데이터 구조 및 알고리즘 그래프 프로젝트

2016170994 김다민

●코드 및 코드 설명

그래프는 파이썬의 자료구조 딕셔너리를 이용해 저장하였다.

각 부분 코드 설명은 코드에 주석으로 달아 놓았다.

from collections import defaultdict  
  
  
class Graph:  
  
 *'''  
 파이썬의 dictionary구조를 이용해 그래프를 저장했으며,  
 count는 DSF알고리즘 진행시 num값을 매길때 사용된다.  
 V는 그래프 vertice 갯수를 나타낸다.  
 '''*

def \_\_init\_\_(self, num\_ver):  
 self.V = num\_ver  
 self.graph = defaultdict(list)  
 self.count = 1  
  
  
 def addEdge(self, v, w):  
 self.graph[v].append(w)  
 self.graph[w].append(v)  
  
 '''   
 v: 지금 방문하는 vertice   
 visited[v] :vertice v의 방문 여부 표시   
 num[]: DSF알고리즘 과정중 지나간 순서를 나타냄   
 parent[v] :DFS tree에서 v의 부모 vertice   
 ap[] :각 인덱스가 articulation point인지 알려주는 배열(리스트)   
 (ex: ap[1]=True이면 1번vertice는 articulation point임)  
 low[v]:DSF path를 지나지 않고 에서 갈 수있는 num이 가장 작은 vertice   
 '''  
  
 def recur\_AP(self, v, visited, is\_ap, parent, low, num):  
  
  
 children = 0  
  
 visited[v] = True  
  
 num[v] = self.count  
 low[v] = self.count  
 self.count += 1  
  
  
  
 for w in self.graph[v]:  
  
 '''  
 w는 v의 인접 vertice들을 가르키고  
 인접노드중 방문한지 않은 vertice가 있다면 visited[w]==False 그 vertice를 자식으로 삼고  
 그자식의 low값과 자신의 low값중 작은 값을 low[v]에 넣는데   
 이과정은 ecursive하게 일어나며 해당 인접노드도 자식을 찾게 하여 DFS tree를 만든다.  
 (왜냐하면 자식의 low값 즉 DFS루트가 아닌 다른루트로 갈수있는 가장 작은 num의 vertice는 부모도 갈수 있기 때문)  
  
 '''  
 if visited[w] == False:  
 parent[w] = v  
 children += 1  
  
  
 self.recur\_AP(w, visited, is\_ap, parent, low, num)  
 low[v] = min(low[v], low[w])  
  
  
  
 '''  
 v 가 root인경우 (parent[v] == -1)  
 DFS tree의 root인 v의 자식이 2개이상이면 articulation point이다.   
 (자식인 두 노드가 속하는 subgraph를 서로 잇는 edge가 없기 때문)  
 '''  
 if parent[v] == -1 and children >= 2:  
 is\_ap[v] = True  
  
  
 '''  
 v가 root가 아닌 경우 (parent[v] != -1)  
 자식의 low값이 v의 num값보다 크면 자식은 v를 거치지 않고는 나머지 그래프와 이어질수 없으므로  
 v는 articulation point이다.  
 '''  
  
 if parent[v] != -1 and low[w] >= num[v]:  
 is\_ap[v] = True  
  
  
 # 아래 과정은 인접노드가 자식이 아닌경우 인데 자식이 아님에도,  
 # 인접노드이기 때문에 지나갈수 있으므로 현재 low[v]와 num[w]를 비교하여 더작은 값을 low[v]에 넣는다.  
  
 elif w != parent[v]:  
 low[v] = min(low[v], num[w])

#초기화 및 출력하는 함수  
 def project3(self):  
  
 *'''  
 모든 vertice에 대해 초기화 ,  
 visited,is\_ap 는 모두 false로 초기화  
 num,low는 무한대로 초기화(왜냐하면 min함수에 영향을 주지 않기위해, verticer갯수보다 큰 아무 숫자로 초기화해도 됨)  
 부모노드는 쓰이지않는 인덱스인 -1로  
 articulation point  
 '''* visited = [False] \* (self.V)  
 num = [float('Inf')] \* (self.V)  
 low = [float('Inf')] \* (self.V)  
 parent = [-1] \* (self.V)  
 is\_ap = [False] \* (self.V)  
  
  
  
 for i in range(self.V):  
 if visited[i] == False:  
 self.recur\_AP(i, visited, is\_ap, parent, low, num)  
  
  
  
 print ("vertice index",end='')  
 for i in range(self.V) :  
 print (" %2d"%i,end='')  
  
 print ("\n num",end='')  
 for i in range(self.V):  
 print (" %2d"%num[i],end='')  
  
 print ("\n low",end='')  
 for i in range(self.V):  
 print(" %2d" % low[i], end='')  
  
 print("\n\nArticulation points: ",end='')  
 for index, value in enumerate(is\_ap):  
 if value == True:  
 print (str(index) + " " ,end='')  
  
  
g1 = Graph(13)  
g1.addEdge(0, 1)  
g1.addEdge(0, 2)  
g1.addEdge(1, 2)  
g1.addEdge(2, 3)  
g1.addEdge(2, 4)  
g1.addEdge(1, 6)  
g1.addEdge(5, 6)  
g1.addEdge(6, 7)  
g1.addEdge(5, 8)  
g1.addEdge(6, 8)  
g1.addEdge(4, 9)  
g1.addEdge(5, 9)  
g1.addEdge(8, 10)  
g1.addEdge(7, 11)  
g1.addEdge(8, 12)  
  
  
g2 = Graph(12)  
g2.addEdge(0, 1)  
g2.addEdge(1, 2)  
g2.addEdge(1, 3)  
g2.addEdge(2, 3)  
g2.addEdge(2, 4)  
g2.addEdge(3, 5)  
g2.addEdge(4, 5)  
g2.addEdge(3, 6)  
g2.addEdge(5, 6)  
g2.addEdge(6, 7)  
g2.addEdge(6, 8)  
g2.addEdge(7, 8)  
g2.addEdge(8, 9)  
g2.addEdge(9, 10)  
g2.addEdge(9, 11)  
g2.addEdge(10, 11)  
  
  
g3 = Graph(12)  
g3.addEdge(0, 1)  
g3.addEdge(0, 2)  
g3.addEdge(2, 3)  
g3.addEdge(2, 4)  
g3.addEdge(2, 5)  
g3.addEdge(4, 5)  
g3.addEdge(4, 6)  
g3.addEdge(5, 6)  
g3.addEdge(3, 7)  
g3.addEdge(6, 7)  
g3.addEdge(7, 8)  
g3.addEdge(6, 9)  
g3.addEdge(6, 10)  
g3.addEdge(8, 11)  
  
  
  
print ("\n\n\n\nGraph1\n\n Original graph")  
print(g1.graph)  
print (" {vertice의 인덱스: [연결된 노드들]}의 형식\n")  
g1.project3()  
  
print ("\n\n\n\nGraph2\n\n Original graph")  
print(g2.graph)  
print (" {vertice의 인덱스: [연결된 노드들]}의 형식\n")  
g2.project3()  
  
print ("\n\n\n\nGraph3\n\n Original graph")  
print(g3.graph)  
print (" {vertice의 인덱스: [연결된 노드들]}의 형식\n")  
g3.project3()

●input

텍스트, 화이트보드이(가) 표시된 사진

자동 생성된 설명

텍스트, 화이트보드이(가) 표시된 사진

자동 생성된 설명

텍스트, 화이트보드이(가) 표시된 사진

자동 생성된 설명

●output

위의 세가지 input을 프로그래밍하여 결과를 출력해보았다.

파이썬의 dictionary자료 구조를 이용하여

original graph를 출력했고 {index:[connected vertice1,connected vertice2, ···]의 형식}

그래프의 각 vertice의 index는 0부터 차례대로 숫자를 매겨 정했으며

DSF의 알고리즘은 index 0부터 진행된다.

num값은 DSF알고리즘상 지나가는 순서를 나타낸 값이고

low값은 DSF 알고리즘에서 나타나는path를 지나지 않고 갈수있는 num이 제일작은

vertice의 num값이다 출력결과는 다음페이지에 달아 놓았다.

그리고 마지막으로 해당 그래프의 articulation point들을 출력했다.

모든 결과가 위의 input과 일치하였으며 성공적으로 프로그래밍 되었다.

출력결과는 다음페이지에 달아놓았다.스크린샷이(가) 표시된 사진

자동 생성된 설명