백준 연동 소크라테스식 AI 문제풀이 튜터

기획 요약

1. 프로젝트 개요

- **서비스명**: 백준 연동 소크라테스식 AI 문제풀이 튜터 (Chrome Extension)
- **형태**: 크롬 익스텐션 기반 학습 보조 도구
- **목표**: 정답을 직접 제공하지 않고, 질문 기반 힌트를 통해 사용자의 사고 과정을 유도하는 AI 튜터 제공
- **기획 배경**: 기존 AI 풀이 도구는 즉각적인 정답 제공으로 학습 효과를 저해함
- **핵심 철학**: AI를 치트 도구가 아닌 사고 훈련용 튜터로 활용

2. 소크라테스식 사고 개요

2.1 정의

> 소크라테스식 사고란, 답을 직접 제시하지 않고 질문을 통해 사고를 검증하고 확장하도록 유도하는 사고 방식이다.

2.2 핵심 원리

1. 가정의 검증
2. 개념 정의의 명확화
3. 근거 요구
4. 반례 및 예외 탐색
5. 논리적 귀결 도출

3. 핵심 기능: 단계별 강제 통제 힌트 시스템

> 본 서비스는 단순 프롬프트 제어가 아닌, **서버 단 로직을 통한 단계별 접근 통제**를 핵심으로 한다.

단계(Level)	접근 방식	제공 내용
0	자동 제공	문제 요약 및 제약 조건 정리
1	사용자 클릭	풀이 방향 힌트
2	사용자 클릭	사고 유도를 위한 소크라테스식 질문
3	사용자 클릭	핵심 아이디어 제시
4	사용자 클릭	접근 방법 및 의사코드
5	명시적 요청	최종 정답 코드 제공

- Level 4까지는 실제 코드 노출을 금지함
- Level 5는 사용자의 명시적 선택 시에만 제공됨

4. 백준 문제 적용용 소크라테스식 힌트 템플릿 (1/2)

Hint 1. 문제 재정의

- 문제에서 최종적으로 요구하는 값은 무엇인가
- 입력 정보 중 실제 계산에 사용되지 않는 요소는 없는가

Hint 2. 제약 조건 분석

- 입력 크기의 상한이 의미하는 바는 무엇인가
- 해당 조건에서 가능한 시간 복잡도는 무엇인가

Hint 3. 상태 및 반복 구조 분석

- 모든 과거 정보가 필요한가
- 요약된 정보로 다음 상태를 계산할 수 있는가

4. 백준 문제 적용용 소크라테스식 힌트 템플릿 (2/2)

Hint 4. 반례 및 경계값 검토

- 최소 입력 및 최대 입력에서의 동작은 어떠한가
- 특수한 값 분포에서도 동일한 로직이 성립하는가

Hint 5. 알고리즘 범주 도출

- 선택 문제인지 여부 (Greedy)
- 이전 상태를 활용하는 문제인지 여부 (DP)
- 정렬 후 규칙성이 발생하는지 여부
- 탐색 또는 그래프 구조 문제인지 여부

5. 기술 아키텍처

5.1 데이터 흐름

백준 문제 페이지 → Chrome Extension → 백엔드 API → LLM

5.2 구성 요소

프론트엔드

- Chrome Extension (Manifest V3)
- 백준 문제 페이지의 DOM을 파싱하여 문제 정보 추출

백엔드

- FastAPI 기반 REST API
- 문제별 세션 관리 및 단계별 응답 통제

AI 모델

- OpenAI gpt-4o-mini

RAG 구조

- 알고리즘 개념, 문제 유형 패턴 데이터 참조
- 문제별 정답 코드 데이터는 저장 및 참조하지 않음

6. 브라우저 기반 Serverless RAG 아키텍처

6.1 핵심 개념

> 본 프로젝트의 RAG(Retrieval-Augmented Generation)는 일반적인 서버 기반 RAG와 달리, **브라우저 확장 프로그램 내부에서 RAG 엔진 전체가 실행**되는 구조를 가진다.

- 브라우저 자체가 RAG 엔진의 실행 주체이며, 서버는 검색·판단·프롬프트 구성 과정에 관여하지 않음
- 별도의 벡터 DB 서버나 백엔드 RAG 파이프라인 없이, 클라이언트(브라우저) 단에서 완결되는 **Serverless RAG** 구현

6. 브라우저 기반 Serverless RAG 아키텍처

6.2 RAG 엔진의 기준 정의

RAG 엔진은 다음의 구성 요소들이 하나의 실행 단위로 결합되어 동작할 때 성립한다.

구성 요소	역할
Query Builder	입력 문제를 검색 가능한 질의 형태로 변환
Knowledge Store	검색 대상이 되는 지식 저장소
Retriever	관련 지식 선택 및 랭킹
Prompt Composer	검색 결과를 프롬프트로 조합
Generator Caller	LLM 호출 및 응답 수신

> 이 다섯 요소가 입력 → 판단 → 변환 → 출력의 흐름을 자율적으로 수행하면, 해당 실행 단위는 "엔진"으로 정의할 수 있다.

6. 브라우저 기반 Serverless RAG 아키텍처

6.3 일반적인 서버형 RAG와의 구조적 차이

서버형 RAG 구조 (일반적 사례)

```
Client(UI)
  ↓
Backend Server (RAG Engine)
 ├ Query Builder
 ├ Retriever
 ├ Vector DB
 ├ Prompt Composer
 └ LLM Caller
```

- RAG 엔진은 서버에 위치
- 클라이언트는 입력·출력 UI 역할만 수행
- 검색·판단 로직은 서버가 담당

본 프로젝트의 브라우저 기반 RAG 구조

```
Browser Extension (RAG Engine)
 ├ Query Builder
 ├ Knowledge Store
 ├ Retriever
 ├ Prompt Composer
 └ LLM Caller
```

- 엔진 전체가 브라우저 프로세스 내부에서 실행
- 서버는 존재하지 않거나, 본 RAG 흐름에 관여하지 않음
- 브라우저는 단순 UI가 아니라 실행 주체

6. 브라우저 기반 Serverless RAG 아키텍처

6.4 실제 코드 기준 구성 요소 매핑

본 프로젝트에서 RAG 엔진의 각 구성 요소는 다음과 같이 브라우저 내 JS 파일에 매핑되어 있다.

RAG 구성 요소	담당 파일	역할 설명
Query Builder	content.js	백준 문제 페이지 DOM에서 문제 텍스트 추출
Knowledge Store	algorithm-db.js	알고리즘 개념·패턴을 담은 정적 JS 객체
Retriever	rag.js	문제 텍스트와 지식 DB를 매칭하여 Top-K 알고리즘 선정
Prompt Composer	ai.js	검색 결과를 시스템 프롬프트에 주입
Generator Caller	ai.js	OpenAI API 직접 호출
Flow Orchestrator	sidepanel.js	사용자 단계(Level) 및 전체 흐름 제어

> 이 파일들이 결합되어 **브라우저 = RAG 엔진**이라는 구조가 성립한다.

6. 브라우저 기반 Serverless RAG 아키텍처

6.5 브라우저 프로세스 내부 실행 흐름

1. 문제 수집 (Query 생성)

- content.js가 백준 문제 페이지의 DOM을 직접 분석
- 문제 설명, 입력/출력 조건을 브라우저에서 즉시 확보
- 서버 전송 과정 없음

2. 검색 쿼리화 및 매칭 (Retrieval)

- rag.js에서 문제 텍스트를 분석
- algorithm-db.js에 정의된 알고리즘 키워드와 매칭
- 키워드 스코어링을 통해 관련 알고리즘 유형 계산

3. Top-K 선택

- 점수가 높은 알고리즘 유형을 선별
- 선택된 알고리즘의 핵심 개념(coreIdea), 접근법(approach)을 컨텍스트로 구성
- 이 단계에서 Retrieval 완료

6. 브라우저 기반 Serverless RAG 아키텍처

6.5 브라우저 프로세스 내부 실행 흐름 (계속)

4. 프롬프트 조합 (Prompt Composition)

- ai.js에서 검색된 컨텍스트를 시스템 프롬프트에 주입
- 문제 레벨 정책(Level 0~5)과 결합
- LLM이 참고할 지식과 발화 범위를 명확히 제한

5. LLM 호출 (Generation)

- 브라우저에서 OpenAI API를 직접 호출
- 서버를 거치지 않는 Client → LLM 구조
- LLM은 판단 주체가 아니라 생성기(Generator) 역할만 수행

6. 브라우저 기반 Serverless RAG 아키텍처

6.6 Knowledge Store 구조 (Serverless DB)

- 별도의 벡터 DB 사용 없음
- algorithm-db.js에 정의된 정적 JS 객체가 지식 저장소 역할 수행

```
const ALGORITHM_DB = {
  dp: {
    name: "다이나믹 프로그래밍",
    keywords: ["최소", "최대", "점화식"],
    coreIdea: "...",
    approach: ["상태 정의", "점화식 도출"]
  }
};
```

- 브라우저 메모리 상에서 즉시 접근 가능
- 네트워크 지연 없음
- 정답 코드 미포함 (개념·패턴 중심)

6. 브라우저 기반 Serverless RAG 아키텍처

6.7 왜 "브라우저가 RAG 엔진"인가

엔진 조건	충족 여부
입력 처리	가능
지식 탐색	가능
판단 및 선택	가능
출력 생성	가능
서버 의존	없음

> 브라우저는 단순히 API를 호출하는 UI가 아니라,
> **무엇을 검색할지 결정**하고, **어떤 지식을 선택할지 판단**하며, **어떤 수준까지 출력할지를 통제**한다.
> 따라서 브라우저는 RAG 실행의 주체, 즉 **엔진**으로 정의된다.

6. 브라우저 기반 Serverless RAG 아키텍처

6.8 구조적 특성 요약

장점

- 서버 비용 없음
- 즉각적인 응답 속도
- 정책·지식 통제 용이
- 교육용·치트 방지 목적에 적합

한계

- 대규모 지식 확장 어려움
- 고도화된 검색(벡터 검색 등) 제한
- 사용자 로그 기반 개인화 분석 불가

본 프로젝트는 RAG의 핵심 구성 요소(Query, Retrieval, Prompt Composition, Generation)를 서버가 아닌 브라우저 확장 프로그램 내부에서 수행하는 **Client-side Serverless RAG 아키텍처**를 구현한 사례이다.

7. 데이터 및 보안 정책

- 사용자 로그인 기능 미제공
- 개인 식별 정보 수집 및 저장 없음
- 사용자 코드 저장 없음
- 오류 로그만 최대 7일 보관
- OpenAI API Key는 서버 환경 변수로 관리

8. MVP 성공 기준

- 사용자가 최종 정답(Level 5) 확인 이전 단계에서 이탈하지 않을 것
- 힌트 단계 체류 시간 증가
- 최종 정답(Level 5) 버튼 사용 비율 50% 이하 유지

9. 기대 효과

- 정답 의존도를 낮춘 자기주도적 문제 해결 유도
- 알고리즘 사고 과정의 구조화
- 코딩 테스트 및 기술 면접 대비 학습 도구로 활용 가능