Shortest path problem

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.

The problem of finding the shortest path between two intersections on a road map may be modeled as a special case of the shortest path problem in graphs, where the vertices correspond to intersections and the edges correspond to road segments, each weighted by the length of the segment.

Definition

A path in an directed graph is a sequence of vertices $P = (v_1, v_2, \ldots, v_n) \in V \times V \times \cdots \times V$ such that v_i is adjacent to v_{i+1} for $1 \leq i < n$. Such a path P is called a path of length $n - 1$ from v_1 to v_n
Let $e_{i,j}$ be the edge incident to both v_i and v_j

Given a real-valued weight function $f : E \to \mathbb{R}$ and an undirected (simple) graph G

Problem is to find the shortest path $P = (v_1, v_2, \ldots, v_n)$ (where $v_1 = v \text{ and } v_n = v')$ that over all possible n minimizes the sum

$$\sum_{i=1}^{n-1} f\left(e_{i,i+1}\right)$$

Dijkstra's algorithm

Dijkstra's algorithm has many variants but the most common one is to find the shortest paths from the source vertex to all other vertices in the graph.

Algorithm Steps:

- Set all vertices distances = ∞ except for the source vertex, set the source distance = 0.

- Push the source vertex in a min-priority queue in the form (distance , vertex), as the comparison in the min-priority queue will be according to vertices distances.
- Pop the vertex with the minimum distance from the priority queue (at first the popped vertex = source).
- Update the distances of the connected vertices to the popped vertex in case of "current vertex distance + edge weight < next vertex distance", then push the vertex
  with the new distance to the priority queue.
- If the popped vertex is visited before, just continue without using it.
- Apply the same algorithm again until the priority queue is empty.

Pseudocode implementation:

```
function Dijkstra(Graph, source):
    dist[source]  := 0                     // Distance from source to sourc
    for each vertex v in Graph:            // Initializations
        if v ≠ source
            dist[v]  := infinity           // Unknown distance function fro
        add v to Q                         // All nodes initially in Q

   while Q is not empty:                  // The main loop
       v := vertex in Q with min dist[v]  // In the first run-through, this
       remove v from Q

       for each neighbor u of v:           // where neighbor u has not yet
           alt := dist[v] + length(v, u)
           if alt < dist[u]:               // A shorter path to u has been
               dist[u]  := alt            // Update distance of u

   return dist[]
end function
```

Full code inplementation could be find here

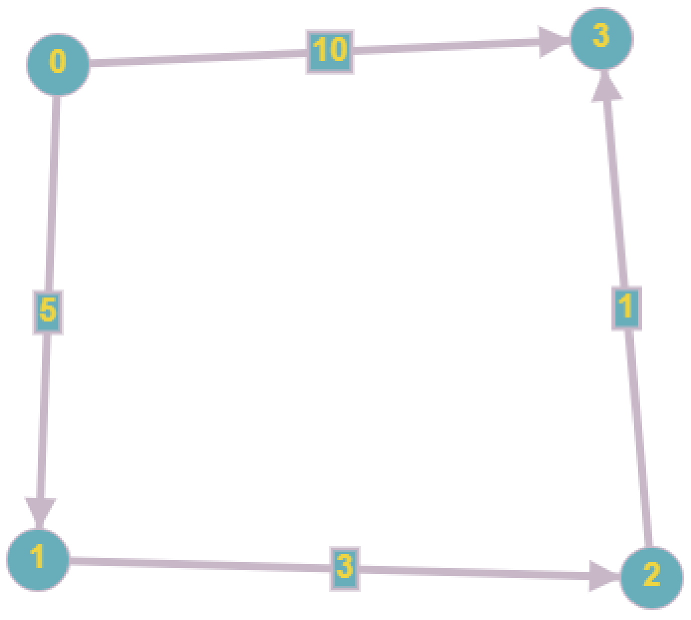
Time Complexity

Time Complexity of Dijkstra's Algorithm is $O(V^2)$ but with min-priority queue it drops down to $O(V + E * log(V))$.[1]

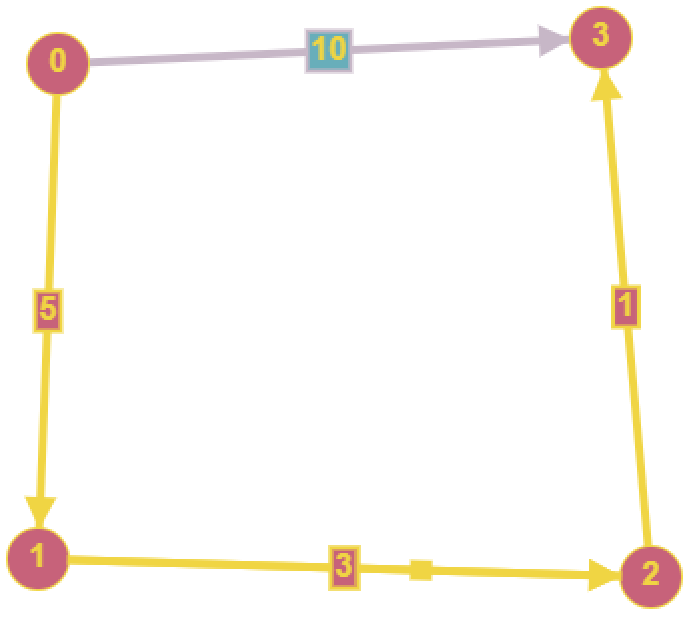
Example solution

Problem is to find shortest path from 0 to 3 vertex

On input we have following graph:



Dijkstra algoritm defines the shortest path is 5+3+1 = 9



Floyd Warshall algorithm

The Floyd-Warshall algorithm is an example of dynamic programming algoritms. It breaks the problem down into smaller subproblems, then combines the answers to those subproblems to solve the big, initial problem.

Algorithm Steps:
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 updates 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 the 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 $dist[i][j]$ as it is.
2. k is an intermediate vertex in shortest path from i to j. We update the value of $dist[i][j]$ as $dist[i][k] + dist[k][j]$ if $dist[i][j] > dist[i][k] + dist[k][j]$

Pseudocode implementation:

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infi
for each edge (_u_,_v_)
   dist[_u_][_v_] ← w(_u_,_v_)  _// the weight of the edge (_u_,_v_)_
for each vertex _v_
   dist[_v_][_v_] ← 0
for _k_ from 1 to |V|
   for _i_ from 1 to |V|
      for _j_ from 1 to |V|
         if dist[_i_][_j_] > dist[_i_][_k_] + dist[_k_][_j_]
             dist[_i_][_j_] ← dist[_i_][_k_] + dist[_k_][_j_]
         end if
```

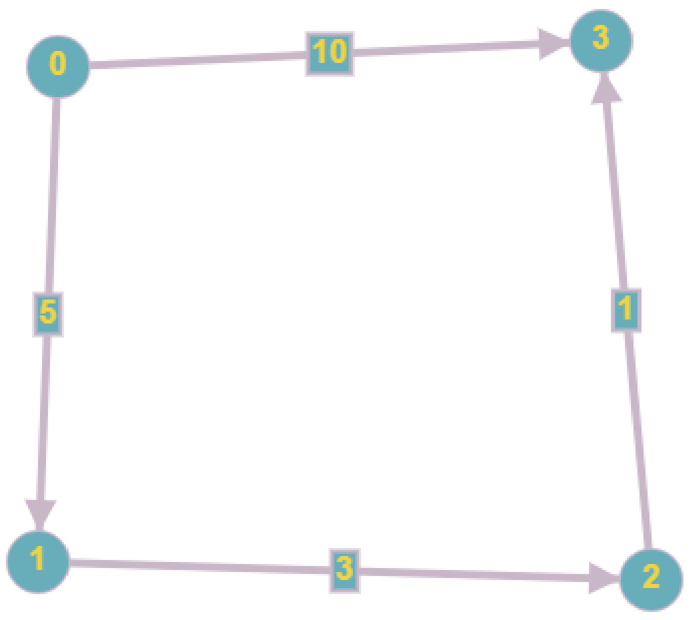
Full code inplementation could be find here

Time Complexity

Time Complexity of Floyd Warshall algorithm is $O(V^3)$[2]

Example solution

On input we have following graph:



On output we have folloving matrix:

```
 0      5      8      9
INF     0      3      4
INF    INF     0      1
INF    INF    INF     0
```

From this matrix we could find that the shortest path between 0 and 3 vertex is 9

Conclusion

Main Purposes:

- Dijkstra's Algorithm is one example of a single-source shortest or SSSP algorithm, i.e., given a source vertex it finds shortest path from source to all other vertices.
- Floyd Warshall Algorithm is an example of all-pairs shortest path algorithm, meaning it computes the shortest path between all pair of nodes.

Comparassion:
The Floyd-Warshall algorithm is effective for calculating all shortest paths in tight graphs when there are a large number of pairs of edges between pairs of vertices. In the case of sparse graphs with edges of non-negative weight, the best choice is to use the Dijkstra algorithm for each possible node. With this choice, the difficulty is$O(|V| * |E|log(V))$ when using a binary heap, which is better than $O(|V|^3)$ in Floyd-Warshell algorithm when $|E|$ is significantly less than $|V|^3$

Google Collab

Full work code and comparison of algorithms you can find below:
Open in Colab

Bibliography

- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). “Section 24.3: Dijkstra’s algorithm”. Introduction to Algorithms (Second ed.). MIT Press and McGraw–Hill. pp. 595–601.
- Cherkassky, Boris V.; Goldberg, Andrew V.; Radzik, Tomasz (1996). “Shortest paths algorithms: theory and experimental evaluation”. Mathematical Programming. : 129–174.
- Abraham, Ittai; Fiat, Amos; Goldberg, Andrew V.; Werneck, Renato F. “Highway Dimension, Shortest Paths, and Provably Efficient Algorithms”. ACM-SIAM Symposium on Discrete Algorithms, pages 782–793, 2010.
- Ahuja, Ravindra K.; Mehlhorn, Kurt; Orlin, James B.; Tarjan, Robert E. (April 1990). “Faster Algorithms for the Shortest Path Problem” (PDF). Journal of the ACM. 37 (2): 213–223. doi:10.1145/77600.77615.
- Thorup, Mikkel (2000). “On RAM priority Queues”. SIAM Journal on Computing. 30 (1): 86–109
- Ahuja, Ravindra K.; Mehlhorn, Kurt; Orlin, James B.; Tarjan, Robert E. (April 1990). “Faster Algorithms for the Shortest Path Problem” (PDF). Journal of the ACM. 37 (2): 213–223.
- Левитин А. В. Глава 9. Жадные методы: Алгоритм Дейкстры // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 189–195. — 576 с. — ISBN 978-5-8459-0987-9
- С. Анисимов. Как построить кратчайший маршрут между двумя точками
- Raman, Rajeev (1997). “Recent results on the single-source shortest paths problem”. SIGACT News. 28 (2): 81–87
- Zhan, F. Benjamin; Noon, Charles E. (February 1998). “Shortest Path Algorithms: An Evaluation Using Real Road Networks”. Transportation Science. 32 (1): 65–73