**CSC 323 Project 4 (C++)**

Tyler Gaugler

Due Date: 11/01/2016

step 0:  
- Open the input file  
- K, numRow, numCol numPts  get from input file.  
- imageArray  Dynamically allocate a 2-D arrays, size numRows X numCols.  
  
        - pointSet  Dynamically allocate the point set, size of numPts  
        - Kcentroids[K]  Dynamically allocate the K centroids struct.  
  
Step 1: call loadPointSet  
Step 2: call assignLabel  
Step 3: call mapPoint2Image  
Step 4: call displayImage // output to output-2  
  
step 5: Go thru the entire pointSet struct array to compute the centroids of the K  clusters. Store the centroids in each Kcentroids[i], i from 1 to K.  
(\*\* You should be able to compute the K centroids just go thru the array only once.)  
  
step 6: 6.1: for each point, p, in the pointSet array  
                compute the distance, dist(p,ci), from  
                        p to the centroids of each Kcentroids[i], i = 1 to K  
           6.2: min\_i <-- determine which dist(p,ci) is minimum  
  
           6.3: if min\_i is not the same as p's old label change p's label to min\_i  
                     And changeLabel set to true  
  
step 7: repeat step 6 until all points in pointSet are process.  
Step 8: repeat step 3 to step 7 until no point changes its label.  
  
Step 9: Output the info of pointSet to Output-1 file.

**Source Code:**

// main.cpp

#include <iostream>

#include <fstream>

#include <string>

#include <stdlib.h>

#include <math.h>

using namespace std;

ifstream inputFile;

ofstream outputFile1;

ofstream outputFile2;

class Point{

public:

int xCoordinate;

int yCoordinate;

int clusterID;

double distance;

void printPoint(void);

Point(int x, int y);

Point(void);

};

Point::Point(int x, int y){

xCoordinate=x;

yCoordinate=y;

}

Point::Point(){

}

// void Point::printPoint(){

// }

class K\_mean{

struct XYCordinate{

int xCord;

int yCord;

};

public:

int K;

XYCordinate\* kCentroids;

int numPoints;

Point\* pointSet;

int pointSetIndex;

int numRow;

int numCol;

int\*\* imageArray;

void loadPointSet(Point\* p);

void assignLabel(Point\* p);

void mapPoint2Image();

void kMeanClustering();

void printPointSet();

void displayImage();

void printToOutput1();

void printToOutput2();

void findCentroids();

int findLength(Point p, int cluster\_id);

K\_mean(int k, int np, int nr, int nc);

};

K\_mean::K\_mean(int k, int np, int nr, int nc){

K=k;

numPoints=np;

numRow=nr;

numCol=nc;

pointSet= new Point[numPoints];

pointSetIndex=0;

imageArray= new int\*[nr];

for(int i = 0; i < nr; ++i){

imageArray[i] = new int[nc];

}

kCentroids= new XYCordinate[K];

}

void K\_mean::loadPointSet(Point\* p){

assignLabel(p);

pointSet[pointSetIndex]= \*p;

pointSetIndex++;

}

void K\_mean::assignLabel(Point\* p){

p->clusterID= (pointSetIndex % K) +1;

}

void K\_mean::mapPoint2Image(){

for(int i=0;i<pointSetIndex;i++){

imageArray[pointSet[i].xCoordinate][pointSet[i].yCoordinate]= pointSet[i].clusterID;

}

}

void K\_mean::displayImage(){

for(int row=0;row<numRow;row++){

for(int col=0;col<numCol;col++){

if(imageArray[row][col]) outputFile2<< imageArray[row][col]<<" ";

else outputFile2<<" ";

}

outputFile2<<endl;

}

}

void K\_mean::printToOutput1(){

outputFile1<<K<<endl<<numPoints<<endl<<"\t"<<numRow<<" "<<numCol<<endl;

for(int i=0;i<pointSetIndex;i++){

outputFile1<<"\t"<<pointSet[i].xCoordinate<<" "<<pointSet[i].yCoordinate<<" "<<pointSet[i].clusterID<<endl;

}

}

void K\_mean::printToOutput2(){

for(int row=0;row<numRow;row++){

for(int col=0;col<numCol;col++){

if(imageArray[row][col]) outputFile2<< imageArray[row][col]<<" ";

else outputFile2<<" 1";

}

outputFile2<<endl;

}

}

void K\_mean::findCentroids(){

for(int i=0;i<pointSetIndex;i++){

kCentroids[pointSet[i].clusterID].xCord+= (pointSet[i].xCoordinate)/K;

kCentroids[pointSet[i].clusterID].yCord+= (pointSet[i].yCoordinate)/K;

}

bool ChangesOccur=false;

while(!ChangesOccur){

ChangesOccur=false;

for(int i=0;i<pointSetIndex;i++){

int min\_i;

for(int c=1;c<=K;c++){

if(findLength(pointSet[i], c) < findLength(pointSet[i], pointSet[i].clusterID)){

pointSet[i].clusterID= c;

ChangesOccur=true;

}

}

}

printToOutput1();

displayImage();

}

}

int K\_mean::findLength(Point p, int cluster\_id){

int centroidx= kCentroids[cluster\_id].xCord;

int centroidy= kCentroids[cluster\_id].yCord;

return sqrt(((p.xCoordinate - centroidx)^2)- ((p.yCoordinate - centroidy)^2));

}

int main(int argc, char\* argv[]){

outputFile1.open(argv[2]);

outputFile2.open(argv[3]);

K\_mean\* kmean;

int line=1;

int k;

int numRow;

int numCol;

int numPoints;

inputFile.open(argv[1]);

if(inputFile.is\_open() && line <= 4){

string data;

while(inputFile >> data){

// cout<<data<<endl;

if(line == 1) k= std::stoi(data);

else if(line == 2) numPoints = std::stoi(data);

else if(line == 3) numRow= std::stoi(data);

else if(line == 4) {

numCol= std::stoi(data);

kmean= new K\_mean(k,numPoints,numRow,numCol);

}

else {

int x= std::stoi(data);

inputFile >> data;

int y= std::stoi(data);

Point\* newPoint= new Point(x, y);

kmean->loadPointSet(newPoint);

}

line++;

}

kmean->printToOutput1();

kmean->mapPoint2Image();

kmean->displayImage();

kmean->findCentroids();

}

outputFile2.close();

outputFile1.close();

}

**Output**

//output1.txt

4

90

80 80

18 58 1

21 52 2

19 46 3

13 46 4

16 38 1

11 36 2

11 58 3

14 42 4

10 42 1

10 41 2

10 44 3

11 54 4

9 57 1

10 52 2

21 41 3

8 46 4

12 38 1

8 35 2

10 38 3

13 49 4

8 46 1

20 55 2

21 53 3

11 48 4

11 53 1

14 57 2

9 49 3

16 39 4

10 35 1

13 51 2

39 14 3

38 9 4

29 20 1

38 11 2

27 21 3

41 24 4

29 23 1

36 10 2

37 19 3

29 10 4

35 26 1

36 20 2

35 23 3

34 18 4

37 24 1

38 56 2

37 60 3

47 54 4

45 48 1

44 60 2

39 51 3

43 52 4

38 61 1

45 63 2

44 55 3

44 47 4

41 66 1

40 55 2

49 52 3

44 53 4

59 30 1

61 36 2

65 34 3

61 38 4

66 25 1

58 39 2

66 39 3

68 30 4

61 28 1

62 25 2

62 30 3

64 20 4

59 24 1

63 28 2

61 30 3

69 24 4

69 32 1

71 30 2

70 38 3

57 39 4

58 43 1

66 28 2

64 21 3

64 39 4

63 41 1

67 44 2

63 29 3

61 21 4

65 43 1

63 24 2

4

90

80 80

18 58 1

21 52 1

19 46 1

13 46 1

16 38 1

11 36 1

11 58 1

14 42 1

10 42 1

10 41 1

10 44 1

11 54 1

9 57 1

10 52 1

21 41 1

8 46 1

12 38 1

8 35 1

10 38 1

13 49 1

8 46 1

20 55 1

21 53 1

11 48 1

11 53 1

14 57 1

9 49 1

16 39 1

10 35 1

13 51 1

39 14 3

38 9 4

29 20 1

38 11 2

27 21 3

41 24 4

29 23 1

36 10 2

37 19 3

29 10 4

35 26 1

36 20 2

35 23 3

34 18 4

37 24 1

38 56 1

37 60 1

47 54 1

45 48 1

44 60 1

39 51 1

43 52 1

38 61 1

45 63 1

44 55 1

44 47 1

41 66 1

40 55 1

49 52 1

44 53 1

59 30 1

61 36 2

65 34 3

61 38 4

66 25 1

58 39 2

66 39 3

68 30 4

61 28 1

62 25 2

62 30 3

64 20 4

59 24 1

63 28 2

61 30 3

69 24 4

69 32 1

71 30 2

70 38 3

57 39 4

58 43 1

66 28 2

64 21 3

64 39 4

63 41 1

67 44 2

63 29 3

61 21 4

65 43 1

63 24 2

//outpout2.txt

2 1

3 1

1 3 2 1 3 2

2 4 1 4 3

1

4 4 2

4 2

1 4

1

3

2

3 2 3

3

4 1 1

4

3 1

2 2

3 1 3

4 2 2 1

3 3

2

4 1

4

4 4 3 2

1 2

4

3

4

2 1

1 1

4 1 3 2 4

2 3

2 2 3 1

4 3 4

3 1

1 2 3

2

4

4 1

3

2

1 1

1 1

1 1 1 1 1 1

1 1 1 1 1

1

1 1 1

1 1

1 1

1

1

1

1 1 1

3

4 1 1

4

3 1

2 2

3 1 1

4 2 1 1

3 1

1

4 1

1

1 1 1 1

1 1

1

1

4

2 1

1 1

4 1 3 2 4

2 3

2 2 3 1

4 3 4

3 1

1 2 3

2

4

4 1

3

2

**Input**

//input.txt

4

90

80 80

18 58

21 52

19 46

13 46

16 38

11 36

11 58

14 42

10 42

10 41

10 44

11 54

9 57

10 52

21 41

8 46

12 38

8 35

10 38

13 49

8 46

20 55

21 53

11 48

11 53

14 57

9 49

16 39

10 35

13 51

39 14

38 9

29 20

38 11

27 21

41 24

29 23

36 10

37 19

29 10

35 26

36 20

35 23

34 18

37 24

38 56

37 60

47 54

45 48

44 60

39 51

43 52

38 61

45 63

44 55

44 47

41 66

40 55

49 52

44 53

59 30

61 36

65 34

61 38

66 25

58 39

66 39

68 30

61 28

62 25

62 30

64 20

59 24

63 28

61 30

69 24

69 32

71 30

70 38

57 39

58 43

66 28

64 21

64 39

63 41

67 44

63 29

61 21

65 43

63 24