삼국지게임 풀이

문제 요약

- 문제에서 제시한 '삼국지 게임' 의 기능 3가지 구현
  - ally: 서로 다른 두 군주가 소속된 동맹을 결합합니다.
  - attack: 방어 영토에 인접한 공격 동맹과 방어 동맹의 인원의 절반을 보내고 차이를 계산합니다. 공격에 성공하면 새로운 장수가 군주가 되고 공격 동맹에 소속됩니다. 전투가 발생한 경우 적대관계가 형성됩니다.
  - recruit: 한 군주의 병사를 늘리거나 군주가 소속된 동맹의 병사를 일괄적으로 늘립니다.

사소한 문제 특징

- 각 군주는 영토 하나만 소유하고 영토를 차지하는 장수도 기존 군주와 겹치지 않음
- 최대 영토 크기는 25 * 25
- 초기 관계는 모두 중립 (초기 동맹 및 적대관계 없음)
- 동맹의 동맹은 동맹 (같은 동맹 소속에서 적대관계 형성 불가)
- 동맹의 적은 적이지만 적의 적은 항상 동맹이 아님
  - A와 B가 적대관계고 B와 C가 적대관계라도 C와 A가 동맹 관계가 아닐 수 있음
- 한 번 생긴 동맹/적대 관계는 사라지지 않음
- ally와 attack은 8,000회, recruit은 13,000회 호출

필요한 자료구조

- 이름을 정수 ID로 바꿔주는 함수
  - H(이름)->ID 함수 (hash table로 구현)
- 각 군주마다 2개의 리스트 (vector 또는 linked list)
  - Alliance(ID)->해당 ID와 동맹인 군주의 ID 리스트
  - Enemies(ID)->해당 ID와 적대인 군주의 ID 리스트
- 현재 군주의 정보 (단순 배열로 저장)
  - Table(ID)->{영토의 병사 수, 위치}
- 각 위치마다 군주 ID (2차원 배열로 저장)
  - F(x, y)->해당 영토를 소유하는 군주 ID

1. Ally

- 입력된 두 군주 이름에 대한 ID 값을 얻습니다.
  - By hash table (앞으로 hash table 비용은 상수로 생략)
- 두 군주가 소속된 각 동맹이 서로 무슨 관계인지 파악합니다.
  - 아무런 관계가 없으면 동맹을 맺습니다.
  - 이미 동맹/적대관계면 문제 명세에 따라 음수를 반환합니다.

1. Ally

- 임의의 두 군주가 소속한 각 동맹의 관계 파악하는 법
  - 둘 중 하나의 Alliance 리스트와 Enemies 리스트를 살펴보면 됩니다.
    - Linear time
- 문제점: 각 군주마다 리스트를 유지하면 메모리와 시간 소요가 큼
  - 매번 업데이트(동맹관계 생성)마다 평균 제곱의 시간 소요 (소속된 군주의 리스트를 모두 최신화 필요)
- 해결책: 각 동맹마다 리스트를 관리할 대표 군주 선정
    - Union-Find

1. Ally

- 동맹 결합 방법 by union-find
  - 두 군주 각각 대표 군주 찾기 (find 연산)
  - 한 군주가 다른 군주의 대표직을 물려 받기 (union 연산)
    - Ex. 한 군주의 부모를 다른 군주로 설정 (대표를 tree의 root로 취급)
    - 대표직을 물려 받은 군주는 물려준 군주의 Alliance와 Enemies 리스트 병합
      - Vector의 insert 또는 linked list의 두 리스트 concatenation 연산으로 가능 (linked list 추천)
      - 긴 vector에 작은 vector를 붙이는 테크닉을 사용하면 시간이 절약되지만 관리하는 리스트가 2개여서 사용하기 어렵고 샘플 테스트케이스에서는 시간차이가 적음
  - 시간복잡도
    - Find: $O(\alpha(N^2))$ 또는 $O(\lg(N^2))$ (구현 따라 다름)
    - Union: O(1) (linked list) 또는 $\Theta(\lg(N^2))$ (vector insert + 큰 리스트에 작은 리스트 붙이기 테크닉)

1. Ally

- 보너스. Union-find를 통해 동맹관계 파악은 find 연산으로 가능
- 페널티. 적대 관계 탐색할 때 각 ID마다 find 연산 취해야 함
  - 상황에 따라 Enemies list가 대표의 자식 군주 ID를 유지
  - Ex. A와 B가 적대관계인 상황에서 B와 C가 동맹을 맺고 C가 대표 군주가 된 경우 A가 소속된 동맹의 Enemies 리스트에 C가 아닌 B가 있음
  - ->find 연산으로 업데이트 필요
- ∴ 관계 파악 기능은 $(O(\alpha(N^2))$ or $O(\lg(N^2)))$ * $O(N^2)$로 가능

2. Attack

- 입력된 두 군주 이름에 대한 ID 값을 얻습니다.
  - By hash table
- 두 군주가 소속된 각 동맹이 서로 무슨 관계인지 파악합니다.
  - 동맹관계면 전투를 하지 않습니다.
  - 그 외의 경우면 방어하는 영토 주변의 군주 ID를 조사합니다.
  - 각 군주가 공격 동맹 소속인지 방어 동맹 소속인지 파악합니다.
    - 공격 인원이 없으면 전투를 하지 않지만 인원이 있으면 전투 시행 및 적대 관계 형성
    - 공격 또는 방어 동맹에 소속되면 문제 명세 따라 방어 영토로 병사 절반 파견
  - 파견된 인원만큼 방어 영토 병사 수 증감합니다.
  - 남은 병사 수가 음수면(공격 인원이 더 많으면) 해당 영토에 새 군주를 선임하고 공격 동맹과 동맹을 체결합니다.

2. Attack

- 방어 영토 주변의 영토가 최대 8개
  - 주변 영토에 있는 군주의 ID는 O(1)에 계산 by 배열
  - 두 군주의 관계 파악 연산을 상수 번 수행 (각 영토마다 공격 방어 2번 수행)
- 전투가 발생하면 각 동맹 Enemies list에 원소 추가
  - 적대 관계는 직접 전투에 포함된 동맹만 적용되므로 단순 list 추가
  - Recall. 적의 적은 아무런 관계를 내포하지 않는다.

2. Attack

- 절반 파견 및 방어 영토 병사 수 증감은 단순 계산으로 가능
  - 현재 군주의 정보를 배열에 저장하여 O(1)에 계산
- 공격을 성공하면 새로운 군주에 대한 자료구조 초기화 및 기타 처리
  - 새 ID부여, Hash table에 삽입
  - list 2개, 현재 군주 정보, 기타 자료구조(ex. Union-find에 쓰이는 자료구조) 초기화
  - 해당 영토의 현재 군주 ID 업데이트
  - 침략당한 군주에 대한 추가 처리 필요 없음
  - 시간복잡도: O(1)
- 기존 공격 동맹과 동맹 체결은 Ally API와 동일

2. Attack

- 총 시간복잡도: $O(\alpha(N^2))$ or $O(\lg(N^2))$
  - 여러 번 관계 파악 연산을 수행하므로 상수가 큼

3. Recruit

- 입력된 군주 이름에 대한 ID 값을 얻습니다.
  - By hash table
- 옵션 0의 경우 해당 군주의 병사 수만 증가합니다.
- 옵션 1의 경우 해당 군주가 소속된 동맹의 모든 군주의 병사 수가 증가합니다.

3. Recruit

- 옵션 0: 해당 군주 정보만 업데이트
  - O(1)
- 옵션 1: 해당 동맹의 Alliance 리스트 내 군주 정보 업데이트
  - Find 연산 + Alliance 리스트 순회 및 해당 군주 정보 업데이트
  - 영토를 빼앗긴 군주(병사 수가 음수)에 대해서는 업데이트를 하지 않음
  - $O(N^2)$

End