**Genetic Algorithm using MapReduce**

The genetic algorithm program was designed to generate an expression which will represent a target number. Each digit and operator in the expression is represented in 4-bits and the series of bits form a chromosome. The program randomly generates an expression and performs the genetic transformations i.e. selection, crossover and mutation, to produce a chromosome with good fitness score. The solution chromosome is a valid chromosome where number is followed by an operator and the fitness score is 1, i.e. it the equivalent to the target number.

Modifying the various factors of the reproduction phase of genetic algorithm, I analyzed –

1. Higher the size of population, less Generations required to find the solution chromosome.
2. Longer the Chromosome length, more generations
3. Lower mutation rate, fitness Score closer to 1
4. 100% Crossover rate, performance of genetic algorithm reduces

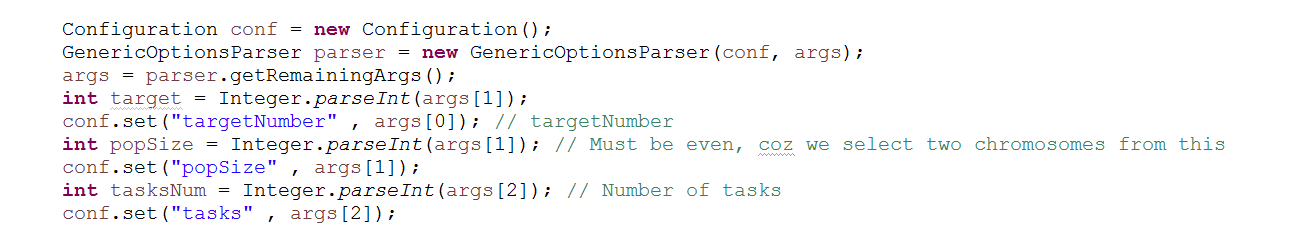
To design a parallelized approach for this problem via Hadoop, I have analyzed that I would need to split the reproduction phase of the algorithm in such a way that the reproduction steps are performed on the local data. After studying the references [1] and [2], I have decided that I would been implementing a compact genetic algorithm design based on Coarse-grained Parallelization Model or Island Model.

In this model, the whole population is divided into islands in the Mapper Phase depending on the number of reducers running and the Reducer performs the selection of two chromosomes, crossover between them, mutation of the genes and then identify the solution chromosome. In my code, the solution chromosome will be emitted in a solutionChromosome.txt. Multiple chains of map reduce job will be generated to achieve this goal.

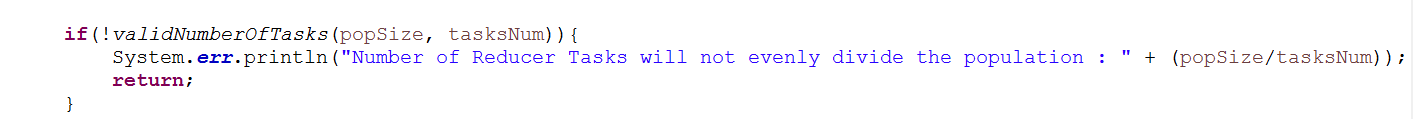
The flow my MapReduce Algorithm is –

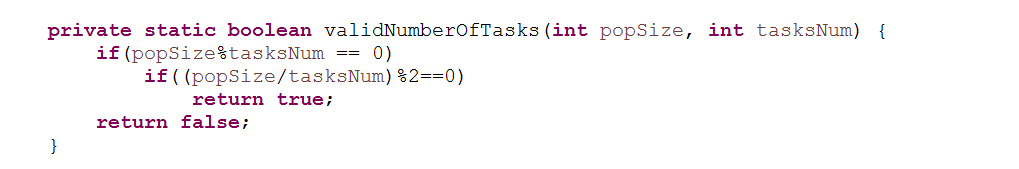
**GaDriver.java**

1. Read the input arguments – TargetNumber, NumOfReducers, PopulationSize



1. Since the population will be divided among the reducers must be even, validate the input arguments such that the division of population among reducers is satisfactory.





1. If valid, generate the initial pool of chromosomes equal to the population size. The file will have “ChromoNumber,binaryChromoString,fitnessScore”



1. Check if whether solutionChromosome.txt file exists or not. If not, continue with the generation steps otherwise stop the chaining of the map reduce jobs.



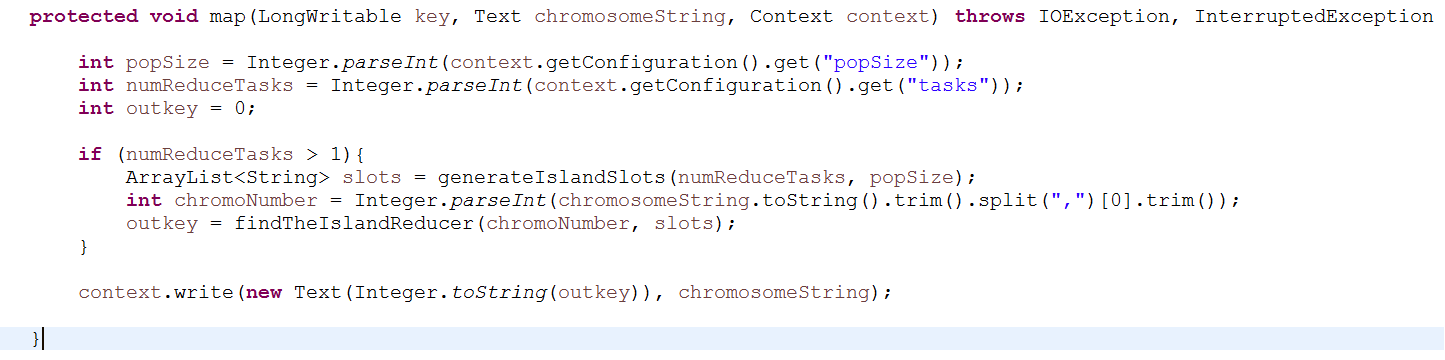
**GaMapper.java**

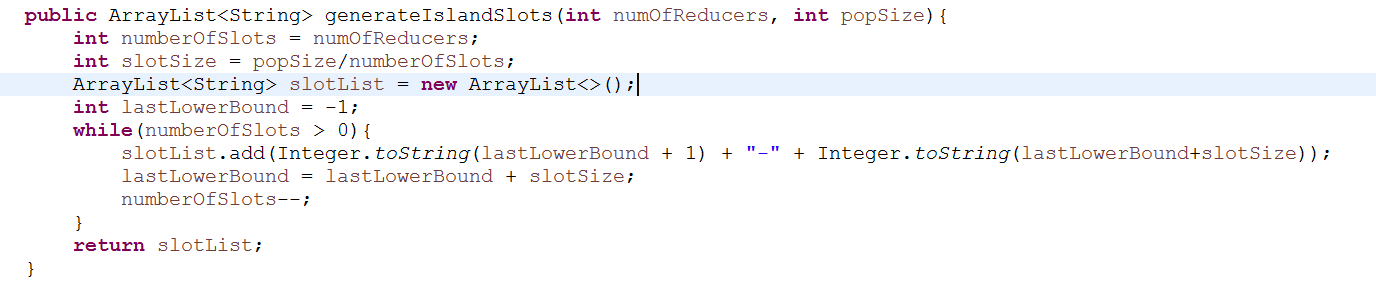
In the Mapper function, I will decide the island for the chromosome. Chromosome belong to same island will have same key so that they can be processed by one reducer.

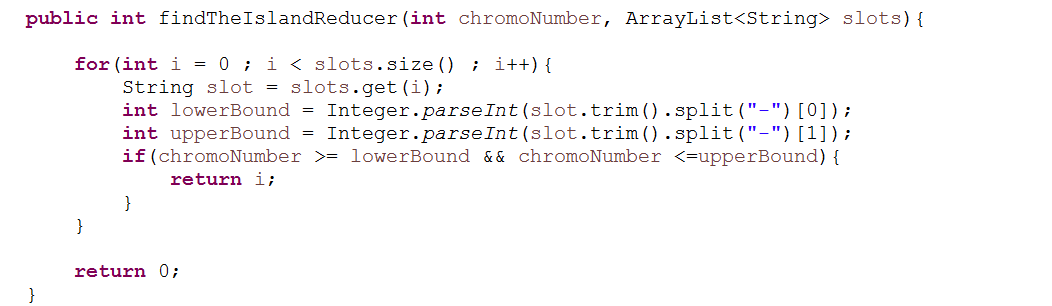
1. If the number of reduces tasks are greater than 1, then we will need to decide the islands. Otherwise it will always to the same single reducer.
2. To decide an island, I use the chromosome number to decide the island it belong to it.

For example – if we have 100 population size and 5 reduces then each island will be of size 20.

The island slots generated for this will be (0-19,20-39,40-59,60-79,80-99) Consider that we got chromosome number 61, then we would need to put in on island 3 since it will be processed by reducer number 3.







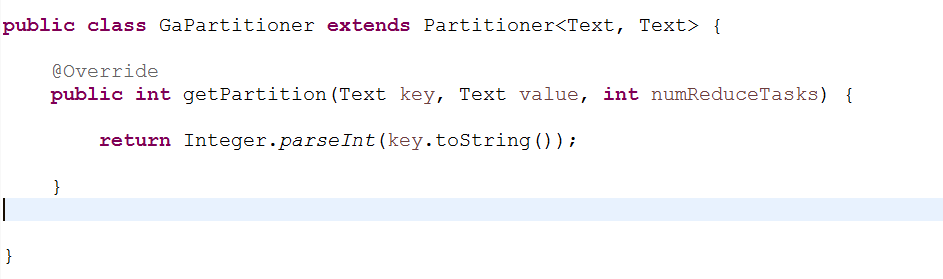
1. The output of this Mapper will be

Key -> Reducer Number

Value -> “chromoNumber,binaryChromoString,fitnessScore”

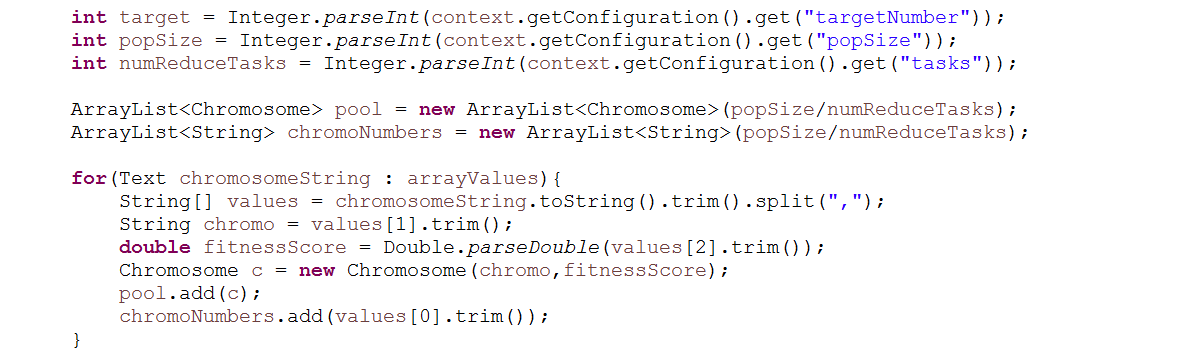
**GaPartitioner.java**

By default, Hadoop uses the value of (HashKey%numberOfReducers) to decide the reducer that will work on that set of values associated with the key. I have override written a custom Partitioner that will return the key number the reducer number, since I am setting the key number in the Mapper. Therefore, I am ensuring that the Reducer-0 processes all the chromosome associated with the Island-0 and nothing else.

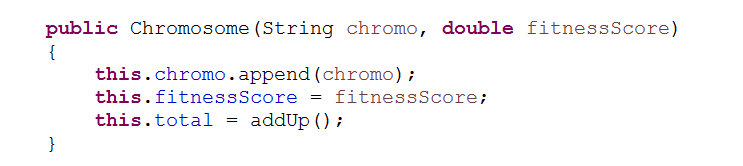


**GaReducer.java**

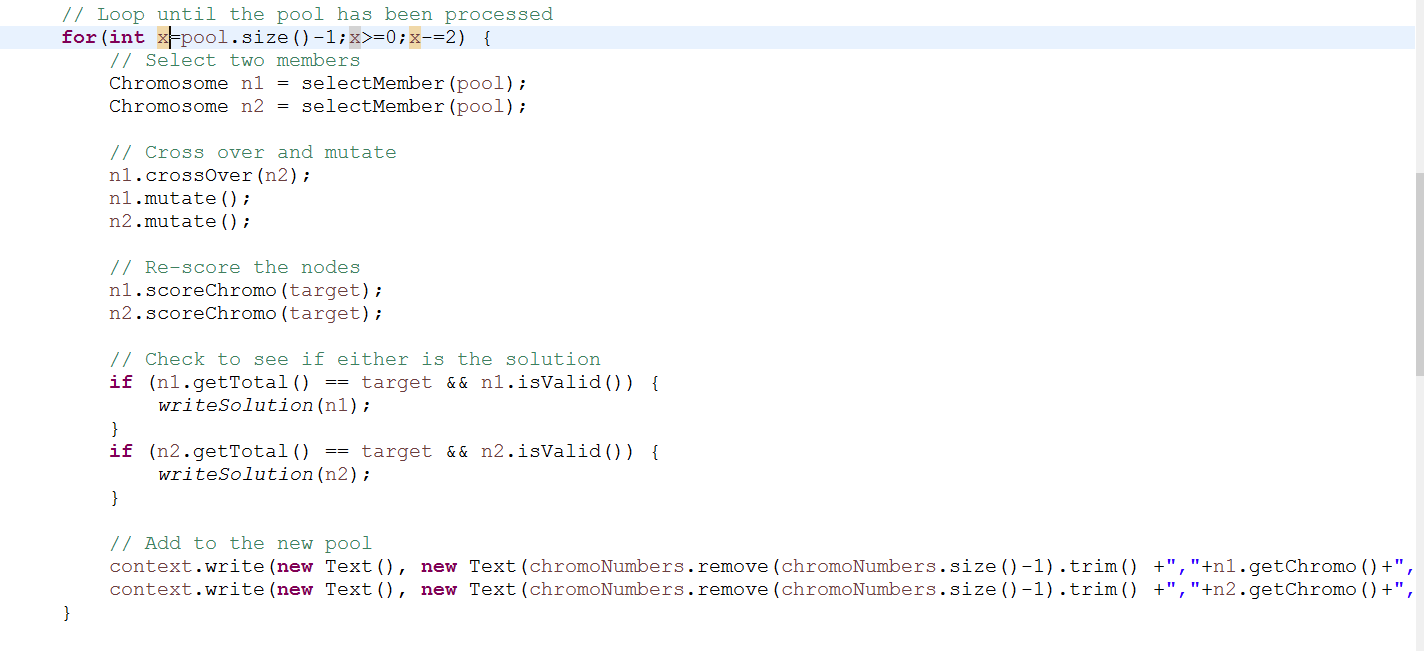
1. In the reduce, I convert the chromosome string into an actual Chromosome Object and add it into the pool of chromosome which will be later used. Also, I have added the chromosome number in a different list as we need to emit the value from the reducer in the same format as the mapper generates it.



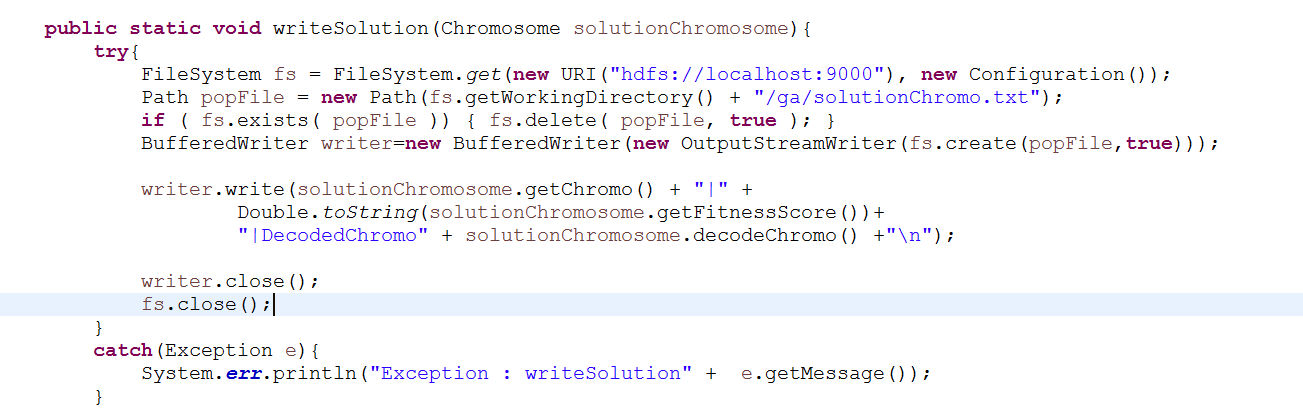
The chromoString is like - 59,0100110100110001101011011101110001000011,0.010526315789473684 So I split it on “,” and get the first part as chromoNumber, second as binaryString and third as fitnessScore. Also, I have added a new constructor in the Chromosome.java class with return a chromosome object when I give it second and third part.



1. The Reducer then perform the steps of genetic algorithm on this pool of chromosomes. It randomly select two chromosomes from this pool. Crossover the bits in the selected chromosome. Mutation the bits in the child chromosomes and then validate if the new child score is valid and its total score is equal to target number.

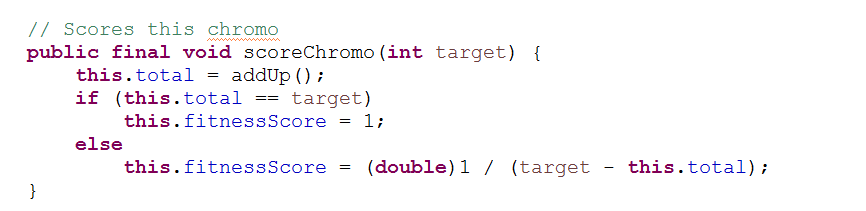


1. If the child chromosome is the solution chromosome, it writes the solution to solutionChromosome.txt. Otherwise, the reducer emits this new child chromosome as the output.



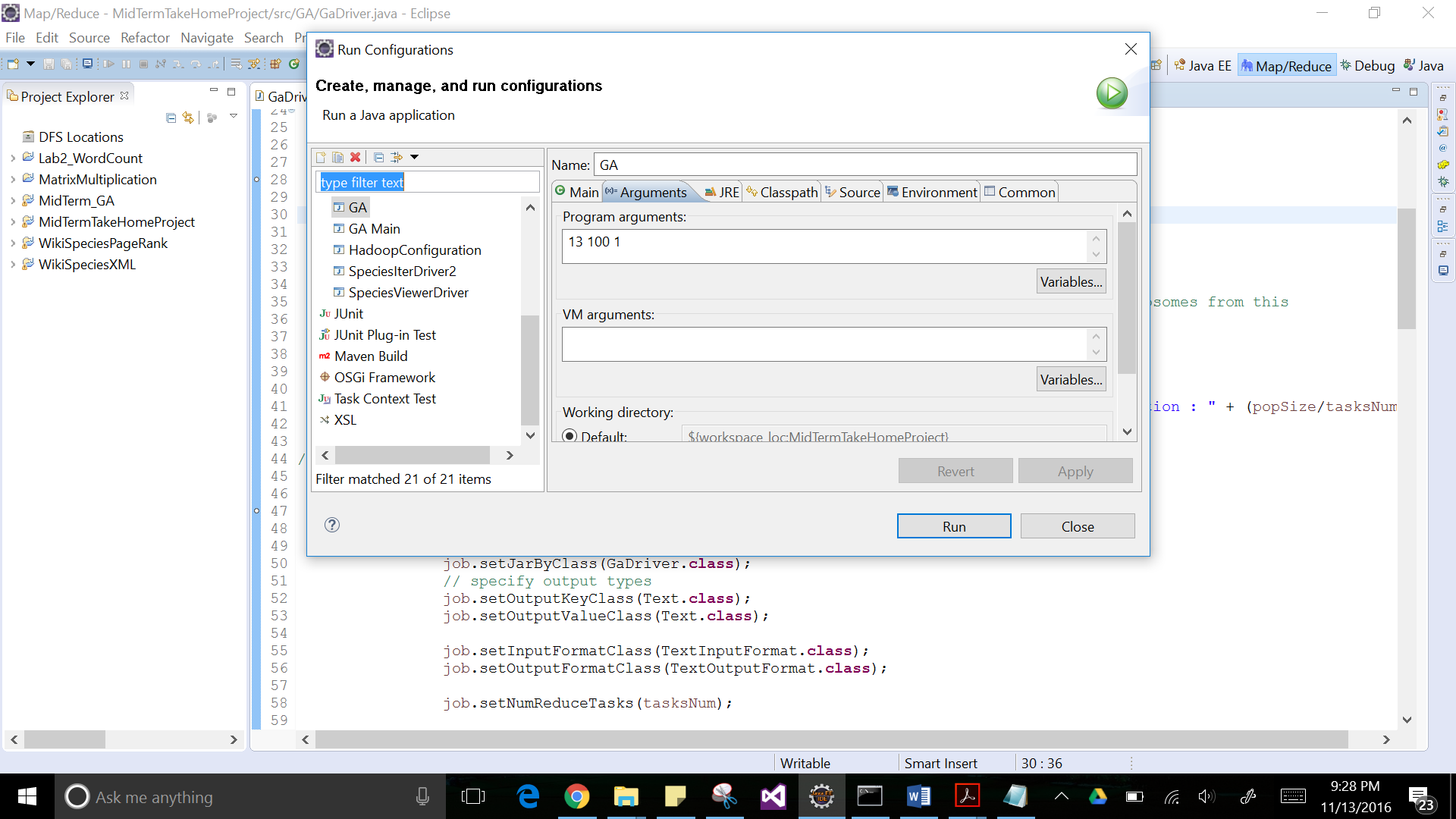
1. The output of the reducer servers as new pool of chromosomes and is the input to the next Mapper cycle.

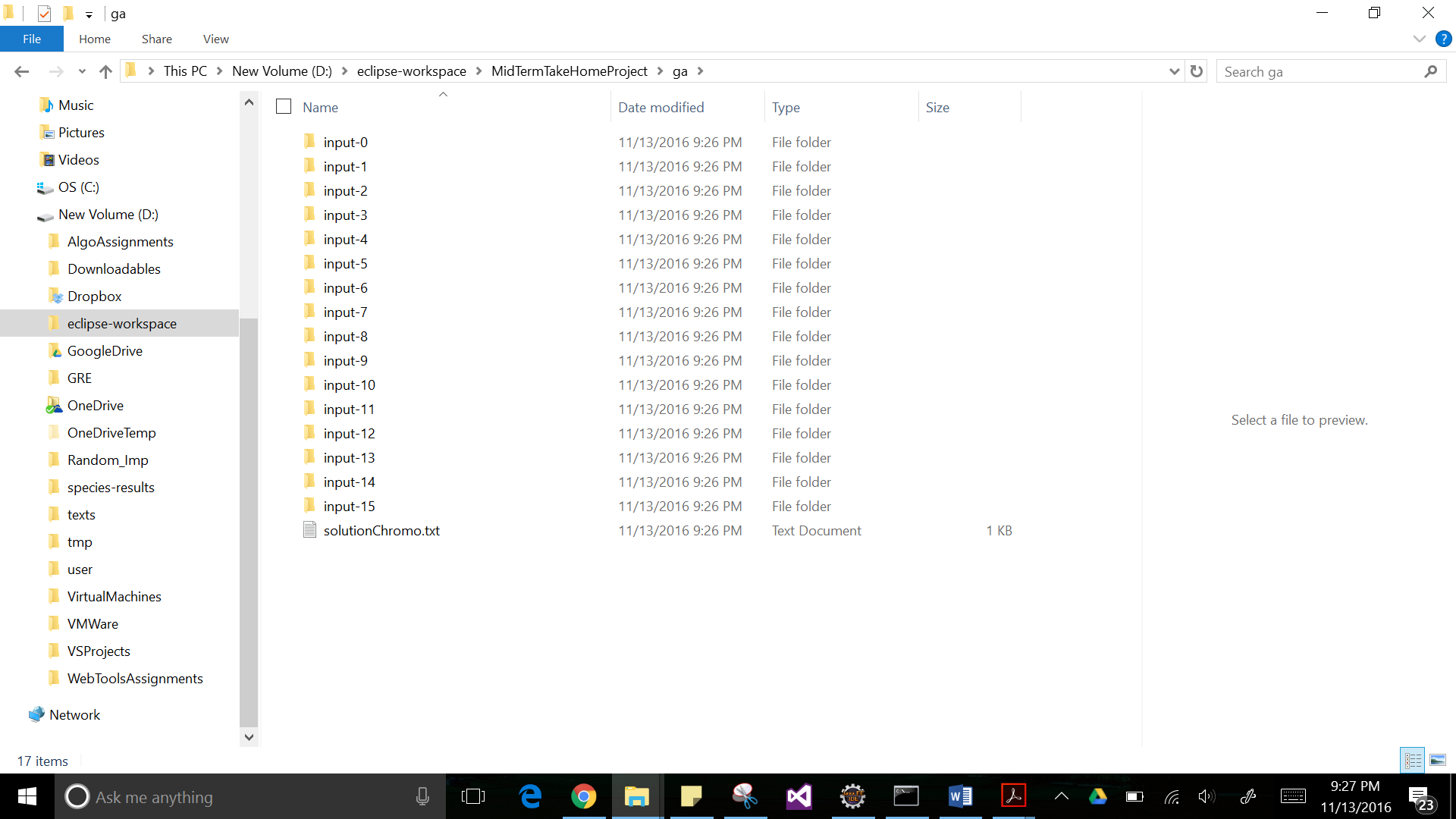
**Chromosome.java**

I have made a change in the scoreChromo function of this class. Since the fitness score the inverse of the difference between the target number and chromosome total, when the difference is 0 I have normalized it to fitness score of 1.

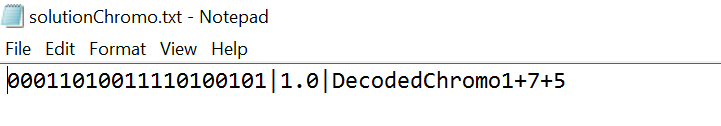
**StandAlone Mode –**

In Stand Alone Mode, the number of reducers are always 1. For the following arguments –

TargetNumber=13, PopulationSize=100

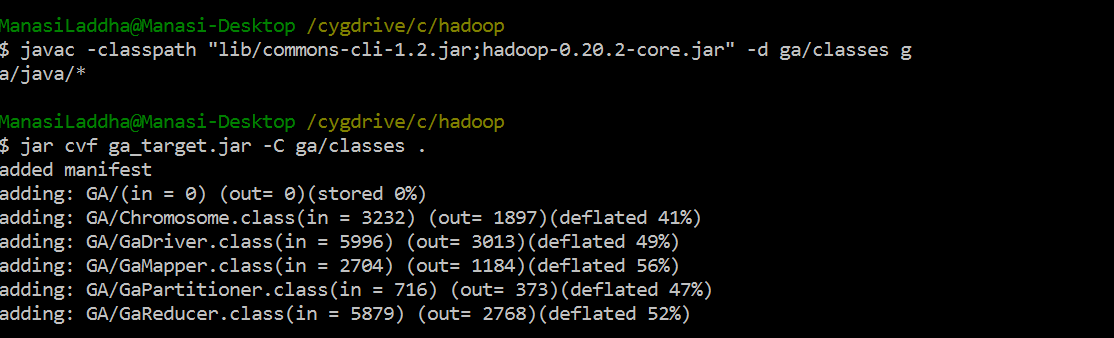
The total generations were 15 to find solution

Chromosome



**PsuedoMode –**

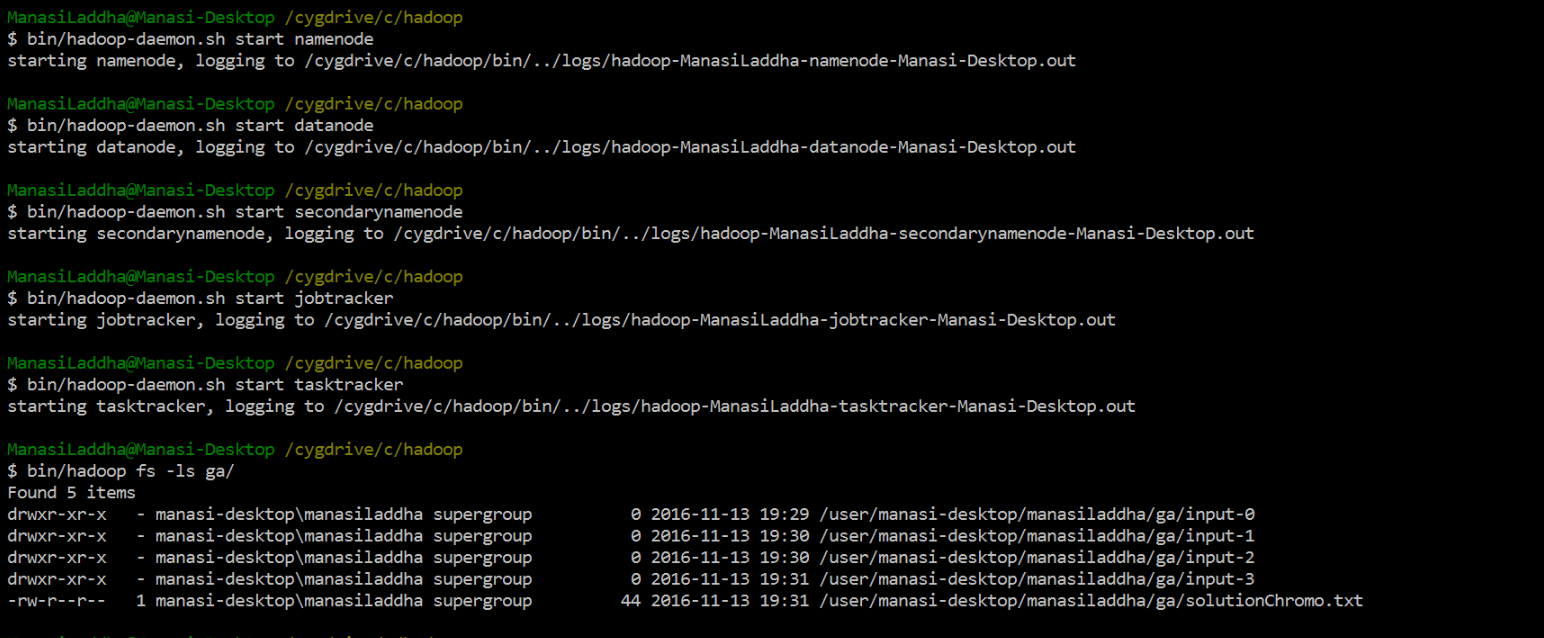
To actually parallelize my code, I decided to run my code in Psuedo Mode. I have first complied my java files and generated the jar.



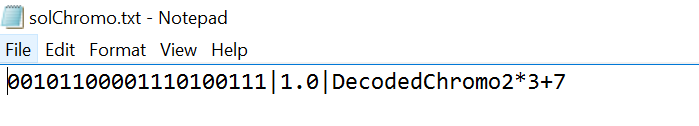
I ran the command with target number is 13 and the population size 500 so each island will have 100 chromosomes.



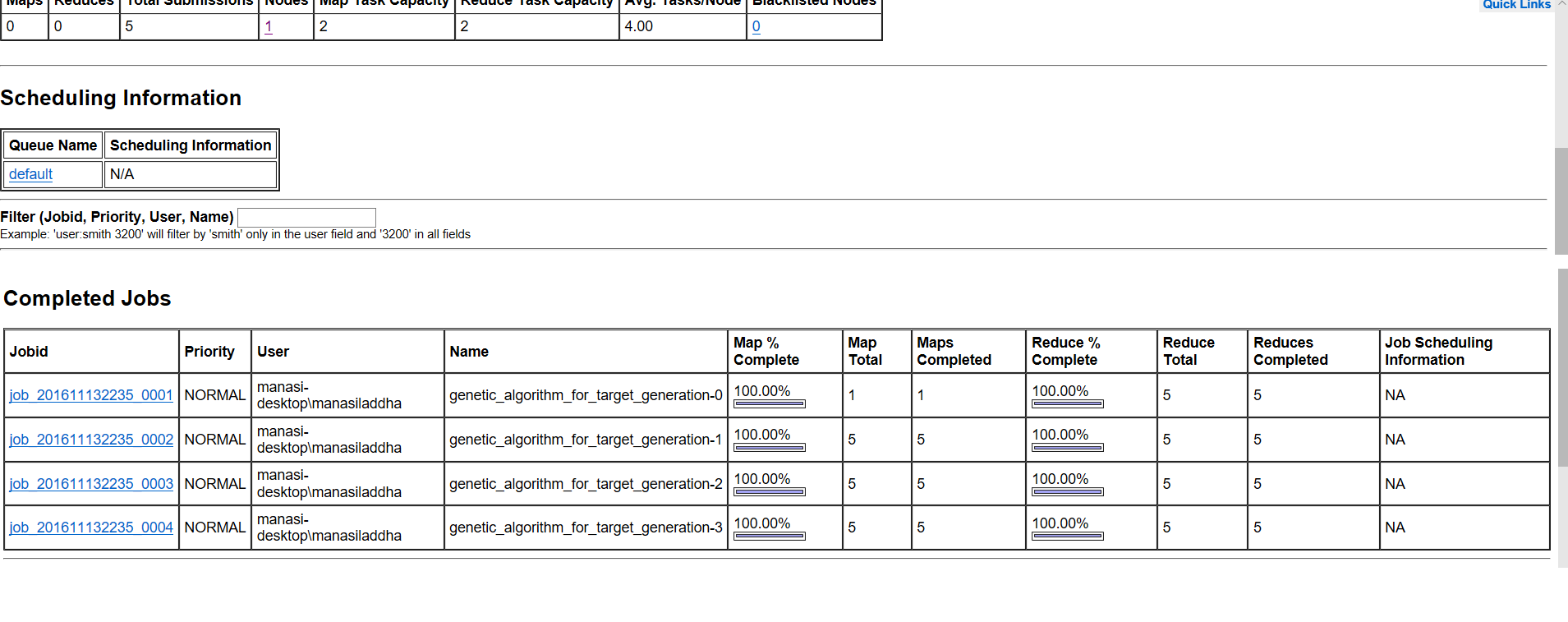
It took me 3 iterations to find the solution chromosome –



The solutionChromosome.txt



The JobTracker Sand –



**References**

[1]Scaling Genetic Algorithms using MapReduce - Abhishek Verma, XavierLlor'a, David E. Goldberg, Roy H. Campbell

[2] A Framework for Genetic Algorithms Based on Hadoop - Filomena Ferrucci, M-Tahar Kechadi, Pasquale Salza, Federica Sarro