Tugas01-Machine Learning
Nama : Muh.Ikhsan
Nim : H071191049

Association Rule Mining

Analisis asosiasi (*Association Rule*) adalah metode machine learning dan data mining digunakan untuk menemukan hubungan yang menarik (*joint value*) item-item yang sering muncul secara bersama-sama dalam suatu database. Aturan ini bisa berupa studi transaksi di supermarket, misalnya seseorang yang membeli susu bayi juga membeli sabun mandi. Di sini berarti susu bayi bersama dengan sabun mandi. Karena awalnya berasal dari studi tentang database transaksi konsumen untuk menentukan kebiasaan suatu produk dibeli bersama produk apa, maka aturan asosiasijuga sering dinamakan *Market Basket Analysis*. Pada *Market Basket Analysis* memungkinkan pengecer untuk mengidentifikasi hubungan item yang sering dibeli orang bersama-sama.

Bentuk umum dari Aturan asosiasi yaitu:

Misalkan $I = I_1, I_2, \cdots, I_m$ merupakan kumpulan item. Sekumpulan item disebut sebagai *itemset. Itemset* yang mengandung k-item disebut *k-itemset.* D merupakan database yang terdiri dari kumpulan transakasi (T), dimana $T \subseteq I$. Setiap transaksi memiliki ID transaksi yang disebut TID.

Sebuah association rule dituliskan dalam bentuk X → Y, dimana $X \subset I$, $Y \subset I$. Association rule berbentuk *"If antecendent then consequent"* (Larose & Larose, 2015). X disebut sebagai Antecedent atau left-hand side (LHS) dan Y disebut Consequent atau right-hand side (RHS).

"If roti dan mentega, maka susu" ("Jika roti dan mentega dibel, maka susu juga akan dibeli pelanggan")

Antecedent : Roti dan Mentega

Consequent : Susu

Itemset : {Roti, Mentega, Susu}

Terdapat dua proses yang dilakukan untuk mendapatkan assosiation rule :

a. Mendapatkan semua frequent itemset

I dikatakan *frequent itemset* apabila nilai support dari item memenuhi nilai *minimum support* (min_sup). Algoritma yang dapat digunakan untuk mencari *frequent itemset* dapat berupa Algoritma Apriori, ataupun FP-Growth.

Support adalah ukuran atau presentasi munculnya sebuah itemset dalam sebuah transaksi. Pertimbangkan itemset_1 = {Roti, Mentega} dan itemset_2 = {Roti, Sampo}. Dalam hal ini transaksi yang memiliki roti dan mentega akan lebih banyak dibandingkan roti dan sampo, sehingga itemset_1 umumnya memiliki *support* yang lebih tinggi daripada itemset_2.

Nilai *support* sebuah item diperoleh dengan rumus berikut:

$$support(A) = \frac{Jumlah\ Transaksi\ Mengandung\ A}{Total\ Transaksi}$$

Sedangkan nilai support dari 2 item diperoleh dari rumus berikut:

$$support(A \cap B) = \frac{Jumlah\ Transaksi\ Mengandung\ A\ dan\ B}{Total\ Transaksi}$$

Jika sebuah *itemset* kebetulan memiliki support yang sangat rendah, kita tidak dapat mengambil informasi yang cukup tentang hubungan antar itemnya dan karenanya tidak ada kesimpulan yang dapat ditarik dari aturan seperti itu.

b. Menghasilkan *association rule* yang kuat dari *frequent itemset*

Setelah semua pola frequensi tinggi ditemukan, barulah dicari aturan assosiatif yang kuat. *Association rule* yang kuat memenuh syarat minimum untuk *confidence* (min_confidence) dengan menghitung confidence(A → B)

Confidence adalah kuatnya hubungan antar item dalam aturan asosiasi. Dapat dikatakan sebagai nilai presentase atau ukuran kemungkinan munculnya *Consecuent* diantara semua grup yang berisi *Antercedent.* Semakin tinggi nilainya, maka semakin besar kemungkinan item *Consecuent* muncul jika diketahui bahwa semua item *Antercedent* muncul dalam itemset tersebut.

Nilai *confidence* dari aturan A → B diperoleh dari rumus berikut :

$$Confidence(A \rightarrow B) = P(B|A) = \frac{Jumlah\ Transaksi\ Mengandung\ A\ dan\ B}{Jumlah\ Transaksi\ mengandung\ A}$$

Sebelum beranjak, pertimbangkan *confidence* untuk {Mentega} → {Roti}, maka dapat diperkirakan nilanya sangat tinggi. Bagaimana dengan {Sikat gigi}→{Susu}? *Confidence* untuk aturan ini akan tinggi juga mengingat {Susu} adalah itemset yang sering muncul pada transaksi lainnya.

Tidak masalah apa yang terdapat pada *antersedent*, *confidence* untuk *association rule* yang memiliki *consecuent* yang sering muncul akan selalu tinggi

Meski suatu aturen memiliki nilai *confidence* yang tinggi, padahal secara intuitif dapat diketahui bahwa aturan ini memiliki asosiasi yang lemah, sehingga terdapat sesuatu yang sedikit menyesatkan tentang nilai kepercayaan yang tinggi ini. Lift diperkenalkan untuk mengatasi tantangan ini.

Lift adalah suatu ukuran untuk mengetahui kekuatan aturan asosisasi (*association rule*) yang telah terbentuk. Nilai *lift ratio* biasanya digunakan sebagai penentu apakah aturan asosiasi valid atau tidak valid. Untuk menghitung lift ratio digunakam rumus sebagai berikut:

$$Lift(A \rightarrow B) = \frac{Confidence(A \rightarrow B)}{Benchmark\ Confidence(A \rightarrow B)}$$

Untuk mendapatkan nilai benchmark confidence sendiri dapat dihitung menggunakan rumus sebagai berikut:

$$Benchmark\ Confidence(A \rightarrow B) = \frac{Nc}{N}$$

dimana :

Nc = Jumlah transaksi dengan item yang menjadi consequent

N = jumlah transaksi basis data

Contoh aturan asosiasi :

{Roti, Mentega} → {Susu} (support = 40%, confidence = 50%)

Association Rules di atas dapat dibaca secara sederhana menjadi : "50% dari transaksi di database yang memuat item roti dan mentega juga memuat item susu. Sedangkan 40% dari seluruh transaksi yang ada di database memuat ketiga item itu." Dapat juga diartikan : "Seorang konsumen yang membeli roti dan mentega punya kemungkinan 50% untuk juga membeli susu. Aturan ini cukup signifikan karena mewakili 40% dari catatan transaksi selama ini.

Market Basket Analysis

Market basket analysis merupakan suatu proses untuk menganalisis kebiasaan pembelian pelanggan dengan menemukan hubungan antara item-item yang dibeli pelanggan dalam keranjang belanjaannya. Informasi yang ditemukan nantinya dapat digunakan untuk mengembangkan strategi pemasaran dalam supermarket sehingga dapat meningkatkan keuntungan yang didapatkan.

Market basket data dapat direpresentasikan dengan format *biner* seperti pada contoh berikut. Dimana setiap baris merupakan data setiap transaksi dan kolom menggambarkan item-itemnya. Format ini biasanya digunakan saat association rule digunakan dalam mengidentifikasi *frequent itemset* (Venkatachari, 2016).

Tabel 1 Binary Database

TID	Roti	Mentega	Susu	Selai
T1	1	1	1	0
T2	1	1	0	1
T3	1	0	0	1
T4	0	1	1	0

Tabel 1 menunjukkan data transaksi. Apabila transaksi mengandung item dalam kolom maka akan diberi nilai 1 namun jika tidak maka akan diberi nilai 0.

Tabel 2 Database Transaksi

TID	**Itemset**
T1	Roti, Mentega, Susu
T2	Roti, Mentega, Selai
T3	Roti, Selai
T4	Mentega, susu

Tabel 2 menunjukkan ID transaksi dan keterangan tentang item apasajakah yang dibeli pada transaksi ID tersebut.

Apriori

Apriori adalah algoritma yang diperkenalkan oleh R. Agrawal dan R. Srikant pada tahun 1994 untuk mencari frequent itemset untuk data *Boolean* pada *association rule* (Han & Kember, 2006). *Apriori* menggunakan metode iteratif dimana k-*itemset* digunakan untuk menentukan (k+1) itemset.

Item I *infrequent* adalah jika:

1) P(I) < *min_support threshold*, maka *infrequent*.
2) P(I+A) < *minimum_support threshold*, maka I+A *infrequent*, dimana A juga termasuk *itemset*.
3) Jika *itemset infrequent* maka semua supersetnya juga akan *infrequent*.

Langkah awal dalam algoritma apriori yang harus dilakukan adalah melakukan *scanning* data untuk menemukan *support* untuk setiap 1-itemset. Hanya item yang memenuhi *minimum support* saja yang akan dilakukan analisis lebih lanjut. Item yang memenuhi *minimum support* ini dinotasikan dengan $L1$. $L1$ akan digunakan untuk menemukan $L2$. $L2$ akan digunakan untuk menemukan $L3$, dan seterusnya sampai tidak ada k-*itemset* yang dapat dibentuk. Untuk menemukan *Lk* dibutuhkan satu kali *full scanning database*.

Langkah-langkah yang diikuti dalam Algoritma Apriori dari data mining adalah:

1. *Join Step:* Langkah ini menghasilkan (k+1) itemset dari K-*itemsets* dengan menggabungkan setiap item dengan dirinya sendiri
2. *Prune Step:* Langkah ini melakukan *scanning* jumlah setiap item di *database.* Jika *candidate item* tidak memenuhi *min_support,* maka item tersebut dianggap *infrequent* dan karenanya akan dihapus. Langkah ini dilakukan untuk memperkeci ukuran *candidate itemsets.*

Cara Kerja Apriori

1. Tentukan *minimum support.*
2. Iterasi 1 : hitung item-item dari *support* (transaksi yang memuat seluruh item) dengan *menscan database* untuk 1-*itemset*, setelah 1-itemset didapatkan, dari 1-*itemset* apakah diatas *minimum support*, apabila telah memenuhi *minimum support*, 1-*itemset* tersebut akan menjadi pola *frequent* tinggi.
3. Iterasi 2 : untuk mendapatkan 2-*itemset*, harus dilakukan kombinasi dari k-*itemset* sebelumnya, kemudian scan *database* lagi untuk hitung item-item yang memuat *support*. itemset yang memenuhi minimum support akan dipilih sebagai pola *frequent* tinggi dari kandidat.
4. Tetapkan nilai k-itemset dari support yang telah memenuhi minimum support dari k-itemset.
5. Lakukan proses untuk iterasi selanjutnya hingga tidak ada lagi k-itemset yang memenuhi minimum support.

Untuk lebih jelasnya, berikut ini adalah contoh penerapan algoritma apriori. Misalkan *support count* yang digunakan adalah 2. Berikut ini adalah *database* transaksi

Tabel 3 Database Transaksi

TID	**Itemset yang dibeli**
T1	A, B, E
T2	B, D
T3	B, C
T4	A, B, D
T5	A, C
T6	B, C
T7	A, C
T8	A, B, C, E
T9	A, B, C

Langkah pertama yaitu scanning *database* dan menghitung jumlah transaksi untuk 1-*itemset*. $C_k = \{A, B, C, D, E\}$.

Berikut ini adalah hasilnya.

Tabel 4 *Support Count 1-itemset*

1-*Itemset*	***Support Count***
A	6
B	7
C	6
D	2
E	2

Karena semua item memenuhi *minimum support* maka $L_1 = \{A, B, C, D, E\}$. Selanjutnya membangkitkan *candidate generation*. C_2 = {AB, AC, AD,AE, BC, BD, BE, CD, CE,DE}. Setelah itu masuk ke tahap *prune* yaitu melakukan scanning kembali apakah subset untuk setiap itemset yang ada pada C_2 merupakan bagian dari L_1. Jika tidak maka *itemset* ini akan dikeluarkan dari C_2. Selanjutnya menghitung *support count* untuk C_2. Berikut adalah hasilnya.

Tabel 5 *Support Count 2-itemset*

2-*Itemset*	***Support Count***
AB	4
AC	4
AD	1
AE	2
BC	4
BD	2
BE	2
CD	0
CE	1
DE	0

Dari tabel di atas dapat dilihat bahwa terdapat tiga *itemset* yang tidak memenuhi minimum support yaitu CD, CE, dan DE sehingga item ini tidak termasuk ke dalam L_2. Jadi L_2 = {AB, AC, AD, AE, BC, BD, BE, CD, CE, DE}. Kembali ke tahap *join step* yaitu membangkitkan candidate generation untuk 3-itemset. Maka didapatkan C_3 = {ABC, ABE, ACE, BCD, BCE, BDE}.

Selanjutnya dilakukan *scanning* kembali pada *database*, ternyata ada *subset* dari C_3 yang bukan merupakan bagian dari L_2, sehingga *itemset* tersebut dikeluarkan dari C_3. Sehingga didapatkan C_3 = {ABC, ABE}. Berikut adalah tabel nilai *support count* untuk C_3.

Tabel 6 *Support Count 3-itemset*

3-*Itemset*	***Support Count***
ABC	2
ABE	2

Ternyata semua data untuk 3-*itemset* ini memenuhi *minimum support*, maka didapatkan L_3 = {ABC, ABE} dan dapat dilanjutkan ke tahap selanjutnya yaitu membangkitkan *candidate generation* untuk 4-*itemset*. Maka C_4 = {ABCE}. Namun setelah dilakukan scanning database ternyata terdapat subset dari itemset ini yang tidak frequent maka C_4 = ∅ sehingga L_4 = ∅. Sehingga proses dihentikan.

FP-Growth

Menggunakan *Apriori* memerlukan beberapa pemindaian database untuk memeriksa jumlah dukungan setiap *item* dan *itemset*. Ketika basis datanya besar, ini akan menghabiskan banyak memori dan daya komputasi. Oleh karena itu dibuatlah algoritma FP-*Growth* untuk mengatasi kekurangan tersebut. Metode ini hanya memindai database dua kali dan menggunakan struktur pohon (FP-*Tree*) untuk menyimpan semua informasi.

Root mewakili *null*, setiap *node* mewakili *item*, sedangkan asosiasi node adalah *itemset* dengan urutan yang dipertahankan saat membentuk pohon.

Pseudocode dan Penjelasan FP-tree :

1. Deduksi item yang sering dipesan. Untuk item dengan frekuensi yang sama, urutannya diberikan menurut urutan abjad.
2. Bangun pohon FP dari data di atas
3. Dari FP-tree di atas, buatlah FP-conditional tree untuk setiap item (atau itemset).
4. Tentukan Frequent Pattern.

Misalkan tabel berikut ini merupakan *database* transaksi suatu supermarket (data asli, belum dilakukan reduksi dimensi) ditampilkan dalam Tabel di bawah dengan nilai *minimum support* yang digunakan adalah 3.

Tabel 7 Database Transaksi Sebelum Reduksi Dimensi

TID	**Item yang dibeli**
1	f,a,c,d,g,i,m,p
2	a,b,c,f,l,m,o
3	b,f,h,j,o
4	b,c,k,s,p
5	a,f,c,e,l,p,m,n

Pertama-tama akan dilakukan scanning database untuk melihat nilai frekuensi setiap item. Frekuensi dari item-item yang ada pada database yaitu <(f:4), (c:4), (a:3), (b:3), (m:3), (p:3), (o:2), (l:2), (d:1), (e:1), (g:1), (h:1), (i:1), (j:1), (k:1), (n:1), (s1)>, dimana angka setelah tanda ":" mengindikasikan nilai *support count*. Setiap item pada tabel diurutkan dengan urutan menurun.

Pola urutan ini penting sebab setiap bagian dari pohon akan mengikuti pola ini. Item dengan nilai support kurang dari min_support yang ditentukan akan dikeluarkan dari pengamatan. Item-item yang memenuhi min_support yaitu u <(f:4), (c:4), (a:3),

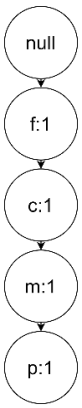
(b:3), (m:3), (p:3)>. Berikut ini adalah kemunculan item yang frequent dalam setiap transaksi yang terbentuk setelah mengeluarkan item yang tidak memenuhi min_support dan diurutkan sesuai frekuensi.

Tabel 8 Database Transaksi Setelah Reduksi Dimensi

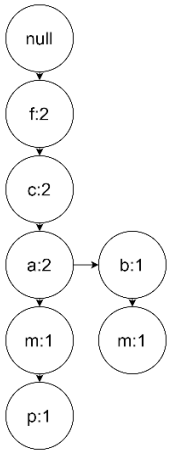
TID	Item yang dibeli
1	f,c,a,m,p
2	f,c,a,b,m
3	f,b
4	c,b,p
5	f,c,a,m,p

Setelah didapatkan daftar transaksi yang hanya berisi item yang frequent, langkah selanjutnya yaitu melakukan scanning yang kedua pada transaksi untuk pembentukan *FP-tree*. Akar dari pohon ini akan diberikan label "null".

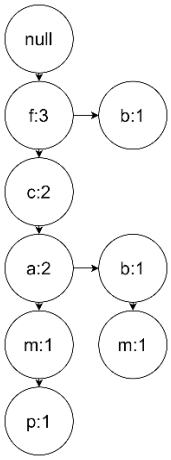
1. Hasil pemindaian transaksi yang pertama digunakan untuk membangun cabang yang pertama pada pohon: {(f:1),(c:1),(a:1),(m:1),(p:1). Frequent item pada transaksi yang ditulis berdasarkan urutan pada list frequent item. Berikut ini adalah gambar yang mengilustrasikan pembentukan FP-tree setelah pemindaian pada TID 1.



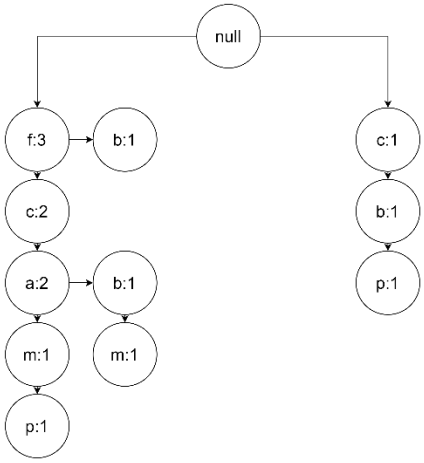
2. Untuk transaksi kedua, karena urutan frequent item <f,c,a,b,m> berbagi awalan <f,c,a> dengan bagian yang sudah ada <f,c,a,m,p> maka count untuk setiap titik sepanjang awalan akan meningkat 1, dan titik baru (b:1) dibentuk dan disambungkan sebagai anak dari (a:2) dan titik baru yang lain (m:1) dibentuk dan disambungkan dengan anak dari (b:1). Gambar berikut mengilustrasikan FP-tree setelah pemindaian TID 2.



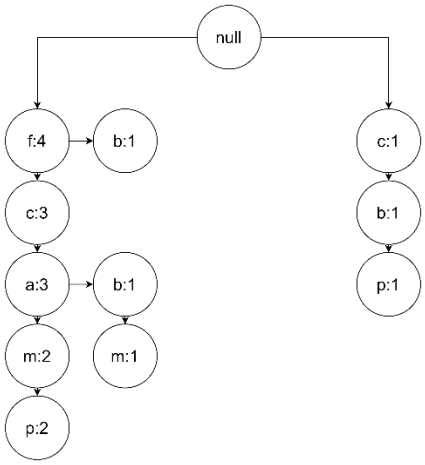
3. Untuk transaksi ketiga, karena frequent item yang tercatat <f,b> hanya berbagi <f> dengan f awalan subtree, maka count f naik 1 dan titik baru (b:1) dibentuk dan disambungkan sebagai anak cabang dari (f:3). Sehingga FP-tree setelah pemindaian TID 3 dapat diilustrasikan dengan gambar sebagai berikut



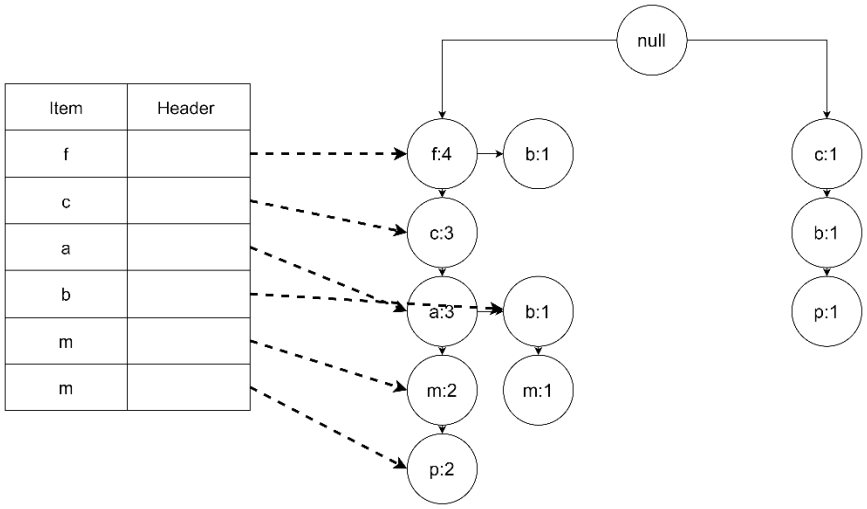
4. Pemindaian pada transaksi keempat membentuk cabang kedua pada pohon <(c:1),(b:1),(p:1). Setelah pemindaian TID 4, ilustrasi FP-tree dapat dilihat pada gambar berikut.



5. Pada transaksi terakhir, karena frequent item <f,c,a,m,p> identik dengan yang pertama maka bagian ini berbagi count pada setiap titik sehingga nilainya bertambah 1. Sehingga FP-tree yang dapat terbentuk setelah pemindaian TID 5 adalah sebagai berikut.



Untuk memfasilitasi pohon tranversal, suatu kepala item pada tabel dibangun dengan setiap poin pada item dihubungkan dengan kejadian pertama yang muncul pada pohon. Titik dengan nama item yang sama akan dihubungkan dengan urutan titik yang tersambung. Setelah melakukan pemindaian pada semua transaksi, maka pohon yang terbentuk dari semua titik-titik yang terhubung tersebut akan terbentuk seperti pada gambar berikut.



Proses penggalian informasi dimulai dari node-link header table yang paling bawah. Berikut ini adalah langkah pembentukan FP-Growth (Samuel, 2007).

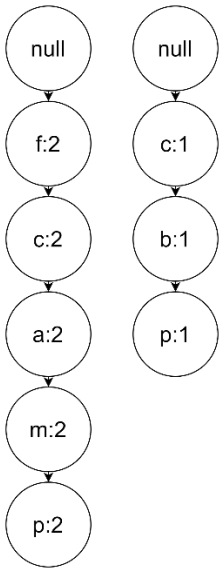
1. Membangun conditional pattern base. Conditional pattern base bisa didapatkan melalui FP-*tree* yang terbentuk. Caranya yaitu dengan melihat FP-*tree* yang berisi akhiran ai . Setiap lintasan yang tidak mengandung ai dibuang. Jadi support count untuk setiap titik adalah nilai support count

untuk titik tersebut yang muncul bersama *ai* . Dalam menyebutkan conditional pattern base, hanya prefix nya saja yang disebutkan.

2. Membangun conditional FP-*tree*. Pada tahap ini support count dari setiap item pada conditional pattern base dijumlahkan. Setiap item yang memiliki support count lebih dari minimal support maka akan dibangkitkan conditional FP-*tree* nya.
3. Pencarian frequent itemset . Jika conditional FP-*tree* merupakan single path, maka untuk mendapatkan frequent itemset dilakukan kombinasi untuk setiap item yang berada pada conditional FP-*tree*. Jika conditional FP-tree bukan merupakan single path maka frequent itemset dibangkitkan secara rekursif.

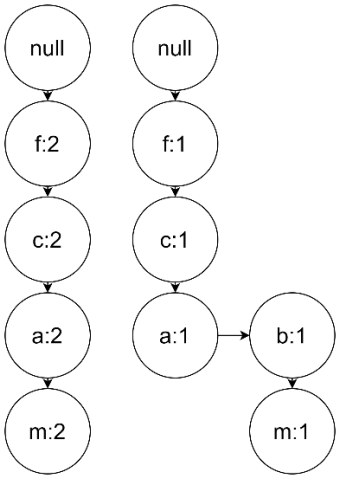
Berikut ini adalah pembentukan FP-Growth untuk contoh sebelumnya.

1. Pembentukan *FP-Growth* untuk *suffix* p
   Berikut ini adalah conditional pattern base untuk suffix p. Dimana untuk semua lintasan yang tidak mengandung p nilai support count sudah dikurangi. Terdapat dua conditional pattern base dengan suffix p yaitu (fcam :2) dan (cb:1). Agar lebih mudah dalam mendapatkan conditional FP-tree untuk suffix p, perhatikan gambar berikut ini.

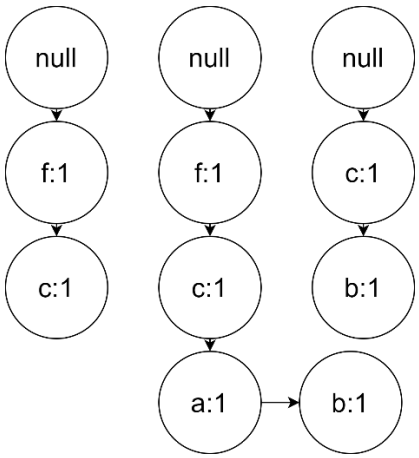


   Berdasarkan Gambar diatas *conditional pattern base* yang memenuhi *minimal support* adalah item c yaitu dengan support *count* 3. Sehingga didapatkan conditional FP-*tree* adalah {(c:3)}|p. Pada suffix p, conditional FP-*tree* yang terbentuk merupakan *single path* sehingga untuk mendapatkan frequent itemset dilakukan dengan mengkombinasikan saja conditional FP-*tree*. Jadi frequent itemset yang terbentuk adalah p, cp.

2. Setelah menghilangkan item yang tidak mengandung dalam lintasan, maka conditional pattern base yang terbentuk adalah (fca:2) dan (fcab:1). Berikut ini adalah gambar untuk conditional pattern yang terbentuk.



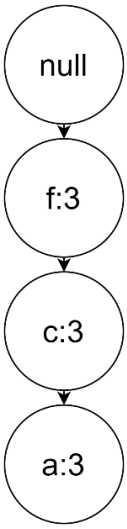
Berdasarkan gambar diatas item yang memiliki *support count* lebih dari atau sama dengan 3 adalah a, c, f. Sehingga *conditional* FP-*tree* yang terbentuk adalah {(f:3, c:3, a:3)}|m. *Conditional* FP-*tree* yang terbentuk merupakan *single path*, sehingga *frequent itemset* didapatkan dengan mencari kombinasinya. *Frequent itemset* yang terbentuk adalah m, fm, cm, am, fcm, fam, cam, fcam.



Pada gambar diatas dapat dilihat bahwa *support count* untuk item f adalah 2 dan untuk c adalah 1. Karena *support count* dari f dan c tidak lebih dari 3 maka untuk suffix b tidak ada *conditional* FP-*tree* yang dibangkitkan. Tidak ada conditional FP-*tree* yang dibangkitkan, sehingga *frequent* itemset yang terbentuk hanya b saja.

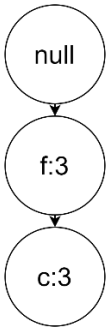
3. Pembentukan FP-*Growth* untuk *suffix* a

Conditional pattern base yang terbentuk untuk *suffix* a hanya ada satu yaitu (fc:3). Berikut ini adalah gambar *conditional pattern* base yang terbentuk untuk *suffix* a.



Conditional pattern base untuk *suffix* a berbentuk *single path* maka semua *item* diatas a pasti memenuhi *minimal support count* karena a memenuhi *minimal support count*. Jadi *conditional* FP-tree yang terbentuk adalah {(f:3, c:3)}|a. Pada suffix b, conditional FP-*tree* yang terbentuk juga merupakan *single path*. Jadi frequent itemset yang terbentuk adalah a, fa, ca, dan fca.

4. Pembentukan FP-*Growth* untuk *suffix* c
   Sama halnya dengan *suffix* a, pada *suffix* c conditional pattern base yang terbentuk hanya ada satu yaitu (f:3). Untuk *suffix* c, *conditional pattern* base yang terbentuk adalah seperti pada gambar berikut.



   Sama halnya dengan *suffix* a, pada *suffix* b *conditional pattern base* juga merupakan single path. Jadi conditional FP-tree yang terbentuk adalah {(f:3)}|c. Frequent itemset yang terbentuk untuk suffix c adalah c dan fc.
5. Pembentukan FP-Growth untuk suffix f conditional pattern base
   Berbeda dengan *suffix* yang lainnya, pada *suffix* f tidak ada conditional *pattern base* yang terbentuk. Karena tidak ada *conditional pattern base* yang terbentuk

maka tidak ada conditional FP-*tree* yang terbentuk dan *fequent itemset* hanya berisi dirinya sendiri yaitu *item* f.

Hasil dari *conditional pattern-bases*, *conditional* FP-*tree* dan *frequent itemset* yang didapatkan dirangkum dalam

Suffix	**Conditional Pattern-base**	**Conditional FP-tree**	**Frequent Itemset**
p	{(f cam:2),(cb:1)}	{(c:3)}|p	p, cp
m	{(f ca:2),(fcab:1)}	{(f:3, c:3, a:3}|m	m, fm, cm, am, fcm, cam,fcam
b	{(f ca:1),(f:1),(c:1)}	∅	b
a	{(fcc:3)}	{(f:3, c:3}|m	a, fa, ca, fca
c	{(f:3)}	{(f:3}|c	c, fc
f	∅	∅	f