A Study on Contrast and Comparison between Bellman-Ford algorithm and Dijkstra’s algorithm

Thippeswamy.K. 
Asst. Professor and HOD, Department of Information Science and Engg., R.L.JIT, Kodigehalli, Doddaballapura, Bangalore, Rural District, Karnataka, INDIA
Email: thippeswamy_yadav@yahoo.co.in.

Hanumanthappa J. 
Assistant Professor, Department of Studies in Computer Science, University of Mysore, Manasagangotri, Mysore, Karnataka, INDIA
Email: hanums_j@yahoo.com

Dr. Manjaiah D.H. 
Associate Professor and Chairman, Department of Computer Science, Mangalore University, Mangalagangothri, Mangalore, Karnataka, INDIA
Email: ylm321@yahoo.co.in

Abstract - In this article we made a study about the two well known shortest path searching algorithms, which are used in routing. They are Bellman-Ford algorithm and Dijkstra’s algorithm. They were compared on the basis of their run time. The analysis of the comparison is given briefly.

1. INTRODUCTION

Today, an internet can be so large that one routing protocol can not handle the task of updating the routing table of all routers. For this reason internet is divided in to an autonomous systems. The routing inside an autonomous system is called as intra domain routing. And the communication between autonomous systems is called inter domain routing. Bellman-Ford algorithm and the Dijkstra’s algorithms are two popular algorithms used in intra domain routing to update the routing tables.

1.1 Bellman-Ford algorithm

The Bellman–Ford algorithm, sometimes referred to as the Label Correcting Algorithm, computes single-source shortest paths in a weighted digraph (where some of the edge weights may be negative). Bellman-Ford is in its basic structure very similar to Dijkstra’s algorithm, but instead of greedily selecting the minimum-weight node not yet processed to relax, it simply relaxes all the edges, and does this \(|V| - 1\) times, where \(|V|\) is the number of vertices in the graph. The repetitions allow minimum distances to accurately propagate throughout the graph, since, in the absence of negative cycles, the shortest path can only visit each node at most once. Unlike the greedy approach, which depends on certain structural assumptions derived from positive weights, this straightforward approach extends to the general case.

Procedure

Bellman-Ford (list vertices, list edges, vertex source)

// This implementation takes in a graph, represented as lists of vertices
// and edges, and modifies the vertices so that their distance and
// predecessor attributes store the shortest paths.

// Step 1: Initialize graph
for each vertex v in vertices:
    if v is source then v.distance := 0
    else v.distance := infinity
    v.predecessor := null

// Step 2: relax edges repeatedly
for i from 1 to size(vertices)-1:
    for each edge uv in edges:
        u := uv.source
        v := uv.destination
        if v.distance > u.distance + uv.weight:
            v.distance := u.distance + uv.weight
            v.predecessor := u

// Step 3: check for negative-weight cycles
for each edge uv in edges:
    u := uv.source
    v := uv.destination
    if v.distance > u.distance + uv.weight:
        error "Graph contains a negative-weight cycle"
1.2 Dijkstra's algorithm

Dijkstra's algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1959, is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, outputting a shortest path tree. This algorithm is often used in routing. For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined.

Procedure

It should be noted that distance between nodes can also be referred to as weight.

1. Create a distance list, a previous vertex list, a visited list, and a current vertex.
2. All the values in the distance list are set to infinity except the starting vertex which is set to zero.
3. All values in visited list are set to false.
4. All values in the previous list are set to a special value signifying that they are undefined, such as null.
5. Current vertex is set as the starting vertex.
6. Mark the current vertex as visited.
7. Update distance and previous lists based on those vertices which can be immediately reached from the current vertex.
8. Update the current vertex to the unvisited vertex that can be reached by the shortest path from the starting vertex.
9. Repeat (from step 6) until all nodes are visited.

2. ALGORITHMS

The algorithms for the above two procedure are given as follows.

2.1 Bellman-Ford algorithm

Input: Edge edges[], int edgecount, int nodecount, int source
Output: Routing table
Begin:

```c
{ int *distance; // Should be allocated
  int i, j;
  if (distance == NULL) Then
    {
      fprintf(stderr, "malloc () failed\n");
      exit(EXIT_FAILURE);
    }
    for (i = 0; i < nodecount; ++i)
      distance[i] = INFINITY;
    distance[source] = 0;
    for (i = 0; i < nodecount; ++i)
      for (j = 0; j < edgecount; ++j)
        if (distance[edges[j].source] != INFINITY)
          {
            int new_distance =
              distance[edges[j].source] + edges[j].weight;
            if (new_distance < distance[edges[j].dest])
              distance[edges[j].dest] = new_distance;
          }
    for (i = 0; i < edgecount; ++i)
      {
        if (distance[edges[i].dest] >
            distance[edges[i].source] + edges[i].weight)
          {
            puts("Negative edge weight cycles detected!");
            free(distance);
            return;
          }
      }
  }
}

Time complexity

Time a n*m +18*m+27*n*m+3
n - Node count
m - Edge count
i.e Time a n*m
```
Time a 27 * Edges * Nodes

**Time a (number of nodes) (number of edges)**

For a general case of number of edges equals to number of nodes we can write (m=n)

Time a 27*n^2 + 28*n + 3

i.e. **Time a 27n^2**

2.2 Dijkstra’s Algorithm

1 function Dijkstra (Graph, source):
2 for each vertex v in Graph: // Initializations
3 dist[v] := infinity // Unknown distance function from source to v
4 previous[v] := undefined // Previous node in optimal path from source
5 dist[source] := 0 // Distance from source to source
6 Q := the set of all nodes in Graph // All nodes in the graph are unoptimized - thus are in Q
7 while Q is not empty: // The main loop
8 u := node in Q with smallest dist[]
9 remove u from Q
10 for each neighbor v of u: // where v has not yet been removed from Q.
11 alt := dist[u] + dist_between (u, v)
12 if alt < dist[v] // Relax (u, v)
13 dist[v] := alt
14 previous[v] := u
15 return previous []

If we are only interested in a shortest path between vertices source and target, we can terminate the search at line 10 if u = target. Now we can read the shortest path from source to target by iteration:

1 S := empty sequence
2 u := target
3 while defined previous[u]
4 insert u at the beginning of S
5 u := previous[u]

**Time complexity**

The running time of Dijkstra's algorithm on a graph with edges E and vertices V can be expressed as a function of |Edges| and |Vertices| using the Big-O notation.

The simplest implementation of the Dijkstra's algorithm stores vertices of set Q in an ordinary linked list or array, and operation Extract-Min(Q) is simply a linear search through all vertices in Q. In this case, the running time is O (|V|^2+|E|) = O(|V|^2).

For a general case of number of edges equals to number of nodes we can write (m=n)

i.e. **Time a 2n^2** (form the above algorithm)

For a case of n=m we can plot the following graph

Number of Nodes vs. time

Form the above graph it is clear that though the nature of the two curves are same i.e O(n^2), the Bellman ford algorithm requires more time than Dijkstra’s algorithm.

The functionality of Dijkstra’s original algorithm can be extended with a variety of modifications. For example, sometimes it is desirable to present solutions, which are less than mathematically optimal. To obtain a ranked list of less-than-optimal solutions, the optimal solution is first calculated. A single edge appearing in the optimal solution is removed from the graph, and the optimum solution to this new graph is calculated. Each edge of the original solution is suppressed in turn and a new shortest-path calculated. The secondary solutions are then ranked and presented after the first optimal solution. Unlike Dijkstra's algorithm, the Bellman-Ford algorithm can be used on graphs with negative edge weights, as long as the graph contains no negative
cycle reachable from the source vertex $s$. (The presence of such cycles means there is no shortest path, since the total weight becomes lower each time the cycle is traversed.) How ever the Bellman-Ford algorithm has another draw back. The Bellman-Ford algorithm does not prevent routing loops from happening and suffers from the count-to-infinity problem. The core of the count-to-infinity problem is that if $A$ tells $B$ that it has a path somewhere, there is no way for $B$ to know if it is on the path. To see the problem clearly, imagine a subnet connected like A-B-C-D-E-F, and let the metric between the routers be "number of jumps". Now suppose that $A$ goes down. In the vector-update-process $B$ notices that its once very short route of 1 to $A$ is down - $B$ does not receive the vector update from $A$. The problem is, $B$ also gets an update from $C$, and $C$ is still not aware of the fact that $A$ is down - so it tells $B$ that $A$ is only two jumps from it, which is false. This slowly propagates through the network until it reaches infinity (in which case the algorithm corrects itself, due to the "Relax property" of Bellman Ford).

3. CONCLUSION

As the analysis shows the Bellman-Ford algorithm solves a problem with a complexity of $27n^2$ but the Dijkstra's algorithm solves the same problem with a lower running time, but requires edge weights to be non-negative. Thus, Bellman–Ford is usually used only when there are negative edge weights.

Both of these functions solve the single source shortest path problem. The primary difference in the function of the two algorithms is that Dijkstra's algorithm cannot handle negative edge weights. Bellman-Ford algorithm can handle some edges with negative weight. It must be remembered, however, that if there is a negative cycle there is no shortest path.

4. REFERENCES

[1] en.wikipedia.org/


5. ACKNOWLEDGEMENT

I would like to thank the Department of Information Science & Engineering, HKBK College of Engineering, Department of Studies in Computer Science, University of Mysore, Manasagangothri & PG Department of Computer Science, Mangalore University, Mangalore for providing support to conduct this research work.

Mr.Hanumantappa.I. is Lecturer at the DoS in CS, University of Mysore, Manasagangothri, Mysore-06 and currently pursuing Ph.D in Computer Science and Engineering, from Mangalore University under the supervision of Dr.Manjaith.D.H on entitled “IPv6 and Multimedia Stuffs”. His teaching and Research interests include Computer Networks,Wireless and Sensor Networks, Mobile Ad-Hoc Networks,Intrusion detection System,Network Security and Cryptography, Internet Protocols, Mobile and Client Server Computing, Traffic management,Quality of Service,RFID,Bluetooth,Unix internals, Linux internal, Kernel Programming ,Object Oriented Analysis and Design etc. His most recent research focus is in the areas of Internet Protocols and their applications. He received his Bachelor of Engineering Degree in Computer Science and Engineering from University B.D.T College of Engineering, Davangere, Karnataka(S),India( C),Kuvempu University,Shimoga in the year 1998 and Master of Technology in CS&Engineering from NITK Surathkal,Karnataka(S),India (C) in the year 2003.He has been associated as a faculty
of the Department of Studies in Computer Science since 2004. He has worked as lecturer at SIR.M.V.I.T.Y.D.I.T.S.V.I.T.of Bangalore. He has guided about 50 Project thesis for BE, B.Tech, M.Tech, MCA, MSc/MS. He has published about 10 technical articles in International and National Peer reviewed conferences. He is a Life member of CSI, ISTE, AMIE, IAENG, Embedded networking group of TIFAC–CORE in Network Engineering, ACM, Computer Science Teachers Association(CSTA). He is also a BOE member of all the Universities of Karnataka, INDIA. He has also visited Republic of China as a Visiting Faculty to teach Computer Science Subjects like OS and System Software and Software Engineering for B.Tech Students of Huancui University in the year 2008. He has also visited Thailand as a Tourist to visit so many places in Thailand.

Mr. B.I.D Kumar is Asst. Professor at the HKBK College of Engineering in Information Science, Bangalore-45. His teaching and research interests include Algorithms, Data Structure, Microprocessor, Computer Networks, Unix Internals, Linux Internal, Kernel Programming, Object Oriented Analysis and Design etc. He received his Bachelor of Engineering Degree in Computer Science and Engineering from Shree Siddhartha Institute of Engineering, Tumkur, Karnataka( S.), India( C.), Bangalore University, Bangalore in the year 2000 and Master of Technology in Computer Science & Engineering from University BDT College of Engineering, Davanagere, Kuvempu University, Shimoga, Karnataka( S.), India( C.) in the year 2005. He has been associated as a faculty of the Department of Information Science since August 2007. In his 7 years of teaching experience, he has guided many Project thesis for BE, B.Tech and involved in technical activities.

Dr. Manjaiah D.H. is currently Reader and Chairman of BoS in both UG/PG in the Computer Science at Dept. of Computer Science, Mangalore University, Mangalore. He is also the BoE Member of all Universities of Karnataka and other reputed universities in India. He received PhD degree from University of Mangalore, M.Tech. from NITK, Surathkal and B.E., from Mysore University. Dr. Manjaiah D.H has an extensive academic, Industry and Research experience. He has worked at many technical bodies like IAENG, WASET, ISOC, CSI, ISTE, and ACS. He has authored more than 25 research papers in international conferences and reputed journals. He is the recipient of the several talks for his area of interest in many public occasions. He is an expert committee member of an AICTE and various technical bodies. He had written Kannada text book, with an entitled, "COMPUTER PARICHAYA", for the benefits of all teaching and Students Community of Karnataka. Dr. Manjaiah D.H.D.H’s areas interest are Computer Networking & Sensor Networks, Mobile Communication, Operations Research, E-commerce, Internet Technology and Web Programming.