Nama : Muhammad Syarif Setiadi

NIM: 1197050091

Minimum Spanning Tree

1. Pengertian

Minimum Spanning Tree adalah himpunan bagian dari himpunan garis-garis suatu graf berbobot tak berarah yang menghubungkan semua titik tanpa membentuk siklus dan dengan total bobot minimum. Dengan kata lain, ini adalah pohon rentang yang total bobotnya minimum.

Diberikan grafik terhubung dan tidak diarahkan,*pohon rentang* dari grafik itu adalah subgraf yang merupakan pohon dan menghubungkan semua simpul bersama. Sebuah grafik dapat memiliki banyak pohon rentang yang berbeda. Sebuah *minimum spanning tree(MST)* atau berat minimum spanning tree untuk graf berbobot, terhubung dan diarahkan adalah pohon rentang dengan berat kurang dari atau sama dengan berat setiap pohon spanning lainnya. Bobot pohon bentang adalah jumlah bobot yang diberikan ke setiap tepi pohon bentang.

Pohon merentang minimum memiliki tepi (V - 1) di mana V adalah jumlah simpul dalam grafik yang diberikan.

Ada dua algoritma untuk mencari Minimum Spanning Tree yaitu Algoritma Kruskal dan Algoritma Prim.

Algoritma Kruskal adalah algoritma untuk mencari pohon merentang minimum secara langsung didasarkan pada algoritma MST (Minimum Spanning Tree) umum. Pada algoritma Kruskal sisi-sisi di dalam graf diurut terlebih dahulu berdasarkan bobotnya dari kecil ke besar. Sisi yang dimasukkan ke dalam himpunan T adalah sisi graf G sedemikian sehingga T adalah pohon. Pada keadaan awal, sisi-sisi sudah diurut berdasarkan bobot membentuk hutan (forest). Hutan tersebut dinamakan hutan merentang (spanning forest). Sisi dari graf G ditambahkan ke T jika tidak membentuk sirkuit di T.

Sedangkan Algoritme Prim adalah sebuah algoritme dalam teori graf untuk mencari pohon rentang minimum untuk sebuah graf berbobot yang saling terhubung. Ini berarti bahwa sebuah himpunan bagian dari edge yang membentuk suatu pohon yang mengandung node, di mana bobot keseluruhan dari semua edge dalam pohon diminimalisasikan. Bila graf tersebut tidak terhubung, maka graf itu hanya memiliki satu pohon rentang minimum untuk satu dari komponen yang terhubung. Algoritme ini ditemukan pada 1930 oleh matematikawan Vojtěch Jarník

Perbedaan prinsip antara algoritma Prim dan Kruskal adalah jika pada algoritma Prim sisi yang dimasukkan ke dalam T harus bersisian dengan sebuah simpul di T, maka pada algoritma Kruskal sisi yang dipilih tidak perlu bersisian dengan simpul di T asalkan penambahan sisi tersebut tidak membentuk sirkuit.

1. Langkah-Langkah mencari MST dengan Algoritma Kruskal

Berikut adalah langkah-langkah untuk mencari MST menggunakan algoritma Kruskal

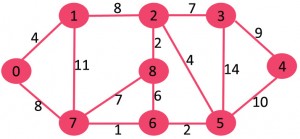
1. *Sortir semua tepi dalam urutan berat yang tidak menurun.*
2. *Pilih tepi terkecil. Periksa apakah itu membentuk siklus dengan pohon perentang yang terbentuk sejauh ini. Jika siklus tidak terbentuk, masukkan tepi ini. Lain, buang.*
3. *Ulangi langkah # 2 hingga ada tepi (V-1) di pohon bentang*

Langkah #2 menggunakan [algoritma Union-Find](https://www.geeksforgeeks.org/union-find/) untuk mendeteksi siklus.

algoritma Union-Find adalah Sebuah [*struktur data disjoint-set*](http://en.wikipedia.org/wiki/Disjoint-set_data_structure)(union-find) dimana struktur data yang melacak suatu set elemen dipartisi menjadi beberapa disjoint (non-overlapping) subset. Sebuah *algoritma union-find* adalah sebuah algoritma yang melakukan dua operasi berguna pada struktur data seperti:  
***Cari:*** . Menentukan subset elemen tertentu dalam ini dapat digunakan untuk menentukan apakah dua elemen berada di bagian yang sama.  
***Union:*** Menggabungkan dua subset menjadi satu subset.  
Pada postingan kali ini, kita akan membahas tentang aplikasi Disjoint Set Data Structure. Aplikasi ini untuk memeriksa apakah grafik yang diberikan berisi siklus atau tidak.

1. Contoh Langkah-Langkah mencari MST dengan Algoritma Kruskal

Perhatikan grafik input di bawah ini.



Grafik tersebut berisi 9 simpul dan 14 sisi. Jadi, pohon merentang minimum yang terbentuk akan memiliki (9 - 1) = 8 tepi.

Setelah menyortir:

Weight Src Dest

1 7 6

2 8 2

2 6 5

4 0 1

4 2 5

6 8 6

7 2 3

7 7 8

8 0 7

8 1 2

9 3 4

10 5 4

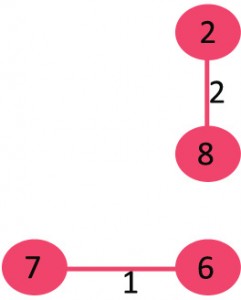
11 1 7

14 3 5

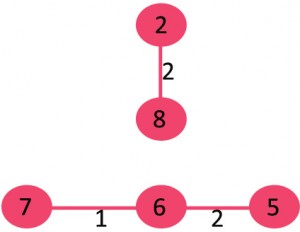
Sekarang pilih semua tepi satu per satu dari daftar tepi yang diurutkan   
**1.***Pilih tepi 7-6:* Tidak ada siklus yang terbentuk, sertakan. 



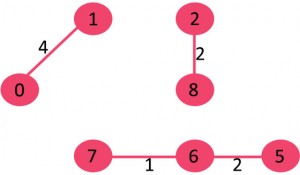
**2.***Pilih tepi 8-2:* Tidak ada siklus yang terbentuk, sertakan. 



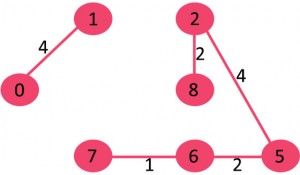
**3.***Pilih tepi 6-5:* Tidak ada siklus yang terbentuk, sertakan. 



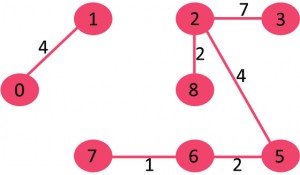
**4.***Pilih tepi 0-1:* Tidak ada siklus yang terbentuk, sertakan. 



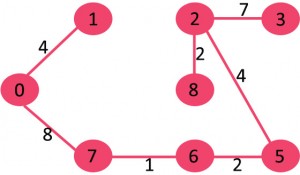
**5.***Pilih tepi 2-5:* Tidak ada siklus yang terbentuk, sertakan. 



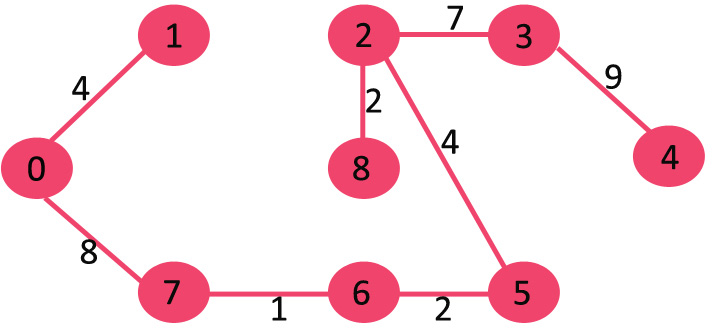
**6.***Pilih tepi 8-6:* Karena memasukkan tepi ini menghasilkan siklus, buang.  
**7.***Pilih tepi 2-3:* Tidak ada siklus yang terbentuk, sertakan. 



**8.***Pilih tepi 7-8:* Karena memasukkan tepi ini menghasilkan siklus, buang.  
**9.***Pilih tepi 0-7:* Tidak ada siklus yang terbentuk, sertakan. 



**10.***Pilih tepi 1-2:* Karena memasukkan tepi ini menghasilkan siklus, buang.  
**11.***Pilih tepi 3-4:* Tidak ada siklus yang terbentuk, sertakan. 



Karena jumlah tepi yang dimasukkan sama dengan (V - 1), algoritme berhenti di sini.

1. Contoh program Algoritma Kruskal

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class Graph {

// A class to represent a graph edge

class Edge implements Comparable<Edge>

{

int src, dest, weight;

// Comparator function used for

// sorting edgesbased on their weight

public int compareTo(Edge compareEdge)

{

return this.weight - compareEdge.weight;

}

};

// A class to represent a subset for

// union-find

class subset

{

int parent, rank;

};

int V, E; // V-> no. of vertices & E->no.of edges

Edge edge[]; // collection of all edges

// Creates a graph with V vertices and E edges

Graph(int v, int e)

{

V = v;

E = e;

edge = new Edge[E];

for (int i = 0; i < e; ++i)

edge[i] = new Edge();

}

// A utility function to find set of an

// element i (uses path compression technique)

int find(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;

}

// A function that does union of two sets

// of x and y (uses union by rank)

void Union(subset subsets[], int x, int y)

{

int xroot = find(subsets, x);

int yroot = find(subsets, y);

// Attach smaller rank tree under root

// of high rank tree (Union by Rank)

if (subsets[xroot].rank

< subsets[yroot].rank)

subsets[xroot].parent = yroot;

else if (subsets[xroot].rank

> subsets[yroot].rank)

subsets[yroot].parent = xroot;

// If ranks are same, then make one as

// root and increment its rank by one

else {

subsets[yroot].parent = xroot;

subsets[xroot].rank++;

}

}

// The main function to construct MST using Kruskal's

// algorithm

void KruskalMST()

{

// Tnis will store the resultant MST

Edge result[] = new Edge[V];

// An index variable, used for result[]

int e = 0;

// An index variable, used for sorted edges

int i = 0;

for (i = 0; i < V; ++i)

result[i] = new Edge();

// Step 1: Sort all the edges in non-decreasing

// order of their weight. If we are not allowed to

// change the given graph, we can create a copy of

// array of edges

Arrays.sort(edge);

// Allocate memory for creating V ssubsets

subset subsets[] = new subset[V];

for (i = 0; i < V; ++i)

subsets[i] = new subset();

// Create V subsets with single elements

for (int v = 0; v < V; ++v)

{

subsets[v].parent = v;

subsets[v].rank = 0;

}

i = 0; // Index used to pick next edge

// Number of edges to be taken is equal to V-1

while (e < V - 1)

{

// Step 2: Pick the smallest edge. And increment

// the index for next iteration

Edge next\_edge = new Edge();

next\_edge = edge[i++];

int x = find(subsets, next\_edge.src);

int y = find(subsets, next\_edge.dest);

// If including this edge does't cause cycle,

// include it in result and increment the index

// of result for next edge

if (x != y) {

result[e++] = next\_edge;

Union(subsets, x, y);

}

// Else discard the next\_edge

}

// print the contents of result[] to display

// the built MST

System.out.println("Following are the edges in "

+ "the constructed MST");

int minimumCost = 0;

for (i = 0; i < e; ++i)

{

System.out.println(result[i].src + " -- "

+ result[i].dest

+ " == " + result[i].weight);

minimumCost += result[i].weight;

}

System.out.println("Minimum Cost Spanning Tree "

+ minimumCost);

}

// Driver Code

public static void main(String[] args)

{

/\* Let us create following weighted graph

10

0--------1

| \ |

6| 5\ |15

| \ |

2--------3

4 \*/

int V = 4; // Number of vertices in graph

int E = 5; // Number of edges in graph

Graph graph = new Graph(V, E);

// add edge 0-1

graph.edge[0].src = 0;

graph.edge[0].dest = 1;

graph.edge[0].weight = 10;

// add edge 0-2

graph.edge[1].src = 0;

graph.edge[1].dest = 2;

graph.edge[1].weight = 6;

// add edge 0-3

graph.edge[2].src = 0;

graph.edge[2].dest = 3;

graph.edge[2].weight = 5;

// add edge 1-3

graph.edge[3].src = 1;

graph.edge[3].dest = 3;

graph.edge[3].weight = 15;

// add edge 2-3

graph.edge[4].src = 2;

graph.edge[4].dest = 3;

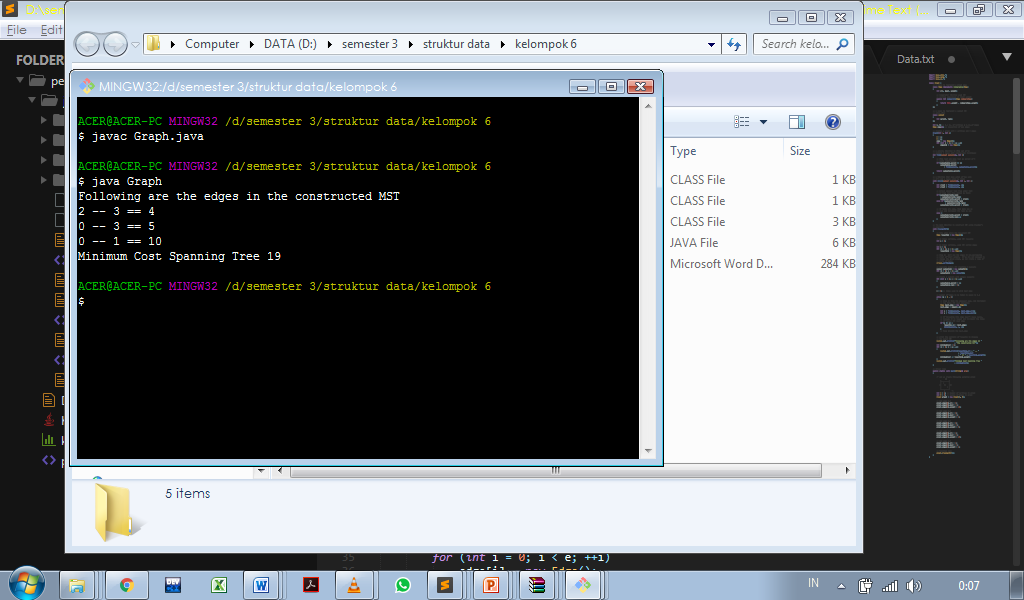
graph.edge[4].weight = 4;

// Function call

graph.KruskalMST();

}

}



1. Langkah-Langkah mencari MST dengan Algoritma Prim
2. Buat set mstSet yang melacak simpul yang sudah termasuk dalam MST.
3. Tetapkan nilai kunci ke semua simpul di grafik masukan. Inisialisasi semua nilai kunci sebagai INFINITE. Tetapkan nilai kunci sebagai 0 untuk simpul pertama sehingga itu diambil terlebih dahulu.
4. Sementara mstSet tidak menyertakan semua simpul
   1. Pilih simpul u yang tidak ada di mstSet dan memiliki nilai kunci minimum.
   2. Sertakan u ke mstSet.
   3. Perbarui nilai kunci dari semua simpul yang berdekatan dari u .

Untuk memperbarui nilai kunci, lakukan iterasi melalui semua simpul yang berdekatan. Untuk setiap simpul v yang berdekatan , jika bobot tepi uv lebih kecil dari nilai kunci sebelumnya v , perbarui nilai kunci sebagai bobot uv  
Ide menggunakan nilai kunci adalah untuk memilih tepi bobot minimum dari pemotongan . Nilai kunci hanya digunakan untuk simpul yang belum termasuk dalam MST, nilai kunci untuk simpul ini menunjukkan tepi bobot minimum yang menghubungkannya ke himpunan simpul yang termasuk dalam MST.

1. Contoh Langkah-Langkah mencari MST dengan Algoritma Prim

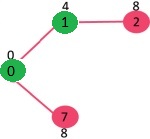
Mari kita pahami dengan contoh berikut: 

[](https://www.geeksforgeeks.org/wp-content/uploads/Fig-11.jpg)

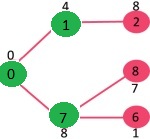
1. Set *mstSet* awalnya kosong dan kunci yang ditetapkan ke simpul adalah {0, INF, INF, INF, INF, INF, INF, INF} di mana INF menunjukkan tak terhingga. Sekarang pilih puncak dengan nilai kunci minimum. Titik 0 dipilih, masukkan ke dalam *mstSet* . Jadi *mstSet* menjadi {0}. Setelah menyertakan ke *mstSet* , perbarui nilai kunci dari simpul yang berdekatan. Simpul yang berdekatan dari 0 adalah 1 dan 7. Nilai kunci dari 1 dan 7 diperbarui sebagai 4 dan 8. Subgraf berikut menunjukkan simpul dan nilai kuncinya, hanya simpul dengan nilai kunci hingga yang ditampilkan. Simpul yang termasuk dalam MST ditampilkan dalam warna hijau.

[](https://www.geeksforgeeks.org/wp-content/uploads/MST1.jpg)

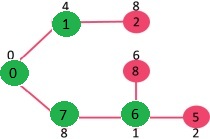
1. Pilih simpul dengan nilai kunci minimum dan belum termasuk dalam MST (bukan dalam mstSET). Titik 1 diambil dan ditambahkan ke mstSet. Jadi mstSet sekarang menjadi {0, 1}. Perbarui nilai kunci dari simpul yang berdekatan dari 1. Nilai kunci dari simpul 2 menjadi 8.

[](https://www.geeksforgeeks.org/wp-content/uploads/MST2.jpg)

1. Pilih simpul dengan nilai kunci minimum dan belum termasuk dalam MST (bukan dalam mstSET). Kita bisa memilih simpul 7 atau simpul 2, biarkan simpul 7 dipilih. Jadi mstSet sekarang menjadi {0, 1, 7}. Perbarui nilai kunci dari simpul yang berdekatan dari 7. Nilai kunci dari simpul 6 dan 8 menjadi terbatas (masing-masing 1 dan 7).

[](https://www.geeksforgeeks.org/wp-content/uploads/MST3.jpg)

1. Pilih simpul dengan nilai kunci minimum dan belum termasuk dalam MST (bukan dalam mstSET). Vertex 6 diambil. Jadi mstSet sekarang menjadi {0, 1, 7, 6}. Perbarui nilai kunci dari simpul yang berdekatan dari 6. Nilai kunci dari simpul 5 dan 8 diperbarui.

[](https://www.geeksforgeeks.org/wp-content/uploads/MST4.jpg)

1. Kami ulangi langkah di atas sampai *mstSet* menyertakan semua simpul dari grafik yang diberikan. Akhirnya, kami mendapatkan grafik berikut.

[](https://www.geeksforgeeks.org/wp-content/uploads/MST5.jpg)

1. Contoh program menggunakan Algoritma Prim

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class MST {

// Number of vertices in the graph

private static final int V = 5;

// 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[], Boolean mstSet[])

{

// Initialize min value

int min = Integer.MAX\_VALUE, min\_index = -1;

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[]

void printMST(int parent[], int graph[][])

{

System.out.println("Edge \tWeight");

for (int i = 1; i < V; i++)

System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);

}

// Function to construct and print MST for a graph represented

// using adjacency matrix representation

void primMST(int graph[][])

{

// Array to store constructed MST

int parent[] = new int[V];

// Key values used to pick minimum weight edge in cut

int key[] = new int[V];

// To represent set of vertices included in MST

Boolean mstSet[] = new Boolean[V];

// Initialize all keys as INFINITE

for (int i = 0; i < V; i++) {

key[i] = Integer.MAX\_VALUE;

mstSet[i] = false;

}

// Always include first 1st vertex in MST.

key[0] = 0; // Make key 0 so that this vertex is

// picked as first vertex

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 thd 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] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

// print the constructed MST

printMST(parent, graph);

}

public static void main(String[] args)

{

/\* Let us create the following graph

2 3

(0)--(1)--(2)

| / \ |

6| 8/ \5 |7

| / \ |

(3)-------(4)

9 \*/

MST t = new MST();

int graph[][] = new int[][] { { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

{ 0, 5, 7, 9, 0 } };

// Print the solution

t.primMST(graph);

}

}

