BAB III

PEMBAHASAN

Dalam hal ini akan dilakukan analisa kompleksitas waktu dari algoritma Prim dan algoritma Kruskal. Dengan memasukkan variabel berupa jumlah simpul, kedua algoritma tersebut akan dibandingkan waktu prosesnya, dan pemakaian memori/variabel.

3.1 *Pseudocode*

Dengan *pseudocode*, algoritma Prim dan algoritma Kruskal akan dianalisa dan ditentukan kompleksitas waktunya.

A. Algoritma Prim

Pseudo code Algoritma Prim sebagai berikut :

```
n = ... {jumlah simpul di dalam graf}
type himpunan_sisi_graf : array[1..n, 1..n] of real
                          {matrik ketetanggan berbobot}
type himpunan_sisi_pohon : array[1..n, 1..n] of integer
                           {sisi-sisi pohon merentang}

function Prim(input M:himpunan_sisi_graf) ⟶ himpunan_sisi_pohon
{menghasilkan pohon merentang minimum}

Deklarasi
  S : sisi_pohon
  p, q, k, index, i, j, min : integer
  TERDEKAT : array[1..n] of integer

Algoritma
  {inisialisasi}
  CariSisiMinimum(p, q)     {sisi e⇒(p, q) dengan bobot minimal}

  {masukan sisi (p, q) ke dalam himpunan S}
  S[i, 1] ⟵ p
  S[i, 2] ⟵ q
  for i ⟵ 1 to n do
        {inisialisasi larik TERDEKAT[1..n}
        if  M[i, q] < M[i, p] then
          TERDEKAT[i] ⟵ q
          else
          TERDEKAT[i] ⟵ p
```

```
      endif
endfor

{sisi (p, q) sudah terpilih, TERDEKAT[p] dan TERDEKAT[q] dinolkan}
TERDEKAT[p]← 0
TERDEKAT[q]← 0

{cari (n-2) buah sisi lain untuk S}
for i← 2 to n-1 do
  {cari simpul j di dalam larik TERDEKAT sedemikian sehingga
   TERDEKAT[j]≠0 dan M[j, TERDEKAT[j]] paling kecil}
   index←1
   min← 9999
   while index ≤ n do
      if TERDEKAT[index] ≠ 0 then
        if M[index, TERDEKAT[index]]< min then
          j← index
          min← M[index, TERDEKAT[index]]
        endif
        index← index + 1
   endwhile
  S[i, 1] ← j
  S[i, 2]← TERDEKAT[j]
  TERDEKAT[j]← 0

  {perbarui TERDEKAT}
  for k← 1 to n do
      if (TERDEKAT[k]≠ 0) and (M[k, j] < M[k, TERDEKAT[k]]) then
         TERDEKAT[k] ← j
      endif
  endfor
endfor

return S
```

Berikut analisa kompleksitas waktunya :

a. Inisialisasi

   `CariSisiMinimum(p, q)` akan dilakukan sebanyak *n* kali

b. Masukan sisi (p, q) ke dalam himpunan S

   `S[i, 1] ← p` akan dilakukan sebanyak 1 kali
   `S[i, 2] ← q` akan dilakukan sebanyak 1 kali

c. Inisialisasi larik TERDEKAT[1..n]

   `TERDEKAT[i] ← q atau TERDEKAT[i] ← p` akan dilakukan sebanyak *n* kali

d. TERDEKAT[p] dan TERDEKAT[q] dinolkan

`TERDEKAT[p]` ← `0` akan dilakukan sebanyak 1 kali
`TERDEKAT[q]` ← `0` akan dilakukan sebanyak 1 kali

c. Cari (n-2) buah sisi lain untuk S

- Pada kalang `while index ≤ n do`
  `j` ← `index` akan dilakukan sebanyak *n* kali
  `min` ← `M[index, TERDEKAT[index]]` akan dilakukan sebanyak *n* kali
- Pada kalang `for k` ← `1 to n do`
  `TERDEKAT[k]` ← `j` akan dilakukan sebanyak n^2 kali

Dengan demikian total kompleksitas waktu untuk algoritma Prim :

T(n) $= O(n + (1 + 1) + (1 + 1) + (n+ n) + n^2)$

$= O(3n + 4 + n^2)$

$= O(n^2)$

B. Algoritma Kruskal

Pseudo code Algoritma Kruskal sebagai berikut :

```
n = ... {jumlah sisi di dalam graf}

function Kruskal (input E : himpunan_sisi)← himpunan_sisi
{menghasilkan pohon merentang minimum}

Deklarasi
  T : himpunan_sisi
  e : sisi

Algoritma
  {mengurutkan sisi-sisi dari yang terkecil hingga terbesar}
   Urutsisi

  {cari n-1 sisi terkecil}
   S ← {}        {S ← himpunan sisi yang telah terpilih}
   while (jumlah sisi di dalam S < n-1) and (E ≠ {}) do
     e ← sisi yang mempunyai bobot terkecil di dalam E
     E ← E - e
     If T U e tidak membentuk sirkuit then
       S ← S U {e}
     endif
   endwhile

return S
```

Berikut perhitungan kompleksitas waktunya :

a. Mengurutkan sisi

`Urutsisi` akan dilakukan sebanyak $n \log n$ kali

b. Mencari n-1 sisi terkecil

`S ← {}` akan dilakukan sebanyak 1 kali
`e ← sisi` akan dilakukan sebanyak n kali
`E ← E - e` akan dilakukan sebanyak n kali
`S ← S U {e}` akan dilakukan sebanyak n kali

Dengan demikian total kompleksitas waktu untuk algoritma Kruskal :

$T(n) = O(n \log n + 1 + (n + n + n))$

$= O(n \log n + 1 + 3n)$

$= O(n \log n)$

3.2 Pemakaian Variabel

Setelah didapat kompleksitas waktu dari algoritma Prim dan Kruskal, kemudian analisa dilakukan dengan memasukan data variabel. Variabel yang dimasukan berupa data n, yaitu banyaknya simpul/sisi pada suatu graf. Pamasukkan variabel dimaksudkan untuk membandingkan waktu proses dari algoritma Prim dan Kruskal. Waktu proses ini dipengaruhi oleh faktor memori dan prosesor komputer yang digunakan. Dengan memasukan variabel, dapat diketahui perbedaan waktu proses algoritma Prim dan Kruskal. Selain itu, dapat diketahui algoritma mana yang mempunyai waktu proses yang lebih cepat.

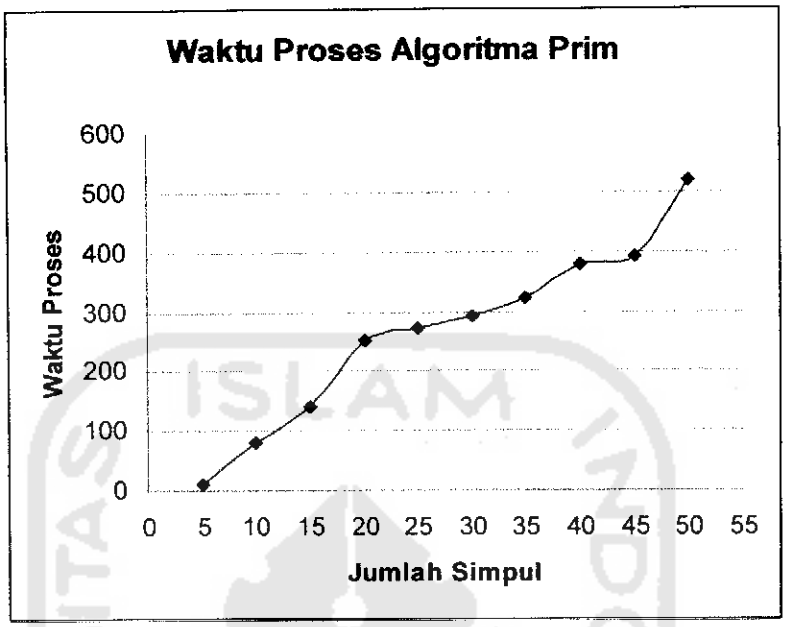
Berikut tabel perbandingan waktu proses dari algoritma Prim dan Kruskal dengan memasukan data variabel:

Tabel 3.1. Waktu Proses

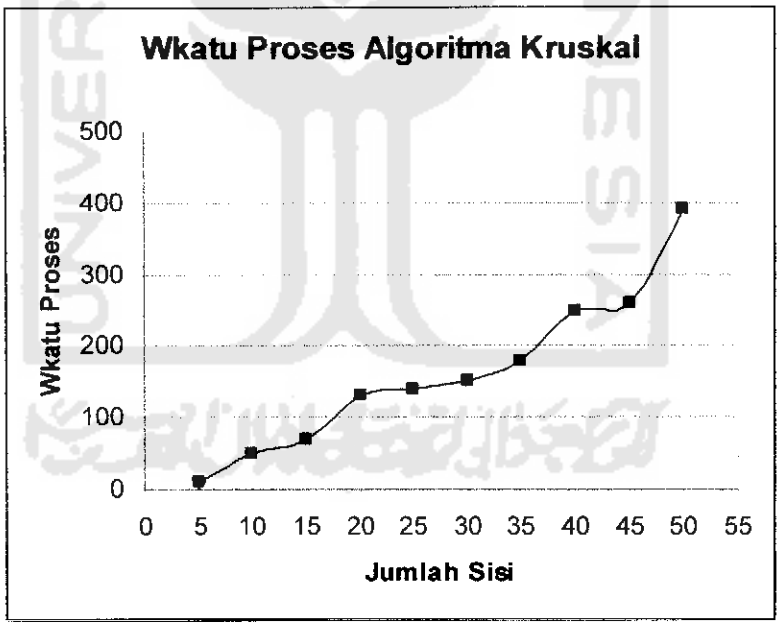
VARIABEL (*n*)	Waktu Proses (dalam Milidetik)	
	Prim	**Kruskal**
5	10	10
10	80	50
15	140	70
20	251	131
25	271	140
30	290	151
35	321	180
40	380	250
45	391	261
50	521	390

Dari data perhitungan kompleksitas waktu dengan memasukan data variabel ke dalam kompleksitas waktu algoritma Prim dan Kruskal, terlihat bahwa waktu proses algoritma Prim dan Kruskal sangatlah berbeda. Waktu proses yang dibutuhkan oleh algoritma Prim jauh lebih lama jika dibandingkan dengan waktu proses algoritma Kruskal. Waktu proses yang lama pada algoritma Prim dapat disebabkan karena kompleksnya proses kompleksitas waktu algoritma untuk memperbarui data *edge* yang terdekat menjadi nol kembali, kemudian untuk mencari *edge* yang terdekat berikutnya yaitu $O(n^2)$. Sedangkan untuk algoritma Kruskal, kompleksnya proses kompleksitas waktu hanya pada saat mengurutkan data *edge* dari yang terkecil hingga terbesar yaitu *n log* (*n*).

Berikut grafik waktu proses algoritma Prim dan Kruskal terhadap masukan variabel.



Gambar 3.1 Grafik waktu proses algoritma Prim



Gambar 3.2 Grafik waktu proses algoritma Kruskal

Dari tabel dan grafik antara algoritma Prim dan Kruskal dapat disimpulkan bahwa algoritma yang mempunyai waktu proses yang lebih cepat dan lebih baik dalam menentukan *minimum spanning tree* adalah algoritma Kruskal.