Verified Dijkstra’s Algorithm for Shortest Path

Yicong Luo, Qingyi He

01 GRAPH DATA STRUCTURE

Graph implemented in a purely functional way

02 DIJKSTRA’S ALGORITHM

An algorithm to search for the shortest paths from a single source in a graph

03 VERIFICATION

Verify the graph and the Dijkstra’s algorithm

04 FUTURE

Try to optimize the algorithm with a verified heap
Verify more properties of the resulted distance

01

GRAPH DATA STRUCTURE

Graph implemented in a purely functional way

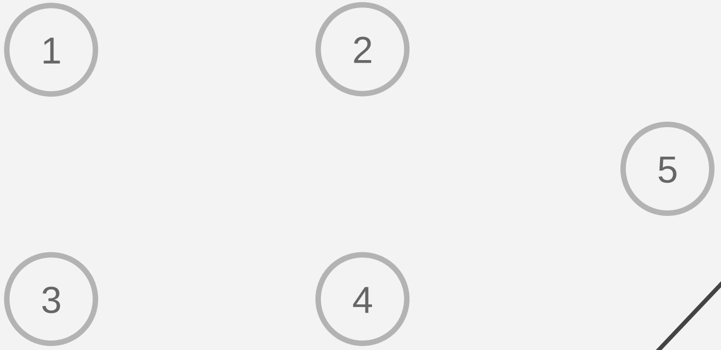
ADJACENCY LISTS

Graph: List[(Int, List[(Int, Distance)])]

Each element of Graph represent a **single node** and **a list of edges** from it, and the edge is represented by a tuple of its **destination** and its **length**

ADJACENCY LISTS

Graph: List[(Int, List[(Int, Distance)])]

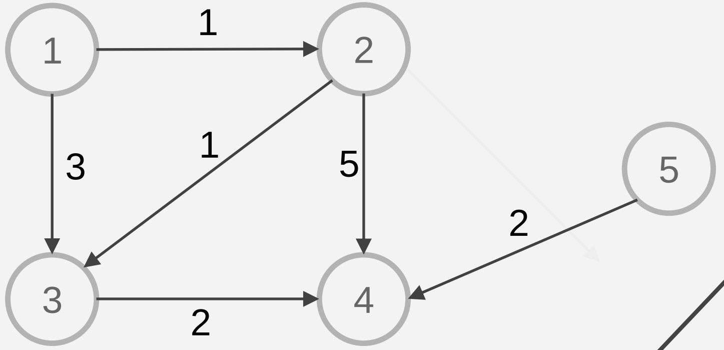


Example:

```
val g = Graph(
  List(
    (1, List(2 -> 1.toDist, 3 -> 3.toDist)),
    (2, List(4 -> 5.toDist, 3 -> 1.toDist)),
    (3, List(4 -> 2.toDist)),
    (4, List()),
    (5, List(4 -> 2.toDist))
  )
)
```

ADJACENCY LISTS

Graph: List[(Int, List[(Int, Distance)])]



Example:

```
val g = Graph(
  List(
    (1, List(2 -> 1.toDist, 3 -> 3.toDist)),
    (2, List(4 -> 5.toDist, 3 -> 1.toDist)),
    (3, List(4 -> 2.toDist)),
    (4, List()),
    (5, List(4 -> 2.toDist))
  )
)
```

02

DIJKSTRA’S ALGORITHM

An algorithm to search for the shortest paths from a single source in a graph

PSEUDO CODE

```
function Dijkstra(Graph, source):

    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q

        for each neighbor v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

From Wikipedia: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

FUNCTIONAL VERSION

```scala
def dijkstra(start: Int): List[Node] = {
  require(graph.get(start) ≠ None())
  iterate(Nil[Node](), prepare(start))
}
```

```scala
// dijkstra main loop
def iterate(seen: List[Node], future: List[Node]): List[Node] = {
  future match
    case Nil() ⇒ seen
    case fu @ Cons(_, _) ⇒
      val (h, t) = getMin(fu)
      iterate(h :: seen, iterOnce(h, t))
}
```

```scala
// update distance from cur
def iterOnce(cur: Node, rest: List[Node]): List[Node] = {
  rest match {
    case Nil()      ⇒ Nil()
    case Cons(h, t) ⇒ Cons(updateDist(cur, h), iterOnce(cur, t))
  }
}
```

```scala
// update distant to node tar (when at node cur)
def updateDist(cur: Node, tar: Node): Node = {
  val nd = cur._2 + distance(cur._1, tar._1)
  (tar._1, if nd ≤ tar._2 then nd else tar._2)
}
```

03

VERIFICATION

Verify the graph and the Dijkstra’s algorithm

VERIFY VALID GRAPH

```scala
def validGraph(graph: List[(Int, List[(Int, Distance)])]): Boolean =
  noDuplicates(graph) &&
    graph.forall(e => noDuplicates(e._2)) &&
    graph.forall(e => e._2.forall((i, _) => graph.get(i) != None())) &&
    graph.forall(n => n._2.forall(z => 0.toDist <= z._2)) &&
    graph.forall { case (n, a) =>
      a.get(n) match {
        case None()  => true
        case Some(d) => d == 0.toDist
      }
    }
```

VERIFY DIJKSTRA’S ALGIRTHM

TERMINATE

The algorithm can finally terminate and never fall into an endless loop.

DISTANCE

For each iteration, the distances of the nodes will monotonically decrease.

VERIFY DIJKSTRA'S ALGIRTHM

```scala
} ensuring (res =>
  res.forall(n => cur._2 <= n._2) &&
    res.map(_._1).content == rest.map(_._1).content &&
    res.size == rest.size &&
    res
      .zip(rest)
      .forall((y, x) =>
        y._1 == x._1 && y._2 <= x._2
      ) // updated dists should be smaller
)
```

TERMINATE

The algorithm can finally terminate and never fall into an endless loop.

```scala
decreases(future.size)
```

```scala
decreases(rest.size)
```

DISTANCE

For each iteration, the distances of the nodes will monotonically decrease.

04

FUTURE

Try to optimize the algorithm with a verified heap
Verify more properties of the resulted distance

WHAT CAN BE DONE NEXT

Use **verified heap** to optimize the step of getting nearest node

Reference: https://stainless.epfl.ch/static/valid/Heaps.html

Verify the **resulted distance list** is correct

- Triangle inequality

THANKS!