LAPORAN PRAKTIKUM

ANALISIS ALGORITMA

Alfari Sidnan Ghilmana

140810180011

Kelas A

Program Studi S-1 Teknik Informatika

Departemen Ilmu Komputer

Fakultas Matematika dan Ilmu Pengetahuan Alam

Universitas Padjadjaran

Pendahuluan

Minimum Spanning Tree

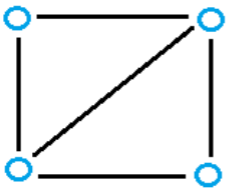
Minimum Spanning Tree (MST) adalah subset dari tepi-tepi dari graf tak-berbobot tepi-terhubung yang terhubung, yang menghubungkan semua simpul secara bersamaan, tanpa siklus apa pun dan dengan total bobot *edge* seminimum mungkin. Sebuah *tree* memiliki satu jalur yang menghubungkan dua simpul. *Spanning tree* dari graf adalah sebuah *tree* yang:

- Berisi semua simpul graf asal
- *Edge* yang dibentuk mencapai semua simpul
- *Acyclic*, artinya graf tidak memiliki node yang kembali ke dirinya sendiri

(a) Cyclic (b) Acyclic (c) Connected (d) Not Connected

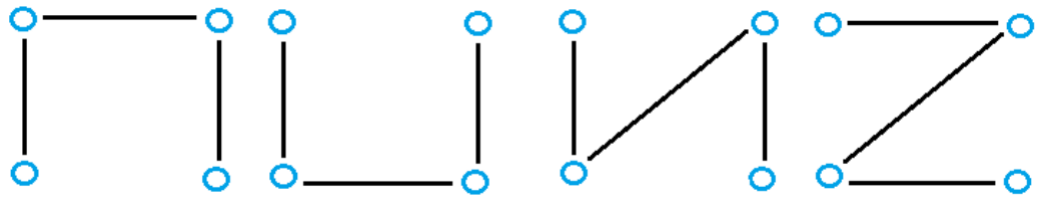
Gambar 1. Ilustrasi Jenis Graf

Misalkan ada sebuah graf sebagai berikut.



Gambar 2. Ilustrasi Graf

Graf di atas memiliki banyak kemungkinan *spanning tree* seperti terlihat di bawah ini.



(1) (2) (3) (4)

Gambar 3. Ilustrasi Kemungkinan *Spanning Tree*

Untuk mendapatkan berbagai kemungkinan terbentuknya Spanning Tree dari sebuah graf dapat menggunakan algoritma Kruskal dan Prim. Kedua algoritma tersebut masuk ke dalam kategori paradigma Greedy dimana solusi yang dipilih saat itu merupakan solusi terbaik tapi belum tentu solusi yang paling optimal. Prinsip algoritma Greedy sendiri adalah “take what you can get now!”.

Algoritma greedy membentuk solusi langkah per langkah (step by step). Terdapat banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya.

Persoalan optimasi (optimization problems): persoalan yang menuntut pencarian solusi optimum. Persoalan optimasi ada dua macam: Maksimasi (maximization) dan Minimasi (minimization) Solusi optimum (terbaik) adalah solusi yang bernilai minimum atau maksimum dari sekumpulan alternatif solusi yang mungkin. Elemen persoalan optimasi: kendala (constraints) danfungsi objektif(atau fungsi optimasi)

Untuk memahami paradigma Greedy pada Algoritma Kruskal dan Prim untuk permasalahan MST dibahas sebagai berikut.

Algoritma Kruskal

Algoritma Kruskal adalah algoritma *minimum spanning tree* yang tujuannya menemukan *edge* dengan bobot sekecil mungkin yang menghubungkan 2(dua) *tree* di *forest.* Berikut cara kerja dari algoritma kruskal.

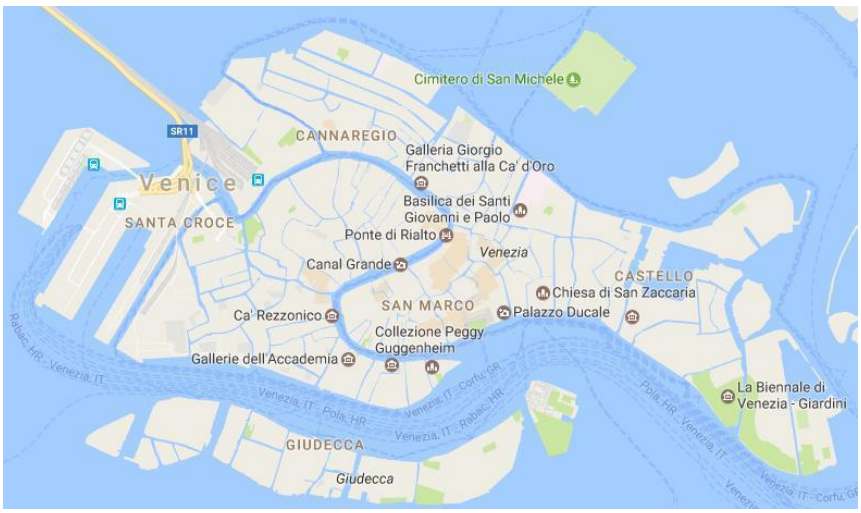
1. Mengurutkan *edges* dari graf menurut bobotnya.
2. T masih kosong. (: *tree*)

3. Pilih sisi *e* (*edge*) dengan bobot minimum yang tidak membentuk sirkuit di T. Masukkan *e* ke dalam T.

4. Ulangi langkah 2 sebanyak n-1 kali.

Contoh Studi Kasus

Terdapat peta dari kota Venice seperti terlihat pada gambar di bawah.

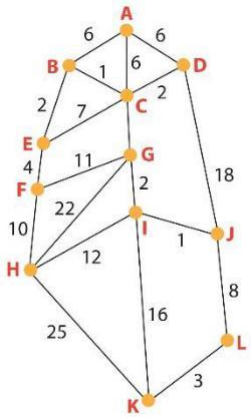


Gambar 4. Peta Kota Venice

Peta tersebut disederhanakan dengan mengubahnya menjadi graf dengan cara memberi nama lokasi penting di peta dengan huruf dan jarak dalam meter.

Cannaregio	Ponte Scalzi	Santa Corce	Dell Orto	Ferroyia	Piazzale Roma	San Polo	Dorso Duro	San Marco	St. Mark Basilica	Castello	Arsenale
A	B	C	D	E	F	G	H	I	J	K	L

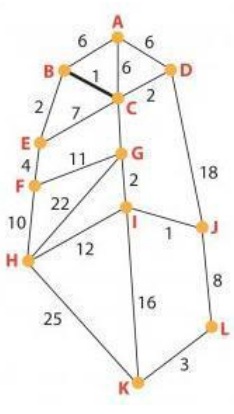
Langkah 1 – Menghapus semua *loop* dan *parallel edges.*

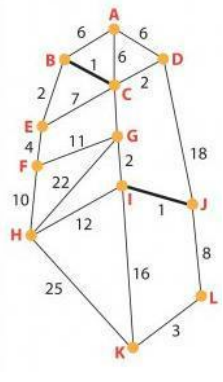


Langkah 2 – Mengatur semua *edge* pada graf dari yang terkecil ke terbesar.

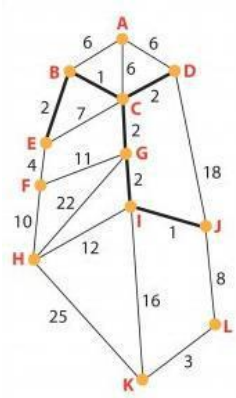
B,C	1
I,J	1
B,E	2
C,G	2
G,I	2
C,D	2
K,L	3
E,F	4
A ,B	6
A,C	6
A,D	6
E,C	7
J,L	8
F,H	10
F,G	11
H,I	12
I,K	16
D,J	18
G,H	22
H,K	25

Langkah 3 – Menambahkan *edge* dengan bobot paling kecil. B dan C terhubung terlebih dahulu karena *edge cost* nya hanya 1. Setelah itu menghubungkan I dan J karena memiliki *edge cost* 1.

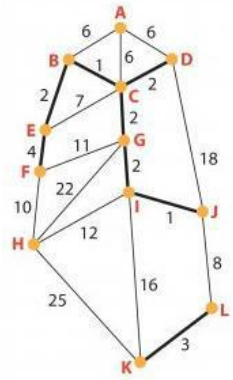




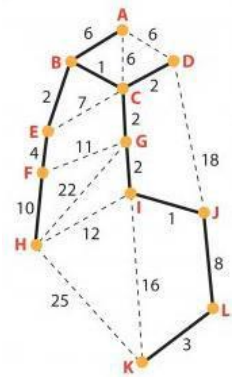
Langkah 4 – Menghubungkan *edges* dengan *cost* 2.



Langkah 5 – Menghubungkan *edges* dengan *cost* 3.



Langkah 6 – *Minimum spanning tree* terbentuk.



Implementasi Algoritma Kruskal dalam Program

Berikut implementasi algoritma Kruskal dalam notasi *pseudocode*.

```
procedure Kruskal(input G : graf, output T : pohon)

{ Membentuk minimum spanning tree T dari graf terhubung G.
  Masukan : graf-berbobot terhubung G = (V, E), yang mana |v| =
  n

  Keluaran : minimum spanning tree T = (V, E')

}

Deklarasi

    i, p, q, u, v : integer
```

Algoritma

{ Asumsi : sisi -sisi dari graf sudah diurut menaik berdasarkan bobotnya } T ← {}

while jumlah sisi T < n - 1 do

e ← sisi yang mempunyai bobot terkecil di dalam E dan bersisian dengan simpul di T

E ← E – (e) *{ e sudah dipilih, jadi hapus e dari E }*

If e tidak membentuk siklus di T then

T ← T ∪ (e) *{ masukkan e ke dalam T yang sudah terbentuk }* endif

endwhile

Algoritma Prim

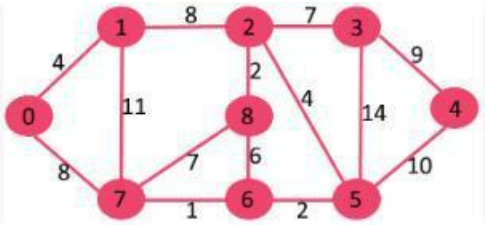
Algoritma Prim merupakan algoritma *minimum spanning tree* yang beroperasi dengan membangun *tree* dengan satu simpul pada satu waktu, dari titik awal yang berubah – ubah, pada setiap langkah menambahkan *cost* yang sekecil mungkin dari *tree* ke titik lain. Berikut proses algoritma prim.

1. Menginisialisasi *tree* dengan *simpul* tunggal, dipilih secara acak dari graf.
2. Membuat *tree* dengan satu *edge*: dari *edge* yang menghubungkan *tree* ke simpul yang belum ada di *tree*, temukan *edge* dengan bobot minimum, dan pindahkan *ke tree*.

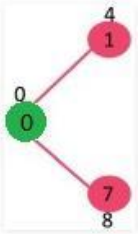
3. Mengulangi langkah 2(dua) sampai semua simpul terdapat di *tree*.

Contoh Studi Kasus

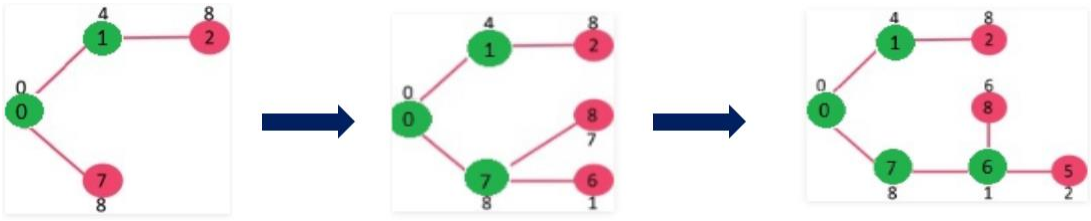
Terdapat graf seperti di bawah ini.



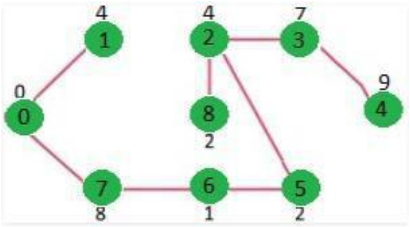
Langkah 1 – Menentukan titik awal dan membuat *subgraph* (simpul yang terdapat di *minimum spanning tree* ditampilkan dalam warna hijau).



Langkah 2 – Menentukan simpul dengan minimum *key value* dan belum termasuk ke *minimum spanning tree*.



Langkah 3 – Mengulangi langkah 2 sampai dengan *minimum spanning tree* mencakup semua simpul yang ada pada graf awal. Sehingga, mendapatkan *minimum spanning tree* sebagai berikut.



Implementasi Algoritma Prim dalam Program

Berikut implementasi algoritma Prim dalam notasi *pseudocode*.

```
procedure Prim(input G : graf, output T : pohon)
{ Membentuk minimum spanning tree T dari graf terhubung G.
  Masukan : graf-berbobot terhubung G = (V, E), yang mana |v| = n
  Keluaran : minimum spanning tree T = (V, E')
}

Deklarasi

    e : edge
```

Algoritma

T sisi e yang mempunyai bobot minimum di dalam E

E ← E – (e) *{ e sudah dipilih, jadi hapus e dari E }*

for I ← 1 to n-2 do

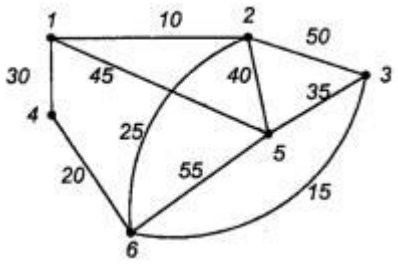
e ← sisi yang mempunyai bobot terkecil di dalam E dan bersisian dengan simpul di T

$T \leftarrow T \cup \{e\}$ *{ masukkan e ke dalam T yang sudah terbentuk }*

E ← E – (e) *{ e sudah dipilih, jadi hapus e dari E }*

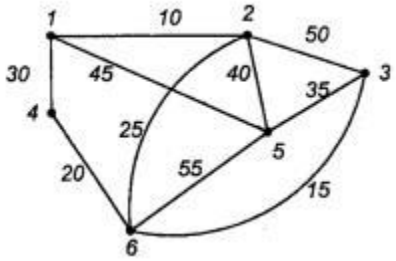
endfor

1. Cari *minimum spanning tree* pada graf di bawah dengan Algoritma Kruskal. Jelaskan langkah demi langkah sampai graf membentuk *minimum spanning tree*.



Jawaban :

Langkah 1 – Menghapus semua *loop* dan *parallel edges.*

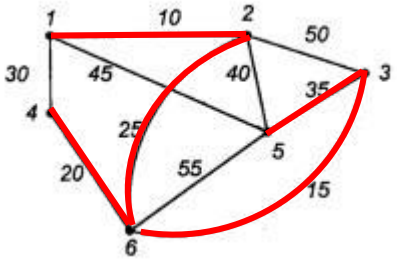


Langkah 2 – Mengatur semua *edge* pada graf dari yang terkecil ke terbesar.

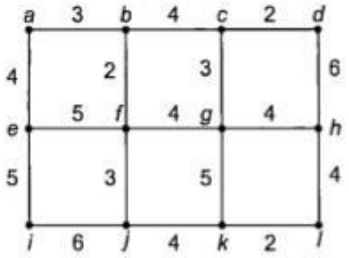
1,2	10
3,6	15
4,6	20
2,6	25

1,4	30
3,5	35
2,5	40
1,5	45
2,3	50
5,6	55

Langkah 3 – Menambahkan *edge* dengan bobot paling kecil. , lakukan dan jangan sampai membentuk sirkuit

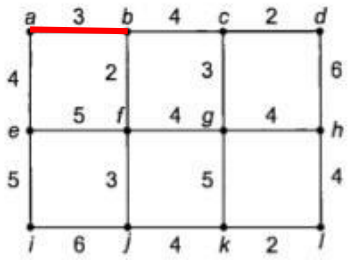


2 Minimum spanning tree terbentukGambarkan 3 buah *minimum spanning tree* yang berbeda beserta bobotnya untuk graf di bawah dengan Algoritma Prim. Jelaskan setiap langkah untuk membangun *minimum spanning tree*.

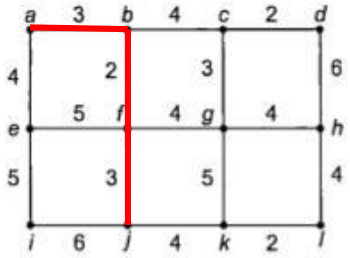


Jawaban :

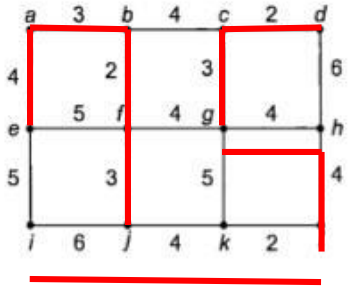
1. Menentukan titik awal graph

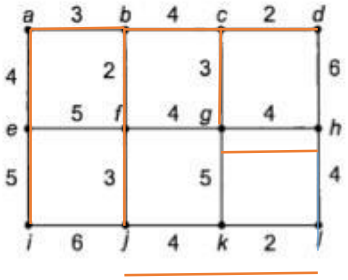


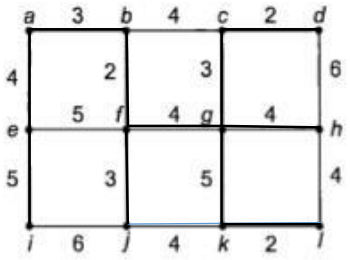
2. Menetukan simpul dengan minimum key value



3. Ulangi langkah kedua sampai semua terdapat tree







3. Apakah semua *minimum spanning tree T* dari graf terhubung *G* harus mengandung jumlah sisi yang sama? Jelaskan alasannya (bukan dengan contoh).

Jawaban :

Iya mengandung jumlah yang sama, karena dalam algoritma tersebut memiliki tujuan untuk mengunjungi semua titik dengan beban yang minimum sehingga semua titik dalam graph tersebut dapat dikunjungi