COP 5536 Advanced Data Structures
Spring 2015 Assignment #1 (Programming Project)

Report

Dijkstra's Algorithm Implementation Using Fibonacci Heap And Implementation of Routing Scheme using binary Trie

Author: VIKRAM KHURANA UFID: 66921015 UF Email: vikram0602@ufl.edu

1. Working Environment

1. Hardware Requirement:
   - Hard Disk space: Minimum 5GB
   - Memory: 512 MB Minimum

2. Operating System:
   - Win32 and above versions like WINDOWS 98/NT/XP/VISTA/7/8
   - Linux

3. Compiler:
   - Standard Java Compiler (JDK)

2. Compile Process

- Open the terminal/cmd and change directory to where the files are located.
- On terminal/cmd run the make file.
- For the first part write:

  java ssp <filename> <source> <destination>
- For part 2:

  java routing <filename1> <filename2> <source> <destination>

3. Classes Used

The three java files includes different classes-
Node class: Class defined for the structure of a node of the graph. Contains label of the node and a list to store the adjacent nodes.
AdjacentNode class: Class for the adjacent nodes to be stored in the adjacency list of a node. Contains label and weight of the adjacent node.
FibHeapNode class: Class for the structure of a Fibonacci heap node. Contains the key, value, degree, parent, left sibling, right sibling, child and cut fields
ssp class: Class containing the main method and all associated methods.
Routing class: used for finding prefix using graph and ip network.
Trie class: Binary Trie Data structure full implementation.

4. Function Prototype

Prototypes in ssp.java and routDijkstra.java:

Function: public static Insert(int key, int value)
Arguments: key and value to be inserted
Return value: Pointer to Node created
Description: For given key, value pair it creates a FibHeapNode object and calls Add() method to add it to the heap.

Function: public void Add(FibHeapNode f)
Arguments: Pointer to a fibonacci heap node
Return value: void
Description: to add the node to the Fibonacci heap and update pointers

Function: public FibHeapNode DeleteMin ()
Arguments: none
Return value: reference to minimum Fibonacci Heap node
Description: Deletes the root node i.e the node with least key value and returns it. Calls Join() method to join two trees of equal degree and updates min element i.e root.

Function: public void DecreaseKey(FibHeapNode node,int newKey)
Arguments: Node to be updated and the new key to be inserted into that node
Return value: void
Description: Updates the key value of a given node with a new key value and applies the cascading cut .

Function: public void Remove(FibHeapNode n)
Arguments: Reference to node to be removed
Return value: void
Description: Removes the given node and its subtree from the heap and updates pointers

Function: public void Join(FibHeapNode node)
Arguments: Reference to node to be joined with root of tree of equal degree
Return value: void
Description: Joins the tree with root as given node with a tree of equal degree by making one tree subtree of the other

Function: public void FileInputGraph()
Arguments: none
Return value: void
Description: Takes number of nodes, source node, number of edges, edges and cost from a file and generates corresponding graph.

Function: public void ShortestPathFibonacciHeap(int source, int destination)
Arguments: destination and source node
Return value: string[]

Description: Computes next shortest distance from source by calling DeleteMin() method and updates the distances of neighboring nodes by calling DecreaseKey() method on neighboring nodes. Finally the distance array consists of the shortest distances from all nodes to the source node. Returns shortest distance between source and destination.

Function: public static void PrintResult(String[] a)
Arguments: input mode
Return value: void
Description: Prints the distance and path at terminal

Prototypes in routing.java

Function: public static String ipToString(String ip)
Arguments: ip address
Return value: string
Description: Function to convert ip address to binary string later on it may be used to put data into the binary trie.

Function: private Node add(Node x, String ip, int vertex, int d)
Arguments: node, corresponding ip , vertex
Return value: Node
Description: Method to add new <ip,nextNode> pairs   to trie

Function: public String longestPrefixOf(String ip)
Return value: string
Description: Function to get the longest prefix in trie for an ip adress

Function: private int longestPrefixOf(Node x, String ip, int d, int length)
Return value: integer
Description: Private function called in public longestPrefixOF for getting the longest prefix match of a given ip address

Function: public int get(String key)
Return value: integer
Description: Function to get the value for a given key i.e. next node for an ip

Function: private Node get(Node x, String key, int d)
Return value: Node
Description: Private function called in get function for getting the value of given node given longest prefix match

Function: public void postOrderRemoval(Node x)
Return value: void
Description: Method for removing the nodes having the same next hope while traversing the tree in postorder.

5. Implementation of Routing Problem

Main Class =Routing

- First we make an arraylist here for nodes number to ip mapping the ip data come from first file argument.
- Then we pass the first file to the routDijkstra/ssp class in order that it may read the graph vertices and edges from there.

- Now what we iterate through the vertices i.e. from 0 to n-1 and find the shortest path of each vertex from the rest using Ssp class. we then use a function for converting ip to binary and find the ip of target that is souruce->target ip and insert it into the trie tree of source node

```
for(int i=0;i<lenght;i++)
        for(int j=0;j<lenght;j++)
                : rest code
```

- We then take the source node and destiantion node and find the shortest path of them using ssp and then there path in the form of binary through trie data structure.

6. Execution Time for Sample cases of Dijkstra

Nodes	source	Destination	Execution time(ms)
1000	0	999	54
1000	5	940	115
5000	0	999	155
5000	5	940	134
5	0	4	0
1000000	0	99999	-(went out of heap memory)