《算法分析与设计》

课后作业

作业编号 作业3

学 号

姓 名

专 业 软件工程

学 院 计算机与人工智能学院

二0二二年三月

第4章 分治法

总体要求：

1. 所有题目均需采用分治法实现。
2. 采用C/C++编程实现题目要求。
3. 分析算法的时间复杂度。
4. 设计具体的实例，分析并验证算法的执行过程及结果是否正确，写出分析及验证过程。

题目1：设A[1]=”\*”,A[2]为如下的形状：

|  |  |  |
| --- | --- | --- |
|  | \* |  |
| \* |  | \* |
|  | \* |  |

也就是A[2]为：

|  |  |  |
| --- | --- | --- |
|  | A[1] |  |
| A[1] |  | A[1] |
|  | A[1] |  |

对应的A[n]为：

|  |  |  |
| --- | --- | --- |
|  | A[n-1] |  |
| A[n-1 |  | A[n-1] |
|  | A[n-1] |  |

输入要求： 输入整数n，占1行。

输出要求：对应的图形。

样例输入：

3

样例输出：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | \* |  |  |  |  |
|  |  |  | \* |  | \* |  |  |  |
|  |  |  |  | \* |  |  |  |  |
|  | \* |  |  |  |  |  | \* |  |
| \* |  | \* |  |  |  | \* |  | \* |
|  | \* |  |  |  |  |  | \* |  |
|  |  |  |  | \* |  |  |  |  |
|  |  |  | \* |  | \* |  |  |  |
|  |  |  |  | \* |  |  |  |  |

题目2：某供电公司在某省各地修建了很多的变电站，这些变电站在出现故障时需要及时的维修，因此派遣距离这些变电站最近的工人去维修将会极大的减少维修的费用。

输入：输入第一行包含一个整数N (1 ≤ *N* ≤ 100000)，表示变电站和工人的数量。其后的2N行，每一行有两个整数X (0 ≤ *X* ≤ 1000000000)，Y (0 ≤ *Y* ≤ 1000000000)，前面的N行表示变电站的位置坐标，后面的N行表示工人的位置坐标。

输出：1行，也就是与这些变电站距离最近的工人所对应的最小距离，精确到小数点后3位。

样例输入：

4

0 0

0 1

1 0

1 1

2 2

2 3

3 2

3 3

样例输出：

1.414

评分标准：（每题100分）

1. 采用分治法实现。（10分）
2. C/C++程序要求（40分）
3. 程序编译、运行正确无误。（10分）
4. 程序书写规范，变量、函数定义符合规范。（10分）
5. 程序符合题目要求（20分）
6. 算法复杂度分析要求（20分）
7. 算法复杂度分析方法及过程正确。（15分）
8. 算法复杂度分析结果正确。（5分）
9. 设计具体实例，分析程序执行过程并验证结果是否正确（30分）
10. 设计符合题目要求的测试数据，并能考虑各种输入情况。要求不少于2组测试数据。（10分）
11. 正确分析每一组测试数据输入时程序的执行过程，包括程序执行过程中中间变量的变化情况以及程序的执行结果。（10分）
12. 正确调试程序，并给出程序执行过程中中间变量的值以及程序的运行结果。（10分）

提交内容： Word文档+程序代码一起打包成rar或zip文件。其中Word文档包含题目要求的所有内容（包括代码）。代码部分除包含在文档外，还应单独以源文件形式提交。如果包含工程文件，应与源文件一起打包。

提交截止时间：2022年4月23日20：00，每延后一天扣20分。

提交方式：电子稿，命名规则：学号\_姓名\_作业4.rar

邮件发送给：[hyhuang@home.swjtu.edu.cn](mailto:hyhuang@home.swjtu.edu.cn)

题目1：

1. 代码

#include<stdio.h>

#include<math.h>

char m[100][100];

void P(int *l*,int *x*,int *y*)

{

*//分治*

    int k = *l* / 3;

    int a, b, c, d, q, w, e, r;*//A[n-1]部分的坐标*

*//横坐标*

    a = *x* ;           *//b*

    b = c = *x* + k; *//a    d*

    d = *x* + 2 \* k;    *//c*

*//纵坐标*

    q = r = *y* + k;    *//w*

    w = *y*;         *//q    r*

    e = *y* + 2 \* k;    *//e*

    if (*l* == 3)*//分治到最小，即A[2]*

    {

*//a           b             c                 d*

        m[*x*][*y* + 1] = m[*x* + 1][*y*] = m[*x* + 1][*y* + 2] = m[*x*+2][*y*+1] = '\*';

        return;

    }

    else

    {

        P(k, a, q);

        P(k, b, w);

        P(k, c, e);

        P(k, d, r);

    }

    return;

}

int main()

{

    int n,l;*//l为格数*

    printf("请输入n:");

    scanf("%d", &n);

    l = pow(3, n - 1);*//边格数*

    for (int i = 1; i <= l; i++)

        for (int j = 1; j <= l; j++)

            m[i][j] = ' ';

    P(l,1,1);

    for (int i = 1; i <= l; i++)

    {

        for (int j = 1; j <= l; j++)

            printf("%c ",m[i][j]);

        printf("\n");

    }

    system("pause");

    return 0;

}

1. 时间复杂度

4^n

分析如下：

该算法采用递归的算法思想，即给定一个3^(n-1)\*3^(n-1)矩阵，初始全赋值为空，将其划分为9个3^(n-2)\*3^(n-2)的子矩阵，每次只递归上、左、右、下四个子矩阵，最终只有一个‘\*’的1\*1的矩阵。A[n]的时间复杂度记为T（n），当n=1时，T（n）=O(1),n>1时T(n)=4T(n-1)+O(1)

则有递推式 T(n)=4T(n-1)+O(1)

=16T(n-2)+5O(1)

=...

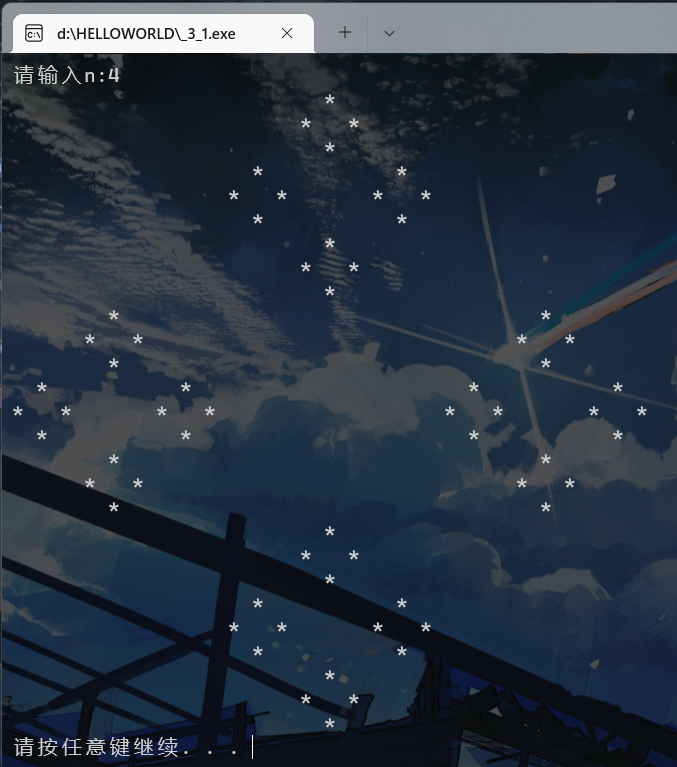
=4^(n-1)\*T(1)+((4^(n-1)-1)/3)O(1)

则该算法的时间复杂度为T（n）=O(4^n)。

1. 设计样例

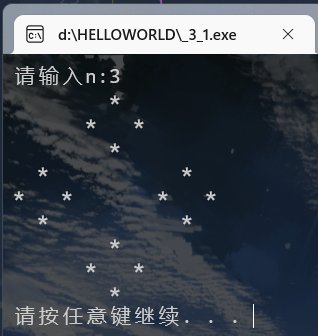
输入：4

输出：

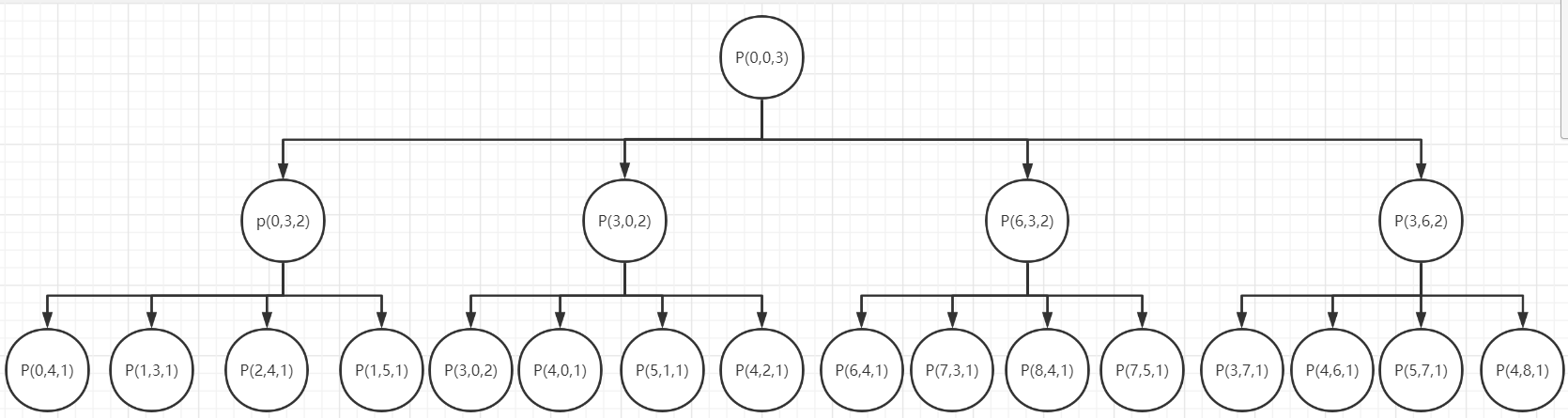


输入：3

输出：



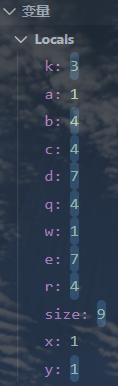
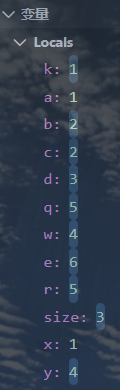
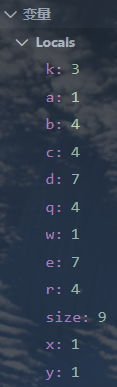
执行过程分析（以n=3为例）：



每个节点即代表执行一次P函数，全部执行完毕后，m数组中‘\*’坐标为：

m[0][4]，m[1][3]，m[2][4]，m[1][5]，m[1][5]，m[3][1]，m[4][0]，m[5][1]，m[4][2]，m[6][4]，m[7][3]，m[8][4]，m[7][5]，m[3][7]，m[4][6]，m[5][7]， m[4][8]

部分中间变量如图所示：



abcd和qwer分别为最小棋盘中‘\*’的横纵坐标，size为当前棋盘的大小，xy为当前棋盘的左上角的横纵坐标。

题目二：

1. 代码：

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<string.h>

#include<algorithm>

#include<iostream>

const double INF = 1e20;

using namespace std;

struct point

{*//点结构*

    int x;*//x坐标*

    int y;*//y坐标*

    int type;

};

double Distance(point *a*,point *b*)

{*//计算两点间距离*

    double s;

    if (*a*.type!=*b*.type)

        s=sqrt((*a*.x-*b*.x)\*(*a*.x-*b*.x)+(*a*.y-*b*.y)\*(*a*.y-*b*.y));

    else

        s = INF;

    return s;

}

bool cmp1(point *a*,point *b*)

{*//按y排序辅助函数*

    return *a*.y<*b*.y;

}

bool cmp2(point *a*,point *b*)

{*//按x排序辅助函数*

    return *a*.x<*b*.x;

}

double ClosestPoint(point *s*[],int *low*,int *high*)

{

    double d1,d2,d3,d;

    int mid,i,j,index;

    point P[10];

    if(*high*-*low*==1)

    {*//只有两个点的时候*

        return Distance(*s*[*low*],*s*[*high*]);

    }

    if(*high*-*low*==2)

    {*//当有三个点的时候*

*//两两计算，找出距离最近的两点*

        d1=Distance(*s*[*low*],*s*[*low*+1]);

        d2=Distance(*s*[*low*+1],*s*[*high*]);

        d3=Distance(*s*[*low*],*s*[*high*]);

        if((d1<d2)&&(d1<d3))

            return d1;

        else if (d2<d3)

            return d2;

        else

            return d3;

    }

*//三个结点以上的采用递归的办法*

    mid=(*low*+*high*)/2;

    d1=ClosestPoint(*s*,*low*,mid);*//左递归*

    d2=ClosestPoint(*s*,mid+1,*high*);*//右递归*

    if(d1<d2)*//比较左右递归所得的最近点对距离*

        d=d1;

    else

        d=d2;

    index=0;

    for(i=mid;(i>=*low*)&&((*s*[mid].x-*s*[i].x)<d);i--)*//记录[mid-d,mid]区域的点*

        P[index++]=*s*[i];

    for(i=mid+1;(i<=*high*)&&((*s*[i].x-*s*[mid].x)<d);i++)*//记录[mid,mid+d]区域的点*

        P[index++]=*s*[i];

    sort(P,P+index,cmp1);*//对给定区间所有元素进行排序*

    for(i=0;i<index;i++)

    {*//找出[mid-d,mid+d]中的最近点对*

        for(j=i+1;j<i+6&&j<index;j++)*//对y坐标排序时间复杂度为O（nlogn）*

        {

            if((P[j].y-P[i].y)>=d)

                break;

            else

            {

                d3=Distance(P[i],P[j]);

                if ((d3 < d) && (P[i].type != P[j].type))

                    d=d3;

            }

        }

     }

     return d;

}

int main()

{

    point p[100];

    int N;

    double minDist;

    scanf("%d", &N);

    if (N == 0)

        return 0;

    for (int i = 0; i < N; i++)

    {

        scanf("%d %d", &p[i].x, &p[i].y);

        p[i].type = 0;

    }

    for (int i = N; i < 2 \* N; i++)

    {

        scanf("%d %d", &p[i].x, &p[i].y);

        p[i].type = 1;

    }

    sort(p, p + 2 \* N, cmp2); *//对所有点在x轴上排序*

    minDist = ClosestPoint(p, 0, 2 \* N - 1);

    printf("%.3lf\n", minDist);

    system("pause");

    return 0;

}

1. 时间复杂度

算法采用分治的思想。在计算过程中，每次获得最短距离时，要记录下点对的位置信息。分治法的递归操作每次都是均等分，所以其递归树为logn层，递归函数中的操作均为线性。

在扫描每一区域的点时，for循环内层为常数阶，因此遍历n个点，时间复杂度不超过O（n）设时间复杂度为T（n）,则有递归公式为：

当n<4时，T（n）=O(1)

当n>=4时，T（n）=2T(n/2)+O(n)

解得T（n）=O(nlog2(n))。

1. 输入样例

样例输入：

4

0 0

0 1

1 0

1 1

2 2

2 3

3 2

3 3

输出结果：

执行过程分析：

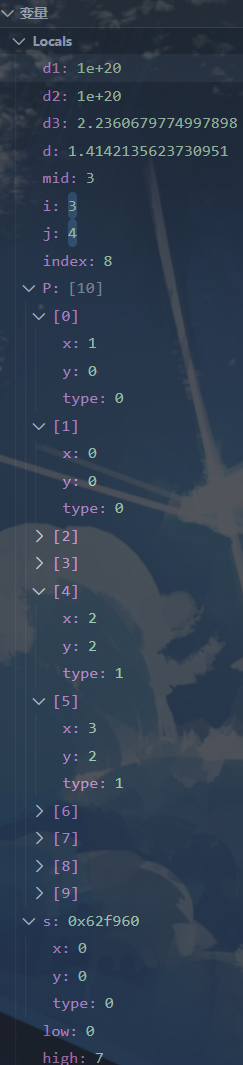
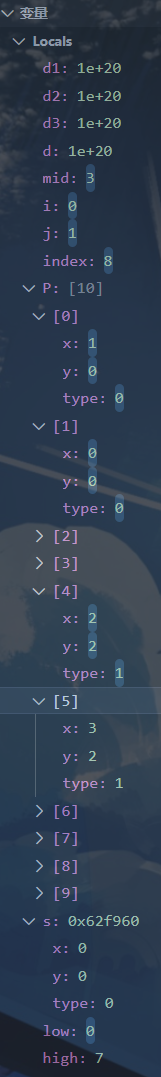
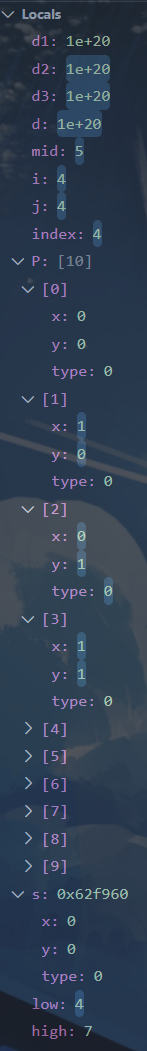
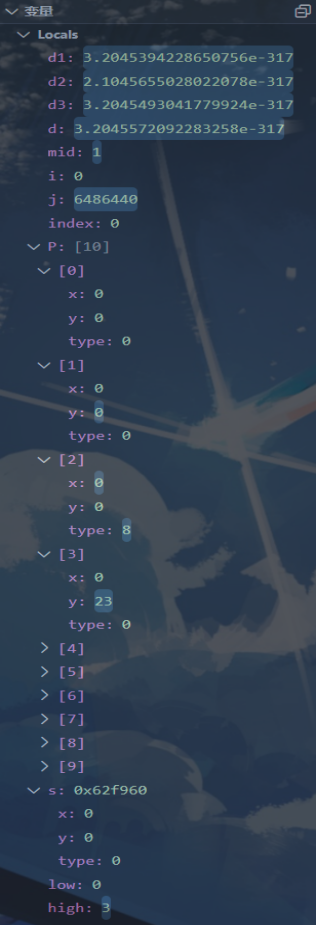
将平面上的点集S按x坐标排序后，若只有1个点，则返回；若有两个点，则直接计算两点距离为最近距离；若有三个点，则两两计算出距离，得到最近距离。

若大于三个点，则将其线性分割成大小大致相等的2个子集S1，S2。最近点对只会出现在以下三种情况中：①点对在子集S1中；②点对在子集S2中；③一个点在S1中，另一个点在S3中。

因此，递归地在S1，S2上解最近点对问题，分别得到S1，S2中的最小距离d1,d2，取d1,d2中的最小距离d和点对。再找出跨越S1，S2的最近点对：合并S1，S2找到点集中x坐标在区间[mid-d,mid+d]范围内的所有点。按y坐标不减排序，循环每个点，找它后面7个点的最小距离，与d比较，比d小则更新d。最后即求得最近点对距离。

需要注意的是，P数组里有两种类型的数据，type=0时代表变电站，type=1时代表工人，在计算距离时，首先需做判断，若两点是类型不同的两类数据，则可以计算距离d，否则将距离d置为INF=1e20，由此实现在更新最短距离时，不会更新同类型数据的距离，从而实现为电站分配距离最近的工人。

部分中间变量如图所示：



d1、d2分别为左右递归所得出的最小距离，P数组的前4个数据为电站，后4个为工人，所求两点距离需满足两数据类型不同，才可更新最短距离，最终d=1.414，输出minDist