EduVision: Journal of Innovations in Pedagogy and Educational Advancements
Volume 01, Issue 06, June, 2025
brightmindpublishing.com
ISSN (E): 3061-6972

PATHFINDING PROBLEM BASED ON GREEDY ALGORITHMS: ANALYSIS USING DIJKSTRA'S ALGORITHM AS AN EXAMPLE

Hamroyeva Ozoda
3rd year student of Computer Science and Programming Technology (by type) at the Faculty of Physics, Mathematics and Information Technologies of Bukhara State University
ozodahamroyevaa@gmail.com

Abstract
This article analyzes the problem of pathfinding based on greedy algorithms. The main focus is on Dijkstra's algorithm, and its operating principle, advantages, and limitations are considered. The problem of pathfinding is relevant in modern technologies, in particular in the fields of navigation systems, transport logistics, games, and artificial intelligence. Dijkstra's algorithm tries to find the overall optimal path by selecting a local optimal solution at each stage. This article analyzes the theoretical foundations of the algorithm, as well as practical examples. It also justifies the greedy approach of Dijkstra's algorithm and briefly compares it with other pathfinding algorithms.

Keywords: Greedy algorithms, Dijkstra's algorithm, pathfinding problem, graph theory, optimal solution, algorithmic approach, navigation systems, transport logistics, artificial intelligence, computer graphics.

Introduction

Today, along with the rapid development of information technologies, the demand for automated systems, intelligent control and solutions based on artificial intelligence is growing in various fields. Especially in the fields of transport, logistics, mobile applications, the gaming industry and robotics, the problem of fast and efficient path finding is taking center stage. Path finding algorithms are important computational tools that allow determining the shortest, safest or cheapest path for a user or system. Algorithmic approaches, in particular greedy algorithms, play a key role in solving such problems. The greedy approach is characterized by its simplicity, speed and ability to work in real time. Such

EduVision: Journal of Innovations in Pedagogy and Educational Advancements
Volume 01, Issue 06, June, 2025
brightmindpublishing.com
ISSN (E): 3061-6972

algorithms make the most profitable choice at each step and strive for the overall optimal solution. This article considers the famous Dijkstra algorithm, which is based on such a strategy. Dijkstra's algorithm is an effective algorithm widely used in path finding problems, built on graph theory and based on the greedy approach. The advantage of this algorithm is that it can determine the path from each node to the shortest distance node. Its use in modern navigation systems, GPS devices, artificial intelligence systems, and even game programming once again proves the relevance of this algorithm. Therefore, the article analyzes in detail the essence of greedy algorithms, how they work, and an effective way to solve the path finding problem using the example of Dijkstra's algorithm.

MAIN BODY

Dijkstra's algorithm was developed by Edsger Dijkstra in 1956 and is designed to find the shortest path from a single starting node to all other nodes in a weighted path network (graph). At each step, the algorithm selects the node with the current shortest distance and updates the distance to its neighbors. This process continues until all nodes have been reached.

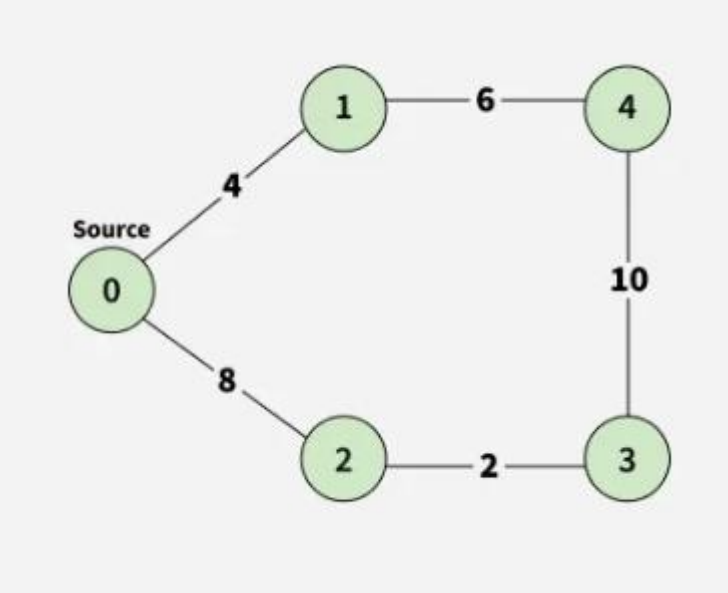
Greedy algorithms are an approach to computing that makes a locally optimal decision at each step in solving a problem. That is, it tries to achieve a general solution by choosing the most profitable or most convenient option each time, and then making the final decision. The main feature of this approach is that it does not consider future steps, but makes the best choice based on the current situation. Greedy algorithms are used to solve many problems quickly and efficiently, including: path finding, weight minimization, interval planning, data compression, etc. However, greedy algorithms do not always guarantee a global optimal solution. Therefore, before using them, it is necessary to determine whether the problem is suitable for the greedy approach.

Given a weighted undirected graph represented as an edge list and a source vertex src, find the shortest path distances from the source vertex to all other vertices in the graph. The graph contains V vertices, numbered from 0 to V - 1.

Note: The given graph does not contain any negative edge.

Examples: Input: src = 0, V = 5, edges[][] = [[0, 1, 4], [0, 2, 8], [1, 4, 6], [2, 3, 2], [3, 4, 10]]

EduVision: Journal of Innovations in Pedagogy and Educational Advancements
Volume 01, Issue 06, June, 2025
brightmindpublishing.com
ISSN (E): 3061-6972



Output: 0 4 8 10 10

Explanation: Shortest Paths:

0 to 1 = 4. 0 → 1

0 to 2 = 8. 0 → 2

0 to 3 = 10. 0 → 2 → 3

0 to 4 = 10. 0 → 1 → 4

➢ **Dijkstra's Algorithm using Min Heap - O(E*logV) Time and O(V) Space**

In Dijkstra's Algorithm, the goal is to find the shortest distance from a given source node to all other nodes in the graph. As the source node is the starting point, its distance is initialized to zero. From there, we iteratively pick the unprocessed node with the minimum distance from the source, this is where a min-heap (priority queue) or a set is typically used for efficiency. For each picked node u, we update the distance to its neighbors v using the formula: dist[v] = dist[u] + weight[u][v], but only if this new path offers a shorter distance than the current known one. This process continues until all nodes have been processed.

Step-by-Step Implementation

Set dist[source]=0 and all other distances as infinity.

Push the source node into the min heap as a pair <distance, node> → i.e., <0, source>.

Pop the top element (node with the smallest distance) from the min heap.

For each adjacent neighbor of the current node:

Calculate the distance using the formula:

EduVision: Journal of Innovations in Pedagogy and Educational Advancements
Volume 01, Issue 06, June, 2025
brightmindpublishing.com
ISSN (E): 3061-6972

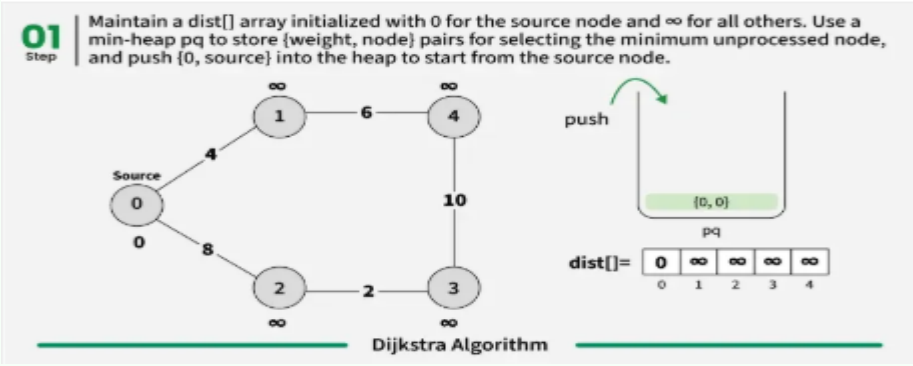
dist[v] = dist[u] + weight[u][v]

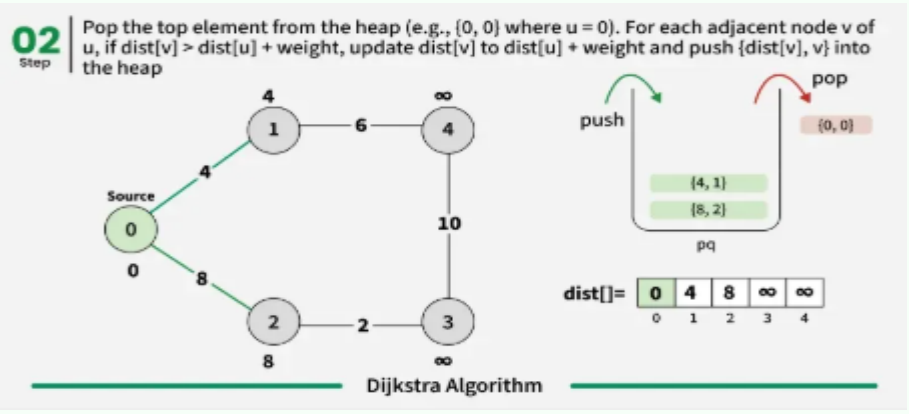
If this new distance is shorter than the current dist[v], update it.

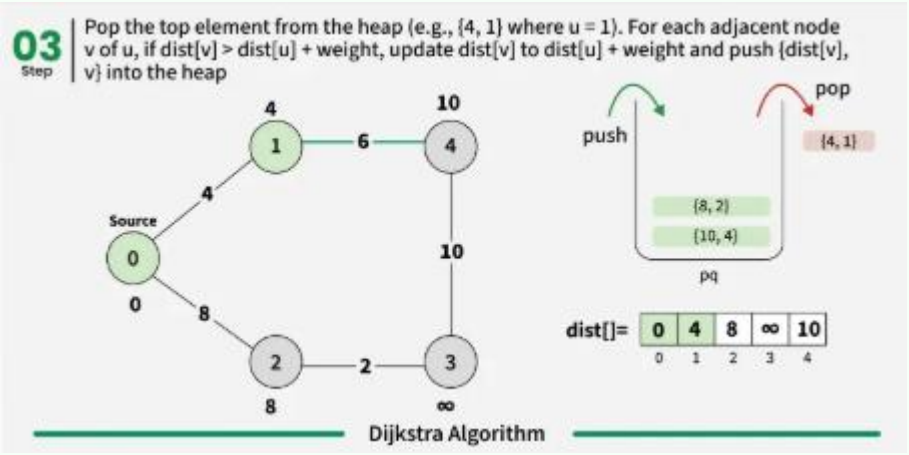
Push the updated pair <dist[v], v> into the min heap

Repeat step 3 until the min heap is empty.

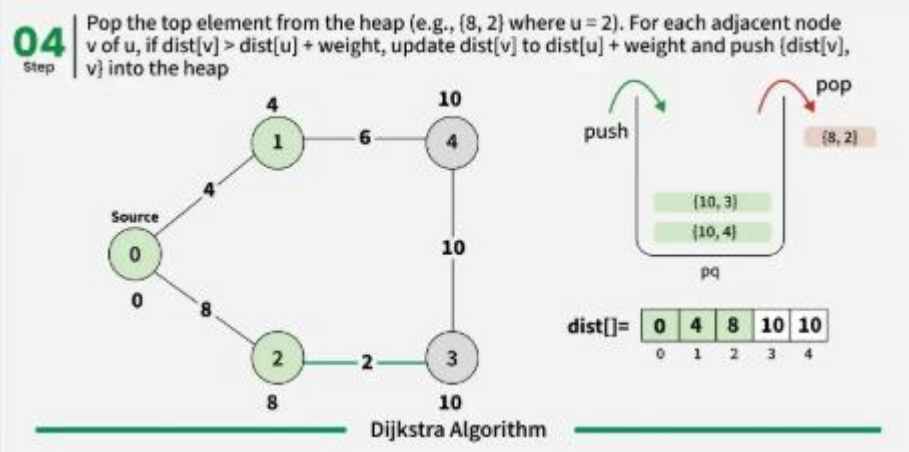
Return the distance array, which holds the shortest distance from the source to all nodes.

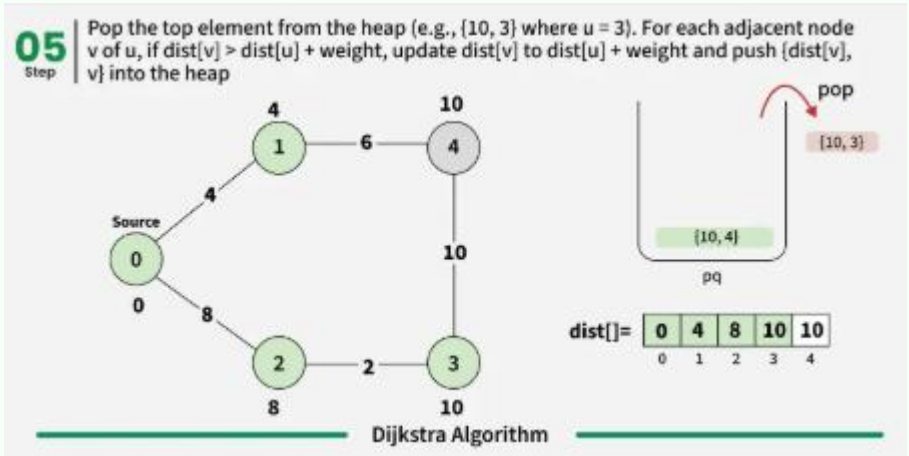


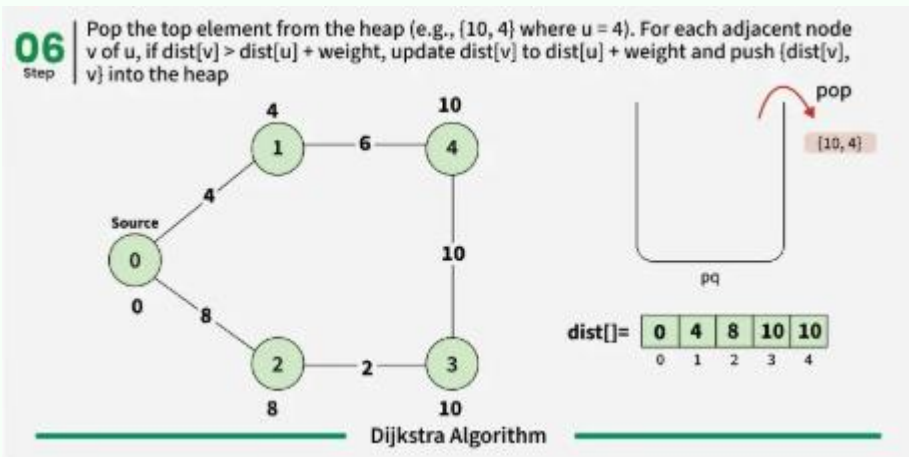




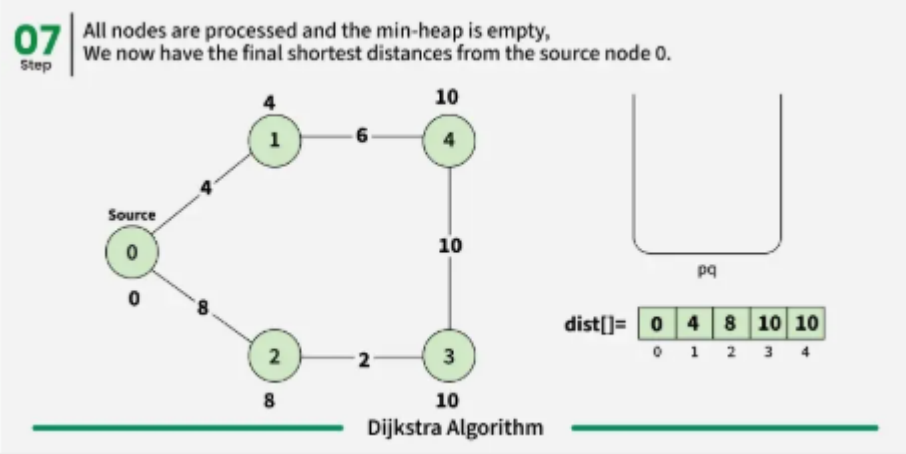
EduVision: Journal of Innovations in Pedagogy and Educational Advancements
Volume 01, Issue 06, June, 2025
brightmindpublishing.com
ISSN (E): 3061-6972







EduVision: Journal of Innovations in Pedagogy and Educational Advancements
Volume 01, Issue 06, June, 2025
brightmindpublishing.com
ISSN (E): 3061-6972



The code in C++:

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
using namespace std;
// Function to construct adjacency
vector<vector<vector<int>>> constructAdj(vector<vector<int>>
                        &edges, int V) {
    // adj[u] = list of {v, wt}
    vector<vector<vector<int>>> adj(V);
    for (const auto &edge : edges) {
        int u = edge[0];
        int v = edge[1];
        int wt = edge[2];
        adj[u].push_back({v, wt});
        adj[v].push_back({u, wt});
    }

    return adj;
}
//Driver Code Ends
// Returns shortest distances from src to all other vertices
vector<int> dijkstra(int V, vector<vector<int>> &edges, int src){
    // Create adjacency list
```

EduVision: Journal of Innovations in Pedagogy and Educational Advancements
Volume 01, Issue 06, June, 2025
brightmindpublishing.com
ISSN (E): 3061-6972


```
vector<vector<vector<int>>> adj = constructAdj(edges, V);
// Create a priority queue to store vertices that
// are being preprocessed.
priority_queue<vector<int>, vector<vector<int>>,
               greater<vector<int>>> pq;
// Create a vector for distances and initialize all
// distances as infinite
vector<int> dist(V, INT_MAX);
// Insert source itself in priority queue and initialize
// its distance as 0.
pq.push({0, src});
dist[src] = 0;
// Looping till priority queue becomes empty (or all
// distances are not finalized)
while (!pq.empty()){

    // The first vertex in pair is the minimum distance
    // vertex, extract it from priority queue.
    int u = pq.top()[1];
    pq.pop();
    // Get all adjacent of u.
    for (auto x : adj[u]){
        // Get vertex label and weight of current
        // adjacent of u.
        int v = x[0];
        int weight = x[1];
        // If there is shorter path to v through u.
        if (dist[v] > dist[u] + weight)
        {
            // Updating distance of v
            dist[v] = dist[u] + weight;
            pq.push({dist[v], v});
        }
    }
```

EduVision: Journal of Innovations in Pedagogy and Educational Advancements
Volume 01, Issue 06, June, 2025
brightmindpublishing.com
ISSN (E): 3061-6972


```
    }
    return dist;
}
//Driver Code Starts
// Driver program to test methods of graph class
int main(){
    int V = 5;
    int src = 0;
    // edge list format: {u, v, weight}
    vector<vector<int>> edges = {{0, 1, 4}, {0, 2, 8}, {1, 4, 6},
                                 {2, 3, 2}, {3, 4, 10}};
    vector<int> result = dijkstra(V, edges, src);
    // Print shortest distances in one line
    for (int dist : result)
        cout << dist << " ";
    return 0;
}
```

Output: 0 4 8 10 10

Time Complexity: O(E*logV), Where E is the number of edges and V is the number of vertices.

Auxiliary Space: O(V), Where V is the number of vertices, We do not count the adjacency list in auxiliary space as it is necessary for representing the input graph.

CONCLUSION

In this paper, we examined the pathfinding problem through the lens of greedy algorithms, with a specific focus on Dijkstra's algorithm. Greedy methods are known for their simplicity and speed, making them suitable for real-time applications where quick decision-making is essential. Dijkstra's algorithm, a classic example of the greedy approach, effectively finds the shortest paths in weighted graphs without negative edge weights. It works by always choosing the next node with the smallest known distance, updating neighboring distances accordingly. This strategy ensures the globally optimal solution, provided the greedy conditions are met. Through theoretical explanation and a step-by-step analysis of a sample graph, we demonstrated how Dijkstra's algorithm

EduVision: Journal of Innovations in Pedagogy and Educational Advancements
Volume 01, Issue 06, June, 2025
brightmindpublishing.com
ISSN (E): 3061-6972

systematically reaches the shortest path results. The algorithm's wide applicability—from navigation systems and GPS tools to robotics and game AI—shows its practical importance in today's technology-driven world. Moving forward, continued research into optimizing greedy algorithms, handling dynamic and large-scale graphs, and integrating other intelligent techniques (such as heuristic or machine learning-based methods) could further enhance pathfinding solutions across various domains.

REFERENCES

1. https://www.geeksforgeeks.org/
2. https://www.geeksforgeeks.org/shortest-path-algorithms-a-complete-guide/
3. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/