SOAL PRAKTIKUM KOM120H STRUKTUR DATA
PERTEMUAN 9 APLIKASI *GRAPH TRAVERSAL*

Dengan representasi *graph* menggunakan *adjacency matrix* seperti pada pertemuan sebelumnya, pada pertemuan ini kita akan membuat implementasi dari deteksi *cycle* dan *topological sort*.

Potongan program di bawah ini mengimplementasikan algoritme **DFS_Visit** dengan menggunakan representasi *adjacency matrix* yang digunakan sebelumnya dan akan dimodifikasi agar dapat melakukan deteksi *cycle*.

```
void DFS_visit(Graph *g, COLOR *vertex_colors, int v)
{
    int i;
    // Tampilkan v
    printf("%d ", v);

    // Tandai v dengan warna GRAY
    vertex_colors[v] = GRAY;

    // Cari vertex yang adjacent terhadap v, jika masih WHITE, panggil DFS_visit vertex itu
    for (i = 0; i < g->n_vertices; i++)
    {
        if (/* Soal 1: Tuliskan kondisi dimana ada vertex i adjacent thd v dan berwarna BUKAN white */)
            printf("Cycle! \n");
        if (g->adjacency_matrix[v][i] == 1 && vertex_colors[i] == WHITE)
            DFS_visit(g, vertex_colors, i);
    }

    vertex_colors[v] = BLACK;
}
```

Soal 1. Lengkapi bagian di atas agar algoritme DFS dapat melakukan deteksi *cycle* pada *graph* masukan!

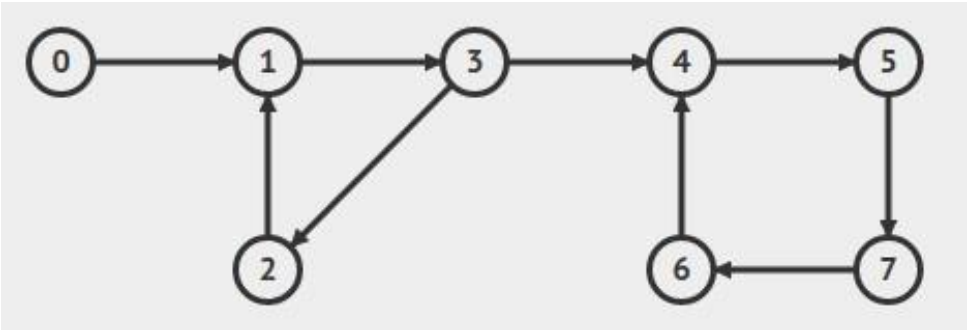
```
void DFS_visit(Graph *g, Color *vertex_colors, int v) {

    int i;

    printf("%d ", v);

    vertex_colors[v] = GRAY;

    for(i = 0; i < g->n_vertices; i++) {
        if(g->adjacency_matrix[v][i] == 1 && vertex_colors[i] != WHITE) {
            printf("Cycle!\n");
        }
        if(g->adjacency_matrix[v][i] == 1 && vertex_colors[i] == WHITE) {
            DFS_visit(g, vertex_colors, i);
        }
    }
    vertex_colors[v] = BLACK;
}
```

Soal 2. Apakah keluaran dari program yang telah dilengkapi di atas jika diberikan *graph* di bawah ini?

```
8 9
0 1
1 3
3 2
2 1
3 4
4 5
5 7
7 6
6 4
0 1 3 2 Cycle!
4 5 7 6 Cycle!
```

Soal 3. Lakukan modifikasi sedemikan rupa sehingga:

- Fungsi/prosedur **DFS_Visit()** tidak lagi mencetak keluaran berupa kata ‘Cycle’ ketika terdapat *cycle* pada *graph* masukan.
- Fungsi/prosedur utama **DFS**() akan mengembalikan keluaran berupa bilangan **0** atau **1**, yang menunjukkan adanya *cycle* atau tidak pada graph masukan **(0** = tidak ada *cycle*, **1** = ada *cycle*)




```
8 9
0 1
1 3
3 2
2 1
3 4
4 5
5 7
7 6
6 4
1
```

```
1 2
2 1
1 4
0
```

```c
#include <stdio.h>
#define MAXNUM_VERTICES 100

typedef enum {WHITE, GRAY , BLACK} Color;

typedef struct {
    int n_vertices;
    int n_edges;
    int adjacency_matrix[MAXNUM_VERTICES][MAXNUM_VERTICES];
} Graph;

int DFS_visit(Graph *g, Color *vertex_colors, int v) {
    int i;

    vertex_colors[v] = GRAY;

    for(i = 0; i < g->n_vertices; i++) {
        if(g->adjacency_matrix[v][i] == 1) {
            if(vertex_colors[i] == GRAY) {
                printf("1");
                return 1;
            }
            if(vertex_colors[i] == WHITE) {
                if(DFS_visit(g, vertex_colors, i)) {
                    return 1;
                }
            }
        }
    }
    vertex_colors[v] = BLACK;
    return 0;
}

int DFS(Graph *g) {
    Color vertex_colors[MAXNUM_VERTICES];
    int i;

    for(i = 0; i < g->n_vertices; i++) {
        vertex_colors[i] = WHITE;
    }
```

```
    for(i = 0; i < g->n_vertices; i++) {
        if(vertex_colors[i] == WHITE) {
            if(DFS_visit(g, vertex_colors, i))
                return 1;
        }
    }
    printf("0");
    return 0;
}

int main() {
    Graph g;
    int n_vertices, n_edges, i, j;
    int a, b;

    scanf("%d %d", &n_vertices, &n_edges);
    g.n_vertices = n_vertices;
    g.n_edges = n_edges;

    for(i = 0; i < n_vertices; i++) {
        for(j = 0; j < n_vertices; j++) {
            g.adjacency_matrix[i][j] = 0;
        }
    }

    for(i = 0; i < MAXNUM_VERTICES; i++) {
        for(j = 0; j < MAXNUM_VERTICES; j++) {
            g.adjacency_matrix[i][j] = -1;
        }
    }

    for(i = 0; i < n_edges; i++) {
        scanf("%d %d", &a, &b);
        g.adjacency_matrix[a][b] = 1;
    }

    DFS(&g);
    printf("\n");

    return 0;
}
```

Selanjutnya, kita akan melakukan modifikasi terhadap DFS untuk dapat melakukan *topological sort*. Pertama, kita tambahkan variabel pencatat waktu *finish* sebagai berikut:

```c
typedef enum {WHITE, GRAY, BLACK} COLOR;

#define Inf 1000000000       // Untuk menyatakan nilai Inf/tak-hingga
int finish_time[MAXNUM_VERTICES];
int time = 0;                // Waktu global

/*!
Prosedur yang mengimplementasikan DFS_visit.
*/
void DFS_visit(Graph *g, COLOR *vertex_colors, int v)
{
    int i;
    // Tampilkan v
    printf("%d ", v);
```

dan inisialisasi nilainya pada prosedur DFS().

```c
/*!
Prosedur yang mengimplementasikan DFS.
*/
void DFS(Graph *g)
{
    COLOR vertex_colors[MAXNUM_VERTICES];
    int i;
    for (i = 0; i < g->n_vertices; i++)
        vertex_colors[i] = WHITE;
    for (i = 0; i < g->n_vertices; i++)
        finish_time[i] = Inf;

    for (i = 0; i < g->n_vertices; i++)
    {
        if(vertex_colors[i] == WHITE)
            DFS_visit(g, vertex_colors, i);
    }

    printf("\n");
}
```

Selanjutnya, kita harus melakukan modifikasi terhadap fungsi DFS_Visit() agar mencatat waktu *finish* dengan benar.

```
/*!
Prosedur yang mengimplementasikan DFS_visit.
*/
void DFS_visit(Graph *g, COLOR *vertex_colors, int v)
{
    int i;
    // Tampilkan v
    printf("%d time = %d ", v, time);

    // Tandai v dengan warna GRAY
    vertex_colors[v] = GRAY;
    // Increment nilai waktu

    // Cari vertex yang adjacent terhadap v, jika masih WHITE, panggil DFS_visit vertex itu
    for (i = 0; i < g->n_vertices; i++)
    {
        if (g->adjacency_matrix[v][i] == 1 && vertex_colors[i] == WHITE)
            DFS_visit(g, vertex_colors, i);
    }

    vertex_colors[v] = BLACK;
    // Increment nilai waktu
    // Catat waktu finish untuk vertex v
}
```

Soal 4. Lengkapi potongan fungsi DFS_Visit() di atas agar dapat mencatat waktu *finish* dari setiap *vertex* dalam *graph* masukan.

```
void DFS_visit2(Graph *g, Color *vertex_colors, int v) {
    int i;

    printf("%d time = %d\n", v, time);

    vertex_colors[v] = GRAY;

    for(i = 0; i < g->n_vertices; i++) {
        if(g->adjacency_matrix[v][i] == 1 && vertex_colors[i] == WHITE) {
            DFS_visit2(g, vertex_colors, i);
        }
    }
    vertex_colors[v] = BLACK;
    finish_time[v] = ++time;
}
```

```
8 9
0 1
1 3
3 2
2 1
3 4
4 5
5 7
7 6
6 4
0 time = 0
1 time = 0
3 time = 0
2 time = 0
4 time = 1
5 time = 1
7 time = 1
6 time = 1
```

Soal 5. Lengkapi potongan kode pada fungsi main berikut untuk menampilkan waktu *finish* setiap *vertex* pada *graph* masukan.

```
DFS(&g);

for (i = 0; i < g.n_vertices; i++)
    // Cetak waktu finish vertex i
printf("\n");
```

```
DFS2(&g);
for(i = 0; i < g.n_vertices; i++) {
    printf("%d ", finish_time[i]);
}
printf("\n");
```

Soal 6. Gunakan fungsi **qsort** dari library **stdlib.h** untuk mengurutkan waktu *finish* dari setiap *vertex* dalam urutan **menurun**, dan kemudian gunakan ini untuk menghasilkan *topological sort* dari *graph* masukan yang diberikan (dengan asumsi tidak ada *cycle*).

```
#include <stdio.h>
#include <stdlib.h>
#define MAXNUM_VERTICES 100
#define Inf 1000000000
```

```c
typedef enum {WHITE, GRAY, BLACK} Color;

int finish_time[MAXNUM_VERTICES];
int time = 0;

typedef struct {
    int n_vertices;
    int n_edges;
    int adjacency_matrix[MAXNUM_VERTICES][MAXNUM_VERTICES];
} Graph;

int compare(const void *a, const void *b) {
    int va = *(const int*)a;
    int vb = *(const int*)b;
    return finish_time[vb] - finish_time[va];
}

int DFS_visit(Graph *g, Color *vertex_colors, int v) {
    int i;

    vertex_colors[v] = GRAY;

    for(i = 0; i < g->n_vertices; i++) {
        if(g->adjacency_matrix[v][i] == 1) {
            if(vertex_colors[i] == GRAY) {
                printf("1");
                return 1;
            }
            if(vertex_colors[i] == WHITE) {
                if(DFS_visit(g, vertex_colors, i)) {
                    return 1;
                }
            }
        }
    }
    vertex_colors[v] = BLACK;
    return 0;
}

void DFS_visit2(Graph *g, Color *vertex_colors, int v) {
    int i;

    vertex_colors[v] = GRAY;

    for(i = 0; i < g->n_vertices; i++) {
```

```
        if(g->adjacency_matrix[v][i] == 1 && vertex_colors[i] == WHITE) {
            DFS_visit2(g, vertex_colors, i);
        }
    }
    vertex_colors[v] = BLACK;
    finish_time[v] = ++time;
}

void DFS2(Graph *g) {
    Color vertex_colors[MAXNUM_VERTICES];
    int i;

    for(i = 0; i < g->n_vertices; i++) {
        vertex_colors[i] = WHITE;
    }
    for(i = 0; i < g->n_vertices; i++) {
        finish_time[i] = Inf;
    }

    time = 0;
    for(i = 0; i < g->n_vertices; i++) {
        if (vertex_colors[i] == WHITE){
            DFS_visit2(g, vertex_colors, i);
        }
    }
}

int main() {
    Graph g;
    int n_vertices, n_edges, i, j;
    int a, b;

    scanf("%d %d", &n_vertices, &n_edges);
    g.n_vertices = n_vertices;
    g.n_edges = n_edges;

    for(i = 0; i < n_vertices; i++) {
        for(j = 0; j < n_vertices; j++) {
            g.adjacency_matrix[i][j] = 0;
        }
    }

    for(i = 0; i < n_edges; i++) {
        scanf("%d %d", &a, &b);
        g.adjacency_matrix[a][b] = 1;
    }
```

```
    DFS2(&g);

    int vertices[MAXNUM_VERTICES];
    for(i = 0; i < g.n_vertices; i++) {
        vertices[i] = i;
    }

    qsort(vertices, g.n_vertices, sizeof(int), compare);

    for(i = 0; i < g.n_vertices; i++) {
        printf("%d ", vertices[i]);
    }
    printf("\n");

    return 0;
}
```

```
8 9
0 1
1 3
3 2
2 1
3 4
4 5
5 7
7 6
6 4
0 1 3 4 5 7 6 2
```

Soal 7. Buatlah sebuah program lengkap yang melakukan hal berikut: diberikan sebuah *graph* masukan **G**, keluaran yang dihasilkan adalah:

- Kata “Cycle” jika **G** mengandung *cycle*.
- *Topological sort* dari semua *vertex* di **G** jika **G** tidak mengandung *cycle*.

```
#include <stdio.h>
#include <stdlib.h>
#define MAXNUM_VERTICES 100
#define Inf 1000000000

typedef enum {WHITE, GRAY, BLACK} Color;

int finish_time[MAXNUM_VERTICES];
```

```c
int time_counter = 0;

typedef struct {
    int n_vertices;
    int n_edges;
    int adjacency_matrix[MAXNUM_VERTICES][MAXNUM_VERTICES];
} Graph;

int DFS_visit(Graph *g, Color *vertex_colors, int v) {
    int i;
    vertex_colors[v] = GRAY;
    for (i = 0; i < g->n_vertices; i++) {
        if (g->adjacency_matrix[v][i] == 1) {
            if (vertex_colors[i] == GRAY) {
                return 1;
            }
            if (vertex_colors[i] == WHITE) {
                if (DFS_visit(g, vertex_colors, i)) {
                    return 1;
                }
            }
        }
    }
    vertex_colors[v] = BLACK;
    return 0;
}

int detect_cycle(Graph *g) {
    int i;
    Color vertex_colors[MAXNUM_VERTICES];
    for (i = 0; i < g->n_vertices; i++) {
        vertex_colors[i] = WHITE;
    }
    for (i = 0; i < g->n_vertices; i++) {
        if (vertex_colors[i] == WHITE) {
            if (DFS_visit(g, vertex_colors, i)) {
                return 1;
            }
        }
    }
    return 0;
}

void DFS_visit2(Graph *g, Color *vertex_colors, int v) {
    int i;
    vertex_colors[v] = GRAY;
```

```
    for (i = 0; i < g->n_vertices; i++) {
        if (g->adjacency_matrix[v][i] == 1 && vertex_colors[i] == WHITE) {
            DFS_visit2(g, vertex_colors, i);
        }
    }
    vertex_colors[v] = BLACK;
    finish_time[v] = ++time_counter;
}

void DFS2(Graph *g) {
    int i;
    Color vertex_colors[MAXNUM_VERTICES];
    for (i = 0; i < g->n_vertices; i++) {
        vertex_colors[i] = WHITE;
        finish_time[i] = Inf;
    }
    time_counter = 0;
    for (i = 0; i < g->n_vertices; i++) {
        if (vertex_colors[i] == WHITE) {
            DFS_visit2(g, vertex_colors, i);
        }
    }
}

int compare(const void *a, const void *b) {
    int va = *(const int*)a;
    int vb = *(const int*)b;

    return finish_time[vb] - finish_time[va];
}

int main() {
    Graph g;
    int n_vertices, n_edges;
    int i, j;
    int a, b;

    scanf("%d %d", &n_vertices, &n_edges);
    g.n_vertices = n_vertices;
    g.n_edges = n_edges;

    for (i = 0; i < n_vertices; i++) {
        for (j = 0; j < n_vertices; j++) {
            g.adjacency_matrix[i][j] = 0;
        }
    }
```

```
    for (i = 0; i < n_edges; i++) {
        scanf("%d %d", &a, &b);
        g.adjacency_matrix[a][b] = 1;
    }

    if(detect_cycle(&g)) {
        printf("Cycle\n");
    } else {
        DFS2(&g);
        int vertices[MAXNUM_VERTICES];
        for(i = 0; i < g.n_vertices; i++) {
            vertices[i] = i;
        }
        qsort(vertices, g.n_vertices, sizeof(int), compare);
        for(i = 0; i < g.n_vertices; i++) {
            printf("%d ", vertices[i]);
        }
        printf("\n");
    }

    return 0;
}
```

```
8 9
0 1
1 3
3 2
2 1
3 4
4 5
5 7
7 6
6 4
Cycle
```