ShortestPath

Cornelius Leary

I wanted to solve graph theory problems.

- I have worked through several already:
  - Breadth-First Search (BFS)
  - Depth-First Search (DFS)
  - Bellman-Ford Algorithm
  - Floyd-Warshall Algorithm
  - Dijkstra’s Algorithm (DA)
- I wanted to be able to more-easily pass valid graphs to the actual algorithms.
- I wanted to be able to reuse as much of the graph-representation and graph-importing programs as I could.

Sample Graph (from “Dijkstra’s algorithm” on Wikipedia)

Start: node 1

Goal: node 5

Shortest Path: 1 → 3 → 6 → 5

Minimized Cost from 1 to 5: 20

1) 1 to 1: 0
2) 1 to 3: 9 total: 9
3) 3 to 6: 2 toal: 11
4) 6 to 5: 9 toal: 20

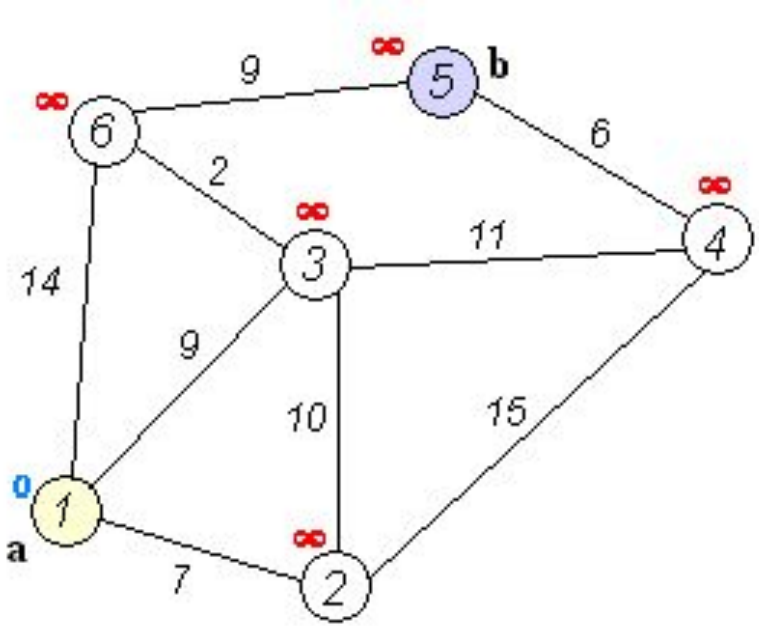
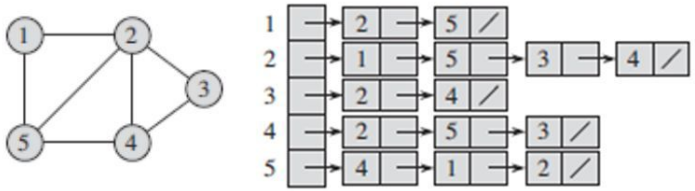


Image: A weighted graph.
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

The underlying graph is represented using an Adjacency List.

Graph representations: Adjacency List

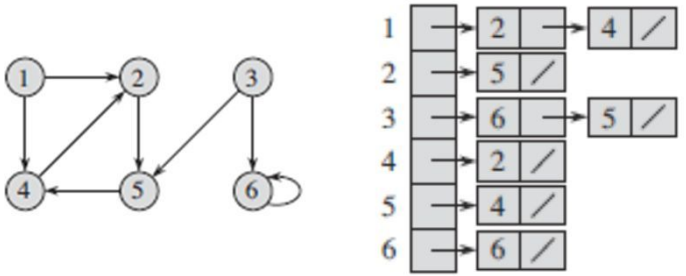
- **Undirected** weighted graph



[CLRS] Fig 22.1

Graph representations: Adjacency List

- **Directed** weighted graph



[CLRS] Fig 22.2

(from CLRS, Ch. 22 - Elementary Graph Algorithms)

Graph G = (V, E)

Adjacency List Representation is useful when graph is sparse ($|E| << |V|^2$)

The *ShortestPath* class imports a graph and finds the shortest path through it.

ShortestPath class

- via *EdgeImporter* object:
  - Reads graph setup information from file, with value on its own line:
    - # of nodes
    - Start node
    - Goal node
  - Reads edges from same file, with each in this format on its own line:

<source name>	*<destination name>*	*<edge weight>*
Integer	*integer*	*double*

- via *Dijkstras_Alg* object:
  - Computes shortest path from start node to goal node
  - Computes cost from start node to every other node in graph

The *EdgeImporter* class imports a graph from a text file.

EdgeImporter class

- Checks if file provided is valid
  - Invalid files:
    - No file name given
    - Non-existent file
    - Incorrect graph setup information
- Sets up internal *EdgeSetBuilder* object
- Reads the graph file
  - Imports edges from file, passing them to *EdgeSetBuilder*
  - Checks if integer-named nodes are zero-based
  - Modifies graphs with non-zero-based node names

The *EdgeSetBuilder* class creates a valid set of edges for a graph algorithm.

EdgeSetBuilder class

- Edges are added:
  - One-at-a-time as a set of *Edge* parameters (method)
    - Must adhere to existing settings
      - Directed / Undirected graph
      - (+)-only or (+) & (-) edge weights
      - Self-Loops allowed / disallowed
  - As an existing vector of *Edge* objects (constructor)
    - Must specify Directed/Undirected
    - (-) edge weights and self-loops are determined from edges in provided set

The *Edge* class represents a graph edge.

Edge class

- Attributes
  - Source name
  - Destination name
  - Edge weight

I want to build a graph-algorithm demonstrator platform in the future.

GUI:

- Qt libraries for GUI
- GraphViz library for graph visualization
- Allow user to choose the algorithm
- Allow user to set up the graph via file
- Allow user to choose the start and goal nodes

Future Algorithms:

- A* Search
- D* (Lite) Search
- RRT's
- PRM's

Thank you for your feedback and your consideration.

https://github.com/Cornelius-Leary-III/graph_search