ASOSIASI TEMPAT WISATA DENGAN KATA KUNCI DI MALANG RAYA DAN SEKITARNYA DENGAN METODE ASSOCIATION RULE MINING MENGGUNAKAN ALGORITMA FP-GROWTH

SKRIPSI

Untuk memenuhi sebagian persyaratan   
memperoleh gelar Sarjana Komputer

Disusun oleh:

Fardan Ainul Yaqiin

NIM: 145150200111180



PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNIK INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

2018

# PENDAHULUAN

## Latar belakang

Objek wisata adalah suatu tempat yang memiliki daya tarik wisatawan untuk berkunjung, dimana objek wisata biasanya mempunyai nilai-nilai alami, budaya, atau sejarah tertentu. Menurut SK MENPARPOSTEL No: KM. 98/PW.102/MPPT-87, objek wisata adalah semua tempat atau keadaan [alam](https://id.wikipedia.org/wiki/Alam) yang memiliki sumber daya wisata yang dibangun dan dikembangkan sehingga mempunyai daya tarik dan diusahakan sebagai tempat yang dikunjungi [wisatawan](https://id.wikipedia.org/wiki/Wisatawan).

Menurut Arief Yahya Menteri Pariwisata Indonesia, wisatawan mancanegara tumbuh sebesar juta pada bulan Agustus 2017 dan jumlah wisatawan nusantara mencapai 200 juta. Untuk Kota Malang sendiri, pertumbuhan ekonomi Kota Malang mencapai 5,61% pada 2015 menurut Kasi Statistik Distribusi Badan Pusat Stastik Kota Malang Erny Fatma *Set*yoharini. Pertumbuhan ekonomi di Kota Malang salah satunya didukung oleh faktor pariwisata. Tingkat pertumbuhan wisatawan per tahun yang besar juga akan berdampak pada semakin banyaknya tempat wisata yang ada sehingga akan membuat wisatawan harus memilih tempat wisata mana yang akan mereka tuju dengan melakukan pertimbangan-pertimbangan yang ada seperti faktor keamanan, ketertiban, kebersihan, kenyamanan, keindahan, keramahan, dan kenangan yang mengacu pada sapta pesona, sehingga sebagai pengelola juga dapat dijadikan acuan pembenahan tempat.

*Association rule mining* diperkenalan untuk memecahkan masalah hubungan antar variable, dimana variable-variabel pada masalah ini adalah keterkaitan tempat wisata dan keadaan tempat wisata tersebut (Piatetsky, 1991). Banyak algoritma untuk menghasilkan *association rule*, beberapa yang terkenal seperti apriori, *Eclat*, dan *FP-growth*. Apriori adalah algoritma untuk *frequent itemset mining* dan *association rule learning* pada *database* transaksional (Agrawal, 1994). Algoritma Ini mengidentifikasi *frequent individual items* dalam *database* dan memperluasnya ke *item set* yang lebih besar dan selama *item set* tersebut masih ada dalam *database*. Algoritma Ini harus membentuk banyak *set* kandidat, juga harus berulang kalimengecek *database* untuk mencocokannya. Hal ini menyebabkan alokasi memori dan *runtime* yang tidak sedikit.

*Eclat*, singkatan dari *Equivalence Class Transformation* adalah algoritma *depth-first search* yang menggunakan *intersection set* (Zaki, 2000). Umumnya algoritma ini cocok untuk eksekusi sekuensial maupun paralel. Pada algoritma ini, semakin besar *tid-list*, maka semakin besar juga dan waktu yang dihabiskan untuk *intersection*. *FP-growth*, singkatan dari *frequent pattern* (Han, 2000). Pembentukan pohon dimulai dari bagian bawah *tabel* header, berurutan sesuai *frequent item* terbanyak, yang kemudian dilanjutkan oleh proses percabangan berdasarkan transaksi selanjutnya sampai transaksi terakhir sehingga terbentuk suatu pohon yang percabangannya membentuk semua asosiasi.

Dari ketiga algoritma *association rules* yang telah dijabarkan diatas, terdapat beberapa perbandingan. Dalam penggunaannya, apriori butuh lebih banyak pengecekan *database*. *Eclat* membutuhkan satu pemindaian *database* dan menemukan *itemset* tingkat berikutnya dengan memotong *itemset* saat itu, sedangkan *FP-growth* memiliki keuntungan dari struktur pohonnya, walaupun struktur datanya lebih kompleks dibandingkan Aprori dan *Eclat* (Solanki, 2014).

Penulis memilih algoritma *FP-growth* ketimbang algoritma apriori maupun *Eclat* dalam pengimplementasiannya dikarenakan *FP-growth* memiliki *runtime* dan penggunaan memori yang lebih sedikit ketimbang apriori, sehingga  *FP-growth* lebih *scalable* karena *running time*-nya linear (Hunyadi, 2011). Selain itu tujuan dari penelitian ini hanyalah sebatas pembentukan asosiasi faktor tempat wisata, sehingga *FP-growth* merupakan algoritma yang lebih cocok daripada algoritma yang lain.

Pada penelitian tentang pariwisata, yaitu Sistem Rekomendasi Tempat Wisata Di Kota Malang Menggunakan Metode *Hybrid Fuzzy-Floyd Warshall* (Firmansyah, 2017), penelitian tersebut berfokus pada rekomendasi tempat wisata. Disitu diberikan sebuah kasus dimana seseorang mempunya nilai-nilai variabel yang kemudian akan diproses untuk mencari tempat wisata yang paling sesuai berdasarkan variabel tersebut menggunakan *fuzzy*. *Set*elah itu menggunakan Floyd Warshall untuk mencari jarak terdekat.

Sedangkan untuk acuan *FP-Growth* sendiri, menggunakan penelitian yang berjudul Pencarian Association Rule Pada Data Pengguna Aplikasi Android Dengan Metode *FP*-*Growth* (Rafsanzani, 2015). Pada penelitian tersebut, bertujuan untuk mencari asosiasi aplikasi yang diinstal, dimana *set*iap aplikasi yang diinstal mempunyai keterkaitan sehingga berguna bagi pengembang aplikasi android untuk mengetahui apa-apa saja aplikasi yang saling berkaitan. Penelitian tersebut memiliki keluaran berupa *FP-tree* yang dibangun dari transaksi id berisi *itemset*, dimana *itemset* tersebut adalah kategori aplikasi android, *set*elah *fp-tree* terbentuk maka akan dilakukan perhitungan *lift ratio* untuk mengetahui kekuatan asosiasi.

Berdasarkan masalah yang d*item*ukan, tujuan dari penelitian ini adalah untuk mencoba mengimplementasikan metode *association rule mining* menggunakan algoritma *FP-growth*, menggunakan tempat-tempat wisata di kota Malang sebagai objeknya dan sifat-sifat dari tempat wisata tersebut sebagai *item set*-nya. Diharapkan dari dibuatnya dasar pengetahuan menggunakan metode *association rules mining* dengan algoritma *FP-growth* ini dapat dijadikan acuan oleh wisatawan untuk memilih tempat wisata, dan dijadikan cerminan bagi pengelola tempat wisata.

## Rumusan masalah

Berdasarkan latar belakang masalah yang telah dijelaskan, maka terdapat beberapa rumusan masalah sebagai berikut:

1. Bagaimana mengimplementasikan algoritma *FP-growth* *association rule mining* pada tempat wisata di Malang?
2. Bagaimana kinerja dari algoritma *FP-growth* untuk pembuatan dasar pengetahuan *association rule mining* tempat wisata di kota Malang?

## Tujuan

Tujuan yang ingin dicapai dari penelitian ini adalah:

1. Mengimplementasikan algoritma apriori pada *association rule* tempat wisata di Malang.
2. Mengetahui kinerja dari algoritma *FP-growth* untuk pembuatan dasar pengetahuan *association rule mining* tempat wisata di kota Malang.

## Manfaat

Adapun manfaat yang dapat didapatkan dari penelitian ini adalah sebagai berikut:

1. Sebagai referensi untuk tempat wisata sehingga membantu wisatawan dalam memilih tempat wisata yang ingin dikunjungi.

2. Cerminan untuk mengembangkan tempat wisatanya bagi pengelola tempat wisata di kota Malang.

3. Sebagai tolak ukur kemampuan dan menerapkan pengetahuan yang diperoleh dalam bangku perkuliahan untuk kasus lapangan.

## Batasan masalah

Pada penelitian ini terdapat batasan masalah agar masalah yang diteliti tidak menyimpang dan lebih terarah, antara lain:

Tempat wisata yang digunakan sebatas tempat wisata di Malang Raya (Kabupaten Malang, Kota Malang, Batu, dan sekitarnya)

Tempat wisata yang digunakan berjumlah empat.

Data yang digunakan adalah data hasil *scraping* komentar pada laman *web* tripadvisor*.*

Kata kunci yang digunakan mengacu pada tujuh sapta pesona

Menggunakan *association rule mining* dengan algoritma *FP-growth.*

## Sistematika pembahasan

Bagian ini berisi struktur skripsi ini mulai Bab Pendahuluan sampai Bab Penutup dan deskripsi singkat dari masing-masing bab. Diharapkan bagian ini dapat membantu pembaca dalam memahami sistematika pembahasan isi dalam skripsi ini.

# LANDASAN KEPUSTAKAAN

## Kajian Pustaka

Pada bagian ini dijelaskan referensi yang digunakan dalam menyelesaikan penelitian ini. Referensi yang dipilih adalah referensi tentang permasalahan yang dibahas dan juga tentang metode yang digunakan. Masalah yang dibahas adalah pembentukan asosiasi tempat wisata dengan kata kunci dan metode yang digunakan adalah metode *FP-growth*.

Penelitian pertama adalah penelitian dari Firmansyah (2017). Penelitian ini memiliki masukan berupa jarak, *budget,* dan waktu pengguna. Penelitian ini menggunakan 2 metode gabungan *Fuzzy* dan *Floyd Warshall.* Keluaran sistem berupa rekomendasi tempat wisata dan jaraknya. Dalam penelitian ini, *set*elah rekomendasi ditetapkan baru akan dicari jaraknya. Pengujian ini bertujuan untuk mendapatkan rekomendasi dan jarak tempat wisata. Penelitian kedua adalah penelitian dari Rafsnani (2015). Penelitian ini menggunakan metode *FP-Growth.* Penelitian ketiga adalah penelitian dari Pinheiro (2013). Penelitian ini menggunakan data pasien yang didiagnosa kanker.

**Tabel 2.1 Kajian Pustaka**

|  |  |  |  |
| --- | --- | --- | --- |
| **Judul** | **Masukan** | **Metode** | **Keluaran** |
| **Proses** | **Analisis** |
| Sistem Rekomendasi Tempat Wisata Di Kota Malang Menggunakan Metode Hybrid *Fuzzy*-Floyd Warshall  (Firmansyah, 2017) | * Data jarak, budget, dan waktu pengguna | *Fuzzy*-Floyd Warshall*.* | * rekomendasi wisata |
| * Proses algoritma *Fuzzy*. * Proses algoritma Floyd Warshall. | * jarak dihitung dari rekomendasi |
| Pencarian Association Rule Pada Data Pengguna Aplikasi Android Dengan Metode FP-Growth (Rafsanzani, 2015) | * Data survey tentang aplikasi yang digunakan | * FP-growth | * aturan asosiasi * minimum support dan inimum confidence berpengaruh pada aturan yang dihasilkan |
| pengenalan pola kanker hati untuk pencegahan dini menggunakan algoritma *FP-growth* (Pinheiro, 2013) | * Data pasien kanker | * FP-growth | * aturan asosiasi * minimum support dan inimum confidence berpengaruh pada aturan yang dihasilkan |

## Tempat Wisata

Tempat wisata adalah suatu tempat atau objek yang menjadi tujuan wisatawan dimana tempat tersebut memiliki daya tarik wisatawan untuk mengunjungi tempat tersebut. Berdasarkan SK MENPARPOSTEL No : KM. 98 / PW.102 / MPPT-87, sebuah tempat wisata adalah suatu bangunan atau alam yang memiliki daya tarik wisatawan yang dikembangkan sedemikian rupa sehingga menjadi tujuan wisata.

Tempat wisata dapat berbentuk objek alamiah seperti gunung, danau, sungai, pantai, laut, atau berupa bangunan seperti museum, benteng, taman hiburan, atau kota tua.

*Set*iap tempat wisata memiliki aspek-aspek tersendiri, seperti kebersihan, medan tempuh, biaya, dan kesenangan. Aspek kebersihan adalah aspek yang menyangkut kebersihan tempat tersebut, kemudian medan tempuh merupakan aspek seberapa sulit menuju tempat tersebut, lalu ada biaya yaitu seberapa banyak uang yang harus dikeluarkan untuk menuju maupun menikmati tempat wisata, dan terakhir aspek kesenangan yang mrupakan seberapa menyenangkan tempat wisata tersebut.

### Tempat Wisata di Malang Raya dan Sekitarnya

Sama seperti kota-kota lainnya di Indonesia, Kota Malang tumbuh dan berkembang *set*elah adanya kehadiran pemerintahan kolonial Belanda. Fasilitas umum dibuat sedemikian rupa untuk memenuhi kebutuhan orang-orang Belanda dan Eropa. Suatu diskriminasi saat itu masih dapat terlihat sampai sekarang, contohnya adalah di sekitar Jalan Besar Idjen atau *Idjen Boulevard*, dimana hanya orang-orang tertentu saja yang bisa tinggal di sana. Sekarang rumah-rumah tersebut menjadi suatu tempat wisata sejarah tersendiri untuk bernostalgia.

Penulis memilih empat tempat wisata di Malang Raya dan sekitarnya yaitu Taman Nasional Bromo Tengger Semeru untuk mewakili tempat wisata alam berupa gunung, kemudian Pantai Goa Cina mewakili tempat wisata alam berupa pantai, lalu Alun-Alun Kota Malang sebagai tempat wisata taman kota, dan terakhir Taman Rekreasi Jatim Park 2 untuk merepresentasikan taman rekreasi, museum, dan kebun binatang.

#### Taman Nasional Bromo Tengger Semeru

Taman Nasional Bromo Tengger terletak di wilayah administratif [Kabupaten Pasuruan](https://id.wikipedia.org/wiki/Kabupaten_Pasuruan), [Kabupaten Malang](https://id.wikipedia.org/wiki/Kabupaten_Malang), [Kabupaten Lumajang](https://id.wikipedia.org/wiki/Kabupaten_Lumajang) dan [Kabupaten Probolinggo](https://id.wikipedia.org/wiki/Kabupaten_Probolinggo). Menurut Departemen Kehutanan pada tahun 2010, tempat ini adalah satu-satunya kawasan konservasi di Indonesia yang memiliki lautan pasir, yaitu Laut Pasir Tengger. Tempat ini mencakup area seluas 5.250 hektar di ketinggian sekitar 2.100 meter, juga berisi gunung tertinggi di Jawa, Gunung Semeru (3,676 m), empat danau dan 50 sungai.

Lautan Pasir Tengger telah dilindungi sejak 1919. Taman Nasional Bromo Tengger Semeru dinyatakan sebagai taman nasional pada tahun 1982 berdasarkan Surat Pernyataan Menteri Pertanian No.736/Mentan/X/1982.

#### Pantai Goa Cina

Pantai Goa China terletak di pesisir selatan di [Dusun Tumpak Awu](https://id.wikipedia.org/w/index.php?title=Dusun_Tumpak_Awu&action=edit&redlink=1), [Desa Sitiarjo](https://id.wikipedia.org/w/index.php?title=Desa_Sitiarjo&action=edit&redlink=1), [Kecamatan Sumbermanjing Wetan](https://id.wikipedia.org/wiki/Sumbermanjing_Wetan,_Malang), [Kabupaten Malang](https://id.wikipedia.org/wiki/Kabupaten_Malang), [Jawa Timur](https://id.wikipedia.org/wiki/Jawa_Timur). Pantai yang memiliki nama asli Pantai Rowo Indah ini dapat diakses melalui jalur lingkar selatan. Saat menuju lokasi terdapat banyak tikungan dan jurang di sisi jalan. Untuk menuju lokasi terdapat rambu penunjuk jalan yang mengarah pada pantai tersebut.

#### Alun-Alun Kota Malang

Taman terbesar di kota Malang tidak lain adalah Alun-Alun Merdeka dan Alun-Alun Tugu (Monumen Tugu Malang). Alun-Alun Kota atau yang biasa disebut Alun-Alun Merdeka ini terletak di depan Kantor Bupati Malang dan merupakan alun-alun tertua yang dibangun pada 1882. Selain itu, Alun-Alun Merdeka juga menyediakan fasilitas bermain anak-anak dan air mancur.

#### Taman Rekreasi Jatim Park 2

Jatim Park 2 terletak di Jl. Oro-Oro Ombo No.9, Temas, Kec. Batu, Kota Batu, Jawa Timur. Merupakan tempat bermain dan belajar yang meliputi Museum Satwa, Batu Secret Zoo, dan Pohon Inn Hotel. Tidak seperti Jatim Park 1, pada Jatim Park 2 lebih menekankan pada tempat pembelajaran ilmu alam, biologi, dan satwa. Terdapat area istirahat yang menyediakan tempat bermain dan penjual makanan.

## Sapta Pesona

Sapta Pesona merupakan suatu program yang dikampanyekan pada program Visit Indonesia Year 1991. Program ini sendiri memiliki tujuan untuk meningkatkan pariwisata dengan membangun pribadi *set*iap lapisan masyarakat yang ada. Sapta Pesona memiliki suatu logo bergambarkan Matahari dengan 7 buah sinarnya yang memiliki masing-masing perwakilan, yaitu keamanan, ketertiban, kebersihan, kesejukan, keindahan, keramahan, dan kenangan.

Pada penelitian ini, sifat-sifat dari sapta pesona akan dijadikan sebagai acuan untuk menyaring komentar yang nantinya akan dijadikan sebagai *item* pada transaksi pembentukan pohon *FP*, dari setiap sifat yang ada akan dicari kata yang mengarah pada sifat tersebut sehingga nantinya akan dikelompokan seperti pada Tabel 2.2 sampai Tabel 2.4, yang kemudian akan dikelompokan lagi menjadi lebih sederhana seperti pada Tabel 2.5.

**Tabel 2.2 Turunan kata sapta pesona 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| antre | awas | bahaya | berantakan | bingung | bohong | bosan |
| ngantri | ati | berbahaya | berserakan | membingungkan | boong | membosankan |
| mengantri | hati |  | brantakan | bingungin | dibohongi | bosenin |
| antri | hati-hati |  |  |  | diboongin | bosen |
|  | ati2 |  |  |  | ketipu | bsn |
|  | ati-ati |  |  |  | ditipu | bsen |
|  | hati2 |  |  |  | penipu | bsan |
|  |  |  |  |  | penipuan | bosn |
|  |  |  |  |  | bhg |  |

**Tabel 2.3 Turunan kata sapta pesona 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| cepat | copet | dekat | galak | mudah | hilang |
| cepet | dicopet | deket | judes | gampang | ilang |
| cpt | kecopetan | dkt | jutek | gmpg | kehilangan |
|  | pencopet |  | cemberut | gmpng | ilg |
|  |  |  | masam | gmpang |  |
|  |  |  | sebal |  |  |
|  |  |  | menyebalkan |  |  |
|  |  |  | nyebelin |  |  |

**Tabel 2.3 Turunan sapta pesona 3**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| khas | kotor | mahal | panas | percuma | ramah | rapi |
| ciri | jorok | menguras | terik | sia sia | sopan | rapih |
| unik | ktr | mhl | pns | sia2 |  | beraturan |
|  | jrk |  |  | percuma |  | tertata |
|  |  |  |  |  |  | tertib |
|  |  |  |  |  |  | rp |
|  |  |  |  |  |  | rpih |
|  |  |  |  |  |  | rpi |

**Tabel 2.4 Turunan kata sapta pesona 4**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| serobot | sesat | sulit | tawa | tenteram |
| nyerobot | tersesat | susah | senyum | tentram |
| menyerobot | nyasar |  | tersenyum | tentrem |
| serobotan |  |  | nyengir |  |
| nyrobot |  |  | lucu |  |
| srobot |  |  | mesem |  |
| srobotan |  |  |  |  |

**Tabel 2.5 Turunan kata sapta pesona 5**

|  |  |  |
| --- | --- | --- |
| bagus | jelek | sejuk |
| gembira | buruk | dingin |
| senang | sedih | segar |
| menyenangkan | menyedihkan | seger |
| baik | parah | mendung |
| indah | sempit | rindang |
| menarik | jlk | dgn |
| nyaman | sdh | dgin |
| ok | sdih | sgr |
| cantik | jlek | sger |
| luas | prah | sgar |
| gaul | smpt | segr |
| mantab | smpit | mndg |
| mantaps | sempt | mendg |
| gede |  | mndung |
| gde |  | rndg |
| kren |  |  |
| oke |  |  |
| kece |  |  |
| bgs |  |  |
| sng |  |  |
| cntk |  |  |
| keren |  |  |
| mantap |  |  |
| super |  |  |
| superr |  |  |
| superrr |  |  |

**Tabel 2.6 Penyederhanaan kelompok hasil turunan kata sapta pesona**

|  |  |
| --- | --- |
| aman | lama |
| antre | lancar |
| awas | macet |
| bagus | mahal |
| bahaya | mudah |
| berantakan | murah |
| bersih | panas |
| bingung | percuma |
| bohong | ramah |
| bosan | rapi |
| buruk | rawan |
| cepat | sedih |
| copet | sehat |
| dekat | sembarangan |
| galak | serobot |
| hilang | sesat |
| jauh | sulit |
| kecelakaan | tawa |
| khas | tenteram |
| kotor | sejuk |

## *Web Scraping*

*Web scraping, web harvesting*, atau *web data extraction* adalah ekstraksi data dari situs *web*. Perangkat lunak *web scraping* dapat mengakses *World Wide Web* secara langsung menggunakan *Hypertext Transfer Protocol*, atau melalui *web browser*. Walaupun *web scraping* dapat dilakukan secara manual oleh pengguna perangkat lunak, istilah ini biasanya mengacu pada proses otomatis yang diimplementasikan menggunakan *bot* atau penjelajah web. Data tertentu dikumpulkan dan disalin dari *web*, biasanya ke pusat basis data lokal atau *spreadsheet*, untuk dijadikan temu kembali atau analisis.

### *Web Scraper Extension*

Pada penelitian ini, untuk pengambilan data akan diambil melalui situs web dengan menggunakan *web scraper extension* pada *web browser* Google Chrome. *Web* *Scraper* adalah *extension*  pada *web browser* Google Chrome untuk melakukan *scraping,* ekstraksi data dari halaman web. Data yang diekstrak nantinya akan diekspor sebagai *CSV*.

## *Association Rule Learning*

*Association Rule Learning* adalah metode pembelajaran berbasis aturan untuk menemukan hubungan yang antara variabel dalam *database* (Gregory, 1991). Berdasarkan konsep Rakesh Agrawal, Tomasz Imieliński dan Arun Swami memperkenalkan *association rules* untuk menemukan keteraturan antara produk dalam data transaksi berskala besar yang dicatat oleh sistem penjualan *point-of-sale (POS)* di supermarket. Misalnya, aturan dapat d*item*ukan pada data penjualan supermarket yang mengindikasikan jika ada yang membeli bawang dan kentang, maka kemungkinan mereka juga akan membeli daging burger.

Selain contoh di atas, penggunaan *association rules* analisis juga digunakan pada [*Web usage mining*](https://en.wikipedia.org/wiki/Web_usage_mining), deteksi intrusi, produksi kontinu, dan bioinformatika. Berbeda dengan [*sequence mining*](https://en.wikipedia.org/wiki/Sequence_mining)*, association rule learning* biasanya tidak mempertimbangkan urutan barang baik dalam transaksi atau lintas transaksi.

**Tabel 2.7 Transaksi *item***

|  |  |
| --- | --- |
| *TID* | *Items* |
| 1 | *Bread, Milk* |
| 2 | *Bread, Diaper, Beer, Eggs* |
| 3 | *Milk, Diaper, Beer, Coke* |
| 4 | *Bread, Milk, Diaper, Beer* |
| 5 | *Bread, Milk, Diaper, Coke* |

Tabel merupakan salah satu contoh data*set* yang memilki *Transaction ID (TID)* yaitu *ID* transaksi dan *Itemsets* yaitu sekumpulan satu atau lebih *item* misal: {*Milk, Bread, Diaper*}.

### *Support*

*Support* adalah indikasi seberapa sering *itemset* muncul di data*set* (Michael, 2005). *Support* dari *itemset* X sehubungan dengan *set* transaksi T didefinisikan sebagai proporsi transaksi t dalam data*set* yang berisi *itemset* X.

(2.1)

Dalam contoh data*set* pada Gambar 2.1, *itemset* X = {*Beer, Diaper*} memiliki *support* 3/5 = 0,6 karena itu terjadi pada 60% dari semua transaksi (3 dari 5 transaksi).

### *Confidence*

*Confidence* adalah indikasi seberapa sering aturan itu terbukti benar (Michael, 2005). Nilai *confidence* dari X ⇒ Y adalah seberapa sering *item* dalam y muncul darlam transaksi yang terdiri dari X

(2.2)

Misalnya, aturan {*Milk, Diaper*} ⇒ {*Beer*} memiliki *confidence* 0.4 / 0.6 = 0,67 dalam *database*, yang berarti bahwa untuk 67% dari transaksi yang mengandung *Milk* dan *Diaper* aturannya benar (67% dari waktu pelanggan membeli *Milk* dan *Diaper*, *Beer* dibeli juga).

### *Lift*

Dalam *data mining* dan *association rule*, *lift* adalah ukuran kinerja dalam memprediksi atau mengklasifikasikan suatu kasus yang diukur terhadap penargetan pilihan acak model. Model penargetan melakukan pekerjaan dengan baik jika respons dalam target jauh lebih baik daripada rata-rata untuk populasi secara keseluruhan (Stéphane, 2011).

(2.3)

Misalnya, aturan {*Milk*, *Bread*} ⇒ {*Diaper*} memiliki lift 0,83.

### Algoritma

Banyak algoritma untuk menghasilkan *association rules* telah diajukan. Beberapa algoritma yang terkenal adalah Apriori, *Eclat* dan *FP-Growth*.

#### Algoritma Apriori

Algoritma Apriori diusulkan oleh Agrawal dan Srikant pada tahun 1994. Apriori dirancang untuk beroperasi pada *database* yang berisi transaksi (misalnya, koleksi barang yang dibeli oleh pelanggan, atau rincian frekuensi situs *web*). Algoritma lainnya dirancang untuk menemukan aturan asosiasi dalam data yang tidak memiliki transaksi (Winepi dan Minepi), atau tidak memiliki cap waktu (sequencing DNA). *Set*iap transaksi dilihat sebagai satu *set* *item* (*itemset*). Dengan ambang *threshold* **C**, algoritma Apriori mengidentifikasi kumpulan *item* yang merupakan himpunan bagian dari *set*idaknya transaksi ***C*** dalam *database*.

Apriori menggunakan pendekatan "*bottom up*", di mana sub*set* yang sering muncul diperpanjang satu *item* pada satu waktu (sebuah langkah yang dikenal sebagai generasi kandidat), dan kelompok kandidat diuji terhadap data tersebut. Algoritma berakhir saat tidak ada perpanjangan lagi.

Apriori menggunakan [*breadth-first search*](https://en.wikipedia.org/wiki/Breadth-first_search)dan[*Hash tree*](https://en.wikipedia.org/wiki/Hash_tree_(persistent_data_structure))untuk menghitung *itemset* kandidat secara efisien. Apriori menghasilkan rangkaian *item* kandidat dengan panjang ***k*** dari *itemsets* dengan panjang ***k - 1***. Kemudian memotong kandidat yang memiliki sedikit *frequent*. Menurut *downwar closure lemma, set* kandidat berisi serangkaian *item* k-panjang *item*. *Set*elah itu, dilakukan pemindaian *database* transaksi untuk menentukan *item* *set* yang sering muncul di antara kandidat.

*Pseudo code* untuk algoritma aprori dijabarkan pada gambar 2.2 ***T*** untuk *database* transaksional, dan *threshold* dari ***ε***. Usual menetapkan notasi teoritis digunakan, namun perhatikan bahwa T adalah multi*set*. ***Ck*** adalah *set* kandidat yang ditetapkan untuk level ***k***. Pada *set*iap langkahnya, algoritma ini diasumsikan menghasilkan *set* kandidat dari *large item sets* pada level sebelumnya, mengikuti *downward closure lemma*. ***count [c]*** mengakses *field* dari struktur data yang mewakili *set* kandidat ***c***, yang pada awalnya diasumsikan nol. Berikut merupakan *pseudocode* algoritma Apriori.

*Ck: Candidate itemset of size k*

*Lk : frequent itemset of size k*

*L1 = {frequent items};*

***for*** *(k = 1; Lk !=∅; k++)* ***do begin***

*Ck+1 = candidates generated from Lk;*

***for each*** *transaction t in database do*

*increment the count of all candidates in Ck+1*

*that are contained in t*

*Lk+1 = candidates in Ck+1 with min\_support*

***end***

***return*** *∪k Lk;*

#### Algoritma Eclat

Algoritma *Eclat* (Equivalence Class Transformation) adalah suatu algoritma untuk mencari koleki *frequent items* (Zaki, 1997), algoritma ini menggunakan *depth first search* pada kisi bagian dan menentukan dukungan *set* *item* dengan memotongkan daftar transaksi. Algoritma *Eclat* awalnya membuat pohon *prefix*, kemudian mempeluas *item* *set* berdasarkan *threshold*-nya dengan pertimbangan *frequent*-nya sampai habis, kemudian diulang lagi. Algoritma ini membuat daftar pengenal transaksi yang mengandung *itemset* di dalamnya, kemudian menetapkan support dari *itemset* tersebut dengan melakukan interseksi. Berikut merupakan *pseudocode* algoritma *Eclat.*

*2: // add to create a new prefix*

*3: // initialize a new equivalence class with the new prefix P*

*10:*

*11: then*

*12:*

*13:*

*14:*

#### Algoritma FP-growth

Algoritma FP-Growth adalah cara alternatif untuk menemukan *frequent itemset* tanpa menggunakan kandidat generasi sehingga meningkatkan kinerjanya dengan cara melakukan *divide and conquer* (Han, 2000).Inti dari metode ini adalah *frequent pattern tree* (pohon FP), yang menyimpan informasi asosiasi *itemset*.

Sederhananya, algoritma ini bekerja sebagai berikut: pertama, kompres *input*-an *database* sehingga membentuk *FP-tree* untuk mewakili *frequent item*. *Set*elah langkah pertama ini, bagi *database* terkompresi menjadi satu *set* *database* kondisional, masing-masing terkait dengan satu *frequent item*. Akhirnya, *set*iap *database* tersebut ditambang secara terpisah. Dengan menggunakan strategi ini, FP-growth mengurangi biaya pencarian untuk menyimpan dan mencari *frequent pattern*, menawarkan selektivitas yang baik.

### Struktur FP-tree

*Frequent Pattern tree* adalah struktur yang menyimpan informasi kuantitatif tentang pola yang sering terjadi dalam *database*.

Pohon *FP* sebagai struktur pohon dapat didefinisikan seperti di bawah ini:

Satu akar diberi label sebagai "*null*" dengan seperangkat *subtitle* *item* prefix sebagai anaknya, dan tabel *frequent-item-header* seperti pada gambar 2.4

*Set*iap *node* di subtree *item*-*prefix* terdiri dari tiga bagian:

*Item*-name: register yang *item*nya ditunjukkan oleh *node*;

*Count*: jumlah transaksi yang ditunjukkan oleh tiap pencarian yang dilalui;

*Node*-link: menghubungkan ke *node* berikutnya di pohon *FP* yang membawa nama *item* yang sama, atau *null* jika tidak ada.

1. *Set*iap entri dalam tabel frequent-*item*-header terdiri dari dua bagian:
   1. *Item*-name: sama dengan *node*;
   2. Kepala *node*-link: pointer ke *node* pertama di *FP-tree* yang membawa nama *item*.

Berikut langkah-langkah untuk pembentukan FP-Tree:

1. Pindai *database* transaksi DB sekali. Kumpulkan F, *set frequent item* dan support tiap *frequent item.item* yang sering, dan dukungan dari *set*iap *item* yang sering. Urutkan F dari besar tekecil berasarkan *frequent itemnya.*
2. Buat akar pohon FP, T, dan tandai sebagai "*null*". Untuk *set*iap trans transaksi di DB lakukan hal berikut:

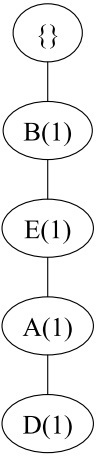
* Pilih *item* pada urutan yang pertama, kemudian buat percabangan sesuai dengan *frequent item*-nya, jika sudah ada cabang yang sesuai maka tambahkannilai pada *node* yang dilalui, jika tidak ada cabang yang sesuai maka buat *node* cabang baru.

Perhatikan transaksi di bawah ini dan minimal support bernilai 3.

Tabel 2.8 Contoh Transaksi

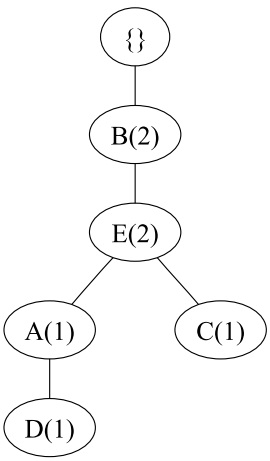
|  |  |
| --- | --- |
| Transaction ID | I(t) |
| 1 | ABDE |
| 2 | BCE |
| 3 | ABDE |
| 4 | ABCE |
| 5 | ABCDE |
| 6 | BCD |

Untuk membangun FP-Tree, support *item* sering dihitung terlebih dahulu dan diurutkan berdasarkan urutan menurun sehingga menghasilkan daftar berikut: {B (6), E (5), A (4), C (4), D (4)} . *Set*elah itu, FP-Tree disusun secara iteratif untuk *set*iap transaksi, dengan menggunakan daftar *item* yang diurutkan seperti yang ditunjukkan pada Gambar 2.5 sampai Gambar 2.10.



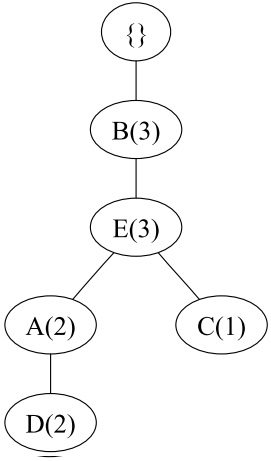
Gambar 2.5 Transaksi 1: BEAD

Pertama bangun sebuah *node* kosong sebagai akar, kemudian pada transaksi pertama, buat *node* sesuai urutan *frequent item*-nya. *Itemset* pada transaksi pertama adalah ABDE, yang menjadi BEAD *set*elah diurutkan sesuai *frequent item*-nya. Sehingga didapatkan seperti Gambar 2.5, yaitu terdapat *node* B(1) di bawah akar yang berarti *node* B baru dilewati satu kali, kemudian dibawah *node* B terdapat *node* E(1), dan *set*erusnya sampai *node* D(1).



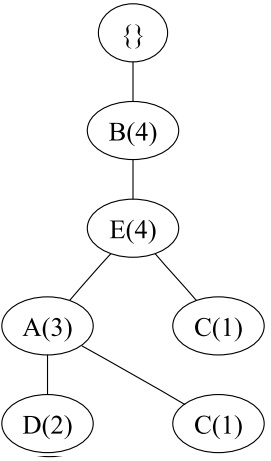
Gambar 2.6 Transaksi 2: BEC

*Set*elah itu masuk ke transaksi kedua, yaitu BCE yang diurutkan menjadi BEC. Karena *node* B sudah terbentuk, maka langsung menambahkan *count* saja tanpa membuat *node* baru, lalu sama seperti *node* B langsung masuk ke *node* E. Dikarenakan cabang pada *node* E baru memiliki *node* A(1), sdangkan pada transaksi membutuhkan *node* C, maka dibuat *node* baru, yaitu C(1) dibawah *node* E bersampingan dengan *node* A sehingga tampak seperti Gambar 2.6.



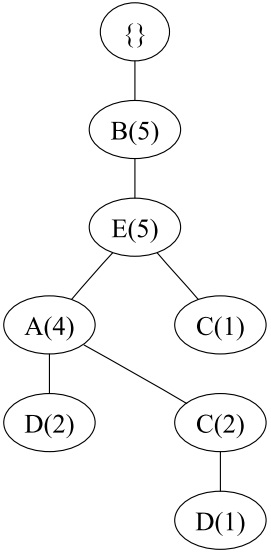
Gambar 2.7 Transaksi 3: BEAD

*Itemset* pada transaksi ketiga adalah ABDE yang berubah menjadi BEAD. Karena transaksi ini sama dengan transaksi pertama, maka hanya perlu menambahkan *count* pada tiap *node* yang dilewati sehingga seperti Gambar 2.7.



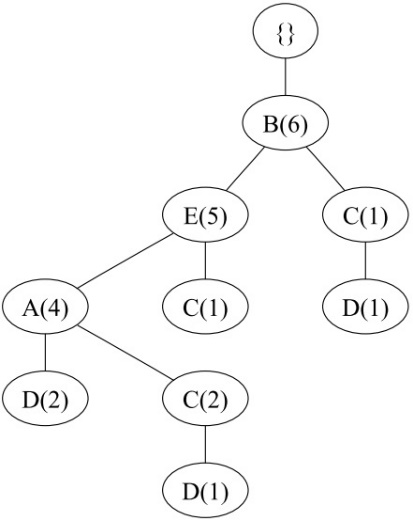
Gambar 2.8 Transaksi 4: BEAC

Transaksi keempat memiliki *itemset* ABCE yang kemudian berubah menjadi BEAC. Pada *node* B, E, dan A hanya perlu melewati dan menambahkan *count*, karena *set*elah *node* A hanya terdapat *node* D dan tidak ada *node* C maka buat *node* baru yaitu *node* C dibawah *node* A disamping *node* D lalu isi *count*­-nya satu seperti pada Gambar 2.8.



Gambar 2.9 Transaksi 5: BEACD

Pada transaksi kelima yaitu ABCDE berubah menjadi BEACD, karena mirip dengan transaksi keempat maka hanya perlu melewati *node*-*node* yang sudah ada yaitu BEAC kemudian menambahkan *count*­-nya kemudian tambahkan *node* baru yaitu *node* D dibawah *node* C seperti pada Gambar 2.9.



Gambar 2.10 Transaksi 6: BCD

Pada transaksi terakhir yaitu transaksi keenam dengan *itemset* BCD, karena tidak ada *node* C *set*elah *node* B maka buat *node* baru dibawahnya yang kemudian dilanjutkan dengan *node* D sehingga akan terbentuk pohon *FP* seperti pada Gambar 2.10.

# METODOLOGI

## Tipe Penelitian

Penelitian ini berupa nonimplementatif yang berfokus pada investigasi terhadap suatu kondisi. Pendekatan ini menggunakan pendekatan deskriptif, mempelajari masalah yang ada pada lingkungan sekitar untuk dicari permasalahannya. Fokus masalah dari penelitian ini adalah mencari asosiasi tempat wisata dengan kata kunci.

## Strategi Penelitian

Pada penelitian ini menggunakan metode penelitian eksperimen sebagai metode yang digunakan. Metode eksperimen yaitu mencari pengaruh variabel tertentu terhadap variabel lainnya dalam kondisi terkontrol. Metode yang digunakan dalam penelitian ini adalah *Association Rule* dengan menggunakan algoritma *FP-growth.*

## Teknik Pengumpulan Data

Data penelitian yang digunakan adalah data yang diambil dari hasil *scraping* dari laman *web* tripadvisor dengan empat tempat wisata, yaitu taman nasional bromo semeru tengger, pantai goa cina, alun-alun kota malang, dan taman rekreasi Jatimpark 2. Jumlah data yang digunakan adalah 200 komentar untuk setiap tempat wisata.

## Implementasi Algoritme

Dalam penelitian pembentukan asosiasi tempat wisata dengan kata kunci menggunakan algoritma *association rule* dengan metode *FP-Growth* dilakukan tahapan-tahapan dalam implementasi:

*Text Processing* data yang diambil dari hasil *scraping* dan dibentuk dalam table transaksi*.*

Pembentukan pohon *FP* menggunakan table transaksi yang telah dibuat.

Keluaran sistem berupa aturan yang terbentuk setelah perhitungan *lift ratio.*

## Jadwal Penelitian

Jadwal penelitian akan dilaksanakan dalam waktu terhitung dari bulan Februari hingga Juni. Berikut ini merupakan jadwal penelitian akan ditunjukan pada tabel.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| No | Uraian | Februari | | | | Maret | | | | April | | | | Mei | | | | Juni | | | |
| Minggu ke- | | | | | | | | | | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |
| 1 | Studi Kepustakaan |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | Pengumpulan Data |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | Implementasi Algoritma |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 | Pengujian dan analisis |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | Kesimpulan dan saran |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

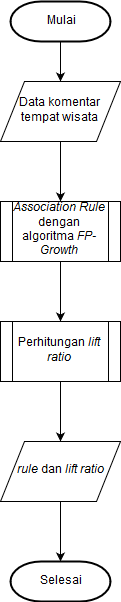
# PERANCANGAN

## Perancangan Proses

Sistem yang akan dibuat merupakan sistem yang dikembangkan untuk mencari pola keterkaitan antara kata kunci dengan tempat wisata. Input yang digunakan dalam analisa adalah komentar tempat wisata.

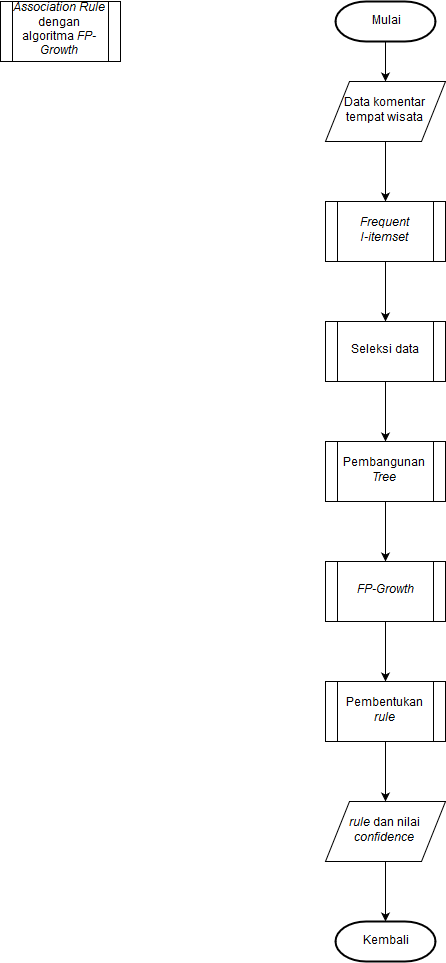
Algoritma yang digunakan dalam sistem ini adalah algoritma *association rule* untuk melakukan *frequent itemset mining*. Untuk mencari *frequent itemset* pada sistem yang akan dibuat menggunakan metode *frequent patter growth*.

*Itemset* yang digunakan pada sistem nantinya akan adalah komentar tempat wisata. Pada Gambar 4.1 menjelaskan tahapan-tahapan untuk mendapatkan *rule.*



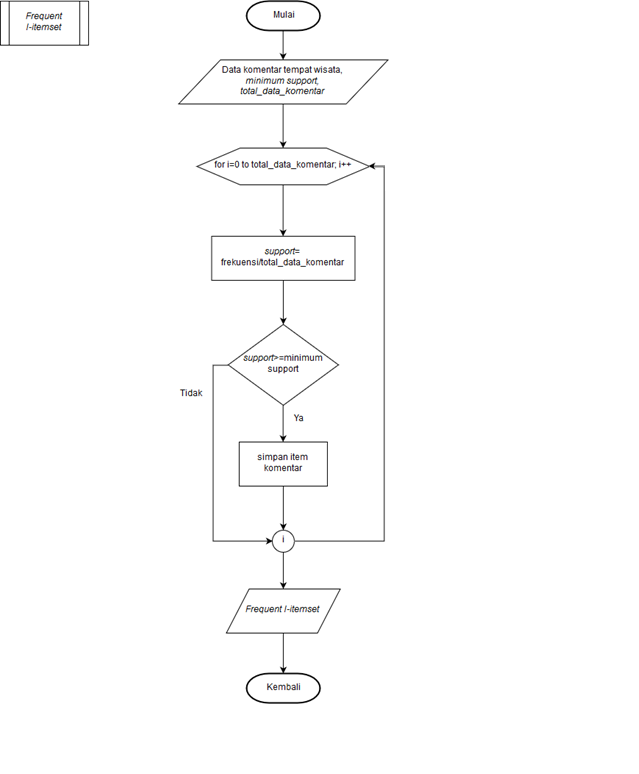
**Gambar 4.1 *Flowchart sistem***

Pada Gambar 4.1, input yang dibutuhkan oleh sistem adalah data komentar tempat wisata. Selanjutnya adalah proses *FP-Growth* dan perhitungan *lift ratio* sehingga keluaran dari proses ini adalah *rule* dan *lift ratio.* Untuk tahap *association rule dengan metode fp-growth* akan dijelaskan pada Gambar 4.2



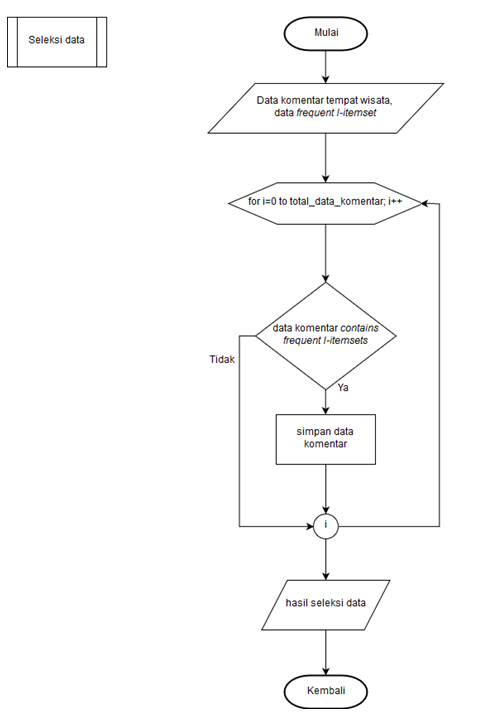
**Gambar 4.2 *Flowchart FP-growth***

Tahapan-tahapan *association rule* dengan *FP-growth* dijelaskan pada Gambar 4.2*.* Pertama sistem akan mencari *frequent* *I*-*itemset*, kemudian melakukan seleksi data yang berisi kata kunci tempat wisata, lalu hasil dari seleksi data digunakan sebagai *input tree*. Hasil pembangunan tree dilakukan dengan proses *FP-growth, set*elah itu dihitung nilai *confidence*-nya, nilai yang memenuhi *minimum confidence* akan dijadikan sebagai *rule*.



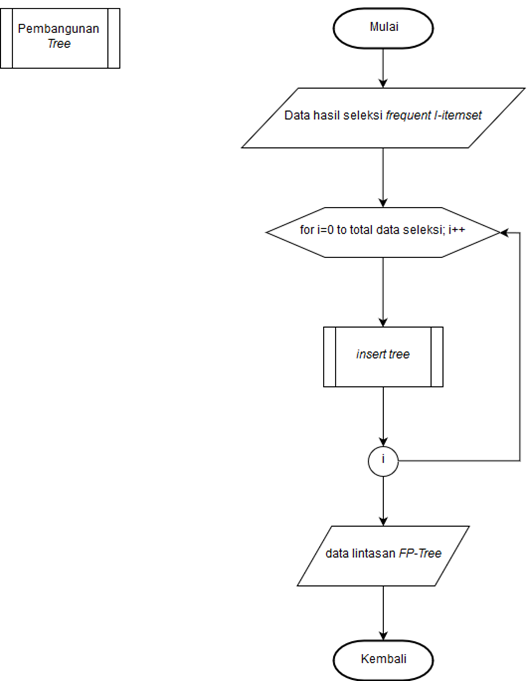
**Gambar 4.3 *Flowchart frequent itemset***

*Item* yang memenuhu nilai *minimum support* akan dijadikan sebagai *Frequent I-itemset* seperti pada Gambar 4.3. *Item* yang memenuhi *minimum support* akan diproses ke pembangunan *tree.*

**

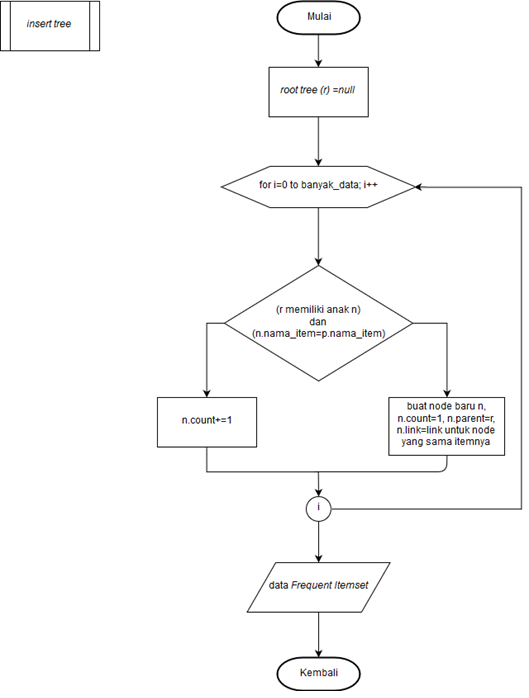
**Gambar 4.4 *Flowchart* seleksi data**

Pada Gambar 4.4 merupakan proses seleksi data. Masukan dari proses ini berupa data komentar dan hasil *frequent I-itemset* untuk menyaring data yang hanya mengandung *frequent I-itemset* dengan keluaran berupa data yang telah diseleksi*.*



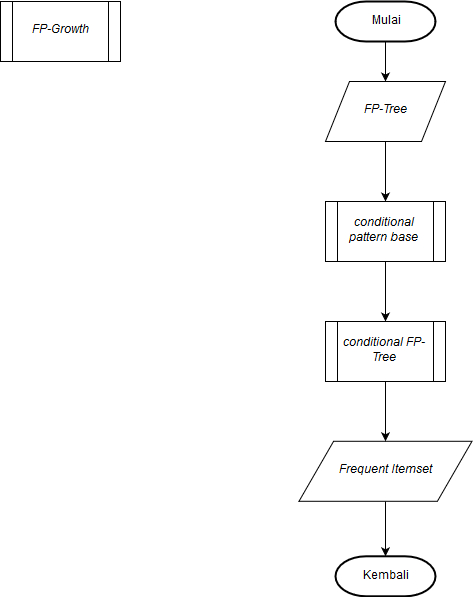
**Gambar 4.5 *Flowchart* pembangunan *tree***

Untuk mendapatkan lintasan *FP-Tree,* setiap data hasil seleksi akan dimasukan ke dalam proses *insert tree* untuk mendapatkan *frequent itemset* seperti pada Gambar 4.5 yang merupakan tahap pembangunan *tree.*



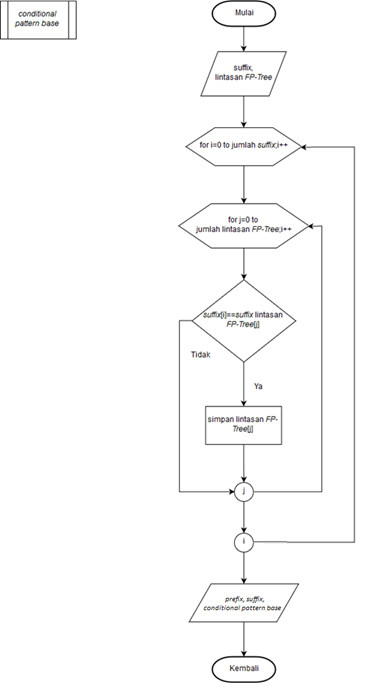
**Gambar 4.6 *Flowchart* *insert tree***

Pada Gambar 4.6 merupakan tahap *insert tree* yang memiliki keluaran berupa data *Frequent Itemset.* Langkah pertama dengan membuat *root* dengan nilai *null*, kemudian dilakukan pembuatan *node* untuk data komentar. Jika pada *tree* belum pernah ada *node* dengan label data tersebut maka dibuat node baru dan diberi nilai *count* 1, namun jika sebelumnya sudah ada maka hanya menambahkan nilai 1 pada *count*.



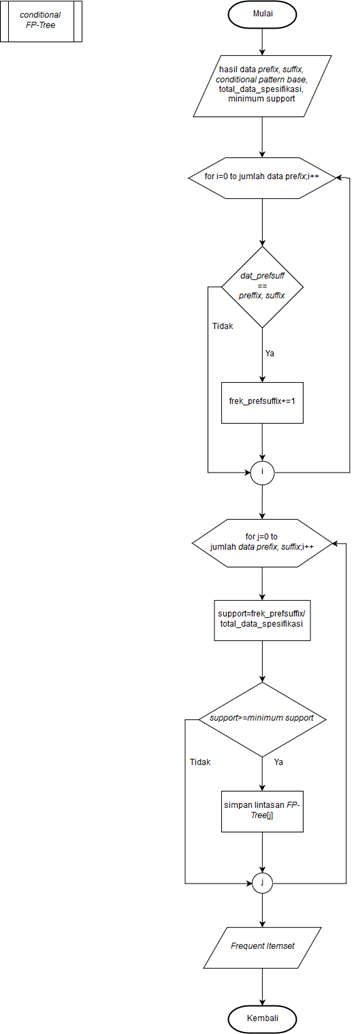
**Gambar 4.7 *Flowchart* *FP-Growth***

Tahapan algoritma *FP-Growth* untuk mendapatkan *Frequent Itemset* dengan cara membangun *conditional pattern base* dan *conditional FP-Tree* dari data hasil pembangunan *FP-Tree* dijelaskan pada Gambar 4.7*.*



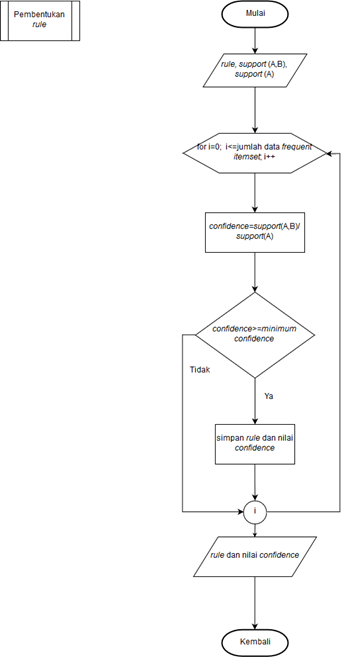
**Gambar 4.8 *Flowchart conditional pattern base***

Pada Gambar 4.8 akan mencari *prefix, suffix,* dan *conditional pattern base* dari lintasan *FP-Tree* yang telah terbentuk melalui proses pembangunan *FP-Tree.* Hasil keluaran proses *conditional pattern base* ini adalah data dari setiap *suffix*.



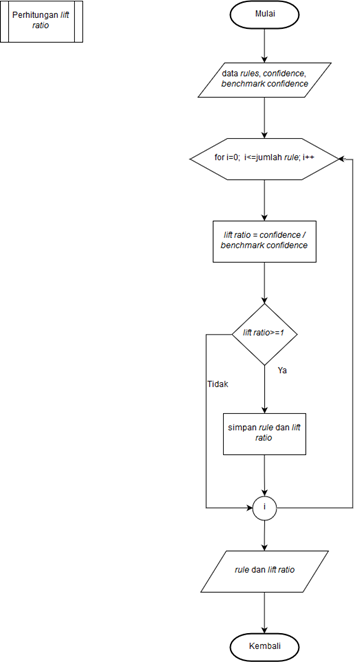
**Gambar 4.9 *Flowchart* *conditional FP-Tree***

Pada tahap *conditional FP-Tree* ini seperti pada Gambar 4.9 akan dicari frekuensi dan nilai support tiap *item* *prefix* dan *suffix* yang mempunyai lintasan yang sama. Dan jika nilai *support* dari kombinasi *item prefix* dan *suffix* tersebut memenuhi nilai *minimum support*akan dijadikan *frequent itemset*. Keluaran dari proses ini adalah *frequent itemset*.



**Gambar 4.10 Pembentukan *rule***

Tahap perhitungan *confidence* dijelaskan pada Gambar 4.10 dengan keluarannya berupa *rule* dan nilai *confidence*. Nilai *confidence* didapatkan dari nilai *support* (A,B) dibagi nilai *support* A. Jika nilai *confidence* yang dimiliki oleh *frequent itemset* tersebut memenuhi *minimum confidence* maka akan dibangkitkan menjadi *rule*. Hasil keluaran proses ini adalah *rule* beserta nilai *confidence*-nya.



**Gambar 4.11 Perhitungan *lift ratio***

Pada Gambar 4.11 merupakan proses perhitungan *lift ratio*. Dari masukan berupa data *rules, confidence,* dan *benchmark confidence* makan akan dilakukan perhitungan *lift ratio* dengan caara membagi *confidence* dengan *benchmark confidence*. Jika hasilnya lebih dari sama dengan 1 maka *rule* dan nilai *lift ratio* tersebut disimpan karena dianggap dapat memberikan manfaat.

## Penghitungan Manual

Pada perhitungan manual ini, digunakan nilai minimum support sebesar 0,2 (20%) dan jumlah data sebanyak 10 buah data transaksi dari komentar tempat wisata Alun-Alun Kota Malan. Langkah-langkah pembentukan *FP-growth* ialah sebagai berikut:

**Langkah 1: Text Processing Data**

**Tabel 4.1 Data Komentar**

|  |  |  |
| --- | --- | --- |
| 1523267074-1351 | https://www.tripadvisor.co.id/Attraction\_Review-g297710-d8555945-Reviews-Malang\_City\_Square-Malang\_East\_Java\_Java.html | Kalau ini jadi Alun Alun malang ya cocok dan cukup. Kalau di jadikan icon masih belum cocok. Alun Alun malang ini skrng jadi bersih dan tamannya ok banget. Dibanding tahun 2000 jauh lah dan lebih bersih. |
| 1523267074-1311 | https://www.tripadvisor.co.id/Attraction\_Review-g297710-d8555945-Reviews-Malang\_City\_Square-Malang\_East\_Java\_Java.html | Kami mengujungi kota malang dan mampir le Alun alun kota malang, alun alun kota ini sangat bersih dan rapih. |
| 1523267074-1357 | https://www.tripadvisor.co.id/Attraction\_Review-g297710-d8555945-Reviews-Malang\_City\_Square-Malang\_East\_Java\_Java.html | taman yang cukup tertata. terdapat air mancur yang indah, tempat duduk, area bermain skate board. cukup nyaman untuk sekedar duduk-duduk di pagi atau sore. |
| 1523267074-1419 | https://www.tripadvisor.co.id/Attraction\_Review-g297710-d8555945-Reviews-Malang\_City\_Square-Malang\_East\_Java\_Java.html | Menjadi salah satu tujuan kami kemarin. Menyempatkan untuk santai sejenak after dr Oen. Cukup jalan kaki dan foto foto cantik. Taman tertata rapi |
| 1523267074-1413 | https://www.tripadvisor.co.id/Attraction\_Review-g297710-d8555945-Reviews-Malang\_City\_Square-Malang\_East\_Java\_Java.html | Alun-alun ini luas dan kini tertata lebih rapi, dengan air mancur yang akan menyala pada hari Minggu dan mempunyai lampu-lampu yang akan terlihat saat malam hari. Cocok juga untuk berolahraga, terdapat tempat bermain untuk anak-anak yang dijaga oleh staf, tempat bermain skateboard dan terkadang ada...Selengkapnya |
| 1523267074-1499 | https://www.tripadvisor.co.id/Attraction\_Review-g297710-d8555945-Reviews-Malang\_City\_Square-Malang\_East\_Java\_Java.html | Alun-alun kota malang semakin cantik *set*elah direnovasi, tersedia bangku-bangku untuk menikmati suasansa kota malang dibawa h pohon tua yang rindang. Tempat terbuka yang baik untuk bermain. |
| 1523267074-1487 | https://www.tripadvisor.co.id/Attraction\_Review-g297710-d8555945-Reviews-Malang\_City\_Square-Malang\_East\_Java\_Java.html | Bisa jadi inspirasi untuk kota kota lain di Indonesia untuk sekelas alun alun ini sgt bersih, tidak ada gembel, pengamen atau preman! taman bersih seger dan nyaman. Bagus! |
| 1523267074-1529 | https://www.tripadvisor.co.id/Attraction\_Review-g297710-d8555945-Reviews-Malang\_City\_Square-Malang\_East\_Java\_Java.html | Alun alun kota malang baru saja sekitar 2 bulan direnovasi, dengan wajah baru, dimana terlihat lebih terbuka, dengan lampu yang terang pada malam hari dan ada kandang binatang antara lain kelinci, serta track jalan sepeda disekeliling alun alun, nampak lebih menyenangkan dibanding wajah lamanya, cukup...Selengkapnya |
| 1523267074-1336 | https://www.tripadvisor.co.id/Attraction\_Review-g297710-d8555945-Reviews-Malang\_City\_Square-Malang\_East\_Java\_Java.html | Sangat gampang untuk menemukan area ini. Tempat yang cocock untuk wisata keluarga dengan air mancur ditengah taman. |
| 1523267074-1451 | https://www.tripadvisor.co.id/Attraction\_Review-g297710-d8555945-Reviews-Malang\_City\_Square-Malang\_East\_Java\_Java.html | Wah kalau semua alun-alun dibuat seperti ini tentu warganya akan senang karena bisa rehat sejenak dari kesibukan rutinitas kerja,bisa bersantai sejenak bagi pelancong dari luar kota seperti kami dari Purwokerto saat reuni di Batu-Malang tanggal 9 April 2016. Senang rasanya bisa berfoto didepan ikon 'Alun-alun...Selengkapnya |

Pada Tabel 4.1 terdapat 10 komentar dari Tripadvisor untuk tempat wisata alun-alun kota malang. Komentar-komentar tersebut akan dijadikan *itemset* untuk setiap *Transaction ID*. Kemudian lakukan tokenisasi dan casefolding sehingga akan seperti Tabel 4.2.

**Tabel 4.2 Data setelah tokenisasi dan *case folding***

|  |  |
| --- | --- |
| *Tid* | *Items* |
| 1 | kalau, ini, jadi, alun, alun, malang, ya, cocok, dan, cukup, kalau, di, jadikan, icon, masih, belum, cocok, alun, alun, malang, ini, skrng, jadi, bersih, dan, tamannya, ok, banget, dibanding, tahun, jauh, lah, dan, lebih, bersih |
| 2 | kami, mengujungi, kota, malang, dan, mampir, le, alun, alun, kota, malang, alun, alun, kota, ini, sangat, bersih, dan, rapih |
| 3 | taman, yang, cukup, tertata, terdapat, air, mancur, yang, indah, tempat, duduk, area, bermain, skate, board, cukup, nyaman, untuk, sekedar, duduk-duduk, di, pagi, atau, sore |
| 4 | menjadi, salah, satu, tujuan, kami, kemarin., menyempatkan, untuk, santai, sejenak, after, dr, oen., cukup, jalan, kaki, dan, foto, foto, cantik., taman, tertata, rapi |
| 5 | alun-alun, ini, luas, dan, kini, tertata, lebih, rapi, dengan, air, mancur, yang, akan, menyala, pada, hari, minggu, dan, mempunyai, lampu-lampu, yang, akan, terlihat, saat, malam, hari, cocok, juga, untuk, berolahraga, terdapat, tempat, bermain, untuk, anak-anak, yang, dijaga, oleh, staf, tempat, bermain, skateboard, dan, terkadang, ada...selengkapnya |
| 6 | alun-alun, kota, malang, semakin, cantik, *set*elah, direnovasi, tersedia, bangku-bangku, untuk, menikmati, suasansa, kota, malang, dibawah, pohon, tua, yang, rindang, tempat, terbuka, yang, baik, untuk, bermain |
| 7 | bisa, jadi, inspirasi, untuk, kota, kota, lain, di, indonesia, untuk, sekelas, alun, alun, ini, sgt, bersih,, tidak, ada, gembel,, pengamen, atau, preman, taman, bersih, seger, dan, nyaman, bagus |
| 8 | alun, alun, kota, malang, baru, saja, sekitar, bulan, direnovasi, dengan, wajah, baru, dimana, terlihat, lebih, terbuka, dengan, lampu, yang, terang, pada, malam, hari, dan, ada, kandang, binatang, antara, lain, kelinci, serta, track, jalan, sepeda, disekeliling, alun, alun, nampak, lebih, menyenangkan, dibanding, wajah, lamanya, cukup...selengkapnya |
| 9 | sangat, gampang, untuk, menemukan, area, ini, tempat, yang, cocock, untuk, wisata, keluarga, dengan, air, mancur, ditengah, taman |
| 10 | wah, kalau, semua, alun-alun, dibuat, seperti, ini, tentu, warganya, akan, senang, karena, bisa, rehat, sejenak, dari, kesibukan, rutinitas, kerja, bisa, bersantai, sejenak, bagi, pelancong, dari, luar, kota, seperti, kami, dari, purwokerto, saat, reuni, di, batu-malang, tanggal, april, senang, rasanya, bisa, berfoto, didepan, ikon, 'alun-alun...selengkapnya |

Setelah itu lakukan *filtering* dengan hanya mengambil kata-kata yang terdapat pada Tabel 2.2 sampai dengan Tabel 2.5, kemudian lakukan *stemming* yaitu tiap-tiap kata yang ada dimasukan kategori masing-masing sehingga hanya terdapat 40 kata yang ada seperti pada Tabel 2.6. Hasil dari filtering dan stemming dapat dilihat pada Tabel 4.3.

**Tabel 4.3 Tabel Transaksi**

|  |  |
| --- | --- |
| *Tid* | *Items* |
| 1 | bersih, bagus, bersih |
| 2 | bersih, rapi |
| 3 | rapi, bagus, bagus |
| 4 | bagus, rapi |
| 5 | bagus, rapi |
| 6 | bagus, sejuk, bagus |
| 7 | bersih, bersih, sejuk, bagus, bagus |
| 8 | bagus |
| 9 | mudah |
| 10 | bagus, bagus |

**Langkah 2: Penambahan kata kunci tempat wisata**

Setelah tabel transaksi terbentuk sepeti pada Tabel 4.3, tambahkan kata kunci tempat wisata yang akan diproses. Dalam kasus ini karena tempat wisata yang dipilih adalah Alun-Alun Kota Malang, maka tambahkan kata kunci ‘aakm’ di setiap transaksi. Tujuan dari penambahan kata ini adalah untuk pembentukan asosiasi. Setelah ditambahkan maka hasilnya akan seperti pada Tabel 4.4.

**Tabel 4.4 Tabel transaksi setelah penambahan kata kunci tempat wisata**

|  |  |
| --- | --- |
| *Tid* | *Items* |
| 1 | bersih, bagus, bersih, aakm |
| 2 | bersih, rapi, aakm |
| 3 | rapi, bagus, bagus, aakm |
| 4 | bagus, rapi, aakm |
| 5 | bagus, rapi, aakm |
| 6 | bagus, sejuk, bagus, aakm |
| 7 | bersih, bersih, sejuk, bagus, bagus, aakm |
| 8 | Bagus, aakm |
| 9 | Mudah, aakm |
| 10 | bagus, bagus, aakm |

**Langkah 3: Menentukan *frequent item***

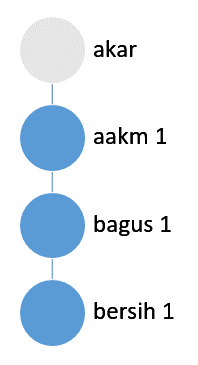
Hitung jumlah kemunculan semua kata, kemudian urutkan berdasarkan kemunculan terbanyak. Setelah dihitung maka hasilnya kata ‘aakm’ memiliki sepuluh kali kemunculan, ‘bagus’ delapan kali, ‘rapi’ empat kali, ‘bersih’ tiga kali, ‘sejuk’ dua kali, dan ‘mudah’ sekali. Setelah itu, tentukan nilai *minimal item support count*, yaitu jumlah minimal kemunculan kata untuk bisa masuk ke tabel transaksi. Pada kasus ini *minimal item support count-*nya adalah dua sehingga kata ‘mudah’ dibuang karena hanya muncul sekali, sehingga terbentuk seperti pada Tabel 4.5.

**Tabel 4.5 Hasil penentuan *frequent item***

|  |  |
| --- | --- |
| *Tid* | *Items* |
| 1 | aakm, bagus, bersih |
| 2 | aakm, rapi, bersih |
| 3 | aakm, bagus, rapi |
| 4 | aakm, bagus, rapi |
| 5 | aakm, bagus, rapi |
| 6 | aakm, bagus, sejuk |
| 7 | aakm, bagus, bersih, sejuk |
| 8 | aakm, bagus |
| 9 | aakm |
| 10 | aakm, bagus |

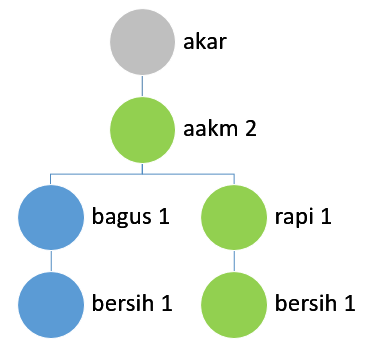
**Langkah 4: Membangun *FP tree***

Langkah selanjutnya adalah membangun pohon *FP* berdasarkan tabel transaksi Tabel 4.5. Awalnya, buat node akar, yang kemudian node berdasarkan *itemset* pertama dengan *item* pertama yaitu ‘aakm’ dengan nilai *count* sebesar satu, lalu dilanjutkan lagi node ‘bagus’ dan ‘bersih’ sehingga tampak seperti pada Gambar 4.12.



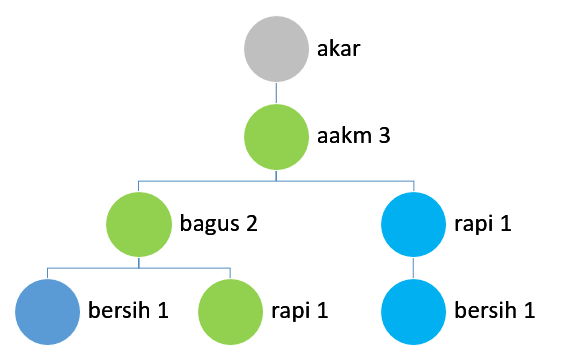
**Gambar 4.12 Pohon *TID* 1**

Pada Gambar 4.13 merupakan proses transaksi kedua {aakm, rapi, bersih}, karena node ‘aakm’ sudah terbentuk sebelumnya maka hanya perlu menambahkan *count*-nya saja sebesar satu, sehingga *count* aakm saat ini adalah dua. Karena belum ada node ‘rapi’ setelah node ‘aakm’ maka buat node baru ‘rapi’ dengan nilai *count* sebesar satu, lalu dibawahnya buat node ‘bersih’. Warna hijau menandakan node yang baru saja dilewati atau dibuat saat transaksi ini.

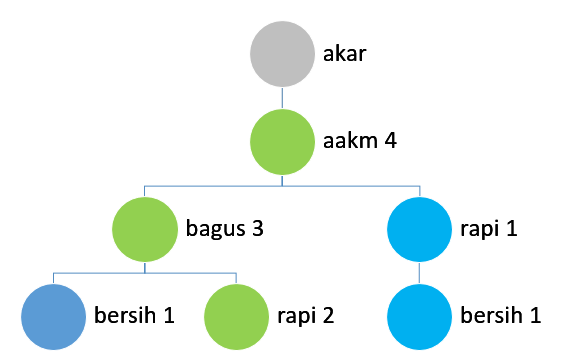


**Gambar 4.13 Pohon *TID* 2**

Selanjutnya adalah transaksi ketiga {aakm, bagus, rapi}, karena node ‘aakm’ dan ‘bagus’ sudah terbentuk maka hanya perlu menambahkan *count* masing-masing node sebesar satu. Sedangkan karena node ‘rapi’ belum ada setelah node ‘bagus’ maka buat node ‘rapi’ dengan *count* bernilai satu seperti pada Gambar 4.14.

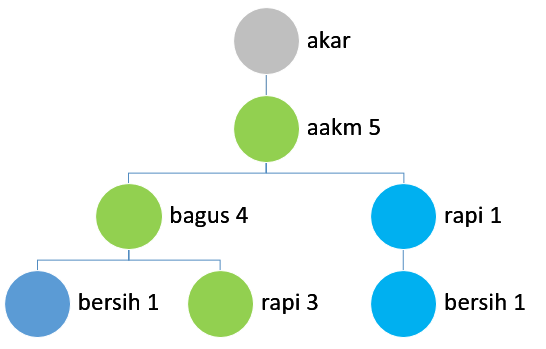


**Gambar 4.14 Pohon *TID* 3**



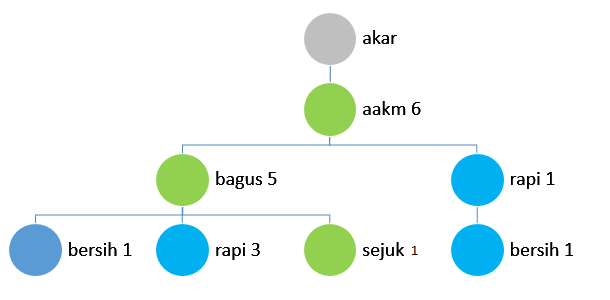
**Gambar 4.15 Pohon *TID* 4**

Pada transaksi keempat dan kelima {aakm, bagus, rapi}, karena transaksi ini sama seperti pada transaksi ketiga maka hanya perlu menambahkan masing-masing *count* sebesar satu dua kali sehingga seperti Gambar 4.15 dan 4.16.



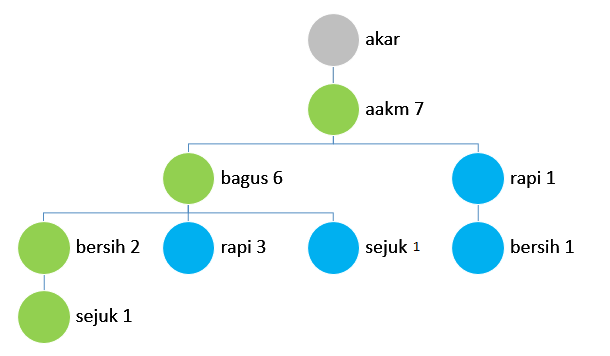
**Gambar 4.16 Pohon *TID* 5**

Gambar 4.17 adalah transaksi keenam {aakm, bagus, sejuk}, karena node ‘aakm’ dan ‘bagus’ sudah ada hanya perlu menambahkan *count*-nya sebesar satu pada masing-masing node. Kemudian buat node baru ‘sejuk’ setelah node ‘bagus’.

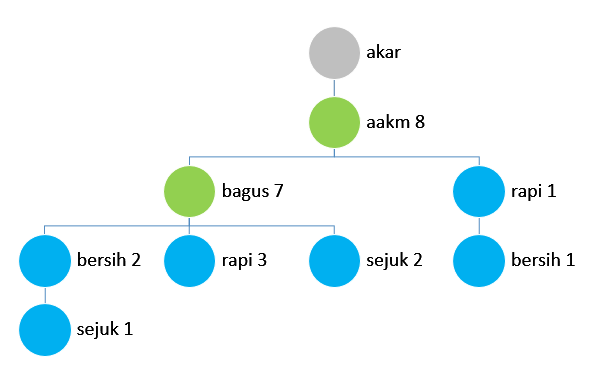


**Gambar 4.17 Pohon *TID* 6**

Selanjutnya adalah transaksi ketujuh {aakm, bagus, bersih, sejuk}, karena node ‘aakm’, ‘bagus’, dan ‘bersih’ sudah ada maka hanya perlu menambah *count* masing-masing node sebesar satu. Kemudian buat node baru ‘sejuk’ ibawah node ‘bersih’ seperti pada Gambar 4.18.

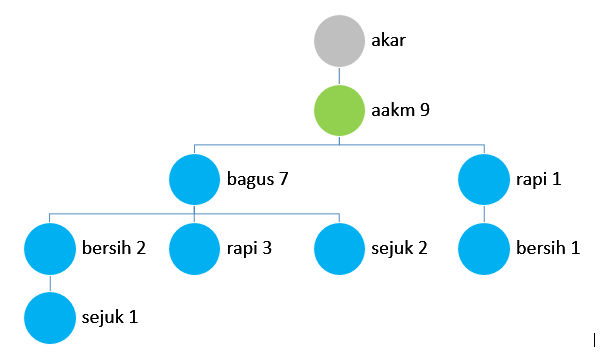


**Gambar 4.18 Pohon *TID* 7**

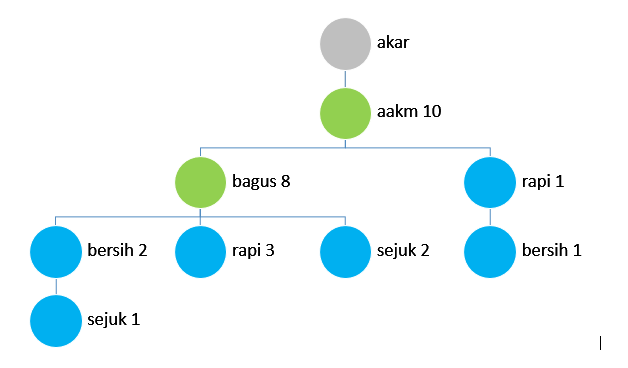


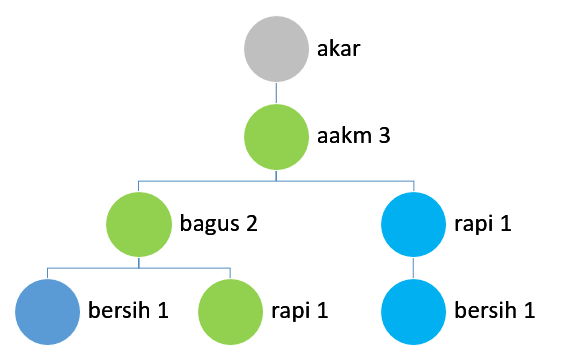
**Gambar 4.19 Pohon *TID* 8**

Gambar 4.19 merupakan transaksi kedelapan {aakm, bagus}, karena kedua node sudah ada maka hanya perlu menambahkan *count* masing-masing sebesar satu, begitu juga dengan transaksi kesembilan yang hanya ada *item* ‘aakm’ seperti pada Gambar 4.20.



**Gambar 4.20 Pohon *TID* 9**



**Gambar 4.****21 Pohon *TID* 10**

Transaksi terakhir yaitu {aakm, bagus}, karena kedua node sudah terbentuk maka hanya perlu menambahkan masing-masing *count* sebesar satu seperti pada Gambar 4.21.

**Langkah 4: Membangun *conditional pattern base***

Dengan cara menelusuri jalur yang dilalui tiap suffix, maka bisa didapatkan *conditional pattern base*-nya. Terdapat dua suffix sejuk, pertama dengan jalur mulai dari akar melalui ‘aakm’, ‘bagus’, dan ‘bersih’, lalu jalur kedua adalah ‘aakm’, dan ‘bagus’ dengan masing-masing *count* bernilai satu karena memang hanya dilalui satu kali. Untuk suffix bersih juga terdapat dua jalur yaitu ‘aakm’, dan ‘bagus’ dengan masing-masing *count* bernilai dua karena dilalui dua kali, lalu jalur ‘aakm’ dan ‘rapi’. Seterusnya hingga suffix ‘aakm’ *null* karena tidak melalui jalur apapun alias turunan langsung dari akar. *Conditional pattern base* yang terbentuk dibuatkan dalam tabel seperti pada Tabel 4.6.

**Tabel 4.6 *Conditional pattern base***

|  |  |
| --- | --- |
| Suffix | Conditional Pattern Base |
| Sejuk | {aakm:1, bagus:1, bersih:1}, {aakm:1, bagus:1} |
| Bersih | {aakm:2, bagus:2}, {aakm:1, rapi:1} |
| Rapi | {aakm:3, bagus:3}, {aakm:1} |
| Bagus | {aakm:8} |
| Aakm | null |

**Langkah 4: Menghitung *support* tiap prefix**

Hitung *support* tiap prefix dari masing-masing suffix dengan persamaan (2.1). Pada suffix ‘sejuk’ terdapat tiga prefix yaitu ‘aakm’, ‘bagus’, dan ‘bersih’. ‘aakm’ memiliki jumlah kemunculan dua kali pada suffix sejuk sehingga sesuai persamaan (2.1) *support*-nya adalah dua dibagi sepuluh, jumlah transaksi, sehingga nilainya 0,2. Nilai *support* dari masing-masing prefix ditunjukan pada Tabel 4.7.

**Tabel 4.7 Nilai *support* masing-masing prefix**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| suffix {sejuk} |  | suffix {bersih} |  | suffix {rapi} |  | Suffix {bagus} |  |
| support {aakm} | 2/10 | support {aakm} | 3/10 | support {aakm} | 4/10 | Support {aakm} | 8/10 |
| support {bagus} | 2/10 | support {bagus} | 2/10 | support {bagus} | 3/10 |  |  |
| support {bersih} | 1/10 | support {rapi} | 1/10 |  |  |  |  |

**Langkah 5: Membuat *conditional FP tree***

Setelah mendapatkan nilai *support* dari masing-masing prefix, buang prefix tersebut jika nilai *support*-nya tidak memenuhi *minimum item support* yaitu 2. Setelah itu buat tabel *conditional FP tree* seperti pada Tabel 4.8.

**Tabel 4.8 *Conditional FP tree***

|  |  |
| --- | --- |
| Suffix | Conditional FP-tree |
| Sejuk | {aakm:2, bagus:2} |
| Bersih | {aakm:3 bagus:2} |
| Rapi | {aakm:4, bagus:3} |
| Bagus | {aakm:8} |

**Langkah 6: Menghitung *lift ratio***

Hitung nilai *confidence* dengan persamaan (2.2), yaitu *support (A,B)* dibagi dengan *support* (A). Kemudian cari nilai *lift ratio* dengan persamaan (2.3) yaitu nilai dari *benchmark confidence* dibagi dengan nilai *confidence*. Hasil perhitungan terdapat pada Tabel 4.9. *Lift ratio* dengan nilai lebih dari sama dengan satu merupakan *rule* yang diambil karena dianggap bermanfaat.

**Tabel 4.9 Hasil Perhitungan *lift ratio***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. | frequent item | Support (A,B) | support (A) | confidence | benchmark confidence | lift ratio |
| 1 | aakm, sejuk | 0.2 | 1 | 0.2 | 0.2 | 1 |
| 2 | bagus, sejuk | 0.2 | 0.8 | 0.25 | 0.2 | 1.25 |
| 3 | aakm, bagus, sejuk | 0.2 | 0.8 | 0.25 | 0.2 | 1.25 |
| 4 | aakm, bersih | 0.3 | 1 | 0.3 | 0.3 | 1 |
| 5 | bagus, bersih | 0.2 | 0.8 | 0.25 | 0.3 | 0.833333333 |
| 6 | aakm, bagus, bersih | 0.2 | 0.8 | 0.25 | 0.3 | 0.833333333 |
| 7 | aakm, rapi | 0.4 | 1 | 0.4 | 0.4 | 1 |
| 8 | bagus, rapi | 0.3 | 0.8 | 0.375 | 0.4 | 0.9375 |
| 9 | aakm, bagugs, rapi | 0.3 | 0.8 | 0.375 | 0.4 | 0.9375 |
| 10 | aakm, bagus | 0.8 | 1 | 0.8 | 0.8 | 1 |

# IMPLEMENTASI

Implementasi pada pencarian asosiasi ini terdapat tiga proses utama yaitu mulai dari proses *text pre-processing*, pembentukan *tree*, kemudian perhitungan *lift ratio*. Di bawah ini akan dijelaskan implementasi dari ketiga proses tersebut untuk mendapatkan asosiasi.

## Implementasi *text pre-processing*

*Text pre-processing* terdiri dari tiga tahapan yaitu, tokenisasi, *filtering*, dan *stemming*. *Text pre-processing* ini merupakan proses awal yang dilakukan oleh sistem sebelum masuk ke tahap pembentukan *tree*.

### Tokenisasi

Tujuan dari tokenisasi adalah untuk mengubah semua huruf kapital yang ada pada dokumen menjadi huruf kecil dan kemudian memecah tiap-tiap kalimat menjadi *term* serta membuang beberapa karakter tanda baca. Proses ini dapat dilihat pada Kode Program 5.1.

|  |  |
| --- | --- |
| 1 | public String tokenizing(String dok) { |
| 2 | dok = dok.toLowerCase(); |
| 3 | dok = dok.replaceAll("[^a-zA-Z ]", ""); |
| 4 | String dok\_token = " "; |
| 5 | String term[]; |
| 6 | term = dok.split(" "); |
| 7 | for (int i = 0; i < *term*.length; i++) { |
| 8 | dok\_token = dok\_token + *term*[i] + " "; |
| 9 | } |
| 10 | return dok\_token.trim(); |
| 11 | } |

**Kode Program 5.1 Implementasi proses tokenisasi**

### *Filtering*

*Filtering* bertujuan untuk menghapus *term* yang ada di dalam daftar *stopwords* sehingga hanya bersisa *term* yang tidak ada di daftar *stopwords*. *Stopword*yang digunakan dalam proses ini adalah *stopword* Fadilah Tala. Proses ini dapat dilihat pada Kode Program 5.2 berikut.

|  |  |
| --- | --- |
| 1 | public String filtering(String dok\_token) throws IOException { |
| 2 | String dok\_filter = ""; |
| 3 | String[] listStop = new String[1000]; |
| 4 | String *term*[]; |
| 5 | try { |
| 6 | BufferedReader br = new BufferedReader(new FileReader(FilterFile)); |
| 7 | String line; |
| 8 | int baris = 0; |
| 9 | while ((line = br.readLine()) != null) { |
| 10 | String data[] = line.split("\t"); |
| 11 | listStop[baris] = data[0]; |
| 12 | baris++; } |
| 13 | } catch (Throwable t) { |
| 14 | System.out.println(t.getMessage()); } |
| 15 | *term* = dok\_token.split(" "); |
| 16 | for (int i = 0; i < *term*.length; i++) { |
| 17 | boolean cek = false; |
| 18 | for (int j = 0; j < listStop.length; j++) { |
| 19 | if (*term*[i].equals(listStop[j])) { |
| 20 | cek = true; } } |
| 21 | if (cek == false) { |
| 22 | dok\_filter = dok\_filter + *term*[i] + " ";} |
| 23 | }return dok\_filter.trim();} |
| 24 |  |

**Kode Program 5.2 Implementasi proses *filtering***

### *Stemming*

Proses *stemming* ini dilakukan menggunakan algoritma Nazif Adrian untuk mendapatkan kata dasar dari semua *term* yang dihasilkan saat proses *filtering* pada data. Proses ini dapat dilihat pada Kode Program 5.3.

|  |  |
| --- | --- |
| 1 | private String stemming(String filtering) throws IOException { |
| 2 | Set dictionary = new HashSet(); |
| 3 | InputStream in = Lemmatizer.class.getResourceAsStream("/root-words.txt"); |
| 4 | BufferedReader br = new BufferedReader(new InputStreamReader(in)); |
| 5 | String line; |
| 6 | while ((line = br.readLine()) != null) { |
| 7 | dictionary.add(line); } |
| 8 | int hitung\_dok = JmlData; |
| 9 | String[] kataUnik = new String[hitung\_dok]; |
| 10 | Lemmatizer lemmatizer = new DefaultLemmatizer(dictionary); |
| 11 | String dokstemm = ""; |
| 12 | String[] *term* = filtering.split(" "); |
| 13 | boolean cek = true; int baris = 0; |
| 14 | for (int i = 0; i < *term*.length; i++) { |
| 15 | *term*[i] = lemmatizer.lemmatize(*term*[i]); |
| 28 | dokstemm = dokstemm + *term*[i] + " "; |
| 29 | return dokstemm.trim(); } |

**Kode Program 5.3 Implementasi proses *stemming***

## Implementasi pembentukan *tree*

Pada tahap implementasi proses pembentukan *tree*, data yang sudah melewati *text processing* akan dijadikan suatu transaksi *itemset*. Setelah transaksi terbentuk barulah dibuat *FP-tree*.

### Membentuk set transaksi

Sebelum masuk ke proses pembentukan *tree*, data yang sudah melewati tahap *text processing* akan dibuat ke dalam bentuk set transaksi, baru kemudian set transaksi tersebut dijadikan masukan untuk pembentukan *tree*. Kode Program 5.4 menunjukan tahap pembentukan set transaksi.

|  |  |
| --- | --- |
| 1 | public class main { |
| 2 |  |
| 3 | public static void main(String[] args) { |
| 4 |  |
| 5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174 | String[] arr;  ArrayList<ArrayList<String>> listOfLists = new ArrayList<ArrayList<String>>();  for (int i = 0; i < obj.size(); i++) {  arr = obj.get(i).split(" ");  ArrayList<String> tmp = new ArrayList<String>();  for (int j = 0; j < arr.length; j++) {  tmp.add(arr[j]);  }  listOfLists.add(tmp);  }  System.out.println(listOfLists);  //------------------------------------------------------------------------------------------------  ArrayList<String> frq = new ArrayList<String>();  ArrayList<Integer> cnt = new ArrayList<Integer>();  // System.out.println("size: "+listOfLists.get(0).size());  for (int i = 0; i < listOfLists.size(); i++) {  for (int j = 0; j < listOfLists.get(i).size(); j++) {  if (frq.size() == 0) {  frq.add(listOfLists.get(i).get(j));  cnt.add(1);  } else {  boolean podo = false;  for (int k = 0; k < frq.size(); k++) {  if (frq.get(k).equals(listOfLists.get(i).get(j))) {  //count+1  // cnt.set(k, 9);  cnt.set(k, cnt.get(k) + 1);  podo = true;  break;  }  }  if (podo == false) {  frq.add(listOfLists.get(i).get(j));  cnt.add(1);  }  }  }  }  System.out.println(frq);  System.out.println(cnt);  //------------------------------------------------------------------------------------------------  for (int i = 0; i < cnt.size() - 1; i++) {  int t = 0;  int index = 0;  for (int j = i; j < cnt.size(); j++) {  if (cnt.get(j) >= t) {  t = cnt.get(j);  index = j;  }  }  //swap  int x = cnt.get(i);  cnt.set(i, t);  cnt.set(index, x);  //swap string  String y = frq.get(i);  frq.set(i, frq.get(index));  frq.set(index, y);  }  System.out.println(cnt);  System.out.println(frq);  //------------------------------------------------------------------------------------------------  int minsupp = 2;  for (int i = 0; i < cnt.size(); i++) {  if (cnt.get(i) < minsupp) {  String a = frq.get(i);  cnt.remove(i);  frq.remove(i);  for (int j = 0; j < listOfLists.size(); j++) {  for (int k = 0; k < listOfLists.get(j).size(); k++) {  if (listOfLists.get(j).get(k).equalsIgnoreCase(a)) {  listOfLists.get(j).remove(k);  }  }  }  }  }  System.out.println(cnt);  System.out.println(frq);  //------------------------------------------------------------------------------------------------  System.out.println(listOfLists);  System.out.println("");  // System.out.println(listOfLists.get(6));  for (int h = 0; h < listOfLists.size(); h++) {  for (int i = 0; i < listOfLists.get(h).size() - 1; i++) {  int indexcount = 0;  int count = 0;  int index = 0;  String a = null;  for (int j = i; j < listOfLists.get(h).size(); j++) {  // System.out.println(listOfLists.get(0).get(j));  for (int k = 0; k < frq.size(); k++) {  // System.out.println(k);  if (listOfLists.get(h).get(j).equalsIgnoreCase(frq.get(k))) {  // System.out.println(listOfLists.get(0).get(j));  indexcount = k;  break;  }  }  if (cnt.get(indexcount) >= count) {  count = cnt.get(indexcount);  index = j;  a = frq.get(indexcount);  // System.out.println(frq.get(indexcount));  }  }  // System.out.println("");  String x = listOfLists.get(h).get(i);  // System.out.println("string x= " + x);  // System.out.println("index count= " + index);  // System.out.println(frq.get(indexcount));  listOfLists.get(h).set(i, a);  // System.out.println(listOfLists.get(6));  listOfLists.get(h).set(index, x);  // System.out.println(listOfLists.get(6));  }  }  // System.out.println(listOfLists.get(6));  System.out.println(listOfLists);  //------------------------------------------------------------------------------------------------  main Tree = new main();  int nilai;  // Tree.sisipDtNode(listOfLists.get(0));  } |

**Kode Program 5.4 Implementasi proses *stemming***

DAFTAR PUSTAKA

Agrawal, Rakesh; and Srikant, Ramakrishnan; Fast algorithms for mining association rules in large databases, in Bocca, Jorge B.; Jarke, Matthias; and Zaniolo, Carlo; editors, Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), Santiago, Chile, September 1994, pages 487-499

Han (2000). "Mining Frequent Patterns Without Candidate Generation". Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. SIGMOD '00: 1–12. doi:10.1145/342009.335372.

Hahsler, Michael (2005). "Introduction to arules – A computational environment for mining association rules and frequent item sets" (PDF). Journal of Statistical Software.

Piatetsky-Shapiro, Gregory (1991), Discovery, analysis, and presentation of strong rules, in Piatetsky-Shapiro, Gregory; and Frawley, William J.; eds., Knowledge Discovery in Databases, AAAI/MIT Press, Cambridge, MA.

Tufféry, Stéphane (2011); Data Mining and Statistics for Decision Making, Chichester, GB: John Wiley & Sons, translated from the French Data Mining et statistique décisionnelle (Éditions Technip, 2008)

Zaki, M. J. (2000). "Scalable algorithms for association mining". IEEE Transactions on Knowledge and Data Engineering. 12 (3): 372–390. doi:10.1109/69.846291.

https://www.cnnindonesia.com/gaya-hidup/20171018122809-307-249199/sektor-pariwisata-tunjukkan-angka-pertumbuhan-yang-baik

<http://traveling.bisnis.com/read/20161227/224/615246/pertumbuhan-ekonomi-2017-malang-pacu-pariwisata>

<https://id.foursquare.com/v/alunalun-kota-malang/4c0a6ff13c70b71346fe275b>

<https://id.foursquare.com/v/jawa-timur-park-2/4c89c457a92fa09394b88bbf?tipsSort=popular>

<https://www.tripadvisor.co.id/Attraction_Review-g297710-d4367540-Reviews-Goa_Cina_Beach-Malang_East_Java_Java.html>

https://www.tripadvisor.co.id/Attraction\_Review-g297710-d6515931-Reviews-Bromo\_Tengger\_Semeru\_National\_Park-Malang\_East\_Java\_Java.html

"Ministry of Forestry: Bromo Tengger Semeru National Park". Archived from the original on March 23, 2010. Retrieved January 3, 2010.

Global Volcanism Program: Mount Semeru". Retrieved 2017-06-12.

Surat Pernyataan Menteri Pertanian No.736/Mentan/X/1982