CpSc 2120: Algorithms and Data Structures

Instructor: Dr. Brian Dean — Fall 2020
Webpage: `http://www.cs.clemson.edu/~bcdean/` — MWF 9:05-9:55
Handout 4: Lab #3 — Flour 132

1 Enforcing Social Distancing

As we all know, maintaining social distancing is one effective line of defense in mitigating the spread of COVID-19. In an attempt to build a high-tech solution to detect violations of social distancing, Evil Professor McFaileykins proposes to outfit all students with GPS transmitter collars, which constantly broadcast the 2D position of each student on a map of campus. Now the only problem is devising an algorithm that can quickly test if students are standing too close to each-other. For starters, Professor McFaileykins wants to be able to quickly determine the closest distance between any pair of students.

In this lab exercise, you will use "spatial hashing" to find the closest pair among 1 million points spread uniformly across a unit square in the 2D plane. Although this problem is easily solved in $\Theta(n^2)$ time by comparing all pairs of points, this solution is too slow for input sizes n on the order of 1 million, as is the case here.

The input file for this lab is:

`/group/course/cpsc212/f20/lab03/points.txt`

It contains 1 million lines, with two space-separated real numbers per line giving the x and y coordinates of a point. All points (x, y) live strictly inside the unit square described by $0 \leq x < 1$ and $0 \leq y < 1$. Remember that the distance between two points (x_1, y_1) and (x_2, y_2) is given by

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

As a small caveat, the C++ library has a function named "distance" already defined (which does something other than computing geometric distance above), so you if you write a function to compute distance, you should probably give it a name other than "distance" or else obscure compiler errors might result.

There are no source code files to start with; you will need to write your program starting from scratch this time.

2 The Approach

To find the closest pair of points quickly, you will divide the unit square containing the points into a $b \times b$ grid of square "cells", each representing a 2D square of size $1/b \times 1/b$. Each point should be "hashed" to the cell containing it[1]. For example, if you are storing the x coordinate of a point in a "double" variable named `x`, then `(int)(x * b)` will scale the coordinate up by b and round down to the nearest integer; the result (in the range $0 \ldots b-1$) can be used as an one of the indices into your 2D array of cells. The other index is calculated the same way, only using the y coordinate.

After hashing, each point needs only to be compared to the other points within its cell, and the 8 cells immediately surrounding its cell – this should result in many fewer comparisons than if we simply compared every point to every other point. You will need to select an appropriate value of b as part of this lab exercise. You may want to consider what are the dangers in setting b too low or too high.

Internally, the grid of cells should be stored as a 2-dimensional array of pointers to linked lists, with each linked list containing the set of points belonging to a single cell. The array of cells must be *dynamically* allocated (with the *new* command) and subsequently de-allocated at the end of your program (with the *delete*) command. This means that, since each cell of the table is itself a pointer to a node of a linked list, the top-level variable representing the 2D array of pointers will be of type "Node ***" (with three *'s!), assuming "Node" is the name of the structure representing a node in your linked lists.

Your program should consist of the following major steps:

1. Allocate and initialize 2D array of pointers to linked lists.
2. Read input file, inserting each point into the appropriate linked list based on the cell to which it maps.
3. For each point, compare it to all the points within its cell and the 8 adjacent cells; remember the smallest distance obtained during this process.
4. De-allocate 2D array and linked lists.
5. Print out minimum distance.

Part of this lab also involves figuring out a good choice for the value of b. Please include in a comment in your code a brief description of why you think your choice of b is a good one.

3 Grading

For this lab, you will receive 8 points for correctness and 2 points for having well-organized, readable code. Zero points will be awarded for code that does not compile, so make sure your code compiles on the lab machines before submitting!

Final submissions are due by 11:59pm on the evening of Sunday, September 13. No late submissions will be accepted.

[1]Note that some might object to the word "hash" here, since there isn't anything about this mapping that is intentionally jumbling things up – the "hash function" here is more of just a re-scaling operation. However, it is mapping elements into cells in a way that is reminiscent of a hash table, so use of the term "hash" is perhaps still reasonable.

4 If Bored...

For those who finish early or are interested in more challenging extensions of the lab, consider the following optional ideas for fun:

- We had earlier assumed that it was only necessary to check each point against the other points in its cell and in the 8 neighboring cells. Is this always a valid assumption, or does it only hold for certain values of b?
- Can you come up with an input that will make this algorithm run very slowly – say, in $\Theta(n^2)$ time?