//Charlie Ang

//March 1, 2017

//CSC 3430

//Homework 5: Implement Closest Point Brute-Force and Divide & Conquer Algorithms

//This program uses the divide and conquer technique in C/C++ to find the smallest distance from a

//given set of points.

//Code borrowed from: http://www.geeksforgeeks.org/closest-pair-of-points/

#include <stdio.h>

#include <float.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#include <iomanip>

#include <iostream>

using namespace std;

// A structure to represent a Point in 2D plane

struct Point

{

int x, y;

};

//Global variables

int distanceCalculations = 0; //num of calculations for brute force

int bruteForcei = 0; //keeping track of point i for two closest points

int bruteForcej = 0; //keeping track of point j for two closest points

int divideConqueri = 0; //keeping track of point i for two closest points

int divideConquerj = 0; //keeping track of point j for two closest points

int divideConqueriStrip = 0;

int divideConquerjStrip = 0; //if closest points are not from closest, then

//closest points are from brute force

float dlfinal = FLT\_MAX;

int finaldli = 0;

int finaldlj = 0;

float drfinal = FLT\_MAX;

int finaldri = 0;

int finaldrj = 0;

float distanceFinal = FLT\_MAX;

int finali = 0;

int finalj = 0;

/\* Following two functions are needed for library function qsort().

Refer: http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ \*/

// Needed to sort array of points according to X coordinate

int compareX(const void\* a, const void\* b)

{

Point \*p1 = (Point \*)a, \*p2 = (Point \*)b;

return (p1->x - p2->x);

}

// Needed to sort array of points according to Y coordinate

int compareY(const void\* a, const void\* b)

{

Point \*p1 = (Point \*)a, \*p2 = (Point \*)b;

return (p1->y - p2->y);

}

//Finds index in P where set of point is

int index(Point P[], Point find, int numberOfPoints)

{

for (int i = 0; i < numberOfPoints; i++)

{

if (P[i].x == find.x && P[i].y == find.y)

return i; //find index where this point lies

}

}

//Function to print all the points

void PrintPoints(Point P[], int n)

{

int count = 0;

//Printing structs of Points

for (int i = 0; i < n - 1; i++)

{

cout << "[" << setw(4) << P[i].x << "," << setw(4) << P[i].y << "], ";

count++;

if (count == 6) //printing six points per line

{

cout << endl;

count = 0;

}

}

cout << "[" << setw(4) << P[n - 1].x << "," << setw(4) << P[n - 1].y << "]" << endl;

}

// A utility function to find the distance between two points

float dist(Point p1, Point p2)

{

// |p1p2| = sqrt((p1x - p2x)^2 + (pyx - p2y)^2)) distance formula

return sqrt((float(p1.x) - float(p2.x))\*(float(p1.x) - float(p2.x)) +

(float(p1.y) - float(p2.y))\*(float(p1.y) - float(p2.y))

);

}

// A Brute Force method to return the smallest distance between two points

// in P[] of size n

float bruteForce(Point P[], int n)

{

float min = FLT\_MAX;

for (int i = 0; i < n; ++i)

for (int j = i + 1; j < n; ++j)

{

distanceCalculations++; //increment brute force comparisons

if (dist(P[i], P[j]) < min)

{

min = dist(P[i], P[j]);

bruteForcei = i; //saves index of smallest point in orig P

bruteForcej = j; //saves index of smallest second smallest point in orig P

}

}

return min;

}

// A utility function to find minimum of two float values

float min(float x, float y)

{

return (x < y) ? x : y;

}

// A utility function to find the distance beween the closest points of

// strip of given size. All points in strip[] are sorted accordint to

// y coordinate. They all have an upper bound on minimum distance as d.

// Note that this method seems to be a O(n^2) method, but it's a O(n)

// method as the inner loop runs at most 6 times

float stripClosest(Point strip[], int size, float d)

{

float min = d; // Initialize the minimum distance as d

qsort(strip, size, sizeof(Point), compareY);

// Pick all points one by one and try the next points till the difference

// between y coordinates is smaller than d.

// This is a proven fact that this loop runs at most 6 times

for (int i = 0; i < size; ++i)

for (int j = i + 1; j < size && (strip[j].y - strip[i].y) < min; ++j)

{

distanceCalculations++;

if (dist(strip[i], strip[j]) < min)

{

min = dist(strip[i], strip[j]);

divideConqueriStrip = i;

divideConquerjStrip = j;

}

}

return min;

}

// A recursive function to find the smallest distance. The array P contains

// all points sorted according to x coordinate

float closestUtil(Point P[], int n)

{

// If there are 2 or 3 points, then use brute force

if (n <= 3)

return bruteForce(P, n);

// Find the middle point

int mid = n / 2;

Point midPoint = P[mid];

// Consider the vertical line passing through the middle point

// calculate the smallest distance dl on left of middle point and

// dr on right side

float dl = closestUtil(P, mid);

//save brute force values from left side

if (dl <= dlfinal)

{

dlfinal = dl;

finaldli = bruteForcei;

finaldlj = bruteForcej;

}

float dr = closestUtil(P + mid, n - mid);

//save brute force values from right side

if (dr <= drfinal)

{

drfinal = dr;

finaldri = bruteForcei;

finaldrj = bruteForcej;

}

// Find the smaller of two distances

float d = min(dl, dr);

if (d == dl) //smallest distance from left side

{

bruteForcei = finaldli;

bruteForcej = finaldlj;

}

else //smallest distance from right side

{

bruteForcei = finaldri + mid; //find actual index in P by adding mid

bruteForcej = finaldrj + mid;

}

if (d < distanceFinal)

{

distanceFinal = d;

finali = bruteForcei;

finalj = bruteForcej;

}

// Build an array strip[] that contains points close (closer than d)

// to the line passing through the middle point

Point \*strip = new Point[n];

int j = 0;

for (int i = 0; i < n; i++)

{

if (abs(P[i].x - midPoint.x) < d)

{

strip[j] = P[i], j++;

}

}

float smallestDistance = min(d, stripClosest(strip, j, d));

if (smallestDistance == d) //closest points were from brute force

{

divideConqueri = bruteForcei;

divideConquerj = bruteForcej;

}

else //closest points were from strip

{

//taking the index from the strip[] find occurance of this point in P sorted

divideConqueri = index(P, strip[divideConqueriStrip], n);

divideConquerj = index(P, strip[divideConquerjStrip], n);

}

if (smallestDistance <= distanceFinal)

{

distanceFinal = smallestDistance;

finali = divideConqueri;

finalj = divideConquerj;

}

delete[] strip;

// Find the closest points in strip. Return the minimum of d and closest

// distance is strip[]

return smallestDistance;

}

// The main functin that finds the smallest distance

// This method mainly uses closestUtil()

float closest(Point P[], int n)

{

qsort(P, n, sizeof(Point), compareX);

// Use recursive function closestUtil() to find the smallest distance

return closestUtil(P, n);

}

//Initializing large array with random unique points

void initpoints(Point P[], int n, int range, int seed = -1)

{

if (seed < 0)

seed = clock();

srand(seed);

//Fill array with unique points[i] for i from 0..n-1

int i = 0, j;

while (i < n)

{

//create random point within range

Point p;

p.x = rand() % range;

p.y = rand() % range;

//search to see if new point is unique

for (j = 0; j < i; ++j)

{

if (P[j].x == p.x && P[j].y == p.y)

{

break; //already existing point

}

}

if (j >= i)

{

//P[i].x = p.x; //Unique point -- save it

//P[i].y = p.y;

P[i] = p;

i++; //move on to next point

}

}

}

// Driver program to test above functions

int main()

{

Point P[] = { { 2, 3 }, { 12, 30 }, { 40, 50 }, { 5, 1 }, { 12, 10 }, { 3, 4 } };

//Point P[] = { { 1, 1 }, { 2, 5 }, { 15, 1 }, { 16, 1 }, { 36, 4 }, { 39, 1 } };

//Point P[] = { { 1, 1 }, { 2, 5 }, { 15, 1 }, { 23, 1 }, { 36, 4 }, { 39, 1 } };

//Point P[] = { { 1, 1 }, { 2,24 }, { 15, 1 }, { 23, 1 }, { 36, 4 }, { 39, 1 } };

//Point P[1000]; //for random array

//initpoints(P, 1000, 10000, 0); //3rd param 10,000 for P[1000]

int n = sizeof(P) / sizeof(P[0]);

Point temp[6]; //temp to copy over orginal array PLUG IN SIZE HERE

copy(begin(P), end(P), begin(temp)); //copy original array into temp

cout << "--------------Closest Points in a plane analysis: Charlie Ang--------------" << endl;

printf("Points[%d] -- Charlie Ang Analysis\n", n);

//Printing Points for P

PrintPoints(P, n);

cout << "Brute Force" << endl;

printf("The smallest distance is %f ", bruteForce(P, n)); //brute force algorithm

int bruteForceiOrig = index(temp, P[bruteForcei], n); //original index i where point is

int bruteForcejOrig = index(temp, P[bruteForcej], n); //original index j where point is

printf("--> P%d[%4d,%4d] : P%d[%4d,%4d]\n", bruteForceiOrig, P[bruteForcei].x, P[bruteForcei].y, bruteForcejOrig, P[bruteForcej].x, P[bruteForcej].y);

cout << "-->\t" << distanceCalculations << " point-to-point distance calculations" << endl;

distanceCalculations = 0; //re-initialize at end of print function

bruteForcei = 0;

bruteForcej = 0;

cout << "Divide & Conquer" << endl;

printf("The smallest distance is %f ", closest(P, n)); //divide and conquer algorithm

int divideConqueriOrig = index(temp, P[finali], n);

int divideConquerjOrig = index(temp, P[finalj], n);

printf("--> P%d[%4d,%4d] : P%d[%4d,%4d]\n", divideConqueriOrig, P[finali].x, P[finali].y, divideConquerjOrig, P[finalj].x, P[finalj].y);

cout << "-->\t" << distanceCalculations << " point-to-point distance calculations" << endl;

cout << "-----------------------------------------------------------------Charlie Ang" << endl;

return 0;

}