ASSIGNMENT 1 – PSSD

JOURNAL

29-Sep-14

From an initial reading of the problem description I was able to see that this was not a trivial problem. It is obviously a graph problem and as such certain decisions need to be made about data structures. These decisions possibly need to be made before creation of an algorithm or concurrently with algorithm design. As my knowledge of graph representations is very basic (I have not worked extensively with graphs before) I will need to research appropriate ways to represent graphs. It is also obvious that I will need to write a function to parse input from a text file. Although seemingly an easy task I have not had to parse anything more than an extremely simple text input before so it will be a good learning experience and help me learn an important skill.

1-Oct-14

From a more detailed reading of the problem description and researching different ways to represent graphs it seems like the most appropriate way to represent the graph will be with an adjacency list. Each town will need to have certain values associated with it obviously an x co-ordinate and y co-ordinate and also a list containing all the towns that this town is adjacent to. I also looked into how to read from a file and write to a file, both processes seem quite simple.

3-Oct-14

The first code I wrote was to parse the input of the text files (as given in the assignment folder), this was mainly a process of trial and error by making minor adjustments and printing out the results of the getline commands to see if I was actually getting what I wanted. The general logic behind the parsing function was to have different modes representing the type of data that is being parsed. While roads are being parsed it will be in “road parsing mode” and will be creating a vector which contains all the links between nodes. The major difficulty I had with parsing input was figuring out a way to convert a given town “name” e.g. 01_0001 to its index into the array of nodes e.g. Nodes[15]. To accommodate for easy conversion between the two representations I used two maps, which essentially serve the role of a hash function. One map converts a town “name” into its index into the array of Nodes and the other converts an index into a town name.

I also created a struct to represent each node. Inside the struct is a double for x, y, a string representing the name of the node and a vector of type node which contains all the nodes that this node is adjacent to.

10-Oct-14

The next class I wrote was one to actually create the graph from the parsed input. A seemingly easy task, but one in which I actually encountered quite a few problems that were mainly as a result of simple oversights. It was very helpful that I had previously created the maps to convert between indexes and town names because this made it a lot easier when it came to creating the adjacency lists for each node. Creating the nodes themselves was quite straightforward as they were simply just structs where all the values can be easily set and altered if necessary.

The major point of issue came from creating the links between all the nodes. I was initially doing the indexing incorrectly simply due to simple oversights. To correct this problem I printed out both my conversion maps to make sure they were correct, which they were, so they were not the problem. I then

went through and printed out each node and all their adjacent nodes and saw that the adjacency lists had not been created properly. I spent some time trying to figure out why this was the case until I realized that it was actually due to the fact that I was testing incorrectly. I had actually created the adjacency lists correctly all along but was not printing my tests correctly which led me to wasting quite a bit of time trying to solve a bug which never really existed.

15-Oct-14

So far I have essentially been avoiding the most difficult part of the assignment, actually coming up with a working algorithm, something, which I believe is quite difficult to do. Intuitively it makes sense to me that one would want to try and make the edges as short as possibly (or possibly just quite short) in order to avoid crossings. This seems to make sense because theoretically the longer an edge is the more "opportunities" it has to cross over with another. I have, however, struggled to come up with any sort of algorithm based around this concept as I am still quite new to graphs in general and have never looked into considering how they can be drawn in such a way as to minimize edge crossings.

Researching into methods I can use to solve this problem there are quite a few mathematically complex ways to eliminate edge crossings, methods which I feel like are far beyond my scope of knowledge. There are however some methods which work by using more real world physics models to generate a graph which is "aesthetically pleasing" i.e. has minimal edge crossings which I think I may be able to adapt for this assignment.

18-Oct-14

Reading more into force based algorithms I have found that they seem to be suitable for the scope of this assignment, possible to code and actually quite easy to understand. From reading academic articles on the general outline of these algorithms I have come to understand how they work in minimizing edge crossings. [12]There are many forced directed methods that give a general outline of how to go about creating a graph with minimal edge crossings. The main concept of these algorithms is that each node is repelled by every other node (calculated by some force based on Coulombs law) and each edge "brings together" its two nodes based on Hookes law. Intuitively these algorithms make sense and do not appear to be too difficult to code, the difficulty seems to be coming up with a good approximation for both Hooke's and Coulumb's law. Another point of difficulty is figuring out how many iterations to run this forced based simulation for as the running time is quite high so obviously cannot just be ran forever.

20-Oct-14

In first attempting to write code based on these algorithms it was obvious that a few changes would be needed before beginning to write actual code. Firstly the structs would need to now each have their own values containing the total vector force acting on them (where the vector corresponds to x and y directed force on the Cartesian plane).

Although conceptually the algorithm is not difficult to understand, implementing it can be quite difficult. For my first approach, based on Coulombs law (which itself is based on inverse distances) I calculated each force to be an incredibly simple (1/(r^2)), making the repulsion forced inversely proportional to the square of the distances between each node-pair. For the attractive force based on Hooke's law I also took

---

[1] http://arxiv.org/pdf/1201.3011.pdf Stephen G. Kobourov , Spring Embedders and Force Directed Graph drawing Algorithms, University of Arizona, 2012

[2] ftp://ftp.mathe2.uni-bayreuth.de/axel/papers/reingold:graph_drawing_by_force_directed_placement.pdf Thomas M. J. Fruchterman, Edward M. Reingold. Graph Drawing by Force Directed Placement, University of Illinois (1991)

another naive approach, I simply calculated the Euclidian distance (as before) and applied a force based on this distance.

The difficulties I had with coding this approach were quite seemingly quite obvious oversights on my behalf, rather than logical errors with the algorithm. Initially I was trying access my Nodes by calling them as such

Node n = allNodes[i]

Where i was the index of the array going through each node (the same process was using to access Node u). Obviously this was incorrect as it was making a copy of the struct each time I was accessing it rather than actually changing the values of the struct. This problem caused me more grief than it probably should have, mainly because I haven't had to use many pointers so far in PSSD. The obvious solution was to simply use a pointer

Node* n = &allNodes[i];

So that when changing the x and y co-ordinates of a nodes I am changing the actual values stored within the node and not changing the values of another struct that was created (which changing its values has no impact on the system).

Initially I tested the code by simply having cout statements to output onto the terminal the new positions of the nodes, then copied them into a template and ran them through the provided count crossings Java file. The outcomes seemed to be reasonably positive, reducing the crossings to at least 10% of the original amount of crossings after running for a number of iterations equal to the amount of towns in the input.

23-Oct-14

The first thing I decided I needed to do was to develop a stronger testing harness as not to waste a lot of time copy and pasting test cases, especially when trying to see the effect of a minor change in code. To write a better testing harness I developed a function which takes in an input (vector of Nodes, amount of towns, roads) and creates a text file in the format required by the CountCrossings class. To do this I essentially followed the same process as parsing the input file but this time around instead of reading data the function was writing data. Once again although this isn't directly related to the creation of an algorithm it is still an important part of writing a program (as testing is extremely important for anything more than a completely trivial question). I felt like even though this was a simple endeavor it allowed me to learn quite a bit about writing to files in C++.

After writing a much more efficient way of testing I began to change the way I was calculating forces (still based of Coulumb's and Hooke's laws) in order to find an answer with minimal crossings. I tried many small changes to these calculations in order to come up with a minimal edge crossing result.

Below are attached images of the tests with the changes to the approximations of Hooke’s and Coulomb’s law indicated. The process of finding appropriate approximations for both laws was one of trial and error until I came up with one that produced good results for all input cases.

INITIAL CROSSINGS

run1in - 37
run2in - 74
run3in - 4700
run4in - 9128
run5in - 66018

Using coulomb approximation $1/d^2$
k
Hooke's approximation of Δx on x
Δy on y

run1 = 5 (removed 32 ~85%)
run2 = 11 (removed 63 ~85%)
run3 = 837 (removed 3863 ~82%)
run4 = 469 (removed 8659 ~94%)
run5 = 4007 (removed 62011 ~93%)
promising initial results

Using coulomb approximation of $1/d^2$
using Hooke approximation of Δx on x, Δy on y } *0.1
(attempting to lessen the effect of the springs)

run1 = 8 (removed 29 ~78%)
run2 = 29 (removed 45 ~60%)
run3 = 386 (removed 4314 ~91%)
run4 = 950 (removed 8178 ~89%)
run5 = 854 (removed 65854 ~98%)

Using coulomb approximation of $1/d^2$
Using Hooke approximation of Δx on x
Δy on y

dampening total force by 90%.

run1 = 8 (removed 29 ~78%)
run2 = 23 (removed 51 ~68%)
run3 = 347 (removed 4353 ~92%)
run4 = 827 (removed 8301 ~90%)
run5 = 2473 (removed 63545 ~96%)

As can be seen overall results are promising but a balance must be found between both forces in order to minimize crossings

Using coulomb approximation
(crocds *2) * $\Delta x / d^2$ on x
(crocds *2) * $\Delta y / d^2$ on y

Hooke's approximation
Δx on x, Δy on y } *0.1

run1 = 4 (removed 33 ~ 89%)
run2 = 9 (removed 69 ~ 93%)
run3 = 59 (removed 4815 ~ 98%)
run4 = 49 (removed 9074 ~ 99%)
run5 = 174 (removed 65844 ~ 99%)

Using coulomb approximation
(crocds *2) * Δx on x
(crocds *2) * Δy on y

Hooke's approximation
Δx on x, Δy on y } *0.5

run1 = 0 (removed 37 100%)
run2 = 0 (removed 74 100%)
run3 = 32 (removed 4688 99%)
run4 = 44 (removed 9084 99.5%)
run5 = 131 (removed 65887 99.6%)

These approximations for both Hooke's and Coulomb's law seem to produce the best output I have been able to generate.

These approximations for the calculations for force were found through trial and error, changing values until more desirable results were found.

27-Oct-14

As the marking script has been posted online I wrote an extremely simple makefile (as my entire code is within a single file) and uploaded the code. Initially I had some problems with the indentation with the makefile but they were easy to fix (I have not had much experience using makefiles). As expected my code passed all of the test oracles resulting in an output of 100% meaning that no further changes need to be made to the code!