ADVANCE DATA STRUCTURE

COP 5536

PROGRAMMING PROJECT

SPRING 2015

Submitted By:

Varun Kumar
UFID -1934 8847
Email:vakumar@cise.ufl.edu

INDEX

1) Compilation Steps:

This project have been implemented in C++ and compiled in Linux (Ubuntu) operating system with g++ compiler

Following steps need to be followed to compile and execute the program.

a) Copy all the files mentioned in zip folder including the makefile in a separate directory

b) Run "**make clean**" to remove all previous output/executable files.

c) Run "**make**" . This will generate two executables **ssp(Part 1)** and **routing(Part 2)**.

d)The two executables generated should be executed in the following way

Part 1 Execution:

```
./ssp --InputGraphFile --sourceVertex --destinationVertex
```

InputFileName is the filename which contains information about graph ie source vertex ,destination vertex and edge weight between them **sourceVertex** is the source vertex from where the shortest path to be calculated .

destinationVertex is the destination vertex upto which the shortest path is be calculated.

Part 2 Execution:

```
./routing  ---InputGraphFile --InputIpFile --sourceVertex
--destinationVertex.
```

InputFileName is same as above.

InputIpFile This file contains the IP address of each vertex.

sourceVertex is same as above.

destinationVertex is same as above.

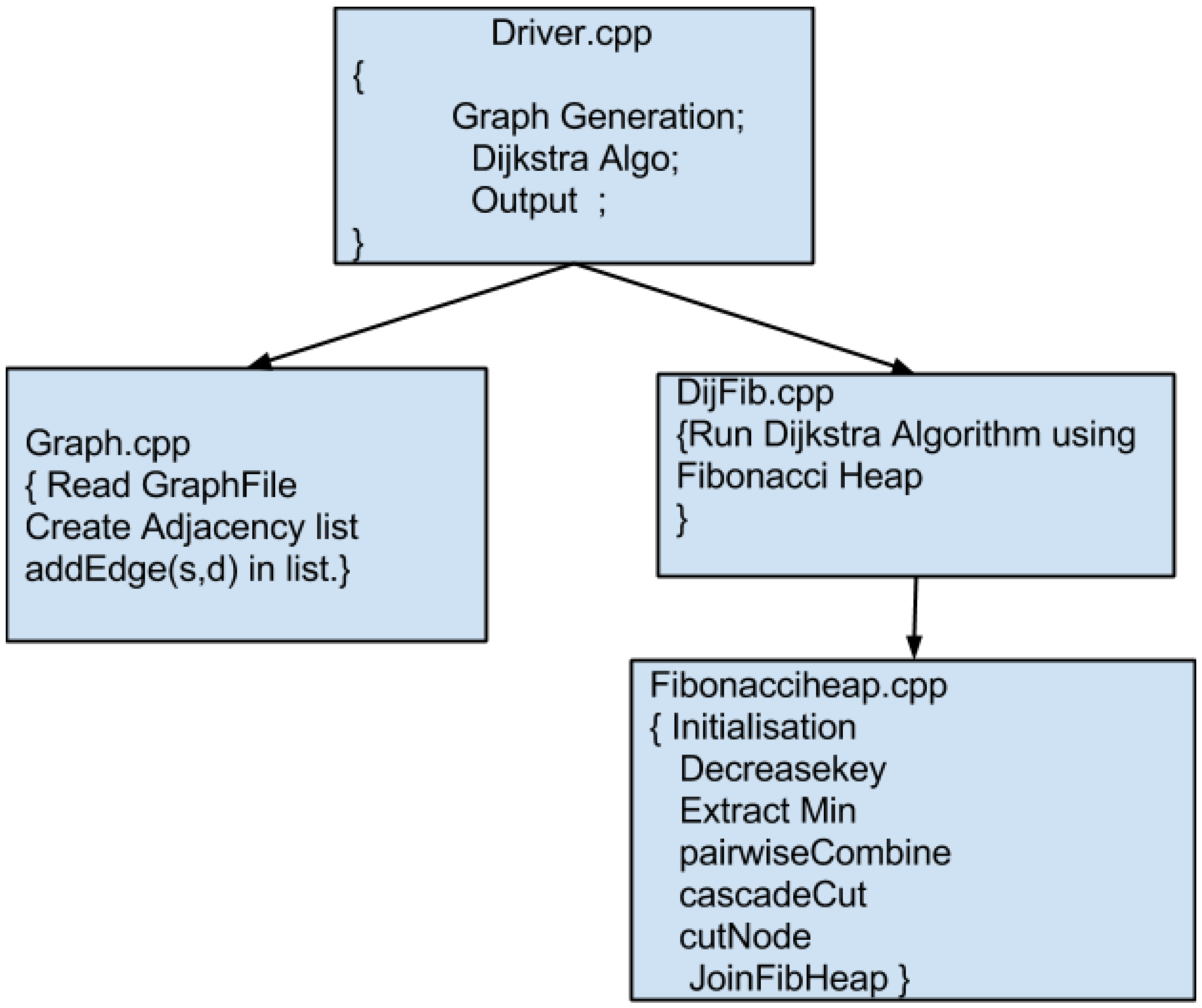
2) Introduction :

The Project is divided into two parts.In Part 1 of project we need to implement Dijkstra Single source shortest algorithm using a Fibonacci.In Part 2 of the project we need to implement a routing scheme (routing) for a network .The data structure to be used for implementing Part 2 is Binary Trie and also Dijkstra algorithm that was being implemented in Part1 was also used .
The benefit of using Fibonacci Heap for Dijkstra is that the complexity of algorithm is O (E+VlogV) time where E is number of edges in the graph which is better as compared to using other data structures like simple heap or array .

3) Class description ,Function and Structure:

A) Part 1:

This part starts with **driver.cpp** file which is the starting point of execution .It takes the input parameters from the console given by the user .After that it calls the other files for graph generation(Graph.cpp) ,executing Dijkstra algorithm (DijFib.cpp)and then it prints the output in the required format on the console .
The High level design of the Part is mentioned in the following diagram.



Detail Description of File and Function:

A) Graph.cpp/Graph.h

This file contains all the class objects ,structures and functions for Graph creation taking input from the file (mentioned by the user in command line argument) and the creates graph in adjacency list form.The GraphNode structure is as follows :

```
struct GraphNode
{
        int des;
        int weight; //Weight present on this node .This weight is used during Dijkstra algorithm
        int parent;   //Parent vertex  of this node
        GraphNode(int d,int w,int p)  //Initialisation of this node
        {
                des=d;
                weight=w;
                parent=p;
        }
}

class Graph
{       int V;      //No of vertices in the node
        list<GraphNode> *adjlst; //Array of lists used for adjacency list
        public:
        int getVertex();
        list<GraphNode>* getList();
         vector<string>Ip; //vector containing  Ip addresses of each  vertex
        Graph(string);
        ~Graph();
      bool EdgePresent(int s,int d);
        void addEdge(int ,GraphNode);
        };
```

i) getVertex()
Returns no of vertex for this graph .This is required since V variable is private
ii) getList()
Returns adjacency list pointer for this graph .This is required since adjacency list variable variable is private
iii) Graph(string)
This function takes input file name as an argument and creates Graph reading from file and storing in adjacency matrix .
iv) EdgePresent(int,int)
This function checks whther the edge between s and d is already present in adjacency list or not .
v) addEdge (int ,GraphNode)
This function add the created Graphnode in adjacency list of the corresponding source node .

B)Fibonacciheap.cpp/Fibonacciheap.h

The Node for Fibonacci Heap is as follows :

```
struct FibHeapNode
{       int degree;  //Degree present for this node
        int weight;  //Weight present on this node
        int vertxId;  //Vertx Id for this node .This vertex id is same as graph vertex number .
        FibHeapNode *parent; //Parent pointer of this node .Required for doubly linked list
        FibHeapNode *left; //left pointer for this node .Required for doubly linked list
        FibHeapNode *right;  //right pointer for this node
        FibHeapNode *child; //child pointer for this node
        bool ChildCut; //childCut Value for this node either true or false
};

class FibHeap
{       FibHeapNode *minNode;
        int nodecount;
        public:
        int getNodeCount()
        {
                return nodecount;
        }
        FibHeapNode *getRoot()
        {
                return minNode;
        }
        FibHeap()
        {
                minNode=NULL;
                nodecount=0;
        }
        void InsertNode(FibHeapNode*);
        FibHeapNode* removeMin();
        FibHeapNode *CreateHeapNode(int ,int );
        void decreaseKey(FibHeapNode *,int  );
        void pairwiseCombine();
        void cascadeCut(FibHeapNode*);
        void cutNode(FibHeapNode *,FibHeapNode *);
        void JoinFibHeap(FibHeapNode *,FibHeapNode *);
};
```

i)InsertNode(FibHeapNode *)
This function inserts the created FibNode into the FibHeap in the top circular doubly linked list .

ii)removeMin:
This function removes the current min node (based on weight) present in the Fib Heap.

iii)decreseKey(FibHeapNode *,int):
This fucntion performs the decrease key operation of the Fib Heap .If the weight of any particular node is changed this function update the node and insert in the appropriate position in the Fib Heap

iv)pairwiseCombine

Pairwise combining repeatdley merge the node of the same degree until all the Min tree in Fibonacci Heap has a distinct degree.It Maintain degree table and merge two tree of the same degree until for reach degree number of tree would be less than one
v) cascadeCut (FibHeapNode *)
Parameter – It take 1 MinFibnode and do cascading.If y is a root list no more cascading required, for y to be in root list it parent would be NULL.If child cut happens first time no more cut and cascading required.Just mark the node to indicate child cut is true If y is marked y is just lost the second child, y is cut from here and added to the root list and cascading cut is called to the parent of the node.
vi)cutNode(FibHeapNode *,FibHeapNode *)
Parameter- MinFibnode x and y
This function Remove x from the child list of y and then decrease degree of y
vii)JoinFibHeap(FibHeapNode *,FibHeapNode *)
This Function take two Fibonacci node and make bigger node is a subtree of the smaller node.

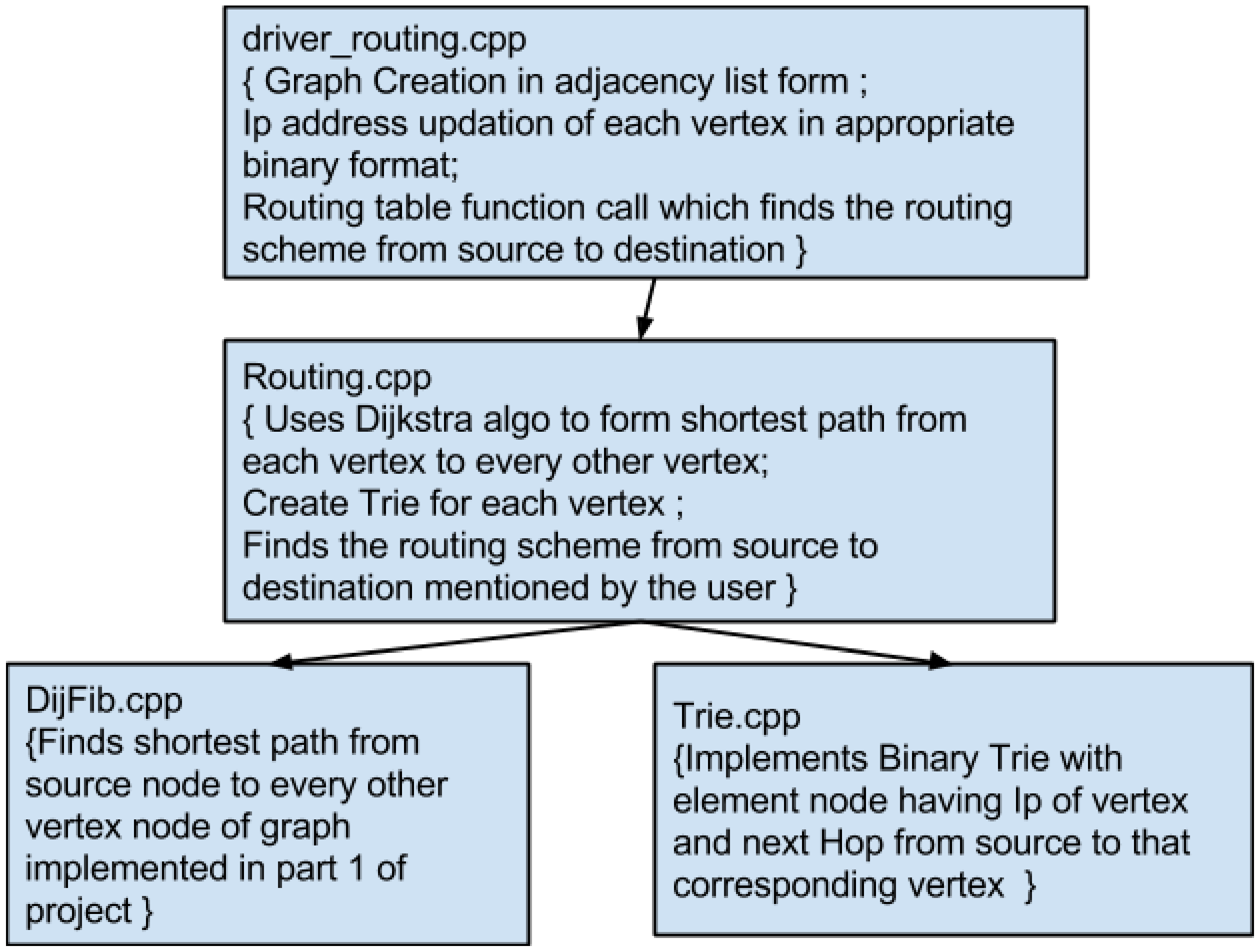
C) DijFib.cpp/DijFib.h

This file is the driver program for implementing the the standard Dijkstra algorithm based on Fibonacci heap .It stores the result in the following structure .

```
typedef struct res
{
vector<int>Dis;// stores the shortest  distance from the source node to the destination node .The destination node is represented as index of the vector .
vector<int>Par;//stores the parent node of the shortest path from source node..The destination node is represented as index of the vector .
}SSP;
```

i)DjkstraFibHeap(Graph *,int s);
This function takes graph pointer and source vertex as an input and after applying dijkstra algorithm returns the shortest path from source vertex to all other vertices present in the graph in the SSP structure as described above .

B)Part 2 :


driver_routing.cpp
{ Graph Creation in adjacency list form ;
Ip address updation of each vertex in appropriate binary format;
Routing table function call which finds the routing scheme from source to destination }
Routing.cpp
{ Uses Dijkstra algo to form shortest path from each vertex to every other vertex;
Create Trie for each vertex ;
Finds the routing scheme from source to destination mentioned by the user }
DijFib.cpp
{Finds shortest path from source node to every other vertex node of graph implemented in part 1 of project }
Trie.cpp
{Implements Binary Trie with element node having Ip of vertex and next Hop from source to that corresponding vertex }

This part starts with driver_routing.cpp file which takes 4 input parameters as in argument from the console .First one is graph file ,then Ip address information of vertices,the source and then destination for which routing scheme has to be generated .The file first calls the Graph generation function which takes the graph information from the file and created adjacency list representation of graph .After that it calls the routing function which does all the important tasks of creation of routing scheme between sources to destination using binary trie .The high level design of this project has been mentioned above .

Detail Description of File and Function:

A)Trie.cpp/Trie.h

These are the most important files for implementation of part 2 of project.These files handles all the important functions and structures of managing Binary Tries.Following are all the important structures and Functions for Binary Tries.

```
typedef struct BranchNode {
        BranchNode *l, *r;
        BranchNode() {
                l = r =  NULL;
        }
}BNode;


typedef struct TrieNodeTag : public BNode
{
        string key;   //Contains Ip address of the node.
        int data;     //Contains the vertex id of the next Hop from the source vertex
        TrieNodeTag();  //Constructor of this structure
        TrieNodeTag(string dIp, int nextNode) {   //Creates of TrieNode Element with Ip and vertex number .
                key = dIp;
                data = nextNode;
        }

}TNode;

/****Structure for storing the result of the routing scheme
typedef struct Trie_Result {
        string IpPrefix;
        int nextHop;
}TrieResult;

class Trie {
private :
        BNode *head;     //root pointer of this trie
public :
        Trie();
        void InsertPair(string destIp, int nextNode);
        TrieResult Search(string destIp);
        BNode* Optimize(BNode *p);
        void OptimizePath();}
```

i) InsertPair(string ,nextNode)

Parameter : Ip address of the destination vertex . Next Vertex present in the shortest path to reach that destination .

This function insert the ip value ,next hop value pair in the binary trie .

ii)Search(string)
Parameter :Ip address of the destination node which is to be searched in the Trie
This function searches the input Ip in the corresponding trie and update the longest matched prefix IP in the TrieResult

iii)Optimize(BNode *)
Parameter :Takes BNode pointer as an argument .For the first call the BNode is the head pointer of the corresponding Trie .This function optimizes the trie based on post order traversal of the Trie .It removes the subTries whose Next Hop is same for all destination .

iv)OptimizePath()
Calls the Optimize function passing head of the Trie as an argument .

B)Routing.cpp/Routing.h

i)routing_table(Graph *,int ,int)
Parameter :Graph in adjacency list form have IP address also present for each vertices ,source and destination vertices for which routing scheme to be generated .
This function builds tries for each vertices of the graph using Dijkstra single source shortest past implemented in part 1 .After that this function call the Search function of Trie to search the IP address of the destination node and updates the result in TrieResult structure .

4) Result:

a) Part 1:

After executing the ssp binary file with giving all the required parameters output will come in the following format on the console :
--Shortest distance value (This represents the shortes distance value from the source to destination node given as an input .
--**The path from source to destination** (In the new line there will be many numbers .These numbers represent the path from source to destination to achieve the above shortest distance.

b) Part 2:

After executing the routing binary file with giving all the required parameters output will come in the following format on the console :

--Shortest distance value (This represents the shortes distance value from the source to destination node given as an input .

---Array of numbers in binary format .(These binary numbers represent the longest prefix match of IP address of the next Hop until we reach the destination node mentioned in the console input.

References:

[1].Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford (2009) [1990]. Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill. ISBN 0-262-03384-4.
[2].http://en.wikipedia.org/wiki/Fibonacci_heap
[3].http://www.sourcetricks.com/2011/06/c-tries.html#.VS7Vv3VGh5R
[4].http://opendatastructures.org/ods-java/13_1_BinaryTrie_digital_sea.html