Comp 446: Algorithms Design and Analysis

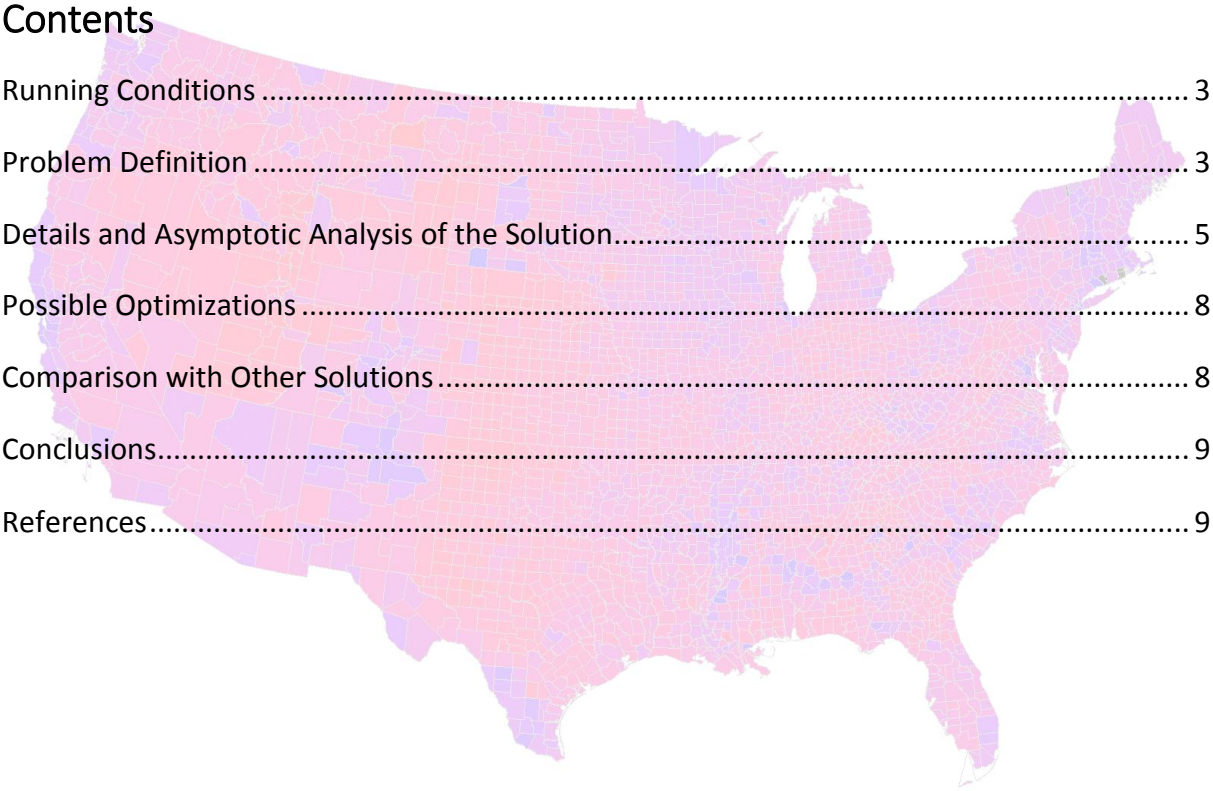
Project: Purple America

OGUN OZ

Instructor: Alptekin KUPCU

Table of Contents

Contents



Running Conditions

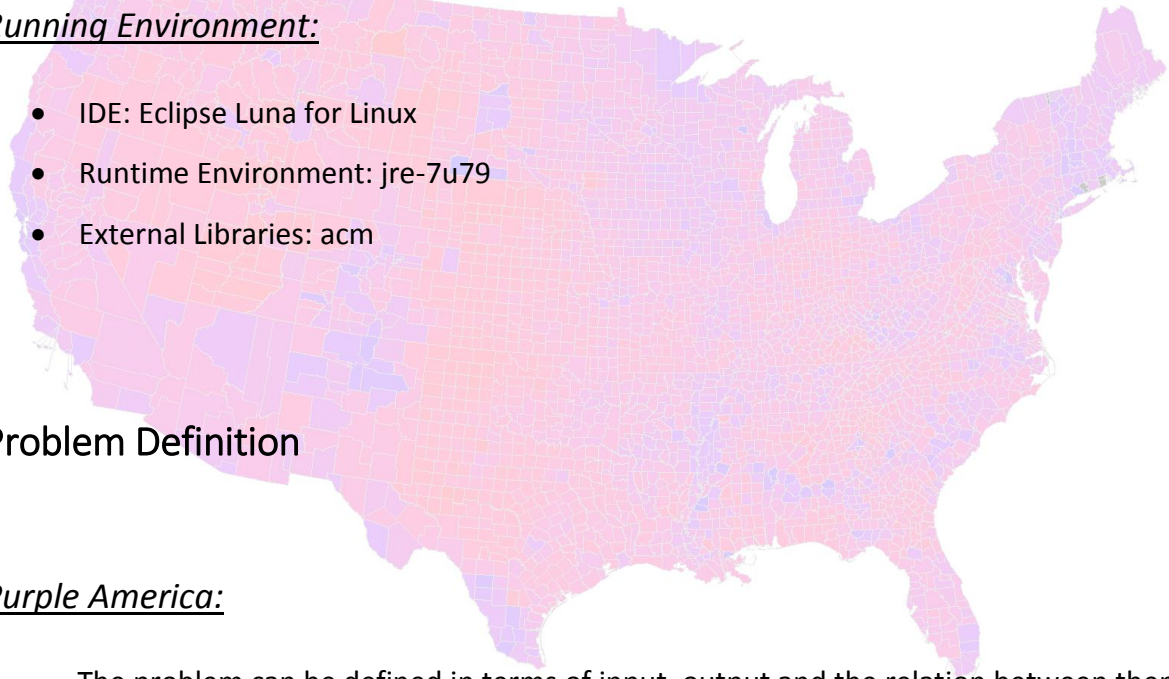
Before defining the problem and analyzing the solution, the running environment and system can be written in the sense of the integrity of the given solution and its source code.

Test Machine:

- ASUS G75VW – Ubuntu 64-bit 14.04
- Processor: Intel® Core™ i7-3610QM CPU @ 2.30GHz × 8
- Graphics: NVidia GeForce GTX 670M 3 GB
- Memory: 15,6 GB

Running Environment:

- IDE: Eclipse Luna for Linux
- Runtime Environment: jre-7u79
- External Libraries: acm



Problem Definition

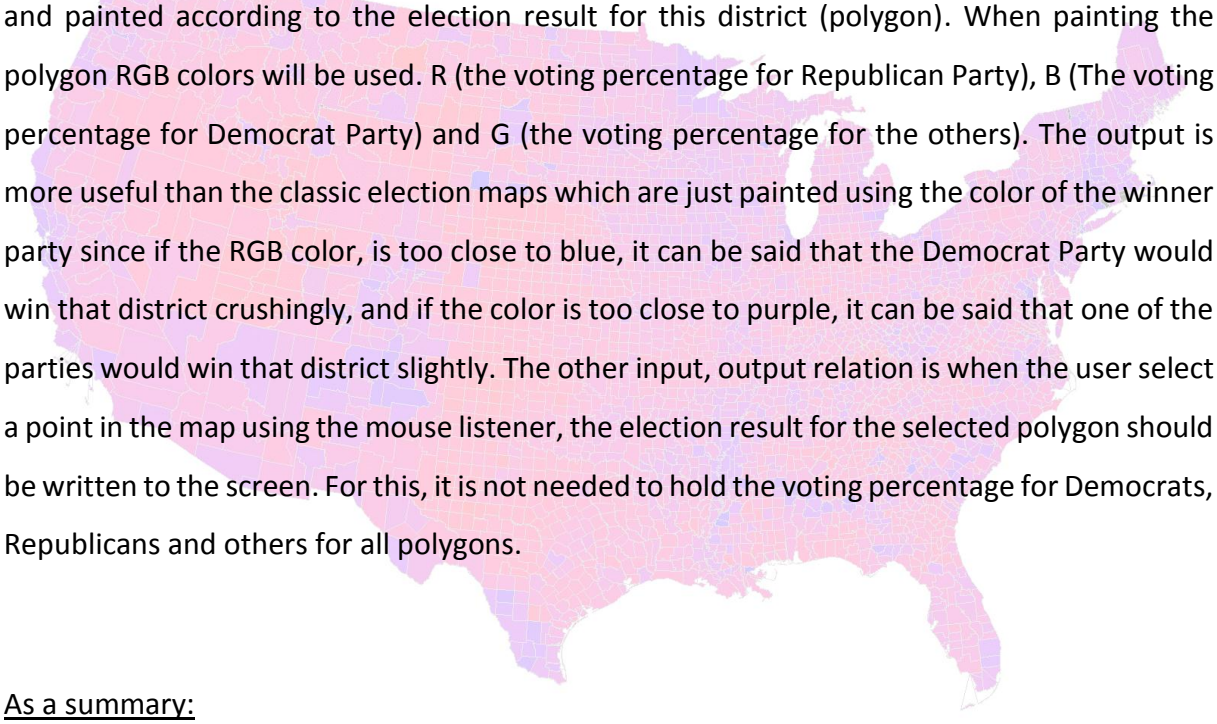
Purple America:

The problem can be defined in terms of input, output and the relation between them. User should give two arguments which are one geometric data file and the election year. In geometric data file, there are coordinates of every district in the map. For instance, if the user wants USA.txt, in geometric data file, there are the coordinates of the states of the America, if the user wants FL.txt, there are the coordinates of the districts of Florida, and if the user wants USA-county.txt, there are the coordinates of the districts of every states of the America. Moreover, beside the geometric data file, there are more inputs which are the election results of the necessary states. Therefore, we can say that the inputs are one geometric file which can be thought the vertices of the polygons, and a list of the election results. So, let's specify

the size of the inputs**. Let v** is the number of all coordinates (vertices) in geometric data file which is the total vertices number of polygons. **Let m** is the state number which will be equal to the number of opened election result file. So, size of the election result data would be n/m. **Let d** is the district (sub region) number in a state. **Let n** is polygon number which is approximately m * d. In the rest of the report, these variables will be utilized as input sizes. There is one more input actually, which is given via mouse listener by user at the running time.

Secondly, the output should be specified properly. The output is a set of filled polygons and the string of election result information for the selected district (polygon).

Finally, the relation between inputs and outputs should be specified precisely. The polygons that represent districts are created using the coordinates in the geometric input file and painted according to the election result for this district (polygon). When painting the polygon RGB colors will be used. R (the voting percentage for Republican Party), B (The voting percentage for Democrat Party) and G (the voting percentage for the others). The output is more useful than the classic election maps which are just painted using the color of the winner party since if the RGB color, is too close to blue, it can be said that the Democrat Party would win that district crushingly, and if the color is too close to purple, it can be said that one of the parties would win that district slightly. The other input, output relation is when the user select a point in the map using the mouse listener, the election result for the selected polygon should be written to the screen. For this, it is not needed to hold the voting percentage for Democrats, Republicans and others for all polygons.



<u>As a summary:</u>

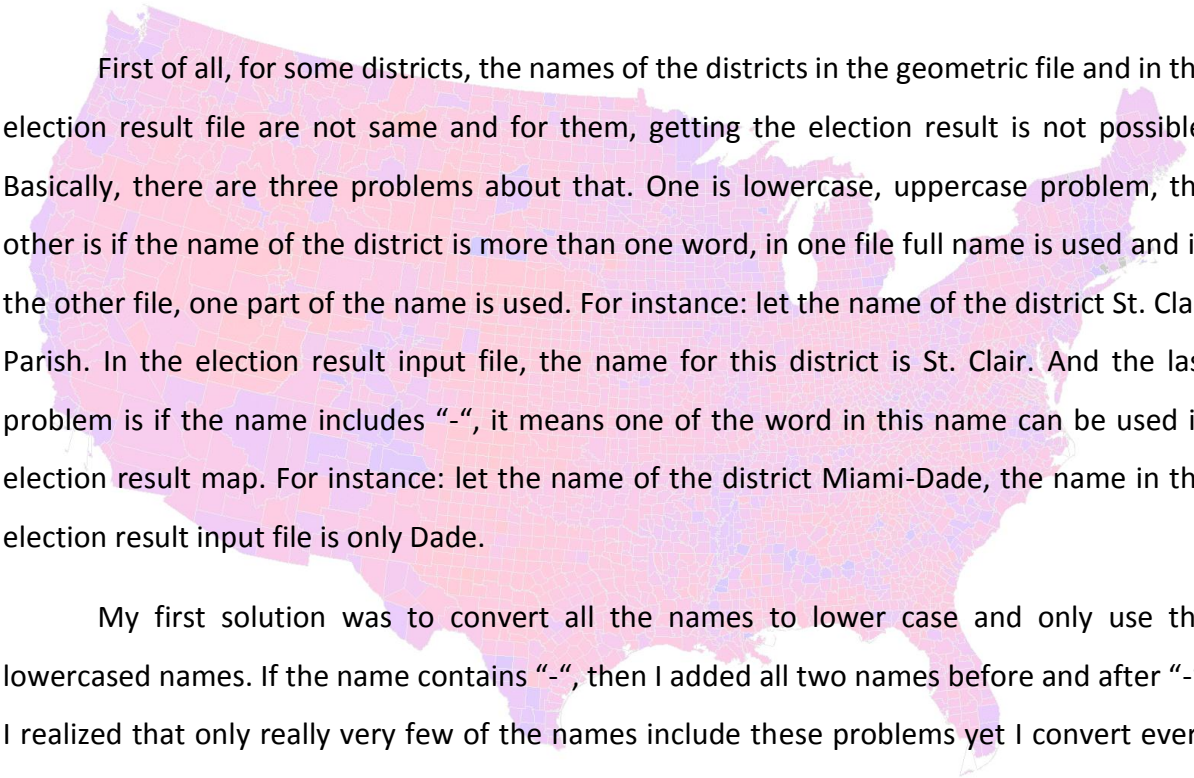
- v = # of all vertices
- n = # of all polygons
- m = # of states
- d = # of districts in a state
- s = # of characters in a district name

Inputs: Coordinates and names of the districts, List of election results per regions, Mouse Listener's coordinates.

Outputs: Filled polygons, a text of election results for one district

Relation: Polygons are created according to the coordinates of the districts, and painted according to the election result for that district using R, G, B colors. The percentages of the parties determine the percentages of red, green and blue color. The election results for the polygon which is selected via Mouse Listener should be shown.

Details and Asymptotic Analysis of the Solution



First of all, for some districts, the names of the districts in the geometric file and in the election result file are not same and for them, getting the election result is not possible. Basically, there are three problems about that. One is lowercase, uppercase problem, the other is if the name of the district is more than one word, in one file full name is used and in the other file, one part of the name is used. For instance: let the name of the district St. Clair Parish. In the election result input file, the name for this district is St. Clair. And the last problem is if the name includes "-", it means one of the word in this name can be used in election result map. For instance: let the name of the district Miami-Dade, the name in the election result input file is only Dade.

My first solution was to convert all the names to lower case and only use the lowercased names. If the name contains "-", then I added all two names before and after "-". I realized that only really very few of the names include these problems yet I convert every character of every name to lowercase, and for all names of districts, I check whether the name contains "-"or not. If the size of the string is s and the total district number is d, therefore the time complexity for this job is **O(n*s)**. I decided to apply these fixing issues, when I cannot find the name. As you remember from the problem definition part, we have to memorize the district names to match the polygon with the election result. So, I have to put election results for all districts and I have to get every election result to paint every district (polygon). So, adding and getting of an ADT, should be fast equally likely. I preferred HashMap which is implemented using a HashTable. We know that putting to and getting from the HashTable, both of them has O(1) expected, O(n) worst case complexity. Putting to an array can be O(1)

in the worst case but getting from an array would be exactly O(n) unfortunately. Moreover, the expected time complexities for HashTable operations are much better than SkipList operations that's costs are O(log(n)) times. So, for holding election results, HashMap is the best solution in terms of the expected time complexities. If there are some problems about I explained in the first paragraph, we can't find in the HashMap and the time cost would be O(n) plus some string operations like converting to lower case or searching word by word takes **O(s** (let say, s is the size of the string)). For all set of string it would be **O(d*s)** time complexity in the worst case. However, if you examine the amortized cost, there are some problems for really few, c constant number of strings and n-c correct strings. It means expected running times for n-c elements is O(1) and for constant c elements is O(n*s). Amortized costs per searching operations would be **O(1)** expectedly and **O(d*s)** in the worst case. Putting operation split the string according to '-'character to fix the last problem. It takes **O(s)** times expectedly and **O(d*s)** times in the worst case.

ElectionMap – ADT Summary

- Searching: O(1) expected, O(d*s) worst case
- Inserting: O(s) expected, O(d*s) worst case
- Space complexity: O(d)

To paint the polygons, we use the ElectionMap which is filled reading the election result file. The significant thing is, we don't have to save every election data to ElectionMap. In my algorithm, the polygons are drawn state by state. It means, the districts of the second state are not drawn before completing all districts of the first state. Actually, it opens a road for Dynamic Programming. Before starting to draw the first district of the first state, we can read the election results for that states. We don't start to draw or read input about the second state before completing drawings and paintings about the first state. So, we don't hold every district's election result at the same time.

The bad news is, after drawing and painting everything, user may want to take the election results for any district. At this point, we have to memorize the election results for every n district. However, memorizing the total vote number is enough for this purpose since getting the R, G, B percentages from the color of the selected polygon is good enough.

Therefore, n times total number should be inserted to an ADT and getting when the user clicks to the polygon. I learned that I can directly takes the Object that I clicked using the mouse's X and Y coordinates. getElementAt(double X, double Y) is the method that is provided me in GCanvas class. When you examine the source code of GCanvas, you see that getElementAt method calls getElement(array of objects at the screen, double X, double Y) method. It searches an object in all objects list to find the correct one. The total object number is n as you know. Then, we can take the necessary polygon in **O(n)** time. Getting the total vote number for a polygon is an **O(1)** expected, **O(n)** worst case problem for a HashTable. Getting the color percentages of the polygon is a constant time problem. So, when the user clicks to a polygon, showing the election results for that polygon has **O(n)** expected and worst case time complexity.

Finally, the running time of the main program starts with the reading a geometric data file. Every line of that file is read, so it takes O(v) time complexity. We have to read the election data file for every included states. Every state's every district should be inserted to HashMap. Insertion has O(1) expected time complexity. We insert d*m elements which is n, so expected running time for total job is O(n). We add every vertex of every polygon, we have totally v vertices, so it takes O(v) time. We have to get election data for every polygon, so it takes O(n) expected O(d*n) worst case time. We have to save every polygon's total vote values. We have n polygons, so it takes O(n) expected, O(n^2) worst case time. Finally, we have to put every polygon's names and total vote values to a HashMap to give to user when she clicks to the polygon. It's cost expectedly, O(n*1) and in the worst case O(n^2). Beside of these, constructing the PurpleAmerica class and mouse operation's time complexities will reach us to the program's final asymptotic analysis.

As a Summary

Complexities of Constructor

- **O(1) time complexity, constant space complexity O(1)**

Complexities of Mouse Operation

- **O(n) expected and O(n) worst case time complexity, O(1) space complexity**

Complexities of the Main Method

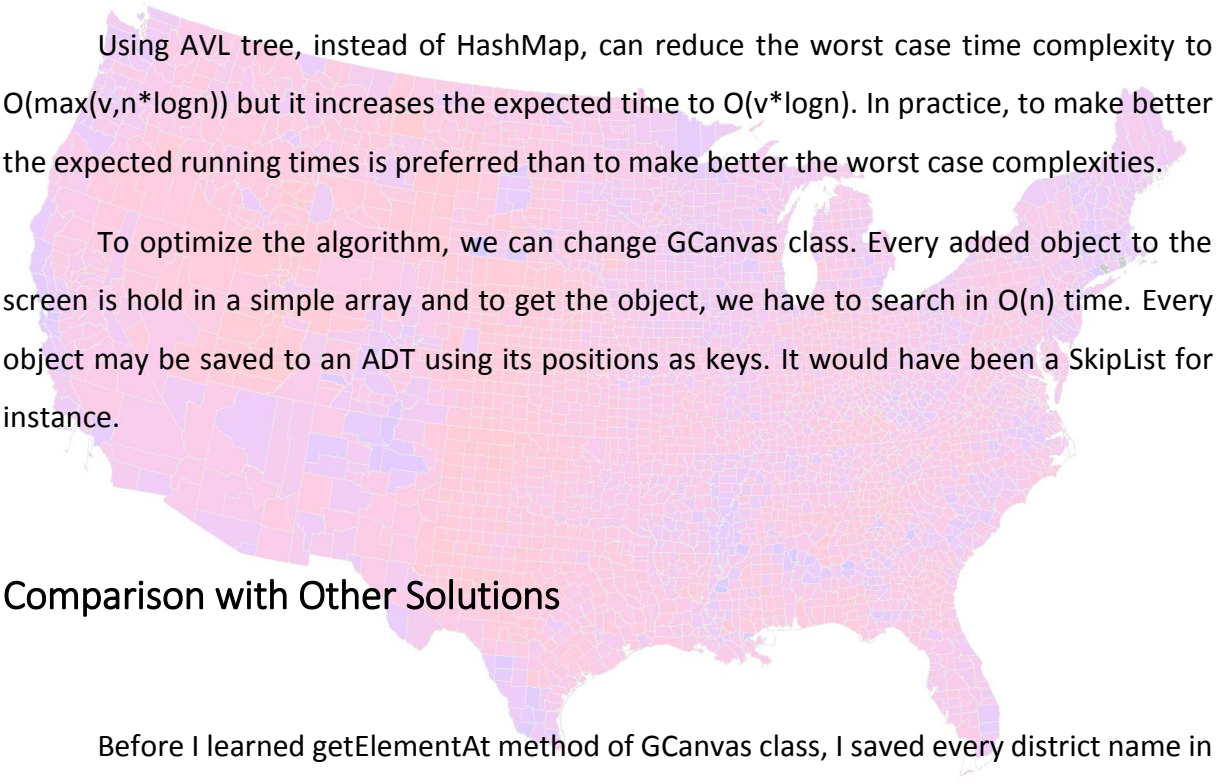
- **O(v) expected and O(max(v, n^2)) worst case time complexity, O(n) space complexity.**

This algorithm has finally O(v) expected, O(max(v,n^2)) time complexity, and O(n) space complexity as it is explained in this section.

Possible Optimizations

Using AVL tree, instead of HashMap, can reduce the worst case time complexity to O(max(v,n*logn)) but it increases the expected time to O(v*logn). In practice, to make better the expected running times is preferred than to make better the worst case complexities.

To optimize the algorithm, we can change GCanvas class. Every added object to the screen is hold in a simple array and to get the object, we have to search in O(n) time. Every object may be saved to an ADT using its positions as keys. It would have been a SkipList for instance.



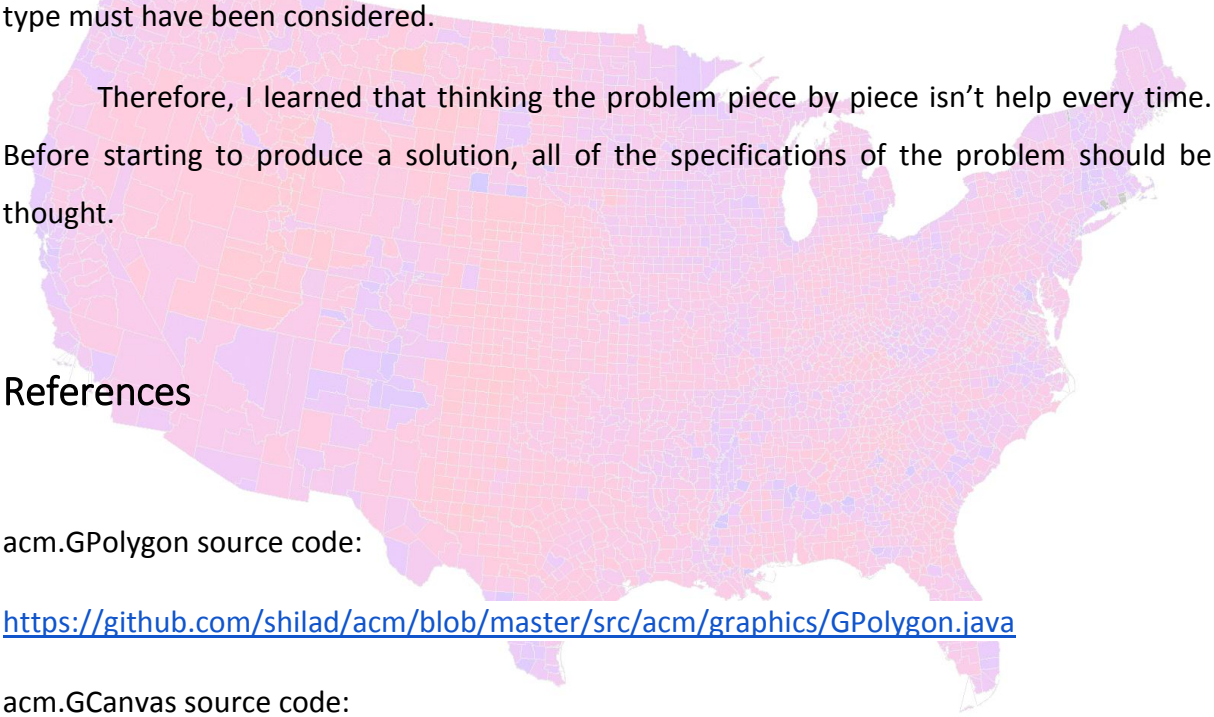
Comparison with Other Solutions

Before I learned getElementAt method of GCanvas class, I saved every district name in its state, and I saved every polygon in its district. This wasn't help me to get the clicked polygon effectively since I had to search the polygons for all states and for all districts in it. I used Java.Polygon class's contains method whether to check it is the clicked polygon. Java.Polygon's contain() method calls its evaluateCrossings() method and this method calls for all vertices v at the screen, linesIntersect() method of Java.Line2D. Getting the election results of the clicked polygon's time complexity would be O(v) instead of O(n) and it is guaranteed that v is much greater than n.

Conclusions

When I see the project, the first thing that I thought was Dynamic Programming because it was possible to solve the first problems first (drawing and painting the districts of a state), and it would lead to the solution of the big problem (drawing and painting the state). This can be applied for all states and whole problem can be solved very easily and efficiently. However, unfortunately, I saw that adding a problem which is not "so" harder than the solved one, can make the problem really hard, and can make the advantages of the solution meaningless. To get the election results for clicked polygon is really affected my simple solution. I had to hold large data in memory, so handling to insert and get data from a data type must have been considered.

Therefore, I learned that thinking the problem piece by piece isn't help every time. Before starting to produce a solution, all of the specifications of the problem should be thought.



References

acm.GPolygon source code:

https://github.com/shilad/acm/blob/master/src/acm/graphics/GPolygon.java

acm.GCanvas source code:

https://github.com/shilad/acm/blob/master/src/acm/graphics/GCanvas.java

acm.GraphicsProgram source code:

https://github.com/shilad/acm/blob/master/src/acm/program/GraphicsProgram.java

Java.Polygon source code:

http://developer.classpath.org/doc/java/awt/Polygon-source.html

Java.Line2D source code:

http://developer.classpath.org/doc/java/awt/geom/Line2D-source.html