

# Cac thua toan to mau

## Dan nhap

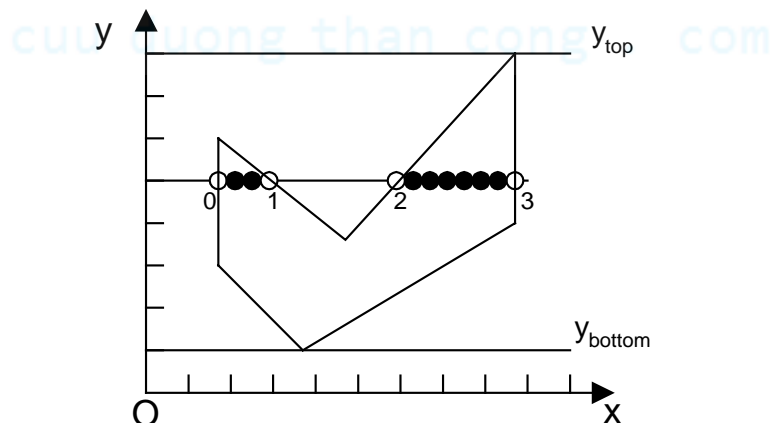
- Môt vung tôi thông nôi xac nôi bô môt nôi kệp kín nôi nôi gôi lă nôi biên. Dăng nôi biên nôi gian thông gáp lă nă gia.
- Côi hai dăng vung tôi thông gáp : tôi bâng môt mau thuan nhât (solid fill) vôi tôi theo môt mau tôi (fill-pattern) nôi nôi
- Viêc tôi mau thông nôi chia lam hai công nôi :
  - ♦ Xac nôi vôi trí cac nôi cên tôi mau.
  - ♦ Quyêc nôi tôi cac nôi trên bâng mau nôi. Công nôi nay thôi sô phôi tăp khi ta cên tôi theo môt mau tôi nôi nôi không phôi tôi thuan môt mau.
- Côi hai cêch tiêp cên chính : tôi mau theo dng quêt vôi tôi mau dă theo nôi biên.
  - ♦ Phôi phâp tôi mau dă theo dng quêt sô xac nôi phâ giao côi cac dng quêt kêt tiêp nhau vôi nôi biên côi vung tôi sau nôi sô tiêp hânh tôi mau cac nôi thôi phâ giao nay. Cêch nay thông nôi dng nêi tôi mau nă gia, nôi tron, ellipse vôi môt sô nôi cong nôi gian khac.
  - ♦ Phôi phâp tôi mau dă theo nôi biên sô băt nău tôi môt nôi bên trong vung tôi vôi tôi nôi loang dă ra cho nêi khi gáp nôi biên. Cêch nay thông nôi dng cho cac dăng nôi biên phôi tăp.

# Thuật toán tìm kiếm theo chiều rộng

Bài toán đặt ra : Cần tìm mau một ña giác cho bôú  $N$  ñanh  $P_i(x_i, y_i), i = 0, \dots, N-1$ . Ña giác này có thể là ña giác loá, ña giác lõm, vạcá ña giác tòi cáđ, ...

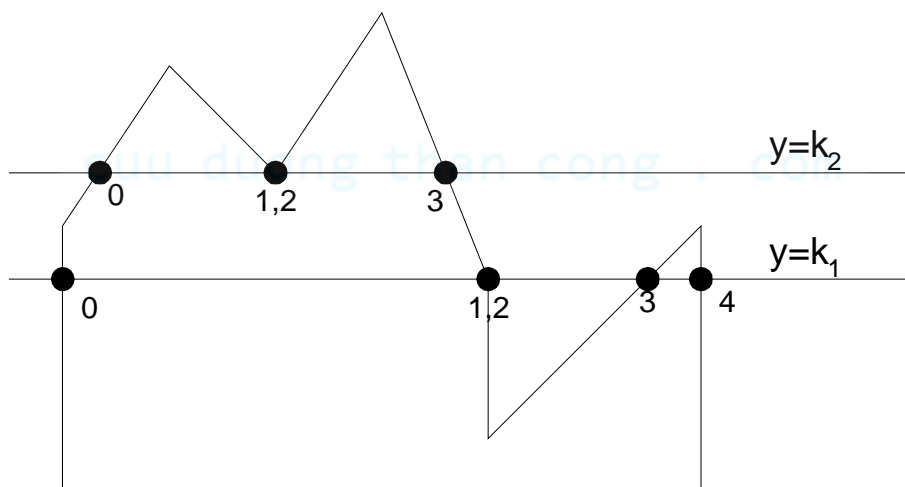
## Tóm tắt các bước chính của thuật toán

- Tìm  $y_{top}$ ,  $y_{bottom}$  lần lượt là giá trị lớn nhất, nhỏ nhất của tập các tung độ của các cạnh của đa giác này cho:  $y_{top} = \max\{y_i, (x_i, y_i) \in P\}$ ,  $y_{bottom} = \min\{y_i, (x_i, y_i) \in P\}$ .
- Với mỗi dòng quét  $y = k$ , với  $k$  thay đổi từ  $y_{bottom}$  đến  $y_{top}$ , làm:
  - ♦ Tìm tất cả các hoành độ giao điểm của dòng quét  $y = k$  với các cạnh của đa giác.
  - ♦ Sắp xếp các hoành độ giao điểm theo thứ tự tăng dần:  $x_0, x_1, x_2, \dots$ ,
  - ♦ Tính diện tích miền thang trên nông thang  $y = k$  lần lượt theo giới hạn bởi các cặp  $(x_0, x_1), (x_1, x_2), \dots, (x_{2k}, x_{2k+1})$ .



## Các vấn đề nảy ra

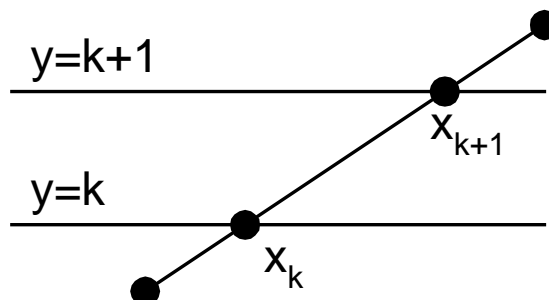
- Hạn chế số cạnh cần tìm giao điểm với mỗi dòng quét vì với mỗi dòng quét, không phải lúc nào tất cả các cạnh của đa giác cũng tham gia cắt dòng quét.
- Xác định nhanh hoành độ giao điểm vì nếu lặp lại thao tác tìm giao điểm của cạnh đa giác với mỗi dòng quét bằng cách giải hệ phương trình sẽ tốn rất nhiều thời gian.
- Giải quyết trường hợp số giao điểm với trường hợp dòng quét đi ngang qua đỉnh : Nếu số giao điểm tìm được giữa các cạnh đa giác và dòng quét là lẻ thì việc nhóm từng cặp giao điểm kế tiếp nhau để hình thành các đoạn tọa độ sẽ không chính xác. Nhiều nay chỉ xảy ra khi dòng quét đi ngang qua các đỉnh của đa giác.
- Ngoài ra, việc tìm giao điểm của dòng quét với các cạnh nằm ngang là một trường hợp đặc biệt cần phải có cách xử lý thích hợp



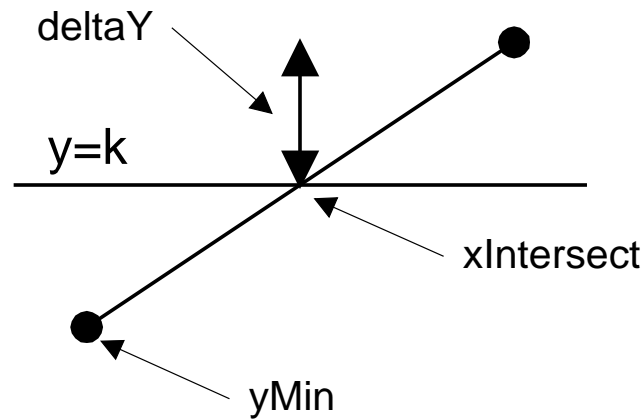
## Tổ chức cấu trúc dữ liệu và thuật toán

- Danh sách các cạnh (Edge Table – ET) : chứa toàn bộ các cạnh của đa giác (**không loại bỏ các cạnh nằm ngang**) được sắp theo thời điểm tăng dần của  $y_{Min}$ .
- Danh sách các cạnh kích hoạt (Active Edge Table – AET) : chứa các cạnh của đa giác có thể cắt ngang với dòng quét hiện hành, các cạnh này được sắp theo thời điểm tăng dần của hoành độ giao điểm giữa cạnh và dòng quét.
- Khi dòng quét đi từ bottom đến top, các cạnh thỏa mãn điều kiện sẽ được di chuyển từ ET sang AET:
  - ♦ Khi dòng quét  $y = k$  bắt đầu cắt một cạnh, nghĩa là  $k \geq y_{Min}$ , cạnh này sẽ được chuyển từ ET sang AET.
  - ♦ Khi dòng quét không còn cắt cạnh này nữa, nghĩa là  $k > y_{Max}$ , cạnh này sẽ bị loại bỏ khỏi AET.
  - ♦ Khi không còn cạnh nào trong ET hay AET nữa, quá trình tô màu kết thúc.
- Nếu tìm giao điểm giữa cạnh đa giác và dòng quét hiện hành nhanh, ta có thể nhận xét :

$$x_{k+1} - x_k = \frac{1}{m} ((k+1) - k) = \frac{1}{m} \text{ hay } x_{k+1} = x_k + \frac{1}{m}.$$



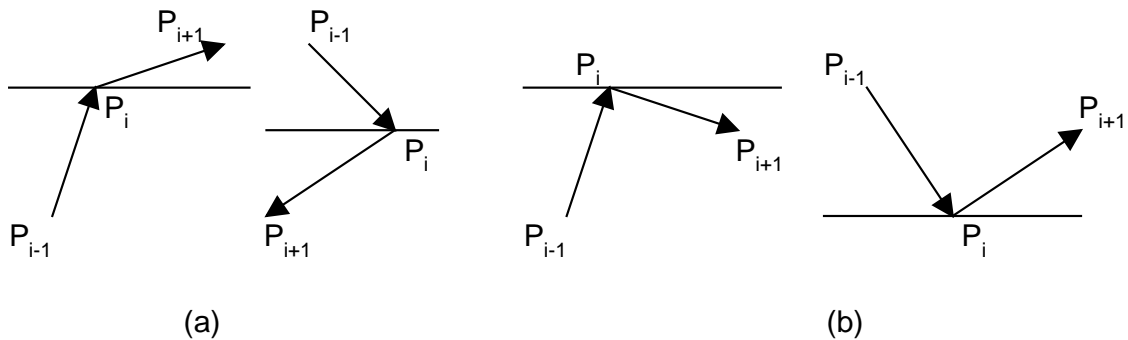
## Neà xuất cấu trúc dữ liệu của một cạnh (EDGE)



- $y_{Min}$ : giá trị tung nhỏ nhất trong 2 đỉnh của cạnh.
- $xIntersect$ : hoành độ giao điểm của cạnh với **dong quét hiện hành**.
- $DxPerScan$ : giá trị  $1/m$  ( $m$  là hệ số góc của cạnh).
- $\delta y$ : khoảng cách từ dòng quét hiện hành tới đỉnh  $y_{Max}$ . Lúc này nếu khi  $k > y_{Max}$  thì thành  $\delta y \leq 0$ .
- Giá trị  $xIntersect$  được khởi gán ban đầu là hoành độ của đỉnh có tung nhỏ là  $y_{Min}$ , và giá trị  $\delta y$  được khởi gán ban đầu là  $y_{Max} - y_{Min} + 1$ .

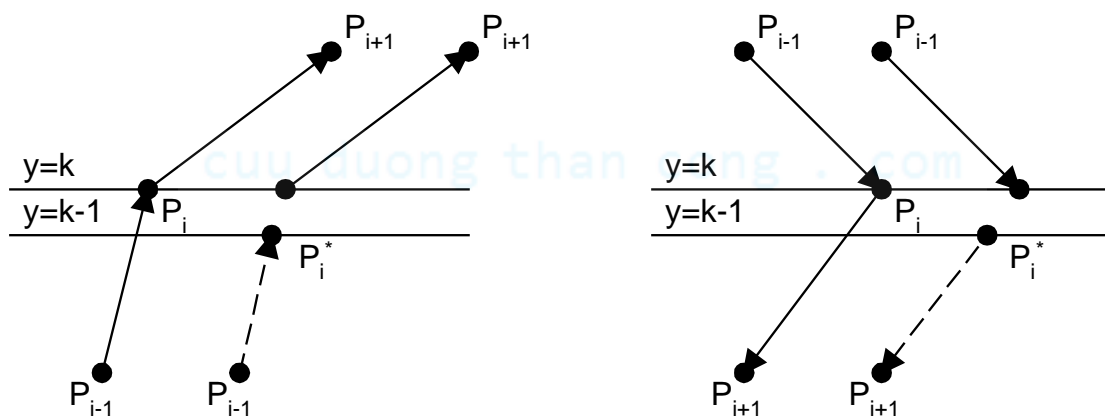
## Giải quyết trường hợp đồng quy qua đỉnh

- Tính một giao điểm nếu chiều của hai cạnh kề của đỉnh đó cùng hướng tăng hay giảm.
- Tính hai giao điểm nếu chiều của hai cạnh kề của đỉnh đó cùng hướng thay đổi, nghĩa là tăng-giảm hay giảm-tăng.

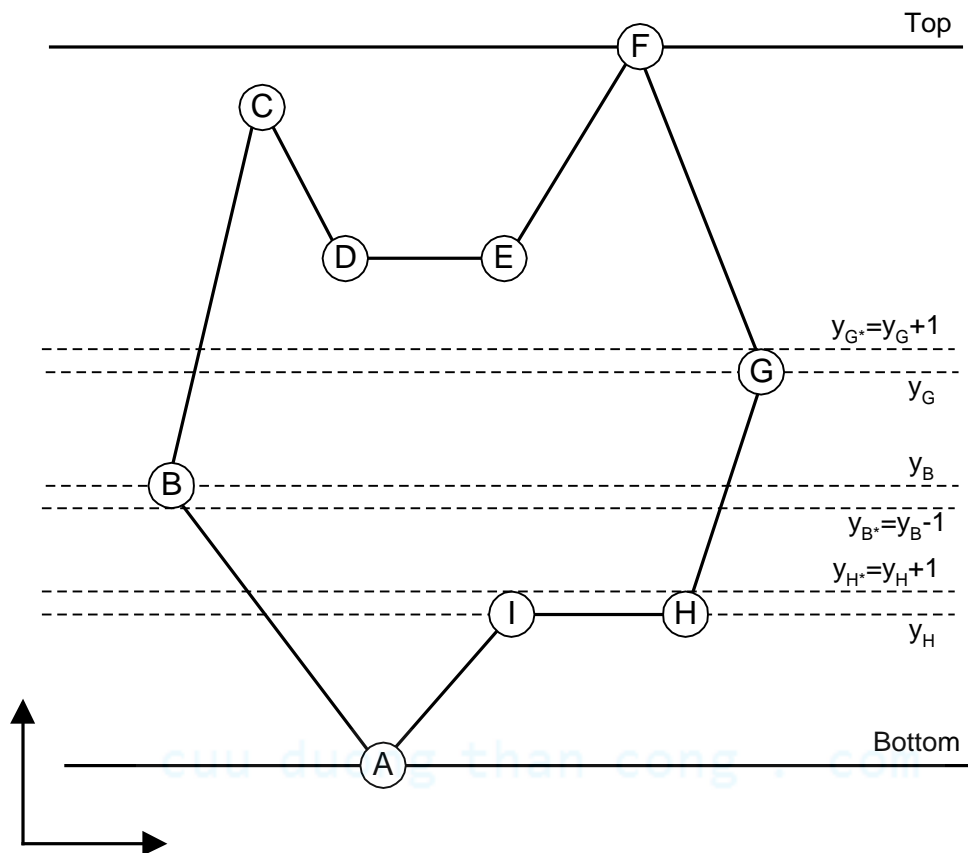


cuuduongthancong.com

- Khi cài đặt nếu phải xét nhiều kiến này cho phức tạp, khi xây dựng dữ liệu cho mỗi cạnh trước khi nối vào ET, người ta sẽ xử lý các cạnh của đỉnh tính hai giao điểm bằng cách loại bỏ một pixel trên cùng của một trong hai cạnh.



## Minh họa thuật toán



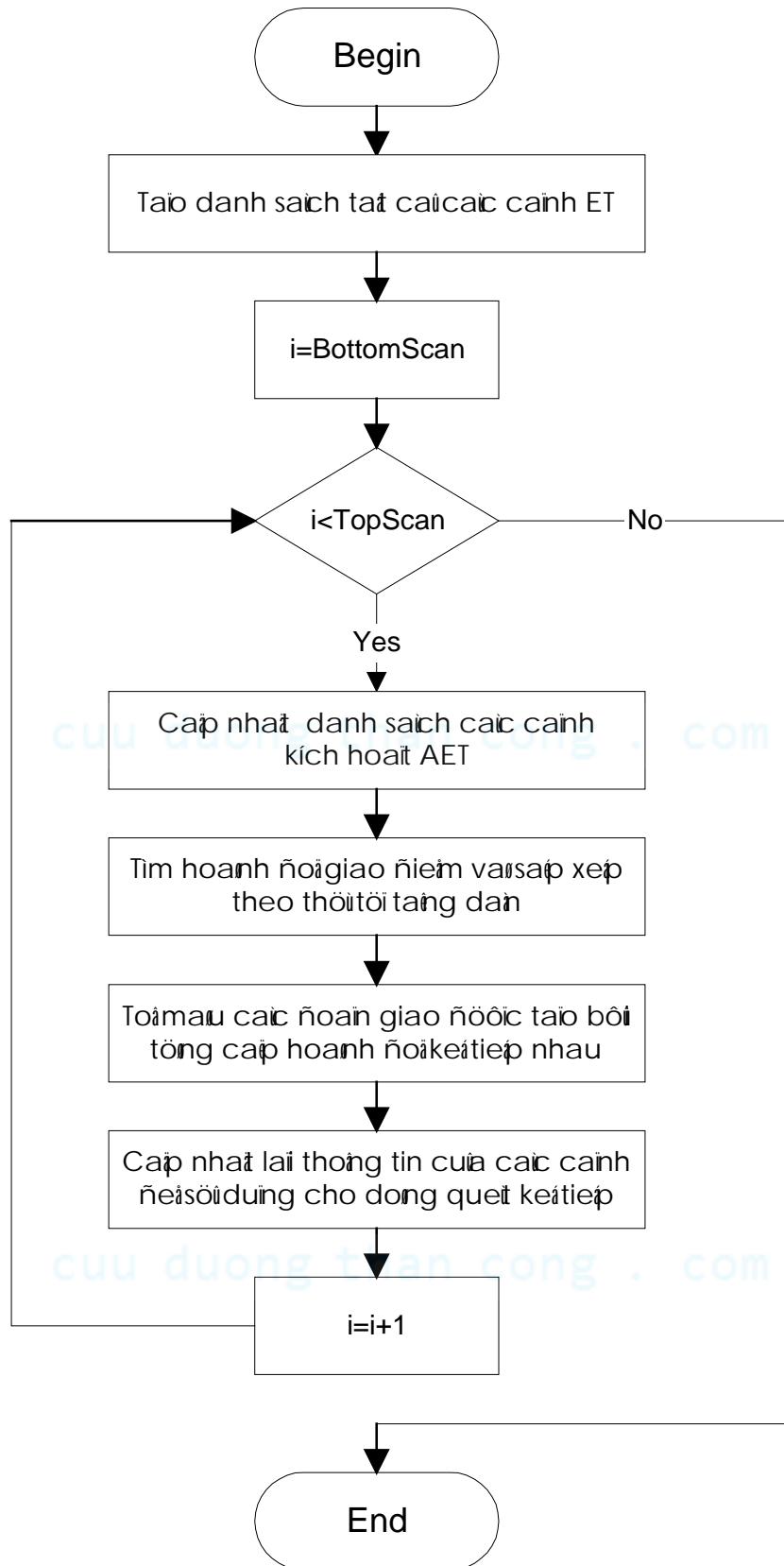
- Ban đầu :
  - ♦ ET :  $AB^*$ , AI,  $H^*G$ , BC,  $G^*F$ , DC, EF. (loại IH và DE)
  - ♦ AET : NULL.
- Khi đang quét nút  $y=y_A$ 
  - ♦ ET :  $H^*G$ , BC,  $G^*F$ , DC, EF. (chuyển  $AB^*$ , AI sang AET)
  - ♦ AET :  $AB^*$ , AI.
- Khi đang quét nút  $y=y_{H^*}$ 
  - ♦ ET : BC,  $G^*F$ , DC, EF. (chuyển  $H^*G$  sang AET)
  - ♦ AET :  $AB^*$ ,  $H^*G$ . (loại AI vì không còn tại dòng quét)

- Khi đồng queit ñait  $y=y_B$ 
  - ♦ ET :  $G^*F$ , DC, EF. (chuyẽn BC sang AET)
  - ♦ AET : BC,  $H^*G$ . (loại  $AB^*$ , sắp xếp lại  $H^*G$  và BC)
- Khi đồng queit ñait  $y=y_{G^*}$ 
  - ♦ ET : DC, EF. (chuyẽn  $G^*F$  sang AET)
  - ♦ AET : BC,  $G^*F$ . (loại  $H^*G$  vì không còn cái đồng queit)
- Khi đồng queit ñait  $y=y_D$ 
  - ♦ ET : NULL. (chuyẽn DC, EF sang AET)
  - ♦ AET : BC, DC, EF,  $G^*F$ . (sắp xếp lại BC,  $GF^*$ , DC, EF)
- Khi đồng queit ñait  $y=y_{C+1}$ 
  - ♦ ET : NULL.
  - ♦ AET : EF,  $G^*F$ . (loại BC, DC vì không còn cái đồng queit)
- Khi đồng queit ñait  $y=y_{F+1}$ 
  - ♦ ET : NULL.
  - ♦ AET : NULL. (loại EF,  $G^*F$  vì không còn cái đồng queit).
- Thuật toán dừng tại ñây.

cuu duong than cong . com



*Lưu nội thuật toán tìm mẫu theo dòng quét*



## Một số hằng định cài đặt

```
#define MAXVERTEX 20
#define MAXEDGE 20
#define TRUE 1
#define FALSE 0
```

```
typedef struct {
    int x;
    int y;
}POINT;
```

```
typedef struct{
    int NumVertex;
    POINT aVertex[MAXVERTEX];
}POLYGON;
```

```
typedef struct {
    int NumPt;
    float xPt[MAXEDGE];
}XINTERSECT;
```

```
typedef struct
{
    int yMin; // Gia tri y nho nhat cua 2 dinh
    float xIntersect; // Hoanh do giao diem cua canh & dong quet
    float dxPerScan; // Gia tri 1/m
    int DeltaY;
}EDGE;
```

```
typedef struct
{
    int NumEdge;
    EDGE aEdge[MAXEDGE];
}EDGELIST;
```

```

void PutEdgeInList(EDGEList &EdgeList, POINT p1, POINT p2, int NextY)
{
    EDGE EdgeTmp;

    EdgeTmp.dxPerScan = float(p2.x-p1.x)/(p2.y-p1.y); // 1/m
    if(p1.y < p2.y)
    {
        /*
            Trường hợp dòng quét đi ngang qua đỉnh là giao điểm
            của 2 cạnh có hướng y cùng tăng
        */
        if(p2.y < NextY)
        {
            p2.y--;
            p2.x -= EdgeTmp.dxPerScan;
        }
        EdgeTmp.yMin = p1.y;
        EdgeTmp.xIntersect= p1.x;
        EdgeTmp.DeltaY = abs(p2.y-p1.y)+1;
    } // if
    else
    {
        /*
            Trường hợp dòng quét đi ngang qua đỉnh là giao điểm của 2 cạnh có
            hướng y cùng giảm
        */
        if(p2.y > NextY)
        {
            p2.y++;
            p2.x+= EdgeTmp.dxPerScan;
        }
        EdgeTmp.yMin = p2.y;
        EdgeTmp.xIntersect= p2.x;
        EdgeTmp.DeltaY = abs(p2.y-p1.y)+1;
    } // else
    // xác định vị trí chèn
    int j = EdgeList.NumEdge;
    while((j>0) && (EdgeList.aEdge[j-1].yMin>EdgeTmp.yMin))
    {
        EdgeList.aEdge[j] = EdgeList.aEdge[j-1];
        j--;
    }
    // tiến hành chèn đỉnh mới vào cạnh
    EdgeList.NumEdge++;
    EdgeList.aEdge[j] = EdgeTmp;
} // PutEdgeInList

```

```

/*
    Tim dinh ke tiep sao cho khong nam tren cung duong thang voi dinh dang
    xet
*/
int FindNextY(POLYGON P, int id)
{
    int j = (id+1)%P.NumVertex;
    while((j<P.NumVertex)&&(P.aVertex[id].y == P.aVertex[j].y))
        j++;
    if(j<P.NumVertex)
        return (P.aVertex[j].y);
    return 0;
} // FindNextY

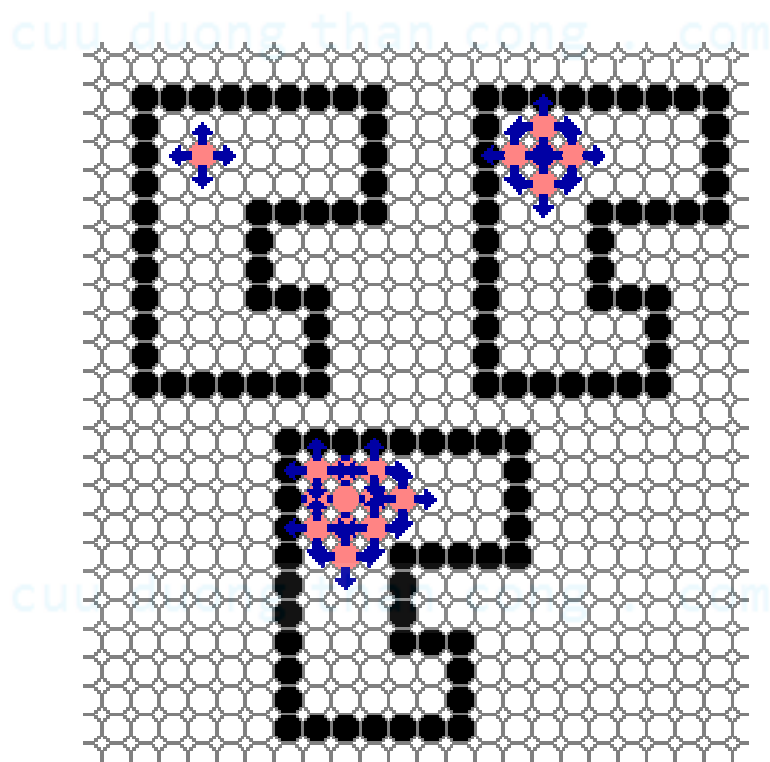
// Tao danh sach cac canh tu polygon da cho
void MakeSortedEdge(POLYGON P, EDGELIST &EdgeList,
                    int &TopScan, int &BottomScan)
{
    TopScan = BottomScan = P.aVertex[0].y;
    EdgeList.NumEdge = 0;
    for(int i=0; i<P.NumVertex; i++)
    {
        // Truong hop canh khong phai la canh nam ngang
        if(P.aVertex[i].y != P.aVertex[i+1].y)
            PutEdgeInList(EdgeList, P.aVertex[i], P.aVertex[i+1], FindNextY(P,
            i+1));
        // Xu li truong hop  canh nam ngang
        else
            if(P.aVertex[i+1].y > TopScan)
                TopScan = P.aVertex[i+1].y;
    }
    BottomScan = EdgeList.aEdge[0].yMin;
} //MakeSortedEdge

// Cap nhat lai hai con tro FirstId, LastId cho biet danh sach cac canh active
void UpdateActiveEdgeList(EDGELIST EdgeList, int yScan, int &FirstId, int
&LastId)
{
    while((FirstId<EdgeList.NumEdge-1) &&(EdgeList.aEdge[FirstId].DeltaY ==
    0))
        FirstId++;
    while((LastId<EdgeList.NumEdge-1)
        &&(EdgeList.aEdge[LastId+1].yMin<=yScan))
        LastId++;
} // UpdateActiveEdgeList

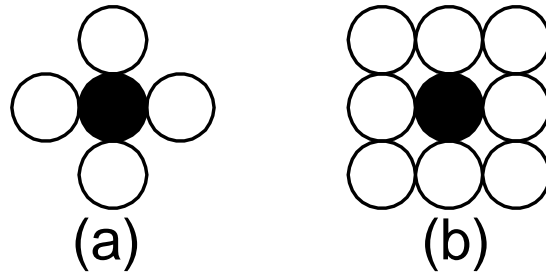
```

# Thuật toán tô màu theo biên

- Bài toán đặt ra : Cần tô màu vùng tô nếu biết một màu của biên vùng tô và một điểm nằm bên trong vùng tô
- Ý tưởng : Bắt đầu từ điểm nằm bên trong vùng tô kiểm tra các điểm lân cận của nó nếu nó có tô hay có phải là điểm có màu trung màu biên hay không, nếu không phải thì ta sẽ tô điểm đó. Quá trình này sẽ lặp lại cho tới khi không còn tô được nữa thì dừng.



- Còi hai quan ñiêm về cách tô màu, ñòi là dùng 4 ñiêm lân cận (hình a) hay 8 ñiêm lân cận (hình b).



- Cài ñặt minh họa thuật toán tô màu theo ñiều kiện biên

```
void BoundaryFill(int x, int y, int FillColor, int BoundaryColor)
{
    int CurrenColor;

    CurrenColor = getpixel(x,y);
    if((CurrenColor!=BoundaryColor)&&CurrenColor!= FillColor))
    {
        putpixel(x,y,FillColor);
        BoundaryFill(x-1, y, FillColor, BoundaryColor);
        BoundaryFill(x, y+1, FillColor, BoundaryColor);
        BoundaryFill(x+1, y, FillColor, BoundaryColor);
        BoundaryFill(x, y-1, FillColor, BoundaryColor);
    }
} // Boundary Fill
```

- Một số nhận xét
  - ♦ Thuật toán còi thể hoạt ñộng không chính xác khi còi một số ñiêm nằm trong vùng tô còi màu là màu cần tô của vùng.
  - ♦ Việc thực hiện ñây qui làm thuật toán không thể dùng cho vùng tô lõn.

- Một cái tiến nhỏ: nhận xét rằng việc gọi thóc hiện nên qui thuật toán cho 4 điểm lân cận của điểm hiện hành không quan tâm tới một trong 4 điểm nào đó xét ô bên trên hay dưới. Ví dụ khi ta xét 4 điểm lân cận của (x, y), thì khi gọi thóc hiện nên qui với điểm hiện hành là một trong 4 điểm trên, (x, y) vẫn nên xem là điểm lân cận của chúng và nên gọi thóc hiện lại.

```
void BoundaryFillEnhanced(int x, int y, int F_Color, int B_Color)
```

```
{
    int CurrenColor;
    CurrentColor = getpixel(x,y);
    if((CurrentColor!=B_Color)&&CurrentColor!= F_Color))
    {
        putpixel(x,y,F_Color);
        FillLeft(x-1, y, F_Color, B_Color);
        FillTop(x, y+1, F_Color, B_Color);
        FillRight(x+1, y, F_Color, B_Color);
        FillBottom(x, y-1, F_Color, B_Color);
    }
} // BoundaryFillEnhanced
```

```
void FillLeft(int x, int y, int F_Color, int B_Color)
```

```
{
    int CurrenColor;
    CurrentColor = getpixel(x,y);
    if((CurrentColor!=B_Color)&&CurrentColor!= F_Color))
    {
        putpixel(x,y,F_Color);
        FillLeft(x-1, y, F_Color, B_Color);
        FillTop(x, y+1, F_Color, B_Color);
        FillBottom(x, y-1, F_Color, B_Color);
    }
} // FillLeft
```

- Một cái tiến khác : không cái nào ã ã ã qui mà toàn theo tổng dòng.

