# <u>REPORT (ADS programming assignment)</u>

**Language Used**: Java
**Compiler**: javac 1.8.0_40
**Runtime Envirnment**: java version "1.8.0_40"

**Zip file consists of**

- 5 Java source files( ssp.java, routing.java, Graph.java, FabonacciHeap.java, BinaryTrie.java)
- Makefile
- Report

**Structure of program:**
Submission consists of five source files. Each each file is having functions/methods. Below are the classes/file and function defined inside them

**1. Graph.java:** It has method specific for creating graph using adjacency list.

- *Graph(String fileName):* It's the constructor method. It takes fileName as a parameter which is being passed during runtime and create a array of linked list of nodes for graph. It calls *public void addEdge(int source, int target, int weight)* mothod for adding the nodes. Also it tells the <u>totalVertex</u> count and total <u>edgeCount</u> by reading the file.
- *public void addEdge(int source, int target, int weight):* Graph is consisting of array of adjacency list of nodes. This method add nodes in the graph.
- *public int getTotalVertex() :* This method returns the value of totalVertex present in the graph.
- *public int getDistance(int src, int dest):* This method returns the edge weight which we have stored while creating the graph.
- *public LinkedList<Integer> getAdjListOfNode(int I):* It lists the adjacency list of node which are still not discovered.

**2. FabonacciHeap.java** This class is having specific to fibonacci heap implementation. It is a heap data structure consisting a collection of min-heap-ordered trees.

- *public DataEntry insertElem(int val, int* priority*):* Inserts the specified element into the Fibonacci heap with the specified priority. Priority is weight+ distance traveled so far. DataEntry is a actually a node in heap. Return type is actually a node showing an entry in the tree.
- *public boolean isEmpty() :* It just check if fibonacci heap is empty or not.
- *public DataEntry min() :* It return pointer to the node whose value is least.
- *public int size() :* Size of heap I.e totak number of nodes present in the Fibonacci heap.
- *public DataEntry deleteMin(): R*emove Minimum element node from the Fibonacci heap.
- *public void decreaseKey(DataEntry node, int newPriority):* Decreases the key of

the specified element to a new priority/weight. This is one of the main method being used in Dijkstra's Single Source Shortest Path (ssp) algorithm

- *public static FabonacciHeap merge(FabonacciHeap one, FabonacciHeap two):* It merges two Fibonacci Heap to get a new Fibonacci Heap
- *private static DataEntry mergeTwoLists(DataEntry d1, DataEntry d2)* : Given two pointers to disjoint circularly- linked lists, merges the two lists together into one circularly-linked list.
- *private void cutNode(DataEntry node) :* It cuts node from the parents Fibonacci heap tree, update the cascading cut. Further merge entry into the top level list.

**3. ssp.java :** It is the main class for running Dijkstra's Single Source Shortest Path (ssp) algorithm. It is taking the parameters passed during run time. It initializes instance for graph class and then calling method which is calculating single sort shortest path i.e *private static String sspCal(Graph g, int sVertx, int dVertex).* Done the error handling also. If source veretx and destination vertex both are between zero and totalVertex given. If it's not then it will throw error.

- *private static String sspCal(Graph g, int sVertx, int dVertex):* It initializes th FabonacciHeap class. We are storing the distance in an array. We are keeping track of the previous visited vertex which will help in tracking the path. We are setting the distance to maximum in beginning and further we are decreasing it using method *public void decreaseKey(DataEntry node, int newPriority)* present in FabonacciHeap class.
  Once we get the destination vertex, we will terminate the program and back track the path and print it.

**4. routing.java** It is main class for running the 2nd part of program. Implement a routing scheme for a network. Each router has an IP address and packets are forwarded to the next hop router by longest prefix matching using a binary trie.

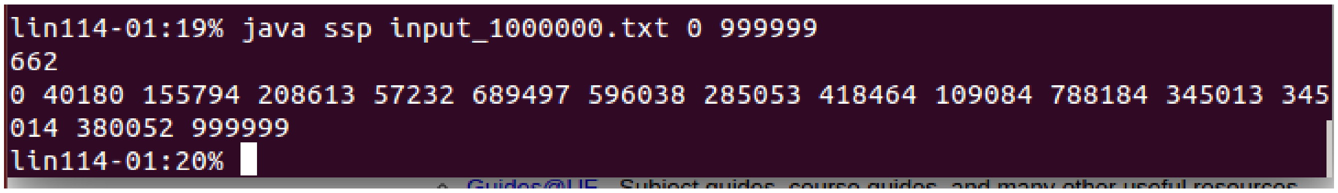
- *private static void createHashMap(String ipFile)*: It creates a hashmap. Key is the vetex number and value is the IP value in binary format
- *private static String ssp(Graph g, int sVertx, int dVertex):* This is same as 1st part. It is returning the Dijkstra's Single Source Shortest Path (ssp) algorithm result.

**5. BinaryTrie.java :** This class is implementation of binary tries and various methods corresponding to it.

- *public void addNode(String ip, int vertex):* It adds nodes in the binary tries.
- *public String longestPrefixMatch(String binaryIP):* It will return the longest prefix match corresponding to an IP address.
- *public void postOrderConcatenate() :* It will concatenate the binary tries if parent is having only one chidren or no children while doing post order traversal
- *public int get(String key)* : function to get next node for an IP address.

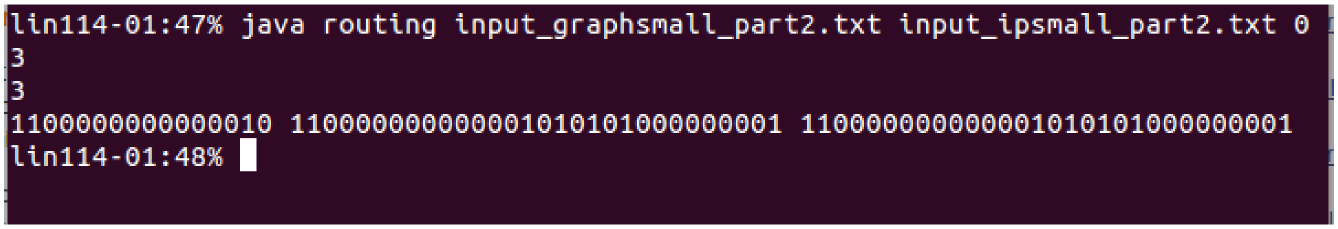
**Steps to run:**
On console terminal type *make* after navigating to the directory where all source code is present. It will generate all the .class files. Below are ways to run part 1 and part 2 of the program
$java ssp <file_name> <source_node> <destination_node>
$java routing <file_name_GraphInput> <file_name_ip_vertexInput> <source_node> <destination_node>

**TestResult:**
Ran ssp on 1Million nodes input provided on thunder machines, it finished in around 35 sec



```
lin114-01:19% java ssp input_1000000.txt 0 999999
662
0 40180 155794 208613 57232 689497 596038 285053 418464 109084 788184 345013 345
014 380052 999999
lin114-01:20%
```

Ran routing on input file provided:



```
lin114-01:47% java routing input_graphsmall_part2.txt input_ipsmall_part2.txt 0
3
3
1100000000000010 11000000000000101010101000000001 11000000000000101010101000000001
lin114-01:48%
```