Kompresi Pohon dengan Kode Prüfer

Yogi Salomo Mangontang Pratama(13511059)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*yogi.salomo@students.itb.ac.id*

***Abstrak*—Makalah ini membahas mengenai salah satu cara kompresi dalam pemrograman, yaitu dengan menggunakan Kode Prüfer. Prinsip dari Kode Prüfer ini adalah mereduksi jumlah dari simpul pada pohon sehingga ukurannya menjadi lebih kecil pada saat pengiriman. Kode Prüfer ini akan diimplementasikan dalam pohon yang merupakan bagian dari teori graf. Nantinya makalah ini akan membahas cara kerja dari Kode Prüfer, membuktikan keabsahannya, serta menyertakan algoritma dasarnya.**

Kata Kunci : Kompresi, Prüfer, Pohon, Graf.

I. PENDAHULUAN

Dalam pemrograman, terutama yang melibatkan jaringan, akan dibutuhkan banyak pengiriman data dari satu komputer ke komputer lainnya. Dengan banyaknya perpindahan data tersebut, maka seorang programmer dituntut untuk mencari metode yang paling efektif, untuk memperkecil ukuran dari data yang dikirimkan.

Begitu banyak cara kompresi yang telah ditemukan, seperti Huffman, DEFLATE, LZW (Lempel-Ziv-Welch), hingga LRZ(LZ-Renau). Kode Prüfer yang akan dibahas pada makalah ini adalah salah satu metode untuk kompresi, yang dikhususkan untuk memperkecil ukuran dari suatu data bertipe pohon.

Di sini akan dibahas metode untuk menyederhanakan struktur dari suatu pohon, terlepas dari isi pohon tersebut. Ibaratkan saja dalam salah satu jejaring sosial, *twitter*. Asumsikan kita ingin mengukur pengaruh sosialnya dengan menghitung jumlah *tweet* yang ditulis seseorang, yang kemudian di-*retweet* oleh *follower*-nya, yang kemudian di-*retweet* kembali, dan seterusnya. Isi dari *tweet*, nama pengguna, serta waktu *retweet*-nya dapat dikirimkan melalui pesan yang berbeda. Yang perlu diamati adalah memetakan struktur pohon untuk setiap simpul yang mewakili persebaran *tweet* tersebut, mulai dari nomor 1 sampai dengan ukuran pohonnya (N). Dan persoalannya adalah membuat suatu metode, sehingga pihak yang menerima pesan memahami bahwa suatu simpul adalah anak ataupun orangtua dari node lainnya, dan dapat membentuk kembali sebuah pohon yang identik.

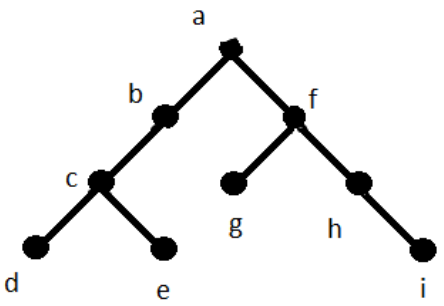
II. TEORI DASAR

II.I. Pohon

Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit. Sifat-sifat dari pohon :

Misalkan G = (V,E) adalah graf tak berarah sederhana dan jumlah simpulnya n, makasemua pernyataan di bawah ini adalah ekivalen :

1. G adalah pohon
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki m = n – 1 buah sisi.
4. G tidak mengandung sirkuit dan memiliki m = n – 1 buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan.



Gambar 2.1 Contoh Sebuah Pohon

Gambar di atas merupakan salah satu contoh pohon. Melalui contoh tersebut kita dapat mengamati berbagai terminologi yang perlu kita pahami sehubungan dengan pembahasan makalah ini antara lain :

- Anak
  Anak adalah simpul yang berasal dari suatu node lainnya.
  Contoh : b adalah anak dari a.
- Orangtua
  Orangtua adalah simpul yang menjadi sumber dari simpul lainnya.
  Contoh : f adalah orangtua dari g dan h.
- Lintasan
  Lintasan adalah simpul-simpul yang dilewati untuk menghubungkan satu simpul dengan

simpul yang lain.
Contoh : lintasan dari a ke I adalah a,f,h,i.

- Saudara kandung
  Saudara kandung adalah node yang memiliki Orangtua yang sama.
  Contoh : g dan h adalah saudara kandung.
- Upapohon
  Upapohon adalah pohon yang menjadi bagian dari suatu pohon. Dapat diperoleh dengan menghilangkan sebuah node dari pohon.
  Contoh : (b (c d e)) adalah upapohon dari Gambar 2.1
- Derajat
  Derajat dari suatu simpul adalah jumlah simpul yang berasal dari simpul tersebut.
  Contoh : derajat dari f adalah 2.
- Daun
  Daun adalah semua simpul yang berderajat nol.
  Contoh : daun d,e,g,i.

II.II Pohon Berlabel

Pohon berlabel adalah suatu pohon yang pada setiap simpulnya diberikan label berupa karakter yang unik.

Cayley(1889) menyatakan dalam rumusnya bahwa untuk n buah simpul, terdapat n^{n-2} buah pohon berlabel. Atau dalam persamaan adalah sebagai berikut :

$$\binom{n-2}{d1-1, d2-1, \ldots dn-1} = \frac{(n-2)!}{(d1-1)!\,(d2-1)!\ldots(dn-1)}$$

Dimana n adalah jumlah dari simpul, serta d1-dn adalah derajat dari masing-masing simpul tersebut.

II.III Kode Prüfer

Berikut adalah teori dari Heinz Prüfer yang isinya kurang lebih sebagai berikut.

II.III.I. Membuat Kode Prüfer dari sebuah Pohon

Diberikan sebuah pohon T dengan vertex set $\{v_1,\ldots, v_n\}$ dengan $v_1 < v_2 < v_n$. Ulangi proses berikut hingga tersisa satu simpul pada pohon :

1. Hapus daun dengan label paling kecil (proses ini akan menghasilkan upagraf dari pohon T)
2. Simpan nilai label dari orangtua daun yang dihapus, ke dalam suatu *string* atau array.

Hasil dari proses ini adalah sebuah *string* dengan panjang n – 2 dalam $\{v_1,\ldots, v_n\}$. *String* tersebut adalah Kode Prüfer dari pohon T tersebut. Pemetaan dari Pohon ke Kode Prüfer ini disebut injektif.

II.III.II Membuat Pohon dari Kode Prüfer

Setelah pohon berhasil dikompresi, dibutuhkan cara untuk meng *encode* kode tersebut sehingga dapat kembali dibaca sebagai pohon. Berikut adalah langkah-langkah yang harus dilakukan :

Diberikan sebuah *string* S dengan panjang n-2 dalam bentuk $\{v_1,\ldots, v_n\}$ dengan $v_1 < v_2 < \ldots < v_n$. Ulangi proses berikut hingga S kosong dan label berukuran dua :

1. Identifikasi nilai terendah yang tidak muncul pada *string*, sebut sebagai v_i, serta elemen pertama pada *string*, sebut sebagai v_j.
2. Tambahkan v_i pada graf yang akan dibentuk, kemudian hubungkan dengan v_j (apabila v_j belum terdapat dalam graf, tambahkan terlebih dahulu)
3. Hapus v_i serta elemen pertama dari *string* tersebut.

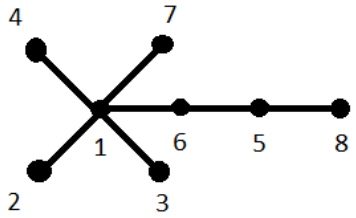
Setelah pengulangan selesai, hubungkan dua nilai yang tersisa pada *string*, maka didapatlah hasil sebuah pohon yang memiliki Kode Prüfer S. Pemetaan dari Kode Prüfer ke dalam pohon disebut bijektif.

III. PEMBAHASAN

A. Pengujian Kode Prüfer dalam Kompresi Pohon

Seperti yang telah dijelaskan pada bagian sebelumnya dari makalah ini, Kode Prüfer berfungsi untuk memperkecil ukuran dari data yang berupa pohon. Langkah-langkah yang harus dilakukan telah dijelaskan pada bagian Teori Dasar. Pada bagian ini, akan diuji apakah langkah-langkah yang telah dijelaskan terbukti dapat membuat suatu pohon menjadi lebih sederhana, dan apakah nantinya kode yang telah dibuat dapat dikembalikan menjadi pohon yang identik tanpa mengetahui bentuk pohon sebelumnya.

Sebagai contoh, apabila kita memiliki sebuah pohon dengan bentuk sebagai berikut :

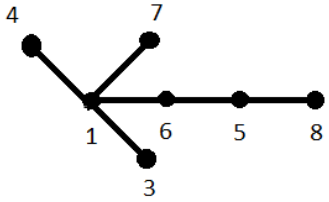


Gambar 3.1 Sebuah pohon berlabel

- Hal yang pertama harus kita lakukan adalah mencari daun dengan label nilai terkecil. Dalam hal ini adalah 2. Setelah itu kita menghapus daun tersebut, dan menambahkan nilai dari label orangtuanya ke dalam kode Prüfer.

  S = (1)

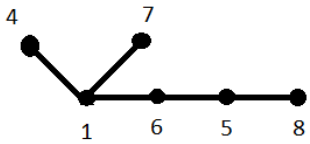
  Dimana S adalah sebuah *string* berisi Kode Prüfer.



Gambar 3.2 Bentuk pohon setelah proses pertama

- Setelah proses pertama, diperoleh pohon dengan bentuk demikian. Dan proses yang sama diulangi kembali, dan kini daun dengan nilai terkecil adalah 3, maka kita kembali menghapus daun tersebut dan menyimpan nilai dari orangtuanya ke dalam S.

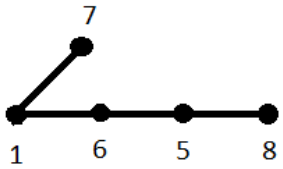
  S = (1, 1)



Gambar 3.3 Bentuk pohon setelah proses kedua

- Ulangi kembali proses yang sama, kali ini dengan daun berlabel 4 bernilai terendah, dan orangtuanya bernilai 1, simpan nilainya ke dalam S.

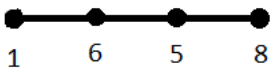
  S = (1, 1, 1)



Gambar 3.4 Bentuk pohon setelah proses ketiga

- Proses yang terakhir menghasilkan pohon seperti pada gambar di atas. Pohon ini hanya memiliki dua daun, dan daun dengan nilai terendah adalah 7, maka kita hapus daun tersebut dan simpan nilai orangtuanya.

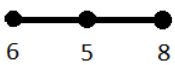
  S = (1, 1, 1, 1)



Gambar 3.5 Bentuk pohon setelah proses keempat

- Dapat kita amati pada gambar di atas, simpul 1 yang tadinya merupakan node dalam, kini menjadi daun, dan nilainya adalah yang terendah, maka kita hapus daun tersebut, dan tambahkan nilai orangtuanya dalam S.

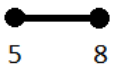
  S = (1, 1, 1, 1, 6)



Gambar 3.6 Bentuk pohon setelah proses kelima

- Kini daun dengan nilai terendah adalah 6. Maka kita hapus daun tersebut dan masukkan nilai orangtuanya ke dalam S.

  S = (1, 1, 1, 1, 6, 5)



Gambar 3.7 Bentuk pohon saat proses berhenti dilaksanakan

- Pada pohon yang kita miliki kini tersisa hanya dua simpul. Hal ini menandakan bahwa proses Encoding dari Kode Prüfer kita telah selesai, dengan Kode Prüfer dari pohon tersebut adalah :

  S = (1, 1, 1, 1, 6, 5)

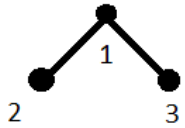
Setelah berhasil mengubah pohon tersebut menjadi Kode Prüfer, kita perlu memeriksa apakah kode tersebut dapat dikembalikan menjadi sebuah pohon yang identik, dengan asumsi kita tidak mengetahui terlebih dahulu bagaimana bentuk dari pohon tersebut.

- Kode Prüfer yang diberikan memiliki 6 masukan. Sesuai dengan teori, maka pohon yang akan dibangun dari kode ini akan memiliki 8 anggota, dengan label dari 1 hingga 8.
- Nilai terkecil yang tidak terdapat pada *string* S adalah 2, dan nilai pertama dari *string* tersebut adalah 1. Maka kita buat sebuah graf yang menghubungkan sebuah simpul berlabel 2 dengan simpul berlabel 1. Nilai 2 ditambahkan pada *string* S, sementara nilai pertama *string* tersebut dihapus.

Gambar 3.8 Bentuk pohon setelah proses pertama

S = (1, 1, 1, 6, 5, 2)

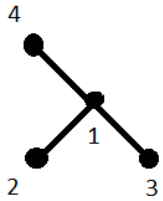
- Sekarang nilai terkecil yang tidak ada pada *string* adalah 3, sementara nilai pertama masih 1. Maka kita menggabungkan simpul 3 dengan simpul 1 yang telah terdapat pada graf. Kemudian menghapus nilai pertama dari *string* S, dan menambahkan nilai 3 di akhir *string* tersebut.



Gambar 3.9 Bentuk pohon setelah proses kedua

S = (1, 1, 6, 5, 2, 3)

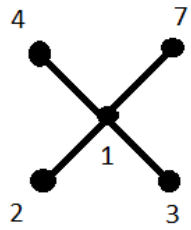
- Nilai terkecil yang tidak terdapat pada *string* tersebut setelah proses sebelumnya dilakukan adalah 4, dan nilai pertama dari *string* tersebut adalah 1. Maka buat simpul 4 dan hubungkan dengan simpul 1 pada graf. Hapus nilai 1 dan tambahkan nilai 4 pada *string* S.



Gambar 3.10 Bentuk pohon setelah proses ketiga

S = (1, 6, 5, 2, 3, 4)

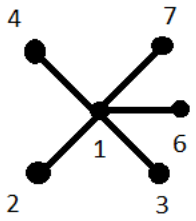
- Nilai awal dari *string* S masih tetap 1, akan tetapi sekarang nilai terkecil yang tidak terdapat pada *string* adalah 7. Maka buat simpul 7 dan hubungkan dengan simpul 1, serta modifikasi *string* S dengan menghapus nilai pertamanya, dan menambahkan 7 pada akhirnya.



Gambar 3.11 Bentuk pohon setelah proses keempat

S = (6, 5, 2, 3, 4, 7)

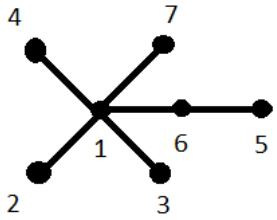
- Sekarang nilai paling pertama dari *string* S adalah 6, akan tetapi nilai terkecil yang tidak terdapat pada *string* tersebut adalah 1. Maka kita harus membuat simpul berlabel 6 dan menghubungkannya dengan simpul 1. Kemudian hapus nilai 6 dan tambahkan 1 pada *string* S.



Gambar 3.12 Bentuk pohon setelah proses kelima

S = (5, 2, 3, 4, 7, 1)

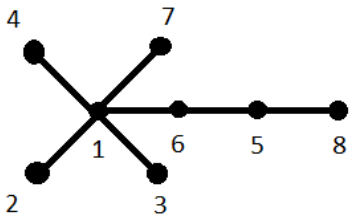
- Nilai terkecil yang belum terdapat pada *string* tersebut kini adalah 6, sementara nilai pertamanya adalah 5. Oleh karena itu, kita buat simpul 5 kemudian hubungkan dengan simpul 6 pada graf. Hapus nilai pertama pada *string* S, kemudian tambahkan nilai 6 pada *string* tersebut.



Gambar 3.13 Bentuk pohon setelah proses keenam

S = (2, 3, 4, 7, 1, 6)

- Dapat kita amati bahwa proses kembali berulang hingga angka yang pertama kali ditambahkan. Berikutnya kita mencari angka berapa yang hilang dari kumpulan angka ini. Karena pohon akan memiliki 8 masukan, maka angka-angka yang belum ada dalam *string* S adalah 5 dan 8. Buat simpul dari angka-angka tersebut, lalu hubungkan pada graf yang telah kita bentuk, sehingga menghasilkan :



Gambar 3.14 Bentuk akhir pohon setelah proses *decoding*

Dapat kita amati bahwa tanpa mengetahui bagaimana bentuk pohonnya, kita telah berhasil membentuk sebuah pohon yang strukturnya identik dengan pohon sebelumnya. Hal ini membuktikan bahwa Kode Prüfer ini sahih. Dan dapat dibandingkan *string* S dengan pohon T dari segi penulisannya :

S = 1 1 1 1 6 5
T = (1 (2) (3) (4) (7) (6 (5 (8))))

S memiliki penulisan yang lebih sederhana, yang akan mengakibatkan ukuran datanya dalam proses algoritma juga lebih sederhana dan ringan. Maka terbukti benar Kode Prüfer ini dapat digunakan untuk melakukan kompresi data yang berupa pohon.

B. Algoritma Sederhana untuk Kode Prüfer

Setelah terbukti bahwa kompresi dengan kode Prüfer adalah metode yang sahih, hal selanjutnya yang perlu dilakukan adalah membuat algoritma dari kompresi kode Prüfer ini sehingga dapat diterapkan dalam dunia pemrograman. Pada kesempatan ini bahasa yang digunakan adalah dalam notasi algoritmik.

```
function BuatKodePrüfer(T : Pohon) → Array of Integer

{ fungsi untuk membuat kode Prüfer dari sebuah pohon berlabel T }

Kamus Lokal

  i, x, y,n : integer

      P : integer[0..n-3]

function HitungNode (T : Pohon) → integer

{ fungsi untuk menghitung jumlah simpul dari sebuah pohon}

Procedure DeleteDaunTerkecil (input : T : Pohon, output : T : Pohon , y : integer)

{Prosedur untuk menghapus sebuah daun terkecil dari pohon, dan mengembalikan pohon yang telah dikeluarkan daunnya beserta integer y yang berisi nilai orangtua dari daun x yang dihapuskan

I.S. : Pohon tidak boleh kosong, x adalah daun dari pohon yang bersangkutan.

F.S. : Pohon yang baru akan berkurang daun yang berlabel x, dan y akan berisi nilai dari label orangtua daun x}

Algoritma

 n ← HitungNode(T)

  for i←0 to n-3 do

      DeleteDaunTerkecil(T, y)

      P[i] ← y


 → P
```

Demikianlah garis besar dari algoritma untuk membuat Kode Prüfer dari sebuah Pohon. Sementara untuk algoritma *decoding* dari Kode Prüfer tersebut menjadi pohon adalah sebagai berikut :

```
function DecodingPrüfer(P : Array of Integer) → Pohon

{ fungsi untuk membongkar sebuah Kode Prüfer menjadi sebuah Pohon }

Kamus Lokal

  i,m, n, x, y, a, b : integer

      T : Pohon

function IsAnggota (P : Array of Integer, x : integer) → boolean

{ fungsi untuk memeriksa apakah sebuah integer adalah anggota dari suatu array of integer}

Procedure DeleteFirstElmt (input : P : Array of Integer, output : P : Array of Integer, y : integer)

{Prosedur untuk menghapus elemen pertama dari sebuah array of integer, dan kemudian memasukkan nilai dari elemen pertama tersbeut ke sebuah integer y.

I.S. : Array tidak boleh kosong.

F.S. : Anggota pertama Array akan terhapus, dan terbentuk Array baru dengan anggota yang telah bergeser satu elemen ke depan, dan y berisi elemen pertama dari array}

Procedure AddToTree (input : T : Pohon, x, y : integer, output : T : Pohon)

{Prosedur untuk Menambahkan daun pada sebuah Pohon. Apabila kedua integer belum ada yang menjadi bagian dari pohon tersebut, maka akan dibuat pohon baru dengan simpul x dan y terhubung.

I.S. : Pohon boleh kosong.

F.S. : Pohon memiliki elemen tambahan x dan y, apabila salah satunya telah terdapat pada pohon, maka integer yang lain akan menjadi anak dari integer yang telah terdapat pada pohon tersebut}

function HitungAnggota (P : Array of Integer) → integer

{ fungsi untuk menghitung jumlah anggota dari sebuah array of integer}
```

Algoritma

```
  m  ← HitungAnggota(P)
     for i←0 to m-1 do
      x ← 1
      while IsAnggota(P,x) do
            x ← x + 1
      {x akan berisi integer terkecil
      yang tidak berada pada array}
      DeleteFirstElmt(P,y)
      AddToTree(T,x,y)
      n ← HitungAnggota(P)
      P[n] ← x
   a ← 1
      while IsAnggota(P,a) do
            a ← a + 1
b ← a + 1
      while IsAnggota(P,b) do
            b ← b + 1
AddToTree(T,a,b)


→ T
```

Demikianlah garis besar dari algoritma untuk mengubah Kode Prüfer kembali menjadi sebuah pohon. Adapun semua fungsi dan prosedur antaranya tidak disertakan realisasinya, akan tetapi manfaatnya telah dijelaskan pada definisi dan spesifikasinya.

C. Sifat-sifat dari Kode Prüfer dan pembuktiannya

Kode Prüfer memiliki beberapa karakteristik yang dapat dijadikan pedoman dalam pengaplikasiannya antara lain :

- Setiap vertex v_i akan muncul sejumlah 1 nilai lebih kecil daripada derajat vertex tersebut.

  Dapat dilihat dari Kode Prüfer contoh, dimana angka 1 muncul sebanyak 4 kali, dan kita lihat bahwa dalam pohon T, simpul 1 berderajat 5 (2,3,4,7,6). Berarti sifat ini terbukti benar.

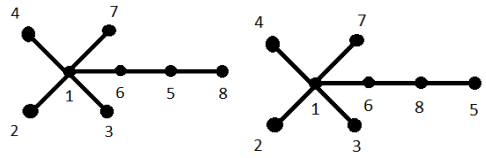
- Setelah daun pertama dihapus dan label orangtuanya disimpan, Kode Prüfer berikutnya sama persis dengan Kode Prüfer dari Upapohon T tanpa simpul vi.

  Misalkan kita menghapus daun 2 dari pohon T yang menjadi contoh, maka Kode Prüfer yang akan kita dapatkan adalah :

  S' = (1, 1, 1, 6, 5)

  Maka sifat inipun terbukti benar adanya.

- Jika daun dengan label terendah berbeda, maka Kode Prüfer dari dua pohon akan berbeda. Sifat ini telah dibuktikan dengan sifat sebelumnya.
- Jika daun dengan label terendah sama, akan tetapi posisi label dari simpul berbeda, akan menghasilkan Kode Prüfer yang berbeda.



Gambar 3.15 Dua Buah Pohon dengan label daun terkecil sama, akan tetapi susunan daunnya berbeda

Dapat dilihat bahwa kedua pohon di atas mirip, akan tetapi posisi label 5 dan 8 berbeda. Dengan menggunakan metode yang dijelaskan di atas, diperoleh Kode Prüfer untuk kedua Pohon :

S1 = (1, 1, 1, 1, 6, 5)

S2 = (1, 1, 1, 8, 1, 6)

Dapat kita amati bahwa Kode Prüfernya berbeda, berarti sifat ini terbukti benar

Ada banyak sifat dari Kode Prüfer yang dapat ditemukan, akan tetapi pada makalah ini pembahasan dibatasi hanya sampai sifat-sifat ini saja.

Demikianlah pembahasan mengenai Kode Prüfer mulai dari pembuktian kebenaran, algoritma, sampai dengan sifat-sifatnya.

V. SIMPULAN

Kode Prüfer merupakan salah satu metode kompresi yang unik, dimana kode ini dikhususkan untuk sebuah graf pohon. Meskipun tingkat kompresinya tidak terlalu besar, akan tetapi dapat dimanfaatkan dalam pemrograman untuk memperingan data yang akan dikirim.

VI. UCAPAN TERIMA KASIH

Pada kesempatan ini penulis ingin mengucapkan syukur kepada Tuhan Yang Maha Esa karena atas berkat-Nyalah makalah ini dapat diselesaikan. Penulis juga mengucapkan terima kasih kepada Bapak Rinaldi Munir, yang berkat pengajaran dan bimbingannya, penulis mendapat ilmu yang berguna bagi penyelesaian makalah ini. Tak lupa juga penulis mengucapkan terima kasih kepada semua pihak yang telah berperan serta dalam proses penyelesaian makalah ini.

VII. REFERENSI

[1] Wang, Xiaodong, dkk., *An Optimal Algorithm for Prufer Codes*, Atlanta : Georgia Institute of Technology, 2009.

[2] Munir, Rinaldi. 2012. Slide Perkuliahan IF2091. Bandung : Institut Teknologi Bandung.
[3] Kenneth H. Rosen, *Discrete Mathematics and Application to Computer Science 7th Edition*, Mc Graw-Hill, 2007. Hal. 745-803
[4] Priell, Nimrod, *On Compressing Trees with Prufer Codes*, URL : www.Educated-guess.com/2011/07/21/on-compressing-trees- with- prufer-codes/ diakses tanggal 16 Desember 2012 pukul 19.00 WIB
[5] "Prufer Code" URL : www.mathworld.wolfram.com/PruferCode.html diakses tanggal 16 Desember pukul 17.00 WIB
[6] "Prufer Codes" URL : www.nd.edu/~dgalvin1/40210/40210_F12/Prufer.pdf diakses tanggal 16 Desember 2012 pukul 19.00

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Desember 2012

Yogi Salomo Mangontang Pratama(13511059)