Journal of the Nigerian Association of Mathematical Physics,
Volume 8 (*November 2004*).

On the application of Dijkstra's algorithms in solving the GSM Network problem

[++]A.W. Gbolagade[1], R.K. Odunaike[2], Z. O. Ogunwobi[1],
A. U. Rufai[1] and K.A. Gbolagade[1].
[1]Department of Mathematical Sciences, Olabisi Onabanjo University, Ago-Iwoye. Ogun State, Nigeria
[2]Department of Physics, Olabisi Onabanjo University, Ago-Iwoye, Ogun State, Nigeria

Abstract.

This paper discusses the transportation problem that arises when two people at different locations are communicating themselves. The authors coded the Dijkstra's algorithm and attempted to make the Dijkstra's algorithm to work like Flodyd's algorithm. Java programming language was used to develop the program for Dijkstra's algorithm. The results were found to be consistent and reproducible.

Keywords: Dijkstra's algorithm, Flodyd's algorithm, Global System for Mobile Communication (GSM), Java language.

pp 319 - 322

1.0 Introduction.

The concept of GSM, its development, the underlying principles for efficient operations and implementation, opportunities, the problem and experiences of its operators and users in Nigeria were examined [3]. In [4], the economic implications of GSM services and the background of telecoms in Nigeria were discussed. Oguike et al, 2002 [1] compared the difference between Dijkstra and Flodyd's algorithms.

The assertion in [1] motivated the authors to implement the Dijkstra's algorithm and made it work like Flodyd's algorithm.

2.0. Dijkstra's algorithm

One of the standard algorithms for determining the shortest route between any two nodes, towns or villages in a local area network or road network is the Dijkstra's algorithm. It can also be used to determine the most efficient message routes between each, two geographic area in a GSM network.

The Dijkstra's algorithm determines the shortest route between a particular node and any other node in the network. It has a special labelling convention that begins by labelling the nodes temporarily and proceeds until all the nodes have been permanently labelled. The special labelling convention that Dijkstra's algorithm uses is of the form $T[x_1, x_2]x_3$, $P[x_1, x_2]x_3$, where T means a temporary label while P means a permanent label, x_1 is the distance between the source node and the node to be labeled, x_3, while x_2 is the node sequence or the node that will be followed in order to reach the node to be labeled, x_3 is the node you are labelling.

The Dijkstra's algorithm terminates when all the nodes have been permanently labeled, but it begins by labelling the source node with the permanent label, P[0, -], *s*, 0 is the distance from the source node to the node *s*, *P* denotes a permanent label, which, -, means there is no sequence node to the source node.

3.0 Formalization of the Dijkstra's algorithm

We used Dijkstra's algorithm developed in [1]

3.1. Initialization

The first step of the Dijkstra's algorithm was to initialize all the variables and sets that would be used. The following variables and sets were initialized as follows:

3.1.1 The variable, *i*, initialized as the source node

3.1.2 The set C_j, initialized to empty, would be used to determine the nodes that have not been labelled permanently and could be reached directly from the source node. This means C_j is the set of all nodes that have not been labelled permanently and can be reached directly from the nodes, *i.*

3.1.3 The set C_t, will be initialized to empty. This set would be the set of all temporary labelled nodes.

3.1.4 The set C_p, will be initialized to empty, this set will be the set of all permanently labelled nodes.

3.2. Labelling the Source Node

Label the source node as follows: $P[0, -]$ I and insert this label into the permanently labelled set. Therefore, the set permanently labelled node becomes $C_p = \{[0, -]i\}$.

3.3. Determination of set, C_j

Determine the set, C_j, as the set of nodes that have not been labelled permanently and can be reached directly from node *i*.

3.4. **Labelling of the nodes in C_j temporary**

Suppose the set C_j is not empty, and then the nodes in the set, C_j will be labelled temporary. In order to do this, the following steps can be followed:

3.4.1 **Count the elements in set, C_j**

Use the variable, C_{nt} to hold the number of elements in the set, C_j

3.4.2 Initialize count variable to zero

3.4.3 Label a node *j* in the set C_j, temporary as follows. $T[a + dij, I]$, where a is the distance label in the permanent label of node *i* and d_{ij} is the distance between node *i* and node *j*. If this node has been labelled temporary, previously, pick the label with minimum distance.

3.4.4 Make this temporary label of node *j* a number of the set of temporary labelled set. This can be expressed as follows: Insert ($T[a + d_{ij}, i]$ *j*, C_t) provided that it is not in set, C_t, otherwise, do not insert it into C_t.

3.4.5 Increase the Count Variable

Add 1 to this variable, C_{to}, this means that $C_{to} = C_{to} + 1$.

3.4.6 Repeat 3.4.3 – 3.4.5, until C_{to} equal C_{nt}.

3.4.7 Suppose the set, C_j is empty, then do not perform steps 3.4.1 – 3.4.5

3.4.8 Remove it from temporary labelled set C_t, this can be expressed as follows: remove ($T[a, b]$ *j*, C_t).

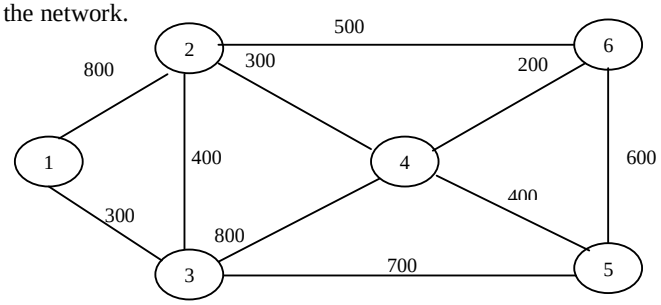
3.4.9 Label it permanently, label the nodes *j* that have picked, permanently. This can be expressed as follows: $P[a, b]j$.

3.5.0 Insert the new permanently labelled node into the set of permanent labels and can be expressed as follows: insert ($P[a, b]$ *c*, C_p)

3.5.1 Let $i = j$ and repeat steps 3.4.3 – 3.5.4, until the temporarily labelled set, C_t becomes empty.

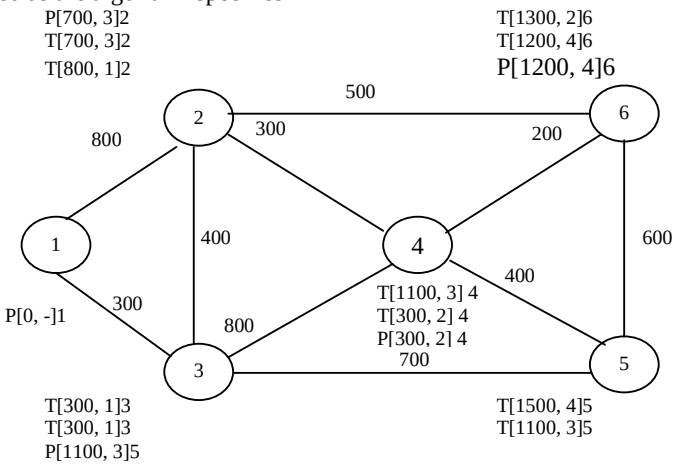
3.5.2 **Illustration**

Using Dijkstra's algorithm to solve a shortest route problem. Given any Tell-mobile phone company services in six geographical areas. The satellite distances (in km) among the six areas are given in the network diagram drawn below. We need to determine the most efficient message routes that should be established between each two areas in the network.



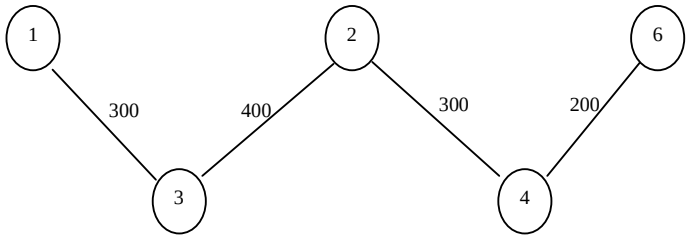
Solution

The Dijkstra's algorithm gives the shortest route between a particular node and any other node in the network. In the light of this, it is necessary to consider particular node and any other node. Taking node 1 as the source node, we proceed as the algorithm specifies



From the diagram, the following permanently labelled set will be obtained: C_p = {[0, -]1, [700, 3], [300, 1]3, [300, 2]4, [1100, 3]5, [1200, 4] 6}. Using the above set of [permanently labelled set, the shortest route between

node 1 and any other node can be determined. Shortest route distance between 1 and 6 is shown below.



The shortest distance is 1200km.

4.0 **The coding of Dijkstra's algorithm to find the shortest routes between cities on a map.**

```
Public interface Routes Map
{
int get Distance (city start, city end);
list getDestination (city city);
using data structures
s, the settled nodes at set
private final set settle Nodes = new Hashet ( );
private Boolean is settled (city v)
{
return settledNodes contains (v);
}
d, the shortest distance list
private final map shortest Distance=new Hashmap ( );
private void setShortest Distance (city city, int distance)
}
public int getshortest Distance (city city)
{
integer d = (integer) shortest Distance get(city);
return (d = = null)? INFINITE_DISTANCE: d int value ( );
π, the predecessors tree
Private final ,ap predecessors= new Hash Map ( );
Private void set predecessor (city a, city b)
{
Predecessors. Put (a, b);
}
public city get predecessors (city city)
{
return (city) predecessors get (city);
}
Q, the unsettled nodes set
Private final comparator shortest Distance comparator=new comparator ( )
}
public int compare (object left, object right)
{
int shortest Distance left=get shortest Distance ((city) left)
int shortest distance Right = get shortest Distance ((city) right);
if (shortest Distance left> shortest Distance Right)
{
return + 1;
}
else if (shortest Distance left < shortest Distance Right)
{
else if (shortest Distance left < shortest Distance Right)
```

```
{
return – 1;

}
else//equal
{
return ((city) left) compare to (right);
}
}
};
Private final shortest unsettled Nodes = new Treeset (shortest Distance comparator);
Private city extract Min ( )
{
city city = (city) unsettled Nodes First ( );
Unsettled Nodes Remove (city);
Return city;
}
```

Please note that the curly brace (}) used in the program is the end statement that is used to match the corresponding (}) begin statement.

5.0 Java idioms used

Java idioms used to flag composition between objects *s*, the settled nodes set. This one is quite straightforward. The java collection includes the set interface, *d*, the shortest distance list.

A straightforward way to implement this in a java is with a map, used to keep the shortest distance value for every node, π, the predecessor tree

The shortest paths are actually stored in reverse order, from destination to source. *Q*, the settled nodes set.

The structure is then looked up for the city with the current shortest distance given by
d ( ).

6.0 Conclusion

We made an attempt to make the Dijkstra's algorithm to work like Flodyd's algorithm first before giving the implementation in Java language. In the program some functions were used. The objective of the program was to compute the shortest route between a particular node and any other node in the network. This was accomplished through designed Java program via the evaluation of the distance between a particular node any the next or any other node. The program shows the applicability of the concept discuss

References

[1] Oguike, O.E. and Agu, M.N (2002): "Proceedings of the computer association of Nigera, vol. 13 ppl 10 – 119.

[2] M. Devargas (1993): "Network Security NCC" Blackwell Ltd. Oxford, England.

[3] B.U. Ezeh (1999): "Object Technology and Enterprise Computing COAN Conference Proceedings. Vol. 10. Pp 110 – 125.

[4] Aliga, P.A., Onianwa, C.U. and Sadiq, I.F. (2003). Proceedings of the Nigeria Computer Society, Vol. 14 pp183 – 198.