DSA6讲解

姜城

2015/05/07

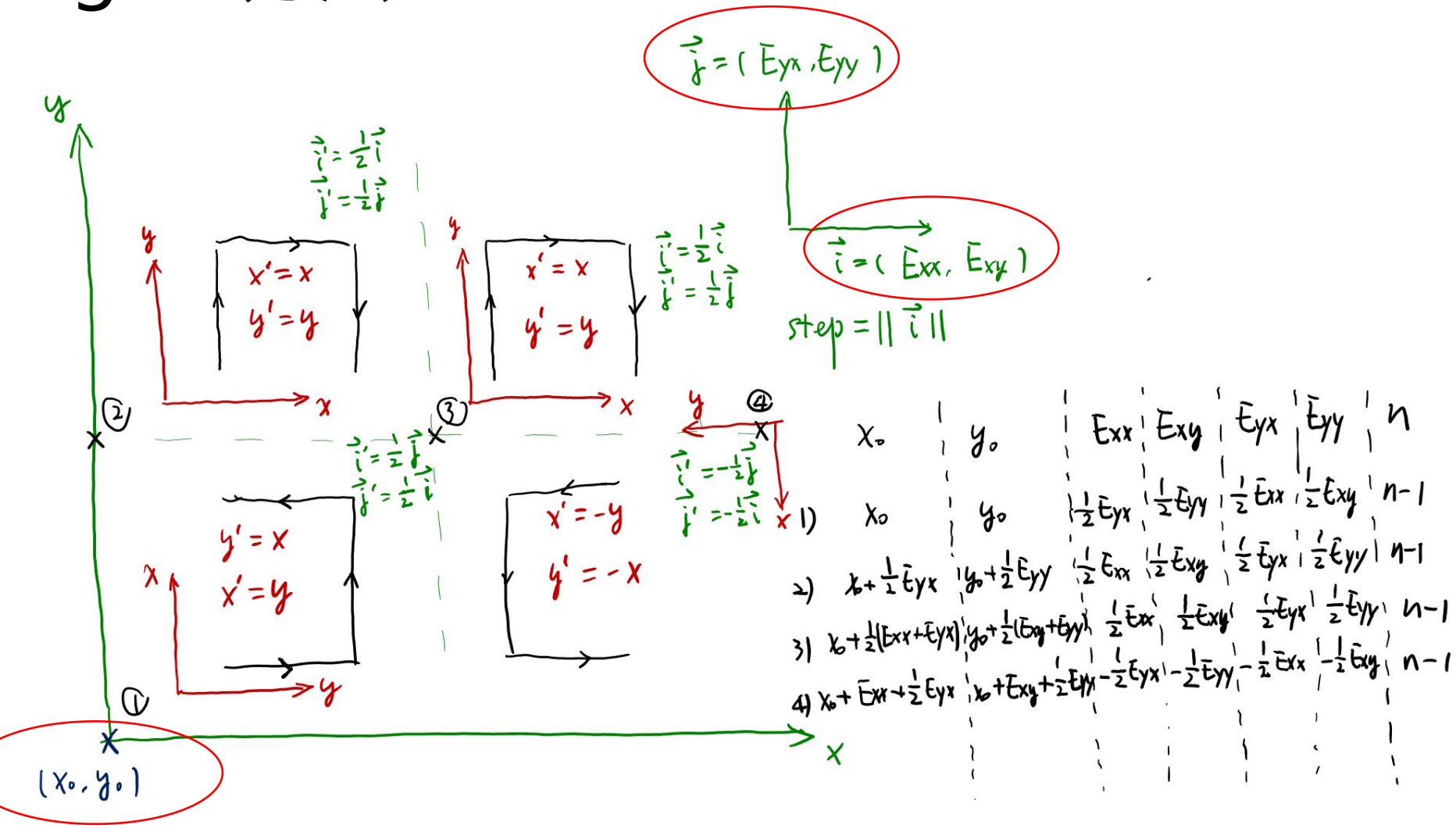
Pascal三角形

```python
def pascalTriangle_impl(numofrow):
    if numofrow==1:
        return [[1]]
    else:
        result=pascalTriangle_impl(numofrow-1)
        lst=result[-1]
        newlist=[1]
        for i in range(len(lst)-1):
            newlist.append(lst[i]+lst[i+1])
        newlist.append(1)
        result.append(newlist)
        return result

def pascalTriangle(numofrow):
    # get the list of rows
    result=pascalTriangle_impl(numofrow)
    # formatted printing
    formatedLst=[]
    for i in result:
        s=str(i).replace(',', '')
        formatedLst.append(s[1:-1])
    maxL=len(formatedLst[-1])
    for i in formatedLst:
        print(i.center(maxL))

if __name__=="__main__":
    pascalTriangle(10)
```

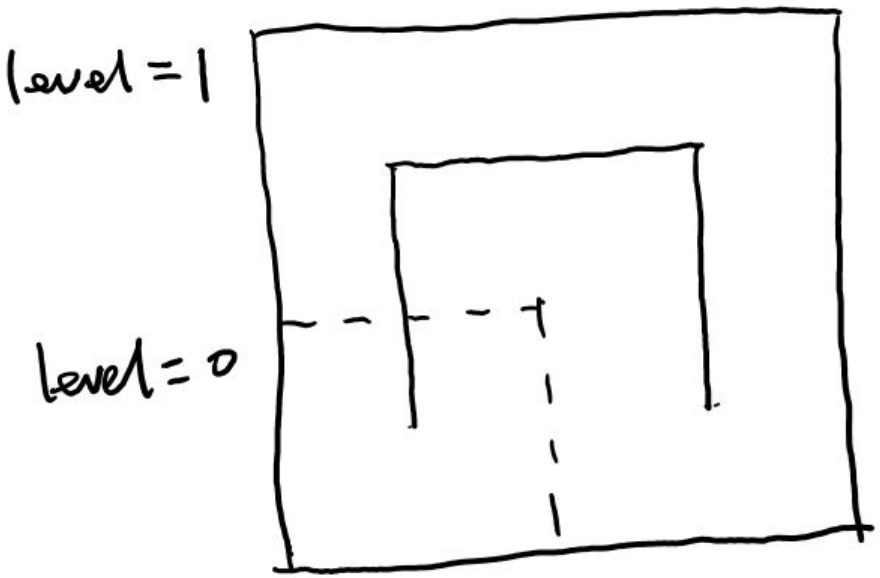
注意：每次递归运算一整行，而不要只运算一个元素。否则递归效率很低。

Hilbert曲线

goto方法



	x_0	y_0	E_{xx}	E_{xy}	E_{yx}	E_{yy}	n
1)	x_0	y_0	$\frac{1}{2}E_{yx}$	$\frac{1}{2}E_{yy}$	$\frac{1}{2}E_{xx}$	$\frac{1}{2}E_{xy}$	$n-1$
2)	$x_0+\frac{1}{2}E_{yx}$	$y_0+\frac{1}{2}E_{yy}$	$\frac{1}{2}E_{xx}$	$\frac{1}{2}E_{xy}$	$\frac{1}{2}E_{yx}$	$\frac{1}{2}E_{yy}$	$n-1$
3)	$x_0+\frac{1}{2}(E_{xx}+E_{yx})$	$y_0+\frac{1}{2}(E_{xy}+E_{yy})$	$\frac{1}{2}E_{xx}$	$\frac{1}{2}E_{xy}$	$\frac{1}{2}E_{yx}$	$\frac{1}{2}E_{yy}$	$n-1$
4)	$x_0+E_{xx}+\frac{1}{2}E_{yx}$	$x_0+E_{xy}+\frac{1}{2}E_{yy}$	$-\frac{1}{2}E_{yx}$	$-\frac{1}{2}E_{yy}$	$-\frac{1}{2}E_{xx}$	$-\frac{1}{2}E_{xy}$	$n-1$



在每一个基本情况下，移动(goto)到基本块的中点

(x_0, y_0)

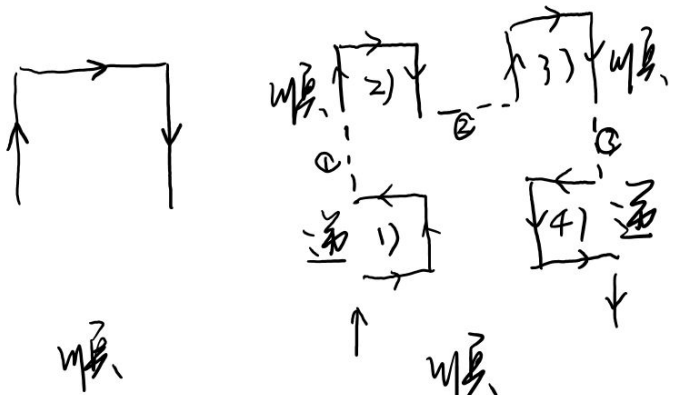
$$X = x_0 + \frac{1}{2}(\bar{E}xx + \bar{E}yx)$$
$$Y = y_0 + \frac{1}{2}(\bar{E}xy + \bar{E}yy)$$

$\Rightarrow$ goto(X, Y)

```python
def hilbert_curve_goto(t, x0, y0, Exx, Exy, Eyx, Eyy, level):
    '''
    方法1：使用goto的方式绘制希尔比特曲线
    :param t: 乌龟
    :param x0: 当前unit的坐标原点X坐标
    :param y0: 当前unit的坐标原点Y坐标
    :param Exx: 当前unit的X方向步长单位向量X坐标（unit边长）
    :param Exy: 当前unit的X方向步长单位向量Y坐标（unit边长）
    :param Eyx: 当前unit的Y方向步长单位向量X坐标（unit边长）
    :param Eyy: 当前unit的Y方向步长单位向量Y坐标（unit边长）
    :param level: 递归层次
    :return: None
    '''
    if level==0:
        X=x0+(Exx+Eyx)/2
        Y=y0+(Exy+Eyy)/2
        t.goto(X, Y)
    else:
        hilbert_curve_goto(t, x0,              y0,              Eyx/2,  Eyy/2,  Exx/2,  Exy/2,  level-1)
        hilbert_curve_goto(t, x0+Eyx/2,        y0+Eyy/2,        Exx/2,  Exy/2,  Eyx/2,  Eyy/2,  level-1)
        hilbert_curve_goto(t, x0+(Exx+Eyx)/2,  y0+(Exy+Eyy)/2,  Exx/2,  Exy/2,  Eyx/2,  Eyy/2,  level-1)
        hilbert_curve_goto(t, x0+Exx+Eyx/2,    y0+Exy+Eyy/2,    -Eyx/2, -Eyy/2, -Exx/2, -Exy/2, level-1)
```

将原点和XY轴的单位步长向量作为参数传入进行递归

forward方法



⇒ 1) 异 2) 同 3) 同 4) 异

连接 ① 如果逆时针，则 right(90)，否则 left(90)，再 forward()

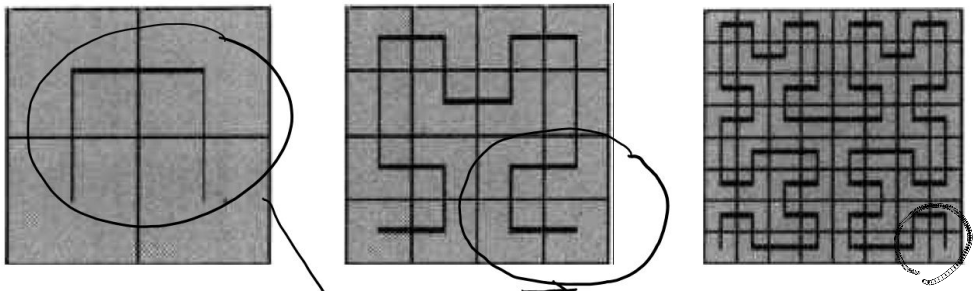
② 如果顺时针，则两次 left(90)，中间 forward()，否则 right(90)

③ 如果顺时针，则 forward()，再 right(90)，否则 left(90)

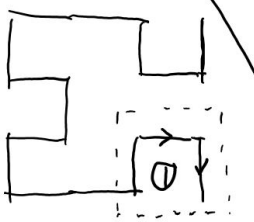
当前 parent

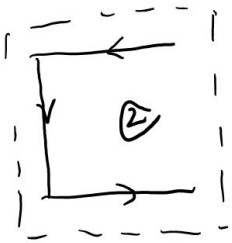
开始处：如果顺时针，right(90)，否则 left(90)

结束处：如果当前 parent 顺时针，right(90)，否则 left(90)



出口





① 其父结构为逆时针，出口向右
子结构出口向下，故左转90°. left(90)

② 其父结构为顺时针，出口向下
子结构为出口向右，故右转90°，right(90)

⇒ 父结构逆，则 left(90)
父结构顺，则 right(90)

```python
def hilbert_curve_forwad(t, level, sideLen):
    '''
    方法2：使用乌龟转向和前进（forward）的方式来绘制希尔伯特曲线
    :param t: 乌龟
    :param level: 递归层次
    :param sideLen: 外框长度
    :return: None
    '''
    s=T.Screen()
    sideLen=float(sideLen)
    # 北方向设为角度0方向，顺时针旋转为正
    s.mode("logo")
    # 将乌龟先移至左下角
    t.up()
    t.goto(-sideLen, -sideLen)
    t.down()
    t.speed(0)
    # 开始进行递归操作，在最高层，总是顺时针的，所以为True
    hilbert_curve_forward_impl(t, level, True, sideLen)
    s.exitonclick()
```

```python
def hilbert_curve_forward_impl(t, level, clockwise, step):
    '''
    hilbert_curve_forward()的递归具体实现
    总的思想是：四个部分的第一个和第四个与父对象的旋转方向相反，第二个和第三个与父对象的旋转方向相同
    :param t: 乌龟
    :param level: 当前递归层次
    :param clockwise: clockwise ~ True    anti-clockwise ~ False
    :param step: 绘制的步长
    :return:
    '''
    if level==0:
        for i in range(2):
            t.forward(step)
            t.right(90) if clockwise else t.left(90)
        t.forward(step)
        # 返回当前绘制的步长
        return step
```

基本情况，需要考虑旋转方向的问题

```
    else:
        # 父层和子层的入口方向会相差90度，在绘制子层之前，应该将方向从父层入口方向转为子层入口方向
        t.right(90) if clockwise else t.left(90)
        # 绘制第一个子部分
        # 该部分方向与父方向相反
        # 注意使用函数返回的步长值来进行forward()的调用
        newStep=hilbert_curve_forward_impl(t, level-1, not clockwise, step/2)
        # 绘制第一和第二部分的连接线
        # 旋转方向与第一子部分的方向(not clockwise)相关
        t.left(90) if (not clockwise) else t.right(90)
        t.forward(newStep)
        # 绘制第二部分
        # 该部分方向与父方向相同
        newStep=hilbert_curve_forward_impl(t, level-1, clockwise, step/2)
        # 绘制第二和第三部分的连接线
        # 旋转方向与第二部分的方向(clockwise)相关
        t.left(90) if clockwise else t.right(90)
        t.forward(newStep)
        t.left(90) if clockwise else t.right(90)
        # 绘制第三部分
        # 该部分方向与父方向相同
        newStep=hilbert_curve_forward_impl(t, level-1, clockwise, step/2)
        # 绘制第三部分和第四部分的连接线
        # 旋转方向与第三部分的方向(clockwise)相关
        t.forward(newStep)
        t.right(90) if clockwise else t.left(90)
        # 绘制第四部分
        # 该部分方向与父方向相反
        newStep=hilbert_curve_forward_impl(t, level-1, not clockwise, step/2)
        # 父层出口方向和子层出口方向相差90度，在子层绘制结束后，应该将方向由子层出口方向转为父层出口方向
        t.right(90) if clockwise else t.left(90)
        # return the step of this level
        return newStep
```

需要考虑父层(level)和子层(level-1)的初始方向问题