ค่ายอบรมโอลิมปิกวิชาการ 2

โครงสร้างข้อมูล: กราฟโดยปริยาย

Data Structure: Implicit graph

รัชดาพร คณาวงษ์

24 มีนาคม 2566

ศูนย์มหาวิทยาลัยศิลปากร

เนื้อหาที่ครอบคลุม

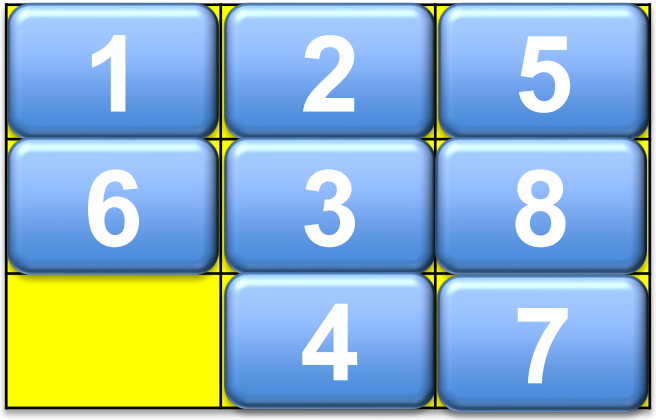
- Implicit graph
- Flood fill algorithm
- Connected componented analysis

กราฟ

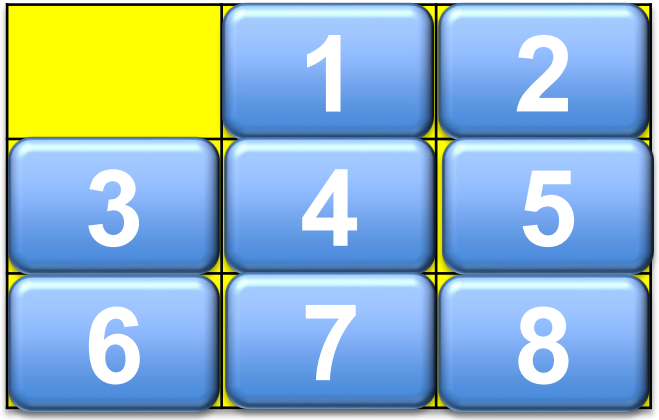
- กราฟปรกติที่เรารู้จักจะแทนโหนดต้องค่าใดๆ และเส้นเชื่อมโหนดคือบอกความสัมพันธ์ของโหนดต่อโหนด
- กราฟมีทั้งแบบ directed graph และ undirected graph และเราสามารถกำหนดค่าน้ำหนักหรือไม่กำหนดบนเส้นเชื่อมก็ได้
- แต่บางปัญหาเราไม่สามารถแทนกราฟได้อย่างปรกติ

เกมส์เรียงตัวเลขตามลำดับ

- บนกระดานจะมีช่องว่างหนึ่งช่อง ส่วนช่องอื่นๆ จะมีตัวเลขอยู่ สามารถเลื่อนตัวเลขในช่องที่ติดกับช่องว่างมาที่ช่องว่างได้ โดยเริ่มต้นจะเป็นตามรูป 1a ให้เรียงตัวเลขให้ได้ตามรูป 1b



รูป 1a เริ่มต้นเกมส์

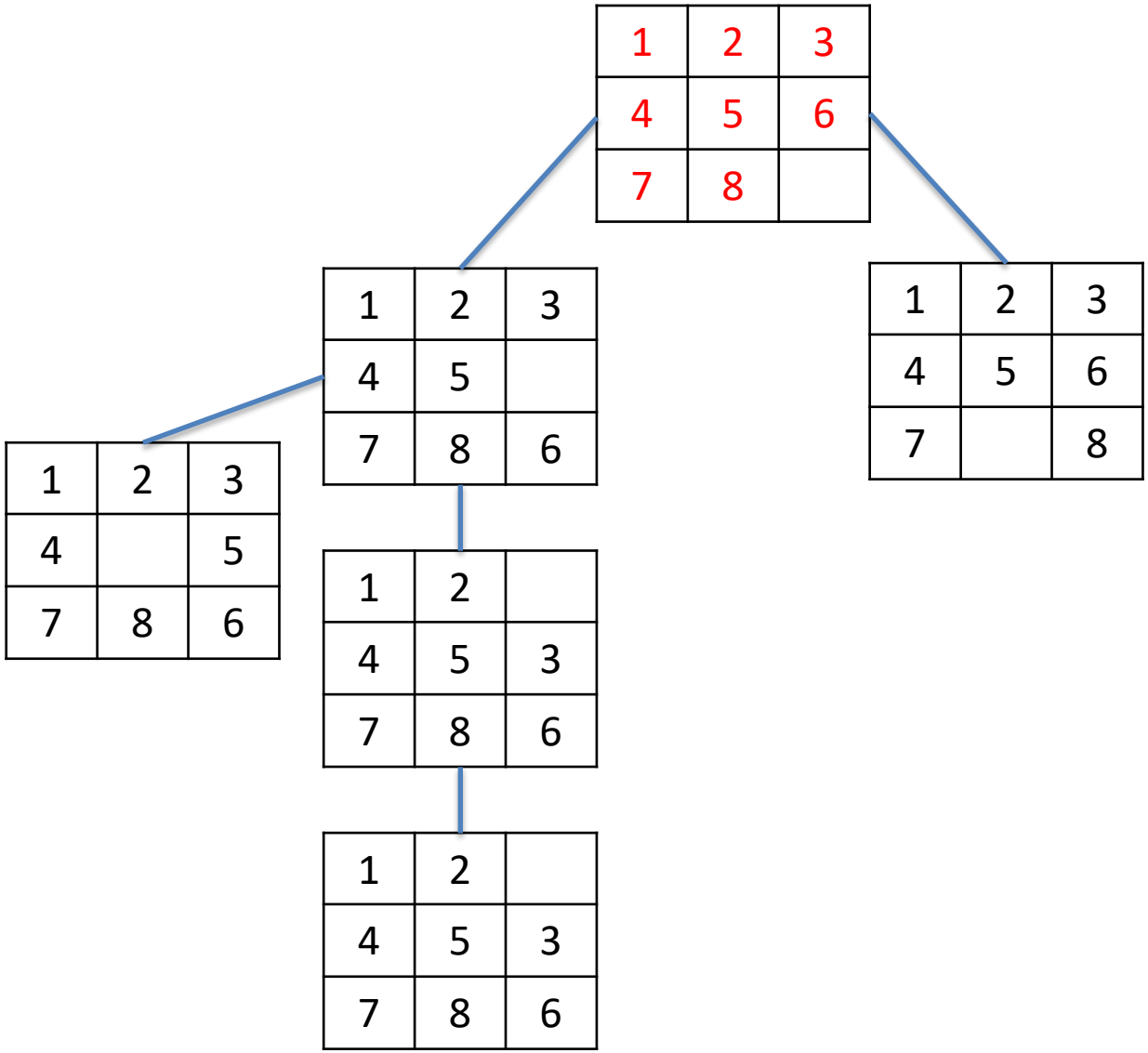


รูป 1b ผลลัพธ์ที่ต้องการ

แก้ปัญหาดูสิ

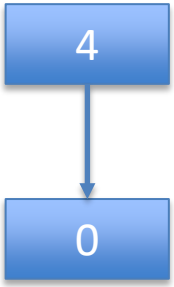
ใช้กราฟแก้ปัญหาได้อย่างไร

- ถ้าเรากำหนดให้โหนดคือรูปแบบการเรียงของตาราง
- และ edge คือตัวเลขที่เลื่อนไปทำให้เกิดการเรียงที่ต่างจากเดิม
- เราจะเขียนกราฟได้อย่างไร
  - กราฟนี้จะต้องมีรูปแบบการเรียงได้ 9! คิดเป็น 362880 รูปแบบซึ่งก็คือจำนวนโหนดที่เราต้องมี
  - แล้ว edge หล่ะ แต่ละโหนดอาจจะมีเส้นเชื่อมที่ทำให้เปลี่ยนรูปแบบการจัดได้ 2, 3 หรือ 4 ขึ้นกับตำแหน่งของช่องว่าง ดังนั้นกราฟนี้อาจจะมีเส้นเชื่อมถึง 1,000,000 หรือมากกว่าได้


1 2 3
4 5 6
7 8
1 2 3
4 5
7 8 6
1 2 3
4 5 6
7 8
1 2 3
4 5
7 8 6
1 2
4 5 3
7 8 6
1 2
4 5 3
7 8 6

	0	1	2	…	362779	362880
0	0	1	1	0	0	0
1	1					
2				1	0	0
…						
362779						
362880						

0
1
2
…
362778
362779
362880



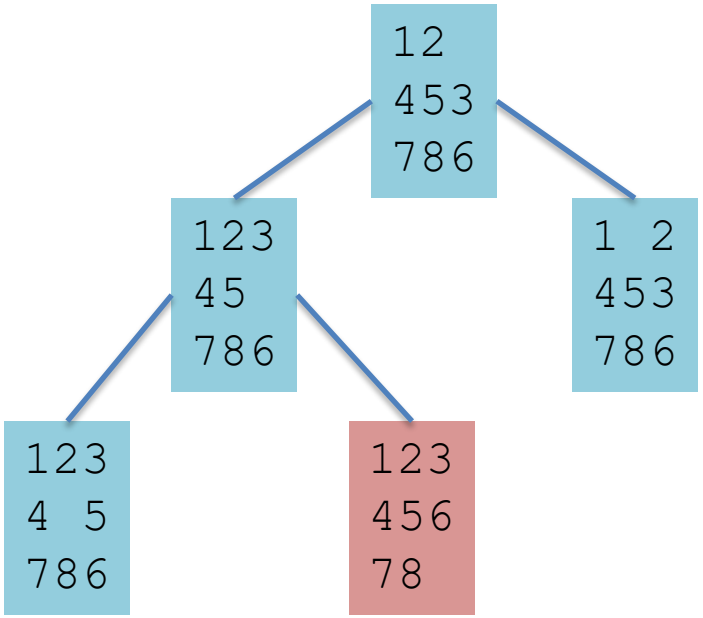
1	2	
4	5	3
7	8	6

1	2	3
4	5	
7	8	6

1	2	3
4	5	6
7	8	

1	2	
4	5	3
7	8	6

1	2	3
4	5	6
7	8	



มีทางอื่นอีกนะ implicit graph ช่วยได้

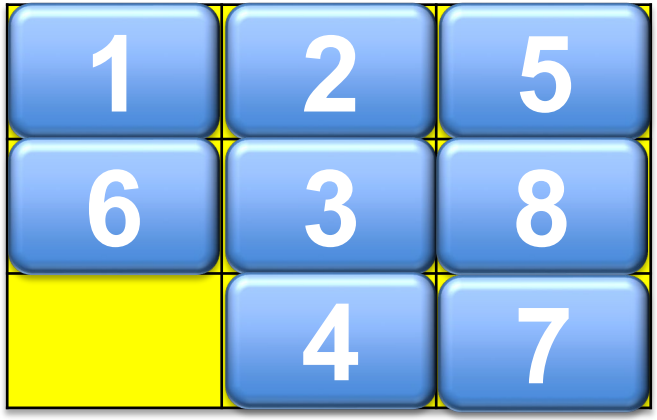
- เราเสนอให้โหนดแทนลักษณะการวางเลขในตารางในเกมส์
  - รูปแบบการวางตัวเลขในตาราง
  - แทนที่จะสร้างทุกรูปแบบ เราก็สร้างเฉพาะบางโหนดเพื่อใช้ในการค้นหาการเล่น
  - Edge เป็นสิ่งที่ต้องมีโดยปริยาย (implicit) เพราะเป็น action ที่ทำให้เกิดโหนดใหม่จากโหนดเดิม

โหนด (A node)

- ลองแปลงรูปแบบของตารางให้อยู่ในรูปข้อความ ซึ่งจะแสดงเป็นโหนด

ถ้าเราเขียนเรียงจากบนลงล่างและซ้ายไปขวาจะได้สตริงดังนี้

“125638047”



จะเห็นว่าตำแหน่งช่องว่างเราแทนด้วย ‘0’ ซึ่งก็คือตำแหน่งที่ 6 นั่นเอง

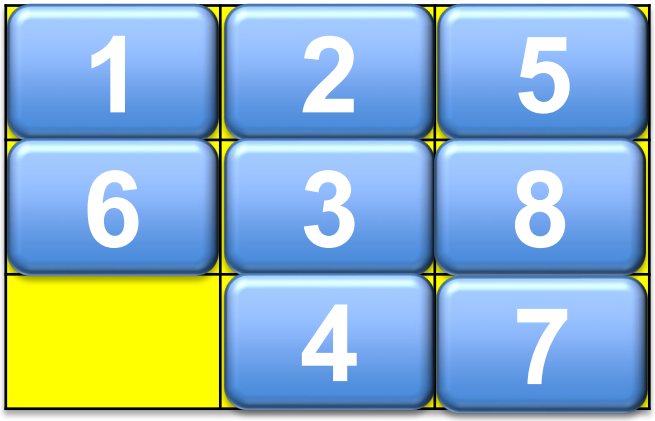
0	1	2
3	4	5
6	7	8

เส้นเชื่อม (An Edge)

- ต้องมีการกำหนดการเลื่อนที่ถูกต้อง ดังนั้นเราจึงต้องกำหนดการดำเนินการที่จะเปลี่ยนรูปแบบการเรียงจากรูปแบบหนึ่งไปเป็นอีกรูปแบบหนึ่ง
- บางครั้งเราก็สร้างกฎเพื่อมากำหนดรูปแบบของการดำเนินการให้สอดคล้องกับปัญหา
- ดังนั้นเส้นเชื่อม (Edge) จึงเป็นสิ่งที่สร้างโหนดใหม่ ซึ่งรับรู้ได้โดยปริยายว่าโหนดที่เป็นเดิมและโหนดใหม่ที่ถูกสร้างจากกฎมีความเชื่อมโยงกันโดยไม่ต้องใช้การเชื่อมต่อกันจริงๆ ในการเก็บข้อมูล

กฎการเลื่อนในเกมส์จัดเรียงตัวเลข

- เกมส์จัดเรียงตัวเลข ต้องการทำการเลื่อนให้ถูกต้องตามกายภาพของตารางซึ่งก็คือเราสามารถเลื่อนช่องที่อยู่ติดกับช่องว่างในแนวตั้งหรือแนวนอนเท่านั้น



โปรดอย่าสนใจตัวเลขที่เห็นในรูปแบบของเกมส์เพราะมันสามารถเปลี่ยนแปลงได้ แต่ให้สนใจตำแหน่งแทน

0	1	2
3	4	5
6	7	8

เช่นจากรูปข้างต้นตำแหน่งที่ว่างคือตำแหน่ง 6 ดังนั้นตัวที่สามารถเลื่อนมาแทนได้คือ ตำแหน่ง 3 และ 7

An Edge

- จากเกมส์เรียงตัวเลข เราสามารถสร้างกฎการเลื่อน (rule หรือ dictionary) เป็นรูปแบบที่แน่นอนในแต่ละกรณีได้ โดยสร้างฟังก์ชันที่รับรู้ตำแหน่งช่องว่างแล้วคืนค่าเป็นตำแหน่งใหม่ที่อาจจะถูกเลื่อนได้ โดยดูจากตำแหน่งที่อยู่ติดกันในแนวนอนและแนวตั้งเป็นหลัก ดังนั้น
  - ตำแหน่งมุมจะมีตัวเลื่อนได้แค่ 2 ตำแหน่ง
  - ตำแหน่งชิดขอบตัวกลางมีตัวเลื่อนได้ 3 ตำแหน่ง
  - ตำแหน่งตรงกลางมีตัวเลื่อนได้ 4 ตำแหน่ง

	1	2
3	4	5
6	7	8

An implementation node

- สำหรับเกมส์เราให้แต่ละโหนดคือสถานะปัจจุบันของเกมส์ที่เปลี่ยนแปลงจากก่อนหน้า ซึ่งเราแทนค่าสถานะเป็นสตริงแล้วจึงใช้ string เก็บสถานะของเกมส์ เพื่อแทนเป็นโหนดได้เลย

```
#include <queue>
#include <list>
list<int> shiftDict(int pos);
string transition(string label, int pos);
bool notInPath(string str,list<string> statepath);
list<string> BFSWithGenerator(string startstate,string finalstate);
int main(){
  string result = "012345678";
  string input = "";
  cin >> input;
  list <string> rset = BFSWithGenerator(input,result);
  for (list<string>::iterator i=rset.begin(); i!=rset.end(); i++)
  {       cout << *i << endl;     }
  return 0;
}
```

An implementation: implicit edges

0	1	2
3	4	5
6	7	8

```
shiftDict[0] = [1,3]
shiftDict[1] = [0,2,4]
shiftDict[2] = [1,5]
shiftDict[3] = [0,4,6]
shiftDict[4] = [1,3,5,7]
shiftDict[5] = [2,4,8]
shiftDict[6] = [3,7]
shiftDict[7] = [4,6,8]
shiftDict[8] = [5,7]
```

```
list<int> shiftDict(int pos){
  list<int> sh;
  switch (pos){
  case 0: sh.push_back (1);
          sh.push_back (3);
          break;
  case 1: sh.push_back (0);
          sh.push_back (2);
          sh.push_back (4);
          break;
  case 2: sh.push_back (1);
          sh.push_back (5);
          break;
  case 3:
  :
  :
  }
}
```

An implementation: Search


```
list<string> BFSWithGenerator(string startstate,string finalstate){
  queue <list <string> > q;
  list <string>  initpath,tmppath,newpath;
  string laststate,newstate;
  list <int>     sft;
  initpath.push_back(startstate);
  q.push(initpath);
  while(!q.empty()){
    tmppath = q.front();
    q.pop();
    list<string>::reverse_iterator index = tmppath.rbegin();
    if (finalstate.compare(*index)==0) return tmppath;
    else laststate = *index;
    sft = shiftDict(laststate.find('0'));
    for (list<int>::iterator j=sft.begin(); j!=sft.end();j++{
      newstate = transition(laststate,*j);
      if (notInPath(newstate,tmppath)){
        newpath = tmppath;
        newpath.push_back(newstate);
        q.push(newpath);            }
     }
} //end while(!q.empty()) }
```

อย่าลืมฟังก์ชันตรวจสอบโหนดใหม่ไม่ย้อนวนกลับ

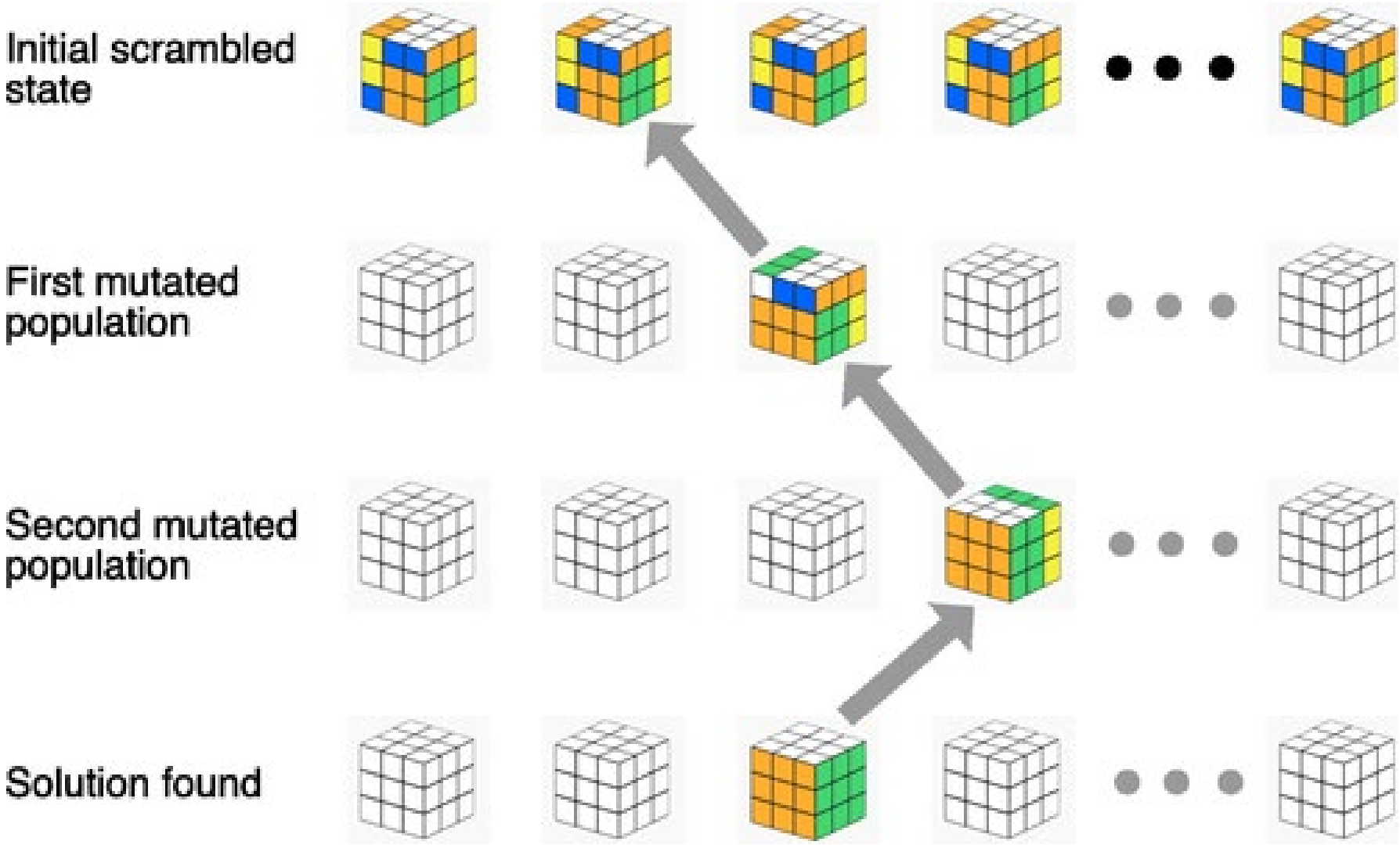
- จากเกมส์จะเห็นว่าช่องว่างสามารถมีตัวเลื่อนที่เลื่อนมาแทนช่องว่างได้มากกว่าหนึ่งตัวแต่จะมีตัวหนึ่งที่อาจจะเป็นตัวที่เพิ่งถูกเลื่อนเมื่อครั้งก่อนเราจึงต้องตรวจสอบว่าโหนดใหม่ ไม่ใช่โหนดที่เพิ่งเลื่อนมา

```
bool notInPath(string str, list<string> statepath){
for (list<string>::iterator i=statepath.begin(); i!=statepath.end();i++)
{
    if (str.compare(*i)==0) return false;
}
return true;
}
```

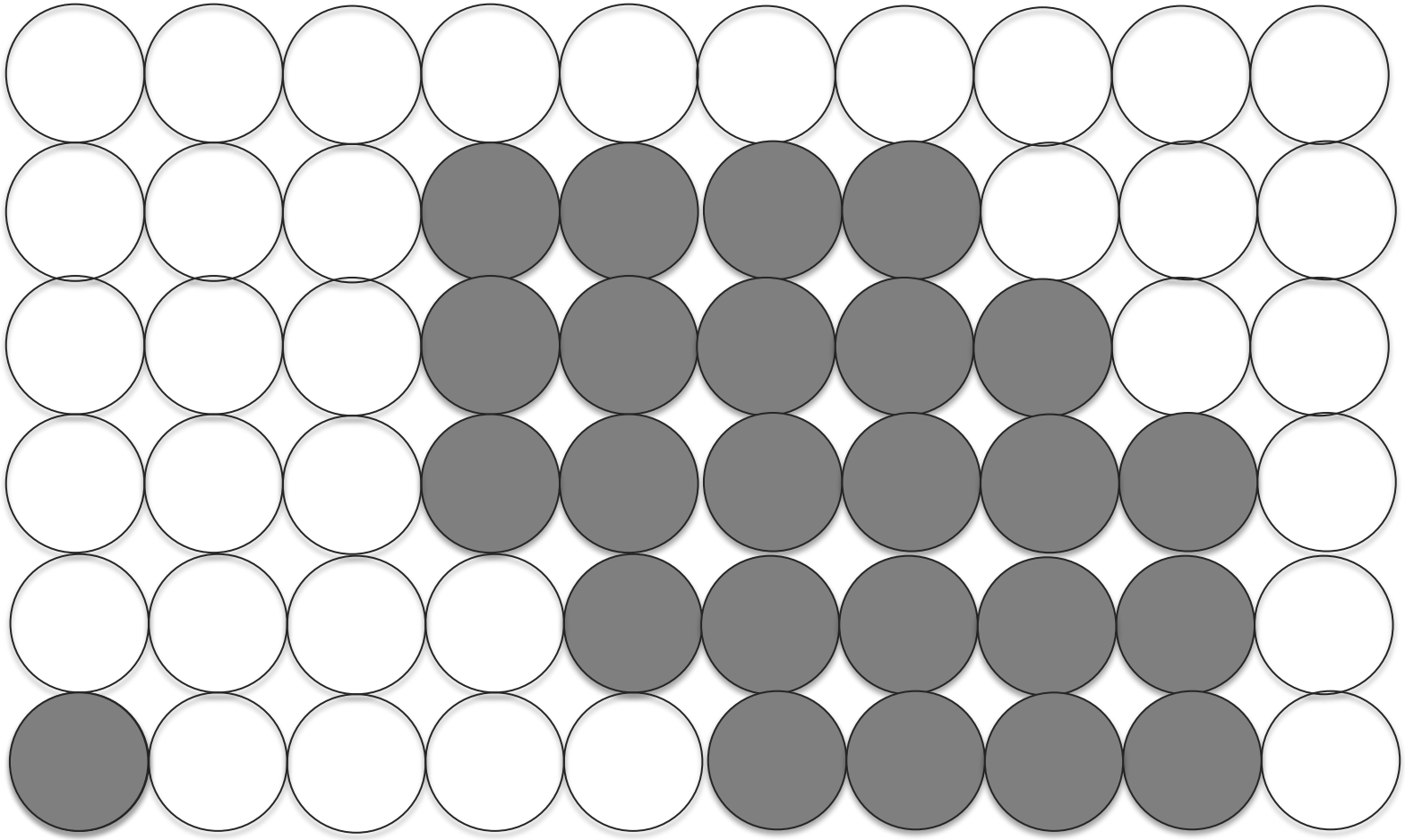
Search ด้วยวิธีอื่นหล่ะ

- นอกจาก Breadth first search แล้ว
- Depth First Search สามารถทำให้หาคำตอบได้ แต่อาจจะได้คำตอบที่นานเท่านั้นเอง
- จากโค้ดตัวอย่างเราสามารถเปลี่ยนจาก Queue เป็น Stack เพื่อให้ทำการสร้างต้นไม้แบบ Depth First Search

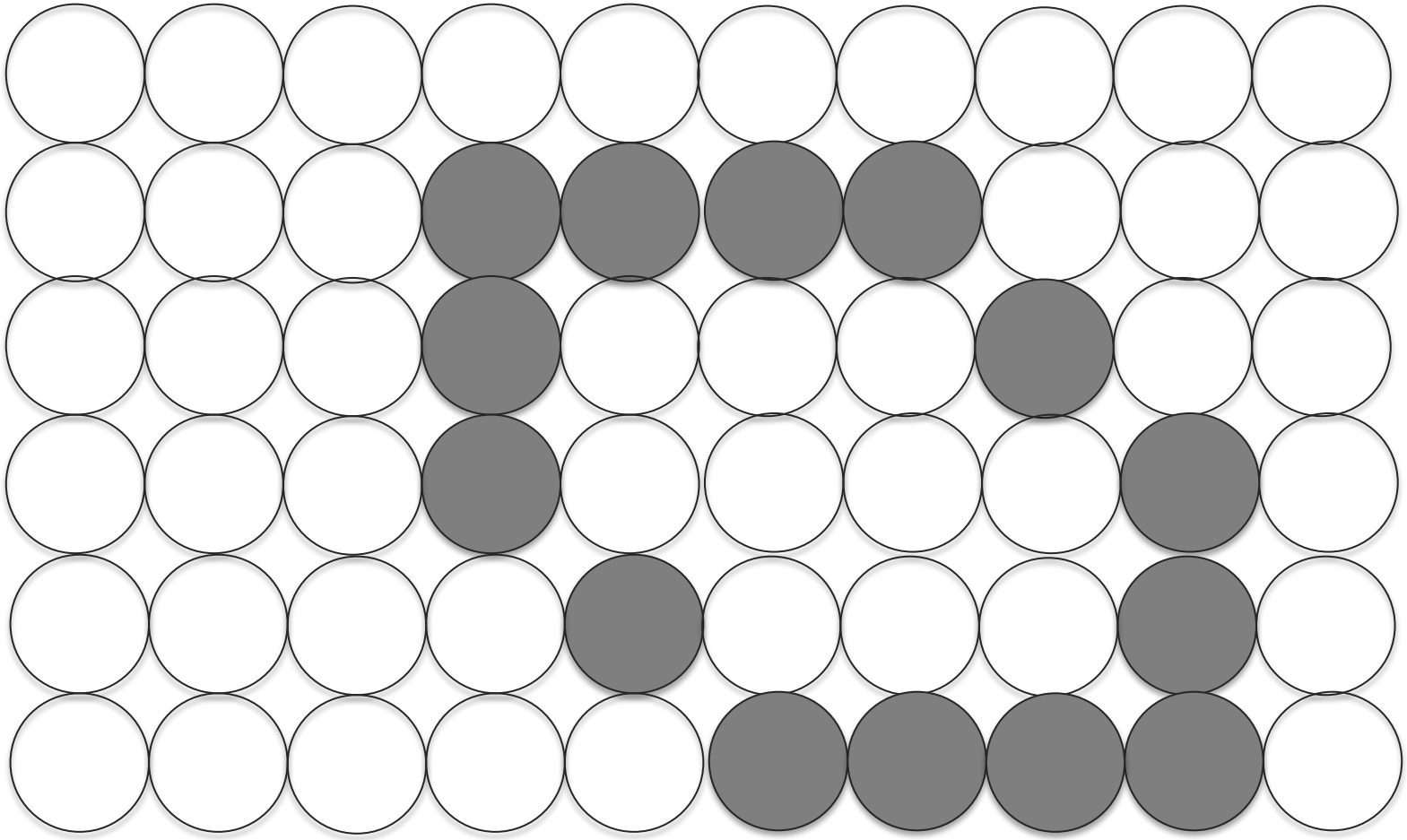
Rubik


Initial scrambled state
First mutated population
Second mutated population
Solution found

อยากเปลี่ยนสีจากสีเทาเป็นสีแดง



อยากระบายสีพื้นที่ในขอบให้หมด



Polygon Filling Algorithms

Highlight all the pixels inside the polygon. There are 2 approaches can be used:

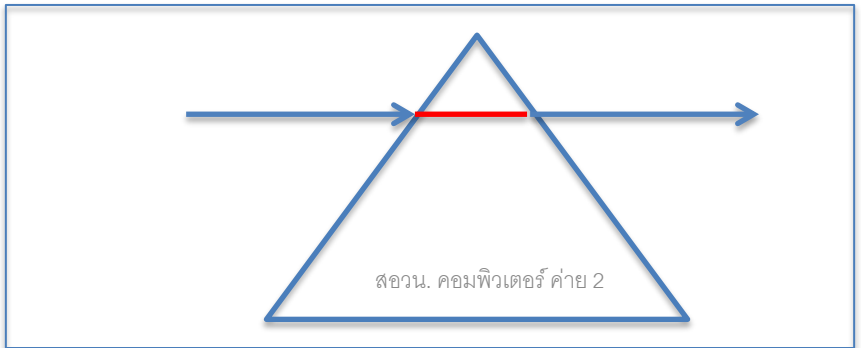
1. Scan Fill
2. Seed Fill (Boundary Fill, Flood Fill)

วิธีแรกที่เราใช้คือ scan line fill

- โดยเรากำหนดให้เส้นทุกเส้นเริ่มจากด้านซ้ายและเริ่มที่แถวบน จากนั้นก็ตรวจเช็คว่าแต่ละจุดที่ลากผ่านมีสีเดิมที่เราต้องการไหม ถ้าใช่ก็ทำการเปลี่ยนเป็นสีใหม่ แต่ถ้าไม่ใช่ไม่ต้องทำอะไร
- วิธีนี้จะต้องใช้การวนเพื่อตรวจสอบทีละจุดสี ดังนั้นจะทำให้ใช้เวลาในการทำงานเท่ากับจำนวนจุดสีที่มี เพราะต้องตรวจสอบหมด ไม่รู้ว่าจุดสีเป็นสีที่ต้องการเปลี่ยนหรือไม่

วิธีแรกที่เราใช้คือ scan line fill

- โดยเรากำหนดให้เส้นทุกเส้นเริ่มจากด้านซ้าย ซึ่งเป็นบริเวณที่อยู่นอกพื้นที่ เมื่อสแกนมาเจอขอบก็ให้เปลี่ยนสถานะว่าเราจะเริ่มระบายสี จากนั้นทำการเปลี่ยนค่าจุดสีเป็นสีใหม่จนกว่าจะเจอขอบอีกด้าน ทำเช่นนี้ในทุกๆ แถว ก็จะทำให้เติมสีได้เต็มพื้นที่
- วิธีนี้จะต้องใช้การวนเพื่อตรวจสอบทีละจุดสี ดังนั้นจะทำให้ใช้เวลาในการทำงานเท่ากับจำนวนจุดสีที่มี เพราะต้องตรวจสอบหมด ไม่รู้ว่าจุดสีเป็นสีที่ต้องการเปลี่ยนหรือไม่



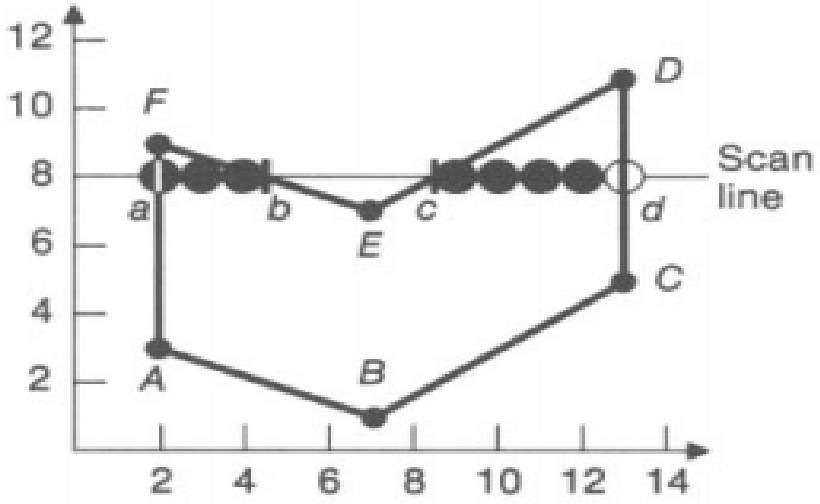
Scan Fill Algorithm

Start with max y and move to min y or vice versa

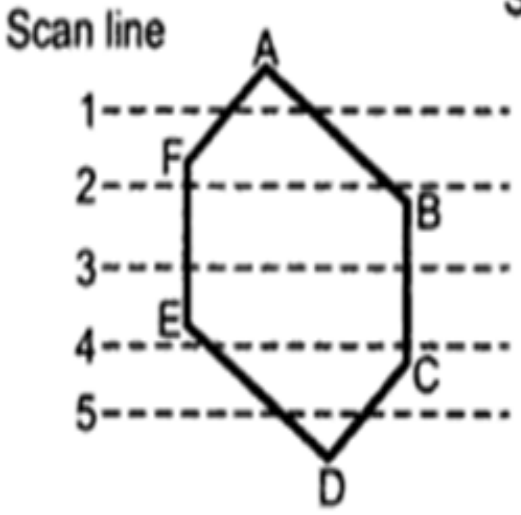
For each scan line:

- **Find the intersections of the scan line with all edges of the polygon.**
- **Sort the intersections by increasing x coordinate.**
- **Fill in all pixels between pairs of intersections**

For scan line number 8 the sorted list of x-coordinates is (2, 4, 9, 13)
Therefore draw line b/w (2,4) & (9,13)



$x_{i+1}=x_i - 1/m$ where m is the slope of edge



Scan line	1	2	3	4	5	
	B → AF	AF	AF	AF	AF	..... y_{max}
	E → AB	B → AB	AB	AB	AB	Sorted by values of y
	FE	E → FE	B → FE	FE	FE	
	BC	BC	E → BC	B → BC	BC	
	ED	ED	ED	E → ED	B → ED	
	CD	CD	CD	CD	E → CD	..... y_{min}

Note : B : indicates beginning of active edge list
E : indicates end of active edge list

Flood Fill Algorithm

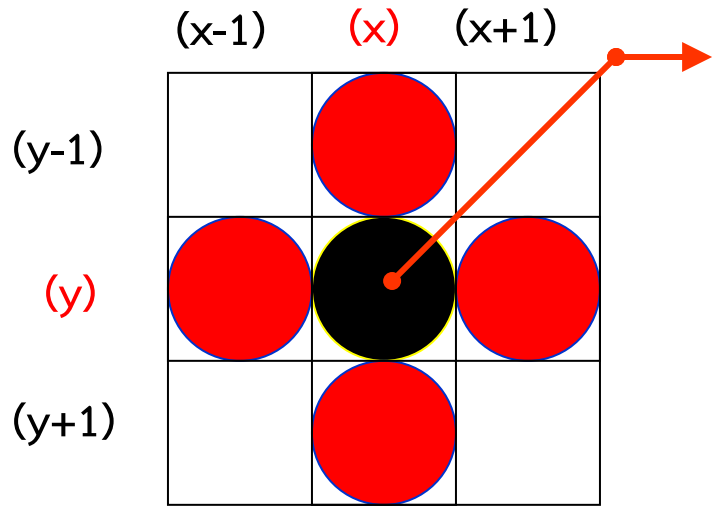
- เป็นอัลกอริทึมที่ใช้ในวงการ Digitam Image Processing
- ใช้ในการแก้ปัญหาการแทนค่าสีเดิมด้วยสีใหม่ หรือระบายสีภายในขอบเขต ตัวอย่างเห็นได้ชัดจาก อุปกรณ์กระป๋องสีของโปรแกรมโฟโตชอป เราสามารถเทสีใหม่ทับสีเดิม เพียงแต่แต้มลงไปแค่จุดเดียว
- หลักการง่ายๆ คือเราจะทำการแทนค่าสีและจุดสีเพื่อนบ้านให้เป็นสีที่ต้องการ และทำการเปลี่ยนสีไปเรื่อยๆ ตราบจนหมดพื้นที่ที่ต้องการทา

จุดสีเพื่อนบ้าน

- จุดสีเพื่อนบ้านคือจุดที่ติดกับจุดสีที่เราให้ความสนใจ
- แต่ละจุดสีมีระยะห่างแค่ 1 เท่านั้น คือต้องมีตำแหน่งอยู่ชิดกัน
- จุดสีเพื่อนบ้านบางจุดอาจอยู่นอกขอบเขตของภาพได้ ถ้าจุดสีที่เราสนใจอยู่ที่ขอบภาพ
- ในการทำ Flood Fill Algorithm เราจะใช้การตรวจสอบจุดสีเพื่อนบ้าน 2 แบบคือ
  - ตรวจสอบสี่จุดรอบจุดสี และ
  - ตรวจสอบแปดจุดรอบจุดสี

การตรวจสอบจุดสีเพื่อนบ้าน 4 จุด

- เราจะพิจารณาจุดสีที่ล้อมรอบจุดสีที่เราสนใจเพียงแค่ 4 จุดเท่านั้น
- ซึ่งเป็นจุดสีที่อยู่แนวนอนและแนวตั้ง



(x,y) เป็นพิกเซลที่เราสนใจ

(x,y-1) เป็นพิกเซลด้านบน

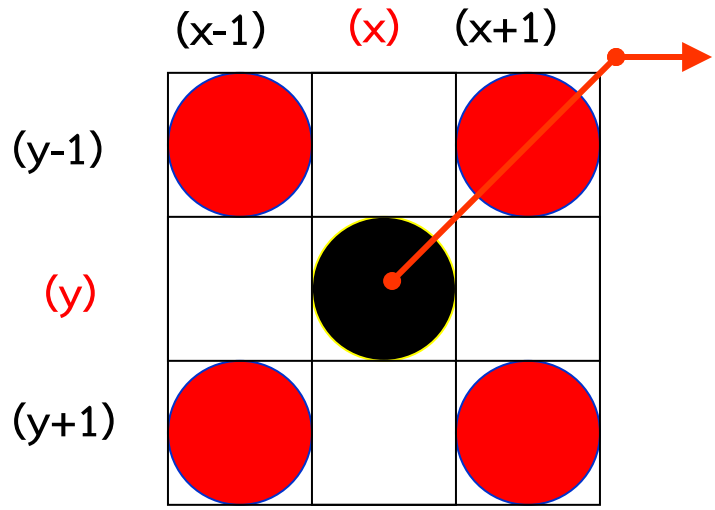
(x-1,y) เป็นพิกเซลด้านซ้าย

(x+1,y) ตำแหน่งจุดสีด้านขวา

(x,y+1) ตำแหน่งจุดสีด้านล่าง

การตรวจสอบจุดสีเพื่อนบ้าน 4 จุด (แนวทะแยง)

- เราจะพิจารณาจุดสีที่ล้อมรอบจุดสีที่เราสนใจเพียงแค่ 4 จุดเท่านั้น
- ซึ่งเป็นจุดสีที่อยู่แนวทะแยงมุม



(x,y) เป็นพิกเซลที่เราสนใจ

(x-1,y-1) เป็นพิกเซลด้านบนซ้าย

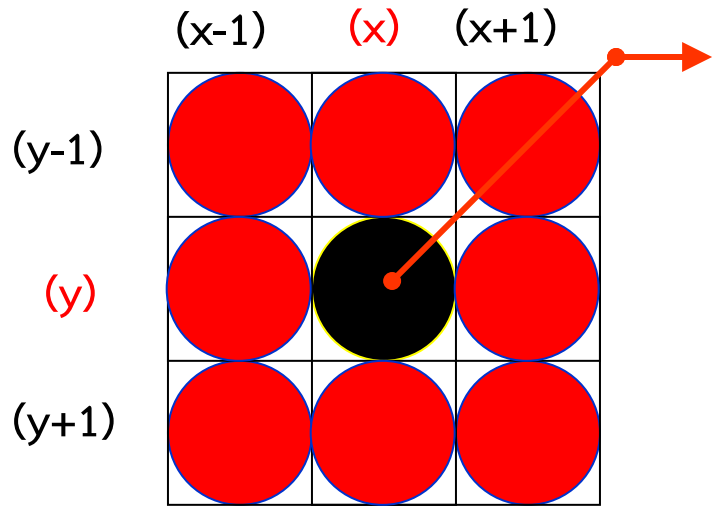
(x+1,y-1) เป็นพิกเซลด้านบนขวา

(x-1,y+1) ตำแหน่งจุดสีด้านล่างซ้าย

(x+1,y+1) ตำแหน่งจุดสีด้านล่างขวา

การตรวจสอบจุดสีเพื่อนบ้าน 8 จุด

- เราจะพิจารณาจุดสีที่ล้อมรอบจุดสีที่เราสนใจทั้ง 8 จุด
- ซึ่งคือจุดสีทั้งแนวนอน แนวตั้ง และจุดสีที่อยู่แนวทะแยงมุม



(x,y) เป็นพิกเซลที่เราสนใจ

(x-1,y-1) (x,y-1) *(x+1,y-1)*

(x-1,y) *(x+1,y)*

(x-1,y+1) *(x,y+1)* *(x+1,y+1)*

Flood Fill Algorithm (ทาสีใหม่ทับสีเก่า)

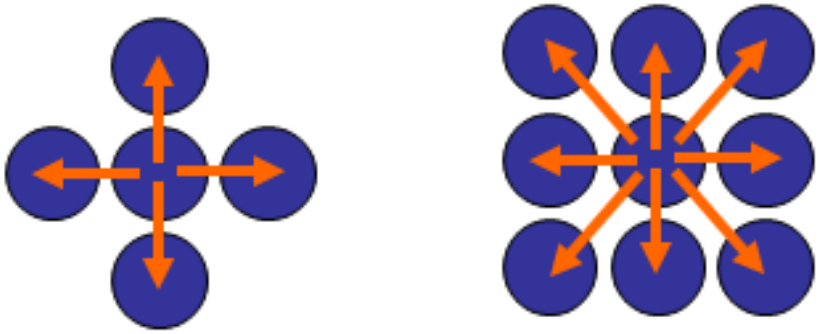

```
void fill( int x, int y, int interiorcolor, int newcolor ) {
   if ( get_pixel( x, y ) == interiorcolor ) {
      put_pixel( x, y, newcolor );
      fill( x -1, y, interiorcolor, newcolor );
      fill( x+1, y, interiorcolor, newcolor );
      fill( x, y -1, interiorcolor, newcolor );
      fill( x, y+1, interiorcolor, newcolor );
  }
}
```

Flood Fill Algorithm (ทาสีในขอบ)


```
void boundaryFill(int x, int y, int fillColor, int borderColor) {
  int interiorColor;
  getPixel(x,y,interiorColor);
  if ((interiorColor!=borderColor)&&(interiorColor!=fillColor)) {
    setPixel(x,y,fillColor);
    boundaryFill(x+1,y,fillColor,borderColor);
    boundaryFill(x-1,y,fillColor,borderColor);
    boundaryFill(x,y+1,fillColor,borderColor);
    boundaryFill(x,y-1,fillColor,borderColor);
  }
}
```

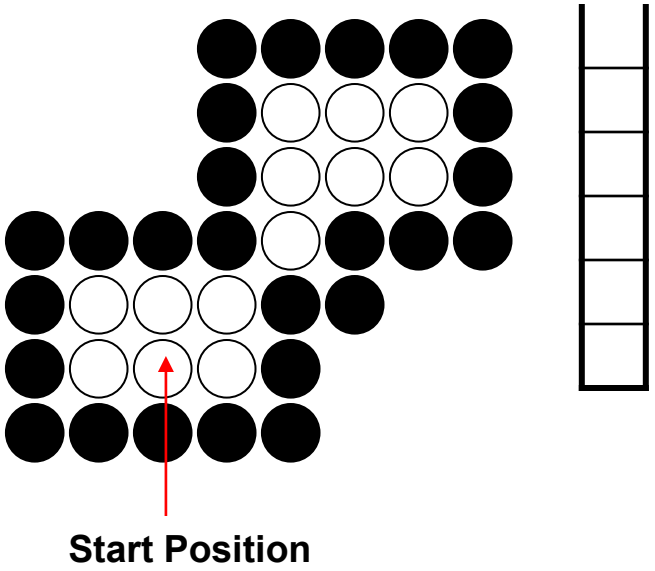
Boundary Fill Algorithm /Flood Fill algorithm

The boundary fill algorithm/ flood fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

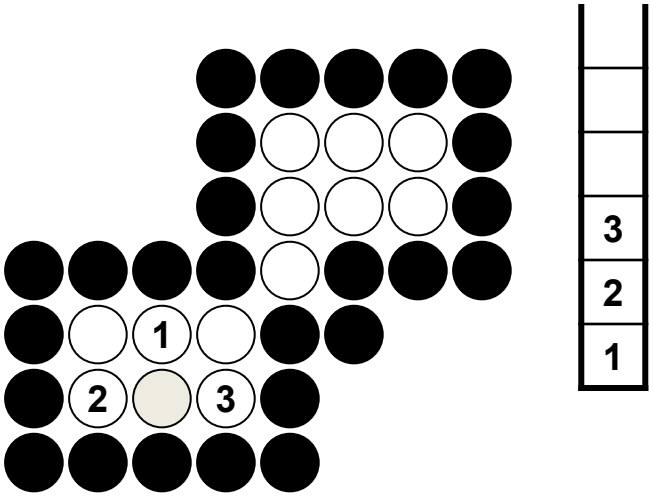


4-connected 8-connected

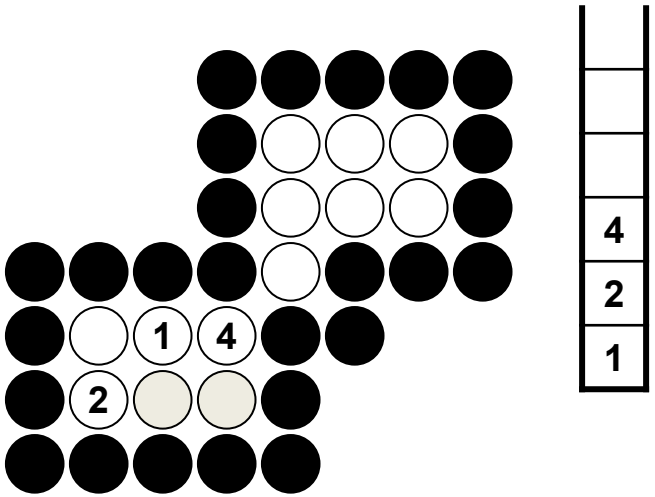
4-connected (Example)



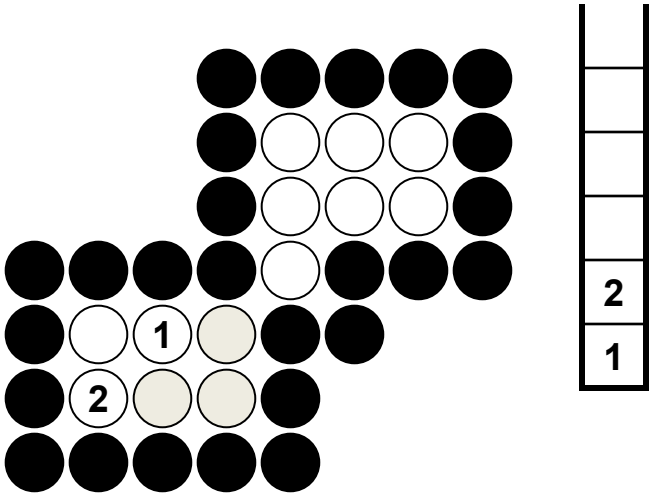
4-connected (Example)



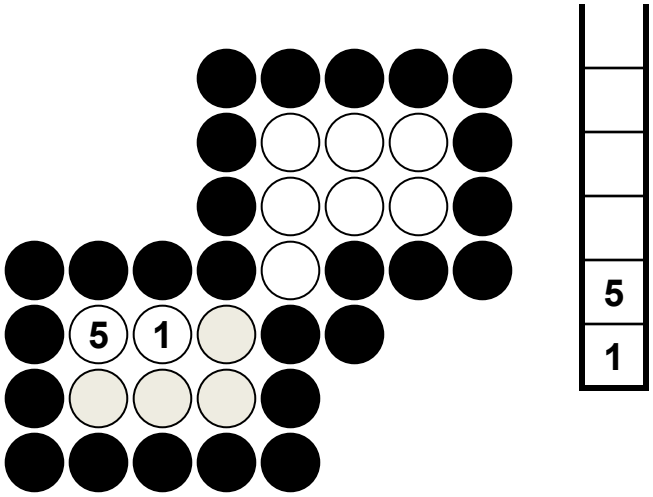
4-connected (Example)



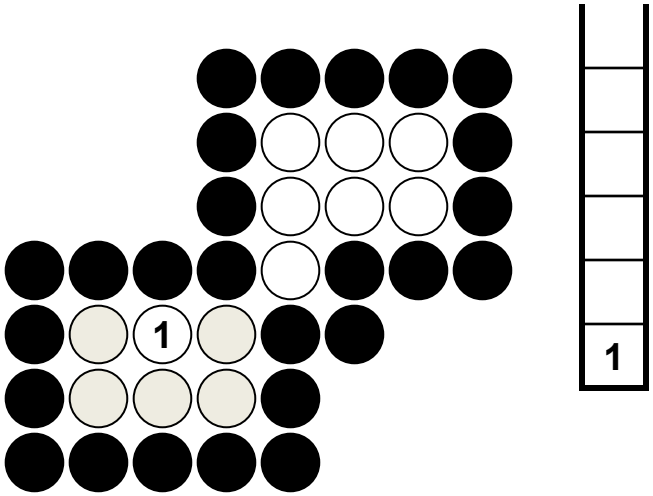
4-connected (Example)



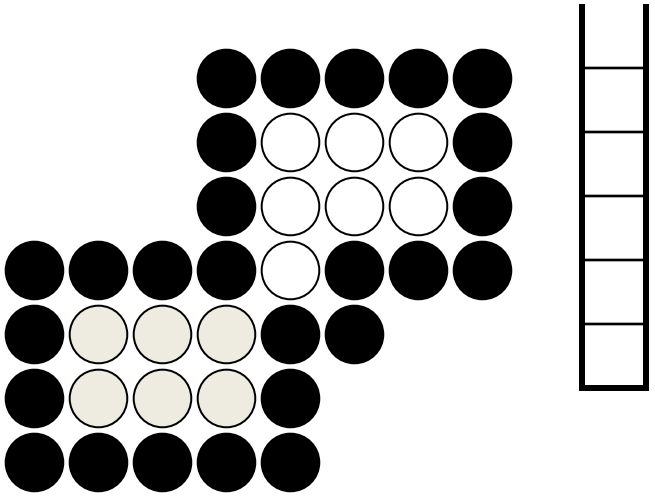
4-connected (Example)



4-connected (Example)

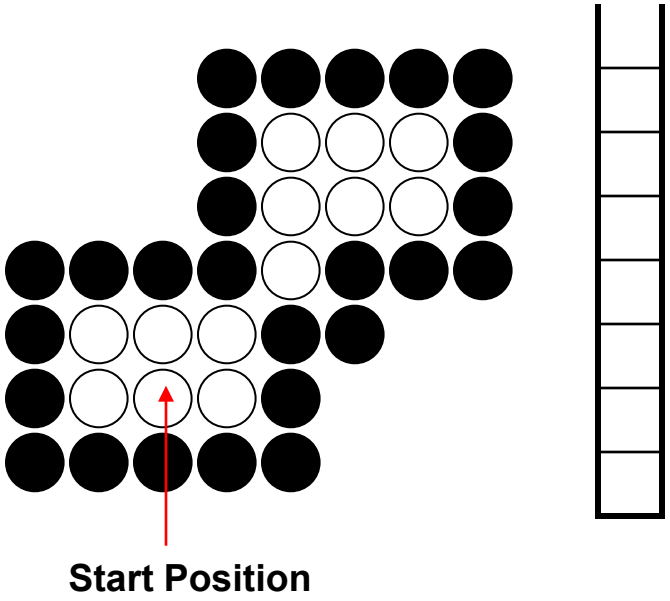


4-connected (Example)

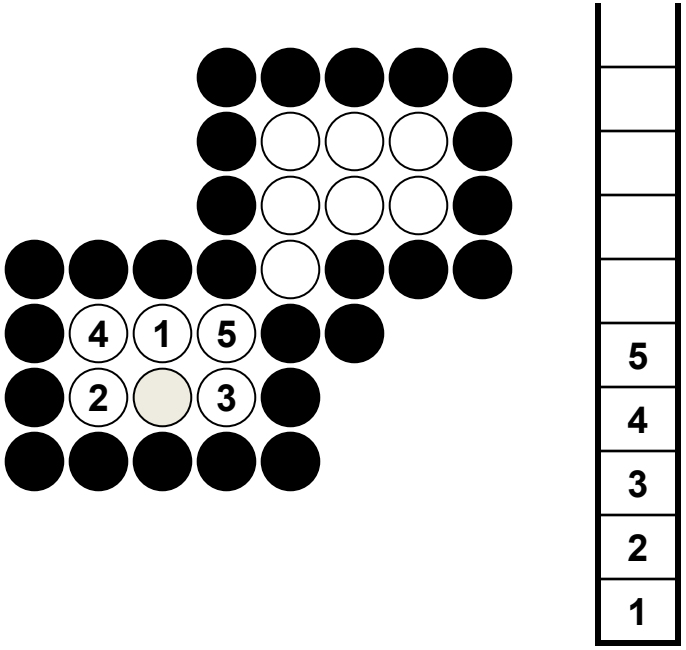


Some region remains unfilled

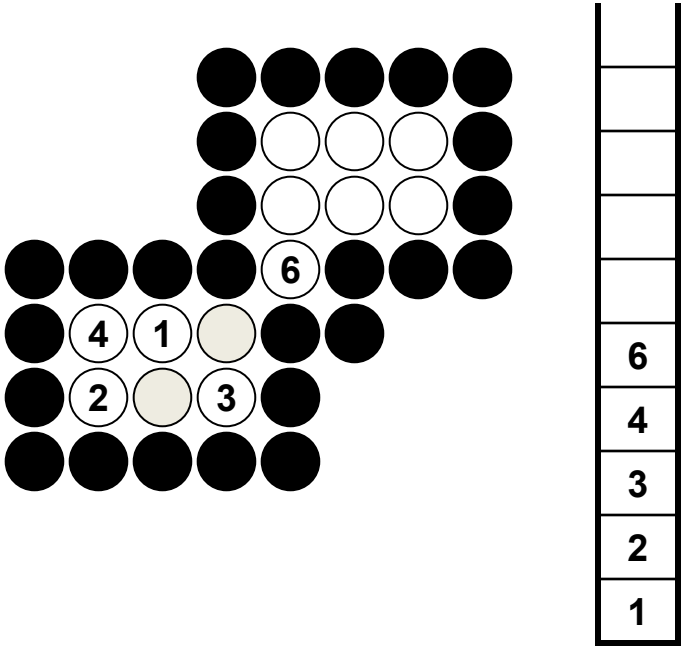
8-connected (Example)



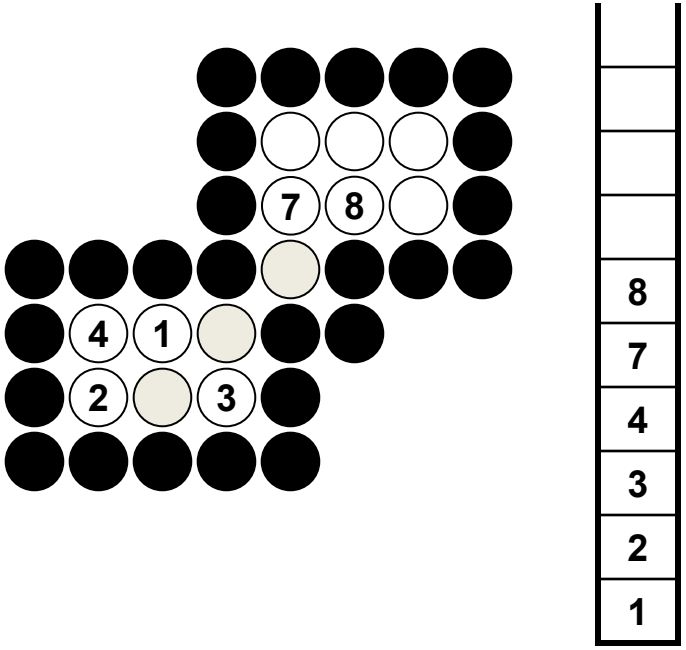
8-connected (Example)



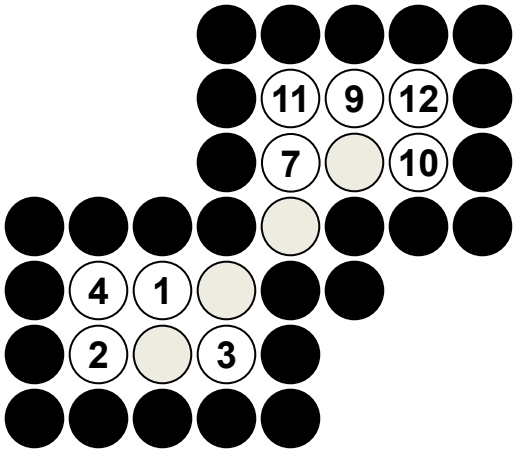
8-connected (Example)

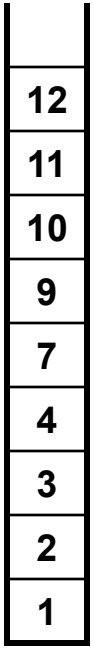


8-connected (Example)

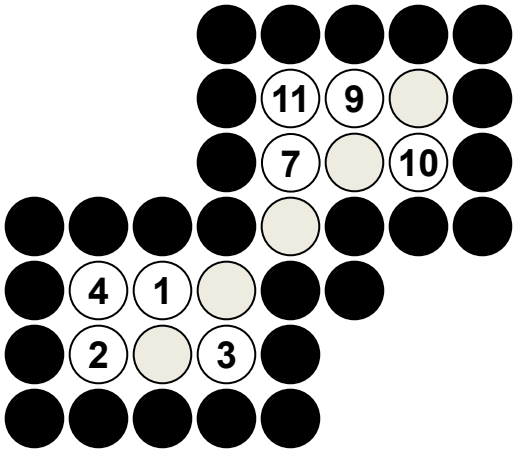


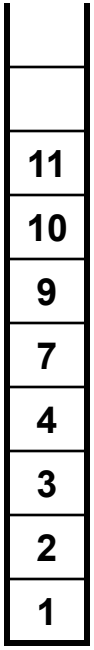
8-connected (Example)



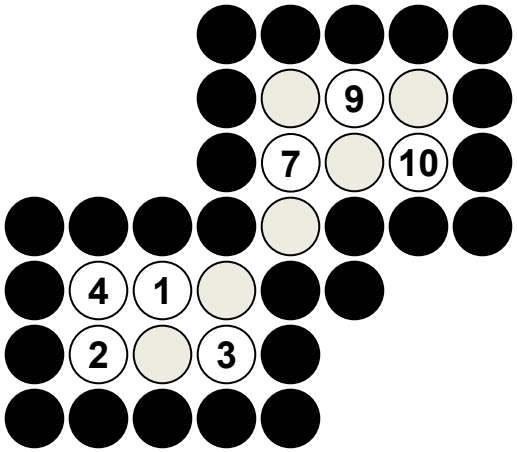


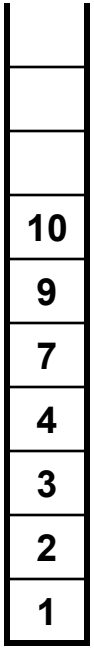
8-connected (Example)



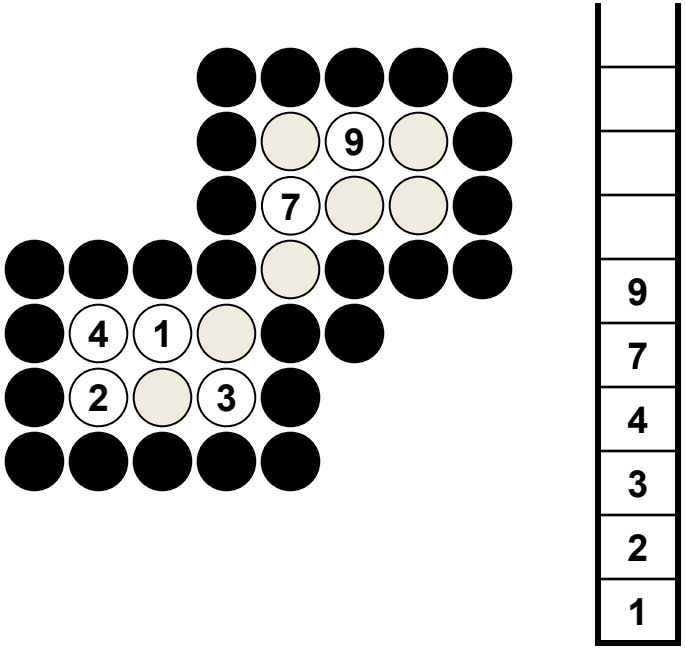


8-connected (Example)

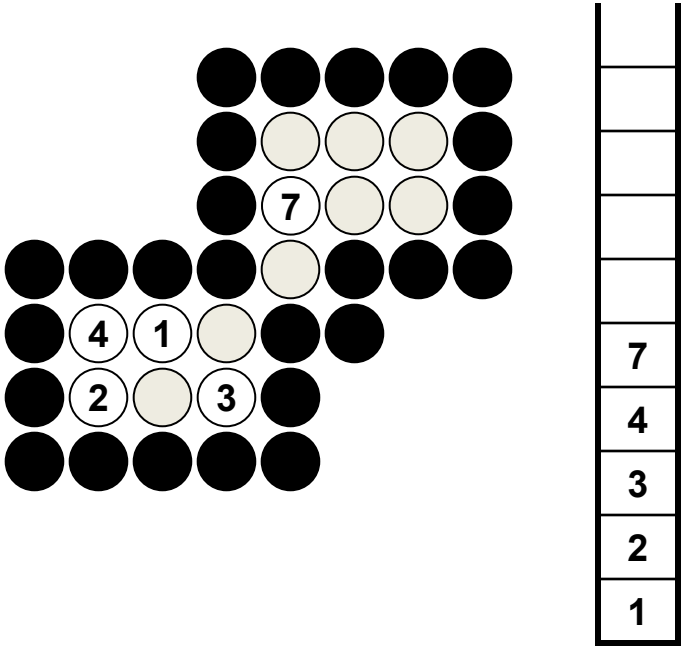




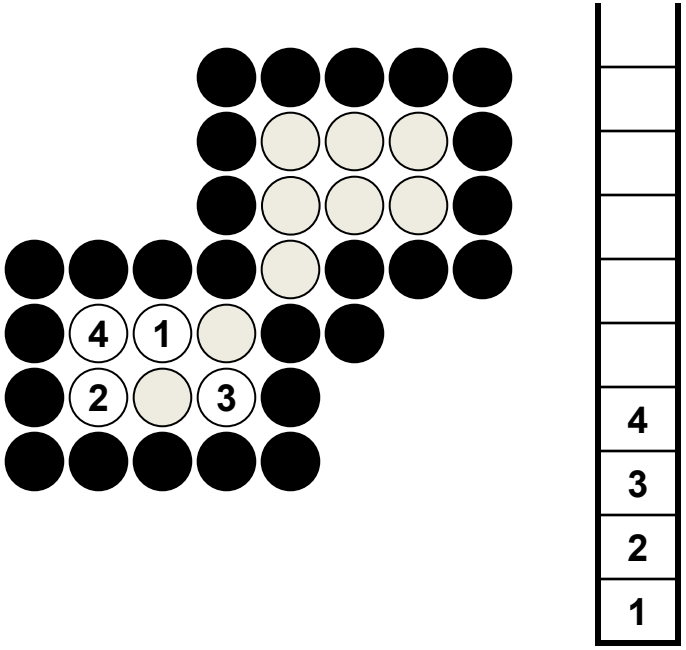
8-connected (Example)



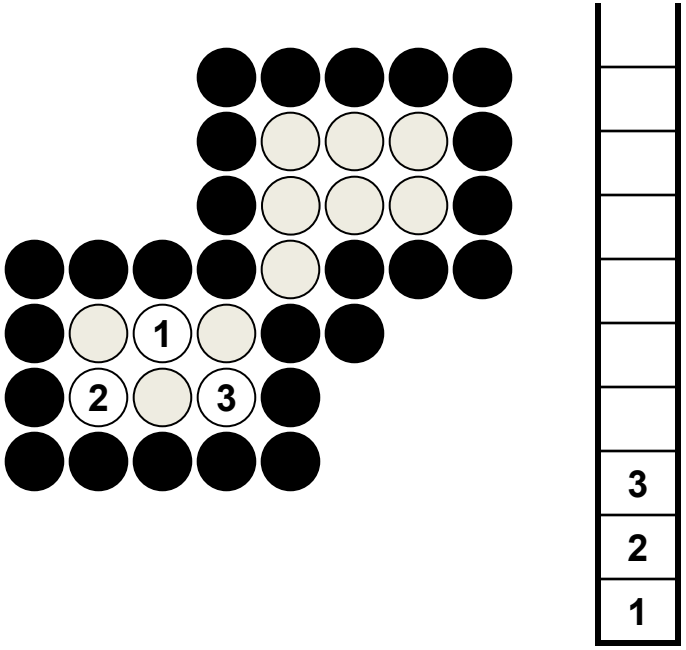
8-connected (Example)



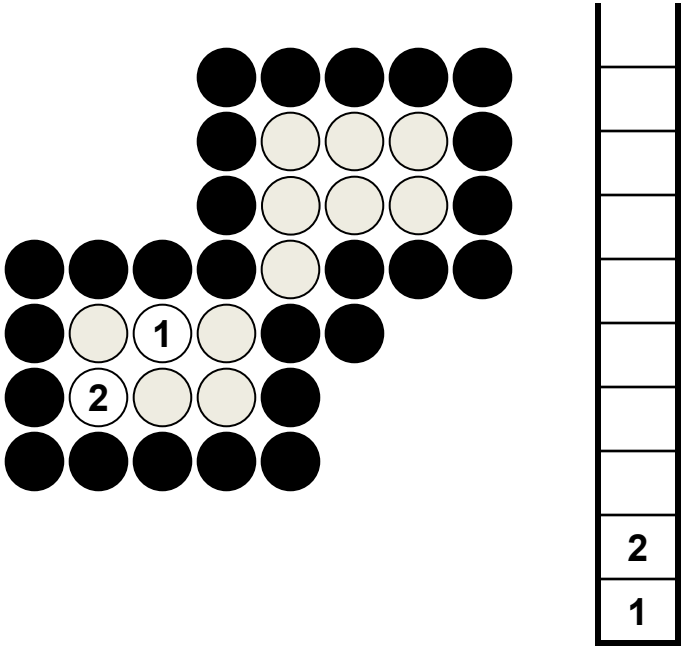
8-connected (Example)



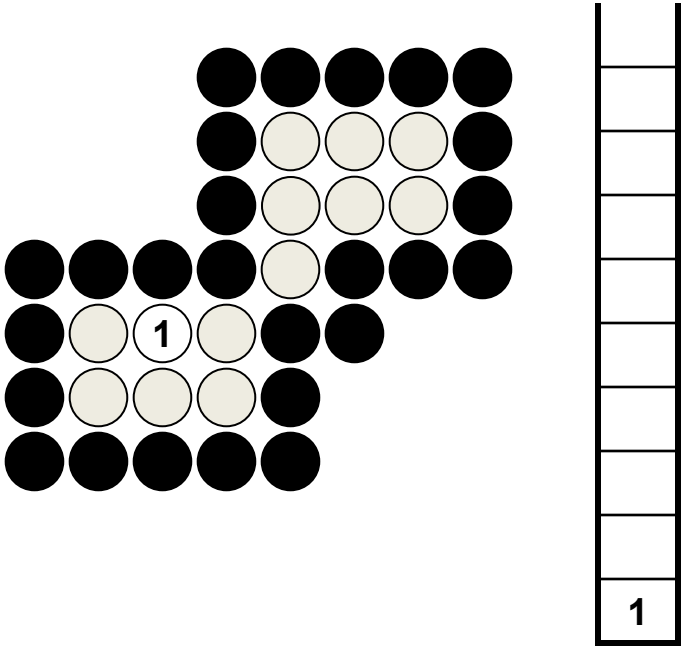
8-connected (Example)



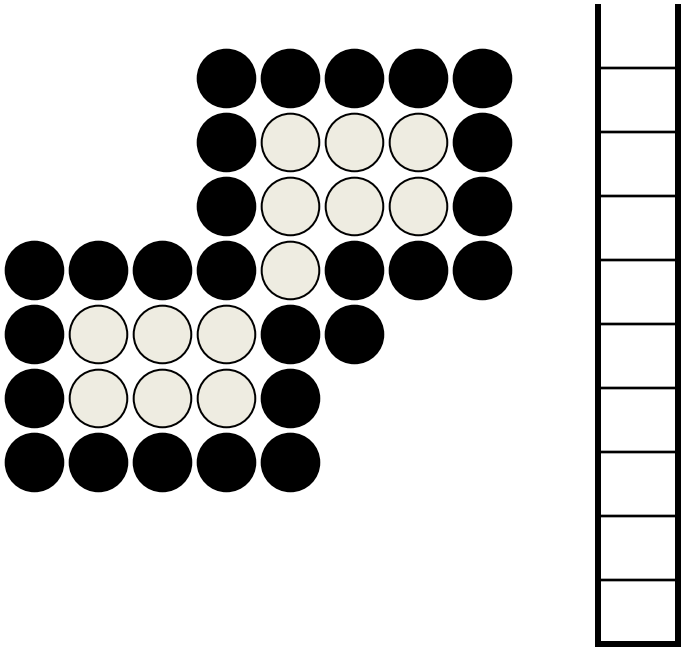
8-connected (Example)



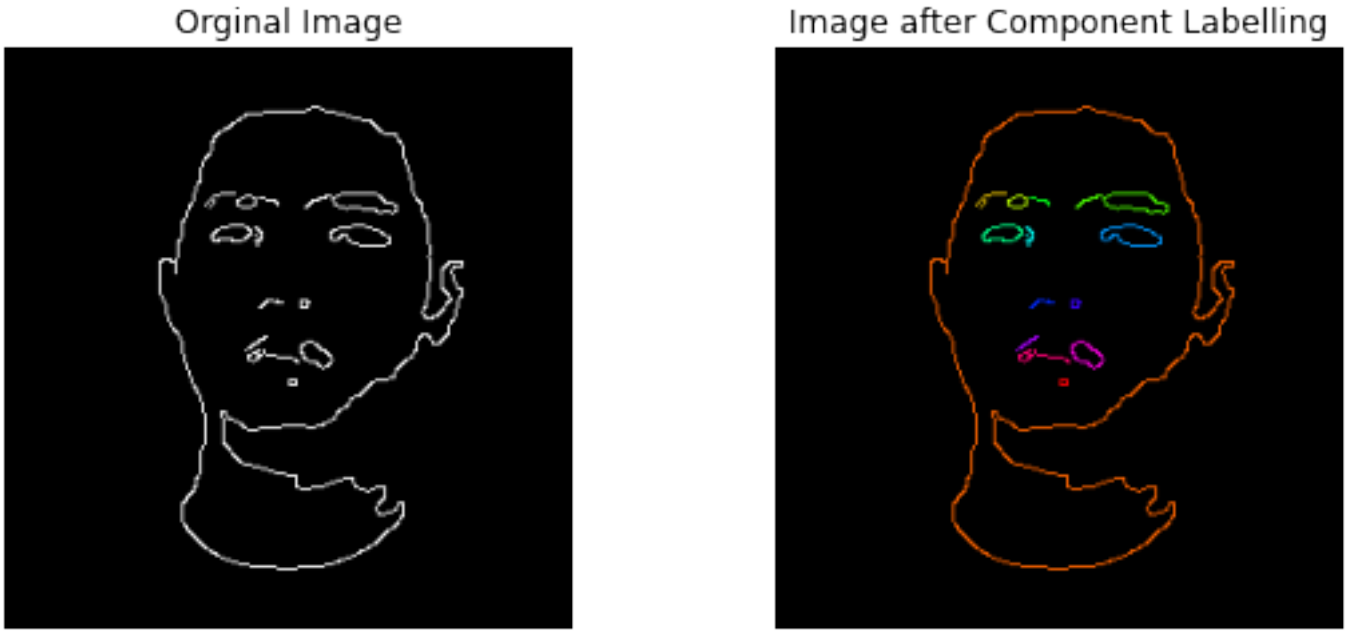
8-connected (Example)



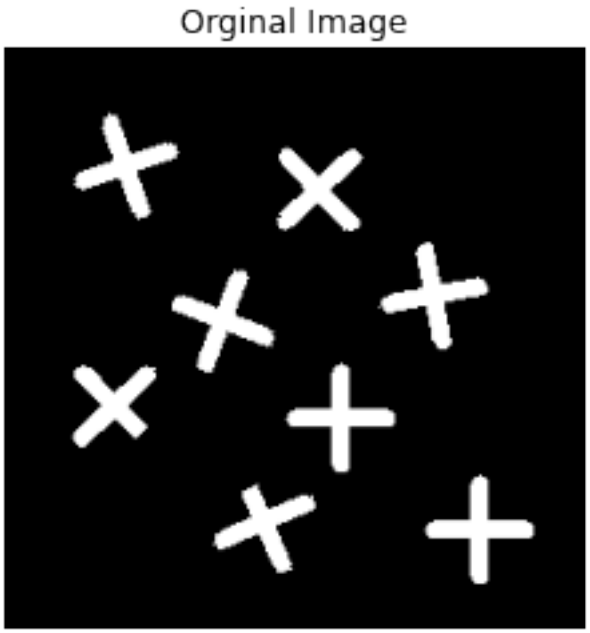
8-connected (Example)

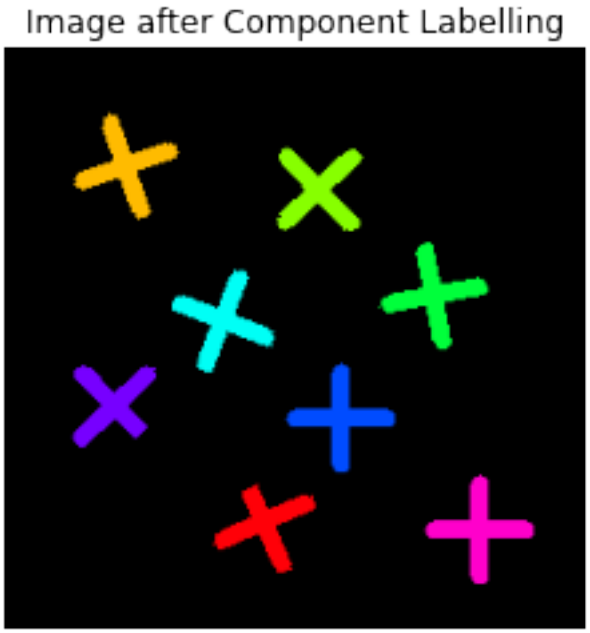


Connected Component Analysis

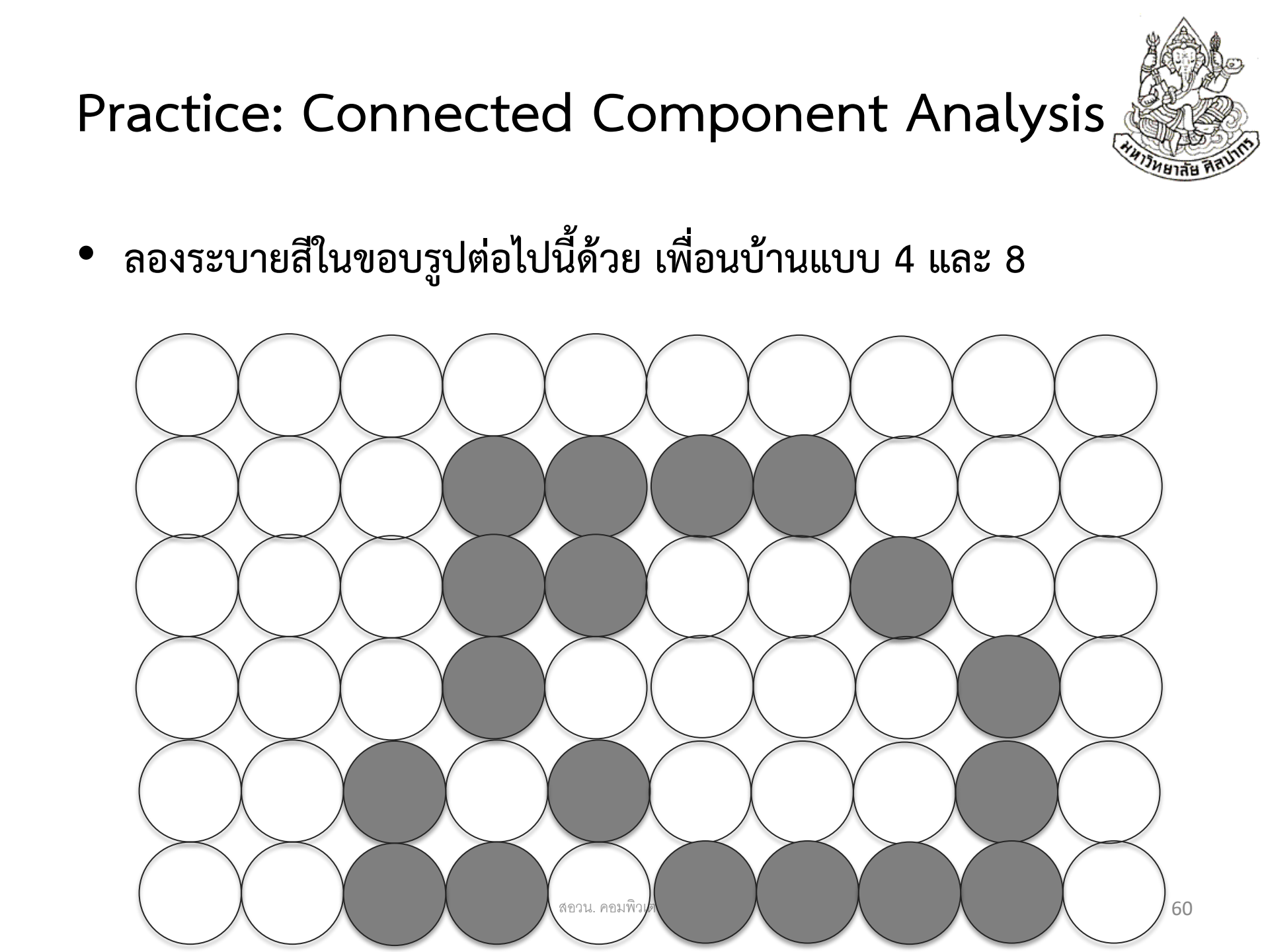


Connected Component Analysis





Practice: Connected Component Analysis



- ลองระบายสีในขอบรูปต่อไปนี้ด้วย เพื่อนบ้านแบบ 4 และ 8

ให้กราฟเราแทนด้วยชุดข้อมูลดังนี้

(A,B,4) (A,C,2) (A,E,3) (B,D,5) (C,D,1) (C,E,6) (C,F,3) (D,F,6) (E,F,2)

1. เริ่มด้วยโหนดใดๆ สมมุติให้เลือก A เป็นโหนดแรก
2. หาโหนดถัดไป เพิ่มในเซตโดยดูจากเส้นเชื่อม (edge) ที่น้อยที่สุดที่ออกจากโหนดที่อยู่ในเซตที่เราเลือก

	A	B	C	D	E	F
{A}	-	4a	2a	∞	3a	∞
{A,C}	-	4a	-	1c	3a	3c
{A,C,D}	-	4a	-	-	3a	3c
{A,C,D,E}	-	4a	-	-	-	2e
{A,C,D,E,F}	-	4a	-	-	-	-
{A,C,D,E,F,B}	-	-	-	-	-	-


มหาวิทยาลัย ศิลปากร

- กราฟบางอย่างมีลักษณะพิเศษที่จะทำให้โหนดหนึ่งเชื่อมไปหาโหนดอีกโหนดหนึ่งด้วยกฎที่แน่นอน และไม่สามารถไปหาโหนดอื่นๆ ได้ตามใจ
- โหนด แทน สถานะของระบบ
- Edge แทน สิ่งที่เปลี่ยนสถานะของระบบ
- ลำดับของ action เพื่อหาผลลัพธ์