Algorithm

고경우 이윤서 백지웅

목차 a table of contents

Part 1

문제 제시

No.11 지속가능한 도시와 공동체

소셜 다이닝 어플리케이션의 핵심 매칭 알고리즘을 개선하여,
생활 속 공동체 형성에 기여하고자 합니다.

알고리즘 최적화는 단순 기술적 개선을 넘어서는 가치를 가집니다.

사용자 간의 연결속도를 극대화하여
사회적 관계망 형성의 가속화를 기대할 수 있습니다.

이는 지속가능발전목표(SDGs)에서
"지속가능한 도시와 공동체" 실현에 이바지 합니다.

빠르고 실현 가능한 매칭 알고리즘을 통해 사용자들간 연결을 용이하게 만들고

새로운 사회적 연결을 만들어내는 긍정적인 변화를 가져올 것입니다.

Part 1

문제 제시

기존의 앱

{ Problem }

기존 매칭 시스템의
비효율적 알고리즘

First

매칭 진행시 유저마다
모든 db의 내용을
재탐색해야 함

Second

불필요한 탐색 진행
선형 탐색 진행(비효율적)

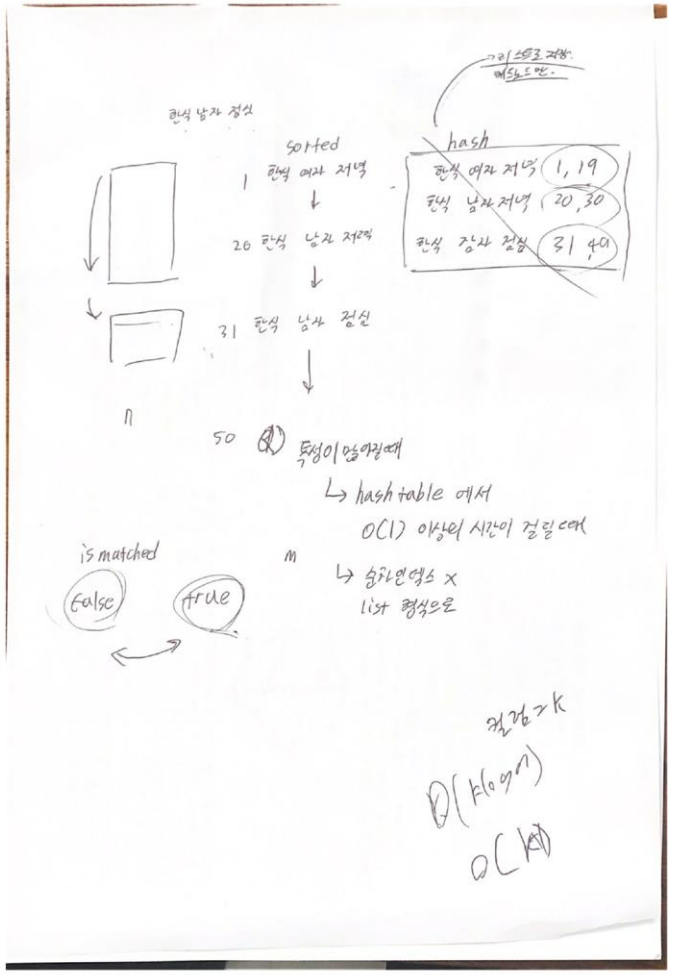
Third

소요 시간 복잡도 : n * O(n) = O(n^2)

Part 2

방법 제시

Part
2
방법 제시



브레인 스토밍

1. db가 언제나 sorted 되어있게 만들자

2. hash table을 사용해서 특성들의 조합만큼의 버킷을 만들자

3. hash table을 리스트 형태로 저장하여 hash table을 갱신할 필요가 없도록 하자

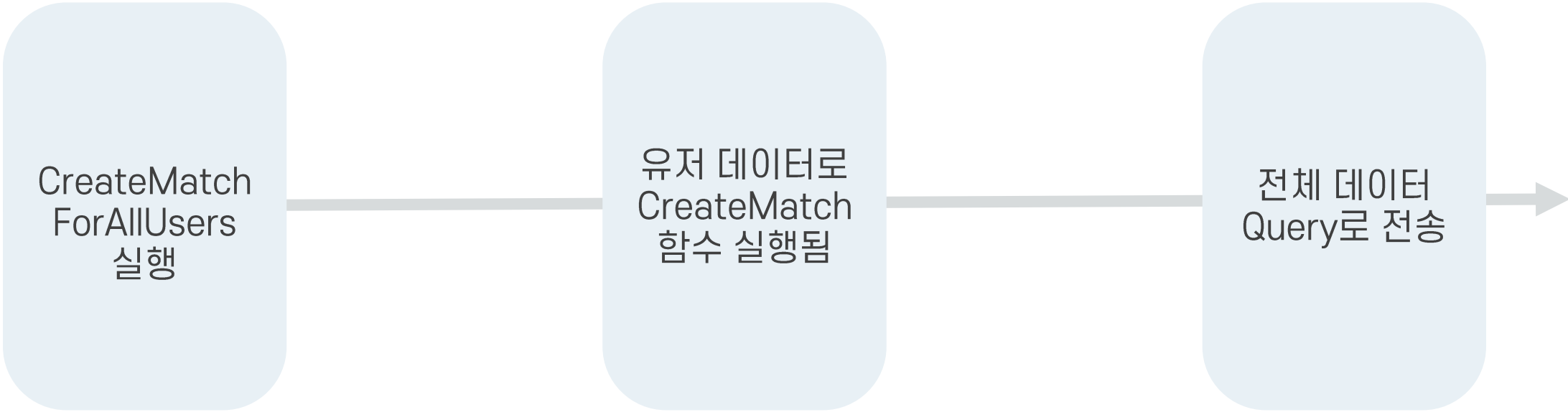
4. 모든 것을 내려다두고 이진탐색트리를 만들고 저장해놓자

소요 시간 복잡도 : n * O(n) = O(n^2)
소요 시간 복잡도 : O(n log n) + O(log n) * k = O(k log n)

Part 2

방법 제시

현재 상황



소요 시간 복잡도 : CreateMatch 실행시마다 O(n) 소요 → $O(n^2)$

방법 제시

개선 상황

각 feature별 value의 곱 (i*j*k개) 이진 탐색 트리 생성 및 저장

CreateMatch 실행 ↓ 유저의 각 feature 별 이진 탐색 트리 테이블 탐색 (log n)

유저의 feature와 비교하여 일치하는 데이터 리스트 생성

일치 리스트의 첫 값과 Match 진행 ↓ 이진 탐색 트리 삭제 알고리즘 실행

소요 시간 복잡도 : O(n log n) + O(log n) = O(n log n)

Part 2

방법 제시

```java
@Transactional
public void createMatchForAllUsers() {
    List<User> users = userRepository.findAll();
    List<Match> matches = new ArrayList<>();
    for (User user : users) {
        if(!user.isMatched() && !(user.getMatchForm()==null) ){
            Match match = createMatch(user);
            if (match != null) {
                matches.add(match);
            }
        }
    }
  /*  return matches; // 모든 매치 결과 반환*/
}

@Transactional
public Match createMatch(User user) {
    lock.lock(); //락 함걸어봄
    try {
        MatchForm matchForm = user.getMatchForm();
        List<User> potentialMatches = userRepository.findMatches
```

Before

```java
// 모든 유저를 트리에 삽입하는 메서드
private void initializeTreeWithUsers() {
    List<User> users = userRepository.findAll();
    for (User user : users) {
        if (!user.isMatched() && user.getMatchForm() != null) {
            addToIndex(user);
        }
    }
}

@Transactional
public void createMatchForAllUsers() {
    List<User> users = userRepository.findAll();
    List<Match> matches = new ArrayList<>();
    for (User user : users) {
        if (!user.isMatched() && user.getMatchForm() != null) {
            addToIndex(user);
            Match match = createMatch(user);
            if (match != null) {
                matches.add(match);
            }
        }
    }
    // return matches; // 모든 매치 결과 반환 (필요 시)
}
```

After

Part 3

해결 과정

사용 기술

이진 탐색 트리를 사용한 테이블 파트 추가 과정 중
Google의 생성형 AI “Gemini” 를 사용하여 코드의 효율성을 높임.

해결 과정

사용 함수

```
public MatchingService(UserRepository userRepository, MatchRepository matchRepository, MatchFormRepository matchFormRepository) {
    this.userRepository = userRepository;
    this.matchRepository = matchRepository;
    this.matchFormRepository = matchFormRepository;
    initializeTreeWithUsers();
}
```

MatchingService

각 특성을 분기로 하는
이진 탐색 트리 테이블 생성

해결 과정

사용 함수

```java
// 모든 유저를 트리에 삽입하는 메서드
private void initializeTreeWithUsers() {
    List<User> users = userRepository.findAll();
    for (User user : users) {
        if (!user.isMatched() && user.getMatchForm() != null) {
            addToIndex(user);
        }
    }
}
```

initializeTreeWithUsers

유저 데이터를 이진 트리에 삽입하는 함수

해결 과정

사용 함수

```java
@Transactional
public Match createMatch(User user) {
    lock.lock(); //락 함걸어봄
    try {
        MatchForm matchForm = user.getMatchForm();
        List<User> potentialMatches = userRepository.findMatchesByPreferences(
                matchForm.getFoodType(), matchForm.getTimeSlot(), matchForm.getPreferGender(), false);

        for (User potentialMatch : potentialMatches) {
            if (!potentialMatch.isMatched() && !potentialMatch.getId().equals(user.getId())
                    && !potentialMatch.getGender().equals(user.getGender())) {
                Match match = new Match(user, potentialMatch, new Date(), matchForm.getTimeSlot(), matchForm.getFoodType());

                potentialMatch.setMatched(true);
                user.setMatched(true);

                userRepository.save(potentialMatch);
                userRepository.save(user);
                return matchRepository.save(match);
            }
        }
        return null; // No match found
    } finally {
        lock.unlock(); // 락 임시
    }
}
```

CreateMatch

매칭 함수

isMatched가 false이고
,
각 feature의 값이
matchform과 같지
않은 경우일 때,

match 실행

해결 과정

사용 함수

```
@Transactional
public void createMatchForAllUsers() {
    List<User> users = userRepository.findAll();
    List<Match> matches = new ArrayList<>();
    for (User user : users) {
        if(!user.isMatched() && !(user.getMatchForm()==null) ){
            Match match = createMatch(user);
            if (match != null) {
                matches.add(match);
            }
        }
    }
  /*  return matches; // 모든 매치 결과 반환*/
}
```

CreateMatchForAllUsers

모든 user에 대해
match를 실행하는 함수

Part 4

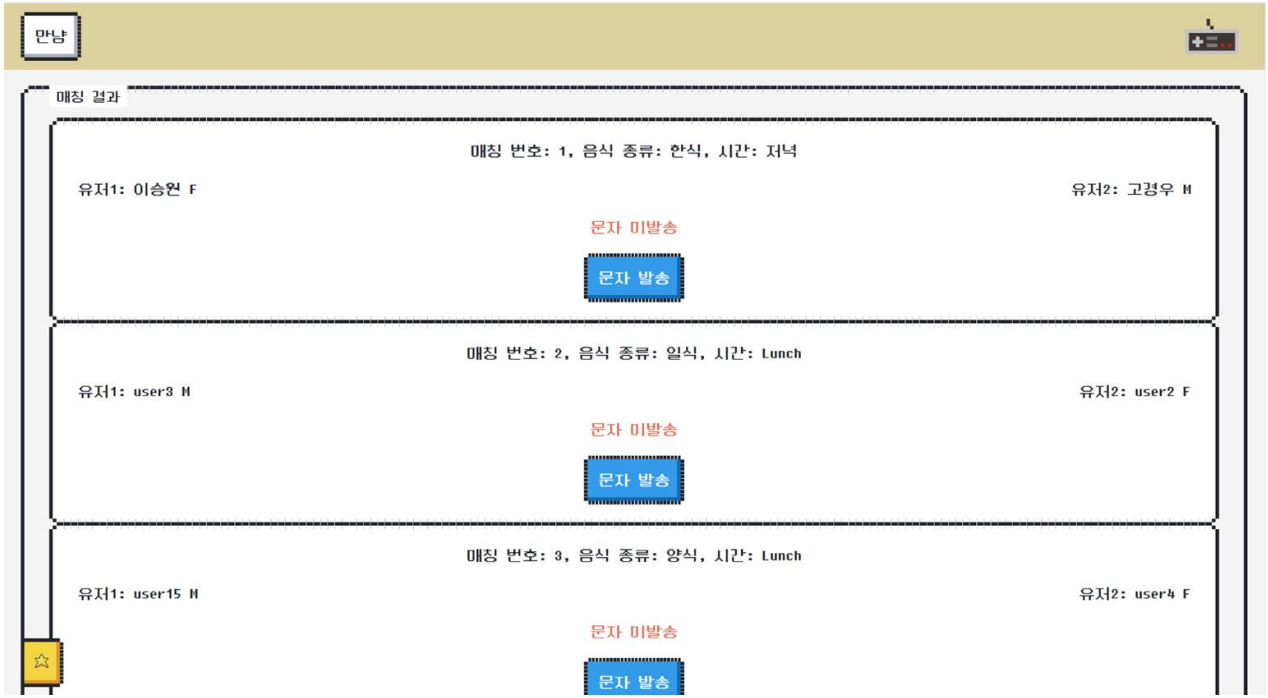
결과

결과 화면

웹 페이지의 결과 화면

매칭 index, 음식 종류, 시간, 매칭된
두 유저 이름이 정상적으로 표시되는 것을
확인할 수 있음.

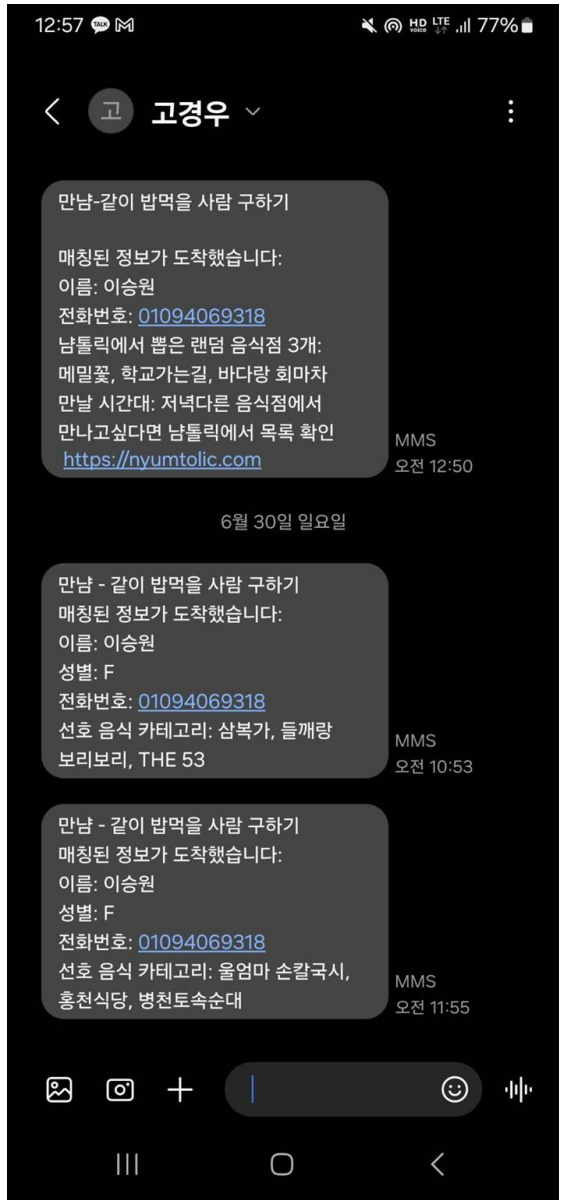
문자 발송 버튼을 누르면 실제로 유저에게
문자가 발송됨.



결과 화면

문자 발송 화면

매칭 정보와 핵심 정보인 선호 음식 종류가 정상적으로 발송된 것을 볼 수 있음.



결과 화면

최종 DB 화면

모든 과정이 완료 된 후 데이터가 정상적으로 저장된 것을 확인.

소요 시간도 이전 코드의 경우보다 더 줄어든 것을 확인.

엄청난 개선 및 확실한 개선

코드 경량화 프로젝트 성공

$O(n^2) \rightarrow O(n \log n)$

SELECT * FROM MATCH;

SENDED	ID	MATCH_DATE	USER1_ID	USER2_ID	FOOD_TYPE	TIME_SLOT
FALSE	1	2024-06-30 11:48:16.667	2	1	한식	저녁
FALSE	2	2024-06-30 11:48:16.685	5	4	일식	Lunch
FALSE	3	2024-06-30 11:48:16.687	17	6	양식	Lunch
FALSE	4	2024-06-30 11:48:16.688	19	10	중식	Evening
FALSE	5	2024-06-30 11:48:16.689	20	13	일식	Evening
FALSE	6	2024-06-30 11:48:16.69	22	3	한식	Evening
FALSE	7	2024-06-30 11:48:16.692	24	11	한식	Lunch
FALSE	8	2024-06-30 11:48:16.693	25	8	양식	Lunch
FALSE	9	2024-06-30 11:48:16.694	26	21	일식	Evening
FALSE	10	2024-06-30 11:48:16.696	28	9	중식	Lunch
FALSE	11	2024-06-30 11:48:16.697	29	12	양식	Lunch
FALSE	12	2024-06-30 11:48:16.698	30	27	일식	Evening
FALSE	13	2024-06-30 11:48:16.7	31	16	일식	Lunch
FALSE	14	2024-06-30 11:48:16.702	34	15	중식	Lunch
FALSE	15	2024-06-30 11:48:16.703	38	7	한식	Evening
FALSE	16	2024-06-30 11:48:16.705	41	32	양식	Evening
FALSE	17	2024-06-30 11:48:16.707	43	40	중식	Lunch
FALSE	18	2024-06-30 11:48:16.708	44	39	양식	Lunch
FALSE	19	2024-06-30 11:48:16.71	57	36	양식	Evening
FALSE	20	2024-06-30 11:48:16.712	61	42	중식	Lunch
FALSE	21	2024-06-30 11:48:16.714	63	14	중식	Evening


THANK YOU


#GDSC