Aplikasi Strategi Algoritma dalam Pembagian Kelompok Tugas Besar

Jan Wira Gotama Putra
13512015
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
wira.gotama@s.itb.ac.id

***Abstract*—Pengerjaan tugas secara berkelompok merupakan hal yang lumrah bagi mahasiswa, tetapi tidak selalu pembagian kelompok mahasiswa tersebut mempunyai kemampuan yang merata. Untuk melakukan pembagian kelompok yang adil maka diperlukan suatu cara agar bisa mendapatkan konfigurasi kelompok mahasiswa yang berkemampuan sama. Saya mengaplikasikan ilmu yang didapat pada kuliah strategi algoritma untuk menciptakan metode pembagian kelompok yang merata.**

***Index Terms*— clustering method, exhaustive search kombinatorial.**

I. PENGENALAN

Insitut Teknologi Bandung adalah sebuah institusi dengan tugas yang banyak. Tugas-tugas di Institut Teknologi Bandung sering dikerjakan dalam format berkelompok, tetapi pembagian kemampuan anggota kelompok sering tidak merata.

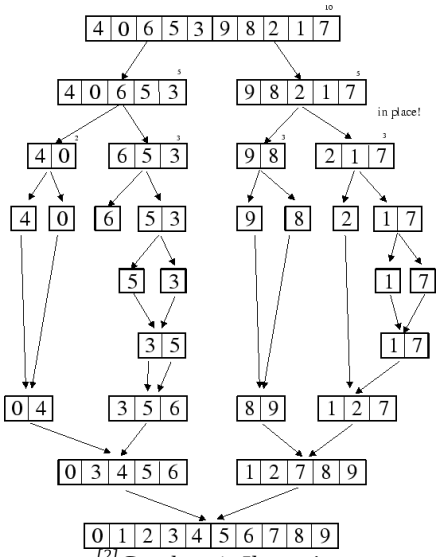
Salah dua topik matakuliah "Strategi Algoritma" adalah *sorting(devide and conquer)* dan *exhaustive search.* Makalah ini akan mengaplikasikan algoritma *sorting(merge sort* dan *quick sort* karena merupakan metode pengurutan tercepat) dan *exhaustive search* untuk menghasilkan pembagian kelompok tugas yang cukup merata tiap kelompoknya.

II. TEORI TERKAIT

1. Algoritma Merge Sort

[1]Merge sort adalah sebuah algoritma pengurutan yang bersifat mudah dibagi tetapi sulit digabung. Algoritma ini mempunyai kompleksitas waktu O(n log n). Berikut adalah tahapan algoritma Merge sort :

1. Larik akan dibagi menjadi 2 bagian kiri dan kanan masing-masing berukuran n/2.
2. Secara rekursif, terapkan algoritma merge sort untuk masing-masing bagian.
3. Gabungkan hasil pengurutan kedua bagian tersebut sehingga diperoleh tabel baru yang terurut.

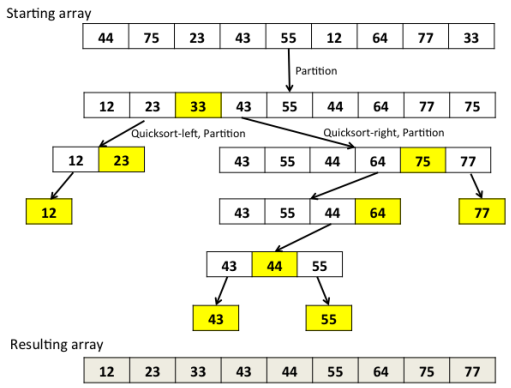


[2]*Gambar 1. Ilustasi merge sort*

2. Algoritma Quick Sort

[1]Quick sort adalah sebuah algoritma pengurutan yang bersifat sudah dibagi tetapi mudah digabung. Suatu larik dibagi menjadi 2 bagian (partisi) A1 dan A2, sehingga setiap elemen pada bagian A1 <= setiap elemen A2. Algoritma ini mempunyai kompleksitas waktu O(n log n). Berikut tahapan algoritma quick sort

1. p adalah elemen terkiri dari tabel, q adalah elemen terkanan dari tabel
2. Pilih elemen dengan indeks tengah sebagai pivot
3. Pindai tabel kiri sampai ditemui A[p] >= pivot
4. Pindai tabel kanan sampai ditemukan A[q] <= pivot
5. Tukarkan A[p] dan A[q]
6. Ulangi langkah dari posisi p+1 dan q-1 sampai p>q
7. Lakukan quicksort untuk bagan kiri tabel (1 sampai q) dan bagian kanan (q+1 sampai akhir tabel)



[3]*Gambar 2. Ilustrasi quick sort.*

3. Algoritma Exhausive Search

Exhaustive Search adalah metode untuk mencari solusi dengan mengenumerasi semua kemungkinan solusi. Exhaustive search biasanya digunakan untuk memecahkan persoalan yang terkait dengan kombinatorial. *Exhaustive Search* tidak sama dengan *brute force* yang lempeng, *Exhaustive Search* pada umumnya tidak mengikutsertakan elemen yang sudah pernah dicoba ke enumerasi berikutnya. Algoritma ini mempunyai kompleksitas O(n!) untuk persoalan kombinatorial.

III. PEMBAHASAN

Pembagian kelompok tugas besar yang cocok dilakukan dengan membuat daftar kunci mahasiswa beserta nilai (nilai bisa saja berupa nilai UTS, IP atau nilai apapun). Kunci mahasiswa berupa NIM, nilai adalah kriteria yang diacu untuk menentukan kemampuan seseorang. Pembagian kelompok yang dihasilkan berupa kelompok-kelompok dengan kemampuan masing-masing kelompok relatif sama. Persoalan ini dapat kita identifikasi merupakan sebuah persoalan dengan jenis kombinatorial, kita dapat memecahkan persoalan ini menggunakan algoritma *exhaustive search*.

Berikut adalah algoritma pembagian kelompok :

1. Urutkan kemampuan mahasiswa berdasarkan nilai.
2. Hitung rata-rata nilai mahasiswa.
3. Tentukan jumlah anggota kelompok (m) .
4. Bagi mahasiswa peta kemampuan menjadi matrix dua dimensi dengan jumlah baris n. Tujuan dari pembuatan matrix ini adalah memetakan kemampuan mahasiswa menjadi beberapa *cluster* kemampuan yang hampir sejajar.
5. Bila jumlah mahasiswa mod m > 0 maka keluarkan m mahasiswa dengan kemampuan menengah dari matrix (terdekat dengan nilai rata-rata). Nantinya mahasiswa dengan kemampuan menengah tersebut akan dimasukkan ke kelompok-kelompok yang sudah dihasilkan.
6. Ambil satu mahasiswa pada baris paling awal, lakukan enumerasi kelompok yang mungkin untuknya.
7. Kelompok yang dimasukkan pada solusi adalah kelompok dengan kemampuan rata-rata paling mendekati rata-rata nilai mahasiswa total.
8. Mahasiswa yang sudah masuk pada kelompok tidak akan diikutsertakan pada enumerasi kelompok berikutnya.
9. Ulangi mulai langkah 5 sampai semua kelompok terbentuk.
10. Masukan mahasiswa dengan kemampuan menengah yang dikeluarkan dari matrix (bila ada) dimulai dari kelompok enumerasi yang rata-rata kemampuannya paling kecil.

Ilustrasi

daftar mahasiswa beserta kemampuan

NIM	nilai
1	100
2	98
3	92
4	86
5	85
6	78
7	72
8	60
9	52
10	48

Rata-rata nilai = 77.2

Kita ingin membentuk kelompok dengan anggota kelompok 3 orang. 10 mod 3 = 1 (artinya ada 1 orang yang dipisahkan terlebih dahulu). Mahasiswa dengan nilai 78 dipisahkan terlebih dahulu karena mempunyai nilai terdekat dengan rata-rata nilai. Matrix yang terbentuk sebagai berikut :

```
100 98 92
86 85 72
60 52 48
```

Mahasiswa yang berada pada baris yang sama dianggap berada pada *level* kemampuan yang sama. Data mahasiswa perlu diurutkan terlebih dahulu agar bisa dibuat matrix seperti ini.

Penentuan kelompok pertama beserta kemampuan

(100,86,60) : 82
(100,86,52) : 79.333
(100,52,48) : 66.67
(100,85,60) : 80.67
(100,85,52) : 79
(100,85,48) : 77.67
(100,72,60) : 77.33 (kelompok terpilih sebagai solusi)
(100,72,52) : 74.67
(100,72,48) : 73.33

Mahasiswa dengan nilai 100, 72 dan 60 terbentuk sebagai 1 kelompok, sehingga masing-masing anggotanya tidak akan diikutsertakan pada penenumerasian kelompok berikutnya. Kemudian kita lanjutkan mengenumerasi kelompok-kelompok berikutnya :

(98,85,52) : 78.33
(98,85,48) : 77
(98,86,52) : 78.67
(98,86,48) : 77.33

(92,85,48) : 75

Mahasiswa dengan nilai 78 yang dikeluarkan dari matrix tadi akan masuk ke dalam kelompok (92.85,48) menjadi (92,85,48,78) dengan rata-rata kemampuan 75.75. Dengan demikian rata-rata kemampuan semua kelompok adalah 77.33, 77.33 dan 75.75 (kemampuan masing-masing kelompok relatif sama).

IV. IMPLEMENTASI DAN ANALISIS

PSEUDOCODE

```
Program Utama;
Kamus
        /*data adalah struktur komposit yang terdiri dari
        NIM dan nilai */
        A, non_matrix : array of data;
        M : matrix of pair;
        i, anggota_kel, x : integer;
        solusi, temp : vector;
        median : real;
        n : integer; //jumlah mahasiswa
Begin
        AmbilDataNilai(A);
        Sort(A);
        Input(anggota_kel);
        buatMatrix(A, M, anggota_kel, non_matrix);
        median <- rataRataNilai(A);
        for i<-1 to n do begin
                temp.clear(); /* Mengosongkan vector
                                temp */
                generateKombinasi(M, i,1,temp
                ,anggota_kel);
                x <- closestToMedian(temp,median);
                solusi <- solusi Union temp[x];
        end;
        insertNonMatrix(non_matrix, anggota, solusi);
        Output(solusi);
End.
```

```
Function rataRataNilai(Input A : array of pair) : real;
/* Mencari rata-rata nilai mahasiswa */
Kamus Lokal
        j : integer;
        sum : real;
Begin
        For j <- 1 to A.size() do begin
                sum <- sum + A[j].nilai;
        End;
        return sum/A.size();
End;
```

```
Function closestToMedian(Input temp : vector, input
median : real) : integer;
/* Mencari indeks temp yang paling dekat dengan nilai
rata-rata mahasiswa */
Kamus Lokal
        beda, minimum : real; idx : integer;
Begin
        idx <- 0; minimum <- 1000; //sebuah nilai besar
        For j<-1 to temp.size() do begin
                beda <- absolut(temp[j].nilai – median);
                If (beda<minimum) then begin
                        minimum <- beda;
                        idx <- j;
                End;
        End;
End;
```

```
Procedure buatMatrix(Input A : array of pair, Output
M : matrix of pair, input jumlah_anggota_kelompok :
integer, output non_matrix : array of pair);
/* matrix dari array A dibuat*/
Kamus lokal
        m, p, x : integer;
Begin
        m <- A.size()/jumlah_anggota_kelompok;
        x <- A.size() mod jumlah_anggota_kelompok
        /* Masukkan anggota array A dengan nilai
menengah  sebanyak x ke dalam non_matrix, remove dari
A */
        //buat matrix M
        P <- 1;
        For i<-1 to jumlah_anggota_kelompok do begin
                For j<-1 to m do begin
                        M[i][j] <- A[p];
                        p <- p+1;
                End;
        End;
End;
```

```
Procedure generateKombinasi(Input matrix M, input
index : integer, input count : integer, output temp :
vector, input jumlah_anggota : integer);
/* kombinasi kelompok bagi mahasiswa pada matrix
M[0][index] dibuat */
Begin
        If (count>jumlah_anggota) then berhenti;
        Else begin
          If (count==1) then temp <- temp Union
          M[1][indeks];
            For i<-1 to kolom_matrix(M) do begin
                If (BelumPernahDipilih(M[count][i])
                begin
                   temp <- temp Union M[count][i];
                   generateKombinasi(M, indeks,
                   count+1, temp);
                end;
             End;
        End;
```

End;

Procedure AmbilDataNilai(Output A : array of pair);
/* Data Mahasiswa diinputkan, kemudian dimasukkan ke dalam A. Data terdiri dari NIM dan nilai */

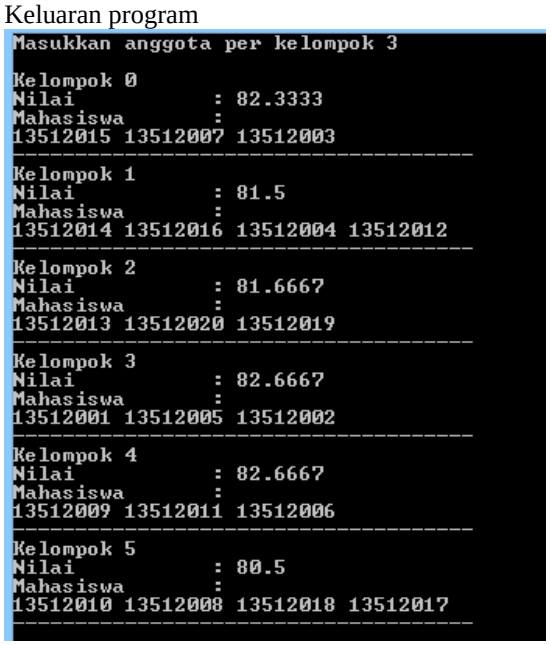
Procedure Sort(Input/Output A : array of pair);
/* Data mahasiswa diurutkan berdasarkan nilai. Pengurutan bisa menggunakan algoritma quicksort ataupun mergesort */

Procedure insertNonMatrix(input non_matrix : array of data, input anggota : integer, input/output solusi : vector);
/* Semua elemen non_matrix dimasukkan ke dalam elemen solusi yang cocok (dimulai dari elemen solusi yang mempunyai nilai terkecil */
kamus lokal
flag : array of boolean;
i, idx, j : integer
min : double
Begin
//flag berikuran sama dengan baris*kolom matrix
for i<-0 to non_matrix.size() begin
min = 1000;
idx = 0;
for j=0 to solution.size() begin
if (solution[j].nilai < min and !flag[j]) begin
min = solution[j].nilai;
idx = j;
end;
end;
solution[idx].nilai = ((solution[idx].nilai * anggota) + non_matrix[i].nilai) / (anggota+1))
solution <- U non_matrix[i]
flag[idx] = true;
end;
End;

Data Uji Mahasiswa 1

```
20
13512001 88
13512002 76
13512003 71
13512004 73
13512005 84
13512006 76
13512007 77
13512008 78
13512009 87
13512010 86
13512011 85
13512012 81
13512013 90
13512014 92
13512015 99
13512016 80
13512017 82
13512018 76
13512019 72
13512020 83
```

Keluaran program

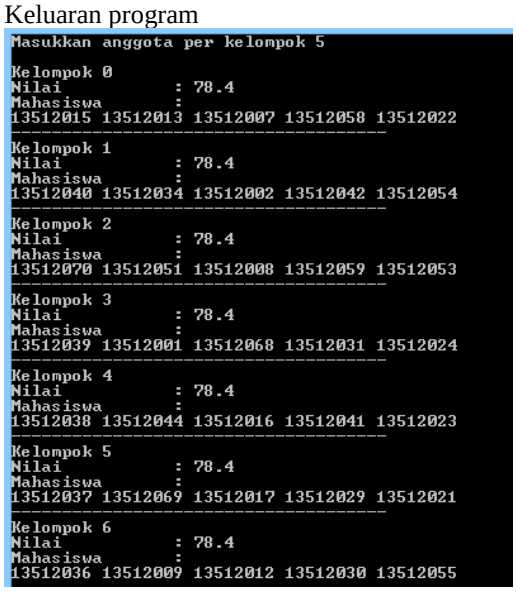


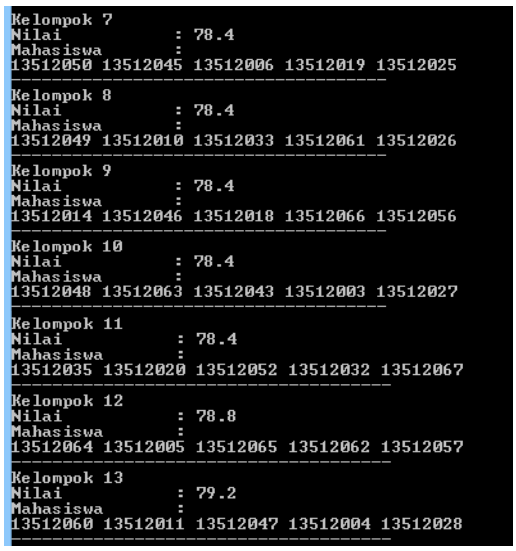
Data Uji Mahasiswa 2

```
70
13512001 88
13512002 76
13512003 71
13512004 73
13512005 84
13512006 76
13512007 77
13512008 78
13512009 87
13512010 86
13512011 85
13512012 81
13512013 90
13512014 92
13512015 99
13512016 80
13512017 82
13512018 76
13512019 72
13512020 83
13512021 61
13512022 60
13512023 61
13512024 62
13512025 63
13512026 64
13512027 65
13512028 66
13512029 67
13512030 68
13512031 69
13512032 71
13512033 80
13512034 90
```

```
13512035 92
13512036 94
13512037 95
13512038 96
13512039 97
13512040 98
13512041 67
13512042 68
13512043 81
13512044 88
13512045 87
13512046 86
13512047 82
13512048 92
13512049 93
13512050 94
13512051 89
13512052 81
13512053 60
13512054 60
13512055 62
13512056 64
13512057 65
13512058 66
13512059 67
13512060 90
13512061 69
13512062 72
13512063 83
13512064 91
13512065 82
13512066 74
13512067 65
13512068 76
13512069 87
13512070 98
```

Keluaran program





ANALISIS

Misal jumlah data yang ada sebanyak **n**. Dari segi waktu pengurutan, program mempunyai kompleksitas waktu O(n log n). Untuk tiap anggota kelompok berjumlah **m** tiap kelompoknya, maka proses pengelompokan mahasiswa membutuhkan kompleksitas O((n/m)!) , hal tersebut menandakan proses yang cukup lambat.

Keberjalanan implementasi menggunakan quicksort lebih cepat pada persoalan ini, karena *quicksort* hanya membutuhkan 1 *array*, sementara *mergesort* memerlukan 2 *array* (1 sebagai input, 1 sebagai hasil), keperluan 2 *array* tersebut menanggung kompleksitas waktu yang cukup banyak seiring dengan perkembangan jumlah data.

V. KESIMPULAN

Dari keluaran program berdasarkan data uji, dapat saya simpulkan bahwa metode pembagian kelompok ini dapat membagi mahasiswa menjadi kelompok-kelompok yang tiap kelompoknya mempunyai kemampuan yang sama relatif satu terhadap lainnya. Metode ini tidak terlalu cepat, tetapi menghasilkan solusi yang bagus.

REFRENSI

[1] Munir, Rinaldi. 2009. Diktat Kuliah IF 2210 Strategi Algoritma 4th ed. Bandung : Program Studi Teknik Informatika STEI ITB.

[2] Munir, Rinaldi. 2014. Slide Kuliah Devide and Conquer.

[3] http://www.cs.swarthmore.edu/~brody/cs35/s14/Labs/images/07/quickSort.png

[4] http://en.wikipedia.org/wiki/Sorting_algorithm

CATATAN TAMBAHAN

Source code dari methode yang telah diimplementasikan dapat diunduh pada pranala berikut ini :
https://www.dropbox.com/s/d1jyk2g2mobulzz/source.cpp

Contoh input data mahasiswa dapat diunduh pada pranala berikut ini :
https://www.dropbox.com/s/q4skpwozf8tben7/data.txt

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 Mei 2014

Jan Wira Gotama Putra
13512015