**Comparison of Shortest Path Searching Algorithms**

**Dijkstra’s Algorithm**

This shortest path algorithm is most commonly used for finding the shortest path from a source vertex to all other vertex. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later. The original Dijkstra’s algorithm does not use min priority queue and runs in O(V2) time where V is the number of nodes. The implementation of this algorithm based on min priority queue implemented using Fibonacci heap brings down the running time by O((E+V)log V). This is asymptotically the fastest known single source shortest path algorithm for arbitrary directed graph with unbounded non-negative weights.

**Applications :**

1) Finding shortest route in a road network (Google Maps).

2) Finding shortest path for a packet in an internet (IP routing).

3) Telephone Networks as well as Flow network.

**Complexity :**

1) Binary Heap as priority queue:- O((E+V) log V).

2) Fibonacci heap : - O(E+V log V).

**FLOYD WARSHALL**

We initialize the solution matrix same as the input graph matrix as a first step. Then we update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and update all shortest paths which include the picked vertex as an intermediate vertex in the shortest path. When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices. For every pair (i, j) of source and destination vertices respectively, there are two possible cases.  
**1)** k is not an intermediate vertex in shortest path from i to j. We keep the value of p[i][j] as it is.  
**2)** k is an intermediate vertex in shortest path from i to j. We update the value of p[i][j] as p[i][k] + p[k][j].

**TIME COMPLEXITY-** O(n3)

**APPLICATIONS -**

* Finding a regular expression denoting the regular language accepted by a finite automaton (Kleene’s algorithm)
* Inversion of real matrices (Gauss–Jordan algorithm).
* Optimal routing.
* Finding a regular expression denoting the regular language accepted by a finite automaton

**Bidirectional Search Algorithm**

This Search Algorithm is commonly used to find the shortest path in a Graph. In this algorithm we start from both the source and the destination vertex and we end the search when both the set of visited edges intersect. This will give us the shortest path. Suppose if branching factor of tree is **b** and distance of goal vertex from source is **d**, then the normal BFS/DFS searching complexity would be O(bd) . On the other hand, if we execute two search operations then the complexity would be O(bd/2) for each search and total complexity would be **O(bd/2).**

This algorithm can be used when :-

1) Both initial and final goals are unique and completely defined.

2) The branching factor is exactly the same in both directions.

**Complexity:-** Time and Space Complexity O(bd/2).

**A\* Search Algorithm**

**A\* Search** is a computer algorithm that is widely used in pathfinding and graph traversal. It enjoys widespread use due to its performance and accuracy. Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute first described the algorithm in 1968. It is an extension of Edsger Dijkstra's 1959 algorithm. A\* achieves better performance by using heuristics to guide its search.

A\* selects the path that minimizes :  **f(n) = g(n) + h(n)**

where n is the last node on the path, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic that estimates the cost of the cheapest path from n to the goal. The heuristic is problem-specific.

**Applications :**

A\* is commonly used for the common pathfinding problem in applications such as games. It also helps in general artificial intelligence engines like chess, Sudoku etc. It finds applications to diverse problems, including the problem of parsing using stochastic grammars in NLP.

**Complexity :**

The [time complexity](https://en.wikipedia.org/wiki/Computational_complexity_theory) of A\* depends on the heuristic. In the worst case of an unbounded search space, the number of nodes expanded is [exponential](https://en.wikipedia.org/wiki/Exponential_time) in the depth of the solution (the shortest path) *d*: *O*(*bd*), where *b* is the [branching factor](https://en.wikipedia.org/wiki/Branching_factor) (the average number of successors per state)

**Worst-case performance :** O(|E|) = O(bd)

**Worst-case space complexity :** O(|V|) = O(bd)

***Performance:***

Calculated on a Peer to Peer Network graph having 16 vertices and 58 edges.

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Algorithm** | **Time** |
| 1. | Dijkstra’s Algorithm | 0.024s |
| 2. | Floyd Warshall Algorithm | 0.026s |
| 3. | Bidirectional Search Algorithm | 0.029s |
| 4. | A\* Search Algorithm | 0.033s |