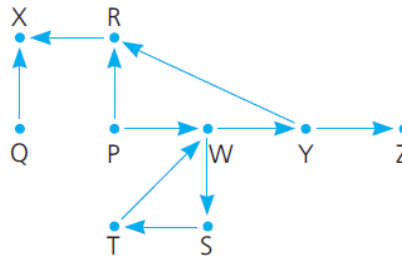


## Project 4: Find Route in Map (Backtracking using a Stack)



For this project you will **design** and **implement** two classes (`City` and `RouteMap`) to find a route, if one exists, from origin to destination city, given a particular map.

This project will build on ideas we have worked with already (reading input from a csv file, working with abstractions - i.e. a `City` and a `RouteMap`).

However this time you will be responsible for the design!



**The idea:** In a map, as the one pictured above, we say that R is **adjacent** to P because there is an arrow going from P to R. However, P is NOT adjacent to R because you cannot go from R to P following a single arrow. A **route** from an origin city to a destination city is a sequence of cities s.t. each city in the sequence is adjacent to the previous one. As we discussed in lecture, it is possible to find out if there is a route from an origin city to a destination city by implementing **backtracking using a stack**

- Starting from the origin, push adjacent cities that have not yet been visited onto the stack, and mark them as having been visited.
- If you reach a dead end (no more unvisited adjacent cities available from the city at the top of the stack), **pop the stack to backtrack**.
- Stop when you either have the destination at the top of the stack (found route!) or the stack is empty (there is no route)

**The input:** The input will be in the following format:

- The first line will be a comma-separated list of city names. For example, for the RouteMap pictured above, **the first line** in the corresponding input file will be:

P,Q,R,S,T,W,X,Y,Z

- The remainder of the file will be a comma separated list of city pairs separated by '–', where the second city in the pair is adjacent to the first. For example, for the RouteMap pictured above, the remainder of the file will be:

P–R,P–W,Q–X,R–X,S–T,T–W,W–Y,W–S,Y–R,Y–Z

You may assume that the city names are unique (no two cities have the same name) but not necessarily single letters, and that the input file is in the correct format.

## Design and Implementation:

You will design and implement the two classes: `City` (`City.hpp` and `City.cpp`) and `RouteMap` (`RouteMap.hpp` and `RouteMap.cpp`)

**Design Requirements** - to successfully implement backtracking using a stack you need to:

- Be able to mark a city as having been visited
- Find out what cities are adjacent to any given city

## Furthermore

- The **RouteMap** class **must have** the following **3 public methods** (you may and should add any public and private members that you deem necessary to implement and support the following)

```
/**
@param input_file_name of a csv file representing a route map where the first
    line is a comma-separated list of city names, and the remainder is a
    comma-separated list of city-pairs of the form A–B, indicating that B
    is adjacent to A (i.e. there is an arrow in the map going from A to B)
@post this depends on your design, the input data representing the map must
    have been stored. Cities must be stored in the same order in which
    they are read from the input file
**/
bool readMap(std::string input_file_name);
```

```

/**
@return a pointer to the city found at position, where
        0 <= position <= n-1, and n is the number of cities,
        and cities are stored in the same order in which they appear
        in the input file
**/
City* getCity(size_t position);

/**
@return true if there is a route from origin to destination, false otherwise
@post if a route is found from origin to destination, it will print it to
      standard output in the form ORIGIN -> ... -> DESTINATION followed by a
      new line
**/
bool isRoute(City* origin, City* destination);

```

For example, with the map in the above example,  
`isRoute(T_ptr, Z_ptr);` will return **true** and print

new line

T -> W -> Y -> Z

**NOTE: `isRoute` MUST explore adjacent cities in the same order in which they were read from the input file.**

**Should you submit a program that mimics the output instead of solving the problem as per this specification, I reserve the right to take away points even if Gradescope gives you full credit. Moreover, if your name is not in your submission's comment preamble you will lose 10 points.**

**Project Notes:** For this project I suggest that you use the STL stack for backtracking (`#include <stack>`) as well as STL vectors or any other container you may deem appropriate to represent and store your data.

<https://en.cppreference.com/w/cpp/container/stack>

**Extra Credit:** Add exception handling for reading an empty file and calling `getCity` with position greater than the number of cities. When catching the exception your program should output "Catching" (for grading purposes output to `cout` not `cerr`). In such cases, a call to `isRoute` should still return **false**.

**Testing:** You should design and test incrementally. Write a main function to test your `City` class. Instantiate `City` objects and test every member function.

Next implement and test your `RouteMap` class, also incrementally. You should come up with different input files and test that your `RouteMap` methods are correct on different test cases (there is a route, there is no route, there is a cycle, etc.)

## **Submission:**

You must **submit** `City.hpp`, `City.cpp`, `RouteMap.hpp` and `RouteMap.cpp` (4 files)

**Your project must be submitted on Gradescope. The due date is Friday April 19 by 6pm. No late submissions will be accepted.**

## **Have Fun!!!!**