Paths and Network Traversal 6/16/2022

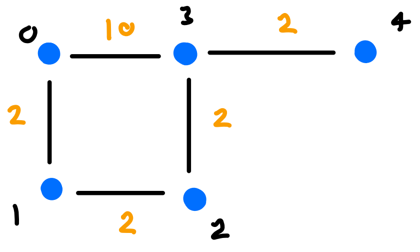
we've already looked at two ways to analyze and understand graphs:

- network centrality: figure out which nodes are most important
- community detection: seperates a graph into groups

Another way we can look at graphs is in terms of how efficiently you can get from one node to another.

Shortest path problem: problem of finding a path between two nodes in a graph such that the sum of the weights of the edges in the path is minimized.

ex)



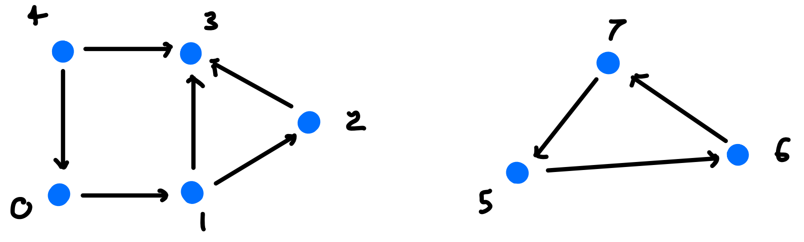
Shortest path from 0→3 : 0→1→2→3

For an unweighted graph, just assume that all of the edges have weight 1.

Applications of the shortest path problem:

- Google maps: finding an optimal route from starting location to ending location
- making schedule of classes: finding shortest path to graduation
- Finding optimal, low price flights
- communication networks: your files are stored in a physical server and you want those files to travel through less nodes to get to the users of your files

For unweighted graphs, we can use an algorithm called Breadth First Search. Breadth First search (or BFS) is an algorithm that finds paths from a given node to all other nodes in the graph that it can reach.



0 can reach 3 and BFS would find the path 0→1→3
0 cannot reach 4 and BFS would determine that there is no path
0 cannot reach 5 and BFS would determine that there is no path

To describe BFS, we first need to learn about queue. A queue is a useful data structure that follows the First In First Out (FIFO, the item in the queue that was put in first is taken out first).

ex)

Q = [ ]	Q = [ ]
Q. enqueue ("task 1")	Q = ["task 1"]
Q. enqueue ("task 2")	Q = ["task 1", "task 2"]
Q. dequeue ()	Q = ["task 2"]
Q. enqueue ("task 3")	Q = ["task 2", "task 3"]

in python, you can use functs append and pop to make a list a queue.

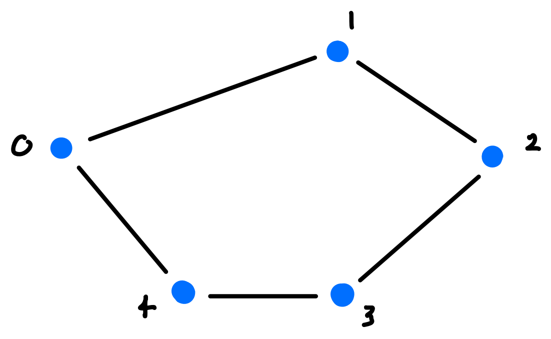
Steps of BFS on a network of n nodes that outputs the shortest path from a given node and all other nodes

```
Create a vector called "visited" of length n of all zeros
Create an empty dictionary called "node Before"
Create an empty queue called "Q"
Create an empty list called "current"
Put the starting node in Q
while Q is not empty:
        set current = Q.dequeue()
        set visited [current] = 1
        loop through all neighbors of current
                if neighbor i has not been visited:   and neighbor i is not already in Q
                        set nodeBefore[neighbor i] = current
                        Q. enqueue (neighbor i)
```

Then we can use nodeBefore to find the shortest path from the starting node to any other node in the graph.

Let's use this algorithm to find the shortest path from $0 \to 2$

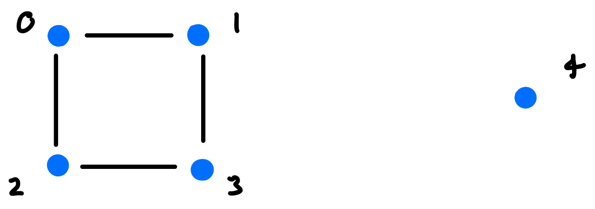
If you missed the demo below, I suggest using one or both of the following resources. They will be different than the algorithm above but still a useful resource: https://youtu.be/pcKY4hjDrxk
https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/



visited : [0 0 0 0 0]
Q : [ ]
current : [ ]
node Before : { 1 : 0,
4 : 0,
2 : 1,
3 : 4 }

nodeBefore [2] = 1
node Before [1] = 0

ex)



use BFS to create nodeBefore.
1) Use nodeBefore to find the shortest path from 2 → 3
2) Figure out how you can use the created variables to identify that there is no path from 2 → 3.