PERBANDINGAN KECEPATAN DALAM PENCARIAN *FREQUENT ITEMSET* ANTARA ALGORITMA *FP-GROWTH* DAN *CUT BOTH WAYS*

SKRIPSI

oleh:
FATMA RIKA FEBRIYANA
0310960030-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2009

UNIVERSITAS BRAWIJAYA

PERBANDINGAN KECEPATAN DALAM PENCARIAN *FREQUENT ITEMSET* ANTARA ALGORITMA *FP-GROWTH* DAN *CUT BOTH WAYS*

SKRIPSI

Sebagai salah satu syarat dalam memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

oleh:
FATMA RIKA FEBRIYANA
0310960030-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2009


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

PERBANDINGAN KECEPATAN DALAM PENCARIAN *FREQUENT ITEMSET* ANTARA ALGORITMA *FP-GROWTH* DAN *CUT BOTH WAYS*

Oleh:
FATMA RIKA FEBRIYANA
0310960030-96

Setelah dipertahankan di depan Majelis Penguji
Pada tanggal 25 Juni 2009
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

Pembimbing I	Pembimbing II
Dian Eka R., SSi.,M.Kom	**Dany Primanita K., ST**
NIP. 132 300 224	**NIP. 132 310 159**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Agus Suryanto, MSc
NIP. 132 126 049

LEMBAR PERNYATAAN

Saya yang bertanda tangan dibawah ini :

Nama **:** Fatma Rika Febriyana
Nim **:** 0310960030
Jurusan **:** Matematika
Penulis skripsi berjudul **:**Perbandingan Kecepatan dalam Pencarian *Frequent Itemset* antara Algoritma *FP-Growth* dan *Cut Both Ways.*

Dengan ini menyatakan bahwa:

1. **Isi dari skripsi yang saya buat ini adalah bener-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam skripsi ini.**
2. **Apabila dikemudian hari ternyata skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menganggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 25 Juni 2009
Yang menyatakan,

(Fatma Rika Febriyana)
NIM. 0310960030

ABSTRAK

Penggalian kaidah asosiasi (*mining association rules*) merupakan salah satu proses *data mining* untuk menemukan pola dan aturan (*rule*) dari sekumpulan data yang besar. Pola-pola ini merupakan kumpulan item (*itemset*) yang sering muncul secara bersamaan (*frequent itemset*) dalam transaksi pada basis data. Proses pencarian *frequent itemset* membutuhkan waktu yang sangat lama, oleh karena itu diperlukan suatu algoritma yang bisa mengefisiensi waktu yang dibutuhkan.

Algoritma *FP-Growth* menerapkan strategi pencarian dengan menggunakan struktur yang sederhana (*FP-tree*) dan memiliki kinerja yang tinggi karena hanya memerlukan dua kali pemeriksaan pada basis data.

Algoritma *Cut Both Ways* (*CBW*) menggunakan gabungan beberapa teknik dan menggunakan *cutting level* (α) untuk membagi ruang pencarian menjadi dua bagian. Nilai *cutting level* merupakan nilai rata-rata dari kardinalitas *frequent itemset,* diharapkan banyak ditemukan *frequent itemset* pada level ini.

Pada tugas akhir ini akan mengimplementasikan proses pencarian *frequent itemset* dengan menggunakan algoritma *FP-Growth* dan *CBW*. Kemudian membandingkan kinerjanya dengan menggunakan beberapa parameter nilai *support* dan diujikan pada beberapa dataset dengan karakteristik yang berbeda.

Hasilnya, algoritma *FP-Growth* mampu menambang *frequent itemset* lebih cepat daripada algoritma *CBW*.

Kata Kunci : *mining association rules, itemset, frequent itemset, support, FP-tree, FP-Growth, cutting level, Cut Both Ways.*

ABSTRACT

Mining association rules is a data mining process to find rule and pattern from a large database. The pattern can be frequent itemset from the transaction of databases. Frequent itemset generation is most time-consuming process, so we need an algorithm that can be eficient a time consuming.

FP-Growth algorithm employs search strategy using compact structure (FP-tree) resulting in a high performance algorithm that requires only two database passes.

Cut Both Ways (CBW) combine a various technic and use cutting level (α) to divide a search space into two different part. Cutting level is an average cardinality of frequent itemsets, expecting that most of the frequent itemsets will apear in this level.

In this final project will implement frequent itemset generation using FP-Growth and CBW algorithm. Then, compare its performance by using different parameter of minimum support and testing it in some dataset with different characteristic.

The result, FP-Growth algorithm can mining frequent itemsets faster than CBW algorithm.

Keywords : mining association rules, itemset, frequent itemset, support, FP-tree, FP-Growth, cutting level, Cut Both Ways.

KATA PENGANTAR

Assalamualaikum Wr. Wb.

Alhamdulillahi robbil'alamin, Segala puji bagi Allah SWT. Tuhan seluruh alam, yang atas rahmat, hidayah, serta semata-mata atas izin-Nyalah maka penulis dapat menyelesaikan skripsi ini. Semoga Allah melimpahkan rahmat atas Nabi Muhammad SAW yang senantiasa memberikan cahaya petunjuk, dan atas keluarganya yang baik dan suci, dengan rahmat dan berkah-Nya menyelamatkan kita pada hari akhirat.

Penulisan skripsi, yang berjudul "Perbandingan Kecepatan dalam Pencarian *Frequent Itemset* antara Algoritma *FP-Growth* dan *Cut Both Ways*", ini masih jauh dikatakan sempurna. Tapi apa yang penulis sampaikan dalam skripsi ini adalah sebuah bentuk usaha untuk memberikan kontribusi dalam bidang *data mining* khususnya dan ilmu komputer pada umumnya. Dan selain itu penulisan skripsi ini sebagai kewajiban atau syarat untuk menempuh gelar sarjana. Semoga skripsi ini bisa bermanfaat bagi penulis sendiri dan seluruh pembaca.

Dalam penyelesain laporan ini, penulis telah mendapat begitu banyak bantuan dari banyak pihak. Penulis ingin menyampaikan penghargaan dan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Ibu Dian Eka R., S.Si., M.Kom, selaku pembimbing utama penulisan skripsi ini.
2. Ibu Dani Primanita, ST, selaku pembimbing pendamping.
3. Bapak Drs. Mardji, M.T, selaku pembimbing akademik.
4. Bapak Wayan Firdaus Mahmudy, S.Si., M.T, selaku ketua program studi Ilmu Komputer Jurusan Matematika FMIPA Universitas Brawijaya.
5. Segenap Bapak Ibu dosen FMIPA UB yang telah membagi ilmunya kepada Penulis.
6. Kedua Orangtua Penulis, Tia, Riko, Mas Rully, Pak Suhandi, Pak Dwi dan Ibu, yang selalu memberi dukungan kepada penulis baik secara spiritual maupun materiil.
7. Seluruh teman-teman penulis yang tidak bisa penulis sebutkan satu persatu di sini.

Wassalamualaikum Wr. Wb.

Malang, 25 Juni 2009

Fatma Rika Febriyana

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR ISTILAH

Algoritma	:	Metode presisi yang digunakan komputer untuk menyelesaikan masalah. Tersusun atas sekumpulan langkah-langkah berhingga, dimana mungkin tiap langkah memerlukan 1 operasi atau lebih.
Association rule	:	hubungan antara *itemset* yang menyatakan bahwa kemunculan suatu *itemset* mempengaruhi kemunculan *itemset* yang lain. *Association rule* dilambangkan dengan X → Y, X dan Y adalah *itemset*. Contoh : {a,b}→{c}.
Basisdata transaksi	:	basisdata yang setiap barisnya merepresentasikan sebuah transaksi. Setiap transaksi mempunyai identitas yang unik (*tid*), dan mempunyai himpunan *item*, misalkan ada sebuah transaksi *T*=(*tid*, *X*) adalah sebuah *tuple* dimana *tid* adalah sebuah identitas transaksi dan *X* adalah sebuah himpunan *item*.
Cutting level	:	Rata-rata kardinalitas dari *frequent itemset.*
Data mining	:	merupakan salah satu proses KDD yang sangat penting dalam usaha menemukan pola-pola yang menarik dari sejumlah data yang besar.
Dataset transaksi	:	himpunan transaksi.
Frequent item	:	*item* yang kemunculannya sama atau lebih besar dari *minimal support*.
Frequent itemset	:	*itemset* yang kemunculannya sama atau lebih besar dari *minimal support*.
Item	:	bagian terkecil dari suatu transaksi, misalkan ada transaksi *T*=(1, {*a b*}), 1 adalah identitas transaksi *T*, {*a b*} adalah

himpunan *item*, *a* dan *b* adalah *item*.

Itemset	:	suatu kumpulan *item*.
Maximal frequent itemset	:	*frequent itemset* yang tidak memiliki *superset* yang juga *frequent itemset,* atau dengan kata lain merupakan *frequent itemset* yang bukan merupakan subset dari *frequent itemset* lainnya. Dari *maximal frequent itemset* inilah akan dibangkitkan kaidah asosiasi.
Minimal support	:	jumlah kemunculan minimal suatu *itemset* untuk dikategoritkan sebagai *itemset* yang frekuen atau disebut juga dengan *frequent itemset* atau *frequent pattern*.
Support	:	*nilai yang menyatakan seberapa sering kemunculan suatu itemset dalam suatu dataset transaksi*

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi telah memungkinkan data dalam jumlah besar terakumulasi, yang biasa disebut dengan basis data. Seiring dengan hal tersebut, dikembangkan pula berbagai cara untuk mendapatkan informasi dari basis data. *Data mining* merupakan salah satu cara untuk menggali informasi yang terpendam tersebut.

Data mining merupakan serangkaian proses untuk menggali nilai tambah dari kumpulan data berupa pengetahuan yang selama ini tidak diketahui secara manual. Ada berbagai macam teknik dalam data mining, dimana penggunaannya tergantung dari permasalahan yang dihadapi. Salah satu tekniknya adalah kaidah asosiasi atau *assosiation rule*.

Kaidah asosiasi digunakan untuk menemukan aturan asosiatif antara suatu kombinasi item. Misalnya, ditemukan pola pada supermarket, yang berupa kombinasi antara item popok dan item bedak bayi, dimana sebagian besar pembeli yang membeli popok cenderung juga untuk membeli bedak bayi.

Syarat untuk membentuk aturan asosiasi tersebut adalah dengan mencari item-item yang sering muncul atau yang biasa disebut *frequent itemset*. Permasalahan efisiensi dalam pencarian *frequent itemset* adalah proses minimalisasi pembacaan basis data dan penghitungan nilai *support*. Pada data yang sangat besar, untuk mendapatkan *frequent itemset* diperlukan waktu yang sangat lama.

Telah banyak dikembangkan algoritma-algoritma kaidah asosiasi dalam mengatasi permasalahan *frequent itemset* diatas. Algoritma tersebut diantaranya: *Apriori* (Agrawal,dkk., 1993), *FP-growth* (J. Han, dkk., 2000), *Cut Both Ways* (Hwung Su dan Yan Lin, 2004), *DHP*, *DIC*, dan yang lainnya.

Dari sekian banyak algoritma tersebut, yang menarik perhatian adalah *FP-growth* dan *Cut Both Ways* atau *CBW*. *FP-growth* yang menggunakan struktur data *FP-tree*, banyak disebut lebih efisien waktunya dalam pencarian *frequent itemset* (Pramudiono, 2003). Sedangkan algoritma *CBW* merupakan algoritma terbaru yang menghandalkan penggabungan beberapa teknik dan menggunakan

cutting level (α) untuk membagi ruang pencarian menjadi dua bagian (Sukarya, 2006).

Dalam skripsi ini akan dibandingkan kecepatan algoritma *FP-growth* dan *CBW* dalam pencarian *frequent itemset*-nya. Pada penelitian ini kedua algoritma akan diujikan pada beberapa dataset dengan karakteristik yang berbeda-beda dan pada berbagai tingkatan nilai *minimal support*.

1.2 Rumusan Masalah

Permasalahan yang akan dijadikan obyek penelitian pada skripsi ini adalah:

1. Bagaimana menerapkan algoritma *FP-growth* dan algoritma *CBW* untuk pencarian *frequent itemset*?
2. Bagaimana menganalisa kecepatan pencarian *frequent itemset* antara algoritma *FP-growth* dan algoritma *CBW,* yang diujikan pada beberapa dataset dengan karakteristik yang berbeda-beda dan pada berbagai tingkatan nilai *minimal support*?

1.3 Tujuan Penelitian

Tujuan dari skripsi ini adalah:

1. Menerapkan algoritma *FP-growth* dan algoritma *CBW* untuk pencarian *frequent itemset,*
2. Menganalisa kecepatan pencarian *frequent itemset* antara algoritma *FP-growth* dan algoritma *CBW,* yang diujikan pada beberapa dataset dengan karakteristik yang berbeda-beda dan pada berbagai tingkatan nilai *minimal support*.

1.4 Manfaat Penelitian

Penelitian ini dapat bermanfaat bagi pengembang perangkat lunak, untuk sebuah perusahaan yang mengutamakan masalah waktu dalam mengelola data, dengan cara membantu memberikan gambaran seberapa cepat pencarian *frequent itemset* dengan menggunakan algoritma *CBW* dibandingkan dengan menggunakan algoritma *FP-growth*.

1.5 Sistematika Penulisan

Pembuatan tugas akhir ini dilakukan dengan pembagian bab sebagai berikut:

BAB I : PENDAHULUAN

Bab ini membahas mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, serta sistematika penulisan tugas akhir.

BAB II : TINJAUAN PUSTAKA

Bab ini mencantumkan beberapa tinjauan pustaka yang berkaitan dengan penelitian ini, diantaranya : *Data Mining*, *Association Rule,* Algoritma *CBW,* Algoritma *FP-growth,* dan teknik-teknik pencarian *frequent itemset* yang digunakan.

BAB III : METODOLOGI DAN PERANCANGAN SISTEM

Bab ini menerangkan beberapa hal mengenai metode penelitian dan langkah-langkah yang harus dilakukan. Beberapa poin yang dibahas ialah: Perangkat Lunak dan Perangkat Keras, Pengumpulan Data, Pola dan Rancangan Penelitian, serta Penerapan Algoritma *CBW* dan *FP-growth*.

BAB IV : HASIL DAN PEMBAHASAN

Bab ini menerangkan proses implementasi dari rancangan penelitian yang dijelaskan pada bab III.

BAB V : KESIMPULAN DAN SARAN

Bab lima ini berisi kesimpulan dari pembahasan dan saran yang diharapkan bermanfaat untuk pengembangan tugas akhir ini selanjutnya.

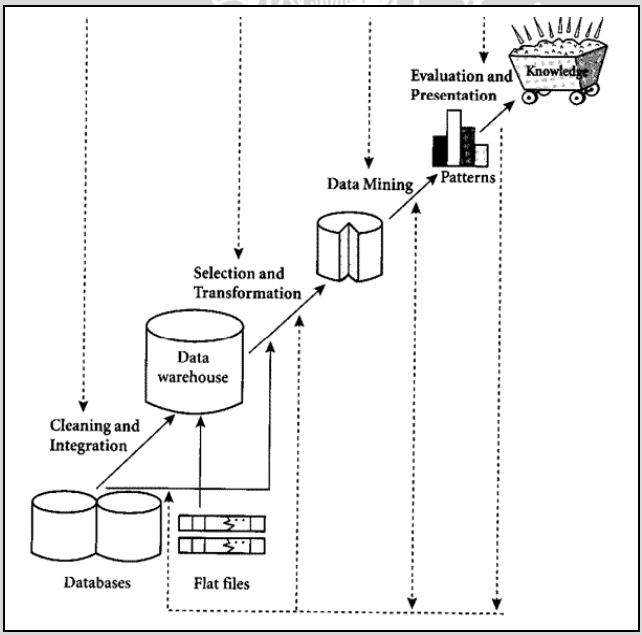

UNIVERSITAS BRAWIJAYA

BAB II
TINJAUAN PUSTAKA

2.1 *Data Mining*

Data mining adalah serangkaian proses untuk menggali nilai tambah dari suatu kumpulan data, berupa pengetahuan yang selama ini tidak diketahui secara manual (Pramudiono, 2003). *Data mining* dipengaruhi oleh beberapa bidang ilmu, diantaranya: *database, information science, high performance computing,* visualisasi, *machine learning,* statistika, jaringan syaraf tiruan, pemodelan matematika, pengolahan citra, pengenalan pola, *information retrival* dan *information extraction.*

Data mining dapat dibagi menjadi beberapa tahap yang diilustrasikan pada Gambar 2.1.



Gambar 2.1 Tahapan *data mining*
Sumber: Iko Pramudiono, 2003

Berdasarkan penggunaannya, *data mining* dibedakan menjadi beberapa macam, seperti: *Transaction mining* (*mining* terhadap data transaksi), *text mining* (*mining* terhadap data teks), *web mining* (*mining* terhadap data yang berhubungan dengan *web*), dan lain-lain.

Karena definisi *data mining* yang luas, ada banyak jenis teknik analisa yang dapat digolongkan dalam *data mining* (Pramudiono, 2003). Beberapa diantaranya yang cukup terkenal adalah: kaidah asosiasi atau *association rule,* klasifikasi atau *classification, clustering,* pola urutan atau *sequential pattern,* dll.

Algoritma kaidah asosiasi diantaranya: Apriori, *FP-growth*, *CBW*, dll. *Classification,* metode yang terkenal decision tree dan algoritmanya: C-45, *Rain Forrest*, dll. Sedangkan *clustering* mempunyai beberapa metode diantaranya: partisi dan hirarki (*bottom-up* dan *top-down*).

2.2 *Association Rule*

Menurut Gopalan (2004), *association rule* atau kaidah asosiasi sejak ditemukan pertama kali oleh Rakesh Agrawal pada tahun 1993, telah menjadi fokus penelitian di berbagai komunitas (misal data mining dan kecerdasan buatan). *Association rule* digunakan untuk menemukan pola dari kumpulan data, dengan cara menemukan aturan-aturan yang mengasosiasikan data yang satu dengan data yang lain. Biasanya yang sering dipakai contoh adalah transaksi jual-beli pada toko serba ada atau sejenisnya. Untuk mencari kaidah asosiasi pada data transaksi tersebut, pertama-tama harus dicari terlebih dahulu item yang paling sering muncul atau *frequent item*, misalnya barang yang sering dibeli oleh pengunjung.

Selanjutnya adalah menentukan *support* dan *confidence* untuk mereduksi ruang pencarian selama proses *mining*. *Support* merupakan persentase kombinasi item atau biasa disebut nilai penunjang, sedangkan *confidence* adalah parameter untuk menilai kuat tidaknya hubungan antar item atau biasa disebut nilai kepastian.

Support dari kaidah asosiasi $X \Rightarrow Y$ adalah rasio dari *record* atau transaksi yang mengandung $X \cup Y$ dengan total *record* dalam basis data. Sementara *minsup* atau *minimal support* menandakan ambang batas atau *treshold* yang menentukan apakah sebuah *itemset* akan dipergunakan pada perhitungan selanjutnya untuk menetukan kaidah asosiasi, yang disebut *frequent itemset*. *Frequent itemset* merupakan

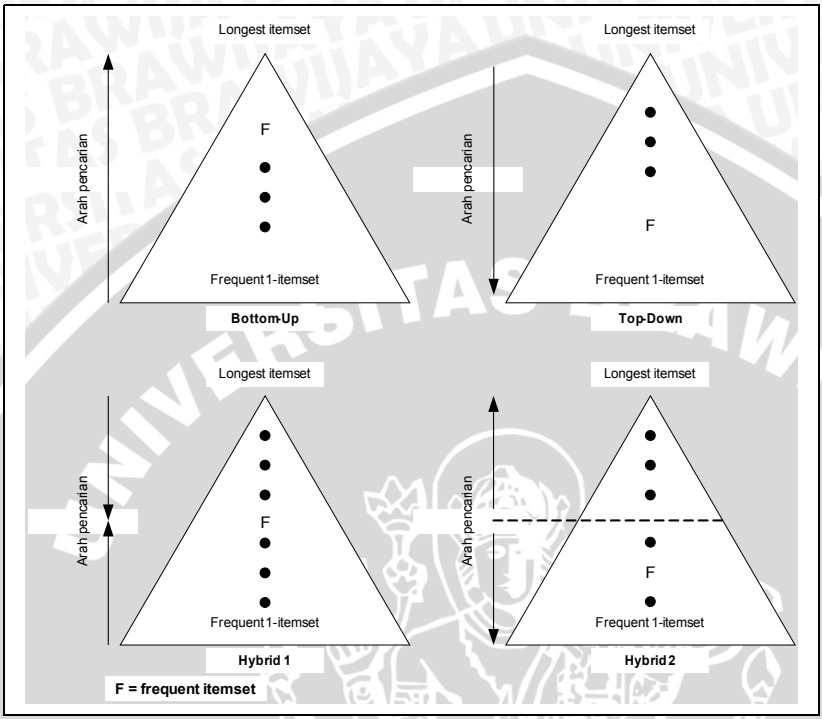
kumpulan item atau *itemset* yang memiliki nilai *support* melebihi *minsup*.

Confidence dari kaidah asosiasi $X \Rightarrow Y$ adalah rasio dari *record* yang mengandung $X \cup Y$ dengan total *record* yang mengandung X. Sementara *minconf* atau *minimal confidence* menandakan ambang batas dari sebuah *itemset* untuk menentukan *confident*. *Confident* merupakan *itemset* yang memiliki nilai *confident* lebih besar dari *minconf*.

Berbagai macam algoritma kaidah asosiasi telah dikembangkan untuk mencari cara agar pencarian *frequent itemsets* lebih efisien, diantaranya: apriori (Agrawal dkk, 1993), *hashing*, OCD, Partition, CBW-*Cut Both Ways* (Hwung Su J., 2004), dan lain-lain. Algoritma-algoritma tersebut mengadopsi teknik yang berbeda dan sudut pandang yang berbeda pula. Akan tetapi dapat dikelompokkan menjadi 3 aspek (Sukarya, 2006), yaitu:

1. Strategi arah pencarian (Gambar 2.2): *top-down, bottom-up,* dan *hybrid*
2. Strategi pencarian: *breadh first search* atau *BFS, dept first search* atau *DFS,* dan *Bi-Directional Search*
3. Strategi perhitungan: *vertical intersection* dan *horizontal counting*

Sedangkan menurut Gopalan (2004), dasar utama algoritma-algoritma *association rule* dalam pengurangan *cost* atau biayanya adalah dengan *Apriori Property*. *Apriori Property* adalah kondisi bila sebuah pola *infrequent* atau tidak sering muncul, maka *superset*-nya tidak akan pernah *frequent.*



Gambar 2.2 Skema Strategi Arah Pencarian *Maximal Frequent Itemset*. Sumber: Sukarya, 2006

2.3 Algoritma *Cut Both Ways* (*CBW*)

Algoritma *Cut Both Ways* (*CBW*) ditemukan oleh Jan-Hwung Su dan Wen-Yang Lin dari Universitas I-Shou Taiwan pada tahun 2004. Algoritma ini menggabungkan beberapa teknik atau strategi. Algoritma ini menggunakan strategi arah pencarian *Hybrid 2* atau *advanced hybrid (*α*-frequent itemset to top* dan α*-frequent itemset to bottom)*. Dimana setelah terbagi 2, strategi *top-down* digunakan untuk menemukan *frequent itemset* yang berada dibawah *cutting level* (α), dikombinasikan dengan strategi pencarian *breadth first search* dan *horizontal counting* untuk penghitungan nilai *support*-nya. Sedangkan strategi *bottom-up* digunakan untuk menemukan *frequent itemset* yang berada diatas *cutting level,* dikombinasikan dengan pencarian *depth first search* dan *vertical intersection* untuk menghitung *support* bagian atas nilai cutting level tersebut (Sukarya, 2006).

Cutting level (α) menurut Hwung Su J. (2004) merupakan pembulatan nilai rata-rata dari jumlah *frequent itemset,* yang secara matematis dapat didefenisikan sebagaimana persamaan 2.1.

$$\alpha = INT\left[\frac{\sum \left|t_{i\perp\, minsupp}\right|}{|D|}\right] \ldots.(2.1)$$

dimana INT[*r*] menandakan pembulatan dari nilai *r*, untuk $r \geq 1$, D adalah tabel transaksi atau jumlah transaksi yang ada, t_i merupakan transaksi ke-i, dan $t_{i\perp minsup}$ adalah kelompok dari *item-item* dalam t_i dengan *support* lebih dari *minsup*, atau secara spesifik seperti persamaan 2.2.

$$t_{i\perp minsup} = \{ x \mid x \in t_i \text{ dan } supp(x) \geq minsup\} \ldots.(2.2)$$

Algoritma *CBW* terdiri dari sebuah algoritma utama yang akan mengeksekusi 3 buah prosedur yaitu: prosedur *Trans*, prosedur *DwnSearch* dan prosedur *UpSearch*.

Algoritma Utama *CBW*

Input: Basis Transaksi (D) dan nilai *minimum support* (minsup);

Output: Gugus *maximal frequent itemset* (F);

1. `Scan D untuk membangkitkan semua frequent 1-itemsets in` F_1;
2. `Trans(D, T,` F_1`,` F_2`,` α`);`
3. *`Dwnsearch(D, DF,`* F_α*`,`* α*`, minsupp);`*
4. *`Upsearch(T, UF,`* F_α*`,`* α*`, minsupp);`*
5. [illegible]

Sumber: Sukarya, 2006

Algoritma ini dimulai dengan melakukan pembacaan pada basis data untuk membangkitkan semua F_1 *(frequent 1-itemset)*, yang berguna untuk menghitung *cutting level*.

Prosedur Trans

```
1. C2 =  kandidat 2-itemset yang dibangkitkan
   dari F1;
2. numf =0;
3. for i = 1 to |D|||||| do
4.       scan ti transaksi ke-i;
5.  hapus item-item pada  ti yang tidak berada
   dalam F1;
6.       numf + =| ti|
7.       if | ti|≥ 3 then
8. tambahkan TlD ini ke dalam TlDlist dari
   setiap item dalam ti;
9.       for setiap 2-subset X dari ti dan X ∈ C2
   do
10.            X.count ++;
11.end for
```

Sumber: Sukarya, 2006

Prosedur *Trans* bertanggung jawab untuk melakukan tiga tugas, yaitu:
(1) menghitung *cutting level* α,
(2) membangkitkan F_2 (*frequent 2-itemset*), dan
(3) mentransformasikan basis data ke vertikal *TIDlist T*.
Setiap kali melakukan *scan* terhadap transaksi *t*, jumlah *frequent item* diakumulasikan untuk perhitungan α selanjutnya. Begitu juga untuk setiap *frequent item*, *TID*-nya ditambahkan ke *TIDlist* jika jumlah dari *t* tidak kurang dari 3. Intinya, untuk semua transaksi yang jumlahnya lebih besar dari α harus disimpan, sebab *UpSearch* dimulai dari level α+1. Kardinalitas (nilai) 3 digunakan karena akan didapatkan hasil yang terbaik untuk memfasilitasi proses *tidlist pruning*. (Hwung Su J., 2004)

Selanjutnya dieksekusi prosedur *DwnSearch*. Menurut Hwung Su J. (2004), setiap transaksi di-*scan*, dan berdasarkan dalil 1, *item-item* yang muncul kurang dari 2 buah *frequent itemset* dalam F_2 dibuang (*pruned*). Berikutnya melakukan enumerasi untuk membangkitkan kandidat *itemset* dengan jumlah antara α dan 3, dan menghitung nilai *support*-nya.

Prosedur DwnSearch

```
1. for i = 1 to |D|||||| do
2.    scan transaksi ke-i, t_i;
3. hapus item-item pada t_i yang kemunculannya
   kurang dari 2 itemset dalam F_2;
4.    for setiap subset X dari t_i dan 3≤|X|≤α do
5.       X.count ++;
6. end for
7. DF = {X|sup (X) ≥ minsup};
```

Sumber: Sukarya, 2006

Prosedur UpSearch

```
1. if Fα = Ø return UF = Ø;
2. baca gugus TIDlists T dan buang non-candidate
   TIDlists;
3. k = α,, F_k=F_α ;
4. repeat
5.   k ++;
6.   C_k = kandidat k-itemsets yang baru, yang
   dibangkitkan dari F_k -1;
7.   for setiap X ⊆ C_k do
8.     tampilkan bit-vector intersection pada X;
9.     hitung support dari X, sup(X);
10. end for
11. F_k = {X| sup (X) ≥ minsupp,X ∈ C_k};
12. UF = UF ∪ F_k
```

Sumber: Sukarya, 2006

Setelah membangkitkan semua *frequent itemset* yang memiliki jumlah tidak kurang dari *α*, *CBW* melakukan prosedur *UpSearch* untuk mencari *frequent itemset* yang mungkin muncul lagi. Di sini pencarian yang digunakan mengikuti paradigma pencarian dari *Apriori*, yaitu membangkitkan *frequent itemset* per level dengan strategi arah pencarian *bottom-up* dimulai dari *frequent itemset* level α. Pertama kali memeriksa jika F_α kosong. Jika tidak, maka membaca *tidlist* T dan membuang *tidlist* untuk itemset yang muncul kurang dari α *frequent itemset* pada F_α. Berikutnya untuk mempercepat

proses interseksi *tidlist* untuk penghitungan nilai *support* digunakan teknik *fast intersection* dan *caching intermediate result* (Sukarya, 2006).

Bila *cutting level* benar atau sesuai, maka *maximal frequent itemset* akan ditemukan pada level ini, sehingga pencarian *DownSearch* hanya dilakukan dengan sekali pembacaan basis data. Hal ini yang diharapkan oleh Sukarya (2006) dan Hwung Su (2004) menjadikan CBW lebih efisien dari algoritma yang lain.

2.4 Algoritma *FP-Growth*

Algoritma *FP-growth* adalah algoritma pencarian pola yang sering muncul (*frequent pattern*) berdasarkan struktur data *FP-tree* (*Frequent Pattern tree*). Menurut Gopalan (2004), algoritma *FP-growth* lebih efisien daripada algoritma-algoritma yang berdasar pada sistem *generate and test*. Karena *FP-growth* menggunakan strategi *devided and conquer*, dimana membagi suatu permasalahan besar menjadi permasalahan-permasalahan yang lebih kecil (*devide*), pembagian dilakukan terus sampai ditemukan bagian masalah kecil yang mudah untuk dipecahkan (*conquer*).

Struktur data *FP-tree* merupakan perluasan dari struktur *prefix tree*. Setiap *node* pada struktur data ini merepresentasikan satu item yang sering muncul (*frequent 1-itemset*). Algoritma *FP-growth* mencari himpunan item yang sering muncul dengan cara membangun *FP-tree* secara rekursif, dan menggabungkan *frequent 1-itemset* yang ada pada (*conditional*) *FP-tree* secara berturut-turut.

FP-tree adalah sebuah *tree*, yang terdiri dari: satu *header table,* satu *root* yang diberi label 'null', dan satu himpunan item *prefix subtree* sebagai node dari anak *root.* Masing-masing *entry* dalam *header table* adalah *frequent item,* dan setiap *record* terdiri dari dua atribut yaitu nama item (*item_name*) dan *head of node-link*. Sedangkan setiap *node* anak terdiri dari atribut: nama item, *count,* dan *node-link*. Dimana *count* menunjukkan jumlah transaksi direpresentasikan oleh cabang yang mengandung *node* tersebut, dan *node-link* menghubungkan node ke node berikutnya pada tree tersebut yang mempunyai nama item yang sama atau *null* jika tidak ada (Maryeti, 2006).

Pembentukan **FP-tree**

Input : Basis Transaksi (D) dan nilai *minimum support* (minsup);

Output : *frequent pattern tree,* FP-tree;

Method : FP-tree dibentuk dengan cara berikut.

1. Transaksi di *database* (DB) ditelusuri sekali agar didapat himpunan *frequent item* (F) dan nilai *support*-nya. Lalu F diurutkan mulai dari yang mempunyai nilai *support* terbesar sampai terkecil, dan hasil pengurutannya dilambangkan dengan L (daftar *frequent item*).
2. Buat *root* dari FP-tree, *T*, dan diberi nilai *null*. ***Foreach*** transaksi *Trans* di *DB* lakukan langkah berikut.

   Pilih dan diurutkan *frequent items* dalam *Trans* (disebut [*p*| P]) mengacu pada L, dimana *p* adalah elemen pertama dan P adalah *item* berikutnya. Panggil ***insert_tree*([*p*|*P*],*T*).**

   Fungsi ***insert_tree*([*p*|*P*],*T*)** terdiri dari langkah-langkah berikut. **if** *T* punya anak *N* dimana *N*.nama-item = *p*.nama-item, maka nilai N bertambah 1; **else** buat node baru *N*, beri nilai 1, N.*parent.link* = *T,* dan *N.link* = node yang sama nama itemnya dihubungkan oleh struktur *node-link*. **if** P tidak kosong, **then** panggil *insert_tree*(*P,N*)secara rekursif.

Sumber: Jiawei Han, 2000

Proses pada algoritma *FP-growth* dimulai dengan memeriksa apakah *tree* mempunyai cabang tunggal. Jika kondisi terpenuhi maka *frequent itemset* didapat dengan mengkombinasikan semua item yang ada pada cabang tersebut, dan tidak perlu melakukan proses *mining* lagi terhadap *FP-tree*. Hal ini merupakan salah satu fitur efisiensi dari *FP-growth*.

Algoritma *FP-growth*
Input : *FP-tree Tree,* Basis Transaksi (D) dan nilai *minimum support* (minsup);
Output : sekumpulan lengkap pola *frequent*;
Method : panggil *FP-growth*(FP-tree, null)
Procedure : *FP-growth*(*Tree*,α)
{
(1) **if** *Tree* mengandung a single path *P* **then**
(2) **for each** kombinasi (β) dari node-node pada *path P* **do**
(3) bangkitkan pola $\beta \cup \alpha$ dengan *support = min-support* node di β;
(4) **else for each** a_i pada *header tree* **do{**
(5) bangkitkan pola $\beta = a_i \cup \alpha$ dengan *support* = a_i.*support*;
(6) bentuk *conditional pattern base* dari β kemudian *conditional FP-tree* dari $Tree_\beta$;
(7) **if** $Tree_\beta \neq null$
(8) **then** call FP-growth ($Tree_\beta$,β) }
}

Sumber: Jiawei Han, 2000

Jika *tree* mempunyai lebih dari satu cabang, maka satu *frequent pattern* dibangkitkan dan juga dibangun *conditional FP-tree* untuk *frequent pattern* tersebut. Proses pembangunan *conditional FP-tree* sama seperti proses pembangunan *FP-tree*. *Conditional FP-tree* dibangun dari pola dasar *conditional* (*conditional pattern base*). *Conditional pattern base* item i_j adalah himpunan *prefix path* dari node i_j yang diperoleh dari penelusuran *node-link*.

Menurut Jiawei Han (2000), keuntungan dari penggunaan struktur data *FP-tree* selain lebih padat (dense) juga tetap lengkap. Informasi yang tidak relefan dikurangi, item-item yang tidak sering muncul (*infrequent*) dihilangkan, akan tetapi tetap menjaga kelengkapan informasi yang dibutuhkan untuk *frequent pattern mining*.

2.5 Analisis algoritma

Menurut buku ajar Metode Numerik 2002, Analisis algoritma sangat membantu di dalam meningkatkan efisiensi program. Kecanggihan suatu program bukan dilihat dari tampilan program,

tetapi berdasarkan efisiensi algoritma yang terdapat didalam program tersebut. Dalam menguji suatu algoritma, dibutuhkan beberapa kriteria untuk mengukur efisiensi algoritma. Terdapat dua tipe Analisis algoritma, yaitu :

1. Memeriksa kebenaran algoritma
   Dapat dilakukan dengan cara perurutan, memeriksa bentuk logika, implementasi algoritma, pengujian dengan data dan menggunakan cara matematika untuk membuktikan kebenaran.
2. Penyederhanaan Algoritma
   Membagi algoritma menjadi bentuk yang sederhana.

Menurut Hariyanto (2003), ada dua fase analisis komputasi, yaitu *A priori* dan *A posteriori*. *A priori* dugunakan untuk menemukan fungsi dan atau parameter-parameter yang relevan, yang membatasi waktu komputasi algoritma. Sedangkan *A posteriori* adalah fase mngumpulkan statistik nyata konsumsi waktu dan ruang suatu algoritma pada mesin dan bahasa pemrograman tertentu.

Menurut Hariyanto (2003), kinerja algoritma dinyatakan dengan notasi big-oh (O), yaitu waktu algoritma berbanding terhadap suatu fungsi tertentu. $f(n)= O(g(n))$ jika dan hanya jika terdapat satu konstanta $c \geq 0$, dan konstanta $n_0 \geq 0$, sehingga $|f(n)| \leq c\ |g(n)|$ untuk semua $n \geq n_0$, dimana n adalah jumlah masukan ke algoritma. Notasi big O menghilangkan semua konstanta dan faktor kecuali yang dominan. Ada beberapa aturan dalam big O, misalkan $T_1(n)$ & $T_2(n)$ adalah waktu jalan dua potongan program P_1 dan P_2, dan $T_1(n)$ & $T_2(n)$ adalah $O(f(n))$ dan $O(g(n))$, maka diperoleh :

a). Aturan penjumlahan: $T_1(n) + T_2(n) = O(\text{maks}\{f(n),g(n)\})$, dan

b). Aturan perkalian: $T_1(n) \cdot T_2(n)= O(f(n)g(n))$


UNIVERSITAS BRAWIJAYA

BAB III
METODOLOGI DAN PERANCANGAN SISTEM

Cakupan pembahasan pada bab ini meliputi metode, rancangan, dan langkah-langkah yang dilakukan dalam penelitian untuk membandingkan algoritma *FP-growth* dan algoritma *Cut Both Ways* dalam pencarian *frequent itemset.*

Langkah-langkah yang dilakukan dalam penelitian ini meliputi :

1. Melakukan studi literatur mengenai *association rule,* algoritma *FP-growth* dan algoritma *Cut Both Ways.*
2. Mengumpulkan data-data yang diperlukan dalam penelitian.
3. Menganalisa dan melakukan perancangan sistem untuk membandingkan keduanya dalam pencarian *frequent itemset.*
4. Mengimplementasikan rancangan yang dilakukan pada tahap sebelumnya menjadi sebuah perangkat lunak untuk membandingkan algoritma dalam pencarian *frequent itemset.*
5. Melakukan uji coba terhadap perangkat lunak menggunakan beberapa data.
6. Melakukan evaluasi dan analisa hasil waktu komputasi yang dihasilkan dari proses uji coba perangkat lunak terhadap data.

3.1 Peralatan Penelitian

Perangkat lunak yang digunakan pada penelitian ini ialah :

1. Sistem operasi Microsoft Windows XP Professional Edition.
2. Visual Studio 2005 sebagai software development dalam mengembangkan aplikasi untuk menguji waktu proses ketiga metode dalam membentuk *Association Rule*.
3. Microsoft SQL Server 2000 sebagai DBMS *(Database Managemen System)*.

Perangkat keras yang digunakan pada pengamatan ini adalah sebuah *PC(Personal Computer)* dengan spesifikasi sebagai berikut :

Processor 1.73 GHz *dual core*
1014 MB RAM
120 GB HDD

3.2 Pengumpulan Data

Pengumpulan data merupakan usaha untuk memperoleh data atau dokumen yang dibutuhkan dalam penelitian dan untuk selanjutnya data tersebut akan diproses sesuai dengan kebutuhan.

3.2.1 Studi literatur

Studi literatur merupakan cara pengumpulan data yang diperoleh dengan mengumpulkan berbagai sumber kepustakaan, baik berupa buku-buku, jurnal, laporan penelitian, dan lain sebagainya untuk ditelaah lebih lanjut sebagai bahan pendukung penelitian.

3.2.2 Pengumpulan data lapangan

Metode ini digunakan untuk mengumpulkan data-data yang dibutuhkan dalam penelitian. Data yang digunakan adalah data asli dan data sintetis. Data asli yang dipilih adalah data yang telah sering diujikan pada penelitian-penelitian sebelumnya (Gopalan, 2004; Grahne, 2005), yaitu: *pumsb*, *mushroom*, dan *connect4*. Sedangkan data sintetis sebanyak 12 buah.

Berikut dataset tersebut:

a. Dataset T3I4D10K,
b. Dataset T3I4D20K,
c. Dataset T3I4D30K,
d. Dataset T3I4D40K,
e. Dataset T3I4D50K,
f. Dataset T3I4D60K,
g. Dataset T3I4D70K,
h. Dataset T3I4D80K,
i. Dataset T3I4D90K,
j. Dataset T3I4D100K,
k. Dataset T10I4D100K, dan
l. Dataset T25I20D100K.

Dimana T menunjukkan rata-rata jumlah item tiap transaksi, I adalah rata-rata maximal *frequent itemset* yang terbentuk, dan D adalah jumlah transaksi. Dataset a sampai k adalah *sparse dataset.* Sedangkan dataset l termasuk *dense dataset.*

Data asli didapat dari *UCI Machine Learning Database Repositori* (http://www.ics.uci.edu/~mlearn/MLRepository.html). Sedangkan data sistetis di bangkitkan menggunakan IBM Data Generator.

3.2.3 Dialog, diskusi, dan konsultasi

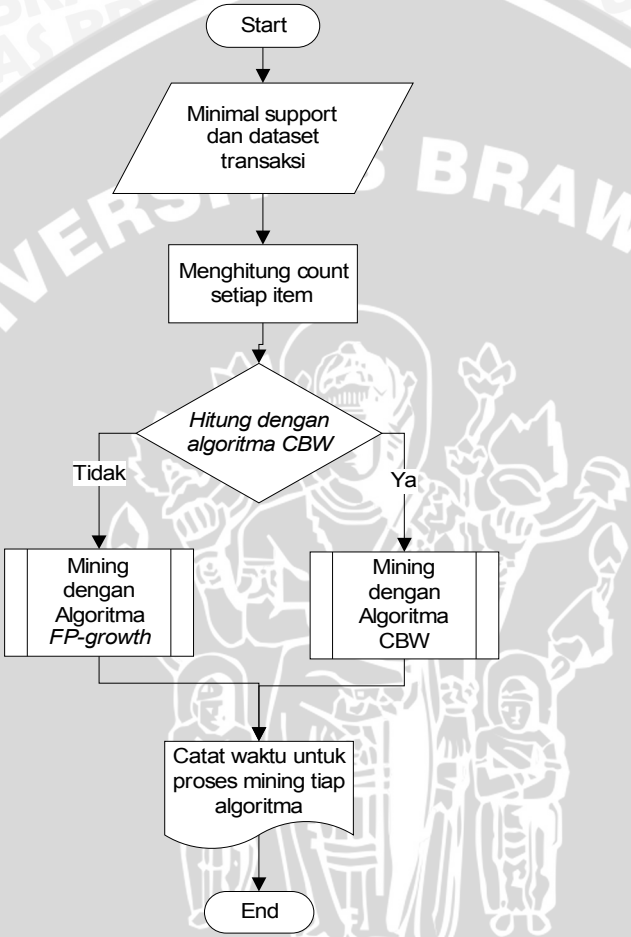
Metode ini dilakukan dengan cara melakukan konsultasi dengan pembimbing skripsi serta melakukan dialog maupun diskusi dengan sumber-sumber lain yang berhubungan dengan penelitian ini untuk memperoleh solusi pengambilan data dan metode pengujian yang efektif.

3.3 Pola dan Rancangan Penelitian

Penelitian ini terdiri dari beberapa tahap, yaitu perancangan dan pembuatan sistem untuk pencarian *frequent itemset* menggunakan algoritma *CBW* dan *FP-growth*, pengujian kebenaran dan kinerja sistem terhadap kedua algoritma dengan menggunakan beragam data, terakhir mengevaluasi dan menganalisa hasil waktu komputasi yang dihasilkan dari proses uji coba perangkat lunak terhadap data.

3.3.1 Rancangan pembuatan sistem

Dalam penelitian ini akan diimplementasikan dua algoritma untuk pencarian *frequent itemset*, yaitu algoritma *CBW* dan *FP-growth*. Metode ini untuk membandingkan *frequent itemset* yang dihasilkan oleh kedua algoritma tersebut. Proses yang dilakukan digambarkan dengan *flowchart* pada gambar 3.1.

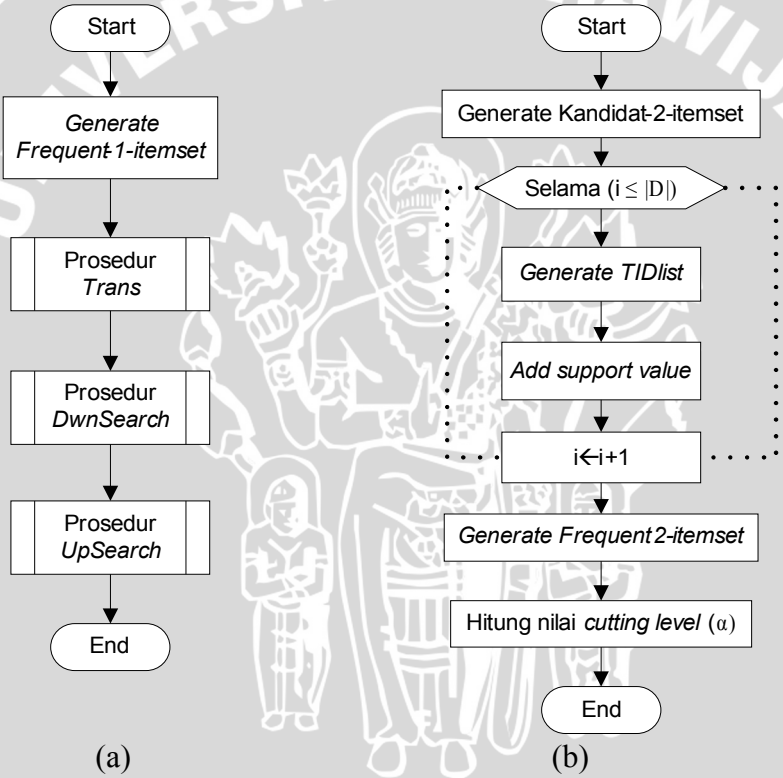


Gambar 3.1 *Flowchart* pencarian *frequent itemset* dengan menggunakan algoritma *CBW* dan *FP-growth*

Penjelasan dari gambar 3.1 adalah:

- Dimulai dengan Pengguna menentukan dataset transaksi yang ingin dicari *frequent itemset*-nya, dan nilai *minimal support* yang diharapkan dari transaksi tersebut.

- Sistem kemudian menghitung *count* atau jumlah setiap item di file transaksi yang berbentuk file teks itu.
- Pengguna menentukan algoritma yang akan digunakan.
- Sistem memproses data yang diinput berdasarkan algoritma yang dipilih.
- Sistem menyimpan data waktu proses dan parameter input ke tabel hasil tes.
- Selesai.



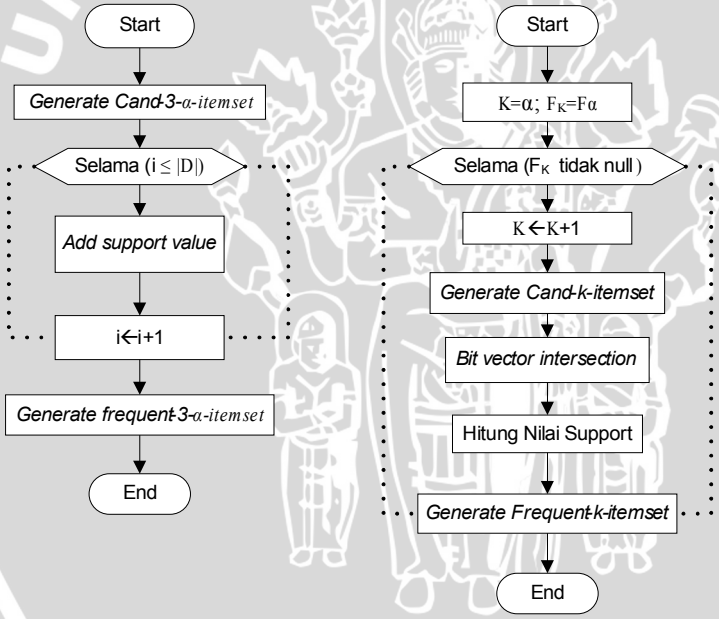
Gambar 3.2 (a) *Flowchart* proses algoritma utama CBW
(b) *Flowchart* prosedur *Trans*

Bila pengguna memilih algoritma CBW, dalam *mining frequent itemset*, maka proses algoritma CBW, berdasarkan flowchart pada gambar 3.2 (a), meliputi:

- sistem akan membangkitkan item yang unik (*frequent* 1-*itemset*), dari dataset yang telah dipilih sebelumnya,
- sistem menjalankan prosedur *Trans*,
- sistem menjalankan prosedur *DwnSearch*, dan
- sistem menjalankan prosedur *UpSearch*.

Pada prosedur *Trans*, hal yang dilakukan sesuai dengan *flowchart* pada gambar 3.2 (b), meliputi:

- sistem membangkitkan kandidat 2-temset,
- untuk setiap transaksi, sistem akan membangkitkan TIDlist dan mengakumulasikan jumlah *frequent item*,
- sistem membangkitkan *frequent 2-itemset,*
- sistem menghitung nilai *cutting level* (α).



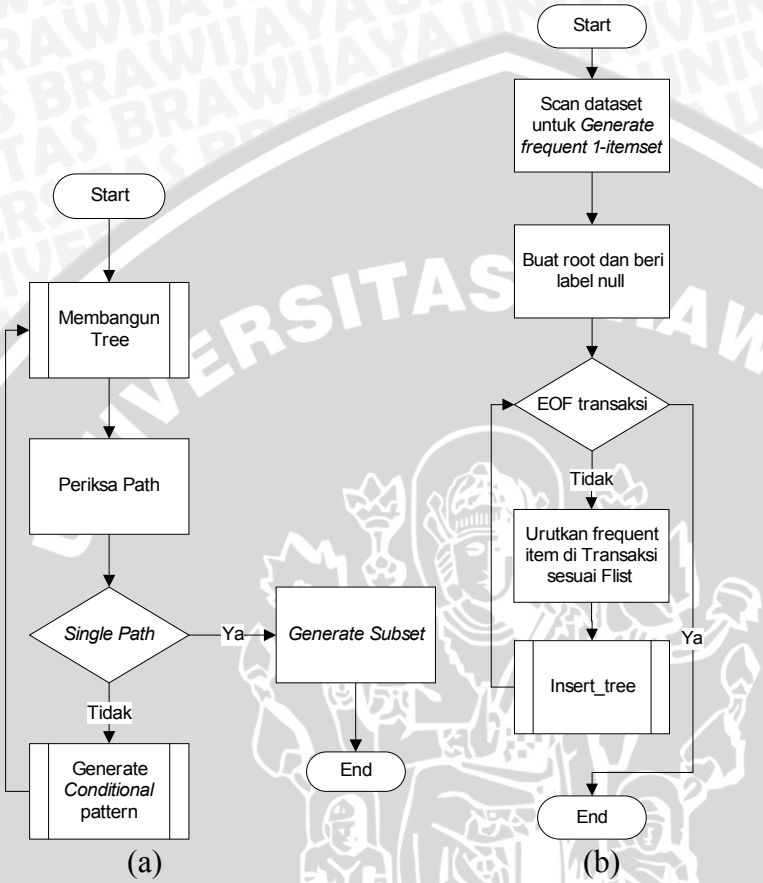
(a) (b)

Gambar 3.3 (a) *Flowchart* Prosedur *DwnSearch*
(b) *Flowchart* Prosedur *UpSearch*

Gambar 3.3 (a) adalah *flowchart* jalannya prosedur *DwnSearch*, dimulai dengan membangkitkan kandidat 3-α-*itemset*, untuk

mendapatkan *frequent* 3-α-*itemset*. Sedangkan gambar 3.3 (b) adalah *flowchart* jalannya prosedur *UpSearch*, yang meliputi:

1. membangkitkan kandidat k-*itemset*, yaitu item yang berada diatas nilai α,
2. menjalankan proses *Bit vektor intersection*,
3. menghitung nilai *support*,
4. membangkitkan *frequent* k-*itemset*,
5. jika jumlah *frequent* k-*itemset* > 0, nilai k bertambah 1 dan ulangi langkah 1; jika tidak maka selesai.



Gambar 3.4 (a) *Flowchart* proses algoritma *FP-growth*
(b) *Flowchart* proses 'Membangun Tree'

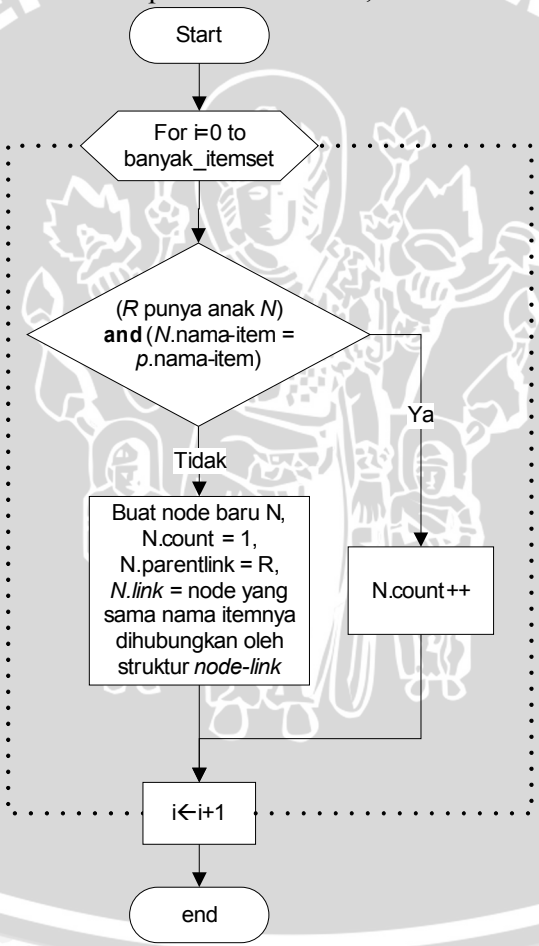
Sedangkan proses dari algoritma *FP-growth*, berdasarkan flowchart pada gambar 3.4(a), meliputi:

- sistem membentuk *FP-tree* dari dataset,
- setelah itu periksa path dari tree yang terbentuk,
- jika bukan *single path*, scan tree dari bawah ke atas untuk membangkitkan *conditional pattern* dari masing-masing item mulai dari item yang kecil nilai *minimal support*-nya. Dan bentuk tree dari masing-masing item,
- jika tree tersebut single path, *generate subset* dari tree tersebut,

- selesai.

Untuk proses 'Membangun Tree', seperti gambar 3.4 (b), adalah:

- sistem me-*scan* dataset, untuk membangkitkan item yang unik (*frequent* 1-*itemset*),
- Buat *root* dan beri nilai *null*
- pada setiap transaksi, urutkan frequent itemnya, disimbolkan dengan [p|P], dimana p elemen pertama dan P elemen yang tersisa,
- lalu masukkan tiap item kedalam tree secara urut (*insert_tree*([p|P], R)). Lakukan sampai transaksi habis,



Gambar 3.4 (c) *Flowchart* proses *InsertTree*

Untuk proses '*InsertTree*', seperti gambar 3.4 (c), adalah:

- Setiap item p pada itemset,
- periksa tree, jika root (R) punya anak (N) dan N.nama_item = p.nama_item, maka N.count++; jika tidak maka buat node baru N, beri nilai 1, parent link merujuk ke R dan node-link dihubungkan ke node dengan nama item yang sama
- Lakukan sampai item habis.
- selesai

3.3.2 Rancangan Desain Data

Struktur data utama pada sistem implementasi mining frequent itemset beserta operasi yang akan dilakukan pada masing-masing struktur, disusun dalam Tabel 3.1.

Tabel 3.1 Struktur Data Implementasi Sistem

No.	**Struktur Data**	
1	Jenis	Struktur data ***Tree***
	Nama	***FP-Tree***
	Deskripsi	untuk menyimpan informasi item serta *count* yang disusun berdasarkan *dataset* transaksi dengan urutan tertentu

	Deklarasi	`FPNode = ^FPTreeNode;`
		`FPTreeNode = record`
		`item : string;`
		`count : integer;`
		`parent, children, sibling, hlink: FPNode;`
		`end;`
		`_minsup, _depth : integer;`
		`_root : FPNode;`
		`_hasSinglePath : boolean;`
		`FrequencyItemCounter : Dictionary<int,int>;`
		`ht : HeaderTable;`

	Operasi	*Create, Addnode, getchild, getsibling, getparent*

2	Jenis	Struktur data *list*
	Nama	***Header_table***
	Deskripsi	list untuk menyimpan informasi item serta *count* yang memiliki nilai count lebih besar atau sama dengan *minimal support*

	Deklarasi	`THeader = ^DataHeader;`
		`DataHeader= record`
		`item : string;`
		`hlink : FPNode;`
		`end;`

	Operasi	*First, next, recordcount, gethlink*
3	Jenis	Struktur data *list*
	Nama	***ItemSet***
	Deskripsi	Untuk menyimpan kumpulan item

	Deklarasi	`ItemSet kumpulan_item = new ItemSet();`
		`item : List<int>;`
		`item_support : int;`

	Operasi	*Add, sort*
4	Jenis	Struktur data *list*
	Nama	***Transaction_table***
	Deskripsi	Untuk menyimpan informasi nomor transaksi, item serta *count.*

	Deklarasi	`List<Transaction>=new List<Transaction>();`
		`Id Transaction : int;`
		`itemset: ItemSet;`

	Operasi	*addItem, sort, getId, getItemset*

5	Jenis	Struktur data *dictionary<key, value>*
	Nama	***TIDList_table***
	Deskripsi	Tabel untuk menyimpan nomor transaksi dari setiap *frequent item*

	Deklarasi	`Dictionary<int, Transaction> TidList = new Dictionary<int, Transaction>();`
		`Item : int;`
		`no_transaksi : Transaksi;`

	Operasi	*Generate, add*
6	Jenis	Struktur data *dictionary<key, value>*
	Nama	***Frequent_Item***
	Deskripsi	Menyimpan koleksi *frequent_item* dan nilai *support*-nya.

	Deklarasi	`Dictionary<int, int> Frequent_Item = new Dictionary<int, int>();`
		`Item : int;`
		`Support : int;`

	Operasi	*Generate, add*
7	Jenis	Struktur data *list*
	Nama	***Candidate_ItemSet***
	Deskripsi	Untuk menyimpan kandidat itemset tiap level.
	Deklarasi	`List<ItemSet> Candidate_Itemset = new List<ItemSet>();` `itemset: ItemSet;`
	Operasi	*Add, clear*

3.3.3 Rancangan Uji Coba

Uji coba sistem untuk membandingkan algoritma dalam pencarian *frequent itemset* ini dibagi dalam dua langkah, yaitu uji coba kebenaran dan uji coba kinerja sistem. Uji coba kebenaran dilakukan untuk menguji validitas sistem aplikasi. Sedangkan uji

coba kinerja dilakukan untuk mengetahui kinerja aplikasi terhadap perubahan-perubahan data transaksi dari berbagai ukuran dan distribusi data (dataset).

Pada uji coba kebenaran, akan dibuktikan hasil yang didapat dari sistem aplikasi apakah sama dengan hasil yang dihitung secara manual. Dataset Contoh seperti yang ditunjukkan pada tabel 3.2 digunakan untuk menguji validitas sistem aplikasi tersebut. Secara manual dan dengan menggunakan sistem aplikasi, akan dicari *frequent itemset* dari dataset tersebut dengan menggunakan algoritma *CBW* dan *FP-growth.* Selanjutnya *frequent itemset* yang terbentuk dan *support* tiap *itemset*-nya harus sama antara hasil perhitungan manual dan hasil dari sistem aplikasi. Bila hasil dari sistem aplikasi tidak sama dengan hasil perhitungan manual, maka kemungkinan besar ada yang kurang atau salah pada pemrograman sistem aplikasinya, dan perlu dilakukan perbaikan dan pengujian ulang sistem.

Tabel 3.2 *Dataset* Contoh

Tid	items
1	002,003,004,005,006
2	001,003
3	002,007
4	001,002,003,004
5	001,002,004
6	003,004,005
7	001,002,004,005
8	002,003,004,008
9	002,003,004,005,006
10	002,003,004,005

Minimal support : 30 %

Karena terdiri dari 10 transaksi, jumlah transaksi yang memenuhi *minsupp* adalah 3 (10 X 30% = 3).

Pada uji coba kinerja, akan dilakukan beberapa kali uji coba terhadap kedua algoritma, dengan menggunakan beberapa dataset yang berbeda, agar dapat diketahui apakah ada perbedaan pada data yang dihasilkan. Selanjutnya dianalisa hasil uji coba kinerja aplikasi

tersebut dalam penghitungan waktu komputasi proses pencarian *frequent itemset* pada berbagai tingkatan nilai *minimal support,* dan jumlah transaksi.

Langkah-langkah yang dilakukan adalah:

1. Dataset dibagi menjadi 2 bagian.
   Untuk analisa waktu pembangkitan *frequent itemset* terhadap penurunan nilai *minimal support*, digunakan data pada tabel 3.3. Sedangkan untuk analisa waktu pembangkitan *frequent itemset* terhadap jumlah transaksi, digunakan data pada tabel 3.4.
2. Seluruh dataset diujikan pada masing-masing algoritma.
3. Setiap proses disimpan hasil yang didapat, yaitu *frequent itemset* dan waktu prosesnya.

Tabel 3.3 *Dataset* untuk analisis waktu pembangkitan *frequent itemset* terhadap penurunan nilai *minimal support*.

Dataset	**Panjang rata-rata item per transaksi**	**Rata-rata *maximal frequent itemset***	**Σ Transaksi**	**Σ Item**
T10I4D100K	10	4	100.000	1000
T25I20D100K	25	20	100.000	10000
pumsb	74	-	49.046	7117
mushroom	23	-	8.124	119
connect4	37	-	67.557	129

Tabel 3.4 *Dataset* untuk analisis waktu pembangkitan *frequent itemset* terhadap jumlah transaksi

Dataset	**Σ Record**	**Σ Transaksi**	**Rata-rata item per transaksi**	**Σ Item**
T3I4D10K	30.100	6.878	4	50
T3I4D20K	60.346	13.824	4	50
T3I4D30K	90.596	20.742	4	50
T3I4D40K	120.968	27.686	4	50
T3I4D50K	150.604	34.582	4	50
T3I4D60K	180.904	41.480	4	50
T3I4D70K	210.311	48.293	4	50
T3I4D80K	239.914	55.065	4	50
T3I4D90K	269.968	61.993	4	50
T3I4D100K	300.198	68.863	4	50

3.4 Penerapan Algoritma *CBW*

Berikut ini penelusuran algoritma *CBW* untuk data pada tabel 3.2 diatas. Sesuai dengan *flowchart* pada gambar 3.2, berikut ini urutan kerja algoritma CBW :

1. Menentukan *frequent* 1-*itemset* dari data transaksi :
   Dengan melakukan pembacaan basis data 1 kali, didapat *frequent* 1-*itemset* atau F_1 seperti pada tabel 3.5.

Tabel 3.5 *Frequent* 1-itemset (F_1)

Item	*Support*
001	4
002	8
003	7
004	8
005	5

2. Menjalankan prosedur *Trans*
   Yang pertama dilakukan adalah membangkitkan kandidat 2-*itemset* dari F_1, seperti pada tabel 3.6.

Dari tabel 3.6, yang dimaksud Tid adalah nomor identitas (ID) / nomor urut transaksi dataset pada tabel 3.2. NumF adalah jumlah *frequent item* pada tiap-tiap transaksi. Yang dimaksud dengan membangkitkan kandidat 2-*itemset* adalah mencari kombinasi item yang ada pada tabel 3.5 di tiap transaksi, misalnya mencari kombinasi item 002 dan 003 apakah ada di transaksi 1. Jika ada, maka diberi tanda dengan angka 1 pada tempat yang telah disediakan. Selanjutnya dihitung total tiap kombinasi item pada dataset tersebut, seperti yang ditunjukkan tabel 3.7.

Tabel 3.6 Pengulangan proses pada prosedur *Trans*

Tid	Trimed Itemset	NumF	Kandidat 2-Itemset									
			001, 002	001, 003	001, 004	001, 005	002, 003	002, 004	002, 005	003, 004	003, 005	004, 005
1	002,003,004,005	4					1	1	1	1	1	1
2	001,003	2		1								
3	002	1										
4	001,002,003,004	4	1	1	1		1	1		1		
5	001,002,004	3	1		1			1				
6	003,004,005	3								1	1	1
7	001,002,004,005	4	1		1	1		1	1			1
8	002,003,004	3					1	1		1		
9	002,003,004,005	4					1	1	1	1	1	1
10	002,003,004,005	4					1	1	1	1	1	1

Tabel 3.7 Kandidat dan *Frequent Itemset* yang dihasilkan prosedur *Trans*

Itemset	*Support*
001,002	3
001,003	2
001,004	3
001,005	1
002,003	5
002,004	7
002,005	4
003,004	6
003,005	4
004,005	5

→

Itemset	*Support*
001,002	3
001,004	3
002,003	5
002,004	7
002,005	4
003,004	6
003,005	4
004,005	5

Untuk setiap *frequent item*, *TID*-nya ditambahkan ke *TIDlist* jika jumlah dari *t* tidak kurang dari 3, seperti pada tabel 3.8. Intinya, untuk semua transaksi yang jumlahnya lebih besar dari α harus disimpan, sebab *UpSearch* dimulai dari level α+1. Kardinalitas (nilai) 3 digunakan karena akan didapatkan hasil yang terbaik untuk memfasilitasi proses *tidlist pruning*.

Tabel 3.8 *TIDList* yang dihasilkan prosedur *Trans*

Item	TID
001	4,5,7
002	1,4,5,7,8,9,10
003	1,4,6,8,9,10
004	1,4,5,6,7,9,10
005	1,7,9,10

Proses terakhir dari prosedur *Trans* adalah menghitung nilai *cutting level* (α), sesuai persamaan 2.3, yaitu:

$\alpha = numf / \left| D \right| = 32 / 10 = 3{,}2 \approx 3$

3. Menjalankan prosedur *DwnSearch*

   Yaitu melakukan enumerasi untuk membangkitkan kandidat *itemset* dengan kardinalitas antara α dan 3, dan menghitung nilai *support*-nya. Karena nilai α adalah 3, maka hanya sekali membangkitkan kandidat 3-*itemset* , yang prosesnya seperti pada tabel 3.9.

Tabel 3.9 Pengulangan proses pada prosedur *DwnSearch*

Tid	Trimed Itemset	Candidate 3-Itemset				
		001,002,004	002,003,004	002,003,005	002,004,005	003,004,005
1	002,003,004,005		1	1	1	1
2	001,003					
3	002					
4	001,002,003,004	1	1			
5	001,002,004	1				
6	003,004,005					1
7	001,002,004,005	1			1	
8	002,003,004		1			
9	002,003,004,005		1	1	1	1
10	002,003,004,005		1	1	1	1

Selanjutnya algoritma mencari itemset yang nilai *support*-nya lebih dari *minimal support*, dan didapat hasil pada tabel 3.10.

Tabel 3.10 *Frequent Itemset* yang dihasilkan prosedur *DwnSearch*

Itemset	*Support*
001,002,004	3
002,003,004	5
002,003,005	3
002,004,005	4
003,004,005	4

4. Menjalankan prosedur *UpSearch*
   Mencari kandidat α+1, didapat: 002,003,004,005. Selanjutnya dicari *bit-vector intersection*-nya untuk menghitung *support*, sebagaimana tabel 3.11.

Tabel 3.11 Interseksi tabel *Bit map*

Item	1	2	3	4	5	6	7	8	9	10
001	0	0	0	0	0	0	0	0	0	0
002	1	0	0	0	0	0	0	0	1	1
003	1	0	0	0	0	0	0	0	1	1
004	1	0	0	0	0	0	0	0	1	1
005	1	0	0	0	0	0	0	0	1	1

Hasilnya pada tabel 3.12.

Tabel 3.12 Kandidat Itemset yang dihasilkan prosedur *Upsearch*

Itemset	***TIDList Intersection***	***Support***
002,003,004,005	1,9,10	3

Hasil akhir dari algoritma CBW terhadap Dataset Contoh, yaitu didapatnya 19 *frequent itemset* seperti pada tabel 3.13.

Tabel 3.13 *Frequent Itemset* hasil algoritma *CBW*

Itemset	***Support***
001	4
002	8
003	7
004	8
005	5
001,002	3
001,004	3
002,003	5
002,004	7
002,005	4
003,004	6
003,005	4
004,005	5
001,002,004	3
002,003,004	5
002,003,005	3
002,004,005	4
003,004,005	4
002,003,004,0005	3

3.5 Penerapan Algoritma *FP-growth*

Berikut ini urutan kerja algoritma *FP-growth*, yang dilakukan pada dataset contoh di tabel 3.2 pada halaman 28:

1. Telusuri Dataset untuk menemukan *frequent items*:
   Setelah didapat, diurutkan berdasarkan jumlah nilai *support*-nya, seperti tabel 3.14.
   Tabel 3.14 *Frequent* 1-itemset

Item	*Support*
002	8
004	8
003	7
005	5
001	4

   Berdasarkan tabel 3.14 item-item pada tabel transaksi juga diurutkan, seperti tabel 3.15
   Tabel 3.15 Dataset Transaksi

Tid	items	(*Ordered*) *frequent items*
1	002,003,004,005,006	002,004,003,005
2	001,003	003,001
3	002,007	002
4	001,002,003,004	002,004,003,001
5	001,002,004	002,004,001
6	003,004,005	004,003,005
7	001,002,004,005	002,004,005,001
8	002,003,004,008	002,004,003
9	002,003,004,005,006	002,004,003,005
10	002,003,004,005	002,004,003,005

2. Telusuri Dataset kembali untuk pembentukan *FP-tree*:
   - Buat *root* beri nilai *null* dan buat *header table*, seperti gambar 3.5.

Header table

Item	Head of node link
002	Null
004	Null
003	Null
005	Null
001	Null

Gambar 3.5 *FP-tree* awal. Sumber: Perancangan.

- Pembacaan transaksi Tid (1)

- Pada gambar 3.6 algoritma menjalankan prosedur *insert_tree*([002|004,003,005], R)

Header table

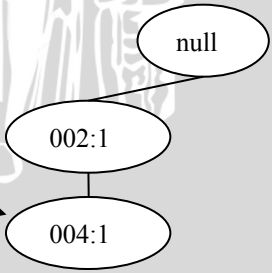
Item	Head of node link
002	
004	Null
003	Null
005	Null
001	Null

Gambar 3.6 *FP-tree* setelah dijalankan prosedur *insert_tree*([002| 004,003,005], R). Sumber: Perancangan.

- Menjalankan prosedur *insert_tree*([004|003,005], 002), seperti gambar 3.7.

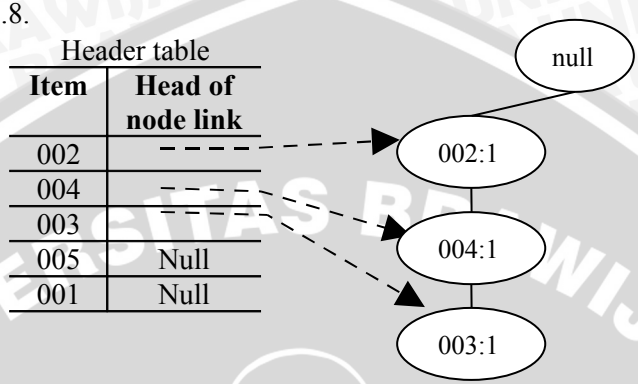
Header table

Item	Head of node link
002	
004	
003	Null
005	Null
001	Null



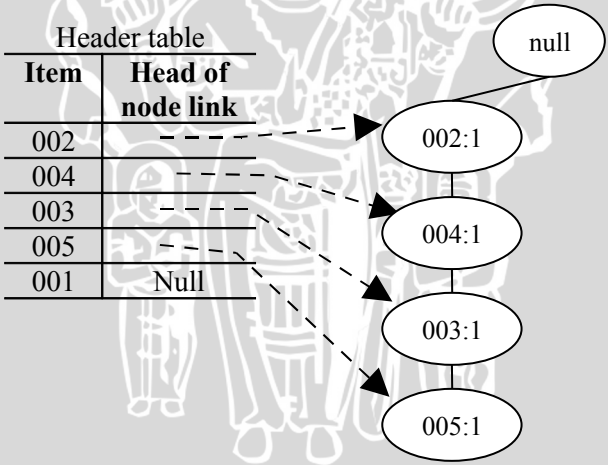
Gambar 3.7 *FP-tree* setelah dijalankan prosedur *insert_tree*([004| 003,005], 002). Sumber: Perancangan.

- Menjalankan prosedur *insert_tree*([003|005], 004), seperti gambar 3.8.



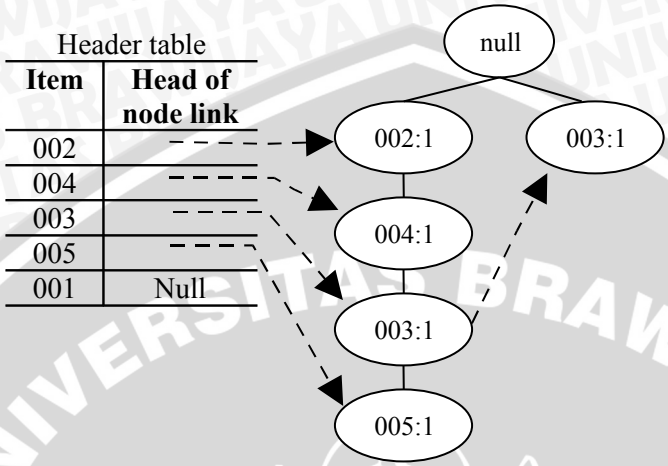
Gambar 3.8 *FP-tree* setelah dijalankan prosedur *insert_tree*([003|005], 004). Sumber: Perancangan.

- Menjalankan prosedur *insert_tree*(005, 003), seperti gambar 3.9.



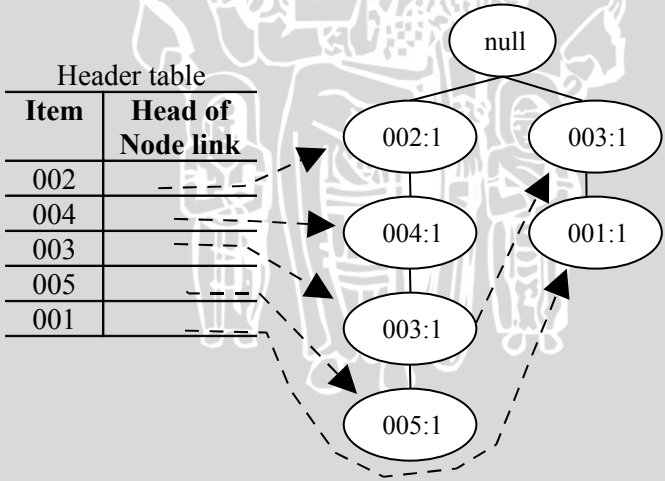
Gambar 3.9 *FP-tree* setelah dijalankan prosedur *insert_tree*(005, 003). Sumber: Perancangan.

- Pembacaan transaksi Tid (2)
- Menjalankan prosedur *insert_tree*([003|001], R), seperti gambar 3.10.



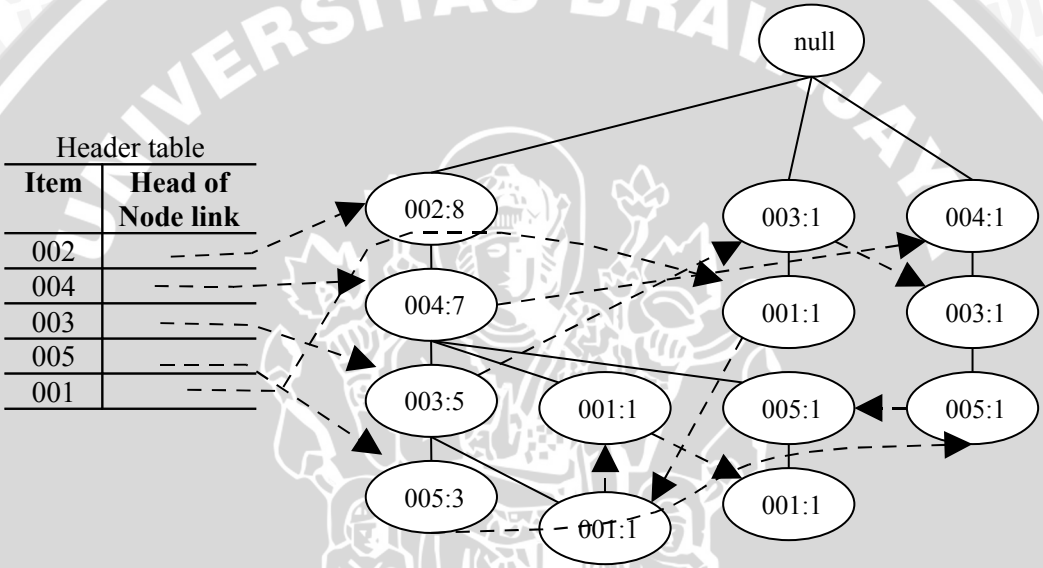
Gambar 3.10 *FP-tree* setelah dijalankan prosedur *insert_tree*([003|001], R). Sumber: Perancangan.

- Menjalankan prosedur *insert_tree*(001, 003), seperti gambar 3.11.



Gambar 3.11 *FP-tree* setelah dijalankan prosedur *insert_tree*(001, 003). Sumber: Perancangan.

- Menjalankan prosedur selanjutnya, sampai terbentuk *FP-tree* yang lengkap seperti gambar 3.12.



Gambar 3.12 *FP-tree* lengkap
Sumber: Perancangan.

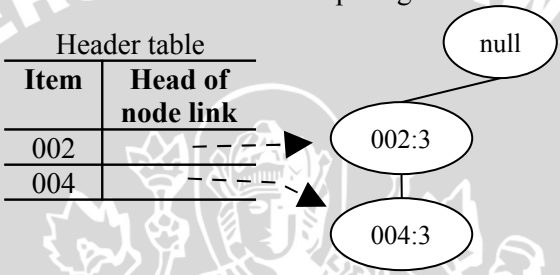
3. Penambangan *frequent itemset* menggunakan algoritma *FP-growth.*

   Proses penambangan ini berdasarkan *FP-tree* lengkap yang telah dibangun. Karena bukan terdiri dari *single path*, maka masing-masing item pada *header* tabel dicari *conditional pattern base*-nya dan dibentuk *Conditional FP-tree*-nya.

   - 001 : 4 (*suffix pattern*)

     Conditional pattern base : {(003:1), (002:1, 004:1, 003:1), (002:1, 004:1), (002:1, 004:1, 005:1)}

     Terbentuk *Conditional FP-tree* seperti gambar 3.13



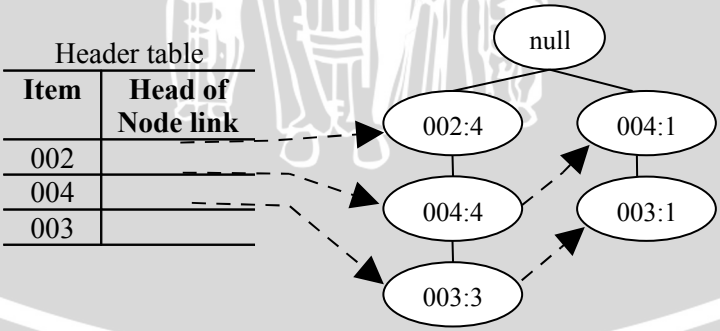
Gambar 3.13 *Conditional FP-tree* untuk 001.
Sumber: Perancangan.

Didapat *frequent itemset*: (002,001), (004,001), dan (002,004,001) masing-masing sebanyak 3.

- 005 : 5 (*suffix pattern*)

  Conditional pattern base : { (002:3, 004:3, 003:3), (004:1, 003:1), (002:1, 004:1) }

  Terbentuk *Conditional FP-tree* : seperti pada gambar 3.14.

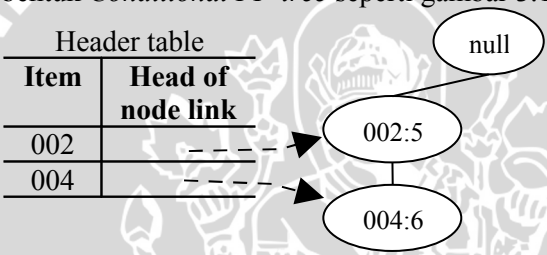


Gambar 3.14 *Conditional FP-tree* untuk 005.
Sumber: Perancangan.

Karena bukan terdiri dari *single path*, maka masing-masing item pada *header* tabel dicari *conditional pattern base*-nya. Hasilnya didapat *frequent itemset* :

- 002,005 : 4
- 004,005 : 5
- 003,005 : 4
- 002,004,005 : 4
- 004,003,005 : 4
- 002,004,003 : 3

- 003 : 7 (*suffix pattern*)
  Conditional pattern base : {(002:5, 004:5), (004:1)}
  Terbentuk *Conditional FP-tree* seperti gambar 3.15



Gambar 3.15 *Conditional FP-tree* untuk 003.
Sumber: Perancangan.

Didapat *frequent itemset* :

- 002,003 : 5
- 004,003 : 6
- 002,004,003 : 5

- 004 : 8 (*suffix pattern*)
  Conditional pattern base : {(002:7)}
  Terbentuk *Conditional FP-tree* seperti gambar 3.16

Gambar 3.16 *Conditional FP-tree* untuk 004.
Sumber: Perancangan.

Dari gambar 3.16 terlihat bahwa *conditional FP-tree* untuk 004 memiliki jalur tunggal.
Didapat *frequent itemset* : {(002,004 : 7)}

- 002 : 8 (*suffix pattern*)
  Conditional pattern base : {.}
  Conditional FP-tree : {.}
  Tidak didapat *frequent itemset*.

Secara keseluruhan, untuk pencarian dengan menggunakan algoritma *FP-growth*, didapat 19 *frequent itemset* seperti ditunjukkan pada tabel 3.16.

Tabel 3.16 *Frequent Itemset* hasil algoritma *FP-growth*

Itemset	***Support***
001	4
002	8
003	7
004	8
005	5
001,002	3
001,004	3
002,003	5
002,004	7
002,005	4
003,004	6
003,005	4
004,005	5
001,002,004	3
002,003,004	5
002,003,005	3
002,004,005	4
003,004,005	4
002,003,004,0005	3

BAB IV
HASIL DAN PEMBAHASAN

4.1 Analisis Algoritma

Berdasarkan penjelasan pada bab 2 sub-bab 2.4, diketahui bahwa pembentukan FP-Tree memerlukan 2 kali pembacaan basis data transaksi: penulusuran pertama untuk mendapatkan *frequent* 1-*itemset* dan penelusuran kedua untuk membangun FP-tree. Proses pembentukan *frequent* 1-*itemset* membutuhkan waktu sebesar O(nm), dimana n adalah jumlah transaksi dan m adalah rata-rata panjang transaksi. Sedangkan proses memasukkan tiap transaksi ke struktur FP-tree waktu yang dibutuhkan sebesar O(|freq(Trans)|), dimana freq(Trans) adalah himpunan *frequent item* di tiap transaksi. Waktu komputasi yang dibutuhkan algoritma FP-growth bergantung kepada FP-tree yang dibangun. Akan didapat waktu minimum (*best-case*) bila database transaksi terdiri dari himpunan item yang sama pada tiap transaksinya, jadi FP-tree yang dihasilkan hanya terdiri dari 1 *path* saja. Waktu maksimum atau *worst-case* akan terjadi bila setiap transaksi di database mengandung atau diawali himpunan item yang unik, akibatnya jumlah *path* dalam FP-tree adalah sebanyak jumlah transaksi. Hal ini dikarenakan harus membangkitkan banyak sub-permasalahan, yaitu membangkitkan *conditional pattern* tiap *frequent itemset*-nya.

Pada algoritma CBW, proses pembentukan *frequent* 1-*itemset* membutuhkan waktu sebesar O(nm), dimana n adalah jumlah transaksi dan m adalah rata-rata panjang transaksi. Untuk membangkitkan kandidat k+1, digabungkanlah tiap anggota dari *frequent* k-*itemset.* Setiap operasi penggabungan membutuhkan paling tidak k-buah perbandingan sebesar $O(\sum_k |F_k| \times |F_k|)$, dimana $|F_k|$ adalah *frequent* k-*itemset*. Waktu minimum (*best-case*) algoritma CBW didapat paling tidak melakukan 3 kali pembacaan basis data transaksi, dan 3 kali pembangkitan kandidat *itemset*. Waktu maksimum atau *worst-case* akan terjadi bila kandidat *longest itemset* (misal, *x*-itemset) berada diatas nilai *cutting level*, jadi algoritma CBW akan membangkitkan kandidat sebanyak *x*.

Dari kedua metode tersebut, kedua algoritma waktu komputasinya bergantung jumlah item dan jumlah transaksi. Pada dataset yang perbandingan jumlah item dan jumlah transaksinya sangat jauh, maka FP-Growth unggul.

4.2 Deskripsi Sistem

Sistem yang akan dikembangkan berupa sistem yang akan mengimplementasikan dua algoritma untuk pencarian *frequent itemset*, yaitu algoritma *CBW* dan *FP-growth*. Pencarian f*requent itemset* bertujuan untuk menemukan asosiasi dan hubungan antar item dalam kumpulan data transaksi atau dataset yang lain dalam jumlah yang besar. Dalam sistem ini digunakan beragam dataset yang memiliki beberapa parameter yang berbeda-beda, dalam membandingkan *frequent itemset* yang dihasilkan oleh kedua algoritma tersebut. Parameter yang digunakan adalah jumlah transaksi, jumlah item, dan rata-rata jumlah item pada tiap transaksi.

Setelah menentukan data transaksi yang akan digunakan dan menentukan nilai *minimal support*-nya, sistem akan men-*scan* data di database satu kali untuk mendapatkan *frequent 1-itemset*. *Frequent 1-itemset* adalah kumpulan item satuan yang sering muncul. Pencarian *frequent 1-itemset* sama-sama dilakukan oleh kedua algoritma pada awal proses pencarian *frequent itemset*. Selanjutnya sistem mencari *frequent itemset* dengan menggunakan algoritma *CBW* dan *FP-growth*.

Output yang diharapkan dari sistem ini adalah berupa *frequent itemset* yang dihasilkan oleh kedua algoritma dan informasi lamanya proses pencarian *frequent itemset*. Proses penghitungan lamanya waktu proses akan dimulai pada saat proses algoritma dijalankan sampai ditemukannya seluruh *frequent itemset*.

4.3 Implementasi Fungsi

Implementasi fungsi selengkapnya dapat dilihat pada Lampiran A, tapi secara garis besar fungsi-fungsi utama yang diimplementasikan adalah sebagai berikut :

1. *FP-Growth* (*ExtractFrequentPattern*)
   Sourcecode 4.1 adalah s*ourcecode* dari proses utama algoritma *FP-Growth* untuk membangkitkan *frequent itemset* per level berdasarkan nilai *minimal support*.

```
...
StartFPtree.SetMinSup(MinSup);
StartFPtree.BuildTree(wholeTransToItemSet);
FPTreeList.Add(StartFPtree);
GivenListNow.Add(new ItemSet());
//Looping on each fptree until there are ones to
process
while (FPTreeList.Count > 0)
{ for (int j = 0; j < FPTreeList.Count; j++)
  { Next = FPTreeList[j];
    Given = GivenListNow[j];
    if (!Next.isEmpty())
    { if (Next.HasSinglePath)
        GenerateCombPattern(Next, Given);
      else{
        // Header table sorting
        Next.SortHeaderTable();
        for (int i = 0; i < Next.GetHTSize();i++)
        {
        ...
       // Here we generate the so called
       //conditional FPTree using a projection
       // of the original database called
       //Conditional Pattern Base
       FPTreeListNext.Add(Next.CreateFPtree(i));
                    ...
        }
      }...
    }...
 }
...
```

Sourcecode 4.1. Sumber: codeplex

2. *GenerateCombPattern*

Sourcecode 4.2 adalah *sourcecode* dari fungsi *GenerateCombPattern* untuk mengkombinasikan semua pattern yang ada dalam tree.

```
...
while (enumerator < max)
{
// Create a new beta result
ItemSet beta = new ItemSet();
betaSupport = 0;
foreach (int item in given.Items)
  beta.Add(item);
index = 0;
combination = enumerator;
while (combination > 0)
{ if ((combination % 2) == 1)
  { beta.Add(itemsetArray[index]);
    if ((betaSupport > supportArray[index]) ||
(betaSupport == 0))
    { betaSupport = supportArray[index];}
   }
   combination = combination >> 1;
   index++;
  }
  enumerator++;
  beta.ItemsSupport = betaSupport;
  result.Add(beta);
}
```

Sourcecode 4.2. Sumber: codeplex

3. *BuildTree*

Sourcecode 4.3 adalah *sourcecode* dari fungsi *BuildTree* untuk membangun *tree* dari dataset yang ada.

```
// Counting the frequency of single items
FrequencyItemCounter=newDictionary<int,int>();
FrequencyItemCounter =
  CountFrequencyItem(startdb);
// Evaluate header table dimension
int HTsize = 0;
foreach (int item in FrequencyItemCounter.Keys)
{if (FrequencyItemCounter[item] >= _minSup)
    HTsize++;
}
ht = new HeaderTable(HTsize);
// Add every frequent single itemset to header
table
foreach (KeyValuePair<int, int> coppia in
FrequencyItemCounter)
{ if (coppia.Value >= _minSup)
    ht.addRecord(coppia.Key, coppia.Value);
}
// create complete FPTree
// Removal of non frequent items, sorting and
final insertion in the FPTree
ItemSet SortedList = new ItemSet();
foreach (ItemSet itemset in startdb)
{ SortedList.Items.Clear();
  for(int i = 0;i < itemset.ItemsNumber; i++)
  {if (FrequencyItemCounter[itemset.Items[i]]
>= _minSup)
   { SortedList.Add(itemset.Items[i]);}
  }
  if (SortedList.ItemsNumber > 0)
  { SortedList.Items.Sort(new
ItemSortingStrategy(FrequencyItemCounter));
    if (_depth < SortedList.ItemsNumber) _depth =
SortedList.ItemsNumber;
      insert_tree(SortedList, Root);
  }
}
startdb = null;
```

Sourcecode 4.3. Sumber: codeplex

4. *insert_tree*

Sourcecode 4.4 adalah s*ourcecode* dari fungsi *insert_tree* memasukkan item sebagai node pada *tree* yang ada.

```
for (int i = 0; i < SortedList.ItemsNumber; i++)
{
...
// If founded node is present in the tree,
// increase it's support and move on to the next
// item updating me reference
if (found != default(FPNode))
{ found.Count += SortedList.ItemsSupport;
  me = found;
}
// Create a new FPNode and set header table head //
and tail corresponding list
else
{ if (me.Children.Length > 0)
    _hasSinglePath = false;
  ...
  newnode.Count = SortedList.ItemsSupport;
  // Temporary Array where to store the new list
  // of child nodes
  FPNode[] tmpArray = new
    FPNode[me.Children.Length + 1];
  for (int k = 0; k < me.Children.Length; k++)
    tmpArray[k] = me.Children[k];
  tmpArray[me.Children.Length] = newnode;
  me.Children = tmpArray;
  newnode.Parent = me;
  // Me reference set to the new node
  me = newnode;
}
}
```

Sourcecode 4.4. Sumber: codeplex

5. *CBWProcess* (*ExtractFrequentPattern*)
   Sourcecode 4.5 adalah s*ourcecode* dari proses utama algoritma CBW untuk membangkitkan *frequent itemset* per level berdasarkan nilai *minimal support*.

```
...
// Compute support for 1-itemset and store in
//FrequentItems dictionary
foreach (Transaction trans in AllTrans)
{ foreach (int product in trans.Itemset.Items)
  { if (FrequentItems.ContainsKey(product))
      FrequentItems[product]++;
    else
      FrequentItems.Add(product, 1);
  }
}
...
Trans(AllTrans, ref TIDList, FrequentItemSets,
   ref alpha, MinSup);
DwnSearch(AllTrans, FrequentItemSets, alpha,
   MinSup);
UpSearch(TIDList,FrequentItemSets, alpha,
   MinSup);
...
```

Sourcecode 4.5. Sumber: Perancangan

6. *Trans*

Sourcecode 4.6 adalah implementasi prosedur *Trans* dari algoritma CBW untuk menentukan nilai α, *TIDList* dan *frequent* 2-*itemset*.

```
...
CandidateItemSets=CandidateGen(FrequentItemSets,
   2);
int numf = 0;
foreach (Transaction trans in AllTrans)
{ // Remove unfrequent item from transactions
  //list
  Transaction ReducedTrans = new Transaction();
  ...
  numf= numf + ReducedTrans.Itemset.ItemsNumber;
  if (ReducedTrans.Itemset.ItemsNumber >= 3)
  { //Insert into TID List
    foreach(int i in ReducedTrans.Itemset.Items)
    { if (TIDList.ContainsKey(i))
        TIDList[i].addItem(trans.Id);
      else
        {Transaction IDList = new Transaction();
         IDList.addItem(trans.Id);
         TIDList.Add(i, IDList);
         }
     }
   }
foreach(ItemSet CanItemSet in CandidateItemSets)
{ if (CanItemSet < trans.Itemset)
    CanItemSet.ItemsSupport++;
}
AllTrans = FlaggedDB;
FrequentItemSets.Clear();
foreach (ItemSet itemset in CandidateItemSets)
{ if (itemset.ItemsSupport >= MinSup)
  { FrequentItemSets.Add(itemset, true);
    Result.Add(itemset);
   }
}
int lengthTrans = AllTrans.Count;
alpha = (int)(numf / lengthTrans);
```

Sourcecode 4.6. Sumber: Perancangan

7. *DwnSearch*

Sourcecode 4.7 adalah implementasi prosedur *DwnSearch* dari algoritma CBW untuk menentukan nilai *frequent itemset* yang levelnya antara 3 sampai α.

```
...
for (int d = 3; d <= alpha; d++)
{ CandidateItemSets =
    CandidateGen(FrequentItemSets, d);
  FrequentItemSets.Clear();
  if (CandidateItemSets.Count <= 0)
    d = alpha + 1;
  foreach (ItemSet s in CandidateItemSets)
    g.Add(s);
}
foreach (Transaction trans in AllTrans)
{
// Remove unfrequent item from transactions list
Transaction ReducedTrans = new Transaction();
if (trans.Itemset.ItemsNumber > 2)
{// Evalute support for the generated frequent
  //candidate
 foreach (ItemSet CanItemSet in g)
 { if (CanItemSet < trans.Itemset)
     CanItemSet.ItemsSupport++;
 }
}
}
foreach (ItemSet e in g)
{if (e.ItemsSupport >= MinSup)
 {//FrequentItemSets.Add(e, true);
  Result.Add(e);
 }
}
```

Sourcecode 4.7. Sumber: Perancangan

8. *UpSearch*

Sourcecode 4.8 adalah implementasi prosedur *UpSearch* dari algoritma CBW untuk menentukan nilai *frequent itemset* yang berada diatas α dengan menggunakan *vertical intersection*.

```
if (FrequentItemSets == null) return;
...
foreach (int item in TIDList.Keys)
{ if (FrequentItems.ContainsKey(item))
    TList.Add(item, TIDList[item]);
}
TIDList = TList;
...
while (IsContinued)
{ k = k + 1;
  //Generate candidat k-itemsets from F(k-1) or
  //F(alpha)
  CandidateItemSets =
    CandidateGen(FrequentItemSets, k);
  FrequentItemSets.Clear();
  foreach (ItemSet x in CandidateItemSets)
  { //bit vector intersection
    ...
    //compute the support of x
    if (d.Count == 1)
      x.ItemsSupport = d[0].ItemsNumber;
    if (x.ItemsSupport >= MinSup)
    { FrequentItemSets.Add(x, true);
      Result.Add(x);
    }
  }
  if (FrequentItemSets.Count <= 0)
   IsContinued = false;
  else
    IsContinued = true;
}
```

Sourcecode 4.8. Sumber: Perancangan

9. *CandidateGen*

Sourcecode 4.9 adalah s*ourcecode* dari Prosedur *CandidateGen* untuk membangkitkan kandidat *itemset* pada satu level tertentu berdasarkan *frequent itemset* yang ditemukan sebelumnya.

```
...
foreach(ItemSet itemset in
  FrequentItemSets.Keys)
{
foreach (int frequentitem in FrequentItems.Keys)
{ if (itemset.Items[itemset.ItemsNumber - 1] <
    frequentitem)
  { NewCandidate = new ItemSet();
    NewCandidate.ItemsSupport = 0;
    foreach (int item in itemset.Items)
      NewCandidate.Add(item);
    NewCandidate.Add(frequentitem);
    //Pruning, based on anti-monotonicity
    //itemset principle (see paper)
    ok = 0;
    for (int i = 0; i < k; i++)
   {ItemSet test = new ItemSet();
    for (int j = 0; j < k; j++)
     {if (j != i)
       test.Add(NewCandidate.Items[j]);
     }
    if (FrequentItemSets.ContainsKey(test))
      ok++;
    else i = k;
   }
   if (ok == k)
     NewCandidateSet.Add(NewCandidate);
  }
  }
}
return NewCandidateSet;
```

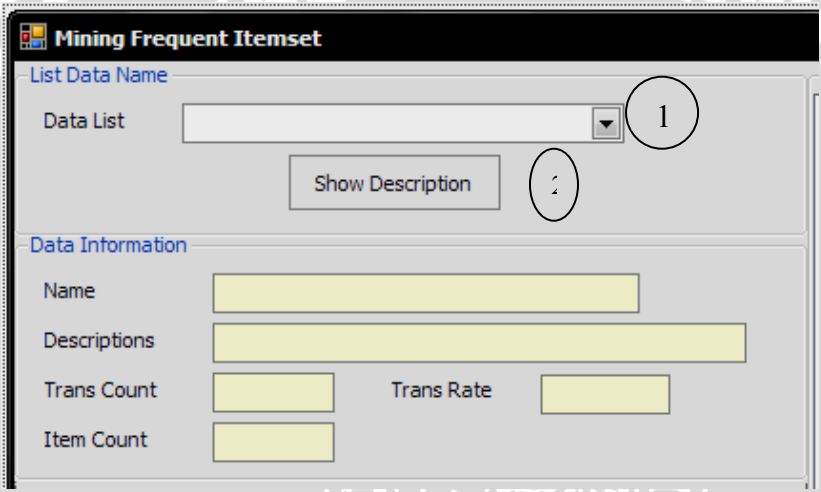
Sourcecode 4.9. Sumber: Perancangan

4.4 Implementasi Antarmuka

Berdasarkan rancangan pada bab 3, maka dibuatlah antarmuka untuk proses mining atau *Mining Frequent Itemset Form*. Antarmuka selengkapnya pada lampiran B.

4.4.1 Antarmuka Pemilihan Dataset

Pada *Mining Frequent Itemset Form* terdapat bagian untuk memilih data yang akan digunakan dan melihat deskripsi dari data terbebut seperti ditunjukkan pada gambar 4.1.

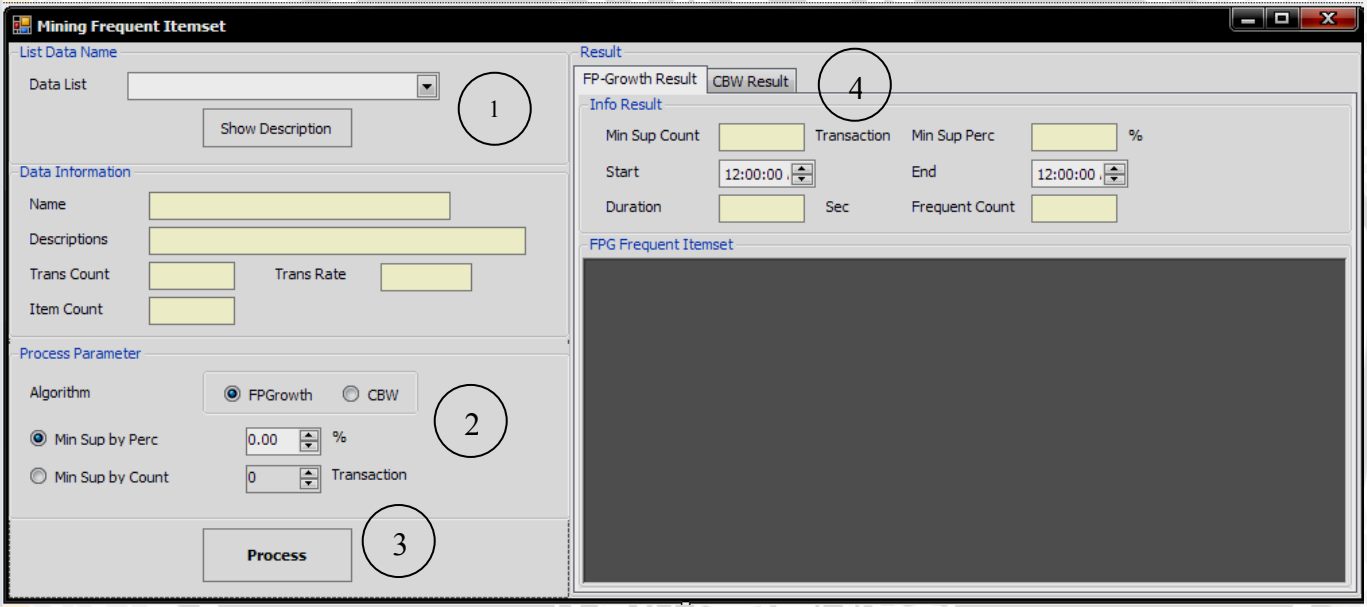


Gambar 4.2 *Select Data*

Pada Gambar 4.1 terdapat beberapa tombol dan pilihan menu.

1. *Combo Box Data List*, untuk menentukan dataset yang akan digunakan.
2. Tombol *Show Description*, untuk menampilkan deskripsi dari data pada *grup box Data Information,* diantaranya: *Name* (nama dataset), *Description* (deskripsi singkat dataset), *Trans Count* (jumlah transaksi), *Item Count* (jumlah item), dan *Trans Rate* (rata-rata jumlah item tiap transaksi).

4.4.2 Antarmuka Proses Mining



Gambar 4.2 Form *Mining Frequent Itemset*

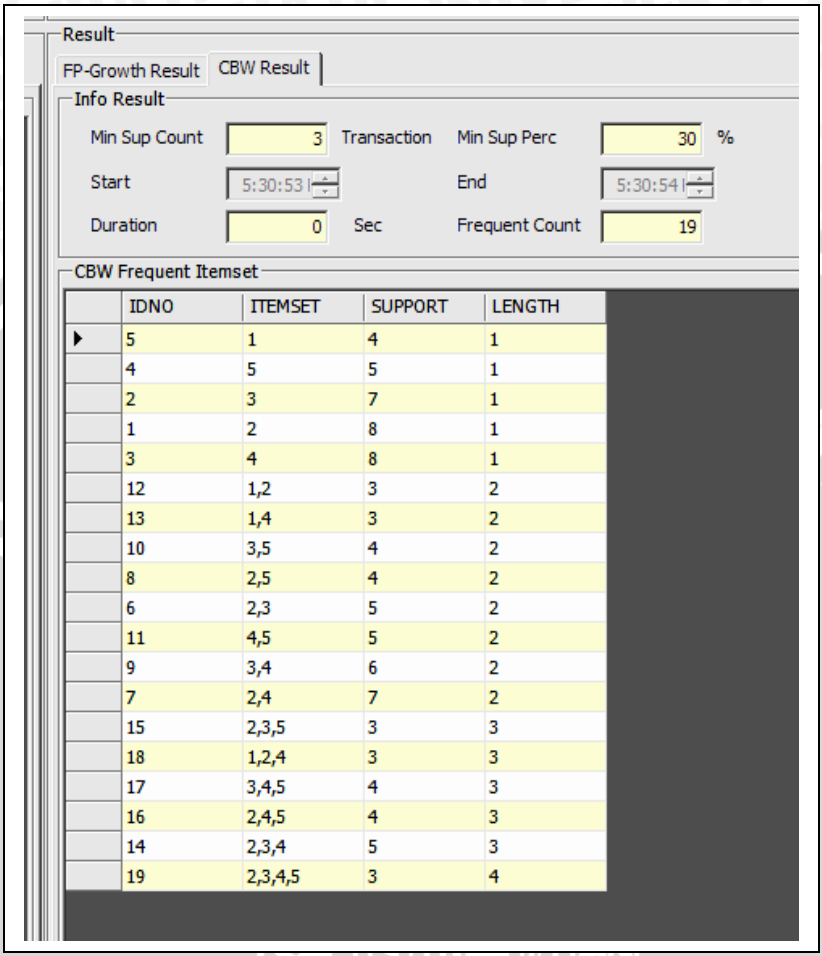
Gambar 4.2 menunjukkan form *Mining Frequent Itemset* secara keseluruhan. Pada antarmuka *Mining Frequent Itemset* tersebut terdapat beberapa bagian,diantaranya.

1. Bagian *Select Data* seperti yang dijelaskan pada sub bab 4.4.1.
2. *Grup box Process Parameter*, yang terdiri dari: *radio button Algorithm*(untuk memilih algoritma yang digunakan untuk mengeksekusi data), dan *radio button Minimal Support* (untuk memilih *minimal support* menggunakan persentase atau bilangan bulat dan sekaligus mengisikan jumlahnya).
3. Tombol Proses(untuk memulai proses *mining*).
4. *Grup box Result,* terdiri dari tab untuk hasi dari proses dengan algoritma *FP-growth* dan tab untuk hasil proses algoritma *CBW*. Keduanya menampilkan hasil dari proses *mining*, berupa: *frequent itemset* yang didapat, waktu awal dan akhir proses, durasi proses serta jumlah *frequent itemset*-nya

4.5 Pengujian Kebenaran Sistem

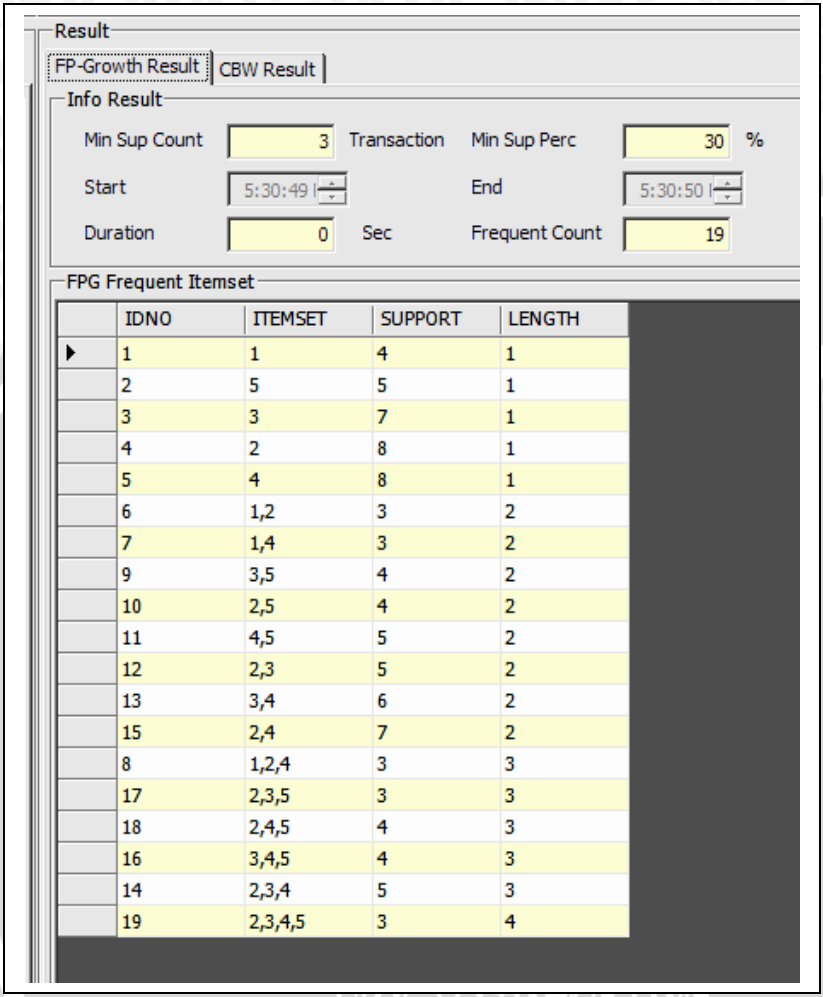
Untuk mengetahui kebenaran keluaran perangkat lunak yang dibangun, dilakukan perbandingan *frequent itemset* yang dihasilkan oleh perangkat lunak dengan *frequent itemset* yang dihitung secara manual. Dalam pengujian ini digunakan *dataset* Contoh yang ditunjukkan pada tabel 3.2 pada halaman 29, yang terdiri dari 10 transaksi, 8 item, dan 36 *record*. *Frequent itemset* hasil perhitungan manual pada algoritma *CBW* dan *FP-Growth* masing-masing ditunjukkan pada tabel 3.13 dan 3.16. Keduanya menunjukkan hasil yang sama yaitu 19 buah *frequent itemset.*

Hasil perhitungan algoritma *CBW* menggunakan perangkat lunak ditunjukkan oleh gambar 4.3. Dan gambar 4.4 adalah keluaran perangkat lunak untuk algoritma *FP-Growth*.



IDNO	ITEMSET	SUPPORT	LENGTH
5	1	4	1
4	5	5	1
2	3	7	1
1	2	8	1
3	4	8	1
12	1,2	3	2
13	1,4	3	2
10	3,5	4	2
8	2,5	4	2
6	2,3	5	2
11	4,5	5	2
9	3,4	6	2
7	2,4	7	2
15	2,3,5	3	3
18	1,2,4	3	3
17	3,4,5	4	3
16	2,4,5	4	3
14	2,3,4	5	3
19	2,3,4,5	3	4

Gambar 4.3 *Frequent Itemset* hasil dari Aplikasi dengan algoritma CBW

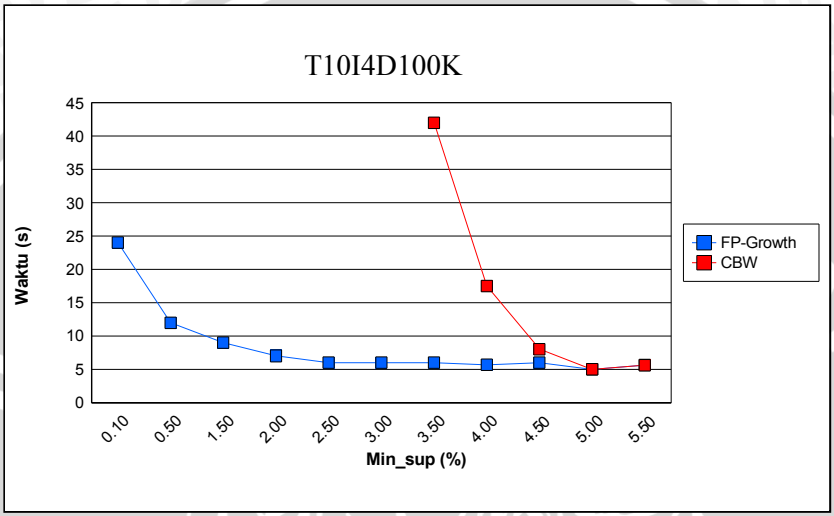


IDNO	ITEMSET	SUPPORT	LENGTH
1	1	4	1
2	5	5	1
3	3	7	1
4	2	8	1
5	4	8	1
6	1,2	3	2
7	1,4	3	2
9	3,5	4	2
10	2,5	4	2
11	4,5	5	2
12	2,3	5	2
13	3,4	6	2
15	2,4	7	2
8	1,2,4	3	3
17	2,3,5	3	3
18	2,4,5	4	3
16	3,4,5	4	3
14	2,3,4	5	3
19	2,3,4,5	3	4

Gambar 4.4 *Frequent Itemset* hasil dari Aplikasi dengan algoritma *FP-Growth*

Dari hasil pengujian didapat bahwa *frequent itemset* hasil keluaran dari perangkat lunak yang dibangun sama dengan hasil perhitungan manual. Dengan ini dapat dibuktikan bahwa output sistem adalah benar. Urutan data yang dihasilkan algoritma FP-Growth dan algoritma CBW diurutkan per level, sehingga memudahkan untuk membanding kebenaran hasil dari dua algoritma ini.

4.6 Analisis Waktu Pembangkitan Frequent Itemset Terhadap *Minimal Support*



Gambar 4.5. Grafik waktu pembangkitan *frequent itemset* terhadap *minimal support* pada *dataset* T10I4D100K

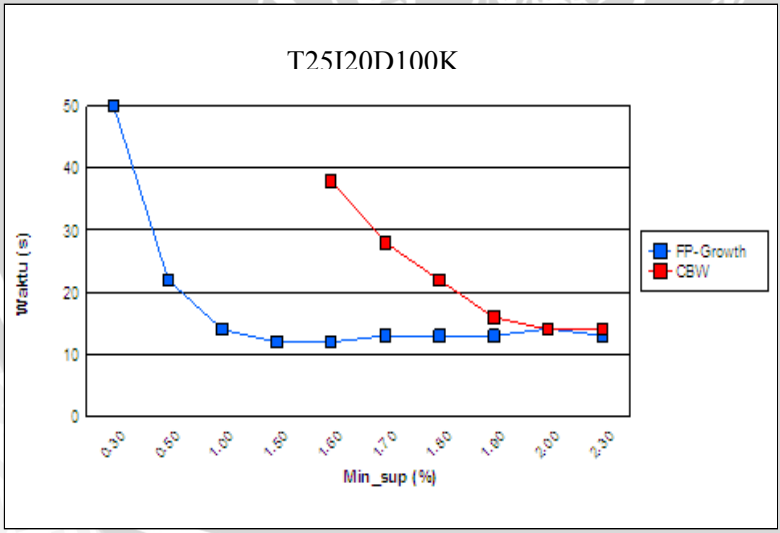
Gambar 4.5 menampilkan grafik waktu dari proses mining terhadap dataset T10I4D100K. Nilai *minimal support* yang digunakan untuk analisis waktu pembangkitan *frequent itemset* pada dataset ini yaitu 0.1%, 0.5%, 1.5%, 2%, 2.5%, 3%, 3.5%, 4%, 4.5%, 5%, dan 5.5%. Alasan pemilihan *minimal support* ini adalah jika dipilih *minimal support* lebih besar dari 5,5% perbedaan performansi kedua algoritma tidak begitu terlihat dan jumlah *frequent itemset* semakin kecil. Sedangkan algoritma CBW hanya diujikan sampai nilai *minimal support* 3.5%, karena jika nilai *minimal support* lebih kecil dari itu waktu yang dibutuhkannya lebih dari 50 detik dan perbedaan performansinya terlalu jauh. *Report* dari dataset T10I4D100K ditunjukkan pada tabel 4.1.

Dataset T10I4D100K termasuk *sparse dataset*, *frequent itemset* yang dihasilkan pendek-pendek dan tidak banyak. Pada gambar 4.5, bentuk grafik waktu jalannya algoritma CBW hampir sama dengan FP-Growth, yaitu semakin kecil nilai *minimal support* waktu yang dibutuhkan akan semakin meningkat tajam. Sedangkan saat jumlah frequent itemset yang dihasilkan semakin sedikit pada

tingkatan *minimal support* tertentu nilainya konstan. Pada gambar 4.5 tersebut juga dapat dilihat bahwa pada tingkatan *minimal support* berapa pun algoritma CBW tidak lebih baik waktunya dari algoritma FP-Growth, pada dataset T10I4D100K.

Tabel 4.2 *Report* percobaan *mining frequent itemset* pada *dataset* T10I4D100K dengan beberapa nilai *minimal support*

No.	Min.Support (%)	∑ Frequent Itemset	Durasi (detik)	
			FP-growth	CBW
1	0,1	12092	24	-
2	0,5	588	12	-
3	1,5	230	9	-
4	2	148	7	-
5	2,5	92	6	-
6	3	46	6	-
7	3,5	32	6	42
8	4	19	5	18
9	4,5	8	6	8
10	5	5	5	5
11	5,5	3	5	5



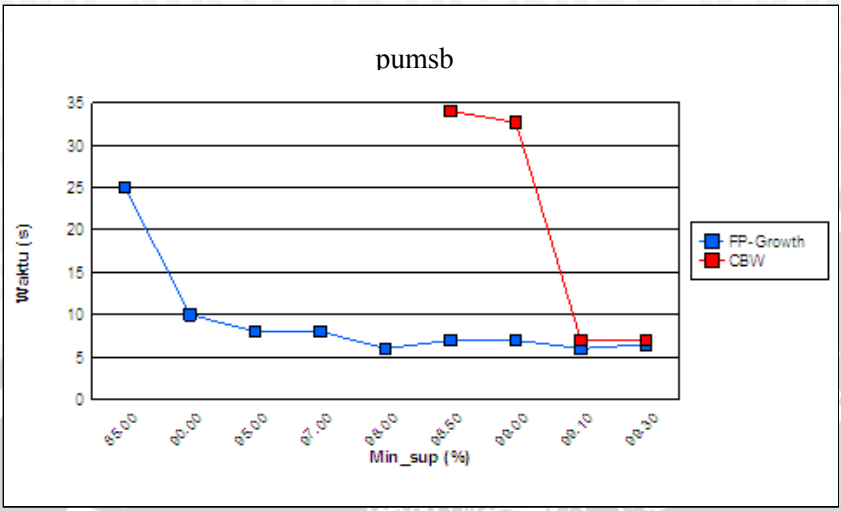
Gambar 4.6. Grafik waktu pembangkitan *frequent itemset* terhadap *minimal support* pada *dataset* T25I20D100K

Gambar 4.6 menampilkan hasil dari proses mining terhadap dataset T25I20D100K pada beberapa tingkatan nilai *minimal support*. Jumlah *frequent itemset* yang dihasilkan meningkat secara exponential pada dataset T25I20D100K, saat nilai *minimal support* semakin kecil. Jumlah *long-frequent itemset* yang dihasilkan sama banyaknya dengan jumlah *short-frequent itemset.* Pada gambar 4.6, waktu jalannya kedua algoritma rata-rata turun saat nilai *minimal support* meningkat. Pada gambar 4.6 tersebut juga dapat dilihat bahwa pada tingkatan *minimal support* berapa pun algoritma CBW tidak lebih baik waktunya dari algoritma FP-Growth, pada dataset T25I20D100K.

Tabel 4.2 menampilkan hasil dari proses mining terhadap dataset T25I20D100K. Nilai *minimal support* yang digunakan untuk analisis waktu pembangkitan *frequent itemset* pada dataset ini yaitu 0.3%, 0.5%, 1%, 1.5%, 1.6%, 1.7%, 1.8%, 1.9%, 2%, dan 2.3%. Alasan pemilihan *minimal support* ini adalah jika dipilih *minimal support* lebih besar dari 2.3% tidak ada *frequent itemset* yang terbentuk. Pada algoritma CBW hanya diujikan sampai nilai *minimal support* 3.5% dan *FP-Growth* 0.3%, jika nilai *minimal support* lebih kecil dari itu waktu yang dibutuhkannya lebih dari 50 detik dan perbedaan performansinya terlalu jauh.

Tabel 4.2 *Report* percobaan *mining frequent itemset* pada *dataset* T25I20D100K dengan beberapa nilai *minimal support*

No.	Min.Support (%)	∑ Frequent Itemset	Durasi (detik)	
			FP-growth	CBW
1	0,3	3061	50	-
2	0,5	1449	22	-
3	1	213	14	-
4	1,5	22	12	-
5	1,6	18	12	38
6	1,7	14	13	28
7	1,8	11	12	21
8	1,9	6	11	15
9	2	3	14	15
10	2,3	1	12	14



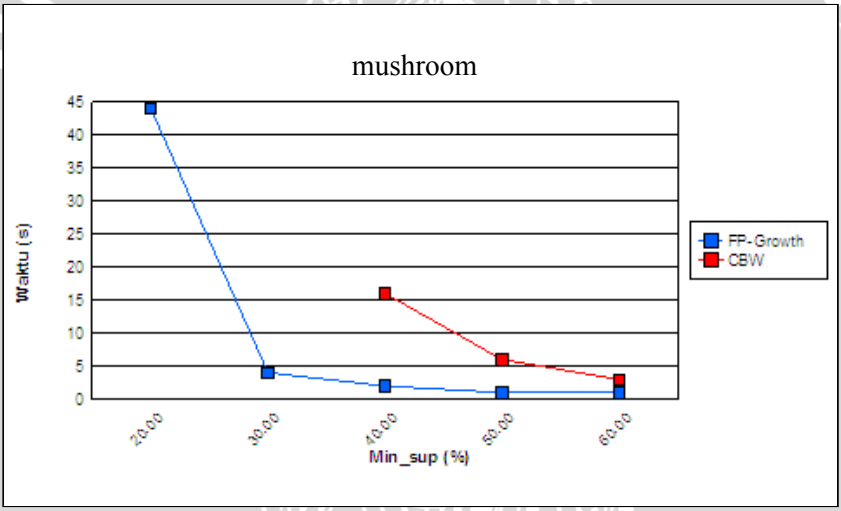
Gambar 4.7. Grafik waktu pembangkitan *frequent itemset* terhadap *minimal support* pada *dataset pumsb*

Dataset *pumsb* merupakan salah satu dari dataset asli yang sering digunakan dalam menghitung waktu pencarian *frequent itemset*, pada beberapa penelitian sebelumnya. Pada gambar 4.7, dapat dilihat bahwa jumlah *frequent itemset* yang dihasilkan meningkat secara exponential pada dataset *pumsb*, saat nilai *minimal support* semakin kecil, dan waktu jalannya kedua algoritma rata-rata turun saat nilai *minimal support* meningkat. Pada gambar 4.7 tersebut juga dapat dilihat bahwa pada tingkatan *minimal support* berapa pun algoritma CBW tidak lebih baik waktunya dari algoritma FP-Growth, pada dataset *pumsb.*

Tabel 4.3 menampilkan hasil dari proses mining terhadap dataset *pumsb*. Nilai *minimal support* yang digunakan untuk analisis waktu pembangkitan *frequent itemset* pada dataset ini yaitu 85%, 90%, 95%, 97%, 98%, 98.5%, 99%, 99.1%, dan 99.3%. Alasan pemilihan *minimal support* ini adalah jika dipilih *minimal support* lebih besar dari 99.3% tidak ada *frequent itemset* yang terbentuk. Pada algoritma CBW hanya diujikan sampai nilai *minimal support* 98.5% dan *FP-Growth* 85%, jika nilai *minimal support* lebih kecil dari itu waktu yang dibutuhkannya lebih dari 50 detik dan perbedaan performansinya terlalu jauh.

Tabel 4.3 *Report* percobaan *mining frequent itemset* pada *dataset pumsb* dengan beberapa nilai *minimal support*

No.	Min.Support (%)	∑ Frequent Itemset	Durasi (detik)	
			FP-growth	CBW
1	85	20533	25	-
2	90	2608	11	-
3	95	172	8	-
4	97	38	8	-
5	98	14	6	-
6	98,5	5	7	34
7	99	3	7	33
8	99,1	2	6	7
9	99,3	1	6	7



Gambar 4.8 Grafik waktu pembangkitan *frequent itemset* terhadap *minimal support* pada *dataset mushroom*

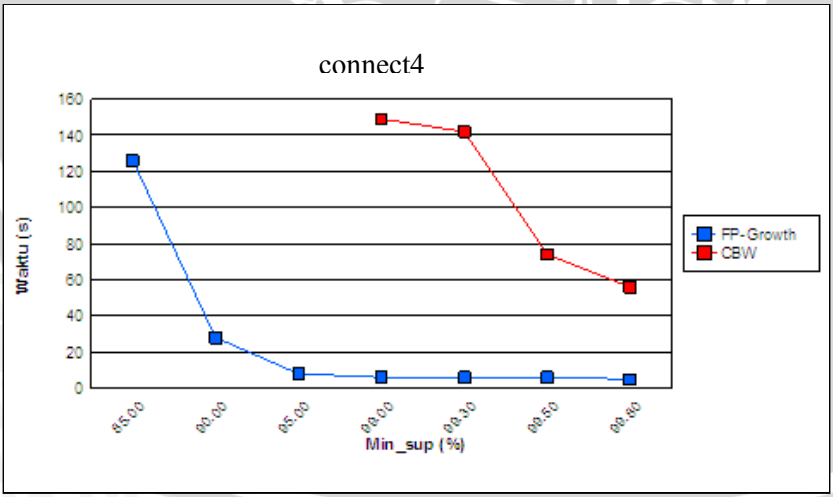
Dataset *mushroom* merupakan salah satu dari dataset asli yang sering digunakan dalam menghitung waktu pencarian *frequent itemset*, pada beberapa penelitian sebelumnya. Pada gambar 4.8 dapat dilihat bahwa pada dataset *pumsb,* jumlah *frequent itemset* yang dihasilkan meningkat saat nilai *minimal support* semakin kecil. Pada gambar 4.8 tersebut juga dapat dilihat bahwa pada dataset

mushroom, pada tingkatan *minimal support* berapa pun algoritma CBW tidak lebih baik waktunya dari algoritma FP-Growth.

Tabel 4.4 menampilkan hasil dari proses mining terhadap dataset *mushroom*. Nilai *minimal support* yang digunakan untuk analisis waktu pembangkitan *frequent itemset* pada dataset ini yaitu 20%, 30%, 40%, 50%, dan 60%. Alasan pemilihan *minimal support* ini adalah jika dipilih *minimal support* lebih besar dari 60% perbedaan performansi kedua algoritma tidak begitu terlihat. Pada algoritma CBW hanya diujikan sampai nilai *minimal support* 40% dan *FP-Growth* 20%, jika nilai *minimal support* lebih kecil dari itu waktu yang dibutuhkannya lebih dari 50 detik dan perbedaan performansinya terlalu jauh.

Tabel 4.4 *Report* percobaan *mining frequent itemset* pada *dataset mushroom* dengan beberapa nilai *minimal support*

No.	Min.Support (%)	∑ Frequent Itemset	Durasi (detik)	
			FP-growth	CBW
1	20	53583	44	-
2	30	2735	4	-
3	40	565	2	16
4	50	153	1	6
5	60	51	1	3



Gambar 4.9. Grafik waktu pembangkitan *frequent itemset* terhadap *minimal support* pada *dataset connect4*

Gambar 4.9 menampilkan hasil dari proses mining terhadap dataset *connect4* pada beberapa tingkatan nilai *minimal support*. *Connect4* merupakan dataset asli yang bersifat *dense* yang menghasilkan banyak sekali pola-pola yang panjang atau *long-frequent itemset*. Pada gambar 4.9, waktu jalannya kedua algoritma rata-rata turun saat nilai *minimal support* meningkat.

Tabel 4.5 menampilkan hasil dari proses mining terhadap dataset *connect4*. Nilai *minimal support* yang digunakan untuk analisis waktu pembangkitan *frequent itemset* pada dataset ini yaitu 85%, 90%, 95%, 99%, 99.3%, 99.5%, dan 99.8%. Alasan pemilihan *minimal support* ini adalah jika dipilih *minimal support* lebih besar dari 99.8% perbedaan performansi kedua algoritma tidak begitu terlihat dan jumlah *frequent itemset* semakin kecil. Pada algoritma CBW hanya diujikan sampai nilai *minimal support* 99% dan *FP-Growth* 85%, jika nilai *minimal support* lebih kecil dari itu waktu yang dibutuhkannya lebih dari 150 detik dan perbedaan performansinya terlalu jauh.

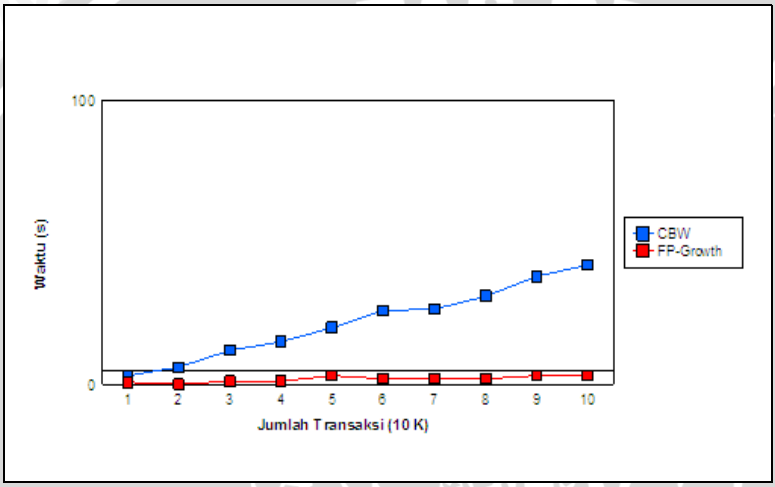
Pada gambar 4.6 tersebut juga dapat dilihat bahwa pada tingkatan *minimal support* berapa pun algoritma CBW tidak lebih baik waktunya dari algoritma FP-Growth, pada dataset *connect4*. Hal tersebut membuktikan bahwa algoritma FP-Growth jauh lebih unggul pada *dense dataset*.

Tabel 4.5 *Report* percobaan *mining frequent itemset* pada *dataset connect4* dengan beberapa nilai *minimal support*

No.	Min.Support (%)	∑ Frequent Itemset	Durasi (detik)	
			FP-growth	CBW
1	85	142271	126	-
2	90	27127	28	-
3	95	2205	8	-
4	99	29	6	149
5	99,3	11	6	142
6	99,5	8	6	74
7	99,8	3	5	56

4.7 Analisis Waktu Pembangkitan Frequent Itemset Terhadap Peningkatan Jumlah Transaksi

Untuk analisis waktu pembangkitan *frequent itemset* terhadap pertambahan jumlah transaksi, digunakan dataset pada tabel 3.4 pada halaman 32 dengan nilai *minimal support* 3%. Gambar 4.10 menunjukkan grafik hubungan waktu pembangkitan *frequent itemset* dengan pertambahan jumlah transaksi.



Gambar 4.10. Grafik hubungan waktu pembangkitan *frequent itemset* dengan penambahan jumlah transaksi pada *minimal support 3%.*

Pada gambar 4.10 dapat dilihat bahwa pertambahan waktu pembangkitan *frequent itemset* cenderung linier. Semakin banyak jumlah transaksi maka waktu pembangkitan *frequent itemset* semakin lama, demikian pula sebaliknya.

Secara logika proses semakin banyak transaksi maka semakin banyak transaksi yang dibaca untuk menentukan nilai *support*. Untuk algoritma *CBW*, walaupun maximal terjadi 4 kali pembacaan basis data, waktu pembentukan kandidat itemsetnya akan terus meningkat bila makin banyak transaksi.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Algoritma FP-Growth dan algoritma CBW telah berhasil diterapkan dalam proses pencarian *frequent itemset.* Kedua algoritma tersebut telah diujikan ke beberapa dataset yang berbeda karakteristiknya dan pada berbagai tingkatan nilai *minimal support.* Dalam pengujian tersebut dapat disimpulkan bahwa:

1. *Frequent itemset* hasil keluaran kedua algoritma dari perangkat lunak yang dibangun sama dengan hasil kedua algoritma dalam perhitungan manual. Dengan ini dapat dibuktikan bahwa output sistem adalah benar.
2. Algoritma FP-Growth mampu menambang *frequent itemset* lebih cepat dibanding algoritma CBW dengan beberapa parameter *minimal support*, karena proses pembacaan basis data yang lebih sedikit. Algoritma CBW melakukan 2 (dua) sampai 4 (empat) kali pembacaan basis data. Sedangkan algoritma FP-Growth dengan menggunakan struktur FP-tree, ia hanya perlu melakukan 2 (dua) kali pembacaan basis data.
3. Pertambahan jumlah transaksi untuk kedua algoritma cenderung linier dengan waktu yang digunakan. Hal ini disebabkan karena semakin banyaknya pembacaan basis data.
4. Performansi algoritma FP-Growth pada saat *minimal support* semakin kecil lebih baik dibanding algoritma CBW. Sedangkan pada saat *minimal support* semakin besar performansi algoritma CBW mendekati performansi algoritma FP-Growth, tetapi tidak bisa lebih baik dan hanya bisa sama untuk yang tidak ditemukan *frequent itemset*.

5.2 Saran

Pada pengembangan selanjutnya, skripsi ini bisa dijadikan acuan untuk menciptakan algoritma baru untuk proses pencarian *frequent itemset* yang lebih cepat waktu komputasinya daripada algoritma CBW dan FP-Growth. Hal tersebut bisa dilakukan dengan memanfaatkan keunggulan dari masing-masing algoritma, yaitu struktur FP-tree pada algoritma FP-Growth dan penggunaan cutting level pada algoritma CBW.

DAFTAR PUSTAKA

Agrawal, R., Imielinski, T., dan A. Swami, A. 1993. *Mining Association Rules between Sets of Items in Large Database*. ACM SIGMOD. Washington DC.

Buku Ajar Metode Numerik. 2002. *ANALISIS ALGORITMA.pdf*. Proyek HEDS.

Database System Research Group (DSRG), http://mirror.cs.curtin.edu.au/other/dsrg/resources/res_arm.html

FIMI Repository. http://www.fimi.cs.helsinki.fi/data/

Gopalan, Raj P., Sucahyo Yudho G.. 2004. *High Performance Frequent Patterns Extraction using Compressed FP-Tree*. Department of Computing, Curtin University of Technology. Australia.

Grahne, G., Zhu, J.. 2005. *Fast Algorithms for Frequent Itemset Mining Using FP-Trees.* IEEE Transactions on Knowlegde and Data Engineering, vol 17, No 10.

Hariyanto, Bambang. 2003. *STRUKTUR DATA, Memuat Dasar Pengembangan Orientasi Objek*. Informatika. Bandung.

Hidayat, Asri. 2006. *Analisis dan Implementasi Algoritma Frequent Pattern Growth* (FP -GROWTH*) untuk Mendapatkan Frequent Itemset pada Data Mining Association Rule*. STT Telkom. Bandung.

http://www.codeplex.com/fpminer/SourceControl/DownloadSourceCode.aspx?changeSetId=7188

http://www.cs.rpi.edu/~zaki/dmcourse/fall03/notes/9-29-03/9-29-03.ps

Hwung Su J., Yang Lin W. 2004. *CBW: An Efficient Algoritma for Frequent Itemset Mining*. The 37th Hawaii International Conference on System Sciences.

J. Han, J. Pei, Y. Yin, and R. Mao. 2000. *Mining Frequent Patterns without Candidate Generation: A Frequentpattern Tree Approach*. Data Mining and Knowledge Discovery: An International Journal, Kluwer Academic Publishers, vol. 8, pp. 53-87.

Maryeti, Sri. 2006. *Analisis Perbandingan Algoritma FP-Growth dan Algoritma Tree Projection dalam Pembangkitan Frequent Pattern*. STT Telkom. Bandung.

Perdana, Fatwa., Djunaidy, Arif. 2006 . *Implementasi Penggalian Kaidah Asosiasi Tanpa Ambang Batas Support dengan enggunakan Algorita Bomo.* Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember. Surabaya.

Pramudiono, Iko. 2003. *Pengantar Data Mining: Menambang Permata Pengetahuan di Gunung Data.* IlmuKomputer.Com.

Sucahyo, Yudho G.. 2003. *Data Mining: Menggali Informasi yang Terpendam.* IlmuKomputer.Com.

Sukarya, Oyo. 2006. *Perbandingan Pencarian Frequent Itemset Menggunakan Algoritma Cut Both Ways dan Algoritma Apriori.*STT Telkom. Bandung.

UCI *Machine Learning Repository.* http://archive.ics.uci.edu/ml/datasets.html

LAMPIRAN A
ALGORITMA IMPLEMENTASI

A.1 FP-Growth

```
using System;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Configuration;
using System.Text;
using FI.TypedDataSet;
using FI.DataAccess;

namespace FI.DAFPGrowth
{
/// <summary>
/// Optimized Implementation of the well-known FPGrowth
algorithm. For more look at original paper:
/// "Mining Frequent Patterns without Candidate
Generation", Jiawei Han,Jian Pei e Yiwen Yin (1999).
/// </summary>
public class FPGrowthDFetcher
{
private List<ItemSet> result; // Final pool list
frequent ItemSets

/// <summary>
/// Frequent pattern extraction method that implement
FPGrowth logic using an
/// iterative pattern growth approach
/// </summary>
/// <param name="allTrans">Total list of input
transaction</param>
/// <returns>Frequent ItemSets</returns>
public void ExtractFrequentPattern(int MinSup)
{
MasterDataFetcher mst = new MasterDataFetcher();

//Prepare data with clean all table to be use
mst.TruncateTable();

//Load transaction itemset
TransactionItemsetTable dsTransItem =
mst.GetTransactionItemsetTable();

result = new List<ItemSet>();
```

```
//We could have used two stacks instead of these
lists!!!
List<FPTree> FPTreeList = new List<FPTree>();
List<FPTree> FPTreeListNext = new List<FPTree>();
List<ItemSet> GivenListNow = new List<ItemSet>();
List<ItemSet> GivenListNext = new List<ItemSet>();
FPTree Next;
ItemSet Given;
ItemSet Beta;

//Convert transaction data become list of itemset
List<ItemSet> wholeTransToItemSet = new List<ItemSet>();
foreach (TransactionItemsetTable.TRANSACTION_ITEMSETRow
tranRow in dsTransItem.TRANSACTION_ITEMSET)
{
if (!tranRow.IsITEMSETNull())
{
string[] sItem = tranRow.ITEMSET.Split(',');
ItemSet x = new ItemSet();
foreach (string item in sItem)
{

if (item.Length >= 1)
{ x.Add(Convert.ToInt32(item)); }
}
wholeTransToItemSet.Add(x);
}
}

FPTree StartFPtree = new FPTree();
StartFPtree.SetMinSup(MinSup);
//Build the first tree on the whole transaction database
list
StartFPtree.BuildTree(wholeTransToItemSet);
FPTreeList.Add(StartFPtree);

//Here our given prefix is null
GivenListNow.Add(new ItemSet());

//Looping on each fptree until there are ones to process
while (FPTreeList.Count > 0)
{
for (int j = 0; j < FPTreeList.Count; j++)
{
Next = FPTreeList[j];
Given = GivenListNow[j];
if (!Next.isEmpty())
{
```

```
// If the FPTree we are examining is composed of a
single path
// we use an optimization based on path node combination
which
// arrest current iteration
if (Next.HasSinglePath)
{
GenerateCombPattern(Next, Given);
}
else
{
// Header table sorting
Next.SortHeaderTable();
//Loop on each header table entry
for (int i = 0; i < Next.GetHTSize(); i++)
{
//New beta ItemSet representing a frequent pattern
Beta = new ItemSet();
//Concatenate with items present in the given ItemSet
foreach (int item in Given.Items)
{
    Beta.Add(item);
}
Beta.Add(Next.GetHTItem(i));
Beta.ItemsSupport = Next.GetHTFreq(i);

// Add beta to frequent pattern result
result.Add(Beta);

// Here we generate the so called Conditional FPTree
using a projection
// of the original database called Conditional Pattern
Base
FPTreeListNext.Add(Next.CreateFPtree(i));

// Insert current beta in next given list
GivenListNext.Add(Beta);
}
FPTreeList[j] = null;
GivenListNow[j] = null;
}
}
}
FPTreeList.Clear();
GivenListNow.Clear();
for (int j = 0; j < GivenListNext.Count; j++)
{
FPTreeList.Add(FPTreeListNext[j]);
GivenListNow.Add(GivenListNext[j]);
```

```
GivenListNext[j] = null;
FPTreeListNext[j] = null;
}
GivenListNext.Clear();
FPTreeListNext.Clear();
}

//Get All Frequent Itemset and update to Database
FrequentItemsetTable dsFrequent = new
FrequentItemsetTable();
foreach (ItemSet items in result)
{
items.Sort();
FrequentItemsetTable.FREQUENT_ITEMSETRow frow =
dsFrequent.FREQUENT_ITEMSET.NewFREQUENT_ITEMSETRow();
frow.ITEMSET = items.ToString() ;
frow.LENGTH = items.ItemsNumber;
frow.SUPPORT = items.ItemsSupport;
dsFrequent.FREQUENT_ITEMSET.AddFREQUENT_ITEMSETRow(frow)
;
}
mst.UpdateFrequentItemsetTable(dsFrequent);
}

/// <summary>
/// Efficient optimization in frequent pattern
generation when an FPTree present only a single path
/// Generated  frequent are automatically inserted in
result (frequent pattern pool).
/// The method enumerates all combinations of node in
FPTree fp using efficiently bit operators
/// </summary>
/// <param name="fp">Single path FPtree </param>
/// <param name="given">ItemSet given prefix</param>
private void GenerateCombPattern(FPTree fp, ItemSet
given)
{
int bits = fp.Depth;
UInt64 enumerator = 1;
UInt64 combination;
UInt64 max = 1;
int index;
int[] itemsetArray = new int[fp.Depth];
int[] supportArray = new int[fp.Depth];
int betaSupport;

fp.Travel = fp.Root;
for (int i = 0; i < fp.Depth; i++)
{
```

```
fp.Travel = fp.Travel.Children[0];
itemsetArray[i] = fp.Travel.Item;
supportArray[i] = fp.Travel.Count;
}
Array.Reverse(itemsetArray);
Array.Reverse(supportArray);
//Max represent the overflow condition
max = max << bits;
//Our enumerator is represented through a 64 bit integer
while (enumerator < max)
{
// Create a new beta result
ItemSet beta = new ItemSet();
betaSupport = 0;
foreach (int item in given.Items)
{
beta.Add(item);
}

index = 0;
combination = enumerator;

while (combination > 0)
{
if ((combination % 2) == 1)
{
beta.Add(itemsetArray[index]);
if ((betaSupport > supportArray[index]) || (betaSupport
== 0))
{
betaSupport = supportArray[index];
}
}
combination = combination >> 1;
index++;
}
enumerator++;
beta.ItemsSupport = betaSupport;
result.Add(beta);
}
}
}
}
```

FP-Tree

```
using System;
using System.Data;
using System.Xml;
using System.Collections;
```

```
using System.Collections.Generic;
using System.Text;
using System.Configuration;
using FI.TypedDataSet;
using FI.DataAccess;


namespace FI.DAFPGrowth
{
public class FPTree
{
private int _minSup;
private FPNode _root;
private bool _hasSinglePath;
private Dictionary<int, int> FrequencyItemCounter;
//LookUp Table(Item,Frequency)
private List<ItemSet> _singleFrequent;
private HeaderTable ht;
private int _depth;
private FPNode _travel;

/// <summary>
/// Get or set the root of the tree
/// </summary>
public FPNode Root
{
get { return _root; }
set { _root = value; }
}

/// <summary>
/// Get or set the minimum support for the fptree
/// </summary>
public int MinSup
{
get { return _minSup; }
set { _minSup = value; }
}

/// <summary>
/// Get or set the frequent 1-itemset
/// </summary>
public List<ItemSet> SingleFrequent
{
get { return _singleFrequent; }
set { _singleFrequent = value; }
}
```

```
/// <summary>
/// Check if the tree has a single path. Test used for
speed up
/// association rule generation
/// </summary>
public bool HasSinglePath
{
get { return _hasSinglePath; }
set { _hasSinglePath = value; }
}

/// <summary>
/// Get or set the tree Depth
/// </summary>
public int Depth
{
get { return _depth; }
set { _depth = value; }
}

/// <summary>
/// Get or set a node used to traverse the fptree
/// </summary>
public FPNode Travel
{
get { return _travel; }
set { _travel = value; }
}

public FPTree()
{
MinSup = default(int);

Root = new FPNode();
Root.Item = default(int);
Root.Parent = default(FPNode);
_singleFrequent = new List<ItemSet>();
_hasSinglePath = true;
_depth = 0;
}

/// <summary>
/// Method invoked for building the first FPTree
representation of the
/// original database
/// </summary>
/// <param name="startdb">The list of database
transactions</param>
public void BuildTree(List<ItemSet> startdb)
```

```
{
// Counting the frequency of single items
FrequencyItemCounter = new Dictionary<int, int>();
FrequencyItemCounter = CountFrequencyItem(startdb);

// Evaluate header table dimension
int HTsize = 0;
foreach (int item in FrequencyItemCounter.Keys)
{
if (FrequencyItemCounter[item] >= _minSup) HTsize++;
}

ht = new HeaderTable(HTsize);

// Add every frequent single itemset to header table
foreach (KeyValuePair<int, int> coppia in
FrequencyItemCounter)
{
if (coppia.Value >= _minSup)
ht.addRecord(coppia.Key, coppia.Value);
}

// create complete FPTree
// Removal of non frequent items, sorting and final
insertion in the FPTree
ItemSet SortedList = new ItemSet();
foreach (ItemSet itemset in startdb)
{
SortedList.Items.Clear();
for (int i = 0; i < itemset.ItemsNumber; i++)
{
if (FrequencyItemCounter[itemset.Items[i]] >= _minSup)
{
SortedList.Add(itemset.Items[i]);
}
}
if (SortedList.ItemsNumber > 0)
{
SortedList.Items.Sort(new
ItemSortingStrategy(FrequencyItemCounter));
if (_depth < SortedList.ItemsNumber) _depth =
SortedList.ItemsNumber;
insert_tree(SortedList, Root);
}
}
startdb = null;
}
```

```
/// <summary>
/// Test the empty condition on tree, used for
optimization
/// purpose
/// </summary>
/// <returns>a boolean indicating if the fptree has
Children or not</returns>
public bool isEmpty()
{
if (_root.Children == default(FPNode[]))
return true;
else
return false;
}

/// <summary>
/// Add an itemset to an FPTree
/// </summary>
/// <param name="SortedList">itemset to be added
(already sorted)</param>
/// <param name="me">node where to start the
insertion</param>
private void insert_tree(ItemSet SortedList, FPNode me)
//proses insert_tree(p[P],R)
{
FPNode found;

//looping on each item and searching if it's present or
not in the FPTree
for (int i = 0; i < SortedList.ItemsNumber; i++)
{
//if the me node has got no Children let's create one
new
if (me.Children == default(FPNode[]))
{
me.Children = new FPNode[0];
}
int j = 0;
found = default(FPNode);
// j represented a sliding index that point to the
position where i want to
// add or insert the current item
while ((j < me.Children.Length) &&
(me.Children[j].Item != SortedList.Items[i]))
{
j++;
}
if (j < me.Children.Length)
{
```

```
found = me.Children[j];
}

// If founded node is present in the tree, increase it's
support and
// move on to the next item updating me reference
if (found != default(FPNode))
{
found.Count += SortedList.ItemsSupport;
me = found;
}
// Create a new FPNode and set header table head and
tail corresponding list
else
{
if (me.Children.Length > 0) _hasSinglePath = false;
FPNode newnode = new FPNode(SortedList.Items[i]);

int index = ht.GetIndex(newnode);
if (ht.GetTail(index) == default(FPNode))
{
ht.SetHead(newnode, index);
ht.SetTail(newnode, index);
}
else
{
ht.GetTail(index).Next = newnode;
ht.SetTail(newnode, index);
}
newnode.Count = SortedList.ItemsSupport;
// Temporary Array where to store the new list of child
nodes
FPNode[] tmpArray = new FPNode[me.Children.Length + 1];
for (int k = 0; k < me.Children.Length; k++)
{
tmpArray[k] = me.Children[k];
}
tmpArray[me.Children.Length] = newnode;
me.Children = tmpArray;
newnode.Parent = me;
// Me reference set to the new node
me = newnode;
}
}
}

public FPTree CreateFPtree(int i)
{
FPNode ResultRootNode = new FPNode();
```

```
FPTree Result = new FPTree();

Result.Root = ResultRootNode;
Result.MinSup = _minSup;

// Counting the frequency of single items
Dictionary<int, int> FrequencyItemCounter2 = new
Dictionary<int, int>();
Result.FrequencyItemCounter = FrequencyItemCounter2;

FPNode node;
//Obtain a reference to head of the list of the i.th
htrecord
node = ht.GetHead(i);

while (node != default(FPNode))
{
int support;
_travel = node;
support = _travel.Count;
_travel = _travel.Parent;
if (_travel.Parent != default(FPNode))
{
while (_travel.Parent != default(FPNode))
{
if (FrequencyItemCounter2.ContainsKey(_travel.Item))
{
FrequencyItemCounter2[_travel.Item] += support;
}
else
{
FrequencyItemCounter2.Add(_travel.Item, support);
}
_travel = _travel.Parent;
}
}
node = node.Next;
}

//Evaluate header table dimension
int HTsize = 0;
foreach (int item in FrequencyItemCounter2.Keys)
{
if (FrequencyItemCounter2[item] >= _minSup) HTsize++;
}

HeaderTable newht = new HeaderTable(HTsize);
Result.ht = newht;
```

```
//Insertion of frequent items in header table
foreach (KeyValuePair<int, int> coppia in
FrequencyItemCounter2)
{
if (coppia.Value >= _minSup)
newht.addRecord(coppia.Key, coppia.Value);
}

// Removal of non frequent items, sorting and final
insertion in the FPTreee
node = ht.GetHead(i);
while (node != default(FPNode))
{
_travel = node;
ItemSet path = new ItemSet();
path.ItemsSupport = _travel.Count;
_travel = _travel.Parent;
if (_travel.Parent != default(FPNode))
{
while (_travel.Parent != default(FPNode))
{
path.Add(_travel.Item);
_travel = _travel.Parent;
}
ItemSet SortedList = new ItemSet();
SortedList.ItemsSupport = path.ItemsSupport;
foreach (int item in path.Items)
{
if (FrequencyItemCounter2[item] >= _minSup)
{
SortedList.Add(item);
}
}
if (SortedList.Items.Count > 0)
{
SortedList.Items.Sort(new
ItemSortingStrategy(FrequencyItemCounter2));
if (Result._depth < SortedList.ItemsNumber)
Result._depth = SortedList.ItemsNumber;
Result.insert_tree(SortedList, ResultRootNode);
}
}
node = node.Next;
}
return Result;
}
}
}
```

A.2 CBW

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.Configuration;
using System.Text;
using System.Windows.Forms;
using FI.TypedDataSet;
using FI.DataAccess;

namespace FI.DAFPGrowth
{
public class CBW
{
private Dictionary<ItemSet, bool> FrequentItemSets; //
Frequent ItemSets generic set
private List<ItemSet> CandidateItemSets; // Candidate
ItemSets generic set
private List<ItemSet> Result; // final frequent ItemSets
result
private Dictionary<int, int> FrequentItems; // Frequent
1-ItemSet look up table (item vs frequency)
private List<Transaction> FlaggedDB; // transaction list
containing only frequent item (unfrequent removed)

/// <summary>
/// Main frequent pattern extraction method that
implement Apriori logic
/// </summary>
/// <param name="AllTrans">Total list of input
transaction</param>
/// <returns>Frequent ItemSets</returns>
public int ExtractFrequentPattern(int MinSup)
{
FrequentItemSets = new Dictionary<ItemSet, bool>();//
(new EqualityComparerItemSet());
FrequentItems = new Dictionary<int, int>();
Result = new List<ItemSet>();
FlaggedDB = new List<Transaction>();
MasterDataFetcher mst = new MasterDataFetcher();
Dictionary<int, Transaction> TIDList = new
Dictionary<int,Transaction>();
int alpha = 0;


//Prepare data with clean all table to be use
mst.TruncateTable();
```

```
//Load transaction itemset
TransactionItemsetTable dsTransItem =
mst.GetTransactionItemsetTable();

//Convert transaction data become list of itemset
List<Transaction> AllTrans = new List<Transaction>();
foreach (TransactionItemsetTable.TRANSACTION_ITEMSETRow
tranRow in dsTransItem.TRANSACTION_ITEMSET)
{
if (!tranRow.IsITEMSETNull())
{
string[] sItem = tranRow.ITEMSET.Split(',');
Transaction x = new Transaction();
x.Id = Convert.ToInt32(tranRow.ID);
foreach (string item in sItem)
{
if (item.Length >= 1) x.addItem(Convert.ToInt32(item));
}
AllTrans.Add(x);
}
}
// Compute support for 1-itemset and store in
FrequentItems dictionary
foreach (Transaction trans in AllTrans)
{
foreach (int product in trans.Itemset.Items)
{
if (FrequentItems.ContainsKey(product))
{
FrequentItems[product]++;
}
else
{
FrequentItems.Add(product, 1);
}
}
}

// Evaluate those 1-itemset whose support is greater
than minimum support
// and add them to both frequent 1-itemset and final
frequent itemsets result
foreach (int product in FrequentItems.Keys)
{
if (FrequentItems[product] >= MinSup)
{
ItemSet frequent = new ItemSet();
frequent.Add(product);
```

```
frequent.ItemsSupport = FrequentItems[product];
FrequentItemSets.Add(frequent, true);
Result.Add(frequent);
}
}
//FrequentItems.Clear();

Trans(AllTrans, ref TIDList, ref alpha, MinSup);
DwnSearch(AllTrans, alpha, MinSup);
UpSearch(TIDList, alpha, MinSup);

//Get All Frequent Itemset and update to Database
FrequentItemsetTable dsFrequent = new
FrequentItemsetTable();
foreach (ItemSet items in Result)
{
items.Sort();
FrequentItemsetTable.FREQUENT_ITEMSETRow frow =
dsFrequent.FREQUENT_ITEMSET.NewFREQUENT_ITEMSETRow();
frow.ITEMSET = items.ToString();
frow.LENGTH = items.ItemsNumber;
frow.SUPPORT = items.ItemsSupport;
dsFrequent.FREQUENT_ITEMSET.AddFREQUENT_ITEMSETRow(frow)
;
}
mst.UpdateFrequentItemsetTable(dsFrequent);

return alpha;
}

public void Trans(List<Transaction> AllTrans, ref
Dictionary<int,Transaction> TIDList, ref int alpha, int
MinSup)
{
//Generate candidat 2-itemsets
CandidateItemSets = CandidateGen(FrequentItemSets, 2);

int numf = 0;
int lengthTrans = AllTrans.Count;
foreach (Transaction trans in AllTrans)
{

// Remove unfrequent item from transactions list
Transaction ReducedTrans = new Transaction();
foreach (int item in trans.Itemset.Items)
{
if (FrequentItems.ContainsKey(item))
{
ReducedTrans.addItem(item);
```

```
}
}
if (ReducedTrans.Itemset.Items.Count > 0)
{
FlaggedDB.Add(ReducedTrans);
}

numf = numf + ReducedTrans.Itemset.ItemsNumber;

if (ReducedTrans.Itemset.ItemsNumber >= 3)
{
//Insert into TID List
foreach (int i in ReducedTrans.Itemset.Items)
{
if (TIDList.ContainsKey(i))
{
TIDList[i].addItem(trans.Id);
}
else
{
Transaction IDList = new Transaction();
IDList.addItem(trans.Id);
TIDList.Add(i, IDList);
}
}
}

foreach (ItemSet CanItemSet in CandidateItemSets)
{
if (CanItemSet < trans.Itemset)
{
CanItemSet.ItemsSupport++;
}
}
}
AllTrans = FlaggedDB;

FrequentItemSets.Clear();
// Check if  every candidate itemset is a frequent or
not and if it is,
// add it to final result
foreach (ItemSet itemset in CandidateItemSets)
{
if (itemset.ItemsSupport >= MinSup)
{
FrequentItemSets.Add(itemset, true);
Result.Add(itemset);
}
}
```

```
//Get alpha value from numf and length all transaction
alpha = System.Math.Abs(numf / lengthTrans); }

public void DwnSearch(List<Transaction> AllTrans, int
alpha, int MinSup)
{
//Build Look Up frequent items table
FrequentItems.Clear();
foreach (ItemSet itemset in FrequentItemSets.Keys)
{
foreach (int item in itemset.Items)
{
if (FrequentItems.ContainsKey(item))
{
FrequentItems[item] += itemset.ItemsSupport;
}
else
{
FrequentItems.Add(item, 0);
}
}
}
List<ItemSet> g = new List<ItemSet>();
for (int d = 3; d <= alpha; d++)
{

CandidateItemSets = CandidateGen(FrequentItemSets, d);
//FrequentItemSets.Clear();
if (CandidateItemSets.Count <= 0)
d = alpha + 1;
else
{
FrequentItemSets.Clear();
foreach (ItemSet s in CandidateItemSets)
{
g.Add(s);
FrequentItemSets.Add(s, true);
}
}
}
foreach (Transaction trans in AllTrans)
{
// Remove unfrequent item from transactions list
Transaction ReducedTrans = new Transaction();
if (trans.Itemset.ItemsNumber > 2)
{
// Evalute support for the generated frequent candidate
foreach (ItemSet CanItemSet in g)
```

```
{
if (CanItemSet < trans.Itemset)
{
CanItemSet.ItemsSupport++;
}
}
}
}
// Check if  every candidate itemset is a frequent or
not and if it is,
// add it to final result
foreach (ItemSet e in g)
{
if (e.ItemsSupport >= MinSup)
{
Result.Add(e);
}
}
}

public void UpSearch(Dictionary<int, Transaction>
TIDList, int alpha, int MinSup)
{
if (FrequentItemSets != null)
{

//Build Look Up frequent items table
FrequentItems.Clear();
foreach (ItemSet itemset in FrequentItemSets.Keys)
{
foreach (int item in itemset.Items)
{
if (FrequentItems.ContainsKey(item))
{
FrequentItems[item] += itemset.ItemsSupport;
}
else
{
FrequentItems.Add(item, 0);
}
}
}

//read the set of tidlists T and prune non-candidate
tidlists
Dictionary<int, Transaction> TList = new Dictionary<int,
Transaction>();
foreach (int item in FrequentItems.Keys)
{
```

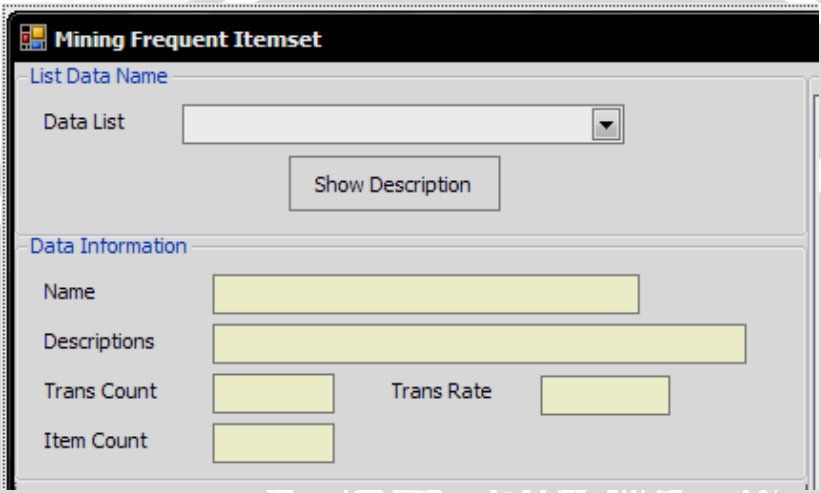
```
if (TIDList.ContainsKey(item))
TList.Add(item, TIDList[item]);
}
TIDList = TList;

bool IsContinued = true;
int k = alpha;
while (IsContinued)
{
k = k + 1;
//Generate candidat k-itemsets from F(k-1) or F(alpha)
CandidateItemSets = CandidateGen(FrequentItemSets, k);

FrequentItemSets.Clear();
foreach (ItemSet x in CandidateItemSets)
{
//bit vector intersection

List<ItemSet> d = new List<ItemSet>();
foreach (int h in x.Items)
{
d.Add(TIDList[h].Itemset);
}

ItemSet a = new ItemSet();
ItemSet b = new ItemSet();
a = d[0];
for (int m = 0; m < x.ItemsNumber - 1; m++)
{
ItemSet t = new ItemSet();
ItemSet u = new ItemSet();

b = d[m + 1];
if(a.ItemsNumber <= b.ItemsNumber)
{
t = a;
u = b;
}
else
{
t = b;
u = a;
}

ItemSet v = new ItemSet();

foreach (int g in t.Items)
{
if (u.Items.Contains(g))
```

```
{
v.Add(g);
}
}
a = v;
if (v.ItemsNumber < MinSup)
m = x.ItemsNumber;
}
//compute the support of x
x.ItemsSupport = a.ItemsNumber;

// Check if  every candidate itemset is a frequent or
not and if it is,
// add it to final result
if (x.ItemsSupport >= MinSup)
{
FrequentItemSets.Add(x, true);
Result.Add(x);
}

}
if (FrequentItemSets.Count <= 0)
IsContinued = false;
else
IsContinued = true;
}
}
}

/// <summary>
/// First step in Apriori algorithm responsible for the
generation of candidate k+1 itemset
/// starting from k frequent itemset. The generated
candidate itemsets could be frequent or not
/// hence the term candidate
/// </summary>
/// <param name="FrequentItemSets">The k generation
Frequent ItemSets</param>
/// <param name="k">the generation (lenght of current
itemsets)</param>
/// <returns>Generated list of candidate
ItemSets</returns>

private List<ItemSet> CandidateGen(Dictionary<ItemSet,
bool> FrequentItemSets, int k)
{
List<ItemSet> NewCandidateSet = new List<ItemSet>();
ItemSet NewCandidate;
int ok;
```

```
//Join Step: a candidate is generated joining two
frequent itemsets following a
//particular query:
//select p.item1,p.item2,. . . ,p.itemk,q.itemk, (k+1
elements)
//from Lk p,Lk q
//where (p.item1 = q.item1,p.item2 = q.item2,. . .
,p.itemk < q.itemk)
foreach (ItemSet itemset in FrequentItemSets.Keys)
{
foreach (int frequentitem in FrequentItems.Keys)
{
if (itemset.Items[itemset.ItemsNumber - 1] <
frequentitem)
{
NewCandidate = new ItemSet();
NewCandidate.ItemsSupport = 0;
foreach (int item in itemset.Items)
{
NewCandidate.Add(item);
}
NewCandidate.Add(frequentitem);

//Pruning, based on anti-monotonicity itemset principle
(see paper)
ok = 0;
for (int i = 0; i < k; i++)
{
ItemSet test = new ItemSet();
for (int j = 0; j < k; j++)
{
if (j != i) test.Add(NewCandidate.Items[j]);
}
if (FrequentItemSets.ContainsKey(test)) ok++;
// This itemset is contained in the frequent list
else i = k; // exit from loop this itemset is not
contained
}
if (ok == k) NewCandidateSet.Add(NewCandidate);
}
}
}
return NewCandidateSet;
}
}
}
```

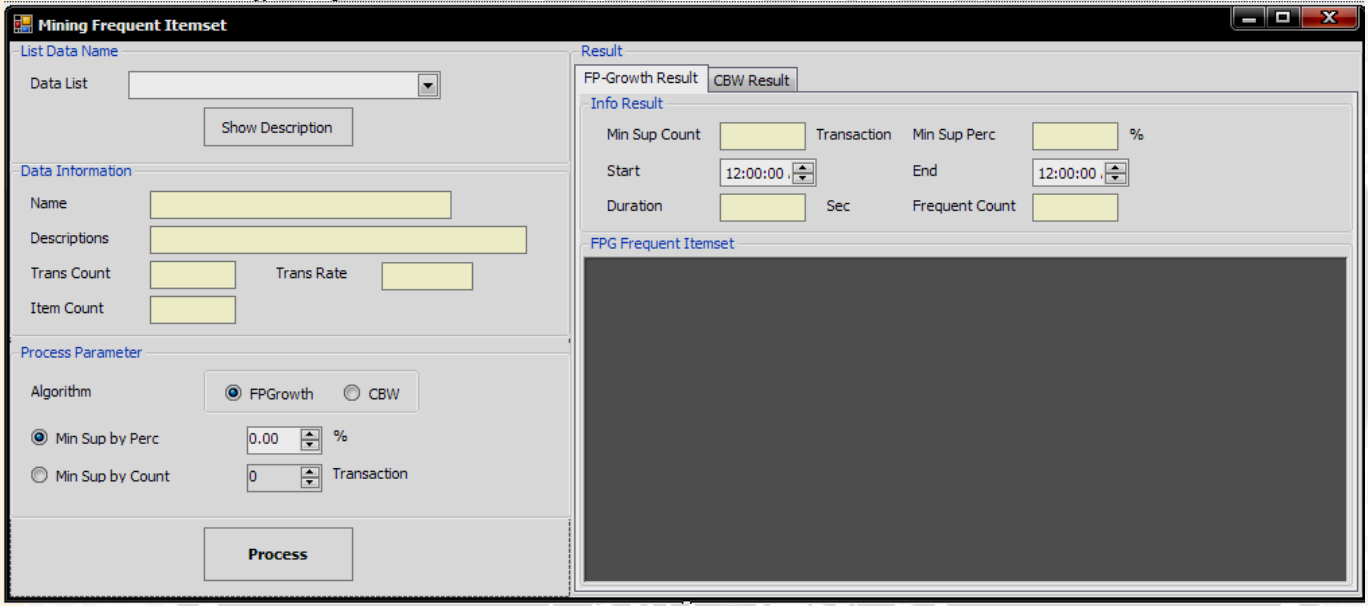
LAMPIRAN B
TAMPILAN APLIKASI

B.1. *Select Data*



Gambar B.3 *Select Data*

B.2. Form *Mining Frequent Itemset*



Gambar B.2 Form *Mining Frequent Itemset*