MINGGU 11
Advance Graphs

DESKRIPSI TEMA

Terdapat dua metode algoritma untuk mencari jarak terdekat pada *graphs*, yaitu **Minimum Spanning Tree** dan **Dijkstra**. **Dijkstra** adalah algoritma untuk menentukan jarak atau lintasan terpendek dari sebuah *graph*. Setiap tahap dari algoritma ini akan menentukan jarang yang paling minimum dari *source* atau asalnya dengan cara mengakumulasi total jarak yang telah ditempuh dari *source* saat tiba di suatu *node*.

Minimum Spanning Tree (MST) adalah *tree* dengan semua *edge*-nya terhubung namun tidak membentuk sebuah siklus (*cycle*). Jumlah *weight* dalam MST yang didapatkan adalah yang paling minimum dari seluruh *edge* yang sudah terhubung. Ada 2 algoritma yang terkenal untuk menemukan MST, yaitu **Algoritma Kruskal** dan **Algoritma Prim**.

Algoritma Kruskal membentuk *spanning tree* dengan menambahkan satu persatu *edge* menjadi *spanning tree*. Algoritma Kruskal mengikuti pendekatan *greedy*, yakni pada setiap iterasi, algoritma tersebut mencari *edge* dengan bobot paling kecil dan menambahkannya ke *growing spanning tree*. Langkat algoritma:

a. Sortir *edge graph* berdasarkan bobot mereka
b. Mulai tambahkan *edge* ke MST dari *edge* dengan bobot terkecil hingga *edge* dengan bobot terbesar
c. Hanya tambahkan *edge* yang tidak membentuk siklus, yaitu *edge* yang hanya terhubung ke komponen yang terputus

Algoritma Prim juga menggunakan pendekatan *greedy* untuk menemukan MST. Dalam Algoritma *Prim*, kita mengembangkan *spanning tree* dari posisi asal. Bagian yang berbeda dari algoritma Kruskal adalah ditambahkannya *edge* ke growing spanning tree, sedangkan pada Algoritma Prim, *vertex* yang ditambahkan ke *growing* spaning *tree*.

CAPAIAN PEMBELAJARAN MINGGUAN (SUB-CAPAIAN PEMBELAJARAN)

1. Mahasiswa mampu mengimplementasikan algoritma Dijkstra menggunakan Bahasa C
2. Mahasiswa mampu mengimplementasikan algoritma Kruskal pada MST menggunakan Bahasa C
3. Mahasiswa mampu mengimplementasikan algoritma Prim pada MST menggunakan Bahasa C

PENUNJANG PRAKTIKUM

1. CodeBlocks
2. Dev-C++ (alternatif)

LANGKAH-LANGKAH PRAKTIKUM

A. Algoritma Dijkstra
   - Buat sebuah *file* dengan nama W11_Dijsktra.c
   - Salinlah *code* berikut ke dalam *file* yang telah dibuat
   - Perhatikan *indentation* dan nomor baris, serta baca komentar yang ada untuk membantu menjelaskan

```c
#include <limits.h>
#include <stdio.h>

// Jumlah vertex
#define V 9

// Fungsi untuk menemukan vertex dengan nilai minimum dari
// himpunan vertex yang tidak termasuk dalam path tree
int minDistance(int dist[], int sptSet[])
{
    // Inisialisasi min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == 0 && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// Fungsi untuk menampilkan distance array yang telah dibuat
void printSolution(int dist[])
{
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}
```

```c
// Fungsi yang mengimplementasikan Algoritma Dijkstra
// Fungsi ini menggunakan representasi Adjacency Matrix
void dijkstra(int graph[V][V], int src)
{
    int dist[V]; // Output array.  dist[i] akan menampung jarak terdekan dist[i]
    // jarak dari src to i

    // akan bernilai true (1) jika vertex i merupakan jalan terpendek
    int sptSet[V];

    // Inisialisasi semua jarak sebagai nilai INFINITE
    // dan stpSet[] bernilai false (0) pada awalnya
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = 0;

    // Jarak dari source ke dirinya sendiri selalu bernilai 0
    dist[src] = 0;

    // Cari Shortest Path
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);

        // Tandai vertex yang dipilih sebagai sudah diproses/dikunjungi
        // Mark the picked vertex as processed
        sptSet[u] = 1;

        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    printSolution(dist);
}
```

```
int main()
{
    int graph[V][V] = { { 0, 4,  0, 0,  0,  0,  0, 8,  0 },
                        { 4, 0,  8, 0,  0,  0,  0, 11, 0 },
                        { 0, 8,  0, 7,  0,  4,  0, 0,  2 },
                        { 0, 0,  7, 0,  9,  14, 0, 0,  0 },
                        { 0, 0,  0, 9,  0,  10, 0, 0,  0 },
                        { 0, 0,  4, 14, 10, 0,  2, 0,  0 },
                        { 0, 0,  0, 0,  0,  2,  0, 1,  6 },
                        { 8, 11, 0, 0,  0,  0,  1, 0,  7 },
                        { 0, 0,  2, 0,  0,  0,  6, 7,  0 } };

    dijkstra(graph, 0);

    return 0;
}

```

B. Algoritma Kruskal

- Buat sebuah *file* dengan nama W11_MST_Kruskal.c
- Salinlah *code* berikut ke dalam *file* yang telah dibuat
- Perhatikan *indentation* dan nomor baris, serta baca komentar yang ada untuk membantu menjelaskan

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

// Struct untuk merepresentasikan sisi dengan bobot di dalam graph
struct Edge {
    int src, dest, weight;
};

// Struct untuk merepresentasikan graph yang terkoneksi
// Tidak berarah namun memiliki bobot
struct Graph {
    // V-> jumlah dari simpul, E-> jumlah dari sisi
    int V, E;

    // graph direpresentasikan sebagai array of edges
    // Selama graph tersebu tidak berarah,
    // sisi dari sumber ke tujuan juga merupakan
    // sisi dari tujuan ke sumber.
    // Keduanya terhitung sebagai 1 sisi di sini
    struct Edge* edge;
};

// Membuat graph dengan V simpul dan E sisi
struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = (struct Graph*)(malloc(sizeof(struct Graph)));
    graph->V = V;
    graph->E = E;

    graph->edge = (struct Edge*)malloc(sizeof( struct Edge) * E);

    return graph;
}

// Struct untuk merepresentasikan subset untuk union-find
struct subset {
    int parent;
    int rank;
};
```

```c
// Fungsi untuk menemukan set dari elemen i
int find(struct subset subsets[], int i)
{
    // find root and make root as parent of i
    // (path compression)
    if (subsets[i].parent != i)
        subsets[i].parent
            = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}

// Fungsi untuk menggabungkan dua set dari x dan y
void Union(struct subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    // Hubungkan tree dengan rank yang lebih kecil di bawah
    // tree dengan rank yang lebih besar
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;

    // Jika rank-nya sama, maka pilih salah satu sebagai root
    // dan rank-nya bertambah 1
    else
    {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

// Bandingkan dua sisi sesuai dengan bobot mereka
// Gunakan qsort() untuk melakukan sorting array of edges
int myComp(const void* a, const void* b)
{
    struct Edge* a1 = (struct Edge*)a;
    struct Edge* b1 = (struct Edge*)b;
    return a1->weight > b1->weight;
}
```

```c
// Fungsi utama untuk membangun MST menggunkana Algoritma Kruskal
void KruskalMST(struct Graph* graph){
    int V = graph->V;
    struct Edgeresult[V]; // Ini akan menyimpan hasil dari MST
    int e = 0; // Index variabel, digunakan untuk result[]
    int i = 0; // Index variabel, digunakan untuk edegs yang sudah diurutkan

    // Step 1: Urutkan semua sisi yang ada berdasarkan bobot mereka.
    // Jika tidak boleh mengubah graph yang ada, maka dapat dibuat salinan array of edges
    qsort(graph->edge, graph->E, sizeof(graph->edge[0]), myComp);

    // Alokasi memori untuk membuat V subset
    struct subset* subsets = (struct subset*)malloc(V * sizeof(struct subset));

    for (int v = 0; v < V; ++v) { // Buat V subset dengan single element
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    // Jumlah dari sisi yang akan diambil sama dengan V-1
    while (e < V - 1 && i < graph->E) {
        // Step 2: Pilih sisi terkecil dan tambahkan index
        // untuk iterasi selanjutnya
        struct Edge next_edge = graph->edge[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        // Jika menambahkan sisi ini tidak membuat siklus,
        // tambahkan  sisi ini ke dalam result dan tambahkan
        // index dari result untuk sisi selanjutnya
        if (x != y) {
            result[e++] = next_edge;
            Union(subsets, x, y);
        } // Else buang sisi selanjutnya
    }

    // print isi dari result[] untuk menampilkan MST yang telah dibuat
    printf("Following are the edges in the constructed MST\n");
    int minimumCost = 0;
    for (i = 0; i < e; ++i){
        printf("%d -- %d == %d\n", result[i].src,
            result[i].dest, result[i].weight);
        minimumCost += result[i].weight;
    }
    printf("Minimum Cost Spanning tree : %d",minimumCost);
    return;
}
```

```
int main(){
    /* Mari buat graph berbobot berikut
            10
        0-------1
        | \     |
       6|  5\   |15
        |     \ |
        2-------3
            4     */
    int V = 4; // Jumlah dari simpul pada graph
    int E = 5; // Jumlah dari sisi pada graph
    struct Graph* graph = createGraph(V, E);

    // tambahkan sisi 0-1
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = 10;

    // tambahkan sisi 0-2
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 6;

    // tambahkan sisi 0-3
    graph->edge[2].src = 0;
    graph->edge[2].dest = 3;
    graph->edge[2].weight = 5;

    // tambahkan sisi 1-3
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 15;

    // tambahkan sisi 2-3
    graph->edge[4].src = 2;
    graph->edge[4].dest = 3;
    graph->edge[4].weight = 4;

    KruskalMST(graph);

    return 0;
}
```

C. Algoritma Prim

- Buat sebuah *file* dengan nama W11_Dijsktra.c
- Salinlah *code* berikut ke dalam *file* yang telah dibuat
- Perhatikan *indentation* dan nomor baris, serta baca komentar yang ada untuk membantu menjelaskan

```c
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>

// Number of vertices in the graph
#define V 6

// A utility function to find the vertex with
// minimum key value, from the set of vertices
// not yet included in MST
int minKey(int key[], bool mstSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

// A utility function to print the
// constructed MST stored in parent[]
int printMST(int parent[], int graph[V][V])
{
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
}
```

```c
// Function to construct and print MST for
// a graph represented using adjacency
// matrix representation
void primMST(int graph[V][V])
{
    // Array to store constructed MST
    int parent[V];
    // Key values used to pick minimum weight edge in cut
    int key[V];
    // To represent set of vertices included in MST
    bool mstSet[V];

    // Initialize all keys as INFINITE
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    // Always include first 1st vertex in MST.
    // Make key 0 so that this vertex is picked as first vertex.
    key[0] = 0;
    parent[0] = -1; // First node is always root of MST

    // The MST will have V vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum key vertex from the
        // set of vertices not yet included in MST
        int u = minKey(key, mstSet);

        // Add the picked vertex to the MST Set
        mstSet[u] = true;

        // Update key value and parent index of
        // the adjacent vertices of the picked vertex.
        // Consider only those vertices which are not
        // yet included in MST
        for (int v = 0; v < V; v++)

            // graph[u][v] is non zero only for adjacent vertices of m
            // mstSet[v] is false for vertices not yet included in MST
            // Update the key only if graph[u][v] is smaller than key[v]
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    // print the constructed MST
    printMST(parent, graph);
}
```

```
79  // driver program to test above function
80  int main()
81  {
82      /* Let us create the following graph
83          2 3
84      (0)--(1)--(2)
85      | / \ |
86      6| 8/ \5 |7
87      | /     \ |
88      (3)-------(4)
89            9         */
90      int graph[V][V] = { { 0, 2, 0, 6, 0 },
91                          { 2, 0, 3, 8, 5 },
92                          { 0, 3, 0, 0, 7 },
93                          { 6, 8, 0, 0, 9 },
94                          { 0, 5, 7, 9, 0 } };
95
96      int graph2[V][V] = { { 0, 3, 0, 0, 6, 5 },
97                           { 3, 0, 1, 0, 0, 4 },
98                           { 0, 1, 0, 6, 0, 4 },
99                           { 0, 0, 6, 0, 8, 5 },
100                          { 6, 0, 0, 8, 0, 2 },
101                          { 5, 4, 4, 5, 2, 0 } };
102
103     // Print the solution
104     primMST(graph2);
105
106     return 0;
107 }
```

D. Tugas

a) Buatlah program yang dapat menentukan jarak minimum dari kota asal ke kota tujuan. Dengan rute 1 arah saja (*undirected graph*). Untuk menentukan jalur antara 1 kota ke kota lainnya, dilakukan dengan menggunakan inputan dengan format Source#Destination#Weight. Source dan *destination* berupa huruf a-z, dan *weight* berupa integer. *Input* akan berhenti ketika *anda* memasukkan salah satu kota yang anda masukkan berupa "-", contohnya - #-#0 atau -#a#0. (*Clue*: Gunakan Dijkstra Algorithm)
Simpan dengan nama **NIM_T1_W11.c**

Contoh input:

[source]#[destination]#[weight]
a#b#2
a#c#5
b#c#3
b#d#7
c#e#6
c#d#1
b#a#3
-#-#0

Source: a

Contoh Output:

Untuk mencapai kota a dari kota a membutuhkan jarak terpendek 0.
Untuk mencapai kota b dari kota a membutuhkan jarak terpendek 2.
Untuk mencapai kota c dari kota a membutuhkan jarak terpendek 5.
Untuk mencapai kota d dari kota a membutuhkan jarak terpendek 6.
Untuk mencapai kota e dari kota a membutuhkan jarak terpendek 11.

b) Buatlah sebuah program MST dengan menggunakan *adjacency matrix*. Program akan meminta *input* berupa *adjacency matrix*. Program memiliki 2 menu utama yaitu algoritma Kruskal dan algoritma Prim. *Output* program berupa *edges* yang dipilih berdasarkan setiap algoritma tersebut dan minimum *cost* dari total *edges* yang dipilih.
Simpan dengan nama **NIM_T2_W11.c**

REFERENSI