

proj 16: Graphs

Introduction

In this problem set you will write a solution to an optimization problem on how to find the shortest route from one building to another on campus given you wish to constrain the amount of time you will spend walking outdoors.

Background

A graph consists of a set of nodes (n_1, n_2, n_3, \dots) and a set of edges (e_1, e_2, e_3, \dots) where an edge connects two nodes that are in the graph. The node n_1 has children nodes if there exists an edge from n_1 to each child node. In Figure 1, Node “a” has children nodes “b” and “c.”

There are two types of edges: directed and undirected. If the edge is directed, then the edge has a specific direction going from start to destination node. Graphs with directed edges are called directed graphs (or digraph).

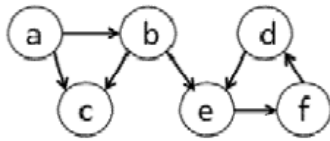


Figure 1. Example of a **directed** graph where each edge has a specific direction.

If the edge is undirected, also known as bidirectional, then it no longer matters which node is the start or destination node because you can traverse the edge from one node to the other in either direction. Essentially, a link in the graph can be represented by a directed edge going from Node “d” to Node “e” and a directed edge going in the reverse direction.

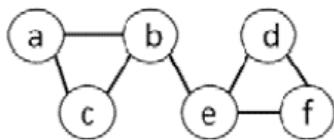
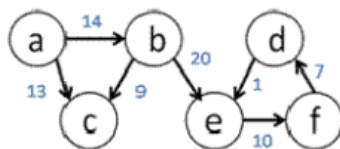


Figure 2. Example of an **undirected** graph where each edge is bidirectional.

An edge can also have a weight. If every edge is associated with a real number (edge weight), then we have weighted graph.



Introduction

From the text input file, campus_map.txt, you will build a representation of a campus map in Python using the graph-related data structures that we provide.

Each line in campus_map.txt has 4 pieces of data in it in the following order separated by a single space (space-delimited): the start building, the destination building, the distance in meters between the two buildings, and the distance in meters between the two buildings that must be spent outdoors. For example, suppose the map text file contained the following line:

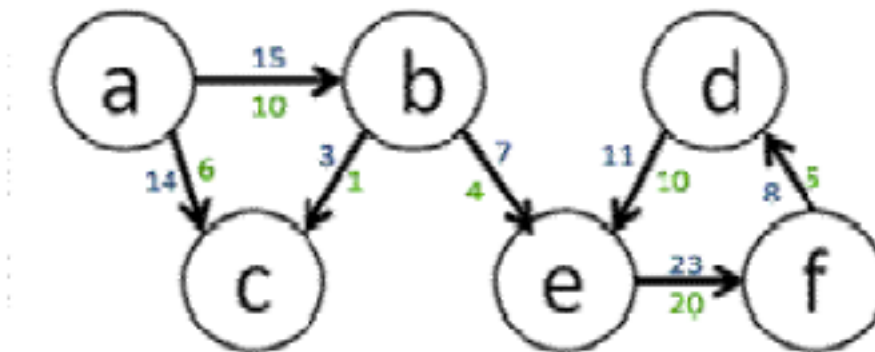
10 32 200 40

This means that the map contains an edge from building 10 (start location) to building 32 (end location) that is 200 meters long, where 40 of those 200 meters are spent outside.

To make the problem interesting, we will say that not every route between a pair of buildings is bi-directional. For example, it may be possible to get from building 54 to building 56, but not the other way around, because the wind that blows away from the first building is too strong.

Problem 1: Creating the Data Structure Representation

In this project, we are dealing with edges that have different weights. In the figure below, the blue numbers show the cost of traversing an edge in terms of total distance traveled, while the green numbers show the cost of traversing an edge in terms of distance spent outdoors. Note that the distance spent outdoors for a single edge is always less than or equal to the total distance it takes to traverse that edge. Now the cost of going from “a” to “b” to “e” is a total distance traveled of 22 meters, where 14 of those meters are spent outdoors. These weights are important when comparing multiple paths because you want to look at the weights associated with the edges in the path instead of just the number of edges traversed.



In graph.py, you’ll find the Digraph, Node, and Edge classes, which do not store information about weights associated with each edge. Extend the classes so that it fits our case of a weighted graph. Think about how you can modify the classes to store the weights shown above. Make modifications directly in graph.py.

Hint: subclass the provided classes to add your own functionality to the new classes.

Deciding what representation to use in order to build up the graph is the most challenging part of the project, so think through the problem carefully.

Problem 2: Building up the Campus Map

Decide how the campus map problem can be modeled as a graph. Write a description of your design approach as a comment under the Problem #1 heading in Proj16.py. What do the graph's nodes represent in this problem? What do the graph's edges represent in this problem?

In the `load_map` function of Proj16.py read in the building data from `mapFilename` and build a directed graph to properly represent the campus map (according to the file). *Hint:* read documentation on file objects in order to figure out how to read each line of the text file. Then parse each line with the help of `string.split`.

Documentation: <https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files>

Problem 3: Find the Shortest Path using Brute Force

We can define a valid path from a given start to end node in a graph as an ordered sequence of nodes $[n_1, n_2, \dots, n_k]$, where n_1 to n_k are existing nodes in the graph and there is an edge from n_i to n_{i+1} for $i=1$ to $k-1$. In Figure 4, each edge is unweighted, so you can assume that each edge is length 1, and then the total distance traveled on the path is 4.

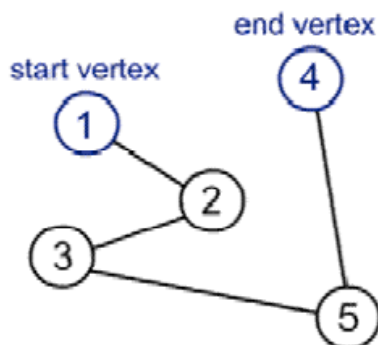


Figure 4. Example of a path from start to end node.

Note that a graph can contain cycles. A cycle occurs in a graph if the path of nodes leads you back to a node that was already visited in the path. When building up possible paths, if you reach a cycle without knowing it, you could get stuck indefinitely by extending the path with the same nodes that have already been added to the path.

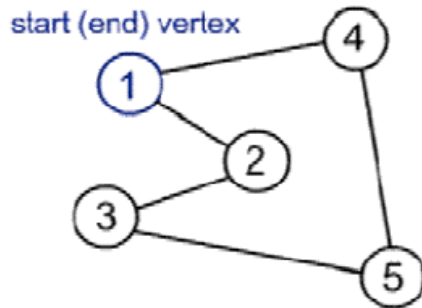


Figure 5. Example of a cycle in a graph.

In our campus map problem, the total distance traveled on a path is equal to the sum of all total distances traveled between adjacent nodes on this path. Similarly, the distance spent outdoors on the path is equal to the sum of all distances spent outdoors on the edges in the path.

Depending on the number of nodes and edges in a graph, there can be multiple valid paths from one node to another, which may consist of varying distances. We define the shortest path between two nodes to be the path with the least total distance traveled. In our campus map problem, one way to find the shortest path from one building to another is to do exhaustive enumeration of all possible paths in the map and then select the shortest one.

How do we find a path in the graph? In the depth-first search algorithm, you try one route at a time while keeping track of routes tried so far. Work off the depth-first traversal algorithm covered in lecture to discover each of the nodes and their children nodes to build up possible paths. Note that you'll have to adapt the algorithm to fit this problem.

Implement the function `bruteForceSearch(digraph, start, end, maxTotalDist, maxDistOutdoor)` so that for a given digraph, you return the shortest path, from the start building to end building, such that the total distance traveled is less than or equal to `maxTotalDist` and that the total distance spent outdoors is less than or equal to `maxDistOutdoor`.

Write a sentence describing what the optimization problem is in terms of what the function to minimize is and what the constraints are for problem 3.

Use the depth-first search approach from lecture to enumerate all possible paths from the start to end node on a given digraph. (Assume the start and end nodes are in the graph). Then select the paths that satisfy the constraint and from that group, pick the shortest path. Return this result as a list of nodes, $[n_1, n_2, \dots, n_k]$, where there exists an edge from n_i to n_{i+1} in the digraph, for all $1 \leq i < k$. If multiple paths are still found, then return any one of them. If no path can be found to satisfy these constraints, then raise a `ValueError` exception. We strongly suggest the use of helper functions to implement `bruteForceSearch`.

Extension

Problem 4: Optimized Method for Finding the Shortest Path

Since enumerating all the paths is inefficient, let's optimize our search algorithm for the shortest path. As you discover new children nodes in your depth-first search, you can keep track of the shortest path that so far that minimizes the distance traveled and minimizes the distance outdoors to fit the constraints.

If you come across a path that is longer than your shortest path found so far, then you know that this longer path cannot be your solution, so there is no point in continuing to traverse its children and discover all paths that contain this sub-path.

Implement the function `directedDFS(digraph, start, end, maxTotalDist, maxDistOutdoor)` that uses this optimized method to find the shortest path in a directed graph from start node to end node such that the total distance travelled is less than or equal to `maxTotalDist` and that the total distance spent outdoors is less than or equal to `maxDistOutdoor`. If multiple paths are still found, then return any one of them. If no path can be found to satisfy these constraints, then raise a `ValueError` exception. We strongly suggest the use of helper functions to implement `directedDFS`.

Test your code by uncommenting the code at the bottom of `Proj16.py`.