




Verified Dijkstra's Algorithm for Shortest Path

Yicong Luo, Qingyi He



GRAPH DATA STRUCTURE

Graph implemented in a purely functional way

01

DIJKSTRA'S ALGORITHM

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

02

VERIFICATION

Verify the graph and the Dijkstra's algorithm

03

FUTURE

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

04



01

GRAPH DATA STRUCTURE

Graph implemented in a purely functional way

-
-
-
-
-
-
-

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

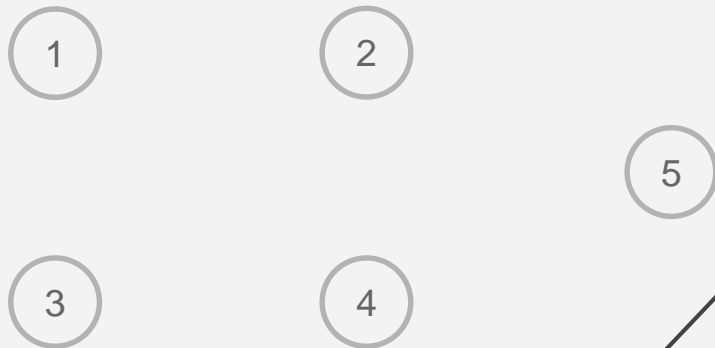
• •
• •
• •
• •
• •
• •
• •

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)])]

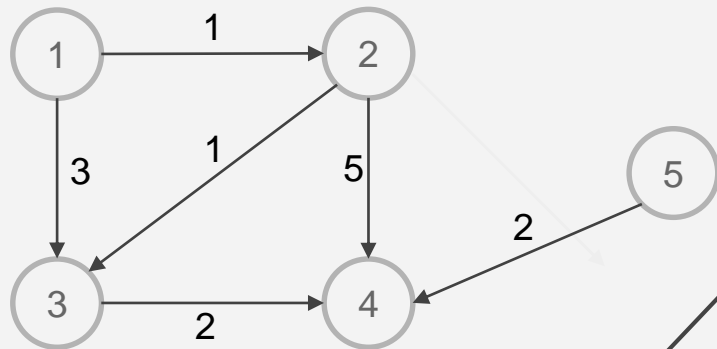


Example:

```
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))  
  )  
)
```

ADJACENCY LISTS

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



A series of dark gray lines forming a large, abstract geometric shape on the left side of the slide. The shape is composed of several connected line segments, creating a series of triangles and quadrilaterals. The lines extend from the top left towards the bottom right, framing the main content area.

02

DIJKSTRA'S ALGORITHM

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

PSEUDO CODE

```
1  function Dijkstra(Graph, source):  
2  
3      for each vertex v in Graph.Vertices:  
4          dist[v]  $\leftarrow$  INFINITY  
5          prev[v]  $\leftarrow$  UNDEFINED  
6          add v to Q  
7      dist[source]  $\leftarrow$  0  
8  
9      while Q is not empty:  
10         u  $\leftarrow$  vertex in Q with min dist[u]  
11         remove u from Q  
12  
13         for each neighbor v of u still in Q:  
14             alt  $\leftarrow$  dist[u] + Graph.Edges(u, v)  
15             if alt < dist[v]:  
16                 dist[v]  $\leftarrow$  alt  
17                 prev[v]  $\leftarrow$  u  
18  
19      return dist[], prev[]
```

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



FUNCTIONAL VERSION

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

```
// 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))  
}
```

```
// 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))  
  }  
}
```

```
// 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

```
140 def validGraph(graph: List[(Int, List[(Int, Distance)])]): Boolean =  
141     noDuplicates(graph) &&  
142     graph.forall(e => noDuplicates(e._2)) &&  
143     graph.forall(e => e._2.forall((i, _) => graph.get(i) != None())) &&  
144     graph.forall(n => n._2.forall(z => 0.toDist <= z._2)) &&  
145     graph.forall { case (n, a) =>  
146         a.get(n) match {  
147             case None() => true  
148             case Some(d) => d == 0.toDist  
149         }  
150     }
```

VERIFY DIJKSTRA'S ALGORITHM



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 ALGORITHM

```
} 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.

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

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

DISTANCE

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

A decorative graphic consisting of several thin, dark gray lines. One line starts from the top left, goes down to a vertex, then up to another vertex, and finally down to a third vertex. Another line starts from the top right and goes down to the same third vertex. A third line starts from the bottom left and goes up to the same third vertex. These lines form a large, open, irregular shape that frames the text on the right side of the slide.

04

FUTURE

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

-
-
-
-
-
-

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

WHAT
CAN BE DONE
NEXT





THANKS!

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**

