2112233062-何智鹏-第一次作业

1、逆序数

1.1 逆序数问题的形式化表示：

如果数列的第 `i` 个和第 `j` 个元素，如果满足 `i < j` 且 `a[i] < a[j]`，则其为一个逆序对数。

要注意的是，一个元素可以不只是在一个逆序对中存在。如果 `k > j > i` 且 `a[i] > a[j] > a[k]`，那么这里有两个含 `a[i]` 的逆序对，分别是 `(a[i], a[j])` 和 `(a[i], a[k])`，`a[i]` 是可以使用多次的。

1.2 设计分析

将序列从中间分开，将逆序对分成三类：

- 两个元素都在左边；
- 两个元素都在右边；
- 两个元素一个在左，一个在右（跨区间）。

这个时候注意到一个很重要的性质，左右半边的元素在各自任意调换顺序，是不影响第三步计数的，因此我们可以数完就给它排序。这么做的好处在于，如果序列是有序的，会让第三步计数很容易。

1.3 算法

C++ | 复制代码

```cpp
class Solution {
public:

    int merge(vector<int>& nums, int l, int r) {
        if (l >= r) return 0;
        int mid = l + r >> 1;
        int res = merge(nums, l, mid) + merge(nums, mid + 1, r);
        int i = l, j = mid + 1;
        vector<int> temp;  //存一下归并过后的结果
        //归并的过程
        while (i <= mid && j <= r) {
            if (nums[i] <= nums[j]) temp.push_back(nums[i ++]);
            else {
                temp.push_back(nums[j ++]);
                res += mid - i + 1;
            }
        }
        //扫尾
        while(i <= mid) temp.push_back(nums[i ++]);
        while(j <= r) temp.push_back(nums[j ++]);

        //将临时数组排好序的结果放回去
        i = l;
        for(auto x : temp) nums[i ++] = x;
        return res;
    }

    int inversePairs(vector<int>& nums) {
        return merge(nums, 0, nums.size() - 1);
    }
};
```

2. 二维空间最近点对

- 通过dl = minDist(lx, ly, m)来完成左半部分的计算；
- 通过dr = minDist(rx, ry, n–m)完成右半部分的计算；
- 最后通过combine(py, n, fx, delta)将两半部分的结果整合在一起；

下面是对于分治算法的调用部分，调用之前需要分别将其中的点按x轴和按y轴进行排序操作，并且将排完序的点放置在新的存储空间中；

C++ 复制代码

```cpp
double callMinDist(node* p, int n){

    node* px = (node*)malloc(n*sizeof(node)); //n主要是用于此处的空间申请;
    node* py = (node*)malloc(n*sizeof(node));

    mergeSortX(p, px, 0, n-1); //按点的x轴值排序;
    copynode(px, p, 0, n-1);

    mergeSortY(p, py, 0, n-1); //按点的y轴值排序;
    copynode(py, p, 0, n-1);

    double min = minDist(px, py, n);
    free(px);
    free(py);
    return min;
}
```

下面就是分治算法的主体：

C++ 复制代码

```cpp
double minDist(node* px, node* py, int n)
{
        int m=n/2;
        double fx = px[m].x;

        node* lx = (node*)malloc(m*sizeof(node));
        node* ly = (node*)malloc(m*sizeof(node));
        node* rx = (node*)malloc((n-m)*sizeof(node));
        node* ry = (node*)malloc((n-m)*sizeof(node));

        copynode(lx, px, 0, m-1);
        copynode(rx, px, m, n-1);
        copynode(ly, py, 0, m-1);
        copynode(ry, py, m, n-1);

        double d1 = minDist(lx, ly, m); //m is number of elements;
        double dr = minDist(rx, ry, n-m);

        double delta = min(d1, dr);
        double d = combine(py, n, fx, delta);
        free(lx);
        free(ly);
        free(rx);
        free(ry);

        return min(delta, d);
}
```

关键之处在于combine函数:

C++ 复制代码

```cpp
double combine(node* py, int n, double lx, double delta)
{
    int num; double d=MAX;
    double tempd;
    node* temp = (node*)malloc(n*sizeof(node));

    int j=0;

    for(int i=0; i<n; i++)
    {
        if(fabs(py[i].x - lx)<= delta){ //求取在区间范围内的点;
            temp[j].x = py[i].x;
            temp[j].y = py[i].y;
            j++;
        }
    }

    num = j;  //temp中的元素

    for(int i=0; i<num; i++)
    {
        for(j=i+1; j<(i+8) && (j<num); j++)
        {
            tempd = dist(&temp[i], &temp[j]);
            if(tempd < d)
                d=tempd;
        }
    }

    free(temp);
    return d;
}
```

3. 时间复杂度推导

3.1 整数乘法

$$T(n)=\begin{cases} O(1) & n=1 \\ 3T(n/2)+O(n)=n^{\log_2^3}=O\left(n^{1.59}\right) & n>1 \end{cases}$$

3.2 矩阵乘法

$$T(n) = \begin{cases} m & \text{若 } n = 1 \\ 7T(n/2) + 18(n/2)^2 a] & \text{若 } n \geqslant 2 \end{cases}$$

3.3 最近点对

$C(n) = 1$

$C(n) = 3$

$C(n) = 2C(n/2) + \Theta(n), n > 3$

解得： $C(n) = \Theta(n \log n)$ 。

$$T(n) = C(n) + \Theta(n \log n) = \Theta(n \log n)$$