Available Online at www.ijcsmc.com/journal

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 7.056

IJCSMC, Vol. 11, Issue. 4, April 2022, pg.1 – 8

Review of Shortest Path Algorithm

Arijeet Banerjee[1]; Pijush Kanti Kumar[2]

[1]Undergrad, Information Technology, Government College of Engineering and Textile Technology, Serampore-712201, India

[2]Asst. Prof, Information Technology, Government College of Engineering and Textile Technology, Serampore-712201, India

[1] arijeetbanerjee98@gmail.com; [2] pijush752000@yahoo.com

DOI: https://doi.org/10.47760/ijcsmc.2022.v11i04.001

Abstract:- *Shortest path algorithm. Graphs are an example of non-linear data structure. A graph is a collection of nodes which are connected by edges. The definition of graph G = (V, E) is basically a collection of vertices and edges. Graphs can be classified on the basis of types of edges. Directed graphs have each of the edges directed which means the edges connecting the two nodes defines the way it is connected from and to. On the other side, undirected graphs have edges which have no direction. The edges of a graph have weights which are associated with it. The weight of an edge can be thought as the cost of the edge. Let's assume there are two vertices representing two cities, then the weight of the edge between the vertices may represent the distance between the cities. Given a given graph and a particular node, we can find a path of least total weight from that node to other vertices of the graph. The total weight of the path will be the sum of the weights of the edges. Graphs can be used in real life to find the shortest path between two destinations, used in social networking sites like facebook and the world wide web where the web pages are represented by the nodes. Dijkstra's Algorithm, Floyd – Warshall, Bellman Ford Algorithm, Johnson's algorithm, A* search algorithm are some of the shortest path algorithms.*

Keywords: Graph, Dijkstra's Algorithm, Floyd-Warshall Algorithm, Bellman–Ford Algorithm, Johnson's algorithm, A* search algorithm

1. RESEARCH OBJECTIVES

To discuss various shortest path algorithms like Dijkstra's, Floyd–Warshall, Bellman Ford Algorithm, Johnson's algorithm, A* search algorithm and their space and time complexities.

2. LITERATURE REIVEW

Graphs are an important data structures which can be represented by adjacency matrix or adjacency lists. Algorithms like Dijkstra's, Floyd etc will compute the shortest distance between the nodes of the graphs. If we represent a graph using an adjacency matrix, say arr[i][j]. If arr[i][j]=1 , this indicates there is an edge between node i and node j. If arr[i][j] = w then w represents the cost of the edge connecting node i and node j. Similarly, we can also use adjacency lists for representing a graph. This is the linked list representation of a graph. We can use an array, say a[] and elements represent a linked list. The a[i] element will represent a linked list of nodes adjacent to i^th nodes. The minimum distance from a node to another can be computed by these shortest path algorithms. Some algorithms compute the distance from a particular node to all the other nodes while some algorithms compute the all source-pair shortest paths.

2.1 DIJKSTRA'S ALGORITHM: EXPLANATION AND IMPLEMENTATION

Dijkstra's algorithm allows us to find the shortest path between any two nodes of a graph. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later. Dijkstra thought about the shortest path problem when working at the Mathematical Center in Amsterdam in 1956 as a programmer to demonstrate the capabilities of a new computer called ARMAC. Here is what Dijkstra said in an interview with Philip L. Frana, Communications of the ACM, 2001:

"What is the shortest way to travel from Rotterdam to Groningen, in general: from given city to given city. It is the algorithm for the shortest path, which I designed in about twenty minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a twenty-minute invention. In fact, it was published in '59, three years later. The publication is still readable, it is, in fact, quite nice. One of the reasons that it is so nice was that I designed

it without pencil and paper. I learned later that one of the advantages of designing without pencil and paper is that you are almost forced to avoid all avoidable complexities. Eventually, that algorithm became to my great amazement, one of the cornerstones of my fame."

— Edsger Dijkstra

This algorithm occurs in many forms. Originally the algorithm was found between two given nodes. A more common variant is where the single node was fixed as the source node and then find the shortest paths from the source to all the other nodes in the graph which produced a shortest path tree. Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. A widely used application of shortest path algorithms is network routing protocols, most notably IS-IS (Intermediate System to Intermediate System) and Open Shortest Path First (OSPF). It is also employed as a subroutine in other algorithms such as Johnson's. The algorithm uses a greedy approach to find the next best solution hoping in the end it will find the best solution for the entire problem. In this algorithm, we start with a weighted graph and first choose the source vertex. The source vertex is assigned a value of zero distance and rest of the vertices are assigned infinity values. Now we choose the vertex with the least distance and traverse the other vertices and update the path. If the distance of the adjacent vertex is lesser than the new path length, we do not update it. Now the chosen vertex become marked. We should avoid the path lengths of already visited vertices. We continue this algorithm until all the vertices have been marked or visited. Below shows the pseudocode of the Dijkstra's algorithm.

```
Dijkstra(Graph, Source)
   for each vertex V in G
      distance[V] = infinite
      previous[V] = NULL
      If V not equal to S, add V to Priority Queue Q
   distance[S] = 0
   while Q is not empty
      U = Extract MIN from Q
      for each unvisited neighbour V of U
         tempDistance = distance[U] + edge_weight(U, V)
         if tempDistance < distance[V]
            distance[V] = tempDistance
            previous[V] = U
   return distance[], previous[]
```

The time Complexity of the dijkstra's algorithm is O(V)^2 where V is the number of vertices when we use two arrays, one for checking whether vertices are marked or not and the second

for storing the distances. If we use a heap or a priority queue we can decrease the time complexity to O(E Log V), where, E is the number of edges and V is the number of vertices. The space Complexity is O(V) where V is the number of vertices. This complexity is achieved when a heap data structure with an array is used.

Dijkstra's algorithm is fast, but it suffers from its inability to deal with negative edge weights. The algorithm may or may not work with negative weights.

2.2 BELLMAN-FORD ALGORITHM: EXPLANATION AND IMPLEMENTATION

The Bellman–Ford algorithm is an algorithm which computes the shortest paths from a single source vertex to all of the other vertices in a weighted digraph. Though it is slower than Dijkstra's algorithm, but more versatile, because of its ability to is handle graphs in which some of the edge weights are negative numbers. The algorithm was first proposed by Alfonso Shimbel (1955), but is instead named after Richard Bellman and Lester Ford Jr., who published it in 1958 and 1956, respectively. Edward F. Moore also published a variation of the algorithm in 1959, and for this reason it is also sometimes called the Bellman–Ford–Moore algorithm. To implement Bellman – Ford algorithm we will start with a weighted graph and choose a starting vertex and assign infinity path values to the other vertices. The starting vertex or the source will be assigned zero path value. Now each edge is visited and path relaxation is done if they are inaccurate. If the distance of the adjacent vertex is lesser than the new path length, we do not update it. In this way the path relaxation is done. This above step is repeated V-1 number of times where V is the number of vertices. This now ensures the shortest distances of the vertices. If the relation step is performed once again and if there is a change is their path lengths then there is a negative edge cycle. Below is the pseudocode of the bellman-ford algorithm.

```
Bellman_Ford(Graph, Source)
  for each vertex V in G
    distance[V] = infinite
      previous[V] = NULL
  distance[S] = 0
  for each vertex V in G
    for each edge (U,V) in G
      tempDistance = distance[U] + edge_weight(U, V)
      if tempDistance < distance[V]
        distance[V] = tempDistance
        previous[V] = U
  for each edge (U,V) in G
    If distance[U] + edge_weight(U, V) < distance[V]
      Return : Negative Cycle Exists
  return distance[], previous[]
```


There are several real-world applications for the Bellman-Ford algorithm, including identifying negative weight cycles calculating the smallest possible heat gain/loss in a chemical reaction. Examining a graph for the presence of negative weight cycles, using negative weights, find the shortest path in a graph are some applications of the bellman ford algorithm. Talking about the time complexity the best case is O(E) and the average and worse case is O(V*E). The space complexity is O(V) where V represents the number of vertices and E represents the number of edges. Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm. If a graph contains a "negative cycle" (i.e., a cycle whose edges sum to a negative value) that is reachable from the source, then there is no cheapest path. Any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman–Ford algorithm can detect and report the negative cycle.

2.3 FLOYD–WARSHALL'S ALGORITHM: EXPLANATION AND IMPLEMENTATION

The Floyd–Warshall algorithm is an example of dynamic programming, and was published in its currently recognized form by Robert Floyd in 1962. The Floyd Warshall Algorithm is for solving the All Pairs Shortest Path problem. The problem is to find the shortest distances between every pair of the vertices in a given edge weighted directed Graph. Floyd-Warhshall algorithm is also called as Floyd's algorithm, Roy-Floyd algorithm, Roy-Warshall algorithm, or WFI algorithm. Below are the steps to implement Floyd-Warshall's algorithm. Firstly we need to create a matrix A0 of dimension n*n where n is the number of vertices. The row and the column are indexed as i and j respectively where i and j will be the vertices or nodes of the graph .Now, Each cell A[i][j] is filled with the distance from the ith vertex to the jth vertex. If there is no path from ith vertex to jth vertex, the cell is left as infinity. Now, we create a matrix A1 using our previous matrix A0. The elements in the first column and the first row are left as they are. Let k be the intermediate vertex in the shortest path from source to destination. In this step, k is the first vertex. A[i][j] is filled with (A[i][k] + A[k][j]) if (A[i][j] > A[i][k] + A[k][j]).That is, if the direct distance from the source to the destination is greater than the path through the vertex k, then the cell is filled with A[i][k] + A[k][j]. Similarly, A2 is created using A1. The elements in the second column and the second row are left as they are and A3 A4 …. AN are created. An gives the shortest path between each vertices.

Below is the pseudocode of the Floyd–Warshall algorithm.

```
n = no of vertices
A = matrix of dimension n*n
for k = 1 to n
   for i = 1 to n
      for j = 1 to n
         Ak[i, j] = min (Ak-1[i, j], Ak-1[i, k] + Ak-1[k, j])
return A
```

The time complexity of the Floyd-Warshall algorithm is O(n3) as we are using three nested loops. The space complexity of the Floyd-Warshall algorithm is O(n2). However, the Floyd-Warshall algorithm does not apply to graphs containing negative weight cycles/cycle but for all pairs of vertices ii and jj for which there doesn't exist a path starting at ii, visiting a negative cycle, and end at jj, the algorithm will then work correctly. Floyd Warshall Algorithm is best suited for dense graphs because its complexity depends on the number of vertices in the given graph. The algorithm also does not return the details of the paths. It is also used to find the Inversion of real matrices and testing of bipartite undirected graphs.

2.4 JOHNSON'S ALGORITHM: EXPLANATION AND IMPLEMENTATION

Johnson's algorithm is used to find the shortest paths between all pairs of vertices in an edge-weighted directed graph. Using Johnson's Algorithm, we can find all pairs shortest path in O (|V|^2 log|V| + |V||E|) time. Johnson's Algorithm is basically a combination of both Dijkstra's Algorithm and Bellman-Ford Algorithm. It works well for negative edges but not for negative edge cycles. It works by using the Bellman–Ford algorithm to compute a transformation of the input graph that removes all negative weights, allowing Dijkstra's algorithm to be used on the transformed graph. It is named after Donald B. Johnson, who first published the technique in 1977. Let the given graph be G. Firstly add a new vertex s to the graph, add edges from new vertex to all vertices of G. Let the modified graph be G'. Run Bellman-Ford algorithm on G' with s as source. Let the distances calculated by Bellman-Ford be h[0], h[1], .. h[V-1]. If we find a negative weight cycle, then return. The negative weight cycle cannot be created by new vertex s as there is no edge to s. All edges are from s. Reweight the edges of original graph. For each edge (u, v), assign the new weight as "original weight + h[u] – h[v]".

Remove the added vertex s and run Dijkstra's algorithm for every vertex.

The time complexity of Johnson's algorithm becomes same as Floyd Warshell when the graph is complete (For a complete graph E = O(V2). But in the case of sparse graphs, the

algorithm performs much better than Floyd Warshell. The main steps in algorithm are Bellman Ford Algorithm called once and Dijkstra called V times. Time complexity of Bellman Ford is O(VE) and time complexity of Dijkstra is O(VLogV). So overall time complexity is O(V2log V + VE).

2.5 A*ALGORITHM: EXPLANATION AND IMPLEMENTATION

A* algorithm is a searching algorithm that is used to find the shortest path between an initial and a final point using its artificial intelligence or brains. It is a handy and smart algorithm that is often used for map traversal to find the shortest path to be taken. A* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It still remains a widely popular algorithm for graph traversal. It searches for shorter paths first, thus making it an optimal and complete algorithm. An optimal algorithm will find the least cost outcome for a problem, while a complete algorithm finds all the possible outcomes of a problem. Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute (now SRI International) first published the algorithm in 1968. It can be seen as an extension of Dijkstra's algorithm. A* achieves better performance by using heuristics to guide its search. What A* Search Algorithm does is that at each step it picks the node according to a value-'f' which is a parameter equal to the sum of two other parameters – 'g' and 'h'. At each step it picks the node/cell having the lowest 'f', and process that node/cell. We define 'g' and 'h' as simply as possible. g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there. h = the estimated movement cost to move from that given square on the grid to the final destination. This is often known as the heuristic, which is nothing but a kind of smart guess. Below is the algorithm.

Make an open list containing starting node
    If it reaches the destination node:
    Make a closed empty list
    If it does not reach the destination node, then consider a node with the lowest f-score in the open list
We are finished
Else:
Put the current node in the list and check its neighbors
    For each neighbor of the current node:
        If the neighbor has a lower g value than the current node and is in the closed list:
Replace neighbor with this new node as the neighbor's parent
Else If (current g is lower and neighbor is in the open list):
Replace neighbor with the lower g value and change the neighbor's parent to the current node.
Else If the neighbor is not in both lists:
Add it to the open list and set its g

Although being the best path finding algorithm around, A* Search Algorithm doesn't produce the shortest path always, as it relies heavily on heuristics / approximations to calculate – h. Considering a graph, it may take us to travel all the edge to reach the destination cell from the source cell [For example, consider a graph where source and destination nodes are connected by a series of edges, like – 0(source) –>1 –> 2 –> 3 (target).So the worst case time complexity is O(E), where E is the number of edges in the graph.

3. CONCLUSIONS AND FUTURE WORK

These algorithms are acceptable in solving the shortest path problem. Dijkstra's algorithm is fast, but it suffers from its inability to deal with negative edge weights. On the other hand, bellman ford is able to deal with negative weights but is a single source shortest path algorithm. The Floyd-Warshall Algorithm is for solving the All-Pairs Shortest Path problem unlike the previous two. A* star algorithm uses "brains" to compute the shortest distance based on smart guesses. In future more artificial intelligence-based algorithms can be developed to compute shortest distances in lesser time and space complexity.

REFERENCES

[1]. Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

[2]. http://theory.stanford.edu/~amitp/GameProgramming/

[3]. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[4]. https://www.geeksforgeeks.org/a-search-algorithm/

[5]. https://mitpress.mit.edu/books/introduction-algorithms-third-edition/

[6]. https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm

[7]. https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

[8]. https://en.wikipedia.org/wiki/A*_search_algorithm

[9]. https://en.wikipedia.org/wiki/Johnson%27s_algorithm

[10].https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm

[11].https://www.programiz.com/dsa/dijkstra-algorithm

[12].https://www.programiz.com/dsa/bellman-ford-algorithm

[13].https://www.programiz.com/dsa/floyd-warshall-algorithm

