***2019***



**算法设计与分析实验报告**



|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1707 |
| 学 号： | U201714786 |
| 姓 名： | 王占成 |
| 完成日期： | 2019.12.20 |

**目 录**

[1 完成情况 4](#_Toc27645536)

[2 poj 1753解题报告 4](#_Toc27645537)

[题目的分析 4](#_Toc27645538)

[算法的设计 4](#_Toc27645539)

[性能的分析 5](#_Toc27645540)

[运行结果的展示 5](#_Toc27645541)

[3 poj 3714解题报告 6](#_Toc27645542)

[题目的分析 6](#_Toc27645543)

[算法的设计 6](#_Toc27645544)

[性能的分析 7](#_Toc27645545)

[运行结果的展示 7](#_Toc27645546)

[4 poj 3233解题报告 7](#_Toc27645547)

[题目的分析 7](#_Toc27645548)

[算法的设计 7](#_Toc27645549)

[性能的分析 9](#_Toc27645550)

[运行结果的展示 9](#_Toc27645551)

[5 poj3269解题报告 9](#_Toc27645552)

[题目的分析 9](#_Toc27645553)

[算法的设计 9](#_Toc27645554)

[性能的分析 11](#_Toc27645555)

[运行结果的展示 11](#_Toc27645556)

[6 总结 11](#_Toc27645557)

[实验总结 11](#_Toc27645558)

[需要进一步完善的地方 12](#_Toc27645559)

[心得体会和建议 12](#_Toc27645560)

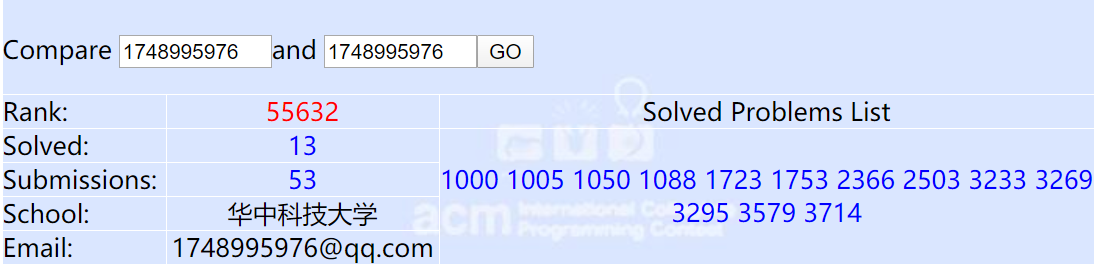
1 完成情况

1. 做的题目

所做的题目有poj1000、poj1005、poj1050、poj1088、poj1723、poj1753、poj2366、poj2503、poj3233、poj3269、poj3295、poj3579、poj3714。

1. 最后AC的题目

AC的题目有poj1000、poj1005、poj1050、poj1088、poj1723、poj1753、poj2366、poj2503、poj3233、poj3269、poj3295、poj3579、poj3714。



2 poj 1753解题报告

## 题目的分析

有4\*4的正方形，每个格子要么是黑色，要么是白色，当把一个格子的颜色改变(黑->白或者白->黑)时，其周围上下左右(如果存在的话)的格子的颜色也被反转，至少反转几个格子可以使 4\*4 的正方形变为纯白或者纯黑。在这里我们可以采用枚举+DFS的思想来解决这个问题。

## 算法的设计

棋盘一共有16个棋子，棋子翻转奇数次与翻转一次的结果相同，而棋子翻转偶数次与不翻转的结果是相同的。所以可以枚举翻1枚棋子，翻2枚棋子……翻16枚棋子的所有情况。不同的棋子进行排列组合，一共有2^16种情况。通过循环依次遍历即可。枚举某一次需要翻的棋子数，然后用DFS搜索翻转这个棋子数的所有情况。在DFS种，需要先选择要翻的第一个棋子，然后下一个要翻的棋子从已选择的这个棋子之后进行选择，一直到选完规定数量的棋子，然后翻转选择的所有棋子并判断所有棋子是否为同一种颜色，如果是，则不用再进行搜索了，当前所得的翻转棋子的数量便是最少的数量。

在这里关键代码便是DFS代码的编写，DFS代码如下：

void DFS(int num,int times,int who) //num表示总共要翻几个棋，times表示已经翻了几个，who表示上一次翻的是哪一个棋

{

if(ans < inf) return; //如果已经找到答案了，就不用往下了

if(times == num) //当翻棋的数量达到了，就进行判断

{

Flip(); //翻棋

if(Judge()) //判断全为同一种颜色

{

ans = num; //记录答案

}

else Flip(); //否则将棋盘恢复原状

return;

}

for(int i=who+1; i<=16; ++i) //从上一个翻的棋子的下一个开始选择，这样可以防止出现重复的情况

{

visit[i] = 1;//选中的标记为 1

DFS(num,times+1,i); //搜索翻下一个的情况，

visit[i] = 0; //清除标记

}

}

## 性能的分析

在实际的算法设计中，为了便于代码的编写，对于循环选择了从1开始而不是从0开始，所以空间复杂度是O(n)。而算法又是对于所有的情况进行枚举，所以时间复杂度为O(2**n**)。

## 运行结果的展示



3 poj 3714解题报告

## 题目的分析

有两组坐标，一组是 n 个发电站的坐标，一组是 n 个士兵的坐标，求某个士兵与离他最近的发电厂之间的距离。在这里可以采用分治算法，这道题目与平面中求两点之间的最短距离相似，只需要额外处理一下士兵与士兵和发电厂与发电厂之间的关系即可。

## 算法的设计

这里可以先分析一下平面分治最小点对问题。已知集合S中有n个点，使用分治法的思想就是将S进行拆分，分为2部分求最近点对。算法每次选择一条垂线L，将S拆分左右两部分为SL和SR，依次找出这两部分中的最小点对距离：δL和δR，记SL和SR中最小点对距离δ = min{δL，δR}。下面最关键的便是子问题合并的问题，若最短的点对分别处于垂线的两边该如何处理？以L为中心，δmin 为半径划分一个长带，最小点对还有可能存在于SL和SR的交界处，p点和q点分别位于L垂线L的两边并且处于这个长带中，p点和q点之间的距离才会小于δ。所以我们可以枚举长带中所有的点，计算两点之间的距离并且与上次得到的最小的距离进行比较，若小于，则取代上次所得的最小距离；否则，最小距离不做改变。在这里我们可以进行简单的优化，若长带中两点之间纵坐标之间的距离大于上次所得的最小值，那么就跳过。

这道题与平面分治点对之间的不同是士兵和士兵之间距离以及发电厂和发电厂之间的距离不能当作最小距离。因此我们可以加一个判断，若计算的两点属于同类事物，那么它们之间的距离就是无限大。

这道题的关键代码就是分治问题求解代码，我们可以将它放在一个函数当中。由于代码不算很大，所以这里给出完整的关键代码，代码如下：

double FindPair(Point \*points, int l,int r)

{// 求出最近点对记录，并将两点记录再a、b中

double distance; //记录集合points中最近两点距离

if (l == r)

return numeric\_limits<double>::max(); //若子集长度小于2，定义为最大距离，表示不可达

if (l + 1 == r)

{//若子集长度等于2，直接返回该两点的距离

distance = Distance(points[l], points[r]);

return distance;

}

//子集长度大于3，进行分治求解

Point \*rectangle = new Point[r - l + 1];

int m = (l + r) >> 1;

distance = min(FindPair(points,l, m), FindPair(points,m + 1, r));

//merge - 进行子集合解合并

//求解跨分割线并在δ×2δ区间内的最近点对

int k = 0;

for (int i = l; i <= r; i++)//取得中线2δ宽度的所有点对共k个

if (abs(points[i].x - points[m].x) <= distance)

rectangle[k++] = points[i];

sort(rectangle, rectangle + k, CompareY);// 以y排序矩形阵内的点集合

for (int i = 0; i < k; i++) {

for (int j = i+1; j < k; j++) {

if (rectangle[j].y - rectangle[i].y >= distance)

break;

if (rectangle[i].camp != rectangle[j].camp)

distance = min(distance, Distance(rectangle[i], rectangle[j]));

}

}

delete rectangle;

return distance;

}

## 性能的分析

这个问题所需要的空间是存储士兵以及发电厂的坐标，空间复杂度为O(n)。对点集S的点坐标进行升序快速排序，复杂度为O(nlogn)，接下来在分治过程中，由于采用的是分治算法，所以总的时间复杂度为O(nlogn)。

## 运行结果的展示



4 poj 3233解题报告

## 题目的分析

给出 n×n矩阵 A 和正整数， 求出 S=A+A2+A3+...+Ak，S的值对m求模，在这里可以采用分治算法进行分治求解。

## 算法的设计

在这里，我们使用类似于二分快速幂算法，可以列出一个表格，将题目转换为两种情况进行处理

|  |  |
| --- | --- |
| k mod 2 = 0 | k mod 2 ≠ 0 |
| s[k]=Ak/2∗s[k/2]+s[k/2] | s[k]=Ak/2∗s[k/2]+s[k/2]+Ak |
| Ak=Ak/2∗Ak/2 | Ak=Ak/2∗Ak/2∗A |

通过这个表格，我们可以使用一个递归，然后采用自底向上的方法不断地填充Ak以及s[k]的值，最后得到结果。只要理解了这个思想，代码的编写也比较简单。具体关键代码如下：

void calculate(int a){ // after calc, r=mat^a, rs=mat^1 + ... + mat^a

if(a == 1){

memcpy(mat\_sum,mat, sizeof(mat));

memcpy(mat\_rem,mat, sizeof(mat));

return;

}

int b = a >>1;

calculate(b);

// mat\_tmp = mat\_rem\*mat\_sum + mat\_sum

matmul(mat\_sum,mat\_rem,mat\_tmp);

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

mat\_tmp[i][j] = (mat\_sum[i][j]+mat\_tmp[i][j])%m;

}

}

// mat\_sum = mat\_tmp

// mat\_tmp = mat\_rem^2

memcpy(mat\_sum,mat\_tmp, sizeof(mat\_tmp));

matmul(mat\_rem,mat\_rem,mat\_tmp);

if(a%2 == 0){//s[k]=Ak/2∗s[k/2]+s[k/2]，Ak=Ak/2∗Ak/2

memcpy(mat\_rem,mat\_tmp, sizeof(mat\_tmp));

return;

}else{//s[k]=Ak/2∗s[k/2]+s[k/2]+Ak，Ak=Ak/2∗Ak/2∗A

matmul(mat\_tmp,mat,mat\_rem);

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

mat\_sum[i][j] = (mat\_sum[i][j] + mat\_rem[i][j])%m;

}

}

}

}

## 性能的分析

本题需要存储s[k]矩阵、Ak矩阵以及临时矩阵的坐标，所以空间复杂度为O(n)。这里我们使用了类似于二分快速幂的算法，时间复杂度为O(logn)。

## 运行结果的展示



5 poj3269解题报告

## 题目的分析

大草原上有好多牛在吃草，给出规定数目的牛的坐标，各个牛所在的位置是互不相邻的，即相邻的坐标不可能同时存在两头牛在吃草。现在想要建一个牛舍，问牛舍建在哪里，所有的牛到牛舍的距离之和最小？即曼哈顿问题求解。在这里我们可以采用中位数并且稍加判断的方法进行求解。

## 算法的设计

这道题可以采用中位数的方法进行求解。但是对于牛的个数又会出现两种情况，一种情况是偶数，另一种情况是奇数，所以我们可以分成两种情况进行处理。在这里，我们为了便于计算，我们在循环当中从1开始。首先分别对所有牛的X坐标和所有牛的Y坐标进行排序。

**当牛的个数n是奇数**

取(x[n/2+1]，y[n/2+1])为牛舍的坐标

若该点为牛所在的坐标，枚举它的上下左右四个方向上的点能求的最小的 d，然后统计这四个点哪些点或哪个点是所有牛回牛舍最近的点，期间也要同时计算距离之和。否则，所得出的点就是答案，只需再计算距离之和即可。

**当牛的个数n是偶数**

取(x[n/2]，y[n/2])和(x[n/2+1]，y[n/2+1])

 由曼哈顿距离的特性知：共有([n/2+1]−x[n/2]+1)\*(n/2+1]−y[n/2]+1)个点，且这个矩阵(包括边)中所有的点到给定的n个点的曼哈顿距离d全部相等。因此首先让方案的数量为矩阵中点的数量，并且记录一次最小距离，然后枚举每个点是否为给定的点，每发现一个点是矩阵中的点，方案数就减一。循环结束，就得出了答案。

关键代码处理如下：

sort(x+1,x+n+1);//对x排序

sort(y+1,y+n+1);//对y排序

if(n%2)//n为奇数时

{

int temp=(n/2)+1;

for(int i=1;i<=n;i++)

{

if(a[i].x==x[temp]&&a[i].y==y[temp])//若点为给出点

{

int Min=INT\_MAX;

for(int l=0;l<4;l++)//枚举四个方向

{

int xx=x[temp]+dx[l];

int yy=y[temp]+dy[l];

int sum=0;

for(i=1;i<=n;i++)//求最小的距离

sum+=abs(a[i].x-xx)+abs(a[i].y-yy);

if(sum<Min)

{

Min=sum;

plan=1;

}

else if (sum==Min)

plan++;

}

printf("%d %d\n",Min,plan);

return 0;

}

else//若点不为给出点

{

minn+=abs(a[i].x-x[temp])+abs(a[i].y-y[temp]);//记录最小距离

plan=1;//方案数为1

}

}

printf("%d %d\n",minn,plan);

}

else//n为偶数时

{

int temp1=n/2,temp2=n/2+1;

plan=(x[temp2]-x[temp1]+1)\*(y[temp2]-y[temp1]+1);//令方案数等于点的个数

for(int i=1;i<=n;i++)

{

minn+=abs(a[i].x-x[temp1])+abs(a[i].y-y[temp1]);//记录最小距离

int x0=a[i].x,y0=a[i].y;

if (x[temp1]<=x0&&x0<=x[temp2]&&y[temp1]<=y0&&y0<=y[temp2])//有牛在该矩阵内(包括边)，方案数减一

plan--;

}

printf("%d %d\n",minn,plan);

}

## 性能的分析

这里需要对所有牛的位置进行存储，空间复杂度为O(n)。在这里排序使用的是快速排序，快速排序的时间复杂度为O(nlogn)。对于牛舍坐标的求解，时间复杂度为O(n)。所以总的时间复杂度为O(n)。

## 运行结果的展示



6 总结

## 实验总结

分治算法和动态规划是算法的重点。而分治算法的重点就是如何找到递归停止的条件以及递归返回之后需要传递给上一层递归的数据是什么。写递归时，千万不要忙于敲代码，要留给思考的时间多一点。

DFS的巧妙之处实在有很多，棋子的翻转便是很好的DFS的实现。但是一开始没有想到DFS而是想到暴力枚举。对于DFS和BFS理解还是有很多欠缺，今后需要多加实践。

## 需要进一步完善的地方

许多算法还有需要完善的地方，譬如分治算法子问题合并的时候可以考虑鸽舍原理进行优化。对动态规划题目的掌握程度还不够熟练，需要多加练习。

## 心得体会和建议

做题的时候首先要学会分析题目，将题目中的关键字以及解题点分析出来。然后根据所学的知识将这个问题的数学模型建立起来，这样问题的求解就会变得简单起来。其实解题最关键的一步便是如何灵活地将题目与算法关联起来，并且建立起二者之间的数学模型。

我对算法的掌握程度比较差，有些思想不能够真正地理解。只有多看书，多敲代码才能让我对一个算法了解的比较透彻。通过算法实验，让我找到了学习算法的方法，同时也让我理解了算法的难度和其乐趣所在。

|  |
| --- |
| 一、原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字: 王占成** |

|  |
| --- |
| 二、对课程设计的学术评语（教师填写） |
|  |
| 三、对课程设计的评分（教师填写） |
| |  |  |  |  | | --- | --- | --- | --- | | 评分项目  （分值） | 报告撰写  （50分） | 课设过程  （50分） | 最终评定  （100分） | | 得分 |  |  |  | |
| **指导教师签字:** |