

RESEARCH ON THE OPTIMIZATION OF DIJKSTRA'S ALGORITHM AND ITS APPLICATIONS

Arjun RK¹, Pooja Reddy², Shama³, M. Yamuna⁴

^{1, 2, 3} School of Computer Science, Vellore Institute of Technology , Vellore, Tamil Nadu, (India)

⁴ School of Advanced Sciences, Vellore Institute of Technology , Vellore, Tamilnadu, (India)

ABSTRACT

The shortest path problem based on the data structure has become one of the hot research topics in graph theory. The shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. As the basic theory of solving this problem, Dijkstra's algorithm has been widely used in engineering calculations. Aiming at the shortcomings of traditional Dijkstra's algorithm, this paper has proposed an optimization method which has mainly improved the nodes selection of the shortest path and data storage structure and organization. Through comparison and analysis, the improved algorithm has been obtained, which has reduced the storage space and improved the operational efficiency.

Keywords: Applications, Directed Graph, Dijkstra's Algorithm, Shortest Path

I. INTRODUCTION

The shortest-route problem determines a route of minimum weight connecting two specified vertices, source and destination, in a weight graph (digraph) in a transportation network. A good algorithm used to find the shortest path between two known vertices is Dijkstra's algorithm.

Dijkstra's algorithms used for calculating the shortest path, which introduced by the famous Dutch computer scientist Edsger W. Dijkstra, was recognized as the best algorithm that can be applied to get the shortest path from a node to any other nodes. However, with the development of the computer, the scale of the problems is increasing continuously, and meanwhile the use of traditional Dijkstra has increased the space and time complexity. Therefore, this paper has proposed the optimization of algorithm based on the data structure, which has very important significance for improving the efficiency of solving the shortest path algorithm.

II. DIJKSTRA'S ALGORITHM

Given a graph and a source vertex in graph, find shortest paths from source to all vertices in the given graph.

We generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, and other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has minimum distance from source.

Below are the detailed steps used in Dijkstra's algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.

2.1 Pseudo Code

```

dist[s] ← 0                                (distance to source vertex is zero)
for all v ∈ V-{s}
    do dist[v] ← ∞                         (set all other distances to infinity)
S←∅                                         (S, the set of visited vertices is initially empty)
Q←V                                         (Q, the queue initially contains all vertices)
while Q ≠ ∅
    do u ← mindistance(Q,dist)             (while the queue is not empty)
        S←S ∪ {u}                           (select the element of Q with the min. distance)
        for all v ∈ neighbors[u]
            do if dist[v] > dist[u] + w(u, v)   (if new shortest path found)
                  then d[v] ← d[u] + w(u, v)      (set new value of shortest path)
                                         (if desired, add traceback code)
return dist

```

III. DRAWBACKS OF THE TRADITIONAL DIJKSTRA'S ALGORITHM:

In network, the traditional Dijkstra's algorithm has a wide application, but it is not difficult to find that its computation has gradually increased with the network increasing in complexity. If directly applied to calculating the best path of the urban road network, this algorithm will need a great amount of computation, and cannot meet the dynamic needs either. In addition, the adjacency matrix and incidence matrix used in the traditional algorithm to store network data will open up a huge storage space to store a large number of invalid OCJ elements and 0 elements, which is bound to cause huge waste of run time and can also reduce the computational efficiency in Matrix algorithm.

IV. OPTIMIZED DIJKSTRA ALGORITHM:

In view of the problems mentioned above in Dijkstra's algorithm, the selection is optimized for the shortest path node, the data storage and organization in this paper.

4.1. Analysis of Optimization Ideas

4.1.1 The Selection of the Shortest Path Nodes and Nodes Ranking

In Dijkstra's algorithm, each node in the network diagram will be changed from unmarked node to the shortest path node. This change requires scanning a large number of stored disorder unmarked nodes one by one, so that the shortest path of the intermediate node can be achieved. If calculating on the basis of this large amount of data, the calculation speed will be certainly affected. Now we can rank the nodes to be scanned according to the label value, and it needs only one iteration to obtain the eligible nodes, which increases the computational efficiency significantly. Here the heap sort method given by J.W.J Williams is used to select the shortest path node. In algorithm optimization, the original sorting nodes are stored disorderly in the one-dimensional array and then was ordered by heap sort to be changed into a small group, when all the nodes can be stored according to the structure and storage order of complete binary tree. After that, the 0 node stored is the heap element, followed by the left subtree and right subtree. The time complexity ($\log N$) is 0 in the heap adjustment process, where N denotes the number of nodes to be ranked. Compared to the next shortest path node selection from a list or an array with disordered structure, this algorithm can highly improve running speed and reduce running time.

4.1.2 Data Storage and Organization

Storing and organizing mass data by the adjacency matrix needs open $uNxN$ storage space (where N represents the number of nodes) for a large sparse graph, which reduces the storage efficiency and computational efficiency greatly, so the adjacency list can be used to store the network expansion structure to reduce the storage space. The adjacency list is a kind of diagram chain stores structure, in which the node elements can be stored in an array.

4.2. The main idea of the improved Dijkstra algorithm :

Give an array T [] and two sets S and adjacent, in which T [] stores the nodes to be sorted, S is the collection of labelled nodes, and adjacent is the collection of the adjacent points. Under initial conditions,

$$\mathbf{T}[] = \mathbf{adj}[v0], \mathbf{S} = \{v0\}$$

First of all, use heap sort to adjust the array T [] into a small heap, take the heap top node which is also the first element of the array as the intermediate nodes, and then add it into the labeled nodes collection S; Then compare the difference sets indicated collection ($\text{adj}[\text{current}] - S$) with the current shortest path of any node V_i in the changed current collection of adjacent nodes; Then find the difference set between the unit set of adjacent nodes of all nodes in the S collection and S (denoted as $U \text{ adj}[S] - S$), put these nodes into the array T in order, cover the node in the original array, and place a counter i to record the number of nodes.

At last, adjust the first i elements in the array into a small heap based on the shortest path value, and take heap top as the next shortest path node and deposit it into S collection. Repeat this iterative cycle, until all the nodes are stored into the S collection. Now calculate the shortest path D [V] from the starting node v0 to any other node in the graph G according to the Dijkstra algorithm based on data structure, in which the storage structure selected in the graph G is the adjacent list.

V. PSEUDO CODE

```
#define n 1000
Void Dijkstra (Mgraph G, int V0,int T[ ],int D[ ],)
{
    int i = 0,j, F[n] , T[n] ,p, k;
    for ( i = 0; i < GVexnum; i + + )
        {F[i] = 0;D [i] = INFNITE; T[V] = 0; }
    for (p = Gvextices[0].firstarc; p; p = p - > nextarc)
        {K = p-> adjvex;D [k] = p-> info; T[i] =D [k] , i + + ; } F[0] = l;D [0] = 0;
    for ( j = l;j < GVexnum;j + +)
        {HeapAdjust (T[n] , i );
        HeapMin (T, current, distance, path); F[current] = 1; }
    for (p = Gvextices[ current ].firstarc; p; p = p-> nextarc)
        { K = p-> adjvex;
        If( ! F[k]&& D[current] + (p-> info) <D [k] ) D [k] =D [current] + (p-> info) ;
        SearchAdj (S, T[n] , i );
        } }
```

VI. ANALYSIS

From above it is clear that the improved Dijkstra's algorithm not only effectively improves the calculations and running efficiency, but also greatly reduces the number of path in analyzing the shortest path.

The traditional algorithm shows an increasing tendency, while the improved algorithm is not the case. In the early stage of the running, the search path of improved Dijkstra's algorithm shows an increasing tendency, but at the latter stage of reaching the end path, it greatly reduced the number of stored paths, while at the end of the algorithm, the path stored in the memory is only 1 that is the shortest path. Therefore, the improved algorithm releases the storage space, reduces data redundancy, and improves the running time and efficiency.

VII. EXAMPLE APPLICATION BUILT USING DEV C++

AIM: Program to find the estimated delivery time of a shipment from a source city to a destination city in a city graph network.

The program takes as input a file which contains the city names and the distance between them in format as shown and the source and destination cities of the shipment.

By Dijkstra's algorithm, the program calculates the shortest path between the 2 cities in the network and displays the shortest distance, the estimated delivery time and the route to be taken to achieve the same.

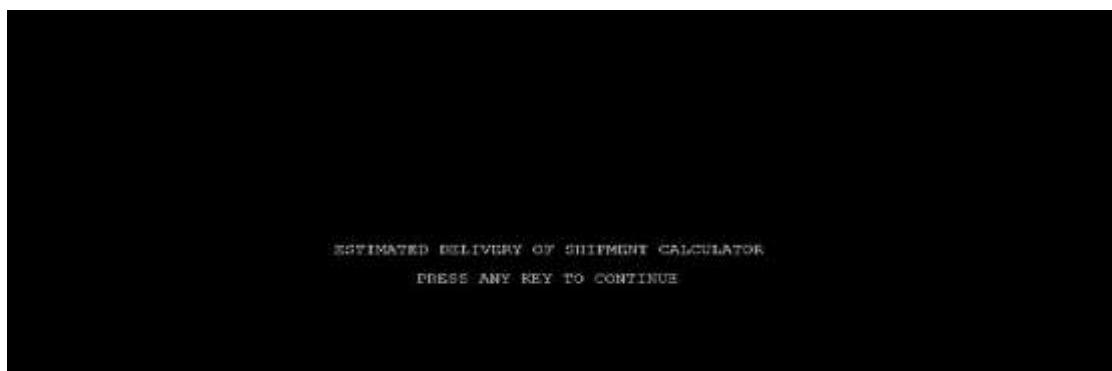
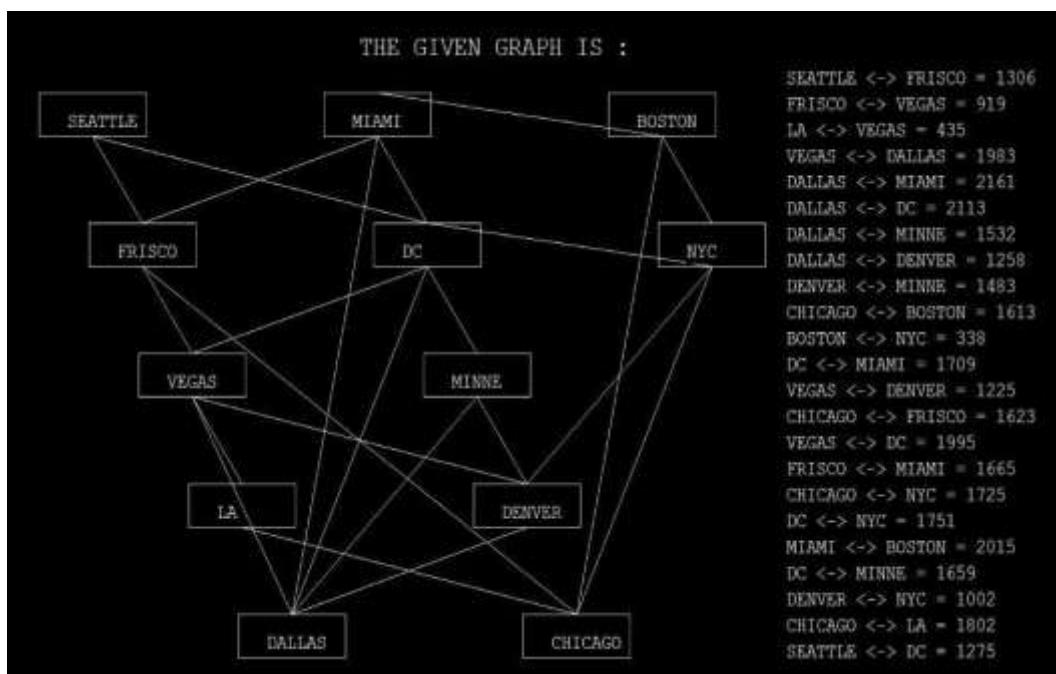


Figure 1 Graphic Window Representing the Output after Compilation of the Above Example

```
Enter the file that contains the graph <graph1.txt and graph2.txt are default>
graph2.txt
The Cities given in the graph are
SEATTLE
FRISCO
VEGAS
LA
DALLAS
MIAMI
DC
MINNE
DENVER
CHICAGO
BOSTON
NYC
Enter source city : CHICAGO
Enter destination city : SEATTLE
->Shortest Distance between CHICAGO and SEATTLE is 2929
And is obtained by going through the nodes
CHICAGO->FRISCO->SEATTLE
->Estimated delivery : 9 days from now
A distance of 300 km is travelled per day!!!
Press any key to continue . . . =
```

Figure 2 Text Window Comprising of Values for the Above Example that are Arbitrarily Chosen

**Figure 3. Graphic Window Showing Arbitrary Distance Values**

graph2.txt - Notepad

```

File Edit Format View Help
SEATTLE FRISCO 1306
FRISCO VEGAS 919
LA VEGAS 435
VEGAS DALLAS 1983
DALLAS MIAMI 2161
DALLAS DC 2113
DALLAS MINNE 1532
DALLAS DENVER 1258
DENVER MINNE 1483
CHICAGO BOSTON 1613
BOSTON NYC 338
DC MIAMI 1709
VEGAS DENVER 1225
CHICAGO FRISCO 1623
VEGAS DC 1995
FRISCO MIAMI 1665
CHICAGO NYC 1725
DC NYC 1751
MIAMI BOSTON 2015
DC MINNE 1659
DENVER NYC 1002
CHICAGO LA 1802
SEATTLE DC 1275

```

Figure 4 Text Window Showing Arbitrary Distance Values

VIII. CONCLUSION

This paper mainly studied the application of the shortest path algorithm based on the data structure, and proposed the improved Dijkstra's algorithm, which optimized selection of the shortest path node and data storage structure and organization. Studies showed that, compared with the traditional Dijkstra's algorithm, the optimized Dijkstra's algorithm which has optimized the space complexity, time complexity and storage combination reduced the storage

space, reduced data redundancy and greatly improved the running rate. It was clearly showed that the optimized algorithm is more applicable to calculate the shortest path.

REFERENCES

- [1] Y. Cao, The Shortest path algorithm in data structures. Yibin University, vol. 6, 2007,pp.82 -84.
- [2] Taha, H. Operations research an introduction, ninth edition. Pearson publisher, 2011.
- [3] Nar, D. Graph theory with applications to engineering and computer science, Prentice Hall, 1997.
- [4] Sneyers J., Schrijvers, T. and Demoen B. Dijkstra's Algorithm with Fibonacci Heaps: An Executable Description in CHR. 20th Workshop on Logic Programming (WLP'06), Vienna, Austria, February 2006.
- [5] Ravi, N., Sireesha, V. Using Modified Dijkstra's Algorithm for Critical Path Method in a Project Network. International Journal of Computational and Applied Mathematics. Volume 5 number 2 pp 217-225. 2010.
- [6] Introduction to Algorithms Book by Charles E. Leiserson, Clifford Stein, Ronald Rivest, and Thomas H. Cormen