Fastest Route

June 9, 2020

1 Fastest Route

sometimes the fastest route is not the direct route. Perhaps taking route $ c $, which as an additional stop is the fastest route

If we have undiscovered weights, we cannot be sure that $ A $ to $ C $ is the sortest path. Maybe $ A $ to $ B $ to $ C $ is the sortest

Notice howerver, that $ A $ to $ B $ is the fastest route, even though we dont know route from $ C $ to $ B $

2 Naive Algorithm

2.1 Optimal Substructure

Observation: > Any subpath of an optimal path is also optimal

Proof: - proof by contridiction - Consider an optional path from $ A $ to $ B $ and two vertices $ u $ and $ v $ on this path. IF there were shorter path from $ u $ to $ v $ we would get a shorter path from $ A $ to $ B $

Corollary: - if $ A -> … u -> B $ is a shortest path from $ A $ to $ B $, then:

$$d(A, B) = d(A, u) + w(u, B)$$

Edge Relaxation: - $ dist[v]$ will be an upper bound on the actual distance from $ A $ to $ v $ - thus the shortest path will be at most $ dist[v] $ - The edge relacation procedure for an edge $ (u, v) $ just checks whether going from $ A $ to $ v $ through $ u $ improves the current vaue of $ dist[v]$

2.2 Naive Approach

Correct Distance: > After the call to Naive Algorithm all the distances are set correctly

Proof: - Assume, for the sake of contradiction, that no edge can be relaxed and there is a vertex $ v $ such that $ dist[v] > d(A, v) $ - Consider a shortest path from $ A $ to $ v $ and let $ u $ be the first vertex on this path with the same property. Let $ p $ be the vertex right before $ u $ - Then $ d(A, p) = dist[p]$ and hence $ d(A, u) = d(A, p) + w(p, u) = dist[p] + w(p, u) $ - $ dist[u] > d(A, u) = dist[p] + w(p, u) => edge(p, u) $ can be relaxed – a contradiction

3 Diklstra's Algorithm

3.0.1 Intuition

- Start at \$ A \$
- we relax all the adges from A, meaning we know that its conenected to \$ B \$ and \$ C \$
- we can be sure distance between \$ A \$ to \$ B \$ is \$ 5 \$
- we dont know to \$ C \$ because \$ B \$ to \$ C \$ might be lessen then \$ A \$ to \$ C \$
- we know distance for \$ B \$ so we mark it with a color
- now we relax all the edges from \$ B \$ and we find \$ B \$ to \$ C \$
- We see that that \$ A \$ to \$ C \$ is now \$ 8 \$
- We then discover more outgoing edges
- The next vertex is \$ E \$ because its the vertex with the min known distance
- while \$ C \$ and \$ D \$ it is possible that their \$ dist \$ values are larger than actual distances

3.1 Main idea of Dijkstra's Algo

Dijkstra: - We maintain a set \$ R \$ of vertices for which \$ dist \$ is already set correctly ("Known Region") - The first vertex added to \$ R R is \$ A \$ - On each iteration we take a vertex outside of \$ R \$ with the minimal \$ dist-value \$, add it to \$ R \$, and relax all its outgoing edges

4 Example of Dijkstra's Algorithm

Process: - Start with \$ 0 \$

- we check the edges and relax them from infinity to their value
- Now we need to select the next node to process
- we select the node with a vale of \$ 3 \$ because its the closest
- we process the second selected node
- We cannot relax the edge going to \$ 10 \$ because \$ 3 + 8 = 11 \$ and the node value is 10.
- next edge has line \$ 3 \$. \$ 3 + 3 = 6 \$ which is less then infinity.
- So we can update this value
- next edge \$ 3 + 5 \$ is less then infinty, so we relax that edge
- we make the \$ 6 \$ node our next node because its the lowest node
- update our nodes from \$ 6 \$. Note that the node which was \$ 10 \$ now becomes \$ 9 \$ because of a edge going from node \$ 6 \$ to it
- move on to the next node which is \$ 7 \$
- finally we selct the last node and select it
- Finally we have to go though all of the nodes and update their values

5 Dijkstra's Algorithm: Implementation

Overview: - Were going to be using a priority queue - `dist[u] <- inf, prev[u] <- NaN`: this line initalizes \$ dist[u] \$ to infinity and : `prev[u]` to None - `prev[u]`: basically keeps track of shortest path - `dist[A] = 0`: first node has zero distance from itself - `H <- MakeQueue(V)`: creates pirority queue - Our priority queue will allow us to select the know min value - `u <- ExtractMin(H)`: remove min value from the queue and assign it to \$ u \$ - `for all (u, v) in E`: we look at all the outgoing edges from \$ u \$ - `if dist[v] > dist[u] + w(u, v)`: check if it is possible to relax $ u $ - `dist[v] <- dist[u] + w(u, v)`: relax the edge - `prev[u] <- u`: if we relaxed the edge $ u $ is the previous to last vetex on the best known path from $ A $ to $ v $ - `ChangePriority(H, v, dist[v]`: change the key of known $ v $ because the key of the nodew must always be eual to its dist value

6 Dijkstra's Algorithm: Proof of Correctness

Critical line is `u <- ExtractMin(H)`

Correct Distance Lemma: > When a node $ u $ is selected via `ExtractMin, dist[u] = d(A, u)`

This means that in the end, this dist-value will also be equal to the distance becasue dist-values are always the upper bound on the correct distance because they are substantiated by a specifc path

Proof: - if node $ D $ has the min dist-value out of all the nodes in the unknown region, then we cannot come to it shorter because at some point nwe need to get from know region to unknown region, via some node which has dist-value at least the same or bigger than the dist-value of $ D $ - And we'll have to add at lease one or more deges which are non-negative and so there's no way to come to the node $ D $ shorter than by the current best path we can know. - So this is a contradiction with the assumptuon that the dist-value of the node $ D $ is not equal to the actual distance

7 Analysis

Core parts: - Initalization which runs porportional to the number of nodes - next is the make queue operation, which depends on impementation - next is the min extract, which is $ v $ nodes - then there is the `changePriority` operation for every relaxed edge

Runing Time: - $ O (V) + T(MakeQueue) + |V| * T(ExtractMin) + |E| * T(ChangePriority)$ - Priority Queue implemented as array: - $ O(|V| + |V| + |V|^2 + |E|) = O(|V|^2) $ - Priority Queue implemented as binary heap: - $ O(|V| + |V| + V|log|V| + E|log|V| = O ((|V| + |E|)log|V|) $ - Binary Heap is worse on a full graph