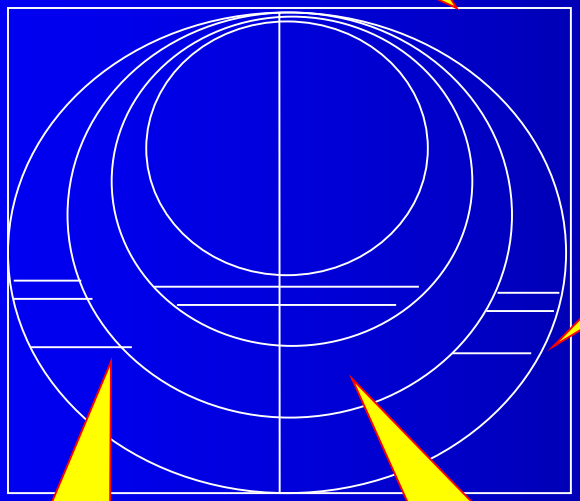
2.3.1.2边界标志算法

基本思想:

- 对多边形的每条边进行直线扫描转换，亦即对多边形边界所经过的象素打上标志。
- 使用一个布尔量 **inside** 来指示当前点是否在多边形内。**inside的初值为假**，当前访问的象素为打上边标志的象素点时，就把**inside取反**。未遇到打标志的象素，inside保持不变。inside为真时，说明该象素在多边形内，应把该象素置为填充颜色。

正方形中内切*n*个圆



inside在这条扫描线上的变化：假、真、假、真、假

inside为真，填色

inside为假，不填色

算法过程

```
void edgemark_fill(多边形定义 polydef, int color)
{
    对多边形polydef 每条边进行直线扫描转换;
    inside = FALSE;
    for (每条与多边形polydef相交的扫描线y )
        for (扫描线上每个象素x ) {
            if(象素 x 被打上边标志)
                inside = ! inside;
            if (inside! = FALSE)
                drawpixel (x, y, color);
            else drawpixel (x, y, background);
        }
}
```

扫描线算法与边界标志算法比较:

用软件实现时，**扫描线算法与边界标志算法**的执行速度几乎相同。但边界标志算法不必建立、维护边表以及对交点排序，**更适合硬件实现**，这时它的执行速度比有序边表算法快一至两个数量级。

2.3.2 区域填充算法

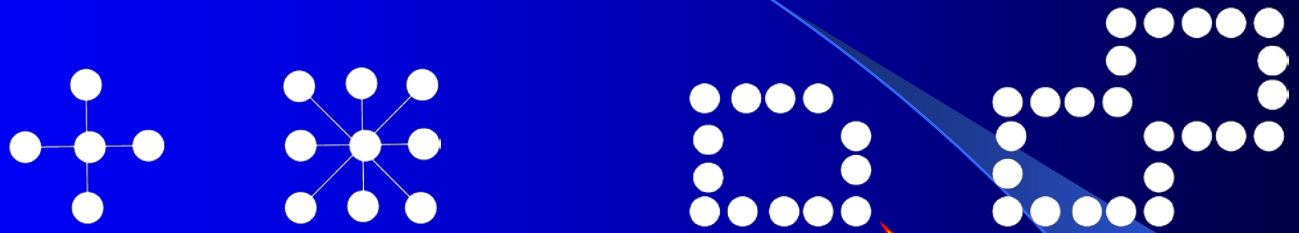
区域指已经表示成点阵形式的填充图形，它是象素的集合。

区域可采用**内点表示**，也可采用**边界表示**。在内点表示中，区域内的所有象素着同一颜色。在边界表示中，区域的边界点着同一颜色。

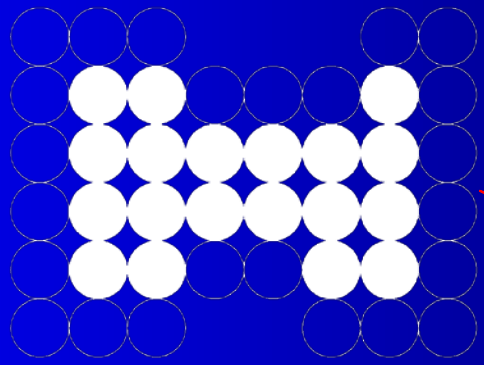
区域填充指先将区域的一点赋予指定的颜色，然后将该颜色扩展到整个区域的过程。

区域填充算法要求区域是**连通的**。而连通区域可分为**4向连通区域**和**8向连通区域**。4向连通区域指的是从区域上一点出发，可通过四个方向，即上、下、左、右移动的组合，在不越出区域的前提下，到达区域内的任意象素；8向连通区域指的是从区域内每一象素出发，可通过八个方向，即上、下、左、右、左上、右上、左下、右下这八个方向的移动的组合来到达。

4向连通区域和8向连通区域



四个方向运动　八个方向运动　四连通区域　八连通区域



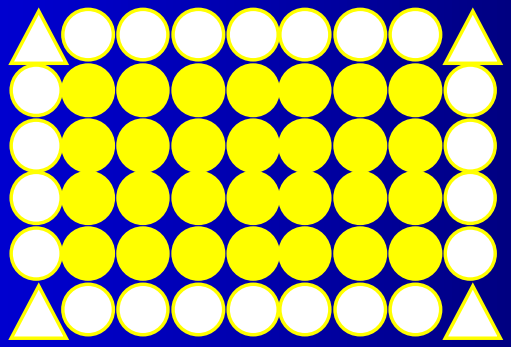
边界表示的区域

内点表示的区域

表示内点　表示边界点

作为4连通区域，其边界只要是8连通就可以了，而作为8连通区域，其边界必须是4连通的。

若将下图●表示的象素组成的区域看着4连通区域，则它的边界由标有●的象素组成；若将该区域看着8连通区域，则它的边界由标有●和▲的象素组成。



2.3.2.1 区域填充的递归算法

内点表示的4连通区域的递归填充算法:

```
void FloodFill4(int x, int y, int oldcolor, int newcolor)
{
    if(getpixel(x,y) = = oldcolor) {      // oldcolor是区域内点原色
        drawpixel(x, y, newcolor);
        FloodFill4(x, y+1, oldcolor, newcolor);
        FloodFill4(x, y-1, oldcolor, newcolor);
        FloodFill4(x-1, y, oldcolor, newcolor);
        FloodFill4(x+1, y, oldcolor, newcolor);
    }
}
```

边界表示的4连通区域的递归填充算法:

```
void BoundaryFill4(int x, int y, int boundarycolor, int newcolor)
{
    int color = getpixel(x, y);
    if(color!=newcolor && color!=boundarycolor) {
        drawpixel(x,y,newcolor);
        BoundaryFill4 (x, y+1, boundarycolor, newcolor);
        BoundaryFill4 (x, y-1, boundarycolor, newcolor);
        BoundaryFill4 (x-1, y, boundarycolor, newcolor);
        BoundaryFill4 (x+1, y, boundarycolor, newcolor);
    }
}
```

区域填充**递归算法**:

算法原理和程序都很简单，但多次递归：费时、费内存，效率不高。为了提高效率，可以采用**扫描线算法**。

2.3.2.2 区域填充的扫描线算法

算法步骤:

- 首先填充**种子点**所在扫描线上的、位于给定区域的**一个区段**
- 然后确定**与这一区段相连通的**上、下两条扫描线上位于给定区域内的区段，并依次保存下来。
- 反复这个过程，直到填充结束。

区域填充的扫描线算法可由下列4个步骤实现:

(1) **初始化**: 堆栈置空。将**种子点**(x, y)入栈。

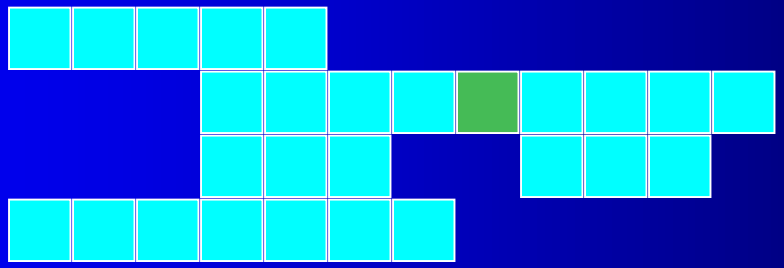
(2) **出栈**: 若栈空则结束。否则取栈顶元素(x, y)，以y作为当前扫描线。

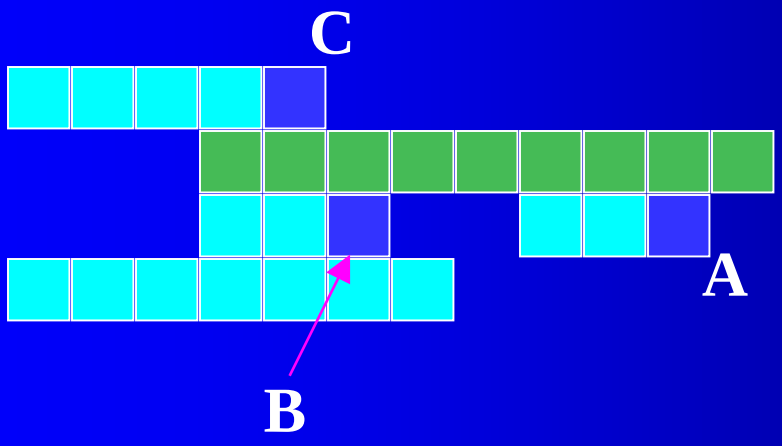
(3) **填充并确定种子点所在区段**: 从种子点(x, y)出发，沿当前扫描线向左、右两个方向填充，直到边界。分别标记区段的左、右端点坐标为xLeft和xRight。

(4) **确定新的种子点**：在区间 [*x*Left, *x*Right]中检查与当前扫描线*y*上、下相邻的两条扫描线是否全为边界象素或者前面已经填充过的象素。若这些扫描线既**不包含边界象素**，也**不包含已填充的象素**，则在 [*x*Left, *x*Right]中把每一区间的**最右象素**作为种子点压入堆栈，返回第(2)步。

上述算法对于每一个待填充区段，只需压栈一次。因此，扫描线填充算法提高了区域填充的效率。

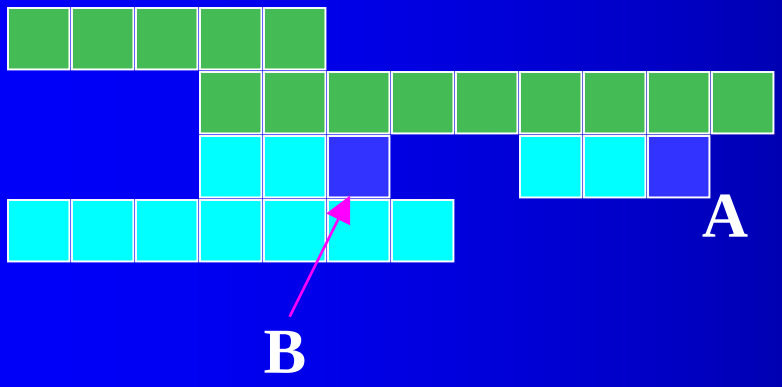
区域填充实例:





Stack

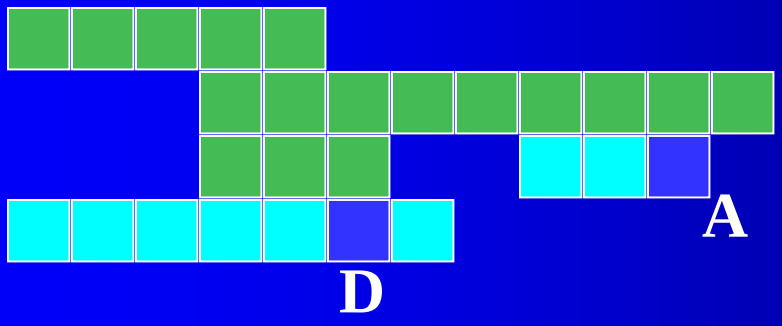
C
B
A



Stack

B

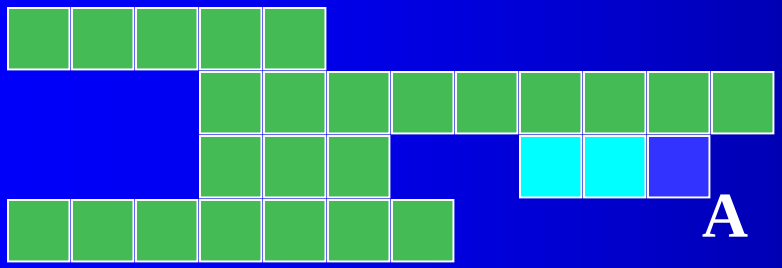
A



Stack

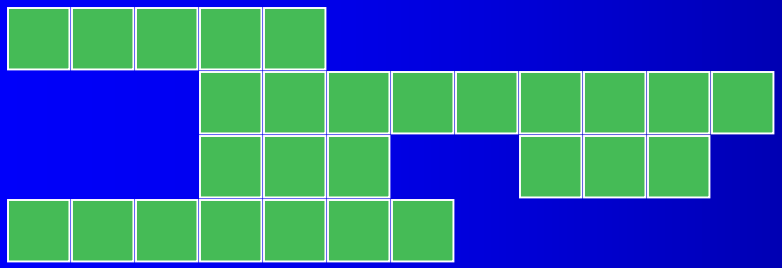
D

A



Stack

A



Stack空！
填充完成！

2.4 字符

字符指数字、字母、汉字等符号。

计算机中字符由一个**数字编码**唯一**标识**。

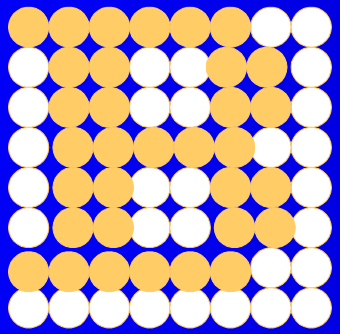
“美国信息交换用标准代码集”简称**ASCII码**。它是用7位（1个字节）二进制数进行编码表示128个字符。

汉字编码的国家标准字符集，每个符号由一个**区码**和一个**位码**（2个字节）共同标识。

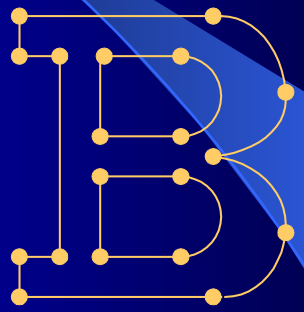
区分ASCII码与汉字编码，采用**字节的最高位**来标识。

点阵字符：每个字符由一个位图表示

矢量字符：记录字符的笔画信息



1	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0



点阵字符　点阵字库中的位图表示　矢量轮廓字符

特点：

- **点阵字符**：存储量大、易于显示
- **矢量字符**：存储量小、美观、变换方便，但需要光栅化后才能显示

字符属性

- 字体　宋体　仿宋体　楷体　黑体　隶书
- 字高　宋体 宋体 宋体 宋体
- 字宽
- 字倾斜角　　倾斜 *倾斜*
- 对齐　　（左对齐、中心对齐、右对齐）
- 字颜色　　红色、绿色