Recursive Shortest Route Algorithm Using Abstract Data Type, Graph

O.E. Oguike
Department of Computer Science
University of Nigeria
Nsukka, Nigeria
osondu.oguike@unn.edu.ng

ABSTRACT
The shortest route algorithm is an important algorithm in the mathematical sciences because it determines the most efficient route between two nodes in a network. Among the variants of the shortest route algorithm that are available, different data structures can be used to define the algorithm. The abstract data type, graph is one of the highly structured data structures or abstract data type that has important applications to mathematical and computer sciences. Mathematically, it can be defined as a collection of two sets, the set of nodes and the set or table/relation of edges. Recursive algorithm is very efficient when implemented, especially when the recursive case has one recursive call, it is also very structured when implemented. This paper presents a recursive design of the Dijkstra’s shortest route algorithm, using the operations of abstract data type, graph.

Keywords- Graph, abstract data type, shortest route algorithm, network, dynamic data structure, recursive algorithm.

1. INTRODUCTION
The shortest route algorithm is an important algorithm in mathematical sciences and computer science in particular. This is because it determines the most efficient route in communication network, water pipeline network, oil pipeline network, road network etc. Variants of the shortest route algorithm exist, which are, Dijkstra’s shortest route algorithm, Floyd’s shortest route algorithm etc. Data structures, like array, list and graph can be used to design the shortest route algorithm. Using the graph data structure to design the shortest route algorithm, graph can be considered as adjacency matrix or adjacency list [1]. In this paper, graph will be considered, based on its mathematical definition. Mathematically, graph is defined as a collection of two sets, the sets of nodes and the table or relation of edges [2].

Based on this mathematical definition of graph, the abstract data types, set and table/relation will be used to form the abstract data type, graph. The underlying ADT that will be used for the set is list, while the underlying ADTs that will be used for the table are set and list. This paper will apply the shortest route algorithm to GSM telecommunication network, with satellite distance between the various nodes of the network [6]. Suppose we have a GSM telecommunication network, as shown in figure 1, with appropriate satellite distance between the nodes of the network.
Table 1: Table of Triple (Edges with Distance Weight)

<table>
<thead>
<tr>
<th>SNode</th>
<th>DNode</th>
<th>Distance</th>
<th>SNode</th>
<th>DNode</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>700</td>
<td>2</td>
<td>6</td>
<td>400</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>700</td>
<td>6</td>
<td>2</td>
<td>400</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>200</td>
<td>3</td>
<td>5</td>
<td>600</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>200</td>
<td>5</td>
<td>3</td>
<td>600</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>300</td>
<td>4</td>
<td>6</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>300</td>
<td>6</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>700</td>
<td>4</td>
<td>5</td>
<td>300</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>700</td>
<td>5</td>
<td>4</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>200</td>
<td>6</td>
<td>5</td>
<td>500</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>200</td>
<td>5</td>
<td>6</td>
<td>500</td>
</tr>
</tbody>
</table>

The abstract data type set will be used to represent the set of nodes of the network. The algorithms for the operations of the ADT, set are emptyset, isemptyset, insertset, removeset and memberset. They will be designed using the algorithms for the operations of the underlying ADT, list, which include, emptylist, isemptystlist, conslist, headlist and taillist. Furthermore, the abstract data type, table of edges will have the following attributes, source node, destination node and satellite distance. The primary keys of the table/relation will be the source node and the destination node. The algorithms for the operations of the abstract data type, table of edges/triple are emptytabletriple, isemptytabletriple, membertabletriple, locatedistance, removetabletriple and insertabletriple. They will be designed using the following algorithms for the operations of set of triple, emptysettriple, isemptysttriple, insertsttriple, removesettriple and membersettriple.

In a similar manner, the algorithms for the operations of set of triple will be designed using the following algorithms for the operations of the ADT, list of triple, emptylisttriple, isemptylsttriple, conslisttriple, sourcehead, destinationhead, distancehead and tailist. The algorithms for the operations of graph are emptygraph, addnode, addedge, isemptygraph, contains, isadjacent, deleternode, deleteredge. They will be designed using the algorithms for the operations of the set of nodes and table of edges/triple. Finally, relevant algorithms that will be used in the recursive Dijkstra’s shortest route algorithm will be designed using the algorithms for the operations of graph. The hierarchy of the underlying ADTs used to represent graph can be shown in figure 2.

2. SURVEY OF RELATED LITERATURE

According to [4], Abstract data type is “a kind of data abstraction where a type’s internal form is hidden behind a set of access functions; values of the type are created and inspected only by calls to the access functions”. Applications of graph algorithm have been the focus of some literature, for different configurations of network. In [2], the authors reviewed the use of fuzzy logic in optimal path selection for an ad hoc network. The authors in [3] outlined the various applications of Graph Theory to Computer Science. According to them, “the major role of graph theory in computer science is in the development of graph algorithms”, which can be used to solve problems that have been modeled using graph, such problem, according to them includes: shortest path algorithm in a network, finding minimal spanning tree, finding graph planarity etc. They also identified some of the languages that can be used to support graph theory concept.

The author in [5] identified other operations of graph, e.g. difference graph, union graph, and intersection graph. These operations, according to him can be used for other applications. Furthermore, the author in [8] presented a version of Dijkstra’s shortest route algorithm, which was based on temporary and permanent labeling of the nodes, until, if possible all the nodes of the graph have been labeled permanently. In [1], [7], presentations of some of the abstract data types, like, list, set, relations and graph were made, with relevant applications. The authors in [9] traced the history of graph theory, according to them; Graph theory, as an area of study within mathematics began with a paper published by Leonhard Euler in 1736. According to them, the study of graph algorithm as an area with computing is relatively new. They also presented two ways of presenting graph, which are adjacency matrix and adjacency list. The authors in [10] suggested a modification of the Dijkstra’s shortest route algorithm, they suggested a multi parameter Dijkstra’s shortest route algorithm that will consider other factors like distance, time and congestion of the edge during the computation of the shortest route.
Furthermore, in [11], the authors used priority queue and C programming language to implement and evaluate the performance of Dijkstra’s shortest route algorithm. Comparative analysis of all the algorithms that solve the shortest route problem were made by the authors in [12], while in [13], the authors surveyed the various applications of shortest path routing, which include, transportation GIS, network analysis, operations research, artificial intelligence and robotics.

3. DESIGN OF ALGORITHMS FOR THE OPERATIONS OF ABSTRACT DATA TYPE, SET OF NODES

The algorithms for the various operations of the abstract data type, set of nodes can be designed, using the various algorithms for the operations of the underlying abstract data type, list. First, the design of the algorithms for the operations of the abstract data type, list will be considered. The structure of the abstract data type list can be defined in a java class called mylist as follows:

```java
Int data
Mylist link
```

The algorithms for the operations of the abstract data type, list can be defined as follows.

**Mylist emptylist()**
1. Determine emptylist
   1.1 emptylist = null
2. Display emptylist
   The algorithm, emptylist returns the null list and it takes nothing as parameter.

**Boolean isemptylist(mylist l)**
1. Read l
2. Determine isemptylist
   2.1 IF l = null THEN
   2.1.1 isemptylist = true
   ELSE
   2.1.2 isemptylist = false
3. Display isemptylist
   The algorithm, isemptylist takes a list as parameter and it returns true if the list is empty, otherwise it returns false.

**int headlist(mylist l)**
1. Read l
2. Determine headlist
   2.1 IF isemptylist(l) THEN
   2.1.1 headlist = "No head for empty list"
   ELSE
   2.1.2 headlist = l.data
3. Display headlist
   The algorithm, headlist takes a list as parameter and it returns the first integer (node) on the left; otherwise, it returns an appropriate error message.

**Mylist taillist(mylist l)**
1. Read l
2. Determine taillist
   2.1 IF isemptylist(l) THEN
   2.1.1 taillist = l
   ELSE
   2.1.2 taillist = l.link
3. Display taillist
   The algorithm, taillist takes a list of integers (nodes) as parameter. If the list is not empty, it removes the integer number on the left hand side of the list, and returns the remaining list; otherwise, it returns the given list.

**Mylist conslist(int x, mylist l)**
1. Read data
   1.1 Read x
   1.2 Read l
2. Determine conslist
   2.1 Declare and allocate temp of the type mylist
   2.2 Temp.data = x
   2.3 Temp.link = l
3. Display conslist
   The algorithm, conslist takes two parameters, an integer and a list of integers. The algorithm makes the integer to be part of list of integers. Having designed the algorithms for the operations of the list abstract data type, these algorithms will be used to design the algorithms for the operations of the abstract data type, set. The following are the algorithms for the operations of the abstract data type, set.

**Myset emptyset()**
1. emptyset = emptylist()
2. Display emptyset
   The algorithm, emptyset takes nothing as parameter and it returns an emptyset.

**Boolean isemptyset(myset s)**
1. Read s
2. Determine isemptyset
   2.1 isemptyset = isemptylist(s)
3. Display isemptyset
   The algorithm, isemptyset takes a set as parameter, and it returns true if the set is empty, otherwise it returns false.

**Boolean memberset(integer x, myset s)**
1. Read data
   1.1 Read x
   1.2 Read s
2. Determine memberset
   2.1 IF isemptyset(s) THEN
   2.1.1 memberset = false
   ELSE
   2.1.2 memberset = memberset(x, taillist(s))
3. Display memberset
   The algorithm, memberset takes two parameters, an integer and a list of integers. If the list is not empty, it removes the integer number on the left hand side of the list, and returns the remaining list; otherwise, it returns the given list.
The algorithm, memberset takes an integer and a set of integers as parameters and it returns true if the integer is a member of the set of integers, otherwise, it returns false.

```
Myset insertset(int x, myset s)
1 Read data
  1.1 Read x
  1.2 Read s
2 Determine insertset
  2.1 IF memberset(x, s) THEN
  2.1.1 insertset = s
  ELSE
  2.1.2 insertset = conslist(x, s)
3 Display insertset
```

The algorithm, insertset takes an integer and a set as parameters and it inserts the integer into the set provided that the integer is not a member of the set.

```
Myset removeset(int x, myset s)
1 Read data
  1.1 Read x
  1.2 Read s
2 Determine removeset
  2.1 IF memberset(x, s) THEN
  2.1.1 IF (x = headlist(s)) THEN
  2.1.1.1 removeset = taillist(s)
  ELSE
  2.1.1.2 removeset = insertset(headlist(s), removeset(x, taillist(s)))
  ELSE
  2.1.2 removeset = s
3 Display removeset
```

The algorithm, removeset takes an integer and a set as parameters and it removes the integer from the set, if it is a member, otherwise, nothing will be removed.

4. DESIGN OF ALGORITHM FOR THE OPERATIONS OF ABSTRACT DATA TYPE, TABLE OF EDGES

The algorithms for the operations of the table of edges can be designed using the algorithms for operations of the underlying abstract data type, set of triples. While the algorithms for the operations of the abstract data type, set of triple can be designed using algorithms for the operations of the abstract data type, list of triple. We begin by designing the algorithms for the operations of list of triples. The structure of the list of triples can be defined in a java class called listriple as follows:

```
int source
int destination
double distance
listriple link
```

The following are the design of algorithms for the operations of the list of triple.

```
listriple emptylistriple()
1 Determine emptylistriple
  1.1 emptylistriple = null
2 Display emptylistriple
```

The algorithm takes nothing as parameter and it returns an empty list of triple.

```
boolean isemptylistriple(listriple l)
1 Read l
2 Determine isemptylistriple
  2.1 IF l = null THEN
  2.1.1 isemptytriple = TRUE
  ELSE
  2.1.2 isemptytriple = FALSE
3 Display isemptylistriple
```

The algorithm, isemptylistriple takes a list of triple as parameter and it returns true if the list of triple is empty, otherwise, it returns false.

```
int sourcehead(listriple l)
1 Read l
2 Determine sourcehead
  2.1 IF isemptylistriple(l) THEN
  2.1.1 sourcehead = "Empty list does not have head"
  ELSE
  2.1.2 sourcehead = l.source
3 Display sourcehead
```

The algorithm, sourcehead takes a list of triple as parameter and it returns an appropriate error message if the list of triple is empty, otherwise, it returns the source attribute of the list of triple.

```
int destinationhead(listriple l)
1 Read l
2 Determine destinationhead
  2.1 IF isemptylistriple(l) THEN
  2.1.1 destinationhead = "Empty list does not have head"
  ELSE
  2.1.2 destinationhead = l.destination
3 Display destinationhead
```

The algorithm, destinationhead takes a list of triple as parameter and it returns an appropriate error message, if the list of triple is empty, otherwise, it returns the destination attribute of the list of triple.

```
Double distancehead(listriple l)
1. Read l
2 Determine distancehead
  2.1 IF isemptylistriple(l) THEN
  2.1.1 distancehead = "Empty list does not have head"
  ELSE
  2.1.2 distancehead = l.distance
3 Display distancehead
```
The algorithm, distancehead takes a list of triple as parameter and it returns an appropriate error message, if the list of triple is empty, otherwise, it returns the distance attribute of the list of triple.

\begin{algorithm}
\caption{Listriple tailistriple(listriple $l$)}
\begin{algorithmic}[1]
\State 1. \textbf{Read} $l$
\State 2. \textbf{Determine tailistriple}
\State 2.1. \textbf{IF} isemptylistriple($l$) \textbf{THEN}
\State 2.1.1. tailistriple = $l$
\State 2.1.2. \textbf{ELSE}
\State 2.1.2. tailistriple = $l$.link
\State 3. \textbf{Display} tailistriple
\end{algorithmic}
\end{algorithm}

The algorithm, tailistriple takes a list of triple, $l$ as parameter and it return $l$, if it is empty, otherwise, it returns a list of triple after taking away the head.

\begin{algorithm}
\caption{listriple conslistriple(int x, int y; double z; listriple $l$)}
\begin{algorithmic}[1]
\State 1. \textbf{Read} data
\State 1.1. \textbf{Read} x
\State 1.2. \textbf{Read} y
\State 1.3. \textbf{Read} z
\State 1.4. \textbf{Read} $l$
\State 2. \textbf{Determine conslistriple}
\State 2.1. \textbf{Declare} and allocate, temp of the type listriple
\State 2.2. Temp.source = x
\State 2.3. Temp.destination = y
\State 2.4. Temp.distance = z
\State 2.5. Temp.link = $l$
\State 2.6. conslistriple = temp
\State 3. \textbf{Display} conslistriple
\end{algorithmic}
\end{algorithm}

The algorithm, conslistriple takes four parameters, which represent the source, destination, distance and the list of triple. The algorithm forms the record of the head and makes it part of the list of triple. These algorithms for the operations of list of triple will be used to design algorithms for the operations of set of triple. The following are the design of algorithms for the operations of set of triple:

\begin{algorithm}
\caption{Setriple insertsetriple(int x; int y; double z; setriple $l$)}
\begin{algorithmic}[1]
\State 1. \textbf{Read} data
\State 1.1. \textbf{Read} x
\State 1.2. \textbf{Read} y
\State 1.3. \textbf{Read} z
\State 1.4. \textbf{Read} $l$
\State 2. \textbf{Determine insertsetriple}
\State 2.1. \textbf{IF} membersetriple(x, y, $l$) \textbf{THEN}
\State 2.1.1. insertsetriple = $l$
\State 2.1.2. \textbf{ELSE}
\State 2.1.2. insertsetriple = conslistriple(x, y, z, $l$)
\State 3. \textbf{Display} insertsetriple
\end{algorithmic}
\end{algorithm}

The algorithm, insertsetriple takes four parameters, which represent the following, source node, destination node, distance and the set of triple. The algorithm uses some of the algorithms for the operations of listriple to determine if the pair of source node and destination nodes is a member of the set of triple. If the pair is a member, it does not insert it into the set of triple, otherwise, it inserts the source node, destination node and the distance into the set of triple.
Setriple removesetriple(int x, y; setriple l)
1 Read data
1.1 Read x
1.2 Read y
1.3 Read l
2 Determine removesetriple
2.1 IF membersetriple(x,y, l) THEN
2.1.1 IF x = sourcehead(l) AND
y = destinationhead(l)
THEN
2.1.1.1 Removesetriple =
tailistriple(l)
ELSE
2.1.1.2 removesetriple =
conslistriple(sourcehead(l),
destinationhead(l),
distancehead(l),
removesetriple(x,
y, tailistriple(l)))
ELSE
2.1.2 removesetriple = l
3 Display removesetriple

The algorithm, removesetriple takes three parameters, which represent the source node, destination node and the set of the triple. The algorithm uses the algorithms for the operations of list of triple and set of triple to determine if the pair of source node and destination node is a member of the set of triple, if it is, it looks for the pair of source node and destination node and removes it from the set of triple, otherwise, nothing will be removed. The algorithms for the operations of the abstract data type, tabletriple can be defined using the algorithms for the operations of the underlying abstract data type, setriple. The primary keys of the abstract data type, tabletriple are source and destination fields. The following are the design of algorithms for the operations of the abstract data type, table of triple.

tabletriple emptytabletriple()
1 Determine emptytabletriple
1.1 emptytabletriple = emptysetriple()
2 Display emptytabletriple

The algorithm, emptytabletriple takes nothing as parameter and it uses the algorithm for the operation of the abstract data type, setriple to return an empty table of triple.

Boolean membertabletriple(int x; int y; tabletriple l)
1 Read data
1.1 Read x
1.2 Read y
1.3 Read l
2 Determine membertabletriple
2.1 membertabletriple =
membersetriple(x, y, l)
3 Display membertabletriple

The algorithm, membertabletriple takes three parameters, which represent the source node, destination node and a table of triple. The algorithm uses the algorithm for the operation of setriple to return true if the pair of the source and destination nodes is in the table of triple, otherwise, it returns false.

Double locatedistance(int x; int y; tabletriple l)
1 Read data
1.1 Read x
1.2 Read y
1.3 Read l
2 Determine locatedistance
2.1 IF membertabletriple(x, y, l) THEN
2.1.1 IF x = sourcehead(l) AND
y = destinationhead(l)
THEN
2.1.1.1 locatedistance =
distancehead(l)
ELSE
2.1.1.2 locatedistance =
locatedistance(x,
y, tailistriple(l))
ELSE
2.1.2 locatedistance = “The pair of primary key does not exist”
3 Display locatedistance

The algorithm, locatedistance takes three parameters, which represent the source, destination nodes, and a table of triple. The algorithm uses the algorithms for the operations of tabletriple to determine if the pair that forms the primary key is a member of the primary key of the table of triple. If it is, it looks for the primary key and returns the corresponding distance attribute; otherwise, it returns an appropriate error message.
Tabletriple insertabletriple(int x; int y; double z; tabletriple l)
1 Read data
1.1 Read x
1.2 Read y
1.3 Read z
1.4 Read l
2 Determine insertabletriple
2.1 insertabletriple = insertsetriple(x, y, z, l)
3 Display insertabletriple

The algorithm, insertabletriple takes four parameters, which represent the source and destination nodes, the distance attribute and a table of triple. The algorithm uses the algorithm for the operation of setriple to insert the source, destination and distance attributes into the table of triple.

Tabletriple removetabletriple(int x;int y; tabletriple l)
1 Read data
1.1 Read x
1.2 Read y
1.3 Read l
2 Determine removetabletriple
2.1 removetabletriple = removesetriple(x, y, l)
3 Display removetabletriple

The algorithm, removetabletriple takes three parameters, which represent the source and destination nodes, and a table of triple. The algorithm uses the algorithm for the operation of setriple to remove the record that has the source and destination nodes as primary key, from the table of triple. The following algorithms will be used to sort the abstract data type, tabletriple in ascending order of distance.

Tabletriple sortabletriple(tabletriple t)
1 Read t
2 Determine sortabletriple
2.1 IF isemptytabletriple(t) THEN
2.1.1 sortabletriple = t
ELSE
2.1.2 sortabletriple = insertordertable(sourcehead(t), destinationhead(t), distancehead(t), sortabletriple(tailistriple(t)))
3 Display sortabletriple

5. DESIGN OF ALGORITHMS FOR THE OPERATIONS OF ABSTRACT DATA TYPE, GRAPH

The algorithms for the operations of the abstract data type, graph can be designed using the algorithms for the operations of the abstract data types, set of nodes and table of edges/triple. Since some of the algorithms return a particular abstract data type, therefore some of the algorithms for the operations of the abstract data type, graph will be considered as two different algorithms. The following are the design of algorithms for the operations of abstract data type, graph.

myset emptygraph1()
1 Determine emptygraph1
1.1 emptygraph1 = emptyset()
2 Display emptygraph1
Tabletriple emptygraph2()
1 Determine emptygraph2
1.1 emptygraph2 = emptytabletriple()
2 Display emptygraph2

The algorithm, emptygraph has been designed as two algorithms, emptygraph1 and emptygraph2. The algorithm, emptygraph1 takes nothing as parameter and it uses the algorithm for the operation of the abstract data type, set of nodes to return an empty set of node. Similarly, the algorithm, emptygraph2 takes nothing as parameter, but it uses the algorithm for the operation of the abstract data type, table of edges to return an empty table of edges.
The algorithm, addnode has been designed as two algorithms, addnode1 and addnode2. Both of them take three parameters, which are the node, x to be added into the set of nodes, s, and the table of edges/triple, t. Addnode1 uses the algorithm for the operation of the abstract data type, set of nodes to add the node into the set of nodes, and it returns a set of nodes. On the other hand, addnode2 returns the table of edges/triple that was passed to it as parameter.

The algorithm, addedge has been designed as two algorithms, addedge1 and addedge2. Both of them take five parameters, which are two integer numbers, x and y that form the new edge and the distance z between them, which will be added into the table of edges, t and a set of node, s. Addedge1 returns the set of nodes, s that was passed to it as parameter, while addedge2 uses the algorithm for the operations of set of nodes to confirm if each of the two nodes that form the edges is a member of the set of nodes. If each of them is a member, it uses the algorithm for the operation of table of edges to insert the triple data, x,y and z into the table of triple t, otherwise, they will not be inserted.

The algorithm, isemptygraph takes two parameters, a set of nodes and a table of triple. The algorithm uses the algorithms for the operations of set of nodes and table of edges to determine if the graph is empty. The graph can only be empty if the set of nodes and the table of edges are empty; otherwise, the graph is not empty.
The algorithm, which is called, contains takes a graph and a node as parameters. The graph consists of a set of nodes and a table of edges. The algorithm uses the algorithm for the operations of set of nodes to determine if the node x is a member of the set of nodes, l.

Boolean isadjacent(int x; int y; myset s; tabletriple t)

1 Read data
  1.1 Read x
  1.2 Read y
  1.3 Read s
  1.4 Read t

2 Determine isadjacent
  2.1 IF membertable(x,y,t) THEN
  2.1.1 isadjacent = true
  ELSE
  2.1.2 isadjacent = false

3 Display isadjacent

The algorithm, isadjacent takes four parameters, which are two nodes, x and y, a set of nodes, s and a table of edges, t. The algorithm uses algorithm for the operation of table of edges to determine if the two nodes x and y are adjacent nodes of the graph.

Myset deletenode1(int x, myset s, tabletriple t)

1 Read data
  1.1 Read x
  1.2 Read s
  1.3 Read t

2 Determine deletenode1
  2.1 IF membertable(x,t) THEN
  2.1.1 deletenode1 = s
  ELSE
  2.1.2 deletenode1 = removeset(x,s)

3 Display deletenode1

The algorithm, deletenode1 has been designed as two algorithms, deletenode1 and deletenode2. Each of them takes three parameters, which are the node x to be deleted from the set of nodes, s and the table of edges, t. The algorithm, deletenode1 first ensures that the node to be deleted is not used to form an edge. Once it confirms this, it removes the node from the set of nodes, otherwise, it does not delete the node from the set of nodes, afterwards, it returns a set of nodes. The algorithm, membertable, which is designed below is used to confirm if the node is not used to form an edge before deleting. The algorithm, deletenode2 returns a table of edges.

Boolean membertable(int x, tabletriple t)

1 Read data
  1.1 Read x
  1.2 Read t

2 Determine membertable
  2.1 IF isemptypabletriple(t) THEN
  2.1.1 membertable = false
  ELSE
  2.1.2 IF x = sourcehead(t) OR x = destinationhead(t) THEN
  2.1.2.1 membertable = true
  ELSE
  2.1.2.2 membertable = membertable(x,tailistriple(t))

3 Display membertable

The algorithm, membertable, takes two parameters, a node x, which is an integer and tabletriple t, which is a table of triple. The algorithm returns true if the node x is used to form an edge, otherwise, it returns false.

Myset deletedge1(int x, int y, myset s, tabletriple t)

1 Read data
  1.1 Read x
  1.2 Read y
  1.3 Read s
  1.4 Read t

2 Determine deletedge1
  2.1 deletedge1 = s

3 Display deletedge1

Tabletriple deletedge2(int x, int y, myset s, tabletriple t)

1 Read data
  1.1 Read x
  1.2 Read y
  1.3 Read s
  1.4 Read t

2 Determine deletedge2
  2.1 deletedge2 = removeset(x,y,t)

3 Display deletedge2

The algorithm, deletedge has been designed as two algorithms, deletedge1 and deletedge2. Both of them take four parameters, which are, two integer numbers that form the edge to be deleted, a set of node and a table of edges. The algorithm, deletedge1 returns the set of nodes, while the algorithm, deletedge2 uses the algorithm for the operation of abstract data type, tabletriple to remove the two integer numbers that form an edge from the table of edges, afterwards, it returns a table of edges.

6 DESIGN OF RECURSIVE DIJKSTRA’S SHORTEST ROUTE ALGORITHM

The algorithms for the operations of the abstract data type, graph will be used as the underlying abstract data type to design the recursive Dijkstra’s shortest route algorithm.
The following algorithms will be used to design the recursive Dijkstra’s shortest route algorithm.

**tabletriple neibourtable(int x, myset s, tabletriple t, tabletriple p)**

1. Read data
   1.1 Read x
   1.2 Read s
   1.3 Read t
   1.4 Read p
2. Determine neibourtable
   2.1 IF isemptyttabletriple(t) THEN
      2.1.1 neibourtable= emptyttabletriple()
   ELSE
      2.1.2 IF (x = sourcehead(t)) 
      AND (NOT memberperm(destinationhead(t), p)) THEN
         2.1.2.1 neibourtable =
         inserttabletriple(sourcehead(s),
         destinationhead(s),
         distancehead(t),
         neibourtable(x, s, tailistriple(t),
         p))
      ELSE
         2.1.2.1 neibourtable =
         neibourtable(x, s, tailistriple(t), p)
3. Display neibourtable

The algorithm, neibourtable determines all the neighbours of a node and put them in a table. It uses an algorithm called, memberperm, which helps to ensure that the neighbours of the node has not been labeled permanently.

**tabletriple templabels(double x, tabletriple t, tabletriple p)**

1. Read Data
   1.1 Read x
   1.2 Read t
   1.3 Read p
2. Determine templabels
   2.1 IF isemptyttabletriple(t) THEN
      2.1.1 templabels =
      emptyttabletriple()
   ELSE
      2.1.2 IF (x = sourcehead(t))
      THEN
         2.1.2.1 templabels =
         templabels(sourcehead(t),
         destinationhead(t),
         distancehead(t)+x,
         templabels(x, tailistriple(t), p))
      ELSE
         2.1.2.2 templabels =
         inserttabletriple(sourcehead(t),
         destinationhead(t),
         distancehead(t)+x,
         templabels(x, tailistriple(t), p))

3. Display templabels

The algorithm, templabels labels as temporary labels all the neighbours of the current source node that have not been labelled as permanent labels. The above algorithm uses the algorithm, destinationmember, which is defined below:

**Boolean destinationmember(int x, listriple t)**

1. Read Data
   1.1 Read x
   1.2 Read t
2. Determine destinationmember
   2.1 IF (isemptyttabletriple(t)) THEN
      2.1.1 destinationmember =
      FALSE
   ELSE
      2.1.2 IF (x = destinationhead(t))
      THEN
         2.1.2.1 destinationmember =
         TRUE
      ELSE
         2.1.2.2 destinationmember =
         destinationmember(x, tailistriple(t))

3. Display destinationmember

The recursive Dijkstra’s shortest route algorithm will use another algorithm, called tabletripleunion, this algorithm takes two tables of triple and joins them together, and the algorithm is defined below:

**Tabletriple tabletripleunion(tabletriple t1, tabletriple t2)**

1. Read Data
   1.1 Read t1
   1.2 Read t2
2. Determine tabletripleunion
   2.1 IF (isemptyttabletriple(t1)) THEN
      2.1.1 tabletripleunion =
      t1
   ELSE
      2.1.2 tabletripleunion =
      inserttabletriple(sourcehead(t1),
      destinationhead(t1),
      distancehead(t1),
      tabletripleunion(tailistriple(t1), t2));
3. Display tabletripleunion
These algorithms can be used to design the recursive Dijkstra’s shortest route algorithm, which is designed below.

```java
tabletriple shortestroute(int sourcenode, tabletriple temp, tabletriple perm, myset s, tabletriple t)
1. Read Data
   1.1 Read sourcenode
   1.2 Read temp
   1.3 Read perm
   1.4 Read s
   1.5 Read t
2. Determine shortestroute
   2.1 IF (isemptytabletriple(temp))
      2.1.1 shortestroute = perm;
   ELSE
      2.1.2 newsourcenode = destinationhead(sortabletriple(temp));
      2.1.3 pivotdistance = distancehead(sortabletriple(temp));
      2.1.4 sampletemp = removetabletriple(sourcehead(sortabletriple(temp)),
                                              destinationhead(sortabletriple(temp)), temp);
      2.1.5 sampleneighbours = null;
      2.1.6 samplenlabels = neighbourtable(newsourcehead(sampletemp), s, t);
      2.1.7 sampletemp = tabletripleunion(sampletemp1, sampletemp);
      2.1.8 sampleperm = insertabletriple(sourcehead(sortabletriple(sampletemp)),
                                         destinationhead(sortabletriple(sampletemp)),
                                         distancehead(sortabletriple(sampletemp)), perm);
      2.1.9 shortestroute = shortestroute(n, sampletemp, sampleperm, s, t);
3. Displaysampleperm
```

7. IMPLEMENTATION OF THE ALGORITHMS

Java programming language was used to implement all the algorithms for the operations of the following abstract data types, mylist, myset, listriple, setriple, tabletriple and graph. Each of these abstract data types was implemented in a Java class, therefore, the class hierarchy is similar to the hierarchy of the abstract data type, which was shown in figure 2. Furthermore, all the algorithms that the recursive Dijkstra’s shortest route algorithm used, including the recursive Dijkstra’s shortest route algorithm were implemented, and a test program was written to test the implemented algorithms. The test program used the implemented algorithms to construct a network as a set of nodes and a table of triple, afterwards the test program used the implemented shortest route algorithm and other implemented algorithms to determine the shortest route between a source node and every other nodes of the network, and the corresponding shortest distance between the source node and any other node of the network.

8 RESULT OF THE IMPLEMENTED ALGORITHM

The test program has been tested and for any source node of the network, it produces the shortest route between the source node and any other node of the network with the corresponding distance. The result is presented as a table of triple, (A, B, C), where letter B denotes the destination node, while letter A denotes the sequence node, and letter C denotes the shortest distance from the source node to node B. Therefore, the result of the shortest route from node 1 to any other node of the network diagram shown in figure 1 can be presented as a table shown below in table 2.

<table>
<thead>
<tr>
<th>Sequence Node</th>
<th>Destination Node</th>
<th>Shortest Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>800</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>800</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>700</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>500</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>200</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Result of the Implemented Algorithm
Figure 3: Shortest route from node 1 to any other node

The result, which is shown in table 2 can be represented in the network diagram in figure 3. The colored route shows the shortest route from node 1 to any of the destination node.

9. CONCLUSION

This paper has been able to use a novel approach to represent graph as a collection of two sets, the set of nodes and the table/relation of triple (edges). The paper has designed novel algorithms for each of the operations of the abstract data types that the graph ADT uses. Furthermore, the recursive Dijkstra’s shortest route algorithm has been designed and implemented using the algorithms for the operations of the ADT, graph, and other ADTs, mylist, myset, listriple, setriple and tabletriple. All the algorithms have been implemented using Java programming language, and a test program has been written to test the implemented algorithms. The result of the test program shows that for a given graph, the shortest route from a given source node to any other node in the network will be determined.

REFERENCES


