Dijkstra's Algorithm Verification

Yazhe Feng

February 13, 2019

1 Dijkstra's Algorithm

1.1 Pseudocode(Idris)

−− data structures
`Node` : (n : Nat) → Fin n
`NodeSet` : List Node −− ‘NodeSet‘ can represents adjacent edges for each node
`Graph` : (List Node, List NodeSet)

`sortNodes` : (weight : Type) →
(gtW : weight → weight → Bool) →
(add : weight → weight → weight) →
(size : Nat) →
(nodes : Vect size (Node m)) →
(dist : Vect m weight) →
(Vect size (Node m))
`sortNodes` w gtW add Z Nil dist = Nil
`sortNodes` w gtW add (S s') (x :: xs) dist
= `insertSort` x (sortNodes w gtW add s xs dist)

`updateDist` : (weight : Type) →
(gtW : weight → weight → Bool) →
(add : weight → weight → weight) →
(size : Nat) →
(cur : Fin size) →
(adj : NodeSet size weight) →
(dist : Vect size weight) →
(Vect size weight)
`updateDist` w gtW add size cur adj dist
= for Node $n \in adj$:
`if` $dist[cur] + weight[cur- > n] < dist[n]$

`then` $dist[n] = dist[cur] + weight[cur- > n]$
`else` continue to the next nodde

−− if unexplored is Nil, then we have calculated the min distance for all nodes
`runDijkstras` : (weight : Type) →
(gtW: weight → weight → Bool) →
(add : weight → weight → weight) →
(size : Nat) →
(size' : Nat) → −− number of unexplored nodes
(graph : Graph size weight) →
(dist : Vect size weight) →
(lte size' size = True) →
(unexplored : Vect size' (Node size)) →
(Vect size weight)
`runDijkstras` _ _ _ _ Z g dist _ Nil = dist
`runDijkstras` w gtW add _ (S s') g dist refl ((MKNode x) :: xs)
= updateDist w gtW add _ x adj_x dist
call (`runDijkstras` _ _ _ _ s' g dist' refl (sortNodes w gtW add s' xs dist))

`dijkstras` : (weight : Type) →
(gtW : weight → weight → Bool) →
(add : weight → weight → weight) →
(size : Nat) →
(source : Node size) →
(graph : Graph size weight) →
(Vect size weight)
`dijkstras` weight gtW add size source g@(nodes, edges)
= `runDijkstras` weight gtW add size size g dist reflProof (sortNodes weight gtW add size nodes)
where
`dist` = list of nodes and their distance to source. $dist[source] = 0$
`reflProof` = proof of `(lte size size = True)`

2 Proof of Correctness

2.1 Assumptions

1. Valid source node: source node provided is in the corresponding graph.
2. Valid nodes: all nodes in the `nodes` list are valid for indexing distance list and

adjacency list.

3. Path : given source s and node n, if distance from s to n is infinity ($dist[n] = infinity$), then there is no path from s to n

2.2 Proof of Termination

As the size of list `unexplored` decreases by one during each call to `runDijkstras`, the function `runDijkstras` is guaranteed to terminate, thus function `dijkstras` terminates.

2.3 Proof of Correctness

Let Let g be the input graph, s be the source node, *dist* be the list of pairs of each node and its distance to s, and *explored* be the list of explored nodes. We want to show that Dijkstra's algorithm indeed calculates the shortest distance from s for each node $n \in g$.

Lemma. *For any node n added to the explored list, $dist[n]$ is indeed the shortest distance value from source s to u.*

Proof. We will prove this lemma by inducting on the size of *explored* list.
Let P(n) be: forall nodes m in *explored* list of size n, $dist[m]$ is the shortest distance value from source s.
`Base Case`: P(0).
When size is 0, *explored* is empty, then according to the algorithm, the first node added to *explored* is the source node s. Since $dist[s] = 0$ is indeed the shortest distance value from s to s, then the lemma holds.

`Inductive Hypothesis`: Suppose P(k) holds, that is, forall nodes m in *explored* list of size k, $dist[m]$ is the shortest distance value from source s
`Inductive Step`: we shall show P(k+1) holds.
Suppose m' is the (k+1)th element added into *explored*. To show $dist[m']$ is indeed the shortest distance value, we shall show for all nodes p that has an edge to m' in graph g, $dist[m'] \leq dist[p] + weight(m', p)$.
`Case 1`: p is not in *explored*.
Since Dijkstra's chooses the node with minimum distance value from the unexplored list, we have $dist[m'] \leq dist[p]$, hence $dist[m'] \leq dist[p] + weight(m', p)$ holds.
`Case 2`: p is in *explored*

□