Penerapan *Divide and Conquer* dalam Membandingkan Alur Proses 2 *Source Code*

Gregorius Ronny Kaluge / 13508019
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18019@students.if.itb.ac.id, ronny.kaluge@gmail.com

***Abstrak*—Dalam kehidupan sehari-hari mungkin kita sering menggunakan fitur *replace* pada *text editor*. Tidak hanya teks biasa yang bisa kita ubah-ubah, *source code* program pun kadang-kadang bisa diubah sehingga untuk membuat *source code* tampak berbeda, kadang-kadang cukup me-*replace* nama variabel menjadi *string* lain. Source code tersebut akan memiliki alur proses yang sama jika di-*compile* meskipun berbeda *source code*. Makalah ini membahas cara membandingkan dua *source code* sehingga bisa diketahui apakah keduanya memiliki alur proses yang persis sama atau tidak. Metode perbandingan akan dilakukan berdasarkan algoritma *divide and conquer*. Untuk implementasi algoritma ini, perlu digunakan *binary search tree*.**

Kata kunci—*replace, text editor, string, source code, compile, divide and conquer, binary search tree*.

I. PENDAHULUAN

Pada mata kuliah yang berhubungan denga pemrograman, sudah sewajarnya ada banyak tugas pemrograman. Kadang-kadang karena alasan-alasan tertentu, mahasiswa malas mengerjakan tugas dan mengopi pekerjaan teman mereka. Agar tidak terlihat berbeda, mereka biasa me-*replace* nama variabel sehingga program mereka tidak terlihat sama dengan milik teman mereka.

Para dosen dan asisten tentunya tidak ingin kebiasaan buruk ini berlanjut. Mereka sudah menyiapkan hukuman khusus bagi mahasiswa yang mengopi perkerjaan teman mereka. Namun, mengecek program mahasiswa satu per satu tentunya bukan pekerjaan mudah, kecuali jika mahasiswanya hanya sedikit (misalnya < 10). Untuk dapat memudahkan tugas mereka, bisa dibuat sebuah program yang membandingkan 2 buah *source code* sehingga diketahui apakah keduanya sama secara proses atau tidak.

II. DIVIDE AND CONQUER

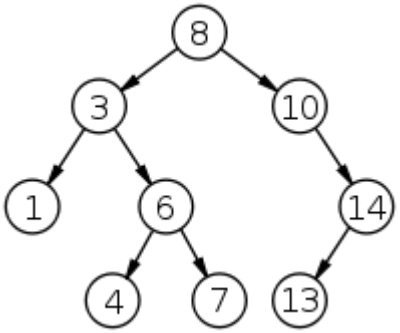
Pada sains komputer, *divide and conquer* adalah paradigma desain algoritma yang penting yang berdasarkan pada rekursif dengan banyak cabang. Algoritma *divide and conquer* bekerja secara rekursif dengan memecah masalah menjadi 2 atau lebih sub masalah dengan tipe yang sama, sampai problem tersebut cukup sederhana untuk langsung diselesaikan. Solusi dari subproblem akan dikombinasikan untuk menghasilkan solusi dari permasalahan yang sebenarnya.

III. BINARY SEARCH TREE

A. Definisi

Pada sains komputer, *binary search tree*(BST) adalah struktur data pohon berbasis *node* yang memiliki sifat-sifat berikut :

- Anak kiri dari sebuah *node* mengandung *key* yang lebih kecil daripada *node* tersebut
- Anak kanan dari sebuah *node* mengandung *key* yang lebih besar daripada *node* tersebut
- Anak kiri dan kanan adalah BST



Gambar 1. Contoh BST dengan ukuran 9

B. Operasi

Ada 4 operasi yang biasa dilakukan pada BST, yaitu *searching*, *insertion*, *deletion*, dan *traversal*. Keempat operasi tersebut memiliki *worst case* O(log n) jika BST seimbang. Namun, jika BST tidak seimbang, *worst case*-nya adalah O(n).

IV. SELF-BALANCING BINARY SEARCH TREE

A. Definisi

Pada sains komputer, *self-balancing binary search tree*

adalah semua BST berbasis *node* yang secara otomatis menjaga ketinggiannya agar seminimal mungkin.

B. Implementasi

Dalam implementasinya, ada beberapa variasi *self-balancing binary search tree*. Struktur data yang populer adalah :

- *AA tree*
- *AVL tree*
- *Red-black Tree*
- *Scapegoat tree*
- *Splay tree*
- *Treap*

C. Aplikasi

Self-balancing binary search tree digunakan untuk membuat *priority queue*, *map* di C++, *dictionary* di C#, dsb. *Binary Search Tree* juga bisa digunakan untuk algoritma *Heap Sort* yang memiliki kompleksitas O(n log n). Pada *self-balacing* BST, *searching*, *insertion*, *deletion*, dan *traversal* bisa dilakukan dengan kompleksitas O(log n).

V. ALGORITMA UNTUK MEMBANDINGKAN

Untuk membandingkan dua *file source code*, dapat digunakan algoritma sebagai berikut :

1. Siapkan sebuah variabel kamus kosong. Variabel ini adalah BST yang *node*-nya berisi pasangan nilai <string1, string2> untuk mengetahui bahwa setiap kemunculan string1 di *file* akan diterjemahkan menjadi string2.
2. Buka *file* pertama, baca *string* baris per baris
3. Untuk setiap *string* yang dibaca per baris, lakukan iterasi dari posisi awal sampai akhir baris menggunakan variabel j
4. Gunakan sebuah variabel, misalnya i yang diinisialisasi dengan 0
5. Jika posisi saat ini adalah tanda baca atau spasi, maka ambil *substring* posisi i .. j-1. Jika *substring* tersebut tidak ada di kamus, tambahkan *substring* tersebut (pastikan string pasangannya pada BST unik). Selanjutnya, tuliskan terjemahan dari *substring* ke *file temporary*, lalu ubah nilai i menjadi j+1
6. Jika posisi saat ini adalah angka dan i = j (artinya karakter pertama kata adalah angka), lakukan penambahan nilai j hingga j mencapai akhir *string* atau karakter ke-j adalah tanda baca atau spasi. Ambil *substring* posisi i .. j-1 dan tuliskan ke *file temporary*.
7. Jika di akhir baris, nilai j > i, ambil substring posisi i .. j-1. Lakukan pengecekan seperti langkah 4.
8. Lakukan langkah 1..7 untuk file kedua dan hasil konversi akan dimasukkan ke *file temporary* lain.
9. Bandingkan isi *file temporary* pertama dan kedua, jika sama berarti kedua *source code* memiliki alur yang sama. Jika berbeda, kemungkinan besar kedua *source code* memiliki alur yang berbeda.

VI. CONTOH SOURCE CODE

Berikut adalah contoh *source code* dalam bahasa C++

```
#include <iostream>
#include <fstream>
#include <map>
#include <vector>
using namespace std;

#define MAX 255

char nama1[MAX], nama2[MAX];
FILE *f, *f2;

void cocokkan (char * nama, char * nama2)
{
	char temp[MAX], temp2[MAX];
	bool cocok = true;
	f = fopen(nama, "r");
	f2 = fopen(nama2, "r");

	while ((fscanf(f, "%s", temp)!= EOF)
&& cocok)
	{
		if(fscanf(f2,"%s",temp2)==EOF)
			cocok = false;
		else
			cocok = strcmp(temp,
temp2) == 0;
	}
	if (cocok)
	{
		if(fscanf(f2,"%s",temp2)==EOF)
			printf("sama\n");
		else
			printf("sepertinya tidak
sama\n");
	}
	else printf("sepertinya tidak
sama\n");
	fclose(f2);
	fclose(f);
}

bool cocok(char c)
{
	return ((c == ' ') || (c == '\t') ||
	(c == '+') || (c == '-') ||
	(c == '*') || (c == '/') ||
	(c == '\\') ||(c == ':') ||
	(c == ',') || (c == ';') ||
	(c == '"') || (c == '\'') ||
	(c == '>') || (c == '<') ||
	(c == '=') || (c == '.') ||
	(c == '&') || (c == '!') ||
	(c == '(') || (c == ')') ||
	(c == '[') || (c == ']') ||
	(c == '{') || (c == '}') ||
	(c== '#') || (c == '?') ||
	(c == '%') || (c == '$') ||
	(c == '^')); //daftar tanda baca dan
	operator yang perlu di skip
}
```

```
void convert(char * nama, char *
nama2)
{
  string temp, temp2;
  map <string, int> a;
  vector<string> s;
  char tempc[100];
  f = fopen(nama2, "w");
  ifstream input( nama , ifstream::in
);
  int i, j, n;
  char c;

  while (getline(input, temp))
  {
    n = temp.length();
    j = 0;
    for (i=0; i<n; ++i)
    {
      c = temp[i];
      if (cocok(c))
      {
        if (j < i)
        {
          temp2 = temp.substr(j, i-j);
          j = a[temp2];
          if (!j) //udah ada
          {
            j = s.size();
            a[temp2] = j;
            if (j > 0)
            {
              itoa(j, tempc, 10);
              s.push_back ("a" +
(string)tempc);
            }
            else s.push_back ("a0");
          }
          fprintf(f, "%s ",
s[j].c_str());
        }
        if ((c!= ' ') && (c!='\t'))
          fprintf(f, "%c", c);
        j = i+1;
      }
      else if ((c <= '9') && (c>='0')
&& (i==j))
      {
        do
          ++i;
        while ((i < n) &&
(!cocok(temp[i])));

        temp2 = temp.substr(j,i-
j).c_str();
        fprintf(f, "%s ",
temp2.c_str());
        j = i+1;
      }
    }
    if (j < i)
    {
      fprintf(f, "%s ", temp.substr(j,
i-j).c_str());
      cout << temp.substr(j, i-j) <<
"...\n";
    }
  }
  input.close();
  fclose(f);
}
```

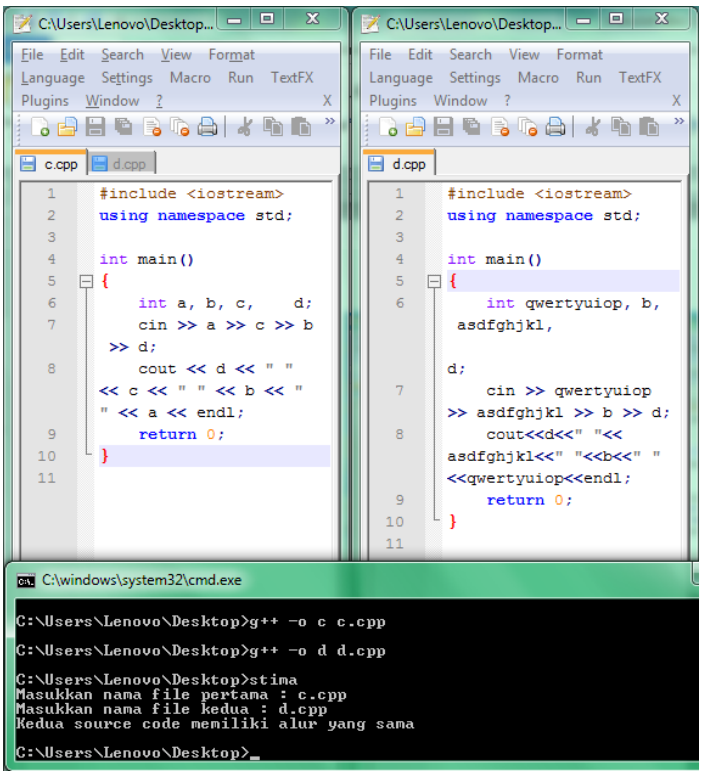
```
int main()
{
  cout << "Masukkan nama file pertama
: ";
  scanf("%s",nama1);

  cout << "Masukkan nama file kedua :
";
  scanf("%s",nama2);

  convert(nama1, "temp1.txt");
  convert(nama2, "temp2.txt");

  cocokkan ("temp1.txt", "temp2.txt");
  return 0;
}
```

Pada *source code* tersebut ada beberapa asumsi yang saya gunakan, yaitu :

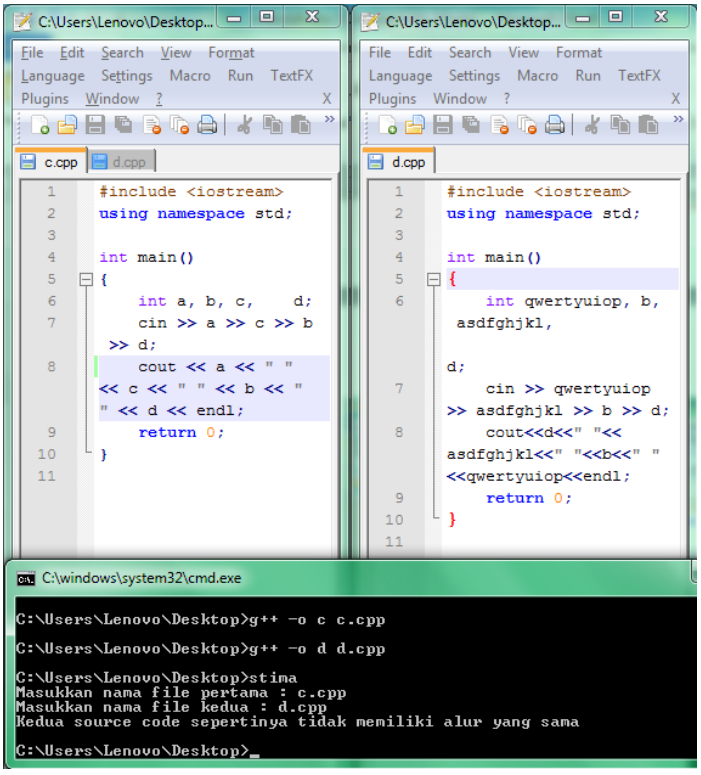
1. Tidak ada komentar pada kedua *source code*, kalaupun ada, keduanya sama persis
2. Kedua *source code* lolos *compile*, benar secara sintaks
3. Pada kasus di mana salah satu *source code* adalah hasil *replace* dari *source code* kedua:
    a. *Replace* dilakukan hanya pada variabel atau fungsi buatan sendiri, per kata dan 1 kata di-replace menjadi 1 kata, tanpa tanda baca tambahan (misalnya (), [], {}, dsb).
    b. Proses *replace* tidak mengubah suatu kata menjadi kata lain yang sudah ada pada *source code* yang sama. Misalnya, me-*replace* int menjadi main pada *source code* C
    c. Tidak ada *file* eksternal yang digunakan pada *source code*

VII. PENGUJIAN

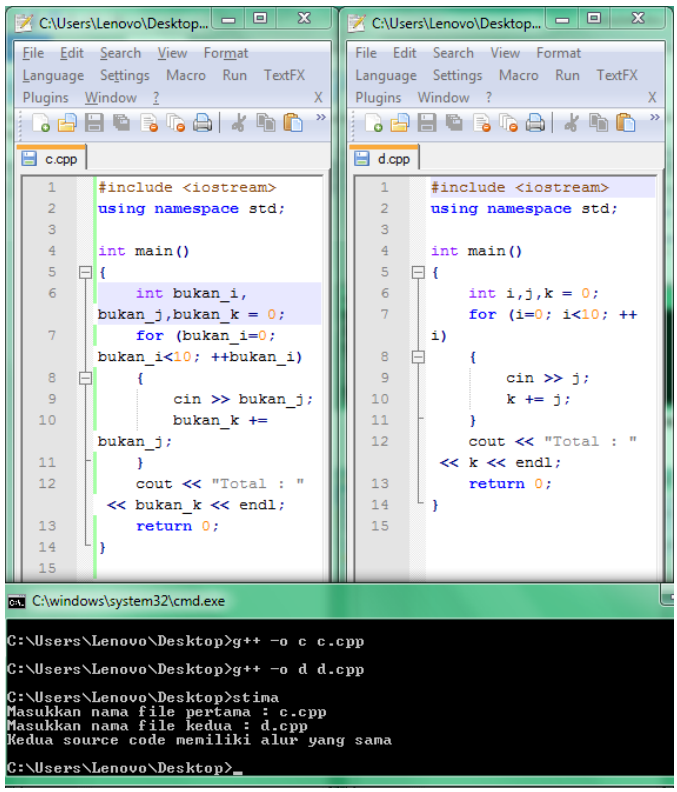
Dari *source code* yang telah dibuat, dilakukan 3 pengujian sebagai berikut:



Gambar 2. Pengujian dengan 2 *source code* yang sama



Gambar 3. Pengujian dengan 2 *source code* yang berbeda



Gambar 4. Pengujian dengan 2 *source code* yang sama

VIII. ANALISIS

Pembuktian kebenaran algoritma ini adalah sebagai berikut :

Jika ada 2 *source code* di mana salah satu *source code* adalah hasil *replace source code* yang satunya (dan memenuhi asumsi di bab 6), urutan deklarasi variabelnya akan sama, begitu pula dengan pemanggilan fungsi , prosedur, dan semua operasi. Bisa dipastikan pula jumlah katanya masih sama, tidak ada penghapusan maupun penambahan kata.

Lalu kita mengganti semua kata (selain operator, tanda baca, dan angka) yang pertama kali kita temukan dengan sebuah *string*, misalnya "a1 ". Lalu semua kata yang kedua kalinya kita temukan dan belum diganti dengan *string* lain, misalnya "a2 ", begitu seterusnya sampai tidak ada lagi kata yang belum diganti. Setiap operator, angka, dan tanda baca yang ditemukan akan langsung dituliskan tanpa dikonversi menjadi *string* lain. Contoh :

Source code :

```
#include <iostream>

int main()
{
        int a1;
        a1 = 1 << 1;
        return 0;
}
```

Akan berubah menjadi :

```
#a0 <a1 >a2 a3 (){a2 a4 ;a4 =1 <<1 a5 0 }
```

Karena urutan deklarasi variabel, prosedur, dan fungsinya sama, variabel yang pertama kali dideklarasikan di *source code* pertama pasti akan menjadi sama dengan variabel yang pertama kali dideklarasikan di *source code* kedua. Hal itu berlaku pula pada prosedur dan fungsi pada kedua *source code*. Akibatnya, jika memang salah satu *source code* dibuat dengan me-*replace* kata-kata dari *source code* yang satunya dan memenuhi asumsi, hasil konversinya akan sama.

Jumlah proses yang dilakukan oleh algoritma ini adalah $n_1 \log k_1 + n_2 \log k_2 + s_1 + s_2$. $n_1 \log k_1$ adalah proses yang dibutuhkan untuk mengonversi semua kata pada *source code* pertama. Ada n_1 kata pada *source code* dengan k_1 variasi, artinya pada kasus terburuk kita harus mencari n_1 kali dan baru memperoleh nilai pada ujung BST. Karena BST yang digunakan sudah seimbang, jumlah ketinggiannya untuk k_1 elemen adalah $\log k_1$ sehingga *traversal* pada *worst case* membutuhkan $\log k_1$ proses. s_1+s_2 proses dihasilkan dari proses perbandingan kedua file hasil konversi. s_1 menyatakan ukuran *file* pertama dan s_2 menyatakan ukuran *file* kedua.

IX. KESIMPULAN

Dari analisis pada bab VIII diperoleh kesimpulan bahwa kompleksitas algoritma ini adalah O(n log k + s). Dengan kompleksitas tersebut, algoritma ini tergolong algoritma *Polynomial*. Untuk pengembangan selanjutnya, perlu dicari algoritma yang lebih umum lagi sehingga bisa mengurangi asumsi yang dibutuhkan.

REFERENSI

[1] http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm
diakses tanggal 8 Desember 2010

[2] http://en.wikipedia.org/wiki/Binary_search_tree
diakses tanggal 8 Desember 2010

[3] http://en.wikipedia.org/wiki/Self-balancing_binary_search_tree
diakses tanggal 7 Desember 2010

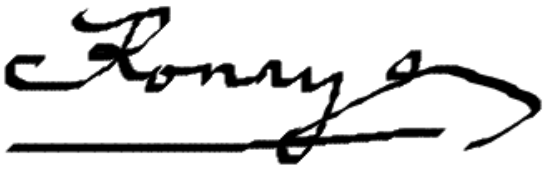
[4] http://www.cplusplus.com/reference/stl/map/
diakses tanggal 8 Desember 2010

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, 29 April 2010

ttd



Gregorius Ronny Kaluge / 13508019