Algorithm 1: Warshell 算法

Input: 集合元素个数 n，关系矩阵 R

Output: 传递闭包关系矩阵 T

```
T ← R;
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            if T_{i,k} = 1 ∧ T_{k,j} = 1 then
                T_{i,j} ← 1
            end
        end
    end
end
```

Algorithm 2: 中点画线法

Input: 直线 L 两端点坐标 $P_0(x_0, y_0), P_1(x_1, y_1)$, 画线颜色 $color$

if $x_0 > x_1$ **then**
    `swap` (P_0, P_1);
end
$a \leftarrow y_0 - y_1, b \leftarrow x_1 - x_0, c \leftarrow x_0 y_1 - x_1 y_0$;
$P_i \leftarrow P_0$;
if $k \in [0, 1]$ **then**
    $d \leftarrow 2a + b$;
    $\Delta P_L \leftarrow (1, 1), \Delta d_L \leftarrow 2(a + b)$;
    $\Delta P_G \leftarrow (1, 0), \Delta d_G \leftarrow 2a$;
else if $k \in (1, +\infty)$ **then**
    $d \leftarrow a + 2b$;
    $\Delta P_L \leftarrow (0, 1), \Delta d_L \leftarrow 2b$;
    $\Delta P_G \leftarrow (1, 1), \Delta d_G \leftarrow 2(a + b)$;
else if $k \in [-1, 0)$ **then**
    $d \leftarrow 2a - b$;
    $\Delta P_L \leftarrow (1, 0), \Delta d_L \leftarrow 2a$;
    $\Delta P_G \leftarrow (1, -1), \Delta d_G \leftarrow 2(a - b)$;
else
    $d \leftarrow a - 2b$;
    $\Delta P_L \leftarrow (1, -1), \Delta d_L \leftarrow 2(a - b)$;
    $\Delta P_G \leftarrow (0, -1), \Delta d_G \leftarrow -2b$;
end
while $P \neq P_1$ **do**
    `putpixel` (P, color);
    if $d < 0$ **then**
        $P \leftarrow P + \Delta P_L, d \leftarrow d + \Delta d_L$;
    else
        $P \leftarrow P + \Delta P_G, d \leftarrow d + \Delta d_G$;
    end
end
`putpixel` (P_1, color);

Algorithm 3: Bresenham 画线法

Input: 直线 L 两端点坐标 $P_0(x_0, y_0), P_1(x_1, y_1)$, 画线颜色 $color$

if $x_0 > x_1$ **then**
    `swap` (P_0, P_1);
end
$\Delta x = x_1 - x_0, \Delta y = y_1 - y_0$;
$P_i \leftarrow P_0$;
if $k \in [0, 1]$ **then**
    $d \leftarrow 2\Delta y - \Delta x$;
    $\Delta P_L \leftarrow (1, 0), \Delta d_L \leftarrow 2\Delta y$;
    $\Delta P_G \leftarrow (1, 1), \Delta d_G \leftarrow 2(\Delta y - \Delta x)$;
else if $k \in (1, +\infty)$ **then**
    $d \leftarrow 2\Delta x - \Delta y$;
    $\Delta P_L \leftarrow (0, 1), \Delta d_L \leftarrow 2\Delta x$;
    $\Delta P_G \leftarrow (1, 1), \Delta d_G \leftarrow 2(\Delta x - \Delta y)$;
else if $k \in [-1, 0)$ **then**
    $d \leftarrow -2\Delta y - \Delta x$;
    $\Delta P_L \leftarrow (1, 0), \Delta d_L \leftarrow -2\Delta y$;
    $\Delta P_G \leftarrow (1, -1), \Delta d_G \leftarrow -2(\Delta y + \Delta x)$;
else
    $d \leftarrow 2\Delta x + \Delta y$;
    $\Delta P_L \leftarrow (0, -1), \Delta d_L \leftarrow 2\Delta x$;
    $\Delta P_G \leftarrow (1, -1), \Delta d_G \leftarrow 2(\Delta x + \Delta y)$;
end
while $P \neq P_1$ **do**
    `putpixel` $(P$, color);
    if $d < 0$ **then**
        $P \leftarrow P + \Delta P_L, d \leftarrow d + \Delta d_L$;
    else
        $P \leftarrow P + \Delta P_G, d \leftarrow d + \Delta d_G$;
    end
end
`putpixel` $(P_1$, color);

Algorithm 4: 中点画圆法

Input: 圆 C 圆心 $C(x, y)$, 半径 R, 画线颜色 *color*

$P_1 \leftarrow (0, R)$;
$d_1 \leftarrow 1 - R$;
$i \leftarrow 1$;
while $P_i \neq (\frac{R}{\sqrt{2}}, \frac{R}{\sqrt{2}})$ **do**
    `DrawCirclePoints` (P_i, color);
    if $d < 0$ **then**
        $P_{i+1} \leftarrow (x_i + 1, y_i), d_{i+1} \leftarrow d_i + 2x_i + 3$;
    else
        $P_{i+1} \leftarrow (x_i + 1, y_i - 1), d_{i+1} \leftarrow d_i + 2(x_i - y_i) + 5$;
    end
    $i \leftarrow i + 1$;
end
`DrawCirclePoints` (P_i, color);

Algorithm 5: Bresenham 画圆法

Input: 圆 C 圆心 $C(x, y)$, 半径 R, 画线颜色 *color*

$P_1 \leftarrow (0, R)$;
$d_1 \leftarrow 2(1 - R)$;
$i \leftarrow 1$;
while $P_i \neq (\frac{R}{\sqrt{2}}, \frac{R}{\sqrt{2}})$ **do**
    `DrawCirclePoints` (P_i, color);
    if $d < 0 \;\&\; 2(d + y) - 1 <= 0$ **then**
        $P_{i+1} \leftarrow (x_i + 1, y_i), d_{i+1} \leftarrow d_i + 2x_i + 3$;
    else
        $P_{i+1} \leftarrow (x_i + 1, y_i - 1), d_{i+1} \leftarrow d_i + 2(x_i - y_i + 3)$;
    end
    $i \leftarrow i + 1$;
end
`DrawCirclePoints` (P_i, color);

Algorithm 6: 多边形逼近画圆法

Input: 圆 C 圆心 $C(x, y)$, 半径 R, 画线颜色 *color*

$n \leftarrow 3\sqrt{R}$;

$\theta \leftarrow \frac{2\pi}{n}$;

$P_1 \leftarrow (0, R)$;

$i \leftarrow 2$;

while $i <= n$ **do**

  $P_i = (x_{i-1}\cos\theta - y_{i-1}\sin\theta, y_{i-1}\cos\theta + x_{i-1}\sin\theta)$;

  `DrawLine` (P_{i-1},P_i, color);

  $i \leftarrow i + 1$;

end

`DrawLine` (P_n,P_1, color);

Algorithm 7: 任意角度画弧法

Input: 圆弧 $\overset{\frown}{SE}$ 所属圆的圆心 $C(x_c, y_c)$, 圆弧起点 $S(x_s, y_s)$, 圆弧终点方位点 $E_{pos}(x_{epos}, y_{epos})$, 画线颜色 $color$

// 求得平移到原点的圆其对应圆心、起点、终点方位点和实际终点

$C' \leftarrow (0,0), S' \leftarrow (x_s - x_c, y_s - y_c)$;

$E'_{pos} \leftarrow (x_{epos} - x_c, y_{epos} - y_c), E' \leftarrow C'E'_{pos} \cap \odot C'$;

// 确定起点和终点所属区域或界点

$region_s = \texttt{GetRegion}(S), region_e = \texttt{GetRegion}(E)$;

// 划分完全绘制区域、部分绘制区域和非绘制区域

`Fill` (state,0);

if $region_s = K_i$ & $region_e = K_j$ **then**

  for $R = R_i \to R_{j-1}$ **do**

    state$[R] \leftarrow 1$;

  end

else if $region_s = K_i$ & $region_e = R_j$ **then**

  for $R = R_i \to R_{j-1}$ **do**

    state$[R] = 1$;

  end

  $MN \leftarrow (\min(x_{K_j}, x_E), \min(y_{K_j}, y_E)), MX \leftarrow (\max(x_{K_j}, x_E), \max(y_{K_j}, y_E))$;

  $Rect \leftarrow (MN, MX)$, state$[region_e] \leftarrow -1$;

else if $region_s = R_i$ & $region_e = K_j$ **then**

  for $R = R_{i+1} \to R_j$ **do**

    state$[R] = 1$;

  end

  $MN \leftarrow (\min(x_{K_{i+1}}, x_S), \min(y_{K_{i+1}}, y_S)), MX \leftarrow (\max(x_{K_{i+1}}, x_S), \max(y_{K_{i+1}}, y_S))$;

  $Rect \leftarrow (MN, MX)$, state$[region_s] \leftarrow -1$;

else

  for $R = R_{i+1} \to R_{j-1}$ **do**

    state$[R] = 1$;

  end

  if S, E 属于同一象限 & S 在 E 顺时针侧 **then**

    $MN \leftarrow (\min(x_S, x_E), \min(y_S, y_E)), MX \leftarrow (\max(x_S, x_E), \max(y_S, y_E))$;

    $Rect \leftarrow (MN, MX)$;

  else

    $MN_1 \leftarrow (\min(x_{K_j}, x_E), \min(y_{K_j}, y_E)), MX_1 \leftarrow (\max(x_{K_j}, x_E), \max(y_{K_j}, y_E))$;

    $MN_2 \leftarrow (\min(x_{K_{i+1}}, x_S), \min(y_{K_{i+1}}, y_S)), MX_2 \leftarrow (\max(x_{K_{i+1}}, x_S), \max(y_{K_{i+1}}, y_S))$;

    $Rect \leftarrow (MN_1, MX_2) \cup (MN_2, MX_2)$;

  end

  state$[region_s] \leftarrow -1$, state$[region_e] \leftarrow -1$;

end

// 八方对称遍历圆像素点，每次生成的像素点需要判断是否在绘制区域内

$\texttt{DrawCircleOfArc}(C, R, color, \text{state}, Rect)$;