**그래프(DFS, BFS)**

그래프란? 정점(vertex)와 간선(edge)로 이루어진 자료구조의 일종

단순 경로, 단순 사이클이란? 한 정점을 두 번 이상 방문하지 않는 경로/사이클

**방향 그래프**: 그래프의 모든 간선이 방향성을 띄는 그래프

**무향 그래프**: 그래프의 모든 간선이 방향성을 띄지 않는 그래프

**다중 간선(Multiple Edges):** 두 정점 사이에 여러 개의 간선이 있는 그래프 (이 때 각 간선은 엄연히 다른 간선이다)

**루프(loop):** 간선의 양 끝 점이 같은 경우

**차수(degree):** 해당 정점에 연결된 간선의 수 – 방향 그래프인 경우 진입 차수(in-degree)와 진출 차수(out-degree)를 나누어서 생각한다.

**연결 요소(connected component):** 원 그래프 G에서 부분 그래프 G’중에 속하는 서로 다른 두 정점의 경로가 존재하는 부분 그래프. 이때 다른 연결 요소의 정점과 연결하는 경로가 있으면 안된다.

**완전 그래프(complete graph):** 서로 다른 두 정점을 골랐을 때 항상 두 정점이 인접하는 그래프

**인접 행렬(adjacency matrix):** 정점의 개수를 v개라고 할 때 v\*v행렬로서 임의의 정점 I,j에 대해 i와 j가 인접하다면 1, 그렇지 않다면 0인 2차원 배열

간선 수가 정점 수보다 크면 사용

**인접 리스트(adjacency list):** 정점의 개수를 v개라고 할 때 v개의 리스트를 가지는 2차원 배열(메모리 상으로 인접 행렬보다 유용, a에서 b로 가는 경로가 있는지를 찾을 때의 시간 복잡도가 상대적으로 크다.)

정점 수가 간선 수보다 크면 사용

|  |  |
| --- | --- |
| DFS(Depth First Search) | BFS(Breadth First Search) |
| 1. 스택을 이용  2. 지금 정점과 인접한 정점 하나씩 방문  3. 갈 수 있는 정점이 없으면 스택에서 pop | 1. 큐를 이용  2. 지금 정점과 인접한 정점을 모두 방문  3. 갈 수 있는 정점이 없으면 큐에 담지 않음 |

#1260 – DFS와 BFS

|  |
| --- |
| 여러 개의 정점으로 이동이 가능한 상황에서 그 중에 제일 값이 작은 노드를 선택해서 이동해야 하기 때문에 dfs를 풀 때에는 깊이 우선 탐색인데, 경로의 총 개수를 세는 경우에는 재귀를 이용하는 것이 편하지만 그게 아니라 하나의 경로만 출력하는 경우리면 스택을 이용해서 현재 이동 가능 경로가 있는지 ‘can\_go’변수를 이용해 확인하면서 깊이 우선으로 탐색을 한다. |

|  |  |
| --- | --- |
| *def* dfs(v):   *#깊이 우선*      visited = [0]\*n      stack = []      stack.append(v)      visited[v] = 1  *print*(v+1, end = *' '*)  *while* stack:          now = stack[-1]          can\_go = False  *for* k in range(n):  *if* board[now][k] == 1:  *if* visited[k] == 0:                      visited[k] = 1                      can\_go = True                      stack.append(k)  *print*(k+1, end = *' '*)  *break*  *if* not can\_go:              stack.pop() | *from* collections *import* deque  *def* bfs(v):   *#너비 우선*  *global* visited      file = deque()      file.append(v)      visited[v] = 1  *while* file:          now = file.popleft()  *print*(now+1, end = *' '*)  *for* k in range(n):  *if* board[now][k] == 1:  *if* visited[k] == 0:                      visited[k] = 1                      file.append(k) |

#1012 - 유기농 배추

|  |
| --- |
| *import* sys  t = int(sys.stdin.readline())  *from* collections *import* deque  *def* count(board,x,y):  *global* ans      dx,dy = [-1,1,0,0],[0,0,-1,1]      file = deque()      file.append((x,y))  *while* file:          a,b = file.popleft()          board[a][b] = 0  *for* k in range(4):              i,j = a+dx[k], b+dy[k]  *if* 0<= i<n and 0<= j<m:  *if* board[i][j] == 1:                      board[i][j] = 0                      file.append((i,j))      ans += 1    *for* \_ in range(t):      m,n,k = map(int, sys.stdin.readline().split())      board = [[0]\*m *for* \_ in range(n)]  *for* \_ in range(k):          a,b = map(int, sys.stdin.readline().split())          board[b][a] = 1      ans = 0  *for* x in range(m):  *for* y in range(n):  *if* board[y][x] == 1:                  count(board, y,x)  *print*(ans) |

#1707 – 이분 그래프

|  |
| --- |
| *import* sys  t = int(sys.stdin.readline())  *def* dfs(v,num):      res[v] = num      ans = True  *for* k in board[v]:  *if* res[k] == -1:  *if* not dfs(k, abs(1-res[v])):                  ans = False  *else*:  *if* res[k] == res[v]:  *return* False  *return* ans  *for* \_ in range(t):      v,e = map(int, sys.stdin.readline().split())      board = [[]*for* \_ in range(v)]  *for* \_ in range(e):          x,y = map(int, sys.stdin.readline().split())          board[x-1].append(y-1)          board[y-1].append(x-1)      res = [-1]\*v      ans = *'YES'*  *for* k in range(v):  *if* res[k] == -1:  *if* not dfs(k,1):                  ans = *'NO'*  *break*  *print*(ans) |

|  |
| --- |
| 이 방법은 기존에 내가 이 문제를 풀었던 방법과 달리 dfs(v,n)에서 v번쨰 노드의 그룹이 정해지지 않았을 때에 n번그룹이라면 과연 이 그래프가 이분 그래프를 충족시킬 수 있을지 True, False로 확인하는 것이다. 한번이라도 노드에서 False가 나온다면 이는 이 그래프가 이분 그래프를 완성시킬 수 없음을 의미하기 때문에 그냥 무조건 ‘NO’를 출력시킨다.  아래 방법은 bfs를 이용해서 직관적으로 해결이 가능해서 직접 해결한 것인데 중요한 것은 하나의 노드만 탐색해서 되는 것이 아니라 res리스트의 -1이 없을 때까지 탐색해야 한다는 것이었다. |

|  |
| --- |
| *def* check(node):      file = deque()      res[node] = 0      file.append(node)  *while* file:          now = file.popleft()  *for* k in board[now]:  *if* res[k] == -1:                  res[k] = abs(res[now]-1)                  file.append(k)  *else*:  *if* res[k] == res[now]:  *return* False  *return* True  *for* \_ in range(t):      v,e = map(int, sys.stdin.readline().split())      board = [[]*for* \_ in range(v)]  *for* \_ in range(e):          a,b = map(int, sys.stdin.readline().split())          board[a-1].append(b-1)          board[b-1].append(a-1)      res = [-1]\*v      ans = *'YES'*  *for* k in range(v):  *if* res[k] == -1:  *if* not check(k):                  ans = *'NO'*  *break*  *print*(ans) |

#11724 – 연결 요소의 개수

|  |
| --- |
| 이전에 풀었던 scc알고리즘에서 제일 중요했던 부분이 바로 이렇게 루트 노드를 찾아내는 방법이었다. 그랬기 때문에 좀더 정확하게, 한번에 해결이 가능했던 것이라고 생각한다.  bfs(v, num)으로 해서 v번 노드의 루트를 너비 우선 탐색으로 num이라는 수로 변경하는 것이고, 이 또한 이전 이분 그래프 문제와 마찬가지로 방문하지 않은 노드가 없을 때까지 시행했다. |

|  |
| --- |
| *import* sys  n,m = map(int, sys.stdin.readline().split())  board = [[]*for* \_ in range(n)]  *for* \_ in range(m):      u,v = map(int, sys.stdin.readline().split())      board[u-1].append(v-1)      board[v-1].append(u-1)  res = [int(i) *for* i in range(n)]  *from* collections *import* deque  visited = [0]\*n  *def* check(v,num):      file = deque()      file.append(v)  *while* file:          v = file.popleft()  *for* k in board[v]:  *if* res[k] == k:  *if* visited[k] == 0:                      res[k] = num                      visited[k] = 1                      file.append(k)  *else*:                      res[k] = num                      check(k,num)  *else*:  *if* res[k] != num:                      res[k] = num                      check(k,num)  *for* k in range(u):  *if* visited[k] == 0:          visited[k] = 1          check(k,k)  *print*(len(set(res))) |

#2667 – 단지 번호 붙이기

|  |
| --- |
| *import* sys  n = int(sys.stdin.readline())  board = [list(map(int, sys.stdin.readline().strip())) *for* \_ in range(n)]  *from* collections *import* deque  *def* move(x,y):  *global* board,ans      file = deque()      count = 1      board[x][y] = 0      file.append((x,y))      dx,dy = [-1,1,0,0],[0,0,-1,1]  *while* file:          i,j = file.popleft()  *for* k in range(4):              a,b = i+dx[k], j+dy[k]  *if* 0<=a<n and 0<=b<n:  *if* board[a][b] == 1:                      board[a][b] = 0                      file.append((a,b))                      count += 1      ans.append(count)  ans = []  cnt = 0  *for* i in range(n):  *for* j in range(n):  *if* board[i][j] == 1:              move(i,j)              cnt += 1  *print*(cnt)  ans.sort()  *for* i in ans:  *print*(i) |

#1697 – 숨바꼭질

|  |
| --- |
| 중요한 것을 중복 방문을 막기 위해 다시 visit리스트를 매번 갱신해 주어야 한다는 것과 스택에서 해당 값을 pop해내기 전에 visit리스트를 갱신해 주어야 한다는 것이다. 그리고 visit리스트는 k도달 지점까지가 아니라 최대 가능 좌표의 값으로 설정해야 함에 주의하자. |

|  |
| --- |
| n,k = map(int, input().split())  visit = [0]\*100001  *from* collections *import* deque  file = deque()  *def* move(v):      file.append((v,0))      visit[v] = 1  *while* file:          a,b = file.popleft()  *if* a == k:  *return* b          num = [a-1, a+1, a\*2]  *for* i in num:  *if* 0<= i <= 100000:  *if* visit[i] == 0:                      file.append((i,b+1))                      visit[i] = 1  *print*(move(n)) |