Voting Tree

Nobody

2022-10-24

1. Introduction

Given two 2-dimentional shapes A and B which are represented as closed polygons, our goal is to find the optimal correspondences (or "match") between points in A and B. A simple example is illustrated below.

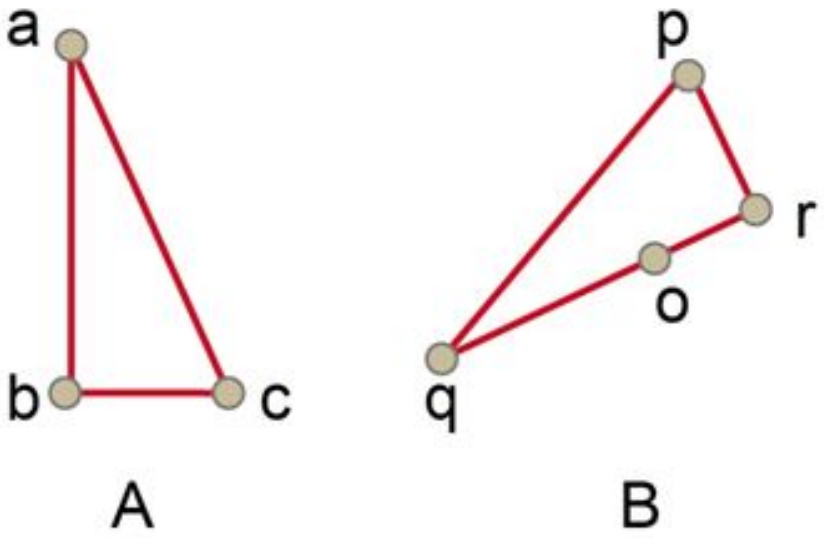

Figure 1: avatar

An optimal correspondence (match) between the two shapes are {(a,q),(b,r),(c,p)}.In this project, we are supposed to solve the aforementioned problem with a combination of various data structures including arrays and voting trees.

2. Algorithm Specification

The implementation of the program consists of two main parts, namely the construction of the tree and the output of the best matching point pairs from the voting table.

Algorithm: Tree Build

Input: Each input file contains one test case. For each case, the first line gives two positive integers M and N which are the total number of points in shape A and B respectively. The next M+N lines each contains the x and the y coordinates (in float type) of a point in shape A and B, respectively, in clockwise order. Hence for each shape, the i-th point given is indexed as i (i=1,. . . ,M or N).

Output: A tree of the all matching point pairs.

Main Idea: Building a tree using recursive methods.The function reads one point at a time from the A and B diagrams respectively, and determines whether the angle decided by the two points is approximate.If the angle difference is too large, return NULL.If the angles are similar, call this function again with the next point as the argument, respectively.If there exists an angle that is 180, it is individually deferred to the next point.In this process, if the point exhaustion occurs, return NULL.

Pseudo Code:

```
Tree TreeBuild(int point1,int point2){

    Determine if there are enough points.
    if not enough, return NULL;

    Create a new tree node, t=(point1,point2);

    if(Angle Matching){
        t->FirstChild=TreeBuild(point1+1,point2+1);
        t->NextSibling=TreeBuild(point1,point2+1);
    }
    else if(flat corner){
        t->FirstChild=NULL;
        t=TreeBuild(point1,point2+1);
    }
    else if(Angle Not Matching){
        return NULL;
    }
    return t;
}
```

Algorithm: Vote and Output

Input: the Tree

Output: Best matching point pairs.

Main Idea: Iterate through all the tree nodes, and each time a node is traversed, a vote is recorded in the corresponding voting table.Check the voting table by row and find the largest element in a row, which is the corresponding point pair.

Pseudo Code:

```
int vote(Tree t){
    when t is not NULL
    v[t->point1.number][t->point2.number]++;
    vote(t->FirstChild);
    vote(t->NextSibling);
}
int Result(int flag){
    for(int i=1;i<=min;i++){
        for(int j=1;j<=max;j++){
            Find the largest element;
            Store to the corresponding location;
            Next line;
        }
    }
    Output the Result;
}
```

3. Testing Results

Table 1 shows some typical test cases for verifying the *Voting Tree* implementation and capturing potential bugs.

All the Test Cases are included in the Appendix B.

Test Cases	Design Purpose	result	status
Case 1	Two shapes similar to example	Result 1	*pass*
Case 2	Two same shapes	Result 2	*pass*
Case 3	Two random shapes (matchable)	Result 3	*pass*
Case 4	Two random shapes (unmatchable)	Result 4	*pass*
Case 5	Two triangles	Result 5	*pass*
...	...	...	...

4. Analysis and Comments

For the runtime of *TreeBuild*, there are build a tree using recursive methods

- building the tree in the recursive method using $O(nlogn)$ time units;
- check whether it is a matchable point pairs takes a constant time c.

Therefore, the worst time complexity can be computed as:

$$T(n) = O(nlogn).$$

Since most of the trees are cut, the actual time spent will be lower.

For the space requirement, the most we need is a n-level binary tree, the space complexity is $\Theta(2^n)$ and should be optimal.

Appendix A: Source Code (in C)

```c
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

int v[101][101]={0};
int chosen[101]={0};
int r[101]={0};
//The vote table.

int left_a,left_b;
//Marks how many points are still available for matching.

int m,n;
int min,max;
//Used to mark the size of the shapes.

struct point {
    int number;
    //Serial number of the point.
    double x;
    double y;
    //Horizontal and vertical coordinates.
};
struct point a[100],b[100];

typedef struct TreeNode *Tree;
struct TreeNode {
    struct point point1;
    struct point point2;
```

```
    Tree  FirstChild;
    Tree  NextSibling;
};
//Build the tree.

Tree TreeBuild(int point1,int point2);
//The function to build the tree.

double math(int i,int j);
//The function to verify that the angles are similar.

int vote(Tree t);
//The function to vote.

int Result(int flag);
//The function to output the final result.

int Max(int i);
//The function to find the largest element in the array.

int main(){
    int flag=0;
    //Used to determine if the output needs to be reversed.
    scanf("%d %d",&m,&n);
    if(m<=n){
        min=m;
        max=n;
        for(int i=1;i<=m;i++){
            a[i].number=i;
            scanf("%lf %lf",&a[i].x,&a[i].y);
        }
        for(int i=1;i<=n;i++){
            b[i].number=i;
            scanf("%lf %lf",&b[i].x,&b[i].y);
        }
    }
    //Input the smaller shape into array a, the larger into array b.
    else{
        flag=1;
        min=n;
        max=m;
        for(int i=1;i<=m;i++){
            b[i].number=i;
            scanf("%lf %lf",&b[i].x,&b[i].y);
        }
        for(int i=1;i<=n;i++){
```

```
            a[i].number=i;
            scanf("%lf %lf",&a[i].x,&a[i].y);
        }
    }
    //Basic inputs.

    left_a=min;
    left_b=max;

    Tree T[max];
    //Used to store the subtrees.

    for(int i=1;i<=max;i++){

        Tree t;
        t=malloc(sizeof(struct TreeNode));
        T[i]=t;
        t->point1=a[1];
        t->point2=b[i];
        //Set the content of the root.

        if(math(1,i)!=1){
            T[i]=NULL;
            continue;
        }
        //If the point pairs in the root do not match, skip.

        left_a--;
        left_b--;
        t->FirstChild=TreeBuild(2,i+1);
        left_a++;
        left_b++;

        //Find the next matching point pair.
        t->NextSibling=NULL;
    }

    for(int i=1;i<=min;i++){
        vote(T[i]);
    }

    if(Result(flag)==0) printf("Unmatchable At All !");

    return 0;
}
```

```
Tree TreeBuild(int point1,int point2){

    if(left_a==0 || left_b==0 || left_b<left_a)    return NULL;

    if(point1>min)    point1-=min;
    if(point2>max)    point2-=max;
    //Ensure that the next point is found clockwise.

    Tree t;
    t=malloc(sizeof(struct TreeNode));
    t->point1=a[point1];
    t->point2=b[point2];
    //t=(point1,point2);

    if(math(point1,point2)==1){

        left_a--;
        left_b--;
        t->FirstChild=TreeBuild(point1+1,point2+1);
        left_b++;
        left_a++;
        //Find the next matching point pair.

        left_b--;
        t->NextSibling=TreeBuild(point1,point2+1);
        left_b++;
        //Find the next suitable subroot.
    }
    //if the angle and the line is suitable, continue.
    else if(math(point1,point2)==0.5){
        t->FirstChild=NULL;

        left_b--;
        t=TreeBuild(point1,point2+1);
        left_b++;
        //Find a point on the extension line.
    }
    //if there is a point on the line, skip the point.
    else{
        return NULL;
    }
    //if the angle and the line is not suitable, and the it is not a direct line, return NU

    return t;
}
```

```
double math(int i,int j){

    double dx1=a[i].x-a[i%min+1].x;
    double dx2=a[i].x-a[(i-2+min)%min+1].x;
    double dx3=b[j].x-b[j%max+1].x;
    double dx4=b[j].x-b[(j-2+max)%max+1].x;

    double dy1=a[i].y-a[i%min+1].y;
    double dy2=a[i].y-a[(i-2+min)%min+1].y;
    double dy3=b[j].y-b[j%max+1].y;
    double dy4=b[j].y-b[(j-2+max)%max+1].y;

    double line1=sqrt(dx1*dx1+dy1*dy1);
    double line2=sqrt(dx2*dx2+dy2*dy2);
    double line3=sqrt(dx3*dx3+dy3*dy3);
    double line4=sqrt(dx4*dx4+dy4*dy4);

    double cos1=(dx1*dx2+dy1*dy2)/(line1*line2);
    double cos2=(dx3*dx4+dy3*dy4)/(line3*line4);

    //The above are the relevant mathematical operations.

    if(cos2<-0.95) return 0.5;
    //Determine if it is a three-point line.

    if(cos1/cos2>1.1 || cos1/cos2<0.9)   return 0;
    //Judging whether the angle is close or not.

    return 1;
}

int vote(Tree t){
    if(t!=NULL){
        v[t->point1.number][t->point2.number]++;
        //Import the matches for each point into the vote table.
        vote(t->FirstChild);
        vote(t->NextSibling);
    }
}

int Result(int flag){
    int level=max-min;

    for(int i=1;i<=max;i++){
        chosen[i]=0;
        r[i]=0;
```

```
    }
    //Initialization

    for(int i=1;i<=min;i++){
        int k=0;
        //Mark the largest element's position.
        for(int j=1;j<=max;j++){
            if(v[i][j]>v[i][k] && chosen[j]==0 && v[i][j]>0){
                k=j;
                chosen[j]=1;
            }
        }
        if(flag==1) r[k]=i;
        else r[i]=k;
    }
    //Transforming content of vote tree into solutions.

    for(int i=1;i<=m;i++){
        if(r[i]==0){
            level--;
            if(level<0) return 0;
        }
    }
    //Check if there is a unique solution.

    for(int i=1;i<=m;i++){
        if(r[i]>0){
            printf("(%d,%d)\n",i,r[i]);
        }
    }
    //Output the solution.
    return 1;
}
//Find the vote table and output the most matching set of point pairs.
```

Appendix B: Test Case

Case1

4 8
3 0
0 4
3 8
9 0
12 0
10.5 2

9 4
10.5 6
12 8
15 4
18 0
15 0

Result1
(1,1)
(2,3)
(3,5)
(4,7)

Case2
4 4 0 0 1 0 1 1 0 1
0 0 1 0 1 1 0 1

Result2
(1,1)
(2,2)
(3,3)
(4,4)

Case3
4 3
6 0
2.99 4
3.01 4
6 4
3 0
0 0
0 4

Result3
(1,3)
(2,1)
(4,2)

Case4
3 4
0 0
1 0
0 1
2 0
3 0
3 1
2 1

Result4

Unmatchable At All !

Case5
3 3 0 1 1 0 0 0 0 1.5 1.5 0 0 0

Result5
(1,1) (2,2) (3,3)

Declaration

I hereby declare that all the work done in this project is of my independent effort.