

实验6 最坏情况为线性时间的选择算法

姓名：邵宁录

学号：2018202195

目录

1. [问题描述](#)
2. [算法基本思路](#)
3. [算法复杂度分析](#)
4. [源码](#)
5. [运行结果截图](#)
6. [问题与总结](#)

一、问题描述

实现最坏情况为线性时间的选择第 k 大数算法，测试时选 $k=1000$ 。

输入：5000个无重复整数，范围是1-50000

输出：第 k 个数的值

二、算法基本思路

总体思路

- 将 n 个元素分成5个一组，最后一组有 $n \bmod 5$ 个元素
- 用插入排序对每一组排序，并取其中值
 - 本实验使用了STL里自带的插入排序 `__insertion_sort`
 - 没有使用新的数组存储 $n/5$ 个组的中值，而是将它们swap到了数组的最前方，再进行递归调用，从而实现了原址排序
- 递归调用此算法，寻找 $n/5$ 个中位数的中位数 x
- 用 x 作为 partition 选择的对象对数组进行划分
- 判断是否找到目标值，如果没有则继续递归搜索

```
if i == k then return k;

else if i < k then 找左区间的第i个最小元

else 找右区间第i-k个最小元
```

代码细节

2.1 读入接口

```
// 从外部读入的接口
void ReadData(const char *filename, vector<int> &array)
{
    string data;

    ifstream infile;
    infile.open(filename, ios::in);

    if (!infile.is_open())
    {
        cout << "[ERROR] Open file failed!" << endl;
        exit(1);
    }

    while (getline(infile, data))
    {
        array.push_back(stoi(data));
    }

    infile.close();
}
```

读入时有两个需要注意的细节：

- 文件未正常打开的异常处理
- 文件最后一行是否是空行的处理

第一个问题的解决非常简单，只需加上一个条件判断语句进行异常处理即可。第二个问题的解决稍微麻烦一些。原先我是用下面这行代码进行读入的：

```
while (!infile.eof())
{
    infile >> data;
    array.push_back(stoi(data));
}
```

但很遗憾，如果最后一行是空行，那么倒数第二行的数据将会被读入两次。因此需要改成用getline函数进行读取。

2.2 partition部分

partition部分分为俩函数：

- 一个是quicksort中的Partition函数
- 一个是经过修改的调用Partition的WorstCaseSelect函数

这部分函数很简单，就不在此赘述。

2.3 WorstCaseSelect部分

```
int WorstCaseSelect(vector<int> &A, int left, int right, int k)
{
    // 边界条件
    if (right - left + 1 < 140)
    {
        // 插入排序，稳定
        __insertion_sort(A.begin() + left, A.begin() + right + 1, cmp);
        return A[left + k - 1];
    }

    // 5个为一组进行插入排序，选出中位数，如果有俩中位数，取较小的
    int t = left;    // 从left开始计数的下标，将每组的中位数交换到这个下标的位置
    for (int i = left; i <= right; i += 5)
    {
        vector<int>::iterator start = next(A.begin(), i);
        // 不足5个就取剩下全部的
        vector<int>::iterator end = i + 5 <= right ? next(A.begin(), i + 5) :
next(A.begin(), right);
        __insertion_sort(start, end, cmp);
        swap(A[t], *(start + (end - start) / 2));
        t ++;
    }
    // 中位数的中位数
    int mid_of_mid = WorstCaseSelect(A, left, t - 1, floor((t - left - 1) /
2));
    // 找出当前选的中间位置
    int mid = WorstCasePartition(A, left, right, mid_of_mid);
    // 当前下标所排的位置
    int cur = mid - left + 1;
    // 三种情况分类讨论
    if (cur == k)    return A[mid];
    else if (cur > k)    return WorstCaseSelect(A, left, mid - 1, k);
    else    return WorstCaseSelect(A, mid + 1, right, k - cur);
}
```

该部分函数的重点是要弄清楚下标之间的关系，并且还得注意设置边界条件 $right - left + 1 < 140$ 时，直接调用插入排序算法返回结果。

三、算法复杂度分析

先说明结论，算法的时间复杂度为 $O(n)$

我们只讨论 n 较大的时候的情况。

在WorstCaseSelect中，对于找到的那个划分的数（中位数的中位数）记为 x ，不算其本身所在的那个组和最后不足5个元素的那个组，大于 x 的数至少有 $3 \left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil \right) - 2 \geq \frac{3n}{10} - 6$ 个。

同理，至少有 $\frac{3n}{10} - 6$ 小于 x 。

因此，在最坏情况下，WorstCaseSelect递归调用最多作用于 $\frac{7n}{10} + 6$ 个元素。

于是乎，可以得到如下的递归式： $T(n) = T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 6\right) + O(n)$

求解方法可以按照算法导论上的指导过程进行，最后得出该算法的时间复杂度为 $O(n)$

四、源码

```
/*
 * @Description: 算法导论第9章选择算法 (worst case select) 实现
 * @Author: rainym00d
 * @Github: https://github.com/rainym00d
 * @Date: 2020-10-21 11:17:45
 * @LastEditors: rainym00d
 * @LastEditTime: 2020-10-22 19:18:36
 */
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <algorithm>

using namespace std;

// 从外部读入的接口
void ReadData(const char *filename, vector<int> &array)
{
    string data;

    ifstream infile;
    infile.open(filename, ios::in);

    if (!infile.is_open())
    {
        cout << "[ERROR] Open file failed!" << endl;
        exit(1);
    }

    while (getline(infile, data))
    {
```

```

        array.push_back(stoi(data));
    }
    infile.close();
}

bool cmp(int a, int b)
{
    return a < b;
}

int Partition(vector<int> &A, int left, int right)
{
    int x = A[right];
    int i = left;
    for (int j = left; j < right; j++)
    {
        if (A[j] <= x)
        {
            swap(A[i++], A[j]);
        }
    }
    swap(A[i], A[right]);

    return i;
}

int WorstCasePartition(vector<int> &A, int left, int right, int x)
{
    int index;
    for (int i = left; i <= right; i++)
    {
        if (A[i] == x)
        {
            index = i;
            break;
        }
    }
    swap(A[index], A[right]);
    return Partition(A, left, right);
}

int WorstCaseSelect(vector<int> &A, int left, int right, int k)
{
    // 边界条件
    if (right - left + 1 < 140)
    {
        // 插入排序, 稳定
    }
}

```

```

        __insertion_sort(A.begin() + left, A.begin() + right + 1, cmp);
        return A[left + k - 1];
    }

    // 5个为一组进行插入排序，选出中位数，如果有俩中位数，取较小的
    int t = left;    // 从left开始计数的下标，将每组的中位数交换到这个下标的位置
    for (int i = left; i <= right; i += 5)
    {
        vector<int>::iterator start = next(A.begin(), i);
        // 不足5个就取剩下全部的
        vector<int>::iterator end = i + 5 <= right ? next(A.begin(), i + 5) :
next(A.begin(), right);
        __insertion_sort(start, end, cmp);
        swap(A[t], *(start + (end - start) / 2));
        t++;
    }
    // 中位数的中位数
    int mid_of_mid = WorstCaseSelect(A, left, t - 1, floor((t - left - 1) /
2));
    // 找出当前选的中间位置
    int mid = WorstCasePartition(A, left, right, mid_of_mid);
    // 当前下标所排的位置
    int cur = mid - left + 1;
    // 三种情况分类讨论
    if (cur == k)
    {
        return A[mid];
    }
    else if (cur > k)
    {
        return WorstCaseSelect(A, left, mid - 1, k);
    }
    else
    {
        return WorstCaseSelect(A, mid + 1, right, k - cur);
    }
}

int main(int argc, char const *argv[])
{
    vector<int> A;
    ReadData("无重复5千整数集范围是1-50000.txt", A);
    // ReadData("test.txt", A);
    cout << "最坏情况为线性时间的选择算法得出的第1000大的数为: " << WorstCaseSelect(A,
0, A.size() - 1, 4001) << endl;

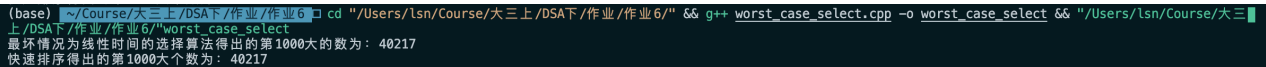
    sort(A.begin(), A.end(), cmp);
}

```

```
cout << "快速排序得出的第1000大的数为：" << A[4000] << endl;

return 0;
}
```

五、运行结果截图



```
(base) ~/Course/大三上/DSA下/作业/作业6 ❏ cd "/Users/lsn/Course/大三上/DSA下/作业/作业6/" && g++ worst_case_select.cpp -o worst_case_select && "/Users/lsn/Course/大三上/DSA下/作业/作业6/"worst_case_select
最坏情况为线性时间的选择算法得出的第1000大的数为：40217
快速排序得出的第1000大的数为：40217
```

六、问题与总结

本次算法实现难度中等，但总体来说还是能够实现的。遇到的问题主要出在下标的计算上容易产生混乱，以及在使用c++的STL时的一点小问题。

我自己在选择算法对比的实验中发现，就期望时间而言，randomized_select算法比最坏情况是线性时间寻找第k大元素的算法的时间复杂度要好很多。我个人猜测原因时它的常数项要小很多。

对比发现，randomized_select比另一种算法少了对于每组元素用插入排序的过程，也少了递归寻找中位数的中位数的过程，因此其 $O(n)$ 的系数要小很多

因此综合考虑的话，randomized_select算法效率更高。