CPL Exam 2

TAs

Greatest Common Divisor

Problem

Description

You are given an array of integers, please find the greatest common divisor (最大公因數) of these integers.

Input

The first line is an integer `N` as the number of the integers in the array.

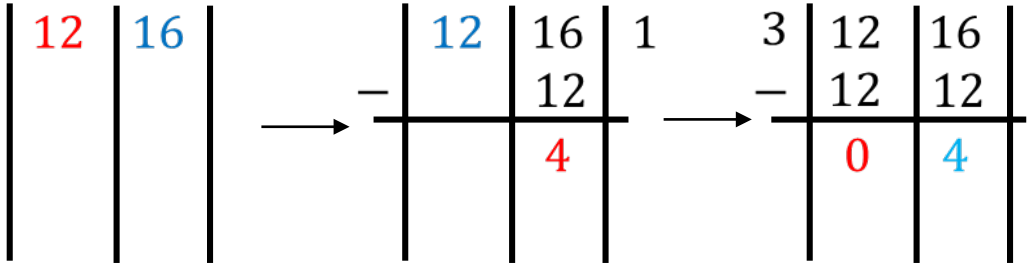
The second line contains `N` integers that are the elements of the array.

Output

Please output an integer that is the greatest common divisor
Remember to add "\n"

Solution

- Think about the Euclidean Algorithm(輾轉相除法)
  - EX: the greatest common divisor of 12 and 16




```
while(x % y != 0){
    int temp = y;
    y = x % y;
    x = temp;
}
gcd = y;
```

  - Notice the red number(smaller) and blue(larger) number in each iteration
    - In each iteration
      - Calculate blue number mod red number
      - The red number is the new blue number in the next iteration
      - Stop when the red number becomes 0
- Apply the Euclidean Algorithm to every element in the array
  - Need to determine the larger number and smaller number.

Overall

```cpp
#include <iostream>

using namespace std;

int GCD(int array[], int N){
    int gcd = array[0];
    for( int i = 1; i < N; i++ ){
        int x, y;
        x = max(gcd, array[i]);
        y = min(gcd, array[i]);
        while(x % y != 0){
            int temp = y;
            y = x % y;
            x = temp;
        }
        gcd = y;

    }
    return gcd;
}

int main(){
    int N;

    cin >> N;
    int array[N];

    for( int i = 0; i < N; i++ )
        cin >> array[i];

    for( int i = 0; i < N; i++ )
        cout << array[i] << " " ;
    cout << endl;

    cout << GCD(array, N) << endl;
    return 0;
}
```

Union Area

Problem

Description

You are given 3 rectangles, please determine the union area (聯集的面積) of these 3 rectangles.

Please follow the template below and complete the code. There are some functions to help you design the union_area() function. It is okay to design your own functions if you wish. You only need to submit th e template part.

- area(): calculate the area of given rectangle
- intersect(): calculate the intersection area of given 2 or 3 rectangles. The intersection part should also be a rectangle, too.
- union_area(): calculate the union area of given rectangles.

Input

3 lines of input. In each line, there are four integers

1. the x coordinate of the left-down position
2. the y coordinate of the left-down position
3. the x coordinate of the right-up position
4. the y coordinate of the right-up position

(每一行都給定左下角座標及右上角座標)

<Note> Each rectangle has length and width less than 20000

Output

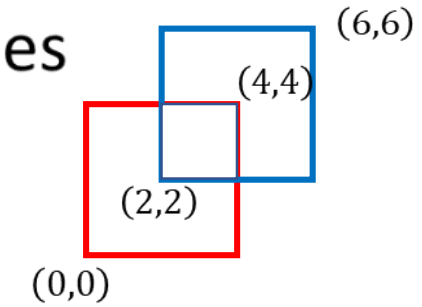
The union area of these three rectangles.

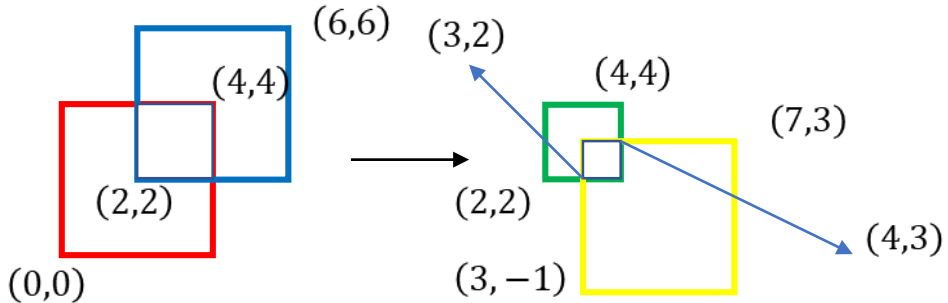
Solution

- area(rectangle rec)
  - Calculate area
- intersect(rectangle rec1, rectangle rec2)
  - Find the intersection of rec1 and rec2
- intersect(rectangle rec1, rectangle rec2, rectangle rec3)
  - Find the intersection of rec1, rec2, and rec3
- union_area(rectangle rec[], int N)
  - Calculate the overall union area

Solution: intersect

- Observe the intersection part of two rectangles
  - EX:
    - rec1: left down(0,0), right up(4,4)
    - rec2: left down(2,2), right up(6,6)
    - The intersection is another rectangle: left down(1,1) right up(2, 2)
      - The left down point is $(\max(rec1.leftdown.x, rec2.leftdown.x), \max(rec1.leftdown.y, rec2.leftdown.y))$
      - The right up point is $(min(rec1.rightup.x, rec2.rightup.x), min(rec1.rightup.y, rec2.rightup.y))$
    - What if add another rectangle rec3: left down(3,-1) right up(7,3)?
      - Just use intersection(rec1, rec2) to intersect rec3





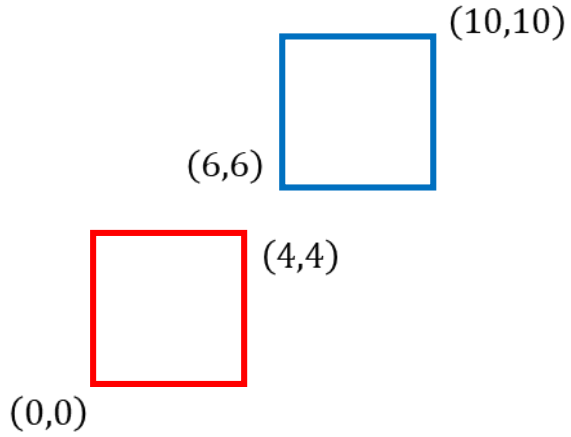
Solution: intersect

```
21  int intersect(rectangle rec1, rectangle rec2){
22      rectangle intersection = rec1;
23      if(rec2.left_down.x > intersection.left_down.x)
24          intersection.left_down.x = rec2.left_down.x;
25      if(rec2.right_up.x < intersection.right_up.x)
26          intersection.right_up.x = rec2.right_up.x;
27      if(rec2.left_down.y > intersection.left_down.y)
28          intersection.left_down.y = rec2.left_down.y;
29      if(rec2.right_up.y < intersection.right_up.y)
30          intersection.right_up.y = rec2.right_up.y;
31      return area(intersection);
32  }
```

```
34  int intersect(rectangle rec1, rectangle rec2, rectangle rec3){
35      struct rectangle intersection = rec1;
36      if(rec2.left_down.x > intersection.left_down.x)
37          intersection.left_down.x = rec2.left_down.x;
38      if(rec2.right_up.x < intersection.right_up.x)
39          intersection.right_up.x = rec2.right_up.x;
40      if(rec2.left_down.y > intersection.left_down.y)
41          intersection.left_down.y = rec2.left_down.y;
42      if(rec2.right_up.y < intersection.right_up.y)
43          intersection.right_up.y = rec2.right_up.y;
44
45      if(rec3.left_down.x > intersection.left_down.x)
46          intersection.left_down.x = rec3.left_down.x;
47      if(rec3.right_up.x < intersection.right_up.x)
48          intersection.right_up.x = rec3.right_up.x;
49      if(rec3.left_down.y > intersection.left_down.y)
50          intersection.left_down.y = rec3.left_down.y;
51      if(rec3.right_up.y < intersection.right_up.y)
52          intersection.right_up.y = rec3.right_up.y;
53
54      return area(intersection);
55  }
```

Solution

- If two rectangles don’t intersect to each other
  - The positions of intersection rectangle would not follow the left-down-right-up from
  - EX:
    - rec1: left down(0,0), right up(4,4)
    - rec2: left down(6,6), right up(10,10)
    - The intersection is another rectangle: left down(6,6) right up(4, 4)
      - Not comply the form



Solution: area

- Need to determine the input rectangle is a valid rectangle
  - Empty intersection

```
15  int area(rectangle rec){
16      if( rec.right_up.x < rec.left_down.x || rec.right_up.y < rec.left_down.y )
17          return 0;
18      return (rec.right_up.x - rec.left_down.x) * (rec.right_up.y - rec.left_down.y);
19  }
```

Solution: union

- Union law of 3 elements:
  - $A + B + C - A \cap B - B \cap C - A \cap C + A \cap B \cap C$

```
57  int union_area(rectangle rec[], int N){
58      int A, B, C, AB, BC, AC, ABC;
59      A = area(rec[0]);
60      B = area(rec[1]);
61      C = area(rec[2]);
62
63      AB = intersect(rec[0], rec[1]);
64      BC = intersect(rec[1], rec[2]);
65      AC = intersect(rec[0], rec[2]);
66
67      ABC = intersect(rec[0], rec[1], rec[2]);
68
69      return A + B + C - AB - BC - AC + ABC;
70  }
```

Overall

```cpp
#include <iostream>

using namespace std;

struct pos{
    int x;
    int y;
};

struct rectangle{
    pos left_down;
    pos right_up;
};

int area(rectangle rec){
    if( rec.right_up.x < rec.left_down.x || rec.right_up.y < rec.left_down.y )
        return 0;
    return (rec.right_up.x - rec.left_down.x) * (rec.right_up.y - rec.left_down.y);
}

int intersect(rectangle rec1, rectangle rec2){
    rectangle intersection = rec1;
    if(rec2.left_down.x > intersection.left_down.x)
        intersection.left_down.x = rec2.left_down.x;
    if(rec2.right_up.x < intersection.right_up.x)
        intersection.right_up.x = rec2.right_up.x;
    if(rec2.left_down.y > intersection.left_down.y)
        intersection.left_down.y = rec2.left_down.y;
    if(rec2.right_up.y < intersection.right_up.y)
        intersection.right_up.y = rec2.right_up.y;
    return area(intersection);
}
```

```cpp
int intersect(rectangle rec1, rectangle rec2, rectangle rec3){
    struct rectangle intersection = rec1;
    if(rec2.left_down.x > intersection.left_down.x)
        intersection.left_down.x = rec2.left_down.x;
    if(rec2.right_up.x < intersection.right_up.x)
        intersection.right_up.x = rec2.right_up.x;
    if(rec2.left_down.y > intersection.left_down.y)
        intersection.left_down.y = rec2.left_down.y;
    if(rec2.right_up.y < intersection.right_up.y)
        intersection.right_up.y = rec2.right_up.y;

    if(rec3.left_down.x > intersection.left_down.x)
        intersection.left_down.x = rec3.left_down.x;
    if(rec3.right_up.x < intersection.right_up.x)
        intersection.right_up.x = rec3.right_up.x;
    if(rec3.left_down.y > intersection.left_down.y)
        intersection.left_down.y = rec3.left_down.y;
    if(rec3.right_up.y < intersection.right_up.y)
        intersection.right_up.y = rec3.right_up.y;

    return area(intersection);
}

int union_area(rectangle rec[], int N){
    int A, B, C, AB, BC, AC, ABC;
    A = area(rec[0]);
    B = area(rec[1]);
    C = area(rec[2]);

    AB = intersect(rec[0], rec[1]);
    BC = intersect(rec[1], rec[2]);
    AC = intersect(rec[0], rec[2]);

    ABC = intersect(rec[0], rec[1], rec[2]);

    return A + B + C - AB - BC - AC + ABC;
}

int main(){
    rectangle rec[3];
    for( int i = 0; i < 3; i++ )
        cin >> rec[i].left_down.x >> rec[i].left_down.y >> rec[i].right_up.x >> rec[i].right_up.y;

    cout << union_area(rec, 3) << endl;
}
```

Password System

Check User

- ***Strcmp***
- If the user is found, return its location.
- Otherwise return -1.

```
12  int checkUser(char *name){
13      for(int i=0; i<20; i++){
14          if(strcmp(name, user[i].username)==0){
15              return i;
16          }
17      }
18      cout << "No person" << endl;
19      return -1;
20  }
```

Check Password

- ***Strcmp***
- ***Strlen***

```
21  bool checkPassword(char *tmp, int pos){
22      if(strcmp(tmp, user[pos].password)==0){
23          cout << "Duplicate password"<< endl;
24          return false;
25      }
26      if(strlen(tmp)<8 || strlen(tmp)>16){
27          cout << "Wrong length"<< endl;
28          return false;
29      }
30      int digital=0, A=0, a=0;
31      for(int i=0; i<strlen(tmp); i++){
32          if(tmp[i]>='0'&&tmp[i]<='9' || tmp[i]>='a'&&tmp[i]<='z' ||tmp[i]>='A'&&tmp[i]<='Z'){
33              if(tmp[i]>='0'&&tmp[i]<='9') digital++;
34              if(tmp[i]>='a'&&tmp[i]<='z') a++;
35              if(tmp[i]>='A'&&tmp[i]<='Z') A++;
36          }
37          else{
38              cout << "Wrong character"<< endl;
39              return false;
40          }
41      }
42      if(digital==0 || A==0 || a==0){
43          cout << "Must contain uppercase, lowercase and numbers"<< endl;
44          return false;
45      }
46      return true;
47  }
```

Set Password

- Note: It needs to be copied together with "\0".

```
48  void setPassword(char *pass, int pos){
49      for(int i=0; i<21; i++){
50          user[pos].password[i] = pass[i];
51      }
52  }
```

```
48  void setPassword(char *pass, int pos){
49      strcpy(user[pos].password, pass);
50  }
```

Complete Code

```
12  int checkUser(char *name){
13      for(int i=0; i<20; i++){
14          if(strcmp(name, user[i].username)==0){
15              return i;
16          }
17      }
18      cout << "No person" << endl;
19      return -1;
20  }
21  bool checkPassword(char *tmp, int pos){
22      if(strcmp(tmp, user[pos].password)==0){
23          cout << "Duplicate password"<< endl;
24          return false;
25      }
26      if(strlen(tmp)<8 || strlen(tmp)>16){
27          cout << "Wrong length"<< endl;
28          return false;
29      }
30      int digital=0, A=0, a=0;
31      for(int i=0; i<strlen(tmp); i++){
32          if(tmp[i]>='0'&&tmp[i]<='9' || tmp[i]>='a'&&tmp[i]<='z' ||tmp[i]>='A'&&tmp[i]<='Z'){
33              if(tmp[i]>='0'&&tmp[i]<='9') digital++;
34              if(tmp[i]>='a'&&tmp[i]<='z') a++;
35              if(tmp[i]>='A'&&tmp[i]<='Z') A++;
36          }
37          else{
38              cout << "Wrong character"<< endl;
39              return false;
40          }
41      }
```

```
42      if(digital==0 || A==0 || a==0){
43          cout << "Must contain uppercase, lowercase and numbers"<< endl;
44          return false;
45      }
46      return true;
47  }
48  void setPassword(char *pass, int pos){
49      strcpy(user[pos].password, pass);
50  }
```

Black or White

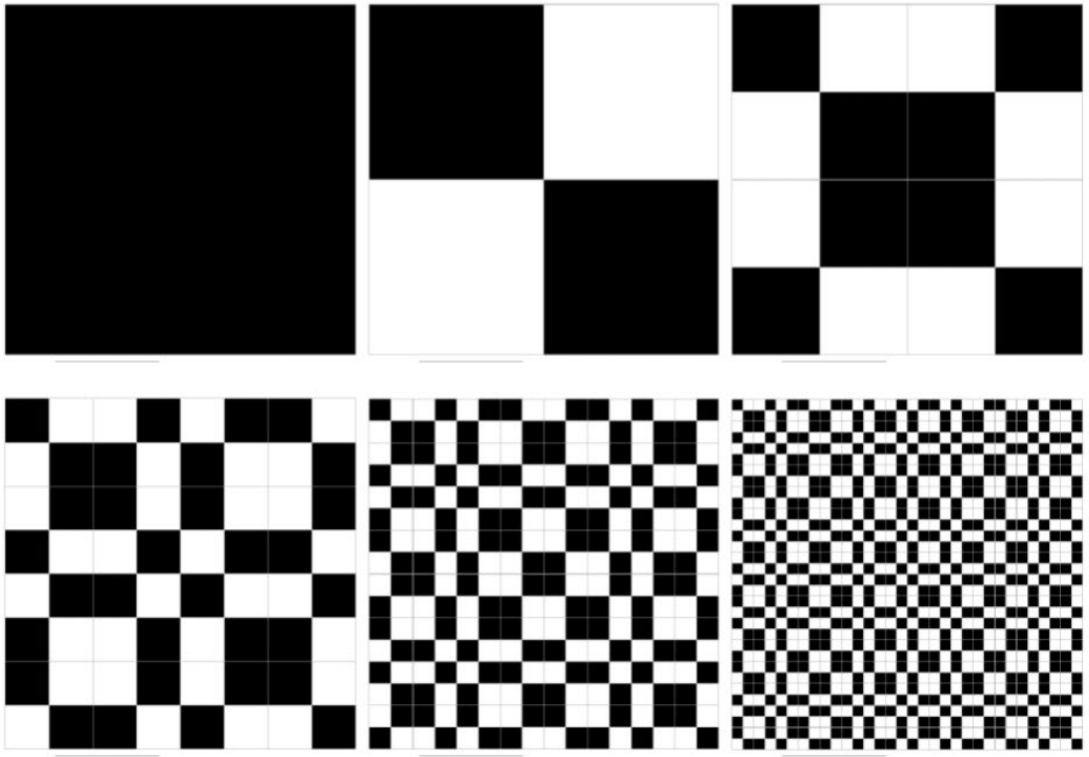
Problem

Description

阿明在公司負責馬賽克拼貼的設計，但他靈感枯竭，所以想用程式生成一些圖案，幫助他完成工作。

對於 N*N 的黑白拼貼，一開始所有的 pixel 的顏色皆相同，接著把它平分成四個小正方形，左上與右下的顏色維持相同，左下與右上的顏色變為相反。重複此過程，直到沒有辦法再平分為止，最後的圖案即為所求。

舉例來說，當**N = 32，初始顏色為黑色**時，整個拼貼的變化過程依序如下：



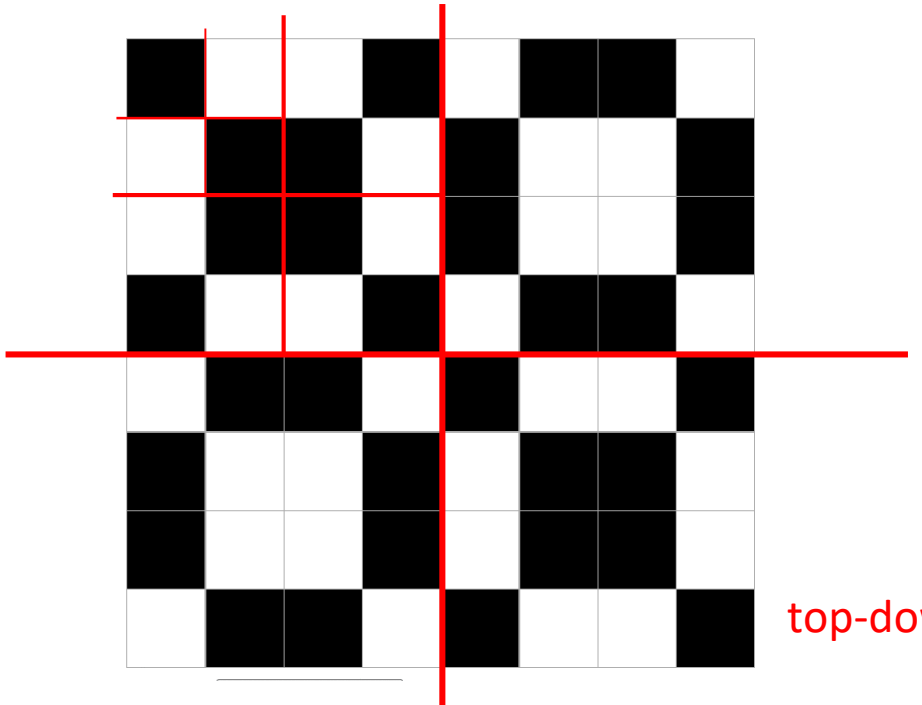
Solution 1: Recursive – main()

1. 動態配置陣列
2. 變數 color 紀錄初始顏色
3. mosaic()
4. 根據布林值輸出對應的字元

```
int main()
{
    int N;
    cin >> N;
    bool** A;
    A = new bool*[N];
    for (int i = 0; i < N; ++i)
        A[i] = new bool[N];

    char c;
    cin >> c;
    bool color = true;
    if (c == 'B')
        color = false;

    mosaic(A, 0, 0, N, color);

    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
            cout << ( A[i][j]? '.':'*' ) << " ";
        cout << endl;
    }

    return 0;
}
```

Solution 1: Recursive – mosaic()

1. 參數: A 為雙重指標，(x, y) 代表區域的起始位置，N 代表區域的邊長。
2. Base case: 當 N 為奇數時，將所有的 pixel 填上相同的顏色即可。
   →容易錯在 N = 1 的 testcase
3. 當 N 為偶數時，將原本的拼貼分為四個子區域個別處理。

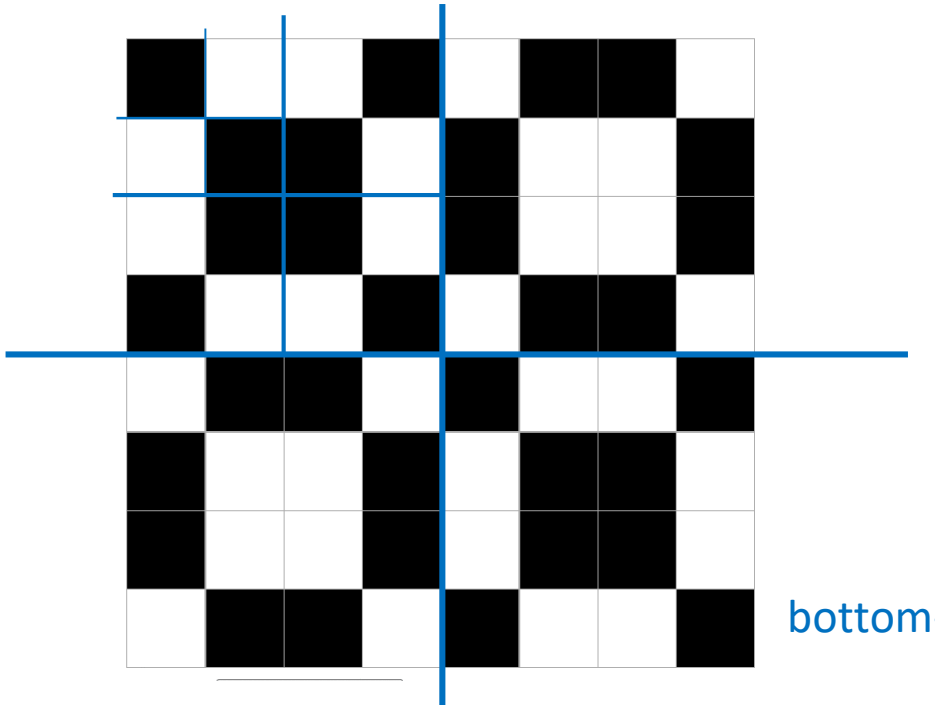


top-down

```
void mosaic(bool **A, int x, int y, int N, bool color)
{
    if (N % 2 == 1)
    {
        for (int i = 0; i < N; ++i)
            for (int j = 0; j < N; ++j)
                A[x+i][y+j] = color;
        return;
    }

    mosaic(A, x, y, N/2, color);
    mosaic(A, x+N/2, y, N/2, !color);
    mosaic(A, x, y+N/2, N/2, !color);
    mosaic(A, x+N/2, y+N/2, N/2, color);
}
```

Solution 2: Iterative

1. 找出最小單位 mini
2. 初始化左上角的最小區塊
3. 把左上角的顏色擴充為兩倍(邊長)
4. 重複3.，直到擴充至整個區域

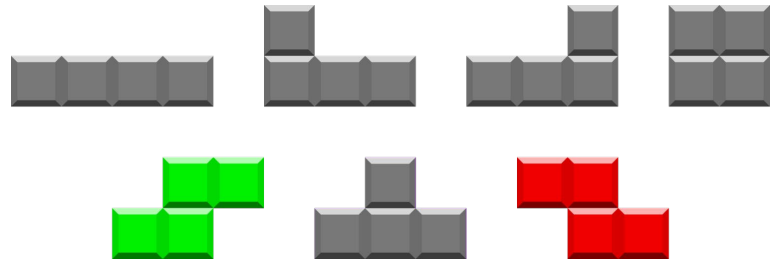


bottom-up

```
void mosaic_adv(char **A, int x, int y, int N, char color)
{
    int mini = N;
    while(mini % 2 == 0) mini /= 2;

    for (int i = 0; i < mini; ++i)
        for (int j = 0; j < mini; ++j)
            A[i][j] = color;

    for (int N_sub = mini*2; N_sub <= N; N_sub*=2)
    {
        for (int i = 0; i < N_sub/2; ++i)
            for (int j = 0; j < N_sub/2; ++j)
            {
                A[i+N_sub/2][j] = '*' + '.' - A[i][j];
                A[i][j+N_sub/2] = '*' + '.' - A[i][j];
                A[i+N_sub/2][j+N_sub/2] = A[i][j];
            }
    }
    return;
}
```

Hatetris

Problem



- 模擬 hatetris 的遊玩過程：
  1. 版面為 20*10，第 0 ~ 3 列是緩衝區，第 4 ~ 19 列則是正式遊玩區域。如果在消除橫列後，緩衝區仍有方塊存在，代表遊戲結束。
  2. 落下的方塊只會有 S 型與 Z 型，考慮到可以旋轉，方塊只會有以下四種形式：
  3. 輸入時除了給定形狀，也會給定該方塊橫跨的 y 座標範圍，因此能確定方塊的型式。舉例來說，若輸入為"S 1 3"，代表方塊為第 1 種形式。(可參考下圖動畫)
  4. 請參考 Hint 中的程式碼，並完成相關的函式，以模擬 hatetris 的遊玩過程。
- 需要實作的函式：
  1. initialize_board: 初始化盤面，所有的格子都沒有方塊存在
  2. place_block: 計算並儲存「方塊落下之後的盤面」
  3. clear_line: 消除填滿的橫列，並將上方的方塊往下移
  4. game_over: 確認遊戲是否結束

Solution - initialize_board() & game_over()

```
void initialize_board(Hatetris & hatetris)
{
    hatetris.board = new char*[20];
    for (int i = 0; i < 20; ++i)
        hatetris.board[i] = new char[10];

    for (int i = 0; i < 20; ++i)
        for (int j = 0; j < 10; ++j)
            hatetris.board[i][j] = '-';

    hatetris.line_cleared = 0;
}
```

```
bool game_over(Hatetris &hatetris)
{
    for (int i = 0; i < 4; ++i)
        for (int j = 0; j < 10; ++j)
            if (hatetris.board[i][j] == '*')
                return true;
    return false;
}
```

Solution - place_block()

1. 根據 head, tail 算出方塊的寬與高
2. 把方塊的形狀映射到二維陣列內

```
57  void place_block(Hatetris &hatetris, char shape, int head, int tail)
58  {
59      int fall = 0;
60      int width = tail-head+1;
61      int height = 5 - width;
62
63      char block[3][3];
64      for (int i = 0; i < 3; ++i)
65          for (int j = 0; j < 3; ++j)
66              block[i][j] = '-';
67
68      if (height == 2)
69      {
70          // - * *    * * -
71          // * * - or - * *
72          // - - -    - - -
73          block[0][1] = '*';
74          block[1][1] = '*';
75
76          if (shape == 'S')
77          {
78              block[0][2] = '*';
79              block[1][0] = '*';
80          }
81          else
82          {
83              block[0][0] = '*';
84              block[1][2] = '*';
85          }
86      }
87      else
88      {
89          // * - -    - * -
90          // * * - or * * -
91          // - * -    * - -
92          block[1][0] = '*';
93          block[1][1] = '*';
94
95          if (shape == 'S')
96          {
97              block[0][0] = '*';
98              block[2][1] = '*';
99          }
100         else
101         {
102             block[0][1] = '*';
103             block[2][0] = '*';
104         }
105     }
```

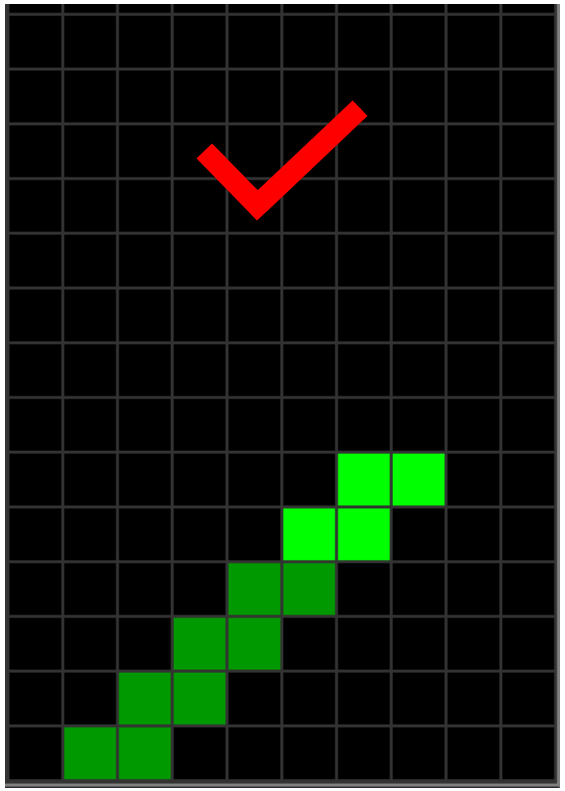
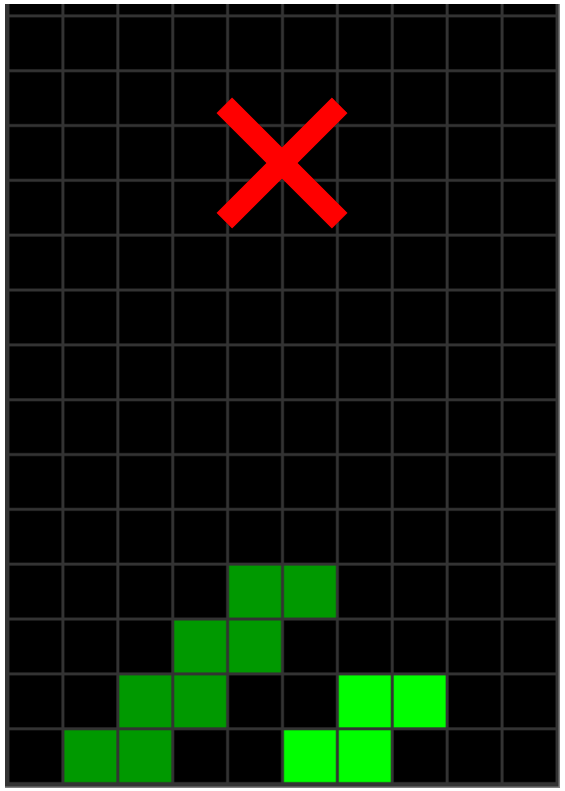
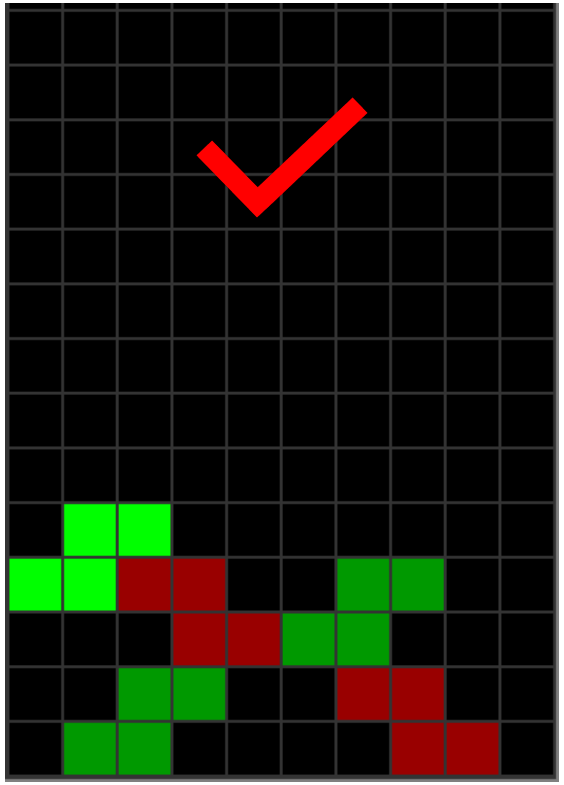
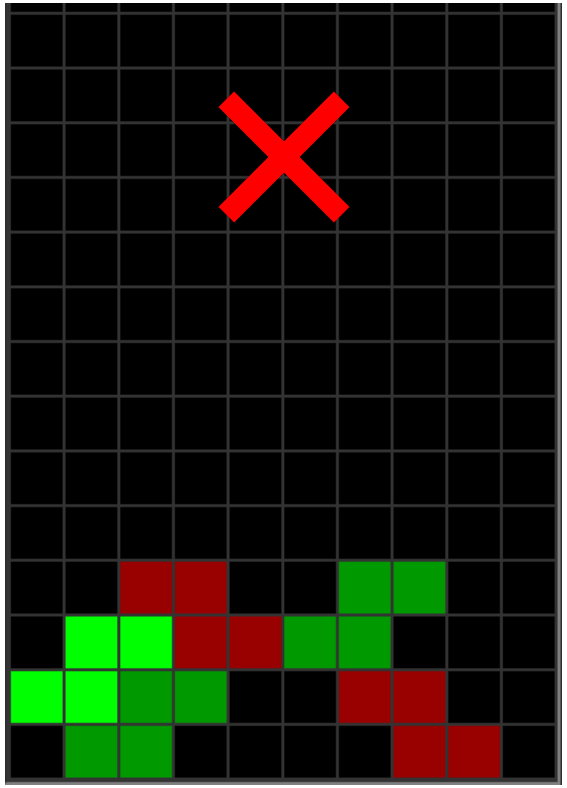
Solution - place_block()

3. 變數 fall 代表方塊下降的格數(從 0 開始)
4. 如果方塊與盤面沒有重疊(empty)，則繼續向下掉落，直到方塊落地或重疊
5. 將方塊填入對應的盤面位置

```
107     bool empty = true;
108     while(empty && (fall+height-1) < 20 )
109     {
110         for (int i = 0; i < height; ++i)
111             for (int j = 0; j < width; ++j)
112                 if (block[i][j] == '*' && hatetris.board[fall+i][head+j] == '*')
113                     empty = false;
114         if (empty)
115             fall++;
116     }
117
118     fall--;
119     for (int i = 0; i < 3; ++i)
120         for (int j = 0; j < 3; ++j)
121             if (block[i][j] == '*')
122                 hatetris.board[fall+i][head+j] = '*';
123 }
```

Solution - place_block()

- 容易犯錯的地方
  1. place block 沒有從上到下判斷
  2. 只依據方塊的最下層判斷

Solution - clear_line()

1. 由下至上確認每列是否已填滿
2. 如果已填滿，將上方的方塊向下移，且消除行數+1

```
void clear_line(Hatetris &hatetris)
{
    for (int i = 19; i >= 0; --i)
    {
        bool full_line = true;
        for (int j = 0; j < 10; ++j)
            if (hatetris.board[i][j] != '*')
                full_line = false;

        if (full_line)
        {
            for (int j = i; j > 0; --j)
                for (int k = 0; k < 10; ++k)
                    hatetris.board[j][k] = hatetris.board[j-1][k];
            i++;
            hatetris.line_cleared++;
        }
    }
}
```

如果消除，下一輪必須再從同一列檢查

Cycle

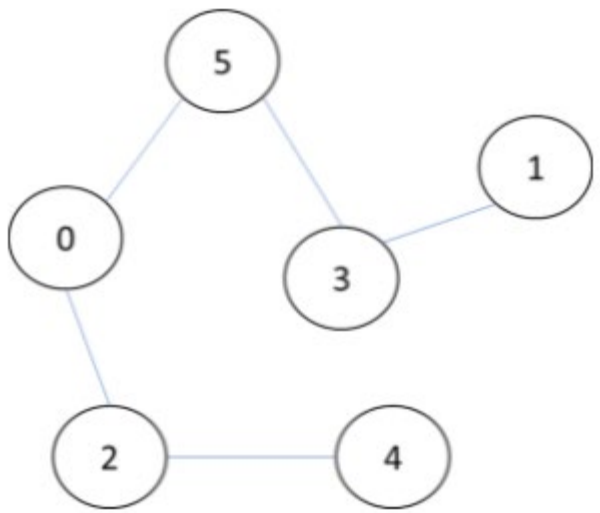
Problem

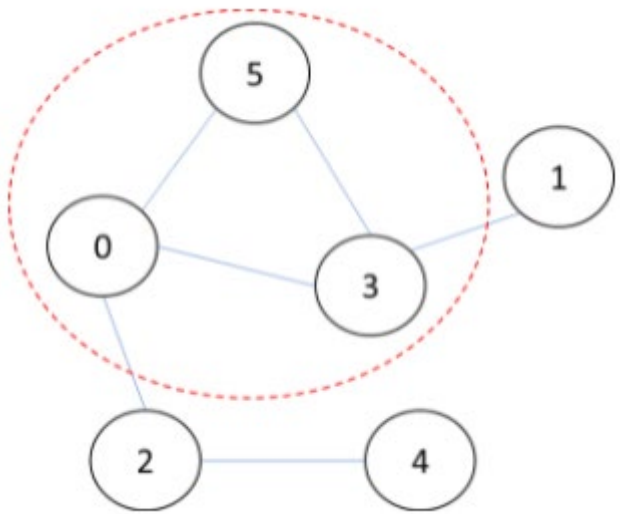
Description

A graph contains two parts as follows:

1. vertex: the node of the graph
2. edge: the link connecting two vertices

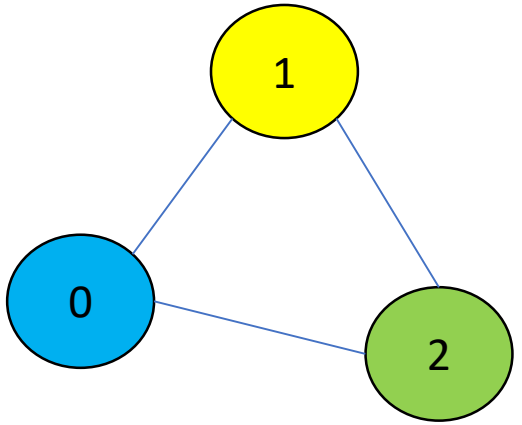
In this problem, you are given a graph. You have to detect if this given graph has a cycle and count the number of subgraphs.

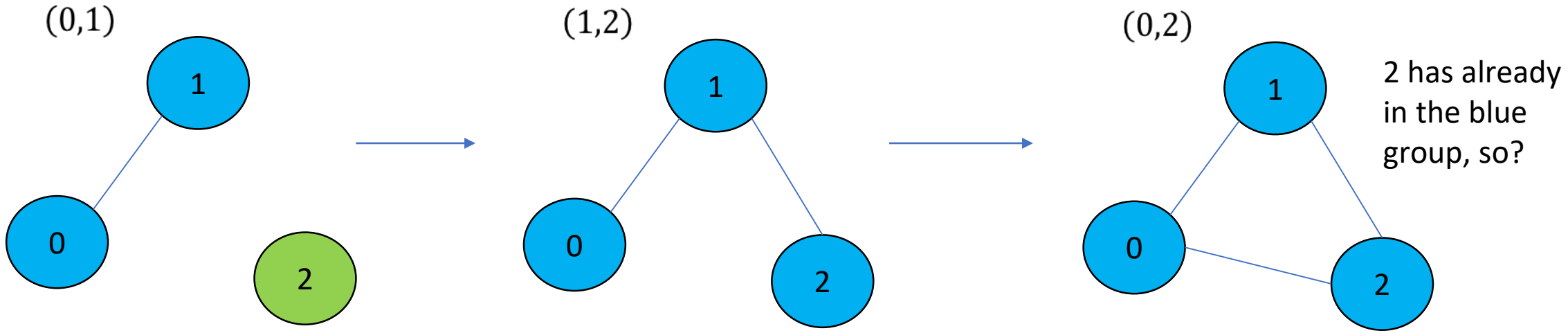




Solution

- Think about dividing the vertices into group when adding a edge
  - EX:
    - Vertex: 0, 1, and 3
    - Edge: (0, 1), (1, 2), (0, 2)



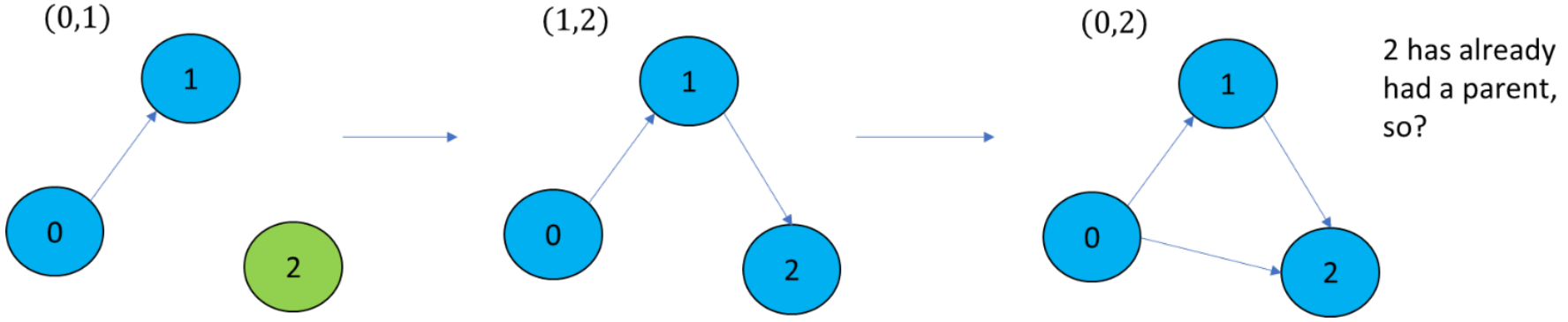


Solution

- How to group the connected vertices?
  - Think about the following structure

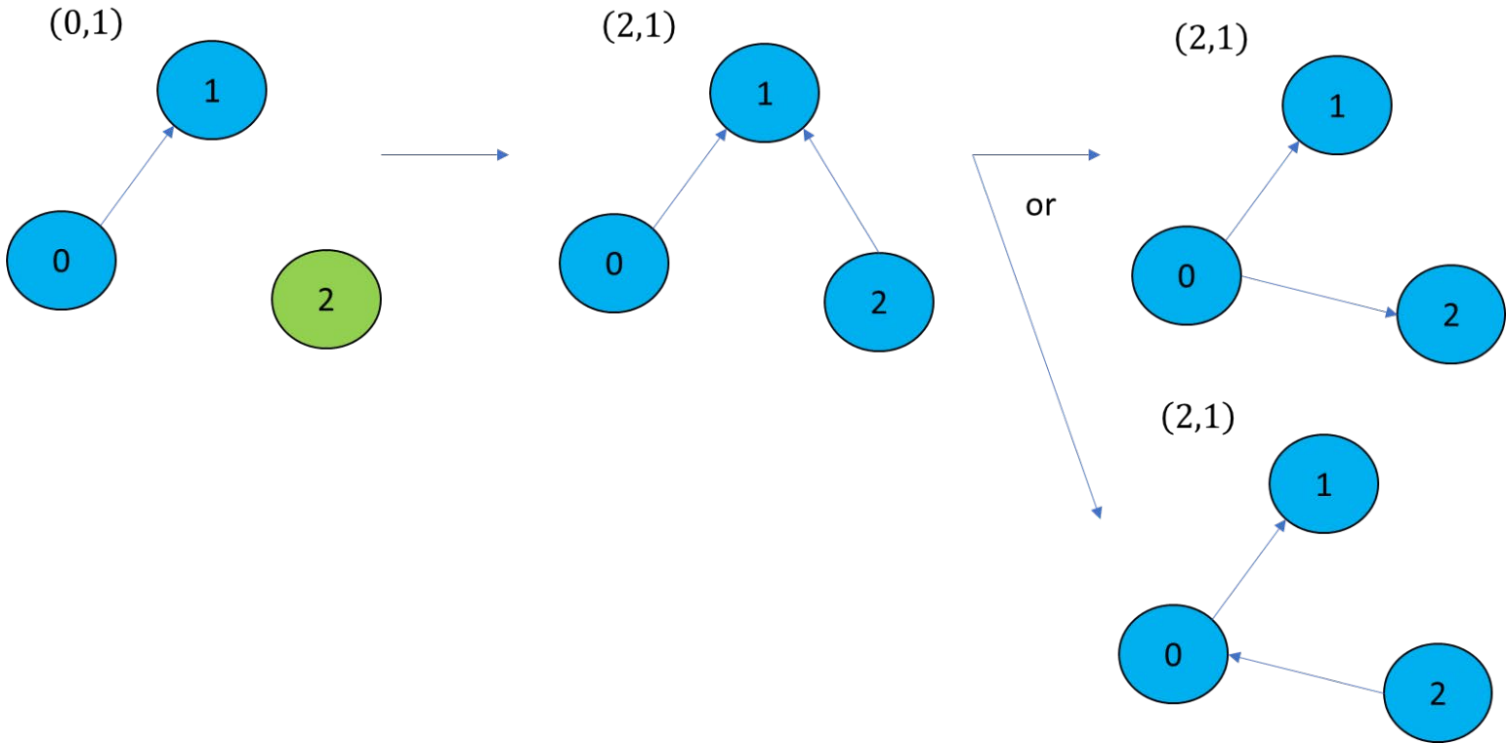
```
10  struct Node{
11      int index;
12      Node* parent;
13  };
14
```

    - Index can help you debug by using cout
    - Node* parent points to the vertex that connects to this vertex.
      - How to initialize, should it point to NULL?
      - No, we can point to the vertex itself for grouping.
    - The vertices in the same group share a common root.



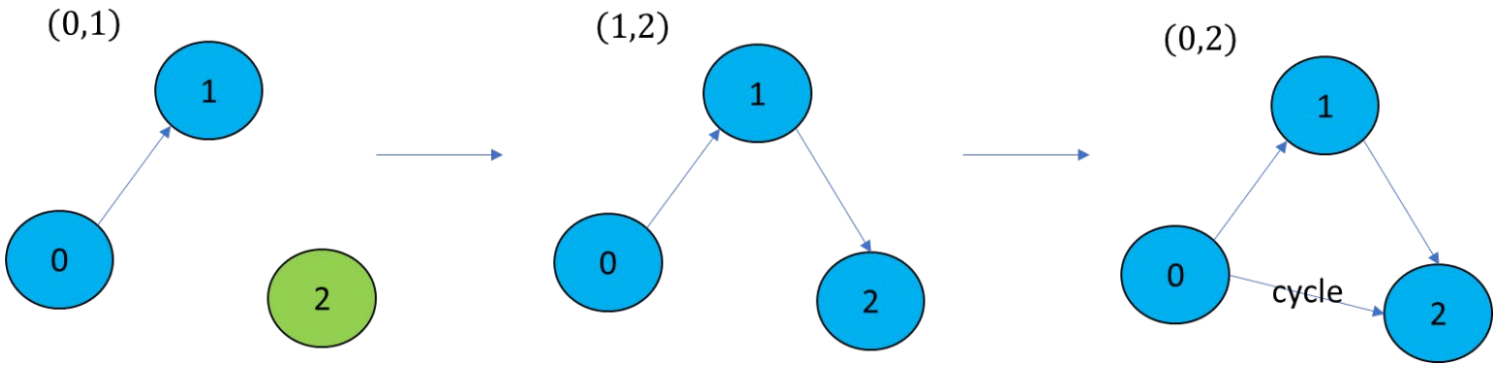
Solution

- If a vertex connect to another vertex which has already had a parent:
  - Check the roots of the two vertices
    - If the roots are different, point the parent of one of the roots to another.



Solution

- If the roots are the same, there is a cycle!



Solution

- How to implement?
  - Write a pointer function to find the root.
    - a while loop to keep updating the pointer
    - When the pointer point to a vertex whose parent is itself, stop

```
15  Node* find(Node *node){
16      // trace back the node until root
17      while(node->parent != node)
18          node = node->parent;
19      return node;
20  }
```

  - When reading an edge, find the roots of the two vertices.
    - If the roots are the same, there is a cycle.
    - If they are different, join the two roots.

```
35      bool cycle = false;
36      for( int i = 0; i < edge; i++){
37          Node* x = find(&node[link[i].start]);
38          Node* y = find(&node[link[i].end]);
39
40          // if the roots of the two nodes are the same, there is a cycle
41          if( x == y )
42              cycle = true;
43
44          // if thet aren't, join the two nodes
45          else
46              y->parent = x;
47      }
```

Solution

- How to determine how many subgraphs are there?
  - Find the root of each vertex, and check how many different roots.
    - Use a table to memorize the roots of each vertex

```
49      int graph_num = 1;
50      Node* root[vertex];
51
52      root[0] = find(&node[0]);
53      for( int i = 1; i < vertex; i++){
54
55          bool not_appear = true;
56          Node* temp = find(&node[i]);
57
58          // check the root is shown in the table or not
59          for( int j = 0; j < graph_num; j++ ){
60              if( temp == root[j] )
61                  not_appear = false;
62          }
63          if( not_appear == true ){
64              root[graph_num] = temp;
65              graph_num++;
66          }
67      }
```

Overall

```cpp
#include <iostream>

using namespace std;

struct Edge{
    int start;
    int end;
};

struct Node{
    int index;
    Node* parent;
};

Node* find(Node *node){
    // trace back the node until root
    while(node->parent != node)
        node = node->parent;
    return node;
}
```

```cpp
int main(){
    int vertex, edge;
    cin >> vertex >> edge;

    struct Edge link[edge];
    struct Node node[vertex];

    for( int i = 0; i < vertex; i++)
        node[i].parent = &node[i];

    for( int i = 0; i < edge; i++ )
        cin >> link[i].start >> link[i].end;

    bool cycle = false;
    for( int i = 0; i < edge; i++){
        Node* x = find(&node[link[i].start]);
        Node* y = find(&node[link[i].end]);

        // if the roots of the two nodes are the same, there is a cycle
        if( x == y )
            cycle = true;

        // if thet aren't, join the two nodes
        else
            y->parent = x;
    }
```

```cpp
    int graph_num = 1;
    Node* root[vertex];

    root[0] = find(&node[0]);
    for( int i = 1; i < vertex; i++){

        bool not_appear = true;
        Node* temp = find(&node[i]);

        // check the root is shown in the table or no
        for( int j = 0; j < graph_num; j++ ){
            if( temp == root[j] )
                not_appear = false;
        }
        if( not_appear == true ){
            root[graph_num] = temp;
            graph_num++;
        }
    }

    if(cycle == true)
        cout << "Cycle detected." << endl;
    else
        cout << "There is no cycle." << endl;
    cout << "Subgraph number: " << graph_num << endl;
    return 0;
}
```

Extension

- The method is a simpler version of disjoint set
- https://zh.wikipedia.org/wiki/%E5%B9%B6%E6%9F%A5%E9%9B%86