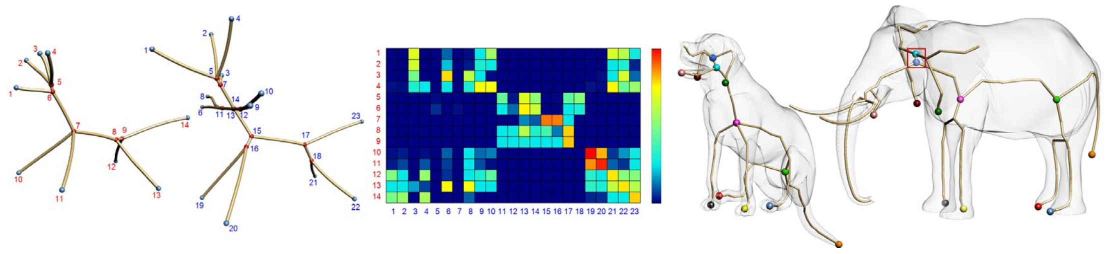
Project 1. Voting Tree

Haoyi Duan

October 26, 2020

Chapter 1: Introduction

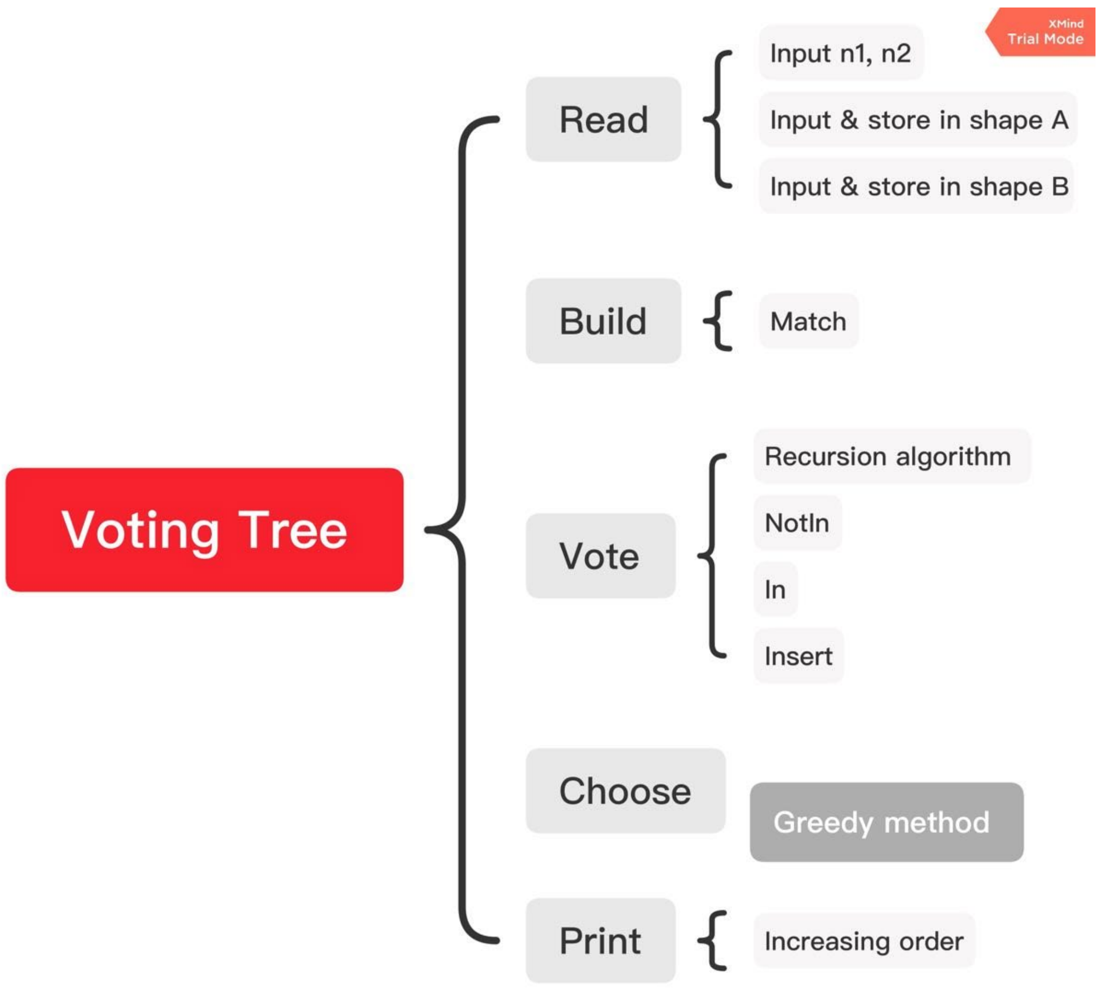


The algorithm in the project "voting tree" challenges the difficult problems of automatic semantic correspondence between two given shapes which are semantically similar but possibly geometrically very different. We argue that the challenging part is the establishment of a sparse correspondence and show that it can be efficiently solved by considering the underlying skeletons augmented with intrinsic surface information. To avoid potentially costly direct search for the best combinatorial match between two sets of skeletal feature nodes, we introduce a statistical correspondence algorithm based on a novel voting scheme, which we call electors voting. The electors are a rather large set of correspondences which then vote to synthesize the final correspondence. The electors are selected via a combinatorial search with pruning tests designed to quickly filter out a vast majority of bad correspondence. This voting scheme is both efficient and insensitive to parameter and threshold settings. The effectiveness of the method is validated by precision-recall statistics with respect to manually defined ground truth. We show that high quality correspondences can be instantaneously established for a wide variety of model pairs, which may have different poses, surface details, and only partial semantic correspondence.

Based on the background above, this project is to write a program to match two groups of two-dimensional points whose coordinates are given in advance. For Example, if A is similar to B, we should find the optimal match between points in A and B. For each test case, print the correspondence points in the best match in the format "(i1,i2)", where i1 is the index of a point in shape A, and i2 in shape B. Each pair in a line, given in increasing order of shape A indices.

Chapter 2: Algorithm Specification

To implement simple shape correspondence, steps are required. Firstly, read the data and store them in appropriate places. Second, Build the voting tree by using recursion algorithm. Third, vote and store the results in an integer array. Fourth, choose the perfect match by using greedy method. Finally, print the result in increasing order. The mind map below may give you a more clear perception.



2.1 Read(void)

First, the program read the input data by function Read(void) and store the data with the use of the data structure struct node. The structure of the struct node is as follows:

```
typedef struct node Node;
struct node
{
    double x, y;
}shapeA[MaxSize], shapeB[MaxSize];
```

Where shapeA[MaxSize] & ShapeB[MaxSize] are used to store the coordinates of points of each shape in increasing order, respectively.

```
typedef struct node Node;
struct node
{
    double x, y;
}shapeA[MaxSize], shapeB[MaxSize];
```

The pseudo-code of the Read(void) below may give you a rough sketch:

```
Read:
input(n1, n2)
for i in range(n1)
  input(xi, yi)
  shapeA[i] = (xi, yi)
for i in range(n2)
  input(xi, yi)
  shapeB[i] = (xi, yi)
```

2.2 Build(SearchTreeADT p, int left, int right)

This function is vital in this project. It uses recursion algorithm to build the voting tree. The input integers left and right represents the index of shape A and Shape B at present. We need to judge:

a. whether the level of the node is less than 3.

b. If the level of the node is equal or more than 3, use Match function to judge whether to terminate.

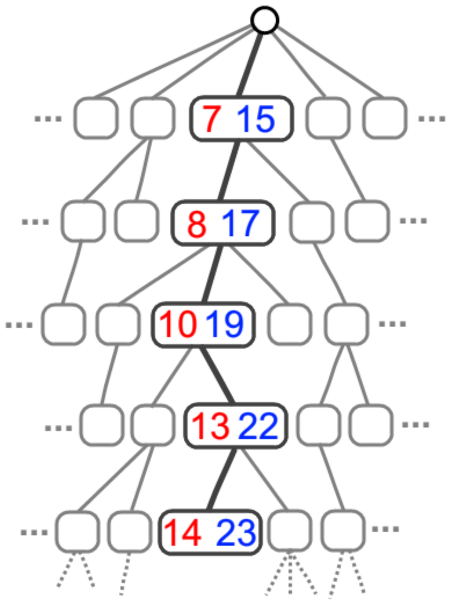
After the work in one step has done, the function will jump to a new Build function. For this process, you can consider the child node as a new root of the subset and the function will do exactly the same work in this new circumstance.

Moreover, the data structure SearchTreeADT is first used in this function. In this structure, p1 & p2 are respectively the index of two points of shape A and shape B, which is a possible match. ChildNum represents the number of child nodes, and Level indicates the level of the present node. SearchTreeADT Last points to the father node, and Next[MaxSize] point to the child nodes, an array is used here because the number of the child nodes may probably more than 1.

```
typedef struct SearchTreeCDT* SearchTreeADT;
struct SearchTreeCDT
{
    int p1, p2;
    int ChildNum;
    int Level;
    SearchTreeADT Last;
    SearchTreeADT Next[MaxSize];
};
```

The pseudo-code of the Build(SearchTreeADT p, int left, int right) is as follows:

```
for i in range(n1):
for i in range(n2):
{
  if level<3 || (level>=3 && Match):
    temp = node
    p->next = temp
  Build(p->next)
}
```



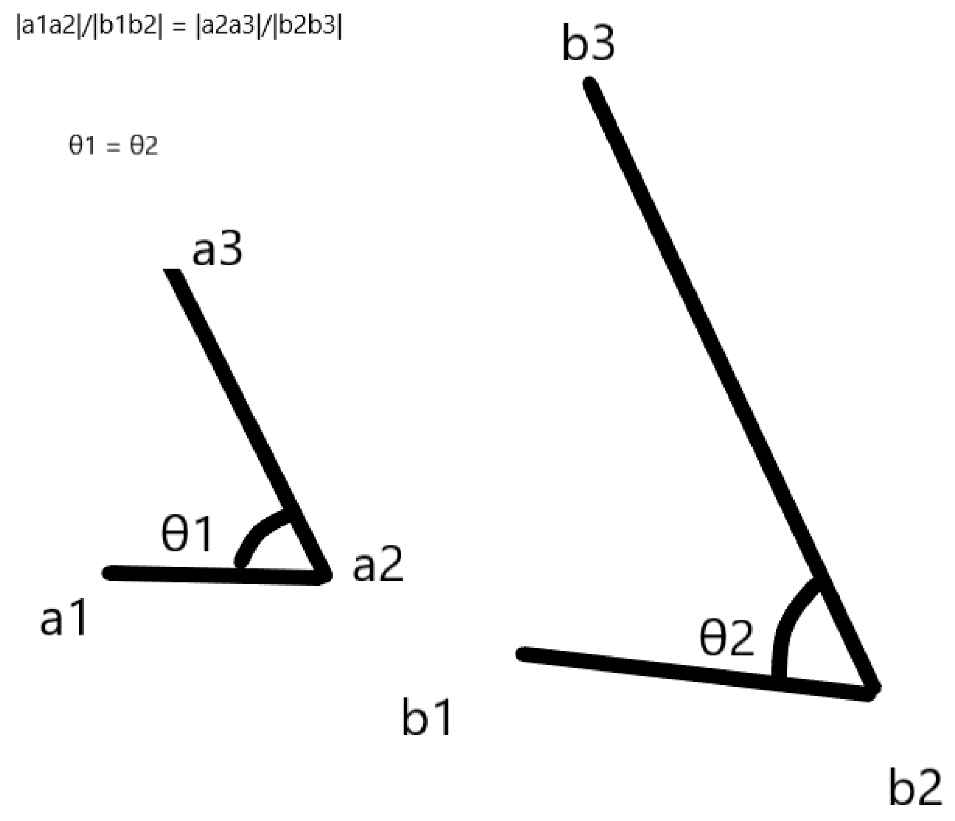
2.3 Match(int p1, int n1, int p2, int n2, int p3, int n3)

The match algorithm is 'ASA Similar Triangles'. It means to judge whether the three nodes (present node, p, p->Last) can be matched, there are two aspects which should take into account.

The first rule is that the ratio of the corresponding side length is close enough;

The second rule is that the angles formed by both sides are close enough.

See relevant picture below.



```
Match

{

aL2 = sqrt((ax1-ax2)*(ax1-ax2) + (ay1-ay2)*(ay1-ay2));

aL2 = sqrt((ax3-ax2)*(ax3-ax2) + (ay3-ay2)*(ay3-ay2));

bL1 = sqrt((bx1-bx2)*(bx1-bx2) + (by1-by2)*(by1-by2));

bL2 = sqrt((bx3-bx2)*(bx3-bx2) + (by3-by2)*(by3-by2));

if (MyAbs(aL1/bL1-aL2/bL2) > epsilon) return false;

CosTheta1 = ((ax1-ax2)*(ax3-ax2)+(ay1-ay2)*(ay3-ay2))/aL1/aL2;

CosTheta2 = ((bx1-bx2)*(bx3-bx2)+(by1-by2)*(by3-by2))/bL1/bL2;

if  (MyAbs(MyAbs(acos(CosTheta1)/acos(CosTheta2))-1)  >  epsilon)
return false;

}
```

```
#define epsilon 0.05
```

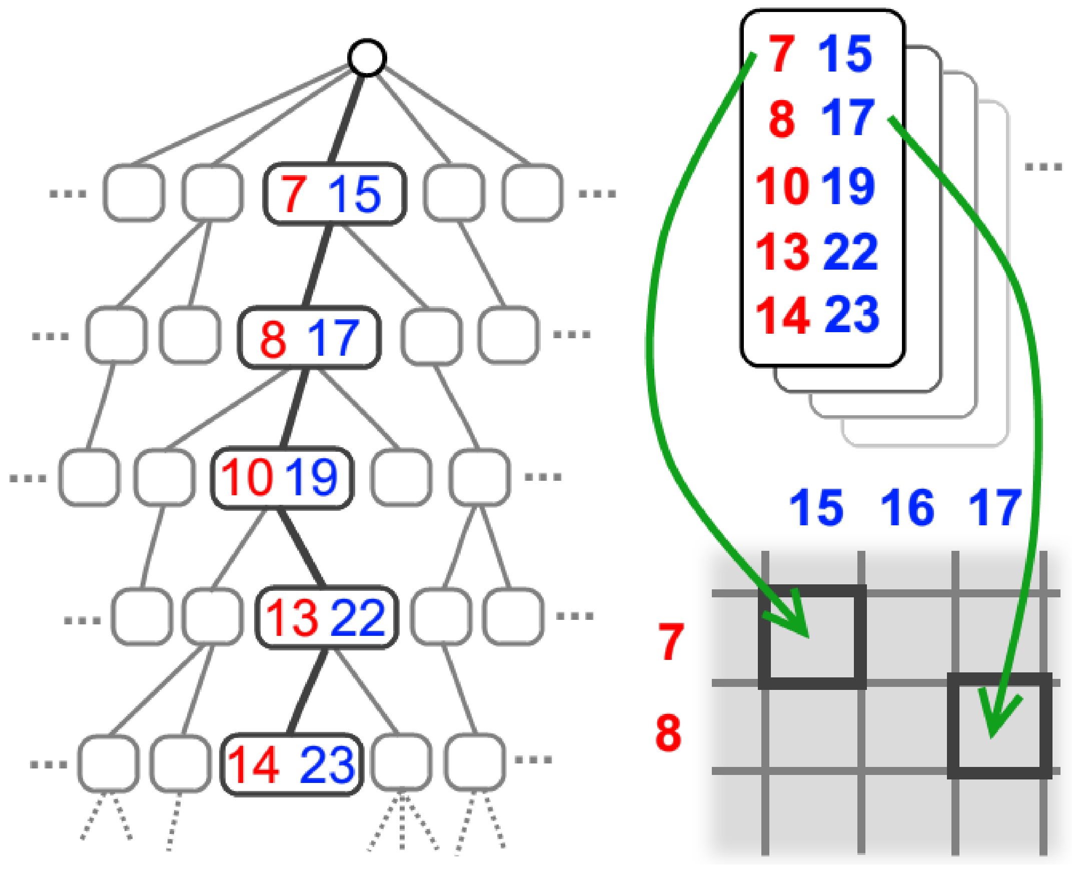
Why epsilon is 0.05? if it is too small, some correspondence will be excluded, but if it is too large, then the role of screening will be very weak. So, after testing several cases, I chose 0.05 in order to consider the both sides.

2.4 Vote(SearchTreeADT p)

After the tree is built, now it is time to vote. The point of each node is the number of path going through this node. So what is the algorithm? In fact, we just need to traverse the hole tree and when it comes to the leaf,

then its direct ancestors, for example, the father, the grandfather, the father of the grandfather, … and so on, all the numbers of path of these 'direct ancestor' nodes are supposed to plus 1, since when it comes to the leaf, it means new path has been found, so it is safe to say that all the direct ancestors have acquired a new elector. What deserves the attention is that the vote function ignore the effects of the nodes with level less than 3 As similar problems are meaningful only when the number of points is at least 3.

```
Vote
{
if p->ChildNum = 0 && p->Level > 2:
while(p){
p->(x, y)
  table[x*row+y] += 1
  p = p->Last;
}
else break
}
```



2.5 SearchTreeADT Greedy(void)

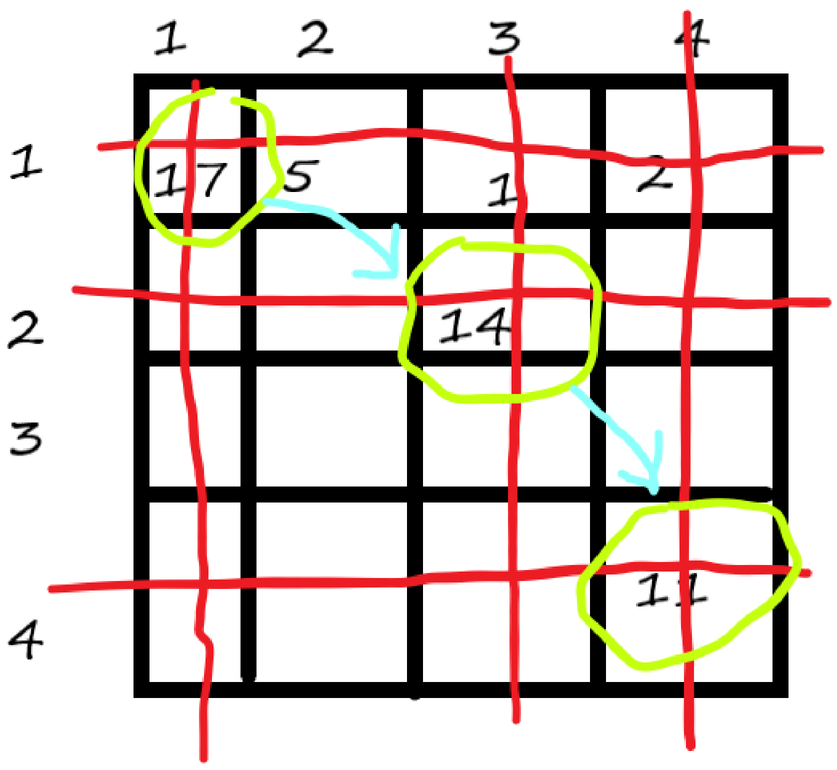
The function Greedy is used to choose the possible correspondence.

```
SearchTreeADT
{
Max=0
for i in range(n1):
  if i in board a: continue
  for j in range(n2):
     if j in board b: ccontinue
     if Max < table[i][j]: Max = table[i][j]
     x=i y=j
if Max: x insert to a, y inset to b, (x, y) insert to list p
else break
}
```

For each circulation, find the eligible nodes who has the maximum number of electors and choose it as a match. when the circulation is over, we will get one or a few results. For the 'one result' circumstance, it is the answer;

But for the 'a few results' condition, further discussion has to be taken. If

Max is equal to 0, it means there is no node available on the table, or the rest of the nodes on the table has the vote value 0. Both of the conditions won't make progress any more. So it is time to terminate. By using greedy algorithm, we can choose matches as below:



Chapter 3: Testing Results

3.1 Simple Test Result

Input

```
3 4
0 4
3 0
0 0
8 4
8 1
4.25 6
8 6
```

and the output is

```
(1, 2)
(2, 3)
```

```
(3, 4)
```
which is as expected. Validation passed.

3.2 Table of Test Cases

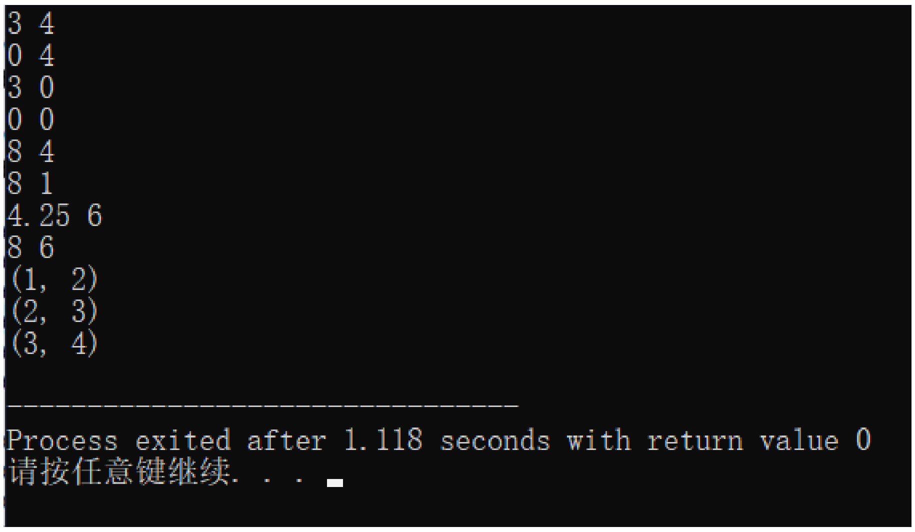
No.	input	Status	Results
1	3 4 0 4 3 0 0 0 8 4 8 1 4.25 6 8 6	pass	(1, 2) (2, 3) (3, 4)
2	3 4 0 0 3 0 3 4 0 0 2 0 3 0 3 4	pass	(1, 1) (2, 3) (3, 4)
3	4 5 0 0 1 0 1 1 0 1 2 0 4 0 4 6 4 2 2 2	pass	(1, 1) (2, 2) (3, 4) (4, 5)
4	4 4 -2 0 0 0 0 2 2 0 2 0 1 1 0 0 0 2	pass	(1, 1) (2, 2) (3, 3) (4, 4)
5	7 7 0 0	pass	(1, 1) (2, 2)

	2 0 3 0 4 0 3 1 2 2 0 2 0 0 2 0 3 0 4 0 5 0 2 2 0 2		(3, 3) (4, 4) (6, 6) (7, 7)
6	4 7 1 0 1 2 0 2 0 0 0 0 2 0 3 1 4 2 3 3 2 1 0 1	pass	(1, 1) (2, 2) (3, 6) (4, 7)
7	4 4 0 0 2 0 3 1 2 2 2 0 2.5 0 2 2 0 2	pass	(1, 1) (2, 3) (4, 4)
8	5 4 0 0 2 0 3 0.5 2 1 1 1 2 0 2 1 2 2 1 2	pass	(1, 1) (2, 2) (4, 3) (5, 4)

9	3 4 0 2 2 0 0 0 8 0 2 6 8 6 8 1	pass	(1, 1) (2, 2) (3, 3)
10	3 6 0 2 2 0 0 0 8 0 2 6 8 6 8 5 8 2 8 1	pass	(1, 1) (2, 2) (3, 3)
11	3 6 0 2 2 0 0 0 8 2 8 1 8 0 2 6 8 6 8 5	pass	(1, 3) (2, 4) (3, 5)
12	3 8 0 2 2 0 0 0 8 2 8 1 8 0 5 3 2 6 7 6 8 6 8 5	pass	(1, 3) (2, 5) (3, 7)
13	6 3 8 2 8 1 8 0	pass	(3, 1) (4, 2) (5, 3)

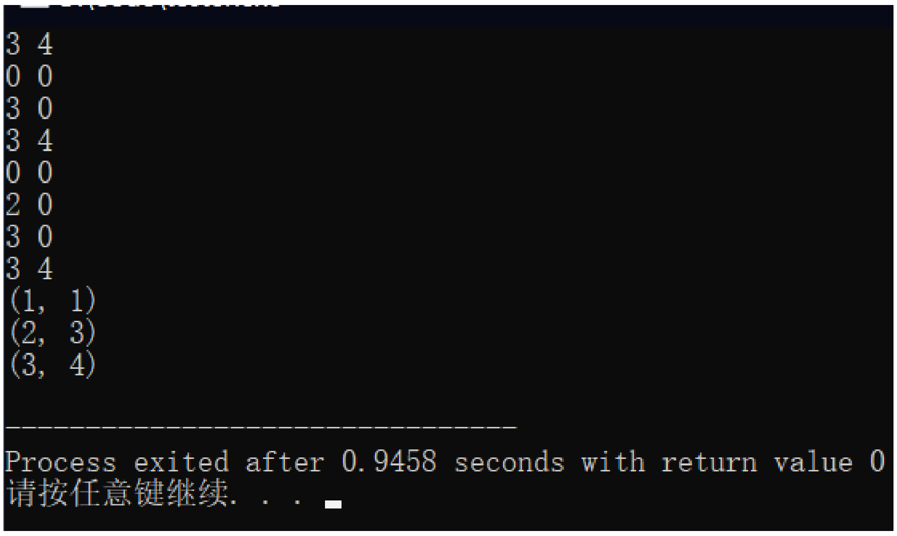
	2 6 8 6 8 5 0 2 2 0 0 0 6 3 8 2 8 1 8 0 2 6 8 6 8 5 0 2 2 0 0 0		
14	9 7 6 8 10 6 10 4 10 3 10 2 10 0 6 2 6 3 6 7 4 2 3 0 2 0 1 0 0 0 0.5 1 1 2	pass	(1, 1) (2, 2) (3, 3) (5, 4) (6, 5) (7, 7)
15	4 4 0 0 2 0 2.5 0.5 0 2 0 0 2.1 0 2.4 0.6 0.1 2	pass	(1, 1) (2, 2) (4, 4)

3.3 Result Pictures



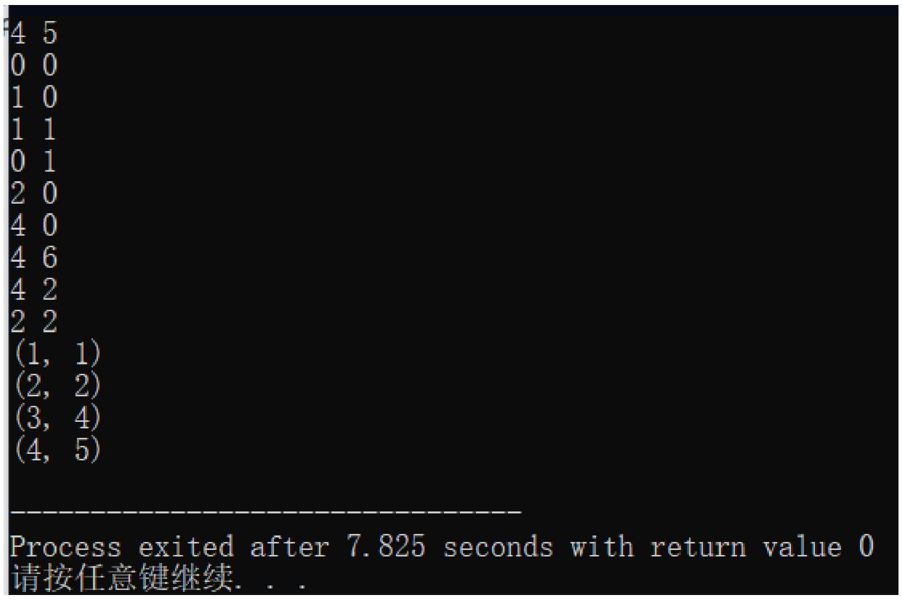
```
3 4
0 4
3 0
0 0
8 4
8 1
4.25 6
8 6
(1, 2)
(2, 3)
(3, 4)

--------------------------------
Process exited after 1.118 seconds with return value 0
请按任意键继续. . .
```



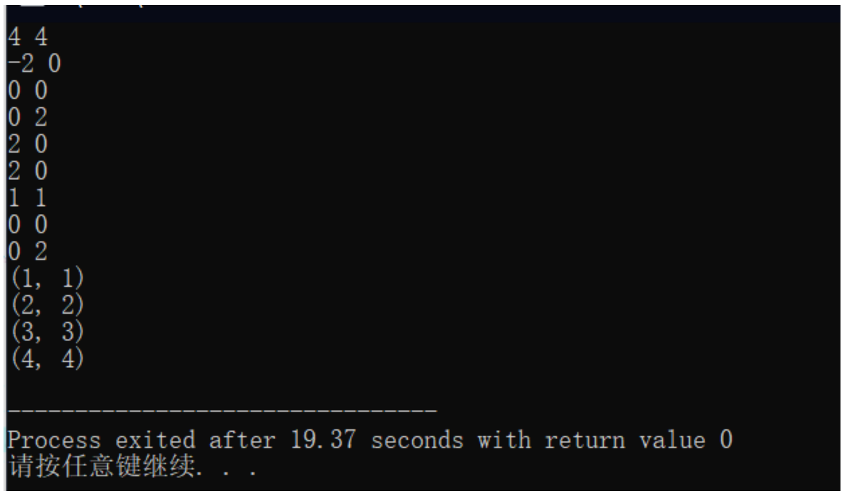
```
3 4
0 0
3 0
3 4
0 0
2 0
3 0
3 4
(1, 1)
(2, 3)
(3, 4)

--------------------------------
Process exited after 0.9458 seconds with return value 0
请按任意键继续. . .
```



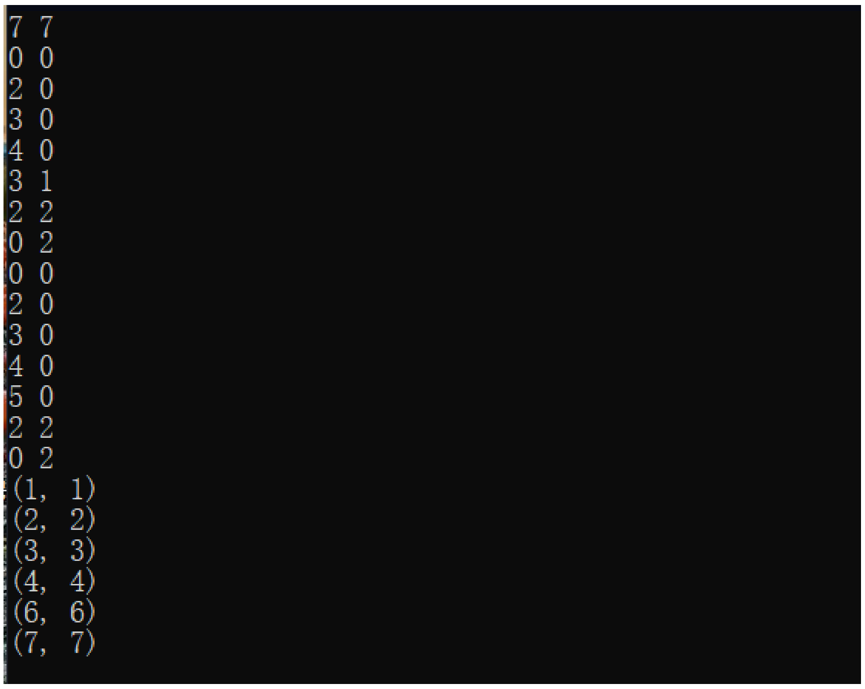
```
4 5
0 0
1 0
1 1
0 1
2 0
4 0
4 6
4 2
2 2
(1, 1)
(2, 2)
(3, 4)
(4, 5)

--------------------------------
Process exited after 7.825 seconds with return value 0
请按任意键继续. . .
```

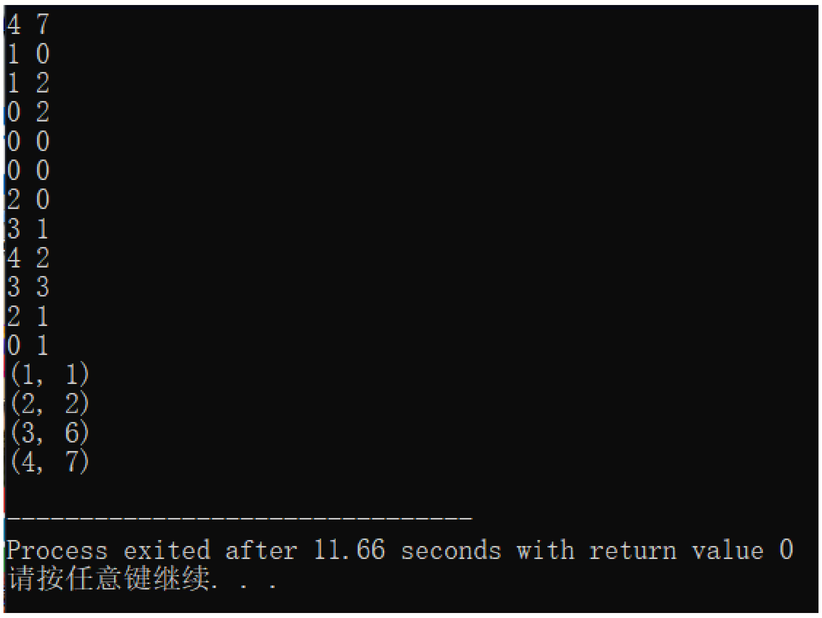


```
4 4
-2 0
0 0
0 2
2 0
2 0
1 1
0 0
0 2
(1, 1)
(2, 2)
(3, 3)
(4, 4)

--------------------------------
Process exited after 19.37 seconds with return value 0
请按任意键继续. . .
```

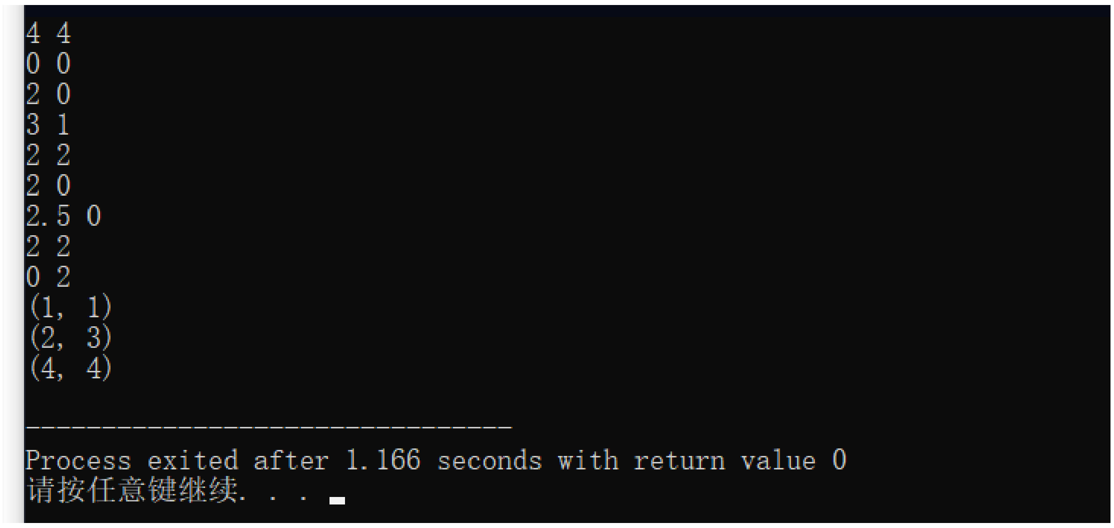


```
7 7
0 0
2 0
3 0
4 0
3 1
2 2
0 2
0 0
2 0
3 0
4 0
5 0
2 2
0 2
(1, 1)
(2, 2)
(3, 3)
(4, 4)
(6, 6)
(7, 7)
```



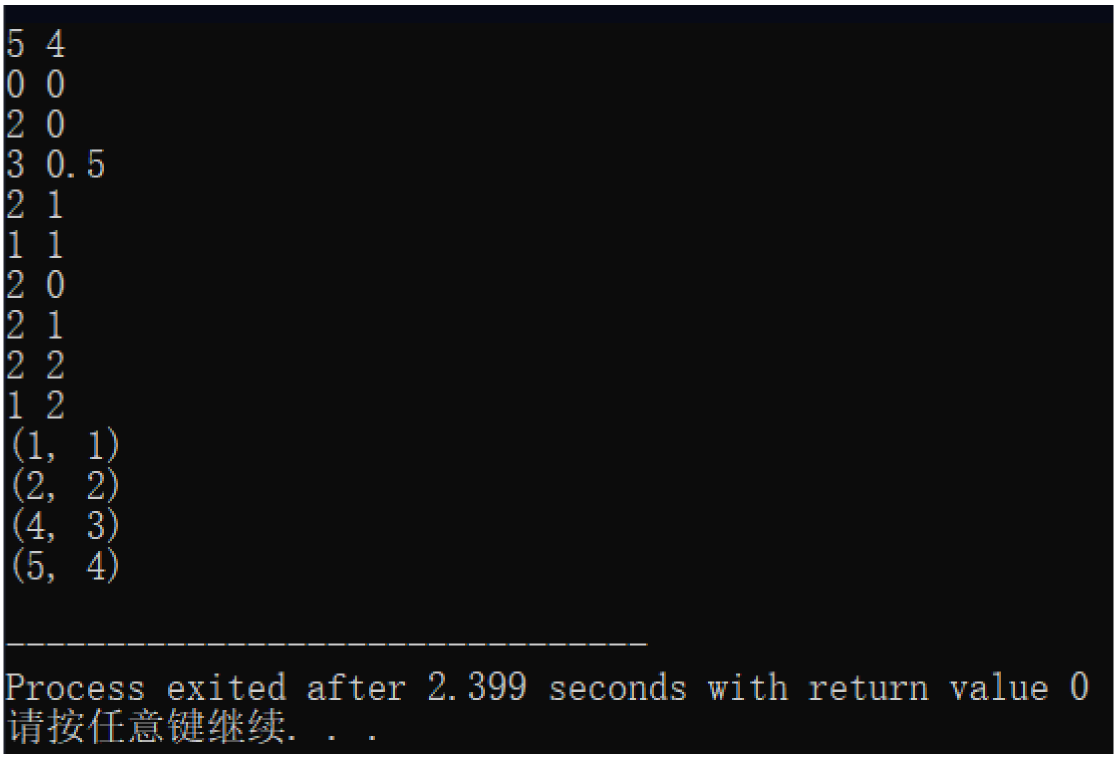
```
4 7
1 0
1 2
0 2
0 0
0 0
2 0
3 1
4 2
3 3
2 1
0 1
(1, 1)
(2, 2)
(3, 6)
(4, 7)

--------------------------------
Process exited after 11.66 seconds with return value 0
请按任意键继续. . .
```



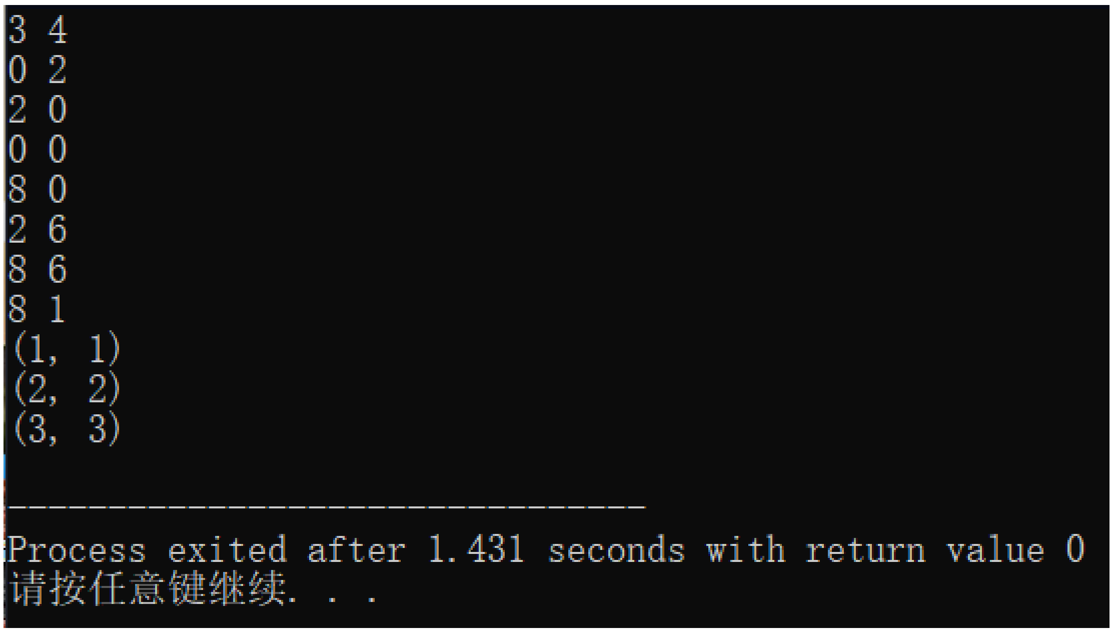
```
4 4
0 0
2 0
3 1
2 2
2 0
2.5 0
2 2
0 2
(1, 1)
(2, 3)
(4, 4)

--------------------------------
Process exited after 1.166 seconds with return value 0
请按任意键继续. . .
```



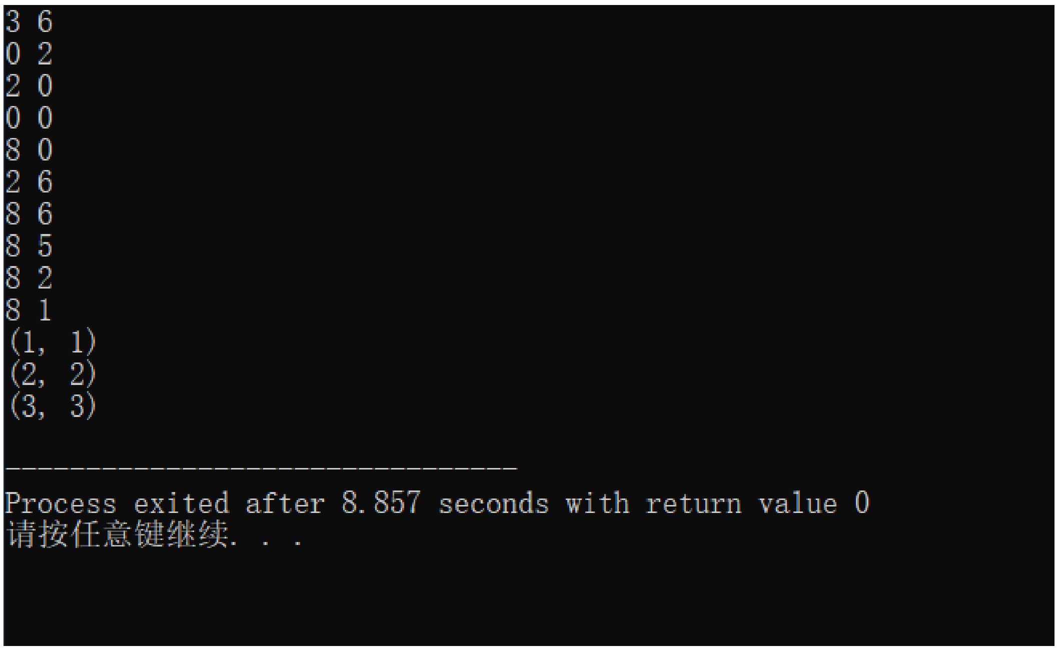
```
5 4
0 0
2 0
3 0.5
2 1
1 1
2 0
2 1
2 2
1 2
(1, 1)
(2, 2)
(4, 3)
(5, 4)

--------------------------------
Process exited after 2.399 seconds with return value 0
请按任意键继续. . .
```



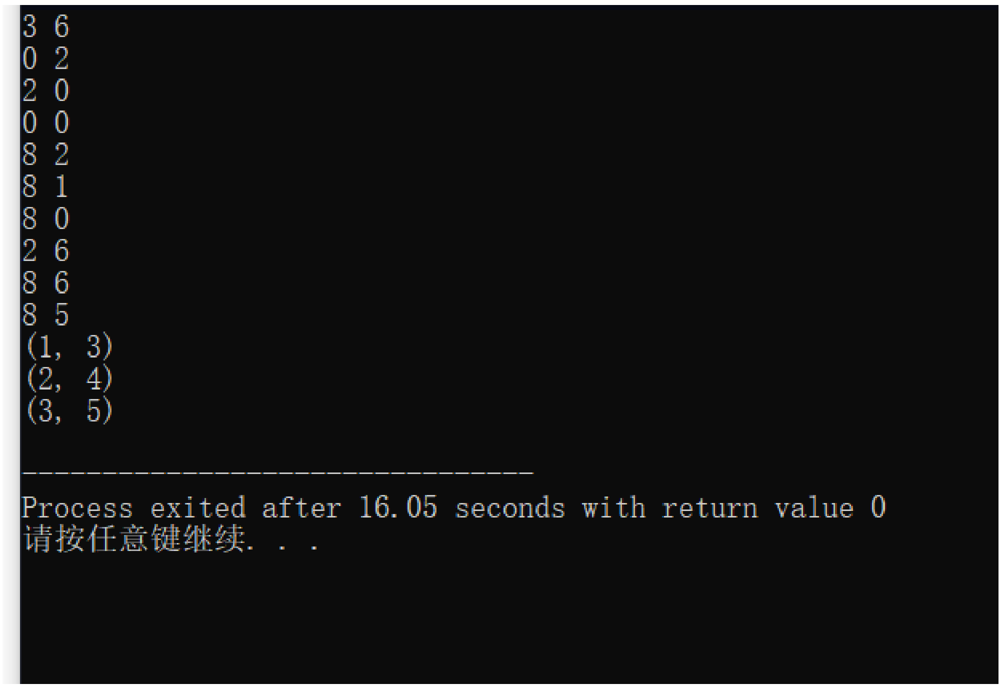
```
3 4
0 2
2 0
0 0
8 0
2 6
8 6
8 1
(1, 1)
(2, 2)
(3, 3)

--------------------------------
Process exited after 1.431 seconds with return value 0
请按任意键继续. . .
```



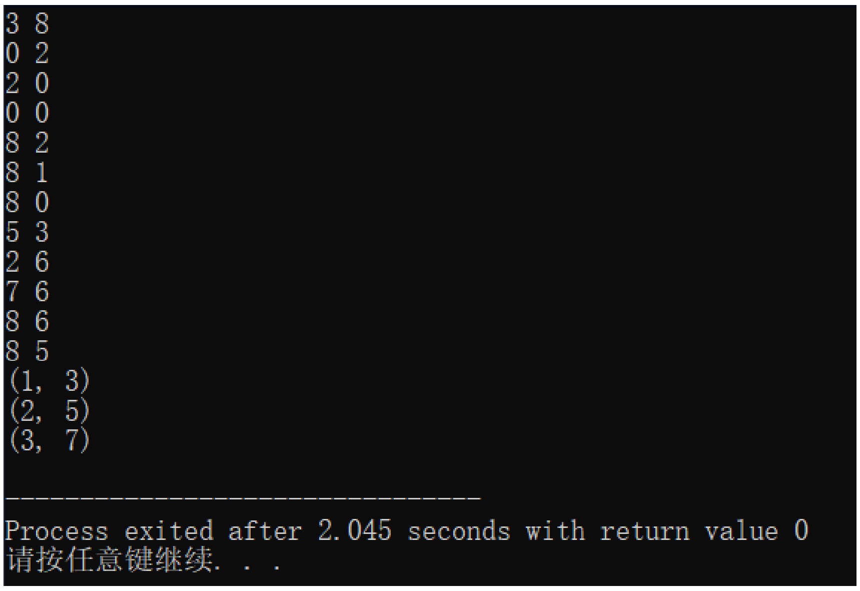
```
3 6
0 2
2 0
0 0
8 0
2 6
8 6
8 5
8 2
8 1
(1, 1)
(2, 2)
(3, 3)

--------------------------------
Process exited after 8.857 seconds with return value 0
请按任意键继续. . .
```



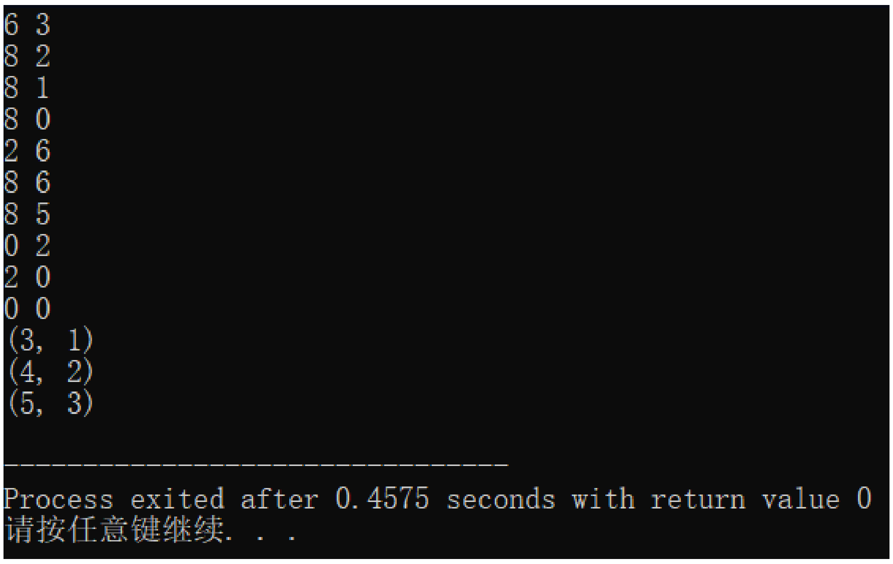
```
3 6
0 2
2 0
0 0
8 2
8 1
8 0
2 6
8 6
8 5
(1, 3)
(2, 4)
(3, 5)

--------------------------------
Process exited after 16.05 seconds with return value 0
请按任意键继续. . .
```



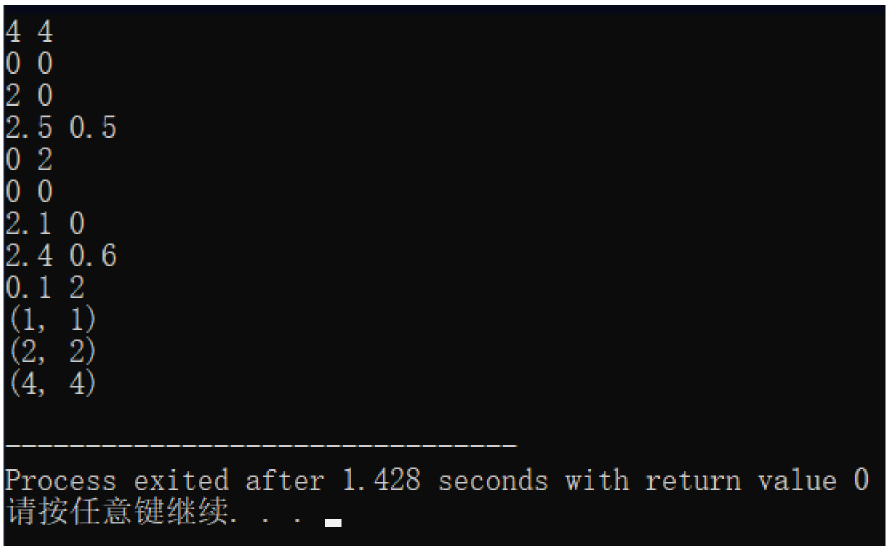
```
3 8
0 2
2 0
0 0
8 2
8 1
8 0
5 3
2 6
7 6
8 6
8 5
(1, 3)
(2, 5)
(3, 7)

--------------------------------
Process exited after 2.045 seconds with return value 0
请按任意键继续. . .
```



```
6 3
8 2
8 1
8 0
2 6
8 6
8 5
0 2
2 0
0 0
(3, 1)
(4, 2)
(5, 3)

--------------------------------
Process exited after 0.4575 seconds with return value 0
请按任意键继续. . .
```



```
4 4
0 0
2 0
2.5 0.5
0 2
0 0
2.1 0
2.4 0.6
0.1 2
(1, 1)
(2, 2)
(4, 4)

--------------------------------
Process exited after 1.428 seconds with return value 0
请按任意键继续. . .
```

In the function Insert, when obj->p1 > temp->p1 but obj->p2 < temp->p2, or similarly, when buffer->p2 > temp->p2, temp should not be added to the list. If you don't consider this condition, you will get bad result.

3.4 Purposes of each test

3.4.1 Triangles with one unrelated node in the end

Input	Output

3 4
0 2
2 0
0 0
8 0
2 6
8 6
8 1
	(1, 1)
	(2, 2)
	(3, 3)

Passed.

3.4.2 Triangles with one unrelated node in the end

Input	Output

3 6
0 2
2 0
0 0
8 0
2 6
8 6
8 5
8 2
8 1
	(1, 1)
	(2, 2)
	(3, 3)

Passed.

3.4.3 Triangles with unrelated nodes both at the first and in the end

Input	Output

3 6
0 2
2 0
0 0
8 2
8 1
8 0
2 6
8 6
8 5
	(1, 3)
	(2, 4)
	(3, 5)

Passed.

3.4.4 Triangles with unrelated nodes at the first, in the middle and in the end

Input	Output

3 8
0 2
2 0
0 0
8 2
8 1
8 0
5 3
2 6
7 6
8 6
8 5
	(1, 3)
	(2, 5)
	(3, 7)

Passed.

3.4.5 Comprehensive test

Unrelated nodes are at the first, int the middle and at the end of both points array. Test the input. Change the input order, test again. Passed.

Input	Output

9 7
6 8
10 6
10 4
10 3
10 2
10 0
6 2
6 3
	(1, 1)
	(2, 2)
	(3, 3)
	(5, 4)
	(6, 5)
	(7, 7)

6 7
4 2
3 0
2 0
1 0
0 0
0.5 1
1 2	

7 9
4 2
3 0
2 0
1 0
0 0
0.5 1
1 2
6 8
10 6
10 4
10 3
10 2
10 0
6 2
6 3
6 7
	(1, 1)
	(2, 2)
	(3, 3)
	(4, 5)
	(5, 6)
	(7, 7)

Passed.

Chapter 4: Analysis and Comments

4.1Analysis

4.1.1 Algorithm 1

The second step, build, involves a branch-and-bound search on a combinatorial tree. Every tree node represents a possible feature-to-feature correspondence, except for the root which is an auxiliary node and represents an empty set. A path from the root to any tree node represents a possible correspondence solution comprising all the feature-to-feature correspondences along the path. The tree is expanded while searching from the root. A node is added only if the new correspondence set including the new node passes a cascade of pruning tests, otherwise the subtree rooted at that node is pruned. Each of the tests efficiently filters away different types of incorrect correspondences by considering different geometry or topology information. The tests are ordered by their time complexities to achieve the best efficiency.

Now the question is, how to analyze the time and space complexities of the algorithms? By using recursion algorithm, for each round the step is combinatorial number, so the time complexity is O(2^n), meanwhile it opens space (SearchTreeADT)malloc(sizeof(struct SearchCDT)), that is to say, its space complexity is linear, equals to O(n1*n2). The time complexity of the function Vote is O(n^3), because we use the recursion algorithm to complement this task, it is O(n^2) for one round, and for n rounds it is O(n^3), meanwhile its space is (SearchTreeADT)malloc(sizeof(struct SearchCDT)), so its space complexity is O(n1*n2). The time complexity of the function Greedy is O(n^3), meanwhile its space is table[MaxSize][MaxSize] , so its space complexity O(n1*n2).

4.1.2 Algorithm 2

Another Algorithm is to gather the vote points of each node and then sum the points of each path, the path which gets the highest mark is supposed to be the most possible correspondence. On the basis of Algorithm 1, it sum the value of each path, adding the time complexity of O(n*n), so the final time complexity is O(n^3+n^2) = O(n^3). The space complexity remains the same, O(n1 *n2). This algorithm has the tendency of Fault tolerance, so it is good at fuzzy processing.

4.2Comments

It is without doubt that this algorithm is not efficient enough. For further discussion, two efforts need to take to make promising progress:

First, free the node (type SeartchTreeADT) when it is useless. During the experiment, I found that when the number of each shape is 20 or so, the computing speed will slow down to a large degree. If the used nodes are free on time, memory will release.

Second, what also need to focus is that although the algorithm is easy to think, it is not effective. The function Vote(), for example, does a lot of repeated work. The algorithm needs to be improved.

Appendix: Source Code (in C)

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>
```

```
#include <math.h>


/*****************************************************************
**************************/

/*DEFINITION*/


/*The maximum number of points of shape A and shape B, respectively.*/

#define MaxSize 100


/*The size of the table is an n1*n2 square,

so it is necessary to define an int array of size 10000,

which is the square of the MaxSize.*/

#define NodeSize 10000


/*Define the number of the results of the voting correspondence in
case the answers are not acctually unique.*/

#define AnswerNum 10


/*The deviation coefficient defined to judge whether to terminate.*/

#define epsilon 0.05


/*****************************************************************
**************************/

/*STRUCTURE*/


/*The struct node, in other words, Node, is used to store the
coordinates of the points.

Use shapeA[MaxSize] & shapeB[MaxSize] to store the data of each
shape.*/

typedef struct node Node;

struct node

{

   double x, y;

}shapeA[MaxSize], shapeB[MaxSize];


/*This structure plays a vital role in the greedy algorithm,

which is used to store the index 'i' & 'j' of the chosen node

in a bid to avoid unnecessary repetition later.*/
```

```
struct board
{
    int data[MaxSize];
    int length;
}a, b;

/*This structure is used to build the voting tree
or connect the expected answer in order to output.*/
typedef struct SearchTreeCDT* SearchTreeADT;
struct SearchTreeCDT
{
    int p1, p2;
    int ChildNum;
    int Level;
    SearchTreeADT Last;
    SearchTreeADT Next[MaxSize];
};

/*n1 & n2 represent the number of points of the two shapes,
namely shapeA & shapeB, respectively.*/
int n1, n2, i;

/*the integer MaxLevel represents he maximum of the length of path.*/
int MaxLevel = 0;

int count = 0, error = 0, SizeFinal = 0, AnswerFinal = 0;

/*This is the voting table, which represents the vote numbers of each
node.
The original votes of all the correspondence are 0.*/
int table[NodeSize] = {0};

/*******************************************************************
**************************/
/*FUNCTIONS*/

/*MyAbs(double x) is a function
```

```
which is able to return the absolute value of the input variable with
double type.*/

double MyAbs(double x);


/*Read() is a function to read the input and store the input to the
appropriate address.*/

void Read(void);


/*Create() is a function to create a head node of the voting tree.*/

SearchTreeADT Create(void);


/*Build(SearchTreeADT p, int left, int right) is a function to build
the voting tree,

what deserves your attention is that it can terminate when the points
are not matched,

and the function designed to judge whether the points in the groups
are matched or not

is in the following.*/

void Build(SearchTreeADT p, int left, int right);


/*Match(int a1, int b1, int a2, int b2, int a3, int b3):

judge whether the points in the groups are matched or not

using mathematical methods.*/

bool Match(int a1, int b1, int a2, int b2, int a3, int b3);


/*The function Vote(SearchTreeADT p) is designed to vote for each
matching.

By caculating the electors of each nodes, we will have somthing to
take into consideration

when deciding which pair is the most perfect correspondence.*/

void Vote(SearchTreeADT p);


void PrintVote(void);


/*Function NotIn(struct board a, int x) is used to judge whether
integer x is in the a.data[MaxSize]

and returns a bool.*/

bool NotIn(struct board a, int x);


/*Function In(int x) is designed to insert the integer x into the
```

```
integer array a.data[MaxSize].*/
void In(int x);


/*The function Insert(SearchTreeADT p, int x, int y)
is used to insert the node(x, y) in the right place.
In this program, the node is placed in increasing order.*/
SearchTreeADT Insert(SearchTreeADT p, int x, int y);


/*The functon Greedy(void) includes the principle of greedy method.
for each step, this argorithm always choose a node from the left
nodes
which has the maximum number of votes.*/
SearchTreeADT Greedy(void);


/*PrintAnswer(SearchTreeADT p) can print the final answer in
increasing order.*/
void PrintAnswer(SearchTreeADT p);


/*******************************************************************
***************************/
/*MAIN*/
int main ()
{
    /*The main function shows the idea of the encapsulation.
    There are several steps included in the main function.
    First, read the input data;
    Second, Build the voting tree;
    Third, vote on the basis of the tree constructed earlier;
    Fourth, make more judegment according to the result;
    Fifth, print the final result, which is a possible match.*/


    Read(); //Read the data.
    SearchTreeADT SearchTree = Create(); //Create the head node.
    Build(SearchTree, 0, 0); //Build the voting tree.


    /*You can use the printf below to analyze the test cases.
    Notice that MaxLevel represents the length of the possible
correspondence,
    but it may not be the final result.
    The integer count represents the nodes chosen,
    and the integer error represents the nodes being rejected.*/
    //printf("MaxLevel = %d count = %d, error = %d\n", MaxLevel,
count, error);


    Vote(SearchTree); //Record the vote result on a table.


    /*You can use the function PrintVote() to print the vote table.*/
    //PrintVote();
    SearchTreeADT Answer = Greedy(); //Use greedy methodn to choose
the match result.
    PrintAnswer(Answer); //Print the final answer in increasing
order.
    return 0;
}


/*****************************************************************
***************************/
/*MyAbs*/
double MyAbs(double x)
{
    if (x < 0) return -x;
    return x;
}


/*****************************************************************
***************************/
/*Read*/
void Read(void)
{
    double xi, yi; //xi & yi temporarily storage the cordinates of
the nodes.
    scanf("%d %d", &n1, &n2);
    for (i = 0; i < n1; i++)
    {
        scanf("%lf %lf", &xi, &yi);
        shapeA[i].x = xi;
        shapeA[i].y = yi;
    }
    for (i = 0; i < n2; i++)
    {
        scanf("%lf %lf", &xi, &yi);
        shapeB[i].x = xi;
        shapeB[i].y = yi;
    }
}

/*******************************************************************
****************************/
/*Create*/
SearchTreeADT Create(void)
{
    int i;
    SearchTreeADT   Head   =   (SearchTreeADT)malloc(sizeof(struct
SearchTreeCDT)); //Open space for the head node.
    Head->ChildNum = 0;
    Head->Level = 0;
    for (i = 0; i < MaxSize; i++)
        Head->Next[i] = NULL;
    return Head;
}

/*******************************************************************
****************************/
/*Match*/
bool Match(int p1, int n1, int p2, int n2, int p3, int n3)
{
    /*The main point in this function is that there are two aspects
to define whether the nodes are a match.
    The first rule is whether the ratio of the corresponding side
length is close enough;
    The second rule is whether the angles formed by the both sides
are close enough.*/

    /*The cordinates of each point*/
    double ax1, ay1, ax2, ay2, ax3, ay3, bx1, by1, bx2, by2, bx3,
by3;


    /*The length triangle side*/
    double aL1, aL2, bL1, bL2;


    ax1 = shapeA[p1-1].x;
    ay1 = shapeA[p1-1].y;
    ax2 = shapeA[p2-1].x;
    ay2 = shapeA[p2-1].y;
    ax3 = shapeA[p3-1].x;
    ay3 = shapeA[p3-1].y;


    bx1 = shapeB[n1-1].x;
    by1 = shapeB[n1-1].y;
    bx2 = shapeB[n2-1].x;
    by2 = shapeB[n2-1].y;
    bx3 = shapeB[n3-1].x;
    by3 = shapeB[n3-1].y;


    aL1 = sqrt((ax1-ax2)*(ax1-ax2) + (ay1-ay2)*(ay1-ay2)); //Caculate
the length of aL1.
    aL2 = sqrt((ax3-ax2)*(ax3-ax2) + (ay3-ay2)*(ay3-ay2)); //Caculate
the length of aL2.
    bL1 = sqrt((bx1-bx2)*(bx1-bx2) + (by1-by2)*(by1-by2)); //Caculate
the length of bL1.
    bL2 = sqrt((bx3-bx2)*(bx3-bx2) + (by3-by2)*(by3-by2)); //Caculate
the length of bL2.
    if (MyAbs(aL1/bL1-aL2/bL2) > epsilon) return false;


    double CosTheta1, CosTheta2;
    CosTheta1  =  ((ax1-ax2)*(ax3-ax2)+(ay1-ay2)*(ay3-ay2))/aL1/aL2;
//Caculate the value of cos(theta1).
    CosTheta2  =  ((bx1-bx2)*(bx3-bx2)+(by1-by2)*(by3-by2))/bL1/bL2;
//Caculate the value of cos(theta2).
    if  (MyAbs(MyAbs(acos(CosTheta1)/acos(CosTheta2))-1)  >  epsilon)
return false;


    return true;
}
```

```
/*******************************************************************
**************************/
/*Build*/
void Build(SearchTreeADT p, int left, int right)
{
    int j, k;
    for (j = left; j < n1; j++)
    for (k = right; k < n2; k++)
    {
        if  (((p->Level  +  1)  <  3)  ||  ((p->Level  +  1)  >=  3  &&
Match(p->Last->p1, p->Last->p2, p->p1, p->p2, j+1, k+1)))
        {
            /*Below are steps to add new node to the p
            The address of the new node is p->Next[p->ChildNum].*/
            p->Next[p->ChildNum]                                    =
(SearchTreeADT)malloc(sizeof(struct SearchTreeCDT));
            p->Next[p->ChildNum]->ChildNum = 0;
            p->Next[p->ChildNum]->Level = p->Level+1;
            p->Next[p->ChildNum]->p1 = j+1;
            p->Next[p->ChildNum]->p2 = k+1;
            p->Next[p->ChildNum]->Last = p;
            int t;
            for (t = 0; t < MaxSize; t++)
                p->Next[p->ChildNum]->Next[t] = NULL;

            /*The maximum depth of the path is updated in real time.*/
            if (MaxLevel < p->Next[p->ChildNum]->Level)
                MaxLevel = p->Next[p->ChildNum]->Level;
           count++; ////////////////////////////////
           /*Use recursion to build the voting tree.
            The key point is to consider the child node as the root
node of the subset.*/
           Build(p->Next[p->ChildNum++], j+1, k+1);
        }
        else error++;
    }
}
```

```
/*******************************************************************
***************************/

/*Vote*/

void Vote(SearchTreeADT p)

{

    int i;

    if (p->ChildNum == 0)

    {

    /*Ignore the effects of the nodes with level less than 3.

        As similar problems are meaningful only when the number of
points is at least 3.*/

        if (p->Level > 2)

        {

        table[(p->p1-1)*n2+(p->p2-1)] += 1;

            while (p->Last->Level >= 1)

        {

            /*When reaching the leaf,

                a new path has been found, in other word,

                the path number of every ancestors of this leaf

                should plus 1.*/

                table[(p->Last->p1-1)*n2+(p->Last->p2-1)] += 1;

                p = p->Last;

            }

        }

    }

    /*Recursion is also used.

    Consider the child node as the root node of the subset

    and then repeat the process.*/

    else

    {

        for (i = 0; i < p->ChildNum; i++)

            Vote(p->Next[i]);

    }

}


/*******************************************************************
```

```
***************************/
/*PrintVote*/
void PrintVote(void)
{
   int i, j;
   for (i = 0; i < n1; i++)
   for (j = 0; j < n2; j++)
   {
      printf("%4d ", table[i*n2+j]);
      if (j == (n2-1)) printf("\n");
   }
}

/*******************************************************************
***************************/
/*NotIn*/
bool NotIn(struct board a, int x)
{
   /*To judge whether the integer x is in the integer array
a.data[MaxSize].*/
   if (!a.length) //There is no integer stored in the a.data[MaxSize]
yet.
      return true;
   else
   {
      int i, flag = 0;
      for (i = 0; i < a.length; i++)
      {
         if (x == a.data[i])
         {
            flag = 1;
            break;
         }
      }
      if (flag) return false;
      return true;
   }
```

```
}


/*******************************************************************
***************************/

/*In*/

void In(int t, int x)

{

   /*Add the integer x to the integer array a.data[MaxSize].*/

   if (t == 0)

      a.data[a.length++] = x; //t=0 represents a.data[MaxSize].

   else if (t == 1)

      b.data[b.length++] = x; //t=1 represents b.data[MaxSize].

}


/*******************************************************************
***************************/

/*Insert*/

SearchTreeADT Insert(SearchTreeADT p, int x, int y)

{

   /*In this part, the data structure SearchTreeADT is used as a
linked list.

   When a node is to be inserted, considering the value of p1 & p2
of this node,

   it will be added at the front of the list, in the middle of the
list

   or at the end of the list.*/

   SearchTreeADT    temp    =    (SearchTreeADT)malloc(sizeof(struct
SearchTreeCDT));

   int i;

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

      temp->Next[i] = NULL;

   temp->p1 = x;

   temp->p2 = y;

   if (!p->Next[0]) //No node has been connected. temp is added at
the top.

      p->Next[0] = temp;

   else

   {

      SearchTreeADT obj = p->Next[0], buffer = Create();
```

```
       if (obj->p1 > temp->p1)

       {

           /*The node is priority to all the other nodes,

           so it is added right after the root.*/

           if (obj->p2 < temp->p2) return p; /*obj->p1 > temp->p1,
but obj->p2 < temp->p2,

                                          it   means   there   is   a
conflict, so the temp should not be added.*/

           temp->Next[0] = p->Next[0];

          p->Next[0] = temp;

       } else

       {

          /*move the node obj*/

           while (obj->p1 < temp->p1)

          {

             buffer = obj;

             obj = obj->Next[0];

             if (!obj) break;

          }

          if (!obj) //obj=NULL, temp is added to the end of the list.

          {

             if (buffer->p2 > temp->p2) return p;

              buffer->Next[0] = temp;

             free(obj);

          } else

          {

             if (obj->p2 < temp->p2 || buffer->p2 > temp->p2) return
p;

              /*temp is added btween the two nodes.*/

              temp->Next[0] = obj;

             buffer->Next[0] = temp;

           }

       }

   }

   return p;

}


/*******************************************************************
```

```
***************************/
/*Greedy*/
SearchTreeADT Greedy(void)
{
    /*For each circulation, find the eligible nodes who has the maximum number of electors
    and choose it as a match, when the circulation is over, we will get one or a few results.
    For the 'one result' circumstance, it is the answer;
    But for the 'a few results' condition, futher discussion has to be taken.*/

    int i, j, x, y, Max;
    a.length = b.length = 0;
    SearchTreeADT p = Create();

    while (true)
    {
        Max = 0; //Max is used to store the maximum of the vote number of the node.
        for (i = 0; i < n1; i++)
        {
            if (!NotIn(a, i)) continue;
            for (j = 0; j < n2; j++)
            {
                if (!NotIn(b, j)) continue;
                if (Max < table[i*n2+j])
                {
                    Max = table[i*n2+j]; //Find the maximum vote value of each term.
                    x = i; //Use x to store i in case it may be used later.
                    y = j; //Use y to store j in case it may be used later.
                }
            }
        }
        if (Max)
        {
            In(0, x); //Insert x into board a.
            In(1, y); //Insert y into board b.
            p = Insert(p, x+1, y+1); //Insert node (x+1, y+1) to the
list p;
        }
        else
        {
            /*If Max is equal to 0, it means there is no node available
on the table,
            or the rest of the nodes on the table has the vote value
0.
            Both of the conditions won't make progress any more.
            So it is time to terminate.*/
            break;
        }
    }
    return p;
}

/*******************************************************************
***************************/
/*PrintAnswer*/
void PrintAnswer(SearchTreeADT p)
{
    /*In this project, the answer is printed in increasing order.*/
    p = p->Next[0];
    while (p)
    {
        printf("(%d, %d)\n", p->p1, p->p2);
        p = p->Next[0];
    }
}
```

Declaration

I hereby declare that all the work done in this project titled "Project 1. Voting Tree" is of my independent effort.