World Airline Route Search

CSCE 3110.501

By: Mario Onofrio, Haley Kang, Marcus Lara Sanchez

Question 1

"I am in city “A”, can I fly to city “B” with less than x connections? Give me the route with the smallest number of connections or tell me there is no such a route."

- How do we traverse the graph?
- How do we print out the path?
- Need a way to make sure that the path does not exceed the x amount of connections.

Meeting Constraints(Q1)

- How do we check?
  - Counter to compare
  - As we jump increment this variable

- How to Traverse to find shortest path?
  - Breadth First Search
  - Uses queue to find shortest path
  - Store cities in separate variable
  - Use stack to reverse queue

Pseudocode

```
queue for bsf q;
map parent (to store origin of each city travelled to)
map, int distance (to store distance from start to each city until destination)
while q is not empty:
     traverse the graph
     if current city is goal and distance is less than maxConnections:
         print path

        if current city is not goal:
            for each neighbor of current city:
                if neighbor is not visited:
                    add neighbor to q
                    update parent and distance maps
    else
        print no route found
```

Question 2

"Give me the route with the smallest number of connections from city “A” to city “D” through city “B” and “C”. (the order of “B” and “C” is not important). Or tell me there is no such a route."

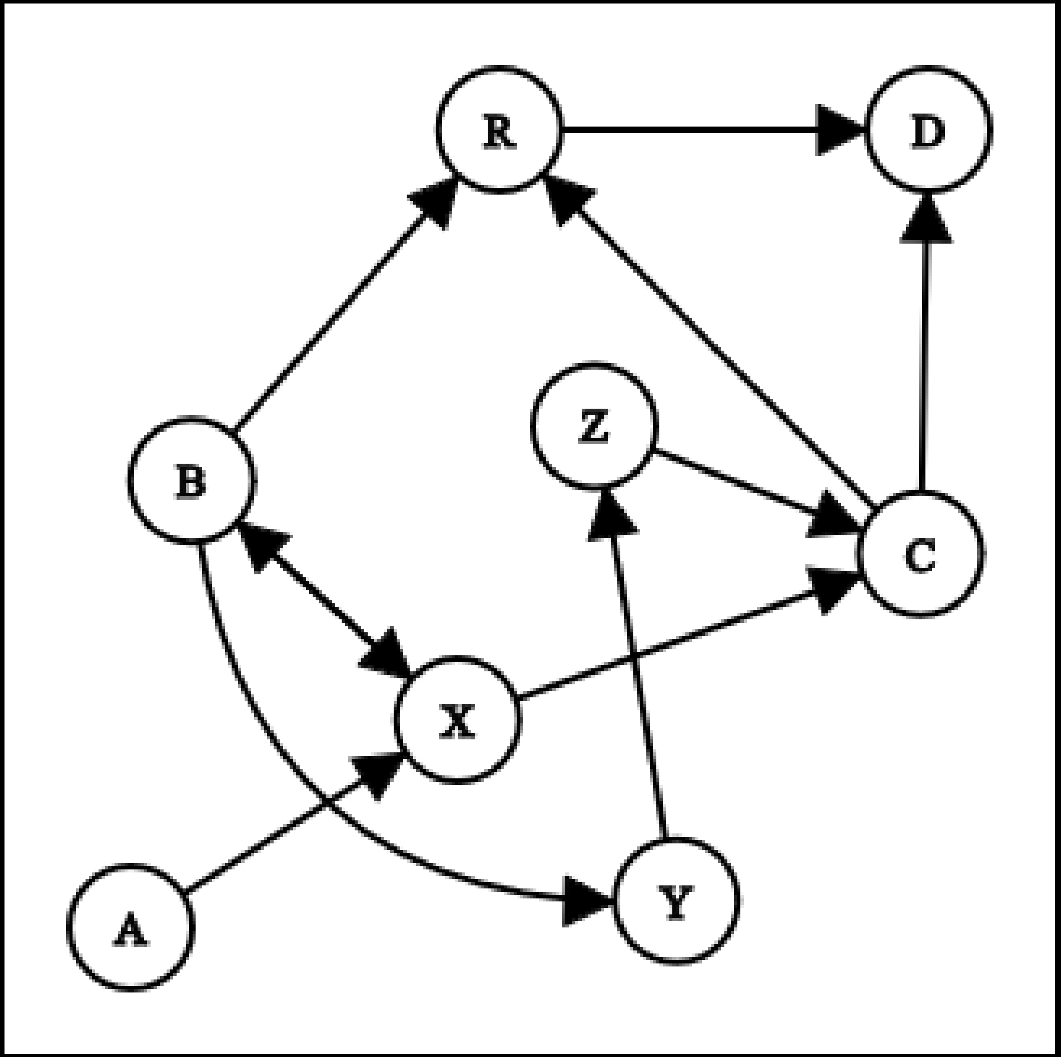
- How do we check for 2 cities?
  - Building off solution to question 1
  - So we use bfs again
  - Use bool to check the node for our current path

Problem!

- Goes through A->X->B->R->Y->Z->C->D
- Correct: A->X->B->X->C->D

- Fix: In order to revisit node
  - Use set<tuple<string rather than just a set<string>
  - Allows for multiple 'states'
    States-instances of paths where some or none of the required cities are visited (4).

T|T	T|F	F|T	F|F



Pseudocode

```
create queee tuple to store city, path, seenB, seenC
set visited with city, seenB, seenC (this is so we can revisit the same node but with different state)
initialize starting point into queue(q.push)



while !q.empty() {
    pop the front of the queue into current
    if current is goal and seenB and seenC are true, we found a path:
        print the path from start to goal through mustPass1 and mustPass2

    if current is in graph:
        for each neighbor of current:
            create a new state with neighbor, new path, new seenB, new seenC
            if this state/node has not been visited:
                add it to visited set
                push the new state into the queue
    }
```

Question 3

"I want to start from city “A”, visit all the cities that can be reached and then come back to “A” using as less connections as possible. Give me a route or tell me there is no such a route. (note: once you come back to “A”, you do not go out anymore)."

- This resembles a Hamiltonian cycle
    - Utilizes a DFS recursive function
- Doesn't need an ADT
    - Valid cites are added to the path and function backtracks if not
- If the path is as big as the graph- that would automatically be the shortest path
- Keep track of the path size- shortest path is stored

Question 4

"I am in city “A”, my friend John is in a different city “B”, and my other friend Ann is in yet another different city “C”. We want to find a city different from the three cities we are in to meet so that the total number of connections among three of us is minimized. Tell me the city we should fly to and the routes for us or tell me there is no such a city."

- How do we figure out the best city for the three friends to meet?
- How do we find the route for each friend?
- How do we know there is no city to meet?

Pseudocode

```
BreadthFirstSearch(city1, parent1);
BreadthFirstSearch(city2, parent2);
BreadthFirstSearch(city3, parent3);
for (iterate through graph) {
    currentCity = iteratorsFirstValue;
    if (totalNumberofConnections < currentMinimumConnections) {
        currentMinimumConnections = totalNumberofConnections;
        bestCity = currentCity;
    }
}

if (bestCity is emptyString) {
    Print("No common meeting city");
}
```