Published in The Startup

Leonidas Boutsikaris Follow
Oct 22, 2019 · 3 min read · Listen

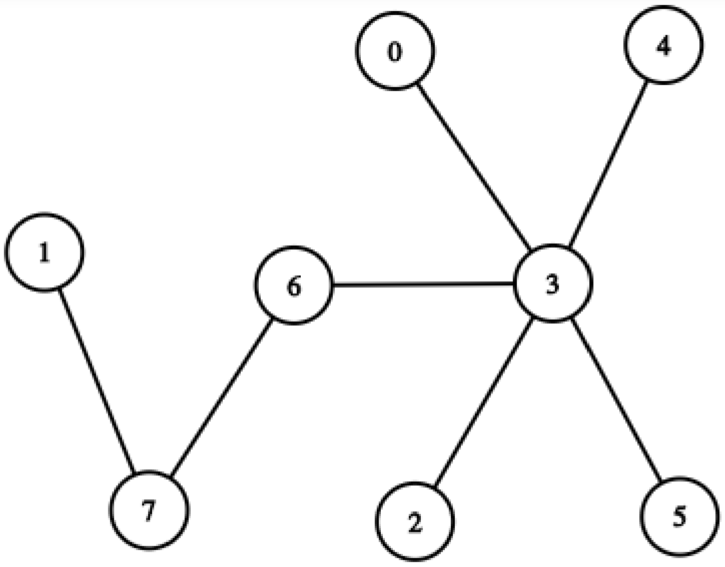
A facility location problem. Where is the optimal placing?

The study of facility location problems (FLP), also known as location analysis, is a branch of operations research and computational geometry concerned with the optimal placement of facilities to minimize transportation costs while considering factors like avoiding placing hazardous materials near housing, and competitors' facilities.

Consider the following scenario

A rural area of a country has decided to improve their health and medicine access. They decided to build a new hospital and a pharmacy but everyone argues about the optimal location of these facilities. Some argue about the population of the cities and others about the distance.

The first step is to create a model of the rural area so we can apply some logic. Using graph theory we will create a non-directed graph of the cities of our rural area. Each vertex will be a city/village and the edges are the connections between them. For example let's take the following graph.



Graph representing the cities of the rural area

Before we try to find the optimal location of the hospital/pharmacy lets define some graph theory terms.

Eccentricity

Eccentricity is the maximum distance from vertex v to any other vertex u of the graph. e(v)= max{d(v, u) | $u \in V(G)$}. For example the Eccentricity of the node 3 is 3 if we consider that every edge has a weight of 1.

The Center of the Graph

The subgraph of a graph G induced by the vertices of G with the minimum eccentricity is called Center of graph G (denoted as center(G)).

In our graph we can see clearly that the center of the graph is the node 3.

Vertex Distance

Vertex Distance is defined as the sum of the distances between a specific vertex and all the vertices of the graph.

distance of v = SUM ($dist(v, u)$) for every $u \in V(G)$

In our case the median of the graph is the node 6.

What about the hospital? what about the pharmacy? Where we should put them?

The hospital must be near the quickest routes. **Take note that an ambulance has to make the same path twice**, one dispatching the vehicle to the location and one returning the patient back to the hospital.

In the other hand the pharmacy just has to be accessible by the citizens.

> The median of the graph can minimize the time because it has the minimum eccentricity. Thats where the hospital must be placed. As you imagined, the center of the graph can be used for the pharmacy because it's more accessible by everyone.

Let's code it.

First thing to do is to transfer the graph in some beep boop computer language. We will use Java.

```java
int[][] inGraph = new int[8][8];

// fill zer0s everywhere
for (int l=0; l<8; l++) {
  for (int c=0; c<8; c++){
      inGraph[l][c] = 0;
  }
}

//fill ones according to the neighbors of the graph

inGraph[0][3]=1;
inGraph[1][7]=1;
inGraph[2][3]=1;
```

```
inGraph[4][3]=1;
inGraph[5][3]=1;
inGraph[6][3]=1;
inGraph[6][7]=1;
inGraph[7][1]=1;
inGraph[7][6]=1;

//we now have the adjacency matrix

printGraph(inGraph);
```

We need to use the Dijkstra algorithm so we can find the all the shortests paths for every node.

```
public static ArrayList<Integer> getShortestPaths(int src, int[][] graph) {
  ArrayList<Integer> totalCosts = new ArrayList<Integer>();
  ArrayList<Integer> prevNodes = new ArrayList<Integer>();
  ArrayList<Integer> minPQ = new ArrayList<Integer>();
  ArrayList<Boolean> visited = new ArrayList<Boolean>();

  minPQ.add(src);
  //1. Set all distances to INFINITY except src node
  for (int i=0; i<graph.length; i++) {
    if (i == src) {
      totalCosts.add(i, 0);
      prevNodes.add(src);
    } else {
      totalCosts.add(i, graph.length + 10);
      prevNodes.add(-1);
    }
    visited.add(false);
  }
  /*
  for (int i=0; i< totalCosts.size(); i++) {
    System.out.println("Index: " + i + " Distance: " + totalCosts.get(i));
  }
  */
  while (!minPQ.isEmpty()) {
    int newSmallest = minPQ.remove(getArrayListMinIndex(minPQ));

    for (int i=0; i<graph[newSmallest].length; i++){
     visited.set(i, true);
     int altPath = totalCosts.get(newSmallest) + neighborDist;

     if (altPath<totalCosts.get(i)) {

      totalCosts.set(i, altPath);
      prevNodes.set(i, newSmallest);
      minPQ.add(i);

     }
    }
   }
  }
 }

 for(int i=0;i<8;i++){
  System.out.print("Cost for "+i+":"+totalCosts.get(i));
  System.out.print(" ////  Previous node visited:"+prevNodes.get(i));
  System.out.println(" ");
 }
 System.out.println(" ");
 return totalCosts;
}
```

These two helper functions will be used by the Dijkstra algorithm to find the minimum and maximum value of the array.

```
public static int getArrayListMaxValue(ArrayList<Integer> srcArray){
 int max = -1;
 int maxIndex = -1;
 for(int i=0; i<srcArray.size(); i++){
  if (srcArray.get(i)>max){
   maxIndex = i;
   max = srcArray.get(i);
  }
 }
 return max;
}

public static int getArrayListMinIndex(ArrayList<Integer> srcArray){
 int min = srcArray.size() + 10;
```

```
      }
    }
    return minIndex;
  }
```

helpers.java hosted with ❤ by GitHub view raw

After getting the paths we can find the eccentricity the center and the median as follows

```
public static ArrayList<Integer> getVertexEccentricity( ArrayList<ArrayList<Integer>> v
  ArrayList<Integer> vertexEccentricity = new ArrayList<Integer>();
  for (int i=0; i<vertexCosts.size(); i++){
    vertexEccentricity.add(getArrayListMaxValue(vertexCosts.get(i)));
  }
  return vertexEccentricity;
}

public static int getCenter(ArrayList<Integer> eccentricities) {
  return getArrayListMinIndex(eccentricities);
}

public static ArrayList<Integer> getVertexDistances(ArrayList<ArrayList<Integer>> verte
  ArrayList<Integer> vertexDistances = new ArrayList<Integer>();
  for (int i=0; i<vertexCosts.size();i++){
    vertexDistances.add(0);
    for (int j=0; j<vertexCosts.get(i).size(); j++){
      vertexDistances.set(i, vertexDistances.get(i) + vertexCosts.get(i).get(j));
    }
  }
  return vertexDistances;
}
public static int getMedian(ArrayList<Integer> vertexDistances) {
  return getArrayListMinIndex(vertexDistances);
}
```

ecce.java hosted with ❤ by GitHub view raw

The main of the class will be as follows.

```
public static void main(String[] args) {
  ArrayList<ArrayList<Integer>> vertexCosts = new ArrayList<ArrayList<Integer>>();
```

```
    for (int i=0; i<inGraph.length; i++){
      vertexCosts.add(getShortestPaths(i, inGraph));
    }
    for (int i=0; i<vertexCosts.size(); i++){
      System.out.println("Vertex: "+ i + " Costs :"+ vertexCosts.get(i));
    }
    vertexEccentricity = getVertexEccentricity(vertexCosts);
    System.out.println("Vertex Eccentricities: " + vertexEccentricity);
    center = getCenter(vertexEccentricity);
    System.out.println("We should place the pharmacy at the center of the graph which is

    vertexDistances = getVertexDistances(vertexCosts);
    System.out.println("Vertex Distances: " + vertexDistances);
    median = getMedian(vertexDistances);
    System.out.println("We should place the hospital at the median of the graph which is

  }
```

main.java hosted with ❤ by GitHub view raw

Conclusion

We checked a simplified version of a problem that can be very tedious if many factors are acknowledged and connected it with graph theory. At first we made a model of the rural cities with a graph. Using the center and the median of a graph we created an automated way of placing these two facilities in the optimal location. Graph theory helps us model and solve problems using existing theorems and applying everyday logic to them. Every application of the graph theory properties depends solely on the problem we're facing, in our case the hospital and the pharmacy locations.

You can check the live demo of the code here

This blog post was made to feed and share our curiosity in collaboration with Nikos Pantelidis.

101