```

# HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
# Path   : uebung13/al/aufgabe02
# Version: Mon May 19 19:00:45 CEST 2025

import sys

from uebung13.al.aufgabe02.dijkstra import Dijkstra
from uebung13.al.aufgabe02.dijkstra_adv import DijkstraADV


if __name__ == ’__main__’:

  dijkstra = Dijkstra()  # without ADV
  #dijkstra = DijkstraADV() # with ADV

  graph = dijkstra.get_graph()

  v_a = graph.insert_vertex("A")
  v_b = graph.insert_vertex("B")
  v_c = graph.insert_vertex("C")
  v_d = graph.insert_vertex("D")
  v_e = graph.insert_vertex("E")
  v_f = graph.insert_vertex("F")

  graph.insert_edge(v_a, v_b, 8)
  graph.insert_edge(v_a, v_c, 2)
  graph.insert_edge(v_a, v_d, 4)
  graph.insert_edge(v_b, v_c, 7)
  graph.insert_edge(v_b, v_e, 2)
  graph.insert_edge(v_c, v_e, 3)
  graph.insert_edge(v_c, v_f, 9)
  graph.insert_edge(v_c, v_d, 1)
  graph.insert_edge(v_d, v_f, 5)

  dijkstra.distances(graph, v_a)

  dijkstra.print_distances()

  if len(dijkstra._parents) != 6:
    print("\nERROR: dijkstra._parents should have a size of 6 !")
    sys.exit(1)




```

```

""" Session-Log:

APQ.insert(0, A)
APQ.insert(9223372036854775807, B)
APQ.insert(9223372036854775807, C)
APQ.insert(9223372036854775807, D)
APQ.insert(9223372036854775807, E)
APQ.insert(9223372036854775807, F)
APQ.remove_min(): (0,A)
Graph.opposite(A, 8): B
Graph.opposite(A, 2): C
Graph.opposite(A, 4): D
APQ.remove_min(): (2,C)
Graph.opposite(C, 2): A
Graph.opposite(C, 7): B
Graph.opposite(C, 1): D
Graph.opposite(C, 3): E
Graph.opposite(C, 9): F
APQ.remove_min(): (3,D)
Graph.opposite(D, 4): A
Graph.opposite(D, 1): C
Graph.opposite(D, 5): F
APQ.remove_min(): (5,E)
Graph.opposite(E, 2): B
Graph.opposite(E, 3): C
APQ.remove_min(): (7,B)
Graph.opposite(B, 8): A
Graph.opposite(B, 7): C
Graph.opposite(B, 2): E
APQ.remove_min(): (8,F)
Graph.opposite(F, 9): C
Graph.opposite(F, 5): D

Distances:
A: 0
B: 7
C: 2
D: 3
E: 5
F: 8

"""

```

```python

# HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
# Path    : uebung13/al/aufgabe02
# Version: Mon May 19 19:00:45 CEST 2025

import sys

from uebung13.graphs.adaptable_heap_priority_queue import AdaptableHeapPriorityQueue
from uebung13.graphs.graph import Graph


class Dijkstra:

  def __init__(self):
    self._graph = None
    self._distances = self._get_distance_map()
    self._locators = self._get_locators_map()
    self._parents = self._get_parents_map()

  def distances(self, graph, s):
    self._graph = graph
    self._apq = self._get_adaptable_priority_queue()

    # TODO Implement here ...


  def get_graph(self):
    return Graph()

  def _get_adaptable_priority_queue(self):
    return AdaptableHeapPriorityQueue()

  def _get_distance_map(self):
    return dict()

  def _get_locators_map(self):
    return dict()

  def _get_parents_map(self):
    return dict()

  def print_distances(self):
    print("\nDistances:")
    for v in self._graph.vertices():
      print(str(v) + ": " + str(self._distances.get(v)))

```