Report on the W-O-M-A-N Puzzle

By Julia Sheng

2018/3/27

Discussion:

I wrote the whole program in Java. Thus I stored the graph using Hashtable, where I stored the state name in key and its bordering states in value. Since there could be more than one bordering states which we need to use each of separately in task2 and we do not know how many are them beforehand, thus I chose the type for value to be Arraylist. Be more specific, every time we read one line from source file, we split the line into words by comma, and store them into a string array called tokens. Since for each line in source file, first word is the name of states and followings would be states that share border with first state in the map, thus key would be tokens[0], and we can add each following bordering state(tokens[1], tokens[2], …) into the corresponding arraylist(value of tokens[0]). In other word, this Hashtable is actually an adjacency list of the US map shown as follows:

G

Key Value

Alabama Florida,Georgia,Mississippi,Tennessee

Alaska

Arizona California,Colorado,Nevada,New Mexico,Utah

Arkansas Louisiana,Mississippi,Missouri,Oklahoma,Tennessee,Texas

California Arizona,Nevada,Oregon

……

For solving the puzzle, we also need to keep track of whether a vertex is visited or not and what is the predecessor of each vertex. I still used Hashtable as the data structure for both of them. The reason is that the vertex is not indexed in an integer but in a string (state name), thus it is hard to map vertex to its value use array or Arraylist where position of each element is indexed in integer. However, in a Hashtable, we can store the name of vertex in key and then we can check the value of that key using .get(key) and alternate it if needed using .replace(value,key).

For the search algorithm part, I tried both Depth-first search (in Dsearch(G,s,t)) and Breadth-first search (in Bsearch(G,s,t)) algorithms. In both data structures, we need to construct a Q to record the vertex u that we can travel to next. I used Stack while implementing Q in Depth-first search algorithm and Queue for Breadth-first search algorithm. The algorithm mainly works as follows: first, add the starting state to Q. While Q is not empty, we pop out an element from Q and set to u; then we search whether u has a bordering state starts by characters ‘W’, ‘O’, ‘M’, ‘A’, ‘N’, if so then we add that bordering state to Q. And when our u is equal to t (target state), then we can return the path using pred. I found the result for both search algorithms are very similar, except in Breadth-first search it travels to one more state Arkansas between Oklahoma and Missouri.

In conclusion, the most important data structures are listed following:

Static Map<String,ArrayList<String>>G =new HashMap<String, ArrayList<String>>();

static Map<String, String> pred = new HashMap<String, String>();

static Map<String, Boolean> visited = new HashMap<String, Boolean>();

Stack<String> Q=new Stack<String>(); /Queue<String> queue = new LinkedList<String>();

Sample runs from Task#1:

Enter a state: Florida

Florida has the following neighbors: Alabama, Georgia

Would you like to ask another query(Y/N)?y

Enter a state: Idaho

Idaho has the following neighbors: Montana, Nevada, Oregon, Utah, Washington, Wyoming

Would you like to ask another query(Y/N)?y

Enter a state: Hawaii

Hawaii has no neighbors.

Program from Task#2

**import** java.io.IOException;

**import** java.net.URL;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.LinkedList;

**import** java.util.Map;

**import** java.util.Queue;

**import** java.util.Scanner;

**import** java.util.Stack;

**public** **class** task2 {

**static** Map<String, ArrayList<String>> *G* = **new** HashMap<String, ArrayList<String>>();

**static** Map<String, String> *pred* = **new** HashMap<String, String>();

**static** Map<String, Boolean> *visited* = **new** HashMap<String, Boolean>();

**public** **static** **void** main(String[] args) {

*readData*();

String s = "Washington";

String t = "District of Columbia";

String path=*DSearch*(*G*, s, t);

**if**(!path.equals("fail")){

System.***out***.println("Yes. To get from "+s +" to "+t+", march as follows: ");

System.***out***.println("Depth-First search: "+path);

System.***out***.println("Breadth-First search: "+*BSearch*(*G*, s, t)); }**else**{

System.***out***.println("No. There is no way to get from "+s +" to "+t);

}

}

**public** **static** String DSearch(Map<String, ArrayList<String>> states,String s,String t) {

**for** (String key : *G*.keySet()) {

*pred*.put(key, **null**);

*visited*.put(key, **false**);

}

Stack<String> Q=**new** Stack<String>();

Q.push(s);

**while**(!Q.isEmpty()){

String u= (String) Q.pop();

**if**(u.equals(t)){

String m=t;

String path=m;

**do**{

m=*pred*.get(m);

path=m+", "+path;

}**while**(!m.equals(s));

**return** path;

}**else**{

*visited*.replace(u,**true**);

ArrayList<String> adj=**new** ArrayList<String>();

adj=*G*.get(u);

**if**(adj.size()!=0){

**for**(**int** i=0;i<adj.size();i++){

String v=adj.get(i);

**char** [] cap=v.toCharArray();

**if**(!*visited*.get(v)&&((cap[0]=='W')||(cap[0]=='O')||(cap[0]=='M')||(cap[0]=='A')||(cap[0]=='N')||(v.equals(t)))){

*pred*.replace(v, u);

Q.push(v);

}

}//for

}//if

}

}//while

**return** "fail";

}

**public** **static** String BSearch(Map<String, ArrayList<String>> states,String s,String t) {

**for** (String key : *G*.keySet()) {

*pred*.put(key, **null**);

*visited*.put(key, **false**);

}

// BFS uses Queue data structure

Queue<String> queue = **new** LinkedList<String>();

queue.add(s);

**while**(!queue.isEmpty()) {

String u= (String)queue.poll();

**if**(u.equals(t)){

String m=t;

String path=m;

**do**{

m=*pred*.get(m);

path=m+", "+path;

}**while**(!m.equals(s));

**return** path;

}**else**{

*visited*.replace(u,**true**);

ArrayList<String> adj=**new** ArrayList<String>();

adj=*G*.get(u);

**if**(adj.size()!=0){

**for**(**int** i=0;i<adj.size();i++){

String v=adj.get(i);

**char** [] cap=v.toCharArray();

**if**(!*visited*.get(v)&&((cap[0]=='W')||(cap[0]=='O')||(cap[0]=='M')||(cap[0]=='A')||(cap[0]=='N')||(v.equals(t)))){

*pred*.replace(v, u);

queue.add(v);

}

}//for

}//if

}

}//while

**return** "fail";

}

**public** **static** **void** readData() {

Scanner inS;

**try** {

URL webFile = **new** URL("https://cs.brynmawr.edu/Courses/cs330/spring2018/USStates.txt");

inS = **new** Scanner(webFile.openStream());

String line = inS.nextLine();

**while** (inS.hasNextLine()) {

line = inS.nextLine();

String[] tokens = line.split(",");

**if** (tokens.length == 1)

*G*.put(tokens[0], **null**);

**else** {

ArrayList<String> state = **new** ArrayList<String>();

**for** (**int** i = 1; i < tokens.length; i++) {

state.add(tokens[i]);

}

*G*.put(tokens[0], state);

}

}

} **catch** (IOException e) {

e.printStackTrace();

System.*exit*(1);

} // catch

}// read data

}

Output from Task#2

Yes. To get from Washington to District of Columbia, march as follows:

Depth-First search: Washington, Oregon, Nevada, Arizona, New Mexico, Oklahoma, Missouri, Nebraska, Wyoming, Montana, North Dakota, Minnesota, Wisconsin, Michigan, Ohio, West Virginia, Maryland, District of Columbia

Breadth-First search: Washington, Oregon, Nevada, Arizona, New Mexico, Oklahoma, Arkansas, Missouri, Nebraska, Wyoming, Montana, North Dakota, Minnesota, Wisconsin, Michigan, Ohio, West Virginia, Maryland, District of Columbia