가희와 함께 하는 3회 코딩테스트

문제		의도한 난이도	출제자
A	가희와 무궁화호	**Medium**	chogahui05
B	가희와 탑	**Hard**	chogahui05
C	가희와 쓰레기 놀이	**Challenging**	chogahui05
D	가희와 베개	**Challenging**	chogahui05
E	가희와 btd5 2	**Challenging**	chogahui05

A. 가희와 무궁화호

`implementation, regex`
출제진 의도 – **Medium**

- 제출 78, 정답 20 (정답률 26.923%)
- 처음 푼 사람: **pichulia**, 28분
- 출제자: chogahui05

A. 가희와 무궁화호

- 먼저 테이블의 정보를 코드에 작성하는 방법을 생각해 봅시다.
  - $table$ 에 역 정보가 매우 많습니다.
  - 이걸 일일히 적기에는 노가다성 작업을 필요로 합니다.

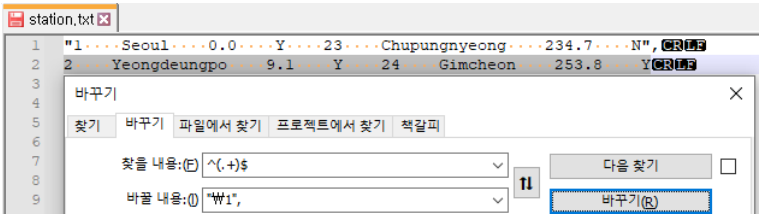
- 표를 그대로 복사하면 아래와 같이 붙여넣기가 될 것입니다.
  - $abcdefgh$
  - 이걸 "$abcdefgh$", 로 바꾸고 싶습니다.
- 그러면 전체 내용을 $capture$ 한 다음에
- $capture$ 한 것의 앞에는 "을, 뒤에는 ", 을 넣으면 됩니다.
- $capture$ 하는 것은 $regex$ 에서 $(pattern)$ 입니다.

A. 가희와 무궁화호

- 전체 내용을 $capture$ 하는 것은 $regex$ 에서 ^(.+)$ 입니다.
  - 정규식에서 맨 앞에 오는 ^ 는 시작
  - 맨 끝에 오는 $는 끝을 의미합니다.
  - 그러면 ^ 과 $ 사이에 있는 (.+)는 전체를 의미하겠네요?
- 따라서 위의 $regex$ 는 전체를 캡쳐하는 것입니다.

A. 가희와 무궁화호

- $notepad$ 에서는 "₩n"을 n 번째로 캡쳐된 것으로 정의하고 있습니다.
- 따라서 바꿀 내용은 "₩1", 로 하면 됩니다.



A. 가희와 무궁화호

- 남은 것은 표정 속도의 계산입니다. 표정 속도는 아래 두 정보를 알면 됩니다.
  - 이동한 거리
  - 걸린 시간

A. 가희와 무궁화호

- 이동한 거리는 각 역간 거리 차이에 절댓값을 씌우면 됩니다.
- 시간은 $hh : mm$ 의 형식을 분으로 변환하면 됩니다.
- 하루가 바뀌는 경우만 처리를 하면 끝납니다.

B. 가희와 탑

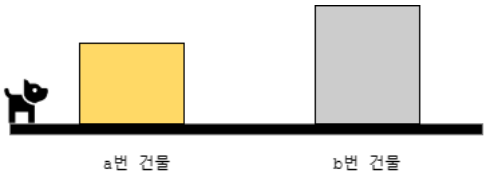
`constructive, greedy`
출제진 의도 – **Hard**

- 제출 248번, 정답 31명 (정답률 13.306%)
- 처음 푼 사람: **shihm1212**, 16분
- 출제자: chogahui05

B. 가희와 탑

- ✓ 불가능한 경우부터 생각해 봅니다.
- ✓ 가희와 단비가 동시에 볼 수 있는 건물의 수는 많아야 1개입니다. 왜?
  - 가희와 단비가 동시에 볼 수 있는 건물의 수는 **2개 이상**이다.
  - 위 명제가 모순이기 때문입니다. 이걸 보여 봅시다.

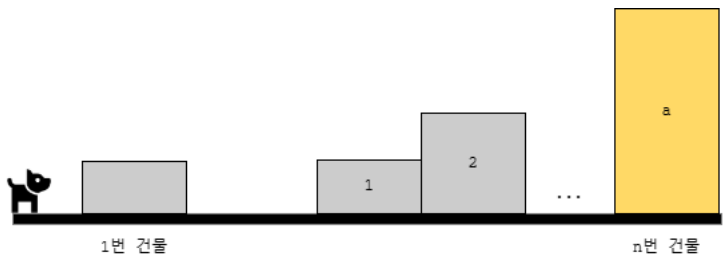
B. 가희와 탑



- 가희와 단비가 2개의 건물을 동시에 볼 수 있다 합시다. a는 b의 왼쪽에 있어요.
  - a의 높이가 b의 높이보다 작다면 가희는 2개의 건물을 볼 수 있어요.
  - 그런데, 단비 입장에서는 a가 안 보입니다. 그렇죠?
  - 고로 모순입니다.

B. 가희와 탑

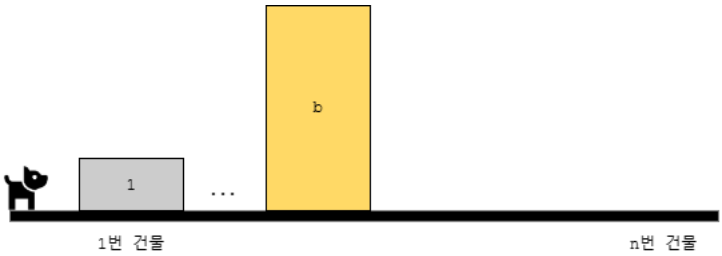
- ✓ $a + b > n + 1$ 인 경우 -1 을 출력해 주면 됩니다.
- ✓ 이제 가능한 경우 중, 코너케이스를 하나씩 정복해 봅시다.
  - $a = b = 1$ 인 경우는 그냥 1 을 n 번 출력해 주면 됩니다.
  - $b = 1$ 인 경우에는 아래와 같이 배치하면 되겠습니다.



B. 가희와 탑

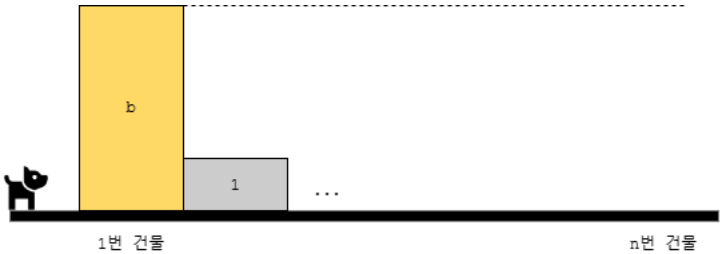
- $a = 1$인 경우에는 어떨까요?
- 생각보다 이 함정에 적지 않게 걸렸으니 차근 차근 해결해 봅시다.
  - 단비가 볼 수 있는 건물이 b개란 이야기는 최대 높이을 가진 건물이 최소 b 이상이란 이야기입니다.
  - 가희는 최대 높이를 가진 건물 b를 볼 수 있습니다. 여기서 문제.
  - 이 건물은 어느 위치에 와야 할까요?

B. 가희와 탑



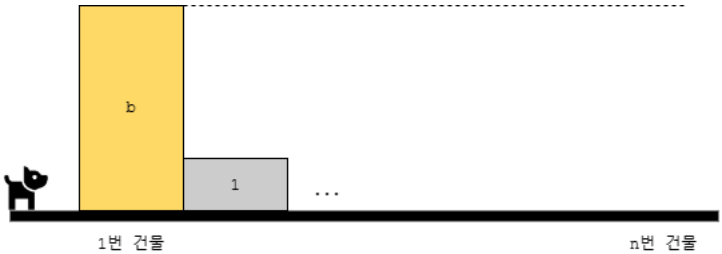
- 만약에 높이 1짜리가 1번째 위치에 왔다고 해 봅시다.
- 그러면 높이가 제일 큰 건물은 다른 위치에 오게 되어 있습니다.
- 가희가 볼 수 있는 건물은 2개 이상이네요?

B. 가희와 탑



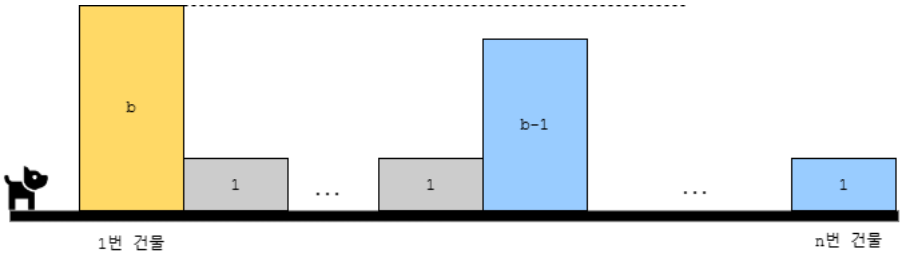
- 따라서 높이가 제일 높은 건물은 1번째 위치에 와야 합니다.
- 가희는 1번 건물을 무조건 볼 수 있기 때문입니다.

B. 가희와 탑



- 그리고 그림을 다시 보면, 2번 건물에 1짜리가 놓여져 있는 것을 볼 수 있습니다.
- 가능하다면 그렇게 놓는 것이 좋습니다. 왜?
- 어짜피 가희는 2번 건물을 못 보기 때문입니다.

B. 가희와 탑

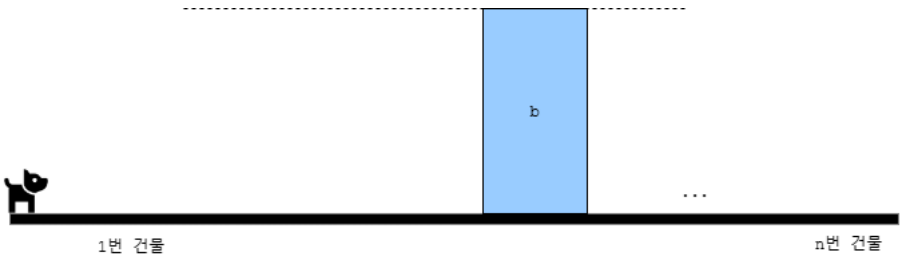


- 다음에 단비는 b개의 건물을 볼 수 있다고 했으므로
- 위와 같이 배치하면 됩니다.
- 조심해야 할 것은 이미 높이가 b인 건물이 1번째 위치에 있다는 것입니다.

- 코너 케이스를 처리했으니 남아 있는 케이스는 2개입니다.
  - $a < b$ 또는 $a = b$인 경우
  - $a > b$

B. 가희와 탑

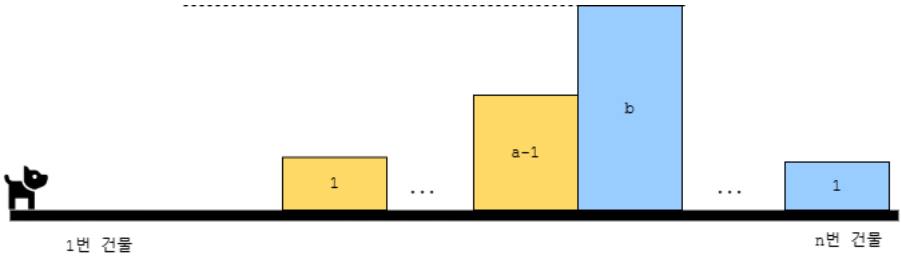
- 먼저 $a < b$이거나 $a = b$인 경우에는 건물의 최대 높이의 최소값은 b입니다.
- 이 사실을 바탕으로 그림을 그려 봅시다.



B. 가희와 탑

- 이제 이 군청색을 기준으로 오른쪽으로는 높이가 1씩 내려가게끔 배치하면 됩니다.
- 왼쪽은 어떻게 배치하면 될까요?
  - 높이가 b인 군청색 건물은 무조건 볼 수 있습니다.
  - 따라서 높이가 1인 건물부터 $a-1$인 건물까지 올라가게끔 배치하면 됩니다.

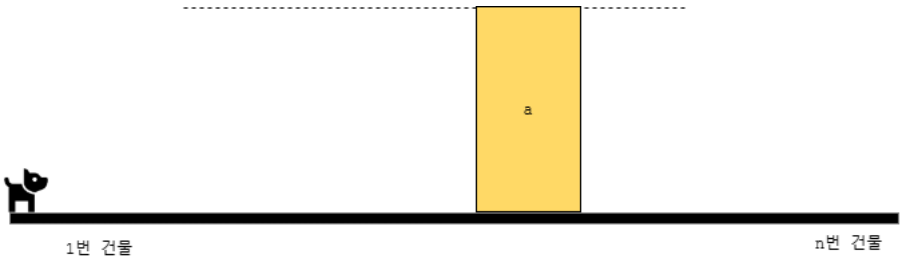
B. 가희와 탑



- 설명한 부분을 그림으로 그리면 위와 같습니다.
- 나머지 부분은 높이가 1짜리인 건물로 도배하면 됩니다.

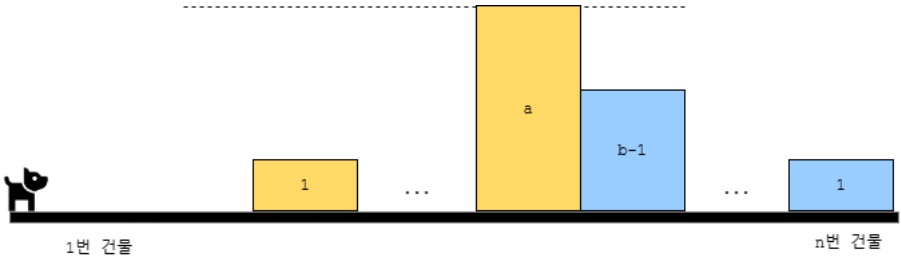
B. 가희와 탑

- 먼저 $a > b$ 인 경우에는 건물의 최대 높이의 최소값은 a 입니다.
- 이 사실을 바탕으로 그림을 그려 봅시다.



- 이제 이 노란색을 기준으로 왼쪽으로는 높이가 1씩 내려가게끔 배치하면 됩니다.
- 오른쪽은 어떻게 배치하면 될까요?
  - 높이가 a인 군청색 건물은 무조건 볼 수 있습니다.
  - 따라서 높이가 1인 건물부터 $b-1$인 건물까지
  - 우측 끝에서부터 좌측으로 올라가게끔 배치하면 됩니다.

B. 가희와 탑



- 설명한 부분을 그림으로 그리면 위와 같습니다.
- 나머지 부분은 높이가 1짜리인 건물로 도배하면 됩니다.

C. 가희와 쓰레기 놀이

`data structure`, `bfs`, `dfs`, `implementation`
출제진 의도 – **Challenging**

- 제출 23번, 정답 4명 (정답률 17.391%)
- 처음 푼 사람: **pichulia**, 131분
- 출제자: chogahui05

C. 가희와 쓰레기 놀이

- 가비지 콜렉터 gc를 구현하라는 문제입니다.
- 삭제 연산은 최대 20번이니 $\mathcal{O}\left(20(O+E)\right)$ 즈음에 처리할 수 있을 겁니다.
- $\mathcal{O}\left(20(O+E)\right)$가 가볍지 않습니다.
- $refence$ 관계가 추가되고 삭제되는 쿼리가 매우 많으니 이 연산을 잘 처리해야 합니다.

C. 가희와 쓰레기 놀이

- 객체의 id는 최대 10^9 입니다.
- 참조 관계의 id도 최대 10^9 입니다.
- 각 객체는 참조 관계로 자기 자신, 혹은 다른 객체와 연결되어 있습니다.

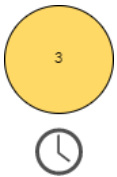
- 따라서 아래와 같은 구조를 생각할 수 있습니다.
  - 객체 id를 key 값으로 가지는 HashMap h_1
  - HashMap h_1의 value는 레퍼런스 id를 가지는 HashMap임
  - 레퍼런스 관계가 추가, 삭제될 때 마다 업데이트
- 연결 관계를 토대로 M이나 m연산이 발생할 때 마다 정리를 해 주면 됩니다. 그런데

- 20번의 M과 m 연산이 가볍지 않습니다. 게다가
  - $hash$에 값을 넣는 연산도 생각보다 가볍지 않은 판에
  - $hash$의 $hash$에서 값을 찾아, 삭제하는 연산은 얼마나 무거울까요?
  - 이러한 연산들이 쌓이면, 3.5초가 넉넉해 보이지만 절대로 넉넉하지 않습니다.
- 핵심 아이디어는 $hash$에서 값을 삭제하는 연산을 없앤다는 것입니다.

- 이게 무슨 소리인가? gc의 핵심은 아래 2가지입니다.
  - 루트로부터 도달 가능한 것들을 $mark$ 합니다.
  - 그렇지 않은 것은 제거합니다.
- 이 연산이 연결 관계가 삭제될 때마다 일어나면 어떨까요? 당연하게도 안 좋을 겁니다.
- $System.gc()$를 호출하지 말아야 하는 이유도 이와 동일합니다.

- 그렇다고 해도 연결 관계를 삭제하는 것을 없애면 안 되지 않나요? 맞습니다.
  - 삭제하는 대신에 메타 정보를 $mark$ 해서 관리합니다.
  - 중요한 건 M 과 m 연산이기 때문입니다.
- 어떤 메타 정보를 어떻게 저장하고 관리해야 하는지가 문제입니다.

C. 가희와 쓰레기 놀이



- 연결 관계 1이 시각 3에 생성되었습니다.
- 이 경우 연결 관계 1은 살아 있는 것입니다.

C. 가희와 쓰레기 놀이

- 연결 관계 1이 시각 3에 생성되었습니다.
- 다음에 연결 관계 1이 시각 4에 제거되었습니다.
- 이 경우 연결 관계 1은 제거된 것입니다.

C. 가희와 쓰레기 놀이

- 연결 관계 1은 시각 3에 생성되었습니다.
- 다음에 연결 관계 1에 대한 최신 이벤트는 시각 4에 있었습니다.
- 생성된 시각보다 최신 이벤트가 더 뒤에 있으니 제거되었다고 하면 되겠네요.

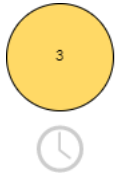
C. 가희와 쓰레기 놀이

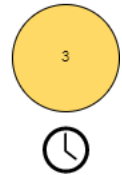
- 연결 관계 1은 시각 3에 생성되었습니다.
- 다음에 연결 관계 1이 시각 4에 제거되었습니다.
- 다음에 연결 관계 1이 시각 5에 다시 생성되었습니다.

C. 가희와 쓰레기 놀이

- ✓ 이 경우, 연결 관계 1을 찾습니다.
- ✓ 다음에 연결 관계 1에 대해 새로운 정보를 업데이트 합니다.
- ✓ 이래도 되긴 합니다만 더 최적화 하는 방법이 없을까요?
  - 이미 기존에 $invalid$ 된 연결 관계는 건너 뛰어도 됩니다.
  - M 과 m 연산을 수행하는 과정에서 건너 뛸 것이기 때문입니다.
  - 대신에 연결 관계 1에 대한 정보를 새롭게 추가하면 됩니다.

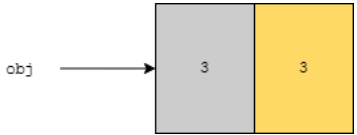
C. 가희와 쓰레기 놀이





✓ 제가 설명한 부분을 도식화 시키면 위와 같습니다.

C. 가희와 쓰레기 놀이



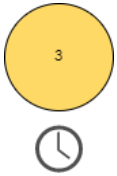
- 오브젝트에 딸린 연결 관계는 위와 같이 그려집니다.
- 회색은 무효화 된 것, 노란색은 유효한 연결 관계입니다.

C. 가희와 쓰레기 놀이

- ✓ 레퍼런스 관계는 추가만 되고 삭제는 안 일어납니다.
- ✓ 순서가 중요하지 않습니다.
- ✓ 따라서 $vector$와 같은 다이나믹 배열이 적당합니다.
  - 객체 id를 key 값으로 가지는 HashMap h_1
  - HashMap h_1의 value는 레퍼런스 정보를 가지는 vector임
  - 레퍼런스 관계가 추가될 때만 추가

- 이제 4, 5, 6번 연산을 어떻게 해야 할지 정리해 봅시다. 먼저 4, 5번 연산.
  - ref_id에 대한 정보를 추가합니다.
  - ref_id가 시각 t에 추가되었다는 정보를 obj에 연결되는 연결 정보에 추가합니다.
  - ref_id가 시각 t에 이벤트가 일어났다는 정보도 추가합니다.

- 6번 연산.
  - ref_id에 대한 정보가 제거되는 것입니다.
  - 그런데 제거되었다는 정보만 $mark$ 하면 됩니다.
  - 시각 t에 ref_id에 대한 이벤트가 일어났다는 정보를 추가합니다.



- 이에 따라 4, 5, 6번 연산을 시뮬레이션 해 봅시다.
- 시각 1에 연결 관계 1이 추가되었습니다.
- obj에 연결 관계 1이 시각 1에 추가되었다는 정보를 추가합니다.
- 연결 관계 1의 이벤트 최신화 시각이 1이라는 정보를 추가합니다.
- 두 수치가 1, 1로 같으므로 유효합니다.

C. 가희와 쓰레기 놀이

- 연결 관계 1이 시각 3에 제거되었습니다. 이벤트 최신화 시각은 3입니다.
- 추가된 시각은 1이지요?
- 추가된 시각 1보다 최신화 시각 3이 뒤에 있으므로 유효하지 않습니다.

- 이제 1번과 2번을 해결해 봅시다.
- 1번과 2번은 연결 관계가 강한 것만 탐색하는지, 약한 것도 같이 탐색하는지가 다릅니다.
- 즉, 연결 관계의 강함 정도가 특정 수치 이상인 것만 탐색하는 것이 공통점입니다.
- 고로 강함의 정도를 bfs 함수의 인자로 넘기면 됩니다. 1번과 2번 해결.

- 연결 관계에는 어떤 정보를 저장하면 좋을까요?
  - id 값
  - 연결하는 오브젝트 2개
  - 연결 관계가 만들어진 시각
  - 연결의 강함 정도

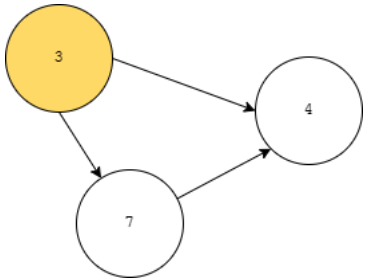
C. 가희와 쓰레기 놀이

- M 과 m 연산에서도 연결 관계가 제거될 수 있습니다. 언제?
- 연결하는 오브젝트 중 하나라도 유효하지 않을 때
- 이 때에도 이벤트 최신화 시각에 반영해 주면 됩니다.

C. 가희와 쓰레기 놀이

- 오브젝트는 삭제를 하지 않나요?
- 삭제를 하면 최대 $\mathcal{O}\left(20O\right)$ 만큼의 무거운 연산을 해야 합니다.
- 오브젝트는 $root$ 일 때 절대로 삭제되지 않습니다.
- 설령 오브젝트가 남아 있다고 해 봅시다.

C. 가희와 쓰레기 놀이



- $root$ 가 아닌 오브젝트에 $root$ 로부터 접근하기 위해서는
- 참조 관계를 최소 하나 이상 거쳐야 합니다.
- 그런데 접근을 할 수 있느냐의 여부는 bfs 단계에서 판정을 하고 있어요.

C. 가희와 쓰레기 놀이

- ✓ $root$ 로부터 접근할 수 없는 것들은 제거되었다고 판정하면 됩니다.
- ✓ $root$ 로부터 접근할 수 없는 것들을 제거하는 연산도 1, 2번에서 수행하고 있습니다.
- ✓ 출력해야 하는 답도 이와 동일합니다.

- 정리하면
  - $hash$ 나 $tree$ 에서 제거 연산은 생각보다 가볍지 않습니다.
  - 고로 이 부분을 없애면 실행 속도가 눈에 띄게 상승합니다.
  - gc 도 참조 관계가 제거될 때 마다 매번 오브젝트를 정리하지 않습니다.

D. 가희와 베개

`bfs`, `dfs`, `graph theory`, `union find`
출제진 의도 – **Challenging**

- 제출 22번, 정답 4명 (정답률 18.182%)
- 처음 푼 사람: **pichulia**, 185분
- 출제자: chogahui05

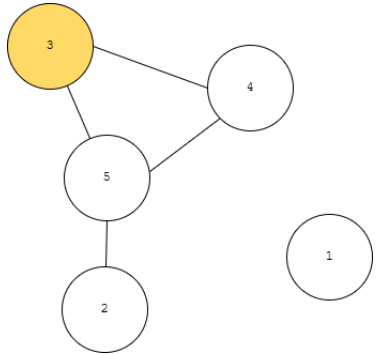
D. 가희와 베개

- ✓ bfs와 dfs를 배울 때 어떤 용어가 생각나시나요?
- ✓ $Component$가 가장 먼저 생각나지 않나요? 그리고 연결 요소
- ✓ 이 문제의 1번째 출제의도입니다. $ConnectedComponent$가 무엇인가? 왜 쓰는가?

D. 가희와 베개

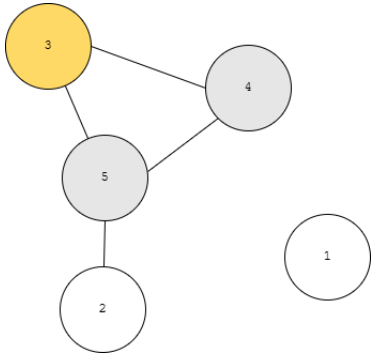
- 이 문제는 양방향 간선으로 연결되어 있습니다. 고로
  - a에서 b로 연결되었다면, b에서 a로도 도달 가능합니다.
  - a에서 b로, b에서 c로 도달 가능하면
    - a에서 c로 도달 가능합니다.
    - c에서 a로 도달 가능합니다.

D. 가희와 베개



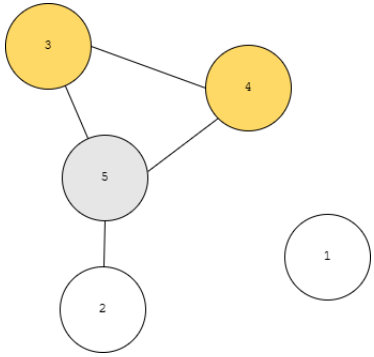
- 이 사실을 bfs와 dfs가 잘 이용하고 있습니다.
- 3번에서 bfs를 수행해 봅시다.

D. 가희와 베개



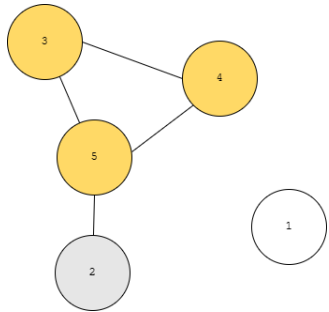
- 3과 인접한 4와 5가 큐에 들어옵니다.
- 고로 3과 4와 5는 서로 갈 수 있습니다.

D. 가희와 베개



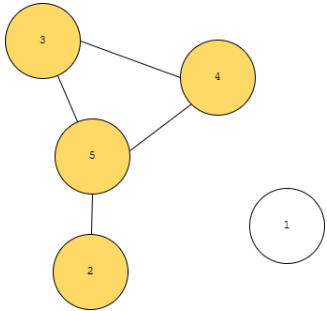
- 다음에 4를 큐에서 빼면서 4와 인접한 5를 방문하려 하는데
- 이미 큐에 있지요?

D. 가희와 베개



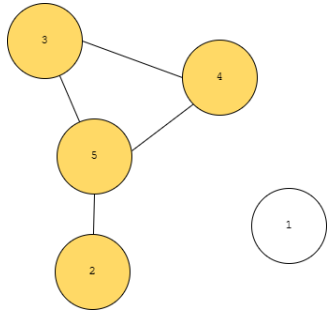
- 다음에 5를 큐에서 빼면서 5와 인접한 2를 방문합니다.
- 그러면 3에서 2로도 방문 가능합니다. 왜?
- 3에서 5로 방문이 가능하고, 5에서 2로도 방문이 가능하기 때문입니다.

D. 가희와 베개



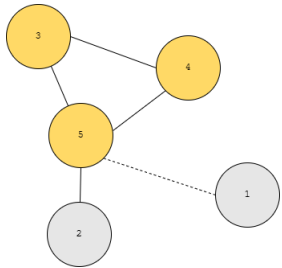
- 더 이상 방문할 곳이 없군요.
- 노란색으로 칠해진 덩어리를 $Component$ 라고 합니다.
- 노란색으로 칠해진 것끼리는 서로 왕래가 가능합니다.

D. 가희와 베개



- 그러면 노란색으로 칠해진 곳에서 하얀색으로 칠해진 곳은 방문 가능할까요?
- 아닙니다. 다른 $Component$ 이기 때문입니다.
- 그림 상으로도 하얀색으로 칠해진 곳과 노란색으로 칠해진 덩어리는 연결되어 있지 않습니다.

D. 가희와 베개



- 노란색으로 칠해진 곳으로부터 하얀색으로 칠해진 곳으로 방문이 가능하다.
- 이 말은 노란색으로 칠해진 어느 한 노드에서 하얀색으로 칠해진 노드로 연결하는 간선이 있다는 것입니다.
- 그렇다면 bfs 과정에서 큐에 넣어졌을 겁니다.

- 간선이 양방향으로만 연결되어 있고
- 노드 $n_1, n_2, ..., n_k$ 가 같은 컴포넌트에 있다면
- 노드 $n_1, n_2, ..., n_k$ 는 서로 방문이 가능합니다.

- 이제 경사로. 경사로는
  - 컴포넌트 2개를 연결하는 역할을 합니다.
  - 고로 컴포넌트를 하나의 노드 N 으로 본다면
  - 컴포넌트 N_a 에서 컴포넌트 N_b 로 갈 수 있게 경사로를 연결할 수 있는지로 문제가 바뀝니다.

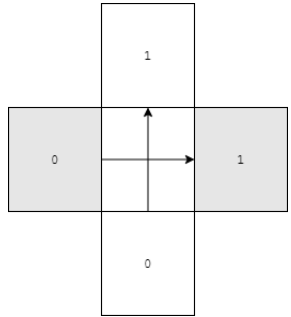
D. 가희와 베개

- 경사로 1개는 많아봐야 컴포넌트 2개를 연결합니다.
- 고로 각 경사로마다 관여하는 컴포넌트 수는 많아야 $2g$ 개임을 알 수 있습니다.
- 그러면 해당 컴포넌트들만 연결해 놓고 검사하면 되겠습니다.

D. 가희와 베개

- 마지막으로 남은 것은 경사로입니다.
- 경사로가 있으나 없으나 별 상관없는 경우가 있습니다. 잘못된 방향으로 놓여진 경우겠지요.
- 잘못된 방향으로 놓여진 경우는 경우의 수에서 제외합니다.

D. 가희와 베개



- 하나의 경사로 위치마다 따져야 할 경우 수는 많아야 2개임을 알 수 있습니다. 왜?
- 경사로와 인접한 방향은 4개인데
- 유효한 방향 하나당 인접한 2개 위치의 고도가 강제되기 때문입니다.

- 정리하면
  - 양방향 간선만 있는 그래프에서 같은 컴포넌트에 속한 위치는 상호간 이동이 가능합니다.
  - bfs나 dfs 등으로 컴포넌트 별로 나눌 수 있습니다.
  - 경사로들은 컴포넌트 둘을 연결하는 역할을 합니다.
  - 검사해야 하는 컴포넌트들은 많지 않습니다.

E. 가희와 btd5 2

`data structure`, `event processing`, `implementation`, `oop`
출제진 의도 – **Challenging**

- 제출 2번, 정답 0명 (정답률 -%)
- 처음 푼 사람: -, -분
- 출제자: chogahui05

E. 가희와 btd5 2

- 하라는 대로만 하면 되긴 합니다만
- 지문이 굉장히 길고 상황도 복잡합니다.
- 천천히 이해해 봅시다.

E. 가희와 btd5 2

- 풍선은 2가지 속성이 있어요.
  - 은신
  - 뾰족한 것에 이뮨이 있습니다.

E. 가희와 btd5 2

- 그런데 원숭이도 2가지 스킬이 있어요.
  - 은신 풍선을 볼 수 있습니다.
  - 납이 씌워진 풍선을 공격할 수 있습니다.

E. 가희와 btd5 2

- 그러면 아래 2가지 상황을 생각해 봅시다.
  - 저격 원숭이가 은신 풍선을 볼 수 없다.
  - 저격 원숭이가 납이 씌워진 풍선에 데미지를 줄 수 없다.
- 이 둘은 뭐가 다를까요?

E. 가희와 btd5 2

- ✓ 전자는 $targetting$ 조차도 불가능합니다. 고로, 데미지 자체를 입지 않았겠지요.
- ✓ 후자는 $targetting$ 은 가능합니다. 이뮨이 있어서 데미지를 못 입혔지만요.
- ✓ 문제. $R1$ 스킬만 배운 원숭이는 은신 풍선을 공격할 수 있을까요?

E. 가희와 btd5 2

- ✓ 이제 구현해야 할 내용을 정리해 봅시다. 먼저 원숭이
  - $targetting$ 합니다.
  - 공격 합니다.
- ✓ 다음 풍선은
  - 맵에서 특정 시간 t에 등장합니다.
  - 시간이 경과해서 사라지거나 원숭이의 공격으로 사라집니다.

E. 가희와 btd5 2

- ✓ 원숭이는 $targetting$ 우선 순위는 3가지 있습니다.
  - $FIRST$
    - 출구로부터 가장 가까운 풍선, 지점에 가장 많이
    - id
  - $LAST$
    - 입구로부터 가장 가까운 풍선, 지점에 가장 적게
    - id
  - $STRONG$
    - 가장 lv 이 높은 풍선
    - id

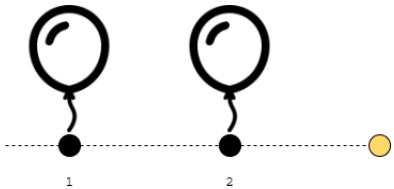
E. 가희와 btd5 2

- 그런데 경로 상에 풍선이 상당히 많이 나올 수 있습니다.
- 고로 경로별로 특정 연산을 효율적으로 할 수 있는 자료구조가 필요합니다.
  - 풍선이 추가되거나 제거됩니다.
  - 원숭이가 **우선 순위**가 제일 높은 풍선을 $targetting$ 합니다.
- c나 c++에서는 map과 set이 이러한 연산을 매우 잘 지원해 줍니다.
- 혹은 $priority_queue$를 쓸 수 있는데 이 방법은 후술하겠습니다.

E. 가희와 btd5 2

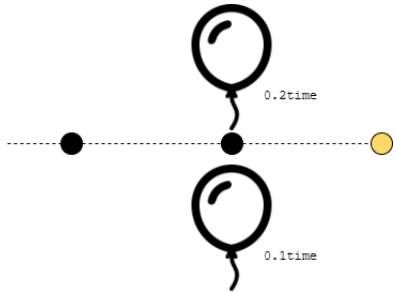
- 2번째 $LAST$
  - 입구로부터 가장 가까운 풍선, 지점에 가장 적은 시간.
  - 시각 1에 등장한 풍선 1과 시각 2에 등장한 풍선 2를 시각 3에 비교해 봅시다.

E. 가희와 btd5 2



- 왼쪽에 있는 것은 시각 1에 등장한 풍선
- 오른쪽에 있는 것은 시각 2에 등장한 풍선입니다.
- 딱 봐도 전자가 후자보다는 입구에서 멀리 있습니다.

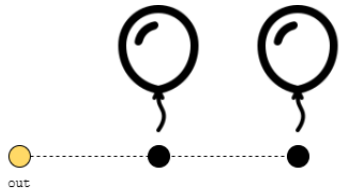
E. 가희와 btd5 2



- 이제 시각 1.8에 등장한 풍선과 시각 1.9에 등장한 풍선 2개를 시각 3에 비교해 봅시다.
- 전자는 위에, 후자는 아래에 있습니다. 머문 시간이 후자가 더 짧습니다.
- 고로 등장 시각이 가장 최근인 것이 $LAST$에 뽑힐 가능성이 높겠네요.

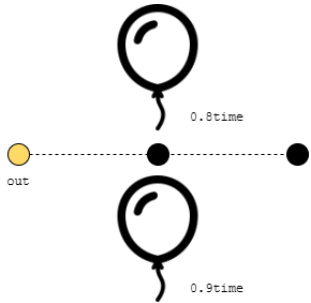
- 1번째 $FIRST$
  - 출구로부터 가장 가까운 풍선, 지점에 가장 많은 시간.
  - 시각 3.1에 사라질 풍선 1과 시각 4.1에 사라질 풍선 2를 시각 2에 비교해 봅시다.

E. 가희와 btd5 2



- 왼쪽에 있는 것은 시각 3.1에 사라질 풍선 1
- 오른쪽에 있는 것은 시각 4.1에 사라질 풍선 2입니다.
- 딱 봐도 전자가 후자보다는 출구에 가까이 있습니다.

E. 가희와 btd5 2



- 이제 시각 4.1에 사라질 풍선과 시각 4.2에 사라질 풍선 2개를 시각 3에 비교해 봅시다.
- 전자는 아래에, 후자는 위에 있습니다. 머문 시간이 전자가 더 깁니다.
- 고로 사라질 시각이 가장 빠른 것이 $FIRST$에 뽑힐 가능성이 높겠네요.

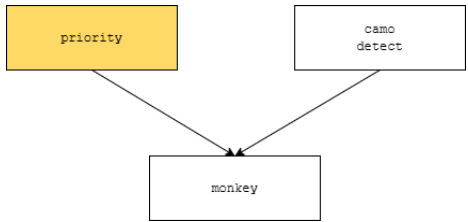
- ✓ 따라서 원숭이의 $targetting$ 우선 순위는 아래와 같이 **정규화**를 할 수 있습니다.
  - $FIRST$
    - ▶ 사라질 시각이 가장 빠른 것
    - ▶ id
  - $LAST$
    - ▶ 등장한 시각이 가장 늦은 것
    - ▶ id
  - $STRONG$
    - ▶ 가장 lv 이 높은 풍선
    - ▶ id

E. 가희와 btd5 2

- ✓ 각 우선순위별로 다른 $queue$ 를 가집니다.
- ✓ 풍선을 $targetting$ 을 할 때 부터 생각해 봅시다.
  - 은신 풍선을 $targetting$ 할 수도 있고 못 할 수도 있습니다.
  - 따라서, 은신인지 아닌지 여부에 따라서 다른 $queue$ 를 돌려야 합니다.
- ✓ 여기까지 정리하면 $queue[priority][camo]$ 로 관리하면 된다는 것이겠군요.

- 이제 원숭이 로직을 정리해 봅시다.
  - 풍선을 $targetting$ 합니다. 설정된 **우선순위** 에 따라
  - 그리고 $camo$ 를 볼 수 있는지 여부에 따라. 그러면
    - 풍선의 타겟 우선 순위에 따라 쓰는 $queue$ 가 달라질 것이고
    - $camo$ 를 볼 수 있는지에 따라 타겟팅 할 수 있는 풍선의 범위가 달라지겠네요.

E. 가희와 btd5 2



- 그림으로 그리면 이런 상황이군요.
- 뭔가 익숙한 용어가 떠오르지 않으시나요? $Dependency\_Injection$

- ✓ 그러면 원숭이가 생성될 때 $priority$ 를 결정하는 것을 $Inject$ 시킵니다.
- ✓ 그리고 원숭이의 우선 순위는 계속 바뀔 수 있네요?
- ✓ 따라서 $set\_priority$ 로도 $Inject$ 시키면 되겠습니다.

- 그리고 $targetting$ 할 수 있는 풍선의 범위도 변합니다.
  - 그러면, 처음에는 $targetting$ 할 수 있는 풍선의 집합이 레벨 1과 2짜리만 있고
  - $L1$ 을 배우면 3과 4짜리도 볼 수 있는 것이잖아요.
- 풍선을 볼 수 있는 집합도 $Inject$ 를 시켜도 되고
- 그냥 $flag$ 만 둬서 처리해도 무난합니다.

E. 가희와 btd5 2

- 한 가지 함정은 $Lead$가 붙은 풍선들입니다.
- 이 풍선은 공격받으면
  - 사라지지 않고 $Lead$만 없어집니다.
  - 따라서, $Lead$는 제거처리 하고, 레벨이 2 낮아진 풍선을 추가해야 합니다.
- 그러면 일정 시각이 지나면 사라지는 풍선은 어떻게 관리할까요?

- 각각의 풍선별로 아래와 같은 정보를 담고 있습니다.
  - 등장하는 시각
  - 사라지는 시각
  - 풍선의 id
- 이 3개의 정보를 가지고 각각의 $queue$에서 찾아서 제거해 주면 됩니다.

E. 가희와 btd5 2

- 원숭이에게 공격받지 않는다면
  - 일정 시각이 지나면 사라집니다.
  - 그러니 $event$ 배열에 풍선이 등장한다와 사라진다를 추가하면 됩니다.
- 이 이벤트 배열에 원숭이에 대한 정보도 추가하면 깔끔합니다.

E. 가희와 btd5 2

- 원숭이에 대한 이벤트는 풍선보다는 쉽습니다. 왜냐?
  - id의 최대값이 10^3 입니다.
  - 고로 원숭이는 그냥 배열로 관리해 주면 됩니다.
  - 그런데 있고 없고는 어떻게 관리하죠?
    - 없으면 nil
    - 있으면 원숭이 객체를 가리키게

E. 가희와 btd5 2

- 이벤트 배열로 관리한다면 시각 t에
  - 공격하는 원숭이가 있으면 공격
  - 해당 시각 이후로 풍선이 사라지면 풍선이 맵에서 사라진다.
- 이를 구현하면 됩니다. 남은 건 입력 $parsing$ 이네요.

E. 가희와 btd5 2

- 입력을 처리하는 좋은 방법
  - 의미 없는 데이터가 들어오면 무시합니다.
  - 초기화 단계에서 수행할 수 있는 것이 있는지 봅니다.
  - 쿼리가 들어오면, 들어온 즉시 처리할 수 있는지 봅니다.
  - 토큰이 여러개가 나올 수 있는 경우
    - 어떤 정보의 토큰 개수가 고정이고
    - 어떤 정보의 토큰 개수가 고정이 아닌지 파악합니다.

E. 가희와 btd5 2

- 먼저 의미 없는 데이터가 있나요? 네.
- 경로에 대한 정보가 좌표로 주어지는 부분입니다. 왜 이게 의미 없을까요?
  - 경로의 길이가 r 이라면 정확히 r 개의 좌표가 주어지기 때문입니다.
  - 지점 또한 t 번째로 나온 좌표와 관련이 없기 때문입니다.
- 따라서 **경로의 길이**만 가지고 오면 됩니다.

E. 가희와 btd5 2

- ✓ 초기화 단계에서 수행할 수 있는 것이 있나요?
- ✓ 이벤트의 목록들을 제외하고 전부 초기화 단계에서 수행할 수 있습니다.
  - 경로에 대한 정보를 초기화 합니다.
  - 맵에 있는 원숭이에 대한 정보들을 초기화 합니다.

E. 가희와 btd5 2

- ✓ 쿼리가 주어지면 즉시 처리할 수 있는 쉬운 방법이 있나요?
- ✓ 이게 중요한 이유는
  - 전처리가 필요한 경우 **전처리 한 데이터**를 저장할 변수가 별도로 필요하기 때문입니다.
  - 그렇지 않으면 쿼리를 받을 때 마다 처리하면 되기 때문입니다.
  - 전자의 대표적인 예시는 $sqrt\_decomposition$ 이 있습니다.

E. 가희와 btd5 2

- 이벤트의 종류는 크게 보면 2가지입니다.
  - 원숭이에 대한 이벤트
  - 풍선에 대한 이벤트
- 원숭이는 쿼리(이벤트)가 들어온 즉시 처리가 가능합니다. 풍선은 어떨까요?

E. 가희와 btd5 2

- 풍선에 대한 이벤트는 2가지로 나눌 수 있습니다.
  - 풍선이 사라진다.
  - 원숭이가 풍선을 공격한다.
- 후자는 원숭이의 타겟 우선순위와 $camo$ 여부로 관리한 set 이나 map 으로 처리할 수 있습니다.
- 전자는 풍선에 대한 등장 시각, 사라진 시각, id 만 있으면 빠르게 처리 가능합니다.

- 토큰이 여러개가 나올 수 있는 경우
  - 어떤 정보의 토큰이 고정이지 않은가요?
  - $balloon\_name$ 만 토큰 개수가 고정이지 않습니다.
    - $Red, Balloon$
    - $Camo, Red, Balloon$
    - $Lead, Balloon$
    - $Camo, Lead, Balloon$

- ✓ $balloon\_name$ 이 나오는 이벤트는 아래와 같습니다.
- ✓ $S.T\ BALLOON\ balloon\_name\ balloon\_id\ route\_id$
  - 2번째 토큰이 $BALLOON$ 인 경우
    - ▶ 3번째 토큰부터 모든 토큰을 다 받습니다.
    - ▶ 마지막 1번째 것과 2번째 토큰을 받고, 해당 토큰들을 제거힙니다.
    - ▶ 그러면 남은 것은 $balloon\_name$ 이네요. 처리하면 되겠습니다.

E. 가희와 btd5 2

- 제 풀이는 set, map과 같이 rb_tree를 이용한 것입니다.
- 하지만 $priority_queue$를 이용하는 방법도 있습니다.
- 핵심은 같습니다만, 이미 제거된 풍선을 어떻게 관리해야 할 지가 문제입니다.

- ✓ 풍선이 제거되었는지 여부를 판정하는 rm_flag 등을 추가합니다.
- ✓ 이제 이 메타 정보를 바탕으로 $targetting$ 할 때,
  - 이미 제거된 경우
    - $queue$ 에서 제거합니다.
    - 다음 원소를 탐색합니다.
  - 제거되지 않은 경우
    - 해당 풍선을 타겟팅 합니다.
    - $queue$ 에서 제거합니다.
    - 해당 풍선의 rm_flag 를 $true$ 로 바꿔줍니다.

- 이것을 $mark$, 그리고 $sweep$ 관점에서 보면
  - rm_flag 는 $mark$ 역할을 하게 됩니다. 제거되었는지 그렇지 않은지.
  - 실제로 $sweep$ 할 때에는 이 $flag$ 에 따라
    - 경로 k 를 탐색할 때 마다
    - rm_flag 가 켜져 있으면 제거하게 됩니다.
- 끝나는 시간도 고려해야 하니, 이 부분은 잘 생각해 보시면 됩니다.