《算法设计与分析》

课程实验报告

专业：计算机科学与技术

班级：2021211307

姓名：陈朴炎

学号：2021211138

目录

一、问题描述

采用线性时间选择算法，根据基站 k-dist 距离，挑选出：

k-dist 值最小的基站

k-dist 值第 5 小的基站

k-dist 值第 50 小的基站

k-dist 值最大的基站

要求：

1. 在排序主程序中设置全局变量，记录选择换份过程的递归层次

2. 参照将以 PPT，将教科书上的“一分为二”的子问题划分方法，改进为“一分为三”，比较这两种划分方式下，选择过程递归层次的差异。

二、算法分析设计

2.1 类定义

C++

```
class BaseStation{
public:
    int ENODEBID;
    double K_DIST;
    BaseStation(){
        ENODEBID = -1;
        K_DIST = 0;
    }
};
```

定义了一个 BaseStation 类，用来存放基站的 ENODEBID 值和 K_DIST 值。同时定义了一个默认构造函数，当测试到 ENODEBID == -1 时，就知道该对象是不合法的。

2.2 宏定义

定义了一个宏 STRAIGHT_SORT_LENGTH

```
C++
#define STRAIGHT_SORT_LENGTH 20
```

表示当待搜索数组小于这个长度时，直接排序来求第 k 小元素。

2.3 全局变量

定义了一个全局变量 recursionDepth

```
C++
int recursionDepth = 0;
```

用来在算法中递归调用，查看函数的递归深度。

2.4 主程序设计

1、主程序首先要定义一个数组，用来存放所有的基站数据

2、第二步打开基站数据文件，读取数据，存放到数组中

3、用一分为三的线性时间搜索，分别找出最小的 K_DIST 值和基站 ID、第 5 小的 K_DIST 值和 ID、第 50 小的 K_DIST 值和 ID 还有最大的 K_DIST 值和 ID，并查看他们的递归深度。

4、用一分为二的线性时间搜索，分别找出最小的 K_DIST 值和基站 ID、第 5 小的 K_DIST 值和 ID、第 50 小的 K_DIST 值和 ID 还有最大的 K_DIST 值和 ID，并查看他们的递归深度。

2.5 分隔算法设计

```
C++
int partition(vector<BaseStation>& arr, int low, int high,
const BaseStation& x = BaseStation()){
    // 在数组中找到划分基准
```

```
    for(int i = 0; i <= high;i++){
        if(x.ENODEBID == arr[i].ENODEBID){
            swapBaseStations(arr, i, high);
            break;
        }
    }

    // 将划分基准放到末尾
    BaseStation pivot = arr[high];
    int i = low -1;
    for (int j = low; j <= high; j++){
        if(arr[j].K_DIST < pivot.K_DIST){
            i++;
            swapBaseStations(arr, i, j);
        }
    }
    swapBaseStations(arr, i+1, high);
    return i + 1;
}
```

该函数有四个参数，作用如下：

arr：这是一个存储 BaseStation 对象的数组引用。该函数将对这个数组进行划分操作。

low：这是数组的低位索引，表示当前操作的数组范围的起始位置。

high：这是数组的高位索引，表示当前操作的数组范围的结束位置。

x：这是一个 BaseStation 类型的常量引用，表示划分基准。默认情况下，它被初始化为一个默认构造的 BaseStation 对象。在函数内部，它的作用是指定划分基准。

该函数的作用是重新排列数组的元素，以确保所有小于基准的元素在其左侧，而所有大于基准的元素在其右侧。

1、寻找基准索引： 第一个循环遍历数组，查找具有 ENODEBID 等于 x.ENODEBID 的元素的索引。一旦找到，它将该元素与索引为 high 的元素交

换。这样做是为了确保基准被放置在数组的末尾。

2、设置基准： 然后将基准设置为索引为 high 的元素。

3、划分数组： 第二个循环从 low 到 high-1 遍历数组。如果元素的 K_DIST 小于基准的 K_DIST，则将其与索引为 i 的元素交换，并递增 i。这个过程有效地将所有小于基准的元素移到其左侧。

4、将基准放置在正确的位置： 最后，将基准与索引为 i + 1 的元素交换，使基准位于数组中的正确位置。

2.6 排序算法设计

```C++
void quickSort(vector<BaseStation>& arr, int low, int high) {
    if (low < high) {
        int pivotIndex = partition(arr, low, high);

        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}
```

该函数是经典的递归快速排序算法，传入一个待排序的数组，以及待排序数组的起始位置和终止位置，得到划分基准的位置，再分别对划分基准的左侧和右侧进行排序。

2.7 线性时间搜索算法设计

```C++
BaseStation select(vector<BaseStation>a, int p, int r, int k){
    printf("%d\t", recursionDepth);
    // 查找区间小，直接排序更快
    if(r - p < 20){
        quickSort(a, p, r);
        return a[p+k-1];
```

```
    }
    // 找到各个区间的中位数，并放到数组的最前面
    for(int i = 0; i <= (r-p-4)/5; i++){
        int s = p + 5 * i;
        int t = s + 4;
        quickSort(a, s, t);
        swapBaseStations(a, p+i, s+2);
    }
    // 选择中位数的中位数作为划分基准 x
    BaseStation x = select(a, p, p+(r-p-4)/5, (r-p+6)/10);
    // 根据 x 将 a[p:r]分成三部分
    int i, j;
    i = partition(a, p, r, x);
    j = i - p + 1; //左子段长度
    if(k == j)
        return a[i];
    else{
        recursionDepth++;
        if(k < j){
            return select(a, p, i-1, k);
        }
        else {
            return select(a, i+1, r, k-j);
        }
    }
}
```

1、如果数组中待搜索的长度足够小，那么就直接对这部分简单排序，找到第 k 小的元素

2、通过循环变量/指针 i，将数组 a[p:r]划分为长度为 5 的 m 个完整子序列，多出来的那一小部分不管。

3、对这些长度为 5 的完整子序列排序，找到每个子序列的中位数，并将这些中位数和数组的前面的元素交换，比如第 i 个完整子序列的中位数就和数组下标的第 i 处的元素交换。

4、找出这些中位数的中位数作为划分基准，这里直接递归调用线性搜索算法，找出 a[p:p+(r-p-4)/5]的第 (r-p+6)/10 小的元素。

5、根据该划分基准，将 a[p:r]划分为 3 部分：

(1) 左子段 a[p:i-1]，长度为 j=i-p+1（包括了划分基准）

(2) 划分基准 a[i]

(3) 右字段 a[i+1:r]，长度为 r-i

6、根据 k 与左子段的长度 j 比较，采用减治法，

(1) 如果 j==k，那么说明找到了第 k 小的元素，返回

(2) 如果 k<j，说明左子段长度比 k 大，要在左子段中找处第 k 小元素

(3) 如果 k>j，说明左子段长度比 k 小，要在右字段中找第 k-j 小的元素

7、为了得出算法的递归调用深度，我们在每次分隔数组的时候将递归调用深度+1，最终能得出递归调用的深度。

三、代码实现

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
#define STRAIGHT_SORT_LENGTH 20
using namespace std;

int recursionDepth = 0;

class BaseStation{
public:
    int ENODEBID;
    double K_DIST;
    BaseStation(){
        ENODEBID = -1;
        K_DIST = 0;
    }
```

```
};
/**
 * @brief 简单的数组内交换函数
 *
 * @param a 数组
 * @param i 待交换的元素下标
 * @param j 待交换的元素下标
 */
void swapBaseStations(vector<BaseStation> & a, int i, int j) {
    BaseStation temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

/**
 * @brief 根据快排修改的划分基准算法，可传入基准数据
 *
 * @param arr 待更新数组
 * @param low 数组低位
 * @param high 数组高位
 * @param x 划分基准
 * @return int 划分基准更新完的位置
 */
int partition(vector<BaseStation>& arr, int low, int high, const BaseStation& x
    = BaseStation()){
    // 在数组中找到划分基准
    for(int i = 0; i <= high;i++){
        if(x.ENODEBID == arr[i].ENODEBID){
            swapBaseStations(arr, i, high);
            break;
        }
    }
    // 将划分基准放到末尾
    BaseStation pivot = arr[high];
    int i = low -1;
    for (int j = low; j <= high; j++){
        if(arr[j].K_DIST < pivot.K_DIST){
            i++;
            swapBaseStations(arr, i, j);
        }
    }
    swapBaseStations(arr, i+1, high);
    return i + 1;
}
```

```
62.
63. /**
64.  * @brief 基于递归的简单快排
65.  *
66.  * @param arr 待排序的数组
67.  * @param low 待排序的区间低位
68.  * @param high 待排序的区间高位
69.  */
70. void quickSort(vector<BaseStation>& arr, int low, int high) {
71.     if (low < high) {
72.         int pivotIndex = partition(arr, low, high);
73.
74.         quickSort(arr, low, pivotIndex - 1);
75.         quickSort(arr, pivotIndex + 1, high);
76.     }
77. }
78.
79. /**
80.  * @brief 一分为三的线性时间选择，找第 k 小元素
81.  *
82.  * @param a 待查找的数组
83.  * @param p 查找的数组区间的低位
84.  * @param r 高位
85.  * @param k 找第几小，比如 k=1 就是找最小的
86.  * @return BaseStation 返回第 k 小的数据
87.  */
88. BaseStation select(vector<BaseStation>a, int p, int r, int k){
89.     // 查找区间小，直接排序更快
90.     if(r - p < STRAIGHT_SORT_LENGTH){
91.         quickSort(a, p, r);
92.         return a[p+k-1];
93.     }
94.     // 找到各个区间的中位数，并放到数组的最前面
95.     for(int i = 0; i <= (r-p-4)/5; i++){
96.         int s = p + 5 * i;
97.         int t = s + 4;
98.         quickSort(a, s, t);
99.         swapBaseStations(a, p+i, s+2);
100.        }
101.        // 选择中位数的中位数作为划分基准 x
102.        BaseStation x = select(a, p, p+(r-p-4)/5, (r-p+6)/10);
103.        // 根据 x 将 a[p:r]分成三部分
104.        int i, j;
105.        i = partition(a, p, r, x);
```

```
106.        j = i - p + 1; //左子段长度
107.        if(k == j)
108.            return a[i];
109.        else{
110.            recursionDepth++;
111.            if(k < j){
112.                return select(a, p, i-1, k);
113.            }
114.            else {
115.                return select(a, i+1, r, k-j);
116.            }
117.        }
118.    }
119.
120.    // 一分为二
121.    BaseStation select_2(vector<BaseStation>a, int p, int r, int k){
122.        recursionDepth++;
123.        // 查找区间小，直接排序更快
124.        if(r - p < STRAIGHT_SORT_LENGTH){
125.            quickSort(a, p, r);
126.            return a[p+k-1];
127.        }
128.        // 找到各个区间的中位数，并放到数组的最前面
129.        for(int i = 0; i <= (r-p-4)/5; i++){
130.            int s = p + 5 * i;
131.            int t = s + 4;
132.            quickSort(a, s, t);
133.            swapBaseStations(a, p+i, s+2);
134.        }
135.        // 选择中位数的中位数作为划分基准 x
136.        BaseStation x = select_2(a, p, p+(r-p-4)/5, (r-p+6)/10);
137.        // 根据 x 将 a[p:r]分成三部分
138.        int i, j;
139.        i = partition(a, p, r, x);
140.        j = i - p + 1; //左子段长度
141.
142.        if(k <= j)
143.            return select_2(a, p, i, k);
144.        else
145.            return select_2(a, i+1, r, k-j);
146.    }
147.
148.    int main(){
149.        vector<BaseStation> datas;
```

```
150.        BaseStation data;
151.        FILE * fp;
152.        fp = fopen("./02-1.csv", "r");
153.        while(fscanf(fp,"%d,%lf", &data.ENODEBID, &data.K_DIST) != EOF){
154.            datas.push_back(data);
155.        }
156.        fclose(fp);
157.
158.        printf("===============Following are divide into THREE: ===============
    \n");
159.        BaseStation result = select(datas, 0, datas.size()-1, 1);
160.        printf("The  smallest k-
    dist station is: %d,%.3lf\nAnd the recursion depth is : %d\n\n", result.ENODEBI
    D, result.K_DIST, recursionDepth);
161.
162.        recursionDepth = 0;
163.        result = select(datas, 0, datas.size()-1, 5);
164.        printf("The 5th smallest k-
    dist  station is: %d,%.3lf\nAnd the recursion depth is : %d\n\n", result.ENODEB
    ID, result.K_DIST, recursionDepth);
165.
166.        recursionDepth = 0;
167.        result = select(datas, 0, datas.size()-1, 50);
168.        printf("The 50th smallest k-
    dist  station is: %d,%.3lf\nAnd the recursion depth is : %d\n\n", result.ENODEB
    ID, result.K_DIST, recursionDepth);
169.
170.        recursionDepth = 0;
171.        result = select(datas, 0, datas.size()-1, datas.size());
172.        printf("The biggest k-
    dist station is: %d,%.3lf\nAnd the recursion depth is : %d\n\n", result.ENODEBI
    D, result.K_DIST, recursionDepth);
173.
174.
175.        printf("===============Following are divide into TWO: ===============\n
    ");
176.        recursionDepth = 0;
177.        result = select_2(datas, 0, datas.size()-1, 1);
178.        printf("The  smallest k-
    dist station is: %d,%.3lf\nAnd the recursion depth is : %d\n\n", result.ENODEBI
    D, result.K_DIST, recursionDepth);
179.
180.        recursionDepth = 0;
181.        result = select_2(datas, 0, datas.size()-1, 5);
```

```
182.        printf("The 5th smallest k-
    dist  station is: %d,%.3lf\nAnd the recursion depth is : %d\n\n", result.ENODEB
    ID, result.K_DIST, recursionDepth);
183.
184.        recursionDepth = 0;
185.        result = select_2(datas, 0, datas.size()-1, 50);
186.        printf("The 50th smallest k-
    dist  station is: %d,%.3lf\nAnd the recursion depth is : %d\n\n", result.ENODEB
    ID, result.K_DIST, recursionDepth);
187.
188.        recursionDepth = 0;
189.        result = select_2(datas, 0, datas.size()-1, datas.size());
190.        printf("The biggest k-
    dist station is: %d,%.3lf\nAnd the recursion depth is : %d\n\n", result.ENODEBI
    D, result.K_DIST, recursionDepth);
191.
192.        return 0;
193.    }
```

四、运行结果及分析

4.1 运行结果

4.1.1 STRAIGHT_SORT_LENGTH 为 20 时的运行结果

```
==============Following are divide into THREE: ==============
The  smallest k-dist station is: 568030,103.075
And the recursion depth is : 15

The 5th smallest k-dist  station is: 567883,126.096
And the recursion depth is : 15

The 50th smallest k-dist  station is: 568074,208.475
And the recursion depth is : 15

The biggest k-dist station is: 568313,2735.798
And the recursion depth is : 15
```

图 4-1 一分为三算法运行结果 1

```
==============Following are divide into TWO: ==============
The  smallest k-dist station is: 568030,103.075
And the recursion depth is : 39

The 5th smallest k-dist  station is: 567883,126.096
And the recursion depth is : 39

The 50th smallest k-dist  station is: 568074,208.475
And the recursion depth is : 39

The biggest k-dist station is: 568313,2735.798
And the recursion depth is : 37
```

图 4-2 一分为二算法运行结果 1

4.1.2 STRAIGHT_SORT_LENGTH 为 10 时的运行结果

```
PS E:\bupt-homework\algorithm\Chapter2__partition\chapter2> cd "e:\bupt-homework\algorithm\Chapter2_
_partition\chapter2\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
==============Following are divide into THREE: ===============
The  smallest k-dist station is: 568030,103.075
And the recursion depth is : 23

The 5th smallest k-dist  station is: 567883,126.096
And the recursion depth is : 23

The 50th smallest k-dist  station is: 568074,208.475
And the recursion depth is : 23

The biggest k-dist station is: 568313,2735.798
And the recursion depth is : 25
```

图 4-3 一分为三算法运行结果 2

```
==============Following are divide into TWO: ===============
The  smallest k-dist station is: 568030,103.075
And the recursion depth is : 61

The 5th smallest k-dist  station is: 567883,126.096
And the recursion depth is : 61

The 50th smallest k-dist  station is: 568074,208.475
And the recursion depth is : 61

The biggest k-dist station is: 568313,2735.798
And the recursion depth is : 59
```

图 4-4 一分为二算法运行结果 2

4.1.3 STRAIGHT_SORT_LENGTH 为 50 时的运行结果

```
PS E:\bupt-homework\algorithm\Chapter2__partition\chapter2> cd "e:\bupt-homework\algorithm\Chapter2_
_partition\chapter2\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
==============Following are divide into THREE: ===============
The  smallest k-dist station is: 568030,103.075
And the recursion depth is : 10

The 5th smallest k-dist  station is: 567883,126.096
And the recursion depth is : 10

The 50th smallest k-dist  station is: 568074,208.475
And the recursion depth is : 10

The biggest k-dist station is: 568313,2735.798
And the recursion depth is : 10
```

图 4-5 一分为三算法运行结果 3

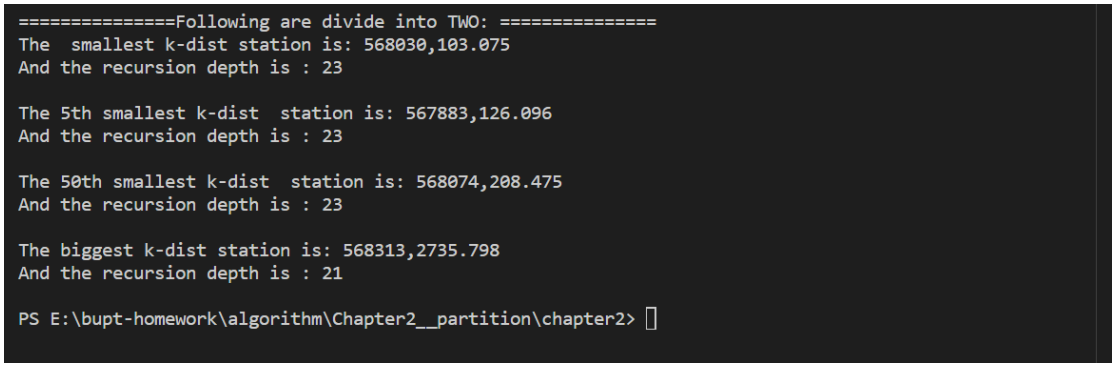

图 4-6 一分为二算法运行结果 3

4.2 运行结果分析

一分为二的快速选择算法和一分为三的快速选择算法的主要区别在于如何处理等于划分基准的元素。

一分为二的快速选择算法：

- 选择基准：选择各个子序列中位数的中位数作为划分基准。
- 划分：根据基准将数组分为两部分，左边的元素小于基准，右边的元素大于基准。
- 递归：根据第 k 小的元素在哪一部分，递归地在该部分进行查找。将划分基准放入下次递归查找中。

一分为三的快速选择算法：

- 选择基准：分别在每五个元素中选择中位数，然后将这些中位数中的中位数作为划分基准。
- 划分：根据基准将数组分为三部分，左边的元素小于基准，中间的元素等于基准，右边的元素大于基准。
- 递归：根据第 k 小的元素在哪一部分，递归地在该部分进行查找。

在一分为三的算法中，由于划分基准的选择更加精细，可以减少递归的深度。这

是因为在一分为三的划分中，等于基准的元素被放在了中间的部分，而不会被继续递归查找。相比之下，一分为二的算法中等于基准的元素可能会被包含在递归的两个分支中，导致递归深度的增加。