Advanced Data Structures Project

Dijikstra Algorithm for shortest path

Simple Scheme Vs Fibonacci Scheme

UFID: 81118155

Name: Sakthivel Manikam Arunachalam

E mail: sarunac1@ufl.edu

Project Deliverable:

ManikamArunachalam_Sakthivel.zip

||

||------------ProjectCOP5536Report.pdf

||------------Project_javadoc (Folder containing Javadoc)

||-----------||-----------------------------||----index.html

||-----------Dijikstra (Netbeans Project Folder)

||-----------executablefiles (Folder containing all the executable class files)

||-----------sourcefiles (Folder containing all the executable class files)

||-----------README.txt

Project Technology and Environment:

- Developed using JAVA according to **JDK 7** specifications.
- Integrated Development Environment : **Netbeans 7.4**
- No external libraries used.

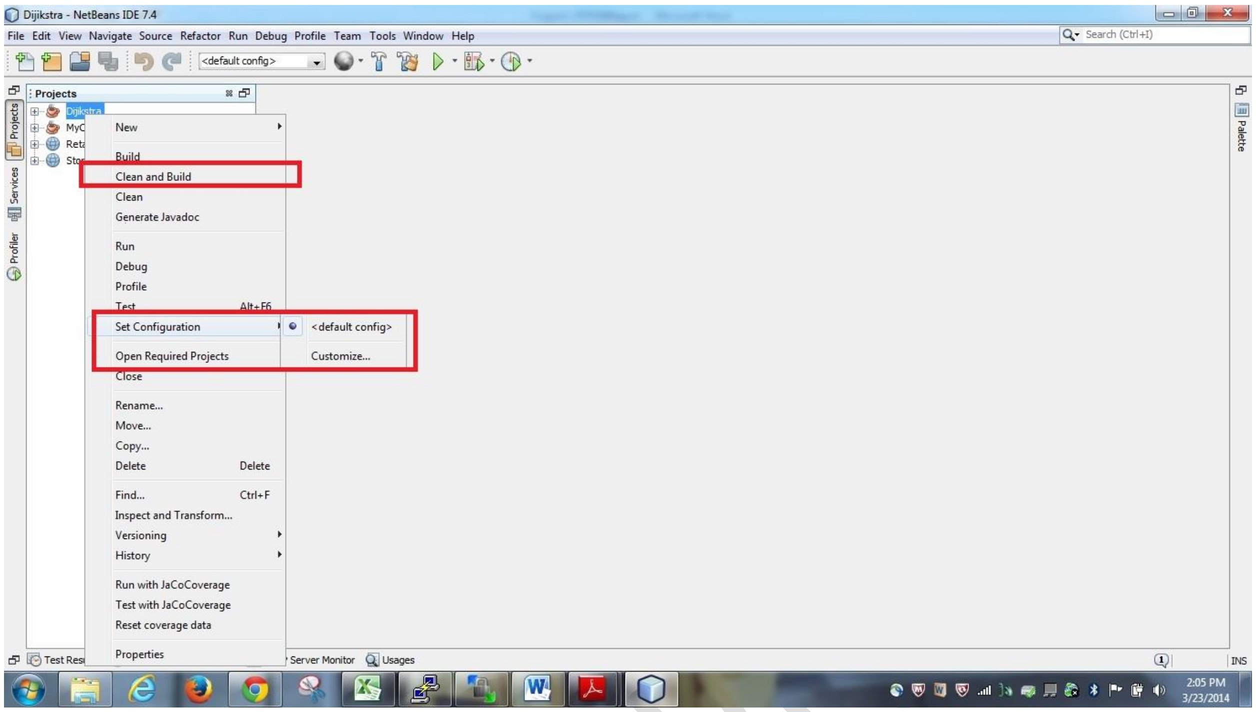
How to compile and run:

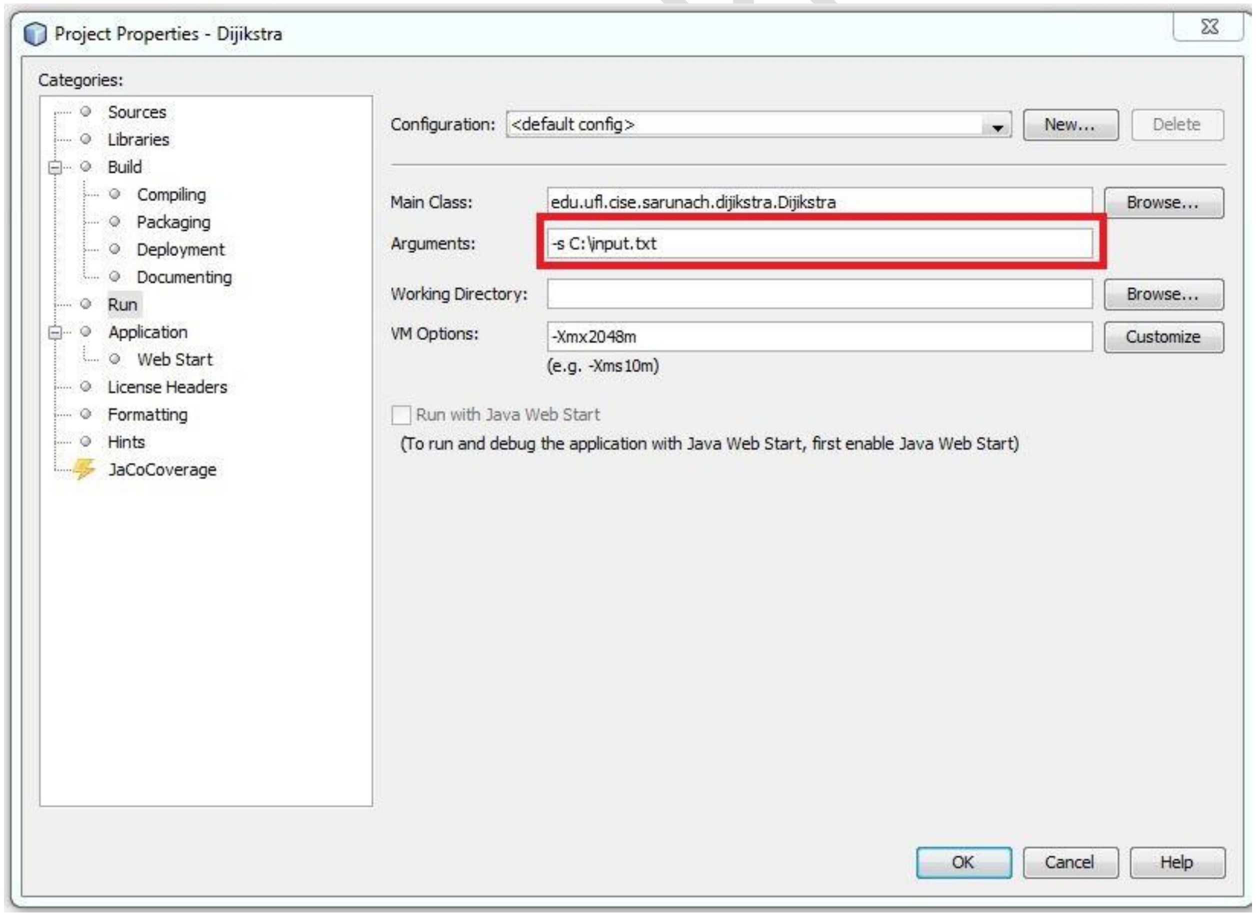
Opening Project in Netbeans:

- Open Netbeans IDE.
- Go to the folder ManikamArunachalam_Sakthivel which is the submitted project folder.
- Open the project directly under the folder called Dijikstra.
- The project is automatically recognized by the Netbeans IDE.
- Now right click the project and click – clean and build.
- This compiles the project.

Running the project in Netbeans:

- Right click the project and click set configuration -> customize.
- Now in the arguments field give the following options as per snapshot and click OK.
- Then right click on the project folder and click Run.
- The output window gives the result.
- If GC error or Heap error occurs, then please add –Xmx2048m in VM Options.





Configuration: <default config>

Main Class: edu.ufl.cise.sarunach.dijikstra.Dijikstra

Arguments: -s C:\input.txt

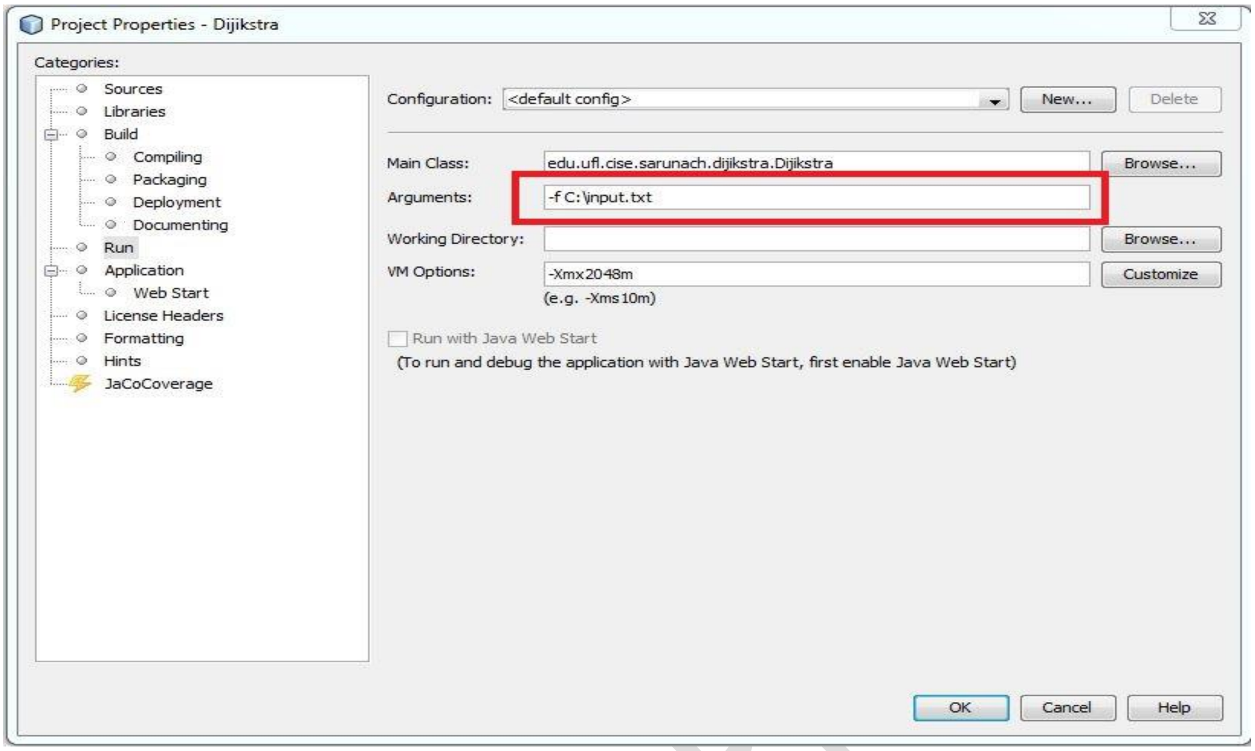
Working Directory:

VM Options: -Xmx2048m

(e.g. -Xms10m)

Run with Java Web Start

(To run and debug the application with Java Web Start, first enable Java Web Start)


Project Properties - Dijikstra

Categories:

- Sources
- Libraries
- Build
  - Compiling
  - Packaging
  - Deployment
  - Documenting
- Run
- Application
  - Web Start
- License Headers
- Formatting
- Hints
- JaCoCoverage

Configuration: <default config>

Main Class: edu.ufl.cise.sarunach.dijikstra.Dijikstra

Arguments: -f C:\input.txt

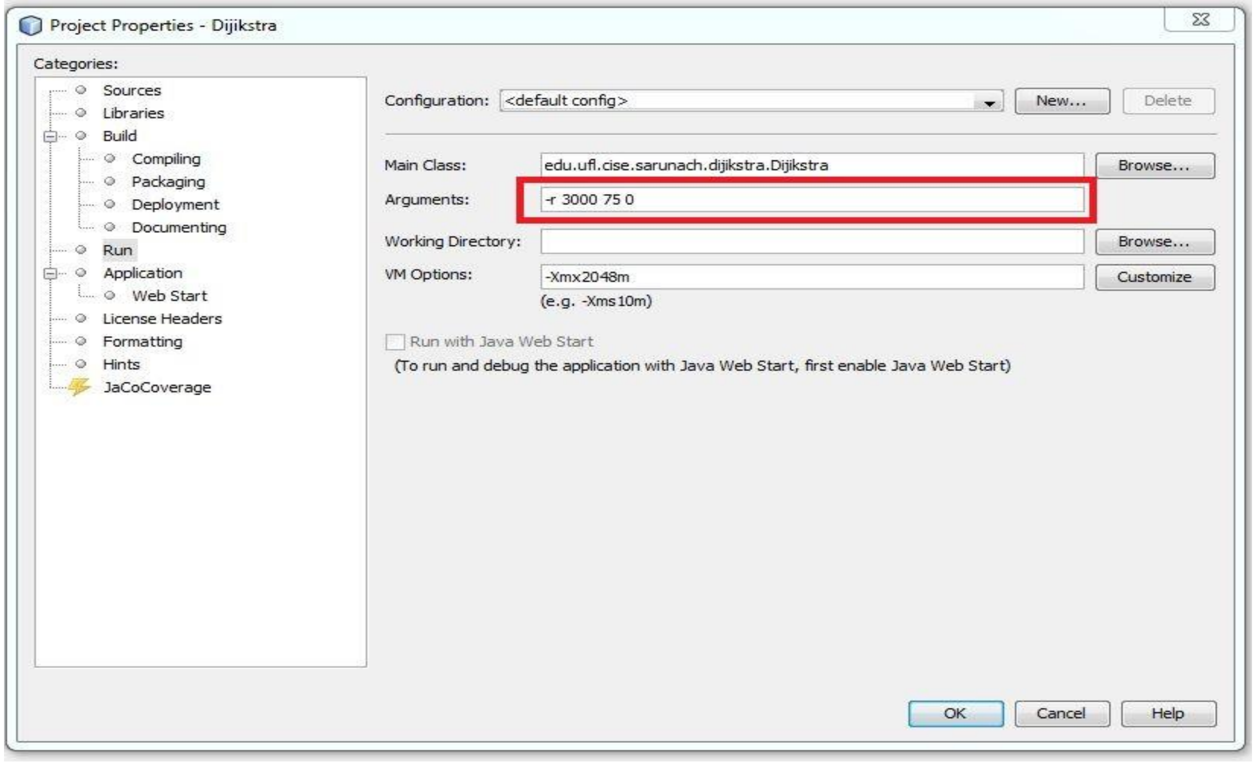
Working Directory:

VM Options: -Xmx2048m

(e.g. -Xms10m)

Run with Java Web Start

(To run and debug the application with Java Web Start, first enable Java Web Start)


Project Properties - Dijikstra

Categories:

- Sources
- Libraries
- Build
  - Compiling
  - Packaging
  - Deployment
  - Documenting
- Run
- Application
  - Web Start
- License Headers
- Formatting
- Hints
- JaCoCoverage

Configuration: <default config>

Main Class: edu.ufl.cise.sarunach.dijikstra.Dijikstra

Arguments: -r 3000 75 0

Working Directory:

VM Options: -Xmx2048m

(e.g. -Xms10m)

Run with Java Web Start

(To run and debug the application with Java Web Start, first enable Java Web Start)

Running project on Linux terminal:

- Copy the folder "sourcefiles" under the submitted project zip file ManikamArunachalam_Sakthivel.zip into the Linux machine.
- Open the Linux terminal and change the current working directory to the directory that contains the source files.
- Now run the command **javac dijikstra.java**.

Running project on Linux terminal:

- Copy the folder "executablefiles" under the submitted project zip file ManikamArunachalam_Sakthivel.zip into the Linux machine.
- Open the Linux terminal and change the current working directory to the directory that contains the executable files.
- Now run the following options to run the program according to the requirements:

Dijikstra Simple Mode commands:

java dijikstra –s filepath

Filepath is the path in which the input file is present.

Output contains the shortest distance to all the nodes with the time taken in milliseconds. Example

```
14 // cost from node 0 to 997

13 // cost from node 0 to 998

12 // cost from node 0 to 999

Simple scheme Time      :240
```

Dijikstra Fibonacci Mode commands:

java dijikstra –f filepath

Filepath is the path in which the input file is present.

Output contains the shortest distance to all the nodes with the time taken in milliseconds. Example

```
14 // cost from node 0 to 997

13 // cost from node 0 to 998

12 // cost from node 0 to 999

Fibonacci Scheme Time    :212
```

Dijikstra Random Mode commands:

java dijikstra –r n d x

n is the number of vertices of the graph.

d is the density of the graph.

x is the source node from which shortest paths are to be found.

Output contains the graph generation time, time taken to run Dijikstra is simple scheme and time taken to run Dijikstra in Fibonacci scheme. Example:

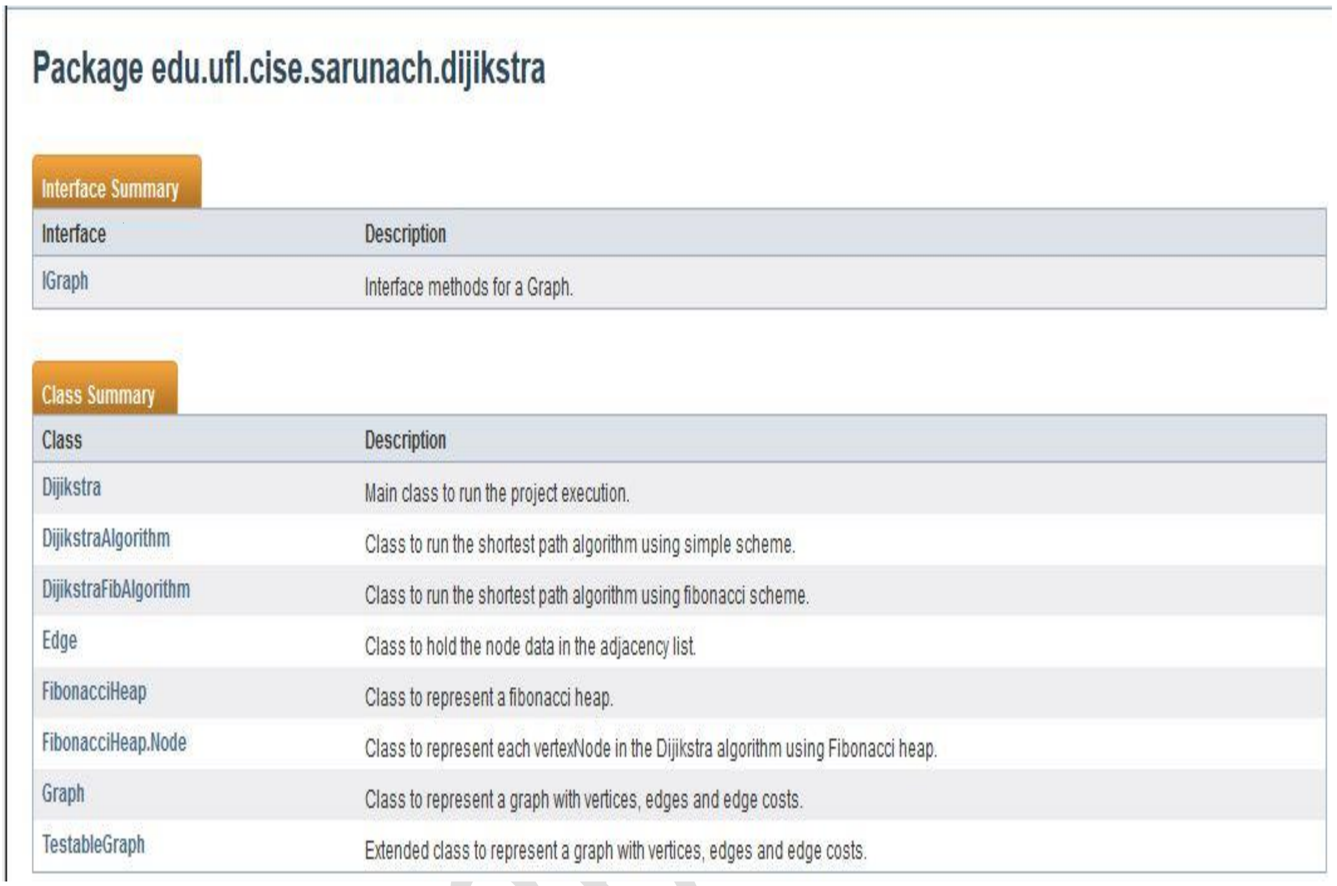
```
Graph Generation Time   : 981
Simple scheme Time      : 192
Fibonacci Scheme Time   : 151
```

Function Prototypes:

ManikamArunachalam_Sakthivel.zip- Unzipped

```
||
||------------Project_javadoc (Folder containing Javadoc)
||-----------||------------------------------||----index.html
```

The index.html is the main page for the project Javadoc which contains all the function prototypes for callable public functions.



Package edu.ufl.cise.sarunach.dijikstra

Interface Summary

Interface	Description
IGraph	Interface methods for a Graph.

Class Summary

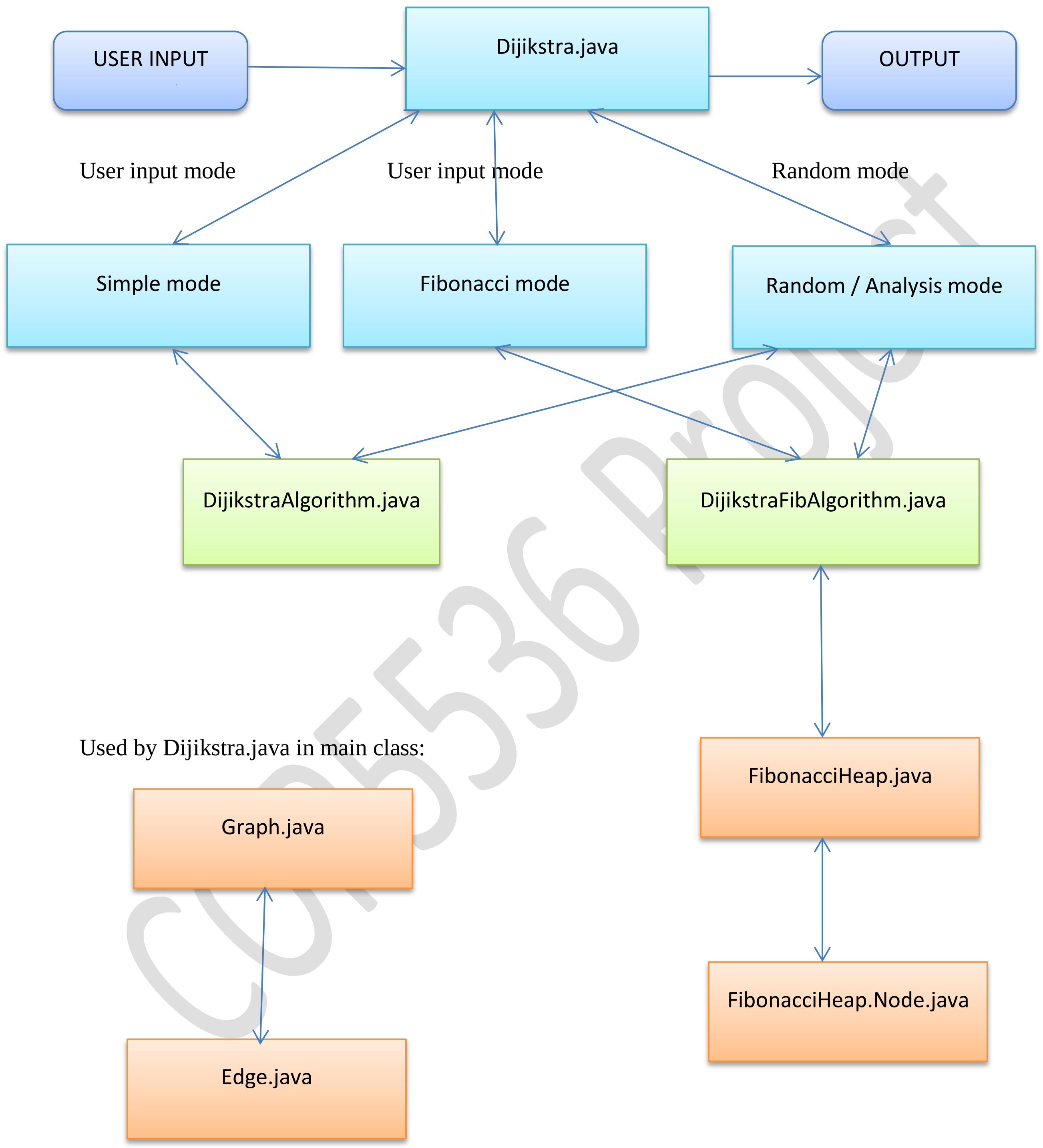
Class	Description
Dijikstra	Main class to run the project execution.
DijikstraAlgorithm	Class to run the shortest path algorithm using simple scheme.
DijikstraFibAlgorithm	Class to run the shortest path algorithm using fibonacci scheme.
Edge	Class to hold the node data in the adjacency list.
FibonacciHeap	Class to represent a fibonacci heap.
FibonacciHeap.Node	Class to represent each vertexNode in the Dijikstra algorithm using Fibonacci heap.
Graph	Class to represent a graph with vertices, edges and edge costs.
TestableGraph	Extended class to represent a graph with vertices, edges and edge costs.

NOTE: The Javadoc has the detailed description of all the methods and classes. Please refer for program functions.

Structure of project and program:

- Dijikstra.java is the main class that communicates with the user via user arguments and output results.
- Dijikstra.java creates objects of class Graph.java.
- Graph.java is used to hold the data of the graph namely vertices, edges, cost etc.
- TestableGraph.java is just an extended class of Graph.java which has an additional print function for debugging capability.
- Graph.java creates objects of Edge.java for storing neighbors and edge costs.
- Dijikstra.java creates objects of two other classes for simple and Fibonacci schemes which are DijikstraAlgorithm.java and DijikstraFibAlgorithm.java respectively.
- DijikstraFibAlgorithm.java additionally creates an object of FibonacciHeap.java to perform Dijikstra algorithm using Fibonacci scheme.
- FibonacciHeap.java has an internal class of Node to store details of nodes in the Fibonacci heap structure.

Program Flow:



Project performance, analysis and conclusions:

Expected Performance:

Fibonacci Heaps:

	Actual	Amortized
Insert	O(1)	O(1)
Remove min (or max)	O(n)	O(log n)
Meld	O(1)	O(1)
Remove	O(n)	O(log n)
Decrease key (or increase)	O(n)	O(1)

Dijikstra using Fibonacci heaps has an overall complexity of O (n log n + e).

Queue using Linked List:

Insert – O (1)

Remove Minimum-O (n)

In case of Sparse Graphs where density of edges is low:

Hence according to the theoretical observations, we can expect the Fibonacci scheme to perform very good in case of sparse graphs where density is low and the number of edges encountered is low. It would outperform the simple scheme by a large performance ratio.

Performance Ratio = time taken by simple scheme / time taken by Fibonacci scheme is HIGH.

In case of Dense Graphs where density of edges is high:

In such cases the performance of a Fibonacci scheme becomes $O(n \log n + n^2)$ which is almost same as performance of $O(n^2)$. The simple scheme in any case performs in time $O(n^2)$. In case of my implementation the simple scheme is implemented using linear structure (array containing distances) which takes slows down the performance in the inner most loop of the Dijikstra.

Performance Ratio = time taken by simple scheme / time taken by Fibonacci scheme is LOW.

Final Expected results:

- **Performance ratio decreases as density increases for a fixed number of vertices.**
- **Performance ratio increases as density decreases for a fixed number of vertices.**
- **When the density of the graph is maximum (100) there is chance that simple scheme performs equally to Fibonacci scheme.**

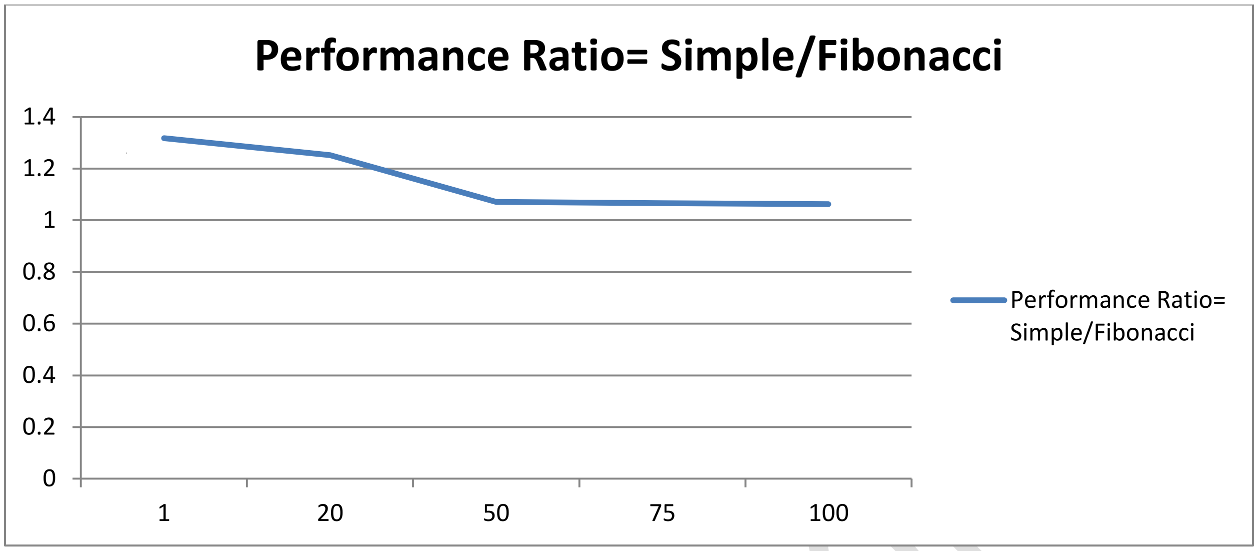
Actual results:

The averages are taken across 10 runs for each combination of densities and number of nodes.

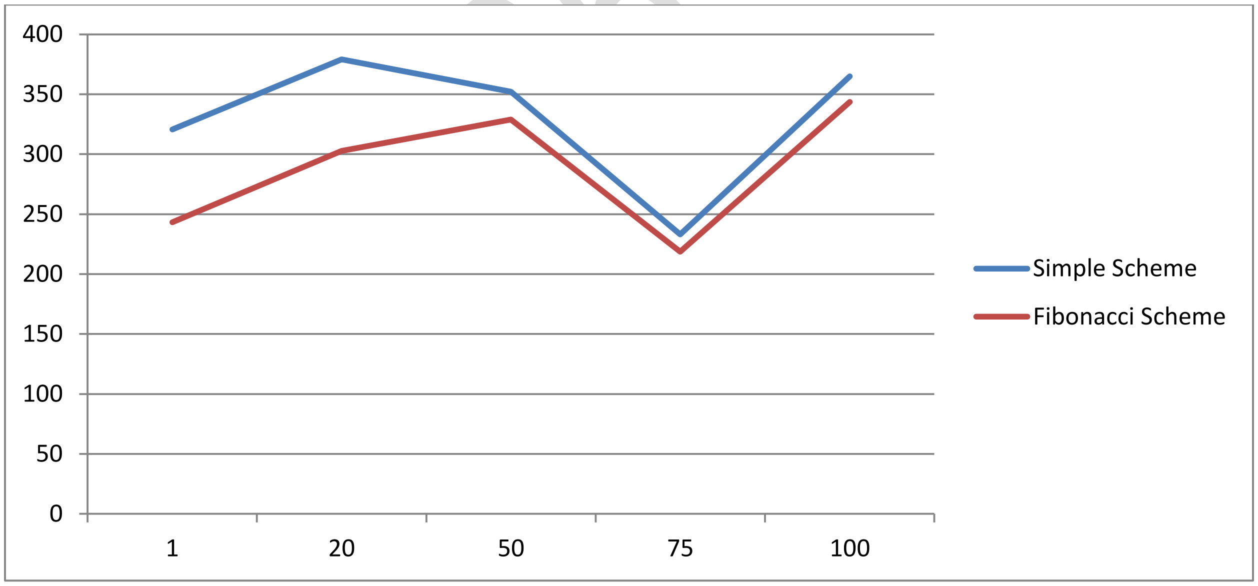
<u>Results for 1000 vertices in random mode with time denoted in milliseconds:</u>

Density	Simple scheme	Fibonacci Scheme	Performance Ratio
1	320.6	243.2	1.318256579
20	379	302.8	1.251651255
50	352	328.8	1.070559611
75	233	218.6	1.065873742
100	365	343.6	1.062281723

Charts:



Graph density is displayed on the X axis.



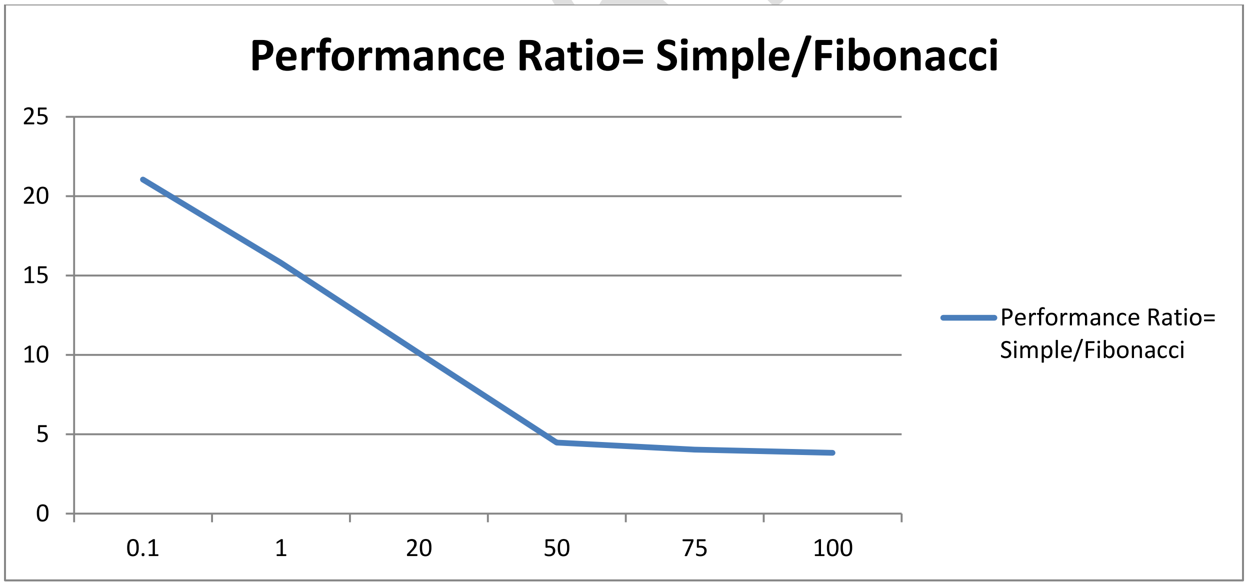
Graph density is displayed on the X axis.

Time in milliseconds is displayed on the Y axis.

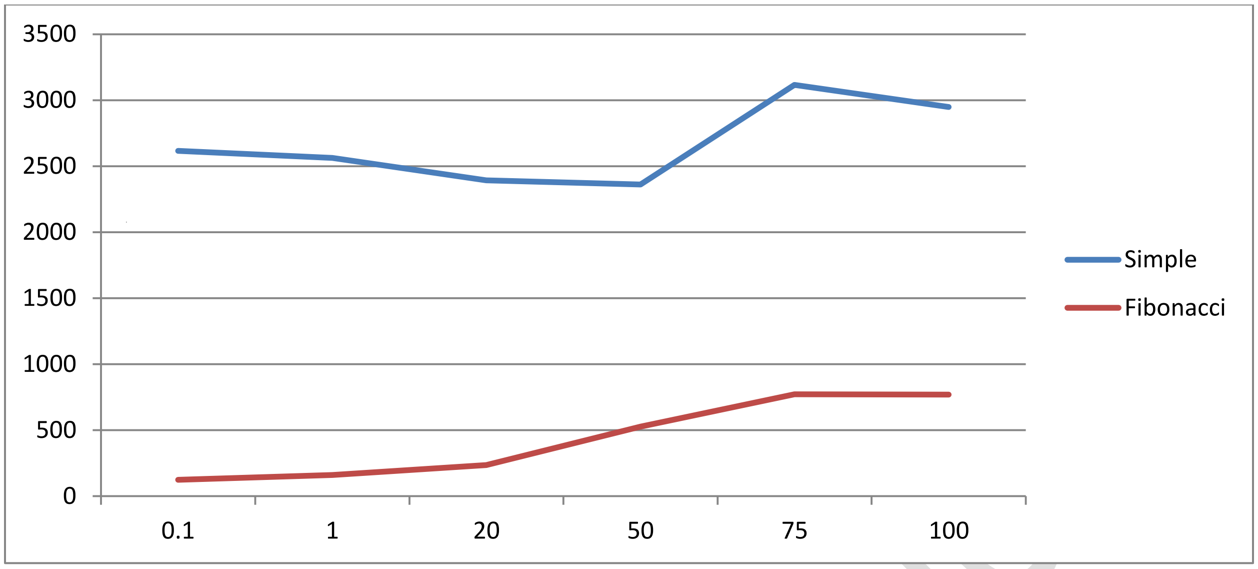
Results for 3000 vertices in random mode with time denoted in milliseconds:

Density	Simple scheme	Fibonacci Scheme	Performance Ratio
0.1	2616.6	124.4	21.03376206
1	2561.6	162	15.81234568
20	2393	236.4	10.12267343
50	2362.2	527.6	4.477255497
75	3114.6	773.2	4.028194516
100	2949.2	769.4	3.833116714

Charts:



Graph density is displayed on the X axis.

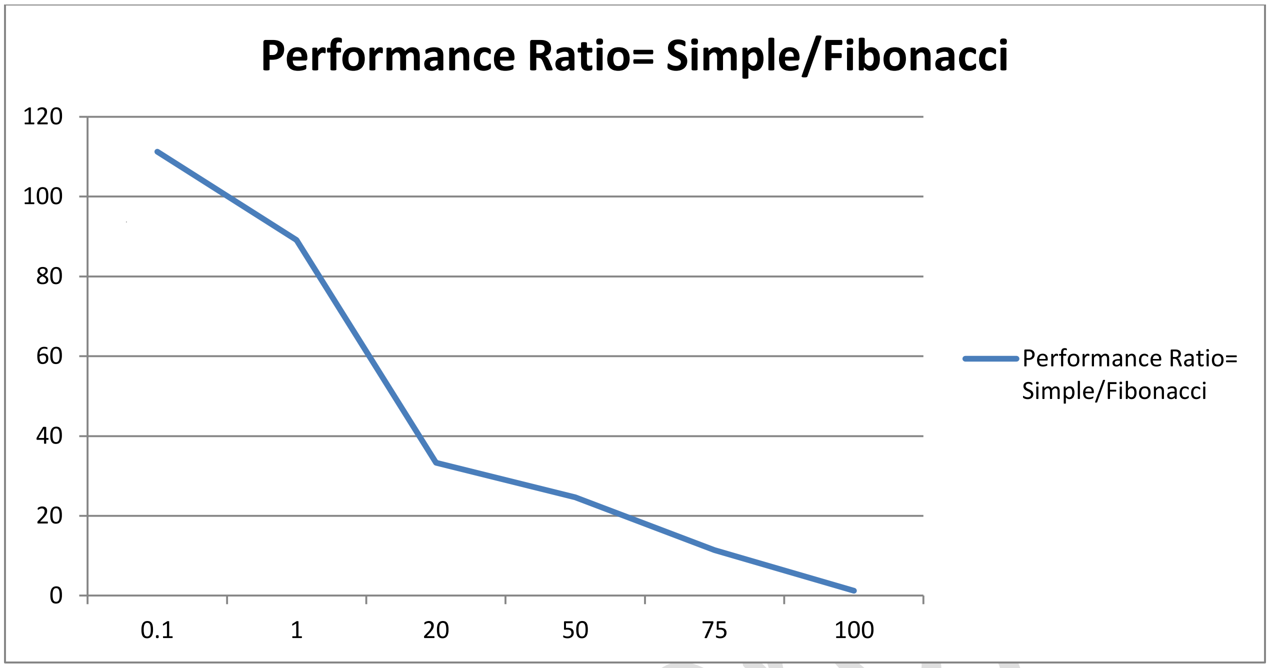


Graph density is displayed on the X axis.

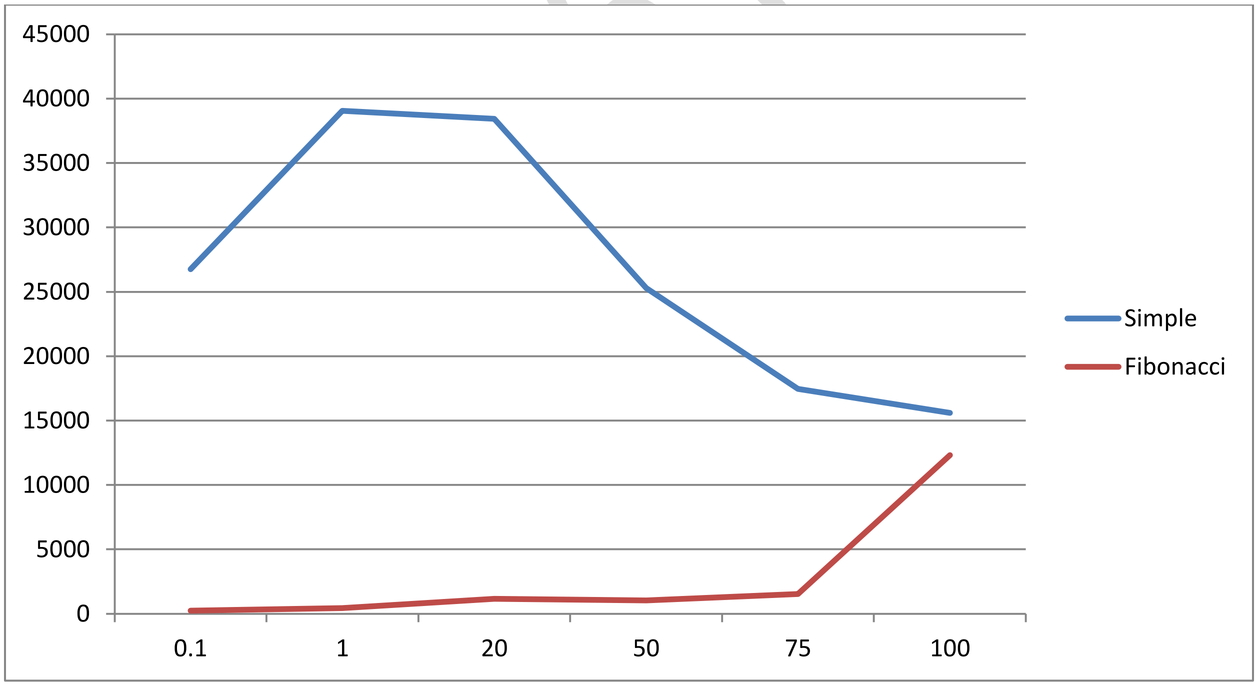
Time in milliseconds is displayed on the Y axis.

Results for 5000 vertices in random mode with time denoted in milliseconds:

Density	Simple scheme	Fibonacci Scheme	Performance Ratio
0.1	26747	240.4	111.2603993
1	39061	438.4	89.09899635
20	38416.8	1152.8	33.32477446
50	25280.4	1024.6	24.67343354
75	17455.8	1526	11.43892529
100	15605.4	12325.6	1.266096579



Graph density is displayed on the X axis.



Graph density is displayed on the X axis.

Time in milliseconds is displayed on the Y axis.

Conclusions and Observations from actual results and analysis **(All readings and observations were made on thunder.cise.ufl.edu server via telnet):**

- The results from the run of 1000, 3000 and 5000 vertices clearly show **that Fibonacci scheme performs better** in case of **low density** graphs and **medium** density graphs.
- The results show **that performance factor of Fibonacci scheme reduces** as the **density increases** keeping **the number of vertices** as a **constant factor**. This evident from the **performance ratio** charts shown for 3000 and 5000 vertices randomized run.
- The performance of Simple scheme and Fibonacci scheme **are almost same** when density is 100 percent.

Conclusion:

Sparse graphs – Fibonacci scheme

Dense graphs – Simple scheme or Fibonacci scheme performs almost equally.