Kompleksitas Algoritma Euclidean dan Stein(FPB Biner)

Okharyadi Saputra (13510072)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
okharyadi@students.itb.ac.id

ABSTRAK

Algoritma yang mangkus adalah algoritma yang meminimumkan kebutuhan waktu dan ruang. Hal ini bergantung pada ukuran masukan yang disimbolkan dengan ***N*** yaitu jumlah data yang diproses. Salah satu jenis algoritma yang ada adalah algoritma untuk mencari nilai FPB(Faktor Persekutuan Terbesar). FPB adalah bilangan terbesar yang dapat membagi dua bilangan atau beberapa bilangan. Ada banyak algoritma yang dapat digunakan untuk mencari nilai FPB seperti Algoritma Euclidean dan Algoritma Stein(Algoritma FPB Biner). Karena itu dalam makalah ini akan dilakukan analisis terhadap kemangkusan algoritma-algoritma tersebut untuk mencari tahu efisiensinya.

Kata kunci- Algoritma mangkus, Euclidean, Stein, FPB, efisiensi.

I. PENDAHULUAN

Terdapat dua macam kompleksitas algoritma, yaitu Kompleksitas waktu dan kompleksitas ruang.

Kompleksitas waktu disimbolkan dengan T(n) dan kompleksitas ruang S(n). Kompleksitas waktu, T(n), diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankkan algoritma sebagai fungsi dari ukuran masukan n. Komplesitas ruang, S(n) diukur dari memori yang digunakan oleh struktur data yang terdapat di dalam algoritma sebagai fungsi dari ukuran masukan n.

Dengan menggunakan besaran kompleksitas waktu/ruang algoritma, kita dapat menentukan laju peningkatan waktu (ruang) yang diperlukan algoritma dengan meningkatnya ukuran masukan n(1).

Di dalam sebuah algoritma terdapat bermacam jenis operasi:

- Operasi baca/tulis,
- Operasi aritmetika (+, -, *, /)
- Operasi pengisian nilai (assignment)
- Operasi pengaksesan elemen larik
- Operasi pemanggilan fungsi/prosedur
- dll

Kompleksitas algoritma dinyatakan secara *asimptotik* dengan notasi big-O. Jika kompleksitas waktu untuk menjalankan suatu algoritma dinyatakan dengan *T(n)*, dan memenuhi

$$T(n) \leq C(f(n))$$

untuk $n \geq n_0$, maka kompleksitas dapat dinyatakan dengan $T(n) = O(f(n))$.

Terdapat 2 jenis penggunaan notasi Big O, yaitu :
1. Infinite asymptotics
2. Infinitesimal asymptotics

Perbedaan kedua jenis penggunaan notasi ini hanya pada aplikasi. Sebagai contoh, pada infinite asymptotics dengan persamaan:

$$T(n) = 4n^2-4n+4$$

Untuk n yang besar, pertumbuhan T(n) akan sebanding dengan n^2 dan dengan mengabaikan suku yang tidak mendominasi, maka kita tuliskan

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

Pada infinitesimal asymptotic**s,** Big O digunakan untuk menjelaskan kesalahan dalam aproksimasi untuk sebuah fungsi matematika, sebagai contoh

$$e^x = 1 + \frac{x}{1} + \frac{x^2}{2} + O(x^3), \quad x \to 0$$

Kesalahannya memiliki selisih

$$e^x - \left(1 + \frac{x}{1} + \frac{x^2}{2}\right)$$

Untuk mencari sebuah FPB(Faktor Persekutuan Terbesar) dari dua buah bilangan bulat, kita dapat menggunakan beberapa algoritma diantaranya:

1. Algoritma Euclidean adalah metode yang efisien untuk menghitung pembagi umum terbesar (FPB) dari dua bilangan bulat tanpa meninggalkan sebuah sisa.

2. Algoritma Stein(Algoritma FPB Biner) adalah algoritma yang menghitung pembagi umum terbesar dari dua bilangan bulat nonnegatif dengan menerapkan identitas-identitas secara berulang kali.

Gambar 1 Euclidean

Algoritma Euclidean memiliki aplikasi teoritis dan praktis. Algoritma ini adalah elemen kunci dari algoritma RSA, sebuah enkripsi public-key metode yang luas digunakan dalam perdagangan elektronik. Hal ini digunakan untuk memecahkan persamaan Diophantine, Algoritma Euclidean juga dapat digunakan dalam metode rantai untuk mencari akar Sturm nyata dari polinomial, dan dalam beberapa algoritma faktorisasi bilangan bulat yang modern.

Contoh:
FPB(24,12)=12 (artinya 12 merupakan bilangan bulat terbesar yang membagi 24 dan 12)
FPB(24,9)=3 (artinya 3 merupakan bilangan bulat terbesar yang membagi 24 dan 9)

Cara Menentukan FPB:

1. **Mengubah ke bentuk perkalian berpangkat.**
   FPB(24,12);
   $24=2^3x3$;
   $12=2^2x3$;
   Ambil bilangan dengan pangkat terkecil yaitu 2^2 dan 3 maka FPB dari 24 dan 12 adalah $2^2x3=12$

2. **Menggunakan tangga bersusun**.

```
* 2 | 24 12
* 2 | 12  6
  2 |  6  3
* 3 |  3  3
      1  1
```

   Tanda bintang menunjukkan kedua bilangan habis dibagi berarti FPB(24,12)=2x2x3=12

3. **Menggunakan Algoritma Euclidean**

4. **Menggunakan Algoritma Stein(Algoritma FPB Biner)**

Pengembangan Algoritma Euclidean
Algoritma Euclidean selain digunakan untuk mencari FPB dari dua bilangan, juga dapat menemukan bilangan x dan y dalam identitas Bezout (persamaan linear diophantine) mx+ny=FPB(m,n);

FPB(312,70)=2 sebagai kombinasi lanjar dari 312 dan 70

Dengan menerapkan algoritma Euclidean untuk memperoleh FPB(312,70)=2

312=4.70 + 32 (i)
70 = 2.32 + 6 (ii)
32 = 5.6 + 2 (iii)
6 = 3.2 + 0 (iv)

Susun pembagian nomor (iii) menjadi
2=32 – 5.6 (iv)
Susun pembagian nomor (iii) menjadi
6=70-2.32 (v)
Sulihkan (v) ke dalam (iv) menjadi
2=32-5.(70-2.32)=1.32-5.70+10.32=11.32-5.70 (vi)
Susun pembagian nomor (i) menjadi
32=312-4.70 (vii)
Sulihkan (vii) ke dalam (vi) menjadi
2=11.32-5.70=11.(312-4.70)-5.70=11.312-49.70

Bilangan x dan y dalam identitas benzout yang didapatkan pada persamaan adalah 11 dan -49.

2. PEMBAHASAN

2.1 Algoritma Euclidean

Konsep Algoritma ini adalah bilangan yang lebih besar (m) dibagi dengan bilangan yang lebih kecil (n). Hasil sisa pembagian m dan n disimpan dalam sebuah bilangan r. Kemudian m diisi dengan nilai n dan n diisi dengan nilai r. Jika hasil sisa pembagian m dan n belum nol maka lakukan terus perulangan pembagian hingga sisanya nol.

1. Jika n=0 maka
   m adalah FPB(m,n);
   stop.
   Tetapi jika n ≠ 0
   Lanjutkan ke langkah 2.
2. Bagilah m dengan n dan misalkan r adalah sisanya.
3. Ganti nilai m dengan nilai n dan nilai n dengan nilai r, lalu ulang kembali ke langkah 1.

Contoh:

FPB(312,70);

312=4.70 + 32
70 = 2.32 + 6
32 = 5.6 + 2
6 = 3.2 + 0

Karena sisa terakhir adalah nol maka FPB dari 312 dan 70 adalah sisa sebelumnya yaitu 2.

FPB(377, 610);

610 =1.377 + 233
377 =1.233 + 144
233 =1.144 + 89
144 =1.89 + 55
89 =1.55 + 34
55 =1.34 + 21
34 =1.21 + 13
21=1.13 + 8
13=1.8 + 5
8 = 1.5 + 3
5 = 1.3 + 2
3 = 1.2 + 1
2 = 1.1 + 1
1 = 1.1 + 0

Karena sisa terakhir adalah nol maka FPB dari 377 dan 610 adalah sisa sebelumnya yaitu 1.

Dalam notasi bahasa C dapat dijelaskan sebagai berikut:

Secara Iteratif:

```
int FPB(int m,int n){

   int r;
   while(n != 0){
         r=m % n;
         m=n;
         n=r;
   }
   return m;

 }
```

Secara Rekursif:

```
int FPB(int m,int n){

  if(n==0){
       return m;
  }
  else{
      FPB(n,m%n);
  }
 }
```

2.1.1 Analisis Kompleksitas Algoritma Euclidean

Kompleksitas dari algoritma mencari FPB ini dapat digambarkan dengan jumlah langkah-langkah algoritma yang dijalankan, dikalikan dengan waktu komputasi dari setiap langkah. Dalam menentukan kompleksitas terburuk dari algoritma ini adalah dengan menggunakan input bilangan fibonaci yang berurutan.maka didapatkan persamaan sebagai berikut:

[n log r = log $r^n \leq$ log Fn ≤ log N => n log N/log r = O(log N)]

Kompleksitas waktu logaritmik ini berarti laju pertumbuhan waktunya berjalan lebih lambat daripada pertumubuhan n.

2.2 Algoritma Stein

Konsep algoritma ini dipakai identitas yang berulang sebagai berikut:

1. gcd(0, *v*) = *v*, karena bilangan apapun membagi 0 dan *v* adalah bilangan terbesar yang membagi *v*. Sama halnya, gcd(*u*, 0) = *u*. gcd(0, 0) tidak dapat didefinisikan.

2. Jika *u* and *v* keduanya genap, maka gcd(*u*, *v*) = 2·gcd(*u*/2, *v*/2), karena 2 adalah pembagi keduanya.

3. Jika *u* genap dan *v* ganjil, maka gcd(*u*, *v*) = gcd(*u*/2, *v*), karena 2 bukan pembagi keduanya. Sama halnya, jika *u* ganjil dan *v* genap, maka gcd(*u*, *v*) = gcd(*u*, *v*/2).

4. Jika *u* dan *v* keduanya ganjil, dan *u* ≥ *v*, maka gcd(*u*, *v*) = gcd((*u*−*v*)/2, *v*). Jika keduanya ganjil dan *u* < *v*, maka gcd(*u*, *v*) = gcd((*v*-*u*)/2, *u*). Ini merupakan kombinasi satu langkah dari algoritma Euclid, yang menggunakan pengurangan setiap langkah, dan sebuah aplikasi dari langkah 3. Pembagian oleh 2 menghasilkan sebuah integer karena perbedaan dari dua angka ganjil adalah genap.

5. Ulangi langkah 3–4 sampai *u* = *v*, (atau satu langkah lagi) sampai *u* = 0. Dalam kasus ini, hasilnya adalah $2^k v$, di mana *k* adalah faktor dari 2 yang ditemukan di langkah 2.

Contoh:

Mencari FPB(1353,1716);

- FPB(1353,1716)
- FPB(1353,1716)
- FPB(1353,858)
- FPB(1353,429)
- FPB(462,429)
- FPB(231,429)
- FPB(231,99)
- FPB(66,99)
- FPB(33,99)
- FPB(33,33)

Maka FPB(1353,1716)=FPB(33,0)=33.

Secara Rekursif:

```
int gcd(unsigned int u, unsigned int
v){
    if(u == v || u == 0 || v == 0)
        return u|v;
    if(u%2 == 0){ // if u is even
        if(v%2 == 0) // if u and v are
even
            return (2*gcd(u/2, v/2));
        else // u is even and v is odd
            return  gcd(u/2, v);
    }
    else if(v%2 == 0) // if u is odd
and v is even
        return gcd(u, v/2);
    else{ // both are odd
        if(u > v)
            return gcd((u-v)/2, v);
        else
            return gcd((v-u)/2, u);
    }
}
```

Secara Iteratif:

```
int FPB(int num1, int num2)
 {
    int pof2, tmp;
    if (!num1 || !num2) {
        return (num1 | num2);
    }
    /* pof2 is the greatest power of 2
deviding both numbers .
    We will use pof2 to multiply the
returning number . */
    pof2 = 0;
    while(!(num1 & 1) && !(num2 & 1)) {
        /* gcd(even1, even1) = pof2 *
gcd(even1/pof2, even2/pof2) */
        num1 >>=1;
        num2 >>=1;
        pof2++;
    }

    do {
        while (!(num1 & 1)) {
            num1 >>=1;
        }
        while (!(num2 & 1)) {
            num2 >>=1;
        }
        /* At this point we know for
sure that
        num1 and num2 are odd */
        if (num1 >= num2) {
            num1 = (num1 - num2) >> 1;
        }
        else {
            tmp = num1;
            num1 = (num2 - num1) >> 1;
            num2 = tmp;
        }
    } while (!(num1 == num2 || num1 ==
0));

    return (num2 << pof2);
 }
```

2.2.1 Analisis Kompleksitas Algoritma Stein

Idenya adalah untuk menerapkan identitas pertama sampai salah satu dari u dan v adalah ganjil. kemudian identitas 2 dan 3 diterapkan, membuat u dan v yang lebih kecil setiap kali. karena salah satu dari u atau v selalu ganjil dari titik ini, kita tidak akan pernah menerapkan identitas pertama lagi. Akhirnya salah satu dari u v menjadi nol, memungkinkan penerapan identitiy terakhir.

Jumlah iterasi dari loop utama adalah $\mathbf{\log_2 (uv)^2 + O(1) = O(n)}$. Setiap iterasi dapat dilakukan dalam waktu O(n), maka total kompleksitas waktu algoritmanya adalah $\mathbf{O(n^2)}$

3. KESIMPULAN

Dari hasil analisis tiap algoritma mencari FPB dapat disimpulkan:

1. Algoritma Stein(FPB Biner) tidak memerlukan perhitungan yang kompleks dibanding algoritma Euclidean. Jadi, perkalian dan pembagian dalam algoritma euclidean diganti dengan shift dalam hal ini,sehingga algoritma Stein 60% lebih efisien dibanding rata-rata pemakaian algoritma Euclidean.

2. Kompleksitas waktu untuk kasus terburuk pada algoritma Euclidean yang dilakukan secara iteratif adalah O(log n).

3. Kompleksitas waktu untuk kasus terburuk pada algoritma Stein yang dilakukan secara iteratif adalah $O(n^2)$.

4. Algoritma Stein(FPB Biner) tidak dapat dipakai sebagai identitas Bezout / persamaan diophantine linear untuk mencari bilangan x dan y dari suatu persamaan mx+ny=FPB(m,n)

REFERENSI

[1]http://www.math.umass.edu/~dhayes/math597a/ugcd2/ugcd2.html. Diakses tanggal 10 Desember 2011

[2]Wikipedia,http://en.wikipedia.org/wiki/Binary_GCD_algorithm. Diakses tanggal 10 Desember 2011.

[3]Wikipedia,http://en.wikipedia.org/wiki/Euclidean_algorithm. Diakses tanggal 10 Desember 2011.

[4]Munir, Rinaldi. 2003. Diktat Kuliah IF2153 Matematika Diskrit. Program Studi Teknik Informatika, Institut Teknologi Bandung.

[5] Rosen, Kenneth H. , ”Discrete Mathematics and Its Applications Fifth Edition”, McGrawHill, 2003.