December 20, 2022 ▪ Data Structure / Graph

G-34: Dijkstra's Algorithm (Intuition and Time Complexity Derivation)

Note: Please watch the previous two videos of this series in order to get an idea of the problem statement as well as the approach required for solving the problem using Dijkstra's Algorithm.

In this particular article, we'll solely be discussing the intuition behind the approach used as well as the full derivation of the time complexity of Dijkstra's Algorithm.

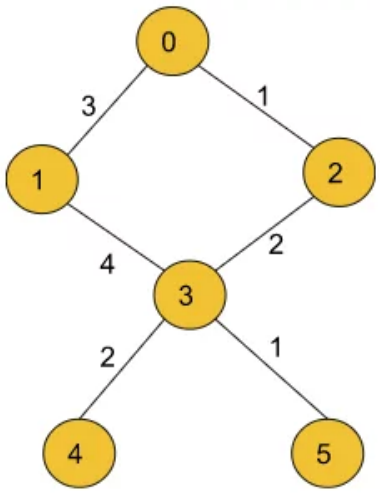
The Intuition behind using Priority Queue and not just Queue in Dijkstra's Algorithm :

Although we can get a solution by just using the Queue data structure also, using a Priority Queue provides us an edge over the

method in which only Queue is used. Now, you may wonder why is that so.

Priority Queue implements Min Heap which eventually gives us the minimum element at the top of the priority queue thus reducing the number of traversals to be done in a graph in case a normal queue is being used. The only difference between a queue and a priority queue is that we have to traverse all connected nodes of a current node and find the minimum among them when we use a normal queue which takes time of O(V). But using the priority queue we can optimize it to O(log V) by reducing the unnecessary exploration of paths.

Let's consider an example of a graph given below:



In this example in order to travel to node 3 from node 0, we will explore both the paths one via node 1 and another via node 2 but in the case of a priority queue, we straightaway get the minimum path i.e., 0->2->3.

Recent Posts

Derivation for Time Complexity of Dijkstra's Algorithm

We know from past lectures that the time complexity of Dijkstra's Algorithm is O(E*logV) where E= number of edges and V= number of nodes.

Now, we will see how can we reach this particular time complexity with the help of a simple derivation:

Pseudocode of Priority Queue :

Derivation:

```
O( V * ( pop vertex from min
heap + no. of edges on each
vertex * push in PQ ))

O( V * ( log(heapSize) + no. of
edges * log(heapSize))

O( V * (log(heapSize) + V-1 *
log(heapSize))    { one vertex
```

```
can have V-1 edges }

O( V * (log(heapSize) * (V-1+1))

O( V * V * log(heapSize))

Now, at the worst case the
heapSize will be equivalent to
v² as if we consider pushing
adjacent elements of a node, at
the worst case each element will
have V-1 nodes and they will be
pushed onto the queue.

O( V * V * log(v²))

O ( v² * 2 log (V))

O ( E * 2 log(V))  { E= v²,
total number of edges}

O ( E * log(V))  Worst case Time
Complexity of Dijkstra’s
Algorithm.
```

> Special thanks to **Priyanshi Goel** for contributing to this article on takeUforward. If you also wish to share your knowledge with the takeUforward fam, please check out this article. If you want to suggest any improvement/correction in this article please mail us at write4tuf@gmail.com

« Previous Post
G-35 : Print Shortest Path – Dijkstra's Algorithm

Next Post »
Number of Operations to Make Network Connected – DSU: G-49.

Load Comments

takeUforward

The best place to learn data structures, algorithms, most asked coding interview questions, real interview experiences free of cost.

Follow Us

DSA Playlist	DSA Sheets	Contribute
Array Series	Striver's SDE Sheet	Write an Article
Tree Series	Striver's A2Z DSA Sheet	
Graph Series	SDE Core Sheet	
DP Series	Striver's CP Sheet	