*Donghang He*

*CS526*

Final project:*shortest path*

## Search process

The two algorithms use the same search process, but there is a difference in judging the next node. Therefore, the two algorithms are not explained separately here. Only talk about the overall process of the project

1. Read the file
   1. Count the row in graph\_input file
   2. Create a 2D array ***graph***
   3. Read each row of the graph\_input file and convert each row to a string array and save in ***graph***
   4. Traverse the entire array ***graph***, save each character in the first column in ***city*** array list. Use string builder to build a pair of the two cities corresponding to the current location. And get the number of current location which is the distance between this two cities. If distance bigger than 0, save it in the ***graphInput*** Hash map, use pair as the key and the distance as the value
   5. Read the direct\_distance file to ***directDistance*** Hash map, use city as key and distance as the value
2. Input start city
   1. Let the user input a city name as the start node
   2. If ***city*** contains this node jump to next step
   3. If not let the user try again
   4. If user enter exit, exit the system
   5. Also, this function will loop by itself
3. Find way for the start node
   1. Use the start node, ***graphInput*** and ***directDistance*** to get the path
   2. Send the arguments to algorithm
4. Algorithm
   1. Father class: method, basic argument and function are set in this file for both algorithm
   2. After the main function instantiated an algorithm class, it will invoke the *getPath* method. And *getPath* will invoke the *getDistance* and send the start node to it.
   3. Get the closest city, for example if our start node is J. Find the node next to the J in the graph and add to list. And the list will be {A, C, I ,K}, then select the closest next node use dd(v)/w(n, v) + dd(v).
   4. Check this city in *getDistance* function, if it is Z just finish, if it is null mean that we reach the end of this we, we need to backtrack to the last point, if it has a node add this node to ***pathGood*** and ***pathBad*** which mean a path with a backtrack and a path without a backtrack, also we add the distance to the ***shorestDistance***.
   5. If finish just print the sequence of all node which equal to ***pathBad***, and the shortest path ***pathGood***, and the shortest distance

## Pseudocodes

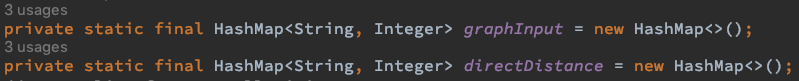
* Algorithm1
* Algorithm getPath()  
   object.getDistance(city)  
    
  Algorithm getDistance(city)  
   nextCity = city  
   if city equals 'Z'  
   then lastCity = city  
   add city to pathGood  
   add city to PathBad  
   else if nextCity is null  
   then from pathBad remove lastCity  
   lastCity = the last one of pathBad  
   add lastCity to pathGood  
   shortestDistance minus the recent add distance  
   else  
   lastPair = nextCity  
   lastCity = second character in nextCity  
    
   for character in nextCity do  
   if usedCityList not contains character  
   then usedCityList add the character  
    
   if pathBad is empty  
   then pathBad add city  
   pathBad add lastCity  
    
   if pathGood is empty  
   then pathGood add city  
   pathGood add lastCity  
    
   shortestDistance plus the distance of nextCity  
    
   if lastCity equals "Z"  
   then print the sequence of all node  
   print the shortest path  
   print the shortest path length  
   else  
   getDistance(lastCity)  
    
  Algorithm getNextCity(city)  
   arrayList adjacentCity = getAdjacentCity(city)  
   smallestDD = null  
   maxDistance = MAX VALUE  
   for cityPair in adjacentCity do  
   nextCity = the second character in cityPair  
    
   if nextCity's distance smaller than maxDistance and usedCityList not contain nextCity  
   then maxDistance = nextCity's distance  
   smallestDD = cityPair  
    
   return smallestDD
* Algorithm2

Same function in algorithm 2 is same as algorithm 1, so at here I only show the pseudocode of different part

Algorithm getNextCity(city)  
 arrayList adjacentCity = getAdjacentCity(city)  
 smallestDD = null  
 maxDistance = MAX VALUE  
 for cityPair in adjacentCity do  
 nextCity = the second character in cityPair  
  
 if nextCity's distance smaller than maxDistance and usedCityList not contain nextCity  
 then maxDistance = nextCity's distance + cityPair's graph distance  
 smallestDD = cityPair  
  
 return smallestDD

## Major data structures

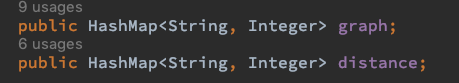
1. Hash map for saving the graph and direct distance



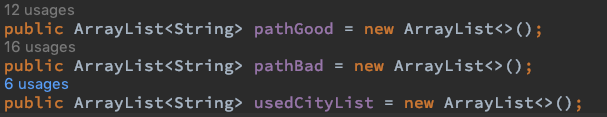
1. Array list for save all cities



1. Hash map for saving the graph and distance in both algorithm which sent from project.java



1. Array list for save the sequence of all node, shortest path and the passing cities



## Input file location

Just at the first line of the project.java file, it is easy to find