah stack/tumpukan.



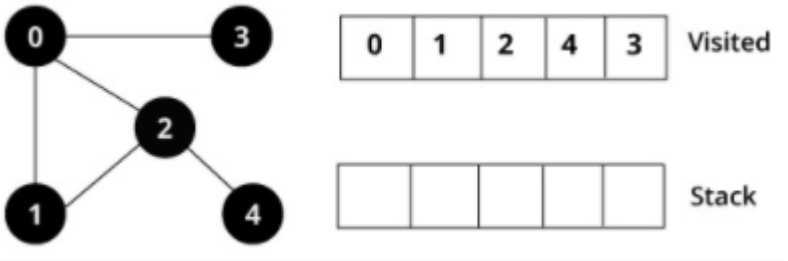

Selanjutnya, kunjungi elemen teratas yang ada pada stack/tumpukan disini contohnya 1 dan carilah titik-titik vertices /simpul yang terhubung dengannya. Karna 0 telah dikunjungi, kunjungi 2.




Titik 2 memiliki titik yang terhubung sejajar yakni titik 4, jadi tambahkan ke atas tumpukkan dan kunjungi titik tersebut.




Setelah mengunjungi elemen terakhir 3. Elemen tersebut tidak memiliki titik yang terhubung dengannya, jadi selesai sudah implementasi dari Depth First Traversal dari grap ini.




## PERCOBAAN

1. DFS 1

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

struct node
{
   int vertex;
   struct node* next;
};

struct node* createNode(int v);

struct Graph
{
   int numVertices;
   int* visited;
   struct node** adjLists; // we need int** to store a two dimensional array. Similary, we need
struct node** to store an array of Linked lists
};

struct Graph* createGraph(int);
void addEdge(struct Graph*, int, int);
```

```
void printGraph(struct Graph*);
void DFS(struct Graph*, int);


int main()
{

   struct Graph* graph = createGraph(4);
   addEdge(graph, 0, 1);
   addEdge(graph, 0, 2);
   addEdge(graph, 1, 2);
   addEdge(graph, 2, 3);

   printGraph(graph);

   DFS(graph, 2);

   return 0;
}

void DFS(struct Graph* graph, int vertex) {
      struct node* adjList = graph->adjLists[vertex];
      struct node* 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;
      }
}


struct node* createNode(int v)
{
   struct node* newNode =(node*) malloc(sizeof(struct node));
   newNode->vertex = v;
   newNode->next = NULL;
   return newNode;
}

struct Graph* createGraph(int vertices)
{
   struct Graph* 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;
}

void addEdge(struct Graph* graph, int src, int dest)
{
    // Add edge from src to dest
    struct node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Add edge from dest to src
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

void printGraph(struct Graph* graph)
{
    int v;
    for (v = 0; v < graph->numVertices; v++)
    {
        struct node* temp = graph->adjLists[v];
        printf("\n Adjacency list of vertex %d\n ", v);
        while (temp)
        {
            printf("%d -> ", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}
```

2. DFS 2

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

#define MAX 5

struct Vertex {
   char label;
   bool visited;
};

//stack variables

int stack[MAX];
int top = -1;
```

```
//graph variables

//array of vertices
struct Vertex* lstVertices[MAX];

//adjacency matrix
int adjMatrix[MAX][MAX];

//vertex count
int vertexCount = 0;

//stack functions

void push(int item) {
  stack[++top] = item;
}

int pop() {
  return stack[top--];
}

int peek() {
  return stack[top];
}

bool isStackEmpty() {
  return top == -1;
}

//graph functions

//add vertex to the vertex list
void addVertex(char label) {
  struct Vertex* vertex = (struct Vertex*) malloc(sizeof(struct Vertex));
  vertex->label = label;
  vertex->visited = false;
  lstVertices[vertexCount++] = vertex;
}

//add edge to edge array
void addEdge(int start,int end) {
  adjMatrix[start][end] = 1;
  adjMatrix[end][start] = 1;
}

//display the vertex
void displayVertex(int vertexIndex) {
  printf("%c ",lstVertices[vertexIndex]->label);
}

//get the adjacent unvisited vertex
int getAdjUnvisitedVertex(int vertexIndex) {
  int i;
```

```
  for(i = 0; i < vertexCount; i++) {
    if(adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == false) {
      return i;
    }
  }

  return -1;
}

void depthFirstSearch() {
  int i;

  //mark first node as visited
  lstVertices[0]->visited = true;

  //display the vertex
  displayVertex(0);

  //push vertex index in stack
  push(0);

  while(!isStackEmpty()) {
    //get the unvisited vertex of vertex which is at top of the stack
    int unvisitedVertex = getAdjUnvisitedVertex(peek());

    //no adjacent vertex found
    if(unvisitedVertex == -1) {
      pop();
    } else {
      lstVertices[unvisitedVertex]->visited = true;
      displayVertex(unvisitedVertex);
      push(unvisitedVertex);
    }
  }

  //stack is empty, search is complete, reset the visited flag
  for(i = 0;i < vertexCount;i++) {
    lstVertices[i]->visited = false;
  }
}

int main() {
  int i, j;

  for(i = 0; i < MAX; i++)    // set adjacency {
    for(j = 0; j < MAX; j++) // matrix to 0
      adjMatrix[i][j] = 0;


  addVertex('S');   // 0
  addVertex('A');   // 1
  addVertex('B');   // 2
  addVertex('C');   // 3
  addVertex('D');   // 4
```

```
  addEdge(0, 1);    // S - A
  addEdge(0, 2);    // S - B
  addEdge(0, 3);    // S - C
  addEdge(1, 4);    // A - D
  addEdge(2, 4);    // B - D
  addEdge(3, 4);    // C - D

  printf("Depth First Search: ");
  depthFirstSearch();

  return 0;
}
```

3. DFS 3

```
#include<stdio.h>

void DFS(int);
int G[10][10],visited[10],n;    //n is no of vertices and graph is sorted in array G[10][10]

int main()
{
   int i,j;
   printf("Enter number of vertices:");

     scanf("%d",&n);

   //read the adjecency matrix
     printf("\nEnter adjecency matrix of the graph:");

     for(i=0;i<n;i++)
      for(j=0;j<n;j++)
                              scanf("%d",&G[i][j]);

   //visited is initialized to zero
  for(i=0;i<n;i++)
      visited[i]=0;

   DFS(0);
}

void DFS(int i)
{
   int j;
     printf("\n%d",i);
   visited[i]=1;

     for(j=0;j<n;j++)
      if(!visited[j]&&G[i][j]==1)
          DFS(j);
}
```

4. DFS 4

```
#include<stdio.h>
#include<conio.h>
```

```
int a[20][20],reach[20],n;
void dfs(int v) {
        int i;
        reach[v]=1;
        for (i=1;i<=n;i++)
          if(a[v][i] && !reach[i]) {
                    printf("\n %d->%d",v,i);
                    dfs(i);
        }
}
 main()
 {
        int i,j,count=0;

        printf("\n Enter number of vertices:");
        scanf("%d",&n);
        for (i=1;i<=n;i++) {
                    reach[i]=0;
                    for (j=1;j<=n;j++)
                      a[i][j]=0;
        }
        printf("\n Enter the adjacency matrix:\n");
        for (i=1;i<=n;i++)
          for (j=1;j<=n;j++)
           scanf("%d",&a[i][j]);
        dfs(1);
        printf("\n");
        for (i=1;i<=n;i++) {
                    if(reach[i])
                      count++;
        }
        if(count==n)
          printf("\n Graph is connected"); else
          printf("\n Graph is not connected");
        getch();
}
```

## DAFTAR PUSTAKA

1. Kernighan, Brian W, & Ritchie, Dennis M. 1988. The Ansi C Programming Language Second Edition, Prentice-Hall.
2. Cipta Ramadhani. 2015. Dasar Algoritma & Struktur Data. Yogyakarta: ANDI.