# Closest Pair - 分治法

## 【算法思想】

首先对所有的点按照x坐标进行从小到大的排序，然后利用二分法进行分割，知道区间范围变为2 或者3，然后求出这2个点或者3个点的距离。

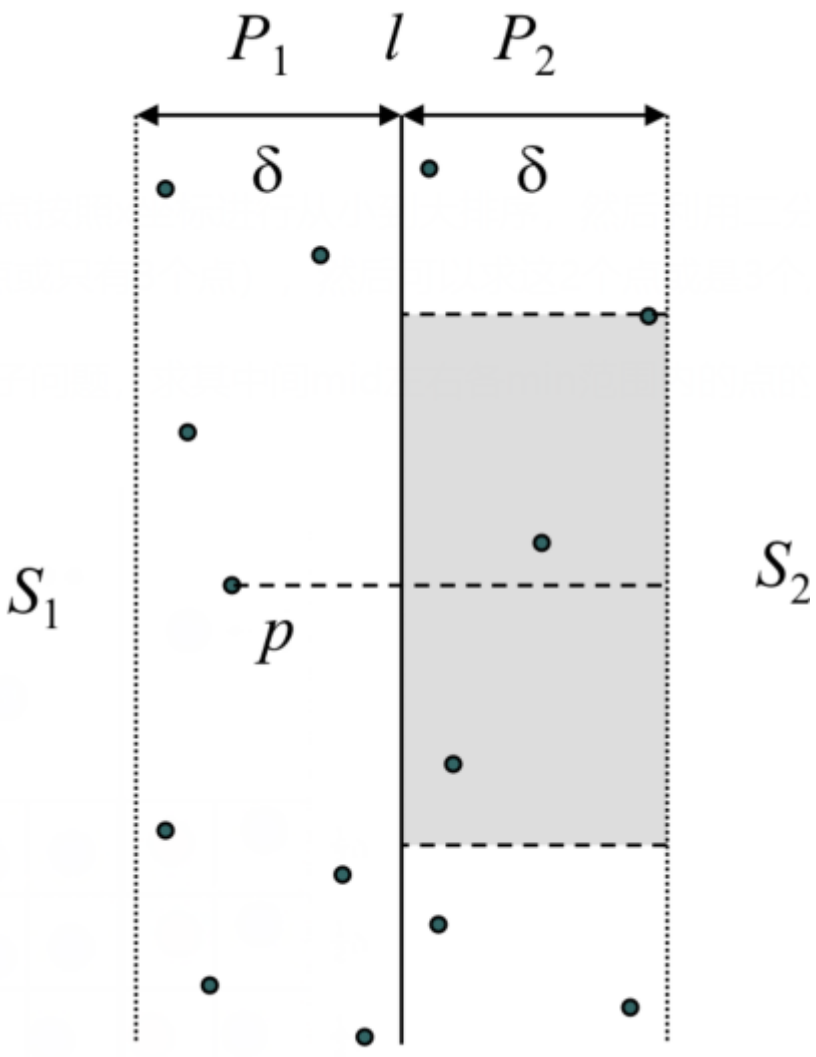
对于每一部分的子问题，还需要考虑边界附近点与另一个集合点之间的距离。

怎样考虑相邻子问题之间两个点的距离是否会小于两个子问题内部的最小距离min_len呢？

一个暴力的方法是嵌套循环遍历两个子问题的所有点求出两个子问题之间的最小距离。显然，时间复杂度为O(n^2)。

我们是否可以做的更好？

考虑下图：




从上图可以看出，实际上我们只需考虑离边界线的距离小于min_len的点就行。

除了考虑x轴上距离边界线小于min_len，在y轴上我们也可以略过很多不需要考察的点。比如：对于在S1集合上的p点，实际上我们只需考虑以它为半径且半径小于min_len点就行，但这种方法比较复杂。我们可以极端的假设p点恰好边界线上，那么需要考察的点实际上是上图灰色部分。因此，实际操作中，我们不仅需要将点以x轴上排序，还需要在y轴上进行排序。

## 【代码】

```
#include<iostream>
#include<string.h>
#include<math.h>
#include <fstream>
#include<iomanip>
#include<algorithm>
using namespace std;
int k=0;


struct Record
{
    int c1,c2;
    double ds;
}record[10000];


struct Points
{
    double x,y;
}*points;


double dis(int a,int b)
{
    double dx = points[a].x-points[b].x,dy = points[a].y-points[b].y;
    return sqrt(dx*dx+dy*dy);
}


double min(double a,double b)
{
    return a>b?b:a;
}


int cmpx(const Points &a,const Points &b)
{
    if(a.x<b.x)
        return 1;
    else
        return 0;
}

int cmpy(const Points &a,const Points &b)
{
    if(a.y<b.y)
        return 1;
    else
        return 0;
}

void storage(double c,int l,int r)
```

```
{   //记录最近点对，用于输出
    record[k].c1 = l;
    record[k].c2 = r;
    record[k++].ds = c;
}


double ClosestPair(int l, int r)
{
    if(1==r-l)
    {
        double d = dis(l,r);
        storage(d,l,r);
        return d;
    }
    if(2==r-l)
    {
        double ll = dis(l,l+1);
        double rr = dis(l+1,r);
        storage(ll,l,l+1);
        storage(rr,l+1,r);
        return min(ll,rr);
    }
    int mid = l+((r-l)>>1);
    double dl = ClosestPair(l,mid);
    double dr = ClosestPair(mid+1,r);
    double Min=min(dl,dr);
    int h1=mid,h2=mid;
    while((points[mid].x-points[h1].x<=Min && h1>=l) || (points[h2].x-points[mid].x<=Min &&
h2<=r))
    {   //得到mid附近2&距离内的区间范围
        h1--;
        h2++;
    }
    sort(&points[l],&points[r],cmpy); //将mid附近2&距离内的点按照y坐标从小到大排序
    for(int i=h1+1;i<h2;i++)
    {
        int k=(i+11)>h2?h2:(i+11);  //只需要比较附近11个点即可
        for(int j=i+1;j<k;j++)
        {
            double d=dis(i,j);
            if(d>=Min)
                break;
            Min = d;
            storage(d,i,j);
        }
    }
    return Min;
}


int main()

{
```

```
    ifstream infile("./points.txt", ios::in);
    char buf[30];
    double (*data)[2];
    char *subarr=NULL;
    char *delims = ",";
    int n;
    cout<<"Please input the number of points(eg.input 10 int this test):"<<endl;
    cin>>n;
    points = new struct Points[n];
    data = new double[n][2];
    if(!infile.is_open())
    {
        cout<<"Error opening file!";
        exit(1);
    }
    int i=0,j=0,p=0;
    while(infile.getline(buf,30)!=NULL)
    {
        j = 0;
        subarr = strtok(buf,delims);
        while(subarr!=NULL)
        {
            data[i][j] = atof(subarr);
            subarr = strtok(NULL,delims);
            j++;
        }

        points[p].x = data[i][0];
        points[p++].y = data[i][1];
        i++;

    }
    infile.close();
    sort(&points[0],&points[n],cmpx); //按横坐标从小到大排序
    double value = ClosestPair(0,n-1);
    int x,y;
    for(i=0;i<k;i++)
    {
        if(record[i].ds==value)
        {

            x = record[i].c1;
            y = record[i].c2;
            cout<<"points("<<points[x].x<<","<<points[x].y<<")and("<<points[y].x<<","
<<points[y].y<<")"<<endl;
        }
    }
    cout<<"have the shortest distance :"<<value<<endl;
    return 0;
}
```