

《算法设计与分析》

实 验 总 结 报 告



专业班级：

CS1706

号：

名：

U201714762

学

姓

梁一飞

指导教师：

完成日期：

韩建军

2019.12.19

计算机科学与技术学院

****

**目 录**

[1 实验总结 1](#_Toc27690924)

[1.1已做的题目 1](#_Toc27690925)

[1.2 AC的题目 1](#_Toc27690926)

[1.3已做但未AC的题目 1](#_Toc27690927)

[1.4收获 2](#_Toc27690928)

[2 实验1:POJ1753 3](#_Toc27690929)

[2.1实验题目 3](#_Toc27690930)

[2.2设计思路 3](#_Toc27690931)

[2.3程序源代码 3](#_Toc27690932)

[2.4运行演示 4](#_Toc27690933)

[3 实验2:POJ2366 5](#_Toc27690934)

[3.1实验题目 5](#_Toc27690935)

[3.2设计思路 5](#_Toc27690936)

[3.3程序源代码 5](#_Toc27690937)

[3.4运行演示 6](#_Toc27690938)

[4 实验3:POJ3233 7](#_Toc27690939)

[4.1实验题目 7](#_Toc27690940)

[4.2设计思路 7](#_Toc27690941)

[4.3程序源代码 7](#_Toc27690942)

[4.4运行演示 9](#_Toc27690943)

[5 实验4:POJ3269 9](#_Toc27690944)

[5.1实验题目 9](#_Toc27690945)

[5.2设计思路 9](#_Toc27690946)

[5.3程序源代码 9](#_Toc27690947)

[5.4运行演示 11](#_Toc27690948)

[附录 实验5-8 AC截图 12](#_Toc27690949)

[附录.1实验5:POJ2503 12](#_Toc27690950)

[附录.2实验6:POJ3714 12](#_Toc27690951)

[附录.3实验7:POJ2506 14](#_Toc27690952)

[附录.4实验8:POJ1723 15](#_Toc27690953)

# 1 实验总结

## 1.1已做的题目

1000

1005

1753

3295

2366

2503

3714

3233

2506

1723

3269

3579

共计：12题

## 1.2 AC的题目

1000

1005

1753

3295

2366

2503

3714

3233

2506

1723

3269

3579

共计：12题

## 1.3已做但未AC的题目

无

共计：0题

## 1.4收获

通过本次算法实验，我学习以及掌握了许多方法，如二分法，分治法，中位数，动态规划等。在做POJ1753时，我掌握了暴力枚举解题的办法，当然，前提是问题规模不复杂，如POJ1753总共就65536种情况，可以很轻松的用dfs解决。在做POJ2366时，简单的使用了hash表，采取用空间换时间的解题策略，也是在问题规模不大时解决问题。在做POJ3233时，为了简化矩阵运算，学习了矩阵快速幂算法，以及分析运算的规律，进一步简化计算，使矩阵运算时间大大减少。在做POJ3269时，一开始毫无头绪，但是当明白牛和牛棚可以简化成坐标系里的点时，问题就成了简单的数学问题，根据曼哈顿距离公式，求得最佳牛棚建设的点，但是这题也有一个坑，就是牛棚的点可能与牛的点重合，此时则需要在这一点附近选一个最佳点，所以在实际编码过程中，一定要多考虑一些因素。

# 2 实验1:POJ1753

## 2.1实验题目

Eg：POJ1753：**Flip Game**

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，问至少反转几 个格子可以使 4\*4 的正方形变为纯白或者纯黑？

## 2.2设计思路

每个棋子最多翻转一次，翻转两次则相当于没有翻转，故只需枚举出1~16所有情况即2^16=65536种即可。

时间复杂度为。

## 2.3程序源代码

代码段1 POJ1753

#include<iostream>

using namespace std;

bool map[16][16] = { false };

bool flag = false;

int step;

int a[6] = { -1,1,0,0,0 }, b[6] = { 0,0,-1,1,0 };   //左右上中下

bool judge()      //判断是否全部同色

{

    for (int i = 1; i <= 4; i++)     //五个位置，除开自身，只要判断四个

        for (int j = 1; j <= 4; j++)

        {

            if (map[i][j] != map[1][1])

                return false;

        }

    return true;

}

void flip(int row, int col)     //翻转棋子

{

    for (int i = 0; i <= 4; i++)

        map[row + a[i]][col + b[i]] = !map[row + a[i]][col + b[i]];

    return;

}

void dfs(int row, int col, int deep)

{

    if (deep == step)       //判断是否从map[1][1]翻转到当前位置

    {

        flag = judge();

        return;

    }

    if (flag || row == 5) return; // dfs的走向是从左到右，从上到下，故当row==5的时候已经实现

    flip(row, col);

    if (col < 4)                  //将翻转后的全部遍历一遍，看是否符合

        dfs(row, col + 1, deep + 1);

    else

        dfs(row + 1, 1, deep + 1);

    flip(row, col);              //若前面的循环内检验出不符合条件，则翻回来

    if (col < 4)

        dfs(row, col + 1, deep);

    else

        dfs(row + 1, 1, deep);

    return;

}

int main()

{

    char fir;

    for (int i = 1; i <= 4; i++)

        for (int j = 1; j <= 4; j++)

        {

            cin >> fir;

            if (fir == 'b') map[i][j] = true;    //黑棋标记为true，白旗为false

        }

    for (step = 0; step <= 16; step++)     //最多需要走4\*4（16）步

    {

        dfs(1, 1, 0);

        if (flag) break;

    }

    if (flag)

        cout << step << endl;

    else

        cout << "Impossible" << endl;

    return 0;

}

## 2.4运行演示



图1 POJ1753AC截图

# 3 实验2:POJ2366

## 3.1实验题目

POJ2366：**Sacrament of the sum**

已知两个给定的序列，一个升序排列，一个降序排列，在这两个序列中各找一个数，它们加 起来恰好等于 10000。

## 3.2设计思路

借助哈希表，将序列a种的所有数（-32678~32767）加上32768后存入一个大小为65536的map数组中，即令map[a]=1；然后令temp1=42768-b。查询map[temp1]是否等于1，若等于1则找到这两个数，否则找不到

时间复杂度：O(n)

## 3.3程序源代码

代码段2 POJ2366

#include<stdio.h>

#define N 65536

#define Ni 50000

#define res 42768

int map[N];

int a[Ni];

int b[Ni];

int i;

int j;

int k;

bool hash(int \*a, int \*b){

    int f;

    int temp1;

    int temp2;

    for (f = 0; f < i; f++) {

        temp1 = a[f]+32768;

        map[temp1] = 1;

    }//将a中出现的数在map中标记为1

    for (f = 0; f < j; f++) {

        temp1 = b[f];

        temp2 = res - temp1;

        if (map[temp2] == 1)

            return true;

    }

    return false;

};//查询res-b中的数在map中是否有标记

int main() {

    for (i = 0; i < N; i++) {

        map[i] = 0;

    };

    scanf\_s("%d", &i);

    for (k = 0; k < i; k++) {

        scanf\_s("%d", a+k);

    };

    scanf\_s("%d", &j);

    for (k = 0; k < j; k++) {

        scanf\_s("%d", b+k);

    };

    if (hash(a, b))

        printf("YES");

    else

        printf("NO");

    return 0;

}

## 3.4运行演示



图2 POJ2366AC截图

# 4 实验3:POJ3233

## 4.1实验题目

POJ3233：**Matrix Power Series**

已知一个 n\*n 的矩阵 A，计算S = A + A 2 + A 3 + … + A k ， S 的值对 m 取模。

## 4.2设计思路

由于S[7]=A+(A+A^4)\*S(3)，S[6]=(1+A^3)\*S(3);可用矩阵快速幂简化计算

时间复杂度：O（lgn\*n^3）

## 4.3程序源代码

代码段3 POJ3233

#include<cstdio>

#include<cstring>

#define N 50

int n, k, M;

struct Matrix

{

    int f[N][N];

};

Matrix Add(Matrix U, Matrix V) //加

{

    Matrix S;

    memset(S.f, 0, sizeof(S.f));

    for (int i = 0; i < n; i++)

        for (int j = 0; j < n; j++)

            S.f[i][j] = (U.f[i][j] + V.f[i][j]) % M;

    return S;

}

Matrix Mul(Matrix U, Matrix V) //乘

{

    Matrix S;

    memset(S.f, 0, sizeof(S.f));

    for (int k = 0; k < n; k++)

        for (int i = 0; i < n; i++)

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

                S.f[k][i] = (U.f[k][j] \* V.f[j][i] + S.f[k][i]) % M;

    return S;

}

Matrix Pow(Matrix S, int k) //矩阵快速幂

{

    if (k == 0)

    {

        memset(S.f, 0, sizeof(S.f));

        for (int i = 0; i < n; i++)

            S.f[i][i] = 1;

        return S;

    }

    if (k == 1)

        return S;

    Matrix X = Pow(S, k / 2);

    if (k % 2)

        return Mul(Mul(X, X), S);

    else

        return Mul(X, X);

}

Matrix Cal(Matrix A, int k) //求解S(K)

{

    if (k == 1)

        return A;

    else

    {

        if (k % 2)

        {

            Matrix B = Add(A, Pow(A, (k + 1) / 2)); //S[7]=A+(A+A^4)\*S(3);

            return Add(A, Mul(B, Cal(A, k / 2)));

        }

        else

        {

            Matrix B = Add(Pow(A, 0), Pow(A, k / 2)); //S[6]=(1+A^3)\*S(3);

            return Mul(B, Cal(A, k / 2));

        }

    }

}

int main()

{

    while (scanf("%d%d%d", &n, &k, &M) == 3)

    {

        Matrix A;

        for (int i = 0; i < n; i++)

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

                scanf("%d", &A.f[i][j]);

        Matrix S = Cal(A, k);

        for (int i = 0; i < n; i++)

        {

            for (int j = 0; j < n - 1; j++)

                printf("%d ", S.f[i][j]);

            printf("%d\n", S.f[i][n - 1]);

        }

    }

    return 0;

}

## 4.4运行演示



图3 POJ3233AC截图

# 5 实验4:POJ3269

## 5.1实验题目

POJ3269：**Building A New Barn**

已知 N 头牛的位置，问牛舍建在哪里，牛到牛舍的距离和最小

## 5.2设计思路

两点曼哈顿距离公式：d=|x1−x2|+|y1−y2|，由于两点曼哈顿距离的特性，单独求 x 与单独求 y 互不影响，题目即为求 |x−x1|+|x−x2|+…+|x−xn| 的最小值，求 |y−y1|+|y−y2|+…+|y−yn| 的最小值，直接求两者中位数即可。对 x、y 各自进行排序比较：1.当n为奇数时，取（ x[n/2+1]，y[n/2+1] ）若该点为给出点，枚举它的上下左右四个方向上的点能求的最小的 d，然后统计当且仅当这4个点的方案数；若该点不为给出点，则直接记录最小距离，方案数为1。2.当n为偶数时，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（ x[n/2+1]−x[n/2]+1）\*（ y[n/2+1]−y[n/2]+1）个点，且它们到给定的n个点的曼哈顿距离和d相等。因此枚举每个点是否为给定的点，求一次最小距离，再先让方案数为点的个数，每次更新减小方案数即可

## 5.3程序源代码

代码段4 POJ3269

#include<iostream>

#include<cmath>

#include<algorithm>

#define MAX 0x3f3f3f3f

#define N 10001

using namespace std;

int dx[4] = { 0,1,0,-1 };//上右下左

int dy[4] = { 1,0,-1,0 };//上右下左

struct Node {

    int x;

    int y;

}a[N];

int x[N], y[N];

int minn, plan;

int main()

{

    int n;

    scanf\_s("%d", &n);

    for (int i = 1; i <= n; i++)

    {

        scanf\_s("%d%d", &a[i].x, &a[i].y);

        x[i] = a[i].x;

        y[i] = a[i].y;

    }

    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 = 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])//更新方案数

                plan--;

        }

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

    }

    return 0;

}

## 5.4运行演示



图4 POJ3269AC截图

# 附录 实验5-8 AC截图

## 附录.1实验5:POJ2503

代码段5 POJ2503

#include <iostream>

#include <cstring>

#include <string>

#include <cstdio>

#include <map>

#include <ctype.h>

#include <stdlib.h>

#include <algorithm>

using namespace std;

int main()

{

    map<string, string> g;

    char str[30], s1[15], s2[15];

    while (gets\_s(str) && strcmp(str, "") != 0)

    {

        sscanf(str, "%s %s", s1, s2);

        g[s2] = s1;

    }//键值对对应

    while (gets\_s(s2))

    {

        if (g.count(s2) == 0)

            printf("eh\n");

        else

            cout << g[s2] << endl;

    }

    return 0;

}



图5 POJ2503AC截图

## 附录.2实验6:POJ3714

代码段6 POJ2503

#include<cstdio>

#include<cstring>

#include<cmath>

#include<algorithm>

using namespace std;

const double MAX = 1e50;

int n;

double ab(double x)

{

    return x > 0 ? x : -x;

}

struct node

{

    double x, y;

    bool bel;//用于分组

}a[200010], temp[200010];

bool cx(const node &a, const node &b)

{

    return a.x < b.x;

}//比较x

bool cy(const node &a, const node &b)

{

    return a.y < b.y;

}//比较y

double min(double x, double y)

{

    return x < y ? x : y;

}//比较值

double dis(node p, node q)

{

    if (p.bel == q.bel) return MAX;//如果两点属于同一组，则距离为MAX

    return sqrt((p.x - q.x)\*(p.x - q.x) + (p.y - q.y)\*(p.y - q.y));

}//计算距离

double make(int l, int r)

{

    if (l == r) return MAX;//二分，只有1个点时。返回MAX，两个点时返回距离

    int m = (l + r) / 2, i, j, k, ll, rr, p, q, x, y, z, cnt = 0;

    double ans = min(make(l, m), make(m + 1, r));

    for (i = l; i <= r; i++)

        if (ab(a[i].x - a[m].x) <= ans)

            temp[++cnt] = a[i];

    sort(temp + 1, temp + cnt + 1, cy);//将求出的x按y排序

    for (i = 1; i <= cnt; i++)

        for (j = i + 1; j <= cnt; j++)

        {

            if (temp[j].y - temp[i].y > ans) break;

            ans = min(ans, dis(temp[i], temp[j]));

        }

    return ans;

}

int main()

{

    int i, j, k, T;

    scanf("%d", &T);

    while (T--)

    {

        scanf("%d", &n);

        for (i = 1; i <= n; i++)

        {

            scanf("%lf%lf", &a[i].x, &a[i].y);

            a[i].bel = 0;

        }

        for (i = 1; i <= n; i++)

        {

            scanf("%lf%lf", &a[i + n].x, &a[i + n].y);

            a[i + n].bel = 1;

        }

        sort(a + 1, a + 2 \* n + 1, cx);

        printf("%.3f\n", make(1, 2 \* n));

    }

}



图6 POJ3714AC截图

## 附录.3实验7:POJ2506

代码段7 POJ2506

#include <iostream>

#include <cstdio>

#include <string>

using namespace std;

string a[300];

string add(string a, string b)

{

    if (a.length() < b.length())

        swap(a, b);

    int i, j, up = 0, temp;

    for (i = a.length() - 1, j = b.length() - 1; i >= 0; i--, j--)

    {

        a[i] = a[i] + (j >= 0 ? b[j] - '0' : 0) + up;

        temp = (a[i] - '0') % 10;

        up = (a[i] - '0' - temp) / 10;

        a[i] = temp + '0';

        if (i == 0 && up != 0) a = "1" + a;//防止最高位进一无法储存；

    }

    return a;

}

int main()

{

    int n;

    a[0] = "1", a[1] = "1";

    for (int i = 2; i <= 250; ++i)

        a[i] = add(add(a[i - 2], a[i - 1]), a[i - 2]);//递推，f(n)=f(n-1)+2f(n-2)；

    while (cin >> n)

        cout << a[n] << endl;

    return 0;

}



图7 POJ2506AC截图

## 附录.4实验8:POJ1723

代码段8 POJ1723

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int maxn = 10010;

int n, mid, ans = 0;

int x[maxn], y[maxn];

//对于y轴，总步数为Σ|y[i]-mid|，对于x轴，总步数为Σ|x[i]-(mid+i)|=Σ|(x[i]-i)-mid|

int main()

{

    scanf("%d", &n);

    for (int i = 1; i <= n; i++) scanf("%d%d", &x[i], &y[i]);

    sort(x + 1, x + n + 1);

    for (int i = 1; i <= n; i++) x[i] -= i;//预处理x[i]-i

    sort(x + 1, x + n + 1);

    mid = x[(1 + n) / 2];

    for (int i = 1; i <= n; i++) ans += abs(mid - x[i]);

    sort(y + 1, y + n + 1);

    mid = y[(1 + n) / 2];

    for (int i = 1; i <= n; i++) ans += abs(mid - y[i]);

    printf("%d\n", ans);

    return 0;

}



图7 POJ1723AC截图