Lab 2: 求平面上n个顶点的最近点对问题

PB21020718 曾健斌

一、实验内容

编程实现求平面上n个顶点的最近点对问题

对文件`data.txt`的所有点，求距离最近的一对及其距离

程序输入：

- 字符串文件`data.txt`，每行三个数，分别表示顶点编号和其横纵坐标

程序输出：

- 输出距离最近的一对顶点编号，及其距离值

二、算法实现

结构体与类定义

首先定义两个结构体：

```
struct Coordinate{
    int index;
    float x;
    float y;
};

struct Pair{
    Coordinate A;
    Coordinate B;
    float distance;
};
```

其中`Coordinate`用于存储每一个点的序号以及坐标信息，`Pair`用于记录点对以及它们之间的距离

下面是算法实现所定义的类：

```
class ClosestPoints
{
private:
    Pair closest;
public:
    vector<Coordinate> Coordinates;             //Coordinates存储原始点对数据
    vector<Coordinate> XSort;                   //XSort中的数组以x坐标为序
    vector<Coordinate> YSort;                   //YSort中的数组以y坐标为序
    ClosestPoints();
    ~ClosestPoints();
    void ReadFile(string filename);
    void Bruteforce(void);                      //1
    void SortCoordinate(void);
    Pair Divide_Sub(vector<Coordinate> X, vector<Coordinate> Y);
    Pair Combine(vector<Coordinate> Yl, vector<Coordinate> Yr, float pivot, Pair p);
    void Divide(void);
    void PrintClosest(void);
};
```

其中`closest`字段用于存储最近点对信息，本实验使用模板类`vector`，并且将借助其高效的按属性划分方法`sort()`来实现后续排序

`//1`中还定义了`Bruteforce()`来暴力求解最近点对问题，意欲与分治算法对比

原始排序以及划分有序数组算法

```
bool
Xcompare(const Coordinate &a, const Coordinate &b)   //此处Xcompare与Ycompare作为sort()的比较函数
{
    return a.x < b.x;
}

bool
Ycompare(const Coordinate &a, const Coordinate &b)
{
    return a.y < b.y;
}

void
ClosestPoints::SortCoordinate()
{
    vector<Coordinate> tmp = Coordinates;
    sort(tmp.begin(), tmp.end(), Xcompare);
    XSort = tmp;
    sort(tmp.begin(), tmp.end(), Ycompare);
    YSort = tmp;
}
```

以上算法使用`std::sort()`自定义比较函数的功能实现原始数组的简易排序

```
vector<vector<Coordinate>>
Seperate(vector<Coordinate> Y, float pivot)
{
    vector<vector<Coordinate>> Ypair;
    vector<Coordinate> Yl, Yr;
    for(int i = 0; i < Y.size(); ++i){
        if(Y[i].x <= pivot){
            Yl.push_back(Y[i]);
        }
        else{
            Yr.push_back(Y[i]);
        }
    }
    Ypair.push_back(Yl);
    Ypair.push_back(Yr);
    return Ypair;                                   //返回 Yl 与 Yr 两个数组，这种写法属于是python 写习惯了
}
```

按照枢轴`pivot`进行对于有序数组 Y 的划分，并**保持其序**

核心分治步骤

Divide

```
Pair
ClosestPoints::Divide_Sub(vector<Coordinate> X, vector<Coordinate> Y)
{
    if(X.size() <= 3)
    {
        return Closestamong3(X);            //1
    }
    vector<vector<Coordinate>> Ypair;
    vector<Coordinate> Xl, Xr, Yl, Yr;
    Pair left, right, Winner;
    float pivot = X[X.size()/2 - 1].x;
    for(int i = 0; i < X.size()/2; ++i){            //此处对 X 与 Y 的处理略有不同，前者之间利用其有序的特性进行对半切分
        Xl.push_back(X[i]);                         //后者则是以 X 的中位数为枢轴调用Seperate()进行划分
    }
    for(int i = X.size()/2; i < X.size(); ++i){
        Xr.push_back(X[i]);
    }
    Ypair = Seperate(Y, pivot);
    Yl = Ypair[0];
    Yr = Ypair[1];
    left = Divide_Sub(Xl, Yl);
    right = Divide_Sub(Xr, Yr);
    if(left.distance < right.distance){
        Winner = left;
    }
    else{
        Winner = right;
    }
    return Combine(Yl, Yr, pivot, Winner);          //2
}
```

Conquer

`//1`用于处理小于三的数组的临界情况，具体实现如下：

```
float
Distance(Coordinate a, Coordinate b)
{
    return sqrt(pow(a.x-b.x, 2) + pow(a.y-b.y, 2));
}


Pair
Closestamong3(vector<Coordinate> X)
{
    Pair c;
    if(X.size() == 2){
        c.distance = Distance(X[0], X[1]);
        c.A = X[0];
        c.B = X[1];
        return c;
    }
    if(Distance(X[0], X[1]) < Distance(X[0], X[2])){
```

```
        if(Distance(X[0], X[1]) < Distance(X[1], X[2])){
            c.distance = Distance(X[0], X[1]);
            c.A = X[0];
            c.B = X[1];
        }
        else{
            c.distance = Distance(X[1], X[2]);
            c.A = X[1];
            c.B = X[2];
        }
    }
    else{
        if(Distance(X[0], X[2]) < Distance(X[1], X[2])){
            c.distance = Distance(X[0], X[2]);
            c.A = X[0];
            c.B = X[2];
        }
        else{
            c.distance = Distance(X[1], X[2]);
            c.A = X[1];
            c.B = X[2];
        }
    }
    return c;
}
```

Combine

`//2`处进行合并操作，实现如下：

```
vector<Coordinate>
CombineSort(vector<Coordinate> A, vector<Coordinate> B)
{                           //本函数用于用切分在buffer中的两个点数组 Yl、Yr 进行重新组合排序
    vector<Coordinate> C;
    int i, j;
    i = j = 0;
    while(i < A.size() || j < B.size()){
        if(i == A.size()){
            while(j < B.size()){
                C.push_back(B[j]);
                ++j;
            }
        }
        else if(j == B.size()){
            while(i < A.size()){
                C.push_back(A[i]);
                ++i;
            }
        }
        else if(A[i].y <= B[j].y){
            C.push_back(A[i]);
            ++i;
        }
        else{
            C.push_back(B[j]);
            ++j;
        }
    }
    return C;
```

```
}

vector<Coordinate>
SetBufferY(vector<Coordinate> Yl, vector<Coordinate> Yr, float pivot, float delta)
{                           //以两子问题得到的最短距离为基准建立一个 pivot 附近的槽
    vector<Coordinate> Yll, Yrr;
    vector<vector<Coordinate>> Ypair;
    Ypair = Seperate(Yl, pivot - delta);
    Yll = Ypair[1];
    Ypair = Seperate(Yr, pivot + delta);
    Yrr = Ypair[0];
    return CombineSort(Yll, Yrr);
}

Pair
ClosestPoints::Combine(vector<Coordinate> Yl, vector<Coordinate> Yr, float pivot,
Pair p)
{
    vector<Coordinate> Yt;
    Pair q = p;
    Yt = SetBufferY(Yl, Yr, pivot, p.distance);
    if(Yt.size() >= 9){
        for(int i = 0; i < Yt.size()-8; ++i){
            for(int j = 0; j < 7; ++j){
                if(Distance(Yt[i], Yt[i + j + 1]) <= q.distance){
                    q.distance = Distance(Yt[i], Yt[i + j + 1]);
                    q.A = Yt[i];
                    q.B = Yt[i + j + 1];
                }
            }
        }
    }
    else{                       //此处务必注意对于buffer区小于9个点的情况作另外的分类
        for(int i = 0; i < Yt.size()-1; ++i){
            for(int j = i+1; j < Yt.size(); ++j){
                if(Distance(Yt[i], Yt[j]) <= q.distance){
                    q.distance = Distance(Yt[i], Yt[j]);
                    q.A = Yt[i];
                    q.B = Yt[j];
                }
            }
        }
    }

    return q;
}

```

最后处理开始的调用：

```
void
ClosestPoints::Divide()
{
    closest = Divide_Sub(XSort, YSort);
}
```

三、实验结果

```
19:02 zeng@ocoubuntu /home/zeng/Documents/alg-lab/alg-lab2/source/cpp_source
% g++ closest_points.cpp main.cpp readfile.cpp timer.cpp -o main
19:03 zeng@ocoubuntu /home/zeng/Documents/alg-lab/alg-lab2/source/cpp_source
% ./main
Point 7119 and Point 5826 are closest pair, the distance is 2.80761
Time: 69.9212ms
Point 5826 and Point 7119 are closest pair, the distance is 2.80761
Bruteforce Time: 2896.18ms
```

可见相比于暴力求解的$O(n^2)$复杂度，时间复杂度为$O(\log n)$的分治算法时间非常之快，可见分治策略影响之深远

四、实验总结

- 本次实验主要是对于最近点对的一种$O(n \log n)$的分治算法的实现
- 从预排序的思想中学习到很多，其**将排序结果保留到后续子问题**的思想很值得在其他算法中借鉴