Penerapan Stuktur FP-Tree dan Algoritma FP-Growth dalam Optimasi Penentuan Frequent Itemset

David Samuel/NIM :13506081 [1)]

1) Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : david_samuel@students.itb.ac.id

Abstrak *- Pengolahan data secara cepat, efisien dan efektif sangat diperlukan oleh manusia seiring dengan perkembangan zaman. Sedangkan di sisi lain, data mentah yang memerlukan pemrosesan jumlahnya sangat banyak sehingga tidak memungkinkan lagi dilakukan pengolahan data secara manual.*

Untuk mengatasi masalah tersebut, telah ditemukan beragam struktur penyimpanan data yang berusaha untuk memperbaiki efisiensi pengolahan dan penggalian data, terutama dalam membangun sebuah struktur data dan mencari frequent itemset dalam sebuah database. Beberapa studi terakhir menggunakan pendekatan paradigma apriori, yang menggunakan metode candidate set generation. Akan tetapi, untuk masukan data yang besar, paragdigma apriori dianggap belum cukup efektif dan efisien.

Dalam makalah ini akan dibahas pembangunan dan aplikasi penggunaan struktur data pohon, yaitu FP-tree (Frequent Pattern Tree), yang merupakan perluasan dari penggunaan pohon dalam struktur data. FP-tree digunakan bersamaan dengan algoritma FP-growth untuk menentukan frequent itemset (data yang paling sering muncul) dari sebuah database, berbeda dengan paradigma apriori yang memerlukan langkah candidate generation, yaitu dengan melakukan scanning database secara berulang-ulang untuk menentukan frequent itemset. Makalah ini juga menyajikan pembahasan mengenai perbandingan kompleksitas waktu antara algoritma FP-growth dan beberapa metode frequent pattern mining lainnya.

Kata Kunci *: algoritma, apriori, FP-Tree, Fp-Growth, frequent pattern mining, frequent itemset*

1. PENDAHULUAN

Penggalian data merupakan salah satu cara yang cukup efektif untuk mengetahui adanya serangkaian pola informasi dari sejumlah besar data yang ada. Pola informasi yang didapat akan menjadi sangat berarti apabila bersifat implisit, belum diketahui sebelumnya, dan bermanfaat.

Pola asosiasi menjadi salah satu fungsionalitas yang paling menarik dalam penggalian data. Dikatakan menarik karena sudah banyak dipakai dalam kehidupan manusia sehari-hari, terutama yang berhubungan dengan data transaksi, misalnya kegiatan e-commerce. Pola asosiasi akan memberikan gambaran mengenai sejumlah atribut, atau sifat tertentu yang sering muncul bersamaan dalam kumpulan data yang diberikan.

FP-Growth adalah salah satu alternatif algoritma yang dapat digunakan untuk menentukan himpunan data yang paling sering muncul (frequent itemset) dalam sebuah kumpulan data. FP-growth menggunakan pendekatan yang berbeda dari paradigma yang selama ini sering digunakan, yaitu paradigma apriori.

Paradigma apriori yang dikembangkan oleh Agrawal dan Srikan (1994), yaitu
anti-monotone Apriori Heuristic: Setiap pola dengan panjang pola k yang tidak sering muncul (tidak frequent) dalam sebuah kumpulan data, maka pola dengan panjang (k+1) yang mengandung sub pola k tersebut tidak akan sering muncul pula(tidak frequent).

Ide dasar paradigma apriori ini adalah dengan mencari himpunan kandidat dengan panjang (k+1) dari sekumpulan pola frequent dengan panjang k, lalu mencocokkan jumlah kemunculan pola tersebut dengan informasi yang terdapat dalam *database*. Adapun hal ini akan mengakibatkan algoritma apriori akan melakukan *scanning database* yang berulang-ulang, apalagi jika jumlah data cukup besar. Berbeda dengan Algoritma FP-growth yang hanya memerlukan dua kali *scanning database* untuk menentukan *frequent itemset*.

Struktur data yang digunakan untuk mencari *frequent itemset* dengan algoritma FP-growth adalah perluasan dari penggunaan sebuah pohon prefix, yang biasa disebut adalah FP-tree. Dengan menggunakan FP-tree, algoritma FP-growth dapat langsung mengekstrak *frequent Itemset* dari FP tree yang telah terbentuk dengan menggunakan prinsip *divide and conquer.*

Pada bagian 2 akan dibahas proses pembentukan FP-tree dari sekumpulan data transaksi. Proses untuk menemukan *Frequent Itemset* dengan algoritma FP-growth akan dibahas pada bagian 3. Bagian 4 akan ditarik kesimpulan dari penggunaan struktur FP-tree dan algoritma FP-growth untuk mendapatkan *frequent itemset.*

2. PEMBANGUNAN FP-TREE

FP-tree merupakan struktur penyimpanan data yang dimampatkan. FP-tree dibangun dengan memetakan setiap data transaksi ke dalam setiap lintasan tertentu dalam FP-tree. Karena dalam setiap transaksi yang dipetakan, mungkin ada transaksi yang memiliki item yang sama, maka lintasannya memungkinkan untuk saling menimpa. Semakin banyak data transaksi yang memiliki item yang sama, maka proses pemampatan dengan struktur data FP-tree semakin efektif. Kelebihan dari FP-tree adalah hanya memerlukan dua kali pemindaian data transaksi yang terbukti sangat efisien.

Misal I= $\{a_1, a_2, \ldots, a_n\}$ adalah kumpulan dari item. Dan basis data transaksi DB = $\{T_1, T_2, \ldots, T_n\}$, di mana T_i (i € [1..n]) adalah sekumpulan transaksi yang mengandung item di I. Sedangkan *support* adalah penghitung (*counter*) frekuensi kemunculan transaksi yang mengandung suatu pola. Suatu pola dikatakan sering muncul (*frequent pattern*) apabila *support* dari pola tersebut tidak kurang dari suatu konstanta ξ (batas ambang minimum *support*) yang telah didefinisikan sebelumnya. Permasalahan mencari pola frequent dengan batas ambang minimum *support count* ξ inilah yang dicoba untuk dipecahkan oleh FP-Growth dengan bantuan Struktur FP-tree.

Adapun FP- tree adalah sebuah pohon dengan definisi sebagai berikut:

- FP-Tree dibentuk oleh sebuah akar yang diberi label *null*, sekumpulan upapohon yang beranggotakan item-item tertentu, dan sebuah tabel *frequent header.*
- Setiap simpul dalam FP-tree mengandung tiga informasi penting, yaitu label item, menginformasikan jenis item yang direpresentasikan simpul tersebut, *support count*, merepresentasikan jumlah lintasan transaksi yang melalui simpul tesebut, dan pointer penghubung yang menghubungkan simpul-simpul dengan label item sama antar-lintasan, dintandai dengan garis panah putus-putus.

Contoh 1 Misalkan diberikan tabel data transaksi sebagai berikut, dengan minimum *support count* ξ=2

No	Transaksi
1	a,b
2	b,c,d,g,h
3	a,c,d,e,f
4	a,d,e
5	a,b,z,c
6	a,b,c,d
7	a,r
8	a,b,c
9	a,b,d
10	b,c,e

Tabel 2.1 Tabel data transaksi mentah

Frekuensi kemunculan tiap item dapat dilihat pada tabel berikut:

Item	Frekuensi
a	8
b	7
c	6
d	5
e	3
f	1
r	1
z	1
g	1
h	1

Tabel 2.2 Frekuensi kemunculan tiap karakter

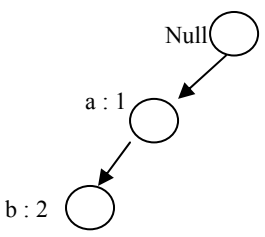
Setelah dilakukan pemindaian pertama didapat *item* yang memiliki frekuensi di atas *support count* ξ=2 adalah a,b,c,d, dan e. Kelima *item* inilah yang akan berpengaruh dan akan dimasukkan ke dalam FP-tree, selebihnya (r,z,g, dan h) dapat dibuang karena tidak berpengaruh signifikan.

Tabel berikut mendata kemunculan item yang *frequent* dalam setiap transaksi, diurut berdasarkan yang frekuensinya paling tinggi.

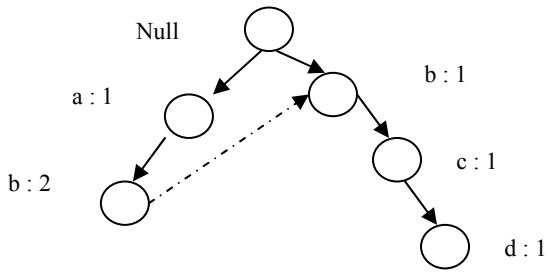
TID	Item
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

Tabel 2.3. Tabel DataTransaksi

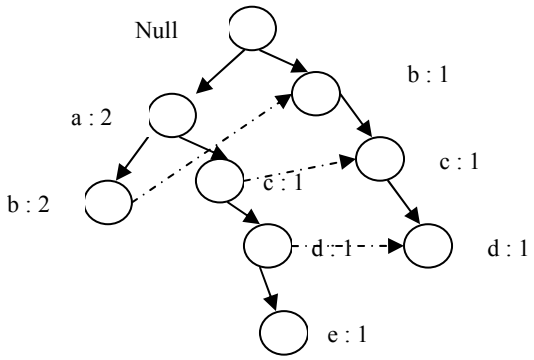
Gambar di bawah ini memberikan ilustrasi mengenai pembentukan FP-tree setelah pembacaan TID 1.



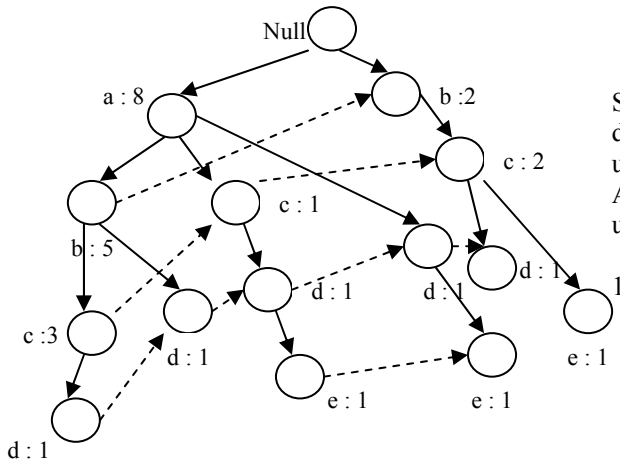
Gambar 2.1 Hasil pembentukan FP-tree setelah pembacaan TID 1



Gambar 2.2 Hasil Pembentukan FP –Tree setelah pembacaan TID 2



Gambar 2.3 Hasil Pembentukan FP-Tree setelah pembacaan TID 3



Gambar 2.4 Hasil Pembentukan FP-Tree setelah pembacaan TID 10

Diberikan 10 data transaksi dengan 5 jenis item seperti pada tabel di atas. Gambar 1 – 4 menunjukkan proses terbentuknya FP –Tree setiap TID dibaca. Setiap simpul pada FP-Tree mengandung nama sebuah item dan *counter support* yang berfungsi untuk menghitung frekuensi kemunculan item tersebut dalam tiap lintasan transaksi.

FP-tree yang merepresentasikan data transaksi pada tabel 2.1 dibentuk dengan cara sebagai berikut:

1. Kumpulan data dipindai pertama kali untuk menentukan *support count* dari setiap *item*. Item yang tidak *frequent* dibuang, sedangkan *frequent item* dimasukkan dan disusun dengan urutan menurun, seperti yang terlihat pada tabel 2.1.
2. Pemindaian kedua, yaitu pembacaan TID pertama {a,b} akan membuat simpul a dan b, sehingga terbentuk lintasan transaksi Null→a→b. *Support count* dari setiap simpul bernilai awal 1
3. Setelah pembacaan transaksi kedua {b,c,d}, terbentuk lintasan kedua yaitu Null→b→c→d. *Support count* masing-masing count juga bernilai awal 1. Walaupun b ada pada transaksi pertama, namun karena *prefix* transaksinya tidak sama, maka transaksi kedua ini tidak bisa dimampatkan dalam satu lintasan.
4. Transaksi keempat memiliki *prefix* transaksi yang sama dengan transaksi pertama, yaitu a, maka lintasan transaksi ketiga dapat ditimpakan di a, sambil menambah *support count* dari a, dan selanjutnya membuat lintasan baru sesuai dengan transaksi ketiga.(lihat gambar 2.3)
5. Proses ini dilanjutkan sampai FP-tree berhasil dibangun berdasarkan tabel data transaksi yang diberikan.

3. PENERAPAN ALGORITMA FP-GROWTH

Setelah tahap pembangunan FP-tree dari sekumpulan data transaksi, akan diterapkan algoritma FP-growth untuk mencari *frequent itemset* yang signifikan. Algoritma FP-growth dibagi menjadi tiga langkah utama, yaitu :

1. Tahap Pembangkitan *Conditional Pattern Base*

   Conditional Pattern Base merupakan subdatabase yang berisi *prefix path* (lintasan prefix) dan *suffix pattern* (pola akhiran). Pembangkitan *conditional pattern base* didapatkan melalui FP-tree yang telah dibangun sebelumnya.

2. Tahap Pembangkitan *Conditional* FP-tree

   Pada tahap ini, *support count* dari setiap item pada setiap *conditional pattern base* dijumlahkan, lalu setiap item yang memiliki jumlah support count lebih besar sama dengan minimum *support count* ξ akan

dibangkitkan dengan *conditional* FP-tree.

3. Tahap Pencarian *frequent itemset*

Apabila Conditional FP-tree merupakan lintasan tunggal (*single path),* maka didapatkan *frequent itemset* dengan melakukan kombinasi item untuk setiap *conditional* FP-tree. Jika bukan lintasan tunggal, maka dilakukan pembangkitan FP-growth secara rekursif.

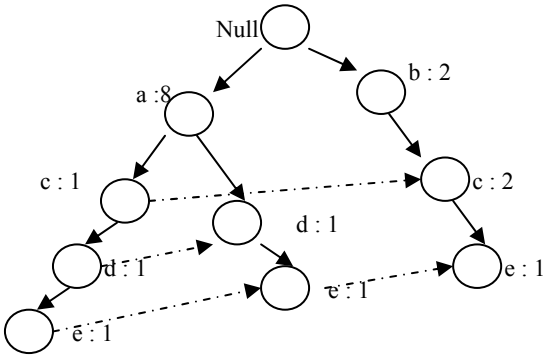
Ketiga tahap tersebut merupakan langkah yang akan dilakukan untuk mendapat *frequent itemset,* yang dapat dilihat pada algoritma berikut :

```
Input: FP-tree Tree
Output: R_t sekumpulan lengkap pola
        frequent
Method: FP-growth(Tree,null)
Procedure: FP-growth(Tree,α)
{
01:if Tree mengandung single path P;
02:then untuk tiap kombinasi (dinotasikan
   β) dari node-node dalam path P do
03:bangkitkan pola β α dengan support =
   minimum support dari       node-node
   dalam β;
04:else untuk tiap a_i dalam header dari
   Tree do {
05:bangkitkan pola
06:bangun β= a_i α dengan
   support = a_i .support
07:if Tree β=Ø
08:then panggil FP-growth(Tree,β) }
}
```

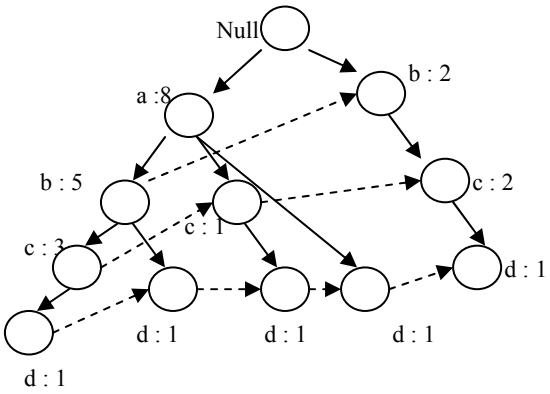
Gambar 3.1 Algoritma FP-Growth

Akan dicoba menerapkan algoritma FP-growth pada kasus contoh 1.

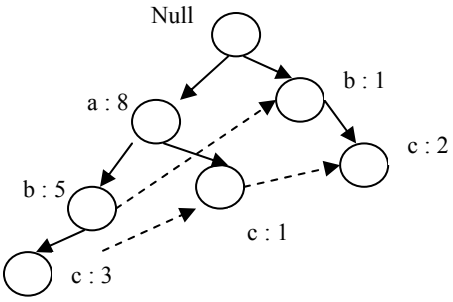
Langkah-langkah yang harus ditempuh akan dijelaskan pada bagian berikut ini.
Untuk menemukan *frequent itemset* dari contoh 1, maka perlu ditentukan upapohon dengan lintasan *y*ang berakhir dengan *support count* terkecil, yaitu e. Berturut-turut ditentukan juga yang berakhir di d,c,b, dan a. Proses pembentukan dapat dilihat pada gambar berikut :



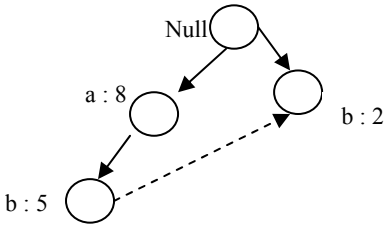
Gambar 3.1 Lintasan yang mengandung simpul e



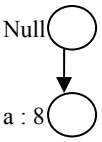
Gambar 3.2 Lintasan mengandung simpul d



Gambar 3.3 Lintasan mengandung simpul c



Gambar 3.4 Lintasan mengandung simpul b



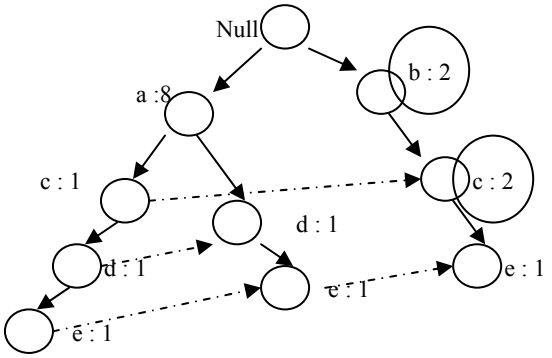
Gambar 3.5 Lintasan mengandung simpul a

Algoritma FP-growth menemukan *frequent itemset* yang berakhiran *suffix* tertentu dengan menggunakan metode *divide and conquer* untuk memecah problem menjadi subproblem yang lebih kecil.

Contohnya, jika kita ingin menemukan semua *frequent itemset* yang berakhiran e. Oleh karena itu, kita harus mengecek apakah *support count* dari e memenuhi minimum *support count* ξ=2. Karena *support count* dari e adalah 3, dan 3≥ ξ, maka e adalah item yang *frequent.*

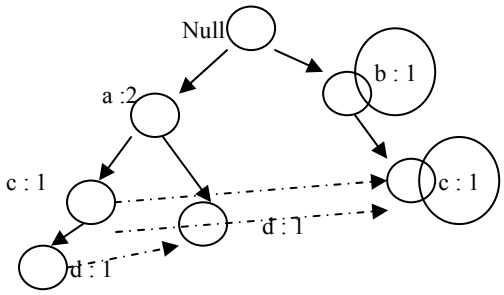
Setelah mengetahui bahwa *item* e adalah *item* yang frequent, maka subproblem selanjutnya adalah menemukan *frequent itemset* dengan akhiran de, ce, be, dan ae. Dengan menggabungkan seluruh solusi dari subproblem yang ada, maka himpunan semua *frequent itemset* yang berakhiran *item* e akan didapatkan.

Untuk lebih memperjelas, dapat dilihat contoh menemukan *frequent itemset* yang berakhiran dengan *item* e di bawah ini



Gambar 3.6 Ada lintasan yang tidak berakhir di e, yaitu Null→b→c

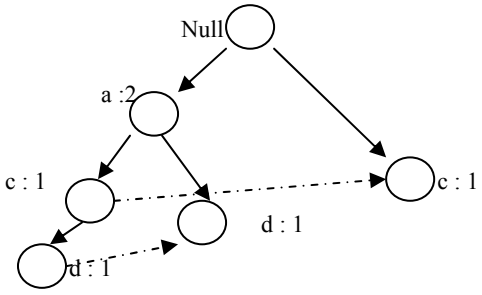
1. Langkah pertama yang dilakukan adalah membangun sebuah upapohon FP-tree dengan hanya menyertakan lintasan yang berakhir di e.
2. *Support count* dari *item* e dihitung dan dibandingkan dengan minimum *support count* ξ*=3*. Karena memenuhi, maka {e} termasuk *frequent itemset*, karena *support count=2.*
3. Karena *item* e *frequent,* maka perlu dipecahkan subproblem untuk menemukan *frequent itemset* yang berkahiran dengan de, ce, be, dan ae. Sebelum memevahkan subproblem ini, maka upapohon FP-tree tersebut harus diubah terlebih dahulu menjadi *conditional* FP-tree. *Conditional* FP-tree mirip dengan FP-tree biasa, namun *conditional FP-tree* dimaksudkan untuk mencari *frequent itemset* yang berakhiran *item* tertentu.
4. *Conditional* FP-tree dapat dibentuk dengan cara :



Gambar 3.7 Semua simpul e dibuang, *Support count* simpul di atasnya sudah diperbaharui

a. Setiap lintasan yang tidak mengandung e dibuang. Pada contoh, lintasan terkanan, terdapat lintasan yang tidak mengandung e, yaitu null→b→c. Lintasan ini dapat dibuang dengan cara mengurangi *support count* menjadi 1, sehingga lintasan tersebut hanya mengandung transaksi {b,c,e}, seperti pada gambar 3.7.

b. Setelah semua lintasan berakhir di e, maka simpul e dapat dibuang, karena setiap nilai *support count* pada simpul orang tuanya telah mencerminkan transaksi yang berakhir di e. Subproblem selanjutnya yang harus dipecahkan adalah mencari lintasan *frequent itemset* yang berakhir di de, ce, be, dan ae.

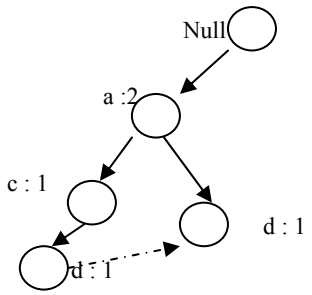


Gambar 3.8 *Conditional FP-tree untuk e* (lintasan mengandung be dihapus, karena tidak *frequent*).

c. Karena nilai *support count* dari b adalah 1, yang berarti transaksi yang mengandung b dan e hanya 1 transaksi, maka berdasarkan prinsip anti-monotone heuristic, simpul b dan lintasan yang mengandung be dapat dibuang, karena jika *item* b tidak frequent, maka setiap transaksi yang berakhiran be juga tidak *frequent.* Terbentuk *Conditional*

FP-tree untuk e, seperti pada gambar 3.8

5. FP-tree menggunakan *Conditional* FP-tree untuk membangun pohon lintasan prefix untuk menemukan *frequent itemset* yang berakhir dengan pasangan *item* de,ce, dan ae.
6. Untuk Lintasa Prefix de, yang dibentuk dari *Conditional FP-tree* untuk *item* e dapat dilihat pada gambar 3.9 berikut



Gambar 3.9 Pohon Prefix yang berakhir di de

7. Dengan menjumlahkan *support count* dari d, yang tidak lain adalah jumlah *frequent itemset* yang berakhir di de, didapat bahwa {d,e} juga termasuk dalam *frequent itemset.*
8. Selanjutnya Algoritma FP-tree akan mengulangi langkah yang sama dengan langkah ketiga, sehingga didapatkan *conditional FP-tree untuk de* hanya berisi satu daun, yaitu a, dengan *support count* 2. Sehingga {a,d,e} termasuk dalam *frequent itemset.*
9. Subproblem berikutnya yaitu dengan menemukan *frequent itemset* yang berakhiran dengan ce. Didapat {c,e} juga merupakan *frequent itemset.* Begitupula dengan {a,e}.

Setelah memeriksa *frequent itemset* untuk beberapa akhiran (*suffix*), maka didapat hasil yang dirangkum dalam tabel berikut:

Suffix	*Frequent Itemset*
e	{e},{d,e},{a,d,e},{c,e},{a,e}
d	{d},{c,d},{b,c,d},{a,c,d},{b,d},{a,b,d},{a,d}
c	{c},{b,c},{a,b,c},{a,c}
b	{b},{a,b}
a	{a}

Tabel 3.1 Hasil *Frequent Itemset*

Dengan metode *divide and conquer* ini, maka pada setiap langkah rekursif, algoritma FP-growth akan membangun sebuah *conditional FP-tree* baru yang telah diperbaharui nilai *support count,* dan membuang lintasan yang mengandung item-item yang tidak *frequent* lagi.

4. KESIMPULAN

Beberapa kesimpulan yang dapat ditarik dari penulisan makalah ini adalah:

1. Struktur pohon yang dipelajari pada mata kuliah matematika diskrit memiliki implementasi yang sangat beragam, khususnya dalam penyimpanan struktur data yang lebih solid, efektif, dan efisien.
2. FP-tree yang terbentuk dapat memampatkan data transaksi yang memilki *item* yang sama, sehingga penggunaan memori komputer lebih sedikit, dan proses pencarian *frequent itemset* menjadi lebih cepat.
3. Dengan menggunakan Algoritma FP Growth, maka pemindaian kumpulan data transaksi hanya dilakukan dua kali, jauh lebih efisien dibandingkan algoritma dengan paradigma apriori.
4. FP-growth merupakan salah satu algoritma yang menjadi dasar perkembangan beberapa algoritma baru yang lebih efektif, karena kelebihannya yaitu tidak melakukan pemindaian data transaksi secara berulang-ulang.

DAFTAR REFERENSI

[1] Borgelt, Christian. " An Implementation of FP-Growth Algorithm" osdm.ua.ac.be/papers/p1-borgelt.pdf Tanggal akses: 29 Desember 2007 Pukul: 18.00 WIB

[2] Han, J., Micheline, K., " Data Mining: Concept and Techniques", *Simon Fraser University, Morgan Kauffman Publisher,* 2000.

[3] Han, J., Yin, Y., "Mining Frequent Pattern Without Candidate Generation" , http://www-faculty.cs.uiuc.edu/~hanj/pdf/sigmod00.pdf Tanggal akses: 27 Desember 2007 Pukul:21.00 WIB

[4] Munir, Rinaldi, "Diktat Kuliah IF2153 Matematika Diskrit", *Departemen Teknik Informatika, Institut Teknologi Bandung,* 2006.

[5] Soelaiman,Rully. Arini, Ni Made. "Analisis Kinerja Algoritma FOLD- Growth dan FP-Growth pada Pengalian Pola Asosiasi", www.si.its.ac.id/Penelitian/JURNAL/ Arin.pdf Tanggal akses: 27 Desember 2007 Pukul 21.00 WIB