JooHyun – Lee (comkiwer)

TS끝말잇기

Hancom Education Co. Ltd.

Problem

Problem

TS끝말잇기

N 명의 플레이어가 M 개의 단어를 사용하여 끝말잇기 게임을 한다.

(3 ≤ N ≤ 50,000, N ≤ M ≤ 50,000)

끝말잇기 게임을 시작하기 전에 먼저 플레이어의 수와 사용할 수 있는 단어 목록이 주어진다.

플레이어 ID는 1부터 N까지의 수이고 같은 ID를 가지는 플레이어는 없다.

단어 목록에서 각 단어는 길이가 2이상 10이하의 영어 알파벳 소문자로 구성되어 있고,
‘＼0’ 문자로 끝나는 문자열이다.

또한, 단어 목록 내의 각 단어는 중복되지 않는다.

Problem

끝말잇기 게임은 여러 개의 라운드로 구성되어 있다.

각 라운드에서는 플레이어 ID를 기준으로 1, 2, 3, .., N 순으로 턴을 진행한다.

라운드를 시작하기 전에 첫번째 턴을 진행할 플레이어 ID와 첫 단어를 정하기 위한 하나의 문자가 주어진다.

플레이어는 자신의 턴이 되면 단어 목록에 있는 단어 중 하나를 조건에 맞게 선택한다.

선택된 단어는 다시 선택하지 않도록 단어 목록에서 삭제한다.

Problem

아래는 선택할 수 있는 단어의 조건이다.

1. 이전 턴에 선택된 단어의 마지막 문자로 시작하는 단어.
   첫번째 턴의 경우 라운드를 시작하기 전 주어진 문자로 시작하는 단어.
2. 게임을 진행하는 동안 한번도 선택된 적이 없는 단어.
3. 1과 2에 해당하는 단어가 여러 개인 경우 사전 순으로 가장 빠른 단어.

Problem

위 조건에 맞는 단어가 더 이상 없는 경우
해당 턴의 플레이어는 게임에서 탈락하고 라운드가 종료된다.

탈락한 플레이어는 다음 라운드부터 턴 진행에서 제외된다.

다음 라운드를 시작하기 전에 이전 라운드에서
선택된 단어를 뒤집은 후 단어 목록에 추가한다.

뒤집은 단어가 이미 선택된 단어라면 단어 목록에 추가하지 않는다.

아래 API 설명을 참조하여 각 함수를 구현하라.

Problem

1. `void init(int N, int M, char words[50000][WLEN])` :

각 테스트 케이스의 맨 처음에 호출된다.

`N` 은 플레이어 수이다. `M` 은 주어지는 단어 목록의 단어 수이다.

`words` 는 끝말잇기 게임에서 사용 가능한 단어 목록이 포함된 배열이다.

단어 목록은 `words` 배열의 인덱스 `0` 부터 순서대로 주어진다.

각 단어는 길이가 `2`이상 `10`이하의 영어 알파벳 소문자로 구성되어 있고, '\0' 문자로 끝나는 문자열이다.

`words` 에 있는 `M` 개의 단어들은 서로 중복되지 않는다.

Parameters

`N` : 플레이어 수 (3 ≤ N ≤ 50,000)

`M` : 단어 목록의 단어 수 (N ≤ M ≤ 50,000)

Problem

2. `int playRound(int pid, char ch)` :

`pid`는 해당 라운드에서 첫번째 턴을 진행할 플레이어 `ID` 이다.

`pid`는 아직 탈락하지 않은 플레이어임을 보장한다.

`ch`는 알파벳 소문자이며 첫번째 플레이어는
단어 목록에서 `ch` 문자로 시작하는 단어를 선택해야 한다.

선택 가능한 단어들 중 `ch` 문자로 시작하는 단어가 존재함을 보장한다.

해당 라운드에서 탈락한 플레이어 `ID`를 반환한다.

해당 함수의 호출 횟수는 `N` 을 넘지 않는다.

Parameters

`pid` : 첫번째 턴을 진행할 플레이어 `ID` ($1 \le pid \le N$)

`ch` : 첫번째 플레이어가 선택할 단어의 첫 문자

Return

해당 라운드가 종료된 후 탈락한 플레이어 `ID`를 반환

Problem

[제약사항]

1. 각 테스트 케이스 시작 시 `init()` 함수가 호출된다.
2. 각 테스트 케이스에서 `playRound()` 함수의 호출 횟수는 `N`을 넘지 않는다.
3. `playRound()` 함수에서 전달하는 `ch` 문자는
   선택 가능한 단어들 중 하나의 첫번째 문자이다.

Problem analysis

Problem analysis : 예제

아래의 [표 1]과 같이 요청이 되는 경우를 살펴보자.

4명의 플레이어가 5개의 단어로 끝말잇기 게임을 시작한다.

순서	함수	반환
1	init(4, 5, [eye, ezone, event, tee, east])	
2	playRound(3, e)	2
3	playRound(3, t)	1
4	playRound(3, e)	4

[표 1]

Problem analysis : 예제

[표 2] 는 init 함수에서 전달한 단어 목록의 단어들을 시작 문자 기준으로 나열한 표이다.

문자	단어들	단어수
e	east, event, eye, ezone	4
t	tee	1

[표 2]

첫번째 라운드에서 첫번째 턴을 진행할 플레이어는 3이며 e 문자로 시작하는 단어를 고른다.
e 문자로 시작하는 단어는 ‘east’, ‘event’, ‘eye’, ‘ezone’ 4개가 존재하며
이 중 사전 순으로 빠른 단어는 ‘east’ 이므로 플레이어 3은 ‘east’ 을 선택한다.
다음 턴에서 t 문자로 시작하는 단어는 ‘tee’ 1개로 플레이어 4는 ‘tee’ 를 선택한다.
다음 턴에서 e 문자로 시작하는 단어는 ‘event’, ‘eye’, ‘ezone’ 3개가 존재하며
이 중 사전 순으로 빠른 단어는 ‘event’ 이므로 플레이어 1은 ‘event’ 를 선택한다.
다음 턴에서 t 문자로 시작하는 단어가 없으므로 해당 턴의 플레이어 2가 탈락한다.
다음 라운드 시작 전 이미 사용한 단어인 ‘east’, ‘tee’, ‘event’ 을 뒤집어
‘tsae’, ‘eet’, ‘tneve’ 를 단어 목록에 추가한다.
[그림. 1] 은 첫번째 라운드에서 각 플레이어가 선택한 단어를 턴 순서대로 나열한 그림이다.

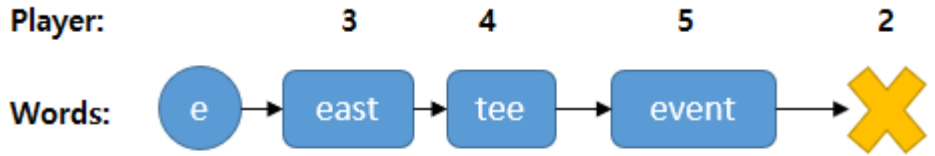

그림. 1

Problem analysis : 예제

[표 3] 은 두번째 라운드 시작 전 단어 목록의 단어들을 시작 문자 기준으로 나열한 표이다.

문자	단어들	단어수
e	eet, eye, ezone	3
t	tneve, tsae	2

[표 3]

두번째 라운드에서 첫번째 턴을 진행할 플레이어는 3이며
t 문자로 시작하는 단어를 고른다.
t 문자로 시작하는 단어는 ‘tneve’, ‘tsae’ 2개가 존재하며
이 중 사전 순으로 빠른 단어는 ‘tneve’ 이므로 플레이어 3는 ‘tneve’ 를 선택한다.
다음 턴에서 e 문자로 시작하는 단어는 ‘eet’, ‘eye’, ‘ezone’ 3개가 존재하며
이 중 사전 순으로 빠른 단어는 ‘eet’ 이므로 플레이어 4는 ‘eet’ 를 선택한다.
다음 턴에서 t 문자로 시작하는 단어는 ‘tsae’ 1개로 플레이어 1은 ‘tsae’ 를 선택한다.
플레이어 2는 이전 라운드에서 탈락했기 때문에 플레이어 3이 다음 턴을 진행한다.
e 문자로 시작하는 단어는 ‘eye’, ‘ezone’ 2개가 존재하며
이 중 사전 순으로 빠른 단어는 ‘eye’ 이므로 플레이어 3는 ‘eye’ 을 선택한다.
다음 턴에서 e 문자로 시작하는 단어는 ‘ezone’ 1개로 플레이어 4는 ‘ezone’ 을 선택한다.
다음 턴에서 e 문자로 시작하는 단어가 없으므로 해당 턴의 플레이어 1이 탈락한다.
다음 라운드 시작 전 이미 사용한 단어인 ‘ezone’ 을 뒤집어 ‘enoze’ 를 단어 목록에 추가한다.
‘tneve’, ‘eet’, ‘tsae’, ‘eye’ 의 경우 뒤집었을 때 이미 선택된 단어이므로 단어 목록에 추가하지 않는다.
[그림. 2] 은 두번째 라운드에서 각 플레이어가 선택한 단어를 턴 순서대로 나열한 그림이다.

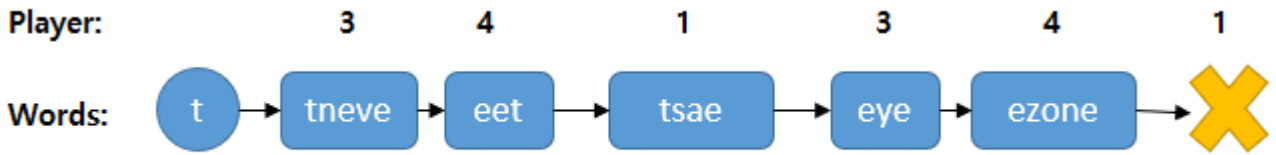

그림. 2

Problem analysis : 예제

[표 4] 은 세번째 라운드 시작 전 단어 목록의 단어들을 시작 문자 기준으로 나열한 표이다.

문자	단어들	단어수
e	enoze	1

[표 4]

세번째 라운드에서 첫번째 턴을 진행할 플레이어는 3 이며 e 문자로 시작하는 단어를 고른다.
e 문자로 시작하는 단어는 'enoze' 1개이므로 플레이어 3은 'enoze' 를 선택한다.
다음 턴에서 e 문자로 시작하는 단어들 중 선택할 수 있는 단어가 없으므로 플레이어 4가 탈락한다.
남아있는 플레이어가 1명이므로 더 이상 라운드를 진행하지 않고 끝말잇기 게임은 종료된다.

[그림. 3] 은 세번째 라운드에서 각 플레이어가 선택한 단어를 턴 순서대로 나열한 그림이다.

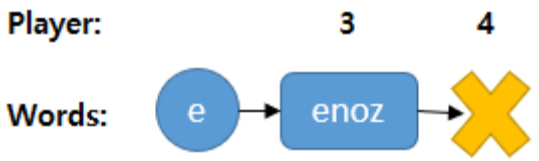

그림. 3

Problem analysis : 제약조건 및 API

- 단어의 수 : 5만(init()으로 주어짐) + 5만(역순생성)
- 플레이어 수 : 5만(아이디는 1 ~ 50,000 범위)
- playRound() 함수 호출 수 : 5만
  : 함수호출수가 5만 플레이어 수도 5만이지만 O(5만 * 5만) 은 될 수 없다.
  : 한번 사용된 단어는 다시 사용될 수 없으므로 최대 10만의 단어만 사용되기 때문이다.

- 단어는 길이 최대 10의 알파벳 소문자이다.
- 끝말잇기 문제이므로 주어진 문자로 시작하는 알파벳상 가장 빠른 단어를
  빠르게 찾을 수 있어야 한다.

- 플레이어를 빠르게 찾고 빠르게 삭제할 수 있어야 한다.

Problem analysis : 제약조건 및 API

- playRound(pid, ch) 호출에 대하여
  - ✓ pid를 빠르게 찾을 수 있어야 한다.
  - ✓ 탈락한 플레이어를 빠르게 제거할 수 있어야 한다.

  플레이어를 어떤 자료구조로 다룰 수 있을까?

  - ✓ ch로 시작하는 단어를 빠르게 찾을 수 있어야 한다.
  - ✓ 끝말로 이을 수 있는 다음 단어를 빠르게 찾을 수 있어야 한다.
  - ✓ 뒤집은 단어가 이미 등장한 적이 있었는지 빠르게 확인할 수 있어야 한다.

  효율적인 단어 검색, 추가, 제거를 위한 자료구조는 어떤 것이 있을까?

Solution sketch

Solution sketch

- 문제에서 등장하는 데이터 타입은
  단어(문자열)와 플레이어 두 가지이다.

- 먼저 문자열을 어떻게 다룰지 생각해 보자.
  - ✓ 시작문자로 분류되어 있고 알파벳 순으로 정렬되어 있다면
    끝말잇기의 다음단어를 찾기 쉬울 것이다.
  - ✓ init()에서 주어지는 단어가 전부라면 deque<string> group[26]에 각 그룹별로 저장하고
    각 그룹별로 정렬한 뒤 사용하면 된다. 그러나 playRound()에서 새로운 단어가
    추가될 수 있다. 사용된 단어를 뒤집어 사용할 수 있기 때문이다.
    그렇다면 다른 대안으로 생각할 수 있는 것은?

Solution sketch

- 두 가지 종류의 데이터 타입이 주어진다.
  문자열과 플레이어 이다.

- 먼저 문자열을 어떻게 다룰지 생각해 보자.
  - ✓ 시작문자로 분류되어 있고 알파벳 순으로 정렬되어 있다면
    끝말잇기의 다음단어를 찾기 쉬울 것이다.
  - ✓ init()에서 주어지는 단어가 전부라면 deque<string> group[26]에 각 그룹별로 저장하고
    각 그룹별로 정렬한 뒤 사용하면 된다. 그러나 playRound()에서 새로운 단어가
    추가될 수 있다. 사용된 단어를 뒤집어 사용할 수 있기 때문이다.
    그렇다면 다른 대안으로 생각할 수 있는 것은?
    priority_queue<string, vector<string>, greater<string>> pq[26]; 이 있을 것이다.
  - ✓ 또한 한 번 등장한 단어는 (사용되었든 사용대기 중이던) 중복하여 등장할 수 없다.
    단어가 등장하였는지 아닌지 확인하는 것으로 족하므로
    unordered<string> entry; 로 관리할 수 있다.

Solution sketch

- 이제 플레이어를 어떻게 다룰지 생각해 보자.

  playRound(int pid, char ch) 에서 함수호출이 50,000번 일어날 수 있으므로
  : pid를 빠르게 찾을 수 있어야 한다.
  : 또한 탈락한 유저는 빠르게 삭제될 수 있어야 한다.

  ✓ 배열을 사용하는 경우: 임의 접근으로 빠르게 찾을 수 있으나 삭제에 O(N)이 걸린다.

  ✓ 리스트를 사용하는 경우: 삭제는 빠르나 찾는데 O(N)이 걸린다.

  ✓ 어떻게 할 수 있을까?

Solution sketch

- 이제 플레이어를 어떻게 다룰지 생각해 보자.

  playRound(int pid, char ch) 에서 함수호출이 50000번 일어날 수 있으므로
  : pid를 빠르게 찾을 수 있어야 한다.
  : 또한 탈락한 유저는 빠르게 삭제될 수 있어야 한다.

  ✓ 배열을 사용하는 경우: 임의 접근을 빠르게 찾을 수 있으나 삭제에 O(N)이 걸린다.

  ✓ 리스트를 사용하는 경우: 삭제는 빠르나 찾는데 O(N)이 걸린다.

  ✓ 어떻게 할 수 있을까?

  ✓ 플레이어를 리스트에 저장하고 iterator 배열을 만들어 임의 접근할 수 있도록 할 수 있다.

  ```
  list<int> player;   // 삭제 O(1), 다음 플레이어 이동 O(1)
  list<int>::iterator its[50005];  // pid로 탐색 O(1)
  ```

Solution sketch

- **이제 문제 해결을 위하여 필요한 자료를 선언해 보자.**

```
// 첫 문자별 단어의 오름차순
priority_queue<string, vector<string>, greater<string>> pq[26];

// 등장한 단어(사용된(선택된) + 사용예정 단어)
unordered_set<string> entry;

// 플레이어를 DLL로 관리, 삭제:O(1), 이동:O(1)
list<int> player;

// player는 1번 id부터 시작한다.(배열크기 여유있게) id로 탐색: O(1)
list<int>::iterator its[50005];

// 뒤집어 재사용할 단어 : 다음 round를 위하여 임시 저장
vector<string> reuse;

int N, M; // N:플레이어수, M: 초기 문자열 수
```

Solution sketch

- 이제 각 함수 별 할 일을 정리해보자.

```
void init(int N, int M, char words[50000][WLEN]) :
```

  - 준비한 자료(pq[26], entry, player 등)를 초기화 한다.

  - 플레이어를 리스트 player에 추가하면서 id별 iterator를
    its[id]에 등록한다.

  - 각 단어를 시작문자를 그룹으로 하여
    : pq[시작문자 –'a']에 추가하고
    : entry 집합에도 추가한다.

Solution sketch


```
int playRound(int pid, char ch)
```

- 리스트에서 pid의 iterator를 O(1)에 얻는다. it = its[pid]
- k = ch – 'a' 로 시작하여 pq[k]에 단어가 있는 동안 다음을 수행한다.
  - top() 우선순위 단어를 선택하고 pq[k]에서 제거
  - 선택된 단어의 끝문자로 다음 k 구하기
  - 다음 플레이어 구하기 : ++it
  - 선택된 단어 뒤집고 entry에 없다면
    entry, reuse에 추가
- reuse에 단어가 있다면 pq[]와 entry에 추가
- 탈락한 유저 pid에 저장하고 player목록에서 삭제
- pid를 반환한다.

Solution sketch

[Summary]

- 문제의 요구사항을 분석하고 적절할 자료구조를 선택 정의할 수 있다.

- list<int>와 list<int>::iterator 배열을 이용하여
  O(1)에 삭제, O(1)에 탐색할 수 있다.

- 어떤 자료가 있다 없다를 확인하는 용도로 unordered_set<data_type>
  또는 hash, trie 등을 사용할 수 있다.

- 우선순위 큐를 사용할 수 있다.

Code example1

: STL

string, list, vector,

unordered_set

Code example

TS끝말잇기

```
#define LM 50000
#include <string>
#include <list>
#include <queue>
#include <algorithm>
#include <unordered_set>
using namespace std;

priority_queue<string, vector<string>, greater<string>> pq[26];  // 첫 문자별 단어의 오름차순
unordered_set<string> entry;     // 사용된(선택된) + 사용예정 단어
vector<string> reuse;            // 뒤집어 재사용할 단어
list<int> player;
list<int>::iterator its[LM + 5]; // player는 1번 id부터 시작한다.(배열 조금 여유있게)
int N, M;
void init(int N, int M, char words[50000][11]) {
    ::N = N, ::M = M;
    for (int i = 0; i < 26; ++i) pq[i] = {};
    entry.clear(), reuse.clear(), player.clear();
    for (int i = 1; i <= N; ++i)
        its[i] = player.insert(player.end(), i);
    for (int i = 0; i < M; ++i) {
        int k = words[i][0] - 'a';
        pq[k].push(words[i]);
        entry.insert(words[i]);
    }
}
```

Code example

TS끝말잇기

```
int playRound(int pid, char ch) {
    auto it = its[pid];                 // 첫 플레이어의 iterator
    int k = ch - 'a';
    reuse.clear();
    while (!pq[k].empty()) {
        string s = pq[k].top();         // 현재 플레이어가 선택한 단어
        pq[k].pop();                    // 단어 목록에서 제거
        k = s.back() - 'a';
        ++it;
        if (it == player.end())         // 끝까지 진행한 경우 다시 첫 플레이어로...
            it = player.begin();
        entry.insert(s);                // 사용된+사용예정 목록에 추가(뒤집힌 단어가 새로운 단어일 수 있
다.)
        reverse(s.begin(), s.end());
        if (entry.count(s) == 0)        // 뒤집은 단어가 사용된+사용예정 단어목록에 없다면
            reuse.push_back(s);         // 다음 라운드에서 사용될 목록에 추가
    }
    for (auto&s : reuse) {              // 뒤집은 단어 시작문자별 pq에 추가
        k = s.front() - 'a';
        pq[k].push(s);
    }
    pid = *it;
    player.erase(it);                   // 탈락된 플레이어 삭제
    return pid;
}
```

Code example

: nonSTL

PQ, Trie, bitwise

circular DLL

Code example


```
const int LM = 50002;
using LL = long long;
int N, M;

struct PQ {      // 55bit정수로 관리
    LL heap[LM];// 시작문자를 그룹으로 단어를 알파벳 오름차순으로 관리
    int hn;
    void clear() { hn = 0; }
    bool empty() { return hn == 0; }
    LL top() { return heap[1]; }
    void push(LL nd) {
        int c = ++hn;
        for (; c > 1 && nd < heap[c >> 1]; c >>= 1)
            heap[c] = heap[c >> 1];
        heap[c] = nd;
    }
    void pop(int c = 2) {
        LL nd = heap[hn--];
        for (; c <= hn && heap[c += (c < hn && heap[c + 1] < heap[c])] < nd; c <<= 1)
            heap[c >> 1] = heap[c];
        heap[c >> 1] = nd;
    }
}pq[27];
```

Code example


```
struct Trie {                       // 실제 단어와 같은 길이로 관리
    int buf[LM * 10][27], bcnt;
    bool word[LM * 10];
    void clear() {
        for (int i = 0; i <= bcnt; ++i) {
            word[i] = 0;
            for (int j = 1; j < 27; ++j) buf[i][j] = 0;
        }
        bcnt = 0;
    }
    void insert(LL n, int p = 0) {
        for (; n; n >>= 5){
            int c = n & 31;
            if (buf[p][c] == 0)    buf[p][c] = ++ bcnt;
            p = buf[p][c];
        }
        word[p] = 1;
    }
```

Code example


```
    void erase(LL n, int p = 0) {
        for (; n; n >>= 5) p = buf[p][n & 31];
        word[p] = 0;
    }
    int search(LL n, int p = 0) {
        for (; n; n >>= 5) {
            p = buf[p][n & 31];
            if (p == 0) return 0;
        }
        return word[p];
    }
}entry;        // 등장한 단어 등록하기
```

Code example


```
LL reuse[LM], rn;
LL reverse(LL tg, LL rv = 0) {        // 뒤집기
    for (; tg; tg >>= 5)
        rv = (rv << 5) + (tg & 31);
    return rv;
}

struct Node { // 플레이어 삭제를 빠르게 하기위하여 circular DLL사용
    int data;
    Node*prev, *next;
    void alloc(int nd, Node*np, Node*nn) {
        data = nd;
        prev = np, next = nn;
        prev->next = next->prev = this;
    }
    void erase() {
        prev->next = next;
        next->prev = prev;
    }
}buf[LM];
```

Code example

TS끝말잇기

```
void init(int N, int M, char words[50000][11]) {
    for (int i = 1; i < 27; ++i) pq[i].clear();
    entry.clear();
    ::N = N, ::M = M;
    buf[1] = { 1, buf + 1, buf + 1 };
    for (int i = 2; i <= N; ++i)        // build circular DLL
        buf[i].alloc(i, buf + i - 1, buf[i - 1].next);

    for (int i = 0; i < M; ++i) {
        LL d = 0, sd;
        int j = 0;
        for (; words[i][j]; ++j)
            d = (d << 5) | (words[i][j] - 96);
        int t = 11 - j;                 // 단어의 마지막 문자 위치 / 5
        int k = words[i][0] - 96;
        sd = (d << (t * 5)) + t;        // 55bit로 맞추기
        pq[k].push(sd);                 // 55bit로 추가
        entry.insert(d);                // 원래 단어로 추가
    }
}
```

Code example

TS끝말잇기

```
int playRound(int pid, char ch) {
    Node*it = buf+pid;                  // memory pool을 이용한 DDL을 구현함으로
    int k = ch - 96;                    // 임의접근과 빠른 삭제가 가능.
    rn = 0;  // reuse cnt;
    while (!pq[k].empty()) {
        LL sd = pq[k].top(); pq[k].pop();
        int t = sd & 31;                    // 마지막 문자 위치
        LL rd = reverse(sd >> (t * 5));  // 순수단어 구하여 뒤집기
        it = it->next;                      // 다음 플레이어로
        k = (sd >> (t * 5)) & 31;           // 마지막 문자로 그룹 번호 구하기
        if (entry.search(rd) == 0) {        // 처음 등장한 단어라면
            entry.insert(rd);               // 원래 단어 길이로 추가
            sd = (rd << (t * 5)) + t;       // 55bit로 추가
            reuse[rn++] = sd;               // 다음 라운드를 위하여 임시 보관
        }
    }
    for (int i = 0; i < rn; ++i) {
        pq[reuse[i]>>50].push(reuse[i]); // 다음 라운드를 위하여 그룹별로 추가
    }
    it->erase();
    return it->data;
}
```

Thank you.