Task: Recommended Jobs Algorithm

Candidate Information

1. **Name**: Himanshu Sharma
2. **Trilogy**: Technical Product Owner at Trilogy

Important Assumptions

1. The algorithm assumes the `*canApply(candidate_id, job_id)*` function accurately filters out jobs a candidate is not eligible to apply for.
2. Only jobs that are marked as 'active' in the jobs table are considered for recommendations.
3. A candidate should not be recommended for a job they have already applied for.
4. The *job_recommendations* mapping is curated manually by hiring managers and is assumed to reflect meaningful relationships.
5. The system is expected to always return at least some recommendations with jobs that may be very loosely related to the candidate's profile. But this handling is better than not showing any job at all.

Important Decisions

1. The recommendation logic uses a **one-hop graph traversal**: recommendations are only fetched directly from previously applied jobs using *job_recommendations*.
2. The candidate's history is checked to avoid re-recommending previously applied jobs.
3. Inactive jobs are filtered out to ensure only valid options are displayed.
4. All eligible job recommendations will be returned without ranking in the current version; future enhancements may introduce prioritization based on candidate engagement signals, potentially leveraging LLMs such as LLaMA for contextual relevance scoring.

Pseudocode

```
# Function to generate job recommendations for a candidate

function recommendedJobs(candidate_id):
    # Retrieve all jobs the candidate has previously applied for
    applied_jobs = getAppliedJobs(candidate_id)

    # Initialize an empty set to store recommended job IDs
    recommendations = set()

    # Loop through each previously applied job
```

```
    for job in applied_jobs:
        # Fetch jobs that are recommended based on this job
        related_jobs = getRelatedJobs(job)

        # Evaluate each related job
        for r_job in related_jobs:
            # Check if the job is active, not already applied to, and the candidate is eligible
            if isActive(r_job) and not hasApplied(candidate_id, r_job) and
canApply(candidate_id, r_job):
                # Add the job to the recommendations set
                recommendations.add(r_job)

    # Return the final list of recommended jobs
    return recommendations
```

Sample Data

Applications:

id	candidate	job	status	last_activity
1	1	PCTO	rejected	2021-01-19 10:00
2	2	VPoE	inProgress	2022-01-19 10:00
3	2	SCTO	rejected	2021-01-19 10:00
4	3	CSA	rejected	2023-01-19 10:00

Job_recommendations:

source_job	recommends_job
PCTO	SCTO
SCTO	CSA
VPoE	SEM

VPoE	SSE
VPoE	CSA
SEM	SEIII
SSE	SEIII
SEIII	SEII
SEII	QA

Jobs:

id	name	compensation	status
PCTO	Product CTO	200	active
SCTO	Subsystem CTO	100	active
CSA	Cloud Software Architect	50	active
SEM	Software Engineering Manager	40	active
SSE	Senior Software Engineer	50	inactive
SEIII	Software Engineer III	30	active
SEII	Software Engineer II	15	active
QA	Manual QA Tester	10	active

Acceptance Tests

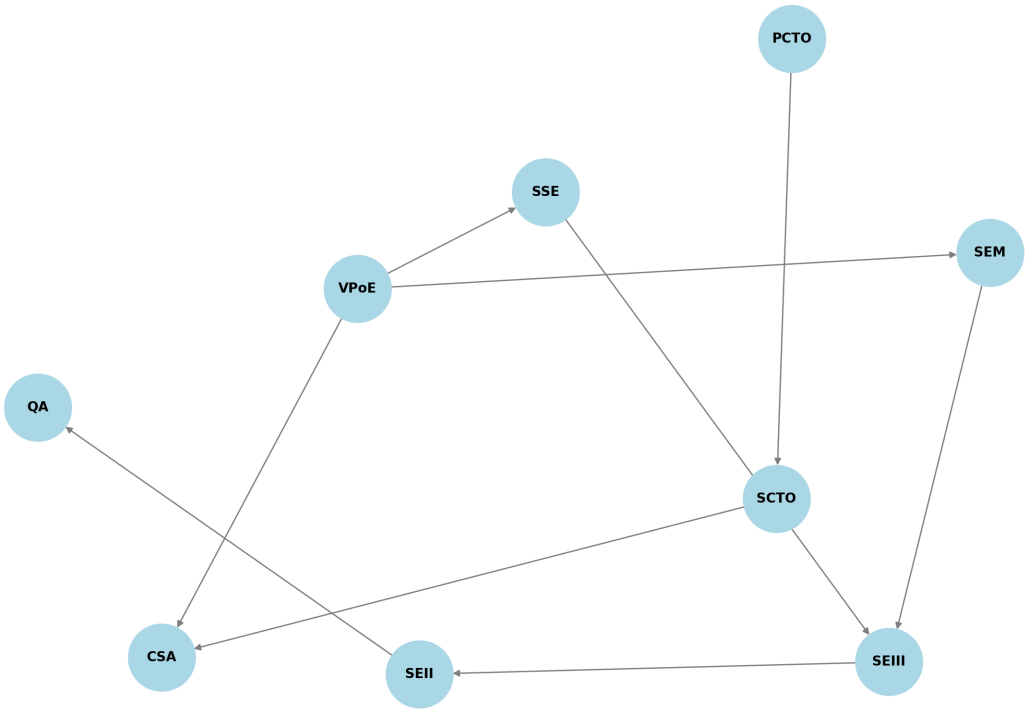
Test 1: Recommend jobs for candidate 1
Input: candidate_id = 1
Expected Output: [SCTO, CSA]
Reason: Candidate applied to PCTO. PCTO recommends SCTO, and STCO recommends CSA. Both are active and eligible.

****

Test 2: Recommend jobs for candidate 2
Input: candidate_id = 2
Expected Output: [SEM, CSA]
Reason: Candidate applied to VPoE and SCTO. VPoE recommends SEM (active), SSE (inactive) and CSA.
****

Test 3: Recommend jobs for candidate 3
Input: candidate_id = 3
Expected Output: []
Reason: CSA not in source_job column in job_recommendations; no recommendations in 1-hop mapping. If extended, SEIII, SEII, QA etc. may be recommended from other chains.
****

Job Recommendation graph



Job Recommendation using LLM

To enhance the relevance of job recommendations and **increase retention of candidates at CROSSOVER**, we propose leveraging a large language model (LLM) to power the *getRelatedJobs(job)* function. Instead of relying solely on manually curated job-to-job mappings, the LLM can analyze job titles, descriptions, and required skills to

identify semantically similar roles. This dynamic approach enables the system to discover relationships between jobs that may not be explicitly linked in the database but are logically connected based on context and capabilities. Over time, this can improve recommendation quality, uncover overlooked job paths, and adapt to evolving job markets without manual intervention.