# Lab 07 / Assignment 02– new task

## Find a route between different connected cities.

(You can also solve the task using recursion, but in this Lab use stack)

The interaction of the program with user may be as follows:

Please enter filename storing a network: pb.txt
Enter the name of starting city: Kasur
Enter the name of destination: Bakhar

Path is Kasur -> Sahiwal -> Jhang -> . . . -> Bhakar

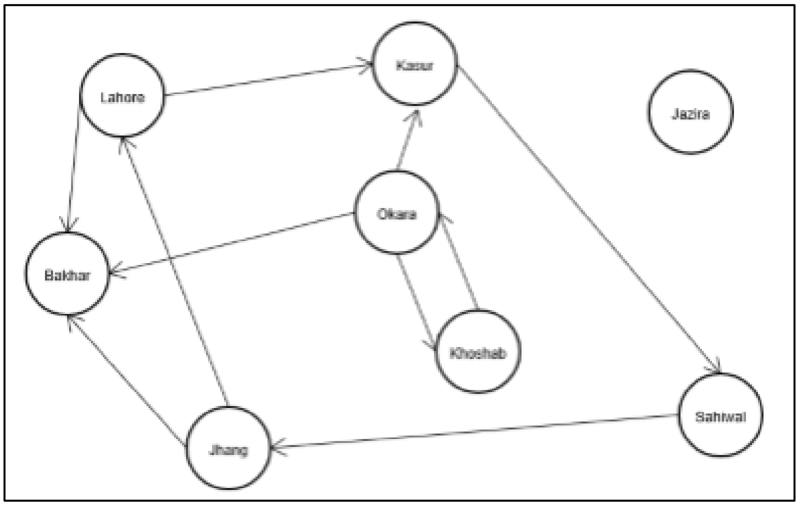
Or Kasur is not a valid city, please re-enter options,

Or Bakhar is not a valid city, please re-enter options,

Or there is not path from Kasur to Bakhar, please re-enter option.

File pb.txt may have data for picture like

```
8
0 Lahore, 2 1 3
1 Kasur, 1 7
2 Jazira, 0
3 Bakhar, 0
4 Okara, 3 1 3 6
5 Jhang, 2 0 3
6 Khosab, 1 4
7 Sahiwal, 1 5
```



**Explaination**: First line consists of the number of cities following by their details one per line. Starting integer on a line is the index no (consecutive integers starting from 0) of a city followed by name and a comma (,). After comma an integer denotes number of outgoing connections following by the indices of those cities.

Read file data into a list of type CityData.

**class CityData**

```
    name # string
    outConCount # int
    outCons # list of ints
    seen # boolean # initialize as false and will be used by main algo
    predecessor # int # initialize as -1 (a sentinel value) and will be used by main algo
```

**Main algorithm:**

```
Search starting city in above list, if found mark it seen, push its index into a stack
while stack is not empty
{
        cc = pop index from stack
        if cc is index of destination
        {
                job done: path is found, display it using predecessor data
        }
        else {
                for all of unseen outCons of cc
                        mark them as seen,
                        set cc as their predecessor
                        and push them into the stack
        }
}
If Stack is empty: Path not found, output not found message.
```

You must complete the following using stack (using recursion, its optional)