IDENTIFIKASI PENGKATEGORIAN *MUSHROOM* MENGGUNAKAN ALGORITMA *APRIORI* DENGAN ALGORITMA *PERFECT HASHING AND PRUNING (PHP)*

SKRIPSI

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

Oleh:
TRIANA FARADILA PUTRI
0610960062-96

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

2011


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

IDENTIFIKASI PENGKATEGORIAN *MUSHROOM* MENGGUNAKAN ALGORITMA *APRIORI* DENGAN ALGORITMA *PERFECT HASHING AND PRUNING (PHP)*

Oleh:
TRIANA FARADILA PUTRI
0610960062-96

Setelah dipertahankan di depan Majelis Penguji
Pada tanggal 27 Juni 2011
dan dinyatakan memenuhi syarat untuk memperoleh gelar Sarjana dalam bidang Ilmu Komputer

Pembimbing I	**Pembimbing II**
Drs. M. Arif Rahman, M.Kom	**Edy Santoso,SSi., M.Kom**
NIP. 196604231991111001	**NIP. 197404142003121004**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf Alghofari, M.Sc
NIP. 196709071992031001


UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini:

Nama	**: Triana Faradila Putri**
Nim	**: 0610960062-96**
Jurusan	**: Matematika**
Penulis Skripsi berjudul	**: Identifikasi Pengkategorian *Mushroom* Menggunakan Algoritma *Apriori* Dengan Algortima *Perfect Hashing and Pruning (PHP)***

Dengan ini menyatakan bahwa:

1. **Isi dari Skripsi yang saya buat adalah benar-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 di kemudian hari ternyata Skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 27 Juni 2011
Yang menyatakan,

Triana Faradila Putri
NIM. 0610960062

IDENTIFIKASI PENGKATEGORIAN *MUSHROOM* MENGGUNAKAN ALGORITMA *APRIORI* DENGAN ALGORITMA *PERFECT HASHING AND PRUNING (PHP)*

ABSTRAK

Dalam kehidupan sehari-hari masyarakat umum belum bisa membedakan *mushroom* yang bisa dikonsumsi dan tidak. Sehingga dibutuhkan suatu sistem untuk pengkategorian *mushroom* dengan menemukan *association rule* sesuai dengan nilai minimum *confidence* dan nilai minimum *support*. Pada penilitian ini, untuk menemukan hubungan antara karakteristik-karakteristik (*association rule*) pada *mushroom*, digunakan dua algoritma, yakni algoritma *Apriori* dan algoritma *Perfect Hashing and Pruning (PHP)*. Hasil dari penelitian ini yaitu menghitung jumlah *rule* dan *lift ratio rule*. Hasil yang didapat dalam penelitian ini yaitu algoritma *Apriori* menghasilkan *rule* yang memiliki manfaat sebanyak 55 dari 96 *rule* (57.29%) sedangkan dengan menggunakan algoritma *Perfect Hashing and Pruning (PHP)* sebanyak 19 dari 38 *rule* (50%).

Kata kunci: *Mushroom*, *Perfect Hashing and Pruning(PHP)*, *Apriori*, *Association rule*.


UNIVERSITAS BRAWIJAYA

IDENTIFICATION OF MUSHROOM CATEGORIZATION USING APRIORI ALGORITHM WITH PERFECT HASHING AND PRUNING (PHP) ALGORITHM

ABSTRACT

In the daily life of the general public can not distinguish the mushrooms can be consumed and no. So it takes a system for categorizing mushrooms with finding association rules in accordance with the minimum confidence and minimum support values. In this study, to find a relationship between the characteristics (association rule) on the mushroom, used two algorithms, the Apriori algorithm and the algorithm of Perfect Hashing and Pruning (PHP). The results of this research is to calculate the amount of lift ratio rule and rule. The results obtained in this research that Apriori algorithm produces the rule is as beneficial as rule 55 of the 96 (57.29%) while using the algorithm of Perfect Hashing and Pruning (PHP) of rule 19 of 38 (50%).

Keywords: Mushroom, Perfect Hashing and Pruning(PHP), Apriori, Association rule.

KATA PENGANTAR

Alhamdulillahi rabbil 'alamin. Puji syukur penulis panjatkan kehadirat Allah SWT, atas segala rahmat dan limpahan hidayah-Nya tugas akhir yang berjudul "Identifikasi Pengkategorian *Mushroom* Menggunakan Algoritma *Apriori* Dengan Algoritma *Perfect Hashing and Pruning (PHP)*" ini dapat terselesaikan. Semoga Allah melimpahkan rahmat atas Nabi Muhammad SAW, makhluk paling mulia yang senantiasa memberikan cahaya petunjuk, dan atas keluarganya dan sahabat-sahabatnya.

Dalam penyelesaian tugas akhir ini, penulis mendapat banyak bantuan baik moral maupun materil dari banyak pihak. Atas bantuan yang telah diberikan, penulis menyampaikan penghargaan dan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Drs. Muh. Arif Rahman, M.Kom., dan Edy Santoso, S.Si., M.Kom., selaku dosen pembimbing, terima kasih atas semua saran, bantuan, kritikan, waktu, dorongan semangat dan bimbingannya.
2. Dr. Abdul Rouf Alghofari, M.Sc., selaku Ketua Jurusan Matematika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Brawijaya.
3. Drs. Marji, MT., selaku Ketua Program Studi Ilmu Komputer Universitas Brawijaya Malang.
4. Agus Wahyu Widodo, ST., selaku dosen Penasihat Akademik.
5. Segenap bapak dan ibu dosen yang telah mendidik dan mengamalkan ilmunya kepada penulis serta segenap staf dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya.
6. Papa, Mama, Abang, dan Almahurmah Kakak terima kasih atas cinta, kasih sayang, doa, dukungan dan semangat yang tiada henti.
7. Om, Tante, Yani, Intan, dan Teh Yuni, selaku orangtua dan keluarga di Malang.

8. Falen, Tyas, Nafis, Restu, Anung, Isrofi, Wisnu, Yoga, Mbak Sulis, Wewo, Welly dan rekan-rekan di Program Studi Ilmu Komputer FMIPA Universitas Brawijaya yang telah banyak memberikan bantuannya demi kelancaran pelaksanaan penyusunan tugas akhir ini.
9. Teman-teman yang di Solok, Padang dan sekitarnya.
10. Pihak lain yang telah membantu terselesaikannnya Tugas Akhir ini yang tidak bisa penulis sebutkan satu-persatu.

Semoga penulisan laporan tugas akhir ini bermanfaat bagi pembaca sekalian. Sebagai manusia, penulis menyadari tugas akhir ini masih jauh dari kesempurnaan dan banyak kekurangan, dengan kerendahan hati penulis mengharapkan kritik dan saran yang membangun dari pembaca.

Malang, 22 Juni 2011

Penulis

DAFTAR ISI


UNIVERSITAS BRAWIJAYA

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR *SOURCECODE*

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

"*Mushroom* (jamur) merupakan organisme yang mirip tumbuhan tetapi tidak memiliki klorofil" (Susilowarno, 2007). Dalam kehidupan sehari-hari, *mushroom* dapat dijumpai pada daerah yang lembab, baik di laut, danau, kulit kayu, ataupun di permukaan tanah. Secara umum *mushroom* dapat diklasifikan menjadi *mushroom* yang dapat dikonsumsi dan *mushroom* yang tidak dapat dikonsumsi (beracun). Dalam kehidupan sehari-hari masyarakat belum bisa membedakan *mushroom* dapat dikonsumsi atau tidak. Oleh karena itu, diperlukan suatu sistem yang dapat digunakan untuk mempelajari tentang *mushroom* dan mengkategorikan *mushroom* yang dapat dikonsumsi atau tidak.

Mushroom mempunyai karakateristik spesifik yang dapat dikenali sebagai penanda apakah *mushroom* tersebut dapat dikonsumsi atau tidak. Karakterisitik tersebut antara lain kepala (bentuk, permukaan, warna), bau yang dikeluarkan oleh *mushroom*, batang (bentuk, permukaan, warna, akar), *ring* (ukuran, tipe), *gill* (pelengkap, warna, ukuran, jarak), warna spora, warna kedung, populasi dan habitat *mushroom* tersebut. Dari karakteristik-karakteristik yang dimiliki *mushroom* tersebut maka dapat dikembangkan menggunakan pengetahuan yang telah ada menjadi suatu sistem yang bisa digunakan untuk mengetahui apakah *mushroom* tersebut layak dikonsumsi atau tidak.

Dalam ilmu komputer dikenal suatu teknik *association rule* yang dikembangkan oleh Rakesh Agrawal pada tahun 1993. *Association rule* adalah teknik mining untuk menentukan aturan asosiatif antara suatu kombinasi item.

Proses penemuan *association rule* melibatkan dua langkah utama yaitu langkah pertama adalah mendapatkan setiap himpunan item yang disebut *itemsets*, dimana nilai kejadian ini berupa minimum *support*, dan *itemset*s ini disebut sebagai *large itemset*s atau *frequent itemset*s. Ukuran *itemset* merepresentasikan jumlah item dalam himpunan ini. Jika ukuran sebuah *itemset* sama dengan *k*, maka *itemset* ini disebut *k-itemset.* Langkah kedua adalah mendapatkan *association rule* dari *frequent itemsets* yang dibangkitkan pada langkah pertama. Dalam langkah ini, dari setiap *frequent itemset f,*

ditemukan semua *subsets* yang tidak kosong dari *f*. Kemudian untuk setiap *subset a*, sebuah aturan dibentuk $a \rightarrow (f-a)$ dibangkitkan jika rasio dari *support* (f-a) terhadap *support* (a) lebih dari atau sama dengan minimum *confidence* (J.D. Holt, and S.M. Chung, 1999).

Beragam algoritma telah ditemukan untuk mendapatkan *frequent itemset*s. Algoritma *Apriori* adalah algoritma paling terkenal untuk menemukan pola frekuensi tinggi. Pada langkah pertama kandidat k-*itemset* dibentuk dari kombinasi *(k-1)-itemset* yang didapat dari iterasi sebelumnya. *Support* dari tiap kandidat k-*itemset* didapat dengan men-*scan database* untuk menghitung jumlah transaksi yang memuat semua item di dalam kandidat *k-itemset* tersebut. Sehingga waktu yang diperlukan bertambah (Pramudiono, 2007).

Untuk mengatasi masalah waktu pada algoritma *Apriori* dapat digunakan algoritma *Perfect Hashing and Pruning* (*PHP*). Penggunaan *PHP* diharapkan dapat memperbaiki efisiensi tahap pembentukan kandidat berikutnya. Hal tersebut dilakukan melalui teknik *hashing* yang dapat mengurangi jumlah transaksi untuk di-*scan* dan memangkas jumlah item dalam setiap transaksi. (S.Ayse Ozel dan H. Altay Güvenir, 2001).

Berdasarkan latar belakang yang telah dikemukan dapat dilihat hubungan antara *mushroom* dengan *association rule,* yaitu karakteristik yang terdapat pada *mushroom* dapat dijadikan sebagai item yang akan dikombinasi sehingga membentuk sebuah aturan apakah *mushroom* tersebut dapat dikonsumsi atau tidak. Dari paparan tersebut maka skripsi ini diberi judul "**Identifikasi Pengkategorian *Mushroom* Menggunakan Algoritma *Apriori* dengan Algoritma *Perfect Hashing and Pruning (PHP)***".

1.2 Rumusan Masalah

Rumusan masalah dalam skripsi ini adalah bagaimana kategori yang dihasilkan algoritma *Perfect Hashing and Pruning* dengan algoritma *Apriori* dalam menemukan *association rule* sesuai dengan minimum *support*, minimun *confidence* yang ditentukan.

1.3 Tujuan Penelitian

Tujuan yang ingin dicapai pada skripsi ini adalah mengetahui dan menganalisa kekuatan *association rule* yang terbentuk dengan menggunakan algoritma *Apriori* dan algoritma *Perfect Hashing and Pruning (PHP)*.

1.4 Batasan Masalah

Batasan masalah yang digunakan pada skripsi ini adalah:

1. Data yang digunakan adalah *dataset mushroom.*
2. Penelitian ini membahas tentang pola keterkaitan antar item
3. Data yang digunakan tidak mengandung *missing value*
4. Data yang digunakan bersifat statis atau tidak mengalami perubahan selama penelitian berlangsung.

1.5 Manfaat Penelitian

Manfaat yang diperoleh dari penulisan skripsi ini adalah memberikan informasi mengenai kategori *mushroom* yang dapat dikonsumsi dan tidak.

1.6 Metode Penelitian

Metode penyelesaian masalah yang dilakukan pada skripsi ini, yaitu :

1. Studi literatur.
   Mempelajari dan mengkaji beberapa literatur (jurnal, buku, dan artikel dari *website*) mengenai *data mining*, algoritma *Apriori* dan algoritma *Perfect Hashing and Pruning (PHP)*.
2. Perumusan masalah dan analisa kebutuhan
   Mengkaji permasalahan sebagai hasil dari studi pustaka dan menganalisis yang dibutuhkan.
3. Perancangan dan implementasi sistem.
   Mengimplementasikan algoritma *Apriori* dan algoritma *Perfect Hashing and Pruning (PHP)* dengan merancang dan membangun sebuah perangkat lunak untuk menemukan *association rule* pada *dataset mushroom*.
4. Uji coba dan analisis hasil implementasi.
   Menganalisa waktu eksekusi untuk algoritma *Apriori* dan algoritma *Perfect Hashing and Pruning (PHP)*.

1.7 Sistematika Penulisan

Tugas akhir ini disusun berdasarkan sistematika penulisan sebagai berikut:

BAB I : PENDAHULUAN

Berisi latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, manfaat penelitian, metode penelitian dan sistematika penulisan.

BAB II : TINJAUAN PUSTAKA

Berisi teori-teori dari berbagai pustaka yang menunjang dalam penulisian skripsi. Teori yang tercakup dalam bab ini yaitu mengenai definisi dan konsep *data mining*, *association rule*, algoritma *Apriori*, algoritma *Perfect Hashing and Pruning* dan *mushroom* (jamur).

BAB III : METODOLOGI DAN PERANCANGAN SISTEM

Bab ini berisi mengenai perancangan perangkat lunak yang dibangun, meliputi analisa umum, perancangan sistem, contoh perhitungan manual, perancangan antarmuka dan perancangan uji coba.

BAB IV : IMPLEMENTASI DAN PEMBAHASAN

Bab ini berisi hasil dari implementasi perangkat lunak yang digunakan untuk menghitung jumlah *rule* yang terbentuk dan menghitung *lift ratio rule* tersebut, pembahasan analisa hasil uji coba dan evaluasi hasil uji coba.

BAB V : KESIMPULAN DAN SARAN

Berisi kesimpulan dari seluruh rangkaian penelitian serta saran kemungkinan pengembangannya.

BAB II
TINJAUAN PUSTAKA

2.1 *Data Mining*

Data mining menyediakan teknologi yang pintar dan kemampuan untuk menjelajahi kemungkinan pengetahuan atau informasi yang tersimpan di sebuah data (Berson, 2000).

Menurut Guidici (2003) *data mining* adalah proses seleksi, eksplorasi, dan pemodelan dari data dalam jumlah besar untuk dengan tujuan memperoleh hasil yang jelas dan bermanfaat untuk pemilik database.

Selain definisi di atas beberapa definisi juga diberikan seperti tertera di bawah ini:

1. *Data Mining* adalah proses untuk menemukan *pattern* yang bernilai dan *relationship* yang tersembunyi dalam *database* yang sangat besar (Seidman, 2000).
2. *Data mining* adalah beberapa cara pengembangan dari ilmu statistik dengan sedikit *Artifical intellingence* dan seperti sebuah mesin yang mempelajari data untuk mengatasi masalah dengan menghasilkan informasi yang tidak kelihatan atau tidak disadari oleh pengguna informasi tersebut (Thearling, 1995).
3. *Data mining*, sering juga disebut *knowledge discovery in database* (KDD), adalah kegiatan yang meliputi pengumpulan, pemakaian data historis untuk menemukan keteraturan, pola atau hubungan dalam set data berukuran besar (Santoso, 2007).

Dari definisi-definisi yang telah disampaikan, hal penting yang terkait dengan *data mining* adalah: (Kusrini, 2009)

1. *Data mining* merupakan suatu proses otomatis terhadap data yang sudah ada.
2. Data yang akan diproses berupa data yang sangat besar.
3. Tujuan *data mining* adalh mendapatkan hubunga atau pola yang mungkin memberikan indikasi yang bermanfaat.

2.2 *Association Rule*

Motivasi awal pencarian *association rule* berasal dari keinginan untuk menganalisa data transaksi supermarket, ditinjau dari perilaku *customer* dalam membeli produk. *Association rule* ini menjelaskan seberapa sering suatu produk dibeli secara bersamaan. Sebagai

contoh, *association rule* "*beer* ⇒ *diaper* (80%)" menunjukkan bahwa empat dari lima customer yang membeli *beer* juga membeli *diaper*.

Penting tidaknya suatu aturan assosiatif dapat diketahui dengan dua parameter, *support* dan *confidence*. Persamaan 2.1 merupakan rumus *support* dan persamaan 2.2 merupakan rumus *confidence*.

$$Support\ (A, B) = \frac{\sum Transaksi\ mengandung\ A\ dan\ B}{\sum Transaksi} \quad (2.1)$$

$$Confidence\ (A \rightarrow B) = \frac{\sum Transaksi\ mengandung\ A\ dan\ B}{\sum Transaksi\ mengandung\ A} \quad (2.2)$$

Kegunaan dari *support* itu sendiri adalah untuk mengukur tingkat intensitas kemunculan suatu *rule*, dimana jika *support* yang dimiliki rendah, maka akan besar kemungkinan rendah juga tingkat keuntungan yang didapatkan dari item-item yang ada pada *rule* tersebut. Sedangkan kegunaan dari *confidence* adalah untuk mengukur tingkat kebenaran(*reability*) dari kesimpulan yang diambil oleh *rule* yang dibuat. Pada implikasi X→Y, jika nilai *confidence* rendah maka kemungkinan munculnya Y yang memuat X semakin rendah pula.

Selain menggunakan *confidence* untuk mengukur tingkat kebenaran, dikenal juga *lift ratio* untuk melihat kuat tidaknya aturan asosiasi dengan membandingkan dengan nilai *benchmark confidence*. Dimana diasumsikan kejadian item dari *consequent* (item setelah maka) dalam suatu transaksi adalah *independent* dengan kejadian dari *antecedent* (item setelah jika) dari suatu aturan asosiasi. *Benchmark confidence* dapat dinyatakan pada persamaan 2.3. Sedangkan untuk mendapatkan *lift ratio* dapat dinyatakan dengan persamaan 2.4. Jadi *lift ratio* adalah perbandingan antara *confidence* untuk suatu aturan dibagi dengan *benchmark confidence*. Apabila nilai *lift ratio* besar dari 1 menunjukkan adanya manfaat dari aturan tersebut. Lebih tinggi nilai *lift ratio*, maka lebih besar kekuatan asosiasi (Santosa, 2007).

$$Benchmark\ Confidence = \frac{jumlah\ transaksi\ dengan\ item\ dalam\ consequent}{jumlah\ transaksi\ dalam\ database} \quad (2.3)$$

$$Lift\ ratio = \frac{confidence}{Benchmark\ confidence} \quad (2.4)$$

Permasalahan untuk menemukan seluruh *association rule* yang ada pada suatu database dapat dibagi menjadi dua fase utama berikut : (Gunawan, 2007).

1. Fase Pencarian *Large Itemset*
   Menemukan seluruh item dari transaksi yang memenuhi *minimum support threshold*. *Support* untuk suatu *itemset* adalah jumlah transaksi dalam database yang mengandung *itemset* tersebut. *Itemset* yang memenuhi persyaratan ini disebut *frequent itemset* (*large itemset*) dan sebaliknya *infrequent itemset* (*small itemset*).
2. Fase *Generate Strong Association Rules*
   Dengan menggunakan *frequent itemset* yang terbentuk dihasilkan (*strong*) *association rules* yang memenuhi *minimum confidence threshold* yang telah dispesifikasikan.

2.3 Algoritma *Apriori*

2.3.1 Definisi

Algoritma *Apriori* pertama kali diajukan oleh R. Agrawal dan R. Srikant tahun 1994. Menurut Savasere (1995), "*Apriori* adalah salah satu pendekatan yang sering digunakan pada *Frequent Itemset Mining*". Prinsip *Apriori* adalah jika sebuah *itemset infrequent*, maka *itemset* yang *infrequent* tidak perlu lagi di*explore* supersetnya sehingga jumlah kandidat yang harus diperiksa menjadi berkurang.

2.3.2 Proses *Apriori*

Sesuai dengan namanya, algoritma ini menggunakan *prior knowledge* mengenai *frequent itemset properties* yang telah dketahui sebelumnya, untuk memproses informasi selanjutnya. *Apriori* menggunakan pendekatan *level-wise search* (*breadth-first search*), dimana *k-itemset* digunakan untuk membentuk *(k+1)-itemset*.

Pertama-tama dicari set dari *frequent 1-itemset*, set ini dinotasikan sebagai L_1, L_1 digunakan untuk menemukan L_2, kemudian set dari *frequent 2-itemset*, digunakan untuk menemukan L_3, dan seterusnya, sampai tidak ada lagi *frequent k-itemset* yang dapat ditemukan. Proses untuk menemukan setiap L_k membutuhkan satu kali pemeriksaan menyeluruh pada *database*, yang artinya jika k=4, maka pemeriksaan terhadap database dilakukan sebanyak empat kali.

Secara umum, algoritma *Apriori* mempunyai langkah-langkah sebagai berikut:

1. Pembentukan 1-*itemset*
   Mendata item-item yang memiliki jumlah kemunculan pada data detail transaksi diatas minimum *support* yang telah ditentukan.
2. Pembentukan kandidat *k-item* atau C_k ($k \geq 2$)
   Kandidat *k-item* diperoleh dengan menyusun kombinasi-kombinasi *k-item* yang item-itemnya terdaftar pada (*k*-1) *itemset*.
3. Memangkas kandidat
   Menghapus kombinasi item dalam daftar kandidat yang sudah terbentuk yang mana subset dari kombinasi item tersebut, tidak terdapat pada *itemset* sebelumnya. *Subset* ialah kombinasi (k-1) item dari dari k-*itemset*. Gambar 2.1 adalah listing untuk pemangkasan kandidat.

```
1  forall itemsets c ∈ C_k do
2    forall (k-1)-subsets s of c do
3      if (s ∉ L_{k-1}) then
4        delete c from C_k ;
```

Gambar 2.1 *Listing Prune Step* Kandidat *Apriori* (Agrawal,1994)

4. Pencocokan data transaksi dengan data kandidat k-item
   Pada langkah ini, item-item pada tiap transaksi disusun membentuk kombinasi *k-item*. Kombinasi-kombinasi tersebut dicocokkan dengat data kombinasi yang ada pada daftar kandidat *k-item*. Jika terdapat kombinasi yang sama, maka nilai *support* atau nilai kemunculan pada kombinasi tersebut ditambahkan dengan 1 (*increment*). Langkah ini terus dilakukan hingga pembacaan data transaksi terakhir.
5. Pembentukan k-*itemset*
   Kombinasi-kombinasi item yang terdaftar pada himpunan *k-itemset* adalah kombinasi-kombinasi item yang terdaftar pada kandidat *k-itemset* dan memiliki nilai kemunculan diatas minimum *support*.
6. Lakukan iterasi
   Mengulangi langkah 2 hingga 5 dengan nilai $k=k+1$ hingga tidak ditemukan kombinasi pada k-*itemset*.

Pseudocode algoritma *Apriori* dapat dilihat pada Gambar 2.2.

```
1   L1 = {large 1-itemsets} ;
2   for (k = 2; Lk-1 ≠ ∅; k++ ) do begin
3      Ck = Apriori-gen(Lk-1);// New Candidate
4      forall transactions t ϵ D do begin
          //Candidates contained in t
5         Ct = subset (Ck, t);
6         forall candidates c ∈ Ct do
7            c.count ++;
8         end
9         Lk = {c ∈ Ck | c.count ≥ minsup}
10  end
11  Answer = ⋃k Lk ;
12
```

Gambar 2.2 Algoritma *Apriori* (Agrawal,1994)

2.4 Algoritma *Perfect Hashing and Pruning* (*PHP*)

Direct Hashing and Pruning (DHP) adalah variasi dari algoritma *Apriori*, dalam arti bahwa DHP menggunakan teknik *hashing* untuk menghapus *itemset*s yang tidak diinginkan untuk generasi selanjutnya dari kandidat *itemset*s (Iti, 2008). Pada algoritma *Apriori*, untuk menemukan *frequent itemset* dengan cara men-*scan database* sehingga ukuran database menjadi besar dan waktu yang dibutuhkan lama. Sedangkan algoritma DHP dapat bekerja lebih cepat dari pada algoritma *Apriori*, dengan menggunakan proses *pruning* setiap transaksi ke *hash table*. Tetapi algoritma DHP memiliki kekurangan yaitu bahwa *hash table* bersaing untuk mendapatkan ruang memori dengan *hash tree* yang digunakan untuk menahan hitungan untuk *itemset*.

Algoritma yang diajukan menggunakan *hashing* sempurna untuk *hash table* pada setiap pass dan juga mengurangi ukuran dari database dengan *pruning* transaksi yang tidak berisi *frequent item*. Sehingga Algoritma ini disebut algoritma *Perfect Hasing and Pruning* (*PHP*) (S.Ayse Ozel dan H. Altay Güvenir, 2001).

Secara umum Algoritma *Perfect Hasing and Pruning* (*PHP*) adalah sebagai berikut.

1. Selama pass pertama, *hash table* dengan ukuran sama dengan item khusus dalam database dibuat. Setiap item khusus dalam database dipetakan ke lokasi berbeda dalam *hash tabel*, dan metode ini disebut *perfect hasing*. Dan metode *add* dari *hass table* menambahkan *entry* baru jika sebuah *entry* untuk *item x* tidak ada

dalam *hash table* dan memulai hitungannya ke 1, jika tidak ini menambahkan hitungan *x* dalam tabel ini dengan 1.

2. Setelah pass pertama, *hash table* berisi jumlah pasti kejadian dari setiap item dalam *database*. Dengan hanya membuat satu pass pada *hash table*, yang ada dalam memori, algoritma ini dengan mudah membangkitkan *frequent 1-itemsets*. Setelah operasi ini, metode *prune* dari *hash tabel* mengurangi semua entry yang dukungannya kurang dari *minimum support*.
3. Dalam pass selanjutnya, algoritma ini mengurangi database dengan membuang transaksi yang tidak memiliki item dari *frequent itemsets*, dan juga memangkas item yang tidak sering dari transaksi ini. Pada saat yang sama, ini menghasilkan kandidat *k-itemset* dan menghitung kejadian *k-itemset*.
4. Pada akhri pass, D_k berisi *database* yang dipangkas, H_k berisi kejadian kandidat *k-itemset*, dan F_k adalah set dari *frequent k-itemset*. Proses ini berlanjut sampai tidak adalah F_k baru yang ditemukan.

Pseudocode algoritma *Perfect Hashing and Pruning* (*PHP*) dapat dilihat pada Gambar 2.3.

```
Input: Database
Output: Frequent k-itemset
/* Database = kumpulan dari transaksi;
    Items = kumpulan item-item;
    transaksi = <TID, {x ∈ Items}>;
   F1  is kumpulan frequent 1-itemsets */

F1 = ∅;
 /*  H1 merupakan tabel hash untuk 2-itemsets
    Baca transaksi dan hitung setiap kejadian dari
  setiap item dan dihasilkan H1 */

for transaksi t ∈ Database do begin
      for item x in t do
           H1.add(x);
end;

//bentuk frequent 1-itemset

for itemset y di H1 do
      if H1.hassupport(y)
      then F1 = F1 ∪ y
end
```

```
//Hapus nilai hash yang tidak memenuhi minimum support

H1.prune(minsup);
D1 = Database;
 //  Dk  merupakan hasil database yang dipruning

/* cari Fk , kumpulan frequent k-itemsets, dimana  k ≥
  2 dan pangkas database */
k = 2;
repeat
       Dk =∅;
       Fk =∅;
      for transaksi  t ∈ Dk-1 do begin
          // w merupakan item di t pada subset k-1
          if  ∀w | w ∉ Fk-1
          then skip t;
          else
              items =∅;
              for k -itemset y di t do
                 if ¬∃z|z = subset k-1 pada y
                    ∧¬Hk-1.hassupport(z)
                 then  Hk.add(y);
                 items=items ∪ y;
            end
            Dk = Dk ∪ y    // bagian dari t yang
                           berisi                //
                           hanya item pada itemset
       end

      for itemset y di Hk do
          if  Hk.hassupport(y)
          then  Fk = Fk ∪ y
      end

//  Hapus  nilai  hash  yang  tidak  memenuhi  minimum
support dari tabel Hk
      Hk.prune (min sup);
      k++;
until  ; Fk-1 =∅;
Answer =  ; ∪kFk ;
```

Gambar 2.3 Algortima *Perfect Hashing and Pruning* (S.Ayse Ozel dan H. Altay Güvenir, 2001).

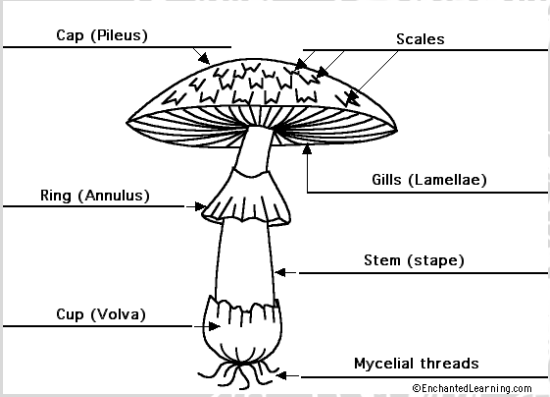
Algoritma ini jelas lebih baik daripada algoritma DHP, karena setelah pembentukan *hash table*, ini tidak perlu menghitung kejadian

kandidat *k-itemset* sebagaimana dalam kasus algoritma DHP. Juga, algoritma yang diajukan bekerja lebih baik daripada algoritma *Apriori*, karena pada setiap iterasi, ukuran *database* direduksi, dan ini memberikan kinerja yang tinggi pada algoritma ketika ukuran database besar dan jumlah *frequent itemsets* relatif kecil.

Sebagai tambahan, setelah setiap iterasi, *database* D_k berisi transaksi dengan *frequent items* saja. Algoritma ini membentuk semua *k-subset* item dalam setiap transaksi dan menyisipkan item yang *k-1* subsetnya adalah besar terhadap *hash table*. Untuk alasan ini, algoritma tidak kehilangan *frequent itemset*. Karena algoritma ini membuat pemangkasan selama penyisipan kandidat *k-itemset* ke H_k, ukuran *hash table* tidak besar dan cocok dengan memori (S.Ayse Ozel dan H. Altay Güvenir, 2001).

2.5 *Mushroom* (Jamur)

"*Mushroom* merupakan organisme yang mirip tumbuhan tetapi tidak memiliki klorofil" (Susilowarno, 2007). Bentuk umum *mushroom* biasanya adalah seperti payung yang dapat dilihat pada gambar 2.4.



Gambar 2.4 Struktur Tubuh *Mushroom*

Dari gambar 2.4 dapat dilihat struktur tubuh *mushroom* terdiri dari:

1. *Cap (Pileus)*
   Cap merupakan bagian atas *mushroom* yang bentuknya menyerupai topi. Ada sejumlah informasi yang didapatkan tentang *mushroom* dari melihat fitur-fitur yang terdapat pada *cap*. Fitur-

fitur yang dapat dilihat tersebut seperti warna, bentuk dan tekstur dari *cap* tersebut.

2. *Scales*
   Scales merupakan potongan jaringan pada permukaan *cap*.
3. *Ring (Annulus)*
   Ring merupakan membran yang terletak di bawah tutup dan melingkari batang. Tidak semua *mushroom* memiliki *ring*.
4. *Gills (Lamellae)*
   Bagian yang memproduksi spora *mushroom* yang subur, *gills* terletak di bawah *cap*. Fitur-fitur yang terdapat pada *gills* antara lain : spasi, warna, tekstur, ketebalan dan kerapuhan *gills* tersebut.
5. *Cup (Volva)*
   Cup merupakan membran tebal yang terdapat di dasar *mushroom*. Tidak semua *mushroom* memiliki *cup*.
6. *Stem (Stape)*
   Bagian dari *mushroom* antara *cap* dan tanah. *Stem* merupakan dukungan utama dari sebuah *mushroom*.
7. *Mycelial threads*
   Mycelial threads merupakan akar *mushroom*.

Secara umum *mushroom* terbagi atas dua kelompok besar, yaitu *mushroom* beracun dan tidak beracun. *Mushroom* tidak beracun adalah *mushroom* yang dapat dikonsumsi sebagai makanan kesehatan yang rata-rata memiliki khasiat memperlancar peredaran darah. Beberapa jenis *mushroom* (jamur) yang dapat dikonsumsi oleh manusia diantaranya : *volvariela volvacea* (jamur merang), *pleurotus* (jamur tiram), jamur *portabella, auricularia polytricha* (jamur kuping hitam), *auricularia auricular judae* (jamur kuping merah), *agaricus campestris* (jamur kancing), jamur kompos, dan *lentinus edulis* (jamur shitake).

Adapun *mushroom* beracun adalah jenis *mushroom* yang tidak dapat diolah sebagai bahan makanan. Keberadaannya hanya sebagai penanda terhadap sesuatu yang lembap, lapuk dan membusuk seperti batang-batang pohon, kayu lapuk, dan makanan yang dibiarkan dalam keadaan terbuka untuk waktu yang lama. Contoh *mushroom* beracun antara lain: *amanita muscaria, lepoita, russula, collybia, boletus*.

Ciri-ciri dari *mushroom* beracun sebagai berikut:

- Warna *mushroom* mencolok seperti merah darah, biru tua, oranye, dan hitam legam. Namun, ada pula yang memiliki warna

lembut seperti kuning muda dan putih menyerupai *mushroom* tidak beracun. *Mushroom* yang berwarna gelap dan dapat dimakan umumnya berwarna cokelat tua.

- *Mushroom* beracun umumnya mengeluarkan bau yang menyengat seperti bau telur busuk dan gas amoniak.
- *Mushroom* beracun memiliki cawan atau cincin pada pangkal batangnya. *Mushroom* yang tidak beracun juga ada yang memiliki cawan seperti *mushroom* merang dan ada pula yang memiliki cincin, seperti *mushroom* kompos.

Umumnya, *mushroom* ini banyak terdapat di tempat-tempat kotor seperti pembuangan sampah dan dekat kandang hewan.

BAB III
METODOLOGI DAN PERANCANGAN SISTEM

Bab ini menerangkan beberapa hal mengenai metode dan perancangan rumusan masalah dalam skripsi ini. Langkah-langkah yang dilakukan dalam penelitian ini yakni:

1. Mempelajari metode yang digunakan dari referensi yang telah ada.
2. Menentukan *dataset* yang akan digunakan.
3. Melakukan perancangan perangkat lunak dengan metode yang digunakan.
4. Mengimplementasikan rancangan yang dilakukan.
5. Uji coba perangkat lunak dengan data yang telah ada.
6. Evaluasi hasil keluaran yang telah dilakukan oleh perangkat lunak.

3.1 Analisa Umum

3.1.1 Deskripsi Umum Sistem

Sistem yang dikembangkan dalam penelitian ini adalah perangkat lunak untuk menemukan hubungan antar item (*Association Rule*). Sistem ini menggunakan algoritma *Apriori* dan algoritma *Perfect Hashing and Pruning*. Sistem ini bertujuan untuk menemukan pola atau hubungan yang terkait antar satu item dengan item lainnya.

Pada sistem ini digunakan *Dataset Mushroom* yang berisi karakteristik/ciri-ciri dari *mushroom*. *Itemset* yang dimaksud dalam sistem ini adalah himpunan karakteristik *mushroom*. Sedangkan *Frequent itemset* yang dimaksud adalah himpunan karakteristik *mushroom* yang sering muncul dalam *database* dan memenuhi *minimum support* yang telah ditentukan. Sedangkan, *Association rule* yang terbentuk adalah *frequent itemset* yang memenuhi *minimum confidence.*

3.1.2 Data Penelitian

Dataset yang digunakan dalam skripsi ini yakni *dataset Mushroom*. Data ini didapat dari situs UCI *Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets/Mushroom)*. Data disimpan pada sebuah tabel yang tidak berealasi. Data terdiri dari 21 atribut yang merupakan ciri-ciri berdasarkan struktur tubuh, populasi dan habitat *mushroom*, yang terdiri dari :

1. *cap-shape*, bertipe data *categorical*
2. *cap-surface*, bertipe data *categorical*
3. *cap-color*, bertipe data *categorical*
4. *bruises*, bertipe data *categorical*
5. *odor*, bertipe data *categorical*
6. *gill-attachment*, bertipe data *categorical*
7. *gill-spacing*, bertipe data *categorical*
8. *gill-size*, bertipe data *categorical*
9. *gill-color*, bertipe data *categorical*
10. *stalk-shape*, bertipe data *categorical*
11. *stalk-root*, bertipe data *categorical*
12. *stalk-surface-above-ring*, bertipe data *categorical*
13. *stalk-surface-below-ring*, bertipe data *categorical*
14. *stalk-color-above-ring*, bertipe data *categorical*
15. *stalk-color-below-ring*, bertipe data *categorical*
16. *veil-color*, bertipe data *categorical*
17. *ring-number*, bertipe data *categorical*
18. *ring-type*, bertipe data *categorical*
19. *spore-print-color*, bertipe data *categorical*
20. *population*, bertipe data *categorical*
21. *habitat*, bertipe data *categorical*

dimana ciri-ciri tersebut digolongkan menjadi 2 kelas yaitu *edible* dan *poisonous*.

3.1.3 Batasan Sistem

Adapun batasan pada sistem yang dibangun adalah:

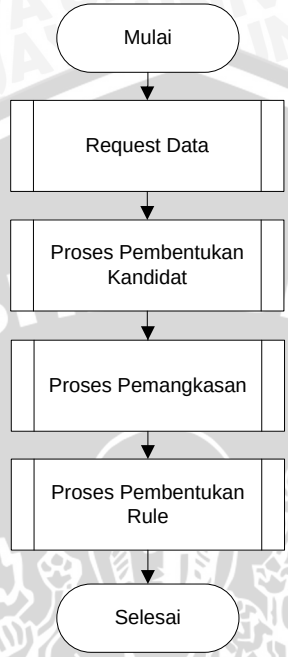
1. *Dataset* yang digunakan merupakan *dataset* yang tidak memiliki relasi (hanya terdiri dari satu tabel).
2. *Dataset* yang digunakan tidak mengandung *missing value.*
3. Nilai *minimum support, minimum confidence* dan jumlah n-itemset ditentukan oleh *user*.

3.2 Perancangan Sistem

Pada subbab ini dijelaskan proses-proses yang terjadi pada sistem. Penemuan *associatin rule* dilakukan dengan algoritma *Apriori* dan algoritma *Perfect Hashing and Pruning.*

3.2.1 Proses *association rule* menggunakan algoritma *Apriori*

Tahapan menemukan *association rule* menggunakan algoritma *Apriori* dapat dilihat pada gambar 3.1



Gambar 3.1 Alur Proses *Association Rule* Menggunakan Algoritma *Apriori*

1. Proses *Request* Data

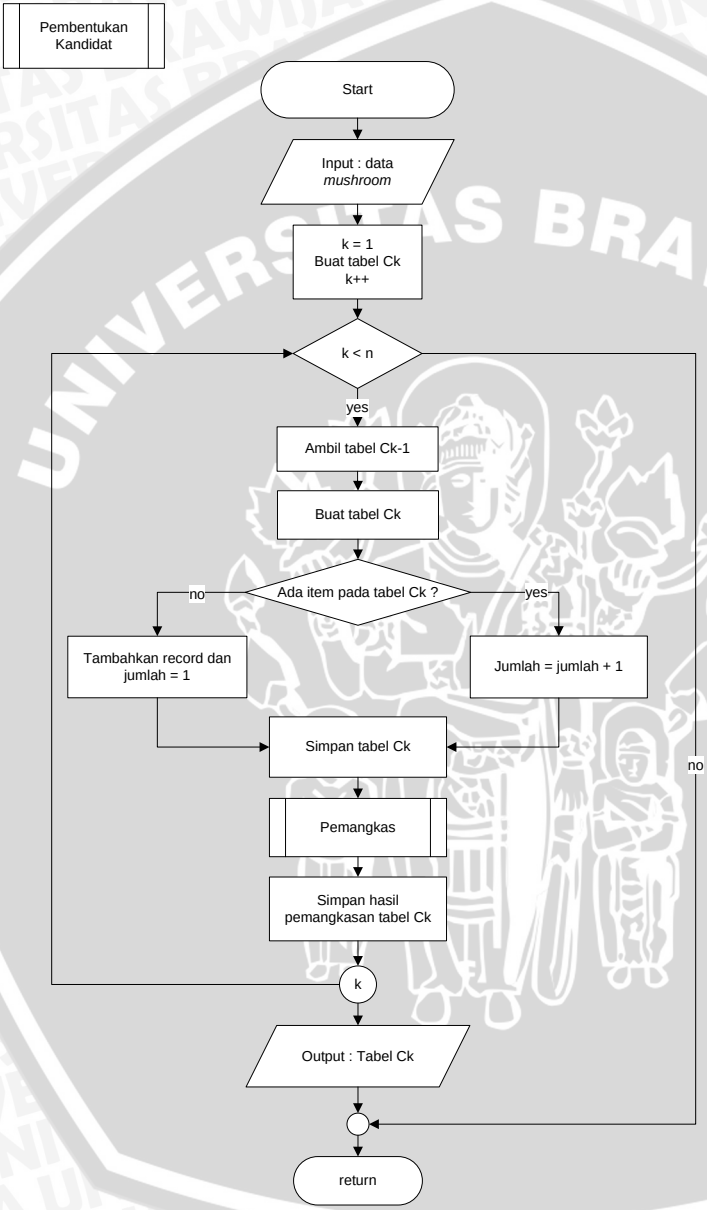
Proses yang pertama kali dilakukan oleh sistem yaitu proses *request* data. Dimana sistem akan mendapatkan informasi dari data yang akan diolah. Adapun informasi yang didapatkan yaitu berupa jumlah *record*, jumlah atribut beserta jumlah nilai yang terkandung didalam atribut tersebut.

2. Proses Pembentukan Kandidat *k-itemset*

Kandidat *k-itemset* didapatkan dengan cara mencari kombinasi dari (*k-1*)-*itemset* yang terbentuk. Item-item tersebut akan dimasukkan kedalam tabel *Ck. Flowcart* pembentukan kandidat dapat dilihat pada gambar 3.2.

Khusus untuk kandidat *1-itemset* didapatkan dengan cara mencari nilai *support* pada masing-masing jenis item dengan menggunakan persamaan 2.1. Nilai *support* diperoleh dengan cara menghitung seberapa banyak kemunculan jenis item tersebut pada *dataset* yang

digunakan. Jika nilai *support* item lebih besar dari nilai minimum *support* maka terbentuklah 1-*itemset*.

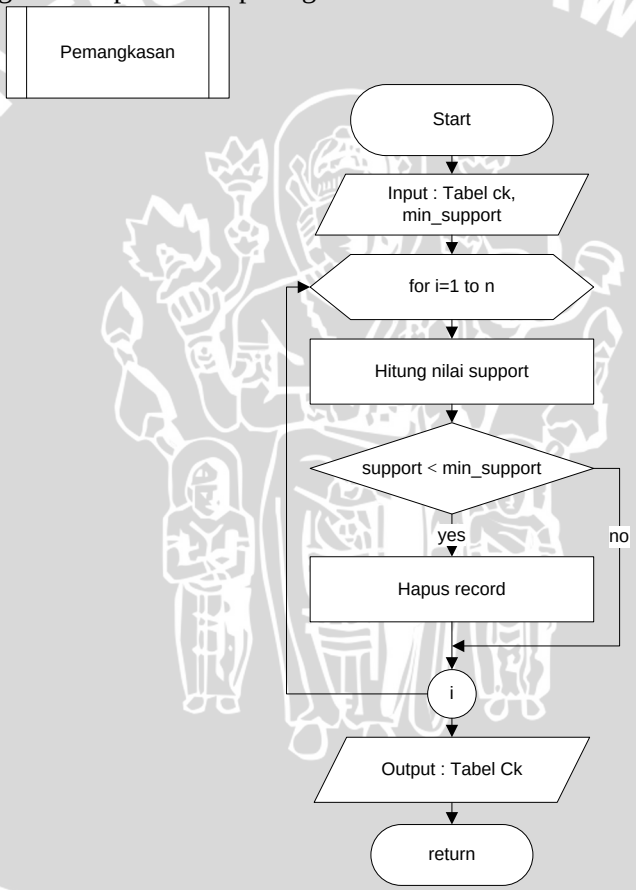


Gambar 3.2 *Flowchart* Pembentukan Kandidat *Apriori*

3. Proses Pemangkasan

Nilai inputan yang dimasukkan adalah tabel C*k* yang berisi kandidat dan nilai minimum *support*. Dari inputan yang dimasukkan dilakukan proses penghitungan nilai *support* menggunakan persamaan 2.1. Pada proses pengecekan nilai *support* dilakukan perulangan, jika nilai *support* tersebut lebih kecil dengan nilai minimum *support* maka kandidat tersebut dihapus dari tabel C*k*.

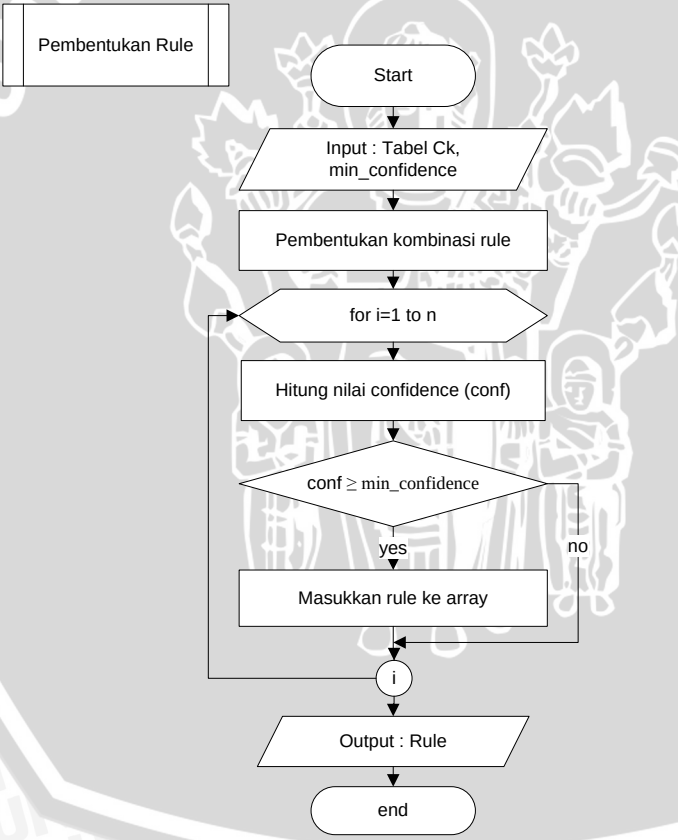
Proses pembentukan kandidat dan proses pemangkasan terus dilakukan sampai tidak ada lagi kandidat yang terbentuk. Alur proses pemangkasan dapat dilihat pada gambar 3.3.



Gambar 3.3 *Flowcart* Pemangkasan

4. Proses Pembentukan Aturan (*Rule*)

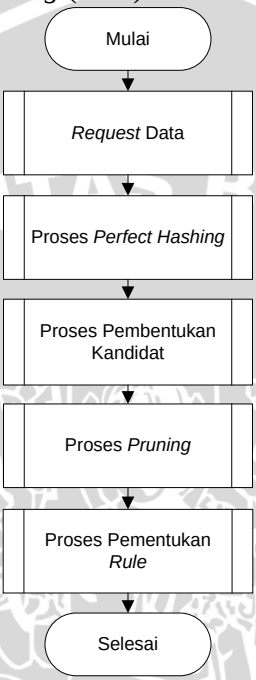
Setelah semua kandidat memenuhi nilai minimum *support*, barulah dicari aturan assosiatif yang memenuhi syarat minimum untuk *confidence* dengan menghitung *confidence* aturan assosiatif. Nilai inputan yang dimasukkan adalah tabel C*k* dan nilai minimum *confidence*. Dari inputan yang dimasukkan dilakukan proses pembentukan kombinasi *rule* dan selanjutnya adalah proses penghitungan nilai *confidence*. Pada proses pengecekan nilai *confidence* dilakukan perulangan, jika nilai *confidence rule* tersebut lebih besar sama dengan nilai minimum *confidence* maka *rule* tersebut dimasukkan kedalam array baru dan jika tidak maka tidak dimasukkan. Pembentukan aturan ini berhenti sampai isi tabel Ck habis. *Flowchart* ditunjukkan pada gambar 3.4.



Gambar 3.4 *Flowchart* Pembentukan Aturan *Apriori*

3.2.2 Proses *association rule* menggunakan algoritma *PHP*

Tahapan menemukan *association rule* menggunakan algoritma *Perfect Hashing and Pruning* (*PHP*) diilustrasikan pada gambar 3.5.



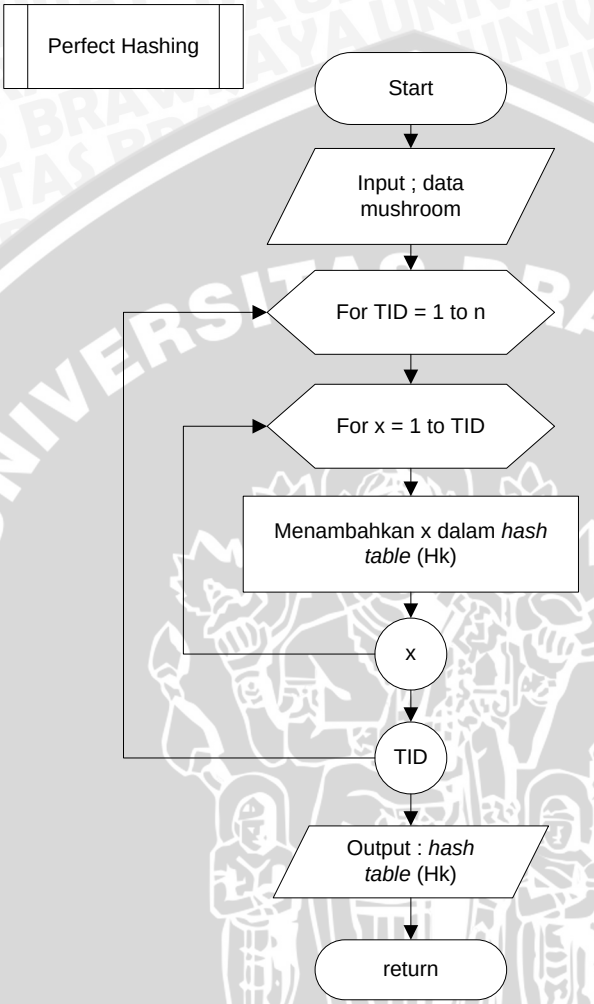
Gambar 3.5 Alur Proses *Association Rule* Menggunakan Algoritma *Perfect Hashing and Pruning*

1. Proses *Request* Data

Proses *request* data pada algoritma *Perfect Hashing and Pruning* sama dengan pada algoritma *Apriori*. Dimana sistem akan mendapatkan informasi dari data yang akan diolah. Adapun informasi yang didapatkan yaitu berupa jumlah *record*, jumlah atribut beserta jumlah nilai yang terkandung didalam atribut tersebut.

2. Proses *Perfect Hashing*

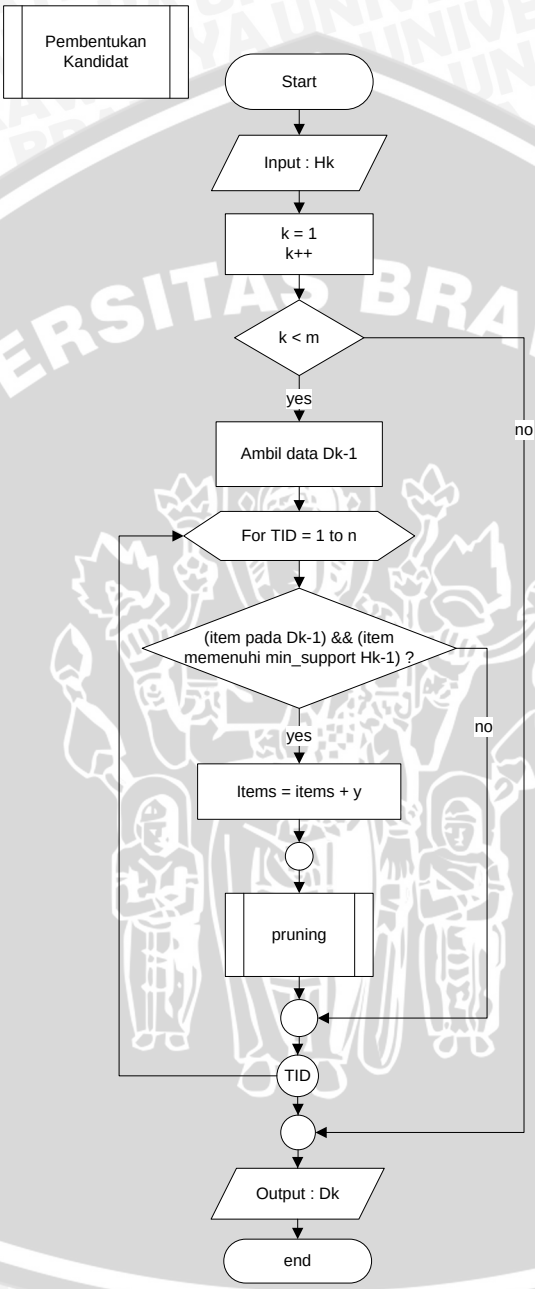
Proses *Perfect Hashing* digunakan untuk membuat tabel *hash* yang sempurna, tanpa ada bentrokan. Semua item yang berbeda dipetakan ke dalam nilai *hash* yang berbeda-beda. Jika item tidak ada dalam tabel *hash* maka ditambahkan perhitungannya dimulai dari 1. *Flowchart Perfect Hashing* ditunjukkan pada gambar 3.6.



Gambar 3.6 *Flowchart Perfect Hashing*

3. Proses pembentukan kandidat

Untuk pembentukan 1-*itemset*, inputan yang dimasukkan adalah *hash table* yang berisi item-item yang terdapat pada data *mushroom.* Sedangkan untuk *k-itemset* pembentukan kandidat diambil dari tabel D*k-1*. Didalam proses ini terdapat proses *pruning. Flowchart* pembentukan kandidat ditunjukkan pada gambar 3.7.



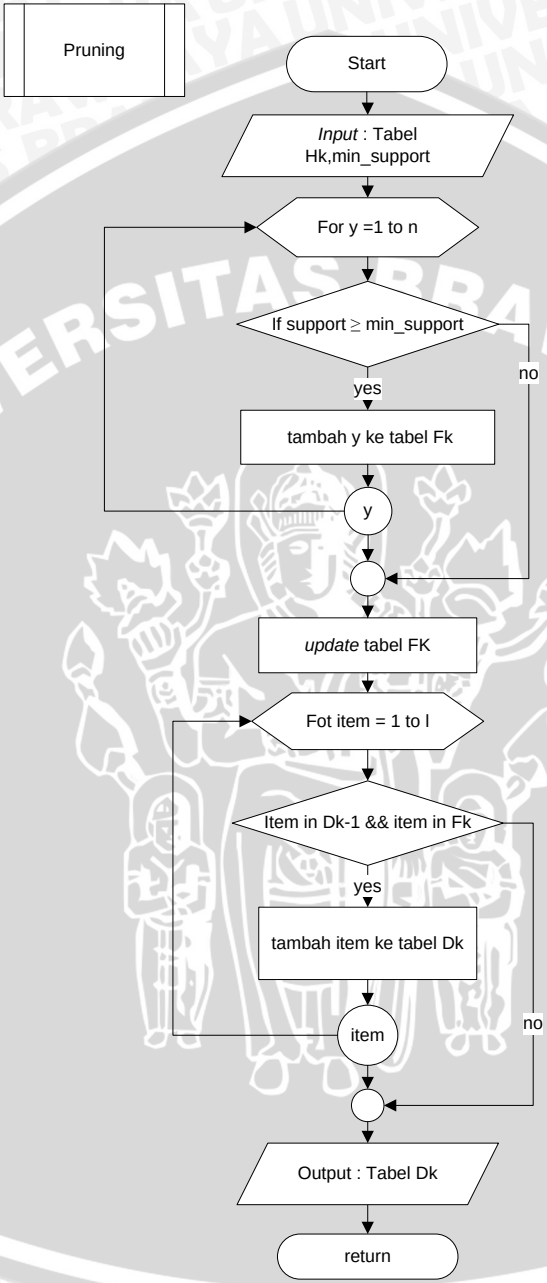
Gambar 3.7 *Flowchart* Pembentukan kandidat

4. Proses *Pruning*

Proses *pruning* dari tabel *hash* mengurangi item yang tidak memenuhi minimum *support* dan juga membuang transaksi yang tidak memiliki item dari *frequent itemsets.* Pada saat yang sama akan menghasilkan *k-itemset* dan menghitung kejadian *k-itemset*.

Nilai inputannya berupa tabel H*k* dan nilai minimum *support*. Dari nilai inputan dihitung nilai *support* masing-masing kandidat. Setelah itu dilakukan pengecekan, apabila nilai *support* kandidat lebih besar sama dengan nilai minimum *support* maka kandidat tersebut masuk ke tabel F*k* jika tidak maka kandidat tersebut tidak dimasukkan ke tabel F*k*.

Setelah itu dilakukan proses *pruning* lagi untuk membentuk tabel D*k*. Proses ini dilakukan dengan melihat item-item yang terdapat pada tabel D*k-1* dan F*k*. Jika item D*k-1* juga terdapat pada F*k* maka item ditambahkan ke dalam tabel F*k*. Proses akan berhenti jika isi tabel F*k* kosong. *Flowchart Pruning* ditunjukkan pada gambar 3.8.



Gambar 3.8 *Flowchart Pruning*

5. Proses Pembentukan Rule

Proses pembentukan *rule* dengan menggunakan algoritma *Perfect Hashing and Pruning* sama dengan algoritma *Apriori* yaitu dengan menghitung nilai *confidence frequent itemset* yang terbentuk. Setelah itu dilakukan pengecekan, jika nilai *confidence* lebih besar sama dengan minimum *confidence* maka terbentuk sebuah *rule* yang dimasukkan ke sebuah tabel baru jika tidak maka tidak ditambahkan.

3.3 Perhitungan Manual

Pada perhitungan manual ini data yang digunakan merupakan sampel data dari *dataset mushroom* yg terdiri dari 22 atribut (termasuk kelas) dan 8416 data. Data yang diambil sebagai sampel hanya 15 dengan 5 atribut. Atribut yang digunakan adalah atribut *type, cap-shape, cap-surface, cap-color, bruises*. Sampel *dataset mushroom* dapat dilihat pada tabel 3.1.

Tabel 3.1 *Dataset Mushroom*

Type	*cap-shape (V01)*	*cap-surface (V02)*	*cap-color (V03)*	*bruises (V04)*
edible	Convex	scaly	white	bruises
edible	Convex	fibrous	brown	no
edible	Sunken	fibrous	gray	no
edible	flat	fibrous	white	no
poisonous	convex	smooth	brown	bruises
poisonous	convex	scaly	white	bruises
poisonous	convex	smooth	brown	bruises
edible	bell	smooth	yellow	bruises
edible	convex	scaly	brown	bruises
edible	bell	scaly	yellow	bruises
edible	bell	scaly	white	bruises
poisonous	flat	scaly	white	bruises
poisonous	flat	smooth	white	bruises
edible	flat	fibrous	brown	no
edible	convex	smooth	yellow	bruises

3.3.1 Perhitungan Manual Menggunakan Algoritma *Apriori*

Langkah 1. Mendata item-item yang terdapat pada *dataset* dapat dilihat pada tabel 3.2.

Tabel 3.2 *C1* (*Candidate 1-itemset*)

Item-1	Jumlah
edible	10
poisonous	5
convex	7
sunken	1
flat	4
bell	3
scaly	6
fibrous	4

Item-1	Jumlah
smooth	5
white	6
brown	5
gray	1
yellow	3
bruises	11
no	4

Langkah 2. Memangkas *itemset* yang tidak memenuhi *minimum support*. Hasil pemangkasan dapat dilihat pada tabel 3.3.

Misalkan *minimum support* = 20% (3 dari 15 transaksi)

Tabel 3.3 L1 (*Large* 1-*itemset*)

Itemset	Jumlah
edible	10
poisonous	5
convex	7
flat	4
bell	3
scaly	6
fibrous	4
smooth	5
white	6
brown	5
yellow	3
bruises	11
no	4

Langkah 3. Mencari kandidat 2-*itemset*. Calon kandidat dapat dilihat pada tabel 3.4

Tabel 3.4 *C2(Candidate 2-itemset)*

Item-1	*Item-2*	Jumlah
edible	convex	4
edible	flat	2
edible	bell	3
edible	scaly	4
edible	fibrous	4
edible	smooth	2
edible	white	3
edible	brown	3
edible	yellow	3
edible	bruises	6
edible	no	4
poisonous	convex	3
poisonous	flat	2
poisonous	bell	0
poisonous	scaly	2
poisonous	fibrous	0
poisonous	smooth	3
poisonous	white	3
poisonous	brown	2
poisonous	yellow	0
poisonous	bruises	5
poisonous	no	0
convex	scaly	3
convex	fibrous	1
convex	smooth	3
convex	white	2
convex	brown	4
convex	yellow	1
convex	bruises	6
convex	no	1
flat	scaly	1
flat	fibrous	2

Item-1	*Item-2*	Jumlah
flat	brown	1
flat	yellow	0
flat	bruises	2
flat	no	2
bell	scaly	2
bell	fibrous	0
bell	smooth	1
bell	white	1
bell	brown	0
bell	yellow	2
bell	bruises	3
bell	no	0
scaly	white	4
scaly	brown	1
scaly	yellow	1
scaly	bruises	6
scaly	no	0
fibrous	white	1
fibrous	brown	2
fibrous	yellow	0
fibrous	bruises	0
fibrous	no	4
smooth	white	1
smooth	brown	2
smooth	yellow	2
smooth	bruises	5
smooth	no	0
white	bruises	5
white	no	1
brown	bruises	3
brown	no	2
yellow	bruises	3

flat	smooth	1
flat	white	3
yellow	no	0

Langkah 4. Memangkas *itemset* yang tidak memenuhi *minimum support*. Hasil pemangkasan dapat dilihat pada tabel 3.5.

Tabel 3.5 Hasil Pemangkasan 2-*itemset*

Item-1	*Item-2*	Jumlah
edible	convex	4
edible	bell	3
edible	scaly	4
edible	fibrous	4
edible	white	3
edible	brown	3
edible	yellow	3
edible	bruises	6
edible	no	4
poisonous	convex	3
poisonous	smooth	3
poisonous	white	3
poisonous	bruises	5
Item-1	*Item-2*	Jumlah
convex	scaly	3
convex	smooth	3
convex	brown	4
convex	bruises	6
flat	white	3
bell	bruises	3
scaly	white	4
scaly	bruises	6
fibrous	no	4
smooth	bruises	5
white	bruises	5
brown	bruises	3
yellow	bruises	3

Langkah 5. Mencari kandidat 3-*itemset*. Calon kandidat dapat dilihat pada tabel 3.6

Tabel 3.6 Kandidat 3-*itemset*

Item-1	*Item-2*	*Item-3*	Jumlah
edible	convex	scaly	2
edible	convex	brown	2
edible	convex	bruises	3
edible	bell	bruises	3
edible	scaly	white	3
edible	scaly	bruises	5
edible	fibrous	no	4
edible	white	bruises	2
edible	brown	bruises	1
edible	yellow	bruises	3
poisonous	convex	smooth	2
poisonous	white	bruises	3
poisonous	convex	bruises	3
poisonous	smooth	bruises	3
convex	scaly	bruises	3
convex	smooth	bruises	3
convex	brown	bruises	3
scaly	white	bruises	4

Langkah 6. Memangkas kandidat 3-*itemset* yang tidak memenuhi *minimum support*. Hasil pemangkasan dapat dilihat pada tabel 3.7.

Tabel 3.7 Hasil pemangkasan 3-*itemset*

Item-1	*Item-2*	*Item-3*	Jumlah
edible	convex	bruises	3
edible	bell	bruises	3
edible	scaly	bruises	4
edible	fibrous	no	4
edible	yellow	bruises	3
poisonous	convex	bruises	3
poisonous	white	bruises	3
poisonous	smooth	bruises	3
convex	scaly	bruises	3
convex	smooth	bruises	3
convex	brown	bruises	3
scaly	white	bruises	4

Langkah 7. Mencari kandidat 4-*itemset*. Calon kandidat dapat dilihat pada tabel 3.8

Tabel 3.8 Kandidat 4-*itemset*

Item-1	*Item-2*	*Item-3*	*Item-4*	Jumlah
edible	convex	scaly	bruises	2
edible	scaly	white	bruises	2

Langkah 8. *Stop* karena kandidat 4-*itemset* tidak memenuhi nilai minimum *support.*

Langkah 9. Menghitung nilai minimum *confidence* dari *2-itemset* dan *3-itemset* yang telah didapatkan. Hasil perhitungan dapat dilihat pada tabel 3.9 untuk *2-itemset* dan tabel 3.10 untuk *3-itemset*.

Tabel 3.9 Nilai Minimum *Confidence* 2-*itemset*

Rules	*Support* (%)	*Confidence* (%)
edible → convex	4/15 = 26.67%	4/10 = 40%
convex → edible	4/15 = 26.67%	4/7 = 57.14%
edible → bell	3/15 = 20%	3/10 = 30%
bell → edible	3/15 = 20%	3/3 = 100%
edible → scaly	4/15 = 26.67%	4/10 = 40%
scaly → edible	4/15 = 26.67%	4/6 = 66.67%
edible → fibrous	4/15 = 26.67%	4/10 = 40%
fibrous → edible	4/15 = 26.67%	4/4 = 100%
edible → white	3/15 = 20%	3/10 = 30%
white → edible	3/15 = 20%	3/6 = 50%
edible → brown	3/15 = 20%	3/10 = 30%
brown → edible	3/15 = 20%	3/5 = 60%
edible → yellow	3/15 = 20%	3/10 = 30%
yellow → edible	3/15 = 20%	3/3 = 100%
edible → bruises	6/15 = 40%	6/10 = 60%
bruises → edible	6/15 = 40%	6/11 = 54.55%
edible → no	4/15 = 26.67%	4/10 = 40%
no → edible	4/15 = 26.67	4/4 = 100%
poisonous → convex	3/15 = 20%	3/5 = 60%
convex → poisonous	3/15 = 20%	3/7 = 42.86%

poisonous → smooth	3/15 = 20%	3/5 = 60%
smooth → poisonous	3/15 = 20%	3/5 = 60%
poisonous → white	3/15 = 20%	3/5 = 60%
white → poisonous	3/15 = 20%	3/6 = 50%
poisonous → bruises	5/15 = 33.33%	5/5 = 100%
bruises → poisonous	5/15 = 33.33%	5/11 = 45.45%
convex → scaly	3/15 = 20 %	3/7 = 42.86 %
scaly → convex	3/15 = 20 %	3/6 = 50 %
convex → smooth	3/15 = 20 %	3/7 = 42.86 %
smooth → convex	3/15 = 20 %	3/5 = 60%
convex → brown	4/15 = 26.67 %	4/7 = 57,14 %
brown → convex	4/15 = 26.67 %	4/5 = 80 %
convex → bruises	6/15 = 40 %	6/7 = 85.71 %
bruises → convex	6/15 = 40 %	6/11 = 54.55 %
flat → white	3/15 = 20%	3/4 = 75%
white → flat	3/15 = 30%	3/6 = 50%
bell→ bruises	3/15 = 20 %	3/3 = 100 %
bruises → bell	3/15 = 20%	3/11 = 27.27%
scaly → white	4/15 = 26.67 %	4/6 = 66.67%
white → scaly	4/15 = 26.67 %	4/6 = 66.67%
scaly → bruises	6/15 = 40 %	6/6 = 100 %
bruises → scaly	6/15 = 40 %	6/11 = 45.46 %
fibrous → no	4/15 = 26.67 %	4/4 = 100 %
no → fibrous	4/15 = 26.67 %	4/4 = 100 %
smooth → bruises	5/15 = 33.33%	5/5 = 100 %
bruises → smooth	5/15 = 33.33 %	5/11 = 54.55%
white → bruises	5/15 = 33.33 %	5/6 = 83.33 %
bruises → white	5/15 = 33.33 %	5/11 = 45.46 %
brown → bruises	3/15 = 20 %	3/5 = 60 %
bruises → brown	3/15 = 20 %	3/11 = 27.27 %
yellow → bruises	3/15 = 20 %	3/3 = 100 %
bruises → yellow	3/15 = 20 %	3/11 = 27.27 %

Tabel 3.10 Nilai Minimum *Confidence* 3-*Itemset*

Itemset	*Support* (%)	*Confidence* (%)
{edible, convex} → bruises	3/15 = 20 %	3/4 = 75%
{edible, bruises} → convex	3/15 = 20 %	3/6 = 50%
{convex, bruises} → edible	3/15 = 20 %	3/6 = 50%
{edible, bell} → bruises	3/15 = 20 %	3/3 = 100%
{edible, bruises} → bell	3/15 = 20 %	3/6 = 50%
{bell, bruises} → edible	3/15 = 20 %	3/3 = 100%
{edible, scaly} → bruises	4/15 = 26.67%	4/4 = 100%
{edible, bruises} → scaly	4/15 = 26.67%	4/6 = 66.67%
{scaly, bruises} → edible	4/15 = 26.67%	4/6 = 66.67%
{edible, fibrous} → no	4/15 = 26.67%	4/4 = 100%
{edible, no} → fibrous	4/15 = 26.67%	4/4 =100%
{fibrous, no} → edible	4/15 = 26.67%	4/4 = 100%
{edible, yellow} → bruises	3/15 = 20 %	3/3 =100%
{edible, bruises} → yellow	3/15 = 20 %	3/6 = 50%
{yellow, bruises} → edible	3/15 = 20 %	3/3 = 100%
{poisonous, convex} → bruises	3/15 = 20 %	3/3 = 100%
{poisonous, bruises} → convex	3/15 = 20 %	3/5 = 60%
{convex, bruises} → poisonous	3/15 = 20 %	3/6 = 50%
{poisonous, smooth} → bruises	3/15 = 20 %	3/3 = 100%
{poisonous, bruises} → smooth	3/15 = 20 %	3/6 = 50%
{smooth, bruises} → poisonous	3/15 = 20 %	3/5 = 60%
{poisonous, white} → bruises	3/15 = 20 %	3/3 = 100%
{poisonous, bruises} → white	3/15 = 20 %	3/5 = 60%
{white, bruises} → poisonous	3/15 = 20 %	3/5 = 60%
{convex, scaly} → bruises	3/15 = 20 %	3/3 = 100 %
{convex, bruises} → scaly	3/15 = 20 %	3/6 = 50 %
{scaly, bruises} → convex	3/15 = 20 %	3/5 = 60 %
{convex, smooth} → bruises	3/15 = 20 %	3/3 = 100 %
{convex, bruises} → smooth	3/15 = 20 %	3/6 = 50 %
{smooth, bruises} → convex	3/15 = 20 %	3/6 = 50 %
{convex, brown} → bruises	3/15 = 20 %	3/4 = 75 %
{convex, bruises}→ brown	3/15 = 20 %	3/6 = 50 %
{brown, bruises} → convex	3/15 = 20 %	3/3 = 100 %
{scaly, white} → bruises	4/15 = 26.67%	4/4 = 100%
{scaly, bruises} → white	4/15 = 26.67%	4/6 = 66.67%

{white, bruises} → scaly	4/15 = 26.67%	4/5 = 80%

Misalkan ditetapkan nilai minimum *confidence* adalah 80% maka aturan yang terbentuk dapat dilihat pada gambar 3.9.

JIKA bell MAKA edible (support=20%, conf=100%)
JIKA fibrous MAKA edible (support=26.67, conf=100%)
JIKA yellow MAKA edible (support=20%, conf=100%)
JIKA no MAKA edible (support=26.67%, conf=100%)
JIKA poisonous MAKA bruises (support=33.33%, conf=100%)
JIKA brown MAKA convex (support=26.67%, conf=80%)
JIKA convex MAKA bruises (support=40%, conf=85.71%)
JIKA bell MAKA bruises (support=20%, conf=100%)
JIKA scaly MAKA bruises (support=40%, conf=100%)
JIKA fibrous MAKA no (support=26.67%, conf=100%)
JIKA no MAKA fibrous (support=26.67, conf=100%)
JIKA smooth MAKA bruises (support=33.33%, conf=100%)
JIKA white MAKA bruises (support=33.33%, conf=83.33%)
JIKA yellow MAKA bruises (support=20%, conf=100%)
JIKA edible dan bell MAKA bruises (support=20%, conf=100%)
JIKA bell dan bruises MAKA edible (support=20%, conf=100%)
JIKA edible dan scaly MAKA bruises (support=26.67%, conf=100%)
JIKA edible dan fibrous MAKA no (support=26.67%, conf=100%)
JIKA edible dan no MAKA fibrous (support=26.67%, conf=100%)
JIKA fibrous dan no MAKA edible (support=26.67%, conf=100%)
JIKA edible dan yellow MAKA bruises (support=20%,conf=100%)
JIKA yellow dan bruises MAKA edible (support=20%,conf=100%)
JIKA poisonous dan convex MAKA bruises (support=20%, conf=100%)
JIKA poisonous dan smooth MAKA bruises (support=20%, conf=100%)
JIKA poisonous dan white MAKA bruises (support=20%, conf=100%)
JIKA convex dan scaly MAKA bruises (support=20%, conf=100%)
JIKA convex dan smooth MAKA bruises (support=20%, conf=100%)
JIKA brown dan bruises MAKA convex (support=20%, conf=100%)
JIKA scaly dan white MAKA bruises (support=26.67%, conf=100%)
JIKA white dan bruises MAKA scaly (support=26.67%,conf=80%)

Gambar 3.9 Aturan yang terbentuk menggunakan Algoritma *Apriori*

3.3.2 Perhitungan manual menggunakan algoritma *Perfect Hashing and Pruning* (*PHP*)

Langkah 1. Membentuk D0 yang berisi data yang terdapat pada dataset, dapat dilihat pada tabel 3.11.

Tabel 3.11 Data *Mushroom* (D0)

TID	*Itemset*
001	edible,convex,scaly,white,bruises
002	edible,convex,fibrous,brown,no
003	edible,sunken,fibrous,gray,no
004	edible,flat,fibrous,white,no
005	poisonous,convex,smooth,brown,bruises
006	poisonous,convex,scaly,white,bruises
007	poisonous,convex,smooth,brown,bruises
008	edible,bell,smooth,yellow,bruises
009	edible,convex,scaly,brown,bruises
010	edible,bell,scaly,yellow,bruises
011	edible,bell,scaly,white,bruises
012	poisonous,flat,scaly,white,bruises
013	poisonous,flat,smooth,white,bruises
014	edible,flat,fibrous,brown,no
015	edible,convex,smooth,yellow,bruises

Langkah 2. Memetakan setiap *item* ke nilai yang berbeda. *Location* merupakan *key* setiap item dan *support* merupakan jumlah *item* yang terdapat pada D0. Hasil pemetaan dapat dilihan pada tabel 3.12.

Tabel 3.12 Kandidat *1-itemset* (H1)

Location	*Itemset*	*Support*
1	edible	10
2	poisonous	5
3	convex	7
4	sunken	1
5	flat	4
6	bell	3
7	scaly	6
8	fibrous	4

Location	*Itemset*	*Support*
9	smooth	5
10	white	6
11	brown	5
12	gray	1
13	yellow	3
14	bruises	11
15	no	4

Langkah 3. *Pruning* untuk item yang tidak memenuhi nilai minimum *support*. Hasil pemangkasan terdapat pada tabel 3.13.

Misalkan *minimum support* = 20% (3 dari 15 transaksi)

Tabel 3.13 *Frequent 1-itemset* (F1)

Itemset	*Support*
H(1)	10
H(2)	5
H(3)	7
H(5)	4
H(6)	3
H(7)	6
H(8)	4
H(9)	5
H(10)	6
H(11)	5
H(13)	3
H(14)	11
H(15)	4

Langkah 4. Menghapus *item* tabel D0 yang tidak memenuhi nilai minimum *support* (tidak terdapat pada tabel F1). Hasil dari pemangkasan dapat dilihat pada tabel 3.14.

Tabel 3.14 Database *Pruning 1-itemset*(D1)

TID	*Itemset*
001	edible,convex,scaly,white,bruises
002	edible,convex,fibrous,brown,no
003	edible,fibrous,no
004	edible,flat,fibrous,white,no
005	poisonous,convex,smooth,brown,bruises
006	poisonous,convex,scaly,white,bruises
007	poisonous,convex,smooth,brown,bruises
008	edible,bell,smooth,yellow,bruises
009	edible,convex,scaly,brown,bruises
010	edible,bell,scaly,yellow,bruises
011	edible,bell,scaly,white,bruises
012	poisonous,flat,scaly,white,bruises
013	poisonous,flat,smooth,white,bruises
014	edible,flat,fibrous,brown,no
015	edible,convex,smooth,yellow,bruises

Langkah 5. Mendata kandidat *2-itemset* ke lokasi yang berbeda. Kandidat *2-itemset* dapat dilihat pada tabel 3.15

Tabel 3.15 Kandidat *2-itemset* (H2)

Loc	*Itemset*	*Supp*
1	edible, convex	4
2	edible, scaly	4
3	edible, white	3
4	edible.bruises	6
5	edible, fibrous	4
6	edible, brown	3
7	edible, no	4
8	edible,flat	2
9	edible,bell	3
10	edible,smooth	2
11	edible, yellow	3
12	poisonous,convex	3
13	poisonous,smooth	3
14	poisonous,brown	2
15	poisonous,bruises	5
16	poisonous,scaly	2
17	poisonous,white	3
18	poisonous,flat	2
19	convex, scaly	3
20	convex, fibrous	1
21	convex, smooth	3
22	convex, white	2
23	convex, brown	4
24	convex, yellow	1
25	convex, bruises	6
26	convex, no	1
27	flat, fibrous	2
28	flat, scaly	1
29	flat, smooth	1
30	flat, white	2
31	flat, brown	1
32	flat, bruises	2
33	flat, no	2
34	bell, scaly	2
35	bell, smooth	2
36	bell, white	1
37	bell, yellow	2
38	bell, bruises	3
39	scaly, white	4
40	scaly, brown	1
41	scaly, yellow	1
42	scaly, bruises	6
43	fibrous, white	1
44	fibrous, brown	2
45	fibrous, no	4
46	smooth, white	1
47	smooth, brown	2
48	smooth, yellow	2
49	smooth, bruises	5
50	white,bruises	5
51	white, no	1
52	brown, bruises	3
53	brown, no	2
54	yellow, bruises	3

Langkah 6. Memangkas kandidat *2-itemset* yang tidak memenuhi minimum *support*. Hasil pemangkasan dapat dilihat pada tabel 3.16.

Tabel 3.16 *Frequent 2-itemset* (F2)

Itemset	*Support*
H(1)	4
H(2)	4
H(3)	3
H(4)	6
H(5)	4
H(6)	3
H(7)	4
H(9)	3
H(11)	3
H(12)	3
H(13)	3
H(15)	5
H(17)	3

Itemset	*Support*
H(19)	3
H(21)	3
H(23)	4
H(25)	6
H(38)	3
H(39)	4
H(42)	6
H(45)	4
H(49)	5
H(50)	5
H(52)	3
H(54)	3

Langkah 7. *Pruning* database D1 yang itemsetnya tidak memenuhi nilai minimum support. Hasil pemangkasan dapat dilihat pada tabel 3.17.

Tabel 3.17 Database *Pruning 2-itemset*(D2)

TID	*Itemset*
001	edible,convex,scaly,bruises
002	edible,convex,fibrous,no
003	edible,fibrous,no
004	edible,fibrous,no
005	convex,bruises
006	convex,scaly,bruises
007	convex,bruises
008	edible,bell,yellow,bruises
009	edible,convex,scaly,bruises
010	edible,bell,scaly,yellow,bruises
011	edible,bell,scaly,bruises
012	poisonous,scaly,bruises
013	bruises
014	edible,fibrous,no
015	edible,convex,yellow,bruises

Langkah 8. Mendata kandidat *3-itemset*. Kandidat *3-itemset* dapat dilihat pada tabel 3.18.

Tabel 3.18 Kandidat *3-itemset* (H3)

Location	*Itemset*	*Support*
1	edible,convex,scaly	2
2	edible,convex,bruises	3
3	edible,scaly,bruises	4
4	edible,fibrous,no	4
5	edible,bell,bruises	3
6	edible,yellow,bruises	3
7	convex,scaly,bruises	2
8	edible,convex,no	1
9	edible,convex,fibrous	1
10	edible,bell,yellow	2
11	edible,bell,scaly	2
12	bell,scaly,yellow	1
13	bell,scaly,bruises	2
14	scaly,yellow,bruises	1
15	edible,convex,yellow	1
16	convex,yellow,bruises	1
17	edible,convex,yellow	1

Langkah 9. Memangkas kandidat *3-itemset* yang tidak memenuhi nilai minimum *support*. Hasil pemangkasan pada tabel 3.19.

Tabel 3.19 *Frequent 3-itemset* (F3)

Itemset	*Support*
H(2)	3
H(3)	4
H(4)	4
H(5)	3
H(6)	3

Langkah 10. *Pruning* database D2 yang itemsetnya tidak memenuhi nilai minimum support. Hasil pemangkasan dapat dilihat pada tabel 3.20.

Tabel 3.20. Database *Pruning 3-itemset* (D3)

TID	*Itemset*
003	edible,fibrous,no
004	edible,fibrous,no
014	edible,fibrous,no

Langkah 11. Proses berhenti karena tidak ada kandidat untuk membentuk 4-*itemset*.

Langkah 12. Menghitung nilai minimum *confidence* dari *2-itemset* dan *3-itemset* yang telah memnuhi minimum *support*, Hasil perhitungan untuk *2-itemset* dapat dilihat pada pada tabel 3.21 dan untuk *3-itemset* pada tabel 3.22.

Tabel 3.21 Perhitungan Nilai *confidence 2-itemset*

Rules	*Support* (%)	*Confidence* (%)
edible → convex	4/15 = 26.67%	4/10 = 40%
convex → edible	4/15 = 26.67%	4/7 = 57.14%
edible → bell	3/15 = 20%	3/10 = 30%
bell → edible	3/15 = 20%	3/3 = 100%
edible → scaly	4/15 = 26.67%	4/10 = 40%
scaly → edible	4/15 = 26.67%	4/6 = 66.67%
edible → fibrous	4/15 = 26.67%	4/10 = 40%
fibrous → edible	4/15 = 26.67%	4/4 = 100%
edible → white	3/15 = 20%	3/10 = 30%
white → edible	3/15 = 20%	3/6 = 50%
edible → brown	3/15 = 20%	3/10 = 30%
brown → edible	3/15 = 20%	3/5 = 60%
edible → yellow	3/15 = 20%	3/10 = 30%
yellow → edible	3/15 = 20%	3/3 = 100%
edible → bruises	6/15 = 40%	6/10 = 60%
bruises → edible	6/15 = 40%	6/11 = 54.55%
edible → no	4/15 = 26.67%	4/10 = 40%
no → edible	4/15 = 26.67	4/4 = 100%
poisonous → convex	3/15 = 20%	3/5 = 60%
convex → poisonous	3/15 = 20%	3/7 = 42.86%
poisonous → smooth	3/15 = 20%	3/5 = 60%

smooth → poisonous	3/15 = 20%	3/5 = 60%
poisonous → white	3/15 = 20%	3/5 = 60%
white → poisonous	3/15 = 20%	3/6 = 50%
poisonous → bruises	5/15 = 33.33%	5/5 = 100%
bruises → poisonous	5/15 = 33.33%	5/11 = 45.45%
convex → scaly	3/15 = 20 %	3/7 = 42.86 %
scaly → convex	3/15 = 20 %	3/6 = 50 %
convex → smooth	3/15 = 20 %	3/7 = 42.86 %
smooth → convex	3/15 = 20 %	3/5 = 60%
convex → brown	4/15 = 26.67 %	4/7 = 57,14 %
brown → convex	4/15 = 26.67 %	4/5 = 80 %
convex → bruises	6/15 = 40 %	6/7 = 85.71 %
bruises → convex	6/15 = 40 %	6/11 = 54.55 %
flat → white	3/15 = 20%	3/4 = 75%
white → flat	3/15 = 30%	3/6 = 50%
bell→ bruises	3/15 = 20 %	3/3 = 100 %
bruises → bell	3/15 = 20%	3/11 = 27.27%
scaly → white	4/15 = 26.67 %	4/6 = 66.67%
white → scaly	4/15 = 26.67 %	4/6 = 66.67%
scaly → bruises	6/15 = 40 %	6/6 = 100 %
bruises → scaly	6/15 = 40 %	6/11 = 45.46 %
fibrous → no	4/15 = 26.67 %	4/4 = 100 %
no → fibrous	4/15 = 26.67 %	4/4 = 100 %
smooth → bruises	5/15 = 33.33%	5/5 = 100 %
bruises → smooth	5/15 = 33.33 %	5/11 = 54.55%
white → bruises	5/15 = 33.33 %	5/6 = 83.33 %
bruises → white	5/15 = 33.33 %	5/11 = 45.46 %
brown → bruises	3/15 = 20 %	3/5 = 60 %
bruises → brown	3/15 = 20 %	3/11 = 27.27 %
yellow → bruises	3/15 = 20 %	3/3 = 100 %
bruises → yellow	3/15 = 20 %	3/11 = 27.27 %

Tabel 3.22 Perhitungan Nilai *confidence* 3-*itemset*

Itemset	*Support* (%)	*Confidence* (%)
{edible, convex} → bruises	3/15 = 20 %	3/4 = 75%
{edible, bruises} → convex	3/15 = 20 %	3/6 = 50%
{convex, bruises} → edible	3/15 = 20 %	3/6 = 50%
{edible, bell} → bruises	3/15 = 20 %	3/3 = 100%
{edible, bruises} → bell	3/15 = 20 %	3/6 = 50%
{bell, bruises} → edible	3/15 = 20 %	3/3 = 100%
{edible, scaly} → bruises	4/15 = 26.67%	4/4 = 100%
{edible, bruises} → scaly	4/15 = 26.67%	4/6 = 66.67%
{scaly, bruises} → edible	4/15 = 26.67%	4/6 = 66.67%
{edible, fibrous} → no	4/15 = 26.67%	4/4 = 100%
{edible, no} → fibrous	4/15 = 26.67%	4/4 =100%
{fibrous, no} → edible	4/15 = 26.67%	4/4 = 100%
{edible, yellow} → bruises	3/15 = 20 %	3/3 =100%
{edible, bruises} → yellow	3/15 = 20 %	3/6 = 50%
{yellow, bruises} → edible	3/15 = 20 %	3/3 = 100%

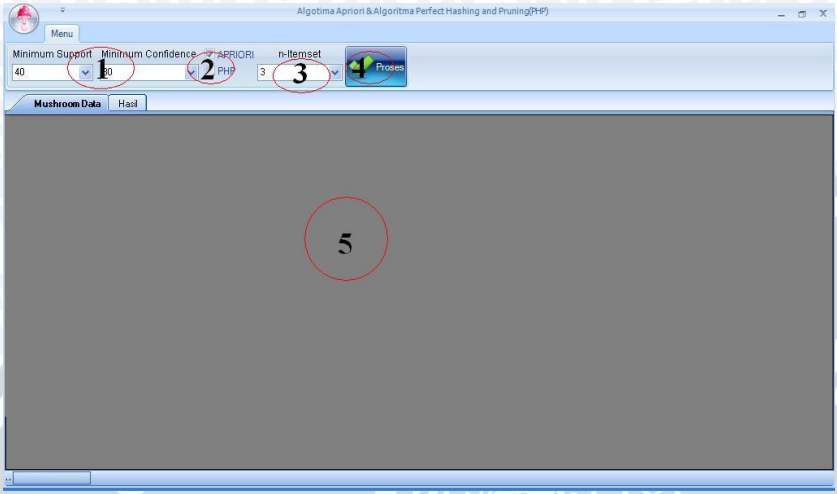
Aturan yang didapatkan dapat dilihat pada gambar 3.10 jika nilai minimum *confidence* = 80%.

JIKA bell MAKA edible (support=20%, conf=100%)
JIKA fibrous MAKA edible (support=26.67, conf=100%)
JIKA yellow MAKA edible (support=20%, conf=100%)
JIKA no MAKA edible (support=26.67%, conf=100%)
JIKA poisonous MAKA bruises (support=33.33%, conf=100%)
JIKA brown MAKA convex (support=26.67%, conf=80%)
JIKA convex MAKA bruises (support=40%, conf=85.71%)
JIKA bell MAKA bruises (support=20%, conf=100%)
JIKA scaly MAKA bruises (support=40%, conf=100%)
JIKA fibrous MAKA no (support=26.67%, conf=100%)
JIKA no MAKA fibrous (support=26.67, conf=100%)
JIKA smooth MAKA bruises (support=33.33%, conf=100%)
JIKA white MAKA bruises (support=33.33%, conf=83.33%)
JIKA yellow MAKA bruises (support=20%, conf=100%)
JIKA edible dan bell MAKA bruises (support=20%, conf=100%)
JIKA bell dan bruises MAKA edible (support=20%, conf=100%)
JIKA edible dan scaly MAKA bruises (support=26.67%, conf=100%)
JIKA edible dan fibrous MAKA no (support=26.67%, conf=100%)
JIKA edible dan no MAKA fibrous (support=26.67%, conf=100%)
JIKA fibrous dan no MAKA edible (support=26.67%, conf=100%)
JIKA edible dan yellow MAKA bruises (support=20%,conf=100%)
JIKA yellow dan bruises MAKA edible (support=20%,conf=100%)

Gambar 3.10 Aturan yang terbentuk menggunakan Algoritma *PHP*

3.4 Perancangan Antarmuka

Antarmuka dari proses pencarian *association rule* diilustrasikan pada gambar 3.11.



Gambar 3.11 Perancangan Antarmuka

Keterangan gambar 3.11 :

1. Parameter
   Menu ini dirancang untuk memilih nilai minimum *support* dan minimum *confedence* sebagai parameter dalam pembentukan *rule.*
2. Pilihan Algoritma
   Menu ini dirancang untuk memilih algortima *Apriori* atau algoritma *PHP* yang akan digunakan untuk menemukan *association rule.*
3. n-itemset
   Menu ini dirancang untuk membatasi pembentukan item sesuai dengan jumlah n-itemset yang dipilih.
4. Proses
   Button ini dirancang untuk mengawali bekerjanya algoritma dalam menemukan *association rule.*
5. Tampilan *dataset*
   Menampilkan isi dari *dataset* yang akan diproses untuk menemukan *association rule.*

3.5 **Perancangan Uji Coba**

Pada subbab ini akan dijelaskan mengenai perancangan uji coba algoritma *Apriori* dan algoritma *PHP* berdasarkan jumlah *rule* yang terbentuk, dan *lift ratio assoction rule*. Rancangan uji coba untuk

kedua algoritma sebagai berikut:

1. Pengujian jumlah rule.

Pengujian ini dilakukan untuk menghitung jumlah *rule* yang terbentuk dari masing-masing algoritma. Parameter yang digunakan pada pengujian ini yaitu minimum *support* dan minimum *confidence*.

Pengujian jumlah *rule* ini menggunakan data sampel yang telah ditetapkan. Tabel 3.24 adalah rancangan tabel uji coba yang digunakan untuk mencatat hasil dalam proses pengujian jumlah *rule*

Tabel 3.24. Uji Coba Jumlah *Rule* yang Terbentuk

Minimum *Support*	Minimum *Confidence*	Jumlah *Rule*	
		Apriori	*Perfect Hashing & Pruning* (*PHP*)
50%	80%		
	90%		
	100%		
60%	80%		
	90%		
	100%		
70%	80%		
	90%		
	100%		

Keterangan :

Minimum *Support* : nilai minimum *support* yang akan diuji

Minimum *Confidence* : nilai minimum *confidence* yang akan diuji

Jumlah *rule apriori* : jumlah *rule* yang terbentuk dengan algortima *Apriori*

Jumlah *rule PHP* : jumlah *rule* yang terbentuk dengan algortima *Perfect Hashing and Pruning*

2. Pengujian *Lift Ratio Rule*

Pengujian ini dilakukan untuk melihat kuat tidaknya *rule* yang terbentuk dari masing-masing algoritma. Pengujian dilakukan denga menghitung *lift ratio* pada persamaan 2.4. Tabel rancangan ujicoba kekuatan *rule* dapat dilhat pada Tabel 3.25.

Tabel 3.25. Uji Coba *Lift Ratio Rule*

Rule	*Confidence*	*Benchmark confidence*	*Lift ratio*

Keterangan :
Rule : *rule* yang terbentuk oleh sistem
Confidence : nilai *confidence* yang dihasilkan *rule*
Benchmark Confidence : nilai *benchmark confidence* dari *rule*
Lift ratio : nilai *lift ratio* yang dihasilkan.

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

4.1 Perangkat Sistem

Perangkat sistem terdiri dari perangkat lunak dan perangkat keras yang digunakan dalam penelitian.

4.1.1 Perangkat Lunak

Perangkat lunak yang digunakan dalam penelitian ni adalah :

1. Sistem operasi Microsoft Windows XP Professional Edition *Service Pack* 3.
2. Microsoft Visual C# 2008 Express Edition sebagai *software development* dalam pembuatan sistem, baik untuk proses menemukan *associotion rule* dengan algoritma *Apriori* dan algoritma *Perfect Hashing and Pruning* serta proses pengujian sistemnya.
3. Microsoft SQL Server 7.0 sebagai DBMS (*Database Management System*).

4.1.2 Perangkat Keras

Perangkat keras yang digunakan dalam penelitian ini adalah :

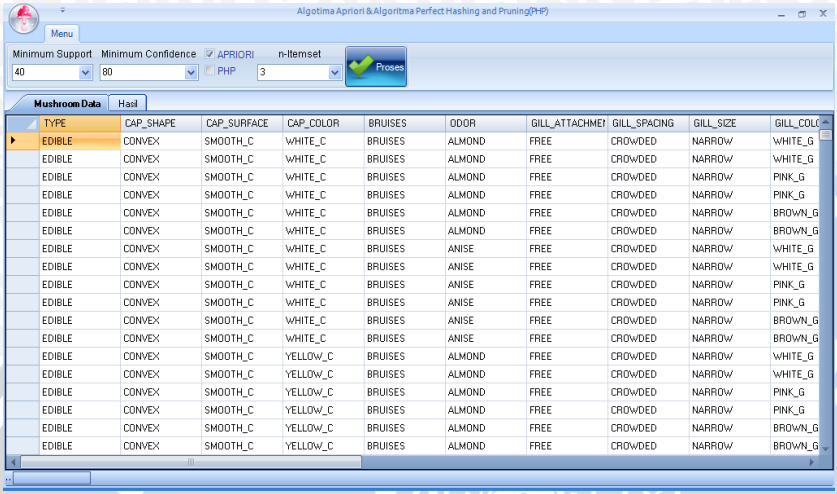
1. Prosesor Intel® Atom Processor N550(1,5 GHz, 1M L2 cache)
2. Memori 1 GB DDR3
3. *Harddisk* dengan kapasitas 160 GB
4. *Keyboard*

4.2 Implementasi Program

Pada subbab implementasi program ini akan dijelaskan mengenai implementasi dari rancangan sistem yang dijelaskan sebelumnya pada bab 3.2.

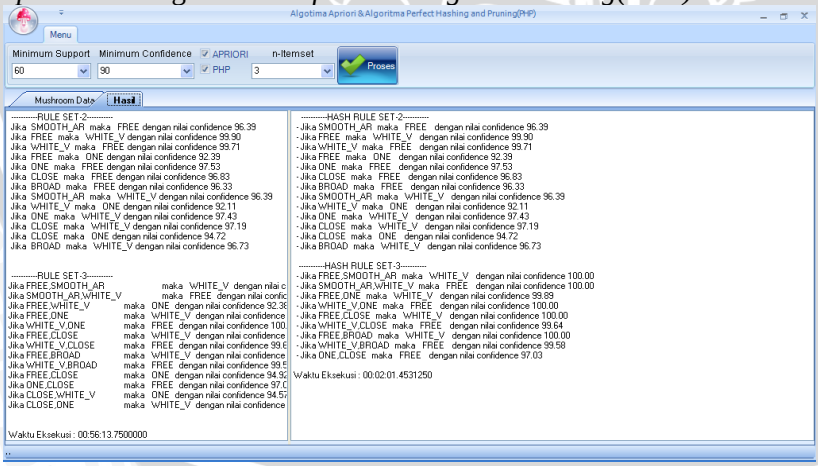
4.2.1 Implementasi Antarmuka

Implementasi antarmuka dari sistem terdiri dari 1 buah *form* yaitu *form Menu*. *Form Menu* ini terdiri dari 2 buah *tab*, *tab* pertama yaitu *Mushroom Data* yang menampilkan dataset yang disimpan dalam database, dan *tab* kedua yaitu *Hasil* yang menampilkan *rule* yang terbentuk.



Gambar 4.1 *Form Menu*

Pada *form Menu* yang diilustrasikan pada gambar 4.1 terdapat 3 buah *ComboBox* yakni *ComboBox Minimum Support, ComboBox Minimum Confidence* dan *ComboBox* Jumlah Itemset, 2 buah *CheckBox* yakni *CheckBox Apriori* dan *CheckBox PHP*, 1 buah *Button* yaitu *Button* Proses, 1 buah *DataGridView* untuk menampilkan dataset dan 2 buah *TextBox* untuk menampilkan hasil dari Algoritma *Apriori* dan Algoritma *Perfect Hashing and Pruning(PHP).*



Gambar 4.2 Tampilan Hasil *Association Rule*

Untuk menemukan *rule*, *User* dapat memilih nilai Minimum *support*, nilai Minimum *confidence* dan batasan jumlah itemset yang akan terbentuk dengan memilih pada masing-masing *ComboBox*. Sedangkan untuk memilih algoritma mana yang akan digunakan untuk menemukan *Association rule* dapat memberi *check* pada *CheckBox Apriori* atau *CheckBox PHP* bahkan dapat dilakukan proses dengan memberi *check* pada kedua *CheckBox*. Setelah itu tekan tombol Proses dan hasil akan keluar pada *TextBox* seperti gambar 4.2.

4.2.2 Algoritma *Apriori*

Dalam algoritma *Apriori* untuk implementasinya maka dilakukan proses pembacaan data, proses pembentukan kandidat 1-*itemset*, proses pemangkasan dan proses pembentukan aturan (*rule*).

4.2.2.1 Proses Pembacaan Data

Dataset yang disimpan pada dabatase dipanggil saat pertama kali proses dijalankan. Dataset akan ditampilkan pada *DataGridView* dangan cara pada *Sourcecode* 4.1.

```
private void Form1_Load(object sender, EventArgs e)
{
this.mushroomFullTableAdapter.Fill(this._MyDatabase
_2DataSet1.MushroomFull);
}
```

Sourcecode 4.1 Pemanggilan Dataset pada Database

Setelah dataset ditampilkan pada *DataGridView* proses selanjutnya yaitu menyimpan dataset tersebut kedalam sebuah *Array*. Proses penyimpanan data pada *Array* dapat dilihat pada *Sourcecode* 4.2.

```
Public void saveToArray()
{
jumBaris = dataGridView1.RowCount;
jumKolom = dataGridView1.ColumnCount;
Data = new string[jumBaris, jumKolom];
for (int i = 0; i <= jumBaris-1; i++)
{
  StringBuilder sb = new StringBuilder();
  List<string> tmpDic = new List<string>();
  for (int j = 0; j <= jumKolom-1; j++)
  {
    Data[i, j] =
Convert.ToString((dataGridView1.Rows[i].Cells[j].Va
lue));
    if (Data[i, j].Trim() != "")
    {
      sb.Append(Data[i, j] + ",");
      tmpDic.Add(Data[i, j]);
    }
  }
if (sb.ToString().Trim()!="")
Dic0.Add(Convert.ToString (i + 1) , sb);
if (tmpDic.Count != 0)
{
  dataItem.Add(dataItem.Count, toStr(tmpDic));
}
}
```

Sourcecode 4.2 Penyimpan dataset ke Array

4.2.2.2 Proses Pembentukan Kandidat 1-*itemset*

Kandidat 1-*itemset* didapatkan dengan cara menghitung frekuensi setiap item yang ditemukan. Proses perhitungan dapat dilihat pada *Sourcecode* 4.3.

```
private void getFreq()
 {
 ItemSet = new Dictionary<string, int>();
 for (int j = 0; j <= jumKolom - 1; j++)
 {
   for (int i = 0; i <= jumBaris-1 ; i++)
   { addItem(Data[i, j]);
     if (!ItemCols.ContainsKey(Data[i, j]))
     { ItemCols.Add(Data[i, j], j);}
     if (!ItemDic.ContainsKey(i))
     {
       List<string> tempList = new List<string>();
       tempList.Add(Data[i, j]);
       ItemDic.Add(i, tempList);
     }
     else
     { ItemDic[i].Add(Data[i, j]);}
   }
 }}
```

Sourcecode 4.3 Frekuensi Itemset

4.2.2.3 Proses Pemangkasan

Setelah dilakukan proses perhitungan jumlah itemset maka dilakukan proses pemangkasan. Proses pemangkasan ini dilakukan sesuai dengan nilai minimum *support* yang dipilih oleh *User*. Proses dari pemangkasan ini dapat dilihat pada *Sourcecode* 4.4.

```
private void pemangkas()
{
  Dictionary<string, int> tempItem = new
Dictionary<string, int>();
  foreach (KeyValuePair<string, int> strItm in
ItemSet)
  {
   if (strItm.Value >= supp)
   {
    tempItem.Add(strItm.Key,strItm.Value);
   }
  }
 ItemSet = new Dictionary<string,int>();
 ItemSet = tempItem;
}
```

Sourcecode 4.4 Pemangkasan Itemset

4.2.2.4 Proses Pembentukan Kandidat n-*itemset*

Proses pembentukan Kandidat n-*itemset* dapat dilihat pada *Sourcecode* 4.5. Proses ini mengkombinasikan item dari *(k-1)-itemset* yang terbentuk. Dan proses ini juga mneghitung jumlah kandidat yag terbentuk.

```
private void prosesApriori()
{
  Dictionary<List<string>, int> iL = new
    Dictionary<List<string>, int>(iC);
  iC = new Dictionary<List<string>, int>();

  for (int i = 0; i < iL.Count; i++)
  {
    for (int j = i + 1; j < iL.Count; j++)
   {    if (iL.ElementAt(i).Key[0] !=
iL.ElementAt(j).Key[0]) break;
    List<string> tempiC = new List<string>();
    tempiC = joinItems(iL.ElementAt(i).Key,
             iL.ElementAt(j).Key, nItem);
   }
  }
}
```

Sourcecode 4.5 Pembentukan Kandidat n-*itemset*

Pada Proses pembentukan Kandidat n-*itemset* ini terdapat fungsi *joinItems*. Untuk menggabungkan item-item tersebut. Fungsi *joinItems* dapat dilihat pada *Sourcecode* 4.6

```
Private List<string> joinItems(List<string> C1,
     List<string> C2, int k)
{
   List<string> result = new List<string>();
   int nTemu = 0;
   for (int i = 0; i < C1.Count; i++)
   {
     if (C1[i] == C2[i])
     {  nTemu++; }
     if (!result.Contains(C1[i]))
         result.Add(C1[i]);
     if (!result.Contains(C2[i]))
         result.Add(C2[i]);
     if (result.Count == k + 1) break;
   }
   if (nTemu < k - 2) result = new List<string>();
       return result;
}
```

Sourcecode 4.6 Fungsi joinItems

4.2.2.5 Proses Pemangkasan n-*itemset*

Setelah didapatkan jumlah dari kandidat *n-itemset* dilakukan pemangkasan untuk jumlah yang kurang sama dari nilai minimum *support*. Proses ini dapat dilihat pada *Sourcecode* 4.7.

```
private void prosesApriori()
{
 if (tempiC.Count >= nItem)
 {
  if (isItemExits(tempiC, iC)) continue;
  int tempSupp = getSupport(tempiC);
  if (tempSupp >= supp) iC.Add(tempiC, tempSupp);
 }
}
```

Sourcecode 4.7 Pemangkasan n-*itemset*

4.2.2.6 Proses Pembentukan Aturan (*Rule*)

Pembentukan aturan dari item-item yang telah melalui proses pemangkasan berdasarkan nilai minimum *support* dan telah dihitung nilai minimum *confidence* nya. Proses pembentukan aturan dapat dilihat pada *Sourcecode* 4.8.

```
private void prosesApriori()
{
  for (int i = 0; i < iC.Count; i++)
 {
  List<string> tmp = new
      List<string>(iC.ElementAt(i).Key);
  Variations<string> variations = new
      Variations<string>(tmp, nItem - 1);

  foreach (IList<string> v in variations)
  { List<string> a = toList(ItoStr(v));
    if ((confidence(getSupportItemsApr(tmp),
     getSupportItemsApr(a))) >= j)

    tbApriori.AppendText(String.Format("Jika {0}
      \t maka   {1}  dengan nilai confidence {2:F}
      \n",    ItoStr(v), v[v.Count - 1],
      confidence(getSupportItemsApr(tmp),
      getSupportItemsApr(a))));
   }
}}
```

Sourcecode 4.8 Pembentukan Aturan

Pada proses pembentukan *rule* terdapat fungsi `getSupportItemsApr` dapat dilihat pada gambar *Sourcecode* 4.9 dan fungsi `confidence` yang dapat dilihat pada *Sourcecode* 4.10.

```
private int getSupportItemsApr(List<string> items)
{
  int result = 0;
  for (int i = 0; i < ItemDic.Count; i++)
  {
    int nTemu = 0;
    for (int j = 0; j < items.Count; j++)
    {
      if
(ItemDic.ElementAt(i).Value.Contains(items[j]))
      {  nTemu++;   }
    } if (nTemu >= items.Count) result++;
  }
  return result;
}
```

Sourcecode 4.9 Fungsi `getSupportItemsApr`

```
public double confidence(double a, double b)
        {
                return(a / b) * 100;
        }
```

Sourcecode 4.10 Fungsi `confidence`

4.2.3 Algoritma *Perfect Hashing and Pruning (PHP)*

Dalam algoritma *Perfect Hashing Pruning (PHP)* untuk implementasinya maka dilakukan proses pembacaan data, proses pembentukan pemetaan, proses *pruning* dan proses pembentukan aturan (*rule*).

4.2.3.1 Proses Pembacaan Data

Dataset yang disimpan pada dabatase dipanggil saat pertama kali proses dijalankan. Dataset akan ditampilkan pada *DataGridView* dangan cara pada *Sourcecode* 4.11.

```
private void Form1_Load(object sender, EventArgs e)
{
this.mushroomFullTableAdapter.Fill(this._MyDatabase
_2DataSet1.MushroomFull);
}
```

Sourcecode 4.11 Pembacaan Data

4.2.3.2 Proses Pemetaan

Pada proses pemetaan terdapat dua subproses. Suproses yang pertama yakni proses pembentukan *Hash Table Item* yaitu proses untuk menginisialisasi item menjadi sebuah indeks dan menghitung jumlah item tersebut. proses pembentukan *Hash Table Item* dapat dilihat pada *Sourcecode* 4.12 .

Subproses yang kedua yakni Proses Pembaruan Kamus Data yaitu mengubah kamus data yang sebelumnya berisi item-item diubah menjadi indeks. Proses Pembaruan Kamus Data dapat dilihat pada gambar *Sourcecode* 4.13.

```
private void pemetaan()
{
   foreach (KeyValuePair<int, string> tmpItem in
         dataItem)
{    string[] tempItem =
               tmpItem.Value.ToString().Split(',');
   for (int j = 0; j < tempItem.Length; j++)
   {
     if (!hashItem.ContainsKey(tempItem[j]))
     {
       string  indx = String.Format("H{0}",
                      hashItem.Count);
       hashItem.Add(tempItem[j], indx) ;
       hashSupport.Add( indx ,  1);
     }
     else
     { hashSupport[hashItem[tempItem[j]]]++; }
   }
}}
```

Sourcecode 4.12 Pembentukan *Hash Table Item*

```
Dictionary<int,string> tempDic = new
          Dictionary<int,string>();
List<string> pattern = new List<string

foreach (KeyValuePair<string,string> tempItem in
        hashItem)
{  pattern.Add(tempItem.Key.ToString());}
foreach (KeyValuePair<int, string> tmpItem in
        dataItem)
{
   List <string> temp =
                toList(tmpItem.Value.ToString());
   List <string> result = new List<string>();
   var hasil = pattern.Intersect(temp);
   foreach (var enms in hasil){
   result.Add(hashItem[enms.ToString()].ToString());
   }
      tempDic.Add(tmpItem.Key, toStr(result));
}
dataItem =  new Dictionary<int,string>(tempDic);
```

Sourcecode 4.13 Pembaruan Kamus Data

4.2.3.3 Proses *Pruning*

Setelah *hash table Item* terbentuk, dapat dilakukan proses *pruning* sesuai nilai minimum *support*. Proses *pruning* dapat dilihat pada *Sourcecode* 4.14. Setelah proses *pruning* dilakukan *hash table item* dan kamus data diperbarui lagi dari item-item yang tidak memenuhi nilai minimum *support*. Untuk proses pembaruan tabel item dapat dilihat pada gambar *Sourcecode* 4.15, sedangkan untuk proses pembaruan kamus data dapat dilihat pada *Sourcecode* 4.16.

```
private void prosesPHP()
{
  Dictionary<string, string> tempHashItem = new
          Dictionary<string, string>(hashItem);
  foreach (KeyValuePair<string, string> tmpHash in
        tempHashItem)
  {
     if (hashSupport[tmpHash.Value] <= supp)
    {
      hashSupport.Remove(tmpHash.Value);
      hashItem.Remove(tmpHash.Key);
    }
  }
}
```

Sourcecode 4.14 Proses *Pruning*

```
private void updateHashData(List<string> pattern)
{
   Dictionary<int ,string> tempDic = new
       Dictionary<int, string>();
   foreach (KeyValuePair<int, string> tmpItem in
          dataItem)
   {
     List<string> temp =
                 toList(tmpItem.Value.ToString());
     List<string> result = new List<string>();
     var hasil = pattern.Intersect(temp);
     tempDic.Add(tmpItem.Key, enumToString(hasil));
   }
   dataItem = new Dictionary<int, string>(tempDic);
}
```

Sourcecode 4.15 Fungsi *updateHashData*

```
private void updateHashItem(List<string> pattern)
{
  List<string> tempListItem = new List<string>();
  Dictionary<string, string> tempHashItem = new
      Dictionary<string, string>(hashItem);
  foreach (KeyValuePair<string, int> de in hC)
  {
    List<string> tmpC = toList(de.Key.ToString());
    for (int i = 0; i < tmpC.Count; i++)
    {
      if (!tempListItem.Contains(tmpC[i]))
    {
      tempListItem.Add(tmpC[i]);
    }
  }
}
```

Sourcecode 4.16 Fungsi *updateHashItem*

4.2.3.4 Proses Pembentukan Aturan (*Rule*)

Pembentukan aturan dari item-item yang telah melalui proses pemangkasan berdasarkan nilai minimum *support* dan telah dihitung nilai minimum *confidence* nya. Proses pembentukan aturan dapat dilihat pada *Sourcecode* 4.17.

```
private void prosesPHP()
{
 foreach (KeyValuePair<string, int> dEnt in hC)
 {  List<string> tmp = new
List<string>(toList(dEnt.Key));
 variations <string> var = new variations
           <string>(tmp, nItem - 1);
 foreach (IList<string> v in var)
 { List<string> a = List(ItoStrHash(v));
   if ((confidence(getHashSupport(tmp),
           getHashSupport(a))) >= j)
    { tbPHP.AppendText(String.Format(" - Jika {0}
      maka   {1}   dengan nilai confidence {2}\n",
      ItoStrHash(v), getItemName(v[v.Count - 1]),
      confidence(getHashSupport(tmp),
      getHashSupport(a))));}
  }
 }}
```

Sourcecode 4.17 Pembentukan Aturan

Pada Proses pembentukan aturan terdapat fungsi `getHashSupport` dapat dilihat pada *Sourcecode* 4.18.

```
private int getHashSupport(List<string> iH)
{
  int result = 0;
  int idi = 0;
  foreach (KeyValuePair<int,string> tmpItem in
dataItem)
  {
    List<string> temp =
           toList(tmpItem.Value.ToString());
    idi++;
    int nTemu = 0;
    for (int j = 0; j < iH.Count; j++)
    {
     if (temp.Contains(iH[j] ))
     {
        nTemu++;
     }
    }
    if (nTemu >= iH.Count) result++;
  }
 return result;
}
```

Sourcecode 4.18 Fungsi `getHashSupport`

4.3 Pengujian Sistem

Data yang digunakan dalam sistem yaitu data *mushroom*. Pada data ini atribut yang digunakan antara lain yaitu *cap-shape, cap-surface, cap-color, bruises, odor, gill-attachment, gill-spacing, gill-size, gill-color, stalk-shape, stalk-root, stalk-surface-above-ring, stalk-surface-below-ring, stalk-color-above-ring, stalk-color-below-ring, veil-color, ring-number, ring-type, spore-print-color, population,habitat.* Dimana atribut-atribut tersebut digolongkan menjadi 2 kelas yaitu *edible* dan *poisonous*. Jumlah data yang digunakan ada 8416 data.

4.3.1 Pengujian Jumlah *Rule*

Pengujian yang dilakukan adalah pengukuran jumlah *rule* yang terbentuk. Pengukuran ini dilakukan untuk memperoleh banyaknya

jumlah aturan (*rule*) yang terbentuk dengan menggunakan algoritma *Apriori* dan Algoritma *Perfect Hashing and Pruning(PHP)*.

Pengukuran jumlah *rule* dilakukan menggunakan data uji dengan parameter nilai minimum *support* dan minimum *Confidence* yang ditentukan oleh *user*. Hasil dari pengukuran jumlah *rule* dapat dilihat pada tabel 4.1.

Tabel 4.1 Hasil Pengujian *Rule* yang terbentuk

Minimum Support	Minimum Confidence	Jumlah *Rule*	
		Apriori	*Perfect Hashing and Pruning (PHP)*
50%	80%	96	38
	90%	79	31
	100%	28	9
60%	80%	33	26
	90%	26	22
	100%	5	5
70%	80%	24	18
	90%	12	17
	100%	2	2

Berdasarkan dari tabel untuk ukuran jumlah *rule* dengan algoritma *Apriori* dan algoritma *Perfect Hashing and Prunin(PHP)*, maka didapatkan bahwa jumlah *rule* ynag dihasilkan algoritma *PHP* lebih sedikit dibandingkan dengan algoritma *Apriori*.

4.3.2 Pengujian *Lift Ratio Rule*

Pengujian ini dilakukan untuk melihat kekuatan *rule* yang dihasilkan oleh sistem dengan menggunakan algoritma *Aporiori* atau algoritma *Perfect Hashing and Pruning (PHP)*. Pengujian dilakukan terhadap *rule* yang dihasilkan dengan parameter minimum *support* 50% dan minimum *confidence* 80%. Untuk algoritma *Apriori* menghasilkan 96 aturan, sedangkan untuk algoritma *Perfect Hashing and Pruning (PHP)* menghasilkan 38 aturan.

Hasil pengujian *lift ratio rule* yang terbentuk dengan menggunakan algoritma *Apriori* dapat dilihat pada Tabel 4.2 dan Lampiran. Sedangkan *lift ratio rule* yang terbentuk menggunakan

algoritma *Perfect Hashing and Pruning (PHP)* dapat dilihat pada Tabel 4.3 dan Lampiran.

Tabel 4.2. Hasil Pengujian *Lift Ratio* yang Terbentuk Menggunakan Algoritma *Apriori*

Rule	*Confidence*	*Benchmark Confidence*	*Lift Ratio*
edible → free	0.96	0.97	0.98
edible → white_v	0.96	0.98	0.98
tapering → free	1.00	0.97	1.03
smooth_ar → free	0.96	0.97	0.99
smooth_br → free	0.96	0.97	0.99
white_ar → free	1.00	0.97	1.03
white_br → free	1.00	0.97	1.03
edible, free → white_v	1.00	0.98	1.02
edible, white_v → free	1.00	0.97	1.03
free, tapering → white_v	1.00	0.98	1.02
tapering, white_v → free	1.00	0.97	1.03
free, tapering → one	1.00	0.92	1.08
tapering, one → free	1.00	0.97	1.03

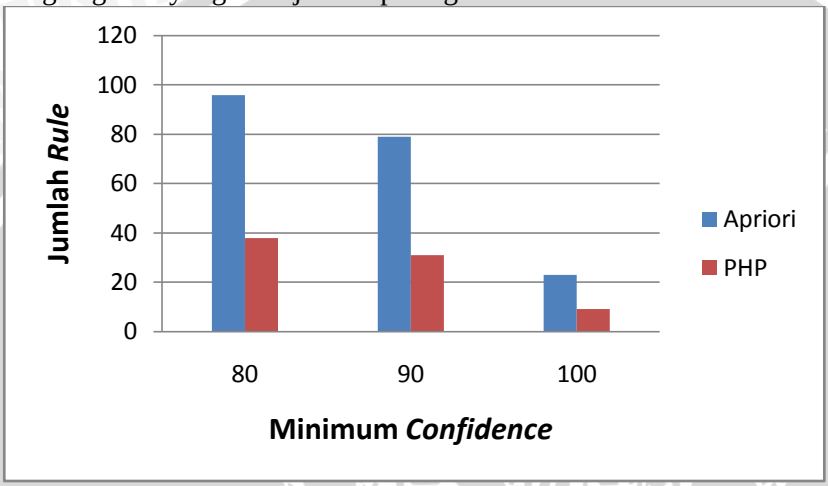
Tabel 4.3. Hasil Pengujian *Lift Ratio* yang Terbentuk Menggunakan Algoritma *Perfect Hashing and Pruning (PHP)*

Rule	*Confidence*	*Benchmark Confidence*	*Lift Ratio*
edible → free	0.96	0.97	0.98
edible → white_v	0.96	0.98	0.98
tapering → free	1.00	0.97	1.03
smooth_ar → free	0.96	0.97	0.99
smooth_br → free	0.96	0.97	0.99
white_ar → free	1.00	0.97	1.03
white_br → free	1.00	0.97	1.03
edible, free → white_v	1.00	0.98	1.02
edible, white_v → free	1.00	0.97	1.03

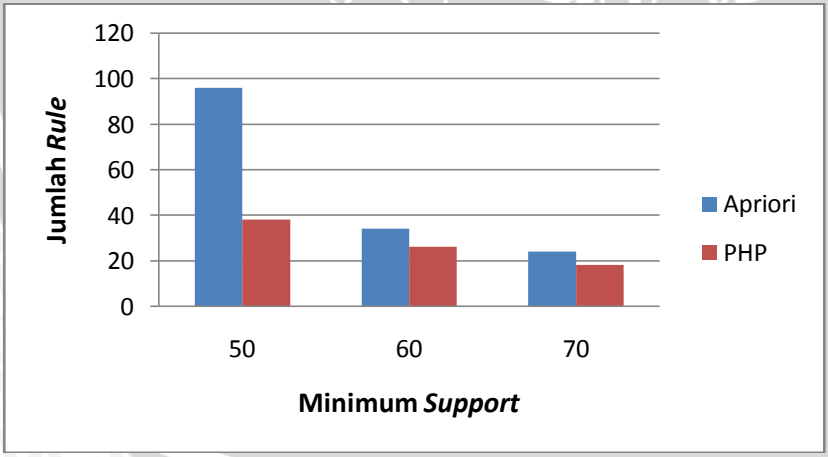
4.4 Analisa Hasil

4.4.1 Analisa Pengujian Jumlah *Rule*

Nilai pengukuran jumlah *rule* yang dihasilkan oleh sistem dengan menggunakan algoritma *Apriori* dan algoritma *Perfect Hashing and Pruning* yang telah ditunjukkan pada subbab 4.3 dapat dinyatakan dengan grafik yang ditunjukkan pada gambar 4.3 dan 4.4.



Gambar 4.3 Grafik Jumlah *Rule* yang Terbentuk Berdasarkan Nilai Minimum *Confidence*



Gambar 4.3 Grafik Jumlah *Rule* yang Terbentuk Berdasarkan Nilai Minimum *Support*

Gambar 4.3 menunjukkan *rule* yang terbentuk menggunakan algoritma *Apriori* dan Algoritma *Perfect Hashing and Pruning* dengan nilai minimum *support* yang sama yakni 50% dan nilai minimum *confidence* yang berbeda yaitu 80%, 90% dan 100%. Jumlah *rule* yang terbentuk untuk nilai minimum *confidence* 80% adalah *Apriori* 96 *rule*, *PHP* 38 *rule*. Untuk nilai minimum *confidence* 90% adalah *Apriori* 79 *rule*, *PHP* 31 *rule*. Dan untuk nilai minimum *confidence* 100% adalah *Apriori* 23 *rule*, *PHP* 9 *rule*.

Gambar 4.4 menunjukkan *rule* yang terbentuk menggunakan algoritma *Apriori* dan Algoritma *Perfect Hashing and Pruning* dengan nilai minimum *support* yang berbeda yaiu 50%, 60% dan 70% sedangkan nilai minimum *confidence*-nya sama yaitu 80%. Jumlah *rule* yang terbentuk untuk nilai minimum *support* 50% adalah *Apriori* 96 *rule*, *PHP* 38 *rule*. Untuk nilai minimum *confidence* 60% adalah *Apriori* 34 *rule*, *PHP* 26 *rule*. Dan untuk nilai minimum *confidence* 70% adalah *Apriori* 24 *rule*, *PHP* 18 *rule*.

Dari gambar 4.3 dan 4.4 dapat dilihat jumlah *rule* yang terbentuk menggunakan algoritma Apriori dan algoritma *Perfect Hashing and Pruning* dipengaruhi oleh nilai minimum *support* dan nilai minimum *confidence* Semakin naik nilai minimum *support* dan nilai minimum *confidence* maka *rule* yang dihasilkan semakin sedikit. Hal ini dikarenakan kandidat-kandidat n-itemset yang terbentuk dipangkas karena tidak memenuhi nilai minimum *support* dan *rule* yang telah memenuhi nilai minimum *support* dipangkas lagi jika nilai minimum *confidence*-nya tidak terpenuhi.

Dari gambar 4.3 dan 4.4. juga dapat dilihat bahwa jumlah *rule* yang terbentuk dengan menggunakan algoritma *Apriori* lebih banyak dibandingkan dengan menggunakan algoritma *Perfect Hashing and Pruning*. Hal ini dikarenakan pada algoritma *Apriori* untuk menemukan kandidat *n-itemset* dilakukan pen-*scan*-an pada database. Sedangkan pada algoritma *Perfect Hashing and Pruning* mengambil kombinasi yang terdapat pada kamus data. Kamus data ini di-*update* dengan cara memangkas item yang tidak memenuhi nilai minimum *support.* Sehingga item yang tidak memenuhi nilai minimum *support* tidak digunakan lagi untuk proses penemuan *n-itemset* yang berikutnya. Dan hasil *rule* yang terbentuk menjadi lebih sedikit.

4.4.2 Analisa *Lift Ratio Rule*

Dari hasil pengujian *lift ratio associaton rule* pada subbab 4.3.2 dapat dilihat bahwa *association rule* yang bermanfaat atau nilai *lift ratio* besar dari 1 dengan menggunakan algoritma *Apriori* sebanyak 55 dari 96 *rule* (57.29%). Sedangkan yang menggunakan algoritma *Perfect Hashing and Pruning (PHP)* sebanyak 19 dari 38 *rule* (50%). Berarti *rule* yang dihasilkan dengan menggunakan algoritma *Apriori* memiliki kekuatan yang lebih dibandingkan dengan *rule* yang dihasilkan menggunakan algoritma *Perfect Hashing and Pruning (PHP)*.

Dari pengujian *lift ratio rule* juga dapat dilihat bahwa *rule* yang memiliki nilai *confidence* yang tinggi tidak selalu diikuti oleh nilai *lift ratio* yang tinggi pula. Artinya tidak semua *rule* yang memiliki tingkat kebenaran (*reability*) yang tinggi juga memiliki kekuatan/manfaat yang besar pula. Atau dengan kata lain, bila *association rule* memiliki nilai yang tinggi untuk *consequent* (item setelah maka) tidak selalu item-item dalam *antecedent* (item setelah jika) dan *consequent* saling independent.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang diperoleh dari skripsi ini adalah :

1. Berdasarkan hasil uji coba jumlah *rule* yang bertujuan mengukur *rule* yang yang terbentuk oleh sistem dalam menemukan *association rule* dengan menggunakan dataset *mushroom* sebagai data tes dengan parameter nilai minimum *support* dan nilai minimum *confidence*, bahwa nilai minimum *support* nilai minimum *confidence* berbanding terbalik terhadap jumlah *rule* yang dihasilkan. Artinya semakin besar nilai minimum *support* dan nilai minimum *confidence* maka semakin sedikit jumlah *rule* yang ditemukan.
2. Berdasarkan uji coba yang telah dilakukan bahwa jumlah *rule* yang ditemukan dengan menggunakan algoritma *Perfect Hashing and Pruning* lebih sedikit dibandingkan dengan menggunakan algoritma *Apriori.*
3. Berdasarkan hasil uji coba *lift ratio rule* untuk mengukur kekuatan *association rule* yang terbentuk, bahwa dengan menggunakan algoritma *Apriori* didapatkan *rule* yang memiliki manfaat sebanyak 55 dari 96 *rule* (57.29%) sedangkan dengan menggunakan algoritma *Perfect Hashing and Pruning (PHP)* sebanyak 19 dari 38 *rule* (50%). Artinya dengan menggunakan algoritma *Apriori* menghasilkan *rule* yang memiliki kekuatan/ manfaat lebih dibandingkan dengan menggunakan algoritma *Perfect Hashing and Pruning(PHP).*
4. Berdasarkan hasil uji coba *lift ratio* bahwa *rule* yang memiliki nilai *confidence* tinggi tidak selalu diikuti dengan nilai *lift ratio* yang tinggi pula.

5.2 Saran

Saran yang mungkin menjadi pertimbangan untuk penelitian selanjutnya adalah :

1. Perlu menghitung jumlah kebenaran dari *rule* yang dihasilkan dengan menggunakan data *testing* (latih).

DAFTAR PUSTAKA

Berson, Alex. Stephen Smith and Kurt Thearling. 1999. *Building Data Mining Applications for CRM*. United State of America: Mc Graw-Hill.

Guidici, P. 2003. *Applied Data Mining Statistical Methods for Business and Industry*. John Wiley & Sons. Chicester.

Gunawan. 2007. *Algoritma Apriori.* Dalam http://www.hansmichael.com/download/dmkdd09-6up-bw.pdf. Diakses tanggal 29 Mei 2010.

Iti. 2008. *Perfect Hashing and Pruning Algorithm*. Dalam http://associationrule.blogspot.com/2008/09/perfect-*hashing*-and-pruning-algorithm.html. Diakses tanggal 10 Mei 2010.

J. D. Holt, dan S. M. Chung, 1999. *Efficient Mining of Association Rules in Text Databases*. CIKM'99, Kansas City, USA.

Kusrini dan Ehma Taufiq Luthfi. 2009. *Algoritma Data Mining*. ANDI Offset: Yogyakarta.

Pramudiono, Iko. 2007. *Algoritma Apriori*. Dalam http://datamining. japati.net/cgi-bin/indodm.cgi? bacaarsiplalu&1172210143&arsiptutorial&1172210174. Diakses tanggal 24 Mei 2010.

Prasetyo, Philips Kokoh. 2006. *Apriori*. Dalam http://philips.wordpress.com. Diakses tanggal 29 Mei 2010.

R. Agrawal, dan R. Srikant. 1994. *Fast Algorithms for Mining Association Rules*. Proc. of the 20th Int'l Conference on Very Large Databases, Santiago, Chile.

S.Ayse Ozel dan H. Altay Güvenir. 2001. *An Algorithm for Mining Association Rules Using Perfect Hashing and Database Pruning*. Bilkent University, Department of Computer Engineering, Ankara, Turkey.

Santosa, Budi. 2007. *Data Mining Teknik Pemanfaatan Data untuk Keperluan Bisnis.* Graha Ilmu . Yogyakarta.

Savasere, E. Omiecienski, and S.Navathe. 1995. *An Efficient Algorithm for Mining Association Rules in Large Databases.* Intenational Conference Very Large Data Bases (VLDB).

Seidman, Claude. 2000. *Data Mining With SQL Server 2000.* Microsoft Press. Washington.

Susilowarno, Gunawan.,dkk. 2007. *Biologi untuk SMA/MA kelas X*. Grasindo. Jakarta.

Suwarningsih, Wiwin. 2008. *Penerapan Association Rule Mining untuk Perancangan Data Mining BDP (Barang Dalam Proses) Obat.* Pusat Penelitian Informatika LIPI. Bandung.

Thearling, Kurt. 1997. *Understanding Data Mining: It's All in The Interaction.* Dalam http://www.tgc.com/dsstar.

LAMPIRAN

Lampiran 1. *Rule* yang Terbentuk Menggunakan Algoritma *Apriori* dengan Minimum *Support* 50% dan Minimum *Confidence* 80%

Rule	*Confidence*
edible → free	95.72%
edible → white_v	95.72%
tapering → free	100.00%
smooth_AR → free	96.39%
smooth_BR → free	96.22%
white_AR → free	100.00%
white_BR → free	100.00%
free → white_v	99.90%
white_v → free	99.71%
free → one	92.39%
one → free	97.53%
free → close	80.59%
close → free	96.83%
broad → free	96.33%
tapering → white_v	100.00%
tapering → one	100.00%
smooth_br → smooth_ar	83.25%
smooth_ar → white_v	96.39%
smooth_ar → one	91.72%
smooth_br → white_v	96.22%
smoot_br → one	91.33%
white_ar → white_v	100.00%
white_ar → one	89.71%
white_br → white_v	92.11%
whie_br → one	97.43%
one → white_v	83.22%
smooth_ar → close	97.19%
close → white_v	80.72%

white_v → close	94.72%
close → one	83.21%
one → close	96.73%
broad → white_v	88.98%
broad → one	96.73%
no → free	95.71%
no → white_v	96.03%
no → one	93.02%
edible, free → white_v	100.00%
edible, white_v → free	100.00%
free, tapering → white_v	100.00%
tapering, white_v → free	100.00%
free, tapering → one	100.00%
tapering, one → free	100.00%
free, smooth_ar → white_v	100.00%
smooth_ar, white_v → free	100.00%
free, smooth_ar → one	91.41%
smooth_ar, one → free	96.06%
free, smooth_ar → close	82.59%
smooth_ar, close → free	95.66%
free, smooth_br → white_v	100.00%
smooth_br, white_v → free	100.00%
free, smooth_br → one	90.99%
smooth_br, one → free	95.86%
free, white_ar → white_v	100.00%
white_ar, white_v → free	100.00%
free, white_ar → one	89.71%
white_ar, one → free	100.00%
free, white_br → white_v	100.00%
white_br, white_v → free	100.00%
free, white_v → one	92.38%
free, one → white_v	99.89%

white_v, one → free	100.00%
free, white_v → close	80.66%
free, close → white_v	100.00%
white_v, close → free	99.64%
free, broad → white_v	100.00%
broad, white_v → free	99.58%
free, one → close	82.79%
free, close → one	94.92%
one, close → free	97.03%
free, broad → one	88.98%
one, broad → free	96.33%
close,broad → free	95.27%
tapering, white_v → one	100.00%
tapering, one → white_v	100.00%
smooth_ar, white_v → one	91.41%
smooth_ar, one → white_v	96.06%
smooth_br, white_v → one	90.99%
smooth_br, one → white_v	95.86%
white_ar, white_v → one	89.71%
white_v, one → white_ar	100.00%
close, smooth_ar → white_v	95.66%
smooth_ar, white_v → close	82.59%
close, white_v →one	94.57%
close, one → white_v	97.03%
white_v, one → close	82.88%
close, broad → white_v	95.80%
close, broad → one	92.12%
one, broad → close	80.43%
broad, white_v → one	88.61%
broad, one → white_v	99.63%
no, free → white_v	99.83%
no, white_v → free	99.50%

no, free → one	93.20%
no, one, → free	95.90%
no, white_v → one	92.73%
no, one → white_v	95.73%

Lampiran 2. *Rule* yang Terbentuk Menggunakan Algoritma *Apriori* dengan Minimum *Support* 50% dan Minimum *Confidence* 90%

Rule	*Confidence*
edible → free	95.72%
edible → white_v	95.72%
tapering → free	100.00%
smooth_ar → free	96.39%
smooth_br → free	96.22%
white_ar → free	100.00%
white_br → free	100.00%
free → white_v	99.90%
white_v → free	99.71%
free → one	92.39%
one → free	97.53%
close → free	96.83%
broad → free	96.33%
tapering → white_v	100.00%
tapering → one	100.00%
smooth_ar → white_v	96.39%
smooth_ar → one	91.72%
smooth_br → white_v	96.22%
smoot_br → one	91.33%
white_ar → white_v	100.00%
white_br → white_v	100.00%
whie_br → one	92.11%
one → white_v	97.43%
close → white_v	97.19%
close → one	94.72%
broad → white_v	96.73%
no → free	95.71%
no → white_v	96.03%
no → one	93.02%
edible, free → white_v	100.00%

edible, white_v → free	100.00%
free, tapering → white_v	100.00%
tapering, white_v → free	100.00%
free, tapering → one	100.00%
tapering, one → free	100.00%
free, smooth_ar → white_v	100.00%
smooth_ar, white_v → free	100.00%
free, smooth_ar → one	91.41%
smooth_ar, one → free	96.06%
smooth_ar, close → free	95.66%
free, smooth_br → white_v	100.00%
smooth_br, white_v → free	100.00%
free, smooth_br → one	90.99%
smooth_br, one → free	95.86%
free, white_ar → white_v	100.00%
white_ar, white_v → free	100.00%
white_ar, one → free	100.00%
free, white_br → white_v	100.00%
white_br, white_v → free	100.00%
free, white_v → one	92.38%
free, one → white_v	99.89%
white_v, one → free	100.00%
free, close → white_v	100.00%
white_v, close → free	99.64%
free, broad → white_v	100.00%
broad, white_v → free	99.58%
free, close → one	94.92%
one, close → free	97.03%
one, broad → free	96.33%
close,broad → free	95.27%
tapering, white_v → one	100.00%
tapering, one → white_v	100.00%

smooth_ar, white_v → one	91.41%
smooth_ar, one → white_v	96.06%
smooth_br, white_v → one	90.99%
smooth_br, one → white_v	95.86%
white_v, one → white_ar	100.00%
close, smooth_ar → white_v	95.66%
close, white_v →one	94.57%
close, one → white_v	97.03%
close, broad → white_v	95.80%
close, broad → one	92.12%
broad, one → white_v	99.63%
no, free → white_v	99.83%
no, white_v → free	99.50%
no, free → one	93.20%
no, one, → free	95.90%
no, white_v → one	92.73%
no, one → white_v	95.73%

Lampiran 3. *Rule* yang Terbentuk Menggunakan Algoritma *Apriori* dengan Minimum *Support* 50% dan Minimum *Confidence* 100%

Rule	*Confidence*
tapering → free	100.00%
white_ar → free	100.00%
white_br → free	100.00%
tapering → white_v	100.00%
tapering → one	100.00%
white_ar → white_v	100.00%
white_br → white_v	100.00%
edible, free → white_v	100.00%
edible, white_v → free	100.00%
free, tapering → white_v	100.00%
tapering, white_v → free	100.00%
free, tapering → one	100.00%
tapering, one → free	100.00%
free, smooth_ar → white_v	100.00%
smooth_ar, white_v → free	100.00%
free, smooth_br → white_v	100.00%
smooth_br, white_v → free	100.00%
free, white_ar → white_v	100.00%
white_ar, white_v → free	100.00%
white_ar, one → free	100.00%
free, white_br → white_v	100.00%
white_br, white_v → free	100.00%
white_v, one → free	100.00%
free, close → white_v	100.00%
free, broad → white_v	100.00%
tapering, white_v → one	100.00%
tapering, one → white_v	100.00%
white_v, one → white_ar	100.00%

Lampiran 4. *Rule* yang Terbentuk Menggunakan Algoritma *Apriori* dengan Minimum *Support* 60% dan Minimum *Confidence* 80%

Rule	*Confidence*
smooth_ar → free	96.39%
free → white_v	99.90%
white_v → free	99.71%
free → one	92.39%
one → free	97.53%
free → close	80.59%
close → free	96.83%
broad → free	96.33%
smooth_ar → white_v	96.39%
whie_v → one	92.11%
one → white_v	97.43%
close → white_v	97.19%
white_v → close	80.72%
close → one	94.72%
one → close	83.21%
broad → white_v	96.73%
broad → one	88.98%
free, smooth_ar → white_v	100.00%
smooth_ar, white_v → free	100.00%
free, white_v → one	92.38%
free, one → white_v	99.89%
white_v, one → free	100.00%
free, white_v → close	80.66%
free, close → white_v	100.00%
white_v, close → free	99.64%
free, broad → white_v	100.00%
broad, white_v → free	99.58%
free, one → close	82.79%
free, close → one	94.92%
one, close → free	97.03%

close, white_v →one	94.57%
close, one → white_v	97.03%
white_v, one → close	82.88%

Lampiran 5. *Rule* yang Terbentuk Menggunakan Algoritma *Apriori* dengan Minimum *Support* 60% dan Minimum *Confidence* 90%

Rule	*Confidence*
smooth_ar → free	96.39%
free → white_v	99.90%
white_v → free	99.71%
free → one	92.39%
one → free	97.53%
close → free	96.83%
broad → free	96.33%
smooth_ar → white_v	96.39%
white_v → one	92.11%
one → white_v	97.43%
close → white_v	97.19%
close → one	94.72%
broad → white_v	96.73%
free, smooth_ar → white_v	100.00%
smooth_ar, white_v → free	100.00%
free, white_v → one	92.38%
free, one → white_v	99.89%
white_v, one → free	100.00%
free, close → white_v	100.00%
white_v, close → free	99.64%
free, broad → white_v	100.00%
broad, white_v → free	99.58%
free, close → one	94.92%
one, close → free	97.03%
close, white_v →one	94.57%
close, one → white_v	97.03%

Lampiran 6. *Rule* yang Terbentuk Menggunakan Algoritma *Apriori* dengan Minimum *Support* 60% dan Minimum *Confidence* 100%

Rule	*Confidence*
free, smooth_ar → white_v	100.00%
smooth_ar, white_v → free	100.00%
white_v, one → free	100.00%
free, close → white_v	100.00%
free, broad → white_v	100.00%

Lampiran 7. *Rule* yang Terbentuk Menggunakan Algoritma *Apriori* dengan Minimum *Support* 70% dan Minimum *Confidence* 80%

Rule	*Confidence*
free → white_v	99.90%
white_v → free	99.71%
free → one	92.39%
one → free	97.53%
free → close	80.59%
close → free	96.83%
whie_v → one	92.11%
one → white_v	97.43%
close → white_v	97.19%
white_v → close	80.72%
close → one	94.72%
one → close	83.21%
free, white_v → one	92.38%
free, one → white_v	99.89%
white_v, one → free	100.00%
free, white_v → close	80.66%
free, close → white_v	100.00%
white_v, close → free	99.64%
free, one → close	82.79%
free, close → one	94.92%
one, close → free	97.03%
close, white_v →one	94.57%
close, one → white_v	97.03%
white_v, one → close	82.88%

Lampiran 8. *Rule* yang Terbentuk Menggunakan Algoritma *Apriori* dengan Minimum *Support* 70% dan Minimum *Confidence* 90%

Rule	*Confidence*
free → white_v	99.90%
white_v → free	99.71%
free → one	92.39%
one → free	97.53%
close → free	96.83%
whie_v → one	92.11%
one → white_v	97.43%
close → white_v	97.19%
close → one	94.72%
free, white_v → one	92.38%
free, one → white_v	99.89%
white_v, one → free	100.00%
free, close → white_v	100.00%
white_v, close → free	99.64%
free, close → one	94.92%
one, close → free	97.03%
close, white_v →one	94.57%
close, one → white_v	97.03%

Lampiran 9. *Rule* yang Terbentuk Menggunakan Algoritma *Apriori* dengan Minimum *Support* 70% dan Minimum *Confidence* 100%

Rule	*Confidence*
white_v, one → free	100.00%
free, close → white_v	100.00%

Lampiran 10. *Rule* yang Terbentuk Menggunakan Algoritma *PHP* dengan Minimum *Support* 50% dan Minimum *Confidence* 80%

Rule	*Confidence*
edible → free	95.72%
edible → white_v	95.72%
tapering → free	100.00%
smooth_ar → free	96.39%
smooth_br → free	96.22%
white_ar → free	100.00%
white_br → free	100.00%
free → white_v	99.90%
white_v → free	99.71%
free → one	92.39%
one → free	97.53%
free → close	80.59%
close → free	96.83%
broad → free	96.33%
tapering → white_v	100.00%
tapering → one	100.00%
smooth_br → smooth_ar	83.25%
smooth_ar → white_v	96.39%
smooth_ar → one	91.72%
smooth_br → white_v	96.22%
smoot_br → one	91.33%
white_ar → white_v	100.00%
white_ar → one	89.71%
white_br → white_v	92.11%
whie_br → one	97.43%
one → white_v	83.22%
smooth_ar → close	97.19%
close → white_v	80.72%
white_v → close	94.72%
close → one	83.21%

one → close	96.73%
broad → white_v	88.98%
broad → one	96.73%
no → free	95.71%
no → white_v	96.03%
no → one	93.02%
edible, free → white_v	100.00%
edible, white_v → free	100.00%

Lampiran 11. *Rule* yang Terbentuk Menggunakan Algoritma *PHP* dengan Minimum *Support* 50% dan Minimum *Confidence* 90%

Rule	*Confidence*
edible → free	95.72%
edible → white_v	95.72%
tapering → free	100.00%
smooth_ar → free	96.39%
smooth_br → free	96.22%
white_ar → free	100.00%
white_br → free	100.00%
free → white_v	99.90%
white_v → free	99.71%
free → one	92.39%
one → free	97.53%
close → free	96.83%
broad → free	96.33%
tapering → white_v	100.00%
tapering → one	100.00%
smooth_ar → white_v	96.39%
smooth_ar → one	91.72%
smooth_br → white_v	96.22%
smoot_br → one	91.33%
white_ar → white_v	100.00%
white_br → white_v	100.00%
whie_br → one	92.11%
one → white_v	97.43%
close → white_v	97.19%
close → one	94.72%
broad → white_v	96.73%
no → free	95.71%
no → white_v	96.03%
no → one	93.02%
edible, free → white_v	100.00%

edible, white_v → free	100.00%

Lampiran 12. *Rule* yang Terbentuk Menggunakan Algoritma *PHP* dengan Minimum *Support* 50% dan Minimum *Confidence* 100%

Rule	*Confidence*
tapering → free	100.00%
white_AR → free	100.00%
white_BR → free	100.00%
tapering → white_v	100.00%
tapering → one	100.00%
white_ar → white_v	100.00%
white_br → white_v	100.00%
edible, free → white_v	100.00%
edible, white_v → free	100.00%

Lampiran 13. *Rule* yang Terbentuk Menggunakan Algoritma *PHP* dengan Minimum *Support* 60% dan Minimum *Confidence* 80%

Rule	*Confidence*
smooth_ar → free	96.39%
free → white_v	99.90%
white_v → free	99.71%
free → one	92.39%
one → free	97.53%
free → close	80.59%
close → free	96.83%
broad → free	96.33%
smooth_ar → white_v	96.39%
whie_v → one	92.11%
one → white_v	97.43%
close → white_v	97.19%
white_v → close	80.72%
close → one	94.72%
one → close	83.21%
broad → white_v	96.73%
broad → one	88.98%
free, smooth_ar → white_v	100.00%
smooth_ar, white_v → free	100.00%
free, one → white_v	99.89%
white_v, one → free	100.00%
free, close → white_v	100.00%
white_v, close → free	99.64%
free, broad → white_v	100.00%
broad, white_v → free	99.58%
one, close → free	97.03%

Lampiran 14. *Rule* yang Terbentuk Menggunakan Algoritma *PHP* dengan Minimum *Support* 60% dan Minimum *Confidence* 90%

Rule	*Confidence*
smooth_ar → free	96.39%
free → white_v	99.90%
white_v → free	99.71%
free → one	92.39%
one → free	97.53%
close → free	96.83%
broad → free	96.33%
smooth_ar → white_v	96.39%
white_v → one	92.11%
one → white_v	97.43%
close → white_v	97.19%
close → one	94.72%
broad → white_v	96.73%
free, smooth_ar → white_v	100.00%
smooth_ar, white_v → free	100.00%
free, one → white_v	99.89%
white_v, one → free	100.00%
free, close → white_v	100.00%
white_v, close → free	99.64%
free, broad → white_v	100.00%
broad, white_v → free	99.58%
one, close → free	97.03%

Lampiran 15. *Rule* yang Terbentuk Menggunakan Algoritma *PHP* dengan Minimum *Support* 60% dan Minimum *Confidence* 100%

Rule	*Confidence*
free, smooth_ar → white_v	100.00%
smooth_ar, white_v → free	100.00%
white_v, one → free	100.00%
free, close → white_v	100.00%
free, broad → white_v	100.00%

Lampiran 16. *Rule* yang Terbentuk Menggunakan Algoritma *PHP* dengan Minimum *Support* 70% dan Minimum *Confidence* 80%

Rule	*Confidence*
free → white_v	99.90%
white_v → free	99.71%
free → one	92.39%
one → free	97.53%
free → close	80.59%
close → free	96.83%
whie_v → one	92.11%
one → white_v	97.43%
close → white_v	97.19%
white_v → close	80.72%
close → one	94.72%
one → close	83.21%
free, white_v → one	92.38%
free, one → white_v	99.89%
white_v, one → free	100.00%
free, close → white_v	100.00%
white_v, close → free	99.64%
one, close → free	97.03%

Lampiran 17. *Rule* yang Terbentuk Menggunakan Algoritma *PHP* dengan Minimum *Support* 70% dan Minimum *Confidence* 90%

Rule	*Confidence*
free → white_v	99.90%
white_v → free	99.71%
free → one	92.39%
one → free	97.53%
close → free	96.83%
whie_v → one	92.11%
one → white_v	97.43%
close → white_v	97.19%
close → one	94.72%
free, white_v → one	92.38%
free, one → white_v	99.89%
white_v, one → free	100.00%
free, close → white_v	100.00%
white_v, close → free	99.64%
one, close → free	97.03%

Lampiran 18. *Rule* yang Terbentuk Menggunakan Algoritma *PHP* dengan Minimum *Support* 70% dan Minimum *Confidence* 100%

Rule	*Confidence*
white_v, one → free	100.00%
free, close → white_v	100.00%

Lampiran 19. Hasil Pengujian *Lift Ratio Rule* yang Terbentuk Menggunakan Algoritma *Apriori*

Rule	*Confidence*	*Benchmark Confidence*	*Lift Ratio*
edible → free	0.96	0.97	0.98
edible → white_v	0.96	0.98	0.98
tapering → free	1.00	0.97	1.03
smooth_ar → free	0.96	0.97	0.99
smooth_br → free	0.96	0.97	0.99
white_ar → free	1.00	0.97	1.03
white_br → free	1.00	0.97	1.03
free → white_v	1.00	0.98	1.02
white_v → free	1.00	0.97	1.02
free → one	0.92	0.92	1.00
one → free	0.98	0.97	1.00
free → close	0.81	0.81	0.99
close → free	0.97	0.97	0.99
broad → free	0.96	0.97	0.99
tapering → white_v	1.00	0.98	1.02
tapering → one	1.00	0.92	1.08
smooth_br → smooth_ar	0.83	0.63	1.32
smooth_ar → white_v	0.96	0.98	0.99
smooth_ar → one	0.92	0.92	0.99
smooth_br → white_v	0.96	0.98	0.99
smoot_br → one	0.91	0.92	0.99
white_ar → white_v	1.00	0.98	1.02
white_ar → one	0.90	0.92	0.97
white_br → white_v	0.92	0.98	0.94
whie_br → one	0.97	0.92	1.06
one → white_v	0.83	0.98	0.85
smooth_ar → close	0.97	0.81	1.20
close → white_v	0.81	0.98	0.83
white_v → close	0.95	0.81	1.17

close → one	0.83	0.92	0.90
one → close	0.97	0.81	1.19
broad → white_v	0.89	0.98	0.91
broad → one	0.97	0.92	1.05
no → free	0.96	0.97	0.98
no → white_v	0.96	0.98	0.98
no → one	0.93	0.92	1.01
edible, free → white_v	1.00	0.98	1.02
edible, white_v → free	1.00	0.97	1.03
free, tapering → white_v	1.00	0.98	1.02
tapering, white_v → free	1.00	0.97	1.03
free, tapering → one	1.00	0.92	1.08
tapering, one → free	1.00	0.97	1.03
free, smooth_ar → white_v	1.00	0.98	1.02
smooth_ar, white_v → free	1.00	0.97	1.03
free, smooth_ar → one	0.91	0.92	0.99
smooth_ar, one → free	0.96	0.97	0.99
free, smooth_ar → close	0.83	0.81	1.02
smooth_ar, close → free	0.96	0.97	0.98
free, smooth_br → white_v	1.00	0.98	1.02
smooth_br, white_v → free	1.00	0.97	1.03
free, smooth_br → one	0.91	0.92	0.99
smooth_br, one → free	0.96	0.97	0.98
free, white_ar → white_v	1.00	0.98	1.02
white_ar, white_v → free	1.00	0.97	1.03
free, white_ar → one	0.90	0.92	0.97
white_ar, one → free	1.00	0.97	1.03
free, white_br → white_v	1.00	0.98	1.02
white_br, white_v → free	1.00	0.97	1.03
free, white_v → one	0.92	0.92	1.00
free, one → white_v	1.00	0.98	1.02
white_v, one → free	1.00	0.97	1.03

free, white_v → close	0.81	0.81	0.99
free, close → white_v	1.00	0.98	1.02
white_v, close → free	1.00	0.97	1.02
free, broad → white_v	1.00	0.98	1.02
broad, white_v → free	1.00	0.97	1.02
free, one → close	0.83	0.81	1.02
free, close → one	0.95	0.92	1.03
one, close → free	0.97	0.97	1.00
free, broad → one	0.89	0.92	0.96
one, broad → free	0.96	0.97	0.99
close,broad → free	0.95	0.97	0.98
tapering, white_v → one	1.00	0.92	1.08
tapering, one → white_v	1.00	0.98	1.02
smooth_ar, white_v → one	0.91	0.92	0.99
smooth_ar, one → white_v	0.96	0.98	0.98
smooth_br, white_v → one	0.91	0.92	0.99
smooth_br, one → white_v	0.96	0.98	0.98
white_ar, white_v → one	0.90	0.92	0.97
white_v, one → white_ar	1.00	0.53	1.88
close, smooth_ar → white_v	0.96	0.98	0.98
smooth_ar, white_v → close	0.83	0.81	1.02
close, white_v →one	0.95	0.92	1.02
close, one → white_v	0.97	0.98	0.99
white_v, one → close	0.83	0.81	1.02
close, broad → white_v	0.96	0.98	0.98
close, broad → one	0.92	0.92	1.00
one, broad → close	0.80	0.81	0.99
broad, white_v → one	0.89	0.92	0.96
broad, one → white_v	1.00	0.98	1.02
no, free → white_v	1.00	0.98	1.02
no, white_v → free	1.00	0.97	1.02
no, free → one	0.93	0.92	1.01

no, one, → free	0.96	0.97	0.98
no, white_v → one	0.93	0.92	1.00
no, one → white_v	0.96	0.98	0.98

Lampiran 20. Hasil Pengujian *Lift Ratio Rule* yang Terbentuk Menggunakan Algoritma *PHP*

Rule	*Confidence*	*Benchmark Confidence*	*Lift Ratio*
edible → free	0.96	0.97	0.98
edible → white_v	0.96	0.98	0.98
tapering → free	1.00	0.97	1.03
smooth_ar → free	0.96	0.97	0.99
smooth_br → free	0.96	0.97	0.99
white_ar → free	1.00	0.97	1.03
white_br → free	1.00	0.97	1.03
free → white_v	1.00	0.98	1.02
white_v → free	1.00	0.97	1.02
free → one	0.92	0.92	1.00
one → free	0.98	0.97	1.00
free → close	0.81	0.81	0.99
close → free	0.97	0.97	0.99
broad → free	0.96	0.97	0.99
tapering → white_v	1.00	0.98	1.02
tapering → one	1.00	0.92	1.08
smooth_br → smooth_ar	0.83	0.63	1.32
smooth_ar → white_v	0.96	0.98	0.99
smooth_ar → one	0.92	0.92	0.99
smooth_br → white_v	0.96	0.98	0.99
smoot_br → one	0.91	0.92	0.99
white_ar → white_v	1.00	0.98	1.02
white_ar → one	0.90	0.92	0.97
white_br → white_v	0.92	0.98	0.94
whie_br → one	0.97	0.92	1.06
one → white_v	0.83	0.98	0.85
smooth_ar → close	0.97	0.81	1.20
close → white_v	0.81	0.98	0.83
white_v → close	0.95	0.81	1.17

close → one	0.83	0.92	0.90
one → close	0.97	0.81	1.19
broad → white_v	0.89	0.98	0.91
broad → one	0.97	0.92	1.05
no → free	0.96	0.97	0.98
no → white_v	0.96	0.98	0.98
no → one	0.93	0.92	1.01
edible, free → white_v	1.00	0.98	1.02
edible, white_v → free	1.00	0.97	1.03