README FILE P1_4035_802142644_172

Kelvin Garcia Muñiz

CIIC4020 - 030

Contents

Project-1-Data-Structures

To Run the Program:

Before executing any of these commands from the CMD or Terminal make sure you are in the correct directory path. If using Eclipse or a similar IDE, these procedures may not be required.

To Run Part 1:

From terminal: java -classpath ./bin p1MainClasses/Part1Main n

```
Where:

  n = 1
      execute Part 1 P1
  n = 2
      execute Part 1 P2
  n = 3
      execute Part 1 P3
  n = 4
      execute Part 1 P4

  If no n is given, then the program will execute all 4 strategies
```

From IDE: Run Part1Main.java from the p1MainClasses package

To Run Part 2:

From terminal: java -classpath ./bin p1MainClasses/Part2Main n m isize fsize istep rep

```
Where:

   n = number of companies

   m = number of crimes

   isize = initial size for experimentations

   fsize = final size for experimentations

   isetp = incremental size step (how much the size increases by trial)

   rep = number of repetitions per size

 * For detailed information head to the in-code implementation *
```

From IDE: Run Part2Main.java, with the desired parameters, from the p1MainClasses package

Part 1:

This part includes a working version of the four strategies to be implemented (P1, P2, P3, P4). Output produces the results for a particular input in which the intersection of all the sets is the set containing the corresponding elements. (See results sample in "Part1 Sample Results Sheet.pdf" or head further into this document)

Part 2:

This part produces the empirical results for the execution times of each one of the four strategies in "part2Results/allResults.txt". (See running results sample graphs in "Part 2 Sample Results Sheets.pdf" or head further into this document).

Understanding the code:

In order to understand the hierarchy of the classes, head to the "UML Kelvin Garcia CIIC4020-030.ucls" file or head further into this document.

Package dataGenerator

Includes:

```
DataGenerator.java (public): generates the data given parameter n, m, and totalSize
DataReader.java (public): reads the data from the files generated
```

Package interfaces

Includes:

```
IntersectionFinder.java (interface) - an objects of type finds the intersection of a
family of sets.

MySet.java (interface) - used by the intersectSets methods that the experiments will
be implementing.
```

Package mySetImplementations

Includes:

```
AbstractMySet.java (public abstract)- an abstract class of MySet.
Set1.java (public) - used by strategy P1 and implements ArrayLists
Set2.java (public) - used by the remaining strategies and implements HashMaps
```

Package p1MainClasses

Includes:

```
FilesGeneratorMain.java (public) - generates the files to be used in Part 1, with the
given number of crimes and companies

Part1Main.java (public) - main class to run Part1 of the project. Outputs the
intersections given by each of the strategies

Part2Main.java (public) - main class to run Part2 of the project. Outputs the
execution times of the strategies given certain parameters. (See "To Run the program"
or head to the in-code implementation of the class for more information)
```

Package setIntersectionFinders

Includes:

```
   AbstractIntersectionFinder.java (public abstract) - an abstract class of
IntersectionFinder
```

Package solutionsAndAidClasses

Includes:

```
P1P2.java (public) - contains the instersectSets method with the implementation used
for P1 and P2

P3.java (public) - contains the instersectSets method with the implementation used
for P3

P4.java (public) - contains the instersectSets method with the implementation used
for P4

Part2Methodology.java (public) - includes constructor needed to run Part 2 in
Part2Main.java with specified parameters. Generates data and saves results

StrategiesTimeCollection.java (public)- includes auxiliary methods to aid in the
management of time data

UnionFinder.java (public) - countains methods unionWriter and arrayToList. The first
is implemented by all strategies and returns the corresponding set representing the
```

```
union of sets given certain parameters. The latter is implemented by P3 and P4 and
converts a given array into an ArrayList
```

Package testerClasses

Includes:

```
   DataGeneratorTester.java (public) - a tester for the DataGenerator.java class
```

Directory doc

Includes:

```
      All proper java documentation for created methods and classes
```

Directory inputFiles

Includes:

```
   Files generated by the FileGenerator.java class
   parameters.txt - parameters used in Part 1
```

Directory part2Results

Includes:

```
   allResults.txt - file contianing results of Part 2
```

UMLKelvinGarciaCIIC4020-030.ucls

UML diagram with class hierarchy

Understanding the different strategies:

P1

```
Finds the intersection by iterating over the sets and removing those elements that do
not belong to that crime. Implements Set1
```

P2

```
Finds the intersection by iterating over the sets and removing those elements that do
not belong to that crime. Implements Set2
```

P3

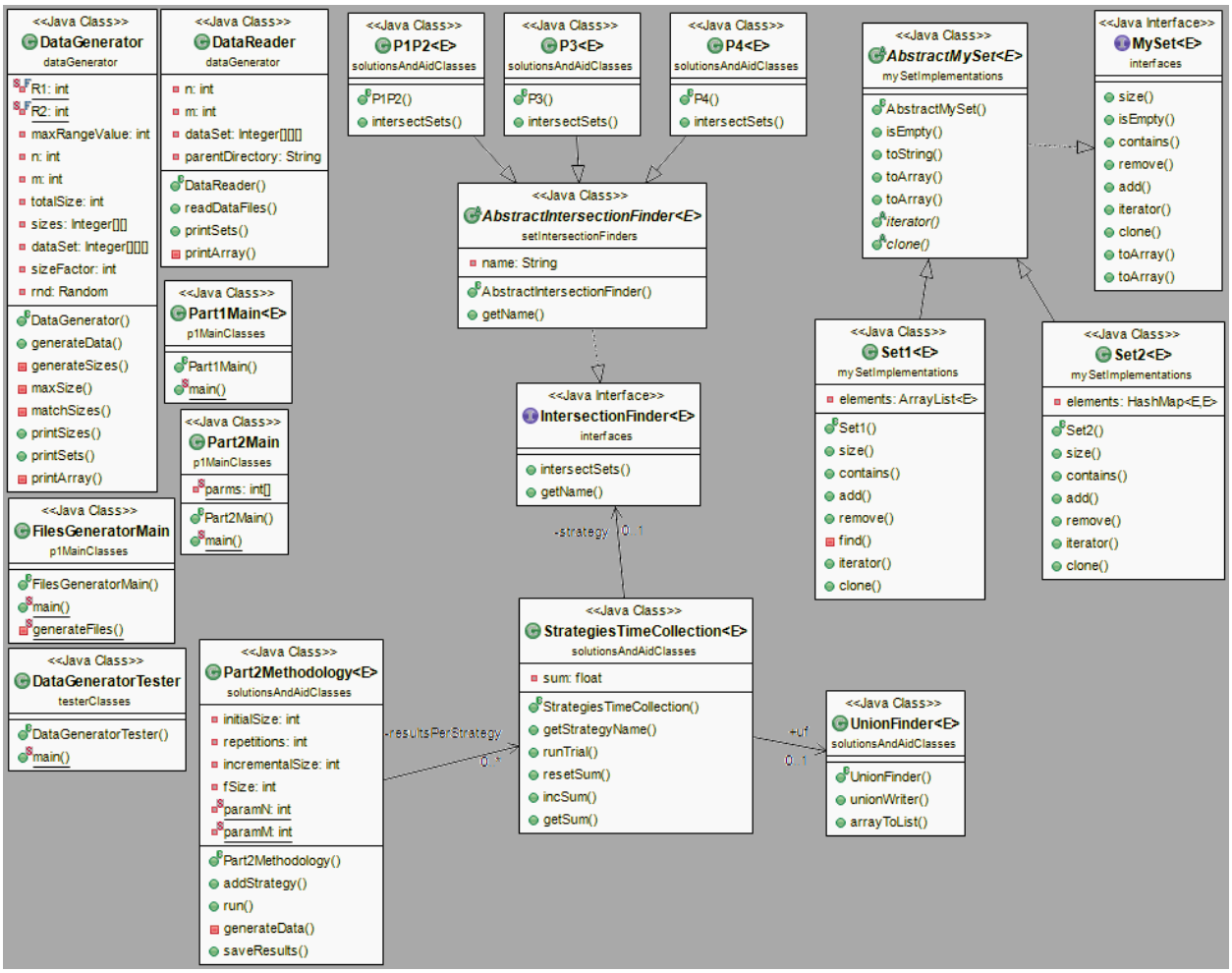
```
Finds the intersection by counting the number of occurrences of each element in the
multiset T0+T1+... + Tm-1. Implements Set2 but unlike P4, P3 iterates over a list to
count these frequencies.
```

P4

```
Finds the intersection by counting the number of occurrences of each element in the
multiset T0+T1+... + Tm-1. Implements Set2 but unlike P3, P4 uses HashMaps to count
these frequencies.
```

To further understand how these strategies are implemented head to the in-code descriptions

UML Diagram


<<Java Class>>
DataGenerator
dataGenerator
R1: int
R2: int
maxRangeValue: int
n: int
m: int
totalSize: int
sizes: Integer[][]
dataSet: Integer[][][]
sizeFactor: int
rnd: Random
DataGenerator()
generateData()
generateSizes()
maxSize()
matchSizes()
printSizes()
printSets()
printArray()
<<Java Class>>
DataReader
dataGenerator
n: int
m: int
dataSet: Integer[][][]
parentDirectory: String
DataReader()
readDataFiles()
printSets()
printArray()
<<Java Class>>
P1P2<E>
solutionsAndAidClasses
P1P2()
intersectSets()
<<Java Class>>
P3<E>
solutionsAndAidClasses
P3()
intersectSets()
<<Java Class>>
P4<E>
solutionsAndAidClasses
P4()
intersectSets()
<<Java Class>>
AbstractMySet<E>
mySetImplementations
AbstractMySet()
isEmpty()
toString()
toArray()
toArray()
iterator()
clone()
<<Java Interface>>
MySet<E>
interfaces
size()
isEmpty()
contains()
remove()
add()
iterator()
clone()
toArray()
toArray()
<<Java Class>>
AbstractIntersectionFinder<E>
setIntersectionFinders
name: String
AbstractIntersectionFinder()
getName()
<<Java Class>>
Part1Main<E>
p1MainClasses
Part1Main()
main()
<<Java Class>>
Part2Main
p1MainClasses
parms: int[]
Part2Main()
main()
<<Java Interface>>
IntersectionFinder<E>
interfaces
intersectSets()
getName()
<<Java Class>>
Set1<E>
mySetImplementations
elements: ArrayList<E>
Set1()
size()
contains()
add()
remove()
find()
iterator()
clone()
<<Java Class>>
Set2<E>
mySetImplementations
elements: HashMap<E,E>
Set2()
size()
contains()
add()
remove()
iterator()
clone()
<<Java Class>>
FilesGeneratorMain
p1MainClasses
FilesGeneratorMain()
main()
generateFiles()
<<Java Class>>
DataGeneratorTester
testerClasses
DataGeneratorTester()
main()
-strategy
0..1
<<Java Class>>
StrategiesTimeCollection<E>
solutionsAndAidClasses
sum: float
StrategiesTimeCollection()
getStrategyName()
runTrial()
resetSum()
incSum()
getSum()
<<Java Class>>
Part2Methodology<E>
solutionsAndAidClasses
initialSize: int
repetitions: int
incrementalSize: int
fSize: int
paramN: int
paramM: int
Part2Methodology()
addStrategy()
run()
generateData()
saveResults()
-resultsPerStrategy
0..*
+uf
0..1
<<Java Class>>
UnionFinder<E>
solutionsAndAidClasses
UnionFinder()
unionWriter()
arrayToList()

Part 2 Results Sample

Size	P1	P2	P3	P4
1000	172121.2	171355.9	323638.84	202655.95
2000	181391.38	325565.84	409844.8	198327.64
3000	154328.06	195490.94	194683.3	119496.58
4000	161234.05	111714.29	160234.38	104471.54
5000	183357.31	120073.1	137331.94	114718.62
6000	247223.34	141203.69	152810.02	127534.47
7000	257154.66	164763.72	174568.3	150338.2
8000	287460.28	181253.9	188886.28	160770.38
9000	327209.12	203712	205778.36	189637.4
10000	366774.53	223878.34	230119.52	194311.19
11000	401133.53	240386.02	243295.08	208472.4
12000	454928.7	258884.27	265062.1	222340.69
13000	477797.62	277982	270991.4	235904.27
14000	523440.47	308564.44	293909.84	263547.47
15000	559140.94	325410.75	305249.9	273710.03
16000	607254.06	348343.16	320678.6	281294.88
17000	624510.9	359344.8	336544.6	291430.7
18000	668455.94	383421.4	347521.56	305814.1
19000	772049.44	417815.97	384096.72	341992.53
20000	798913.4	447321.16	398508.1	354251.12
21000	800486.5	463971.72	409171.16	359558.1
22000	876268.75	500141.2	435473.1	386376.38
23000	921024.75	526132.6	442661.16	397279.2
24000	976563.8	538916.5	450082	412570.47
25000	1018493.44	567963.4	486422.7	434984.72
26000	1069910.2	595485.25	496574.3	439039.75
27000	1128668.5	633749.7	516254	461288.2
28000	1183262.4	645630.94	564610.06	476575.9
29000	1243785.5	666167.4	540434.9	480101.88
30000	1312824.6	690637.4	552235.06	494310.56
31000	1361529.5	718869.6	578609	527472.6
32000	1420665.2	742725.7	588269.06	529604.06
33000	1480072.5	766291.1	612201.3	541245.75
34000	1578542.4	834041.1	656766.8	564715.9
35000	1625260.5	830638.56	631471.5	570580.06

36000	1685059.2	849115.4	639708.6	580091.75
37000	1777947.4	883844.75	666608.1	618100.44
38000	1813635	897089.25	678244.5	617746
39000	1907990.5	930600.2	694708.2	628962.8
40000	2080795.9	1013269.44	769613.06	679270.75
41000	2078625.2	1019373.44	725022.6	660222.2
42000	2166675	1028897.1	755702	681165.75
43000	2182217	1073550.8	791207.1	711485.7
44000	2303476.5	1102426.4	787201.5	727273.94
45000	2378600.8	1134686.1	827316.8	750970
46000	2539609.8	1269505.6	870039.94	813441.5
47000	2676483.8	1238923	879052.8	817201.94
48000	2612312.8	1225310.4	881923.1	787487.06
49000	2718309.8	1292550.5	872328.6	801915.75
50000	2850762	1327711.6	899686.06	832061.44

Parameters for this run: 10, 50, 1000, 50000, 1000, 200