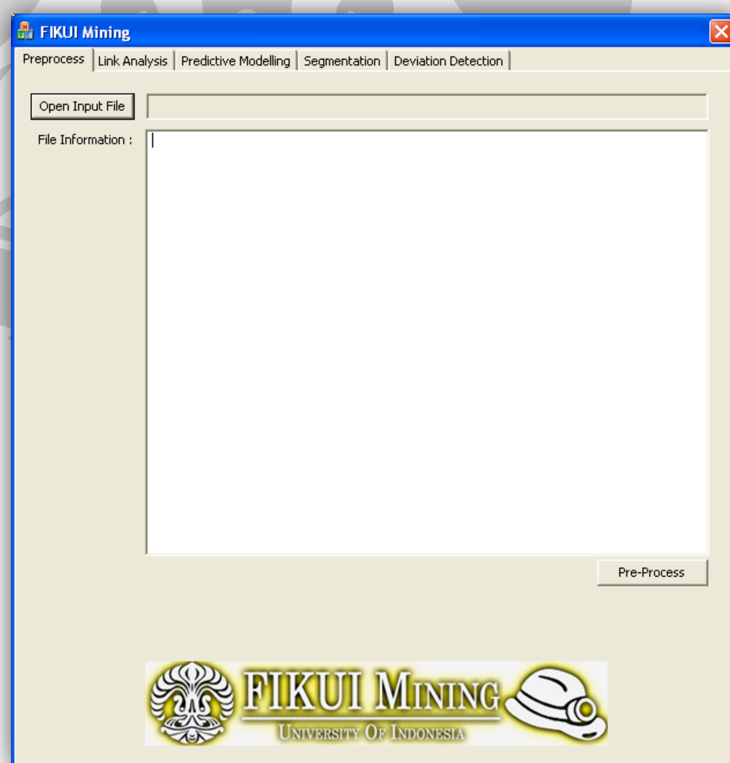
BAB III ANALISIS DAN PERANCANGAN

Bab ini berisi penjelasan tentang analisis kebutuhan dan perancangan perangkat lunak hasil implementasi. Analisis kebutuhan meliputi daftar fitur yang diperlukan untuk dibuat pada tahap implementasi. Perancangan perangkat lunak meliputi pembuatan desain *input*, *class*, *function* dan *output*.

3.1 ANALISIS KEBUTUHAN

Perangkat lunak hasil penerapan yang dibuat oleh penulis akan menjadi salah satu bagian dari *FIKUI Mining* **[RAS08]**. Berikut adalah tampilan versi *FIKUI Mining* yang digunakan oleh penulis dalam Tugas Akhir ini.



Gambar 3.1 – *FIKUI Mining*

Pada *FIKUI Mining* telah terdapat beberapa fitur yang digunakan oleh semua algoritma yang diimplementasikan di dalamnya seperti fitur *preprocessing*, fitur membuka *input file*, fitur tampilan untuk memasukkan nilai input, dan fitur untuk menyimpan hasil eksekusi dalam wujud *output file* **[RAS08]**.

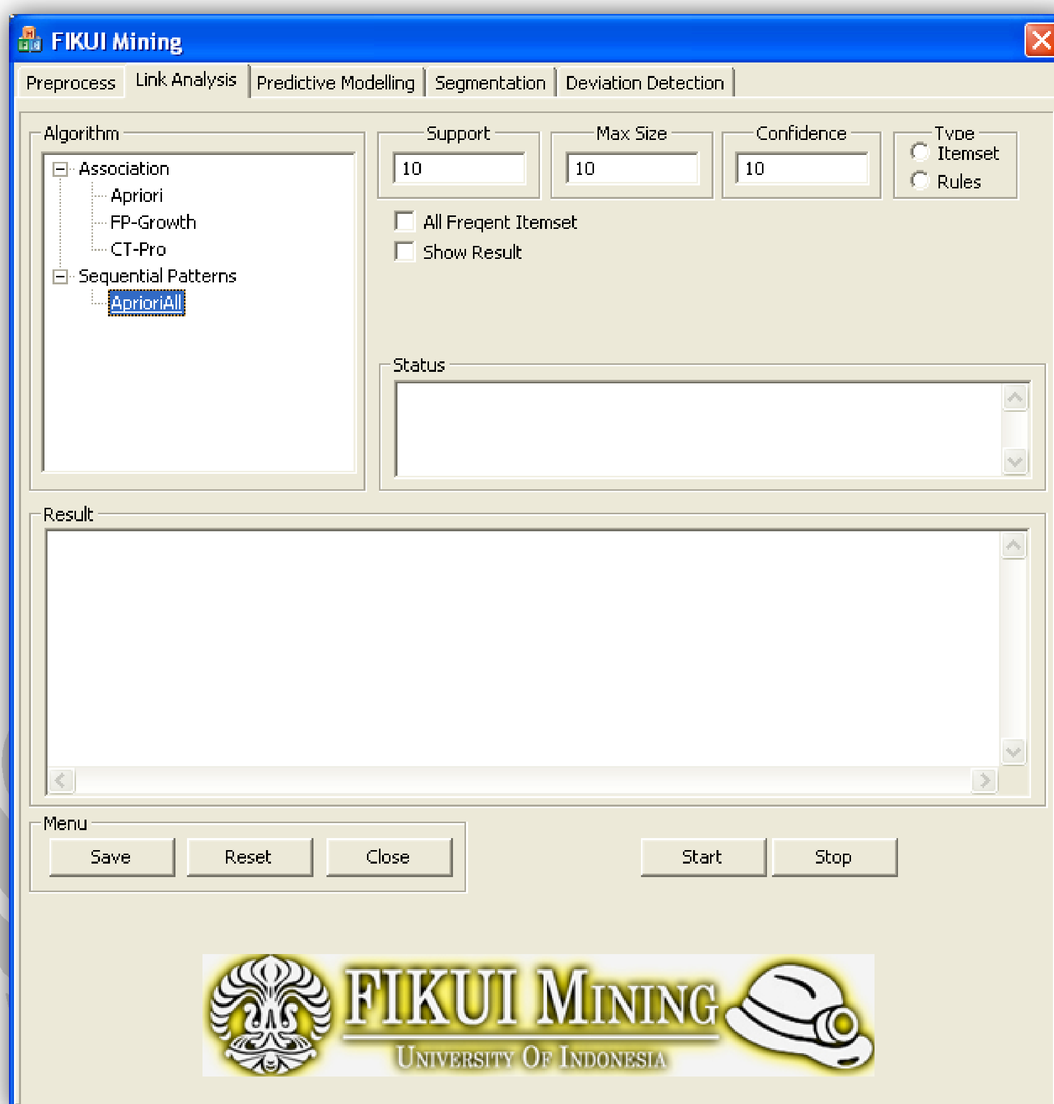
Dari konsep yang telah dijelaskan pada bab sebelumnya, terdapat beberapa fitur yang dibutuhkan untuk mecari pola sekuensial dengan menggunakan algoritma *AprioriAll* yaitu:

1. Fitur untuk membuka *input file*.
2. Fitur untuk memasukkan nilai *support*.
3. Fitur untuk menerapkan *Mining Sequential Patterns* (MSP).
4. Fitur untuk menampilkan *output*.
5. Fitur untuk menyimpan *output* dalam wujud *output file*.

Semua fitur kecuali fitur untuk menerapakan MSP, telah tersedia pada FIKUI *Mining*. Oleh karena itu penulis hanya perlu berfokus pada pembuatan semua *class* beserta semua *function* yang mendukungnya.

Selain itu, penulis juga perlu mempersiapkan semua *dataset* yang diperlukan untuk menguji hasil implementasi. Penjelasan mengenai perancangan dan pembuatan dataset dijelaskan pada bab V tentang hasil pengujian.

Pencarian pola..., Fandi, FASILKOM UI, 2009



Gambar 3.2 – Submenu *Sequential Patterns*

Pada gambar di atas, dapat dilihat telah terdapat *input form* dan *control* yang dimiliki *FIKUI Mining* pada bagian *Link Analysis* seperti *Algorithm*, *Support*, *Max Size*, *Confidence*, *Type*, *Status*, *Result*, dan *Menu*. Namun, karena pencari pola sekuensial hanya membutuhkan dua *input* yaitu *input file* dan nilai *support*, maka *input form* lainnya tidak dibutuhkan.

3.2 PERANCANGAN

Untuk melakukan implementasi *Mining Sequential Patterns* (MSP), penulis harus merancang format *input*, *MSP class*, *MSP_itemset class* dan format *output*.

3.2.1 Perancangan *Input*

Sesuai dengan konsep yang digunakan pada bab sebelumnya, *database* terdiri dari *customer ID*, *transaction time* dan *item* yang dibeli. Berikut ini adalah contoh struktur *input file* yang digunakan untuk merepresentasikan *database* pada contoh tabel 2.9.

```
@data
1|30
1|90
2|10 20
2|30
2|40 60 70
3|30 50 70
4|30
4|40 70
4|90
5|90
```

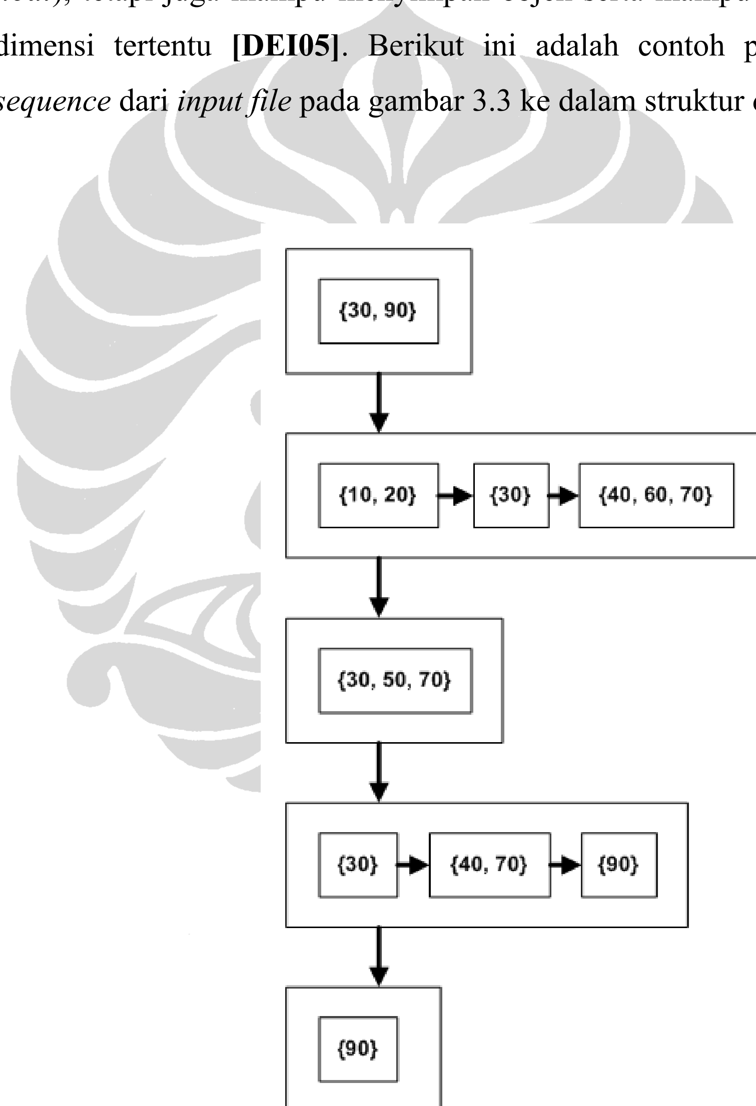
Gambar 3.3 – Tampilan *Input*

Pada baris pertama ditandai dengan **@data** mengikuti format *input file* yang dimiliki oleh FIKUI *Mining*. Setiap baris merepresentasikan transaksi yang terurut berdasarkan *transaction time*. Pada masing-masing baris, karakter pertama merupakan *customer ID* kemudian diikuti oleh tanda *pipe* " | ", kemudian diikuti dengan *item* yang dibeli pada transaksi tersebut dan dipisahkan oleh spasi. *Customer ID* diwakili dengan bilangan bulat berurutan mulai dari 1, 2, 3, …, dan seterusnya. Baris terakhir merupakan baris kosong sebagai tanda berakhirnya *input file*.

Pencarian pola..., Fandi, FASILKOM UI, 2009

3.2.2 Perancangan Struktur Data

Mengingat jumlah transaksi yang terdapat pada database bisa banyak dan berbeda-beda antara satu *input file* dengan yang lain, maka dibutuhkan struktur data yang bisa menyimpan pertambahan informasi secara dinamis. Oleh karena itu, penulis memilih struktur data *vector*. Pada dasarnya struktur data *vector* merupakan *array* yang tidak perlu didefinisikan panjangnya karena panjang *vector* bertambah secara dinamis sesuai dengan pertambahan elemen. Selain itu, struktur data *vector* tidak hanya mampu menyimpan jenis data primitif (contoh *int*, *double*, *float*), tetapi juga mampu menyimpan objek serta mampu dideklarasikan dengan dimensi tertentu **[DEI05]**. Berikut ini adalah contoh penyimpanan *customer sequence* dari *input file* pada gambar 3.3 ke dalam struktur data *vector*.

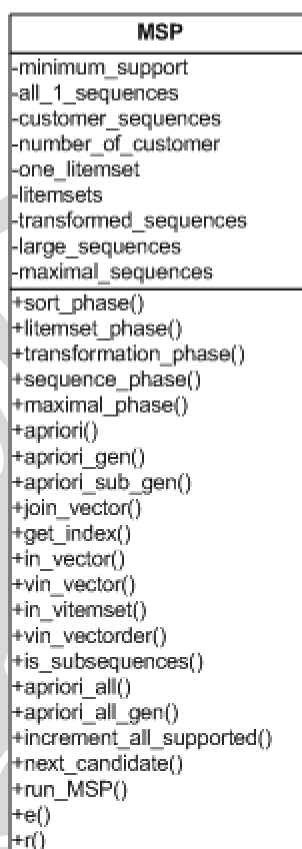


Gambar 3.4 – Struktur Data *Vector*

Pencarian pola..., Fandi, FASILKOM UI, 2009

3.2.3 Perancangan *MSP Class*

MSP class merupakan *class* utama hasil penerapan yang dilakukan oleh penulis pada Tugas Akhir ini. Semua proses pencarian pola sekuensial terjadi pada *class* ini. Berikut ini adalah *class diagram* untuk *MSP class*.



Gambar 3.5 – *MSP Class Diagram*

Dari *class diagram* di atas, terdapat beberapa variabel yang menyimpan hasil utama dari proses yang terjadi pada *class* tersebut tersebut yaitu:

- **`minimum_support`**, menyimpan nilai *minimum support*.
- **`all_1_sequences`**, menyimpan semua *1-sequence*.
- **`customer_sequences`**, menyimpan semua *customer sequence*.
- **`number_of_customer`**, menyimpan jumlah *customer*.

- `one_litemset`, menyimpan *large itemset* yang terdiri dari satu *item*.
- `litemsets`, menyimpan semua *large itemset*.
- `transformed_sequences`, menyimpan *transformed sequence* dalam representasi *mapping number*.
- `large_sequence`, menyimpan semua *large sequence*.
- `maximal_sequence`, menyimpan hasil akhir proses MSP berupa *maximal sequence*.

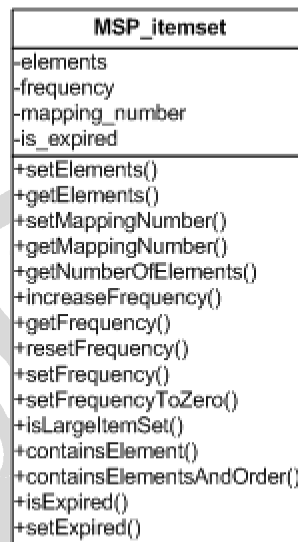
Tabel berikut ini adalah deskripsi kegunaan masing-masing *function* yang terdapat pada *MSP class*.

Function	Deskripsi
`sort_phase`	implementasi dari *sort phase*
`litemset_phase`	implementasi dari *litemset phase*
`transformation_phase`	implementasi dari *transformation phase*
`sequence_phase`	implementasi dari *sequence phase*
`maximal_phase`	implementasi dari *maximal phase*
`apriori`	implementasi dari algoritma *Apriori*
`apriori_gen`	implementasi dari *candodate generation*
`apriori_sub_gen`	implementasi dari *sub-candidate generation*
`join_vector`	menggabungkan isi dari dua *vector*
`get_index`	mendapatkan *index* dari sebuah *vector*
`in_vector`	mencari elemen pada sebuah *vector*
`vin_vector`	mencari *vector* 1 dimensi pada *vector* 2 dimensi
`in_vitemset`	mencari objek *MSP_itemset* pada sebuah *vector*
`vin_vectorder`	mencocokan urutan *vector* pada *vector* yang lain
`is_subsequences`	mencari elemen *subsequence* dari sebuah *vector*
`apriori_all`	implementasi dari algoritma *AprioriAll*
`apriori_all_gen`	implementasi *candidate generation* untuk *AprioriAll*
`increment_all_supported`	menyaring *candidate* yang *frequent*
`next_candidate`	memeriksa ukuran *candidate* berikutnya
`run_MSP`	menjalankan MSP
`e`	mencetak *output* pada satu baris yang sama
`r`	mencetak *output* dan pindah baris

Tabel 3.1 – Deskripsi *MSP Class*

3.2.4 Perancangan *MSP_itemset Class*

MSP_itemset class merupakan *class* yang merepresentasikan *itemset* beserta semua artibut dan operasi yang diperlukan. Berikut ini adalah *class diagram* untuk *MSP_itemset class*.



Gambar 3.6 – *MSP_itemset Class Diagram*

Dari class diagram di atas, terdapat beberapa variabel yaitu:

- `elements`, yang menyimpan semua *item* anggota dari *MSP_itemset*.
- `frequency`, jumlah kemunculan *MSP_itemset* ini terhadap *customer*.
- `mapping_number`, bilang bulat yang mewakili *MSP_itemset* ini.
- `is_expired`, tanda yang menujukkan sudah atau belumnya *MSP_itemset* ini terdaftar pada satu baris *customer*.

Pencarian pola..., Fandi, FASILKOM UI, 2009

Tabel berikut ini adalah deskripsi kegunaan masing-masing *function* yang terdapat pada *MSP_itemset class*.

Function	Deskripsi
`setElements`	memasukkan elemen ke *MSP_itemset* ini
`getElements`	mendapatkan elemen
`setMappingNumber`	menentukan *mapping number*
`getMappingNumber`	mendapatkan *mapping number*
`getNumberOfElements`	mendapatkan banyaknya jumlah *item* pada elemen
`increaseFrequency`	menambah nilai frekuensi dengan satu
`getFrequency`	mendapatkan nilai frekuensi
`resetFrequency`	mengubah nilai frekuensi menjadi satu
`setFrequency`	menentukan nilai frekuensi
`setFrequencyToZero`	mengubah nilai frekuensi menjadi nol
`isLargeItemSet`	memeriksa apakah frekuensi ≥ *minimum support*
`containsElement`	memeriksa apa sebuah *item* termasuk elemen
`containsElementsAndOrder`	memeriksa keberadaan dan urutan *itemset* dalam elemen
`isExpired`	mendapatkan status *is_expired*
`setExpired`	menentukan nilai *is_expired*

Tabel 3.2 – Deskripsi *MSP_itemset*

3.2.5 Perancangan *Output*

Output yang akan ditampilkan pada jendela *output* pada FIKUI Mining adalah *output* utama setiap fase yang dilalui pada *Mining Sequential Patterns* (MSP). Berikut ini adalah tampilan *output* tesebut.

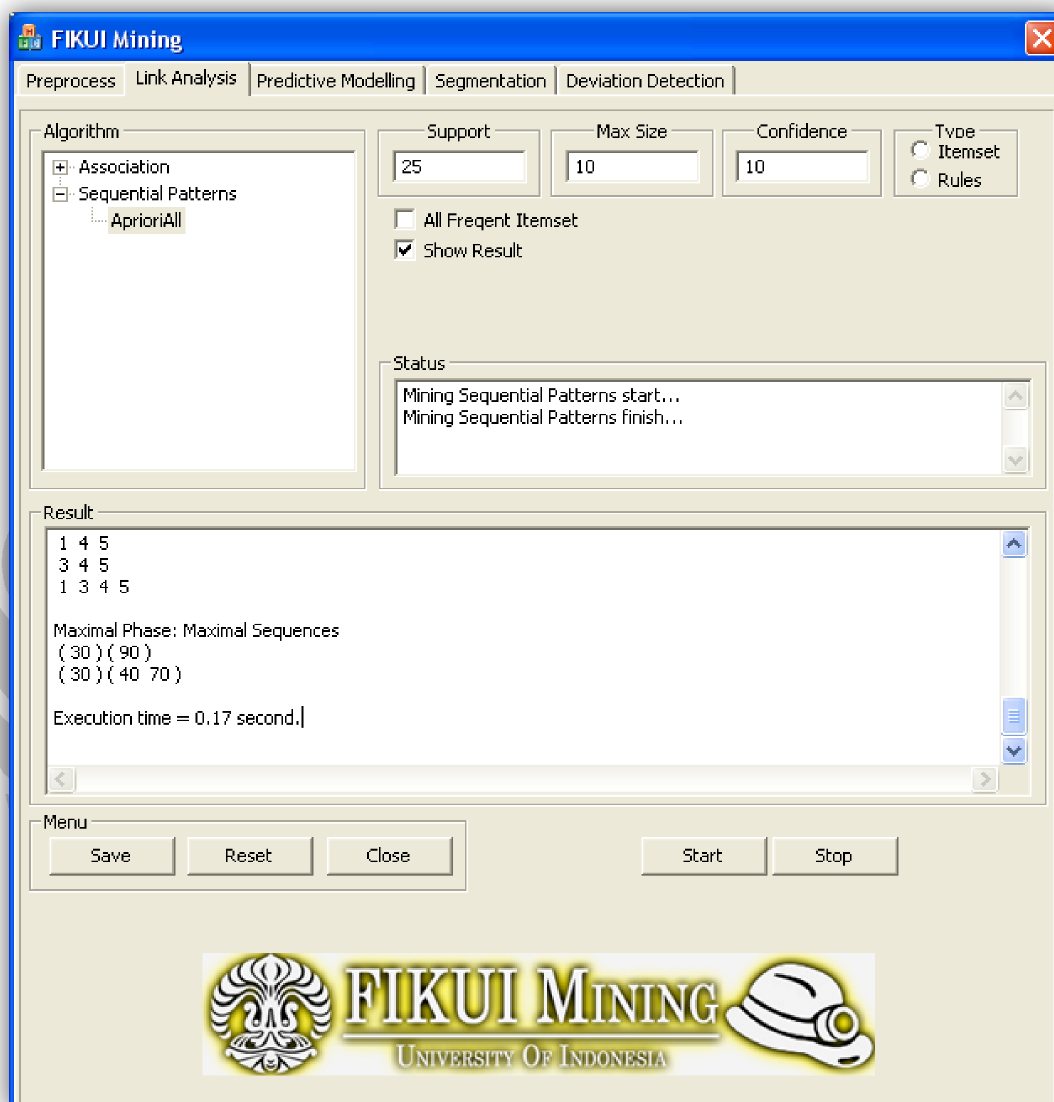
```
Number of customer   = 5
Minimum support      = 2.00 customer

Sort Phase: Customer Sequence
1 -> (30) (90)
2 -> (10 20) (30) (40 60 70)
3 -> (30 50 70)
4 -> (30) (40 70) (90)
5 -> (90)

Litemsets Phase: Litemsets
(30)    -> 1
(40)    -> 2
(70)    -> 3
(40 70) -> 4
(90)    -> 5

Transformation Phase: Transformed Sequence
1 -> {1} {5}
2 -> {1} {2 3 4}
3 -> {1 3}
4 -> {1} {2 3 4} {5}
5 -> {5}

Sequence Phase: Large Sequence
1  5
1  2
1  3
1  4
2  3
2  4
3  4
1  2  3
1  2  4
1  3  4
2  3  4
1  2  3  4

Maximal Phase: Maximal Sequence
(30) (90)
(30) (40 70)

Execution time = 0.16 second.
```

Gambar 3.7 – Tampilan *Output*

Dari gambar 3.7, berikut ini adalah penjelasan untuk masing-masing *output* utama yang ditampilan, yaitu:

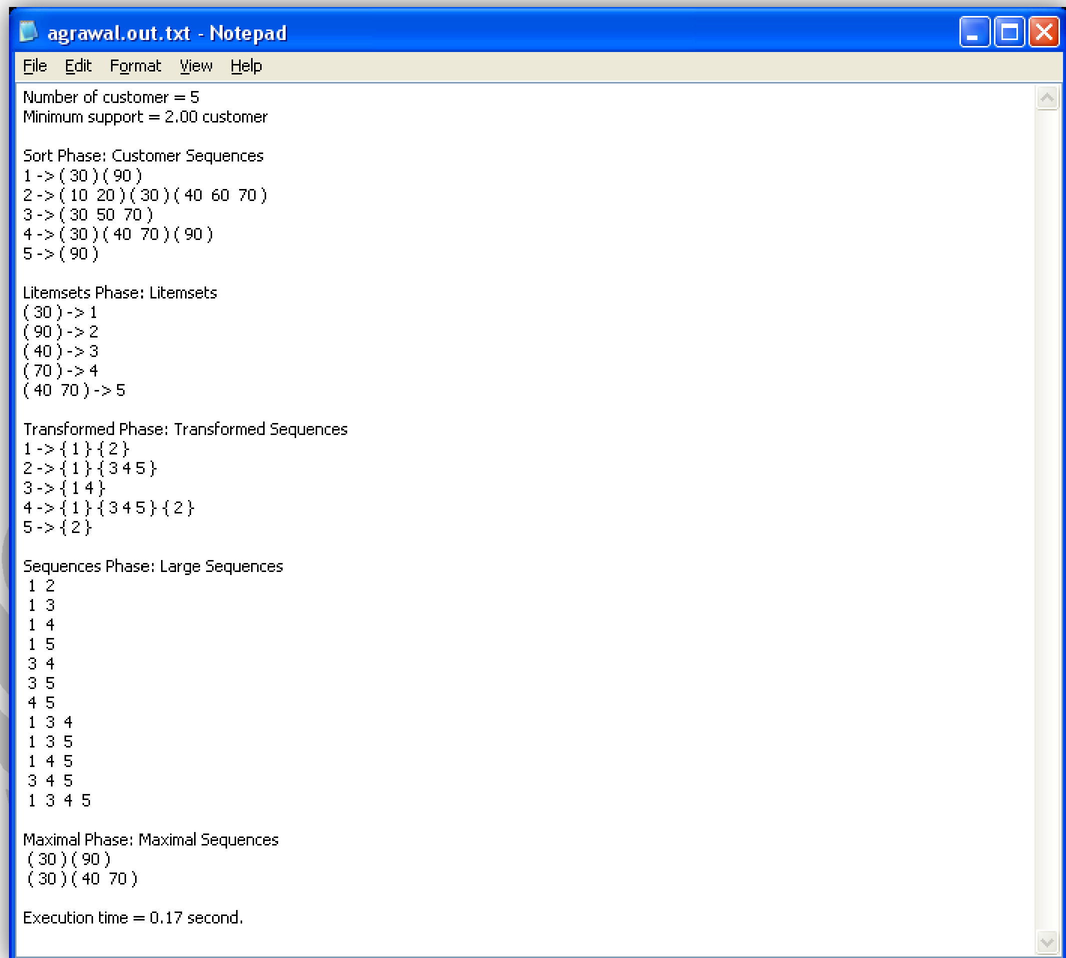
- ***Number of customer***, menampilkan jumlah *customer* pada *database*.
- ***Minimum support***, menampilkan hasil perkalian nilai *support* yang ditentukan oleh pengguna dengan jumlah *customer*.
- ***Sort Phase: Customer Sequence***, menampilkan *customer sequence* pada masing-masing *customer* (diwakili dengan bilangan bulat).
- ***Litemset Phase: Litemsets***, menampilkan semua *large itemset* (*litemset*) yang dipetakan dengan bilangan bulat berurutan (*mapping number*).
- ***Transformation Phase: Transformed Sequence***, menampilkan *transformed sequence* dalam representasi *mapping number*.
- ***Sequence Phase: Large Sequence***, menampilkan *large sequence* dalam representasi *mapping number*.
- ***Maximal Phase: Maximal Sequence***, menampilkan *maximal sequence* sebagai hasil akhir dari implementasi.
- ***Execution time***, menampilkan total waktu ekskusi program mulai dari dijalankannya *sort phase*.

Berikut ini adalah *screenshot* yang menampilan *output* pada bagian *result FIKUI Mining*.



Gambar 3.8 – Jendela *Result*

Berikut ini adalah tampilan isi dari *output file* yang disimpan menggunakan *FIKUI Mining*.



```
agrawal.out.txt - Notepad
File  Edit  Format  View  Help

Number of customer = 5
Minimum support = 2.00 customer

Sort Phase: Customer Sequences
1 -> ( 30 ) ( 90 )
2 -> ( 10  20 ) ( 30 ) ( 40  60  70 )
3 -> ( 30  50  70 )
4 -> ( 30 ) ( 40  70 ) ( 90 )
5 -> ( 90 )

Litemsets Phase: Litemsets
( 30 ) -> 1
( 90 ) -> 2
( 40 ) -> 3
( 70 ) -> 4
( 40  70 ) -> 5

Transformed Phase: Transformed Sequences
1 -> { 1 } { 2 }
2 -> { 1 } { 3 4 5 }
3 -> { 1 4 }
4 -> { 1 } { 3 4 5 } { 2 }
5 -> { 2 }

Sequences Phase: Large Sequences
 1  2
 1  3
 1  4
 1  5
 3  4
 3  5
 4  5
 1  3  4
 1  3  5
 1  4  5
 3  4  5
 1  3  4  5

Maximal Phase: Maximal Sequences
 ( 30 ) ( 90 )
 ( 30 ) ( 40  70 )

Execution time = 0.17 second.
```

Gambar 3.9 – *Output File*

BAB IV IMPLEMENTASI

Bab ini berisi penjelasan hasil penerapan algoritma *AprioriAll* untuk mendapatkan pola sekuensial (*sequential patterns*) melalui lima fase sesuai dengan konsep yang telah dijelaskan pada bab sebelumnya. Hasil penerapan akan dijelaskan dengan menggunakan *pseudocode* yang merepresentasikan cara kerja *source code* hasil penerapan. Proses penerapan ini disebut *Mining Sequential Patterns* (MSP).

Penulis melakukan implementasi *Mining Sequential Patterns* (MSP) dalam bahasa pemograman C++ dengan menggunakan Microsoft Visual Studio 2005. Struktur data yang digunakan dalam implementasi adalah ***vector***, yang merupakan bagian dari *Standard Template Library* (STL) dalam bahasa pemograman C++ **[DEI05]**.

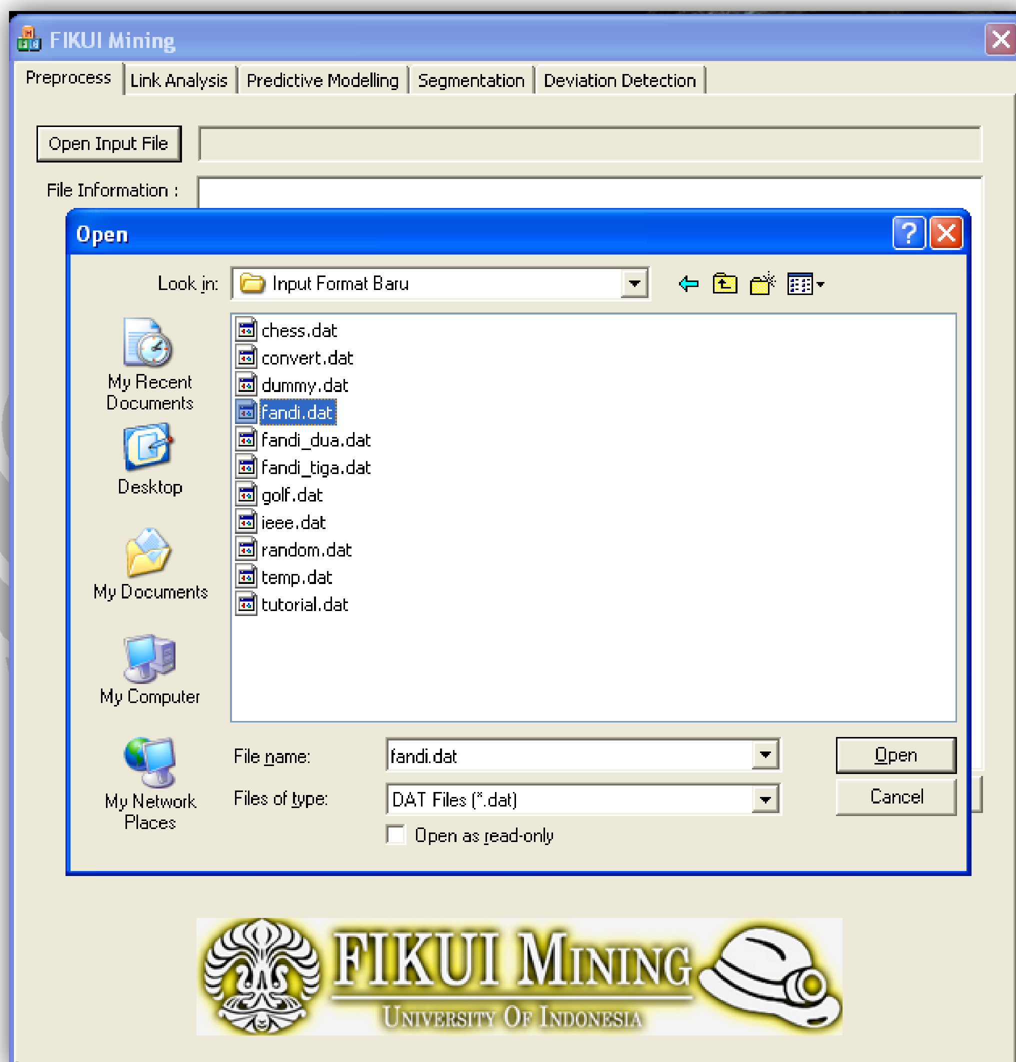
Implementasi *Mining Sequential Patterns* (MSP) diwujudkan dalam *MSP class*. Setiap fase dalam MSP diimplementasikan dalam wujud *function* dari *MSP class*. Selain itu, penulis juga membuat beberapa *function* tambahan yang dibutuhkan untuk membantu proses MSP seperti *function apriori*, *apriori_all* dan lainnya.

Implementasi objek *itemset* diwujudkan dalam *MSP_itemset class*. Implementasi ini diperlukan karena setiap *itemset* memiliki informasi yang selalu dibawanya seperti daftar elemen anggotanya (*item*), frekuensi, nomor pemetaan (*mapping number*) dan status *expired*.

Penulis melakukan implementasi *source code* melalui pemahaman konsep MSP dan panduan dari hasil implementasi algoritma *AprioriSome* untuk mencari pola sekuensial dalam bahasa PHP dan *database* MySQL yang telah dibuat oleh Pawel Rendak dan Lukasz Barcikowski **[REN09]**.

4.1 PEMBACAAN *INPUT*

Tahap pertama dari implementasi MSP adalah pembacaan input. Dengan menggunakan fitur pembacaan *input file* yang telah tersedia dalam FIKUI *Mining* pada *tab Preprocess*, maka *input file* siap diproses.



Gambar 4.1 – Membuka *Input File*

Selain *input file*, dibutuhkan nilai *support* yang ditentukan oleh pengguna. Nilai support dimasukkan pada *Link Analysis* → *Support* dan dengan memilih *Algorithm* → *Sequential Patterns* → *AprioriAll*. Sedangkan bagian yang lain seperti *Max Size*, *Confidence*, *Type*, *All Frequent Itemset*, dan *Show Result* tidak digunakan pada MSP.

Berikut ini adalah *pseudocode* pembacaan input.

```
foreach line from input_file
{
     tokenizing line by delimiter "|";
     added second_token to customer_database[first_token];
}
```

Gambar 4.2 – *Pseudocode* Pembacaan Input

Dengan menggunakan contoh tabel 2.9 (lihat di halaman 16) dan nilai *support* yang diinginkan 25% maka diperoleh `customer_database` sebagai berikut.

Index	Elements
1	<30> <90>
2	<10 20> <30> <40 60 70>
3	<30 50 70>
4	<30> <40 70> <90>
5	<90>

Tabel 4.1 – *customer_database*

Struktur data yang digunakan dalam implementasi adalah *vector*. Pada tabel di atas, merupakan wujud dari *vector* yang terdiri dari lima elemen (ditandai oleh index bernomor 1, 2, 3, 4, 5). Masing-masing elemen terdiri dari *vector* berjenis *string* (ditandai oleh tanda kurung siku < >).

Setiap tabel yang digunakan pada bab ini merupakan representasi struktur data *vector* dalam implementasinya.

4.2 *SORT PHASE*

Setelah pembacaan *input file* selesai dilakukan, diperoleh **customer_database**, kemudian langkah berikutnya adalah mendapatkan dua *output* berikut ini melalui *sort phase*:

- **all_1_sequences** merupakan semua *sequence* yang panjangnya satu atau dianggap sebagai kumpulan masing-masing *item* yang terdapat pada **customer_database**.
- **customer_sequences** diperoleh dengan mengubah masing-masing elemen dari **customer_database** menjadi objek *MSP_itemset*.

Berikut ini adalah *pseudocode* untuk *sort phase*.

```
foreach itemset from customer_database
{
     added itemset to customer_sequences;

     foreach item from itemset
     {
          if(item not contained in all_1_sequences)
          {
               added item to all_1_sequences;
          }
     }
}
```

Gambar 4.3 – *Pseudocode Sort Phase*

Berikut ini adalah *customer sequence* yang dihasilkan.

Customer ID	Customer Sequence
1	(30) (90)
2	(10 20) (30) (40 60 70)
3	(30 50 70)
4	(30) (40 70) (90)
5	(90)

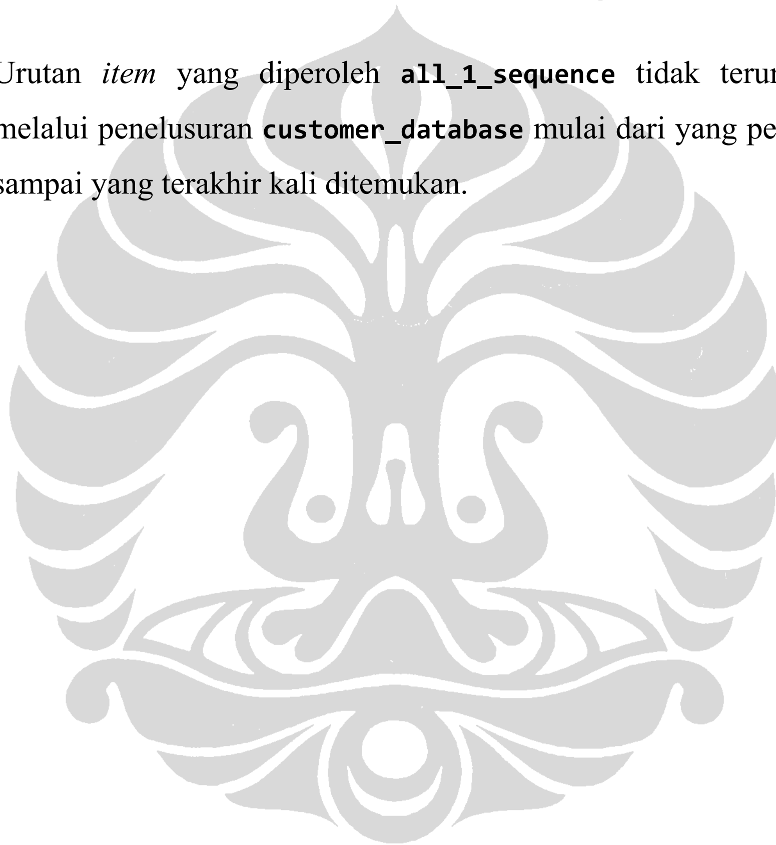
Tabel 4.2 – *customer_sequences*

Berikut ini adalah **`all_1_sequences`** yang dihasilkan.

all_1_sequences
30
90
10
20
40
60
70
50

Tabel 4.3 – *all_1_sequences*

Urutan *item* yang diperoleh **`all_1_sequence`** tidak terurut karena diperoleh melalui penelusuran **`customer_database`** mulai dari yang pertama kali ditemukan sampai yang terakhir kali ditemukan.



4.3 *LITEMSET PHASE*

Dengan membentuk **1_litemsets** dari **all_1_sequences** dan membentuk **transaction_database** dari **customer_sequences**, pada fase ini digunakan algoritma *Apriori* untuk mendapatkan semua *large itemset* (**litemsets**) terhadap seluruh transaksi.

Berikut ini adalah *pseudocode* untuk *litemsets phase*.

```
foreach 1-sequence from all_1_sequences
{
     if(1-sequence.frequency >= minimum_support)
     {
          added 1-sequence to 1_litemsets;
     }
}

foreach transaction from customer_sequences
{
     add transaction to transaction_database;
}

litemsets = apriori(minimum_support, 1_litemsets, transaction_database);
```

Gambar 4.4 – *Pseudocode Litemset Phase*

Berikut ini adalah **litemsets** yang dihasilkan.

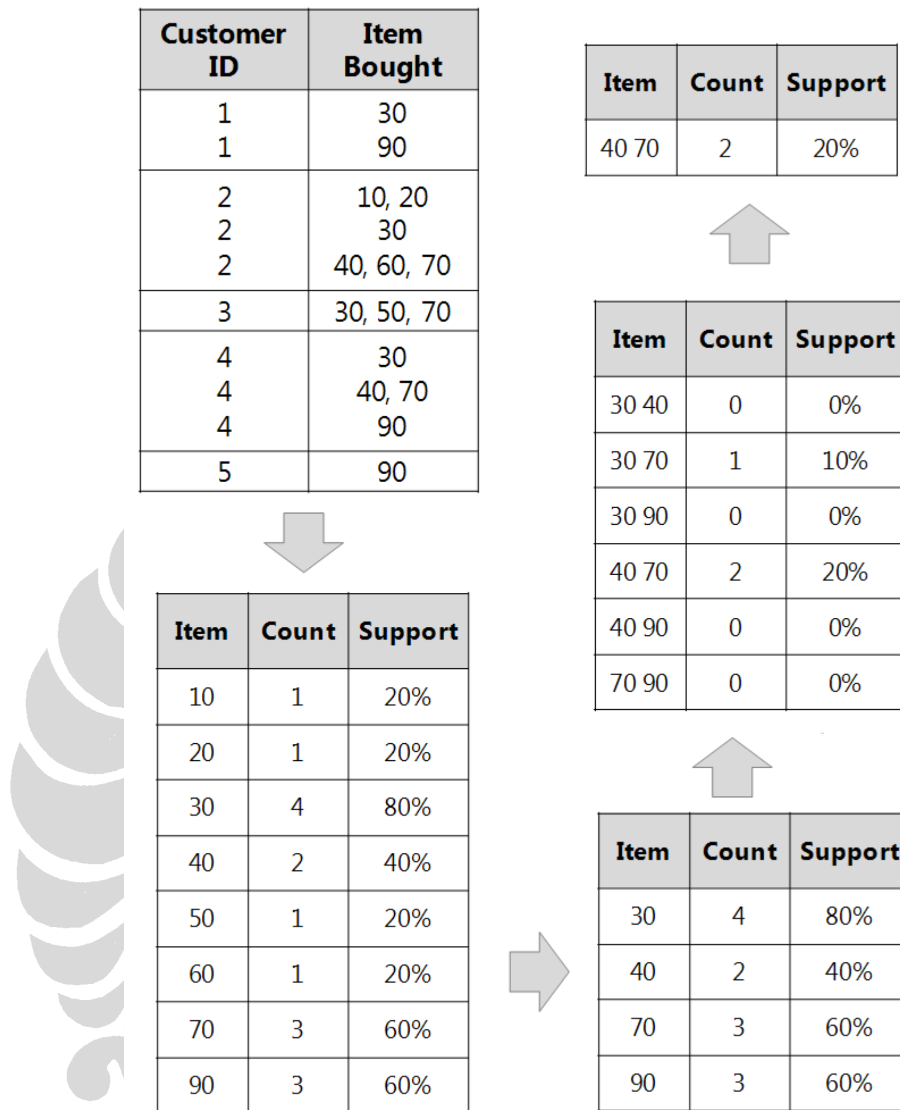
Litemset	Mapping Number
30	1
90	2
40	3
70	4
40 70	5

Tabel 4.4 – *litemsets*

Mapping number diberikan berupa bilangan bulat (*integer*) secara berurutan mulai dari 1, 2 , 3, … dan seterusnya.

Pencarian pola..., Fandi, FASILKOM UI, 2009

Berikut ini adalah tahapan yang terjadi pada saat pengunaan algoritma *Apriori*.



Customer ID	Item Bought
1 1	30 90
2 2 2	10, 20 30 40, 60, 70
3	30, 50, 70
4 4 4	30 40, 70 90
5	90

Item	Count	Support
10	1	20%
20	1	20%
30	4	80%
40	2	40%
50	1	20%
60	1	20%
70	3	60%
90	3	60%

Item	Count	Support
30	4	80%
40	2	40%
70	3	60%
90	3	60%

Item	Count	Support
30 40	0	0%
30 70	1	10%
30 90	0	0%
40 70	2	20%
40 90	0	0%
70 90	0	0%

Item	Count	Support
40 70	2	20%

Gambar 4.5 – Tahapan Algoritma *Apriori*

Dari pembentukan *candidate* yang terdiri dari satu item (**`1_litemsets`**), kemudian dilakukan penghilangan semua *candidate* yang tidak memenuhi **`minimun_support`**. Kemudian hasilnya digunakan untuk membentuk *candidate* yang terdiri dari dua *item*. Proses pembentukan dan penghilangan *candidate* ini dilakukan berulang-ulang sampai diperoleh hanya sisa satu *candidate*. Semua *candidate* yang memenuhi **`minimum_support`** disebut **`litemsets`**.

Pencarian pola..., Fandi, FASILKOM UI, 2009

4.4 *TRANSFORMATION PHASE*

Tujuan fase ini adalah mendapatkan **transformed_sequences** melalui cara mengganti **customer_sequences** dengan *mapping number* yang bersesuaian dengan **litemsets**.

Berikut ini adalah *pseudocode* untuk *transformation phase*.

```
foreach sequence from customer_sequences
{
     foreach mapping_number from litemsets
     {
          result = replace sequence with appropiate mapping_number;
     }
     added result to transformed_sequences;
}
```

Gambar 4.6 – *Pseudocode Transformation Phase*

Berikut ini adalah **transformed_sequences** yang diperoleh.

Customer ID	Transformed Sequence
1	{ 1 } { 2 }
2	{ 1 } { 3 4 5 }
3	{ 1 4 }
4	{ 1 } { 3 4 5 } { 2 }
5	{ 2 }

Tabel 4.5 – *transformed_sequences*

Pencarian pola..., Fandi, FASILKOM UI, 2009

4.5 *SEQUENCE PHASE*

Ketiga fase sebelumnya hanya bertujuan untuk mengubah **`customer_database`** menjadi **`transformed_sequences`** agar lebih efisien diproses menggunakan algoritma *AprioriAll*. Tujuan fase ini adalah mendapatkan **`large_sequences`** yang terdiri dari semua *large sequence* dalam wujud *mapping number*.

Berikut adalah *pseudocode* untuk *sequences phase*.

```
foreach 1_sequence from all_1_sequences
{
     if(1_sequence.frequency >= minimum_support)
     {
          replace 1_sequence with appropiate mapping_number;
          added 1_sequence to large_1_sequences;
     }
}

large_sequences = apriori_all( minimum_support,
                               large_1_sequences,
                               transformed_sequences );
```

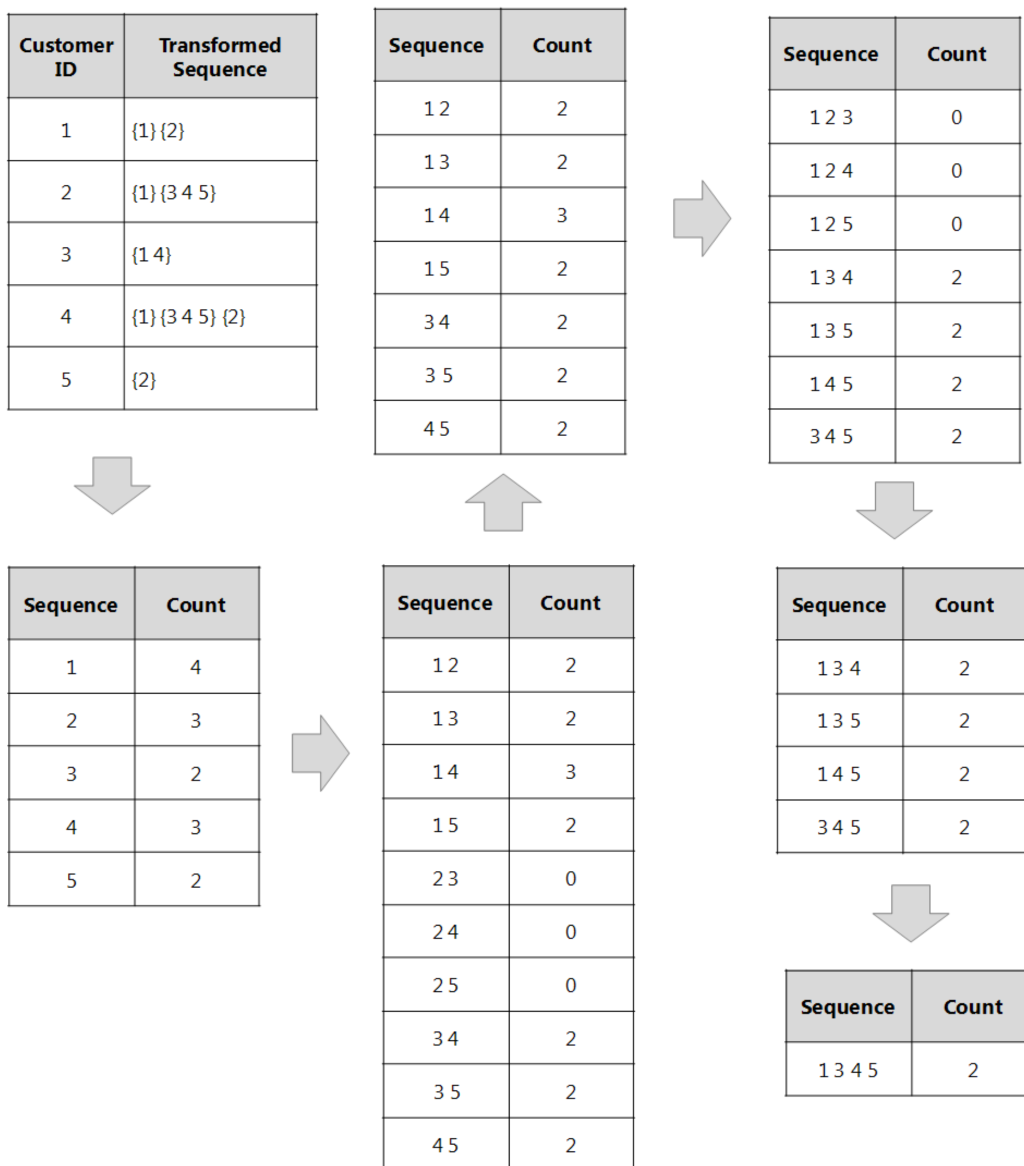
Gambar 4.7 – *Pseudocode Sequence Phase*

Berikut ini adalah **`large_sequences`** yang dihasilkan.

Large Sequence
1 2
1 3
1 4
1 5
3 4
3 5
4 5
1 3 4
1 3 5
1 4 5
3 4 5
1 3 4 5

Tabel 4.6 – *large_Sequences*

Berikut ini adalah tahapan yang terjadi pada pengunaan algoritma *AprioriAll*.



Customer ID	Transformed Sequence
1	{1} {2}
2	{1} {3 4 5}
3	{1 4}
4	{1} {3 4 5} {2}
5	{2}

Sequence	Count
1	4
2	3
3	2
4	3
5	2

Sequence	Count
1 2	2
1 3	2
1 4	3
1 5	2
2 3	0
2 4	0
2 5	0
3 4	2
3 5	2
4 5	2

Sequence	Count
1 2	2
1 3	2
1 4	3
1 5	2
3 4	2
3 5	2
4 5	2

Sequence	Count
1 2 3	0
1 2 4	0
1 2 5	0
1 3 4	2
1 3 5	2
1 4 5	2
3 4 5	2

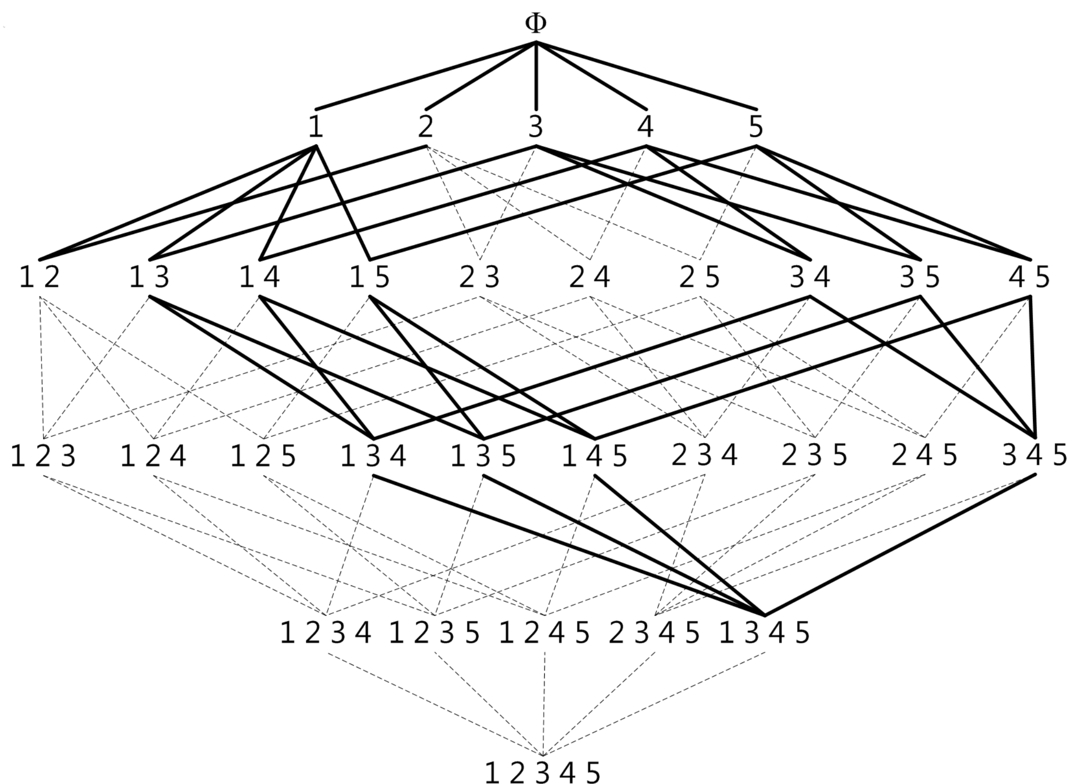
Sequence	Count
1 3 4	2
1 3 5	2
1 4 5	2
3 4 5	2

Sequence	Count
1 3 4 5	2

Gambar 4.8 – Tahapan Algoritma *AprioriAll*

Proses yang terjadi pada algoritma *AprioriAll* mirip dengan algoritma *Apriori*. Dimulai dengan membentuk *candidate* yang terdiri dari dua *sequence* dengan menggunakan `large_1_sequence` yang kemudian diikuti dengan penghilangan *candidate* yang tidak memenuhi `minimum_support`. Semua *sequence* yang memenuhi nilai `minimum_support` disebut `large_sequences`.

Berikut ini adalah representasi *tree* pada saat proses *candidate generation* yang terjadi selama algoritma *AprioriAll* dijalankan.



Gambar 4.9 – *Candidate Generation Tree*

Dimulai dari kelima elemen pada **`large_1_sequences`** dibentuk 10 *candidate* yang memiliki panjang masing-masing dua elemen, kemudian dari setiap *candidate* yang memenuhi nilai **`minimum_support`** dibentuk kembali empat *candidate* yang panjangnya tiga elemen. Dari ketiga elemen tersebut hanya dapat dibentuk satu *candidate* akhir yang panjangnya empat elemen.

Pada gambar di atas, garis tebal menandakan semua *candidate* yang memenuhi nilai **`minimun_support`** dan digunakan untuk membentuk *candidate* pada tahap berikutnya. Garis putus-putus menandakan bahwa *candidate* tersebut dilalui karena tidak memenuhi nilai **`minimum_support`** sehingga tidak akan digunakan pada tahap berikutnya.

Pencarian pola..., Fandi, FASILKOM UI, 2009

4.6 *MAXIMAL PHASE*

Dari semua **`large_sequences`** yang dihasilkan pada fase sebelumnya akan disaring untuk mendapatkan **`maximal_sequences`**.

Berikut ini adalah *pseudocod*e untuk **`maximal_sequences`**.

```
foreach sequence s1 in large_sequences
{
     foreach sequence s2 in large_sequences
     {
          if s2 not is_subsequences from s1
          {
               added s2 to maximal_sequences;
               delete s2 from large_sequences;
          }
          else
          {
               delete s2 from large_sequences;
          }
     }
}
```

Gambar 4.10 – *Pseudocode Maximal Phase*

Berikut ini adalah **`maximal_sequences`** yang dihasilkan.

Maximal Sequence
(30) (90)
(30) (40 70)

Tabel 4.7 – *maximal_sequences*

4.7 PENAMPILAN *OUTPUT*

Berikut ini adalah *output* akhir dari contoh yang telah dibahas dan ditampilkan pada bagian *result* pada FIKUI *Mining*.

```
Number of customer = 5
Minimum support = 2.00 customer

Sort Phase: Customer Sequences
1 -> ( 30 ) ( 90 )
2 -> ( 10  20 ) ( 30 ) ( 40  60  70 )
3 -> ( 30  50  70 )
4 -> ( 30 ) ( 40  70 ) ( 90 )
5 -> ( 90 )

Litemsets Phase: Litemsets
( 30 ) -> 1
( 90 ) -> 2
( 40 ) -> 3
( 70 ) -> 4
( 40  70 ) -> 5

Transformed Phase: Transformed Sequences
1 -> { 1 } { 2 }
2 -> { 1 } { 3 4 5 }
3 -> { 1 4 }
4 -> { 1 } { 3 4 5 } { 2 }
5 -> { 2 }

Sequences Phase: Large Sequences
 1  2
 1  3
 1  4
 1  5
 3  4
 3  5
 4  5
 1  3  4
 1  3  5
 1  4  5
 3  4  5
 1  3  4  5

Maximal Phase: Maximal Sequences
->  1  2
->  1  3  4  5

Execution time = 0.16 second.
```

Gambar 4.11 – Penampilan *Output*

4.8 OPTIMASI HASIL IMPLEMENTASI

Beberapa optimasi untuk meningkatkan efisiensi yang telah dilakukan oleh penulis dalam tahap implementasi ini yaitu:

- Dengan menentukan *customer ID* yang berupa bilangan bulat yang berurutan, maka tidak diperlukan algoritma pengurutan (*sorting*) untuk mengurutkan transaksi yang terdapat pada *input file*. Dengan menggunakan struktur data *vector* maka *customer ID* diwakili dengan nomor *index* yang dimiliki *vector* tersebut.
- Penggunaan variabel yang sama berulang kali untuk mencegah pemborosan pengunaan *memory* komputer. Apabila terdapat variabel yang telah selesai digunakan, baik yang berjenis primitif ataupun objek, akan digunakan kembali oleh proses lain yang membutuhkannya sehingga tidak perlu mendeklarasikan variabel baru sejenis.
- Untuk setiap *vector* yang sudah selesai digunakan dan nilainya tidak diperlukan lagi, maka *vector* tersebut dikosongkan isinya. Hal ini juga untuk mencegah pemborosan pengunaan *memory* komputer.
- Dalam beberapa proses sering dibutuhkan suatu tahapan yang sama sehingga, penulis membuat beberapa *function* pembantu yang bisa digunakan berulang kali dan bersamaan. Misalnya **`apriori_sub_gen()`**, **`in_vector()`**, **`vin_vector()`**, **`in_vitemset()`**, **`vin_vectorder()`**, dan lainnya (lihat tabel 3.1).

BAB V HASIL PENGUJIAN

Bab ini berisi penjelasan tentang cara pengujian, hasil pengujian dan analisis hasil pengujian. Pengujian dibagi menjadi dua bagian yaitu pengujian dengan menggunakan *dataset* berukuran kecil dan pengujian dengan menggunakan *dataset* berukuran besar.

5.1 LINGKUNGAN PENGUJIAN

Sebelum mencapai tahap pengujian dengan menggunakan *dataset* yang berukuran besar, penulis telah melakukan berkali-kali pengujian dengan menggunakan *dataset* yang berukuran kecil untuk memastikan bahwa hasil implementasi mampu menghasilkan *output* yang benar.

Pengujian hasil implementasi dengan menggunakan *dataset* yang telah ditentukan oleh penulis berlangsung di komputer dengan spesifikasi berikut :

- *Processor* : Intel Pentium IV 2.66 GHz
- *Memory* : 448 MB RAM
- *Harddisk* : 19,5 GB
- *Operating System* : Microsoft Windows XP Professional SP2

Pada saat pengujian, tidak ada perangkat lunak yang menggunakan CPU *resources*, dengan kata lain tidak ada program lain yang bekerja sehingga perangkat lunak hasil implementasi bisa menggunakan sampai 99% CPU *resources*. Selain itu, perangkat lunak hasil implementasi bekerja pada *hard drive* yang sama dengan tempat *dataset* berada. Semua ini dipersiapkan agar hasil pengujian mencapai *performance* yang maksimal.

Pencarian pola..., Fandi, FASILKOM UI, 2009

5.2 HASIL PENGUJIAN UNTUK *DATASET* KECIL

Tujuan utama pengujian dengan menggunakan *dataset* yang berukuran kecil adalah memastikan bahwa perangkat lunak telah diimplementasikan dengan benar sehingga mampu menghasilkan *output* yang sesuai. Untuk pengujian ini, penulis menggunakan *dataset* yang telah dibahas pada bab pendahuluan dan bab implementasi yaitu *dataset* yang dibuat oleh Agrawal dkk **[AGR95]**.

```
@data
1|30
1|90
2|10 20
2|30
2|40 60 70
3|30 50 70
4|30
4|40 70
4|90
5|90
```

Gambar 5.1 – *Dataset* Agrawal

Pengujian berhasil dilakukan dengan menghasilkan *output* akhir (*maximal sequence*) yang sesuai dengan contoh **[AGR95]** dalam waktu **0,17 detik** dan nilai *support* 25%.

```
( 30 ) ( 90 )
( 30 ) ( 40  70 )
```

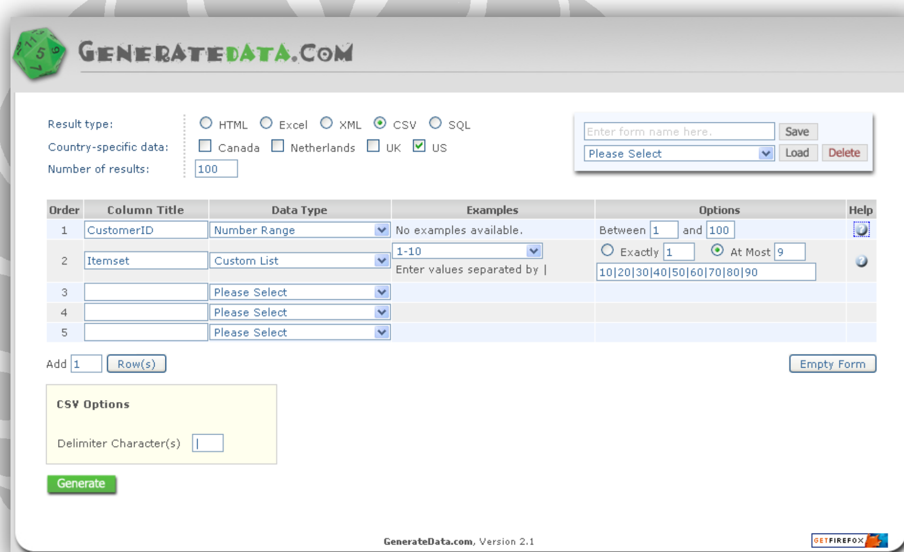
Gambar 5.2 – *Output* Agrawal

Waktu ini diukur mulai dari *sort phase* dijalankan sampai *output* akhir (*maximal sequence*) ditampilkan pada jendela *result*.

5.3 HASIL PENGUJIAN UNTUK *DATASET* BESAR

Sebelum melakukan pengujian untuk *dataset* yang berukuran besar, penulis harus menentukan karakteristik *dataset* yang akan digunakan dan membuat *dataset* yang bisa diuji dalam waktu relatif tidak terlalu lama.

Untuk membuat *dataset*, penulis menggunakan perangkat lunak yang khusus dibuat dengan tujuan menghasilkan *dataset* untuk keperluan eksperimen. Perangkat lunak ini dibuat dalam bahasa PHP dan menggunakan *database* MySQL **[KEE09]**. Perangkat lunak ini disebut *data generator*.



Gambar 5.3 – *Data Generator*

Dilihat dari bentuk *transaction database* yang direpresentasikan oleh *dataset* diperoleh tiga besaran yang mungkin mempengaruhi *performance* yaitu:

- Nilai *support* yang didefinisikan pengguna.
- Jumlah transaksi (*number of transaction*), direpresentasikan oleh jumlah baris pada *dataset*.
- Ukuran *itemset* (*itemset size*) masing-masing transaksi, direpresentasikan oleh jumlah *item* pada setiap baris *dataset*.

Selain itu juga terdapat beberapa karakteristik tambahan yang mungkin dimiliki data pada pencarian pola sekuensial sebagai berikut.

Simbol	Nama Karakteristik	Keterangan
C	*number of customers*	jumlah semua *customer*
T	*number of transactions*	jumlah semua transaksi yang terdapat pada *dataset*
I	*number of items*	jumlah semua *item* yang diperdagangkan
|C|	*average number of transactions per customer*	rata-rata jumlah transaksi per *customer*, yang terdapat pada *customer sequences*
|T|	*average number of items per transaction*	rata-rata jumlah *item* per transaksi, yang terdapat pada *customer sequences*
S	*number of maximal sequence*	jumlah *maximal sequence* yang ditemukan
|MS|	*average number of transaction per maximal sequence*	rata-rata jumlah transaksi per *maximal sequence* yang ditemukan
|TS|	*average number of item per transaction in maximal sequence*	rata-rata jumlah *item* per *transaction* pada setiap *maximal sequence* yang ditemukan
CD	*customer availability ratio*	jumlah *customer* yang hadir di *dataset* dibagi dengan jumlah semua *customer*

Tabel 5.1 – Karakteristik Data

Beberapa karakteritik di atas digunakan untuk penamaan *dataset*. Contoh apabila terdapat *dataset* **`C10-T3-I5_1.dat`** artinya |C| = 10, |T| = 3, I = 5, dan termasuk *dataset* urutan ke-1 pada jenisnya.

5.3.1 *Performance vs Support*

Pengujian ini bertujuan untuk mendapatkan hubungan antara *performance* dengan nilai *support*. Untuk pengujian ini digunakan tiga *dataset* dengan karakteristik sebagai berikut.

Dataset	C	T	I	|C|	|T|	S	|MS|	|TS|	CD
C10-T3-I5_1.dat	100	1000	5	10	3	1	1	2	1
C10-T3-I5_2.dat	100	1000	5	10	3	2	1	2	1
C10-T3-I5_3.dat	100	1000	5	10	3	2	1	2	1

Tabel 5.2 – Karakteristik Dataset untuk *Support* 90%

Dataset	C	T	I	|C|	|T|	S	|MS|	|TS|	CD
C10-T3-I5_1.dat	100	1000	5	10	3	4	2,75	1,14	1
C10-T3-I5_2.dat	100	1000	5	10	3	5	2	1,4	1
C10-T3-I5_3.dat	100	1000	5	10	3	4	2	1,5	1

Tabel 5.3 – Karakteristik Dataset untuk *Support* 75%

Dataset	C	T	I	|C|	|T|	S	|MS|	|TS|	CD
C10-T3-I5_1.dat	100	1000	5	10	3	4	2,25	1,22	1
C10-T3-I5_2.dat	100	1000	5	10	3	4	2,25	1,44	1
C10-T3-I5_3.dat	100	1000	5	10	3	5	2,2	1,54	1

Tabel 5.4 – Karakteristik Dataset untuk *Support* 50%

Dataset	C	T	I	|C|	|T|	S	|MS|	|TS|	CD
C10-T3-I5_1.dat	100	1000	5	10	3	2	3	1,17	1
C10-T3-I5_2.dat	100	1000	5	10	3	1	4	1,5	1
C10-T3-I5_3.dat	100	1000	5	10	3	4	2,75	1,36	1

Tabel 5.5 – Karakteristik Dataset untuk *Support* 25%

Dataset	C	T	I	|C|	|T|	S	|MS|	|TS|	CD
C10-T3-I5_1.dat	100	1000	5	10	3	1	4	1,25	1
C10-T3-I5_2.dat	100	1000	5	10	3	1	4	1,5	1
C10-T3-I5_3.dat	100	1000	5	10	3	1	4	1,5	1

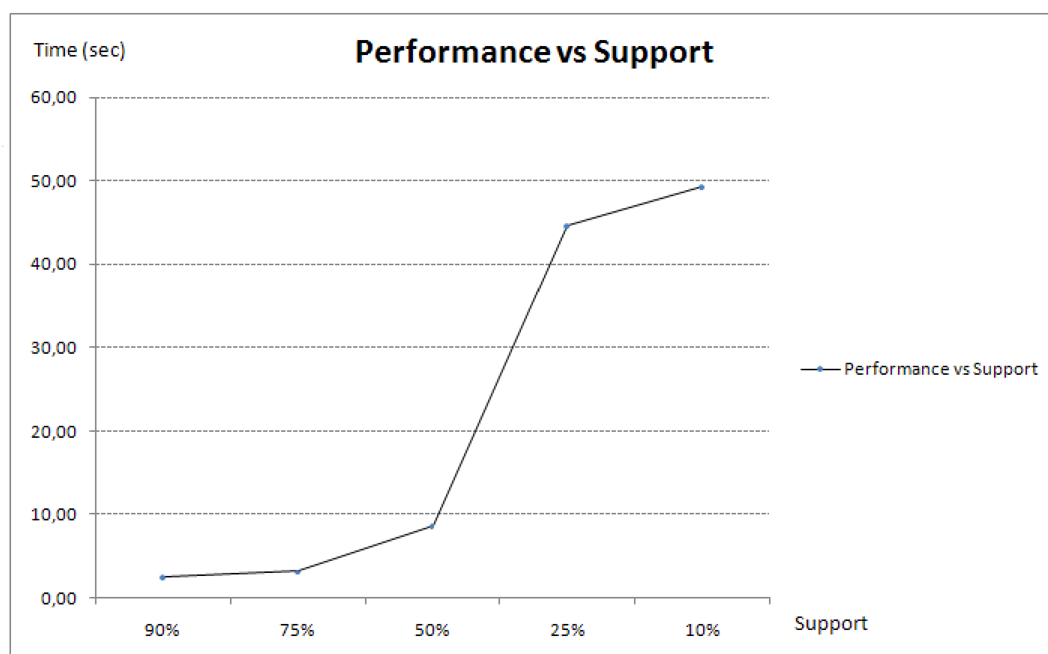
Tabel 5.6 – Karakteristik Dataset untuk *Support* 10%

Berikut ini adalah hasil pengujian untuk ketiga *dataset* yang berupa waktu eksekusi (detik) untuk setiap nilai *support*.

Dataset	Support				
	90%	75%	50%	25%	10%
C10-T3-I5_1.dat	2,28	2,59	4,34	10,67	10,69
C10-T3-I5_2.dat	2,56	3,59	11,56	68,45	68,58
C10-T3-I5_3.dat	2,58	3,17	9,80	54,55	68,52
Waktu rata-rata	**2,47**	**3,12**	**8,57**	**44,56**	**49,26**

Tabel 5.7 – Hasil Pengujian *Performance vs Support*

Berikut ini adalah grafik hasil rata-rata pengujian *performance vs support*.



Gambar 5.4 – Hasil Pengujian *Performance vs Support*

Dari grafik di atas terlihat bahwa pergerakan waktu berbanding linear mulai dari nilai *support* 90% ke nilai support 50%. Pergerakan waktu berbanding eksponensial ketika nilai *support* lebih kecil dari 50%. Pergerakan waktu berbanding linear kembali mulai ketika nilai *suppot* 25% sampai nilai *support* 10%. Semakin kecil nilai *support* yang diberikan, semakin besar waktu yang dibutuhkan untuk memprosesnya.

Hal ini terjadi karena semakin kecil nilai *support*, maka semakin banyak *sequence* yang memenuhi nilai *minimun support*, sehingga semakin banyak pula *candidate* yang terbentuk. Akibatnya semakin besar waktu yang dibutuhkan untuk membentuk *candidate* tersebut.

Semua waktu pada grafik di atas diperoleh dari rata-rata waktu hasil pengujian beberapa *dataset*, sehingga secara umum hubungan nilai *support* dan waktu (*performance*) adalah berbanding terbalik.

5.2.2 *Performance vs Transaction*

Pengujian ini bertujuan untuk mendapatkan hubungan antara *performance* dengan jumlah transaksi (*number of transaction*). Untuk pengujian ini digunakan sepuluh *dataset* dengan nilai *support* 90% dan karakteristik sebagai berikut.

Dataset	C	T	I	|C|	|T|	S	|MS|	|TS|	CD
C100-T3-I5_1.dat	100	10000	5	10	3	4	1,25	3	1
C100-T3-I5_2.dat	100	10000	5	10	3	2	1	2	1
C150-T3-I5_1.dat	100	15000	5	10	3	6	1,16	2	1
C150-T3-I5_2.dat	100	15000	5	10	3	5	1	2	1
C200-T3-I5_1.dat	100	20000	5	10	3	4	1	2	1
C200-T3-I5_2.dat	100	20000	5	10	3	4	1,5	2	1
C250-T3-I5_1.dat	100	25000	5	10	3	3	1	3	1
C250-T3-I5_2.dat	100	25000	5	10	3	3	1	3	1
C300-T3-I5_1.dat	100	30000	5	10	3	2	1	3,5	1
C300-T3-I5_2.dat	100	30000	5	10	3	2	1	3	1

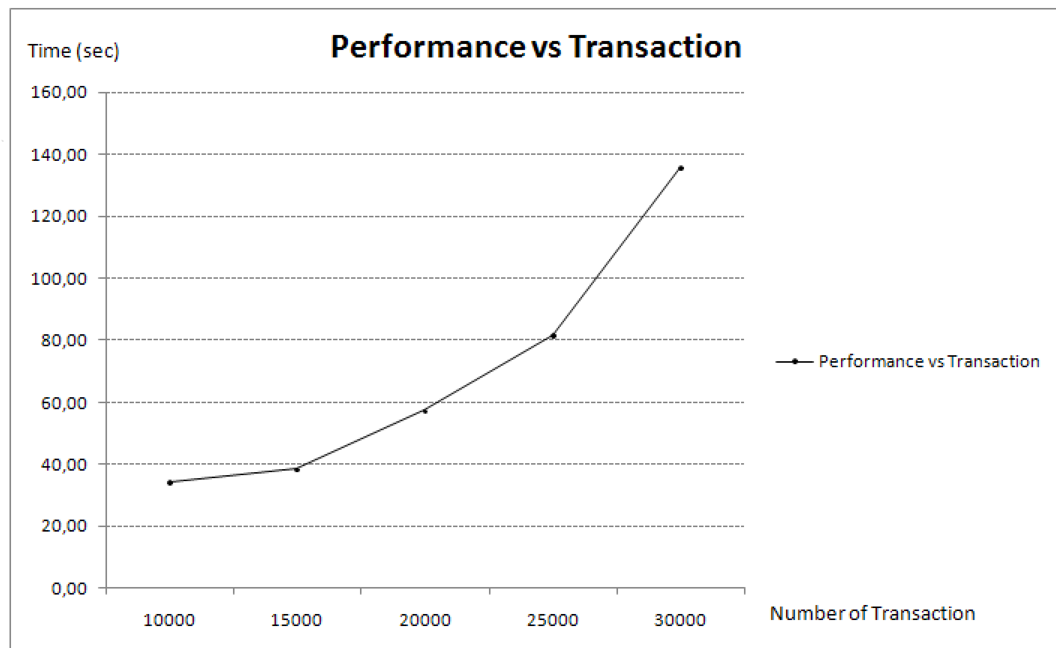
Tabel 5.8 – Karakteristik Dataset untuk *Performance vs Transaction*

Berikut ini adalah hasil pengujian untuk kesepuluh *dataset* yang berupa waktu eksekusi (detik) untuk setiap nilai jumlah transaksi.

Dataset	Jumlah Transaksi				
	10000	15000	20000	25000	30000
C100-T3-I5_1.dat	46,7				
C100-T3-I5_2.dat	21,80				
C150-T3-I5_1.dat		40,06			
C150-T3-I5_2.dat		36,84			
C200-T3-I5_1.dat			45,88		
C200-T3-I5_2.dat			68,59		
C250-T3-I5_1.dat				94,81	
C250-T3-I5_2.dat				68,20	
C300-T3-I5_1.dat					181,98
C300-T3-I5_2.dat					89,41
Waktu rata-rata	**34,25**	**38.45**	**57,23**	**81,50**	**135,70**

Tabel 5.9 – Hasil Pengujian *Performance vs Transaction*

Berikut ini adalah grafik hasil rata-rata pengujian *performance vs transaction*.



Gambar 5.5 – Hasil Pengujian *Performance vs Transaction*

Dari grafik di atas terlihat bahwa pergerakan waktu berbanding linear mulai dari jumlah transaksi 10000 sampai jumlah transaksi 30000. Semakin besar jumlah transaksi, semakin besar waktu yang dibutuhkan untuk memprosesnya.

Hal ini terjadi karena semakin banyak jumlah transaksi, semakin banyak jumlah iterasi yang dibutuhkan untuk membentuk *customer sequence*, *transformed sequence*, dan *output* lainnya.

Semua waktu pada grafik di atas diperoleh dari rata-rata waktu hasil pengujian beberapa *dataset*, sehingga secara umum hubungan jumlah transaksi (*number of transaction*) dan waktu (*performance*) adalah berbanding lurus.

5.2.3 *Performance vs Itemset*

Pengujian ini bertujuan untuk mendapatkan hubungan antara *performance* dengan ukuran rata-rata *itemset* pada masing-masing transaksi (|T|). Untuk pengujian ini digunakan delapan *dataset* dengan nilai *support* 90% dan karakteristik sebagai berikut.

Dataset	C	T	I	|C|	|T|	S	|MS|	|TS|	CD
C10-T2-I3_1.dat	100	1000	3	10	2	1	1	2	1
C10-T2-I3_2.dat	100	1000	3	10	2	1	1	3	1
C10-T3-I5_1.dat	100	1000	5	10	3	2	1	3	1
C10-T3-I5_2.dat	100	1000	5	10	3	3	1	2,33	1
C10-T4-I7_1.dat	100	1000	7	10	4	3	1	3,67	1
C10-T4-I7_2.dat	100	1000	7	10	4	4	1	2,75	1
C10-T5-I9_1.dat	100	1000	9	10	5	5	1	3	1
C10-T5-I9_2.dat	100	1000	9	10	5	8	1	2,5	1

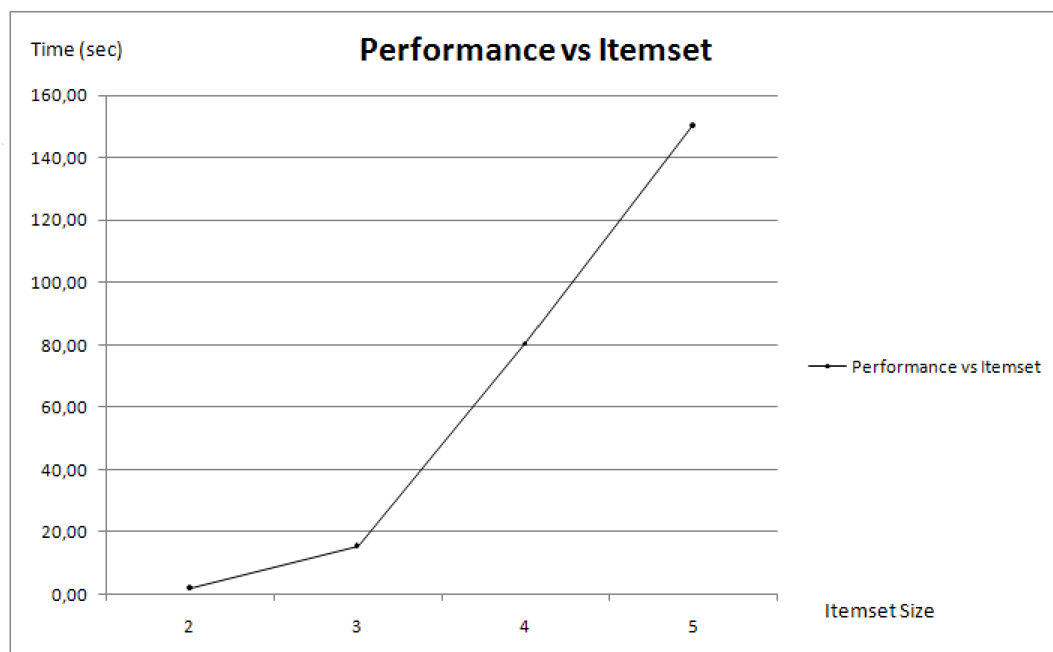
Tabel 5.10 – Karakteritik Dataset untuk *Performance vs Itemset*

Berikut ini adalah hasil pengujian untuk kedelapan *dataset* yang berupa waktu eksekusi (detik) untuk setiap ukuran rata-rata *itemset* pada masing-masing transaksi (|T|).

Dataset	Rata-rata Jumlah Item per Transaksi			
	2	3	4	5
C10-T2-I3_1.dat	1,25			
C10-T2-I3_2.dat	2,64			
C10-T3-I5_1.dat		26,19		
C10-T3-I5_2.dat		4,97		
C10-T4-I7_1.dat			134,66	
C10-T4-I7_2.dat			26,34	
C10-T5-I9_1.dat				241,01
C10-T5-I9_2.dat				60,36
Waktu rata-rata	**1,95**	**15,58**	**80,5**	**150,69**

Tabel 5.11 – Hasil Pengujian *Performance vs Itemset*

Berikut ini adalah grafik hasil rata-rata pengujian *performance vs itemset*.



Gambar 5.6 – Hasil Pengujian *Performance vs Itemset*

Dari grafik di atas terlihat bahwa pergerakan waktu berbanding linear mulai dari *itemset* berukuran rata-rata 2 ke *itemset* berukuran rata-rata 3. Kemudian pergerakan waktu naik tajam untuk *itemset* ukuran rata-rata lebih dari 3. Semakin besar ukuran rata-rata *itemset* per transaksi, semakin besar waktu yang dibutuhkan untuk memprosesnya.

Hal ini terjadi karena semakin besar ukuran rata-rata *itemset* per transaksi, semakin banyak kemungkinan jumlah *candidate* yang akan terbentuk pada *litemset phase* (dalam kasus ini menggunakan algoritma *Apriori*).

Semua waktu pada grafik di atas diperoleh dari rata-rata waktu hasil pengujian beberapa *dataset*, sehingga secara umum hubungan ukuran *itemset* (*itemset size*) dan waktu (*performance*) adalah berbanding lurus.