|  |
| --- |
| **온라인 저지 문제 집합 추천 알고리즘** |
|  |
| Online Judge Problem Set Recommendation |
|  |
| **요 약**  2022년 현재 많은 사람들이 프로그래밍 대회, 코딩 테스트 등을 준비하기 위하여 온라인 저지를 사용하고 있다. 많은 온라인 저지 사이트들 중, 오랜 기간 운영되고 있는 Baekjoon Online Judge(이하 BOJ)는 사용자에게 문제를 추천해주는 기능이 없다. 이에 본 알고리즘(혹은 프로그램)은 사용자의 BOJ 아이디를 입력 받고, 다른 사용자들이 해결한 문제들 중 나와 비슷한 사용자의 데이터와 전체 문제 데이터를 이용하여 사용자에게 문제들을 추천 하는 것을 목표로 한다. | | |

**1. 서 론**

온라인 저지(Online Judge)는 알고리즘이나 프로그램 등을 해결하고 시험해 볼 목적으로 만들어진 온라인 사이트, 시스템이다. 사용자는 주로 사이트에 회원가입을 하여 해당 사이트를 이용한다. 문제가 주어지면 사용자는 해당 문제의 입력을 받는 프로그램을 작성하여 문제의 test case(문제의 입력들을 말한다. 각 입력에 해당하는 답이 존재하며 적절한 출력으로 문제를 해결한다.)마다 올바른 답을 도출해야 한다. 모든 test case를 통과하는 것을 문제를 해결했다고 한다. 현대 사회에서는 많은 IT 기업들이 채용이나 여타 목적을 위하여 코딩테스트를 시행하고 있다. 또한 IT에 관하여 사람들의 관심이 많아지고 이에 따라 다양한 프로그래밍 대회들이 주최되고 있다. 이를 준비하기 위하여 온라인 저지를 이용하는 사람들이 많다. 하지만 처음 온라인 저지를 이용하거나 문제에 대한 경험이 많이 없을 경우, 사용자들은 어떠한 문제를 풀어봐야 하는지조차 모르는 경우가 있다. 혹은 어느 정도 경험이 쌓였다고 해도 무작정 여러 문제들을 풀어보거나 하는 사람들도 있다. 본 논문은 이들 중 BOJ를 이용하는 사람들을 대상으로 앞서 말한 것들을 해결할 방안으로 문제 추천 알고리즘을 제시한다.

BOJ는 2010년에 개설되어 현재까지 많은 사람들이 이용하고 있다. BOJ는 23492개의 채점 가능한 문제를 가지고 있으며, 다양한 프로그래밍 언어를 지원한다. 또한 사용자들이 만든 문제집(문제들을 모은 것으로 주제에 관련된 문제들이 포함되어 있다.), 대회, 해결한 문제들을 기준으로 한 랭킹, 게시판 등의 기능을 사용자에게 제공한다. (2022년 10월 24일 기준) 하지만 BOJ에는 문제를 추천해주는 기능이 없다(예를 들어 ‘별 찍기 1’라는 문제의 경우, 비슷한 유형의 문제인 ‘별 찍기 2’와 같은 것은 문제의 페이지에 적혀 있긴 하다. 하지만 이는 출제자의 재량이다). 만약 어떤 사용자가 ‘다이나믹 프로그래밍’이라는 문제의 tag를 가진 문제를 풀었다고 가정하자. 그렇다면 이 사용자가 앞서 풀어본 문제를 더 연습하고 싶거나 다른 사람들이 해결한 문제들에 대해서 추천을 받고 싶은 경우가 생길 것이다. 이러한 경우를 해결하기 위해서는 사용자에게 문제를 추천해줄 시스템이 필요하다.

데이터를 모으는 방법은 BOJ 사이트를 직접 crawling 하고, Solvedac (BOJ 미러 사이트이자 커뮤니티 프로젝트이다. 사용자들이 직접 문제의 난이도를 평가하고 기여하여 집단지성을 이용하는 것으로 문제의 난이도를 정한다. 이 사이트 또한 BOJ 사용자의 데이터를 가지고 있다.) API를 이용하여 사용자에 대한 정보와 문제에 대한 정보를 구한다.

**2. 용 어**

rank는 백준에서의 해결한 문제 수에 따른 순위를 말한다. target user (TU)는 문제를 추천 받고 싶어하는 사용자를 말한다. 이 때, TU는 solvedac와 BOJ가 연동이 되어있다고 가정한다. similar boundary (SB)는 TU의 rank로부터 얼만큼 구할 것인지에 사용된다. similar user (SU)는 TU의 rank을 기준으로 하여 구한 사용자들을 말한다. (예로 들어 TU의 rank가 6325등이고, similar boundary가 5라 하자. 이 때 SU는 6320~6324, 6326~6330등의 사용자를 말한다.) error users (EU)는 정보의 일부가 없는 사용자를 말한다. 주로 Solvedac에 가입이 되어 있지 않거나, 해결한 문제의 대부분을 데이터로서 사용할 수 없는 경우이다. 해결한 문제는 BOJ에서 주어진 문제에 대하여 모든 test case를 통과하고 해결한 문제들을 말한다. 문제의 id는 BOJ에서 생성된 문제를 식별하기 위한 번호로 최대 5자리의 자연수이다. tier는 문제나 사용자의 수준을 말한다. unrated, 브론즈 5 ~ 1, 실버 5 ~ 1, 골드 5 ~ 1, 플래티넘 5 ~ 1, 다이아 5 ~ 1, 루비 5 ~ 1, 마스터로 구분되어 있으며 언급된 순서대로 각 0부터 31까지 수 하나에 대응된다. (예시: 브론즈 1은 5, 골드 2는 14) rating은 사용자가 해결한 문제들 중 Solvedac에 등록된 문제(BOJ에 모든 문제가 등록이 되어있으나 사용자들의 난이도 평가가 부족하여 해결한 문제에 들어가지 않는 경우가 있다. 즉 이는 BOJ에서 해결한 문제의 수와 Solvedac에서 해결한 문제의 수가 다름을 말한다. 본 알고리즘에서는 BOJ에서 해결한 문제의 수를 말하며 이용한다.)에서 tier가 높은 (숫자가 높은, 어려운) 문제 상위 100개를 기준으로 계산된 사용자의 수준을 말한다. 클수록 더 어려운 문제를 해결했음을 의미한다. tag는 문제의 태그를 말한다. BOJ에는 190개의 태그가 존재한다. (예로 들어 백준 1865번 ‘웜홀’ 문제의 경우 ‘그래프 이론’, ‘벨만-포드’의 두 가지 태그를 가지고 있다.) zss는 Tree edit distance를 계산할 수 있는 python module을 말한다. zss의 기원은 알고리즘을 만든 사람들인 Zhang-Shasha에서 왔다.

**3. 알고리즘**

3.1 데이터 생성

3.1.1 문제에 대한 정보

문제에 대하여 필요로 하는 것은 문제의 id, 문제의 제목, 문제의 tier, 문제의 tags가 필요하다. 또한 문제들은 새로 생기거나 없어지는 주기가 긴 편이므로 매번 crawl 하는 것보다 한 번 csv 파일을 만드는 것이 좋다. crawling은 파이썬의 requests와 json module (이하 파이썬 module들에 대해서 module은 생략), Solvedac API를 사용한다. requests를 사용하여 query를 날릴 때 필요한 인자로 page라는 것이 있다. 이는 한 page에 50문제씩 있기에 1 page부터 494 page까지 for loop을 이용해 문제를 가져오는 것으로 정한다. 따라서 본 프로그램이 추천할 수 있는 문제는 BOJ 기준 24700번까지로 잡는 것이 안정적이다. (모종의 이유로 없어진 문제들도 존재하기 때문에 실제 최대 번호는 2022년 10월 23일 기준 25880번까지이다.) 이후 pandas의 to\_csv() 함수를 사용하여 csv 파일로 만든다. column은 문제 id, 문제 hash (문제에 100을 곱한 값), 문제 제목, 문제 tier, 문제 tags로 구성된다. 총 용량은 1.6MB 정도이고 파일명은 ‘problem\_list2.csv’이다. 이렇게 crawl 한 문제들의 정보는 ‘main.py’에서 사용된다.

3.1.2 사용자에 대한 정보

사용자들에 대해서 필요로 하는 것을 구하는 과정은 TU, SU에 따라 구분될 필요가 있다. 먼저 TU는 입력 받은 사용자의 아이디를 사용한다. Solvedac API와 requests를 이용하여 이 아이디를 query문의 handle에 넘긴 후, rating, tier을 구한다. 그리고 BeautifulSoup과 requests를 이용하여 html을 parsing한다. 여기서 BOJ 기준 해결한 문제 수와 rank를 구한다. (Solvedac API를 통하여 얻을 수 있는 값들과는 다른 값이다.) 이렇게 구한 값들을 dictionary 형태로 저장한다. 이후 SU를 생성하기 위해서는 먼저 SU에 어떤 사람들이 들어가야 하는지, 미리 정해진 SB (본 알고리즘에서는 10으로 설정. 이후 결론에서 추가 서술)를 기준으로 TU rank의 upper bound (rank + SB), lower bound (rank - SB)에 해당하는 사용자들을 crawl 하여 리스트를 생성한 후 TU와 같이 rating과 tier, BOJ 기준 해결한 문제 수, rank를 구한다. SU를 생성할 때 주의 할 점으로 SB에 따라서 원하는 SU들이 다른 페이지에 있을 수 있다. 그 이유는 한 페이지 당 100등을 간격으로 페이지가 나뉘기 때문이다. 이러한 상황을 구분하여 구현을 하였는데, 해당하는 코드에 따라 SB의 최대 값은 99로 한다. 마지막으로 TU, SU가 해결한 문제들의 id들을 BOJ에서 BeautifulSoup과 requests를 이용하여 crawl 한다.

3.2 알고리즘

3.2.1 간략한 알고리즘 흐름에 대한 설명

문제들을 추천하기 위한 알고리즘을 서술하기 전에 간략하게 요약을 한다. 입력으로 사용자의 아이디 (TU)를 받는다. 그리고 TU와 비슷한 사용자(SU)들을 찾은 뒤, 정보들을 crawling 한다. TU와 SU들이 해결한 문제들의 tag를 tag 별로 수를 세고, 이를 값으로 가지는 tag table을 만든다. tag table의 row는 사용자들, column은 tag가 될 것이다. 이후 TU와 SU의 rating 차이와 cosine similarity 구하고, tag가 클 수록 부모 노드가 되는 tree를 생성하여 tree edit distance를 구한다. 앞서 구한 값들을 이용하여 사용자에 따른 문제별 problem score을 계산하고 table로 만든 다음 정렬한다(1). 그리고 사용자들이 해결한 문제들을 기준으로 Apriori Algorithm을 사용하여 frequent item set들의 table을 생성한다. 이러한 item set 들의 item(즉 문제)이 가지는 problem score의 총 합이 가장 큰 순으로 정렬을 한다(2). 마지막으로 (1)에서의 상위 5개 문제와 (2)에서의 상위 1개 문제 집합의 문제들을 사용자에게 추천한다.

3.2.2 세부 설명

번호와 순서에 따라 세부적으로 알고리즘을 설명하도록 하겠다. 1) 프로그램은 문제를 추천 받고자 하는 사용자의 아이디를 입력으로 받는다. 이 아이디가 TU가 된다. 2) ‘get\_problem.py’를 이용하여 생성된 ‘problem\_list2.csv’ 파일을 읽어들여 BOJ 문제의 정보를 가져온다. 각 문제는 문제 id, 문제 제목, 문제 tier, 문제 tag들의 정보를 갖는다. 3) TU의 rating, tier, 해결한 문제 수, rank를 구한다. 4) TU의 정보를 이용하여 SU들의 리스트를 생성한다. 이 때 solvedac에 연동이 되어 있지 않은 SU는 error user에 추가하고 SU 집합에서 제외한다. 5) SU 각각의 rating, tier, 해결한 문제 수, rank를 구한다. 6) TU의 해결한 문제들의 id들을 가져온다. 7) SU 각각의 해결한 문제들의 id들을 가져온다. 8) 6, 7의 과정에서 각 사용자들의 tier와 문제의 tier을 비교하여 아래로 2, 위로 1만큼 이내에 있는 문제들만 가져온다. (예를들어 사용자의 tier가 골드 2라면 가져오는 문제들의 tier은 골드 4부터 골드 1 사이의 문제들만 해당된다.) 이렇게 가져온 문제들의 tag를 이용하여 사용자별 전체 tag set을 구하고, 각 tag를 가지는 문제들이 몇 개가 있는지 센다. 9) SU들이 해결한 문제들의 전체 tag set을 구한다. 이 때 사용자들이 해결한 전체 문제 수의 10% 보다 작은 tag들은 제외한다. 그 이유는 전체 해결한 문제 수에서 지배적인 문제의 tag를 사용하고자 하는 목적과 해결한 전체 문제 수의 10% 이하의 tag는 큰 의미를 가지지 않기 때문이다. 10) column이 tag, row가 사용자, 사용자들이 해결한 문제가 가지는 tag의 수를 값으로 가지는 tag table을 생성한다. 첫번째 row는 tag set의 tag가 되고, 두번째 row는 TU가 해결한 문제들에 따른 tag 별 총수를 값으로 가지는 row가 된다. 이후 row들은 SU들에 관한 row이다. 11) tag table을 이용하여 TU에 대한 SU들의 cosine similarity를 소수점 2번째 자리까지 구하고, z-score로 normalization한다. normalization을 하는 이유는 TU와 SU에 따라 cosine similarity 값의 범위가 변하기 때문에 일관되게 cosine similarity를 반영하기 위해서이다. 12) TU와 SU에 대하여 사용자의 tag 수가 클 수록 부모 노드가 되는 unary tree를 생성한다. 예로 들어 ‘BFS’ tag의 수가 120, ‘DFS’ tag의 수가 110이면 ‘BFS’가 ‘DFS’의 부모 노드가 된다. tag들은 tag의 수에 따라 정렬을 한다. 그리고 사용자 각각의 tag를 label로 가지는 Node를 생성한다(zss module의 Node를 사용). 그런 다음 zss를 이용하여 TU와 SU의 tree edit distance를 구한다. tree edit distance를 구하는 이유는 사용자가 해결한 문제들의 수로 우선순위를 가지는 순서를 고려하고자 하기 때문이다. cosine similarity로는 사용자가 어떠한 tag를 가지는 문제들을 더 많이 풀고 선호하는 지에 대한 반영을 할 수 없다. 예로들어 편의상 tag를 a, b, c, d, e로 생각했을 때, 어떠한 사용자 X가 a tag를 가진 문제를 100개를 해결했고 그 다음으로 많이 해결한 문제가 b tag를 가진 문제 80개를 해결했다고 하자. 이후의 c, d, e의 순서대로 문제들을 해결했다고 가정하자. 그리고 이 사용자와 비교할 사용자 Y가 있고, Y는 b, c, d, e, a의 순서대로 문제를 해결했다고 하자. 그렇다면 X와 Y를 비교했을 때 둘 다 b, c, d, e의 문제 수 순서로 문제들을 해결해왔기 때문에 zss를 이용하여 tree edit distance를 구한다면 (X는 a→b→c→d→e, Y는 b→c→d→e→a의 트리를 가진다. 화살표의 왼쪽이 부모 노드, 오른쪽이 자식 노드) 2가 된다.(Y의 b에 부모 노드로 a를 추가, e의 자식 노드인 a를 제거) 따라서 해결한 문제의 수에 따른 경향성을 고려할 수 있기 때문에 이를 이용한다. 13) column이 문제 번호, row가 TU와 SU인 weighted problem score table을 생성한다. table의 값은 cosine similarity, tree edit distance, TU와 SU의 rating 차이를 고려한 problem score이 된다. 이는 아래와 같은 식으로 정의한다.

(csSU는 SU의 cosine similarity, tedSU는 SU의 tree edit distance, rTU는 TU의 rating, rSU는 SU의 rating을 뜻한다.) 위 공식을 살펴보면 먼저 SU의 cosine simiarity를 10으로 나눈 값을 곱한다. 그리고 SU의 tree edit distance의 역수를 곱한다. 이는 tree edit distance의 값이 클 수록 TU와 SU가 비슷하지 않을 정도가 크다는 것을 의미하므로 TU와 SU가 비슷하다면 problem score의 값을 높이기 위해서 역수를 취한 것이다. 이후 TU의 rating을 TU의 rating과 SU의 rating 차의 절댓값으로 나눈 값을 곱한다. 이는 TU와 SU의 rating이 서로 비슷할 수록 문제의 난이도가 TU에게 적절하다고 생각할 수 있다. 따라서 서로의 rating차가 작을 수록 이 값이 커지게 되며 (분모에 있으므로) problem score에 적절하게 반영된다. 이렇게 구한 problem score들을 내림차순으로 정렬한다. 14) Apriori algorithm을 이용하여 frequent item set들을 구한다. mlxtend를 사용하며, minimum support 값은 0.3으로 한다. 또한 TU가 해결한 문제에 대해서는 추천을 하지 않을 것이므로 SU가 해결한 문제들에서 TU가 해결한 문제를 제외한 문제들에 대해서만 고려한다. frequent item set의 item set들에 대해서 하나의 set에 속하는 item (즉 문제)들 각각의 problem score의 합이 가장 높은 frequent item set 하나를 구한다. 예를 들어 편의상 문제들을 a, b, c라고 하고 problem score이 각각 1, 2, 3이라고 하자. 그리고 frequent item set이 {a, b}, {a, c} 라고 한다면 각각의 problem score의 합은 3, 4가 된다. 이 때 사용자에게 {a, c}의 조합을 추천한다는 것이다. 15) 13에서 구한 table으로 가장 problem score이 높은 5개의 문제와 14에서 구한 item set에 속한 문제들을 문제 id, 문제 제목, 문제 tier, 문제 tags와 함께 출력한다.

**4. 결 과**

본 논문의 알고리즘을 적용하여 코드[[1]](#footnote-1)를 실행시켜서 얻은 결과는 다음과 같다. 먼저 TU는 ‘gnaroshi’, similar boundary는 10으로 하고 main.py를 실행했을 때 그림 1과 같은 결과를 얻을 수 있다. (2022년 10월 25일 기준)

Text

Description automatically generated

# 그림 1 main.py의 결과

2번째 줄은 사용자의 입력을 의미하며 각각 TU의 아이디, similar boundary를 의미한다. 3번째 줄은 TU에 대한 정보를 의미한다. 4~5번째 줄은 TU를 기준으로 구한 SU의 아이디 목록이다. 6~7번째 줄은 EU에 대한 정보를 말한다. 9번째 줄은 cosine similarity를 의미한다. 11번째 줄은 tree edit distance를 의미한다. 13~15번째 줄은 계산된 frequent item set들 중에서 problem score의 합이 가장 큰 item set에 속한 문제들의 정보이다. 17~21번째 줄은 problem score이 가장 큰 상위 5개의 문제에 대한 정보를 의미한다.

실행시간에 대해서 similar boundary(이하 SB)에 따른 실행시간은 SB가 5일 때 7~9초, SB가 10일 때 18~24초, SB가 20일 때 평균 33초이다. 사용자의 rank에 따른 실행시간은 일반적으로 rank가 높을 수록 해결한 문제의 수가 많아져서 소요되는 실행시간이 길어진다.

**5. 결론 및 고찰**

먼저 위 알고리즘이 추천한 문제들에 대해서 생각을 해보자. 먼저 추천된 문제들이 정말로 사용자들이 원하는 것인가? 이는 나와 비슷한 사용자들이 해결한 모든 문제에 대해 추천을 했기에 사용자가 원하는 문제들이 아닐 가능성이 있다. 예를 들어 어려운 문제들을 풀어보고 싶은 사용자들이나 어떠한 유형의 문제에 대해서 추천을 받고 싶어하는 사용자가 있을 수 있다. 전자의 경우, 문제들을 rating에 따라 걸러내는 작업에서 범위를 넓히고, similar boundary를 크게 하여 어느정도 해결할 수 있다. 따라서 사용자의 선호에 따라 similar boundary의 값을 변경하며 문제를 추천 받을 수 있다. 하지만 similar boundary가 커질 수록 SU들 중 많이 해결한 공통 문제가 우선적으로 추천되는 경향을 볼 수 있게 된다. 즉, 사용자가 well known 문제나 많은 사용자들이 해결해본 문제를 해결하지 않을 경우, 문제의 tier에 따른 문제들이 적게 추천된다는 것이다. 후자의 경우는 사용자가 원하는 tag를 입력받고 비슷한 사용자들이 해결한 문제들 중 해당하는 tag를 가진 문제들만 뽑아내는 식으로 데이터의 전처리를 해주면 해결 할 수 있다. 그렇다면 반대로 사용자가 많이 풀어보지 않은 문제에 대해서 추천을 원하는 경우에는 어떻게 할 수 있을까. 이는 tree edit distance를 본 알고리즘에서 역수를 취한 것 대신 본래의 distance를 이용한다면 해결할 수 있다. 즉 나와 반대로 문제를 해결한 사용자가 problem score에 기여하는 정도를 높이는 것이다.

위 알고리즘을 이용하여 7명의 사용자들에게 문제를 추천해주었을 때 받은 평가 중 고려해볼 만한 평가는 다음과 같다. “풀어보지 못한 문제와 시도했다가 실패했던 문제를 추천해줘서 좋았다.”, “처음 풀어보는 유형의 문제가 추천되지 않아 좋았다.”, “문제의 수가 조금 더 많으면 좋을 것 같다.”. 첫번째 평가는 해결한 문제들에 대해서 알고리즘이 작동하기 때문에 어찌보면 당연한 결과이다. 여기서 더 나아가서 알 수 있는 것은 다양한 사용자들이 공통적으로 해결한 문제들을 추천 받는 다는 것을 의미한다. 두번째 평가는 사용자와 비슷한 사용자가 문제를 추천하는 데에 기여하는 정도가 크므로 사용자가 많이 해결해본 문제들의 유형을 추천하기 때문이다. 세번째 평가의 경우 고려할 수 있는 것으로 추천하는 문제의 수를 임의의 수로 입력을 받게 프로그램을 수정하면 될 것이다.

실행시간의 대부분을 차지하는 것은 crawling이다. 위 프로그램을 개선시킬 수 있는 방안으로 일정한 주기로 사용자들의 데이터를 미리 crawling 하는 것이다. 혹은 처음 crawling 한 데이터를 일정한 주기로 업데이트 하며 사용하는 방식을 사용하면 좋을 것이다.

**참고문헌 및 사이트**

“Baekjoon Online Judge”. Baekjoon Online Judge, 2022년 10월 24일 접속, [https://www.acmicpc.net](https://www.acmicpc.net/)

“Baekjoon Online Judge - 알고리즘 - 나무위키”. 나무위키, 2022년 10월 16일 수정, 2022년 10월 24일 접속, [https://namu.wiki/w/Baekjoon Online Judge](https://namu.wiki/w/Baekjoon%20Online%20Judge)

“[Solved.ac](http://solved.ac/)”. solved.ac, 2022년 10월 25일 접속, <https://solved.ac/>

“@solvedac/unofficial-documentation”, solvedac github, 2022년 2월 수정, 2022년 10월 18일 접속, <https://solvedac.github.io/unofficial-documentation/#/>

“Tree Edit Distance (and Levenshtein Distance)”, youtube, 2021년 1월 27일 수정, 2022년 10월 18일 접속, <https://www.youtube.com/watch?v=6Ur8B35xCj8>

“Tree edit distance in Python — Zhang-Shasha v1.1”, pythonhosted, 2013년 수정, 2022년 10월 21일 접속, <https://pythonhosted.org/zss/#examples>

Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal of Computing, 18:1245–1262, 1989.

1. https://github.com/Airsleep/AJOU222\_ITIP1\_Project [↑](#footnote-ref-1)