Penggunaan Prüfer Code untuk Menghasilkan Kasus Uji Berbentuk Pohon

Ahmad Romy Zahran - 13520009[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13520009@std.stei.itb.ac.id

***Abstrak*—Kasus uji berguna mengetes kebenaran desain dan implementasi algoritma. Kasus uji acak yang banyak dapat memeriksa lebih menyeluruh dan memastikan suatu program benar dengan lebih akurat. Makalah ini menampilkan program yang membuat kasus uji berbentuk pohon. Program memanfaatkan Prüfer code, barisan unik untuk suatu pohon berlabel yang menjamin sisi-sisi pada kasus uji membentuk pohon. Hasilnya program berjalan cepat dengan kompleksitas O(n) dan menghasilkan pohon yang bervariasi.**

***Keywords*—debug, kasus uji, pengujian, pohon, Prüfer code.**

I. PENDAHULUAN

Kasus uji atau *test* case banyak digunakan dalam kompetisi pemrograman kompetitif untuk menilai solusi peserta. Selain itu, kasus uji juga berguna dalam mengetes kebenaran desain dan implementasi algoritma. Setelah dites, kebenaran program tersebut menjadi semakin menyakinkan atau justru ditemukan *counter test case* yang membuktikan kesalahan atau *bug* dalam program itu. Kasus uji yang baik dapat membedakan solusi peserta yang benar dan salah, dan dapat sepasti mungkin memastikan suatu program itu benar. Untuk akurasi itu, dibutuhkan kasus uji dalam jumlah cukup banyak hingga puluhan atau ratusan. Kasus uji dalam pemrograman kompetitif non interaktif memiliki format tertentu sehingga dapat dibuat program yang menghasilkannya lalu eksekusinya diiterasi sekian kali. Program tersebut dapat ditujukan untuk kasus uji spesial atau kasus uji random. Kasus uji random dalam jumlah besar dapat digunakan untuk memeriksa lebih menyeluruh. Makalah ini akan menampilkan program (dibuat dalam bahasa C++) yang membuat kasus uji berbentuk pohon. Kasus uji yang dibuat berupa bilangan banyak simpul pohon, diikuti dengan sisi-sisinya berupa dua bilangan yang dihubungkan oleh sisi tersebut. Makalah ini memanfaatkan teori Prüfer code agar dijamin kasus uji yang dibuat berisi sisi-sisi yang membentuk pohon dan dapat melingkupi berbagai variasi pohon. Dalam makalah ini juga disertakan gambar graf yang dibuat dengan graf editor dari https://csacademy.com/app/graph_editor/.

II. LANDASAN TEORI

A. Pohon

Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit. Pohon dengan *n* simpul memiliki $n - 1$ sisi [4]. Derajat dari suatu simpul adalah banyak tetangganya (simpul yang terhubung langsung). Pohon berlabel artinya pohon tersebut diperhatikan nomor simpulnya , tidak hanya bentuk geometri saja. Daun adalah simpul dengan derajat 1.

B. Pengujian dengan Bash Script

Bila program yang ingin diuji adalah A dan solusi atau pembandingnya adalah sol_A, pengujian dapat dilakukan seperti Errichto dalam [2], yaitu dengan menjalankan bash s.sh dengan s.sh adalah bash script berikut.

```
$ s.sh
1   for((i=1; i<=100; i++)); {
2       echo $i
3       ./gen $i > intc
4       ./A < intc > out1
5       ./sol_A < intc > out2
6       diff -w out1 out2 || break
7   }
```

Gambar 1. Cuplikan bash script s.sh, dari [2]

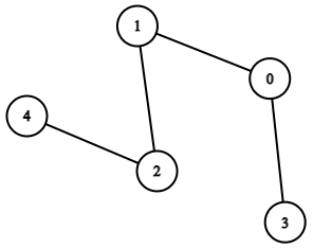
Baris kedua untuk mengetahui kasus uji keberapa yang sedang dijalankan. Baris ketiga terdapat program gen dengan parameter *i* yang menulis kasus uji secara random ke file intc. Program gen inilah yang akan dibuat. Baris keempat dan kelima menyalurkan input kasus uji ke dua program dan menyimpan outputnya di out1 dan out2. Baris keenam membandingkan out1 dan out2 dengan mengabaikan perbedaan *whitespace* dan menghentikan pengujian bila berbeda. Semua itu diiterasi hingga 100 kali.

Skrip tersebut dapat dimodifikasi sesuai keperluan. Batasan $i \leq 100$ dapat dihilangkan untuk pengujian mandiri. Solusi sol_A dapat diganti dengan solusi yang lebih lambat, misal didapat dari cara *brute force*. File input dapat dibuat dan disimpan semua terlebih dahulu. File output juga dapat disimpan semua.

C. Prüfer Code

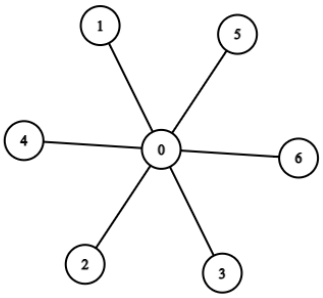
Prüfer code atau Prüfer sequence adalah barisan unik yang merepresentasikan pohon berlabel tertentu. Prüfer code ditemukan oleh Heinz Prüfer pada tahun 1918 yang saat itu dia gunakan untuk membuktikan formula Cayley [3]. Pohon berlabel dengan *n* simpul dengan indeks dimulai dari 0 memiliki Prüfer code berupa barisan *n-2* bilangan yang setiap bilangan

berinterval [0, *n-1*]. Adapun untuk n=1, Prüfer code tidak berlaku dan makalah ini hanya meninjau pohon dengan simpul minimal dua. Bila *n-2* bilangan tersebut berbeda, pohon yang terbentuk akan berupa garis seperti gambar ini.



Gambar 2. Pohon dengan Prüfer code 0 1 2

Bila *n-2* bilangan tersebut sama, akan terbentuk pohon dengan 1 pusat. Contohnya barisan 0 0 0 0 0.



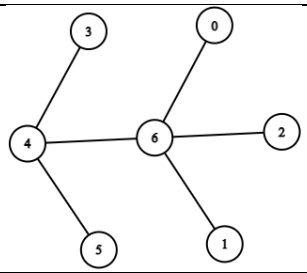
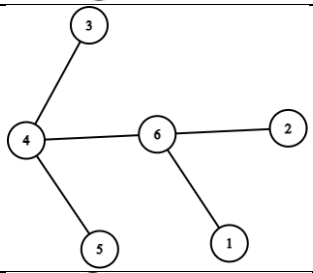
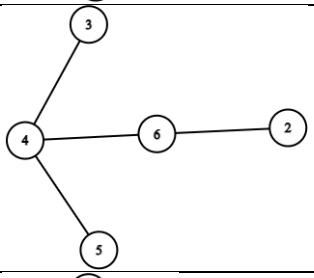
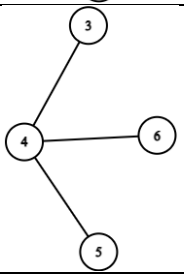
Gambar 3. Pohon dengan Prüfer code 0 0 0 0 0

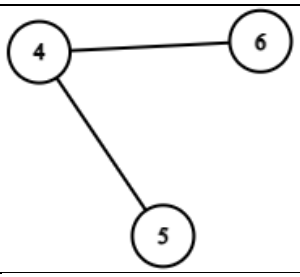
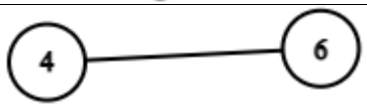
Referensi [1] menjelaskan algoritma untuk membentuk Prüfer code dari sebuah pohon adalah dengan mengulangi *n-2* kali prosedur berikut.

1. Dari pohon, cari daun dengan nomor terkecil.
2. Hapus daun dan sisinya dari pohon dan tulis nomor simpul yang bertetangga dengannya (dijamin hanya ada 1 simpul).

Contoh untuk Prüfer code 6 6 6 4 4 ada pada tabel berikut.

Tabel I. Contoh Prosedur Pembentukan Prüfer code dari Suatu Pohon

Iterasi ke-	Pohon	Prüfer code sementara	Keterangan
0			
1		6	Daun nomor terkecil = 0 Tetangga daun terkecil = 6
2		6 6	Daun nomor terkecil = 1 Tetangga daun terkecil = 6
3		6 6 6	Daun nomor terkecil = 2 Tetangga daun terkecil = 6

4		6 6 6 4	Daun nomor terkecil = 3 Tetangga daun terkecil = 4
5		6 6 6 4 4	Daun nomor terkecil = 5 Tetangga daun terkecil = 4

Dari algoritma ini juga dapat disimpulkan.

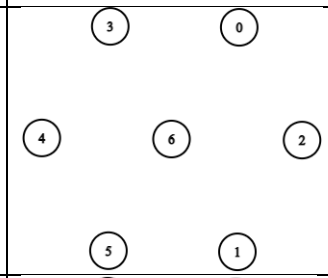
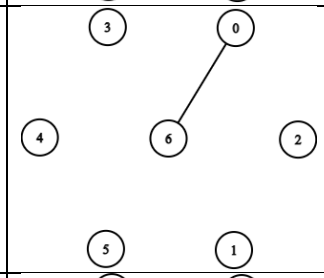
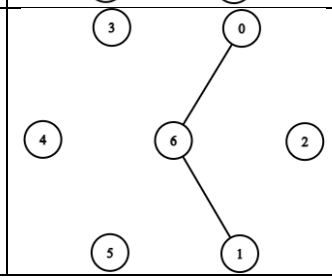
1. Pohon yang tersisa setelah Prüfer code dibuat terdiri dari 2 simpul dan salah satunya bernomor $n-1$.
2. Setiap simpul muncul di Prüfer code sebanyak derajatnya dikurangi satu. Karena untuk menghapus suatu simpul, kita perlu menghapus semua tetangganya hingga tersisa satu tetangga.

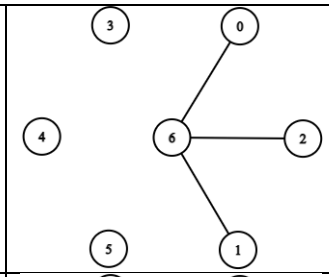
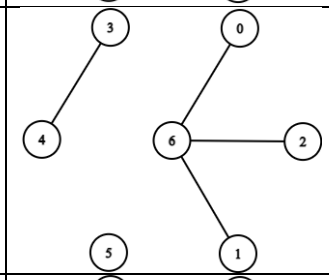
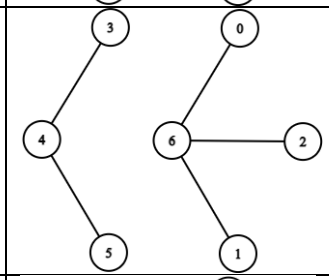
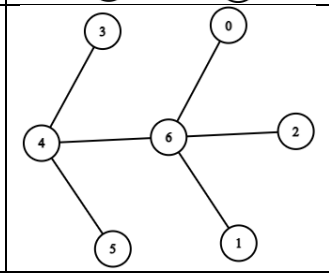
Fakta kedua ini dimanfaatkan untuk membentuk pohon dari sebuah Prüfer code, yakni banyak simpul adalah ukuran barisan ditambah dua, daun dari pohon adalah bilangan yang tidak muncul di Prüfer code, dan derajat dari simpul-simpul adalah satu ditambah banyak kemunculannya di Prüfer code. Kita simpan daftar daun dan derajat simpul lalu lakukan prosedur berikut untuk setiap bilangan di Prüfer code dari awal.

1. Buat dan tulis sisi yang menghubungkan bilangan di Prüfer code dengan daun terkecil pada daftar.
2. Hapus daun tersebut dari daftar dan kurangi derajat bilangan di Prüfer code. Bila derajatnya menjadi satu, masukan ke daftar daun.

Pada daftar daun akan tersisa dua bilangan dengan derajat 1, buat sisi yang menghubungkan keduanya dan kita selesai dengan $n-1$ sisi. Contoh untuk Prüfer code 6 6 6 4 4 ada pada tabel berikut.

Tabel II. Contoh Prosedur Pembentukan Pohon dari Suatu Prüfer code

langkah ke-	Pohon	Prüfer code yang tersisa	Keterangan
0		6 6 6 4 4	Daftar daun: {0,1,2,3,5} Daftar derajat: {1,1,1,1,3,1,4}
1		6 6 4 4	Daftar daun: {1,2,3,5} Daftar derajat: {0,1,1,1,3,1,3} Sisi yang dibuat: {0,6}
2		6 4 4	Daftar daun: {2,3,5} Daftar derajat: {0,0,1,1,3,1,2} Sisi yang dibuat: {1,6}

3		4 4	Daftar daun: {3,5,6} Daftar derajat: {0,0,0,1,3,1,1} Sisi yang dibuat: {2,6}
4		4	Daftar daun: {5,6} Daftar derajat: {0,0,0,0,2,1,1} Sisi yang dibuat: {3,4}
5			Daftar daun: {4,6} Daftar derajat: {0,0,0,0,1,0,1} Sisi yang dibuat: {5,4}
6			Daftar daun: {} Daftar derajat: {0,0,0,0,0,0,0} Sisi yang dibuat: {4,6}

Setiap Prüfer code membentuk pohon dan setiap pohon dapat dibentuk Prüfer codenya dan dibentuk kembali dari Prüfer code itu. Artinya ada bijeksi atau korespondensi satu-satu antara Prüfer code dan pohon. Dari sini, dapat diturunkan formula Cayley, yaitu banyaknya pohon merentang dengan *n* simpul adalah n^{n-2}.

III. Impelementasi Algoritma

A. Implementasi Prüfer Code ke Kasus Uji Pohon

```
vector<pair<int, int>> pruefer_decode(vector<int> const& code) {
    int n = code.size() + 2;
    vector<int> degree(n, 1);
    for (int i : code)
        degree[i]++;

    int ptr = 0;
    while (degree[ptr] != 1)
        ptr++;
    int leaf = ptr;
```

(a)

```
    vector<pair<int, int>> edges;
    for (int v : code) {
        edges.emplace_back(leaf, v);
        if (--degree[v] == 1 && v < ptr) {
            leaf = v;
        } else {
            ptr++;
            while (degree[ptr] != 1)
                ptr++;
            leaf = ptr;
        }
    }
    edges.emplace_back(leaf, n-1);
    return edges;
}
```

(b)

Gambar 4. Cuplikan Fungsi pruefer_decode: (a) bag. 1, (b) bag. 2. Sumber: dokumentasi pribadi, dikutip dari [1]

Fungsi pruefer_decode itu menerima barisan Prüfer code dan mengembalikan vector sisi dari pohon. Algoritma yang digunakan adalah algoritma pada landasan teori dengan optimasi pencarian daun terkecil tanpa menyimpan daftar daun. Caranya dengan menggunakan variabel *ptr* yang mencari daun terkecil pertama. Variabel *ptr* memiliki sifat interval [0,ptr] hanya terdapat 1 daun setiap sebelum penghapusan. Setiap penghapusan, daun terkecil berikutnya adalah tetangga dari

daun terkecil sebelumnya bila tetangga ini lebih kecil dari *ptr* dan berderajat 1. Bila tidak, *ptr* akan iterasi hingga menemui daun terkecil dan sifat *ptr* tetap. Dapat dilihat banyak langkah *ptr* linear sehingga kompleksitas waktu dari fungsi ini adalah O(n).

B. Implementasi gen.cpp

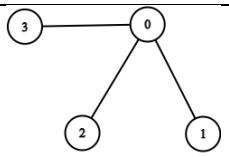
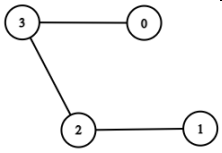
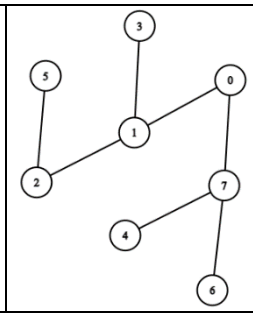
```cpp
#include <bits/stdc++.h>
using namespace std;

vector<pair<int, int>> pruefer_decode(vector<int> const& code) {

int rand(int a, int b) {
    return a + rand()%(b-a+1);
}

int main(int argc, char* argv[]) {
    srand(atoi(argv[1])); //atoi convert string to int
    int n = rand(2,10);
    vector<int> code(n-2);
    for(int i=0;i<n-2;i++) {
        code[i] = rand(0, n-1);
    }
    vector<pair<int,int>> edges = pruefer_decode(code);
    cout <<n <<"\n";
    for(auto [u,v] : edges) {
        cout <<u <<" " <<v <<"\n";
    }
    return 0;
}
```

Gambar 5. Cuplikan program gen.cpp, sumber: dokumentasi pribadi

Untuk menghasilkan kasus uji pohon yang random, program *gen* menerima parameter *i* (lihat landasan teori B) dan menjadikannya seed untuk rand dengan fungsi srand(). Nilai *n* diacak dari 2 sampai 10 karena ingin dibuat dua ke atas dan nilai bilangan Prüfer code diacak dari 0 hingga $n-1$. Hasilnya dimasukkan pada fungsi pruefer_decode dan dihasilkan daftar sisinya. Program *gen* lalu menulis *n* dan $n-1$ sisinya setiap baris. Nilai *n* juga dapat diacak dalam rentang yang lebih luas seperti [2,5000] untuk menguji batas waktu dan memori pada kontes. Selain itu lebih disarankan memakai batas atas yang kecil dengan banyak iterasi untuk mengetes variasi yang menyeluruh.

Setelah diuji dengan nilai *i* berbeda, didapat program menghasilkan pohon dengan cukup cepat. Contoh dapat dilihat pada tabel berikut.

Tabel III. Hasil Kasus Uji dan Pohon yang Bersesuaian

Kasus Uji	Pohon
PS D:\B\T> ./gen 11 4 1 0 2 0 0 3	
PS D:\B\T> ./gen 0 4 0 3 1 2 2 3	
PS D:\B\T> ./gen 4 8 3 1 4 7 5 2 2 1 1 0 0 7 6 7	

Dari Tabel II dapat dilihat kasus uji dapat berupa pohon dengan satu pusat, pohon berupa garis, dan variasi pohon lain.

IV. KESIMPULAN

Hasil program *gen* yang mengimplementasikan teori Prüfer code berjalan sesuai yang diharapkan. Program berjalan cepat dengan kompleksitas O(n) dan menghasilkan pohon yang bervariasi yang dapat dilihat pada Tabel III. Program yang dibuat mengambil indeks awal pohon 0, bila dibutuhkan pohon berindeks awal 1 dapat dilakukan modifikasi.

Kasus uji yang dikeluarkan program *gen* saat ini bersifat acak. Untuk pengujian yang urut dan lengkap akan memakan waktu lama, namun dapat dilakukan dengan melihat Prüfer code sebagai bilangan basis n. Pengujian adalah fase yang penting di kompetisi ataupun praktik kehidupan nyata. Sayangnya makalah ini belum dapat menunjukkan contoh pengujian dan *counter test case*-nya.

V. UCAPAN TERIMA KASIH

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena atas rahmat dan berkat-Nya makalah ini dapat selesai. Penulis juga ingin mengucapkan terima kasih kepada:

1. Orang tua penulis yang selalu memberi dukungan,
2. Bapak dan Ibu dosen pengampu mata kuliah Matematika Diskrit IF2120 terutama Pak Rinaldi Munir yang sudah membimbing Kelas K1 selama satu semester ini,
3. Teman-teman yang saling memberi dukungan,
4. Graf editor dari CSAcademy tempat penulis membuat gambar-gambar graf untuk makalah ini.
5. Errichto dan web cp-algorithms tempat penulis belajar banyak.
6. Pihak-pihak lain yang tidak sempat disebutkan.

REFERENSI

[1] cp-algorithms.com, "Prüfer code". https://cp-algorithms.com/graph/pruefer_code.html. [diakses 13 Desember 2021]

[2] K. Debowski, "How to test your solution in Competitive Programming, on Linux?". https://www.youtube.com/watch?v=JXTVOyQpSGM. [diakses 13 Desember 2021]

[3] Prüfer, H. (1918). "Neuer Beweis eines Satzes über Permutationen". Arch. Math. Phys. 27: 742–744.

[4] R. Munir. "Pohon (Bag. 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf. [diakses 14 Desember 2021]

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, 15 Desember 2021

Ahmad Romy Zahran (13520009)