# BAB 2 LANDASAN KEPUSTAKAAN

Pada bagian bab ini penulis menjelaskan teori yang digunakan sebagai landasan dalam melakukan penelitian. Teori mengenai algoritma *FIN*, tahapan-tahapan algoritma *FIN* seperti konstruksi *POC-tree.*

## 2.1 Tinjauan Pustaka

Penelitian untuk menemukan *frequent itemset* pada *big data*. Penelitian untuk menganalisis kebiasan konsumen dalam membeli produk. Algoritma *FIN* dengan konsep *Map-Reduce* digunakan untuk melakukan *mining frequent itemset* yang diproses secara pararel untuk meningkatkan performa *mining*. Hasil penelitian menunjukkan performa algoritma *FIN* lebih cepat dari pada algoritma *Apriori* maupun *FP-Growth* (Prakash, 2016).

Penelitian sentimen konsumen terhadap suatu produk berdasarkan *review* yang dikirim konsumen pada situs *website reseller*. Informasi ini dapat digunakan pemilik usaha untuk mengetahui pendapat sebenarnya dari konsumen setelah membeli salah satu produk. Analisis sentimen menganalisis *review* konsumen dengan mengidentifikasi kata yang sering digunakan untuk berbagai *review* yang dikirim konsumen, setelah membeli suatu produk melalui *mining frequent itemset*. Algoritma yang digunakan untuk *mining frequent itemset* adalah *FIN* dan *FP-Growth*. Peneliti membandingkan performa algoritma *FIN* dan *FP-Growth* dalam melakukan *mining frequent itemset*. Hasil dari penelitian adalah algoritma *FIN* menghasilkan polaritas positif dengan konsumsi maksimal memori 37, 94 Mb. Sedangkan algoritma *FP-Growth* juga menghasilkan polaritas positif dengan konsumsi maksimal memori yang lebih tinggi yaitu 43,98 Mb. Berdasarkan hasil penelitian, algoritma *FIN* mengkonsumsi memori lebih rendah dari pada algoritma *FP-Growth* dalam melakukan *mining frequent itemset* (Lobo, 2016).

Penelitian mengimplementasikan algoritma *FIN* untuk *mining frequent itemset* pada *big data* menggunakan *framework Apache Spark* yang diproses secara pararel. Untuk mengetahui efektifitas algoritma *FIN*, peneliti membandingkan dengan algoritma lain yaitu *PFP*. Algoritma *FIN* dalam melakukan *mining frequent itemset* menggunakan struktur data *Nodesets*. *FIN* dapat mendekomposisi permasalahan *frequent itemset* yang berskala besar menjadi unit-unit *task*. Dimana setiap unit *task* dapat dieksekusi secara pararel tanpa *communication overhead* yang tidak diperlukan. Selain itu strategi *hash-based load balancing* diadopsi untuk mengoptimasi penggunaan *resource* dan memaksimalkan *throughput*. Peneliti melakukan uji coba dengan data set uji yang terdiri dari 129 *item* dengan 67.557 transaksi. Hasil penelitian kecepatan eksekusi *FIN* untuk *minimum support* 0,40 adalah 66,67 detik. Sedangkan untuk kecepatan eksekusi *PFP* untuk minimal support 0,40 adalah 198 detik. Hasil perbandingan menunjukkan bahwa algoritma *FIN* memiliki kecepatan eksekusi lebih cepat dari pada algoritma *FPF* (Lin, 2016).

Penelitian tentang menemukan susunan *layout* makanan prasmanan dan menu paket menggunakan metode *Association Rule*. Peneliti dalam melakukan *frequent itemset mining* menggunakan algoritma *Apriori*. Hasil *Association Rule* untuk minimum *support* 30% dan minium *confidence* 62% pada sistem yang telah dibangun memberikan informasi mengenai susunan *layout* makanan prasmanan yang sesuai dengan pola kebiasaan konsumen. Hasil *Association Rule* dengan nilai minimum *support* dan *confidence* juga dapat digunakan untuk membuat rekomendasi menu paket hemat kepada konsumen (Widiati, 2014).

Tabel 2.1 Tinjauan Pustaka

| No | Judul | Objek | Metode | Hasil |
|---|---|---|---|---|
| 1 | *FIN algorithm for generating frequent itemset in big data.* | Melakukan mining *frequent itemset* pada *big data*. | Algoritma *FIN* | Algoritma *FIN* digunakan untuk melakukan proses pararel pada fungsi *hadoop Map-Reduce.* Menunjukkan efisiensi eksekusi waktu dan penggunaan *space* pada *big data*. |
| 2 | *Sentiment Analysis using FP-Growth and FIN algorithm* | Menganalisis sentimen konsumen berdasarkan review konsumen pada suatu produk melalui *mining frequent itemset*. | Algoritma *FIN* dan Algoritma *FP-Growth* | Algoritma *FIN* membutuhkan konsumsi memori yang lebih rendah dari pada algoritma *FP-Growth*. |
| 3 | *PFIN: A Parallel Frequent Itemset Mining Algorithm Using Nodesets.* | Menemukan *freqent itemset* pada *database big data* menggunakan algoritma *FIN*, yang diproses secara *pararel* menggunakan proses distribusi memori pada *framework Spark*. | Algoritma *FIN* | Penelitian dibandingkan antara menemukan *frequent itemset* menggunakan algoritma *FIN* dan algoritma pararel lain yaitu *PFP*. Menunjukkan algoritma *FIN* lebih kompetitif dalam performa *scalability* dan kecepatan eksekusi. |
| 4 | Implementasi *Association Rule* terhadap penyusunan layout makanan dan penentuan paket makanan hemat di RM Roso Echo dengan algoritma *Apriori*. | Menemukan susunan layout makanan prasmanan dan menu paket menggunakan metode *association rule*. | Algoritma *Apriori* | Hasil *association rule* untuk *minSup* 30% dan *minConf* 62% pada sistem yang telah dibangun memberikan informasi mengenai susunan *layout* makanan prasmanan yang sesuai dengan pola kebiasaan konsumen serta dapat memberikan rekomendasi menu paket hemat kepada konsumen. |

| | | | | |
|---|---|---|---|---|
| 5 | Implementasi *Association Rule Mining* untuk Menentukan Menu Paket Makanan dengan Algoritma *FIN* menggunakan *NodeSets* (Studi Kasus R.M. Lesehan Nova Sragen) | Menentukan kombinasi makanan dan minuman pada menu paket yang representatif dengan pilihan pengunjung. | Algortima *FIN* | Pada nilai minimum *support* = 11, telah menghasilkan variasi menu paket yang proporsional, serta telah representatif dengan pilihan konsumen. |

## 2.2 Menu

Daftar *item* makanan dan minuman yang dapat dipilih oleh *customer* pada saat memesan *item-item* di rumah makan. Menu sering disebut daftar menu baik oleh *customer* maupun pengelola rumah makan. Daftar menu dapat berupa buku, *pamflet*, maupun lembaran (Spang, 2000). Untuk melindungi dari minuman yang tumpah dan noda makanan, menu dilaminating dengan bahan plastik. Menu digunakan oleh tipe rumah makan yang baru membuatkan masakan dan minuman apabila *customer* memesan *item* masakan tertentu. Menu adalah bentuk kesepakatan secara tertulis antara *customer* dan pengelola rumah makan mulai dari harga *item* yang tidak dapat ditawar dan *item* masakan yang di hidangkan harus sesuai dengan yang tertera di menu (Spang, 2000).

Menu yang disediakan oleh R.M. Lesehan Nova Sragen terdiri dari daftar *item* makanan yang berjumlah 156 *item* dan daftar *item* minuman yang berjumlah 44 *item*. Menu dibuat dengan bentuk buku terdiri dari 8 halaman yang dilaminating dengan bahan plastik. Bagian luar buku menu dilindungi dengan *cover* berbahan *board* karton yang membuat buku menu bersifat kaku dan lebih tahan lama.

## 2.3 Menu Paket

Daftar menu yang setiap pilihan *item*nya adalah kombinasi antara *item* makanan dan *item* minuman dengan nominal harga untuk setiap *item* adalah penjumlahan nominal harga minuman dan makanan (Gloria, 2007). Pada daftar menu paket memiliki jumlah *item* pilihan lebih sedikit dari pada daftar menu utama. Tujuan utama menu paket adalah memudahkan *customer* untuk memilih *item* pesanan dengan kombinasi *item* makanan dan minuman serta harga total yang telah ditetapkan sebelumnya oleh pihak pengelola rumah makan. Sasaran utama menu paket adalah rombongan *customer* (Gloria, 2007). Menu paket memudahkan setiap anggota rombongan untuk memilih kombinasi *item* makanan dan minuman dengan waktu relatif lebih cepat dari pada memilih *item* makanan dan minuman pada menu utama, yang memiliki variasi pilihan *item* yang lebih banyak. Menu paket juga memiliki tujuan lain yaitu untuk menurunkan tingkat keanekaragaman *item* menu pesanan. Apabila tingkat keanekaragaman *item* menu pesanan rendah maka proses pembuatan sampai menghidangkan masakan akan lebih cepat, dibandingkan apabila tingkat keanekaragaman *item* menu pesanan yang tinggi (Gloria, 2007).

Menu Paket yang sudah ada di R.M. Lesehan Nova Sragen terdiri dari daftar paket berjumlah 30 *item* paket. Menu paket dibuat berdasarkan bahan baku makanan seperti ayam, bebek, ikan lele, ikan nila, dan ikan gurami. Bahan-bahan tersebut diproses dengan cara digoreng, dimasak maupun dibakar dikombinasikan dengan minuman es teh maupun es buah. Menu Paket dicetak dalam bentuk pamflet yang dilaminating dengan plastik, untuk menjaga menu paket dapat bertahan lama.

### 2.4 *Data Mining*

Teknologi yang digunakan oleh pihak pengelola usaha untuk menemukan perilaku konsumen dalam membeli produk usaha. *Data mining* dapat berupa program yang digunakan pengelola usaha untuk menganalisis dan menggali data yang disimpan pada *database* (Kumar, 2006). *Data mining* digunakan untuk menemukan *trend* sebagai peluang usaha di masa depan. *Data mining* juga dapat mengatasi kesalahan pakar pada bidang bisnis, disebabkan pakar tidak mengetahui tentang hal atau kejadian tertentu pada bisnis, dengan cara memprediksi informasi berdasarkan pola yang tersembunyi pada *database*. Secara garis besar *data mining* mencari informasi yang bernilai pada *database*. *Data mining* terdiri dari Klasifikasi, Klastering dan *Association Rule*.

### 2.5 *Association Rule*

*Association Rule* berguna untuk menemukan relasi-relasi yang menarik yang tersembunyi pada kumpulan data yang besar. Relasi-relasi yang d*item*ukan dapat ditampilkan dalam bentuk *association rule* atau *set of frequent items* (Steinbach, 2006). *Rule* menandakan terdapat hubungan kuat antara penjualan produk *A* dengan penjualan produk *B* karena *customer* yang membeli produk *A* juga membeli produk *B*. Pelaku usaha dapat menggunakan *rule* ini untuk mendapatkan peluang untuk menjual secara bersamaan produk-produk kepada *customer*.

*Association Rule* dapat digunakan pada *Market Basket Data* yang ditampilkan pada format bineri. Baris menunjukkan transaksi, kolom menunjukkan *item*. Diketahui $A=\{a_1,a_2,...,a_n\}$, *A* adalah himpunan semua *item* yang ada pada *Market Basket Data*, dan $W=(w_1,w_2,...,w_n)$, *W* adalah himpunan untuk semua transaksi. Untuk setiap transaksi *W* terdapat *item* yang menjadi bagian dari himpunan *A*. *Item*set memiliki jumlah *support* yang menunjukkan jumlah transaksi yang terdapat pada setiap *itemset* (Steinbach, 2006).

*Association rule* adalah ekspresi implikasi dengan bentuk $X \rightarrow Y$, *X* dan *Y* adalah *itemset* yang tidak beririsan, $X \cap Y = \emptyset$. Kekuatan *association rule* dapat diukur menggunakan *support* dan *confidence* yang dimiliki *association rule* (Agarwall, 1998). *Support* menentukan seberapa sering rule di aplikasikan pada himpunan data, sedangkan *confidence* menentukan berapa frekuensi *item Y* terlibat transaksi dengan *X*.

***Support***

$$s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N} \quad (2.1)$$

$s(X \rightarrow Y)$ = Nilai *support* asosiasi *item X* terhadap *item Y*.

$\sigma(X \cup Y)$ = jumlah *item X* dan *item Y* bersamaan ada pada 1 transaksi untuk seluruh data transaksi di *database*.

N = jumlah seluruh transaksi pada *database*.

***Confidence***

$$c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \quad (2.2)$$

$c(X \rightarrow Y)$ = Nilai *confidence* asosiasi *item* X terhadap *item* Y.

$\sigma(X \cup Y)$ = jumlah *item* X dan *item* Y bersamaan ada pada 1 transaksi untuk seluruh data transaksi di *database*.

$\sigma(X)$ = jumlah keseluruhan *item* X pada *database*.

Penghitungan *support* diperlukan karena rule yang memiliki nilai *support* yang rendah dapat terlihat hanya dengan perubahan sederhana dan tidak berguna dari perspektif bisnis ,karena tidak menguntungkan apabila mempromosikan *item-item* kepada *customer* yang jarang membelinya bersamaan (Navathe, 2005). *Confidence* menghitung kebenaran inferensi yang dibuat oleh *rule*. Pada rule *X* → *Y*, nilai *confidence* yang lebih tinggi akan menghasilkan kemungkinan *Y*, tampil lebih banyak pada transaksi yang mengandung *X*. *Confidence* juga memberikan estimasi probabilitas bersyarat untuk *Y* jika *X* ada.

*Pada Association Rule Mining* terdapat contoh permasalahan seperti berikut: Diberikan himpunan transaksi *T*, temukan semua *rule* yang memiliki *support* ≥ *minsup* dan *confidence* ≥ *minconf*, *minsup* adalah *threshold support* dan *minconf* adalah *threshold confidence*. Untuk menyelesaikan permasalahan tersebut terdapat berbagai metode atau algoritma seperti *Brute Force*, *Apriori*, *Eclat* dan *FP-Growth*. Algoritma *Apriori* dan *FP-Growth* terus berkembang memunculkan algoritma-algoritma baru yang menyempurnakan dari kedua algoritma tersebut. Algoritma *Improve Apriori*, *AprioriDp* adalah perkembangan dari algoritma *Apriori*. Sedangakan penyempurnaan algoritma *FP-Growth* seperti Algoritma *DynFp-Growth*, *FP-Bonsai*, *AFOPT*, *NONORDFP*, *FP-Growth**, *PPV*, *PrePost*, dan *FIN* (Kumar, 2006). Berbagai macam algoritma untuk menyelesaikan permasalahan diatas terdapat persamaan proses yaitu pada tahap: Pertama, *Frequent Itemset Generation.* Kedua, *Rule Generation*.

*Frequent Itemset Generation* adalah tahap untuk menemukan semua himpunan *item* yang memenuhi *threshold minsup*. *Itemset* yang dibentuk disebut sebagai *Frequent Itemset*. Sedangkan *Rule Generation* adalah untuk mengekstraksi semua rule yang memiliki nilai *confidence* tinggi dari himpunan *frequent itemset* yang telah d*item*ukan pada tahap sebelumnya. *Rule-rule* yang didapatkan pada proses ini disebut *strong rule* (Tan, 2006).

### 2.6 Algoritma *FIN*

Algoritma *FIN* adalah algoritma untuk menemukan *frequent itemset* menggunakan struktur data *novel* berupa *nodesets.* Untuk menghindari pengulangan pencarian, algoritma *FIN* menggunakan strategi *pruning* disebut sebagai *promotion.* Memiliki persamaan dengan *Children-Parent Equivalence pruning* untuk mengurangi ruang pencarian (Deng, 2014). Tugas utama dari *mining frequent itemset* adalah menemukan semua *itemset* yang nilai *support*-nya kurang dari *minsup* dikalikan jumlah transaksi (Long, 2014).

Algoritma *FIN* termasuk algoritma yang berdasar pada *Pattern Growth* karena tidak membentuk *candidate itemset*, tetapi mengkonstruksi struktur kompleks yang mengandung informasi *frequent itemset* didalam *database* (Yin, 2000). Hasil perkembangan dari algoritma *FP-Growth*, *PPV*, dan *Prepost*. Algoritma *FIN* menggunkan struktur data *Nodeset* yaitu mempresentasikan *itemset* hanya dengan angka *pre-orde*. Memperbaiki kedua algoritma sebelumnya yaitu *PPV* dan *Prepost*, yang menggunakan struktur data *Node-List* dan *N-List* yang mengharuskan menggunakan angka *pre-order* dan *post-order* secara bersamaan. Menyebabkan algoritma tersebut membutuhkan konsumsi memori yang lebih tinggi apabila dibandingkan dengan algoritma *FIN* (Deng, 2014). Algoritma *FIN* menggunakan *POC-tree* untuk mengidentifikasi *frequent k-itemset*.

Untuk mengimplementasikan *Association Rule Mining* menggunakan algoritma *FIN*, maka diperlukan tahapan sebagai berikut:

1. Mengkonstruksi *POC-tree*.
2. Mengidentifikasi semua *frequent 1-itemsets*.
3. Memindai *POC-tree* untuk mendapatkan semua *frequent 2-itemsets* dan semua *Nodesets*.
4. Untuk meningkatkan efisiensi *mining frequent itemsets*, algoritma *FIN* mengadopsi *promotion* sebagai strategi *pruning*.

#### 2.6.1 *POC-tree (Pre-Order Coding tree)*

Struktur dari *POC-tree* adalah pertama terdiri dari satu *root* dengan label "*null*", dan himpunan *item prefix tree* sebagai anak. kedua setiap *node* pada *item prefix subtree* terdiri dari 4 bagian yaitu *item-name*, *count*, *children-list*, *pre-order*. *Item-name* adalah meregistrasi nama *item* pada *node*, *count* adalah jumlah transaksi berdasarkan *path* mencapai suatu *node*, *children-list* adalah semua anak pada *node*, *pre-order* adalah tingkat *pre-order* pada *node* (Deng, 2014).

Langkah-langkah untuk membangun *POC-tree*:

1. Memindai *database* berdasarkan *minSup* satu kali untuk mendapatkan himpunan *frequent 1-itemset*(F1) dan nilai *support*-nya.
2. Menyusun F1 berdasarkan urutan nilai *support* terbesar sampai terkecil. Kemudian menyimpan urutan pada L1.
3. Membuat *root* untuk *POC-tree* dengan label "*null*".
4. Untuk setiap transaksi pada *database* lakukan hal berikut:
   a) Hapus *item-item* yang tidak *infrequent* dari setiap transaksi.
   b) Urutkan *item-item* pada transaksi sesuai dengan urutan L1.
   c) Jadikan hasil *item-item* terurutkan menjadi bentuk [*p* | *P*], *p* adalah elemen pertama dan *P* adalah daftar sisa.
   d) Panggil method *insert tree*([*p* | *P*], *root*), dengan parameter [*p* | *P*] dan *root*.
5. Proses didalam method *insert tree* sebagai berikut:

   Jika *root* memiliki anak N dan N.*item-name* = p.*item-name* maka *count* pada N bertambah 1.

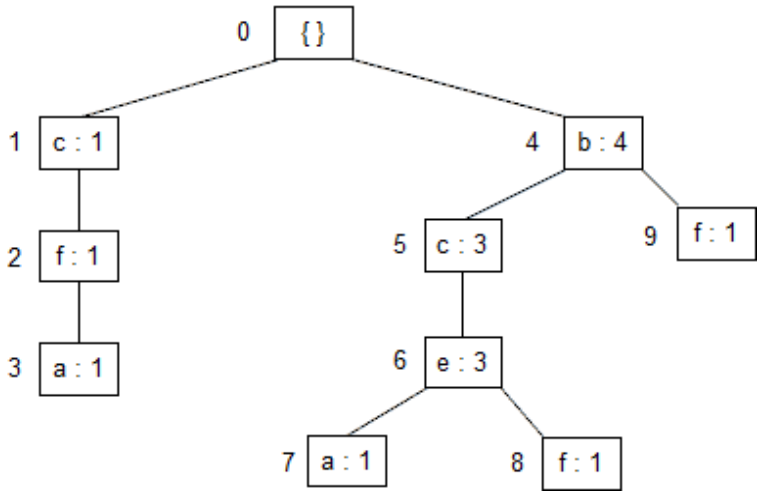
   Jika tidak, buat *node* baru, yang nilai *count* untuk *node* tersebut diinisalisasi dengan 1, dan tambahkan *node* tersebut sebagai anak dari *root*.

   Jika nilai *P* tidak kosong maka panggil method *insert tree*(P, N) secara rekursif, *N* adalah Node anak.
6. Pindai *POC-tree* untuk membuat *pre-order* untuk setiap *node* dengan *pre-order traversal*.

Tabel 2.2 Transaksi pada *database*

| **ID** | ***Item*** | ***Frequent Item* Terurut** |
|---|---|---|
| 1 | a , c, g, f | c, f, a |
| 2 | e , a, c, b | b, c, e, a |
| 3 | e , c, b , i | b, c, e |
| 4 | b , f, h | b, f |
| 5 | b , f, e, c, d | b, c, e, f |

Tabel 2.2 sebagai ilustrasi transaksi yang tersimpan pada *database*. Data tersebut digunakan untuk membuat *POC-tree* dengan langkah-langkah yang dijelaskan sebelumnya menghasilkan *POC-tree* sebagai berikut:




Gambar 2.1 Hasil konstruksi *POC-tree*

Penjelasan untuk simbol pada *POC-tree* pada Gambar 2.1 sebagai berikut:

1. Huruf *a* sampai *f* yang berada didalam kotak adalah *item-name* dari *node*.
2. Nilai angka yang berada setelah tanda (:) didalam kotak adalah *count register* untuk setiap *node*.
3. Nilai angka yang berada diluar kotak sebelah kiri adalah *pre-order* untuk setiap *node*.

### 2.6.2 *NodeSet*

*Nodeset* adalah suatu struktur data *novel* yang memiliki properti *N-info*. *N-info* untuk suatu *node N* pada *POC-tree* adalah *pre-order* dan *count register*, dengan bentuk (*pre-order*, *count register*) untuk *N-info* pada setiap *node N*. Terdapat jenis *Nodeset* berdasarkan jumlah *k-itemset*.*Nodeset* dengan *1-itemset*, *Nodeset* dengan *2-itemset*, dan *Nodeset* dengan *3-itemset* (Deng, 2014).

Untuk membangun *Nodeset* dengan *1-itemset*. Digunakan hukum pertama *Nodeset* yaitu untuk *frequent item i*, dengan bentuk *Nodeset* adalah {($pre_1$, $c_1$), ($pre_2$, $c_2$), ...,($pre_m$, $c_m$)}. Maka nilai *support* sama dengan $c_1 + c_2 + ... + c_m$. Simbol $pre_n$ adalah *prepost* ke-N, dan $c_n$ adalah *count register* ke-N (Deng, 2014). Berdasarkan hukum pertama *Nodeset*, penulis mendapatkan nilai *Nodeset f* pada *POC-tree* Gambar 2.1 adalah {(2,1), (8,1), (9,1)} dengan nilai *support f* = 3. Didapatkan dari *support(f)* = $c_1 + c_2 + c_3 = 1 + 1 + 1 = 3$. *Nodeset* dengan *1-itemset* digunakan untuk mencari *Nodeset* dengan *2-itemset*.

Untuk membuat *Nodeset* dengan 2-*itemset* diperlukan 2 *node* yang memiliki *ancestor* yang sama pada *POC-tree*. *Nodeset* dengan 2-*itemset* $i_1i_2$, disimbolkan sebagai *Nodeset*($i_1$,$i_2$). Berikut adalah hasil dari Nodeset 1-*itemset* untuk semua *node* untuk *POC-tree* Gambar 2.1:

Tabel 2.3 Hasil Nodeset 1-*item*set

| No | *Nodeset 1-itemset* | Anggota *Nodeset 1-itemset* |
|---|---|---|
| 1 | b | {(4, 4)} |
| 2 | c | {(1, 1), (5, 3)} |
| 3 | e | {(6, 3)} |
| 4 | f | {(2, 1), (8, 1), (9, 1)} |
| 5 | a | {(3, 1), (7, 1)} |

Berdasarkan daftar *Nodeset* dengan *1-itemset* pada Tabel 2.2, untuk *Nodeset 1-itemset f* adalah {(2,1), (8,1), (9,1)}. Untuk membuat *Nodeset* dengan *2-itemset*, misal *Nodeset bf* maka penulis melakukan seperti berikut:

1. Mengecek seluruh anggota *Nodeset f*.
2. Pengecekan dimulai dari anggota *Nodeset f* (2,1). Anggota *Nodeset f* (2,1) memiliki *ancestor c*, sehingga tidak sesuai apabila ingin menyusun *Nodeset 2-itemset bf*. Disebabkan apabila ingin membentuk *Nodeset 2-itemset bf* diperlukan *ancestor* b.
3. Pengecekan dilanjutkan pada anggota *Nodeset f* (8,1) dan anggota *Nodeset f* (9,1). Kedua anggota *Nodeset f* yaitu (8,1) dan (9,1) memiliki *ancestor* sama yaitu b. Sehingga dapat dimasukkan ke dalam *Nodeset 2-itemset bf,* karena sesuai dengan *ancestor*-nya. Nilai *Nodeset bf* = {(8,1), (9,1)}.

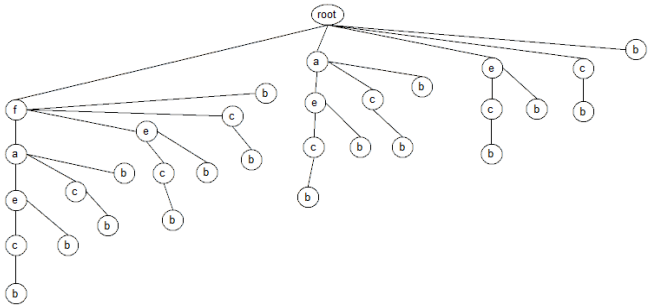
*Untuk dapat membuat Nodeset* dengan *2-itemset* maka harus memenuhi hukum kedua *Nodeset* yaitu apabila himpunan P = ($p_1p_2$, $p_1$, $p_2$ |Ɛ $L_1 \wedge p_1$ berelasi dengan $p_2$) maka anggota himpunan P dapat untuk menyusun *nodeset* dengan *2-itemset*. Nodeset dengan 2-*item*set dalam bentuk {($pre_1$,$c_1$), ($pre_2$,$c_2$), ..., ($pre_m$,$c_m$)}, maka nilai *support* P sama dengan $c_1 + c_2 + ... + c_m$ (Deng, 2014). Simbol $pre_n$ adalah *preorder* dan simbol $c_n$ adalah *count register*. Berdasarkan hukum *Nodeset* kedua maka nilai *support* untuk *Nodeset* dengan 2-*itemset bf*{(8,1), (9,1)} = 2. Didapatkan dari *support(bf)* = 1 + 1 = 2.

*Nodeset 3-itemset* adalah *Nodeset* yang memiliki *itemset* lebih dari 2. Terdapat himpunan P=$p_1p_2p_3..p_k$ dengan $p_j \in L_1$. Diasumsikan $Nodeset_{P1}$ adalah $P_1$=$p_1p_3...p_k$ dan $Nodeset_{P2}$ adalah $P_2$=$p_2p_3...p_k$. *Nodeset* untuk P disimbolkan sebagai $Nodeset_P$. Sehingga $Nodeset_P$ = $Nodeset_{P1} \cap Nodeset_{P2}$ (Deng, 2014). Sebagai contoh, menggunakan *Nodeset 2-itemset* untuk membangun *Nodeset 3-itemset*. *Nodeset 2-itemset* tersebut adalah *Nodeset bf* = {(8,1),(9,1)} dan *Nodeset cf* = {(2,1),(8,1)}. Berdasarkan aturan *Nodeset 3-itemset* maka dapat dibuat *Nodeset bcf* adalah {(8,1),(9,1)} ∩ {(2,1),(8,1)} = {(8,1)}. *Nodeset 3-itemset*

tetap menggunakan hukum *Nodeset* Ketiga yaitu *Nodeset P* dengan bentuk {($pre_1$,$c_1$), ($pre_2$,$c_2$), ...,($pre_3$,$c_3$)}, yang memiliki *support P* sama dengan $c_1 + c_2 + ... + c_M$.

### 2.6.3 *Set-enumeration tree*

*Set-enumeration tree* digunakan untuk merepresentasikan ruang pencarian, yang digunakan algoritma *FIN* untuk meningkatkan efisiensi *mining frequent itemset*. Dimisalkan himpunan *item I = {$i_1$,$i_2$,,...,$i_m$}* . Untuk mengkonstruksi *set-enumeration tree* dibutuhkan tahapan seperti berikut. Pertama, dibuat *root* untuk *tree*. Kedua, membuat *node child m* dari *root* yang diregistrasi dengan $i_{JS}$, direpresentasikan sebagai *m 1-itemsets*. Ketiga, membuat *node child* (m-$j_S$) merepresentasikan *itemsets*{$i_{JS+1}i_{JS}i_{JS-1}...i_{J1}$},{$i_{JS+2}i_{JS}i_{JS-1}...i_{J1}$},...,{$i_Mi_{JS}i_{JS-1}...i_{J1}$} dan meregistrasi $i_{JS+1}$,$i_{JS+2}$,...,$i_M$. *Set-enumeration tree* dibuat dengan mengeksekusi tahap ketiga secara berulang-ulang sampai semua daun *node* terbentuk. Hukum pertama *set-enumeration tree* yaitu apabila *item i* dan *itemset* $P(i \notin P)$ , jika nilai *support P* sama dengan nilai *support* $P \cup (i)$ , dan nilai *support* $A \cup P$ , dimana $A \cap P = \emptyset \wedge i \notin A$ , sama dengan nilai *support* $A \cup P \cup (i)$ (Deng, 2014).




Gambar 2.2 *set-enumeration tree*

Berdasarkan Gambar *set-enumeration tree* pada Gambar 2.2, direpresentasikan *itemset {bceaf}* dan *registrasi item b* pada bagian *node* di bagian kiri bawah *set-enumeration tree*. Algoritma *FIN* menggunakan hukum 1 *set-enumeration tree* untuk mempersempit ruang pencarian. Untuk melakukan proses *pruning* maka sebagai contoh pada Gambar 2.2 nilai *support af* sama dengan nilai *support eaf* pada *subtree*, yang *root*-nya memiliki *node* yang di-*registrasi e* dan direpresentasikan dengan *eaf* pada bagian kiri pada Gambar 2.2.