

北京邮电大学软件学院

2017-2018 学年第一学期实验报告

课程名称: 算法分析与设计

项目名称: 实验四: 分治法

项目完成人:

姓名: 刘禾子 学号: 2017526019

指导教师: 李朝晖

日 期: 2017 年 12 月 04 日

一、 实验目的

- 1、 深刻理解并掌握分治的设计思想;
- 2、 提高应用分治法设计算法的技能;

二、 实验内容

基本题 1： 元素选择

给定序列中 n 个元素和一个整数 k , $1 \leq k \leq n$, 输出这 n 个元素中第 k 小元素的值及其位置。编写并调试程序（不排序）。

基本题 2： 最大子段和问题

给定由 n 个数组成的序列 (a_1, a_2, \dots, a_n) , 最大子段和问题要求该序列求解和为最大的连续子段。

基本题 3:

在一个按照东西和南北方向划分成规整街区的城市里, n 个居民点散乱地分布在不同的街区中。用 x 坐标表示东西向, 用 y 坐标表示南北向。各居民点的位置可以由坐标 (x, y) 表示。街区中任意 2 点 (x_1, y_1) 和 (x_2, y_2) 之间的距离可以用数值 $|x_1 - x_2| + |y_1 - y_2|$ 度量居民们希望在城市中选择建立邮局的最佳位置, 使 n 个居民点到邮局的距离总和最小。

编程任务:

给定 n 个居民点的位置, 计算 n 个居民点到邮局的距离总和的最小值。

提高题: 棋盘覆盖问题

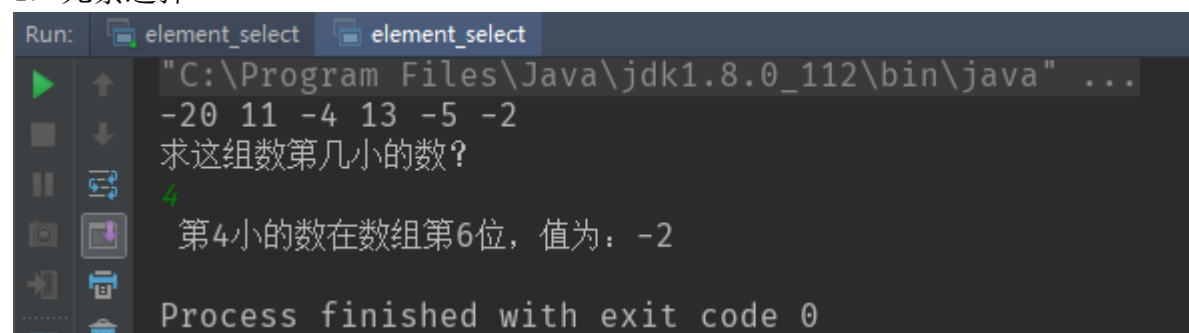
在一个 $2k \times 2k$ 个方格组成的棋盘中, 恰有一个方格与其它方格不同, 称该方格为一特殊方格, 且称该棋盘为一特殊棋盘。在棋盘覆盖问题中, 要用 4 种不同形态的 L 型骨牌覆盖给定的特殊棋盘上除特殊方格以外的所有方格, 且任何 2 个 L 型骨牌不得重叠覆盖。

三、 实验环境

IntelliJ IDEA Community Edition 2017.2.4

四、 实验结果

1. 元素选择



```
Run: element_select element_select
"C:\Program Files\Java\jdk1.8.0_112\bin\java" ...
-20 11 -4 13 -5 -2
求这组数第几小的数?
第4小的数在数组第6位, 值为: -2
Process finished with exit code 0
```

2. 最大子段和问题

```
Run: element_select sub_segment
"C:\Program Files\Java\jdk1.8.0_112\bin\java" ...
-20 11 -4 13 -5 -2
最大子段和为 20
从第2个元素到第4个元素:
Process finished with exit code 0
```

3. 邮局选址问题

```
Run: Post_locate
"C:\Program Files\Java\jdk1.8.0_112\bin\java" ...
请输入居民点个数:
5
请分别输入5个居民点的坐标:
1 2
2 2
1 3
3 -2
3 3
5个居民点到邮局的距离总和最小值为: 10.0
```

4. 棋盘覆盖问题

```
Run: Chess_Board
"C:\Program Files\Java\jdk1.8.0_112\bin\java" ...
输入棋盘大小k值 (2^k*2^k):
2
请输入特殊方格的坐标x,y:
0 1
2 0 3 3
2 2 1 3
4 1 1 5
4 4 5 5
Process finished with exit code 0
```

五、 附录

1. 元素选择

(1) 问题分析

给定一个无重复的无序数组，寻找这个数组中第 k 小的数，就是找到最小的前 k 个数，需要快速定位需要用到快速排序的核心思想（分治递归）；

(2) 设计方案

首先随机选定一个主元素 key ，经过一趟比较之后，比 key 小的数在 key 的左边，比 key 大的数在 key 的右边， key 的位置在之后的排序过程中不会改变，一趟遍历完成以后 key 的位置是 k ，即 key 就是第 k 小的元素，直接返回 k ，前 k 个数就是最小的

k 个数了;

(3) 算法分析

该算法实际上相当于快速排序的一部分，总的时间复杂度为 $O(n)$ 。

(4) 源代码

```
import java.util.Random;
import java.util.Scanner;

public class element_select {
    private static int[] nums={-20,11,-4,13,-5,-2};
    private static int[] nums_copy={-20,11,-4,13,-5,-2}; //用于对照的原数组
    private static Random rand=new Random();
    public static void main(String args[]) {
        for (int i=0;i<nums.length;i++) {
            System.out.print(nums[i]+" ");
        }
        System.out.println();
        System.out.println("求这组数第几小的数? ");
        Scanner sc=new Scanner(System.in);
        int k=sc.nextInt();
        System.out.println(RandomFind(0, nums.length-1, k));
    }

    private static int RandomFind( int p, int r, int k) {
        if (p==r) {
            for(int i=0;i<nums_copy.length;i++) {
                if (nums_copy[i]==nums[p])
                    System.out.print(" 第"+k+"小的数在数组第"+(i+1)+"位， 值为: ");
            }
            return nums[p];
        }

        int q=RandomPartition(p,r);
        if ((q+1)==k)
            return nums[q];
        else if ((q+1)<k)
            return RandomFind(q+1, r, k);
        else
            return RandomFind(p, q-1, k);
    }

    private static int RandomPartition( int p, int r) {
        int pivot=rand.nextInt(r-p+1)+p;
        int x=nums[pivot];
        int i=p, j=r;
        while (true) {
            while (nums[i]<x)
                i++;
            while(nums[j]>x)
                j--;
            if (i<j) {
                int temp=nums[i];
                nums[i]=nums[j];
                nums[j]=temp;
            }else
                return j;
        }
    }
}
```

2. 最大子段和问题

(1) 问题分析

给定由 n 个整数组成的序列 (a_1, a_2, \dots, a_n) ，最大子段和问题要求该序列形如 的最大值 $(1 \leq i \leq j \leq n)$ ，当序列中所有整数均为负整数时，其最大子段和为 0。例如，序列 $(-20, 11, -4, 13, -5, -2)$ 的最大子段和为：20

(2) 设计方案

划分：按照平衡子问题的原则，将序列 (a_1, a_2, \dots, a_n)

划分成长度相同的两个子序列 $(a_1, \dots, a_{\lfloor n/2 \rfloor})$ 和 $(a_{\lfloor n/2 \rfloor + 1}, \dots, a_n)$ ，则会出现以下三种情况：

- ① a_1, \dots, a_n 的最大子段和 = $a_1, \dots, a_{\lfloor n/2 \rfloor}$ 的最大子段和；
 - ② a_1, \dots, a_n 的最大子段和 = $a_{\lfloor n/2 \rfloor + 1}, \dots, a_n$ 的最大子段和；
 - ③ a_1, \dots, a_n 的最大子段和 = $\sum_{k=i}^j a_k$ ($k=i$ 到 j)，且 $1 \leq i \leq n/2, n/2+1 \leq j \leq n$ 。(2)
- 求解子问题：对于划分阶段的情况①和②可递归求解，情况③需要分别计算

$$s1 = \max \sum_{k=i}^{\lfloor n/2 \rfloor} a_k (1 \leq i \leq \lfloor n/2 \rfloor)$$

$$s2 = \max \sum_{k=\lfloor n/2 \rfloor + 1}^j a_k (\lfloor n/2 \rfloor + 1 \leq j \leq n)$$

则 $s1+s2$ 为情况③的最大子段和。

合并：比较在划分阶段的三种情况下的最大子段和，取三者之中的较大者为原问题的解。

(3) 算法分析

分析算法的时间性能，对应划分得到的情况①和②，需要分别递归求解，对应情况③，两个并列 for 循环的时间复杂性是 $O(n)$ ，所以，存在如下递推式：

$$T(n) = \begin{cases} 1 & n=1 \\ 2T(n/2) + n & n>1 \end{cases}$$

时间复杂度为 $O(n \log_2 n)$ 。

(4) 源代码

```
public class sub_segment {
    private static int begin=0;
    private static int end=0;
    public static void main(String args[]) {
        int[] a={-20,11,-4,13,-5,-2};
        for (int anA : a) {
            System.out.print(anA + " ");
        }
        System.out.println();
        System.out.println("最大子段和为 "+MaxSum(a, 0, 5));
        System.out.println("从第"+(begin+1)+"个元素到第"+(end+1)+"个元素: ");
    }
}
```

```

    }

    private static int MaxSum(int[] a, int left, int right) {
        int sum, midSum, leftSum, rightSum;
        int center, s1, s2, lefts, rights;
        if (left==right){
            sum=a[left]; //序列长度为1 直接求解
        }else {
            center=(left+right)/2;
            leftSum=MaxSum(a, left, center);
            rightSum=MaxSum(a, center+1, right);
            s1=0; lefts=0;
            for (int i=center; i>=left; i--) {
                lefts+=a[i];
                if (lefts>s1) {
                    s1=lefts;
                    begin=i;
                }
            }
            s2=0; rights=0;
            for (int j=center+1; j<right; j++) {
                rights+=a[j];
                if (rights>s2) {
                    s2=rights;
                    end=j;
                }
            }
            midSum=s1+s2;
            if (midSum<leftSum)
                sum=leftSum;
            else sum=midSum;
            if (sum<rightSum)
                sum=rightSum;
        }
        return sum;
    }
}

```

3. 邮局选址问题

(1) 问题分析

要使 n 个居民点到邮局的距离总和最小，用数值 $|x_1-x_2|+|y_1-y_2|$ 度量，即要使得邮局的 x 和 y 与每个居民点的坐标差值总和最小，实际上就是选取 x 和 y 的中位数 mid_x 和 mid_y

(2) 设计方案

分别用 $x[n]$ 和 $y[n]$ 两个数组存取 n 个居民点的横纵坐标，对 $x[n]$ ， $y[n]$ 分别进行从小到大的排序，若 n 是偶数，则中位数是有序数组最中间的两个数求和除以二 $(x[n/2-1]+x[n/2])/2$ ，若 n 是奇数，则中位数是有序数组最中间的一个数 $x[n/2]$ ， $y[n]$ 的中位数同理。最后求中位数与各个坐标的差值总和。

(3) 算法分析

该算法的时间复杂度实际上是花费在排序上，排序的时间复杂度越低，算法的性能越好，这里选用的是 java 中封装好的排序算法 `Arrays.sort()`，使用了快速排序和优化的合并排序，最好的时间复杂度是 $O(n)$ ，最差则退化到 $O(n^2)$ 。

(5) 源代码

```

import java.util.Arrays;
import java.util.Scanner;

public class Post_locate {

```

```

public static void main(String args[]){
    int n;
    System.out.println("请输入居民点个数: ");
    Scanner input=new Scanner(System.in);
    n=input.nextInt();
    int[] x=new int[n];
    int[] y=new int[n];
    System.out.println("请分别输入"+n+"个居民点的坐标: ");
    for(int i=0;i<n;i++){
        x[i]=input.nextInt();
        y[i]=input.nextInt();
    }
    input.close();
    Arrays.sort(x);
    Arrays.sort(y);
    double mid_x,mid_y;
    if (n%2==0){
        mid_x=(x[n/2-1]+x[n/2])/2;
        mid_y=(y[n/2-1]+y[n/2])/2;
    }else {
        mid_x=x[n/2];
        mid_y=y[n/2];
    }
    double sum=0;
    for (int i=0;i<n;i++){
        sum+=Math.abs(y[i]-mid_y);
        sum+=Math.abs(x[i]-mid_x);
    }
    System.out.println(n+"个居民点到邮局的距离总和最小值为: "+sum);
}
}

```

4. 棋盘覆盖问题

(1) 问题分析

分治法求解棋盘覆盖问题的技巧在于划分棋盘，使划分后的子棋盘的大小相同，并且每个子棋盘均包含一个特殊方格，从而将原问题分解为规模较小的棋盘覆盖问题。 $k>0$ 时，可将 $2k \times 2k$ 的棋盘划分为 4 个 $2^{k-1} \times 2^{k-1}$ 的子棋盘，这样划分后，由于原棋盘只有一个特殊方格，所以，这 4 个子棋盘中只有一个子棋盘包含该特殊方格，其余 3 个子棋盘中没有特殊方格。为了将这 3 个没有特殊方格的子棋盘转化为特殊棋盘，以便采用递归方法求解，可以用一个 L 型骨牌覆盖这 3 个较小棋盘的会合处，从而将原问题转化为 4 个较小规模的棋盘覆盖问题。递归地使用这种划分策略，直至将棋盘分割为 1×1 的子棋盘。

(2) 设计方案

- ① 棋盘：可以用一个二维数组 `board[size][size]` 表示一个棋盘，其中 $\text{size}=2^k$ 。为了在递归处理的过程中使用同一个棋盘，将数组 `board` 设为全局变量；
- ② 子棋盘：整个棋盘用二维数组 `board[size][size]` 表示，其中的子棋盘由棋盘左上角的下标 `tr`、`tc` 和棋盘大小 `s` 表示；
- ③ 特殊方格：用 `board[dr][dc]` 表示特殊方格，`dr` 和 `dc` 是该特殊方格在二维数组 `board` 中的下标；
- ④ L 型骨牌：一个 $2k \times 2k$ 的棋盘有一个特殊方格，所以，用到 L 型骨牌的个数为 $(4k-1)/3$ ，将所有 L 型骨牌从 1 开始连续编号，用一个全局变量 `t` 表示。

(3) 算法分析

设 $T(k)$ 是覆盖一个 $2k \times 2k$ 棋盘所需时间，从算法的划分策略可知， $T(k)$ 满足如下递推式：

$$T(k) = \begin{cases} O(1) & k=0 \\ 4T(k-1) + O(1) & k>0 \end{cases}$$

解此递推式可得 $T(k)=O(4k)$ 。

(4) 源代码

```
import java.util.Scanner;

public class Chess_Board {
    private static int t=0;    //L 型骨牌编号
    public static void main(String args[]){
        int k,dr,dc;
        System.out.println("输入棋盘大小 k 值 (2^k*2^k):");
        Scanner input =new Scanner(System.in);
        k=input.nextInt();
        int size= (int) Math.pow(2,k);
        int[][] board=new int[size][size];
        for (int i=0;i<size;i++){
            for (int j=0;j<size;j++){
                board[i][j]=0;
            }
        }
        System.out.println("请输入特殊方格的坐标 x,y:");
        dr=input.nextInt();dc=input.nextInt();
        ChessBoard(0,0,dr,dc,size,board);
        for (int[] aBoard : board) {
            for (int j = 0; j < board[0].length; j++) {
                System.out.print(aBoard[j] + " ");
            }
            System.out.println();
        }
    }

    private static void ChessBoard(int tr, int tc, int dr, int dc, int size,int[][] board) {
        int s,t1;    //t1 表示本次覆盖所用 L 型骨牌的编号
        if(size==1)
            return;
        t1=++t;
        s=size/2;
        if (dr<tr+s&&dc<tc+s)    //特殊方格在左上角子棋盘中
            ChessBoard(tr, tc, dr, dc, s, board); //递归处理子棋盘
```



```

else{
    board[tr+s-1][tc+s-1]=t1;
    ChessBoard(tr, tc, tr+s-1, tc+s-1, s, board);
}
if (dr<tr+s&&dc>=tc+s)
    ChessBoard(tr, tc+s, dr, dc, s, board);
else {
    board[tr+s-1][tc+s]=t1;
    ChessBoard(tr, tc+s, tr+s-1, tc+s, s, board);
}
if (dr>=tr+s&&dc<tc+s) {
    ChessBoard(tr+s, tc, dr, dc, s, board);
}else {
    board[tr+s][tc+s-1]=t1;
    ChessBoard(tr+s, tc, tr+s, tc+s-1, s, board);
}
if (dr>=tr+s&&dc>=tc+s) {
    ChessBoard(tr+s, tc+s, dr, dc, s, board);
}else {
    board[tr+s][tc+s]=t1;
    ChessBoard(tr+s, tc+s, tr+s, tc+s, s, board);
}
}
}

```

5. 调试心得

经过本次实验，我深刻地理解了分治的核心思想，将一个难以直接解决的大问题划分成一些规模较小的子问题，分别求解各个子问题，再合并子问题的解得到原问题的解，同时在调试过程中温习了快速排序以及合并排序的算法思想。