TUGAS AKHIR – KI1502

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI MATRIKS PADA SYSTEMATIC NODAL EQUATIONS: STUDI KASUS PROBLEM SPOJ KLASIK ELEC – ELECTRICAL ENGINEERING

MUHAMMAD IMADUDDIN
NRP 5111100042

Dosen Pembimbing I
Arya Yudhi Wijaya, S.Kom., M.Kom.

Dosen Pembimbing II
Rully Soelaiman, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

UNDERGRADUATE THESIS – KI1502

ALGORITHM DESIGN AND ANALYSIS OF MATRIX COMPUTATION ON SYSTEMATIC NODAL EQUATIONS: A CASE STUDY AT SPOJ CLASSICAL PROBLEM ELEC – ELECTRICAL ENGINEERING

MUHAMMAD IMADUDDIN
NRP 5111100042

Supervisor I
Arya Yudhi Wijaya, S.Kom., M.Kom.

Supervisor II
Rully Soelaiman, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2015

LEMBAR PENGESAHAN

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI MATRIKS PADA SYSTEMATIC NODAL EQUATIONS: STUDI KASUS PROBLEM SPOJ KLASIK ELEC – ELECTRICAL ENGINEERING

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Dasar dan Terapan Komputasi
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh
MUHAMMAD IMADUDDIN
NRP : 5111100042

Disetujui oleh Dosen Pembimbing Tugas Akhir

1. Arya Yudhi Wijaya, S.Kom., M.Kom. .....
   NIP: 198409042010121002 (Pembimbing 1)
2. Rully Soelaiman, S.Kom., M.Kom. .....
   NIP: 197002131994021001 (Pembimbing 2)

SURABAYA

JUNI, 2015

DESAIN DAN ANALISIS ALGOTIMA KOMPUTASI MATRIKS PADA SYSTEMATIC NODAL EQUATIONS: STUDI KASUS PROBLEM SPOJ KLASIK ELEC – ELECTRICAL ENGINERRING

Nama : **Muhammad Imaduddin**
NRP : **5111100042**
Jurusan : **Teknik Informatika**
Fakultas Teknologi Informasi ITS
Dosen Pembimbing I : **Arya Yudhi Wijaya, S.Kom., M.Kom.**
Dosen Pembimbing II : **Rully Soelaiman, S.Kom., M.Kom.**

ABSTRAK

Topik Tugas Akhir ini diangkat dari permasalahan pada Online Judge SPOJ dengan kode soal ELEC yang berjudul Electrical Engineering. Pada permasalahan ini diberikan beberapa jaringan resistor, dan untuk tiap jaringan resistor diminta untuk menghitung impedansi ekuivalen dari beberapa pasang node yang ada dalam jaringan tersebut. Salah satu cara yang paling umum dalam menghitung impedansi ekuivalen adalah dengan mentransformasi bagian-bagian kecil jaringan resistor sesuai dengan susunannya (seri, paralel, atau wye-delta), namun cara tersebut sulit untuk diimplementasikan secara efisien untuk jaringan yang besar dan kompleks.

Metode yang digunakan pada Tugas Akhir ini dalam mencari impedansi ekuivalen adalah dengan membuat Systematic Nodal Equations dan menerapkan teknik komputasi matriks yang efisien dalam mencari solusinya. Teknik komputasi matriks yang digunakan adalah dekomposisi LU yang memiliki kompleksitas waktu $O(n^3)$ untuk setiap penghitungan nilai impedansi ekuivalen pada jaringan resistor dengan n node.

Hasil uji coba terhadap implementasi yang dilakukan menunjukkan bahwa solusi yang digunakan sudah memenuhi

batasan-batasan yang ada pada permasalahan Electrical Engineering pada SPOJ.

Kata kunci: Dekomposisi LU, Impedansi Ekuivalen, Jaringan Resistor, Systematic Nodal Equations.

ALGORITHM DESIGN AND ANALYSIS OF MATRIX COMPUTATION ON SYSTEMATIC NODAL EQUATIONS: A CASE STUDY AT SPOJ CLASSICAL PROBLEM ELEC – ELECTRICAL ENGINEERING

Name	**: Muhammad Imaduddin**
NRP	**: 5111100042**
Department	**: Department of Informatics Faculty of Information Technology ITS**
Supervisor I	**: Arya Yudhi Wijaya, S.Kom., M.Kom.**
Supervisor II	**: Rully Soelaiman, S.Kom., M.Kom.**

ABSTRACT

This undergraduate thesis is based on a problem on SPOJ Online Judge coded ELEC with the title Electrical Engineering. In this problem we are given with some resistor networks, and for each network we are needed to calculate the equivalent impedance on some pairs of nodes inside the network. One commonly known method for calculating equivalent impedance on a resistor network is to transform small parts of the network in accordance to its arrangement (serial, parallel, or wye-delta shaped), but its hard to implement that in an efficient way for a large and complex network.

The method used in this undergraduate thesis for finding the value of equivalent impedance is by building Systematic Nodal Equations and applying an efficient matrix computation technique to find the solution. In this case the used matrix computation technique is LU Decomposition which has the time complexity of $O(n^3)$ for each calculation of equivalent impedance on a resistor network with n nodes.

The test result on implementation that has been done shows that the used solution meets the requirements of Electrical Engineering problem on SPOJ.

Keywords: Equivalent Impedance, LU Decomposition, Resistor Network, Systematic Nodal Equations.

KATA PENGANTAR

Puji syukur penulis kehadirat Tuhan Yang Maha Esa karena berkat rahmat dan karunia-Nya penulis dapat menyelesaikan Tugas Akhir yang berjudul

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI MATRIKS PADA SYSTEMATIC NODAL EQUATIONS: STUDI KASUS PROBLEM SPOJ KLASIK ELEC – ELECTRICAL ENGINEERING

Tugas Akhir ini merupakan salah satu syarat untuk memperoleh gelar Sarjana Komputer di Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya.

Penulis ingin menyampaikan terima kasih yang sebesar-besarnya atas dukungan dan semangat yang diberikan dan membantu penulis baik secara langsung maupun tidak dalam menyelesaikan Tugas Akhir ini. Penulis ingin mengucapkan terima kasih kepada:

1. Tuhan Yang Maha Esa karena berkat rahmat dan karunianya penulis berhasil menyelesaikan Tugas Akhir dengan baik.
2. Orang tua dan keluarga penulis yang selalu memberikan doa dan dukungan selama penulis kuliah di Jurusan Teknik Informatika ITS.
3. Ibu Dr. Eng. Nanik Suciati, S.Kom., M.Kom. selaku Ketua Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember.
4. Bapak Radityo Anggoro, S.Kom., M.Sc. selaku Koordinator Tugas Akhir.
5. Bapak Arya Yudhi Wijaya, S.Kom., M.Kom. selaku Dosen Pembimbing I Tugas Akhir yang telah memberikan bimbingan dan dukungan selama penulis menyelesaikan Tugas Akhir.

6. Bapak Rully Soelaiman, S.Kom., M.Kom. selaku Dosen Pembimbing II Tugas Akhir yang telah memberikan banyak ilmu selama penulis berkuliah di Jurusan Teknik Informatika ITS serta seluruh dukungan dan bantuan selama penulis menyelesaikan Tugas Akhir.
7. Bapak dan Ibu Dosen Jurusan Teknik Informatika ITS yang telah memberikan ilmu selama penulis berkuliah di Jurusan Teknik Informatika ITS.
8. Seluruh staf dan karyawan Jurusan Teknik Informatika ITS yang telah membantu selama penulis berkuliah di Jurusan Teknik Informatika ITS.
9. Teman-teman penulis terutama Ivan, Tracy, Andy, Yunus, Sindu, Indra, Bayu, Mas Teguh, Mas Fajrin dan Mas Angga atas ilmu dan pengalaman yang berharga selama penulis berkuliah di Jurusan Teknik Informatika ITS.
10. Seluruh pihak lain yang tidak bisa saya sebutkan satu persatu yang telah mendukung penulis.

Penulis mohon maaf apabila terdapat kekurangan dalam penulisan Tugas Akhir ini. Kritik dan saran penulis harapkan untuk perbaikan dan pembelajaran di kemudian hari. Semoga Tugas Akhir ini dapat memberikan manfaat yang sebesar-besarnya.

Surabaya, Juni 2015

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

Pada bab ini penulis menjelaskan tentang latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, dan sistematika penulisan Tugas Akhir.

1.1 Latar Belakang

Dalam sebuah jaringan resistor, impedansi ekuivalen antara dua node dapat dihitung menggunakan beberapa cara. Salah satu cara yang yang paling umum adalah dengan mentransformasi bagian-bagian dalam jaringan berdasarkan susunannya (seri, paralel, atau berbentuk *wye* maupun *delta*), sehingga dalam jaringan tersebut hanya tersisa dua node saja. Untuk jaringan yang cukup besar dan kompleks, cara tersebut sulit diimplementasikan ke dalam program secara efisien. Cara lain yang lebih mudah adalah dengan membuat sebuah sistem persamaan linier yang disebut *Systematic Nodal Equations* dari jaringan yang diberikan, kemudian mencari solusi dari sistem persamaan tersebut..

Topik Tugas Akhir ini diangkat dari permasalahan pada *Online Judge* SPOJ dengan kode soal ELEC yang berjudul *Electrical Engineering*. Pada permasalahan ini diberikan beberapa jaringan resistor, dan untuk tiap jaringan resistor diminta untuk menghitung impedansi ekuivalen dari beberapa pasang node yang ada dalam jaringan tersebut.

Pada Tugas Akhir ini akan diimplementasikan *Systematic Nodal Equations* dengan menggunakan algoritma komputasi matriks dalam proses penyelesaiannya. Algoritma komputasi matriks yang digunakan harus efisien agar dapat memenuhi batasan-batasan yang diberikan pada *Online Judge* SPOJ.

1.2 Rumusan Masalah

Berikut beberapa hal yang menjadi rumusan masalah dalam Tugas Akhir ini:

1. Bagaimana desain algoritma dan struktur data yang efisien untuk permasalahan komputasi matriks pada *Systematic Nodal Equations*?
2. Bagaimana implementasi algoritma dan struktur data yang efisien berdasarkan desain yang telah dilakukan?
3. Bagaimana uji coba untuk mengetahui kebenaran dan kinerja dari implementasi yang telah dilakukan?

1.3 Batasan Masalah

Berikut beberapa hal yang menjadi batasan masalah dalam Tugas Akhir ini:

1. Implementasi dilakukan menggunakan bahasa pemrograman C++.
2. Solusi yang diimplementasikan hanya untuk jaringan resistor dengan banyak node [1..100], banyak resistor antar node [0..1000], banyak pasangan node yang dicari impedansi ekuivalennya [0..10], dan nilai absolut bagian real maupun imajiner dari impedansi pada tiap resistor [0,001..1000].
3. Pada seluruh bagian proses penghitungan, matriks yang terbentuk merupakan matriks non singular.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah mengimplementasikan solusi untuk permasalahan komputasi matriks pada *Systematic Nodal Equations* dan melakukan uji coba untuk mengetahui kebenaran dan kinerja dari implementasi yang telah dilakukan.

1.5 Metodologi

Berikut metodologi yang digunakan dalam Tugas Akhir ini:

1. Penyusunan proposal Tugas Akhir
   Pada tahap ini dilakukan penyusunan proposal Tugas Akhir yang berisi definisi permasalahan komputasi matriks pada *Systematic Nodal Equations* beserta gambaran umum mengenai solusi untuk permasalahan tersebut.

2. Studi Literatur
   Pada tahap ini dilakukan studi literatur mengenai solusi untuk permasalahan komputasi matriks pada *Systematic Nodal Equations*. Literatur yang digunakan antara lain buku referensi dan artikel yang didapatkan dari internet.
3. Desain
   Pada tahap ini dilakukan desain algoritma serta struktur data dari solusi untuk permasalahan komputasi matriks pada *Systematic Nodal Equations*.
4. Implementasi
   Pada tahap ini dilakukan implementasi solusi untuk permasalahan komputasi matriks pada *Systematic Nodal Equations* berdasarkan analisis dan desain yang telah dilakukan sebelumnya.
5. Uji coba dan evaluasi
   Pada tahap ini dilakukan uji coba untuk menguji kebenaran dan kinerja dari implementasi yang telah dilakukan. Pengujian kebenaran dilakukan dengan melakukan pengiriman kode program ke *Online Judge* SPOJ sesuai dengan permasalahan yang terkait dan melihat umpan balik yang diberikan. Pengujian kinerja dilakukan dengan membandingkan kompleksitas yang didapat dari hasil pengujian dengan kompleksitas yang didapat dari hasil analisis. Selain itu dilakukan evaluasi untuk mencari bagian-bagian yang masih bisa dioptimasi dan melakukan perbaikan jika terdapat kesalahan.
6. Penyusunan buku Tugas Akhir
   Pada tahap ini dilakukan penyusunan buku Tugas Akhir yang berisi dokumentasi mengenai solusi untuk permasalahan komputasi matriks pada *Systematic Nodal Equations*.

1.6 Sistematika Penulisan

Berikut sistematika penulisan buku Tugas Akhir ini:

1. **BAB I: PENDAHULUAN**
   Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, dan sistematika penulisan Tugas Akhir.
2. **BAB II: TINJAUAN PUSTAKA**
   Bab ini berisi daftar teori mengenai permasalahan dan algoritma yang digunakan dalam Tugas Akhir ini.
3. **BAB III: DESAIN**
   Bab ini berisi desain algoritma serta struktur data yang digunakan dalam Tugas Akhir ini.
4. **BAB IV: IMPLEMENTASI**
   Bab ini berisi implementasi berdasarkan desain algoritma serta struktur data yang telah dilakukan.
5. **BAB V: UJI COBA DAN EVALUASI**
   Bab ini berisi uji coba dan evaluasi dari implementasi yang telah dilakukan.
6. **BAB VI: KESIMPULAN DAN SARAN**
   Bab ini berisi kesimpulan dari hasil uji coba yang telah dilakukan dan saran mengenai hal-hal yang masih bisa dikembangkan.

BAB II
TINJAUAN PUSTAKA

Pada bab ini penulis menjelaskan tentang beberapa tinjauan pustaka mengenai permasalahan dan algoritma yang digunakan dalam Tugas Akhir.

2.1 Sphere Online Judge (SPOJ)

Sphere Online Judge (SPOJ) adalah sistem penilaian *online* yang berisi berbagai jenis permasalahan pemrograman yang dapat diselesaikan oleh pengguna dengan mengirim kode sumber program yang berisi solusi dari permasalahan yang diberikan. Setiap permasalahan mempunyai format masukan yang diberikan dan format keluaran yang diminta. Selain itu setiap permasalahan juga mempunyai batasan tertentu termasuk lingkungan penilaian, batasan waktu, batasan memori, dan batasan dari permasalahan yang dideskripsikan.

Kode sumber program yang diterima sistem akan dikompilasi dan dijalankan pada lingkungan penilaian sistem. Program akan diuji menggunakan masukan dari berkas masukan permasalahan kemudian hasil keluaran program akan dibandingkan dengan berkas keluaran permasalahan. Sistem akan memberikan umpan balik kepada pengguna antara lain:

1. ***Accepted***, artinya program tidak melanggar batasan yang diberikan dan hasil keluaran program sama dengan berkas keluaran permasalahan.
2. ***Wrong Answer***, artinya hasil keluaran program tidak sama dengan berkas keluaran permasalahan.
3. ***Time Limit Exceeded***, artinya program melanggar batasan waktu yang diberikan.
4. ***Memory Limit Exceeded***, artinya program melanggar batasan memori yang diberikan.
5. ***Runtime Error***, artinya terjadi *error* pada saat program dijalankan.
6. ***Compile Error***, artinya terjadi *error* pada saat kompilasi kode sumber program.

Selain itu sistem juga memberikan umpan balik kepada pengguna mengenai waktu dan memori yang dibutuhkan program pada saat diuji menggunakan masukan dari berkas masukan permasalahan.

2.2 Permasalahan Electrical Engineering pada SPOJ

Salah satu permasalahan yang terdapat pada SPOJ adalah *Electrical Engineering* dengan kode ELEC. Deskripsi permasalahan tersebut ditunjukkan dalam **Gambar 2.1** dan **Gambar 2.2**.

ELEC - Electrical Engineering

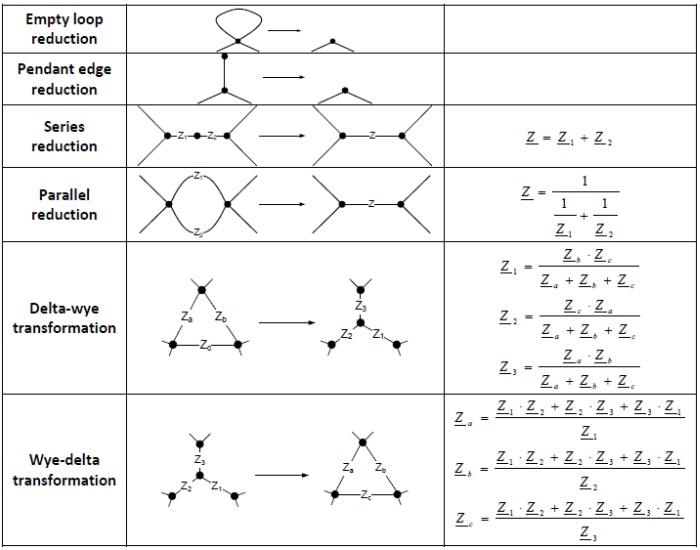
#bubble-cup-8-round-2

The electrical engineers' indefatigable strive towards environmentally friendly energy production translated into the recent boom of hydro, solar, wind and geothermal power plants. While the production side seems ready, these ambitious projects have their bottleneck in the transportation and distribution: Besides the energy losses that occur during transportation over long distances, the renewable energy sources cannot provide power on demand – they must be taken as provided by nature. Used at large scale in today's networks, unreliable green energy can disrupt the balance of power grids easily and cause huge damage along with large-scale power outages.

Serious effort is thus put on researching transient and dynamic phenomena in power grids. You are offered a position in the lab for linear and planar distribution networks. Given a description of the distribution network's line impedances Z_i , you are to find the equivalent impedance between some couples of nodes. The knowledge of such equivalent impedances may speed up the network analysis considerably! Impedances are complex number whose real part represents the resistive line behaviour while the imaginary part stands for the capacitive (negative) or inductive (positive) characteristic. Lines are bidirectional, that is impedance(a,b) equals impedance (b,a).

It was proven that any linear and planar graph (can be drawn in a way that its edges intersect only at their endpoints) can be reduced into a single equivalent edge that represents the equivalent impedance between its ending nodes, using the following six transformations:

Gambar 2.1 Deskripsi Permasalahan Electrical Engineering (1)



Gambar 2.2 Deskripsi Permasalahan Electrical Engineering (2)

Deskripsi singkat dari permasalahan tersebut adalah diberikan jaringan resistor yang direpresentasikan oleh graph tidak berarah dengan *N* node dan *C* edge/resistor. Setiap node diberi nomor mulai dari 1 sampai *N*. Setiap edge memiliki sebuah nilai yang merepresentasikan nilai impedansi dari dua node yang dihubungkannya. Kemudian akan diberikan *Z* pasang node untuk dihitung impedansi ekuivalen dari setiap pasang node tersebut. Impedansi ekuivalen antara dua node merupakan nilai impedansi pada edge yang menghubungkan kedua node tersebut setelah graph direduksi menjadi sebuah edge saja, yaitu edge yang menghubungkan kedua node tersebut, dengan menggunakan aturan-aturan transformasi yang ditunjukkan oleh **Gambar 2.2**.

Berikut merupakan format masukan dari permasalahan tersebut:

1. Masukan terdiri dari beberapa kasus uji, format masukan setiap kasus uji didefinisikan pada poin 2 hingga 5.
2. Baris pertama berisi 3 buah bilangan bulan *N*, *C* dan *Z* yang merepresentasikan jumlah node, jumlah edge, dan jumlah pasangan node yang akan dihitung nilai impedansi ekuivalennya.
3. *C* baris berikutnya masing-masing berisi 4 buah bilangan *U*, *V*, *Wr* dan *Wi* yang merepresentasikan tiap edge. *U* dan *V* merupakan nomor node yang dihubungkan. *Wr* dan *Wi* merupakan bilangan real yang merepresentasikan bagian real dan bagian imajiner dari nilai impedansi edge tersebut.
4. *Z* baris berikutnya masing-masing berisi dua buah bilangan *A* dan *B* yang merepresentasikan pasangan node yang akan dihitung nilai impedansi ekuivalennya.
5. Masukan berakhir jika $N = C = Z = 0$.

Berikut merupakan format keluaran dari permasalahan tersebut:

1. Untuk setiap kasus uji, format keluaran didefinisikan pada poin 2 dan 3.
2. Untuk masing-masing pasangan node yang akan dihitung nilai impedansi ekuivalennya, jika pasangan node tersebut terhubung, keluarkan 2 buah bilangan real dengan ketelitian 2 angka di belakang koma yang merepresentasikan bagian real dan bagian imajiner dari nilai impedansi ekuivalennya. Jika pasangan node tersebut tidak terhubung, keluarkan “no connection” tanpa tanda petik.
3. Keluaran diakhiri dengan sebuah baris kosong.

Berikut merupakan batasan dari permasalahan tersebut:

1. $1 \leq N \leq 100$
2. $0 \leq C \leq 1.000$
3. $0 \leq Z \leq 10$
4. $1 \leq U, V, A, B \leq N$
5. $10^{-3} \leq |Wr|, |Wi| \leq 10^{3}$
6. Lingkungan penilaian Intel Pentium G860 3 GHz.
7. Batasan waktu 0.079 detik.

8. Batasan ukuran kode sumber 50.000 B.
9. Batasan memori 1536 MB.

Contoh masukan dan keluaran dari permasalahan tersebut ditunjukkan dalam **Gambar 2.3** dan **Gambar 2.4**.

```
5 10 3
3 1 12.317 -0.779
5 3 30.107 0.289
5 1 27.447 -22.649
4 2 15.351 24.371
5 5 19.63 -3.549
2 2 11.841 18.757
4 5 4.834 -16.542
3 5 5.022 -22.387
2 5 24.768 -22.356
5 2 27.351 12.053
1 2
2 3
3 3

10 10 4
9 8 6.36 17.411
1 3 27.596 -6.484
9 10 4.735 -8.282
8 8 6.901 27.939
8 4 14.894 3.729
5 4 14.311 -2.422
10 10 11.009 6.225
4 4 3.196 -32.703
10 9 15.282 -14.799
3 9 20.473 27.158
10 9
8 1
2 9
9 6
```

Gambar 2.3 Contoh Masukan Permasalahan Electrical Engineering

```
23.37 -7.26
19.61 -6.97
0.00 0.00

3.79 -5.46
54.43 38.09
no connection
no connection
```

Gambar 2.4 Contoh Keluaran Permasalahan Electrical Engineering

Gambar 2.3 menunjukkan contoh masukan dengan 2 kasus uji yang dipisahkan oleh sebuah baris kosong. Baris pertama pada kasus uji pertama menunjukkan nilai *N* atau jumlah node 5, nilai *C* atau jumlah edge 10 dan nilai *Z* jumlah pasangan node yang ingin dicari impedansi ekuivalennya 3. Tiap baris pada 10 baris berikutnya menunjukkan deskripsi edge, diawali dengan edge pertama yang menghubungkan node 3 dan node 1 dan memiliki nilai impedansi (12,317 - 0,779i). Setelah itu terdapat 3 baris di mana tiap baris berisi 2 node yang ingin dicari impedansi ekuivalennya, diawali dengan node 1 dan node 2.

Gambar 2.4 menunjukkan keluaran dari 2 kasus uji untuk contoh masukan pada **Gambar 2.3**. Kasus uji pertama memiliki 3 baris keluaran, sesuai dengan nilai *Z* pada masukan. Baris pertama menunjukkan bahwa nilai impedansi ekuivalen antara node 1 dan 2 adalah (23,27 – 7,26i), baris kedua menunjukkan bahwa nilai impedansi ekuivalen antara node 2 dan 3 adalah (19,61 – 6,97i) dan baris ketiga menunjukkan bahwa nilai impedansi ekuivalen antara node 3 dan 3 adalah (0.00 + 0.00i). Baris ketiga pada keluaran kasus uji kedua berisi “no connection” yang menunjukkan bahwa node 2 dan 9 pada graph yang terbentuk pada kasus uji kedua tidak terhubung.

2.3 Systematic Nodal Equations

Dalam sebuah jaringan dengan n node, tegangan antara sebuah node sembarang dengan $(n-1)$ node lainnya membentuk sekumpulan variabel yang independen. Sebagai contoh, pada

jaringan dalam **Gambar 2.5**(a), tegangan antar node yang terbentuk adalah $v_1(t)$, $v_2(t)$, $v_3(t)$ dan $v_4(t)$.

Semua tegangan antar node dibuat dengan tanda polaritas acuan negatif berada pada node yang sama, seperti yang ditunjukkan dalam **Gambar 2.5**(b), kemudian setiap node diberi nomor sesuai dengan **Gambar 2.5**(c). Node 0 (atau disebut juga node acuan) adalah node yang digunakan oleh semua tegangan antar node. Tegangan v_3 misalnya, adalah tegangan antara node 3 dan node 0, dengan tanda polaritas acuan positif berada pada node 3.

Jika keempat tegangan tersebut diketahui, tegangan lain maupun arus manapun pada jaringan tersebut akan dapat dicari dengan mudah. Pada **Gambar 2.6**, dapat diketahui bahwa:

$$v_5 = v_3 - v_2$$

$$i_2 = i_3 - 6\sin t = \frac{v_3 - v_4}{10} - 6\sin t$$

Dengan menerapkan hukum arus Kirchhoff, sebuah persamaan yang independen bisa didapatkan untuk setiap node. Untuk node 1, sesuai dengan **Gambar 2.7**(a),

$$i_a + i_b + i_c = 0$$

Dengan mensubstitusi arus pada resistor dengan tegangan pada node,

$$\frac{v_1}{4} + \frac{v_1 - v_2}{5} + \frac{v_1 - v_2}{7} = 0$$

Untuk node 2, sesuai dengan **Gambar 2.7**(b),

$$i_d + i_e + i_f + i_g = 0$$

$$\frac{v_2 - v_1}{7} + \frac{v_2 - v_1}{5} + \frac{v_2 - v_3}{8} + \frac{v_2 - v_4}{9} = 0$$

Untuk node 3, sesuai dengan **Gambar 2.7**(c),

$$i_h + i_i = 6\sin t$$

$$\frac{v_3 - v_2}{8} + \frac{v_3 - v_4}{10} = 6\sin t$$

Untuk node 4, sesuai dengan **Gambar 2.7**(d),

$$i_j + i_k = -6\sin t - 3\cos 2t$$

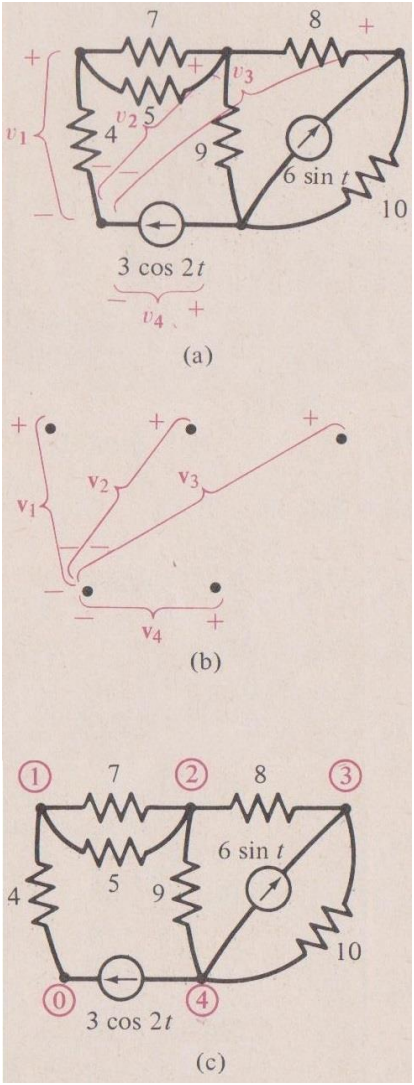
$$\frac{v_4 - v_2}{9} + \frac{v_4 - v_3}{10} = -6\sin t - 3\cos 2t$$

Dengan menyusun ulang persamaan-persamaan dari 4 node di atas, bisa didapatkan sebuah *Systematic Nodal Equations*

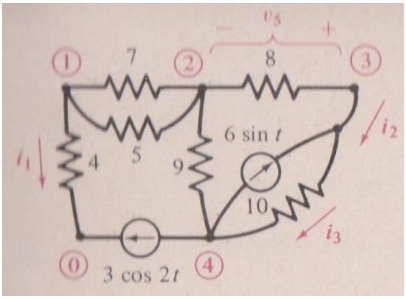
$$\begin{aligned}
\left(\frac{1}{4}+\frac{1}{5}+\frac{1}{7}\right)v_1 - \left(\frac{1}{5}+\frac{1}{7}\right)v_2 - (0)v_3 - (0)v_4 -= 0 \\
-\left(\frac{1}{5}+\frac{1}{7}\right)v_1 + \left(\frac{1}{5}+\frac{1}{7}+\frac{1}{8}+\frac{1}{9}\right)v_2 - \left(\frac{1}{8}\right)v_3 - \left(\frac{1}{9}\right)v_4 = 0 \\
-(0)v_1 - \left(\frac{1}{8}\right)v_2 + \left(\frac{1}{8}+\frac{1}{10}\right)v_3 - \left(\frac{1}{10}\right)v_4 = 6\sin t \\
-(0)v_1 - \left(\frac{1}{9}\right)v_2 - \left(\frac{1}{10}\right)v_3 + \left(\frac{1}{9}+\frac{1}{10}\right)v_4 = -6\sin t - 3\cos 2t
\end{aligned} \qquad (1)$$

yang kemudian dapat dicari nilai dari tegangan antar node v_1, v_2, v_3 dan v_4.

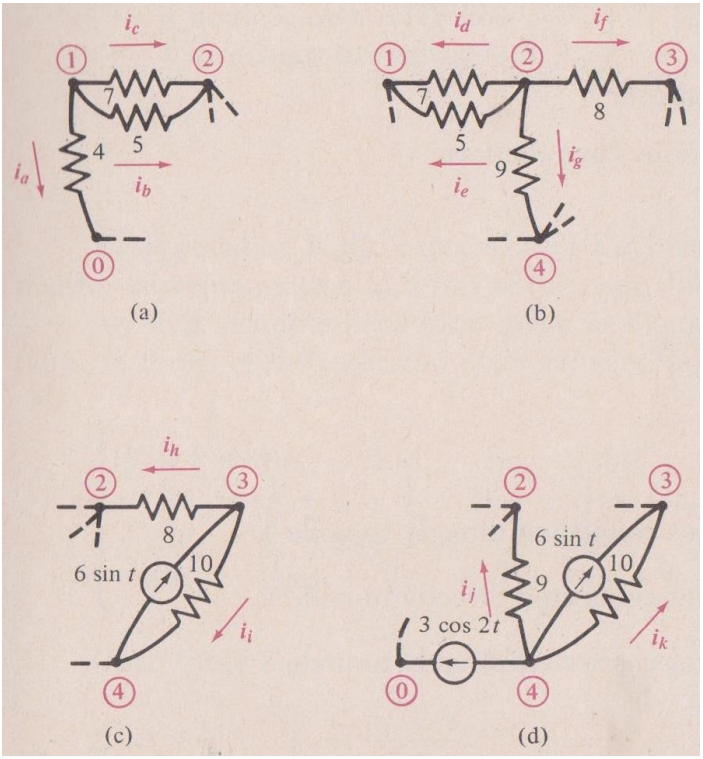
Persamaan untuk node acuan (dalam kasus ini node 0) tidak perlu diperhitungkan karena merupakan kombinasi linier dari persamaan-persamaan node lainnya.



Gambar 2.5 Jaringan dengan 5 Node



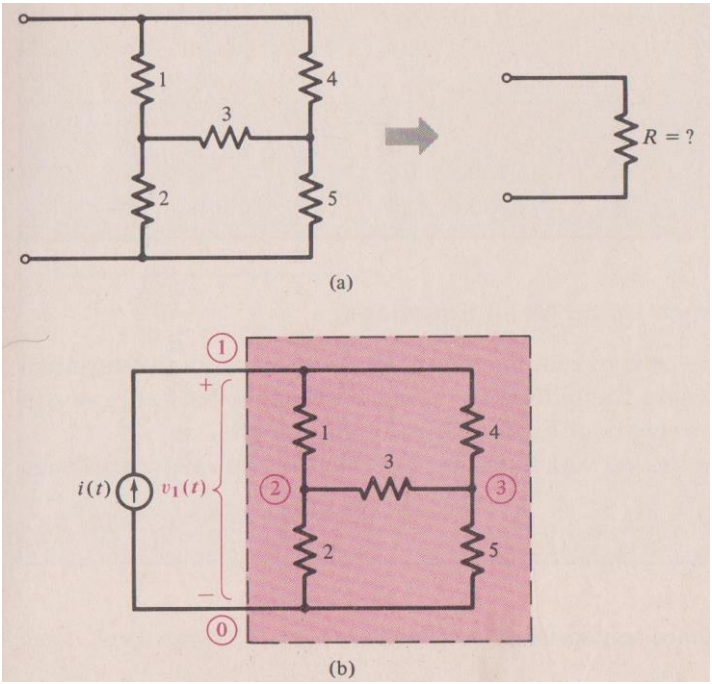
Gambar 2.6 Mencari Nilai Variabel Lain dalam Jaringan



Gambar 2.7 Analisis untuk Node 1 hingga 4

2.4 Hambatan Ekuivalen Menggunakan Nodal Equations

Nodal equations dapat digunakan untuk menghitung hambatan ekuivalen pada sebuah jaringan resistor, seperti contoh jaringan dengan 2 terminal yang ditunjukkan dalam **Gambar 2.8**(a). Misalkan diberikan arus $i(t)$ antara kedua terminal seperti pada **Gambar 2.8**(b), kemudian ditentukan node acuan untuk tegangan $v(t)$. Tegangan $v(t)$ akan sebanding dengan $i(t)$, dan rasio dari keduanya merupakan hambatan ekuivalen antara 2 terminal pada jaringan.



Gambar 2.8 Jaringan Resistor dengan 2 Terminal (a), Arus Diberikan pada Jaringan Resistor (b)

Pada contoh tersebut, *Systematic Nodal Equations* yang terbentuk adalah

$$\left(\frac{1}{1}+\frac{1}{4}\right)v_1-\left(\frac{1}{1}\right)v_2-\left(\frac{1}{4}\right)v_3=i(t)$$
$$-\left(\frac{1}{1}\right)v_1+\left(\frac{1}{1}+\frac{1}{2}+\frac{1}{3}\right)v_2-\left(\frac{1}{3}\right)v_3=0$$
$$-\left(\frac{1}{4}\right)v_1-\left(\frac{1}{3}\right)v_2+\left(\frac{1}{3}+\frac{1}{4}+\frac{1}{5}\right)v_3=0$$

di mana arus $i(t)$, meskipun tidak ditentukan, dianggap memiliki nilai yang diketahui. Setiap tegangan dan arus dalam jaringan akan sebanding dengan $i(t)$.

Dengan menghilangkan bentuk pecahan pada *Systematic Nodal Equations* di atas, didapatkan

$$5v_1-4v_2-v_3=4i(t)$$
$$-6v_1+11v_2-2v_3=0$$
$$-15v_1-20v_2+47v_3=0$$

Dengan menggunakan aturan Cramer dapat dihitung

$$v_1(t)=\frac{\begin{vmatrix}4i(t) & -4 & -1\\ 0 & 11 & -2\\ 0 & -20 & 47\end{vmatrix}}{\begin{vmatrix}5 & -4 & -1\\ -6 & 11 & -2\\ -15 & -20 & 47\end{vmatrix}} \qquad (2)$$

$$=\frac{4i(t)\begin{vmatrix}11 & -2\\ -20 & 47\end{vmatrix}}{5\begin{vmatrix}11 & -2\\ -20 & 47\end{vmatrix}+4\begin{vmatrix}-6 & -2\\ -15 & 47\end{vmatrix}-\begin{vmatrix}-6 & 11\\ -15 & -20\end{vmatrix}}$$

$$=\frac{1908i(t)}{5(477)+4(-312)-(285)}$$

$$=\frac{1908i(t)}{852}$$

Dan hambatan ekuivalen dari jaringan dengan 2 terminal tersebut adalah

$$R_{equiv} = \frac{v_1(t)}{i(t)} = \frac{1908}{852}\Omega$$

Metode yang sama juga dapat digunakan dalam menghitung impedansi ekuivalen dalam sebuah jaringan resistor.

2.5 Dekomposisi LU

Dekomposisi LU merupakan metode dekomposisi/faktorisasi sebuah matriks persegi menjadi matriks segitiga bawah dan matriks segitiga atas. Dengan kata lain, bisa didapatkan persamaan

$$A = LU$$

Di mana A merupakan matriks persegi, L merupakan matriks segitiga bawah, dan U merupakan matriks segitiga atas.

Terdapat beberapa algoritma dekomposisi LU antara lain:

1. Algoritma Doolittle yaitu dekomposisi LU di mana matriks segitiga bawah yang dihasilkan merupakan matriks segitiga unit.
2. Algoritma Crout yaitu dekomposisi LU di mana matriks segitiga atas yang dihasilkan merupakan matriks segitiga unit.

2.5.1 Dekomposisi LU Menggunakan Algoritma Doolittle

Diberikan sebuah matriks A berukuran $n \times n$ sehingga

$$\begin{aligned} A &= \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \\ &= LU \\ &= \begin{bmatrix} 1 & & & 0 \\ l_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \dots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ & u_{22} & \dots & u_{2n} \\ & & \ddots & \vdots \\ 0 & & & u_{nn} \end{bmatrix} \end{aligned} \quad (3)$$

Maka nilai elemen-elemen matriks L dan U dapat dicari dengan prosedur berikut:

$$\begin{aligned}&\text{Untuk tiap } i = 1,2,\dots,n,\\ &\quad u_{ii} = a_{ii} - \textstyle\sum_{k=1}^{i-1} l_{ik}u_{ki}\\ &\quad \text{Untuk tiap } j = i+1, i+2, \dots, n,\\ &\qquad u_{ij} = a_{ij} - \textstyle\sum_{k=1}^{i-1} l_{ik}u_{kj}\\ &\qquad l_{ji} = \frac{a_{ji} - \sum_{k=1}^{i-1} l_{jk}u_{ki}}{u_{ii}}\end{aligned} \tag{4}$$

Dari hasil dekomposisi yang didapat, determinan dari matriks A dapat dihitung dengan menggunakan persamaan:

$$\det(A) = \det(L) \times \det(U)$$

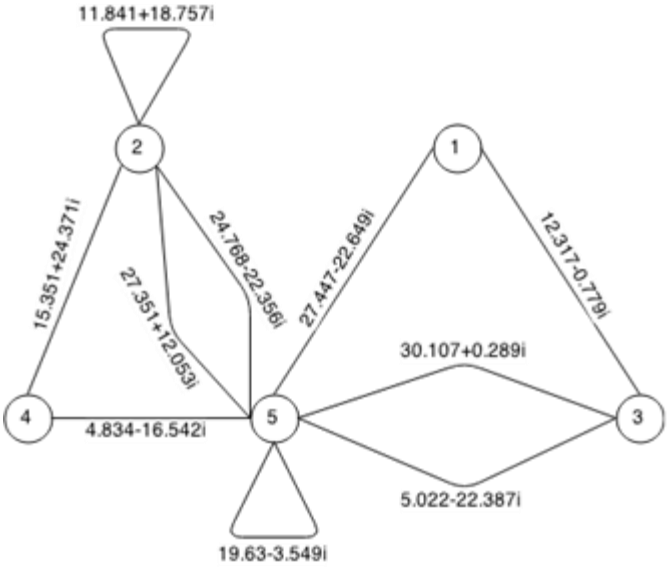
Determinan dari matriks segitiga merupakan perkalian dari elemen-elemen di diagonal utamanya, sehingga

$$\begin{aligned}\det(A) &= (1 \times 1 \times \dots \times 1) \times (u_{11} \times u_{22} \times \dots \times u_{nn})\\ &= u_{11} \times u_{22} \times \dots \times u_{nn}\end{aligned} \tag{5}$$

Untuk matriks berukuran besar, cara penghitungan determinan tersebut relatif lebih cepat jika dibandingkan dengan cara *naive* menggunakan metode ekspansi Laplace.

2.6 Analisis Solusi pada Contoh Kasus Permasalahan Electrical Engineering SPOJ

Pada subbab 2.2 telah diberikan contoh masukan dan keluaran untuk permasalahan *Electrical Engineering* pada SPOJ. Untuk contoh masukan pertama pada **Gambar 2.3** dengan banyak node 5 dan banyak edge 10, graph yang terbentuk setelah informasi semua edge dimasukkan ditunjukkan dalam **Gambar 2.9**. Selanjutnya untuk pasangan node pertama yang ingin dicari impedansi ekuivalennya, yaitu node 1 dan node 2, *Systematic Nodal Equations* yang terbentuk adalah



Gambar 2.9 Graph Kasus Uji Pertama pada Contoh Masukan SPOJ Electrical Engineering

$$(0.10 + 0.02i)v_1 + (-0.08 - 0.01i)v_3 + (0)v_4 + (-0.02 - 0.02i)v_5 = 1$$
$$(-0.08 - 0.01i)v_1 + (0.12 + 0.05i)v_3 + (0)v_4 + (-0.04 - 0.04i)v_5 = 0$$
$$(0)v_1 + (0)v_3 + (0.03 + 0.03i)v_4 + (-0.02 - 0.06i)v_5 = 0$$
$$(-0.02 - 0.02i)v_1 + (-0.04 - 0.04i)v_3 + (-0.02 - 0.06i)v_4 + (0.13 + 0.12i)v_5 = 0$$

di mana node 2 dijadikan sebagai node acuan dan arus yang diberikan antara node 1 dan node 2 adalah 1 Ampere. Persamaan pertama merupakan persamaan untuk node 1, persamaan kedua merupakan persamaan untuk node 3, persamaan ketiga adalah persamaan untuk node 4, dan persamaan terakhir adalah persamaan untuk node 5. Selanjutnya dari persamaan-persamaan tersebut dicari

$$v_1(t) = \frac{\begin{vmatrix} 1 & -0.08-0.01i & 0 & -0.02-0.02i \\ 0 & 0.12+0.05i & 0 & -0.04-0.04i \\ 0 & 0 & 0.03+0.03i & -0.02-0.06i \\ 0 & -0.04-0.04i & -0.02-0.06i & 0.13+0.12i \end{vmatrix}}{\begin{vmatrix} 0.10+0.02i & -0.08-0.01i & 0 & -0.02-0.02i \\ -0.08-0.01i & 0.12+0.05i & 0 & -0.04-0.04i \\ 0 & 0 & 0.03+0.03i & -0.02-0.06i \\ -0.02-0.02i & -0.04-0.04i & -0.02-0.06i & 0.13+0.12i \end{vmatrix}}$$

Untuk memudahkan penghitungan nilai determinan, dilakukan dekomposisi LU pada kedua matriks di atas, sehingga didapatkan

$$v_1(t) = \frac{\begin{vmatrix} 1 & -0.08-0.01i & 0 & -0.02-0.02i \\ 0 & 0.12+0.05i & 0 & -0.04-0.04i \\ 0 & 0 & 0.03+0.03i & -0.02-0.06i \\ 0 & -0.42-0.18i & -1.07-0.79i & 0.15+0.02i \end{vmatrix}}{\begin{vmatrix} 0.10+0.02i & -0.08-0.01i & 0 & -0.02-0.02i \\ -0.76-0.12i & 0.06+0.05i & 0 & -0.06-0.05i \\ 0 & 0 & 0.03+0.03i & -0.02-0.06i \\ -0.24-0.12i & -1.00+0.00i & -1.07-0.79i & 0.10-0.01i \end{vmatrix}}$$

Matriks L dan matriks U yang merupakan hasil dekomposisi dari tiap matriks disimpan dalam 1 matriks yang sama dengan mengabaikan diagonal utama matriks L yang bernilai 1 semua. Determinan dari masing-masing matriks adalah perkalian elemen-elemen diagonal utama matriks L dan matriks U, sehingga

$$\begin{aligned} v_1(t) &= \frac{(0.0003382011 + 0.000811551i)}{(0.0000033542 + 0.000035771i)} \\ &= 23.37 - 7.26i \end{aligned}$$

Selanjutnya impedansi ekuivalen dapat dicari dengan

$$Z_{equiv} = \frac{v_1(t)}{i(t)} = \frac{23.37 - 7.26i}{1} = 23.37 - 7.26i$$

Nilai ini sesuai dengan keluaran pertama yang ditunjukkan dalam **Gambar 2.4**. Untuk contoh masukan yang selanjutnya dapat diselesaikan menggunakan cara yang sama.

2.7 Pembuatan Data Generator untuk Uji Coba

```
GenerateData()
1.  // buat 10 berkas dengan jumlah node yang bervariasi
2.  for num_nodes = 10 to 100 increment by 10 do
3.     filename = num_nodes + ".txt"
4.     myfile = open file filename
5.     // buat 10 test case per berkas
6.     for tc = 1 to 10 do
7.            print_on_myfile(num_nodes+" 1000 10")
8.            // buat 1000 edge per test case
9.            for e = 1 to 1000 do
10.                  // bangkitkan deskripsi edge
    secara acak pada rentang sesuai batasan permasalahan
11.                  n1 = random between 1 and
    num_nodes
12.                  n2 = random between 1 and
    num_nodes
13.                  re = random between 0.001 and 1000
14.                  im = random between 0.001 and 1000
15.                  im_sign = random between 1 and 2
16. if im_sign = 1 then
17.                           im = im * -1
18.                  print_on_myfile(n1+" "+n2+" "+re+"
    "+im)
19.           // cari 10 impedansi ekuivalen per test
    case
20.           for i = 1 to 10 do
21.                  n1 = random between 1 and
    num_nodes
22.                  n2 = random between 1 and
    num_nodes
23.                  print_on_myfile(n1+" "+n2)
24.    print_on_myfile("0 0 0")
25.    save myfile
```

Gambar 2.10 Pseudocode Data Generator

Pembuatan data generator dilakukan untuk membuat kasus uji yang akan digunakan untuk uji coba yang akan dijelaskan pada bab V. Data generator ini disesuaikan dengan format masukan yang telah dijelaskan pada subbab 2.2. Pseudocode data generator ditunjukkan dalam **Gambar 2.10**.

Data generator akan membuat beberapa berkas uji coba. Masing-masing berkas memiliki banyak node pada jaringan resistor yang bervariasi pada rentang 10 hingga 100 dengan interval nilai 10. Banyak kasus uji, banyak edge pada jaringan resistor, dan banyak pasangan node yang ingin dicari nilai impedansi ekuivalennya dibuat sama untuk semua berkas uji coba, yaitu 10 kasus uji, 1000 edge dan 10 pasangan node. Untuk setiap edge yang akan dibuat, dua node yang dihubungkan akan dipilih secara acak. Nilai bagian real dari impedansi dibangkitkan secara acak pada rentang [0.001..1000] dengan ketelitian 3 angka di belakang koma, sedangkan nilai bagian imajiner dari impedansi juga dibangkitkan secara acak pada rentang ±[0.001..1000] dengan ketelitian 3 angka di belakang koma. Setiap pasangan node yang akan dihitung impedansi ekuivalennya dipilih secara acak dari node-node yang tersedia.

BAB III
DESAIN

Pada bab ini penulis menjelaskan tentang desain algoritma serta struktur data yang digunakan dalam Tugas Akhir.

3.1 Deskripsi Umum Sistem

Sistem akan menerima masukan berupa banyak node, banyak edge, banyak pasangan node yang akan dicari nilai impedansi ekuivalennya, deskripsi dari setiap edge, dan pasangan-pasangan node yang akan dicari nilai impedansi ekuivalennya. Untuk setiap deskripsi edge yang dimasukkan, sistem akan melakukan update terhadap persamaan-persamaan pada *Systematic Nodal Equations*. Setelah semua deskripsi edge sudah dimasukkan, sistem akan melakukan pengelompokan terhadap node-node yang terhubung dengan memberikan nomor kelompok untuk setiap node. Kemudian untuk setiap pasangan node yang akan dicari nilai impedansi ekuivalennya, sistem akan mengecek terlebih dahulu apakah kedua node tersebut terhubung. Jika terhubung, maka sistem akan menghitung nilai impedansi ekuivalennya dan mengeluarkannya. Jika tidak terhubung, maka sistem akan mengeluarkan “no connection”. Pseudocode untuk fungsi Main ditunjukkan dalam **Gambar 3.1**.

3.2 Desain Algoritma

Sistem terdiri dari 4 fungsi utama yaitu UpdateEquations, UpdateNodeGroup, EquivalentImpedance dan Determinant. Pada subbab ini akan dijelaskan maksud dan desain algoritma dari masing-masing fungsi tersebut.

3.2.1 Desain Fungsi UpdateEquations

Fungsi UpdateEquations digunakan untuk melakukan update pada *Systematic Nodal Equations* sesuai dengan penjelasan teori pada subbab 2.3 untuk setiap deskripsi edge yang dimasukkan. Pseudocode fungsi ini ditunjukkan dalam **Gambar 3.2**.

```
Main()
26. input number of node, number of edge, number of
    query
27. if number of node, number of edge, number of query
    is not 0
28.     input description of edges
29.     for each edge(u, v) which u!=v
30.             UpdateEquations(u, v, impedance)
31.     UpdateNodeGroup()
32.     input queries
33.     for each query(node1, node2)
34.             if node1 and node2 is connected
35.                     output EquivalentImpedance(node1,
    node2)
36.             else
37.                     output “no connection”
38.     output empty line
```

Gambar 3.1 Pseudocode Fungsi Main

```
// parameters:
//      u               : first node
//      v               : second node
//      impendace       : value of impedance on the edge
UpdateEquations(u, v, impedance)
1.  equations_coefficient[u][u] += 1/impedance
2.  equations_coefficient[v][v] += 1/impedance
3.  equations_coefficient[u][v] -= 1/impedance
4.  equations_coefficient[v][u] -= 1/impedance
```

Gambar 3.2 Pseudocode Fungsi UpdateEquations

Pada sistem persamaan (1) di subbab 2.3, dapat dilihat bahwa nilai koefisien pada tiap variabel tegangan pada node berkaitan dengan jumlah nilai kebalikan impedansi edge-edge yang terhubung pada node tersebut. Baris 1 hingga 4 pada **Gambar 3.2** melakukan update terhadap koefisien-koefisien pada persamaan-persamaan di *Systematic Nodal Equations* berdasarkan deskripsi edge yang diberikan. *equations_coefficient* menyimpan koefisien-koefisien tersebut, dengan *equations_coefficient[i][j]* menandakan koefisien dari v_j pada persamaan untuk node *i*.

3.2.2 Desain Fungsi UpdateNodeGroup

Fungsi UpdateNodeGroup digunakan untuk mengelompokkan node-node yang terhubung. Pseudocode fungsi ini ditunjukkan pada **Gambar 3.3**.

```
// node_group is initialized with -1 for all nodes
UpdateNodeGroup()
1.  for each node u which node_group[u] != -1
2.      node_group[u] = u
3.      for each node v connected to u
4.              node_group[v] = u
```

Gambar 3.3 Pseudocode Fungsi UpdateNodeGroup

node_group menyimpan nomor kelompok dari tiap node, dengan *node_group[i]* menandakan nomor kelompok dari node *i*. Awalnya semua node memiliki *node_group* bernilai -1 yang berarti belum memiliki kelompok. Fungsi ini dimulai dengan melakukan iterasi untuk tiap node yang belum memiliki kelompok, kemudian menandai node tersebut dan semua node lain yang terhubung dengan nomor kelompok yang sama.

3.2.3 Desain Fungsi EquivalentImpedance

Fungsi EquivalentImpedance digunakan untuk menghitung impedansi ekuivalen dari 2 node yang terhubung sesuai dengan penjelasan teori pada subbab 2.3 dan 2.4, dengan menetapkan 1 Ampere sebagai nilai arus yang diberikan. Pseudocode fungsi ini ditunjukkan pada **Gambar 3.4**.

Baris 1 dan 2 melakukan pengambilan node-node yang terhubung dengan node *node1* dan *node2* yang kemudian disimpan pada *con_nodes*. Baris 3 melakukan pembuatan matriks koefisien node-node yang ada pada *con_nodes* yang kemudian disimpan pada *con_matrix*. Nilai-nilai koefisien diambil dari nilai-nilai yang telah dihasilkan oleh fungsi UpdateEquations.

Baris 5 hingga 7 mengganti kolom pada *con_matrix* yang berisi koefisien node *node1* dengan vektor bagian kanan dari *Systematic Nodal Equations* untuk node-node yang ada pada *con_nodes*, seperti contoh pada persamaan (2) di subbab 2.4. Baris

4 dan 8 menghitung nilai impedansi ekuivalen yang akan dikembalikan oleh fungsi ini pada baris 9. Karena arus yang digunakan bernilai 1 Ampere, maka nilai impedansi ekuivalennya sama dengan nilai tegangan yang dihitung menggunakan aturan Cramer.

```
// parameters:
//     node1   : first node
//     node2   : second node
// returns:
//     equivalent impedance between node1 and node2
EquivalentImpedance(node1, node2)
1.  for each node u connected to node1 and node2 && u !=
    node2
2.      con_nodes.add(u)
3.  con_matrix = coefficient matrix of nodes in
    con_nodes
4.  eq_impedance = con_matrix.determinant
5.  for each node u in con_nodes
6.      con_matrix[u][node1] = 0
7.  con_matrix[node1][node1] = 1
8.  eq_impedance = con_matrix.determinant/eq_impedance
9.  return eq_impedance
```

Gambar 3.4 Pseudocode Fungsi EquivalentImpedance

3.2.4 Desain Fungsi Determinant

Fungsi Determinant digunakan untuk menghitung determinan dari sebuah matriks persegi yang diberikan dengan menggunakan metode dekomposisi LU yang telah dijelaskan pada subbab 2.5.1. Pseudocode fungsi ini ditunjukkan pada **Gambar 3.5**.

Elemen-elemen matriks awal akan disalin ke *lu_element* yang ditunjukkan oleh baris 1 dan 2. Penyalinan isi matriks awal diperlukan karena matriks awal masih diperlukan pada proses selanjutnya, sehingga isinya tidak boleh berubah. Fungsi LUDecompose pada baris 3 melakukan dekomposisi LU pada *lu_element* sesuai persamaan (3) pada subbab 2.5, yang kemudian nilai elemen-elemennya akan berubah sesuai dengan hasil dekomposisi. Pseudocode untuk fungsi LUDecompose sendiri

ditunjukkan dalam **Gambar 3.6 dan** sesuai dengan persamaan (4) pada subbab 2.5.1.

Baris 4 hingga 6 pada **Gambar 3.5** menghitung nilai determinan yang akan dikembalikan, yaitu hasil perkalian dari elemen-elemen diagonal utama dari *lu_element* sesuai seperti pada persamaan (5) di subbab 2.5.1.

```
// paramaters:
//     size          : size of the matrix
//     element[][]   : elements of the matrix
//returns:
//     determinant of the matrix
Determinant(size, element)
1.  for each element[i][j] where i,j<=size
2.     lu_element[i][j] = element[i][j]
3.  LUDecompose(size, lu_element)
4.  determinant = 1;
5.  for each lu_element[i][i] where i<=size
6.     determinant *= lu_element[i][j]
7.  return determinant
```

Gambar 3.5 Pseudocode Fungsi Determinant

```
// parameters:
//     size          : size of the matrix
//     element[][]   : elements of the matrix
LUDecompose(size, element)
1.  for i = 1 to size do
2.     for j = i+1 to size do
3.             lu[j][i] = lu[j][i] / lu[i][i]
4.             for k = i+1 to size do
5.                     lu[j][k] = lu[j][k] –
    (lu[j][i]*lu[i][k])
```

Gambar 3.6 Pseudocode Fungsi LUDecompose

3.3 Desain Struktur Data

Permasalahan Electrical Engineering membutuhkan pemrosesan bilangan kompleks, sehingga struktur data yang sesuai dibuat untuk menampung jenis data bilangan kompleks. Struktur data ini berisi dua bilangan real A dan B yang merepresentasikan bilangan kompleks yang berbentuk $(A + Bi)$ serta beberapa

definisi operasi aritmetika dasar pada bilangan kompleks, seperti penjumlahan, pengurangan, perkalian dan pembagian. Selain itu juga terdapat sebuah fungsi untuk mengecek kesamaan dua buah bilangan kompleks. Pseudocode struktur data bilangan kompleks ditunjukkan dalam **Gambar 3.7**.

```
Complex
1.  real_part
2.  imaginary_part
3.  define operator +
4.  define operator –
5.  define operator *
6.  define operator /
7.  IsEqual(other)
8.      if (real_part == other.real_part &&
    imaginary_part == other.imaginary_part)
9.              return true
10.     else return false
```

Gambar 3.7 Pseudocode Struktur Data Complex

Struktur data lain yang dibuat adalah struktur data untuk matriks persegi dengan elemen bertipe bilangan kompleks. Struktur data ini berisi informasi ukuran matriks, elemen-elemen pada matriks, serta sebuah fungsi untuk menghitung determinan matriks yang sesuai dengan penjelasan pada subbab 3.2.4. Pseudocode struktur data ini ditunjukkan dalam **Gambar 3.8**.

```
ComplexMatrix
1.  size
2.  element
3.  Determinant(size, element)
```

Gambar 3.8 Pseudocode Struktur Data ComplexMatrix

BAB IV
IMPLEMENTASI

Pada bab ini penulis menjelaskan tentang implementasi berdasarkan desain algoritma serta struktur data yang telah dilakukan.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang digunakan sebagai berikut:

1. Perangkat Keras
   Processor Intel® Core™ i5-4200U CPU @ 1.60GHz
   RAM 8.00 GB
2. Perangkat Lunak
   Sistem Operasi Windows 8.1
   Integrated Development Environment Code::Blocks 13.12

4.2 Implementasi Struct Complex

Struct Complex digunakan untuk merepresentasikan bilangan kompleks. Implementasi struct Complex ditunjukkan dalam **Kode Sumber 4.1**, **Kode Sumber 4.2**, **Kode Sumber 4.3**, **Kode Sumber 4.4**, **Kode Sumber 4.5**, **Kode Sumber 4.6**, dan **Kode Sumber 4.7**.

Terdapat dua variabel dalam struct Complex seperti yang ditunjukkan dalam **Kode Sumber 4.1**, yaitu *real* yang menyimpan bagian real dari bilangan kompleks serta *imag* yang menyimpan bagian imajiner dari bilangan kompleks.

```
1.  struct Complex
2.  {
3.      // struct Complex menyimpan nilai bagian real dan
    imajiner pada sebuah bilangan kompleks
4.     double real, imag;
```

Kode Sumber 4.1 Variabel pada Struct Complex

Terdapat juga dua macam konstruktor untuk struct Complex seperti yang ditunjukkan pada **Kode Sumber 4.2**.

Konstruktor pertama tidak memiliki parameter dan memberi nilai awal 0 untuk variabel *real* dan *imag*. Konstruktor kedua memiliki dua buah parameter yang masing-masing akan menjadi nilai dari variabel *real* dan *imag*.

```
5.      // constructor default untuk menginisialisasi
    bilangan kompleks dengan nilai 0
6.      Complex()
7.      {
8.          real = imag = 0.0;
9.      }
10.     // constructor berparameter untuk
    menginisialisasi bilangan kompleks dengan nilai yang
    diinginkan
11.     Complex(const double& a, const double& b)
12.     {
13.         real = a;
14.         imag = b;
15.     }
```

Kode Sumber 4.2 Constructor pada Struct Complex

Sebagian besar isi struct Complex merupakan *operator overloading* yang mendefinisikan operasi-operasi aritmetika dasar dan logika. **Kode Sumber 4.3** menunjukkan *operator overloading* untuk operasi perkalian. Terdapat 3 macam operasi perkalian pada struct Complex, yaitu perkalian dengan variabel bertipe Complex menggunakan operator "*", perkalian dengan variabel bertipe Complex menggunakan operator "*=", serta perkalian dengan variabel bertipe double menggunakan operator "*=".

Kode Sumber 4.4 menunjukkan dua macam *operator overloading* untuk operasi pembagian, yaitu pembagian dengan variabel bertipe Complex menggunakan operator "/" dan pembagian dengan variabel bertipe Complex menggunakan operator "/=".

Kode Sumber 4.5 menunjukkan penjumlahan dengan variabel bertipe Complex menggunakan operator "+=", sementara **Kode Sumber 4.6** menunjukkan pengurangan dengan variabel bertipe Complex menggunakan operator "-=".

`

Selain untuk operasi-operasi aritmetika dasar, terdapat juga *operator overloading* untuk mengecek kesamaan dengan variabel Complex lain, menggunakan operator "==" serta *operator overloading* untuk melakukan pemberian nilai menggunakan operator "=" seperti yang ditunjukkan pada **Kode Sumber 4.7**.

Terdapat juga *operator overloading* yang melibatkan struct Complex yang terdapat di luar definisi struct Complex, yaitu definisi operasi pembagian variabel bertipe double oleh variabel bertipe Complex. Implementasinya terdapat pada **Kode Sumber 4.8**.

```
16.     // definisi operasi perkalian dua buah bilangan
    kompleks
17.    Complex operator*(const Complex& r) const
18.    {
19.        return Complex(real*r.real-imag*r.imag,
    real*r.imag+imag*r.real);
20.    }
21.    // definisi operasi perkalian dua buah pada
    bilangan kompleks dengan operator compound
    assignment
22.    Complex& operator*=(const Complex& r)
23.    {
24.        double re = real*r.real-imag*r.imag;
25.        double im = real*r.imag+imag*r.real;
26.        real = re;
27.        imag = im;
28.        return *this;
29.    }
30.    // definisi operasi perkalian sebuah bilangan
    kompleks dengan bilangan desimal
31.    Complex& operator*=(const double& r)
32.    {
33.        real *= r;
34.        imag *= r;
35.        return *this;
36.    }
```

Kode Sumber 4.3 Operasi Perkalian pada Struct Complex

```
37.    // definisi operasi pembagian bilangan kompleks
    dengan bilangan kompleks lain
38.    Complex operator/(const Complex& r) const
39.    {
40.        double divisor = r.real*r.real+r.imag*r.imag;
41.        return
    Complex((real*r.real+imag*r.imag)/divisor,
    (imag*r.real-real*r.imag)/divisor);
42.    }
43.    // definisi operasi pembagian bilangan kompleks
    dengan bilangan kompleks lain dengan operator
    compound assignment
44.    Complex& operator/=(const Complex& r)
45.    {
46.        double divisor = r.real*r.real+r.imag*r.imag;
47.        double re =
    (real*r.real+imag*r.imag)/divisor;
48.        double im = (imag*r.real-
    real*r.imag)/divisor;
49.        real = re;
50.        imag = im;
51.        return *this;
52.    }
```

Kode Sumber 4.4 Operasi Pembagian pada Struct Complex

```
53.    // definisi operasi penjumlahan dua buah bilangan
    kompleks dengan operator compound assignment
54.    Complex& operator+=(const Complex& r)
55.    {
56.        real += r.real;
57.        imag += r.imag;
58.        return *this;
59.    }
```

Kode Sumber 4.5 Operasi Penjumlahan pada Struct Complex

4.3 Implementasi Struct ComplexMatrix

Struct ComplexMatrix digunakan untuk merepresentasikan matriks persegi yang memiliki elemen bertipe bilangan kompleks. Struct ini berisi variabel *size* yang menyimpan ukuran matriks, array dua dimensi *element* yang menyimpan elemen-elemen matriks, serta fungsi *Determinant* untuk

`

menghitung determinan dari matriks yang diimplementasikan sesuai dengan desain pada subbab 3.2.4. Implementasi struct ComplexMatrix ditunjukkan dalam **Kode Sumber 4.9**

Agar lebih efisien, proses dekomposisi LU juga dilakukan dalam fungsi *Determinant* yaitu pada baris 13 hingga baris 22, sehingga nilai determinan yang akan dikembalikan dapat dihitung di tengah-tengah proses dekomposisi LU yang ditunjukkan pada baris 15.

```
60.     // definisi operasi pengurangan bilangan kompleks
    dengan operator compound assignment
61.     Complex& operator-=(const Complex& r)
62.     {
63.         real -= r.real;
64.         imag -= r.imag;
65.         return *this;
66.     }
```

Kode Sumber 4.6 Operasi Pengurangan pada Struct Complex

```
67.     // definisi perbandingan kesamaan dua buah
    bilangan kompleks
68.     bool operator==(const Complex& r) const
69.     {
70.         double dr = real-r.real;
71.         double di = imag-r.imag;
72.         // EPS merupakan nilai yang sangat kecil
    (10^-6), digunakan untuk menghindari permasalahan
    teknis yang disebabkan oleh ketelitian tipe data
73.         return dr < EPS && dr > -EPS && di < EPS &&
    di > -EPS;
74.     }
75.     // definisi assignment nilai bilangan kompleks
    dari bilangan kompleks lain
76.    Complex& operator=(const Complex& r)
77.     {
78.         real = r.real;
79.         imag = r.imag;
80.         return *this;
81.     }
82. };
```

Kode Sumber 4.7 Operasi Lain pada Struct Complex

```
1.  // definisi pembagian bilangan desimal dengan
    bilangan kompleks
2.  Complex operator/(const double& l, const Complex& r)
3.  {
4.      double divisor = r.real*r.real+r.imag*r.imag;
5.      return Complex(l*r.real/divisor, -
    l*r.imag/divisor);
6.  }
```

Kode Sumber 4.8 Operasi Pembagian Double dengan Struct Complex

4.4 Implementasi Fungsi UpdateEquations

Fungsi UpdateEquations diimplementasikan sesuai dengan desain pada subbab 3.2.1. Implementasi fungsi ini ditunjukkan dalam **Kode Sumber 4.10**.

Variabel *equations_coeffs* merupakan variabel global yang menyimpan koefisien-koefisien pada *Systematic Nodal Equations*, dengan *equations_coeffs[i][j]* menandakan koefisien dari v_j pada persamaan untuk node *i*. Nilai 1/*impendance* disimpan dalam variabel sementara agar penghitungan tidak dilakukan secara berulang-ulang.

4.5 Implementasi Fungsi UpdateNodeGroup

Fungsi UpdateNodeGroup diimplementasikan sesuai dengan desain pada subbab 3.2.2. Implementasi fungsi ini ditunjukkan dalam **Kode Sumber 4.11**.

Variabel *num_nodes* dan *node_group* merupakan variabel global. *num_nodes* menyimpan banyak node dalam graph/jaringan resistor, sedangkan *node_group* merupakan array 1 dimensi di mana *node_group[i]* menyimpan nomor kelompok dari node *i*.

Fungsi UpdateNodeGroup melakukan iterasi untuk setiap node yang belum memiliki kelompok, kemudian memberi nomor kelompok yang sama untuk node tersebut dan node-node lain yang terhubung. Dalam mencari node-node yang terhubung dibutuhkan fungsi pembantu UpdateNodeGroupDFS yang ditunjukkan dalam **Kode Sumber 4.12**.

`

```
1.  struct ComplexMatrix
2.  {
3.       // dimensi matriks persegi
4.       int size;
5.       // nilai elemen-elemen dalam matriks
6.       Complex element[MAXN][MAXN];
7.       // fungsi untuk menghitung nilai determinan
    dalam dari matriks menggunakan dekomposisi LU
8.       Complex Determinant()
9.       {
10.          Complex lu[size][size];
11.          Complex det(1.0, 0.0);
12.          Complex temp;
13.          // elemen matriks disalin ke variabel lain
    untuk didekomposisi agar nilainya tidak berubah
14.          for (int i=0; i<size; ++i)
15.              for (int j=0; j<size; ++j)
16.                  lu[i][j] = element[i][j];
17.          // dekomposisi LU. Nilai determinan dihitung
    juga dalam proses ini dan disimpan pada variabel det
18.          for (int p=0; p<size; ++p)
19.          {
20.              det *= lu[p][p];
21.              for (int i=p+1; i<size; ++i)
22.              {
23.                  temp = lu[i][p] /= lu[p][p];
24.                  for (int j=p+1; j<size; ++j)
25.                      lu[i][j] -= temp * lu[p][j];
26.              }
27.          }
28.          // kembalikan nilai variabel det sebagai
    nilai determinan
29.          return det;
30.      }
31. };
```

Kode Sumber 4.9 Implementasi Struct ComplexMatrix

Variabel *is_connected* merupakan variabel global berbentuk array 2 dimensi dengan tipe data boolean yang menyimpan informasi adanya edge pada 2 node. Jika *is_connected[i][j]* bernilai true, maka terdapat edge yang

menghubungkan node *i* dan node *j*, sedangkan jika bernilai false maka tidak terdapat edge yang menghubungkan node *i* dan node *j*.

```
1.  inline void UpdateEquations(const int& node1, const
    int& node2, const Complex& impedance)
2.  {
3.      Complex temp_complex = 1.0/impedance;
4.
5.      // menambah koefisien pada variabel v[node1]
    pada persamaan di node1
6.      equations_coeffs[node1][node1] += temp_complex;
7.      // menambah koefisien pada variabel v[node2]
    pada persamaan di node2
8.      equations_coeffs[node2][node2] += temp_complex;
9.      // mengurangi koefisien pada variabel v[node2]
    pada persamaan di node1
10.     equations_coeffs[node1][node2] -= temp_complex;
11.     // mengurangi koefisien pada variabel v[node1]
    pada persamaan di node2
12.     equations_coeffs[node2][node1] -= temp_complex;
13. }
```

Kode Sumber 4.10 Implementasi Fungsi UpdateEquations

```
1.  void UpdateNodeGroup()
2.  {
3.      // untuk setiap node,
4.      for (int i=0; i<num_nodes; i++)
5.          // jika node tersebut belum memiliki nomor
    kelompok,
6.          if (node_group[i] == -1)
7.          {
8.              // beri nomor kelompok, lalu
9.              node_group[i] = i;
10.             // beri nomor kelompok yang sama untuk
    semua node lain yang terhubung dengannya
11.             UpdateNodeGroupDFS(i, i);
12.         }
13. }
```

Kode Sumber 4.11 Implementasi Fungsi UpdateNodeGroup

`

```
1.  // parameter group merupakan nomor kelompok yang
    akan diberikan, dan parameter node merupakan node
    yang sedang dikunjungi pada DFS
2.  void UpdateNodeGroupDFS(const int& group, const int&
    node)
3.  {
4.      for (int i=0; i<num_nodes; i++)
5.          // untuk node lain yang terhubung ke node
    saat ini dan belum memiliki nomor kelompok,
6.          if (node_group[i] == -1 &&
    is_connected[node][i])
7.          {
8.              // beri nomor kelompok pada node tersebut
    dan node-node lain yang terhubung.
9.              node_group[i] = group;
10.             UpdateNodeGroupDFS(group, i);
11.         }
12. }
```

Kode Sumber 4.12 Implementasi Fungsi UpdateNodeGruopDFS

Untuk setiap node yang belum memiliki kelompok, fungsi UpdateNodeGroupDFS melakukan *depth-first search* pada graph dimulai dari node tersebut untuk mencari node-node lain yang terhubung, lalu memberi nomor kelompok yang sama pada node-node tersebut dengan node awal. Parameter *group* menunjukkan nomor kelompok dari node awal, sementara parameter *node* menunjukkan node yang dikunjungi saat ini pada proses *depth-first search*.

4.6 Implementasi Fungsi EquivalentImpedance

Fungsi EquivalentImpedance diimplementasikan sesuai dengan desain pada subbab 3.2.3. Implementasi fungsi ini ditunjukkan dalam **Kode Sumber 4.13**.

Variabel *connected_nodes_coeffs* digunakan untuk menyimpan koefisien-koefisien pada *Systematic Nodal Equations* untuk node-node yang terhubung dengan node *node1* (kecuali node *node2* yang menjadi node acuan). Variabel *connected_nodes* digunakan untuk menyimpan node-node yang terhubung dengan node *node1* (kecuali node *node2*). Variabel *node1_index*

digunakan menyimpan index dari node *node1* dalam array *connected_nodes*.

Pada baris 6 hingga 12 dilakukan pencarian node-node yang terhubung dengan node *node1*, kemudian ditampung pada *connected_nodes*. Pada baris 13 hingga 15, *connected_nodes-_coeffs* diisi dengan menggunakan nilai pada variabel global *equations_coeffs*. Pada baris 17 hingga 19 dilakukan penggantian nilai kolom ke *node1_index* pada *connected_nodes_coeffs* dengan vektor bagian kanan dari *Systematic Nodal Equations* untuk node-node yang ada pada *connected_nodes*. Baris 16 menghitung nilai determinan dari *connected_nodes_coeffs* sebelum kolomnya diubah, dan baris 20 menghitung nilai determinannya setelah kolomnya diubah, kemudian menggunakan kedua nilai tersebut untuk mengembalikan nilai impedansi ekuivalen antara node *node1* dan node *node2*.

4.7 Implementasi Fungsi Main

Fungsi Main diimplementasikan sesuai dengan desain pada subbab 3.1. Implementasi fungsi ini ditunjukkan dalam **Kode Sumber 4.14** dan **Kode Sumber 4.15**.

Karena array dalam bahasa pemrograman C++ memiliki nilai index terkecil 0 sementara permasalahan *Electrical Engineering* pada SPOJ menggunakan 1 sebagai nilai node terkecil maka semua masukan mengenai node akan dikurangi nilainya dengan 1 untuk menyesuaikan, seperti yang ditunjukkan pada baris 14-15 serta baris 25-26.

Baris 8 hingga 10 melakukan *reset* terhadap beberapa variabel global agar siap digunakan untuk kasus uji selanjutnya. Baris 16 melakukan pengecekan untuk mengabaikan edge yang kedua ujungnya berada pada node yang sama karena tidak akan berpengaruh pada penghitungan impedansi ekuivalen.

Baris 27 melakukan pengecekan terhadap 2 node yang ingin dicari impedansi ekuivalennya. Jika kedua node tersebut sama, maka sistem akan mengeluarkan “0.00 0.00” sebagai nilai impedansi ekuivalennya tanpa melalui proses penghitungan terlebih dahulu.

`

```
1.  // mencari impedansi ekuivalen antara node1 dan
    node2
2.  Complex EquivalentImpedance(const int& node1, const
    int& node2)
3.  {
4.       // matriks yang menyimpan koefisien variabel
    tegangan dari node-node yang terhubung dengan node1
    dan node2
5.       ComplexMatrix connected_nodes_coeffs;
6.       // connected_nodes menyimpan node-node yang
    terhubung, node1_index menyimpan index dari node1
    pada connected_nodes
7.       int connected_nodes[MAXN], node1_index;
8.       connected_nodes_coeffs.size = 0;
9.       // mengisi connected_nodes
10.      for(int i=0; i<num_nodes; i++)
11.          if (i != node2 && node_group[i] ==
    node_group[node1])
12.          {
13.              if (i == node1)
14.                  node1_index =
    connected_nodes_coeffs.size;
15.
    connected_nodes[connected_nodes_coeffs.size++] = i;
16.          }
17.      // mengisi connected_nodes_coeffs berdasarkan
    informasi dari connected_nodes
18.      for (int i=0; i<connected_nodes_coeffs.size;
    i++)
19.          for (int j=0; j<connected_nodes_coeffs.size;
    j++)
20.              connected_nodes_coeffs.element[i][j] =
    equations_coeffs[connected_nodes[i]][connected_nodes
    [j]];
21.      // menghitung nilai determinan dari
    connected_nodes_coeffs saat ini
22.      Complex retval =
    connected_nodes_coeffs.determinant();
```

Kode Sumber 4.13 Implementasi Fungsi EquivalentImpedance (1)

```
23.     // mensubtisuti kolom ke node1_index pada
    connected_nodes_coeffs dengan bagian kanan dari
    systematic nodal equations, sesuai dengan aturan
    Cramer
24.     for (int i=0; i<connected_nodes_coeffs.size;
    i++)
25.
       connected_nodes_coeffs.element[i][node1_index] =
    kComplexZero;
26.    connected_nodes_coeffs.element[node1_index][node1
    _index] = Complex(1.0, 0.0);
27.     // mengembalikan nilai impedansi ekuivalen,
    yaitu hasil bagi determinan connected_nodes_coeffs
    yang sudah disubtitusi kolom dengan determinan
    awalnya, sesuai dengan aturan Cramer
28.     return
    connected_nodes_coeffs.determinant()/retval;
29. }
```

Kode Sumber 4.14 Implementasi Fungsi EquivalentImpedance (2)

```
1.  int main()
2.  {
3.      int node1, node2, i;
4.      double real, imag;
5.      Complex result;
6.      // baca masukan jumlah node, jumlah edge, dan
    jumlah penghitungan impedansi ekuivalen. Keluar dari
    program jika ketiganya bernilai 0
7.      while(scanf("%d%d%d", &num_nodes, &num_edges,
    &num_queries), (num_nodes||num_edges||num_queries))
8.      {
9.          // reset nilai variabel-variabel yang akan
    digunakan
10.         memset(node_group, -1, sizeof(node_group));
11.         memset(equations_coeffs, 0,
    sizeof(equations_coeffs));
12.         memset(is_connected, 0,
    sizeof(is_connected));
13.         // baca masukan deskripsi edge
14.         for(i=0; i<num_edges; ++i)
```

Kode Sumber 4.15 Kode Sumber Fungsi Main (1)

`

```
15.         {
16.             scanf("%d%d%lf%lf", &node1, &node2,
    &real, &imag);
17.             --node1;
18.             --node2;
19.              // abaikan edge yang loop
20.             if (node1 == node2)
21.                 continue;
22.              // tandai bahwa kedua node yang
    dihubungkan edge sudah terhubung
23.             is_connected[node1][node2] =
    is_connected[node2][node1] = 1;
24.              // update koefisien pada systematic
    nodal equation berdasarkan edge yang baru saja
    dimasukkan
25.             UpdateEquations(node1, node2,
    Complex(real, imag));
26.         }
27.          // update nomor kelompok dari setiap node
28.         UpdateNodeGroup();
29.         for(i=0; i<num_queries; ++i)
30.         {
31.              // baca masukan pasangan node yang akan
    dicari impedansi ekuivalennya
32.             scanf("%d%d", &node1, &node2);
33.             --node1;
34.             --node2;
35.              // jika kedua node sama, nilai impedansi
    ekuivalen antara kedua node tersebut adalah 0
36.             if (node1 == node2)
37.                 printf("0.00 0.00\n");
38.              // jika kedua node tidak terhubung,
    keluarkan “no connection”
39.             else if (node_group[node1] !=
    node_group[node2])
40.                 printf("no connection\n");
41.              // jika kedua node terhubung, hitung dan
    keluarkan nilai impedansi ekuivalennya
42.             else
43.             {
```

Kode Sumber 4.16 Kode Sumber Fungsi Main (2)

```
                result = EquivalentImpedance(node1,
    node2);
                printf("%.2lf %.2lf\n", result.real,
    result.imag);
            }
        }
        printf("\n");
    }
}
```

Kode Sumber 4.17 Kode Sumber Fungsi Main (3)

`

BAB V
UJI COBA DAN EVALUASI

Pada bab ini penulis menjelaskan tentang uji coba dan evaluasi dari implementasi yang telah dilakukan

5.1 Lingkungan Uji Coba

Lingkungan implementasi yang digunakan sebagai berikut:

1. Perangkat Keras
   Processor Intel® Core™ i5-4200U CPU @ 1.60GHz
   RAM 8.00 GB
2. Perangkat Lunak
   Sistem Operasi Windows 8.1
   Integrated Development Environment Code::Blocks 13.12

5.2 Skenario Uji Coba

Pada subbab ini akan dijelaskan skenario uji coba yang akan dilakukan. Skenario uji coba terdiri dari uji coba kebenaran dan uji coba kinerja.

5.2.1 Uji Coba Kebenaran

Uji coba kebenaran dilakukan dengan mengirimkan kode sumber implementasi yang telah dilakukan ke situs *Online Judge* SPOJ. Permasalahan yang akan diselesaikan adalah *Electrical Engineering* seperti yang telah dijelaskan pada subbab 2.2. Setelah kode sumber implementasi dikirimkan ke situs SPOJ, dapat dilihat umpan balik sistem pada situs SPOJ seperti yang dijelaskan pada subbab 2.1. Hasil uji coba pada situs SPOJ ditunjukkan dalam **Gambar 5.1**.

Gambar 5.1 Hasil Uji Coba pada Situs SPOJ

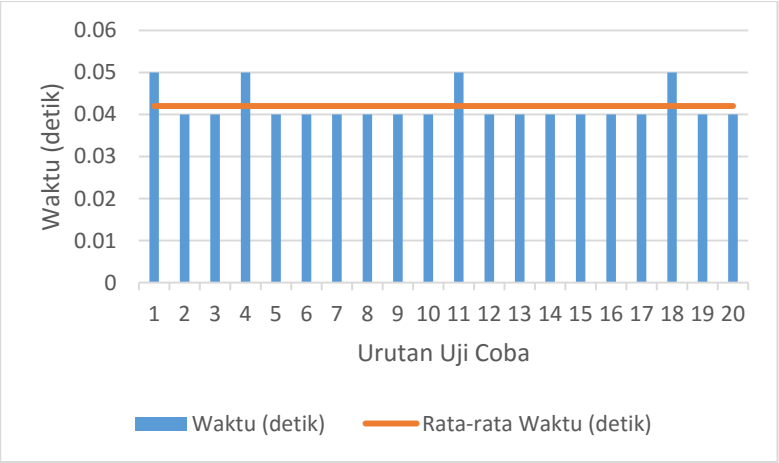
Dari hasil uji coba yang telah dilakukan, kode sumber program yang dikirimkan mendapat umpan balik ***Accepted***. Waktu

yang dibutuhkan program adalah 0.04 detik dan memori yang dibutuhkan program adalah 3,0 MB. Hal tersebut membuktikan bahwa implementasi algoritma komputasi matriks pada *Systematic Nodal Equations* yang telah dilakukan berhasil menyelesaikan permasalahan pada studi kasus *Electrical Engineering* dengan benar.

Setelah itu dilakukan pengiriman kode sumber implementasi sebanyak 20 kali untuk melihat variasi waktu dan memori yang dibutuhkan program. Hasil uji coba pada situs SPOJ sebanyak 20 kali ditunjukkan dalam **Tabel 5.1**, **Gambar 5.2**, dan **Gambar A.1**.

Tabel 5.1 Hasil Uji Coba pada Situs SPOJ Sebanyak 20 Kali

No	Hasil	Waktu (detik)	Memori (MB)
1	Accepted	0.05	3.0
2	Accepted	0.04	3.0
3	Accepted	0.04	3.0
4	Accepted	0.05	3.0
5	Accepted	0.04	3.0
6	Accepted	0.04	3.0
7	Accepted	0.04	3.0
8	Accepted	0.04	3.0
9	Accepted	0.04	3.0
10	Accepted	0.04	3.0
11	Accepted	0.05	3.0
12	Accepted	0.04	3.0
13	Accepted	0.04	3.0
14	Accepted	0.04	3.0
15	Accepted	0.04	3.0
16	Accepted	0.04	3.0
17	Accepted	0.04	3.0
18	Accepted	0.05	3.0
19	Accepted	0.04	3.0
20	Accepted	0.04	3.0



Gambar 5.2 Grafik Hasil Uji Coba pada Situs SPOJ Sebanyak 20 Kali

Dari hasil uji coba yang telah dilakukan, seluruh kode sumber program yang dikirimkan mendapat umpan balik ***Accepted***. Waktu yang dibutuhkan program minimum 0,04 detik, maksimum 0,05 detik dan rata-rata 0,042 detik. Memori yang dibutuhkan sebesar 3,0 MB untuk semua percobaan.

5.2.2 Uji Coba Kinerja

Untuk graph/jaringan resistor dengan n node , algoritma dekomposisi LU memiliki kompleksitas waktu $O(n^3)$ untuk setiap penghitungan determinan matriks dalam proses penghitungan impedansi ekuivalen. Uji coba kinerja dilakukan dengan membuat beberapa graph acak dengan banyak node yang bervariasi. Banyak kasus uji, banyak edge dan banyak pasangan node yang akan dihitung impedansi ekuivalennya dibuat tetap karena tidak memiliki pengaruh yang signifikan terhadap kompleksitas waktu dan agar pengujian dapat menunjukkan pengaruh banyak node terhadap waktu yang dibutuhkan program.

Dalam pembuatan setiap deskripsi edge, dua node yang dihubungkan oleh edge tersebut dipilih secara acak dari node-node

yang ada. Nilai bagian real dari impedansi dibangkitkan secara acak pada rentang [0.001..1000] dengan ketelitian 3 angka di belakang koma, sedangkan nilai bagian imajiner dari impedansi juga dibangkitkan secara acak pada rentang ±[0.001..1000] dengan ketelitian 3 angka di belakang koma. Setiap pasangan node yang akan dihitung impedansi ekuivalennya dipilih secara acak dari node-node yang tersedia.

5.2.2.1 Pengaruh Banyak Node Terhadap Waktu

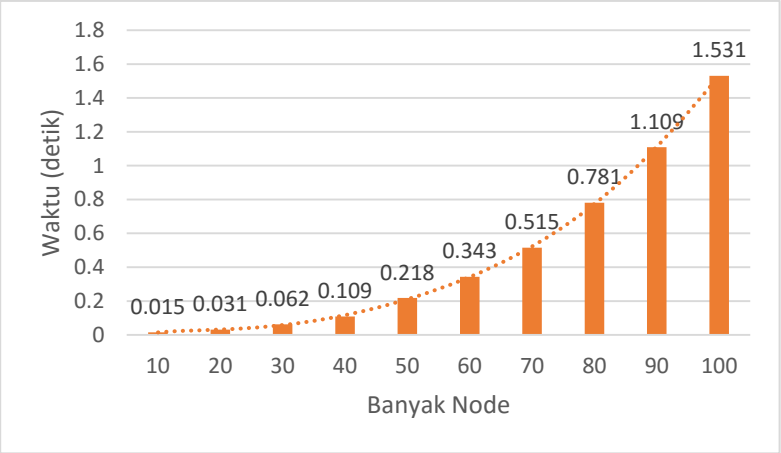
Banyak kasus uji dibuat tetap yaitu 10. Banyak node dibuat bervariasi antara 10 hingga 100 dengan interval nilai 10. Banyak edge dibuat tetap yaitu 1000. Banyak pasangan node yang ingin dicari impedansi ekuivalennya dibuat tetap yaitu 10. Kemudian dicatat waktu yang dibutuhkan sistem untuk setiap 10 kasus uji yang telah dibuat. Hasil uji coba ditunjukkan dalam **Tabel 5.2** dan **Gambar 5.3**.

Tabel 5.2 Hasil Uji Coba Pengaruh Banyak Node Terhadap Waktu

No	Banyak Node	Waktu (detik)
1	10	0.015
2	20	0.031
3	30	0.062
4	40	0.109
5	50	0.218
6	60	0.343
7	70	0.515
8	80	0.781
9	90	1.109
10	100	1.531

Dari hasil uji coba yang telah dilakukan, grafik pertumbuhan waktu yang dibutuhkan program mendekati kurva polinomial berderajat 3 seiring dengan pertumbuhan banyak node. Hal tersebut membuktikan bahwa implementasi yang telah

dilakukan sesuai dengan kompleksitas waktu dari algoritma dekomposisi LU yang dipengaruhi banyak node secara kubik.



Gambar 5.3 Grafik Hasil Uji Coba Pengaruh Banyak Node Terhadap Waktu

LAMPIRAN A

muhammad imaduddin: submissions

< Previous 1 2 3 4 5 Next >

ID		DATE	PROBLEM	RESULT	TIME	MEM	LANG
14413234		2015-06-08 04:18:30	Electrical Engineering	accepted edit ideone it	0.05	3.0M	C++ 4.3.2
14413233		2015-06-08 04:18:27	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413232		2015-06-08 04:18:25	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413231		2015-06-08 04:18:22	Electrical Engineering	accepted edit ideone it	0.05	3.0M	C++ 4.3.2
14413230		2015-06-08 04:18:19	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413229		2015-06-08 04:18:16	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413228		2015-06-08 04:18:14	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413227		2015-06-08 04:18:12	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413225		2015-06-08 04:18:07	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413224		2015-06-08 04:18:07	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413223		2015-06-08 04:18:04	Electrical Engineering	accepted edit ideone it	0.05	3.0M	C++ 4.3.2
14413221		2015-06-08 04:18:02	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413220		2015-06-08 04:17:59	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413218		2015-06-08 04:17:55	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413217		2015-06-08 04:17:54	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413216		2015-06-08 04:17:52	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413215		2015-06-08 04:17:49	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413214		2015-06-08 04:17:46	Electrical Engineering	accepted edit ideone it	0.05	3.0M	C++ 4.3.2
14413213		2015-06-08 04:17:42	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2
14413212		2015-06-08 04:17:39	Electrical Engineering	accepted edit ideone it	0.04	3.0M	C++ 4.3.2

Gambar A.1 Hasil Uji Coba pada Situs SPOJ Sebanyak 20 Kali

DAFTAR PUSTAKA

[1] Hostetter, Gene H. 1984. Engineering Network Analysis.

[2] Jensen, Randal W. dan Watkins, Bruce O. 1974. Network Analysis Theory and Computer Methods.

[3] Press, H. William, Saul A. Teukolsky, William T. Vetterling dan Brian P. Flannery. 2007. Numerical Recipes in C, The Art of Scientific Computing, Third Edition.

[4] "SPOJ.com – Problem ELEC," [Online]. Avaliable: http://www.spoj.com/problems/ELEC/.

BIODATA PENULIS

Muhammad Imaduddin, lahir di Surabaya pada 3 Oktober 1992, anak pertama dari 3 bersaudara. Penulis telah menempuh pendidikan formal mulai dari jenjang TK hingga S-1 di TK Dorowati Surabaya (1997-1999), SDN Manukan Kulon IV Surabaya (1999-2005), SMPN 1 Surabaya (2005-2008), SMAN 5 Surabaya (2008-2011), dan Jurusan Teknik Informatika Fakultas Teknologi Informasi (FTIF) Institut Teknologi Sepuluh Nopember (ITS) Surabaya (2011-2015). Pada tahun 2010 penulis pernah meraih medali perunggu pada Olimpiade Sains Nasional (OSN) tingkat SMA. Ketika masih menempuh perkuliahan di Jurusan Teknik Informatika FTIF-ITS, penulis beberapa kali menjadi finalis kompetisi pemrograman nasional yang diselenggarakan di Institut Teknologi Bandung (ITB), Universitas Indonesia (UI), Universitas Gadjah Mada (UGM) dan Bina Nusantara University (BINUS). Penulis juga pernah meraih Juara III pada kompetisi Gemastik Kategori Debugging (2013) serta Juara II pada kompetisi Gemastik Kategori Pemrograman (2014). Selain itu penulis juga pernah menjadi asisten mata kuliah Pemrograman Terstruktur (2012) di Jurusan Teknik Informatika FTIF-ITS.