C++ Programming: The Euclidean Distance

Nthikeng Letsoalo

University of the Witwatersrand — August 18, 2018

Introduction

The euclidean distance is a well known distance metric for calculating the straight line distance between 2 points. Given two points **p** and **q** the euclidean distance is given by:

$$d(\mathbf{p},\mathbf{q}) = \sqrt{(p_0 - q_0)^2 + (p_1 - q_1)^2 + ... + (p_D - q_D)^2}$$

$$d(\mathbf{p},\mathbf{q}) = \sqrt{\sum_{i=0}^{D}(p_i - q_i)^2} \tag{1}$$

Where D is the dimension of the points.

Make sure you fully understand the formula given above as you will be using it to code the majority of the tasks found in this lab. If you are unsure of what this formula represent and how it works – remember that *Google* is your friend and it will not judge you for not knowing this elementary formula

1 Compiling Your Code

To compile this particular lab run the following commands:

Command Line

```
$ make
$ ./euclideandistance
```

2 Maximum Distance in Array

Given a set P of N euclidean points, your task is to find the maximum distance between all pairs of distances in the set. Thats is, find the points that are furthest apart. For a guide, consider the diagram shown below.

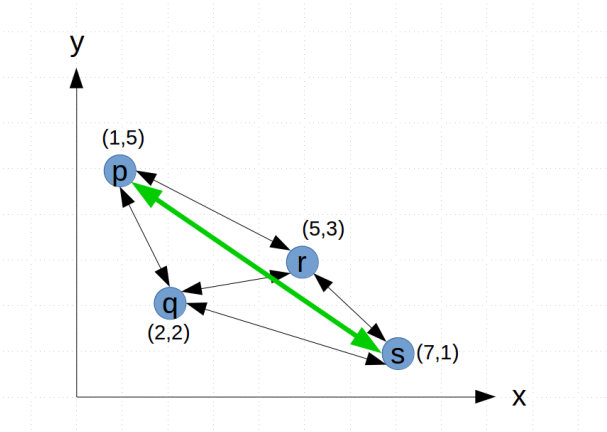

Figure 1: Maximum Distance

It is easy to see from the diagram that the points **p**=[1,5] and **s**=[7,1] will give you the maximum distance because they are the pair that is furthest apart. You can verify this by computing all the distances for points and arranging in ascending order.

2.1 Case 1: 1-Dimensional Points

To start off, we will consider 1-D points, which just means that all your points are single valued real numbers. i.e. The set P is given by $P = \{p_1, p_2, ..., p_n\}$ where $p_i \in \mathbb{R}$. This is convenient because it allows us to just store the numbers in normal C++ arrays.

Now your task is to complete the following functions so you that you can solve the *maximum distance* problem:

Listing 1: functions to complete

```
1  void generate1D_Points(double *arrPoints, size_t n){
2      //TODO: write funcitonality for this method here
3  }
4
5  double MaxDistance1D(double *arrPoints, size_t n){
6      //TODO: write funcitonality for this method here
7  }
```

When you are done with the code above complete the main function so that you can compile and run your code

Listing 2: main function

```
1
2  int main(void){
3      /*TODO: 1.) create an appropriate array to store the euclidean points
4       *      2.) call the appropriate generate1D/2D/3D_Points function
5       *      3.) call the appropriate MaxDistance1D/2D/3D function to calculate the m
6       *      4.) output the maximum distance to the terminal
7       */
8  }
```

Question 1

Consider the algorithm you have designed to solve the problem. Using Big-O notation, what is the complexity of your algorithm?

(a) $O(n)$.

(b) $O(2n)$.

(c) $O(2n + 2n)$.

(d) $O(n^2)$

2.2 Case 2: 2-Dimensional Points

Now that you have completed the one-dimensional case, let us have a look at the 2D case. This means that all our points will have an x and y coordinate. i.e. the set P is given by $P = \{(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)\}$ where $(x_i, y_i) \in \mathbb{R}^2$. This case is represented by the illustration in figure 1.

By now you should realize that C++ arrays cannot hold more than one element at a particular index. To overcome this limitation we are going to create our own **data structure**. To do this we are going to use the *struct* construct found in C++. The *struct* allows us to define a data structure to suit our needs. For example if we want to create a data structure to represent a 2-dimensional euclidean point we can achieve it in the following manner.

Listing 3: C++ struct

```
1  struct point2D{
2      double x;
3      double y;
4  };
```

In order to use the data structure that you have created, you declare it the same way you would with any variable. So if we wanted to have a point $\mathbf{q} = [1, 7]$ we would create the variable in the following manner:

Listing 4: Using your created struct

```
1  int main(void){
2      point2D q; // declaring a variable (object) q of data type point2D
3                 //
4      //Method 1 for initializing q
5      q.x = 1;  // assigning the x member/attribute of q to 1
6      q.y = 7;  // assigning the x member/attribute of q to 1
7
8      //Method 2 for initializing q
9      q = {1,7}
10
11     //Note that to access elements you just use dot notation.
12     cout<<"q=["<<q.x<<","<<q.y<<"]"<<endl;
13     return 0;
14 }
```

Now the question is: How would you create an array P of 2D euclidean points?

Info: A **data structure** is a collection of elements that represent a more complex entity. These elements are often referred to as members or attributes and can be of any data type. One way to think about it is that creating a **data structure** simply means that you're creating your own **data type**.

Using the ideas mentioned above, your task is to solve the maximum distance problem for the 2-dimensional case. I have already written the `generate2D_points` function for you, all you have to do is complete the code for `MaxDistance` and complete the `main` function

Question 2

Which of the following ways is the correct way to declare an array P of 2-dimensional points with length 10?

(a) `point2D *P = new point2D[10];`

(b) `point2D P[10];`

(c) `point2D P[10]= { } ; .`

(d)
```
point2D *P;
P = new point2D[10];
```

2.3 Case 3: 3-Dimensional Points

Now that you have dealt with the 2D case. You are fit to write a 3D solution to the maximum distance problem from scratch. So let's get coding...

3 Extend yourself: n-Dimensional Points