[22.04.2] 성적 조회

TS점수조회

김 태 현

문 제

정보 등록 : {학생 ID(1 ~ 1,000,000,000), 학년(1,2,3), 성별(male, female), 점수(0~300,000)}

정보 삭제 : 학생 ID

정보 검색 : 주어진 학년, 성별 만족하며, 특정 점수 이상의 학생 중 점수가 가장 낮은 학생의 ID
주어지는 학년, 성별은 여러 개일 수 있음

addScore() 호출 횟수 <= 20,000

모든 함수 호출 횟수 <= 80,000

```
void init()
```

모든 정보 초기화

```
int addScore(int id, int grade, char gender[], int score)
```

{id, grade, gender, score} 등록
학년, 성별이 각각 grade, gender 인 학생들 중 1)점수가 가장 높은 학생 ID, 2) 같다면 ID가 가장 큰 학생 ID 반환

```
int removeScore(int id)
```

학생ID가 id인 정보 제거
제거 후, 해당 학생의 학년과 성별이 동일한 학생 중, 1)점수가 가장 낮은 학생 ID, 2) 같다면 ID가 가장 작은 학생 ID반환
id인 학생이 없거나 , id인 학생과 학년, 성별이 같은 학생이 없다면 0 반환

```
int get(int gradeCnt, int grade[], int genderCnt, char gender[][7], int score)
```

grade[], gender[] 인 학생들 중, 점수가 score 이상이면서 1) 점수가 가장 낮은 학생 ID, 2) 같다면 ID가 가장 작은 학생 ID 반환

주요 로직

1. 학생 ID(큰 정수)로 정보 접근 -> hash
2. 학생 ID가 id인 정보 접근/삭제
3. 특정 학년,성별의 가장 높은점수(큰ID) 학생 ID
   낮은점수(낮은ID) 학생 ID
4. {학년, 성별} 에 포함되는 학생들 중, 특정 점수 이상이면서 가장 낮은점수(낮은 ID)의 학생 ID

문제 분석

- addScore() <= 20,000 회 : 최대 data 수 = 20,000개
- 모든 함수 호출 <= 80,000 회
- 모든 함수에서 특정 조건의 학생 검색 필요

  매번 linear search 시, O(data 수 * 호출 횟수) = 시간 초과

학생 ID가 id인 정보 접근/삭제

어떤 자료구조를 쓰냐에 따라 달라짐

- **list** : iterator 활용하여 O(1)
- **vector** : O(n)
- **set** : iterator 활용시 O(1) , 활용안할시 O(log n)
- **unordered_set** : O(1)

특정 학년,성별의 가장 높은점수(큰ID) / 낮은점수(낮은ID) 학생 ID

linear search 시, 오래 걸리므로 자료 저장을 유리하게 할 필요가 있음

1. grade, gender 별로 분류하여 관리, 총 6개
2. 우선순위 순으로 관리
    - set/map : {score, id} 오름차순 한 개
    - priority_queue : {score, id} 큰 기준 / 작은 기준 두 개

{학년, 성별} 에 포함되는 학생들 중, 특정 점수 이상이면서 가장 낮은점수(낮은ID)의 학생 ID

- 분류된 개수가 6개 뿐이므로 포함되는 구간을 전부 확인
- set/map의 lower_bound() 를 통해 선택되는 노드 중 { score, id } min 값 반환

자료 구성

unordered_map: *student*

key:id	value:{grade, gender, iterator}
500	1, 1, it_500
400	3, 0, it_400
300	2, 0, it_300

grade : 1,2,3
gender : 1(male), 0(female)

set: *priority[grade][gender]*

addScore()

priority[grade][gender]에 {score, id} 등록
student[id] = {grade, gender, 등록한 iterator}
priority[grade][gender] 마지막 노드 id 반환

removeScore()

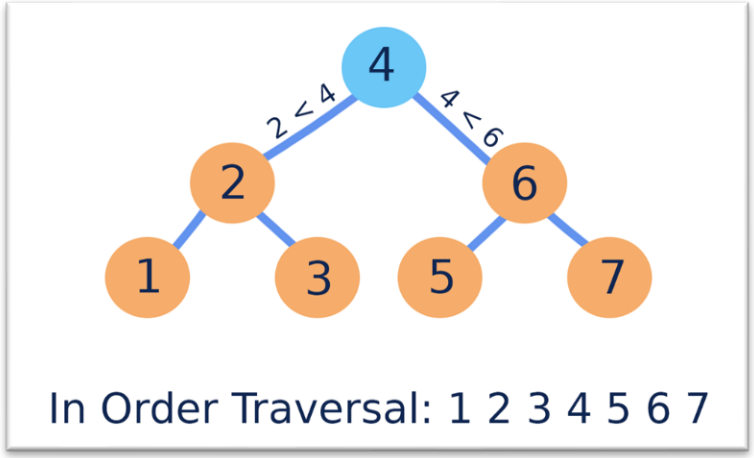
student[id]에 기록된 iterator로 priority에서 삭제
student[id] 삭제
priority[grade][gender] 첫번째 노드 id 반환

get()

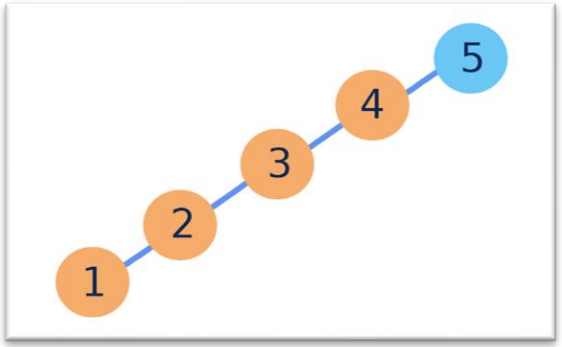
모든 grade, gender 쌍의 priority에서 lower_bound로 찾아진 노드들 중 가장 작은 값 반환

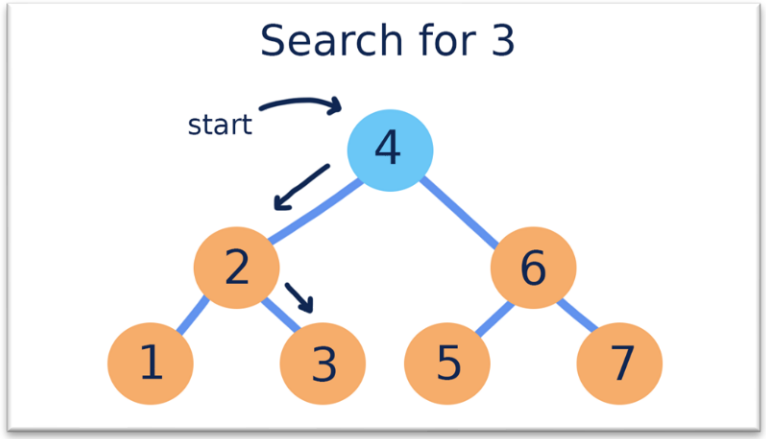
Binary Search Tree

non-stl 일 경우, set 대신



- binary tree
- left child : 작은 값
- right child : 큰 값
- search time complexity : worst case O(n)





이를 개선한 구조 : self-balancing binary search tree
ex) Red-Black tree, AVL tree, Splay tree , …

bucket 분할하는 경우?

score	info
0 ~ 1000	{id, grade, gender, score}, ..
1001 ~ 2000	
..	
290,001 ~ 300,000	

- 평균적으로는 data가 고르게 분포되어 검색량을 줄일 수 있지만 고르게 분포된다라는 보장이 없다.
- worst case는 한 개의 구간에 모든 data가 존재하는 경우이며, 이 경우는 linear search 보다도 비효율적이다.
- Pro검정의 경우, 일반적으로 random case이긴 하나, 출제자의 의도에 따라서 얼마든지 저격 data를 넣을 수도 있다.
- 따라서, 이 해법은 input case에 의존해야하는 해법이며 합격을 보장받지는 못한다.

addScore()

마지막 구간부터 감소하며 노드 있는 구간 선택
해당 구간에서 가장 큰 값 검색

removeScore()

처음 구간부터 증가하며 노드 있는 구간 선택
해당 구간에서 가장 작은 값 검색

get()

입력된 score의 구간부터 증가하며 score 이상의 가장 작은 값 선택


감사합니다