ADS Programming Project

Name: Prachi Mittal

UFId: 41454991

Email: prachim89@ufl.edu

1 CONTENTS

2 PROJECT OBJECTIVE

The project is divided in two parts.

Part 1: Implement Dijkstra's single source shortest path for undirected graphs using Fibonacci Heap.

Part2: Implement a routing scheme for a network where each router has an IP address and packets are forwarded to the next hop router by longest prefix matching using a binary trie.

3 PROJECT ENVIRONMENT

3.1 LANGUAGE AND DEVELOPMENT ENVIRONMENT

The program is implemented in **JAVA (*v 1.8*)**. Eclipse Luna Service Release 2 (4.4.2) is the development environment

3.2 COMPILER

The program has been compiled using: ***javac 1.7.0_67***

3.3 COMPILER INSTRUCTIONS

The program files can be compiled through makefile

To compile the program, execute the following command:

make

On successful compilation, all the dependent java files will be compiled and class files will be produced in the same directory.

4 FUNCTION PROTOTYPES

4.1 FIBONACCIHEAP.JAVA

public FibNode<T> enqueue(T value, double priority) {

/* - Inserts the specified element into the Fibonacci heap with the specified priority. */}

public FibNode<T> min() {

/*Returns an Entry object corresponding to the minimum element of the Fibonacci heap empty.*/}

public static <T> FibonacciHeap<T> merge(FibonacciHeap<T> one, FibonacciHeap<T> two) {

/* Given two Fibonacci heaps, returns a new Fibonacci heap that contains all of the elements of the two heaps*/**}**

public FibNode<T> dequeueMin() {

/* Dequeues and returns the minimum element of the Fibonacci heap*/**}**

public void decreaseKey(FibNode<T> entry, double newPriority) {

/* Decreases the key of the specified element to the new priority*/ **}**

private void cutNode(FibNode<T> entry) {

/* Cuts a node from its parent. If the parent was already marked, recursively cuts that node from its parent as well*/**}**

4.2 SSP.JAVA

public static ArrayList<String> runDijkstra(File file, String srcId, String dstId) throws IOException {

/*Creates paths from source to detination. Calls Fibonacci heap and extracts the path with minimum weight*/ **}**

4.3 BINARYCONVERSION.JAVA

public static ArrayList<String> convertIP(File input) throws IOException {

/*Reads the input file line by line skipping empty lines and call processIP to convert IP to binary*/ **}**

public static String processIP(String ip) {

/*Splits the line with . and converts all four parts to corresponding 8 bit binary number. Thus a 32 bit binary IP is returned*/ **}**

4.4 TRIENODE.JAVA

public TrieNode subNode(char c){

```
/*Defines structure of each node in Trie*/}
```

4.5 TRIE.JAVA

```
public void insert(String ip,int vertex){
```

/*Forms a Trie using characters in ip address and inserts key i.e. vertex in the leaf node*/ **}**

```
public Integer search(String ip){
/*Searches for key value at end of a ip in Trie*/ }

public void postTraversal(TrieNode node){
/*Do a post order traversal on Trie. Compresses the trie in following conditions
```

- `If node.left.vertex == node.right.vertex`
- `If node.right == null && node.left != null`

```
    •  If node.left == null && node.right != null
*/}
```

4.6 ROUTING.JAVA

`public static void main(String[] args) throws IOException {`

`/*Starting with source node, keep each node as destination, run dijkstra,` get first hop value in shortest path. Now in Trie for source node, traverse with IP of every other node and insert hop values. Store the Trie in hash map with key as the source node you started with. Now search IP with longest prefix match with final destination node, get the first hop, then refer the hashmap with key value as that hop value and get next hop. Repeat till you hit final destination. Print prefix on your way.*/ **}**

5 PROGRAM STRUCTURE

5.1 SSP

Ssp.java(main()→`runDijkstra())→FibonacciHeap.java`

In this part we implement Dijkstra's single source shortest path algorithm to find shortest path from source node to destination node and the corresponding minimum weight. The input file containing undirected information is read, a Hash Map is used to store the graph using source as the key. These are then added to FibonacciHeap with infinite priority. To apply Dijkstra's, remove the minimum node and find all nodes that are connected to it. In the end find the path from which destination is reached. Fibonacci Heap is maintained to find minimum weighted edge.

5.2 ROUTING

routing.java(main())→`BinaryConversion.java→new Trie()→ssp.java(runDijktra())→Trie.java(insert()→postTraversal()→search())`

Routing takes input undirected graph, IP addresses of nodes, source and destination nodes. First we convert the IP Addresses to 32 bit Binary notation. We start with source node, construct a trie for it. This trie is constructed by running Dijkstra from this node to every other node as destination. We insert first hop value in the leaf nodes of Trie. These tries are stored in Hash Map. After insertion, Tries are compressed by pot order traversal. On these compressed Tries we apply search. Start with source node, we take First hop for destination node in first trie, then search the trie corresponding to first Hop and check the hop value there. Repeat till we hit the destination node. As output we print the total weight of shortest path and Longest prefix matched IP sequence for IP addresses of each node along shortest path.

6 EXECUTION

6.1 SSP

To run Dijkstra's single source shortest path algorithm, execute the following command

java *ssp file_name source_node destination_node*

This reads the input from a file ‘file-name’. The arguments ‘source_node’ and ‘destination_node’ are integers representing the vertex numbers of the source node and the destination node for Dijkstra’s algorithm respectively.

6.2 Routing

To run routing, execute the following command

***java** routing file_name_1 file_name_2 source_node destination_node*

This reads the input graph from a file ‘file-name_1’ and IP addresses from ‘file_name_2’. The arguments ‘source_node’ and ‘destination_node’ are integers representing the vertex number of the source node and the destination node respectively for a packet.

7 Expected Output

7.1 SSP

The output for Single source shortest path(ssp) consists of two lines. The first line has one integer, the total weight. The second line shows the path starting with source_node and ending with destination_node separated by a space between nodes.

An example of output when the source is 0 and the destination is 2:

```
13

0 1 2
```

7.2 Routing

The output of routing consists of two lines. The first line has one integer, the total weight. The second line shows the matched prefixes of each node on the path starting with source_node and ending with destination_node.

An example of output when the source is 0 and the destination is 3:

```
3
1100000000000010 110000000000001010101000000001 110000000000001010101000000001
```