Aplikasi Pohon Keputusan pada Fungsi *Random*

Yollanda Sekarrini - 13512051
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512051@std.itb.ac.id

***Abstract*—Fungsi *random* merupakan salah satu fungsi dalam bahasa pemrograman yang berguna untuk mengacak suatu data, bisa dalam kumpulan integer, character, atau tipe bentukan. Setiap kali dijalankan, fungsi akan menghasilkan keluaran yang berbeda dan memungkinkan untuk menghasilkan keluaran yang sama. Tapi tak jarang seorang *programmer* membutuhkan fungsi *random* yang tidak hanya mengacak, tapi juga melakukan fungsi – fungsi tertentu seperti pengacakan suatu integer pada interval tertentu dan pengacakan yang tak berulang. Untuk mendapatkan fungsi – fungsi *random* tersebut, seorang *programmer* membutuhkan kodingan tambahan. Proses pengacakan tersebut dapat digambarkan memalui pohon keputusan sehingga pemahaman terhadap kinerja fungsi dapat mudah dimengerti.**

***Kata Kunci*—Acak, Fungsi *Random*, Pohon, Pohon Keputusan.**

I. PENDAHULUAN

Pohon merupakan graf tak-berarah yang bersifat terhubung dan tidak mengandung sirkuit. Pohon telah digunakan sejak tahun 1857 oleh Arthur Cayley untuk menghitung jumlah senyawa kimia. Bahkan melalui teori pohon ini, Cayley berhasil mengemukakan sebuah teorema yang bernama teorema Cayley yang menyatakan bahwa jika n adalah bilangan bulat lebih dari satu, maka banyaknya pohon berlabel yang berbeda dari n simpul adalah n^{n-2}. Teori yang sederhana tetapi banyak mendapat pengakuan dari ilmuwan dunia.

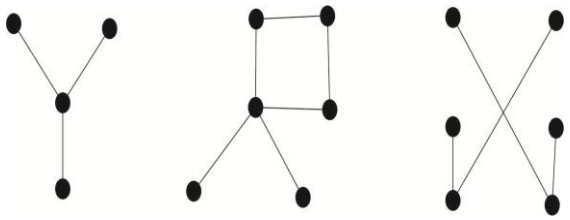
Di dalam ilmu komputer, pohon memiliki banyak terapan diantaranya, pohon ekspresi (*expression tree)*, pohon pencarian biner (*binary search tree*), kode prefix (*prefix code*), kode Huffman (*Huffman code*), dan pohon keputusan (*decision tree*). Di dalam makalah ini, terapan pohon yang digunakan adalah pohon keputusan. Pohon keputusan merupakan salah satu pemodelan persoalan yang terdiri dari berbagai keputusan dan mengarah ke keputusan tersebut. Karena kegunaanya itulah, proses pemilihan dari fungsi *random* akan dapat digambarkan.

Hampir seluruh aplikasi menggunakan fungsi *random.* Oleh karena itu, pengetahuan tentang fungsi *random* akan sangat membantu seorang *programmer* dalam membuat variasi program. Selain itu, fungsi *random* juga mempunyai banyak variasi, dua diantaranya adalah fungsi *random* yang hanya mengacak dari interval tertentu dan fungsi *random* yang tak berulang. Untuk mendapatkan fungsi *random* yang menjalankan fungsi tersebut, khusunya dalam bahasa C, dibutuhkan tambahan kodingan.

II. TEORI

A. Pohon

Pohon dalam bidang Matematika Diskrit berarti graf tak-berarah yang bersifat terhubung dan tidak mengandung sirkuit. Karena pohon mengacu ke teori graf, pohon dilambangkan dengan *G = (V, E). V* adalah vertex (simpul) dan *E* adalah edge (sisi).



Gambar 1. Contoh graf berbentuk pohon, graf sirkuler, dan graf tak-terhubung

Dari gambar di atas, terlihat bahwa graf pertama dari kiri adalah graf yang terhubung dan tidak mempunyai sirkuit. Itulah yang disebut dengan pohon. Sedangkan dua graf lainnya bukan pohon.

Sifat- sifat pohon diantaranya :

- Setiap pasang simpul di dalam pohon terhubung dengan lintasan tunggal (unik).
- Pohon merupakan graf terhubung.
- Pohon tidak memiliki sirkuit.
- Jika n adalah banyaknya simpul pada pohon dan m adalah banyaknya sisi, maka *m = n - 1*.
- Penambahan satu sisi pada pohon akan mengakibatkan terbentuknya satu sirkuit.

- Semua sisi dari pohon adalah jembatan yang berarti, jika salah satu sisi dihapus akan menyebabkan pohon terpecah menjadi dua bagian.

Beberapa terminologi dalam pohon, yaitu :

1. *Child* (anak)

Suatu simpul disebut dengan *child* jika terdapat satu simpul sebelumnya yang bersisian.

2. *Parent* (orangtua)

Lawan dari *child*, suatu simpul disebut *parent* jika terdapat satu atau lebih simpul setelahnya yang bersisian

3. .*Path* (lintasan)

Lintasan berarti urutan simpul – simpul dari simpul asal ke simpul tujuan. Sedangkan panjang lintasan adalah jumlah sisi yang dilalui dalam lintasan.

4. *Descendant* (keturunan)

Suatu simpul disebut *descendant* jika terdapat dua/lebih simpul sebelumnya.

5. *Ancestor* (leluhur)

Lawan dari *descendant*, suatu simpul disebut *ancestor* jika terdapat dua/lebih simpul sebelumnya.

6. *Sibling* (saudara kandung)

Sibling yaitu suatu simpul yang mempunyai orangtua yang sama dengan simpul yang lainnya.

7. *Leaf* (daun)

Leaf yaitu suatu simpul yang tidak mempunyai anak.

8. *Subtree* (subpohon)

Subtree yaitu sekumpulan simpul, berawal dari akar, diakhiri dengan simpul kosong. Daun juga disebut subpohon yang terdiri dari akar sebagai orangtua dan simpul kosong sebagai anaknya.

9. *Degree* (derajat)

Degree yaitu jumlah subpohon (atau jumlah anak) yang terdapat pada suatu simpul.

10. *Level* (tingkat atau aras)

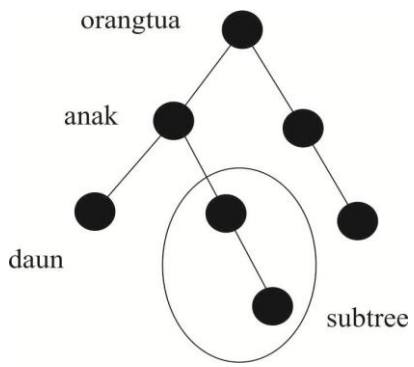
Level yaitu jumlah simpul dari simpul tersebut sampai ke akar dikurang satu sehingga *level* dari akar adalah nol. Sebagian referensi menyatakan bahwa *level* dari akar satu, tapi pada makalah ini, acuan *level* nol terletak pada akar.

11. *Internal Nodes* (simpul dalam)

Internal Nodes yaitu satu/lebih simpul sebelum daun.

12. *Depth* (kedalaman)

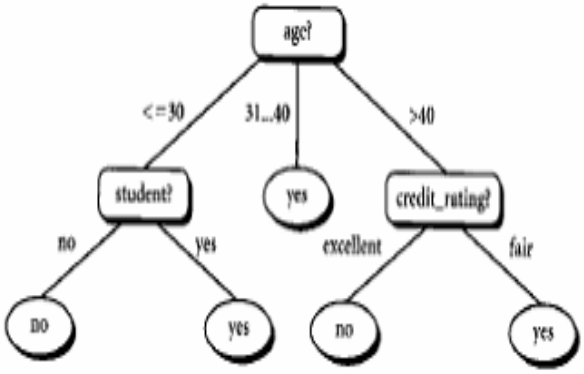
Depth yaitu *level* maksimum dari suatu pohon.



Gambar 2. Salah satu contoh pohon beserta keterangannya

Pohon mempunyai banyak terapan seperti, pohon ekspresi (*expression tree)*, pohon pencarian biner (*binary search tree*), kode prefix (*prefix code*), kode Huffman (*Huffman code*), dan pohon keputusan (*decision tree*). Salah satu terapan yang dibahas pada makalah ini adalah pohon keputusan (*decision tree*).

Pohon keputusan merupakan pohon yang tiap simpulnya memodelkan persoalan kemudian diarahkan ke daun berdasarkan keputusan. Dari pengertian di atas, dapat disimpulkan bahwa daun adalah solusi dari persoalan tersebut.



Gambar 3. Contoh pohon keputusan
Sumber: http://fairuzelsaid.files.wordpress.com/2009/11/image002.gif

Pohon keputusan memiliki banyak kelebihan dalam memodelkan persoalan, diantaranya:

1. Mudah untuk diinterpretasikan.
2. Mudah unutk dimengeti.
3. Memungkinkan untuk menjelajah semua kemungkinan.
4. Lebih akurat dalam hal pemodelan.
5. Pengambilan keputusan yang rumit dapat diubah menjadi lebih sederhana.

Walaupun pohon keputusan memiliki banyak kelebihan, setiap metode tentu ada kelemahannya. Diantara kelemahan dari pemodelan persoalan menggunakan pohon keputusan adalah :

1. Hasil atau solusi dari pohon keputusan bergantung kepada desain dari pohon itu dibuat.
2. Adanya kesulitan dalam pembuatan pohon keputusan yang optimal.
3. Adanya kesulitan membuat pohon keputusan yang lebih umum.

B. Fungsi *random*

Fungsi *random* merupakan fungsi pengacak suatu data sehingga saat program dijalankan, fungsi akan menghasilkan keluaran yang berbeda dan mungkin sama.

Fungsi *random* dalam bahasa C yaitu `rand()`, `random()` dan `randomize()`. Banyak variasi fungsi *random* dalam C, tapi pada makalah ini, lebih memfokuskan pada fungsi `rand()`. Untuk menggunakan fungsi `rand()`, diawal program, harus meng-*include*-kan *library* `<stdlib.h>`.

Ada beberapa hal yang harus diperhatikan dalam membuat fungsi *random* menggunakan `rand()` diantaranya :

1. *Scaling*

Fungsi `rand()`akan mengacak bilangan bertipe integer dari nol sampai batas maksimal nilai yang di-*return*. Oleh karena itu dibutuhkan fungsi modulo sebagai pembatas nilai yang ingin diacak. Sebagai contoh, seorang *programmer* membutuhkan suatu program yang mengacak suatu bilangan integer dari 1 sampai 5, maka fungsi modulo yang harus diketik oleh *programmer* tersebut adalah :

```
rand () % 5 + 1
```

Contoh lain, mengacak bilangan integer dari 2 sampai 7, maka ketikannya adalah :

```
rand () % 6 + 2
```

Dengan memainkan operator tambah pada fungsi *random* interval, pengacakan dapat dibentuk sedemikian mungkin sesuai keinginan sang *programmer*.

2. *Seeding*

Fungsi `rand()`secara *default* akan mengacak bilangan integer dengan pengulangan yang sama tiap kalinya. Oleh karena itu, perlu dilakukannya *seeding* agar data yang diacak berbeda keluarannya tiap kali fungsi dijalankan. Fungsi yang melakukan hal tersebut adalah `srand()`. Dengan memasukkan `time(NULL)` sebagai parameternya, fungsi `rand()` akan menghasilkan keluaran yang berbeda berdasarkan jumlah detik terhitung dari 1 Januari 1970 sebagai *seed_number*-nya.

Contoh program yang menjalankan fungsi `rand()` :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    /* KAMUS */
    int i, jumlah;

    /* ALGORITMA */
    srand(time(NULL));
    printf("Jumlah data: ");
    scanf("%d", &jumlah);
    for (i = 1; i <= jumlah; i++) {
        printf("%d\n", rand() % 10 +
1;
    }
}
```

Gambar 3. Mengacak integer dari 1-10 sebanyak jumlah data masukan *user*.

Fungsi *random* menjadi salah satu kebutuhan bagi *programmer* untuk diketahui. Banyak program yang menggunakan fungsi ini, seperti

- Windows Media Player, menggunakan fungsi *random* untuk pengacakan urutan lagu.
- Microsoft Office Excel bahkan mempunyai fungsi *random* sendiri.
- Permainan Acak Kata, Tebak Angka dan Hangman. Jelas menggunakan fungsi *random.* Hampir semua permainan yang diprogram menggunakan fungsi *random.*

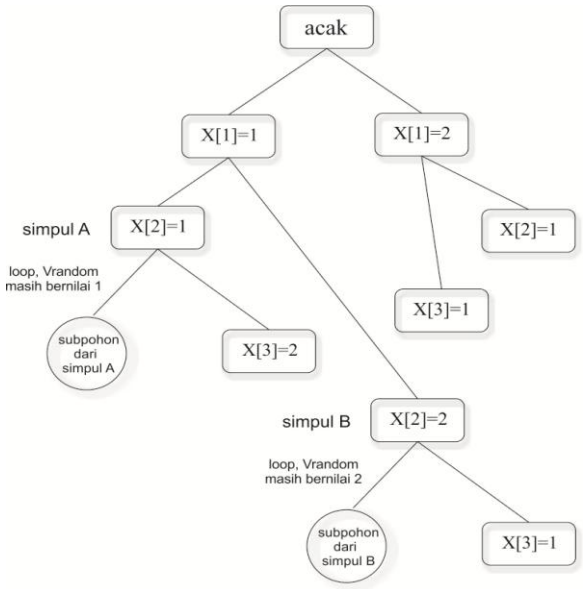
III. PENERAPAN POHON KEPUTUSAN PADA FUNGSI *RANDOM*

Pohon keputusan memiliki peran untuk mengarahkan suatu persoalan ke solusinya. Dengan peran tersebut, tahap demi tahap proses pengacakan fungsi *random* `rand()` akan dapat digambarkan.

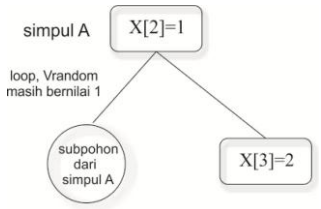
Pertama-tama, tentukan fungsi *random*-nya. Contoh fungsi *random* yang mengacak integer 1, 1, dan 2 :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    /* KAMUS */
    int Vrandom;
    int V1, V2;
    int X[4];
    int i;

    /* ALGORITMA */
    srand(time(NULL));
    Vrandom = rand() % 2 + 1; /*
random */
    V1 = Vrandom;  /* V1 = 1 | 2 */
    X[1] = V1;
    if (V1==1) {  /* acak nilai 1 |
2 */
      Vrandom = rand() % 2 + 1;
      V2 = Vrandom;
      X[2] = V2;
      Vrandom = rand() % 2 + 1;
      while(Vrandom==V2) {
            Vrandom = rand() % 2 +
1;
      }
      /* Vrandom != V2 */
      X[3] = Vrandom;
    }
    else {  /* V1==2 */
      X[2] = 1;
      X[3] = 1;
    }
    for (i=1; i<=3; i++) {
      printf ("Angka ke-%d = %d\n",
i, X[i]);
}
```

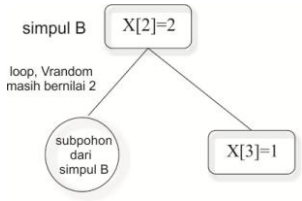
Dari fungsi *random* tersebut, terdapat *loop* `while` yang terus terjadi bila `Vrandom` masih bernilai sama dengan simpul sebelumnya. Oleh karena itu, pohon keputusan dibuat menjadi bentuk pohon rekursif yang semu. Pohon keputusan tersebut adalah :



Gambar 4. Pohon keputusan dari fungsi *random* yang mengacak integer 1, 1, dan 2



Gambar 5.a. Subpohon dari simpul A



Gambar 5.b. Subpohon dari simpul B

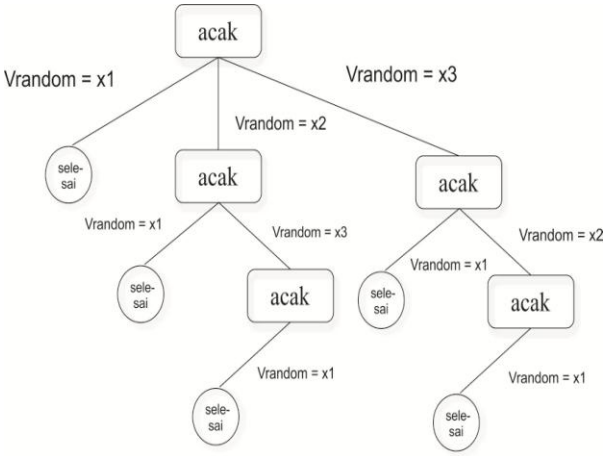
Dari gambar tersebut, terlihat bahwa pohon seolah-olah kembali ke simpul sebelumnya. Hal tersebut dibuat agar hasil acakan masih terdiri dari 1, 1, dan 2. Hasil dari pohon keputusan tersebut merupakan hasil yang kurang baik karena sebuah pohon seharusnya tidak dapat kembali ke simpul sebelumnya.

Ditentukan kembali fungsi *random* yang akan direpresentasikan dalam bentuk pohon keputusan. Fungsi *random* ini merupakan fungsi *random* tak berulang yang akan mengacak suatu integer x1, x2 dan x3.

Fungsi *random*-nya adalah :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    /* KAMUS */
    int i,V1,V2;
    int Vrandom;

    /* ALGORITMA */

    srand(time(NULL));
    Vrandom = rand() % 3 + 1;
    printf("Angka ke-%d = %d\n", 1,
Vrandom);
    V1 = Vrandom;
    Vrandom = rand() % 3 + 1;
    while(Vrandom == V1) {
        Vrandom = rand() % 3 + 1;
    }
    printf("Angka ke-%d = %d\n", 2,
Vrandom);
    V2 = Vrandom;
    while((Vrandom == V1)||( Vrandom
== V2)) {
        Vrandom = rand() % 3 + 1;
    }
    printf("Angka ke-%d = %d\n", 3,
Vrandom);
}
```

Dari fungsi *random* tersebut, dapat dibuat pohon keputusannya sebagai berikut :



Gambar 6. Pohon keputusan dari fungsi *random* jika nilai acak yang ingin dikeluarkan adalah x1.

Pohon keputusan tersebut menggambarkan proses fungsi *random* tak berulang yang terdiri dari tiga integer yang akan diacak. Setiap integer yang telah dikeluarkan tidak akan diacak lagi untuk pengacakkan berikutnya. Gambar di atas, menunjukkan proses terpilihnya integer

pertama yang dilambangkan dengan x1.

Dari pohon tersebut terlihat bahwa x1 mempunyai lima kemungkinan lintasan yang akan terpilih, yaitu :

- Acak, x1 (langsung terpilih diawal pengacakkan)
- Acak, x2, acak, x1
- Acak, x3, acak, x1
- Acak, x2, acak, x3, acak, x1
- Acak, x3, acak, x2, acak, x1

Jika beruntung, x1 akan terpilih saat pertama kali diacak. Tetapi jika tidak, x1 akan keluarkan saat pengacak yang ketiga dan tentunya akan memakan waktu yang lebih lama.

Dari dua contoh penerapan pohon keputusan terhadap fungsi *random* tersebut ada beberapa hal yang perlu diperhatikan, yaitu :

- Kedalaman pohon keputusan sama dengan banyaknya nilai yang akan di acak (jika pengacakan tidak berulang).
- Semakin banyak nilai yang diacak, maka kemungkinan lintasan dari nilai tersebut juga semakin banyak.
- Sulit untuk memodelkan pengacakkan nilai yang banyak.
- Hanya dapat memodelkan persoalan terpilihnya satu bilangan acak. Belum bisa memodelkan persoalan yang lebih umum.
- Terdapat suatu pengulangan pada pohon yang seharusnya itu tidak terjadi.

IV. KESIMPULAN

Pohon keputusan mempunyai banyak kelebihan dalam hal memodelkan persoalan. Contoh aplikasi pohon keputusan yaitu pengurutan tiga elemen, kemungkinan terpilihnya seorang karyawan berdasarkan syarat/keputusan tertentu dan pemilihan satu koin yang *defect* dari tiga koin yang tidak diketahui beratnya masing-masing. Pohon keputusan menjadi suatu metode sederhana yang dapat menyelesaikan persoalan yang rumit.

Tapi dari persoalan di atas, penulis menyadari bahwa, penerapan pohon keputusan untuk memodelkan persoalan pohon keputusan kurang tepat. Karena persoalan fungsi *random* merupakan persoalan yang harusnya berulang-ulang. Pohon tidak dapat melakukan hal yang berulang-ulang. Terbukti dari definisinya yaitu graf yang terhubung dan tidak mempunyai sirkuit yang berarti setiap solusi akan unik dan tidak akan kembali ke keputusan sebelumnya. Jika memaksakan penerapan pohon keputusan terhadap fungsi *random* yang berulang tersebut, maka pohon yang dapat dibentuk sangat terbatas. Seperti yang telah dicontohkan di atas, pohon keputusan yang terbentuk hanya terbatas pada proses pemilihan satu bilangan acak. Tidak dapat memodelkan pemilihan bilangan acak yang lebih umum.

Maka dari itu, pemodelan persoalan fungsi *random* lebih cocok menggunakan graf biasa yang tentunya mempunyai sisi sirkuit agar dapat memodelkan pengulangan pengacakan suatu data.

REFERENSI

Munir, Rinaldi. "Diktat Kuliah IF2120 Matematika Diskrit". Program Studi Teknik Informatika, 2006, Bandung, Indonesia.

http://fairuzelsaid.files.wordpress.com/2009/11/image002.gif . Diakses pada tanggal : 15 Desember 2013.

http://filiadielias.blogspot.com/2013/02/c-code-cara-membuat-angka-random.html . Diakses pada tanggal : 15 Desember 2013

http://pohonkeputusan-sukoand.blogspot.com/2011/01/pohon-keputusan.html . Diakses pada tanggal : 15 Desember 2013

http://gudangiclan.blogspot.com/2011/12/membuat-fungsi-random-huruf-dan-angka.html . Diakses pada tanggal : 16 Desember 2013.

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, 16 Desember 2013

ttd

Yollanda Sekarrini (13512051)