from \_\_future\_\_ import annotations  
from typing import List, Optional  
from generic\_search import bfs, Node, node\_to\_path  
  
MAX\_NUM: int = 3  
  
  
class MCState:  
 def \_\_init\_\_(self, missionaries: int, cannibals: int, boat: bool) -> None:  
 self.wm: int = missionaries # 서쪽 강둑에 있는 선교사 수  
 self.wc: int = cannibals # 서쪽 강둑에 있는 식인종 수  
 self.em: int = MAX\_NUM - self.wm  
 self.ec: int = MAX\_NUM - self.wc  
 self.boat: bool = boat  
 # end of \_\_init\_\_  
  
 def \_\_str\_\_(self) -> str:  
 return ("서쪽 강둑에는 {}명의 선교사와 {}명의 식인종이 있다.\n"   
 "동쪽 강둑에는 {}명의 선교사와 {}명의 식인종이 있다.\n"  
 "배는 {}쪽에 있다.")\  
 .format(self.wm, self.wc, self.em, self.ec, ("west" if self.boat else "east"))  
 # end of \_\_str\_\_  
  
 def goal\_test(self) -> bool:  
 return self.is\_legal and self.em == MAX\_NUM and self.ec == MAX\_NUM  
 # end of goal\_test  
  
 @property  
 def is\_legal(self) -> bool:  
 if self.wm < self.wc and self.wm > 0:  
 return False  
  
 if self.em < self.wc and self.em > 0:  
 return False  
  
 return True  
 # end of is\_legal  
  
 def successors(self) -> List[MCState]:  
 sucs: List[MCState] = []  
 if self.boat:  
 if self.wm > 1:  
 sucs.append(MCState(self.wm - 2, self.wc, not self.boat))  
 if self.wm > 0:  
 sucs.append(MCState(self.wm - 1, self.wc, not self.boat))  
 if self.wc > 1:  
 sucs.append(MCState(self.wm, self.wc - 2, not self.boat))  
 if self.wc > 0:  
 sucs.append(MCState(self.wm, self.wc - 1, not self.boat))  
 if (self.wc > 0) and (self.wm > 0):  
 sucs.append(MCState(self.wm - 1, self.wc - 1, not self.boat))  
 else:  
 if self.em > 1:  
 sucs.append(MCState(self.wm + 2, self.wc, not self.boat))  
 if self.em > 0:  
 sucs.append(MCState(self.wm + 1, self.wc, not self.boat))  
 if self.ec > 1:  
 sucs.append(MCState(self.wm, self.wc + 2, not self.boat))  
 if self.ec > 0:  
 sucs.append(MCState(self.wm, self.wc + 1, not self.boat))  
 if (self.ec > 0) and (self.em > 0):  
 sucs.append(MCState(self.em + 1, self.ec + 1, not self.boat))  
 return [x for x in sucs if x.is\_legal]  
  
  
def display\_solution(path: List[MCState]):  
 if len(path) == 0:  
 return  
 old\_state: MCState = path[0]  
 print(old\_state)  
 for current\_state in path[1:]:  
 if current\_state.boat:  
 print("{}명의 선교사와 {}명의 식인종이 동쪽 강둑에서 서쪽 강둑으로 갔다.\n"  
 .format(old\_state.em - current\_state.em, old\_state.ec - current\_state.ec))  
 else:  
 print("{}명의 선교사와 {}명의 식인종이 서쪽 강둑에서 동쪽 강둑으로 갔다.\n"  
 .format(old\_state.wm - current\_state.wm, old\_state.wc - current\_state.wc))  
  
 print(current\_state)  
 old\_state = current\_state  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start: MCState = MCState(MAX\_NUM, MAX\_NUM, True)  
 solution: Optional[Node[MCState]] = bfs(start, MCState.goal\_test,  
 MCState.successors)  
 if solution is None:  
 print("답을 찾을 수 없습니다.")  
 else:  
 path: List[MCState] = node\_to\_path(solution)  
 display\_solution(path)