BAB 1

ANALISIS DAN PERANCANGAN

Pada bab ini, penulis akan menjelaskan apa saja yang dilakukan dalam pengembangan *Agglomerative Hierarchical Clustering* untuk Spark. Pengembangan dilakukan untuk mencapai tujuan yaitu mendapatkan pola dari dataset yang diolah. Pola yang ingin didapatkan meliputi perhitungan rata-rata, nilai maksimum, nilai minimum dan nilai standar deviasi dari setiap attribut yang ada pada data. Selain itu, perlu didapatkan juga jumlah anggota pada setiap *cluster* yang dihasilkan dari algoritma *Agglomerative Hierarchical Clustering*.

1.1 Analisis Perangkat Lunak

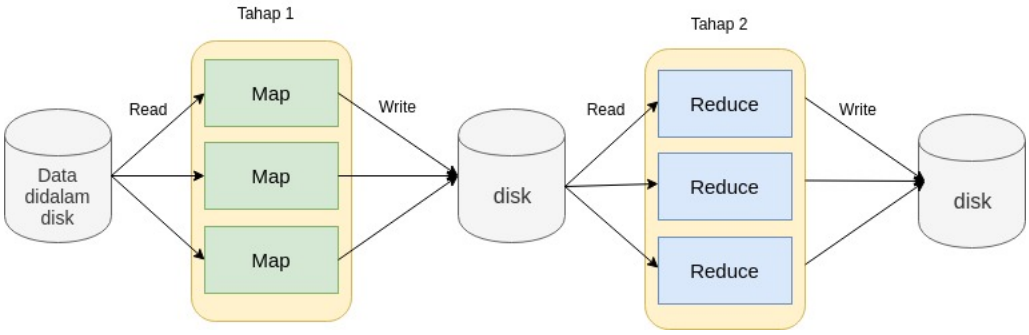
Pada bagian ini akan dijelaskan analisis algoritma *Agglomerative Hierarchical Clustering* pada MapReduce, kebutuhan perangkat lunak, dan masalah yang ingin diselesaikan.

1.1.1 Identifikasi Masalah

Dalam bidang *big data*, volume data yang sangat besar harus disimpan dalam tempat penyimpanan yang sangat besar. Volume data *big data* dapat mencapai *peta bytes*. Volume yang terlalu besar akan meningkatkan biaya dan menghabiskan tempat penyimpanan data. Volume data perlu direduksi agar menghemat tempat dan biaya.

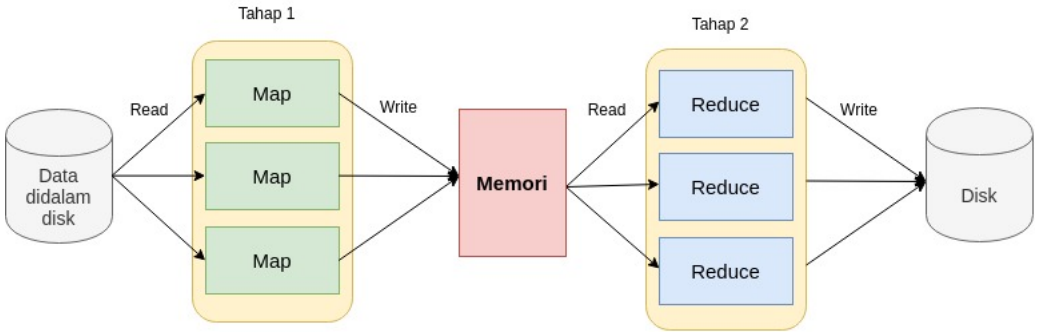
Teknologi Hadoop MapReduce dan algoritma *Agglomerative Hierarchical Clustering* dapat digabungkan sebagai solusi untuk mereduksi data. Algoritma *Agglomerative* dapat mereduksi data dengan mengambil pola-pola dari *clusters* yang dibentuk. Sistem terdistribusi Hadoop membantu dalam proses membagikan dan memecah tugas agar dapat dikerjakan secara parallel. Dengan begitu, proses reduksi data dengan algoritma *Agglomerative* akan lebih cepat.

Tetapi teknologi Hadoop masih terlalu lambat dalam mereduksi data. Hal ini disebabkan karena Hadoop banyak melakukan penulisan dan pembacaan kepada disk. Proses *disk* I/O pada Hadoop sangat tinggi dan meyebabkan algoritma *Agglomerative* berjalan sangat lambat pada Hadoop. Pada setiap tahap Hadoop akan menuliskan hasilnya kepada *disk* dan akan dibaca kembali oleh tahap selanjutnya dari *disk* seperti pada Gambar 1.1.



Gambar 1.1: Gambar penulisan kepada disk di MapReduce

Solusinya adalah menggabungkan teknologi terdistribusi lainnya dengan algoritma *Agglomerative* untuk mereduksi data. Spark, sistem terdistribusi yang menyimpan data pada memori dapat menggantikan Hadoop MapReduce. Kecepatan memori lebih cepat dibanding disk merupakan salah satu faktor mengapa Spark akan memproses data dengan kecepatan yang lebih tinggi. Pembacaan han penulisan akan dilakukan kepada memori. Gambar 1.2 adalah contoh ilustrasi tahap proses data di Spark.



Gambar 1.2: Gambar penulisan kepada memori di Spark

1.1.2 Kebutuhan Perangkat Lunak

Dalam melakukan perancang perlu diketahui dulu kebutuhan perangkat lunak. Perangkat lunak yang dirancang harus dapat menghasilkan pola yang nantinya dapat dibandingkan dengan hasil pola dari perangkat lunak MapReduce. Pola yang dibutuhkan diantaranya:

1. Jumlah objek pada *cluster*.
2. Nilai minimum setiap atribut pada *cluster*.
3. Nilai maksimum setiap atribut pada *cluster*.
4. Nilai rata-rata setiap atribut pada *cluster*.
5. Nilai standar deviasi setiap atribut pada *cluster*.

1.1.3 Analisis Agglomerative Hierarchical Clustering MapReduce

Agglomerative Hierarchical Clustering MapReduce dibagi menjadi dua bagian. Bagian pertama terkait tahap *map* dan bagian kedua terkait tahap *reduce*. Tahap *map* akan dijelaskan dengan *pseudocode* berikut ini 1

Algorithm 1: Algoritma *Mapper*

Masukan : Data mentah (**TO**)); jumlah partisi (**n**)
Keluaran : *key* = sebuah bilangan bulat ε {1 ... *n*} *value* = teks dari sekumpulan nilai atribut yang telah diproses sebelumnya

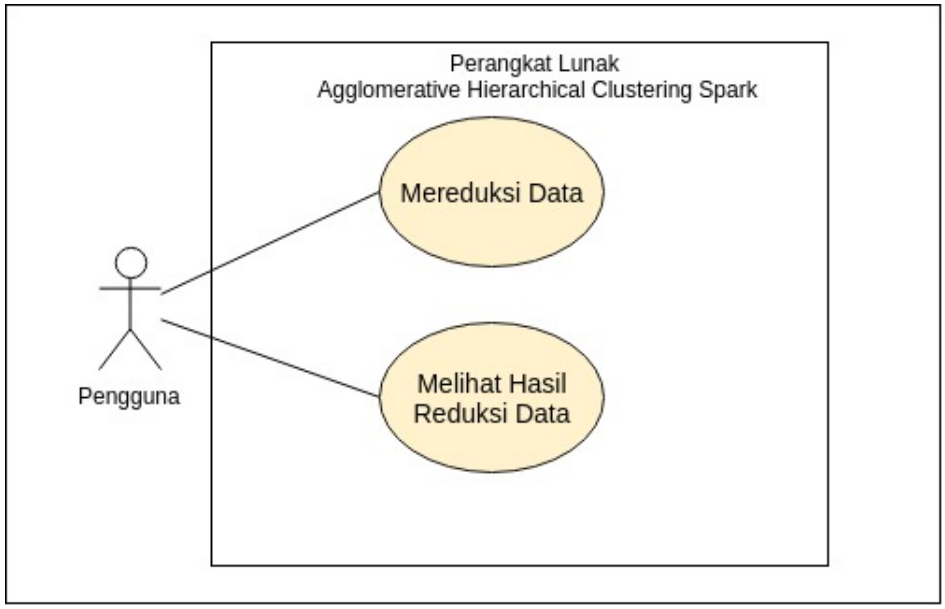
```
1 begin
2     value ← membaca baris dan memproses atributnya
3     key ← sebuah bilangan acak k, dimana 1 ≤ k ≤ n
4     mengamblikan pasangan <key, value> sebagi hasil
5 end
```

1.2 Perancangan Perangkat Lunak

Pada bagian ini, akan dijelaskan perancangan perangkat lunak. Perancangan termasuk diagram *use case*, skenario, diagram kelas, dan rancangan antarmuka.

1.2.1 Diagram Use Case dan Skenario

Diagram *use case* merupakan sebuah pemodelan untuk perilaku dari perangkat lunak yang akan dibuat. Diagram *use case* digunakan untuk mengetahui fungsi apa saja yang ada dalam perangkat lunak. Fungsi-fungsi dari perangkat lunak akan dioperasikan oleh satu pengguna. Cara kerja dan perilaku dari perangkat lunak akan dijelaskan dalam bentuk diagram *use case*. Diagram *use case* dapat dilihat pada Gambar 1.3.



Gambar 1.3: Gambar diagram *use case* perangkat lunak *Agglomerative Hierarchical Clustering*

Bedasarkan gambar diagram *use case* diatas, berikut adalah skenario yang ada:

1. Nama *use case*: Mereduksi data

    - Aktor: Pengguna

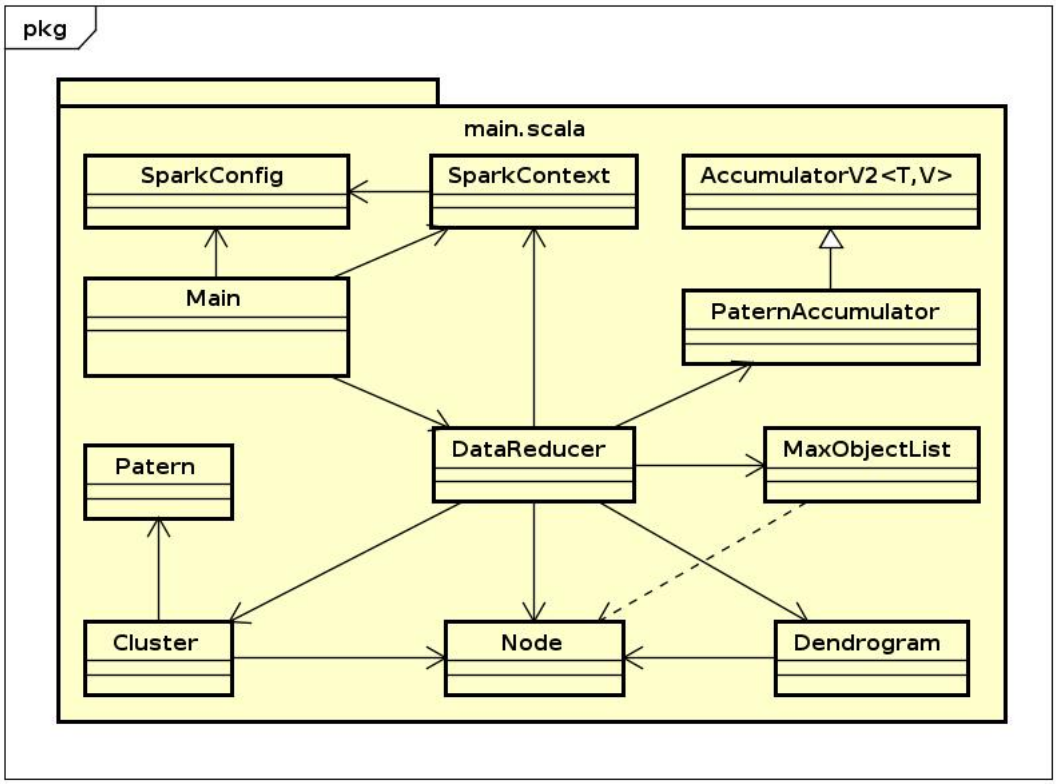
- Pre-kondisi: data yang akan diolah dimasukan kepada HDFS
- Pra -kondisi: hasil reduksi disimpan pada HDFS
- Deskripsi: Fitur untuk menjalankan program untuk mereduksi data
- Langkah-langkah:
  (a) Pengguna mengisi JAR *path*, *input path*, dan *output path*
  (b) Pengguna mengisi jumlah *executor* dan besar *executor memory*
  (c) Pengguna mengisi jumlah partisi, batas maksimum objek, tipe metode, dan *cut-off distance*
  (d) Pengguna menekan tombol *submit*
  (e) Sitem melakukan pengolahan data dengan algoritma *Agglomerative Hierarchical Clustering* pada *cluster* Hadoop
  (f) Sistem membuka tab baru untuk melihat tahap dan progres program
  (g) Sistem menyimpan hasil reduksi pada HDFS

2. Nama *use case*: Melihat data

- Aktor: Pengguna
- Pre-kondisi: data yang akan ditampilkan sudah disimpan pada HDFS
- Pra -kondisi: menampilkan data yang ada pada HDFS
- Deskripsi: fitur untuk melihat data hasil reduksi
- Langkah-langkah:
  (a) Pengguna mengisi *path* dimana data disimpan pada HDFS
  (b) Sistem menampilkan data-data pada direktori tersebut

1.2.2 Diagram Kelas

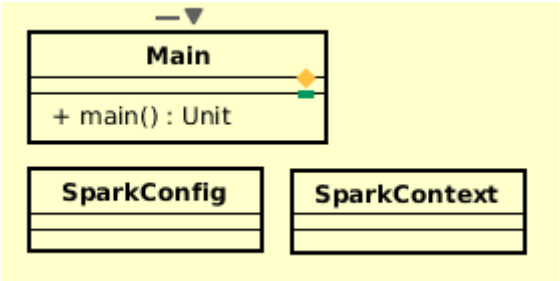
Pada bagian ini akan dijelaskan diagram kelas dari perangkat lunak. Diagram kelas dapat dilihat pada Gambar 1.4.



Gambar 1.4: Gambar diagram kelas

Bedasarkan Gambar 1.4, berikut ini adalah penjelasan kelas-kelas yang digunakan:

- **Main, Spark Config, dan Spark Context**

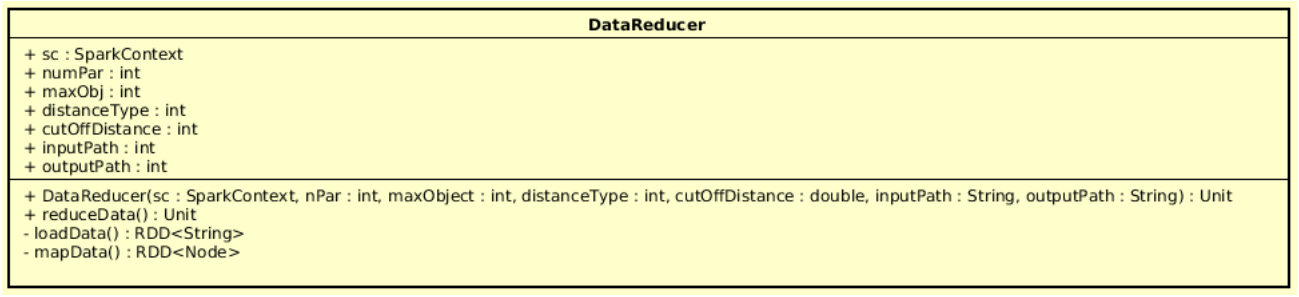


Gambar 1.5: Gambar kelas Main, SparkConfig, SparkContext

Berikut adalah penjelasan dari ketiga kelas pada Gambar 1.5:

  - Main: kelas Main memiliki *method main* yang merupakan titik masuk dari program. *Method* ini merupakan *method* pertama yang akan dieksekusi ketika program dijalankan.
  - SparkConfig: kelas SparkConfig digunakan untuk mengatur konfigurasi untuk Spark. Pengaturan nama aplikasi, jumlah *core*, besar *memory*, dan lainya dapat diatur pada kelas ini.
  - SparkContext: kelas ini merupakan titik masuk untuk layanan-layan dari Apache Spark.

- **DataReducer**

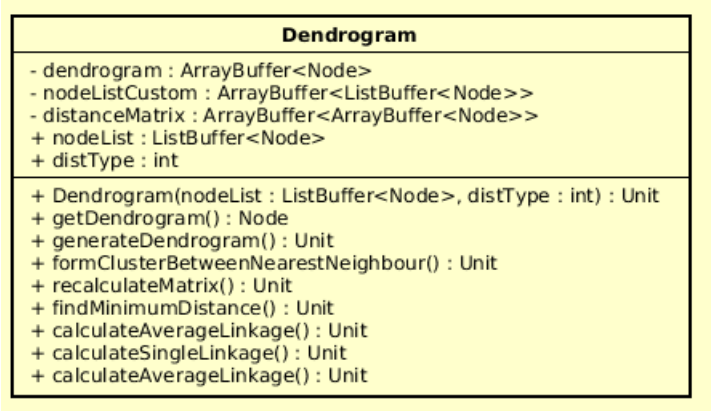


Gambar 1.6: Gambar kelas DataReducer

Kelas DataReducer dirancang untuk memproses data. Proses reduksi secara parallel dilakukan pada kelas ini. Proses pemuatan dan penyimpanan data dilakukan pada kelas ini. Bedasarkan Gambar 1.6, berikut adalah penjelasan dari *methods* pada kelas DataReducer:

1. loadData: *method* untuk memuat data bedasarkan *intput path* yang diberikan.
2. mapData: *method* untuk memisahkan atribut dan membungkus attribut dalam objek Node agar lebih mudah diproses. *Method* ini akan mengembalikan RDD bertipe Node.
3. reduceData: *method* dimana proses reduksi data secara parallel terjadi. Data akan dipecah menjadi beberapa partisi pada method ini. Partisi ini akan diproses secara parallel. Method ini juga bertangung jawab untuk meyimpan pola hasil reduksi kepada HDFS.

- **Dendrogram**



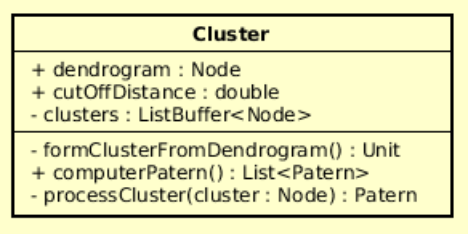
Gambar 1.7: Gambar kelas Dendrogram

Kelas Dendrogram dirancang untuk memproses data dan membangun dendrogram sesuai algoritma *Agglomerative Hierarchical Clustering*. Bedasarkan Gambar 1.7, berikut adalah penjelasan *methods* pada kelas Dendrogram:

1. getDendrogram: *method* ini mengembalikan dendrogram.
2. generateDendrogram: *Method* untuk membangun dendrogram bedasarkan algoritma *Agglomerative Hierarchical Clustering*.
3. formClusterBetweenNearestNeighbour: method untuk menggabungkan *cluster* terdekat.
4. recalculateMatrix: *method* untuk menghitung ulang matriks jarak.
5. findMinimumDistance: *method* untuk mencari jarak minimum antara dua *cluster*.

6. calculateCentroidLinkage: method untuk mencari jarak antara centorid dua cluster.
7. calculateSingleLinkage: method untuk mencari jarak minimum antara dua cluster.
8. calculateCompleteLinkage: method untuk mencari jarak maksimum antara dua cluster.
9. calculateDistance: method untuk mencari jarak antara dua buah Node bedasarkan atributnya.

- **Cluster**

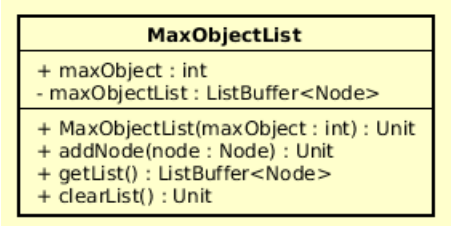


Gambar 1.8: Gambar kelas Cluster

Kelas Cluster dirancang untuk mengolah mengolah*cluster* untuk menghasilkan pola dengan memotong *cluster*. Bedasarkan Gambar 1.8, berikut adalah penjelasan *methods* pada kelas Cluster:

1. formClusterFromDendrogram: method ini bertugas untuk memotong *dendrogram* menjadi beberapa *cluster*.
2. computePatern: method untuk mengolah potongan-potongan *cluster* menjadi pola dengan memanggil *method* processCluster.
3. processCluster: method untuk memproses *cluster* dan membuat pola bedasarkan atribut-atribut pada *cluster*.

- **MaxObjectList**

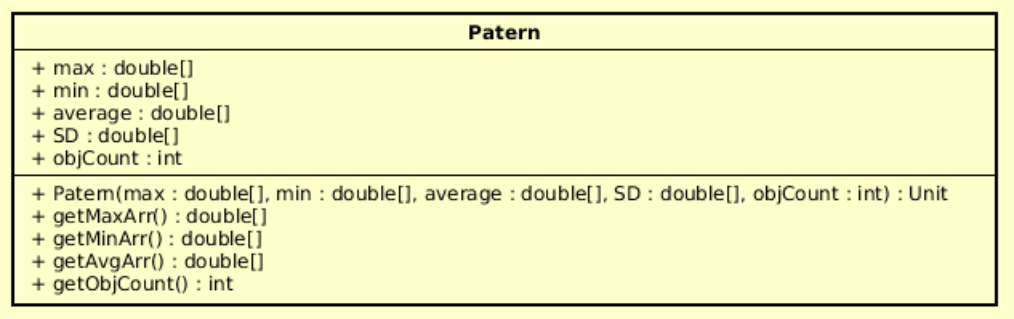


Gambar 1.9: Gambar kelas MaxObjectList

Kelas MaxObjectList dirancang untuk membatasi jumlah objek yang akan diolah pada kelas Dendrogram. Bedasarkan Gambar 1.8, berikut adalah penjelasan *methods* pada kelas MaxObjectList:

1. addNode: method untuk menambahkan Node pada pada *list*
2. getList: method ini mengembalikan *list* berisi objek Node.
3. clearList: method untuk mengosongkan *list*.

- **Patern**

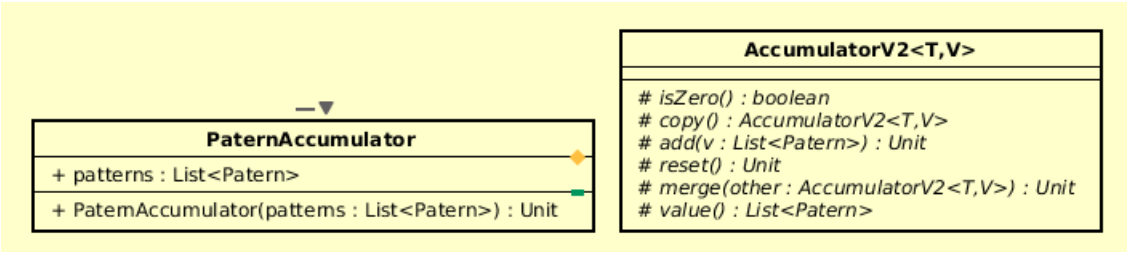


Gambar 1.10: Gambar kelas Patern

Kelas Patern dirancang untuk merepresentasikan pola pada *cluster*. Bedasarkan Gambar 1.8, berikut adalah penjelasan *methods* pada kelas MaxObjectList:

1. getMaxArr: *method* ini mengembalikan *array* berisi nilai maksimum dari setiap atribut.
2. getMinArr: *method* ini mengembalikan *array* berisi nilai minimum dari setiap atribut.
3. getAvgArr: *method* ini mengembalikan *array* berisi nilai rata-rata dari setiap atribut.
4. getSDArr: *method* ini mengembalikan *array* berisi nilai standar deviasi dari setiap atribut.
5. getObjCount: *method* ini mengembalikan jumlah objek.

- **PaternAccumulator dan AccumulatorV2**

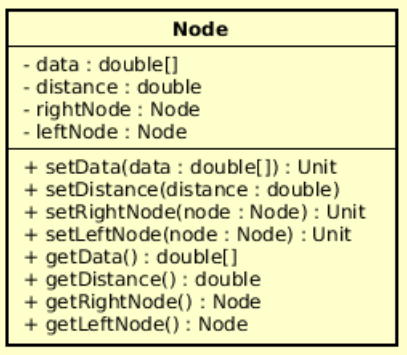


Gambar 1.11: Gambar kelas PaternAccumulator dan AccumulatorV2

Kelas PaternAccumulator dirancang untuk mengumpulkan pola-pola hasil reduksi. Kelas ini merupakan anak dari kelas abstrak *AccumulatorV2* yang setiap *method* harus di-*override* pada kelas anaknya. Bedasarkan Gambar 1.11, berikut adalah penjelasan *methods* pada kelas PaternAccumulator:

1. isZero: *method* untuk mengetahui apakah list *masih* kosong atau tidak.
2. copy: *method* untuk menduplikat objek PaternAccumulator.
3. add: *method* untuk menambahkan *list* berisi pola-pola.
4. merge: *method* untuk menggabungkan dua objek PaternAcumulator menjadi satu.

- **Node**



Gambar 1.12: Gambar kelas Node

Kelas Node digunakan untuk membentuk pohon yang merepresentasikan *dendrogram*. Selain itu kelas ini digunakan untuk merepresentasikan anggota pada *cluster*. Bedasarkan Gambar 1.12, berikut adalah penjelasan *methods* pada kelas Node:

1. setData: *method* untuk memasukan nilai-nilai atribut.
2. setDistance: *method* untuk megubah nilai jarak.
3. setRightNode: *method* untuk menambahkan anak kanan Node.
4. setLeftNode: *method* untuk menambahkan anak kiri Node.
5. getData: *method* ini mengembalikan nilai-nilai atribut.
6. getDistance: *method* ini mengembalikan jarak.
7. getRightNode: *method* ini mengebalikan anak belah kanan dari Node.
8. getLeftNode: *method* ini mengebalikan anak belah kiri dari Node.

1.2.3 Rancangan Antarmuka

DAFTAR REFERENSI

LAMPIRAN A

KODE PROGRAM

Listing A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

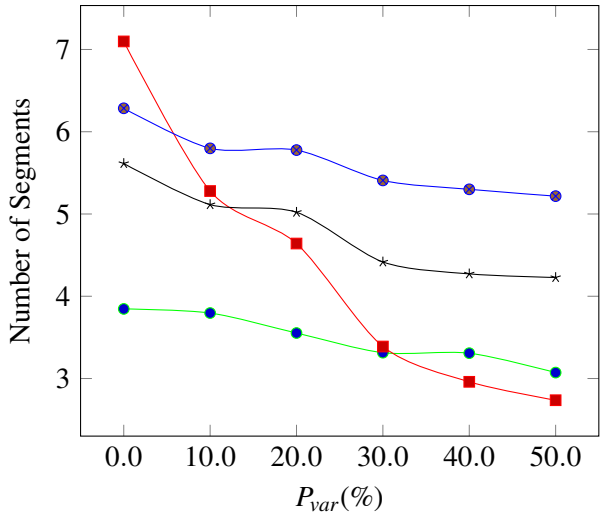
Listing A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

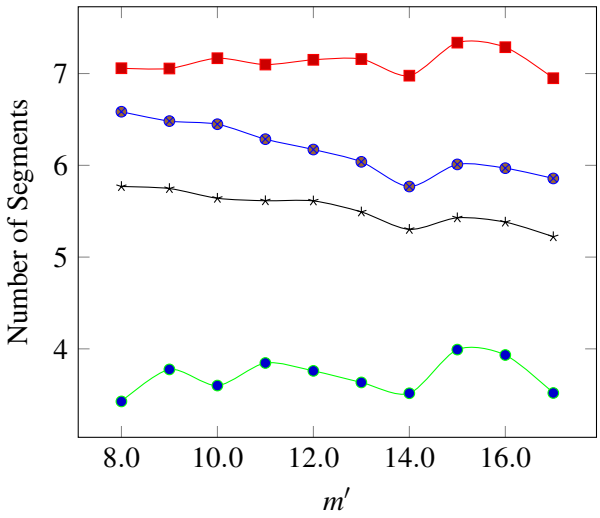
LAMPIRAN B

HASIL EKSPERIMEN

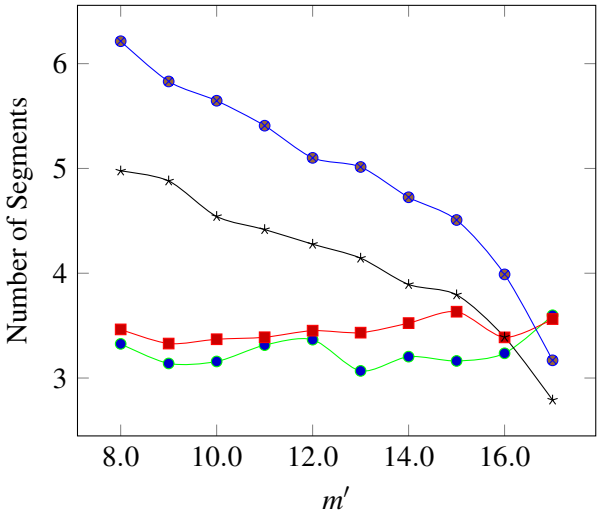
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



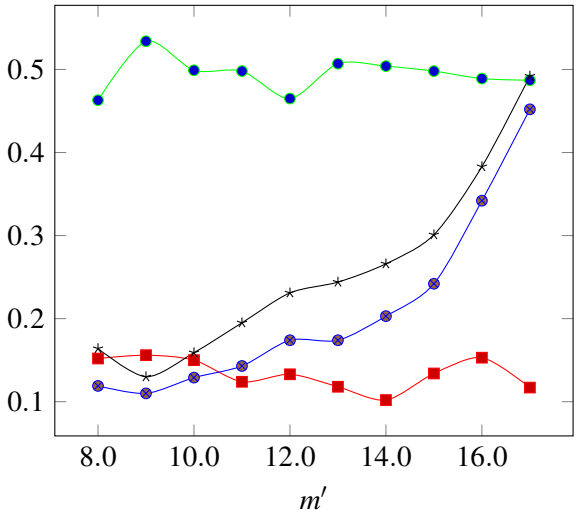
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4