**CS469 Data Structures and Algorithms**

**HOS07 Dijkstra’s Algorithm**

05/10/2023 Review by Christopher Sharp

02/12/2024 Reviewed by Anh Nguyen

09/29/2024 Reviewed by Shahid Khan

School of Technology and Computing (STC) @City University of Seattle (CityU)

**Before You Start**

* The document’s examples are written in Python. If you don’t know Python programming language, please finish the Python tutorial in Module00 folder before you start the assignment.
* Some steps are not explained in the tutorial**.** If you are not sure what to do:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

Students will be able to:

- Use Dijkstra’s Algorithm to find the fastest path of a graph

- Understand the differences between weighted and unweighted graphs, and what circumstances to use each in

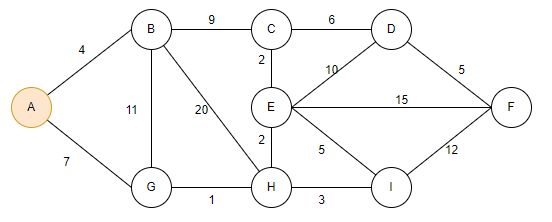
**Resources**

* Dijkstra’s Algorithm in Python, GeekforGeeks - <https://www.geeksforgeeks.org/python-program-for-dijkstras-shortest-path-algorithm-greedy-algo-7/>
* Dijkstra’s Algorithm - <https://www.programiz.com/dsa/dijkstra-algorithm>
* Dijkstra’s Algorithm Theory and Intuition - <https://stackabuse.com/courses/graphs-in-python-theory-and-implementation/lessons/dijkstras-algorithm/>

# Data Structure

In HOS06 Breadth-First graphs, from last week, you learned how to make an algorithm and graph that would take the shortest path possible; this is an unweighted graph. Dijkstra’s Algorithm is used to calculate the fastest path possible, using a weighted graph.

The shortest path for a graph is given by the source to all vertices in the graph. We start by picking the vertex with the shortest current cost and add it to the visited list.

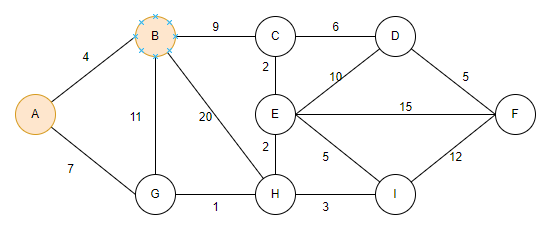


The starting point is vertex A, and we will set the initial cost from vertex A to other vertices to infinite for learning purposes, but it could be anything.

|  |  |
| --- | --- |
| **Vertex** | **Cost to get to vertex from A** |
| A | 0 |
| B | INF |
| C | INF |
| D | INF |
| E | INF |
| F | INF |
| G | INF |
| H | INF |
| I | INF |

After picking our vertex with the lowest cost, A, it will be marked as visited. The starting node will always have the lowest cost, so it will always be marked first.

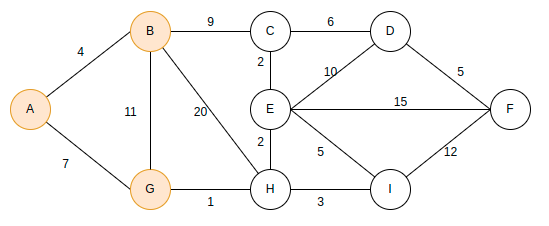
From A, we can move to either B or G. The cost from A to B is 4, and from A to G is 7. Since 4 is lower than 7, B will be the next to visit and we will transverse to B.



Upon traversing, we mark B as visited, and observe the adjacent vertices C, H and G.

* A => B => C cost is 4 + 9 = 13 < INF. The cost from A to C is updated to 13
* A => B => H cost is 4 + 20 = 24 < INF. The cost from A to H is updated to 24
* A => B => G cost is 4 + 11 = 15 > 6 (A => G). The cost from A to G remains 7

|  |  |
| --- | --- |
| **Vertex** | **Cost to get to vertex from A** |
| A | 0 |
| B | 4 |
| C | 13 |
| D | INF |
| E | INF |
| F | INF |
| G | 7 |
| H | 24 |
| I | INF |

The next vertex to visit is vertex G. We mark it as visited and update its adjacent vertices’ costs (Vertex H).

* A => G => H cost is 7 + 1 = 8 < INF. The cost from A to H is updated to 8

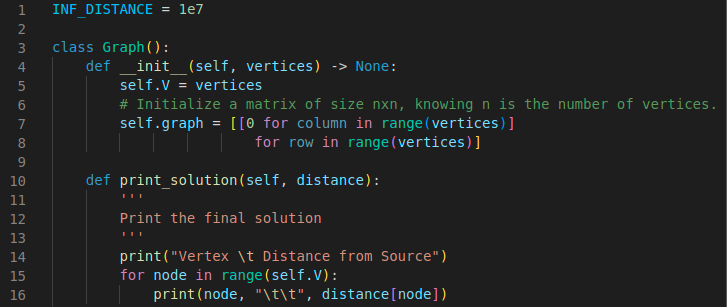
|  |  |
| --- | --- |
| **Vertex** | **Cost to get to vertex from A** |
| A | 0 |
| B | 4 |
| C | 13 |
| D | INF |
| E | INF |
| F | INF |
| G | 7 |
| H | 8 |
| I | INF |

Now that we have a decent idea of how the graph works and how the costs are calculated to find the fastest path possible, we can start looking at how we would code this in python.

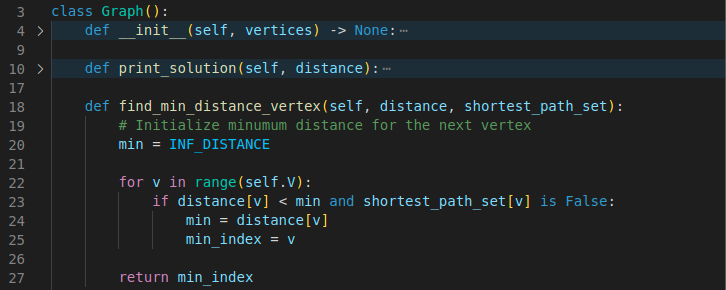
# Algorithm

Note that this program is for an adjacency matrix representation of the graph. An adjacency matrix is a way of representing a graph as a matrix of Booleans. If you’d like to know more, you can read more [here](https://www.programiz.com/dsa/graph-adjacency-matrix#:~:text=An%20adjacency%20matrix%20is%20a,direct%20path%20between%20two%20vertices.).

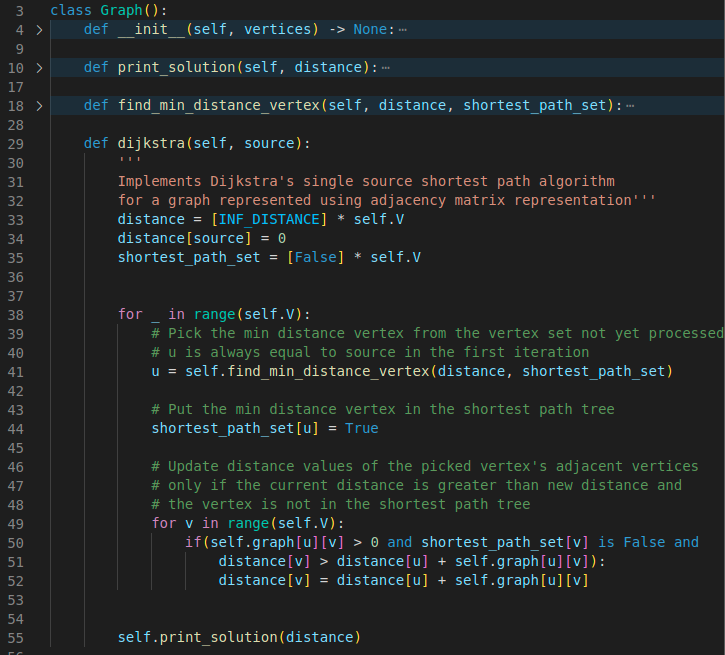
*Please make a file named* ***Dijkstra.py*** *and follow along with the examples.*



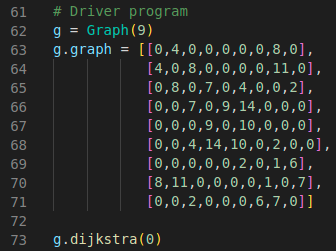
We start off by initializing the graphic and defining the vertices.



To calculate the distances (costs) between the vertices.



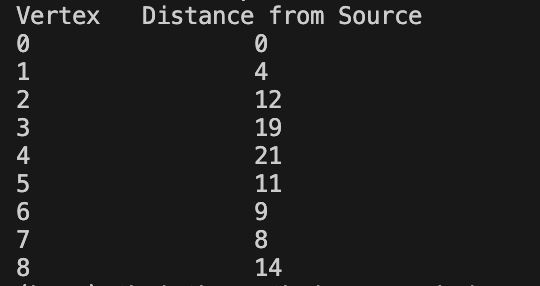
Starting the count and looping through the vertices.



Here’s the grid to copy and paste (may need to adjust formatting after paste):

|  |
| --- |
| g.graph = [[0,4,0,0,0,0,0,8,0],  [4,0,8,0,0,0,0,11,0],  [0,8,0,7,0,4,0,0,2],  [0,0,7,0,9,14,0,0,0],  [0,0,0,9,0,10,0,0,0],  [0,0,4,14,10,0,2,0,0],  [0,0,0,0,0,2,0,1,6],  [8,11,0,0,0,0,1,0,7],  [0,0,2,0,0,0,6,7,0]] |

Your output should look like this.



***Save your code file in the module folder for your deliverable.***

# Push Your Work to GitHub

Open terminal and make sure you’re in the repository folder. (i.e: hos07\_courseName\_GitHubUserName)

**Type the following command to upload your work**:

|  |
| --- |
| >>>> git add.  >>>> git commit -m “Submission for HOS07”  >>>> git push origin master |