BÁO CÁO ĐÁNH GIÁ BÀI TẬP VỀ NHÀ NHÓM 12

Nhóm 1:

1. Abtraction

Cho 1 ma trận có giá trị 0 hoặc 1, đếm số vùng 1 (vùng chỉ có 1 số 1 hoặc có nhiều số 1 kề nhau).

2. Pattern Recognition

Các số 1 nằm kề nhau theo hàng cột và đường chéo

3. Decomposition

Đã có thể giải quyết bài toán mà không cần phân rã.

4. Algorithm Design

Khi duyệt xung quanh 1 điểm, ta duyệt luôn xung quanh của xung quanh để xem có các 1 kết nối với nhau hay không, nếu có thì mark True ở ma trận visit để không duyệt điểm đó nữa.

4. Algorithm Design

Khi duyệt xung quanh 1 điểm, ta duyệt luôn xung quanh của xung quanh để xem có các 1 kết nối với nhau hay không, nếu có thì mark True ở ma trận visit để không duyệt điểm đó nữa.

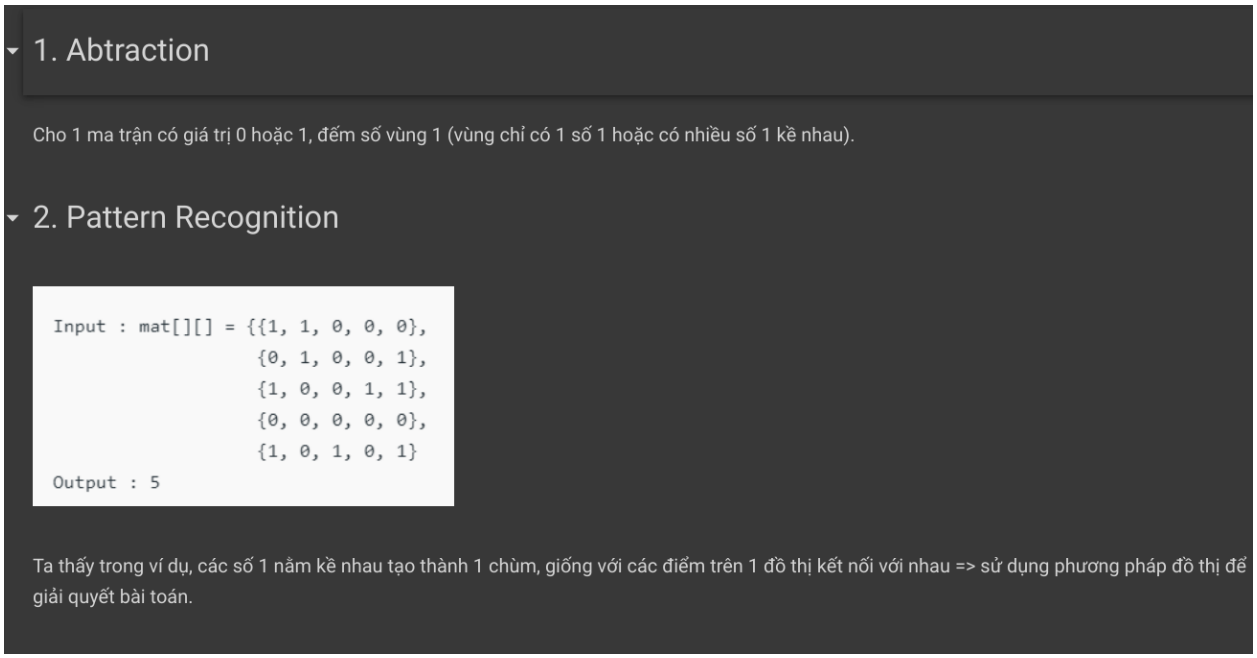
```
class Graph:
    def __init__(self, row, col, g):
        self.ROW = row
        self.COL = col
        self.graph = g

    def isSafe(self, i, j, visited):
        return (i >= 0 and i < self.ROW and j >= 0 and j < self.COL and not visited[i][j] and self.graph[i][j])

    def DFS(self, i, j, visited):
        rowNbr = [-1, -1, -1,  0, 0,  1, 1, 1];
        colNbr = [-1,  0,  1, -1, 1, -1, 0, 1];
        visited[i][j] = True
        for k in range(8):
            if self.isSafe(i + rowNbr[k], j + colNbr[k], visited):
                self.DFS(i + rowNbr[k], j + colNbr[k], visited)
```

```
    def countIslands(self):
        visited = [[False for j in range(self.COL)]for i in range(self.ROW)]
        count = 0
        for i in range(self.ROW):
            for j in range(self.COL):
                if visited[i][j] == False and self.graph[i][j] == 1:
                    self.DFS(i, j, visited)
                    count += 1
        return count

graph = [[1, 1, 0, 0, 0],
         [0, 1, 0, 0, 1],
         [1, 0, 0, 1, 1],
         [0, 0, 0, 0, 0],
         [1, 0, 1, 0, 1]]

row = len(graph)
col = len(graph[0])

g = Graph(row, col, graph)
print(g.countIslands())
```

Độ phức tạp: O(E+V) với E là số cạnh, V là số đỉnh

Nhóm làm tốt, trình bày rõ ràng, sử dụng DFS để giải quyết được bài toán đưa ra.

Nhóm 2:



1. Abtraction

Cho 1 ma trận có giá trị 0 hoặc 1, đếm số vùng 1 (vùng chỉ có 1 số 1 hoặc có nhiều số 1 kề nhau).

2. Pattern Recognition

```
Input : mat[][] = {{1, 1, 0, 0, 0},
                   {0, 1, 0, 0, 1},
                   {1, 0, 0, 1, 1},
                   {0, 0, 0, 0, 0},
                   {1, 0, 1, 0, 1}
Output : 5
```

Ta thấy trong ví dụ, các số 1 nằm kề nhau tạo thành 1 chùm, giống với các điểm trên 1 đồ thị kết nối với nhau => sử dụng phương pháp đồ thị để giải quyết bài toán.

3. Decomposition

Đã có thể giải quyết bài toán mà không cần phân rã.

4. Algorithm Design

Khi duyệt xung quanh 1 điểm, ta duyệt luôn xung quanh của xung quanh để xem có các 1 kết nối với nhau hay không, nếu có thì mark True ở ma trận visit để không duyệt điểm đó nữa.

```
[ ]  from collections import deque

     def isLand(map, i, j, vis):

       return ((i >= 0) and (i < 5) and
           (j >= 0) and (j < 5) and
         (map[i][j] and (not vis[i][j])))

     def solve(map, vis, search_i, search_j):
       #Duyệt xung quanh điểm đang xét từ trên xuống, từ trái qua phải
       row = [-1, -1, -1, 0, 0, 1, 1, 1]
       col = [-1, 0, 1, -1, 1, -1, 0, 1]

       q = deque()
       q.append([search_i, search_j])
       vis[search_i][search_j] = True
```

Sửa hàm isLand: i<row and j< col

```
       #duyệt điểm hiện tại, đồng thời duyệt luôn nếu điểm xung quanh điểm hiện tại có điểm xung quanh = 1 hay không
       while (len(q) > 0):
         temp = q.popleft()

         i = temp[0]
         j = temp[1]

         for k in range(8):
           if (isLand(mat, i + row[k], j + col[k], vis)):
             vis[i + row[k]][j + col[k]] = True
             q.append([i + row[k], j + col[k]])

     def countIslands(map):

       vis = [[False for i in range(5)]
             for i in range(5)]

       count = 0

       for i in range(5):
         for j in range(5):
           if (map[i][j] and not vis[i][j]):
             solve(map, vis, i, j)
             count += 1

       return count
```

```
map = [[ 1, 1, 0, 0, 0 ],
       [ 0, 1, 0, 0, 1 ],
       [ 1, 0, 0, 1, 1 ],
       [ 0, 0, 0, 0, 0 ],
       [ 1, 0, 1, 0, 1 ]]

print(countIslands(map))
```

```
5
```

Độ phức tạp: O(E+V) với E là số cạnh, V là số đỉnh

Nhìn chung nhóm làm tốt, sử dụng BFS giải quyết được yêu cầu bài toán đưa ra, sai sót một chút ở hàm isLand.

Nhóm 3:

1. Abstraction

Cho một ma trận có giá trị 0 hoặc 1, đếm số vùng 1 (2 hay nhiều số 1 kề nhau).

2. Pattern Recognition

Dùng DFS để duyệt.

3. Decomposition

Không cần phân rã.

4. Algorithm Design

```
rowIndex = [-1, -1, -1, 0, 0, 1, 1, 1]
colIndex = [-1, 0, 1, -1, 1, -1, 0, 1]

def DFS(land, row, col):
    if row < 0 or row >= numRow or col < 0 or col >= numCol or land[row][col] != 1:
        return
    land[row][col] = 0
    for i in range(len(rowIndex)):
```

```
            DFS(land, row + rowIndex[i], col + colIndex[i])

    def countLand(land):
        count = 0
        for row in range(numRow):
            for col in range(numCol):
                if land[row][col] == 1:
                    count += 1
                    DFS(land, row, col)
        return count

    numRow, numCol = map(int, input().split())
    land = []
    for i in range(numRow):
        temp = list(map(int, input().split()))
        land.append(temp)
    print(countLand(land))
```

```
5 5
1 1 0 0 0
0 1 0 0 1
1 0 0 1 1
0 0 0 0 0
1 0 1 0 1
5
```

Độ phức tạp thuật toán: $O(E+V)$ với E là cạnh và V là đỉnh.

Nhóm làm tốt, đúng yêu cầu bài, sử dụng DFS để giải.

Nhóm 4:

Abstraction:

Tìm số thành phần liên thông của các node có giá trị 1 trong đồ thị.

Decomposing

Bài toán đã đủ nhỏ để giải quyết, Không cần phân rã

Pattern Recognition

Ta thấy trong ví dụ, các số 1 nằm kề nhau tạo thành 1 chùm, giống với các điểm trên 1 đồ thị kết nối với nhau => sử dụng phương pháp đồ thị để giải quyết bài toán.

→ Dùng DFS hoặc BFS để duyệt

Algorithm Design

Sử dụng DFS

Duyệt từng phần tử, nếu phần tử đó có giá trị là 1 thì đây là 1 thành phần liên thông

Duyệt tiếp ra 8 phần tử xung quanh và đánh dấu đã duyệt

Programming

```
arrow = [-1, -1, -1, 0, 0, 1, 1, 1]
arcol = [-1, 0, 1, -1, 1, -1, 0, 1]

def DFS(matrix, row, col):
    if row < 0 or row >= num_row or col < 0 or col >= num_col or matrix[row][col] != 1:
        return
    matrix[row][col] = 0
    for i in range(len(arrow)):
        DFS(matrix, row + arrow[i], col + arcol[i])

def rs(mat):
    count = 0
    for row in range(num_row):
        for col in range(num_col):
            if matrix[row][col] == 1:
                count += 1
                DFS(mat, row, col)
    return count

print('Nhap lan luot so hang, so cot cua ma tran');
num_row, num_col = map(int, input().split())
matrix = []
print('Nhap tung phan tu cua ma tran');
for i in range(num_row):
    tmp = list(map(int, input().split()))
    matrix.append(tmp)
print("\n",rs(matrix))
```

```
Nhap lan luot so hang, so cot cua ma tran
3 3
Nhap tung phan tu cua ma tran
1 0 0
0 1 1
1 1 1

 1
```

Làm bài tốt, vận dụng DFS để giải bài toán.

Nhóm 5:

Abstraction:

Tìm số thành phần liên thông của các node có giá trị 1 trong đồ thị.

Pattern Recognition

- Dùng DFS hoặc BFS để duyệt

Algorithm Design

- Duyệt từng phần tử, nếu phần tử đó có giá trị là 1 thì đây là 1 thành phần liên thông
- Duyệt tiếp ra 8 phần tử xung quanh và đánh dấu đã duyệt

```
xung_quanh_dong = [-1, -1, -1, 0, 0, 1, 1, 1]
xung_quanh_cot = [-1, 0, 1, -1, 1, -1, 0, 1]

def DFS(islandMatrixMap, dong, cot):
    if dong < 0 or dong >= so_dong or cot < 0 or cot >= so_cot or islandMatrixMap[dong][cot] != 1:
        return
    islandMatrixMap[dong][cot] = 0
    for i in range(len(xung_quanh_dong)):
        DFS(islandMatrixMap, dong + xung_quanh_dong[i], cot + xung_quanh_cot[i])
```

```
def isLand(islandMatrixMap):
    count = 0
    for dong in range(so_dong):
        for cot in range(so_cot):
            if islandMatrixMap[dong][cot] == 1:
                count += 1
                DFS(islandMatrixMap, dong, cot)
    return count

so_dong, so_cot = map(int, input().split())
islandMatrixMap = []
for i in range(so_dong):
    temp = list(map(int, input().split()))
    islandMatrixMap.append(temp)
print(isLand(islandMatrixMap))
```

```
5 5
1 1 0 0 0
0 1 0 0 1
1 0 0 1 1
0 0 0 0 0
1 0 1 0 1
5
```

Độ phức tạp thuật toán: $O(nm)$ với n = so_dong, m = so_cot

Nhóm làm tốt, giải thích rõ ràng, sử dụng DFS để giải quyết vấn đề.

Nhóm 6:

Decomposition

Xác định các phần tử có giá trị 1 xung quanh 1 phần tử đang xét

Pattern Recognition

Bài toán xác định các phần tử lân cận

Abstraction

Cho một mảng 2 chiều với các phần tử mang giá trị 0 và 1. Các phần tử có giá trị 1 đứng lân cận nhau tạo thành một chuỗi các phần tử. Tính toán xem có bao nhiêu chuỗi như vậy

Algorithm Design

1. Tạo 1 mảng visited có kích thước bằng kích thước của mảng nhập vào và có giá trị là False
2. Duyệt từng phần tử trong mảng đầu vào kết hợp kiểm tra điều kiện nếu phần tử tương ứng trong mảng visited = True thì bỏ qua, còn nếu False thì chúng ta sẽ xét. Mỗi lần duyệt xong phần tử nào thì phần tử tương ứng trong mảng visited sẽ có giá trị là True.
3. Xét:

    Nếu giá trị đang xét == 0 thì xét phần tử tiếp theo.

    Nếu giá trị đang xét == 1 thì thực hiện:

        Thêm phần tử hiện tại vào queue.

        Xét các phần tử xung quanh phần tử đang xét, nếu các phần tử xung quanh có giá trị là 1 và có phần tử tương ứng trong mảng visited là False thì thêm vào queue.

        Tương tự, xét các phần tử còn lại có trong queue cho đến khi queue không còn phần tử nào. Sau mỗi lần duyệt qua phần tử ta đánh dấu phần tử tương ứng trong mảng visited tương ứng là True. Kết thúc, ta sẽ tìm ra được một chuỗi.

```
def COUNT(matrix,m,n):
    visited =[[0 for x in range (n)]for x in range (m)]                    #Tạo mảng visited để đánh dấu những phần tử đã đi qua
    count=0
    for i in range (m):
        for j in range (n):                                                # Xét duyệt đồ thị
            if (matrix[i][j]==1 and visited[i][j]==0):                     #Trường hợp phần tử=1
                queue=[]                                                   # Tạo queue để lưu các phần tử lân cận
                queue.append([i,j])
                while queue:
                    temp=queue.pop(0)                                      # Đưa phần tử ra khỏi queue
                    visited[temp[0]][temp[1]]=1
                    c=temp[0]-1                                            # Tạo chỉ số của các phần tử lân cận
                    if c<=-1:                                              #Loại bỏ trường hợp phần tử đang xét nằm ở rìa
                        c=0
```

```
                    d=temp[0] + 2
                    if d>=m:
                        d=m
                    e=temp[1] - 1
                    if e<=-1:
                        e=0
                    f=temp[1] + 2
                    if f>=n:
                        f=n
                    for a in range (c,d):
                        for b in range (e,f):
                            if (matrix[a][b]==1 and visited[a][b]==0):
                                queue.append([a,b])
                count+=1
            else: visited[i][j]=1
    return count


matrix=[[1,1,0,0,0],
        [0,1,0,0,1],
        [1,0,0,1,1],
        [0,0,0,0,0],
        [1,0,1,0,1]]

matrix2=[[1,0,0,0,0],
        [0,1,1,0,1],
        [1,0,0,1,1],
        [0,0,0,0,0],
        [1,0,0,0,1]]
```

```
#Thêm phần tử lân cận vào queue nếu đáp ứng điều kiện

#Trường hợp phần tử=0
```

```
m=n=5
print(COUNT(matrix,m,n))
print(COUNT(matrix2,m,n))
```

```
5
3
```

Trình bày dễ hiểu, kỹ càng, giải thích rõ ràng, hiểu vấn đề, hoàn thành tốt. Dùng BFS để giải bài toán.

Nhóm 7:

Abstraction

Cho một ma trận nhị phân, đếm số vùng 1 (vùng có các số 1 đứng lân cận nhau chỉ đc coi là 1 vùng)

Decomposion

Tại mỗi vị trí có có giá trị một, tìm tất cả các vị trí lân cận và đánh dấu lại để không đếm lại nhiều lần.

Pattern Recognition

Dùng DFS

Algorithm Design

- Duyệt từng phần tử trong ma trận, nếu phần tử có giá trị một thì cộng biến đếm thành phần liên thông lên 1
- Đồng thời duyệt 8 vị trí lân cận và đánh dấu lại vị trí đã duyệt qua.

```
[ ]  around_row = [-1, -1, -1, 0, 0, 1, 1, 1]
     around_col = [-1, 0, 1, -1, 1, -1, 0, 1]

     def DFS(matrix, row, col):
         if row < 0 or row >= num_row or col < 0 or col >= num_col or matrix[row][col] != 1:
             return
         matrix[row][col] = 0
         for i in range(len(around_row)):
             DFS(matrix, row + around_row[i], col + around_col[i])

     def land(matrix):
         count = 0
         for row in range(num_row):
             for col in range(num_col):
                 if matrix[row][col] == 1:
                     count += 1
                     DFS(matrix, row, col)
         return count

     num_row, num_col = map(int, input().split())
     matrix = [[ 1, 1, 0, 0, 0 ],
           [ 0, 1, 0, 0, 1 ],
           [ 1, 0, 0, 1, 1 ],
           [ 0, 0, 0, 0, 0 ],
           [ 1, 0, 1, 0, 1 ]]

     print(land(matrix))
```

```
5 5
5
```

Nhóm làm tốt. Dùng DFS để giải quyết bài toán.

Nhóm 8:

1. Abstraction

Cho một mảng 2 chiều với các phần tử mang giá trị 0 và 1. Các phần tử có giá trị 1 đứng lân cận nhau tạo thành một chuỗi các phần tử. Tính toán xem có bao nhiêu chuỗi như vậy

2. Decomposition

Xác định các phần tử có giá trị 1 xung quanh 1 phần tử đang xét

3. Pattern Recogntion

Bài toán xác định các phần tử lân cận

4. Alogorithm Design

- Lần lượt kiểm tra các phần tử theo thứ tự trong ma trận:

1. Nếu có phần tử bằng 1. Nếu có thì số vùng đất tăng lên 1.
2. Kiểm tra các vùng lân cận vị trí vừa xác định được xem có phần tử nào bằng 1 nữa không. Nếu có thì đánh dấu để không xét điểm đó ở bước 1 nữa.

- In ra kết quả số vùng đất tìm được

5. Độ phức tạp của thuật toán

Độ phức tạp thuật toán: O(nm) với n = num_row, m = num_col

```python
from queue import Queue

def Check(x, y, mat):
  n = len(mat)
  if x < 0 or y < 0 or x>=n or y>=n or mat[x][y] == 0:
    return False
  return True

def BFS(u, v, mat, check):
  q = Queue()
  q.put((u, v))
  dx = [-1, -1, 0, 1, 1, 1, 0, -1]
  dy = [0, -1, -1, -1, 0, 1, 1, 1]

  while q.empty() == False:
    x, y = q.get()
    check[x][y] = 1
    for i in range(8):
      u = x + dx[i]
      v = y + dy[i]
    if Check(u, v, mat):
      q.put((u, v))
```

```
def count_connected_component(mat):
  n = len(mat)
  cnt = 0
  check = [[0] * n] * n
  for i in range(n):
    for j in range(n):
      if mat[i][j] and check[i][j] == 0:
        BFS(i, j, mat, check)
        cnt += 1
        print(i, j)
  print(cnt)

mat = [[1, 1, 0, 0, 0],
       [0, 1, 0, 0, 1],
       [1, 0, 0, 1, 1],
       [0, 0, 0, 0, 0],
       [1, 0, 1, 0, 1]]
count_connected_component(mat)
```

Sử dụng BFS để giải quyết bài toán, hoàn thành tốt, rõ ràng.

Nhóm 9:

Abstraction:

Tìm số thành phần liên thông của các node có giá trị 1 trong đồ thị.

Decomposition

Bài toán đã đủ nhỏ để giải quyết.

Pattern Recognition

- Dùng DFS hoặc BFS để duyệt

Algorithm Design

- Duyệt từng phần tử, nếu phần tử đó có giá trị là 1 thì đây là 1 thành phần liên thông
- Duyệt tiếp ra 8 phần tử xung quanh và đánh dấu đã duyệt

```
[ ] around_row = [-1, -1, -1, 0, 0, 1, 1, 1]
    around_col = [-1, 0, 1, -1, 1, -1, 0, 1]
```

```python
def DFS(matrix, row, col):
    if row < 0 or row >= num_row or col < 0 or col >= num_col or matrix[row][col] != 1:
        return
    matrix[row][col] = 0 # Xung quanh toàn là nước êiiiii :))
    for i in range(len(around_row)):
        DFS(matrix, row + around_row[i], col + around_col[i])

def count_land(matrix):
    count = 0
    for row in range(num_row):
        for col in range(num_col):
            if matrix[row][col] == 1:
                count += 1
                DFS(matrix, row, col)
    return count

num_row, num_col = map(int, input().split())
matrix = []
for i in range(num_row):
    temp = list(map(int, input().split()))
    matrix.append(temp)
print(count_land(matrix))
```

```
5 5
1 1 0 0 0
0 1 0 0 1
1 0 0 1 1
0 0 0 0 0
1 0 1 0 1
5
```

Độ phức tạp thuật toán: $O(nm)$ với n = num_row, m = num_col

Hoàn thành tốt. Tương tự như các nhóm trên, nhóm cũng sử dụng DFS để giải quyết vấn đề

Nhóm 11:

1 Abstraction

Cho một ma trận nhị phân, đếm số vùng 1 (vùng có các số 1 đứng lân cận nhau chỉ đc coi là 1 vùng)

2 Decomposition

Tìm thành phần liên thông của node có giá trị 1

3 Pattern Regconition

Bài toán đồ thị

4 Algorithm Design

- Tạo 2 mảng index_row, index_col.
- Duyệt thành phần liên thông của mỗi phần tử trong mảng land(mảng đầu vào) nếu bằng 1 thì tăng biến đếm count và xét các vị trí xung quanh bằng hàm tìm liên thông DFS.
- Nếu vị trí hợp lệ và khác 1 thì gán bằng 0 và tiếp tục tìm liên thông điểm đó nếu không thì trả về.
- Kết quả là biến đếm count.

```
[ ]  index_col = [-1, 0, 1, -1, 1, -1, 0, 1]
     index_row = [-1, -1, -1, 0, 0, 1, 1, 1]
```

```
    def DFS(land, row, col):
      #if row<0 or num_row>row  or col<0 or num_col>col or land[row][col] != 1:
        #return
      if row<0 or num_row<=row  or col<0 or num_col<=col or land[row][col] != 1:
        return
      land[row][col] = 0 # danh dau da xet
      for i in range (8): # xung quanh co 8 o
          DFS(land, row + index_row[i], col + index_col[i])

    def count_land(land):
      count = 0
      for row in range(num_row ):
        for col in range(num_col ):
          if land[row][col]==1:
            count+=1
            #print("row: ", row, "col", col, "matrix: ", land[row][col])
            #print("----", count,"--")
            DFS(land, row, col)
      return count

    print("Nhap so hang, so cot: ")
    num_row, num_col = map(int, input().split())
    print("Nhap du lieu manh dat:")
```

```python
print("Nhap so hang, so cot: ")
num_row, num_col = map(int, input().split())
print("Nhap du lieu manh dat:")

land = []
for row in range(num_row):
  #row = map(int, input().split())
  row = list(map(int, input().split()))
  land.append(row)
print("So manh dat: ",count_land(land))
```

```
Nhap so hang, so cot:
5 5
Nhap du lieu manh dat:
1 1 0 0 0
0 1 0 0 1
1 0 0 1 1
0 0 0 0 0
1 0 1 0 1
So manh dat:  5
```

Hoàn thành tốt, hiểu vấn đề.

Nhận xét:

- Các nhóm hoàn thành bài tập đầy đủ và đúng hạn.
- Các nhóm đều nắm rõ và vận dụng thành thạo Computational Thinking để giải quyết vấn đề.
- Vận dụng tốt các phương pháp để giải quyết vấn đề.