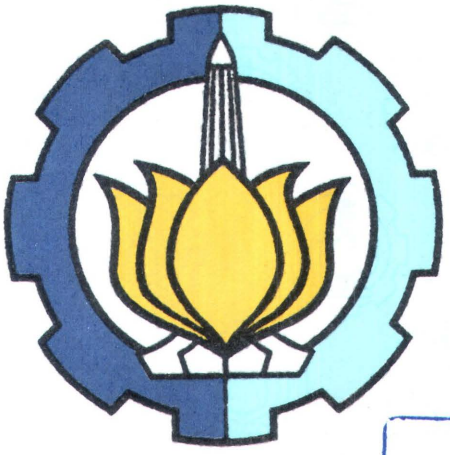
25032 /H06

MILIK PERPUSTAKAAN ITS

IMPLEMENTASI ALGORITMA CHARM-L DALAM MENGGALI *FREQUENT CLOSED ITEMSET*

TUGAS AKHIR

RSSI
005.1
Yen
I-1
2006



PERPUSTAKAAN ITS

Tgl. Terima	21-2-06
Terima Dari	H
No. Agenda Prp.	229287

Disusun Oleh :

YENNY KURNIAWATI
5201.100.005

PROGRAM STUDI SISTEM INFORMASI
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2006

IMPLEMENTASI ALGORITMA CHARM-L DALAM MENGGALI *FREQUENT CLOSED ITEMSET*

TUGAS AKHIR

Diajukan untuk Memenuhi Sebagian Persyaratan

Memperoleh Gelar Sarjana Komputer

Pada

Program Studi Sistem Informasi

Fakultas Teknologi Informasi

Institut Teknologi Sepuluh Nopember

Surabaya

Mengetahui/Menyetujui

Dosen Pembimbing I

10/2 '06

Rully Soelaiman, S.Kom, M.Kom.

NIP. 132.085.802

SURABAYA

JANUARI 2006

ABSTRAK

Himpunan *frequent closed itemset* dapat digunakan untuk mengetahui *support* dari semua *frequent itemset* secara tepat, dan jumlahnya lebih sedikit daripada jumlah *frequent itemset* yang ditemukan pada minimum support yang sama. Algoritma CHARM-L merupakan algoritma yang efisien untuk menggali struktur lattice dari semua *frequent closed itemset*.

Dalam penelitian ini dibuat aplikasi implementasi algoritma CHARM-L yang dapat menghasilkan *frequent closed itemset*. Algoritma ini melakukan enumerasi *closed itemset* dalam ruang pencarian *IT-Search Tree* yang dapat dipecah menjadi beberapa bagian *subtree* yang *independent*, berdasarkan sifat *equivalence classes*. Pencarian *itemset* dilakukan pada setiap *subtree* secara terpisah dengan metode *Hybrid Search* yang didasarkan pada sifat pasangan *itemset-tidset* guna mempercepat proses pencarian. Format *diffset* digunakan sebagai representasi data vertikal sehingga lebih menghemat memori. Pemangkasan *non-closed itemset* dilakukan dengan operasi irisan himpunan yang sederhana.

Dari hasil uji coba didapatkan bahwa aplikasi ini dapat menemukan semua itemset dalam *frequent closed itemset*. Semakin kecil nilai *minimum support* yang dimasukkan, maka *frequent closed itemset* yang dihasilkan akan semakin besar.

Kata kunci : *CHARM-L, closed itemset, frequent itemset, closed itemset lattice, data mining, minimum support, basis data transaksi.*

KATA PENGANTAR

Puji syukur penulis ucapkan kehadirat Tuhan YME yang telah memberikan rahmat dan karunianya kepada penulis, sehingga dengan rahmat-Nya jugalah penulis dapat menyelesaikan tugas akhir ini dengan baik.

Tugas akhir dengan judul "IMPLEMENTASI ALGORITMA CHARM-L DALAM MENGGALI *FREQUENT CLOSED ITEMSET*" ini disusun guna memenuhi sebagian persyaratan untuk memperoleh gelar sarjana pada Program Studi Sistem Informasi, Institut Teknologi Sepuluh Nopember Surabaya.

Pada kesempatan ini penulis menyampaikan ucapan terima kasih kepada semua pihak yang telah membantu penulis baik dalam pelaksanaan tugas akhir hingga selesainya penyusunan laporan tugas akhir ini. Ucapan terima kasih penulis sampaikan kepada :

- Seluruh keluarga di rumah, Mama, Papa, ketiga kakakku atas kasih sayang, doa, bantuan, dan dukungannya kepada penulis selama ini.
- Bapak Rully Soelaiman, S.Kom, M.Kom selaku dosen pembimbing, atas bantuannya selama penulis menimba ilmu di Program Studi Sistem Informasi, FTIF ITS Surabaya, khususnya selama proses pembuatan tugas akhir ini.
- Titus, atas dukungan, sharing, dan bantuannya selama ini; juga keluarganya yang sering ikut direpotkan penulis selama proses pembuatan tugas akhit ini.
- Diyah, Beny, Soni, Yudi, atas dukungan dan bantuannya selama ini.
- Semua teman–teman angkatan 2001 dan adik-adik di Keluarga Mahasiswa Sistem Informasi-ITS Surabaya.
- Semua Bapak dan Ibu Dosen Pengajar di Program Studi Sistem Informasi, FTIF ITS Surabaya yang telah memberikan ilmunya kepada penulis selama ini.
- Seluruh staff dan karyawan Program Studi Sistem Informasi dan Fakultas Teknologi Informasi atas bantuan dan dukungannya selama penulis menimba ilmu di Program Studi Sistem Informasi, FTIF ITS Surabaya.

- Teman-teman seperjuangan di R6, Ninik, Prima, Mbak Titik yang selalu memberikan dukungan, hiburan dan semangat dalam mengerjakan tugas akhir ini.
- Dan semua pihak yang tidak dapat penulis sebutkan satu persatu yang telah mendukung penulis selama ini.

Penulis menyadari bahwa dalam tugas akhir ini masih banyak terdapat kekurangan dan jauh dari sempurna, oleh karena itu penulis sangat mengharapkan saran dan kritik yang membangun dari semua pihak. Akhir kata penulis berharap semoga laporan ini bermanfaat bagi pembaca.

Surabaya, Januari 2006

BAB I

PENDAHULUAN

BAB I

PENDAHULUAN

1.1. LATAR BELAKANG

Dewasa ini, penggunaan teknologi informasi dan basis data dalam perusahaan semakin meluas, terutama karena kemampuannya yang luar biasa dalam menampung dan memproses data untuk berbagai aspek dan fungsi dalam perusahaan. Dengan semakin banyaknya data yang disimpan tersebut, adakalanya perusahaan mengalami kesulitan, misalnya dalam mengolah data berukuran besar dan kompleks seperti data *history* dan data sensus, yang ukurannya jauh lebih besar dan kompleks daripada data operasional sehari-hari. Di sisi lain, pengolahan dan pengelolaan data berukuran besar dan kompleks ini penting bagi perusahaan untuk menggali informasi yang berguna dalam mendukung penentuan strategi bisnis perusahaan.

Data mining merupakan cara menggali dan menganalisa data berukuran besar, untuk menemukan pola yang dapat membantu dalam pengambilan keputusan. Penggalian pola berulang (*frequent pattern*) atau *itemset* tersebut merupakan masalah penting dan mendasar dalam banyak aplikasi data mining, misalnya dalam penentuan kaidah asosiasi, korelasi, kaidah sekuensial, pola-pola multidimensi dan untuk berbagai pencarian penting lainnya [3].

Kebanyakan algoritma untuk penggalian pola yang telah diajukan selama ini merupakan varian dari algoritma Apriori [1][5]. Apriori menggunakan *breadth-first search* (*BFS*) yang mendata tiap-tiap *frequent itemset*. Apriori menggunakan sifat *downward-closure* dari support sebuah itemset, yaitu sifat yang menyatakan bahwa semua subset dari sebuah *frequent itemset* harus berulang (*frequent*), sifat ini digunakan untuk memangkas lingkup pencarian. Algoritma Apriori cukup mampu untuk mengatasi data-set yang jenisnya tidak banyak, misalnya data keranjang belanja, sedangkan untuk data level menengah seperti data

telekomunikasi dan data sensus, kemampuan dari algoritma ini sangat menurun, sedangkan pada kenyataan sehari-hari, sulit ditemukan suatu problem yang polanya sederhana.

Salah satu solusi untuk meningkatkan efisiensi dalam menggali *frequent pattern* yang rumit yaitu dengan hanya menggali *frequent closed itemset*. Sebagian besar algoritma yang umum digunakan untuk penggalian *frequent closed set* ini hanya menghasilkan keluaran berupa daftar dari *closed set* yang telah didapatkan. Di sisi lain, terdapat kebutuhan akan konstruksi *lattice* dari *frequent closed set* tersebut. Hal ini penting untuk mengefisienkan proses pembangkitan kaidah dari pola-pola yang telah dihasilkan itu, dan juga sangat berguna dalam visualisasi kaidah-kaidah yang telah dibangkitkan.

Pada tugas akhir ini dikembangan sebuah aplikasi untuk menggali *frequent closed itemset* yang lebih optimal dengan algoritma baru, yaitu CHARM-L. Algoritma CHARM-L ini dinilai sebagai algoritma terbaru yang paling efisien dalam menggali *frequent closed itemset* [1]. Kelebihan dari algoritma ini adalah pada ruang pencarian utama yang berupa *IT-Search Tree* yang dapat dipecah menjadi beberapa bagian *subtree independent*, berdasarkan pada *equivalence classes*. Pencarian *itemset* dapat dilakukan pada setiap *subtree* secara terpisah. Pencarian *frequent closed itemset* dalam algoritma ini dilakukan secara lebih efisien dengan menggunakan metode *Hybrid Search* yang didasarkan pada sifat pasangan *itemset-tidset* sehingga dapat melompati beberapa level pencarian. Kelebihan lainnya adalah pengunaan *diffset* sebagai representasi data vertikal sehingga lebih menghemat memori. Algoritma ini juga menggunakan operasi himpunan berupa operasi irisan yang sederhana, selain itu algoritma ini tidak memerlukan struktur data internal yang kompleks.

1.2. TUJUAN

Tujuan dari tugas akhir ini adalah untuk membangun suatu aplikasi yang mengimplementasikan algoritma CHARM-L dalam menggali *frequent closed itemset* dari suatu data set sintetis.

1.3. PERMASALAHAN

Permasalahan pada tugas akhir ini antara lain adalah:

1. Bagaimana mengimplementasikan algoritma CHARM-L pada aplikasi yang akan dibuat.
2. Bagaimana merancang keluaran dan antar muka yang mudah dimengerti oleh pengguna.
3. Bagaimana melakukan uji coba dan evaluasi untuk aplikasi yang akan dibuat.

1.4. BATASAN MASALAH

Batasan permasalahan dalam tugas akhir ini antara lain:

1. Data sintetis yang digunakan sebagai masukan sistem telah disesuaikan dengan kebutuhan aplikasi.
2. Ukuran data masukan akan disesuaikan dengan kemampuan komputasi perangkat keras yang akan digunakan.
3. Penelitian dititik beratkan pada penerapan algoritma CHARM-L untuk penggalian *frequent closed itemset*.
4. Perangkat lunak ini hanya menggali *frequent closed itemset*.

1.5. METODOLOGI

Penyusunan tugas akhir ini dibagi menjadi beberapa tahapan sebagai berikut:

1. **Studi kepustakaan**

   Aktivitas yang dilakukan pada tahapan ini adalah mengumpulkan dan mempelajari berbagai literatur yang terkait dengan penyusunan sistem ini, yaitu yang berhubungan dengan algoritma CHARM-L, pembangkitan data set sintetis untuk masukan sistem, dan berbagai literatur lain yang masih dalam lingkup permasalahan tugas akhir ini.

2. **Desain aplikasi**

   Aktivitas yang dilakukan adalah menyusun algoritma CHARM-L secara terinci dalam menggali *frequent closed itemset* sehingga diketahui kebutuhan akan masukan, proses, dan keluaran dari sistem yang dibuat, sehingga dengan demikian dapat dibuat suatu perencanaan yang rinci mengenai sistem tersebut. Perencanaan ini meliputi perancangan terhadap aplikasi yang dibuat, antara lain dengan menentukan struktur data yang digunakan, proses-proses yang harus dilaksanakan, dan penentuan rancangan antar muka dari aplikasi tersebut.

3. **Pembuatan aplikasi**

   Aktivitas yang dilakukan yaitu membangun aplikasi untuk menggali *frequent closed itemset* dari suatu data set sintetis, sesuai dengan perencanaan yang telah dibuat pada tahap sebelumnya. Aplikasi ini dibangun dengan menggunakan bahasa Java, dengan menggunakan compiler Eclipse v.3.0.

4. **Uji coba dan evaluasi**

   Aktivitas yang dilakukan pada tahapan ini adalah menguji aplikasi yang dibuat dengan menggunakan data masukan yang telah dipersiapkan sebelumnya. Hasil uji coba ini akan dievaluasi untuk menentukan perlu tidaknya dilakukan perbaikan pada aplikasi yang telah dibuat.

5. **Penyusunan buku tugas akhir**

   Aktivitas yang dilakukan pada tahapan ini adalah menyusun buku tugas akhir sebagai dokumentasi yang menjelaskan dasar teori dan metode yang digunakan serta hasil yang telah didapatkan dari proses uji coba.

1.6. SISTEMATIKA PENYUSUNAN

Dalam penyusunannya, laporan tugas akhir ini dikelompokkan menjadi lima bab, yang membahas bagian–bagian berikut :

BAB I Pendahuluan

Bab ini menjelaskan beberapa hal pokok dari tugas akhir ini, antara lain: latar belakang yang mendasari tugas akhir ini, tujuan serta manfaat tugas akhir, permasalahan dan batasannya, serta metodologi pelaksanaan tugas akhir dan sistematika dari penulisan buku tugas akhir ini.

BAB II Dasar Teori

Pada bab ini dijelaskan mengenai konsep data mining yang mendasar antara lain mengenai konsep nilai support minimal. Selain data mining, bab ini juga menjelaskan mengenai konsep *lattice* yang merupakan representasi hubungan *subset-superset* dari *closed itemset* yang dihasilkan.

BAB III Penerapan Algoritma CHARM-L untuk menemukan kaidah asosiasi

Bab ini menjelaskan langkah-langkah yang ditempuh dalam menjalankan algoritma CHARM-L mulai dari pembacaan basis data sebagai masukan sampai proses berakhir dan dihasilkan keluaran berupa *frequent closed itemset*.

BAB IV Perancangan dan Pembuatan Aplikasi

Bab ini menjelaskan tahapan-tahapan dalam proses perancangan dan pembuatan aplikasi data mining menggunakan algoritma CHARM-L. Dalam bab ini dijelaskan analisa kebutuhan sistem, perancangan data, proses, dan antarmuka. Perancangan data menjelaskan data masukan, proses dan keluaran dari aplikasi ini. Perancangan proses menjelaskan proses yang terjadi, digambarkan dalam bentuk Diagram Aliran Data (DAD), dan perancangan antarmuka menjelaskan mengenai antarmuka yang dibuat.

BAB V Uji Coba Aplikasi

Bab ini menjelaskan proses uji coba yang dilakukan terhadap aplikasi yang telah dibuat dengan menggunakan masukan berupa data sintetis sebagai data percobaan. Hasil uji coba ini kemudian akan dievaluasi untuk mengetahui apakah aplikasi ini sudah menghasilkan keluaran yang benar atau aplikasi ini masih perlu diperbaiki lebih lanjut.

BAB VI Penutup

Bab ini merupakan bagian terakhir dari buku Tugas Akhir ini yang berisi kesimpulan yang didapat selama pembuatan tugas akhir ini. Selain itu, bab ini juga berisi saran untuk pengembangan perangkat lunak yang lebih lanjut.

BAB II

DASAR TEORI

BAB II

DASAR TEORI

Bab ini akan menjelaskan konsep *data mining* yang mendasar antara lain mengenai penggalian *frequent pattern*, konsep nilai *support* dan *support* minimal, kaidah asosiasi, dan nilai *confidence* minimal. Selain itu, juga dijelaskan tentang sifat-sifat pasangan *itemset-tidset*, konsep pemecahan ruang pencarian *IT-Search Tree* dengan dasar *equivalence classes*, *diffset* sebagai representasi data vertikal, dan konsep dasar pada *lattice* yang berhubungan dengan enumerasi *itemset*.

2.1. PENGGALIAN FREQUENT PATTERN

Berikut ini dijelaskan istilah-istilah yang digunakan dalam konsep penggalian *frequent pattern*[1]: ***I*** merupakan himpunan item dan ***D*** merupakan sebuah basis data transaksional, dengan setiap transaksinya memiliki sebuah penanda unik (***tid***) dan mengandung sekumpulan item. Himpunan semua tid dinotasikan sebagai ***T***. Suatu himpunan $X \subset I$ disebut juga ***itemset***, dan suatu himpunan $Y \subset T$ disebut sebagai ***tidset***. Sebuah *itemset* dengan item sebanyak *k* disebut sebagai ***k-itemset***. Untuk kemudahan, *itemset* {*A, C, W*} ditulis sebagai *ACW* dan *tidset* {*2, 4, 5*} ditulis sebagai *245*. Untuk sebuah *itemset X*, maka *tidset* yang bersesuaian dinotasikan dengan *t*(*X*), yaitu himpunan semua *tid* dari transaksi yang mengandung *X* sebagai subsetnya, dinyatakan dengan $t(X) = \cap_{x \in X} t(x)$. Untuk sebuah *tidset Y*, maka *itemset* yang besesuian dinotasikan sebagai *i*(*Y*), yaitu himpunan semua item yang terjadi pada transaksi dengan *tid* pada *Y*, dinyatakan dengan $i(Y) = \cap_{y \in Y} i(y)$. Notasi $X \times t(X)$ digunakan untuk menyatakan pasangan *itemset-tidset* dan biasa disebut sebagai ***IT-pair***.

Support dari suatu *itemset X* dinotasikan dengan $\sigma(X)$, merupakan banyaknya transaksi dimana *itemset X* tersebut merupakan *subset*nya, dinyatakan dengan

$\sigma(X) = |t(X)|$. Sebuah *itemset* dinyatakan ***frequent*** jika supportnya lebih besar atau sama dengan ***minimum support*** (***min_sup***) yang disyaratkan oleh pengguna, yaitu jika $\sigma(X) \geq \min_\sup$. Suatu *frequent itemset* disebut **maksimal** (***maximal itemset***) jika *itemset* tersebut bukan merupakan *subset* dari *frequent itemset* yang lain. Jika $c : P(I) \rightarrow P(I)$ menyatakan ***closure operator***, yang didefinisikan dengan $c(X) = i(t(X))$, dengan $X \subset I$. Sebuah *frequent itemset* disebut ***closed*** jika dan hanya jika $c(X) = X$. Dengan kata lain, *frequent itemset X* disebut *closed* jika tidak terdapat *superset* $Y \supset X$ dengan $\sigma(X) = \sigma(Y)$.

Basis data contoh ditunjukkan pada tabel 2.1 [1]. Tampak bahwa pada basis data contoh terdapat 5 item yang berlainan, $I = \{A,C,D,T,W\}$, dalam 6 transaksi, $T = \{1,2,3,4,5,6\}$. Tabel sebelah kanan menunjukkan seluruh *frequent itemset* yang terkandung pada minimal 3 transaksi, sesuai dengan syarat bahwa *min_sup* = 50%.

Tabel 2.1 Basis data contoh

Basis Data

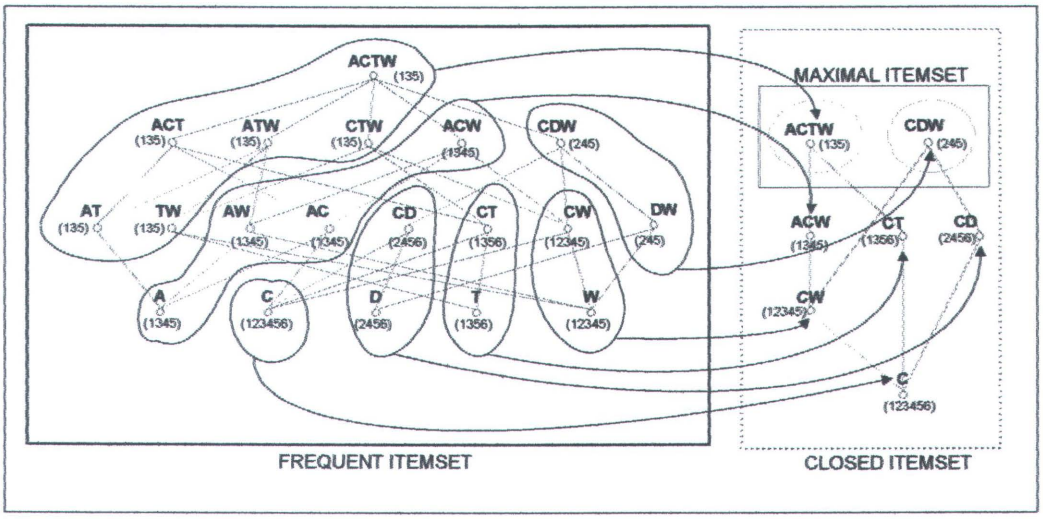
Transaksi	Item
1	ACTW
2	CDW
3	ACTW
4	ACDW
5	ACDTW
6	CDT

Seluruh Frequent *Itemset*

Minimum support = 50%

Support	*Itemset*
100% (6)	C
83% (5)	W, CW
67% (4)	A, D, T, AC, AW, CD, CT, ACW
50% (3)	AT, DW, TW, ACT, ATW, CDW, CTW, ACTW

Gambar 2.1 menunjukkan seluruh *frequent itemset* yang telah didapatkan beserta *tidset*nya masing-masing, ditampilkan dalam sebuah *subset lattice*. Dibandingkan dengan ke-19 buah *frequent itemset*, hanya terdapat 7 *closed itemset*, yang masing-masing didapatkan dengan membuang semua *itemset* yang mempunyai *tidset* yang sama, tampak dalam gambar yaitu yang ditunjukkan dengan area tertutup. Dari *closed itemset lattice* dapat dilihat bahwa terdapat 2 *maximal itemset*, yang ditandai dengan lingkaran, yaitu *ACTW* dan *CDW*.



Gambar 2.1 *Frequent*, *Closed*, dan *Maximal Itemset*

MILIK PERPUSTAKAAN ITS

2.2. KONSEP LATTICE

Berikut ini merupakan beberapa terminologi dari teori *lattice* yang berhubungan dengan penggalian *itemset* [2]:

Definisi 1 : Jika P merupakan sebuah himpunan, maka sebuah **urutan parsial** (***partial order***) dari P merupakan suatu relasi biner $\leq$, sedemikian sehingga untuk semua $X, Y, Z \in P$, relasinya adalah:

1. Refleksif : $X \leq X$.
2. Antisimetrik : $X \leq Y$ dan $Y \leq X$, menghasilkan $X = Y$.
3. Transitif : $X \leq Y$ dan $Y \leq Z$, menghasilkan $X \leq Z$.

Himpunan P dengan relasi $\leq$ disebut sebagai **himpunan terurut** (***ordered set***).

Sebagai contoh, jika $P = \{\{\},\{A\},\{A,C\}\}$ maka himpunan P merupakan *ordered set* karena berlaku ketiga sifat di atas untuk semua $X, Y, Z \in P$.

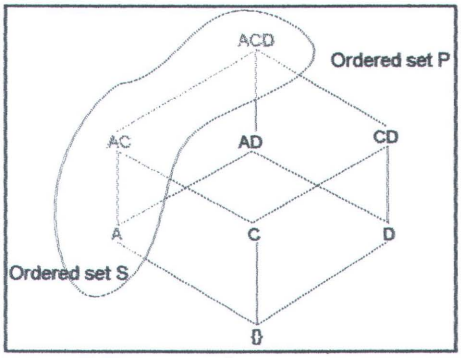
Definisi 2 : P merupakan sebuah himpunan terurut dan $X, Z, Y \in P$, maka X dikatakan **tertutup oleh** Y, dinotasikan dengan $X \lceil Y$, jika $X < Y$ dan $X \leq Z < Y$, menghasilkan $Z = X$, yaitu jika tidak terdapat elemen Z pada P dengan $X < Z < Y$.

Sebagai contoh pada gambar 2.2, untuk $P = \{\{\},\{A\},\{A,C\}\}$, maka X dikatakan tertutup oleh Y untuk $X = \{\}$ dan $Y = \{A\}$.

Untuk $X = \{\}$ dan $Y = \{A,C\}$ maka tidak dapat dikatakan X tertutup oleh Y karena $X < Y$ namun terdapat $Z = \{A\}$ dimana $\{\} \leq \{A\} < \{A,C\}$ namun $\{\} \neq \{A\}$.

Definisi 3 : Jika P merupakan sebuah himpunan terurut dan $S \subseteq P$, maka elemen $X \in P$ merupakan **batas atas (batas bawah)** dari S jika $s \leq X$ ($s \geq X$) untuk semua $s \in S$. Batas atas terkecil, disebut ***join*** dari S dinotasikan sebagai $\vee S$, dan batas bawah terbesar, disebut juga ***meet*** dari S dinotasikan sebagai $\wedge S$.
Elemen terbesar pada P dinotasikan sebagai $\top$, disebut juga **elemen puncak**, dan elemen terkecil pada P, dinotasikan sebagai $\perp$, disebut juga **elemen dasar**.

Contoh dapat dilihat pada gambar 2.2. Jika himpunan $P = \{\{\},\{A\},\{C\},\{D\},\{A,C\},\{A,D\},\{C,D\},\{A,C,D\}\}$ dan himpunan $S = \{\{A\},\{A,C\},\{A,C,D\}\}$ dengan $S \subseteq P$, maka batas bawah dari S yaitu elemen $\{A\}$, dan batas atasnya adalah elemen $\{A,C,D\}$. *Join* dari $S = \{A,C,D\}$ dan *meet* dari $S = \{A\}$. Elemen puncak dari P yaitu $\{A,C,D\}$ dan elemen dasarnya yaitu $\{\}$.



Gambar 2.2 Himpunan terurut P dan subsetnya, S

Definisi 4 : Jika L merupakan himpunan terurut, maka L disebut sebuah ***join*** (***meet***) ***semilattice*** jika $X \vee Y$ ($X \wedge Y$) ada untuk semua $X, Y \in L$.
L disebut sebuah ***lattice*** jika L merupakan *join semilattice* sekaligus *meet semilattice*, yaitu jika $X \vee Y$ maupun $X \wedge Y$ ada untuk setiap pasang elemen $X, Y \in$

L. L disebut ***complete lattice*** jika $\vee S$ maupun $\wedge S$ ada untuk semua subset $S \subseteq L$. Sebuah himpunan terurut $M \subset L$ merupakan ***sublattice*** dari L jika $X, Y \in M$ berimplikasi pada $X \vee Y \in M$ dan $X \wedge Y \in M$.

Sebagai contoh, untuk himpunan $L_1 = \{\{A\},\{C\},\{A,C\}\}$, maka L_1 merupakan *join semilattice* karena terdapat elemen $X \vee Y = \{A,C\}$, namun himpunan L_1 tersebut bukan merupakan *meet semilattice* karena tidak terdapat elemen $X \wedge Y = \{\}$. Jika $L_2 = \{\{\},\{A\},\{C\},\{A,C\},\{A,D\},\{A,C,D\}\}$, maka himpunan L_2 merupakan sebuah *lattice* karena terdapat $X \vee Y$ maupun $X \wedge Y$ untuk setiap pasang elemen $X, Y \in L_2$. *Ordered set* $M = \{\{\},\{A\},\{C\},\{A,C\}\}$ merupakan contoh *sublattice* dari L_2.

Untuk suatu himpunan S, himpunan terurut $P(S)$, yaitu ***power set*** **dari** ***S***, merupakan sebuah *complete lattice* dimana *join* dan *meet* dilakukan dengan operasi gabungan dan irisan, secara berurutan dapat ditulis seperti di bawah ini:

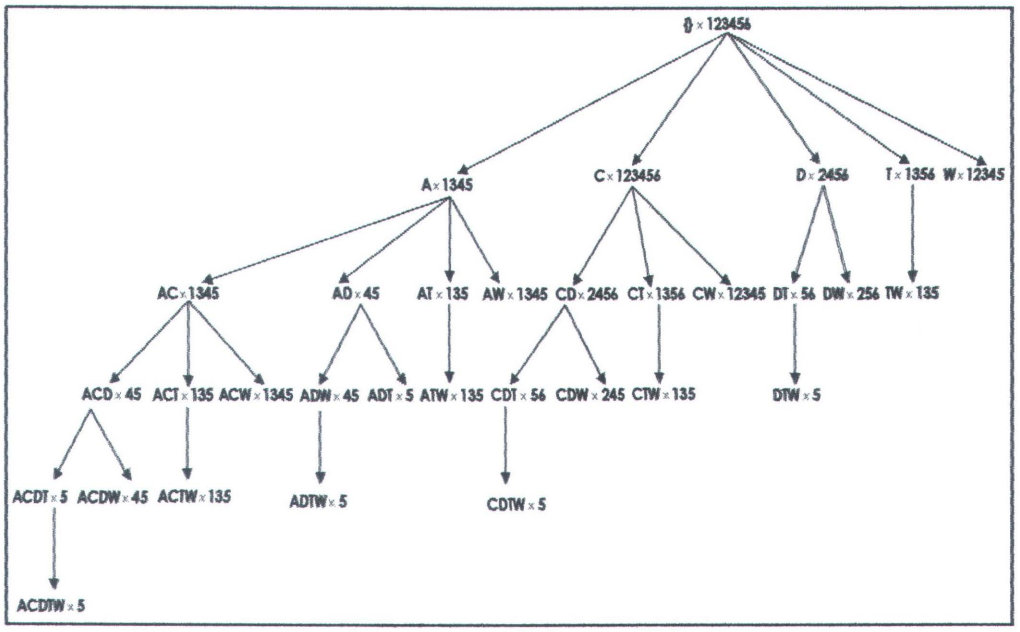
$$\vee\{A_i \mid i \in I\} = \bigcup_{i \in I} A_i \qquad \wedge\{A_i \mid i \in I\} = \bigcap_{i \in I} A_i$$

Elemen puncak dari $P(S)$ yaitu $\top = S$ dan elemen dasarnya yaitu $\perp = \{\}$. Untuk tiap $L \subseteq P(S)$, L disebut sebagai ***lattice of sets*** jika L bersifat tertutup dalam operasi gabungan dan irisan yang terbatas, sebagai contoh, $(L\ ;\subseteq)$ merupakan sebuah *lattice* dengan urutan parsial yang dispesifikasikan dengan relasi subset $\subseteq$, $X \vee Y = X \cup Y$ dan $X \wedge Y = X \cap Y$

2.3. *IT-SEARCH TREE* DAN *EQUIVALENCE CLASSES*

Dengan I merupakan himpunan item, didefinisikan suatu fungsi $p(X,k) = X[1:k]$ sebagai prefiks dari X sepanjang k dan ***prefix-based equivalence relation*** θ_k pada *itemset* sebagai berikut: $\forall X,\ Y \subseteq I$, $X \equiv_{\theta_k} Y \Leftrightarrow p(X,k) = p(Y,k)$.[1] [2] Artinya, dua *itemset* berada dalam satu kelas yang sama jika keduanya mempunyai prefiks yang sama sepanjang k.

Ruang pencarian *itemset* direpresentasikan sebagai sebuah *IT-Search Tree*, ditunjukkan pada gambar 2.3. Tiap node dalam *IT-Search Tree* tersebut berupa sebuah pasangan *itemset-tidset* (*IT-pair*) $X \times t(X)$, yang sebenarnya merupakan *prefix-based class*. Semua *child* dari sebuah *node* X termasuk dalam *equivalence class* dari node tersebut karena masing-masing mempunyai prefiks X yang sama. ***Equivalence class*** dinotasikan sebagai $[P] = \{l_1, l_2, ..., l_n\}$, dengan P merupakan *parent node* (prefiksnya) dan tiap-tiap l_i merupakan sebuah item tunggal, merupakan representasi dari node $Pl_i \times t(Pl_i)$.



Gambar 2.3 *IT-Search Tree*

Berikut ini merupakan definisi dari *equivalence classes*:

Definisi 5 : Jika P merupakan sebuah himpunan, maka sebuah **relasi ekuivalen** (***equivalence relation***) pada P merupakan suatu relasi biner $\equiv$, sedemikian sehingga untuk semua $X, Y, Z \in P$, relasinya adalah:

1. Refleksif : $X \equiv X$.

2. Simetrik : $X \equiv Y$, menghasilkan $Y \equiv X$.
3. Transitif : $X \equiv Y$ dan $Y \equiv Z$, menghasilkan $X \equiv Z$.

Equivalence relation ini mempartisi himpunan P menjadi subset-subset disjoinnya yang akan disebut sebagai ***equvalence classes***. *Equivalence class* dari elemen $X \in P$ dinyatakan dengan $[X] = \{Y \in P \mid X \in Y\}$.

Sebagai contoh, gambar 2.3 merupakan *IT-Search Tree* untuk basis data contoh pada Tabel 2.1. Node root merupakan kelas [] = {*A*,*C*,*D*,*T*,*W*}. Node child paling kanan dari root yaitu kelas [*A*] = {*C*,*D*,*T*,*W*}, yaitu semua itemset yang mengandung *A* sebagai prefiksnya. Tiap anggota kelas mewakili sebuah node dari parent node. Sebuah kelas mewaliki item-item dengan mana prefiksnya dapat diperluas untuk mendapatkan sebuah node frequent yang baru. Subtree dari prefiks yang *infrequent* tidak perlu diuji lebih lanjut.

Kelebihan pendekatan dengan *equivalence class* ini yaitu pada pemecahan ruang pencarian utama menjadi beberapa subproblem yang independen. Untuk tiap subtree yang berakar pada node *X*, masing-masing dapat diperlakukan seperti suatu permasalahan baru; pada subtree dapat dilakukan enumerasi itemset, kemudian diberikan prefiks *X* pada masing-masing itemset tersebut, dan demikian seterusnya.

Enumerasi *frequent pattern* berlangsung secara terus-menerus dalam kerangka *IT-Tree*. Pada tiap node atau kelas prefiks tertentu dapat dilakukan operasi irisan tidset untuk tiap pasangan elemen dalam kelas dan dilakukan pemeriksaan apakah *min_sup* sudah terpenuhi; perhitungan *support* berjalan bersamaan dengan pembangkitan itemset. Tiap *frequent itemset* merupakan kelas didalam node tersebut, yang memiliki elemen-elemen sendiri yang dapat diperluas secara rekursif. Dapat dinyatakan sebagai berikut, untuk suatu kelas itemset dengan prefiks [*P*], $[P] = \{l_1, l_2,...,l_n\}$, dapat dilakukan operasi irisan tidset $t(l_i)$ dan $t(l_j)$, dengan $j > i$ untuk mendapatkan sebuah kelas baru hasil perluasan $[Pl_i] = \{l_j \mid j > i$ dan $\sigma(Pl_il_j) \geq min_sup\}$. Pada contoh, dari root kosong [] = {*A*,*C*,*D*,*T*,*W*}, dengan *min_sup* = 50%, didapatkan kelas [*A*] = {*C*,*T*,*W*}, [*C*] = {*D*,*T*,*W*}, dan [*D*] ={*W*},

dan [*W*] ={ }. Perlu diingat bahwa kelas *A* tidak memuat elemen *D* karena *AD* *infrequent*.

2.4. SIFAT DASAR PASANGAN ITEMSET-TIDSET

Untuk dua node manapun dalam *IT-tree*, $X_i \times t(X_i)$ dan $X_j \times t(X_j)$, jika $X_i \subseteq X_j$, maka $t(X_j) \subseteq t(X_i)$. Misalnya untuk $ACW \subseteq ACTW$, $t(ACW) = 1345 \supseteq 135 = t(ACTW)$. Didefinisikan suatu fungsi $f : P(I) \mapsto N$ sebagai suatu pemetaan satu-satu dari *itemset* ke integer. Untuk *itemset* X_i dan X_j, dikatakan $X_i \leq_f X_j$ jhj $f(X_i) \leq f(X_j)$. Fungsi *f* mendefinisikan urutan keseluruhan (***total order***) bagi setiap set dari seluruh *itemset*. Misal, jika *f* merupakan urutan berdasarkan abjad, maka *itemset* $AC \leq AD$, sedangkan jika *f* mengurutkan *itemset* berdasarkan *support*nya, maka $AD \leq AC$ jika $\sigma(AD) \leq \sigma(AC)$.

Ada empat sifat dasar *IT-pair* yang akan digunakan menggali *closed set* dengan cepat [1]. Jika node dalam proses adalah node $P \times t(P)$, dengan $[P] = \{l_1, l_2, ..., l_n\}$ merupakan *prefix class*-nya. Jika X_i menunjukkan notasi dari *itemset* Pl_i, maka tiap anggota [*P*] merupakan sebuah *IT-pair* $X_i \times t(X_i)$.

Teorema 1. Jika $X_i \times t(X_i)$ dan $X_j \times t(X_j)$ merupakan dua anggota manapun dari kelas [*P*], maka keempat sifat ini berlaku :

1. Jika $t(X_i) = t(X_j)$, maka $c(X_i) = c(X_j) = c(X_i \cup X_j)$
2. Jika $t(X_i) \subset t(X_j)$, maka $c(X_i) \neq c(X_j)$, namun $c(X_i) = c(X_i \cup X_j)$.
3. Jika $t(X_i) \supset t(X_j)$, maka $c(X_i) \neq c(X_j)$, namun $c(X_j) = c(X_i \cup X_j)$.
4. Jika $t(X_i) \not\subseteq t(X_j)$ dan $t(X_i) \not\supseteq t(X_j)$,

   maka $c(X_i) \neq c(X_j) \neq c(X_i \cup X_j)$.

Pembuktian Teorema 1:

1. Jika $t(X_i) = t(X_j)$, maka $i(t(X_i)) = i(t(X_j))$, yang berarti $c(X_i) = c(X_j)$. Selain itu, $t(X_i) = t(X_j)$ berakibat pada $t(X_i \cup X_j) = t(X_i \cap X_j) = t(X_i)$, sehingga $i(t(X_i \cup X_j)) = i(t(X_i))$ yang berarti $c(X_i \cup X_j) = c(X_i)$. Sifat ini memungkinkan penggantian X_i dengan $X_i \cup X_j$, sehingga elemen X_j dapat dibuang dengan mempertimbangan *closure*nya dengan $X_i \cup X_j$.
2. Jika $t(X_i) \subset t(X_j)$, maka $t(X_i \cup X_j) = t(X_i \cap X_j) = t(X_i) \neq t(X_j)$, sehingga $c(X_i \cup X_j) = c(X_i) \neq c(X_j)$. Sifat ini memungkinkan penggantian X_i dengan $X_i \cup X_j$ karena keduanya mempunyai *closures* yang identik, namun elemen X_j tidak dapat dibuang dari class $[P]$ karena akan menghasilkan *closures* yang berbeda.
3. Jika $t(X_i) \supset t(X_j)$, maka $t(X_i \cup X_j) = t(X_i \cap X_j) = t(X_j) \neq t(X_i)$, sehingga $c(X_i \cup X_j) = c(X_j) \neq c(X_i)$. Sifat ini memungkinkan penggantian X_j dengan $X_i \cup X_j$ karena keduanya mempunyai *closures* yang identik, namun elemen X_i tidak dapat dibuang dari kelas $[P]$ karena akan menghasilkan *closures* yang berbeda.
4. Jika $t(X_i) \not\subseteq t(X_j)$ dan $t(X_i) \not\supseteq t(X_j)$, maka dapat dipastikan bahwa $t(X_i \cup X_j) = t(X_i \cap X_j) \neq t(X_i) \neq t(X_j)$, yang berakibat pada $c(X_i \cup X_j) \neq c(X_i) \neq c(X_j)$. Dengan sifat ini maka tidak ada elemen dalam kelas yang dapat dieliminasi karena melakukan eliminasi elemen X_i maupun X_j menghasilkan *closures* yang berbeda (tidak satupun yang merupakan subset dari lainnya).

Fungsi pengurutan berdasarkan sifat pasangan itemset-tidset

Dalam menspesifikasikan fungsi *f* yang mendefinisikan urutan keseluruhan (***total order***) untuk setiap set dari seluruh *itemset*, dapat dilakukan berbagai metode pengurutan. Metode yang biasa dilakukan yaitu mengurutkan berdasarkan abjad, namun metode apapun dapat dilakukan. Pendekatan yang paling baik adalah

dengan mengurutkan *itemset* berdasarkan *support*nya, dengan tujuan meningkatkan peluang untuk memangkas elemen dalam kelas [P].

Kesimpulan yang dapat diambil dari keempat sifat dasar pasangan *itemset-tidset* adalah, bahwa Kondisi 1 dan Kondisi 2 lebih menguntungkan daripada dua kondisi lainnya. Untuk Kondisi 1, kedua *itemset* memiliki *closure* yang identik sehingga elemen l_j dapat dibuang dari kelas [P_i] dan P_i dapat digantikan dengan elemen $P_i \cup l_j$. Untuk Kondisi 2, juga dapat dilakukan penggantian elemen P_i dengan $P_i \cup l_j$. Jelas bahwa untuk kedua kondisi ini, tidak ada elemen apapun yang perlu ditambahkan ke dalam kelas [P_i]. Dengan semakin banyaknya Kondisi 1 dan Kondisi 2 yang terjadi, maka level pencarian yang akan dilakukan akan semakin sedikit, sedangkan terjadinya Kondisi 3 dan Kondisi 4 menghasilkan penambahan pada himpunan node baru, sehingga membutuhkan level permrosesan lebih panjang. Karena yang diinginkan adalah $t(l_i) = t(l_j)$ (Kondisi 1) atau $t(l_i) \subset t(l_j)$ (Kondisi 2), maka sebaiknya *itemset* diurutkan berdasarkan peningkatan *support*nya [1].

2.5. *SUBSUMPTION CHECK* DALAM LATTICE

Jika X_i dan X_j merupakan dua itemset, dikatakan itemset X_i meliputi (***subsume***) itemset X_j jika dan hanya jika $X_j \subset X_i$ dan $\sigma(X_j) = \sigma(X_i)$. Sebelum sebuah *frequent itemset* ditambakan pada himpunan *frequent closed itemset* C, perlu diperiksa apakah *frequent itemset* tersebut sudah termasuk dalam salah satu *frequent closed itemset* yang sudah didapatkan.

Hal ini perlu dilakukan karena dalam penggalian dengan metode *depth-in-first*, dimungkinkan terjadi kasus seperti berikut ini: sebuah *closed set* Y ditambahkan pada C. Pada pemrosesan cabang selanjutnya dimungkinkan adanya pembangkitan *itemset* X yang tidak dapat diperluas lagi, dengan $X \subseteq Y$ dan $\sigma(Y) = \sigma(X)$. Pada kondisi ini, X bukan merupakan *closed set* karena sudah termasuk

dalam Y dan X tidak ditambahkan pada C. Hal ini sesuai dengan sifat dari *frequent closed itemset* yang telah dijelaskan pada subbab 2.1.

Untuk melakukan *subsumption checking* terhadap *itemset* X, perlu dicari semua *closed itemset* yang merupakan *superset* dari X, dinotasikan dengan himpunan S. Jika support X sama dengan support *closed itemset* $Z \in S$ manapun, maka X tidak ditambahkan ke dalam *closed set* C dan proses *subsumption checking* selesai. Penggunaan *subsumption check* dalam CHARM-L akan dijelaskan pada subbab 3.2.2, sedangkan contoh penggunaan *subsumption check* pada algoritma CHARM-L ditunjukkan pada subbab 3.2.4.

2.6. DIFFSET SEBAGAI PENGGANTI TIDSET

Dalam memanipulasi *IT-pair*, digunakan format data vertikal yang merupakan representasi penyimpanan untuk *tidset* tiap item dari basis data. Dalam tugas akhir ini, yang digunakan adalah format data vertikal yang disebut ***diffset***. *Diffset* ini menyimpan jalur perubahan *tids* suatu pola kandidat dari *parent frequent pattern*nya. Perubahan ini dicatat sepanjang perjalanan dari suatu node ke node *children*nya, dimulai dari node *root*.

Secara formal dapat dituliskan, bagi sebuah kelas dengan prefiks P, notasi $d(X)$ menyatakan *diffset* dari X, yang terkait dengan sebuah prefiks *tidset*, yang merupakan keseluruhan *tids* pada waktu tertentu. Dalam metode vertikal biasa, yang tersedia untuk suatu kelas tertentu yaitu *tidset* dari prefiks, $t(P)$, dan *tidset* dari setiap anggota kelasnya, $t(PX_i)$. Diasumsikan PX dan PY merupakan dua anggota dari kelas P. Dari definisi *support*, terbukti bahwa $t(PX) \subseteq t(P)$ dan $t(PY) \subseteq t(P)$ bernilai benar. Dengan demikian, maka dapat diperoleh *support* dari PXY dengan memeriksa kardinalitas dari $t(PX) \cap t(PY) = t(PXY)$.

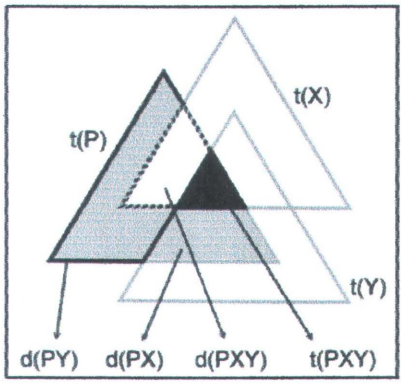
Jika yang tersedia untuk perhitungan bukanlah $t(PX)$ melainkan $d(PX)$, yang nilainya sama dengan $t(P) - t(X)$, yaitu perbedaan *tids* X dari P. Dengan cara yang sama dapat diperoleh $t(PY)$. Hal pertama yang perlu dicatat adalah *support* dari

suatu *itemset* bukan lagi merupakan kardinalitas dari *diffset*, namun nilainya menjadi $\sigma(PXY) = \sigma(P) - |d(PXY)|$.

Jadi, $d(PXY)$ harus dihitung. Dari definisi, $d(PXY) = t(PX) - t(PY)$. Namun yang tersedia adalah *diffset* dan bukannya *tidset* seperti yang diperlukan oleh ekspresi di atas. Hal ini dapat diselesaikan dengan cara berikut ini:

$$\begin{aligned} d(PXY) &= t(PX) - t(PY) = t(PX) - t(PY) + t(P) - t(P) \\ &= (\, t(P) - t(PY)\,) - (\, t(P) - t(PX)\,) \\ &= d(PY) - d(PX). \end{aligned}$$

Dengan kata lain, $d(PXY)$ tidak dihitung dari perbedaan *tidset*, $t(PX) - t(PY)$, namun dengan perbedaan dari *diffset*, $d(PY) - d(PX)$. Gambar 2.4 menunjukkan perbedaan daerah untuk *diffset* dan *tidset* untuk suatu kelas *prefiks* dan dua dari anggotanya.



Gambar 2.4 *Diffsets* : prefiks P

Tidset dari *P*, ditunjukkan dalam Gambar 2.4 dengan segitiga yang ditandai dengan $t(P)$, merupakan keseluruhan dari *tids* yang berkaitan. Daerah berwarna kelabu yang menunjukkan $d(PX)$ dan daerah dengan garis warna hitam menunjukkan $d(PY)$. Perlu diingat bahwa baik $t(PXY)$ maupun $d(PXY)$ merupakan *subset* dari *tidset prefiks PX*. Penggunaan *diffset* biasanya akan lebih kecil daripada menyimpan *tidset* bersama tiap *child* karena hanya perubahan yang berpengaruh lah yang dicatat dari sebuah *node* ke *node childrenn*ya. *Diffset* juga semakin kecil dengan semakin panjangnya *itemset* yang ditemukan.

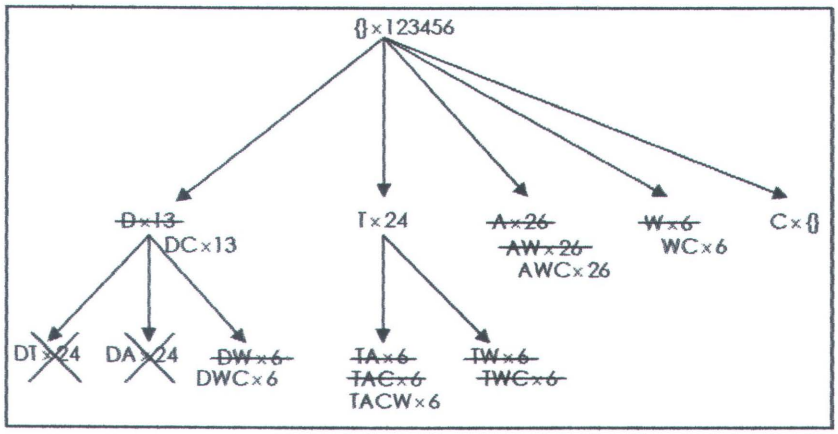
Operasi pembedaan dan pengujian subset

Dengan asumsi bahwa basis data awal disimpan dalam format *tidset*, namun pemrosesan selanjutnya dilakukan dalam format *diffset*. Dengan tersedianya *diffset* sebagai ganti *tidset* untuk tiap-tiap *itemset*, terdapat permasalahan baru mengenai bagaimana cara menghitung perbedaan kombinasi *itemset* yang baru terbentuk dari induknya secara langsung. Yang perlu dilakukan adalah melakukan pemindaian secara linear terhadap kedua *diffset*, menyimpan *tids* dalam salah satunya, dan tidak pada lainnya. Permasalahan utama adalah bagaimana melakukan pemrosesan subset secara efektif, yang diperlukan untuk memilih salah satu dari keempat sifat pasangan *itemset-tidset* yang bersesuaian.

Dalam mencari perbedaan dari dua himpunan, yang disimpan adalah banyaknya ketidaksesuaian dari kedua *diffset*, yaitu keadaan dimana sebuah *tid* terjadi di sebuah *itemset* dan tidak di itemset yang lain. Jika $m(X_i)$ dan $m(X_j)$ menyatakan banyaknya ketidaksesuaian pada *diffset* $d(X_i)$ dan $d(X_j)$, maka ada 4 keadaan yang mungkin terjadi:

1. Kemungkinan 1. $m(X_i) = 0$ dan $m(X_j) = 0$, maka $d(X_i) = d(X_j)$ atau $t(X_i) = t(X_j)$.
2. Kemungkinan 2. $m(X_i) > 0$ dan $m(X_j) = 0$, maka $d(X_i) \supset d(X_j)$ atau $t(X_i) \subset t(X_j)$.
3. Kemungkinan 3. $m(X_i) = 0$ dan $m(X_j) > 0$, maka $d(X_i) \subset d(X_j)$ atau $t(X_i) \supset t(X_j)$.
4. Kemungkinan 4. $m(X_i) > 0$ dan $m(X_j) > 0$, maka $d(X_i) \neq d(X_j)$ atau $t(X_i) \neq t(X_j)$.

Dengan demikian, maka dapat dilakukan perhitungan support, pengujian subset, kesamaan dan ketidaksamaan secara bersamaan dengan perhitungan *diffset* itu sendiri. Gambar 2.5 di bawah ini menunjukkan pencarian *closed set* dengan menggunakan *diffset* sebagai ganti *tidset*. Dalam proses ini, tahap demi tahapnya sama dengan jika mengunakan *tidset*, walaupun operasi pembedaan dilakukan dengan *diffset* (kecuali untuk node root yang masih menggunakan *tidset*).



Gambar 2.5 Proses pencarian dengan format *diffset*

BAB III

PENERAPAN ALGORITMA CHARM-L DALAM MENGGALI *FREQUENT CLOSED ITEMSET*

BAB III

PENERAPAN ALGORITMA CHARM-L DALAM MENGGALI *FREQUENT CLOSED ITEMSET*

3.1. PENYIAPAN DATA ITEM

Proses penyiapan data item ini meliputi proses pembentukan item utama dan proses pengurutan item utama.

3.1.1. Pembentukan Item Utama

Penggalian *frequent closed itemset* dimulai dengan membentuk item utama terlebih dahulu, yaitu *1-itemset* yang mempunyai nilai *support* ≥ *minimum support*, dinotasikan sebagai F_1. Kandidat item utama ini didapatkan dari pembacaan masukan basis data transaksional, dan dipilih mana yang memenuhi ambang *minimum support*. Kandidat terpilih inilah yang menjadi item utama (F_1), yang akan digunakan untuk pemrosesan lebih lanjut. Sebagai contoh, digunakan kembali basis data transaksional pada Tabel 2.1, dituliskan kembali pada Tabel 3.1.

Tabel 3.1 Basis data contoh

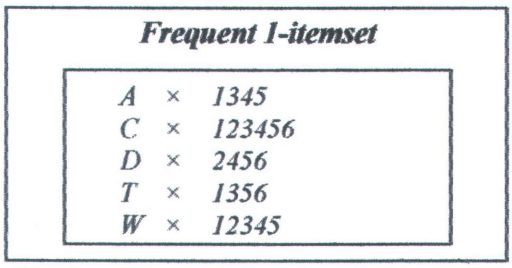
Transaksi	Item
1	*ACTW*
2	*CDW*
3	*ACTW*
4	*ACDW*
5	*ACDTW*
6	*CDT*

Dari pembacaan pada basis data contoh di atas, maka didapatkan 5 kandidat item utama yaitu $I = \{A, C, D, T, W\}$. Nilai support dari masing-masing kandidat item utama di atas adalah sebagai berikut:

- Item *A* : 4 transaksi
- Item *C* : 6 transaksi

- Item *D* : 4 transaksi
- Item *T* : 4 transaksi
- Item *W* : 5 transaksi

Jika ambang batas minimum support = 50% (3 transaksi), maka kelima item diatas memenuhi syarat tersebut, sehingga kelima kandidat item diatas merupakan item utama / *frequent 1-itemset* (F_1). Tiap-tiap anggota F_1 merupakan pasangan *itemset-tidset* (*IT-pair*), dituliskan seperti gambar 3.1 berikut ini.



Gambar 3.1 *Frequent 1-itemset* : *IT-pair*

Dalam algoritma CHARM-L ini digunakan *diffset* sebagai format data vertikal untuk menggantikan nilai *tidset* tiap *itemset*. Karena himpunan *frequent itemset* dapat digambarkan dalam bentuk *lattice*, maka dapat dicari elemen dasarnya dengan operasi *meet itemset* dan *join tidset* dari *semua frequent 1-itemset* sehingga didapatkan elemen dasarnya adalah $\varnothing \times 123456$. Dengan demikian, F_1 ini dapat direpresentasikan dalam bentuk *equivalence class* sebagai berikut: $[\varnothing \times 123456] = \{A \times 1345, C \times 123456, D \times 2456, T \times 1356, W \times 12345\}$. Dari representasi *equivalence class* tersebut, maka dapat dicari tiap-tiap *diffset* dari tiap-tiap anggota F_1, yang hasilnya dituliskan pada tabel 3.2.

Tabel 3.2 *Frequent 1-itemset* dalam format *diffset*

Itemset X	*t(P) - t(X)*	*Diffset X*
A	$t(\varnothing) - t(A) = 123456 - 1345$	*26*
C	$t(\varnothing) - t(C) = 123456 - 123456$	$\varnothing$
D	$t(\varnothing) - t(D) = 123456 - 2456$	*13*
T	$t(\varnothing) - t(T) = 123456 - 1356$	*24*
W	$t(\varnothing) - t(W) = 123456 - 12345$	*6*

3.1.2. Pengurutan *Frequent 1-itemset*

Sebelum dilakukan pencarian *frequent closed itemset* dengan algoritma CHARM-L, elemen kelas [*P*] harus diurutkan terlebih dahulu. Pada level *root*, node itemset diurutkan berdasarkan bobotnya. Dengan $x, y \in I$, didefinisikan **bobot (*weight*) dari item *x*** sebagai $w(x) = \sum_{x,y \in F_2} \sigma(xy)$, yaitu jumlah seluruh *support* dari *frequent 2-itemset* yang mengandung item *x*. Untuk perhitungan bobot, perlu dibentuk *frequent 2-itemset* terlebih dahulu. Gambar 3.2 menunjukkan kandidat dari *frequent 2-itemset* yang dituliskan dalam bentuk *IT-pair*, beserta supportnya masing-masing.

Kandidat Frequent 2-itemset

X × t(X)	Support
AC × 1345	4
AD × 45	2
AT × 135	3
AW × 1345	4
CD × 2456	4
CT × 1356	4
CW × 12345	5
DT × 56	2
DW × 245	3
TW × 135	3

Frequent 2-itemset

X × t(X)	Support
AC × 1345	4
AT × 135	3
AW × 1345	4
CD × 2456	4
CT × 1356	4
CW × 12345	5
DW × 245	3
TW × 135	3

Gambar 3.2 *Frequent 2-itemset*

Dari *frequent 2-itemset* yang telah didapatkan pada gambar 3.2, maka bobot untuk tiap-tiap F_1 dapat dihitung dan hasilnya dituliskan pada Tabel 3.3, sehingga kelas [*P*] diurutkan menjadi $\{D, T, W, A, C\}$.

Tabel 3.3 *Frequent 1-itemset* beserta bobotnya

$X \in F_1$	$XY \in F_2$	$w(X)$
A	*AC, AT, AW*	4+3+4 = 11
C	*AC, CD, CT, CW*	4+4+4+5 = 17
D	*CD, DW*	4+3 = 7
T	*AT, CT, TW*	3+4+3 = 10
W	*AW, CW, DW, TW*	4+5+3+3 = 15

3.2. PENCARIAN DENGAN ALGORITMA CHARM-L

Berikut ini akan dijelaskan proses-proses yang terjadi dalam penggalian menggunakan algoritma CHARM-L. Setelah semua proses dijelaskan, pada subbab terakhir diberikan contoh penggalian *frequent closed itemset* menggunakan algortima CHARM-L.

3.2.1. Algoritma CHARM-L

Metode pencarian yang digunakan yaitu algoritma CHARM-L yang menghasilkan *frequent closed itemset*. Algoritma ini bekerja dengan terus menerus menggali ruang *itemset* dan ruang *tidset* di dalam ruang pencarian *IT-tree*. Pencarian didasarkan pada sifat pasangan *itemset-tidset*, dengan tujuan memangkas level pencarian untuk segera menemukan *closed itemset* tanpa harus mengenumerasi setiap kandidatnya. Setiap kali CHARM-L membangkitkan sebuah node *closed itemset*, *closed itemset* tersebut ditandai dengan sebuah penanda unik yang disebut ***cid***. Dalam CHARM-L, tiap elemen $I_i \times t(I_i) \in P$ diasosiasikan dengan suatu ***cidset*** yang dinotasikan dengan ***C***, yaitu himpunan semua *cid* dari *closed itemset* yang merupakan superset dari Pl_i. Begitu suatu *closed itemset* ditemukan, algoritma ini segera membentuk struktur *lattice*nya.

```
CHARM-L (D , min_sup):
1.  [∅] = { I_i × d(I_i) : I_i ∈ I ∧ σ(I_i) ≥ min_sup };
2.  CHARM-L-EXTEND ([∅], L_c = {∅});
3.  return L //lattice dari closed sets

CHARM-L-EXTEND ([P], L_c):
4.  for each I_i × d(I_i), C(I_i) ∈ [P]
5.      P_i = P ∪ I_i dan [P_i] = ∅
6.      UPDATE-C (I_i,[P])
7.      for each I_j × d(I_j), C(I_j) ∈ [P], j > i
8.          X = I_j dan Y = d(I_j) - d(I_i) dan C (X) = C (I_i) ∩ C (I_j)
9.          CHARM-PROPERTY (X × Y, I_i, I_j, P_i, [P_i], [P])
10.     L_n = SUBSUMPTION-CHECK-LATTICE-GEN (L_c , P_i , C (P_i))
11.     CHARM-L-EXTEND ([P_i], L_n)
12.     delete [P_i]
```

Gambar 3.3 Algoritma CHARM-L

```
CHARM-PROPERTY (X × Y, I_i, I_j, P_i, [P_i], [P]):
13. if ( σ(X) ≥ min_sup) then
14.     if m(I_i) = 0 dan m(I_j) = 0 then          //Kondisi 1
15.         Buang elemen I_j dari [P]
16.         P_i = P_i ∪ I_j
17.     else if m(X_i) > 0 dan m(X_j) = 0 then     //Kondisi 2
18.         P_i = P_i ∪ I_j
19.     else if m(X_i) = 0 dan m(X_j) > 0 then     //Kondisi 3
20.         Buang elemen I_j dari [P]
21.         X × Y masuk pada [P_i]                 //gunakan fungsi pengurutan f
22.     else if m(X_i) > 0 dan m(X_j) > 0 then     //Kondisi 4
23.         X × Y masuk pada [P_i]                 //gunakan fungsi pengurutan f
```

Gambar 3.4 Metode CHARM-PROPERTY

Algoritma CHARM-L ini dituliskan dalam bentuk *pseudocode* seperti pada gambar 3.3 di atas. Dengan L menyatakan ***closed itemset lattice*** dan L_r adalah ***node root*** **dari** ***lattice***, diinisialisasi $L_r = \varnothing$. Metode CHARM-L ini bekerja sebagai berikut :

- CHARM-L dimulai dengan inisialisasi kelas *parent* $[P]$ dengan *frequent item* tunggal (Baris 1). Elemen $[P]$ ini diurutkan terlebih dahulu dengan fungsi pengurutan f.
- Dilakukan pemanggilan pada prosedur CHARM-L-EXTEND dengan parameter *parent equivalence class* $[P]$ dan *lattice* root L_r (Baris 2). Pemrosesan utama dalam algoritma CHARM-L ini dilakukan pada prosedur CHARM-L-EXTEND ini.

CHARM-L-EXTEND mengambil L_c, *node lattice* dalam proses (yang diinisialisasi dengan *node root*), dan sebuah *equivalence class* dari *IT-pair*, $[P]$. CHARM-L-EXTEND bekerja dengan memeriksa tiap kombinasi *IT-pair* yang menjadi anggota kelas $[P]$, sebagai berikut.

Pada Gambar 3.3 baris 5, tiap I_i membangkitkan sebuah prefiks baru $P_i = P \cup I_i$, dengan kelas $[P_i]$, yang diinisialisasi kosong.

MILIK PERPUSTAKAAN ITS

Pada baris 8, kedua *IT-pair* dikombinasikan untuk menghasilkan pasangan baru $X \times Y$, dengan $X = l_j$ dan $Y = t(l_i) \cap t(l_j)$. Jika terdapat $C(l_i)$ dan $C(l_j)$, maka dapat diperoleh himpunan *closed itemset* yang mengandung Pl_i dan Pl_j dengan cara mengiriskan kedua *cidset*, yaitu $C(X) = C(l_i) \cap C(l_j)$, seperti yang dilakukan pada baris 8.

Melalui CHARM-PROPERTY, tiap *IT-pair* $l_i \times t(l_i)$ (baris 4), akan dikombinasikan dengan *IT-pair* lain $l_j \times t(l_j)$ sesudahnya (baris 7), sesuai dengan fungsi pengurutan *f*. Metode CHARM-PROPERTY ini dituliskan dalam bentuk pseudocode pada gambar 3.4.

Dengan metode SUBSUMPTION-CHECK-LATTICE-GEN, CHARM-L mengenumerasi semua *closed itemset* yang tidak termasuk dalam *closed itemset* lain (baris 10). Fungsi ini juga membangkitkan *node lattice* baru L_n untuk *closed set* yang baru terbentuk dan menempatkannya pada posisi yang tepat dalam *closed itemset lattice L*. Node lattice baru L_n ini menjadi node dalam proses pada pemanggilan rekursif CHARM-L-EXTEND selanjutnya (baris 11).

Karena daftar *closed superset* dari l_i dapat berubah ketika sebuah *closed itemset* ditambahkan ke dalam *lattice*, maka suatu pengujian dilakukan pada baris 6 untuk melakukan pembaharuan nilai pada $C(l_i)$ untuk setiap elemen yang tersisa dalam kelas [*P*] dengan pemanggilan prosedur UPDATE-C.

3.2.2. Subsumption Check dan Pembangkitan *Lattice*

Sesuai dengan teori pada subbab 2.5, untuk memeriksa apakah sebuah *itemset* P_i bersifat *closed* (Gambar 3.3 baris 10), maka dilakukan pemanggilan pada metode SUBSUMPTION-CHECK-LATTICE-GEN, yang urutan prosesnya dituliskan dalam bentuk *pseudocode* pada gambar 3.5 berikut ini.

```
SUBSUMPTION-CHECK-LATTICE-GEN (Lc , X, C (X)):
1.  S = {Z ∈ C | Z.cid ∈ C (X)}
    //eliminate subsumed itemsets
2.  for each Z ∈ S do
3.      if σ(X) = σ(Z) then return Lc.
//X menjadi child dari Lc
4.  Ln = X
5.  Lc.children.add(Ln) , Ln.parents.add(Lc)
//Penyesuaian Lattice
6.  S^min = { Z ∈ S|Z is Minimal}
7.  for all Z ∈ S^min do
8.      Ln.children.add(Z), Z.parents.add(Ln)
9.      for all Zp ∈ Z.parents do
10.         if Zp⊂ Ln then
11.             Zp.children.remove(Z), Z.parents.remove(Zp)
12. return Ln
```

Gambar 3.5 *Subsumption checking* dan pembangkitan *lattice*

Metode ini mengambil *node lattice* dalam proses, *itemset X* yang baru terbentuk dan *cidset C(X)* sebagai masukan. Langkah pertama yang dilakukan yaitu memeriksa apakah *X* sudah **termasuk/terliputi** (***subsume***) dalam *itemset* lain. Untuk itu, dicari semua *closed itemset S* yang merupakan *superset* dari *X*. Jika *X* mempunyai *support* yang sama dengan *support superset* $Z \in S$ manapun, maka *X* sudah termasuk sehingga proses berhenti dan fungsi ini mengembalikan nilai L_c (Baris 3). Jika tidak, sebuah node *lattice* baru diinisialisasi sebagai $L_n = X$ (Baris 4).

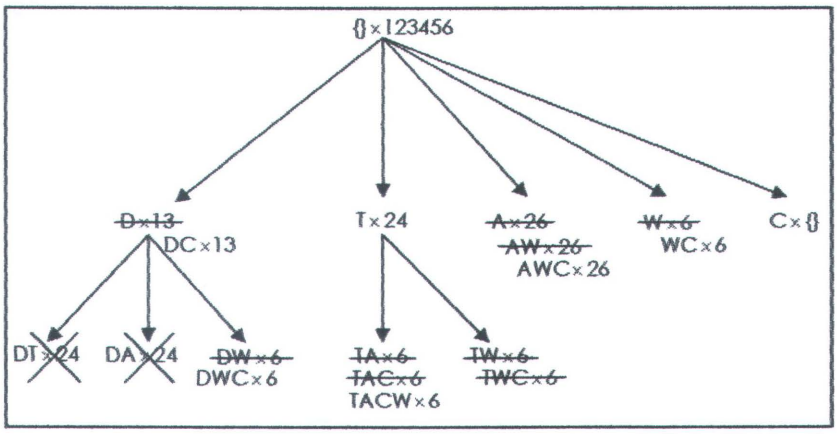
Untuk membangkitkan struktur *lattice* dari *node lattice* baru tersebut, maka dicari kumpulan ***parents*** (*subset* langsungnya), dan kumpulan ***children*** (*superset* langsungnya). Node baru L_n ditambahkan sebagai *child* dari node dalam proses, L_c, dan L_c sebagai *parent* dari L_n (Baris 5). Dari himpunan semua *closed superset* dari L_n (misalnya $L_n = X$), maka dapat dicari *minimal superset*, S^{min} (Baris 6). Tiap-tiap *minimal superset* $Z \in S^{min}$ kemudian menjadi *child* dari L_n (dan L_n menjadi *parent* dari *Z*) (Baris 8). Langkah terakhir, untuk setiap *parent* dari *Z*, dinotasikan dengan Z_p, maka penunjuk *childern*nya harus disesuaikan; *Z* dibuang

dari daftar *children* dari Z_p (dan Z_p dari daftar parent Z) (Baris 9-10). Kemudian, *lattice node* L_n menjadi nilai kembalian untuk metode ini.

3.2.3. Mengupdate C

Prosedur *update-C* pada algoritma CHARM-L (Baris 6) dijalankan karena setelah pemanggilan secara rekursif pada prosedur CHARM-L-EXTEND (Baris 11), maka *closed set* yang baru mungkin telah dibangkitkan. Dengan adanya *closed set* baru tersebut, maka perlu dilakukan pembaruan nilai *cidset* untuk setiap *IT-pair* yang masih tersisa dalam kelas [*P*]. Dapat dituliskan sebagai berikut, untuk setiap *IT-pair* $l_j \times t(l_j) \in [P]$, dengan $l_j \geq_f l_i$, UPDATE-*C* menambahkan semua *cids* dari semua *closed set* yang baru dibangkitkan pada $C(l_j)$.

3.2.4. Contoh penggalian dengan algoritma CHARM-L



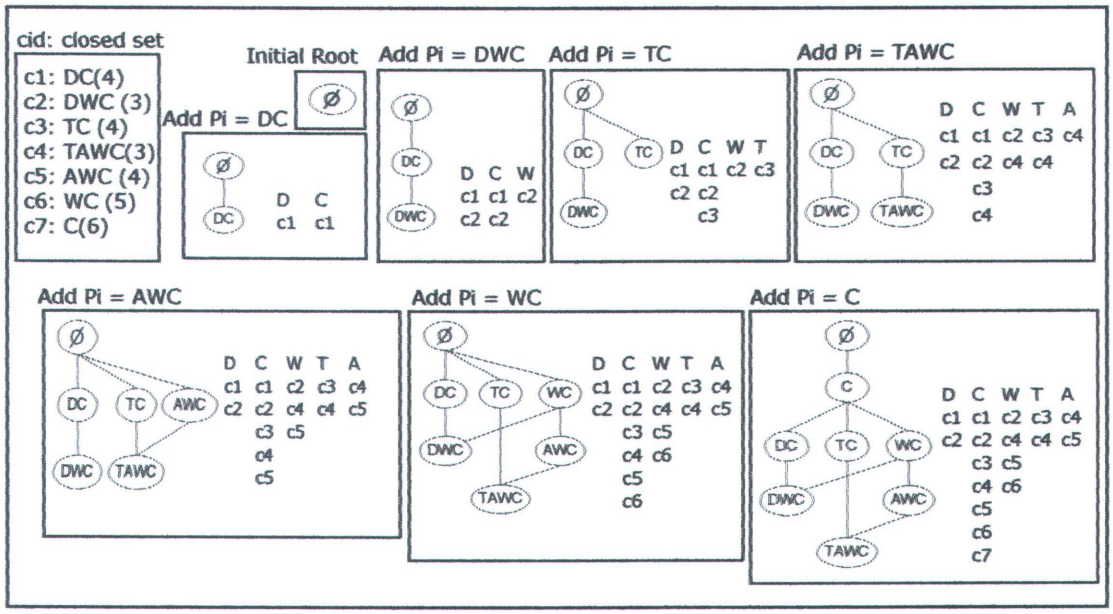
Gambar 3.6 Proses pencarian dengan format diffset

Pada gambar 3.6 ditunjukkan bagaimana algoritma CHARM-L bekerja untuk basis data contoh pada Tabel 3.1. Berikut merupakan urut-urutan proses dalam menggali *frequent closed itemset* untuk basis data contoh tersebut.

- Metode CHARM-L (Gambar 3.3 baris 1) menginisialisasi kelas root dengan F_1 yang telah diurutkan, $[\varnothing] = \{D\times13, T\times24, A\times26, W\times6, C\times\varnothing\}$.

- Prosedur CHARM-L-EXTEND pada baris 4 memproses node *D×13*, sehingga pada baris 5 nilai $P_i = D$.
- Pada Baris 7, nilai P_i dikombinasikan dengan elemen berikutnya. *DT* dan *DA* merupakan *infrequent itemset*, maka keduanya dipangkas (Gambar 3.6).
- Pemrosesan selanjutnya pada node *D* dan node *W*. Pada prosedur CHARM-PROPERTY, $m(D) > 0$ dan $m(W) > 0$. Karena Kondisi 4 tercapai, nilai *W* dimasukkan ke *equivalence class* [*D*] (Baris 23), sehingga nilai [*D*] = {*W*}.
- Pemrosesan selanjutnya pada node D dan node C, dan didapatkan $m(D) > 0$ dan $m(C) = 0$, sehingga dalam kasus ini berlaku Kondisi 2. Hal ini berakibat pada pengantian elemen *D* dengan *DC*, yang juga mengubah kelas [*D*] dengan [*DC*], dan [*DW*] menjadi [*DWC*].
- DC tidak dapat diperluas lagi kemudian ditambahkan ke *lattice*. Pada pemanggilan rekursif CHARM-L-EXTEND berikutnya dengan kelas [*DC*], hanya terdapat 1 elemen *DWC* sehingga proses melakukan *subsumption check* (baris 10) dan *DWC* ditambahkan ke *lattice*. Ketika proses kembali ke node *D* (sekarang *DC*), maka cabang *IT-tree* tersebut terselesaikan.
- Pada pemrosesan node *T* dan node *A*, $m(T) > 0$ dan $m(A) > 0$, berlaku Kondisi 4 sehingga *A* diinsertkan ke kelas [*T*].
- Pada pemrosesan node *T* dan node *W*, $m(T) > 0$ dan $m(W) > 0$ sehingga kelas [*T*] menjadi [*T*] = {*A*, *W*}. Kemudian didapatkan $m(T) > 0$ dan $m(C) = 0$, setiap *T* digantikan oleh *TC*, sehingga [*T*] berubah menjadi [*TC*] = {*A*, *W*}.
- CHARM-L kemudian melakukan subsumption check untuk node *TC* dan *TC* ditambahkan ke *lattice*. Pada pemanggilan rekursif untuk memproses [*TC*], *TAC* dikombinasikan dengan *TWC* yang menghasilkan $m(TAC) = 0$ dan $m(TWC) = 0$. Kondisi 1 terpenuhi, berakibat *TAC* menjadi *TACW*, dan *TWC* dibuang. Karena *TACW* tidak dapat diperluas lagi, maka *TACW* ditambahkan ke *lattice* dan cabang *TC* selesai diproses.
- Pemrosesan cabang lainnya menunjukkan terjadinya Kondisi 2, sehingga tidak lagi dilakukan pemanggilan rekursif terhadap CHARM-L-EXTEND, dan semua *closed frequent itemset* yang didapatkan dari proses ini ditunjukkan dengan *IT-pair* yang tidak dicoret pada gambar 3.6.

Pembangunan *closed itemset lattice* dilakukan dalam metode SUBSUMPTION-CHECK-LATTICE-GEN, ditunjukkan pada gambar 3.7. Ketika sebuah *closed set* ditemukan, dilakukan pemberian nilai *cid* dan nilai tersebut dimasukkan pada *lookup table*, gambar 3.7 kotak paling kiri. Pada inisialisasi awal, *lattice* hanya mempunyai root $L_r = \varnothing$, dan nilai *cids* untuk semua item masih kosong. *Closed itemset* pertama yang ditambahkan dalah $P_i = DC$, dengan node dalam proses $L_c = L_r$, sehingga ditambahkan $L_n = DC$ sebagai child dan memperbarui nilai *cidset* dari C dan D, seperti yang tampak pada gambar 3.6 (dengan C(C) = {c1} dan C(D) = {c2}). Kemudian ditambahkan $P_i = DWC$, dengan $L_c = DC$, DWC ditambahkan sebagai node baru L_n dan menjadi child dari DC; nilai *cidset*nya kemudian diperbarui. Proses berlanjut terus hingga semua *closed set* ditemukan sehingga tampak seperti gambar 3.7, kotak paling kiri bawah.



Gambar 3.7 Pertumbuhan lattice dan cidsetnya

BAB IV

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK

BAB IV

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK

Dalam bab ini dijelaskan mengenai tahapan proses perancangan dan pembuatan perangkat lunak. Perancangan dan pembuatan perangkat lunak ini meliputi data yang digunakan, alur proses yang terjadi, antarmuka untuk pengguna, struktur data serta fungsi-fungsi yang digunakan dalam perangkat lunak.

4.1. ANALISA KEBUTUHAN SISTEM

Dari penjelasan pada bab 2 dan bab 3 mengenai penggalian *frequent closed itemset* dan algoritma CHARM-L, maka dapat diketahui bahwa aplikasi ini harus memenuhi hal-hal berikut ini :

1. aplikasi ini membutuhkan masukan dari pengguna berupa tabel transaksi dari basis data, dan parameter *minimum support* sebagai batasan aplikasi.
2. aplikasi ini menghasilkan keluaran berupa sebuah file teks yang memuat daftar seluruh *frequent closed itemset* yang dihasilkan selama proses penggalian.
3. aplikasi ini maupun hasil keluarannya sebaiknya memiliki sifat *usability*, sehingga pengguna dapat mengoperasikan aplikasi dengan lebih mudah.

4.2. PERANCANGAN PERANGKAT LUNAK

Sub bab ini terdiri dari 3 bagian yaitu perancangan data, perancangan proses, dan perancangan antar muka untuk pengguna. Berikut ini dijelaskan masing-masing bagian tersebut.

4.2.1. Perancangan Data

Tujuan dari perancangan data adalah untuk mengetahui data seperti apa yang dibutuhkan untuk pemrosesan perangkat lunak. Perancangan data ini digolongkan

menjadi 3 bagian yaitu perancangan data masukan, perancangan data proses, dan perancangan data luaran.

4.2.1.1. Data masukan

Data masukan yang digunakan adalah tabel dari data transaksi pembelian yang diambil dari suatu basis data. Data yang diperlukan ini terdiri atas 2 kolom, sebuah kolom penanda transaksi (*tid*), dan sebuah kolom item. Sebuah tabel transaksi pembelian dapat terdiri dari beberapa jenis item, pengguna dapat bebas memilih kolom mana yang ingin digunakan. Untuk mempermudah proses data mining, maka tabel data transaksi yang akan digunakan harus dipersiapkan dahulu dalam format yang sesuai. Sebagai contoh data masukan, data transaksi pada Tabel 3.1 harus diubah terlebih dahulu sehingga formatnya menjadi seperti Tabel 4.1 di bawah ini.

Tabel 4.1 Perancangan data transaksional

TID	ITEM
1	A
1	C
1	T
1	W
2	C
2	D
2	W
3	A
3	C
3	T
3	W
4	A
...	...

Data masukan lainnya yang diperlukan adalah parameter berupa nilai support minimal yang diinginkan oleh pengguna, sebagai batasan aplikasi. Selain itu penguna juga harus menspesifikasikan tempat penyimpanan file teks sebagai file keluaran dari aplikasi ini.

4.2.1.2. Data proses

Data proses adalah data yang digunakan selama proses berjalan. Dalam perangkat lunak ini akan digunakan beberapa data proses, antara lain data hasil pemindaian basis data, data item utama, dan data *frequent itemset*. Berikut ini akan dijelaskan ketiga macam data proses tersebut.

1. **Data hasil pemindaian basis data**

   Data yang dihasilkan dari pemindaian basis data adalah tidset dari semua item yang terdapat dalam tabel transaksi. Data tidset ini dinyatakan dalam deretan bit, dengan urutan bit dari kiri ke kanan menyatakan transaksi dari item. Sebagai contoh, tidset dari item A adalah (010111) dan tidset dari item C adalah (0111111). Data hasil pemindaian ini disimpan dalam sebuah tabel Item_Utama.

   Adapun tabel Item_Utama mempunyai karakteristik sebagai berikut:

   - Level item. Karena data yang disimpan dalam tabel ini merupakan item utama, maka level dari semua item yang tersimpan tersebut adalah '1'.
   - Nama item. Menyimpan nama dari suatu item.
   - Tidset item. Menyimpan informasi mengenai transaksi-tansaksi mana yang memuat item tersebut.
   - Support item. Menyimpan suppot dari masing-masing item, yang didapatkan dari banyaknya tidset dari item tersebut

2. **Data *frequent itemset***

   Data *frequent itemset* ini menyimpan informasi dari setiap kumpulan item yang memenuhi batas nilai support minimal. Data ini meliputi data item utama dan data *frequent n-itemset*.

   Data item utama merupakan data hasil pemindaian basis data yang telah terseleksi berdasarkan batas minimal support yang telah diberikan oleh pengguna, karena itu item data ini disebut juga dengan *frequent 1-itemset* (F_1).

Karena item-item F_1 ini didapatkan dari seleksi pada data hasil pemindaian, maka pada mulanya data F_1 ini masih dalam format tidset, padahal untuk menggali *frequent closed itemset* dengan algoritma CHARM-L, digunakan format diffset sebagai pengganti tidset.

Oleh karena itu perlu dilakukan proses pengubahan tidset menjadi diffset sebelum data F_1 ini diproses lebih lanjut. Sedangkan data *frequent n-itemset*, dengan n>1, merupakan kumpulan item yang supportnya memenuhi batas minimum support. Data *frequent n-itemset* ini sudah disimpan dalam format diffset.

3. **Data node *IT-tree***

   Data *node IT-Tree* ini menyimpan informasi karakteristik dari setiap *node IT-Tree*, sebagai representasi ruang pencarian dalam algoritma ini. *Node IT-tree* ini pada dasarnya merupakan *equivalence class*, karakteristik *node IT-Tree* ini yaitu *node* itu sendiri merupakan sebuah *frequent itemset*, dengan *childrem*nya yang masing-masing merupakan itemset tunggal (*1-itemset*).

4. **Data node *lattice***

   Data *node lattice* ini menyimpan informasi karakteristik dari setiap *node frequent closed itemset lattice*, sebagai representasi struktur *lattice* yang merupakan hasil dari penggalian ini. *Node lattice* ini sendiri merupakan sebuah *frequent itemset*, dan mempunyai *parents* dan *children*.

4.2.1.3. Data luaran

Data luaran yang dihasilkan berupa daftar *frequent closed set* yang sesuai dengan spesifikasi minimum support yang ditentukan oleh pengguna. Untuk memperjelas hasil keluaran, dituliskan beberapa informasi yang sesuai, antara lain: karakteristik dari tabel data masukan, minimum support yang dispesifikasikan, dan informasi waktu untuk masing-masing proses yang dilakukan dalam perangkat lunak ini. Selain ditampilkan dalam antarmuka pengguna, data luaran ini akan disimpan

dalam sebuah file teks. Berikut ini dijelaskan tiap-tiap data luaran yang akan diberikan kepada pengguna, yaitu:

a. **Nama tabel data transaksi**

   Merupakan nama tabel data transaksi yang diberikan oleh pengguna

b. **Jumlah transaksi**

   Menyatakan banyaknya transaksi yang ada dalam tabel data trasaksi

c. **Jumlah item**

   Menyatakan banyaknya item yang terjadi dalam data transaksi

d. **Jumlah record**

   Menyatakan jumlah record atau baris tabel data transaksi

e. **Minimum support**

   Merupakan nilai minimum support yang diberikan oleh pengguna

f. **Durasi pemindaian data**

   Menyatakan waktu yang dibutuhkan oleh perangkat lunak untuk melakukan proses pemindaian dengan pemanggilan *method* `Scanning()`.

g. **Durasi penyaringan data**

   Menyatakan waktu yang digunakan oleh perangkat lunak untuk melakukan penyaringan data berdasarkan nilai minimum support yang diberikan pengguna. Penyaringan data ini dilakukan dengan pemanggilan *method* Filtering().

h. **Durasi pengurutan data**

   Menyatakan lamanya waktu yang digunakan untuk menjalankan proses pengurutan Item Utama dengan pemanggilan *method* Ordering F1().

i. **Durasi proses penggalian**

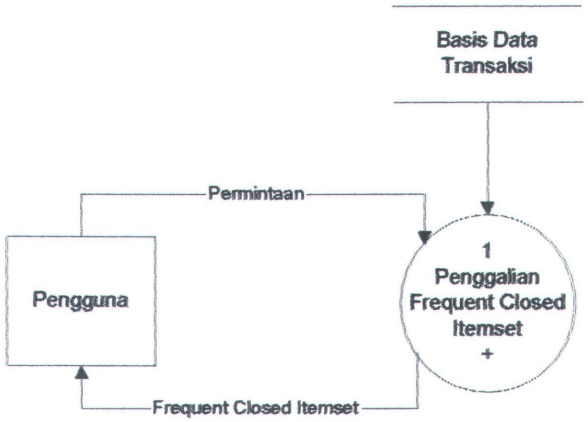
   Menyatakan lamanya waktu yang digunakan oleh perangkat lunak untuk melakukan penggalian *frequent closed itemset* dengan menggunakan algoritma CHARM-L.

j. **Daftar *Frequent closed itemset***

   Merupakan daftar *frequent closed itemset* yang dihasilkan oleh proses penggalian dengan algoritma CHARM-L. *Frequent closed itemset* ini ditampilkan berdasarkan tingkatnya.

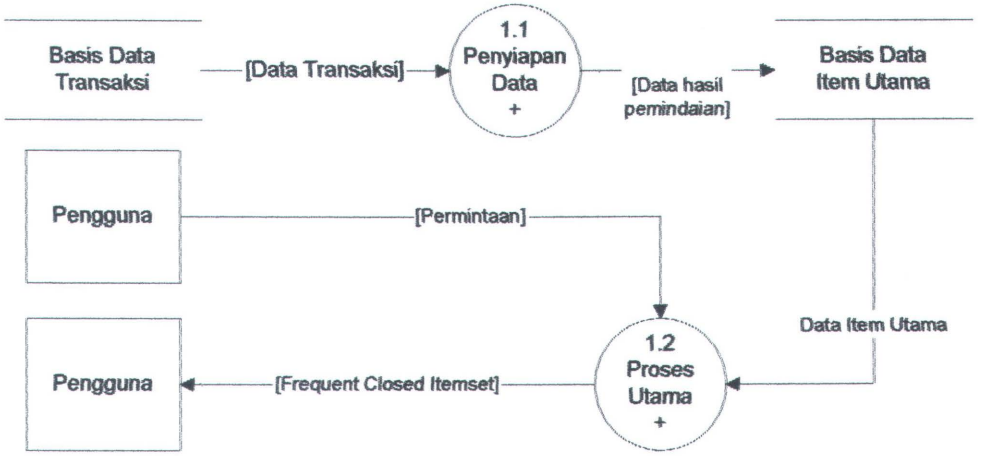
4.2.2. Perancangan Proses

Perancangan proses dari perangkat lunak ini digambarkan dalam bentuk data flow diagram atau Diagram Aliran Data (DAD). Diagram aliran data ini menjelaskan proses-proses yang digunakan pada tugas akhir ini dari tingkat 1 sampai dengan tingkat 3.



Gambar 4.1 DAD level 1

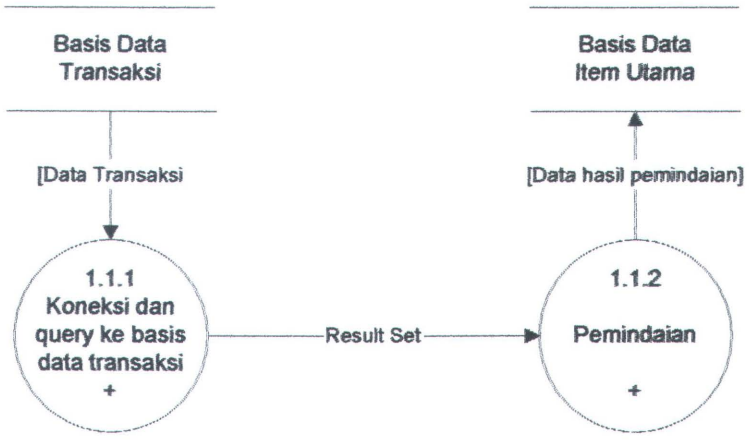
Diagram untuk penggalian *frequent closed itemset* pada tingkat 1 dapat dilihat pada gambar 4.1. Proses dimulai dengan adanya masukan dari pengguna berupa data transaksi yang akan digunakan dan nilai support minimal. Selanjutnya, proses penggalian *frequent closed itemset* dapat dilakukan. Dari proses tersebut akan dihasilkan semua *frequent closed itemset* yang sesuai dengan permintaan pengguna.



Gambar 4.2 DAD level 2

DAD level 2 dapat dilihat pada gambar 4.2. Pada diagram ini ditunjukkan bahwa proses penggalian *frequent closed itemset* pada tingkat 1 dijabarkan menjadi 2 buah proses, yaitu proses penyiapan data dan proses utama. Proses penyiapan data akan merubah data transaksi awal menjadi data masukan yang sesuai untuk proses utama. Data yang dihasilkan dari proses penyiapan data ini disimpan dalam basis data Item Utama. Basis data inilah yang dijadikan masukan bagi proses utama dimana penggalian *frequent closed itemset* dilakukan berdasarkan permintaan dari pengguna.

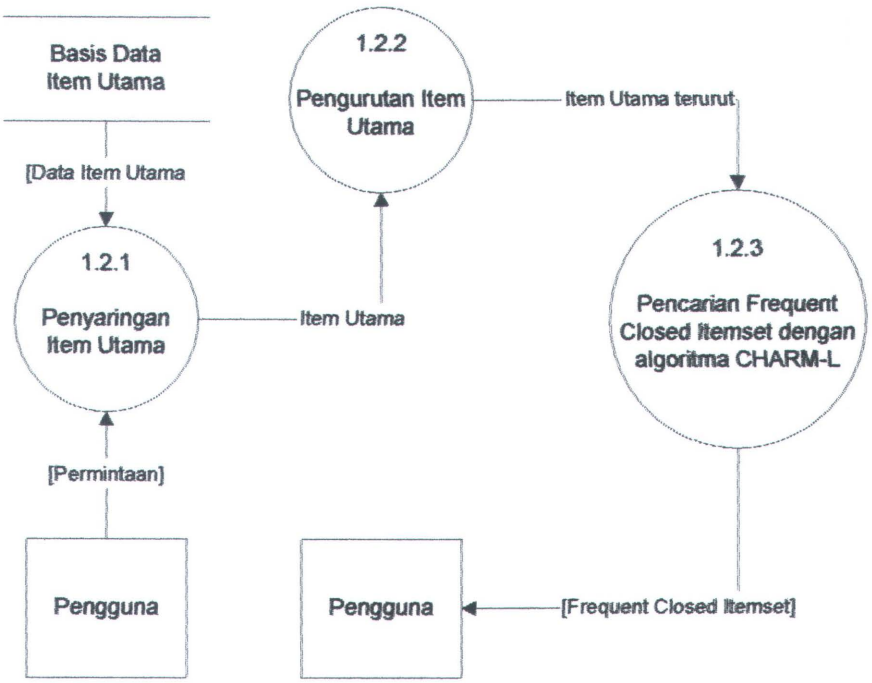
Penjabaran yang lebih detil dari proses penyiapan data dapat dilihat pada gambar 4.3. Proses penyiapan data ini diawali dengan dilakukannya koneksi dan *query* ke basis data. Proses *query* ini dilakukan untuk mencari item-item pada basis data. Pada proses pemindaian basis data, hasil query dari proses sebelumnya akan diproses kemudian disimpan dalam tabel Item_Utama. Tabel inilah yang akan dipergunakan untuk proses utama.



Gambar 4.3 DAD proses penyiapan data level 2

Proses selanjutnya yang yaitu penggalian *frequent closed itemset* seperti ditunjukkan pada gambar 4.4, yang merupakan penjabaran dari proses utama pada level 2. Proses utama ini meliputi proses-proses ini telah dijelaskan pada bab 3. Pada DAD proses utama ini terdapat 2 proses yaitu proses pembentukan item utama, proses pengurutan item utama dan proses penggalian dengan algoritma

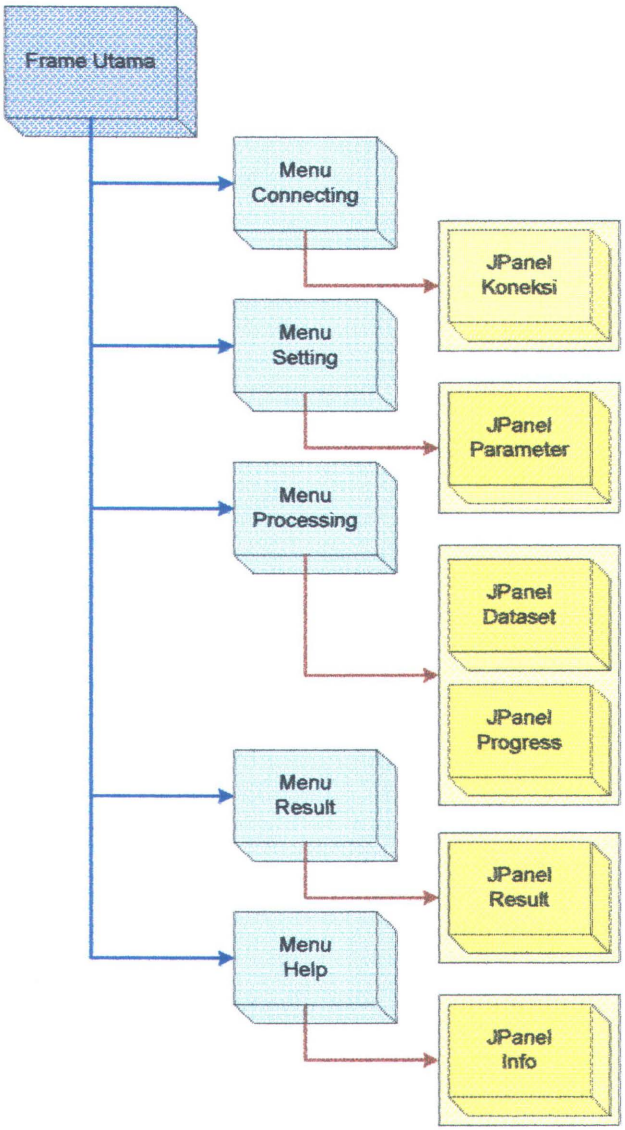
CHARM-L. Dengan menggunakan data Itemset dan permintaan dari pengguna, proses pembentukan item utama akan mencari item-item pada basis data itemset yang memenuhi nilai support minimal. Hasil dari proses ini diurutkan terlabih dahulu, kemudian hasil pengurutan ini akan menjadi masukan pada proses pencarian *frequent closed itemset* dengan algoritma CHARM-L. Dari proses inilah nantinya akan dihasilkan daftar *frequent closed itemset* yang memenuhi nilai support minimal yang diberikan oleh pengguna.



Gambar 4.4 DAD proses utama level 2

4.2.3. Perancangan Antarmuka

Bagian ini menjelaskan mengenai perancangan antarmuka dari perangkat lunak. Struktur rancangan dari perangkat lunak ini terdiri dari 5 menu yaitu menu Connecting, menu Setting, menu Processing, menu Result dan menu Help seperti yang ditunjukkan pada Gambar 4.5.



Gambar 4.5 Desain struktur antarmuka perangkat lunak

- Menu Connecting

  Dalam menu ini, pengguna diminta untuk mengisikan parameter-parameter yang digunakan untuk melakukan koneksi ke basis data untuk mendapatkan data transaksi yang akan diproses.

- Menu Setting

  Menu ini digunakan pengguna untuk memilih tabel yang digunakan sebagai data transaksi, beserta kolom transaksi dan kolom itemnya, dan juga untuk

menentukan tempat penyimpanan file yang akan digunakan untuk menyimpan keseluruhan hasil *frequent closed itemset* yang dihasilkan.

- Menu Processing

  Menu ini menampilkan informasi mengenai semua parameter yang telah dimasukkan oleh pengguna pada menu-menu sebelumnya, dan akan memulai proses penggalian frequent closed set jika tombol Processing ditekan. Pengguna juga dapat melihat informasi sejauh mana penggalian telah dilakukan dari progress bar di menu ini.

- Menu Result

  Menu ini menampilkan hasil penggalian *frequent closed itemset* beserta informasi mengenai parameter yang telah dimasukkan oleh pengguna untuk memperjelas dari hasil keluaran.

- Menu Help

  Menu ini menampilkan informasi langkah-langkah untuk menggunakan perangkat lunak ini.

4.3. PEMBUATAN PERANGKAT LUNAK

Berdasarkan rancangan yang dibuat, berikut dijelaskan mengenai lingkungan implementasi yang digunakan, implementasi data, implementasi proses, dan implementasi antarmuka dari perangkat lunak.

4.3.1. Lingkungan Implementasi

Implementasi perangkat lunak dilakukan menggunakan bahasa pemrograman Java dengan kompiler j2sdk1.5.0_02 dan editor Eclipse Platform versi 3.0.

4.3.2. Implementasi Data

Pada tahap implementasi data dibagi menjadi tiga bagian, yaitu data masukan, data proses, dan data luaran. Setiap bagian implementasi data akan dijelaskan pada subbab-subbab berikut.

4.3.2.1. Implementasi Data Masukan

Berdasarkan rancangan pada gambar 4.1 maka untuk implementasi data masukan, data dibagi menjadi dua bagian menurut asal data, yaitu :

a. Data Masukan Pengguna

Data ini merupakan data masukan yang diberikan oleh pengguna ke dalam perangkat lunak. Data masukan yang diperlukan oleh perangkat lunak dari pengguna yaitu data untuk melakukan koneksi basis data, nama tabel basis data sebagai sumber data transaksi, data persentase minimum support sebagai parameter batasan perangkat lunak, dan data lokasi penyimpanan file keluaran. Implementasi keempat jenis data tersebut ke dalam perangkat lunak dijelaskan pada tabel 4.2.

Tabel 4.2 Penjelasan variabel untuk data masukan pengguna

No	Variabel	Keterangan
1	`hostTF`	*Text field* untuk mengisikan nama komputer *server* basis data
2	`sidTF`	*Text field* untuk mengisikan nama basis data pada *server*
3	`portTF`	*Text field* untuk mengisikan port komputer *server* basis data
4	`userTF`	*Text field* untuk mengisikan nama pengguna *server* basis data
5	`passTF`	*Text field* untuk mengisikan kata kunci pengguna *server* basis data
6	`tabelCombo`	*Combo box* untuk memilih nama tabel yang akan digunakan sebagai data sumber
7	`itemCombo`	*Combo box* untuk memilih nama kolom yang akan digunakan sebagai kolom item data sumber
8	`tidCombo`	*Combo box* untuk memilih nama kolom yang akan digunakan sebagai kolom tid data sumber
9	`supportTF`	*Text field* untuk mengisikan parameter batasan minimum support
10	`fileTF`	*Text field* untuk lokasi file keluaran yang diinginkan

b. Data Masukan Hasil *Query* Basis Data

Data ini didapat oleh perangkat lunak setelah mengeksekusi sebuah *query* ke dalam basis data sesuai dengan parameter koneksi basis data yang telah diberikan oleh pengguna. Terdapat tiga data masukan hasil *query*, yaitu :

- *Query* nama tabel

  Data ini adalah data nama tabel yang terdapat pada basis data pengguna. Data inilah yang digunakan pengguna untuk memilih nama tabel yang akan digunakan dalam proses penggalian itemset. Untuk mendapatkan data *query* ini, perangkat lunak mengeksekusi *method* `getTable()` pada kelas `dataset` untuk mendapatkan nilai kembalian berupa String[] yang akan menjadi masukan bagi combobox tabelCombo. Implementasi *method* ini dalam perangkat lunak ditunjukkan pada gambar 4.6.

```
1   public String[] getTable () throws SQLException {
2     rs = stmt.executeQuery("select count(tname) from tab");
3     rs.next();
4     int size = Integer.parseInt(rs.getString("count(tname)"));
5     rs = stmt.executeQuery("select * from tab");
6     int count = 0;
7     String [] result = new String [size];
8     while (rs.next()){
9       result[count] = rs.getString("tname");
10      count++;
11    }
12    return result;
13  }
```

Gambar 4.6 Kode program *method* `getTable()`

- *Query* nama kolom

  Data ini adalah nama kolom yang terdapat pada tabel yang dipilih pengguna. Data ini digunakan untuk mengambil nama kolom yang akan digunakan sebagai sumber data untuk kolom item (`combobox ItemCombo`) dan kolom tid (`combobox tidCombo`). Untuk mendapatkan data set ini, perangkat lunak harus mengeksekusi *method* `getTableColumn()`, sehingga menghasilkan nilai kembalian berupa `String[]`. Implementasi *method* `getTableColumn()` ditunjukkan pada gambar 4.7.

```
1  public String[] getTableColumn (String tab_name)
                                          throws SQLException {
2    rs = stmt.executeQuery("select * from " + tab_name);
3    int size = rs.getMetaData().getColumnCount();
4    String [] result = new String [size];
5    for(int a=1 ; a<=size ; a++) {
6      rs.next();
7      result[a-1] = rs.getMetaData().getColumnName(a);
8    }
9    rs.close();
10   return result;
11 }
```

Gambar 4.7 Kode program *method* `getTableColumn()`

- *Query* data transaksi

  Data ini adalah tabel data transaksi yang dipilih pengguna. Data ini digunakan sebagai masukan untuk proses pemindaian data. Untuk mendapatkan data ini, perangkat lunak harus mengeksekusi *method* `selectTable()`, sehingga menghasilkan nilai kembalian berupa `ResultSet`. Implementasi *method* `selectTable()` ditunjukkan pada gambar 4.8.

```
1.  public ResultSet selectTable (String tab_field, String tab_name,
                    String constrain, String order) throws SQLException {
2.     rs = stmt.executeQuery("select "+ tab_field +" from " + tab_name + "
                           where "+ constrain +" order by "+ order);
3.     return rs;
4.  }
```

Gambar 4.8 Kode program *method* `selectTable ()`

4.3.2.2. Implementasi Data Proses

Dalam melakukan penggalian *frequent closed itemset*, perangkat lunak ini membutuhkan tempat penyimpanan informasi untuk setiap *frequent itemset* yang sedang dienumerasi. Untuk keperluan tersebut, berdasarkan penjabaran pada subbab 3.2, maka dibuat 3 struktur data berdasarkan karakteristik yang diperlukan oleh perangkat lunak ini, yaitu kelas `clsFreqItemset` yang menyimpan karakteristik tiap-tiap frequent itemset, kelas `clsITTreeNode` yang menyimpan karakteristik *IT-tree*, sebagai ruang pencarian dalam algoritma ini, dan kelas `clsLatticeNode` yang menyimpan tiap-tiap *frequent closed itemset* yang ditemukan oleh perangkat lunak ini dalam struktur *lattice*.

Berikut ini akan dijelaskan mengenai masing-masing kelas struktur data serta kegunaannya.

- **clsFreqItemset**

Kelas ini digunakan untuk membuat sebuah obyek yang dapat menyimpan karakteristik dari sebuah *frequent itemset*. Karakteristik yang disimpan dalam sebuah *frequent itemset* yaitu kode itemset (`temp_kode`), diffset (`diffset`), support (`support`), level (`level`), dan cidset (`cidset`). Untuk setiap itemset yang diproses, kode itemset berfungsi untuk menyimpan informasi item-item yang membentuk sebuah itemset, misalnya untuk itemset *DC*, kode itemset akan berisi {*D*,*C*} dan untuk itemset *DWC* kode itemset akan berisi {*D*,*W*,*C*}. Level itemset menyatakan tingkatan dari itemset. Misalnya itemset *DC* memiliki level 2 dan itemset *DWC* memiliki level 3. Sedangkan cidset berfungsi untuk menyimpan informasi mengenai *frequent closed itemset* mana saja yang mengandung item tersebut. Implementasi kelas `clsFreqItemset` untuk menyimpan sifat-sifat frequent itemset ditunjukkan pada gambar 4.9.

```
1  class clsFreqItemset{
2    private int level;
3    private BitSet diffset;
4    private int support;
5    private BitSet cidset;
6    private Hashtable temp_kode;
7
8    clsFreqItemset() {
9    }
10
11   void AddCidset(int cid){
12     this.cidset.set(cid);
13   }
14 }
```

Gambar 4.9 Kode program kelas `clsFreqItemset`

- **clsITTreeNode**

Kelas ini digunakan untuk membuat sebuah obyek yang menyimpan karakteristik dari sebuah node dalam ruang pencarian *IT-tree*. Variabel yang disimpan adalah node itu sendiri dan node childrennya. Node itu sendiri disimpan dalam variabel yang diberi nama `curr_node`, sedangkan node children disimpan dalam variabel `children`. Gambar 4.10 ini menujukkan implementasi kelas `clsITTreeNode` untuk menyimpan node dalam *IT-tree*.

```
1    class clsITTreeNode{
2      private clsFreqItemset curr_node;
3      private Vector children;
4
5      clsITTreeNode(){
6      }
7
8      void addChild(clsFreqItemset child_node){
9        this.children.addElement(child_node);
10     }
11
12     void removeChild(clsFreqItemset child_node){
13         this.children.removeElement(child_node);
14     }
15   }
```

Gambar 4.10 Kode program kelas `clsITTreeNode`

- **clsLatticeNode**

Kelas ini digunakan untuk membuat sebuah obyek yang menyimpan karakteristik sebuah node dalam struktur *lattice*. Variabel yang disimpan adalah karakteristik node *frequent* itu sendiri beserta node parents dan childrennya. Karakteristik node frequent itu sendiri disimpan dalam variabel yang diawali dengan nama `curr_node_`, node parentsnya dalam variabel `parents`, sedangkan node children dalam variabel `children`. Gambar 4.11 ini menujukkan implementasi kelas cls*Lattice*Node untuk menyimpan node *lattice*.

```
1    class clsLatticeNode {
2        private Vector parents;
3        private Vector children;
4        private int curr_node_level;
5        private int curr_node_support;
6        private BitSet curr_node_cidset;
7        private Hashtable curr_node_kode;
8
9        clsLatticeNode() {
10       }
11
12       void addParent(clsLatticeNode parent_node){
13          this.parents.addElement(parent_node);
14       }
15
16       void removeParent(clsLatticeNode parent_node){
17          this.parents.removeElement(parent_node);
18       }
19
20       void addChild(clsLatticeNode child_node){
21          this.children.addElement(child_node);
22       }
23
24       void removeChild(clsLatticeNode child_node){
25          this.children.removeElement(child_node);
26       }
27   }
```

Gambar 4.11 Kode program kelas `clsLatticeNode`

4.3.2.3. Implementasi Data Luaran

Setelah proses penggalian *frequent closed itemset* selesai, perangkat lunak ini akan memberikan hasil penggalian *frequent closed itemset* kepada pengguna. Untuk keperluan tersebut, maka berikut ini dijelaskan masing-masing variabel data luaran yang digunakan untuk menyampaikan informasi hasil penggalian kepada pengguna, yaitu:

a. Informasi data transaksi

Informasi data transaksi ini berasal dari pengguna sendiri atau dari basis data, dan sebenarnya bukan merupakan hasil keluaran dari proses penggalian *frequent closed itemset*. Informasi ini tetap ditampilkan untuk memperjelas hasil penggalian. Variabel yang menyimpan informasi data transaksi ini yaitu:

- `String set_Table` : menyimpan nama tabel data transaksi
- `int numberOfTrans` : menyimpan nilai banyaknya transaksi dalam tabel data transaksi
- `int item_in_process` : menyimpan nilai banyaknya item dalam tabel transaksi
- `int numberOfRecords` : menyimpan nilai banyaknya record (baris) tabel data transaksi
- `int set_minsup` : menyimpan nilai minimum support yang dispesifikasikan pengguna

b. Data durasi pemrosesan dan data penggunaan memori

Data ini memberi informasi mengenai durasi dari masing-masing proses yang dilakukan sepanjang proses penggalian frequent closed itemset dan informasi mengenai banyaknya memori yang digunakan dalam pemrosesan. Variabel yang menyimpan informasi data transaksi ini yaitu:

- `String ScanTime` : menyimpan durasi proses pemindaian
- `String FilterTime` : menyimpan durasi proses penyaringan item utama
- `String OrderingTime` : menyimpan durasi proses pengurutan item utama
- `String CHARMLTime` : menyimpan durasi proses penggalian frequent closed itemset

- `int MaxMemUsed` : menyimpan nilai penggunaan memori maksimal selama perangkat lunak berjalan
- `int MinMemUsed` : menyimpan nilai penggunaan memori minimal selama perangkat lunak berjalan

c. **Data *frequent closed itemset***

Merupakan hasil keluaran utama dari proses penggalian ini. Data inilah yang yang memuat seluruh *frequent closed itemset* yang dihasilkan sepanjang proses penggalian., *frequent closed set* ini ditampilkan dalam tiap level. Data *frequent closed itemset* ini disimpan dalam 2 variabel, `Hashtable ClosedSet` yang menyimpan seluruh *frequent closed itemset* dan `ArrayList LookUpLevel` untuk membantu menampilkan *frequent closed itemset* dalam tiap levelnya.

4.3.3. Implementasi Proses

Implementasi proses penggalian *frequent closed itemset* oleh perangkat lunak dibagi menjadi dua bagian sesuai dengan proses yang ada dalam DAD Level 2 pada gambar 4.3, yaitu proses penyiapan data dan proses utama, yang masing-masing dijelaskan dalam subbab-subbab berikut.

4.3.3.1. Implementasi Proses Penyiapan Data

Implementasi proses penyiapan data dapat dibedakan menjadi 2 proses sesuai dengan DAD proses penyiapan data level 2, gambar 4.4, yaitu proses koneksi dan query ke basis data transaksi, dan proses pemindaian.

4.3.3.1.1. Proses koneksi dan query ke basis data transaksi

Proses penyiapan data dimulai dengan melakukan koneksi dan query ke basis data. Proses query ini bertujuan untuk mencari item-item tunggal pada basis data. Proses query ini dilakukan dengan pemanggilan method selecTable(), implementasi dari method ini dituliskan pada gambar 4.12

MILIK PERPUSTAKAAN ITS

```
1.  public ResultSet selectTable (String tab_field, String tab_name,
                        String constrain, String order) throws SQLException {
2.      rs = stmt.executeQuery("select "+ tab_field +" from " + tab_name + "
                                where "+ constrain +" order by "+ order);
3.      return rs;
4.  }
```

Gambar 4.12 Kode program *method* `selectTable ()`

4.3.3.1.2. Proses pemindaian

Proses pemindaian basis data ini dilakukan dengan pemanggilan *method* `Scanning()`.Data transaksi hasil proses query diproses agar menjadi data masukan yang sesuai untuk proses utama kemudian disimpan pada basis data item utama. Hal ini dilakukan dengan tujuan menghemat waktu pemindaian, agar pengguna dapat melakukan proses penggalian *frequent closed itemset* pada sebuah tabel dengan parameter minimum support yang berbeda-beda, tanpa perlu melakukan pemindaian data berkali-kali. Gambar 4.13 menunjukkan implementasi dari *method* `Scanning()`.

4.3.3.2. Implementasi Proses Utama

Implementasi proses utama dapat dibedakan menjadi 3 proses sesuai dengan DAD proses utama level 2, gambar 4.4, yaitu proses penyaringan item utama, proses pengurutan item utama dan proses pencarian *frequent closed itemset* dengan algoritma CHARM-L

4.3.3.2.1. Proses penyaringan Item Utama

Proses ini bertujuan mengambil data hasil pemindaian pada tabel Item Utama sesuai dengan parameter minimum support, dilakukan dengan pemanggilan *method* `filtering()`. Data yang dihasilkan dari proses `filtering` ini disebut sebagai data Item Utama. Variabel yang disimpan dalam sebuah data Item Utama yaitu variabel level item, nama item, tidset item, dan support item. Data inilah akan dijadikan masukan bagi proses penggalian *frequent closed itemset*. Kode program dari *method* `filtering()`, ditunjukkan pada Gambar 4.14.

```
1    void Scanning(String Tid, String ItemName,String table_name) {
2       //Variabel TIDLIST
3       ...
4       //Variable Scan
5       ...
6       try {
7            dataset oraData = new dataset();
8            oraData.getConnect(conn_host,conn_port,conn_sid,conn_user,
                                conn_pass);
9            numberOfRecord = oraData.getTableRows(this.set_table);
10           numberOfTrans = oraData.getTableRowsDistinct(this.set_TID,
                                                  this.set_table);
11           ResultSet Data = oraData.selectTable(set_TID+", "+set_item,
                                          set_table, set_item+",IDKEY");
12           boolean str_check = true;
13           while(Data.next()) {
14           int transaksi = Data.getInt(1);
15           NameItem = Data.getString(2);
16           if (str_check==true){
17               tesString = NameItem;
18               str_check = false;
19           }
20           boolean tes = true;
21           while (tes==true) {
22           if (NameItem.equals(tesString)){
23               int tesaja = add+maximum_bit;
24                   if (transaksi <(add+maximum_bit)){
25                     int tempcount = (int)Math.pow(2,(maximum_bit-1)-
                                                 (transaksi-add));
26                     count_konv += tempcount;
27                     supp ++; tes = false;
28                   }else{
29                     add = add+maximum_bit;
30                     tesaja = add;
31                     if (count_konv != 0) inp = (char)count_konv;
32                     else                 inp = kosong;
33                     str = str+inp; count_konv = 0;
34                   }
35           }else{
36               add = 0; inp = (char)count_konv; str = str + inp;
37               oraData.insertbig(level,item_ke,tesString,str,supp,
                                  TIDLISTBIG");item_ke = item_ke + 1;
38               tesString = NameItem;
39               supp = 0; count_konv = 0; str = "";
40           }
41           }//while
42           }  // while(RS.next())
43           Data.close();
44           inp = (char)count_konv;
45           if((int)inp==39) inp = (char)40;
46           str = str+inp;
47          oraData.insertbig(level,item_ke,tesString,str,supp,"TIDLISTBIG");
48           item_in_process = item_ke+1;
49           oraData.closing();
50           oraData = null;
51           System.gc();
52       } catch(Exception ex) {
53           tampilan.errorMsg(new String("Error occured in scanning"));
54           ex.printStackTrace();
55       }
56   }//(Scan_actionPerformed)
```

Gambar 4.13 Kode program *method* `Scanning ()`

```
1.  Vector filtering(int minsup){
2.  Vector ItemsetList = new Vector();
3.  ResultSet RS;
4.  int level,indek,support; int kode = 65;
5.  String name, String tid="";
6.  long len_clob; String substr; int bufint=0; String strtemp;
7.  BitSet tempbit = new BitSet(); Integer Int = new Integer(0);
8.
9.  try {
10. dataset oraData = new dataset();
11. oraData.getConnect(conn_host,conn_port,conn_sid,conn_user,conn_pass);
12. //setting progressbar
13. ...
14.
15. RS = oraData.selectTable("LEVELS,INDEK,NAME,SUPPORT,TIDLIST01,
        TIDLIST02","TIDLISTBIG","SUPPORT >= "+this.mintrans,
        "LEVELS,INDEK");
22. while(RS.next()) {
23. tid="";
24. SwingUtilities.invokeAndWait(runner);
25. int value = tampilan.processBar.getValue();
26. tampilan.processBar.setValue(value+1);
27. int incdif = 1;
28. Hashtable tempkode = new Hashtable();
29. tempbit = new BitSet();
30. level   = RS.getInt(1);
31. indek   = RS.getInt(2);
32. name    = RS.getString(3);
33. support = RS.getInt(4);
34. String  tid1     = RS.getString(5);
35. String  tid2     = RS.getString(6);
36. if (tid2==null)  tid = tid+tid1;
37. else                  tid = tid+tid1+tid2;
38. String kode_itemset = "";
39. kode_itemset = kode_itemset+(char)kode;
40. kode +=1;
41. for (int i=0;i< tid.length();i++){
42. bufint = (int)tid.charAt(i);
43. strtemp = Int.toBinaryString(bufint);

44. int dif = incdif * this.maximum_bit - strtemp.length();
45. for (int m=0;m<strtemp.length();m++){
       a.  if (strtemp.charAt(m)=='1'){
       b.  tempbit.set(m+dif); }
46. }//for m
47. incdif+=1;
48. }
49. tempkode.put(kode_itemset,Int);
50. Item.put(kode_itemset, name);
51. clsFreqItemset FreqItemset = new clsFreqItemset(level, tempbit,
                                        i.  support,new BitSet(),tempkode);
52. Freq01.put(kode_itemset,FreqItemset);
53. orderingF1 = false;
54. ItemsetList.add(FreqItemset);
55. }//while(RS.next())
56. RS.close(); oraData.closing(); oraData = null; System.gc();
57. }catch(SQLException e) {
58. tampilan.errorMsg(new String("Error occured in filtering"));
59. e.printStackTrace();
60. }
61. return ItemsetList;
62. }//----------filtering end----------
```

Gambar 4.14 Kode program *method* `filtering ()`

4.3.3.2.2. Proses pengurutan Item Utama

Proses ini sesuai dengan fungsi pengurutan pada subbab 3.1.2, dan diimplementasikan dalam *method* `OrderingF1()`. Pembentukan F_2 dilakukan pada baris 3 sampai dengan baris 14, setelah itu dilakukan perhitungan bobot masing-masing F_1 (baris 18-27). Setelah bobot masing-masing F1 telah didapatkan, maka F1 diurutkan berdasarkan bobotnya (baris 30-46). Data yang dihasilkan ini menjadi masukan bagi proses penggalian dengan algoritma CHARM-L. Kode program dari *method* `ordering()`, ditunjukkan pada Gambar 4.15.

```
1.  Vector orderingVector (Vector F1){
2.  //============================== construct F2
3.    Vector InputNext = (Vector)F1.clone();
4.    for (int i=0;i<(InputNext.size()-1);i++){
5.      clsFreqItemset temp01 = (clsFreqItemset)InputNext.elementAt(i);
6.      for (int j=i+1;j<InputNext.size();j++){
7.        clsFreqItemset temp02 = (clsFreqItemset)InputNext.elementAt(j);
8.         Hashtable code_temp = (Hashtable)joinCodeItemset
                              (temp01.getKode(),temp02.getKode()).clone();
9.         BitSet diff_temp = getDiffset(temp01, temp02);
10.        int support_temp = count_support(temp01, diff_temp);
11.        Integer supp_temp = new Integer(support_temp);
12.        if(support_temp > (mintrans-1))Freq02.put(code_temp,supp_temp);
13.      }//(j)
14.    }//(i)
15. //============================== construct F2 ends
16. //============================== count weight of F1
17.   for(Enumeration e = Freq01.keys(); e.hasMoreElements();) {
18.     String F01 = e.nextElement().toString();
19.     int weight = 0;
20.     for(Enumeration f = Freq02.keys(); f.hasMoreElements();) {
21.       Hashtable F2 = (Hashtable)f.nextElement();
22.       String F02 = getString(F2);
23.       if(F02.contains(F01))weight+= ((Integer)Freq02.get(F2)).intValue();
24.     }
25.     F1Weight.put(F01,new Integer(weight));
26.    }
27. //============================== count weight of F1 ends
28. //============================== ordering Vector by weight
29.   Item [] sorted_index = new Item[F1.size()];
30.   Vector item_ordered = new Vector();
31.   for (int i=0; i<F1.size(); i++){
32.     clsFreqItemset temp = (clsFreqItemset)F1.elementAt(i);
33.     Enumeration e = temp.getKode().keys();
34.     while (e.hasMoreElements()){
35.     String item_temp = e.nextElement().toString(); int weight_temp = 0;
36.     if(F1Weight.containsKey(item_temp))weight_temp =
                              ((Integer)F1Weight.get(item_temp)).intValue();
37.     sorted_index[i] = new Item(temp, weight_temp);
38.     }
39.     e = null;
40.   }
41.   Arrays.sort(sorted_index);
42.   for (int i=0; i<sorted_index.length; i++){
43.     item_ordered.addElement((clsFreqItemset)sorted_index[i].item);
44.   }
45.   return item_ordered;
46. }
```

Gambar 4.15 Kode program *method* `OrderingVector ()`

4.3.3.2.3. Proses pencarian FCI dengan algoritma CHARM-L

Setelah Item Utama telah diurutkan, maka proses penggalian *frequent closed itemset* dengan algoritma CHARM-L mulai dilakukan dengan pemanggilan *method* `ChARML()`. Berdasarkan pseudocode algortima CHARM-L pada gambar 3.3, maka proses mula-mula menginisialisasi node awal dari *IT-tree* sebagai ruang pencarian. Node awal inilah yang akan diproses untuk pertama kalinya, dengan node *childrem*nya merupakan item utama yang telah diurutkan. Selain itu dilakukan juga inisialisasi node awal dari *lattice* yang akan menjadi keluaran bagi aplikasi ini. Setelah proses inisialisasi nilai awal selesai, maka proses utama dalam algoritma ini dijalankan dengan pemanggilan *method* `ChARMLextend()`. Implementasi dari *method* `ChARML()` ini dapat dilihat pada gambar 4.16

```
1.  void ChARML(Vector f1){
2.  clsITTreeNode start = new clsITTreeNode(root);
3.  start.setChild(f1);
4.  clsLatticeNode latt = new clsLatticeNode(root);
5.  ChARMLextend(start, latt);
6.  }
```

Gambar 4.16 Kode program method `ChARML()`

Proses utama dalam algoritma CHARM-L dijalankan pada *method* `ChARMLextend()`. *Method* ini mengambil masukan berupa sebuah *node IT-tree* sebagai implementasi dari kelas prefiks, dan sebuah node *lattice* sebagai node *lattice* dalam proses. Pemanggilan *method* `ChARMLextend()` ini bertujuan untuk menilai tiap-tiap kombinasi itemset yang dihasilkan dalam *node IT-tree* tersebut apakah itemset tersebut merupakan *frequent closed itemset* atau bukan.

Dengan l_i dan l_j merupakan itemset anak dari node dalam proses, untuk setiap itemset l_i, masing-masing dikombinasikan dengan itemset berikutnya, l_j. Tiap-tiap l_i membangkitkan sebuah node baru, $P_i = P \cup l_i$, yang item anaknya diinisialisasi dengan nilai kosong. Kemudian kedua itemset tersebut dikombinasikan untuk mendapatkan itemset baru, $X \times Y$, dengan $X = l_j$ dan $Y = t(l_i) \cap t(l_j)$. Setelah itemset X didapatkan, dicari kondisi mana yang dapat dijalankan berdasarkan sifat dasar pasangan itemset-tidset dengan memanggil *method* `ChARMLproperty()`.

Method `ChARMLextend()` merupakan implementasi dari pseudocode CHARM-L-EXTEND pada gambar 3.3. Pseudocode pada gambar 3.3 baris 5 diimplementasikan pada gambar 4.17 baris 8-23. Method update-C pada pseudocode baris 6 diimplementasikan pada gambar 4.18. Pseudocode baris 8 diimplementasikan pada gambar 4.17 baris 25-37. Method CHARM-PROPERTY, gambar 3.4 baris 13-23 diimplementasikan pada gambar 4.19. Pseudocode baris 10 dan 11 diimplementasikan pada gambar 4.17 baris 44 dan 45.

```
1.  void ChARMLextend(clsITTreeNode P, clsLatticeNode Lc){
2.    clsFreqItemset P_node = P.getCurrNode();
3.    Vector child_P = (Vector)P.getChild();
4.    for(int i=0 ; i<child_P.size() ; i++){
5.      clsFreqItemset li = (clsFreqItemset)child_P.elementAt(i);
6.      //----------------------
7.      //Updating cid ....
8.      ...
9.      //Updating cid ends....
10.     //----------------------
11.     int level = li.getLevel();
12.     BitSet cid_Pi = (BitSet)li.getCidset().clone();
13.     if(P_node!=root)cid_Pi.and(P_node.getCidset());
14.     clsFreqItemset node_Pi = new clsFreqItemset();
15.     node_Pi.setKode((Hashtable)joinCodeItemset(li.getKode(),
                                                   P_node.getKode()).clone());
16.     node_Pi.setLevel(level+P_node.getLevel());
17.     node_Pi.setCidset(cid_Pi); node_Pi.setDiffset(getDiffset(P_node,li));
18.     node_Pi.setSupport(count_support(P_node,li.getDiffset()));
19.     clsITTreeNode Pi = new clsITTreeNode(node_Pi);
20.     for(int j=i+1 ; j<child_P.size() ; j++){
21.       clsFreqItemset lj = (clsFreqItemset)child_P.elementAt(j);
22.       boolean frequent = false;
23.       //cek F2 frequent pa nggak!!!!!!!!!
24.       if (P_node.equals(root) && Freq02.containsKey(joinCodeItemset
                                        (li.getKode(),lj.getKode()))){
25.         frequent=true;
26.       }else if(!P_node.equals(root)) frequent = true;
27.       if(frequent){
28.         Hashtable item_X = (Hashtable)lj.getKode().clone();
29.         BitSet diff_Y = this.getDiffset(li,lj);
30.         int supp_X = this.count_support(li, diff_Y);
31.         BitSet cid_X = (BitSet)li.getCidset().clone();
32.         cid_X.and(lj.getCidset());
33.         clsFreqItemset X = new clsFreqItemset(level, diff_Y, supp_X,
                                                  cid_X, item_X);
34.         //CHARM-PROPERTY....
35.         ...
36.         //ends of CHARM-PROPERTY
37.       }    }
38.     if(node_Pi.getSupport()>=mintrans){
39.       clsLatticeNode Ln = SubsumptionCheckLattGen(Lc, node_Pi);
40.       ChARMLextend(Pi, Ln); }
41.     Pi=null;
42.   }
43.   System.gc();
44. }
```

Gambar 4.17 Kode program *method* `ChARMLextend()`

```
===== UPDATE-C =====
1.  if(!ClosedSet.isEmpty()){
2.  for (int a=1 ; a<=ClosedSet.size() ; a++){
3.  Hashtable        closed        =        ((clsLatticeNode)ClosedSet.get(new
    Integer(a))).getKode();
4.  //update child prefiks
5.  boolean masuk = true;
6.  for (Enumeration e= P_node.getKode().keys();e.hasMoreElements();){
7.  if(!closed.containsKey(e.nextElement())){
8.  masuk = false;
9.  break;
10. }
11. }
12. if(masuk && P_node.getCidset().length()-1<a){
13. P_node.AddCidset(a);
14. }
15. //update child child IT-tree
16. for (int b=i ; b<child_P.size() ; b++ ){
17. clsFreqItemset chld = (clsFreqItemset)child_P.elementAt(b);
18. if(closed.containsKey(chld.getKode().keys().nextElement())&&
    chld.getCidset().length()-1<a){
19. chld.AddCidset(a);
20. }
21. }
22. }
23. }
```

Gambar 4.18 Kode program *method* `updateC()`

```
===== CHARM PROPERTY =====
1.  if(X.getSupport() >= mintrans){
2.    int m_i = countM(li,lj);
3.    int m_j = countM(lj,li);
4.    if(m_i==0 && m_j==0){
5.      //== Property 1 ==
6.      P.removeChild(lj);j--;
7.      node_Pi.setLevel(lj.getLevel()+node_Pi.getLevel());
8.      node_Pi.setKode(this.joinCodeItemset(node_Pi.getKode(),
                                                (Hashtable)lj.getKode()));
9.      node_Pi.getCidset().and(X.getCidset());
10.   }else if(m_i>0 && m_j==0){
11.     //== Property 2 ==
12.     node_Pi.setLevel(lj.getLevel()+node_Pi.getLevel());
13.     node_Pi.setKode(this.joinCodeItemset(node_Pi.getKode(),
                                                (Hashtable)lj.getKode()));
14.     node_Pi.getCidset().and(X.getCidset());
15.   }else if(m_i==0 && m_j>0){
16.     //== Property 3
17.     P.removeChild(lj);j--;
18.     Pi.addChild(X);
19.     Pi.setChild(orderingVector(Pi.getChild()));
20.   }else if(m_i>0 && m_j>0){
21.     //== Property 4 ==
22.     Pi.addChild(X);
23.     Pi.setChild(orderingVector(Pi.getChild()));
24.   }
25. }
```

Gambar 4.19 Kode program *method* `ChARMproperty()`

`ChARMLproperty()` bertugas untuk memotong level pencarian dengan cara memanipulasi kelas prefiks [*P*] berdasarkan sifat dasar pasangan itemset-tidset. Kelas prefiks ini diimplementasikan sebagai sebuah `ITTreeNode`. Pemanggilan `ChARMLproperty()` dapat mengubah kelas [*P*] yang sedang digunakan (Kondisi 1 dan 3), mengubah prefix P_i (Kondisi 1 dan 2), juga dapat menginsertkan *IT-pair* yang baru dibangkitkan pada kelas [P_i] (Kondisi 3 dan 4).

Untuk memeriksa apakah itemset P_i yang baru terbentuk merupakan *frequent closed itemset*, maka dilakukan pemanggilan pada *method* `SubsumptionCheckLattGen()`. *Method* ini mengambil masukan berupa sebuah itemset dan sebuah *lattice node* dalam proses. Proses pertama yang dilakukan adalah melakukan pemeriksaan apakah itemset tersebut telah termasuk dalam *frequent closed itemset* yang telah ditemukan sebelumnya. Jika itemset tersebut belum termasuk, maka sebuah *lattice node* baru akan dibentuk dan diinisialisasi sesuai dengan nilai itemset tersebut. Node *lattice* baru itu akan ditambahkan sebagai node child dari *lattice node* dalam proses. Kemudian *method* ini akan merekonstruksi ulang struktur *lattice* yang telah ada agar sesuai dengan penambahan node *lattice* baru tersebut.

Setelah selesai dengan *method* `SubsumptionCheckLattGen`, maka semua itemset l_j telah diproses. Tahap selanjutnya adalah memproses kelas prefiks P_i yang baru terbentuk secara *depth-in-first* dengan melakukan pemanggilan rekursif `ChARMLextend()`. Setelah proses tersebut selesai, maka semua itemset yang mengandung itemset P_i telah ditemukan, dan proses akan kembali ke awal `ChARMLextend()` untuk memproses item l_i berikutnya.

Method `SubsumptionCheckLattGen()` merupakan implementasi dari pseudocode pada gambar 3.5, ditunjukkan pada gambar 4.20. Pseudocode pada gambar 3.5 baris 1 diimplementasikan pada gambar 4.20 baris 3-6. Pseudocode baris 2-3 diimplementasikan pada baris 8-10. Pseudocode baris 4-5 diimplementasikan pada baris 11-19. Pseudocode baris 6 diimplementasikan pada gambar 4.20 baris 21-24.

Pseudocode baris 8 diimplementasikan pada baris 28-29. Pseudocode baris 11 diimplementasikan pada baris 35-36.

```
1.  clsLatticeNode SubsumptionCheckLattGen(clsLatticeNode Lc, clsFreqItemset
    X){
2.
3.    //=======getSuperset================================
4.    Vector S = new Vector();
5.    . . .
6.    //end of getSuperset================================
7.
8.    for (int i=0;i<S.size();i++){
9.      if(((clsLatticeNode)S.elementAt(i)).getSupport()==X.getSupport())
                                                              return Lc;
10.   }
11.   clsLatticeNode Ln = new clsLatticeNode(X);
12.   count_cid++;
13.   ClosedSet.put(new Integer(count_cid), Ln);
14.   int lvl = Ln.getLevel();
15.   Item temp = new Item (new Integer(count_cid),lvl);
16.   LookUpLevel.add(temp);
17.
18.   Lc.addChild(Ln);
19.   Ln.addParent(Lc);
20.
21.   //=======getSupersetMinimum=========================
22.   Vector Smin = (Vector)S.clone();
23.   . . .
24.   //end of getSupersetMinimum=========================
25.
26.   for(int i=0 ; i<Smin.size() ; i++){
27.     clsLatticeNode Zmin = (clsLatticeNode)Smin.elementAt(i);
28.     Ln.addChild(Zmin);
29.     Zmin.addParent(Ln);
30.     Vector child_Zmin = Zmin.getParents();
31.     for(int j=0 ; j<child_Zmin.size() ; j++){
32.       clsLatticeNode p_Zmin = (clsLatticeNode)child_Zmin.elementAt(j);
33.       CharSequence p_Zmin_kode = getString(p_Zmin.getKode())
                    .subSequence(0,getString(p_Zmin.getKode()).length());
34.       if(p_Zmin.getLevel()<Ln.getLevel()&& getString(Zmin.getKode())
                                                  .contains(p_Zmin_kode)){
35.         p_Zmin.removeChild(Zmin);
36.         Zmin.removeParent(p_Zmin);
37.       }
38.     }
39.   }
40.   return Ln;
41. }
```

Gambar 4.20 Kode program *method* `SubsumptionCheckLattGen()`

4.3.4. Implementasi Antarmuka

Pada subbab ini akan dijelaskan mengenai implementasi dari desain perangkat lunak secara detail. Perangkat lunak ini dibangun dengan mengimplementasikan penggunaan komponen javax.swing pada bahasa pemrograman Java. Perangkat lunak ini membutuhkan interaksi pengguna berupa parameter masukan dan eksekusi tombol secara bertahap. Untuk mempermudah pengguna dalam menjalankan perangkat lunak dengan langkah-langkah yang benar, maka pada

setiap panel dialog disediakan tombol yang akan menuntun pengguna untuk melakukan proses selanjutnya. Namun agar perangkat lunak ini lebih interaktif, maka pengguna juga dapat kembali ke proses sesaat sebelumnya dengan mengeksekusi menu-menu yang telah disediakan, tanpa harus mengulang pemrosesan sejak awal.

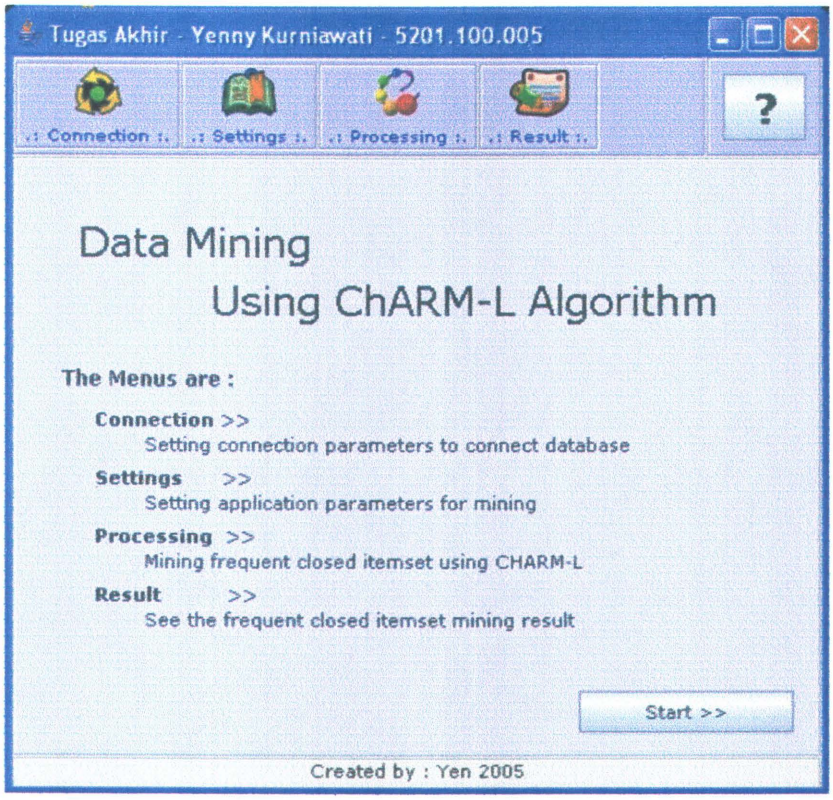
4.3.4.1. Antarmuka Frame Utama

Frame utama dalam perangkat lunak ini merupakan tampilan awal yang muncul begitu perangkat lunak ini dijalankan. Frame utama ini hanya berisi informasi mengenai apa fungsi dari perangkat lunak ini dan kegunaan dari navigation menu yang disediakan. Navigation menu yang disediakan ditampilkan pada gambar 4.21, dan penjelasan mengenai masing-masing fungsi dalam navigation menu ditampilkan pada Tabel 4.3. Gambar 4.22 menunjukkan tampilan dari frame utama.

Gambar 4.21 *Navigation Menu* dari perangkat lunak

Tabel 4.3 Penjelasan *Navigation Menu* dari perangkat lunak

Navigation Menu	Penjelasan
Connection	Menu untuk menampilkan dialog Connection
Settings	Menu untuk menampilkan dialog Settings
Processing	Menu untuk menampilkan dialog Processing
Result	Menu untuk menampilkan dialog Result
Help	Menu untuk menampilkan informasi perangkat lunak



Gambar 4.22 Antarmuka Frame Utama

4.3.4.2. Antarmuka Dialog Connection

Antarmuka dialog Connection ditunjukkan pada gambar 4.23. Fungsi dari dialog `Connection` ini adalah untuk menerima parameter masukan dari pengguna yang akan digunakan sehubungan dengan koneksi basis data pengguna. Komponen yang terdapat dalam dialog ini meliputi:

- 2 *label* yang menunjukkan tipe basis data yang digunakan, yaitu Oracle
- 1 *label* dan 1 textField untuk data masukan nama komputer basis data
- 1 *label* dan 1 textField untuk menentukan port yang digunakan pada komputer basis data
- 1 *label* dan 1 *textField* untuk data masukan nama basis data yang digunakan
- 1 *label* dan 1 *textField* untuk data masukan nama pengguna
- 1 *label* dan 1 *passwordField* untuk data masukan password pengguna

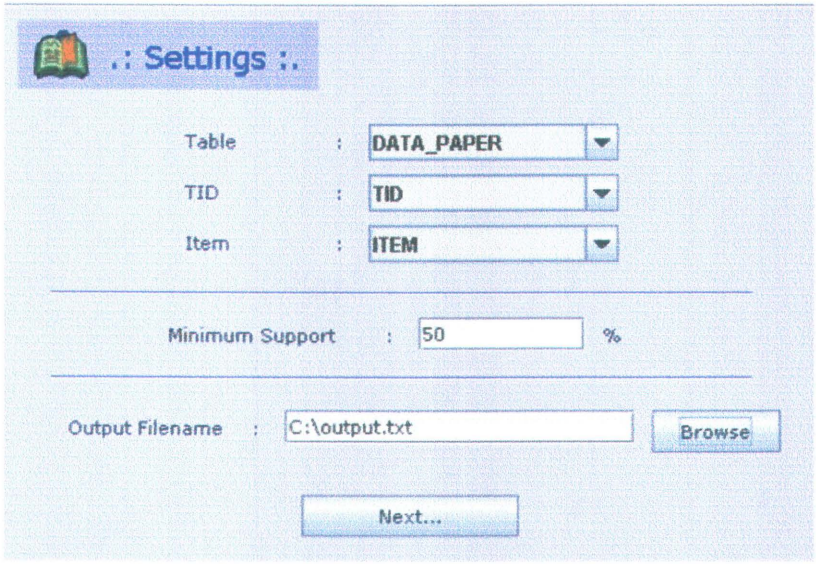
- 1 tombol "`Connecting...`" yang akan mengeksekusi perintah untuk melakukan koneksi ke basis data sesuai dengan parameter yang telah dimasukkan oleh pengguna.

4.3.4.3. Antarmuka Dialog Settings

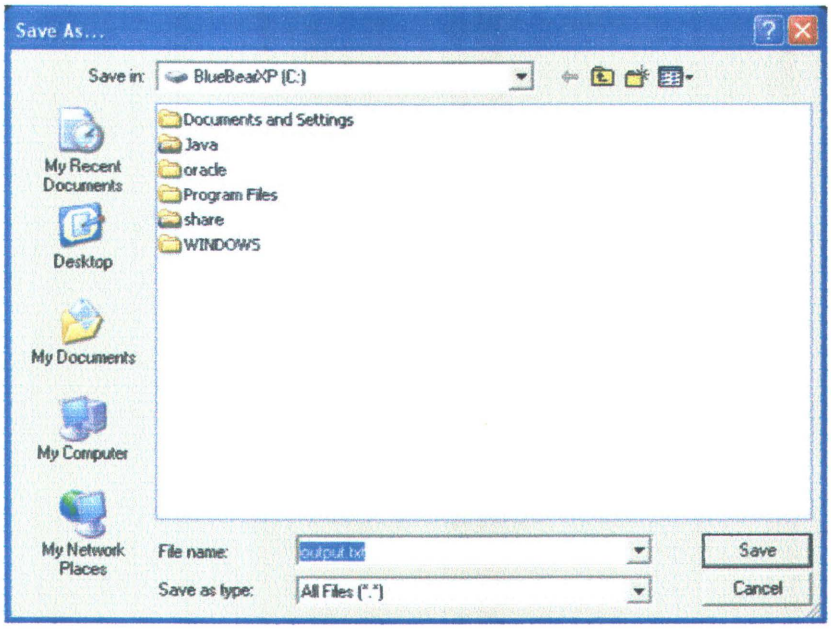
Fungsi dari dialog ini yaitu untuk menerima masukan pengguna yang berhubungan dengan parameter proses penggalian *frequent closed itemset*. Dalam dialog ini pengguna juga diminta untuk memasukkan nama dan lokasi untuk file teks keluaran. Berikut ini dijelaskan masing-masing komponen yang terdapat dalam dialog Settings.

- 1 *label* dan 1 *combobox* untuk memilih nama tabel dalam basis data yang ingin digunakan
- 1 *label* dan 1 *combobox* untuk memilih nama kolom yang akan digunakan sebagai kolom transaction id (*tid*)
- 1 *label* dan 1 *combobox* untuk memilih nama kolom yang akan digunakan sebagai kolom item
- 1 *label* dan 1 *textfield* yang digunakan untuk masukkan parameter minimum support yang diinginkan
- 1 *label* dan 1 *textfield* yang digunakan untuk masukan parameter nama dan lokasi file teks hasil keluaran.
- 1 tombol "`Browse`" yang akan membuka sebuah FileDialog untuk mempermudah pengguna dalam menentukan lokasi file keluaran yang diinginkan
- 1 tombol "`Next...`" yang akan memberikan navigasi ke langkah selanjutnya.

Antarmuka dialog Settings ditunjukkan pada gambar 4.23, sedangkan FileDialog untuk navigasi nama dan lokasi file output ditampilkan pada gambar 4.24.



Gambar 4.23 Antarmuka Dialog Settings



Gambar 4.24 FileDialog untuk memilih nama dan lokasi file output

4.3.4.4. Antarmuka Dialog Processing

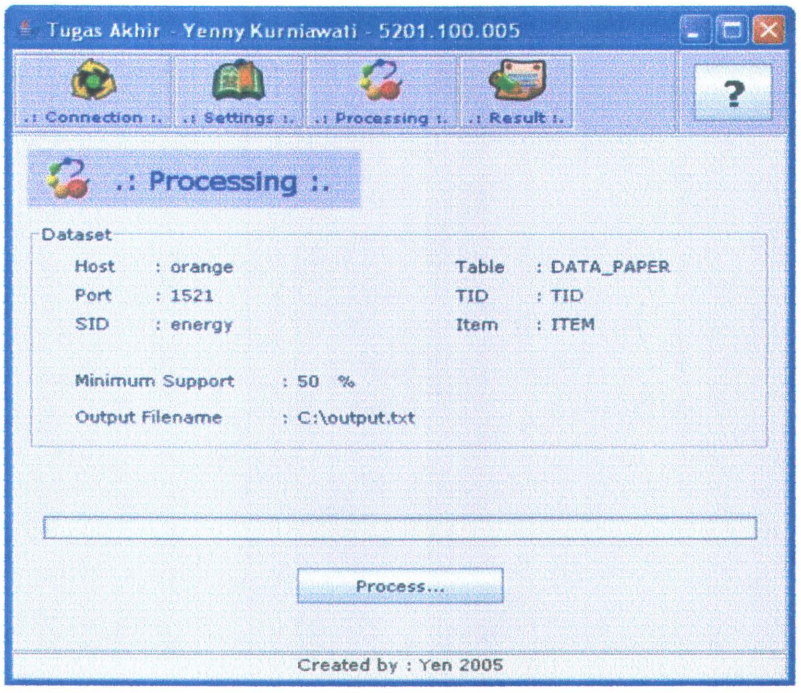
Fungsi dari dialog Processing ini adalah untuk menginformasikan pengguna kembali mengenai berbagai parameter yang telah dimasukkan ke dalam aplikasi.

Dari dialog inilah pengguna dapat mengeksekusi tombol "Process..." untuk mulai menjalankan proses penggalian *frequent closed itemset*. Jika proses penggalian telah selesai dilakukan, maka tombol "Process..." tersebut akan digantikan dengan tombol "See Result", yang akan mengarahkan pengguna untuk menuju ke tahap berikutnya yaitu melihat hasil keluaran.

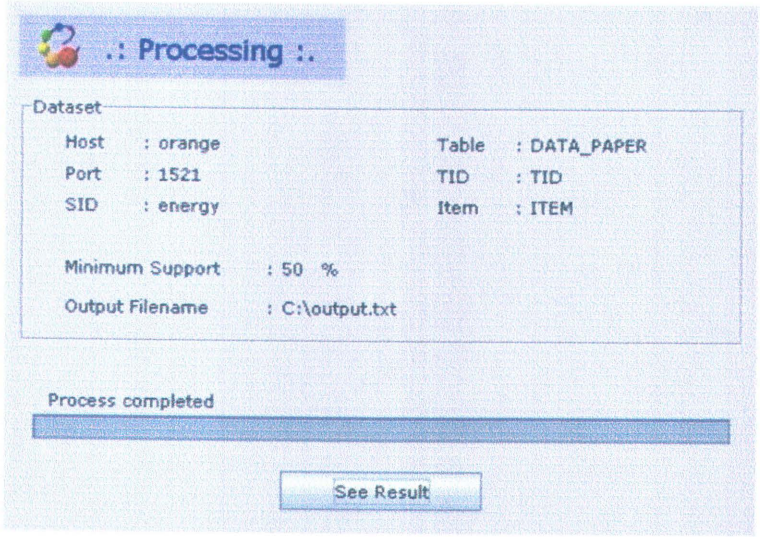
Dialog ini terdiri dari 2 panel, yaitu :

- Panel dataset yang berisi berbagai parameter yang telah dimasukkan oleh pengguna
- Panel proses, yang berisi sebuah progressbar, sebuah tombol "Process..." dan sebuah tombol "See Result". Progressbar ini ditampilkan agar pengguna dapat melihat sejauh mana proses penggalian telah berjalan.

Gambar 4.25 menunjukkan antarmuka Dialog Processing sebelum proses penggalian dijalankan, sedangkan gambar 4.26 menunjukkan antarmuka setelah proses penggalian selesai.



Gambar 4.25 Antarmuka Dialog Processing – sebelum penggalian



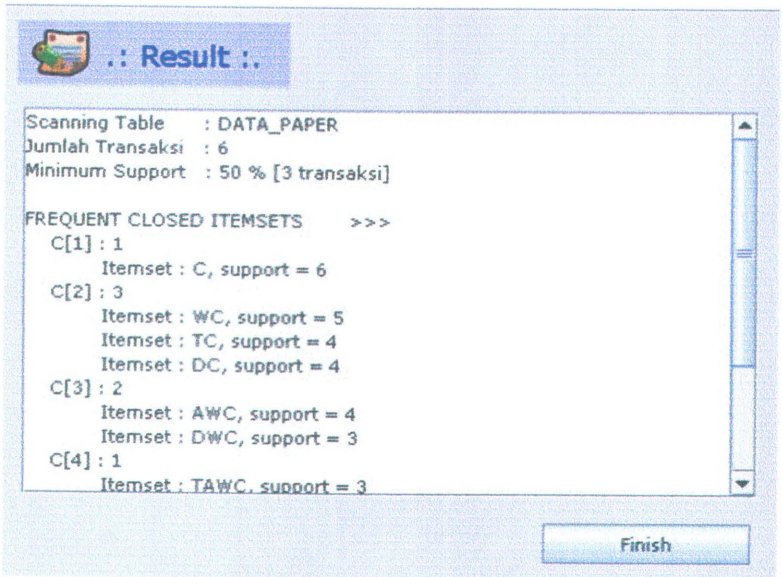
Gambar 4.26 Antarmuka Dialog Processing – setelah penggalian

4.3.4.5. Antarmuka Dialog Result

Dialog ini berisi tentang informasi hasil keluaran dari proses penggalian *frequent closed itemset* mining, ditampilkan dalam komponen JTextArea. Dialog ini terdiri dari 2 komponen yaitu:

- 1 *textArea* yang berfungsi menampilkan hasil proses penggalian
- 1 tombol yang digunakan untuk menutup aplikasi

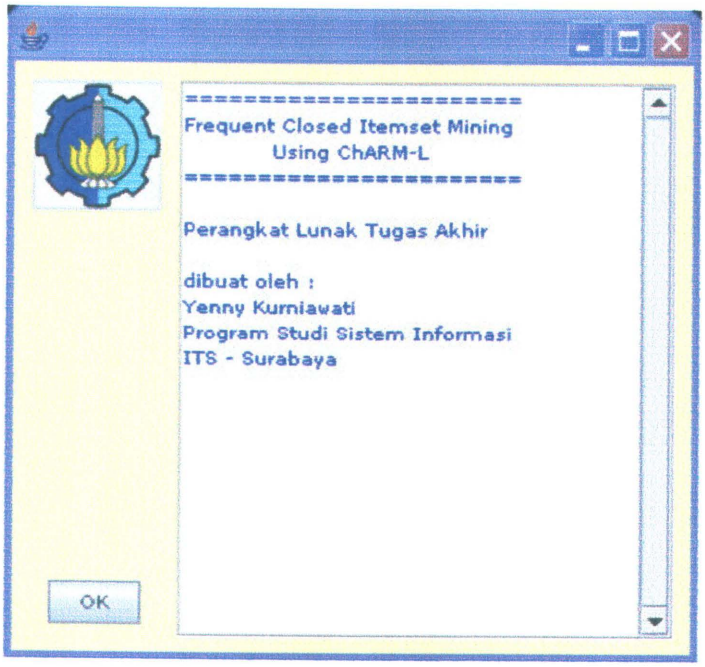
Antarmuka untuk dialog Result ditampilkan pada gambar 4.27.



Gambar 4.27 Antarmuka Dialog Result

4.3.4.6. Antarmuka Dialog Help

Dialog ini berisi informasi mengenai pembuatan perangkat lunak, antarmuka dari dialog ini ditampilkan pada gambar 4.28.



Gambar 4.28 Antarmuka Dialog Help

BAB V

UJI COBA DAN EVALUASI

BAB V

UJI COBA DAN EVALUASI

5.1. LINGKUNGAN UJI COBA

Perangkat keras yang digunakan dalam uji coba uni adalah komputer dengan prosesor AMD Athlon XP 2100+, harddisk 80GB ATA 7200 rpm, dan DDR memori sebesar 512MB. Sistem operasi yang digunakan adalah Windows XP Service Pack 1. Basis data yang digunakan adalah Oracle DBMS versi 8.1.7.

5.2. PENGUJIAN DATA KECIL

Pengujian data kecil ini dilakukan terhadap perangkat lunak dengan masukan berupa tabel basis data yang berisi transaksi dalam jumlah kecil. Dalam pengujian ini, data transaksi yang digunakan sebagai masukan yaitu data transaksi seperti yang telah dituliskan dalam tabel 3.1. Data transaksi tersebut harus disesuaikan dulu formatnya sehingga menjadi seperti yang telah dituliskan pada tabel 4.1. Data transaksi ini akan menghasilkan 7 buah *frequent closed itemset* seperti yang telah ditunjukkan pada gambar 2.1

Setelah perangkat lunak CHARM-L dijalankan dengan minimum support 50%, maka akan mendapatkan hasil keluaran berupa frequent closed itemset sebagai berikut :

```
Parameter          >>>
Scanning Table     : DATA_PAPER
Jumlah Transaksi   : 6
Jumlah Item        : 5
Jumlah Record      : 23
Minimum Support    : 50 % [3 transaksi]

FREQUENT CLOSED ITEMSETS        >>>
  FCI[1] : 1
       Itemset : C, support = 6
  FCI[2] : 3
       Itemset : WC, support = 5
       Itemset : TC, support = 4
       Itemset : DC, support = 4
  FCI[3] : 2
       Itemset : AWC, support = 4
```

```
     Itemset : DWC, support = 3
FCI[4] : 1
     Itemset : TAWC, support = 3
```

Hasil tersebut menunjukkan kesesuaian dengan *frequent closed itemset* yang ditunjukkan pada gambar 2.1.

5.3. DATA BESAR YANG DIGUNAKAN

Berikut akan dijelaskan mengenai masing-masing karakteristik data masukan yang digunakan dalam pengujian data besar. Selain itu juga dijelaskan cara penggunaan perangkat lunak data

5.3.1. Karakteristik Data Masukan

Uji coba pada perangka lunak ini dilakukan terhadap beberapa data sintetis yang berbeda. Sebagian dataset yang digunakan didapatkan dari *FIMI* (*Frequent Itemset Mining Implementation*) *Repository* [6], sedangkan dataset lainnya didapatkan dari pembangkitan data dengan menggunakan data generator yang disediakan pada *IlliMine Project* [7]. Pada tabel 5.1 berikut ini diperlihatkan karakteristik dari setiap data.

Tabel 5.1 Karakteristik data masukan

Data	Record	Item	Transaksi	Rata-rata item/transaksi
chess	118252	76	3196	37
mushroom	186852	120	8124	23
connect	2904951	130	67557	43
pumsb*	2475947	7117	49046	50
T40I10D100K	3960507	1000	100000	40
Trans100K	996279	1000	100000	10
Trans200K	1991009	1000	200000	10
Trans300K	2986057	1000	300000	10
Trans400K	3982351	1000	400000	10
Trans500K	4979533	1000	500000	10

5.3.2. Pembangkitan Data Sintetis dengan Data Generator

Perangkat lunak pembangkit data yang digunakan untuk membangkitkan data sintetis dalam pengujian ini yaitu perangkat lunak pembangkit data untuk pola asosiasi dan sekuensial yang disediakan oleh *IlliMine Project* [7]. Perangkat lunak ini dapat menghasilkan data *large frequent itemset* dengan taksonomi maupun tanpa taksonomi. Perangkat lunak ini terletak dalam folder `illimine/packages/association/data_generator/`. Dalam tugas akhir ini digunakan data masukan tanpa taksonomi, sehingga penggunaan perangkat lunak tersebut menjadi sebagai berikut:

```
bin/assoc_data_generator lit [pilihan]
```

Berikut merupakan pilihan parameter yang digunakan untuk melakukan pembangkitan data tanpa taksonomi.

- `-ntrans` banyaknya_transaksi_dalam_ribuan (default : 1000)
- `-tlen` rata-rata item dalam transaksi (default : 10)
- `-nitems` banyaknya_item_yang_berbeda_dalam_ribuan (default : 100)
- `-fname <namafile>` (hasil keluaran berupa namafile.data dan namafile.pat)
- `-ascii` (default : `false`)

Perangkat lunak ini memungkinkan 2 macam format file data keluaran, format biner dan format ascii, berdasarkan pada pilihan parameter `-ascii` yang dispesifikasikan pengguna. Dalam tugas akhir ini digunakan format ascii untuk data keluaran. Pada file data keluaran dengan format ascii, nilai pertama yang keluar adalah nilai banyaknya item dalam transaksi tersebut, diikuti dengan tiap item dalam transaksi yang dituliskan secara terurut. Setelah itu transaksi selanjutnya dituliskan dengan cara yang sama. Dari penjelasan di atas maka pemanggilan yang dilakukan untuk membangkitkan data transaksi `trans100k` menjadi sepert berikut:

```
bin/assoc_data_generator lit -ntrans 100 -tlen 10 -nitems 1 -ascii
                         -fname trans100k
```

Berikut merupakan 5 transaksi awal dari data trans100k hasil pembangkitan:

```
6 77 595 639 858 875 889
8 149 239 383 529 643 821 832 835
9 7 25 48 306 477 484 661 679 864
7 255 486 684 688 700 706 983
5 480 645 744 924 965
```

5.4. PENGUJIAN DATA BESAR

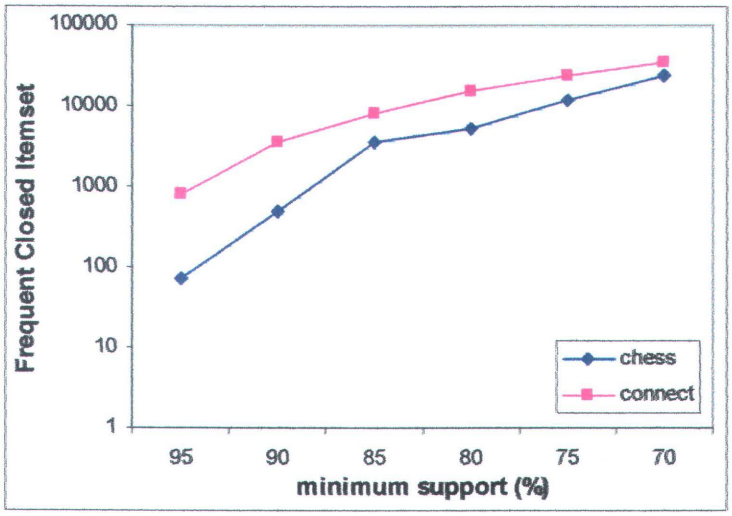
Pengujian data besar ini dilakukan untuk menguji kebenaran dan kehandalan perangkat lunak.. Berikut ini dijelaskan masing-masing pengujian yang telah dilakukan terhadap perangkat lunak dengan menggunakan data-data di atas.

5.4.1. Dataset chess

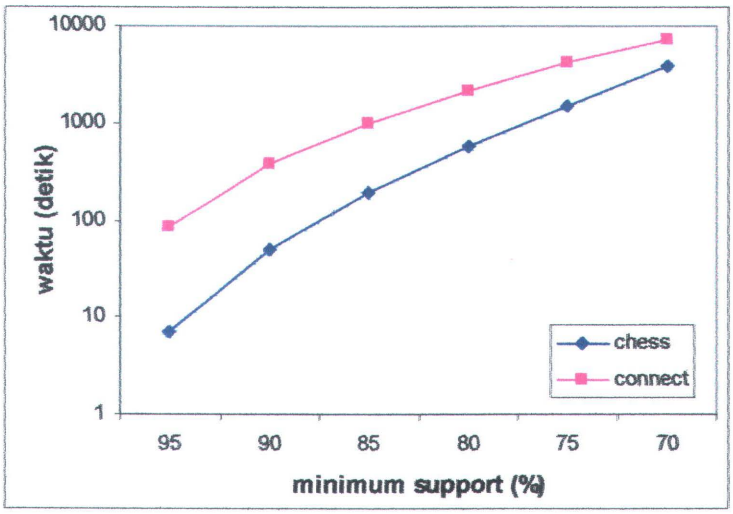
Dataset ini merupakan salah satu data yang didapatkan dari *FIMI Repository*. Data ini terdiri dari 3196 transaksi dengan rata-rata panjang transaksi adalah 32 item. Pada uji coba ini digunakan data chess sebagai data masukan dengan berbagai nilai minimum support dalam range 70% - 95%. Banyaknya *frequent closed itemset* yang dihasilkan dari proses penggalian ditunjukkan pada gambar 5.1. Waktu proses penggalian *frequent closed itemset* ditunjukan pada gambar 5.2.

5.4.2. Dataset connect

Dataset ini merupakan salah satu data yang didapatkan dari *FIMI Repository*, namun dataset ini juga tersedia sebagai dataset contoh dalam *IlliMine Project*. Data ini terdiri dari 67557 transaksi dengan rata-rata panjang transaksi adalah 43 item. Pada uji coba ini digunakan data connect sebagai data masukan dengan berbagai nilai minimum support dalam range 70% - 95%. Banyaknya *frequent closed itemset* yang dihasilkan dari proses penggalian ditunjukkan pada gambar 5.1. Waktu proses penggalian *frequent closed itemset* ditunjukan pada gambar 5.2.



Gambar 5.1 Perbandingan support - jumlah FCI pada dataset chess dan connect



Gambar 5.2 Perbandingan support - waktu komputasi pada chess dan connect

Dari kedua grafiks di atas dapat dilihat bahwa semakin kecil nilai minimum support yang diberikan, maka akan didapatkan lebih banyak FCI, dan waktu komputasi akan semakin lama. Untuk uji coba pada chess dengan nilai minimum support 70% dan 75%, terjadi lonjakan waktu komputasi. Hal ini disebabkan jumlah *frequent itemset* yang diproses semakin besar, sehingga pemanggilan CHARM-L-EXTEND secara rekursif juga semakin banyak. Dari kedua grafiks di

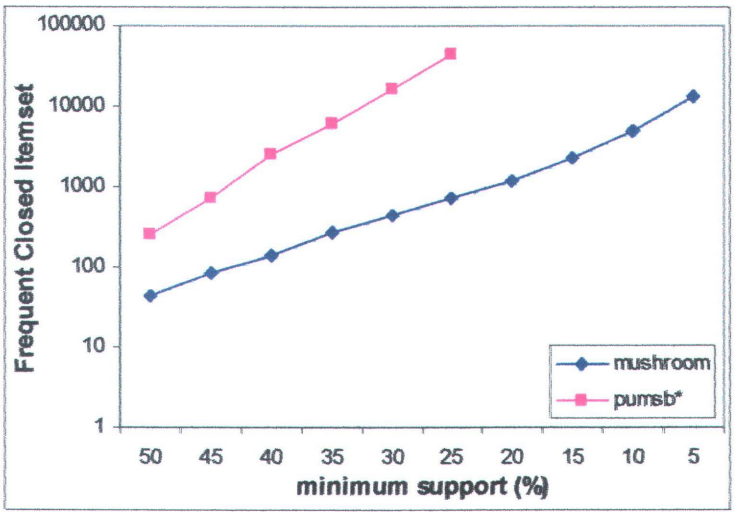
atas tampak bahwa pernambahan jumlah FCI berbanding lurus dengan waktu yang dibutuhkan untuk melakukan komputasi.

5.4.3. Dataset Mushroom

Dataset ini merupakan salah satu data yang didapatkan dari FIMI *Repository*. Data ini terdiri dari 8124 transaksi dengan rata-rata panjang transaksi adalah 23 item. Pada uji coba ini digunakan data mushroom sebagai data masukan dengan berbagai nilai minimum support dalam range 5% - 50%. Banyaknya *frequent closed itemset* yang dihasilkan dari proses penggalian ditunjukkan pada gambar 5.3. Waktu proses penggalian *frequent closed itemset* ditunjukan pada gambar 5.4.

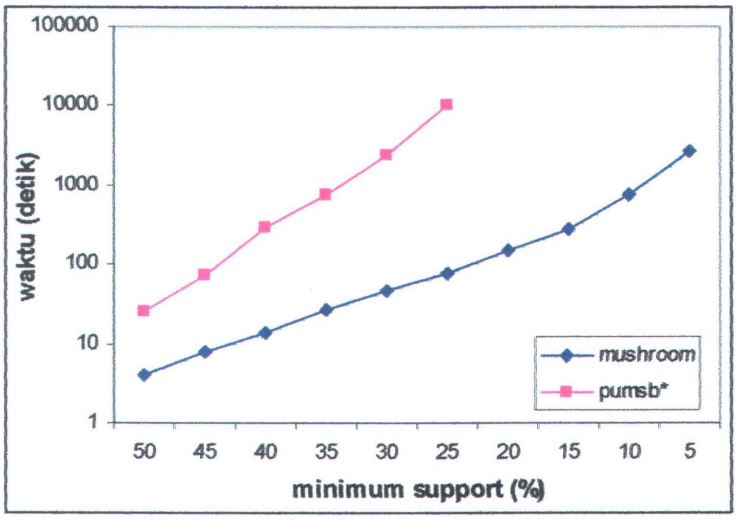
5.4.4. Dataset pumsb*

Dataset ini merupakan salah satu data yang didapatkan dari *FIMI Repository*. Data ini terdiri dari 49046 transaksi dengan rata-rata panjang transaksi adalah 50 item. Pada uji coba ini digunakan data pumsb* sebagai data masukan dengan berbagai nilai minimum support dalam range 25% - 50%. Banyaknya *frequent closed itemset* yang dihasilkan dari proses penggalian ditunjukkan pada gambar 5.3 Waktu proses penggalian *frequent closed itemset* ditunjukan pada gambar 5.4.



MILIK PERPUSTAKAAN ITS

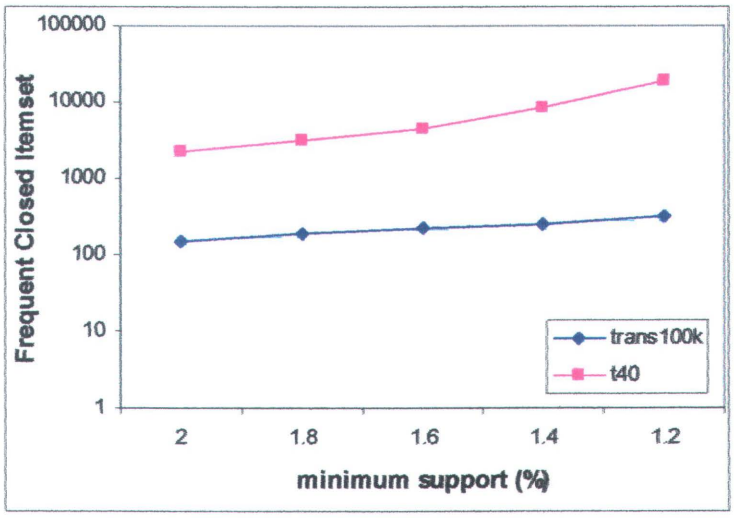
Gambar 5.3 Perbandingan support - jumlah FCI pada mushroom dan pumsb*



Gambar 5.4 Perbandingan support - waktu komputasi pada mushroom dan pumsb*

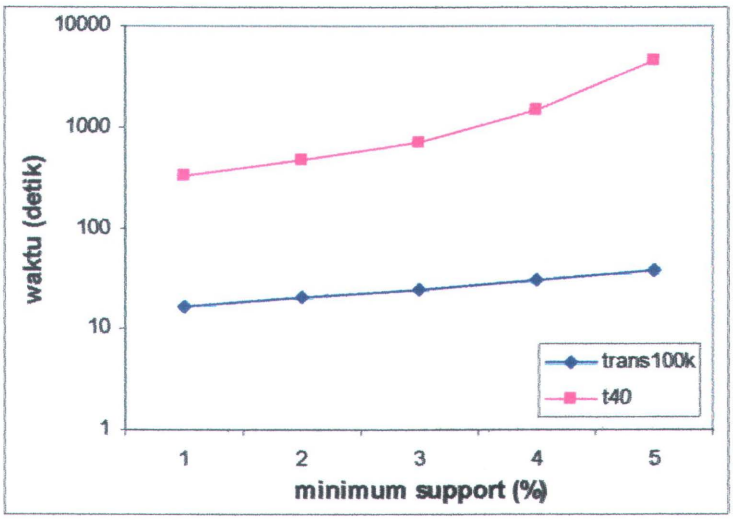
Dari kedua grafiks tersebut tampak bahwa pernambahan jumlah FCI berbanding lurus dengan waktu yang dibutuhkan untuk melakukan komputasi.

5.4.5. Dataset T40I10D100K



Gambar 5.5 Perbandingan support - jumlah FCI pada T40 dan trans100k

Dataset ini merupakan salah satu data yang didapatkan dari *FIMI Repository*. Data ini merupakan data transasksi sintetik yang terdiri dari 100000 transaksi dengan rata-rata panjang transaksi adalah 40 item. Pada uji coba ini digunakan data T40 ini sebagai data masukan dengan berbagai nilai minimum support dalam range 1.2% - 2%. Banyaknya *frequent closed itemset* yang dihasilkan dari proses penggalian ditunjukkan pada gambar 5.5. Waktu proses penggalian *frequent closed itemset* ditunjukan pada gambar 5.6.



Gambar 5.6 Perbandingan support - waktu komputasi pada T40 dan trans100k

Dari kedua grafiks tersebut menunjukkan bahwa T40, yang rata-rata item/transaksinya lebih panjang daripada trans100k, menghasilkan FCI lebih banyak daripada trans100k pada nilai minimum support yang sama.

5.4.6. Dataset Trans100K

Dataset ini merupakan salah satu data yang dihasilkan dari pembangkitan data menggunakan perangkat lunak *data generator* yang disediakan dalam *IlliMine Project*. Data ini terdiri dari 100000 transaksi dengan rata-rata panjang transaksi adalah 10 item. Pada uji coba ini digunakan data Trans100K ini sebagai data masukan dengan berbagai nilai minimum support dalam range 0.07% - 2%. Banyaknya *frequent closed itemset* yang dihasilkan dari proses penggalian

ditunjukkan pada gambar 5.5 dan gambar 5.7. Waktu proses penggalian *frequent closed itemset* ditunjukan pada gambar 5.6 dan gambar 5.8.

5.4.7. Dataset Trans200K

Dataset ini merupakan data yang didapatkan dari pembangkitan data dengan perangkat lunak *data generator* dari *IlliMine Project*. Data ini terdiri dari 200000 transaksi dengan rata-rata panjang transaksi adalah 10 item. Pada uji coba ini digunakan data Trans200K ini sebagai data masukan dengan berbagai nilai minimum support dalam range 0.075% - 0.5%. Banyaknya *frequent closed itemset* yang dihasilkan dari proses penggalian ditunjukkan pada gambar 5.7. Waktu proses penggalian *frequent closed itemset* ditunjukan pada gambar 5.8.

5.4.8. Dataset Trans300K

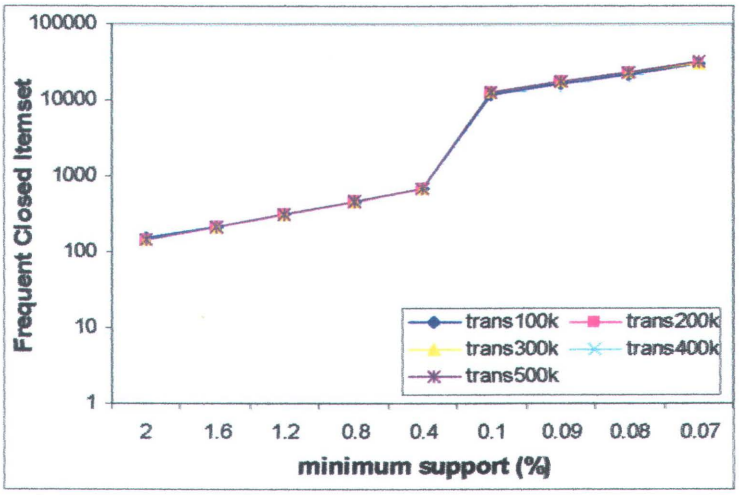
Dataset ini merupakan salah satu data hasil pembangkitan dangan perangkat lunak *data generator* dari *IlliMine Project*. Data ini terdiri dari 300000 transaksi dengan rata-rata panjang transaksi adalah 10 item. Pada uji coba ini digunakan data Trans300K ini sebagai data masukan dengan berbagai nilai minimum support dalam range 0.067% - 0.67%. Banyaknya *frequent closed itemset* yang dihasilkan dari proses penggalian ditunjukkan pada gambar 5.7. Waktu proses penggalian *frequent closed itemset* ditunjukan pada gambar 5.8.

5.4.9. Dataset Trans400K

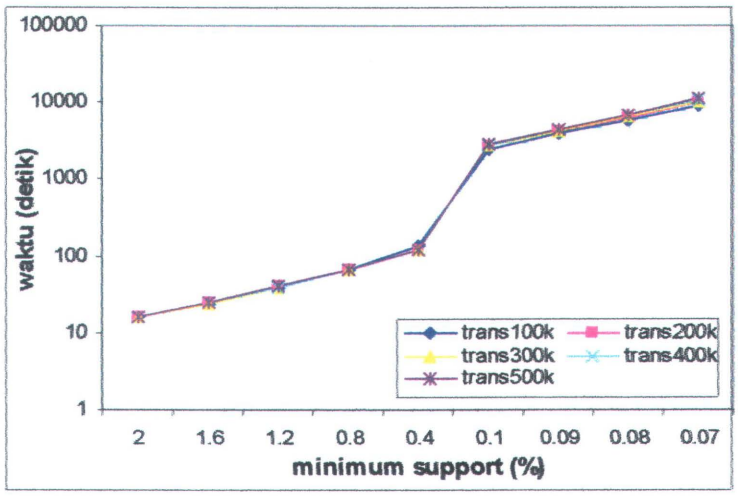
Dataset ini merupakan salah satu data hasil pembangkitan dangan perangkat lunak *data generator* dari *IlliMine Project*. Data ini terdiri dari 400000 transaksi dengan rata-rata panjang transaksi adalah 10 item. Pada uji coba ini digunakan data Trans400K ini sebagai data masukan dengan berbagai nilai minimum support dalam range 0.07% - 0.5%. Banyaknya *frequent closed itemset* yang dihasilkan dari proses penggalian ditunjukkan pada gambar 5.7. Waktu proses penggalian *frequent closed itemset* ditunjukan pada gambar 5.8.

5.4.10. Dataset Trans500K

Dataset ini merupakan salah satu data hasil pembangkitan dangan perangkat lunak *data generator* dari *IlliMine Project*. Data ini terdiri dari 500000 transaksi dengan rata-rata panjang transaksi adalah 10 item. Pada uji coba ini digunakan data Trans500K ini sebagai data masukan dengan berbagai nilai minimum support dalam range 0.07% - 0.4%. Banyaknya *frequent closed itemset* yang dihasilkan dari proses penggalian ditunjukkan pada gambar 5.7. Waktu proses penggalian *frequent closed itemset* ditunjukan pada gambar 5.8.



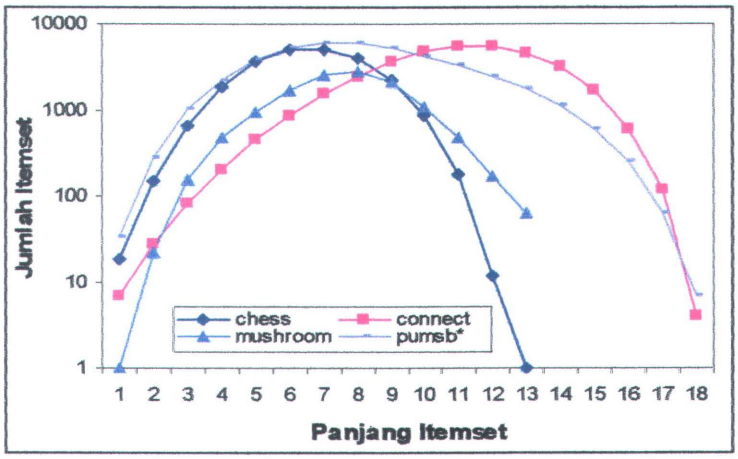
Gambar 5.7 Perbandingan support - jumlah FCI pada Trans100k - Trans500K



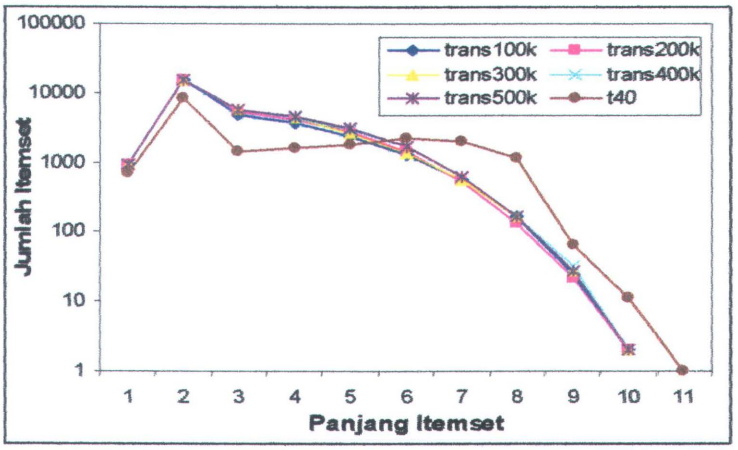
Gambar 5.8 Perbandingan support - waktu komputasi pada Trans100k - Trans500K

Dari kedua grafiks pada gambar 5.7 dan gambar 5.8, tampak bahwa pernambahan jumlah FCI juga diikuti dengan penambahan waktu yang dibutuhkan untuk melakukan komputasi, dan pertumbuhan jumlah transaksi tidak terlalu mempengaruhi jumlah FCI yang dihasilkan dan waktu komputasi yang diperlukan. Dari semua percobaan di atas dapat disimpulkan bahwa penambahan jumlah FCI memang berbanding lurus dengan penambahan waktu yang dibutuhkan untuk melakukan penggalian.

Dari hasil uji coba yang dilakukan di atas dengan berbagai dataset masukan, didapatkan sejumlah *frequent closed itemset* yang bervariasi dalam hal jumlah dan panjangnya. Hal ini bergantung pada karakteristik dataset itu sendiri dan juga pada batasan minimum support yang ditentukan. Adapun banyaknya *frequent closed itemset* yang dihasilkan dari masing-masing dataset berdasarkan nilai minimum support tertentu dapat dilihat pada gambar 5.9 untuk dataset real dan pada gambar 5.10 untuk dataset sintetik.

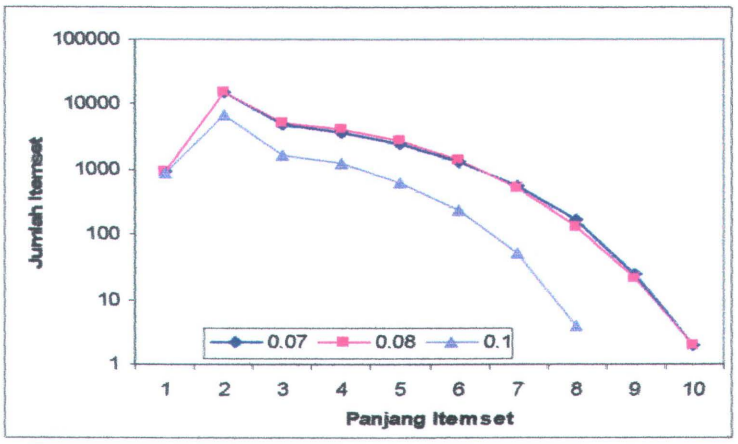


Gambar 5.9 Distribusi FCI berdasarkan panjang itemset pada dataset real



Gambar 5.10 Distribusi FCI berdasarkan panjang itemset pada dataset sintetik

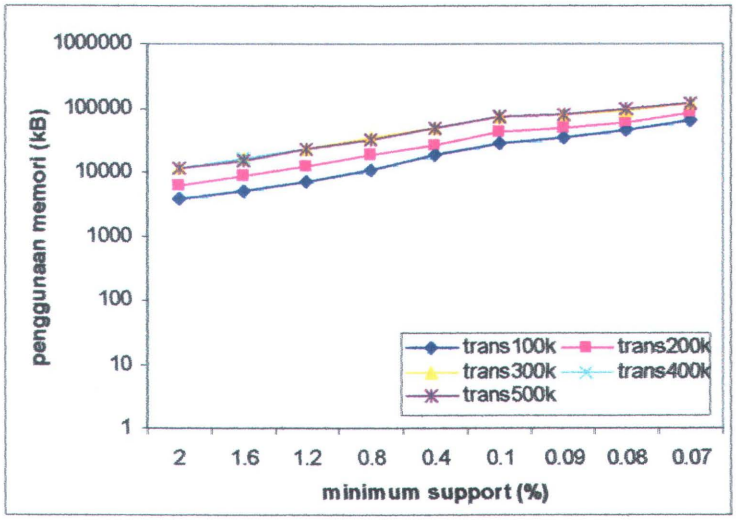
Dari gambar 5.11 tampak bahwa dengan semakin kecil minimum support, maka grafiks distribusi FCI berdasarkan panjang itemsetnya lebih mendekati distribusi dataset tersebut sebenarnya. Dari gambar 5.9 dan gambar 5.10 tampak bahwa dataset chess, dataset connect dan dataset pumsb* memiliki kecenderungan distribusi yang sama, yaitu distribusinya simetris, dataset mushroom dan dataset T40I10D100K memiliki kecenderungan distribusi bi-modal, dan dataset Trans100K sampai dataset Trans500K memiliki distribusi simpang kanan (*right skewness distribution*). Perlu diingat bahwa dataset Tran100K sampai Trans500K mempunyai karakteristik yang sama kecuali pada jumlah transaksi. Dari sini dapat disimpulkan bahwa distribusi dataset tidak terpengaruh oleh banyaknya transaksi.



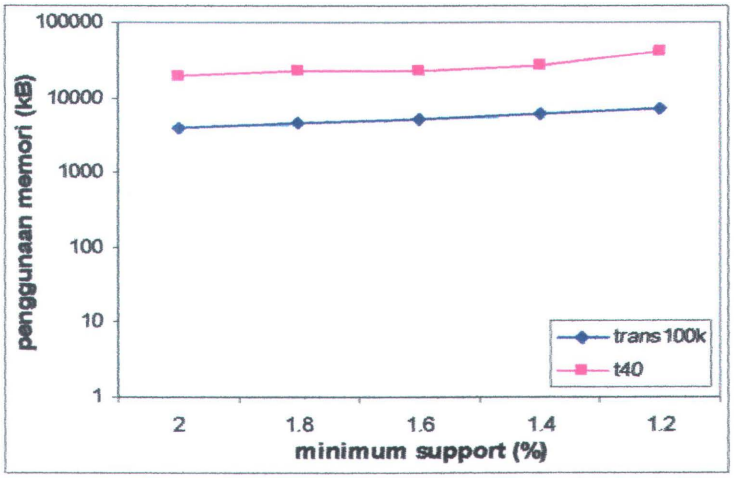
Gambar 5.11 Distribusi FCI berdasarkan panjang itemset untuk dataset trans100k dengan berbagai minimum support

5.5. PENGUJIAN UTILISASI MEMORI

Pengujian ini dilakukan untuk melihat utilisasi memori dari perangkat lunak yang telah dibuat. Penggunaan memori terbesar terjadi pada saat pemindaian basis data. Penggunaan memori algoritma CHARM-L ini relatif lebih kecil daripada algoritma lainnya karena algoritma CHARM-L hanya melakukan sekali pemindaian ke basis data, selain itu dalam pemrosesannya algoritma CHARM-L ini menggunakan format diffset yang relatif jauh lebih kecil daripada penggunaan format tidset. Gambar 5.12 dan 5.13 menunjukkan uji coba utilisasi memori untuk data T40I10D100K, dan data Trans100K sampai Trans500K.



Gambar 5.12 Penggunaan memori pada dataset Trans100K-Trans500k



Gambar 5.13 Penggunaan memori pada dataset T40 dan Trans100k

Dari uji coba penggunaan memori tersebut, semuanya memiliki kecenderungan yang sama yaitu penggunaan memori yang meningkat seiring dengan berkurangnya minimum support. Hal ini diakibatkan oleh semakin banyaknya frequent itemset yang diproses dan semakin banyaknya *frequent closed itemset* yang dihasilkan sehingga membutuhkan memori lebih besar. Gambar 5.12 menunjukkan penggunaan memori untuk pertumbuhan data yaitu jumlah transaksi. Dari kelima grafiks tersebut tampak bahwa penggunaan memori tidak terlalu bertambah seiring dengan pertumbuhan jumlah transaksi.

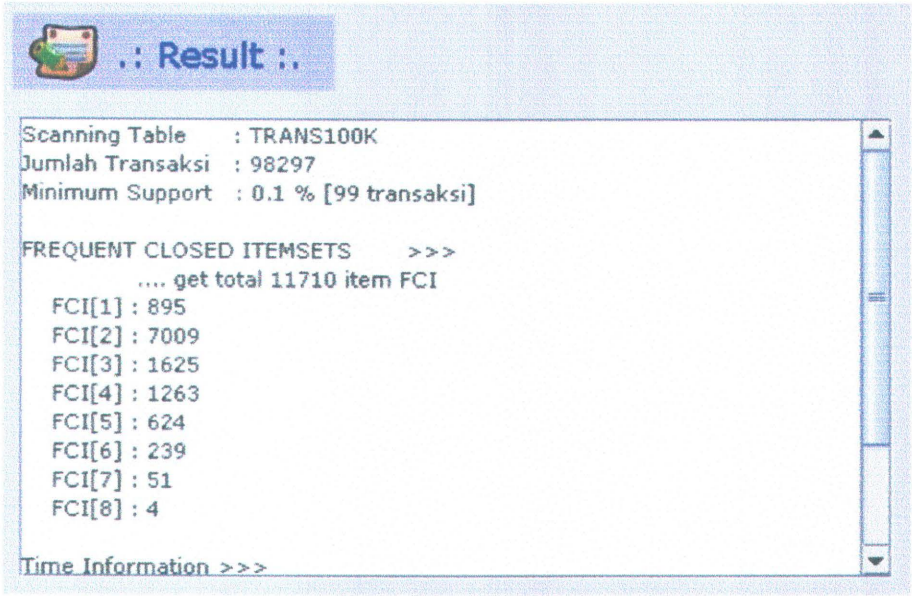
Gambar 5.13 menunjukkan penggunaan memori untuk pertumbuhan data yaitu panjang item per transaksi. Kedua grafiks tersebut menunjukkan kisaran penggunaan memori yang cukup berbeda. Dari grafik tersebut dapat diambil kesimpulan bahwa pertumbuhan rata-rata item per transaksi berpengaruh pada penggunaan memori.

5.6. UJI COBA KEBENARAN

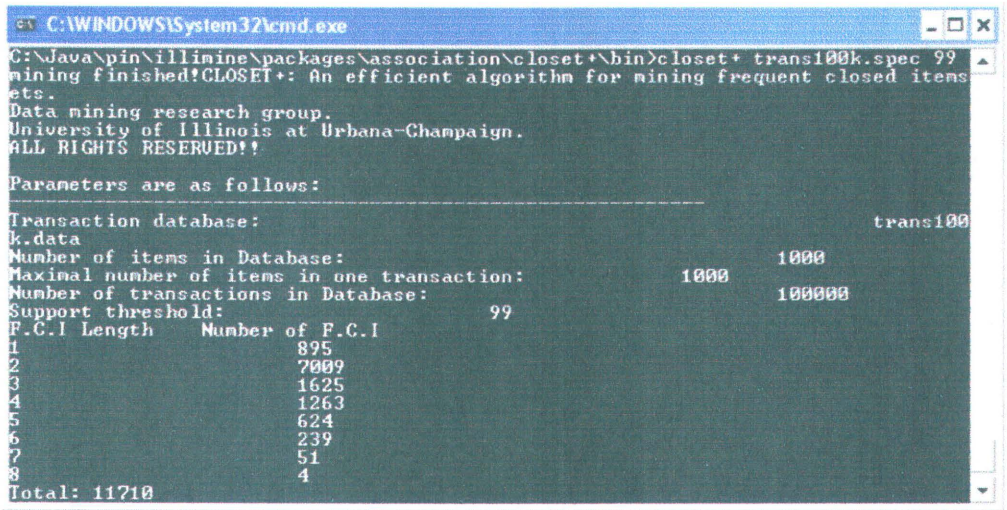
Untuk menguji kebenaran hasil keluaran perangkat lunak ini, dilakukan uji coba kebenaran dengan melakukan penggalian terhadap data dengan berbagai minimum support yang berbeda. Selanjutnya hasil penggalian tersebut dibandingkan dengan perangkat lunak pembanding, dalam hal ini yang dibandingkan yaitu jumlah keseluruhan FCI hasil penggalian dan jumlah FCI untuk tiap levelnya. Jika FCI yang didapatkan dari perangkat lunak CHARM-L sudah sesuai dengan FCI yang didapatkan dari perangkat lunak pembanding, maka proses yang dilakukan sudah benar. Sebagai perangkat lunak pembanding digunakan perangkat lunak data mining Closet+ yang disediakan dalam *Illimine Project*. Proses pengujian ini akan menggunakan dataset Trans100K.

Gambar 5.37 menunjukkan penggalian FCI dengan perangkat lunak CHARM-L yang dibangun, dilakukan pada dataset Trans100K dengan minimum support 0.1% atau sama dengan 99 transaksi. Pada gambar 5.38 ditunjukkan penggalian untuk data yang sama dengan nilai support yang sama. Kedua gambar tersebut

menunjukkan kesesuaian hasil, dengan jumlah seluruh FCI yang dihasilkan adalah 11710 item.



Gambar 5.14 Penggalian dengan perangkat lunak CHARM-L



Gambar 5.15 Penggalian dengan perangkat lunak closet+ dari IlliMine

Dari tabel 5.2 tampak bahwa hasil penggalian dengan perangkat lunak CHARM-L yang dibuat telah sesuai dengan hasil penggalian dengan perangkat lunak Closet+, sehingga dapat disimpulkan bahwa proses yang dilakukan sudah benar.

Tabel 5.2 Penggalian untuk dataset Trans100K untuk berbagai minimum support

Minsup (%)	Mintrans	CHARM-L			Closet+		
		Total FCI	k	FCI[k]	Total FCI	k	FCI[k]
1.2	1180	316	1	316	316	1	316
0.8	787	465	1	465	465	1	465
0.4	492	685	1	675	685	1	675
			2	10		2	10
0.1	99	11710	1	895	11710	1	895
			2	7009		2	7009
			3	1625		3	1625
			4	1263		4	1263
			5	624		5	624
			6	239		6	239
			7	51		7	51
			8	4		8	4
0.08	79	21756	1	910	21756	1	910
			2	11720		2	11720
			3	3423		3	3423
			4	2622		4	2622
			5	1716		5	1716
			6	897		6	897
			7	371		7	371
			8	92		8	92
			9	5		9	5
0.07	69	29145	1	921	29145	1	921
			2	15224		2	15224
			3	4937		3	4937
			4	3570		4	3570
			5	2420		5	2420
			6	1314		6	1314
			7	564		7	564
			8	168		8	168
			9	25		9	25
			10	2		10	2

BAB VI

PENUTUP

BAB VI

PENUTUP

Bab ini berisi kesimpulan dari tugas akhir yang dibuat berdasarkan hasil uji coba. Selain itu juga disertakan kemungkinan pengembangan yang dapat dilakukan pada tugas akhir ini.

6.1. KESIMPULAN

Dari hasil uji coba perangkat lunak yang telah dilakukan, didapatkan beberapa kesimpulan sebagai berikut:

- Algoritma CHARM-L berhasil diimplementasikan untuk menggali *frequent closed itemset*.
- Dari hasil uji coba dapat disimpulkan bahwa parameter minimum support mempengaruhi kinerja perangkat lunak. Waktu komputasi, penggunaan memori, dan jumlah *frequent closed itemset* yang dihasilkan semakin besar untuk nilai support minimal yang semakin kecil. Sebagai contoh untuk uji coba dengan data sebanyak 100000 transaksi dan jumlah item sebanyak 1000 buah, didapatkan bahwa untuk minimum support berturut-turut sebesar 1%, 0.75%, 0.5%, 0.25%, dibutuhkan waktu komputasi berturut-turut sebanyak 56, 85, 131, dan 942 detik.
- Penggunaan memori akan semakin besar seiring dengan semakin besarnya data transaksi. Sebagai contoh, untuk data transaksi sebesar 100000, untuk nilai support 0.1% penggunaan memori yang terjadi sebanyak 22605 kB danuntuk data transaksi sebesar 500000 penggunaan memori yang terjadi sebesar 62985 kB.

6.2. KEMUNGKINAN PENGEMBANGAN

Pada tugas akhir ini dilakukan pembuatan perangkat lunak penggalian *frequent closed itemset* dengan algoritma CHARM-L. Algoritma CHARM-L ini tidak membutuhkan memori yang besar dalam proses penggaliannya, lamanya proses

disebabkan oleh banyaknya komputasi yang harus dilakukan. Pengembangan yang mungkin dilakukan adalah dengan meningkatkan kegunaan dari perangkat lunak ini, yaitu dengan membangun perangkat lunak lain untuk menggali kaidah asosiasi dari *frequent closed itemset* yang dihasilkan oleh algoritma CHARM-L ini.

DAFTAR PUSTAKA

DAFTAR PUSTAKA

[1] M.J. Zaki dan Ching-Jui Hsiao, "*Efficient Algorithms for Mining Closed Itemset and Their Lattice Structure*", *IEEE Transaction on Knowledge and Data Engineering*, vol 17, no. 4, pp. 462-478, April 2005

[2] M.J. Zaki, "*Scalable Algorithms for Association Mining*", *IEEE Transaction on Knowledge and Data Engineering*, vol 12, no. 3, pp. 372-390, Mei-Juni 2000

[3] Jiawei Han dan Micheline Kamber, "*Data Mining: Concept and Techniques*", Morgan Kaufmann, 2000

[4] Hera Hikmah Evianti, "*Perancangan dan Pembuatan Perangkat Lunak Data Mining untuk Pencarian Kaidah Asosiasi dengan metode Bottom-Up*", Tugas Akhir Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember, 2002

[5] R. Agrawal, et al, "*Fast Discovery of Association Rules*", *Advanced in Knowledge Discovery and Data Mining*, U. Fayyad et all., eds., pp. 307-328, Menlo Park, Calif.: AAAI Press, 1996

[6] Frequent Itemset Mining Dataset Repository written by Bart Goethals, Online, Available from [http://fimi.cs.helsinki.fi/data/], 2004

[7] IlliMine written by Data Mining Research Group, DAIS Research Laboratory Department of Computer Science, University of Illinois at Urbana-Champaign, Online, Available from [http://illimine.cs.uiuc.edu/], 2005

LAMPIRAN

LAMPIRAN A

HASIL UJI COBA PERANGKAT LUNAK

A.1. HASIL UJI COBA UNTUK DATASET MUSHROOM

minsup(%)	mintrans	time(sec)	memory(kB)	Total FCI	k	FCI Length[k]
25	19660	75	1835	729	1	1
					2	21
					3	75
					4	121
					5	159
					6	181
					7	111
					8	41
					9	14
					10	4
					11	1
20	19660	146	1975	1197	1	1
					2	21
					3	99
					4	169
					5	234
					6	290
					7	231
					8	97
					9	33
					10	12
					11	4
					12	4
					13	0
					14	1
					15	1
15	1219	281	3587	2261	1	1
					2	21
					3	119
					4	254
					5	397
					6	545
					7	497
					8	265
					9	114
					10	34
					11	7
					12	4
					13	0
					14	1
					15	2

A.2. HASIL UJI COBA UNTUK DATASET TRANS400K

minsup(%)	mintrans	time(sec)	memory(kB)	Total FCI	k	FCI Length[k]
0.8	3147	66	33095	464	1	827
0.4	1574	124	47787	674	1	668
					2	6
0.1	394	4627	61793.5	12392	1	893
					2	6914
					3	1703
					4	1397
					5	838
					6	426
					7	165
					8	50
					9	6
0.09	354	4449	81074	17208	1	904
					2	8966
					3	2566
					4	2243
					5	1482
					6	753
					7	222
					8	60
					9	11
					10	1
0.08	315	6723	93435	22698	1	912
					2	11557
					3	3596
					4	3008
					5	2121
					6	1001
					7	380
					8	109
					9	13
					10	1

LAMPIRAN B

CONTOH KELUARAN PERANGKAT LUNAK

B.1. KELUARAN UNTUK BASIS DATA CONTOH

```
*****************************************************************
          Tugas Akhir (Yenny Kurniawati - 5201 100 005)
*****************************************************************

Parameter         >>>
Scanning Table      : DATA_PAPER
Jumlah Transaksi    : 6
Jumlah Item         : 5
Jumlah Record       : 23
Minimum Support     : 50 % [3 transaksi]


FREQUENT CLOSED ITEMSETS        >>>

  FCI Length [1] : 1
  FCI Length [2] : 3
  FCI Length [3] : 2
  FCI Length [4] : 1


{C} : 6
{W,C} : 5
{T,C} : 4
{D,C} : 4
{A,W,C} : 4
{D,W,C} : 3
{T,A,W,C} : 3


Time Information >>>
Scanning Time       : 0:0:0:711
Filtering Time      : 0:0:0:280
Ordering Time       : 0:0:0:20
ChARM-L Time        : 0:0:0:821

Memory usage        : 1122.5 kB
```

```
{36,60,58} : 3051
{36,60,52} : 3041
{36,60,29} : 3039
{29,52,58} : 3169
{62,52,58} : 3049
{62,29,58} : 3045
{40,52,58} : 3158
{62,40,58} : 3045
{40,29,58} : 3154
{7,52,58} : 3064
{40,29,52} : 3144
{7,29,58} : 3068
{7,29,52} : 3058
{60,52,58} : 3137
{7,40,58} : 3049
{60,29,58} : 3135
{7,40,52} : 3039
{60,29,52} : 3125
{7,40,29} : 3043
{60,40,58} : 3123
{60,40,52} : 3113
{60,40,29,52} : 3100
{36,29,52,58} : 3072
{36,40,52,58} : 3061
{36,40,29,58} : 3057
{36,40,29,52} : 3047
{36,60,52,58} : 3040
{36,60,29,58} : 3038
{40,29,52,58} : 3143
{7,29,52,58} : 3057
{7,40,52,58} : 3038
{60,29,52,58} : 3124
{7,40,29,58} : 3042
{60,40,52,58} : 3112
{60,40,29,58} : 3110
{36,40,29,52,58} : 3046
{60,40,29,52,58} : 3099


Time Information >>>

Scanning Time       : 0:0:0:281
Filtering Time      : 0:0:0:297
Ordering Time       : 0:0:0:0
ChARM-L Time        : 0:0:8:16

Memory Usage        : 1974.5 kB
```