제한된 프로그램 소스 집합에서 표절 탐색을 위한 적응적 알고리즘

지정훈° 우균 조환규
부산대학교 컴퓨터공학과
{jhji°, woogyun, hgcho}@pusan.ac.kr

An Adaptive Algorithm for Plagiarism Detection in a Controlled Program Source Set

Junghoon Ji° Gyun. Woo, Hwangyu Cho
Dept of Computer Engineering, Pusan National University

요 약

본 논문에서는 대학생들의 프로그래밍 과제물이나 프로그래밍 경진대회에 제출된 프로그램과 같이 동일한 기능을 요구받는 프로그램 소스 집합들에서 표절 행위가 있었는지를 탐색하는 새로운 알고리즘을 제시한다. 본 논문에서는 프로그램의 소스 집합에서 추출된 키워드들의 빈도수에 기반한 로그 확률값을 가중치로 하는 적응적(adaptive) 유사도 행렬을 만들어 이를 기반으로 주어진 프로그램의 유사구간을 탐색하는 지역정렬(local alignment) 방법을 소개한다. 우리는 10여개 이상의 프로그래밍 대회에 제출된 실제 프로그램으로 본 방법론을 실험하였다. 실험결과 이 방법은 이전의 고정적 유사도 행렬(일치 +1, 불일치 -1, 갭(gap)을 이용한 일치 -2)에 의한 유사구간 탐색에 비하여 여러 장점이 있음을 알 수 있었으며, 보다 다양한 표절탐색 목적으로 제시한 적응적 유사도 행렬이 응용될 수 있음을 알 수 있었다.

1. 서 론

문서에 기초한 창작물의 표절(plagiarism)은 갈수록 사회문제화 되고 있다. 특히 표절을 검출하는 문제는 프로그래밍 분야에서 더욱 심각하게 대두된다. 자연어로 작성된 과제들은 강사가 채점을 위하여 읽어보는 단계에서 적절한 수준으로 적발할 수 있지만 프로그램 소스와 같이 그 언어가 자연언어와 매우 다를 경우에는 표절 탐색이 더욱 어렵다. 특히 수강생의 수가 많은 원격강의나 e-Learning을 이용한 강의에서는 정확하면서도 자동화된 표절검사 시스템이 필수적으로 요구된다[1].

일반 문서 표절연구에서와 마찬가지로 프로그램 표절 연구에는 몇 가지 문제가 있다. 먼저 두 프로그램 중 어떤 하나가 다른 하나를 표절하였는지를 판별하는 객관적인 기준이 없다. 이 문제는 확률 문제로 귀결될 수밖에 없다. 즉 어떤 길이의 프로그램 A가 다른 프로그램 B와 표절없이 거의 유사할 경우에 대한 확률모델을 제시하고 그에 근거하여 두 프로그램의 표절에 대한 가성을 밝히는 것이 최선의 방책이라고 할 것이다.

표절 연구의 또 다른 어려움은 표절에 대한 구체적인 정량적 평가를 하는 것이다. 다시 말해서 프로그램 A가 다른 프로그램 B를 표절했다고 하면 어떤 부분을 어떻게 표절했는지 밝혀야 한다. 단순한 구문수준(syntax level)에서의 표절은 쉽게 판별할 수 있지만 내부 자료구조의 표절 알고리즘의 표절을 정량화하여 밝혀내기란 매우 어려운 작업이다. 따라서 본 논문에서는 두 프로그램의 유사도를 일단 구문수준에서 시작하여 좀 더 확장된 형태로 고려할 수 있는 연구에 집중하고자 한다.

표절연구에서의 마지막 어려움은 실험자료로 쓰일 실제적인 표절 자료를 확보하는 것이다. 개발된 시스템이 잘 동작하는지를 증명하기 위해서는 실제적인 상황에서 제대로 동작함을 보여야 하는데, 표절행위는 일반적으로 불법적인 행위로 인식되기 때문에 정상적인 방법으로 표절 자료를 구하기란 매우 힘들다. 한편 인위적으로 표절한 프로그램을 양산할 수도 있지만 이는 실제상황에서 만들어진 것이 아니므로 정확한 의미에서 표절데이타로 보기는 힘들다.

본 연구에는 위에서 제시한 여러 문제에도 불구하고 현재까지 개발된 기존의 표절 탐색보다는 진일보된 새로운 방법을 제시하고 그 방법이 보다 타당함을 다양한 데이터와 실험의 특성값을 비교함으로 보이고자 한다. 본 논문에서는 특정 프로그램 집합으로 제한된 환경에서의 표절 검사에 초점을 맞추고 있다. 여기서 제한된 환경에서의 프로그램들이란, 입력과 출력, 형식, 프로그램의 기능이 매우 엄격하게 정의된 프로그래밍 과제물이라든지, 프로그래밍 경진대회에서 제출된 문제들에 대한 프로그램을 말한다. 본 논문에서는 이러한 제한된 환경에서의

프로그램 집합에 대해서 본 논문에서 제시하는 방법의 우수성을 보이고자 한다.

2. 관련연구

국내에서 관련 연구는 주로 문서들의 군집화(clustering)나 정보 조회(information retrieval)에 관한 것이었다. 표절에 관한 연구는 최근의 일인데 그 중 대표적인 것은 생물학에서 사용되는 유전자 정렬기법을 원용하여 프로그램의 유사도를 비교한 논문이 있다[2]. 그리고 실제 국내외에서 연구된 표절탐색 방법을 조사한 자료조사 논문이 있다[3].

지금까지 알려진 프로그램 소스의 표절검사 방법은 크게 특정한 키워드가 프로그램에 나타난 횟수를 주요 특성변수로 파악하여 이들의 형태를 비교하는 특성변수 계산법(attribute counting)과 프로그램의 파스 트리(parse tree)와 같이 프로그램의 특정한 구조를 분석하여 그것을 비교하는 구조기반 비교방법으로 나눌 수 있다. 특성변수 계산법은 일반적인 문서의 유사도나 군집화에 보편적으로 쓰이는 방식으로 각 문서의 주요 키워드의 빈도수를 정규화한 백터를 만들어 이들 간의 상관계수를 계산하여 그로부터 두 문서의 유사도를 추출하는 방식이다.

구조기반 비교에 의한 프로그램 소스 비교는 크게 3단계로 구성되어있다. 1단계는 프로그램 소스에서 주요 객체(keywords)등을 추출하여 비교가 용이하도록 새로운 객체를 만드는 작업이다. 이 작업에는 프로그램 소스 전체를 하나의 문자 스트링으로 보고 특별한 작업 없이 그대로 이용하는 방법[4]과 전체 프로그램을 프로그램 parsing tree로 재배열하여 그렇게 배열된 트리의 순회법(traversal)을 이용하여 어떤 선형의 순서를 만들어 내는 방법, 그리고 프로그램의 구문단계에서 대략적인 수행을 하여 그 수행되는 함수들의 순서에 따라 주요 키워드를 추출하여 새롭게 정렬하는 방법이 있다[2].

프로그램 소스 상에서 각 함수들의 위치는 큰 의미를 가지지 못하므로 이들을 하나의 스트링으로 보고 기존의 스트링을 비교하는 알고리즘을 사용하는 것은 표절 공격에 취약할 수 있다. 물론 이런 공격에 대비하기 위하여 일치되는 부분 블록(substring)들을 차례대로 지워나가는 Greedy-String-Tiling 방법이 흔히 사용되고 있다. 지금까지 알려진 가장 안정적인 도구인 JPlag와 YAP계열 모두 Greedy-String-Tiling 방법에 기초하고 있다[5,6].

3. 적응적 유사도 행렬

3.1 프로그램 선형화

이 절에서는 지역정렬 기법에 의해 두 프로그램의 유사도를 적응적으로 측정하기 위해 필요한 이론적 모델을 기술한다. 지역정렬 기법을 통해 두 프로그램의 유사도를 비교할 때에는 프로그램 선형화, 지역정렬, 유사도 산출 등의 과정을 거친다(그림 1참고).

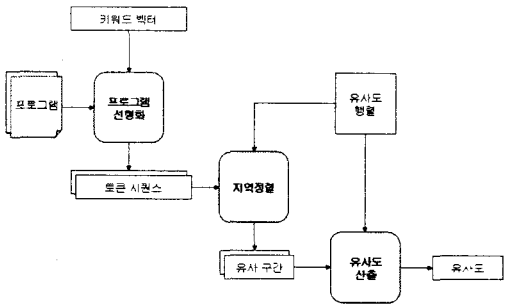


그림 1 지역 정렬 알고리즘에 의한 유사도 산출 과정

그림 1에서 첫 번째 단계인 토큰 추출은 두 프로그램에서 원하는 토큰을 추출하는 단계이다. 이 단계를 위해서는 우리가 살펴보고자 하는 토큰 집합을 정해야 한다. 토큰 집합은 통상적으로 해당 프로그래밍 언어의 키워드 집합이 된다. 지역정렬 알고리즘에서 사용할 키워드 백터를 K라고 하자. 지역정렬에서 고려할 키워드 개수를 r이라고 하면 키워드 백터 $K = \langle k_1, k_2, \ldots, k_r \rangle$로 정의할 수 있다.

프로그램 선형화에 사용되는 키워드 백터 K는 프로그래밍 언어에 종속적이다. 이 논문에서 구현한 유사도 검출 프로그램은 C/C++를 대상으로 하고 있다. 현재 구현된 유사도 검출 프로그램에서는 C/C++ 언어 키워드 중 81개의 키워드를 선정하여 프로그램 선형화에 이용하고 있다(r = 81).

3.2 적응적 유사도 행렬

그림 1의 두 번째 단계인 지역 정렬 단계에서는 유사도 행렬에 따라 두 토큰 스트림을 비교한다. 이 때 유사도 행렬이 사용되는데 키워드의 개수가 r일 때 유사도 행렬 M은 $(r + 1) \times (r + 1)$ 행렬이다. 행 수 및 열 수가 키워드 개수보다 하나 많은 이유는 갭(gap)을 나타내는 특수기호 k_{r+1} = '_'이 포함되었기 때문이다. 갭 기호 '_'는 두 서열의 길이를 맞추어 주기 위해 넣어주는 특별

한 기호이다.

기존의 지역정렬 알고리즘에서는 유사도 행렬의 점수를 정수로 나타낸다. 키워드 k_i, k_j가 일치할 경우에는 $M(k_i,k_j)$는 1, 일치하지 않으면 -1을 부여하고 있다. 갭을 이용하여 일치시켜야 하는 경우에는 -2점을 부여한다. 이렇게 정수 단위로 유사도 행렬을 정의하는 방법은 기존 생물학적 DNA 서열에 대해 유사도를 산출할 때 사용하던 방법이다.

이 절에서 설명할 알고리즘은 특정 프로그램 집합에 따라 유사도 행렬을 적응적으로 변경한다. 같은 목적으로 작성된 프로그램 그룹에 대해서 키워드 빈도는 매우 다르게 나타날 수 있는데, 적응적 지역정렬 알고리즘에서는 키워드 빈도를 바탕으로 유사도 행렬을 결정한다. 적응적 지역정렬에서는 빈도가 높은 키워드의 일치에 대해서는 낮은 점수를 부여하고, 빈도가 낮은 키워드의 일치에 대해서는 높은 점수를 부여한다.

적응적 유사도 행렬을 결정하기 위해서는 먼저 키워드 빈도를 산출해야 한다. n개의 프로그램으로 구성된 프로그램 그룹 $\mathbb{P}=\{p_1,p_2,...,p_n\}$가 있다고 할 때, 키워드 빈도 백터 $f^{\mathbb{P}}=\langle f_1^{\mathbb{P}},f_{2,...,}^{\mathbb{P}}f_r^{\mathbb{P}}\rangle$ 는 각 키워드 k_i가 해당 프로그램 그룹 내의 모든 프로그램에서 사용된 회수로부터 결정된다. 키워드 k_i가 프로그램 그룹 $\mathbb{P}$에서 사용된 횟수를 $occur(\mathbb{P}, k_i)$라고 하면, k_i가 사용된 빈도 $f_1^{\mathbb{P}}$는 다음과 같이 정의한다.

$$f_i^{\mathbb{P}}=occur(\mathbb{P},k_i)/\sum_{i=1}^{r}occur(\mathbb{P},k_i)$$

정의에 따르면 임의의 키워드 k_i의 빈도 $f_i^{\mathbb{P}}$는 0이상 1이하의 값($0 \le f_1^{\mathbb{P}} \le 1$)이며, 키워드 백터의 모든 원소의 합은 1이다($\sum_{i=1}^{r} f_1^{\mathbb{P}}=1$).

유사도 행렬은 키워드 빈도에 따라 적응적으로 정한다. 본 논문에서는 두 키워드의 빈도 곱의 로그 값으로 유사도 점수를 정했는데, 어떤 신호값(signal)의 세기(intensity)를 상대적으로 비교하기 위하여 그 값에 log를 취한 값(log odd)을 취하는 것은 공학에서 흔히 이용하는 방법이다. 본 유사도 검출의 경우에는 상대적인 비율을 그대로 이용하는 것보다 log를 취한 값을 사용하는 것이 바람직하다고 판단하였다. 왜냐하면 키워드 빈도의 차이가 크므로 키워드의 상대적인 비율을 그대로 매칭 가중치에 반영하는 것은 바람직하지 못하기 때문이다.

이상에서 설명한 것과 같이 유사도 행렬의 값은 키워드 곱의 로그 값으로 정한다. 구체적으로 말해서, 키워드 k_i와 k_j가 일치할 때의 점수는 음의 로그로 유사도 점수를 나타내고, 불일치일 경우에는 양의 로그로 벌점을 나타낸다. 키워드 빈도는 0과 1사이의 수치이므로 결과적으로 일치할 경우에는 양의 값을, 불일치일 경우에는 음의 값을 갖게 된다. 프로그램 그룹 $\mathbb{P}$에 대한 유사도 행렬 $M^{\mathbb{P}}$를 정의하면 다음과 같다.

$$M^{\mathbb{P}}(k_i,k_j)=\begin{cases}-\alpha\log_2 f_i^{\mathbb{P}} f_j^{\mathbb{P}} & k_i=k_j\\ \beta\log_2 f_i^{\mathbb{P}} f_j^{\mathbb{P}} & k_i\neq k_j\\ 4\beta\log_2 f^{\mathbb{P}} & k_i,k_j \text{ 중 하나가 '_'이고 다른 것의 빈도가 } f^{\mathbb{P}} \text{일 때}\end{cases}$$

위와 같이 유사도 행렬을 결정하면 일치와 불일치에 대한 상대적 가중치를 매개변수 α와 β로 조정할 수 있다는 장점이 있다.

3.3 프로그램간 유사도 계산

두 프로그램의 토큰 시퀀스에 대해 유사한 구간(aligned subrange)이 발견되면, 위에서 정의한 유사도 행렬을 이용하여 유사도 점수를 산출한다. 두 프로그램 A와 B를 정렬하여 유사구간을 산출하는 함수를 $align$이라고 하고 이 함수가 유사구간 쌍$a_1,a_2,...,a_m,b_1,b_2,...,b_n$에 대해 다음과 같이 키워드 쌍의 백터 형태로 유사구간을 산출한다고 하자.

$$align(A,B)=\langle(a_1,b_1),(a_2,b_2),...,(a_m,b_n)\rangle$$

이 때, 이 구간의 유사도 점수는 다음 함수로 정의된다.

$$SIM_{abs}(A,B)=\sum_{(a.b)\in align(A,B)} M^{\mathbb{P}}(a,b)$$

이는 정수 단위의 유사도 행렬을 이용하는 기존 방법에서의 유사도 산출 방법과 거의 동일하다. 기존 방법에서의 프로그램 유사도 점수 산출에 대해서는 [7]를 참고하기 바란다.

유사도 점수 SIM_{abs}를 0~100% 구간으로 정규화한 것을 유사도라고 한다. 정규화된 유사도를 산출하는 방식은 두 가지가 있다. 일반적으로 사용되는 방식은 두 프로그램의 유사도 점수 두 배를 각 프로그램에 대하여 자신과 정렬시킨 후 유사도 점수의 합으로 나누는 방식이다. 이 방식의 유사도 $SIM_{sum}(A,B)$의 유사도 산출은 다음과 같다.

$$SIM_{sum}(A,B)=\frac{2SIM_{abs}(A,B)}{SIM_{abs}(A,A)+SIM_{abs}(B,B)}$$

유사도 점수를 정규화하는 다른 방법으로서 $SIM_{abs}(A,A)$와 $SIM_{abs}(B,B)$중 작은 값으로 나누는 방

식이 있다. 이 방식은 두 프로그램의 길이 차이로 인한 유사도($SIM_{sum}(A,B)$)가 낮아지는 것을 보완한다. 작은 점수를 기준으로 정규화한 유사도를 본 논문에서는 $SIM_{\min}(A,B)$로 정의한다.

$$SIM_{\min}(A,B)=\frac{SIM_{abs}(A,B)}{\min\{SIM_{abs}(A,A),SIM_{abs}(B,B)\}}$$

다음 절에서는 이상에서 기술한 적응적 유사도와 기존 유사도의 분포 차이를 살펴본다. 특히 적응적 유사도에 대해서는 매개변수 α값과 β값을 변경시킴으로써 어떻게 분포가 변화하는지 살펴본다.

4. 실험 및 평가

적응적 지역정렬 알고리즘의 효과를 실험해보기 위해, 간단한 실험을 수행하였다. 고정적 유사도 행렬(fixed similarity matrix)을 채택하였을 경우와 적응적 유사도 행렬(adaptive similarity matrix)을 채택하였을 경우에 대하여 지역정렬 알고리즘을 수행하였다. 각 경우에 대하여 유사도 점수 분포와 유사도가 높은 구간을 비교하여 살펴보았다.

4.1 테스트 데이터

테스트 데이터 프로그램으로는 ACM 국제 대학생 프로그래밍 경진대회(ICPC : International Collegiate Programming Contest) 2005년도 아시아 지역 예선 문제에 대한 출전자가 제출된 프로그램을 사용하였다.

순번	프로그램 그룹	파일 개수	순서쌍 수	소스코드 라인 수			
				최대	최소	평균	표준편차
1	ICPC05 1	153	11,628	144	21	44.46	15.85
2	ICPC05 2	109	5,886	139	24	65.44	22.86
3	ICPC05 3	38	703	88	28	61.47	29.27
4	ICPC05 4	44	946	91	23	45.25	14.28

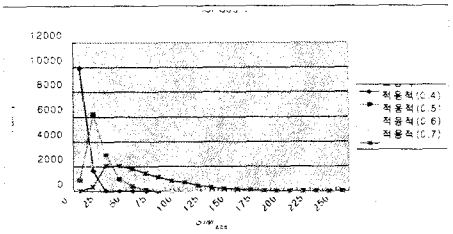
표 1 : 표절의심 코드를 포함하는 ICPC05 데이터 셋

과거 발표된 연구결과에서는 인위적인 표절을 수행한 프로그램에 대해 표절 검사를 수행하였다. 그러나 이러한 인위적인 표절은 실제 현장에서 행하여지고 있는 표절과는 차이가 있다. 따라서 우리는 실제 프로그래밍 경진대회나 실제 과제 제출 프로그램을 대상으로 하여 표절을 검사하였으며, 그 결과 표절로 의심되는 프로그램을 찾을 수 있었다. 표1은 표절로 의심되는 프로그램 쌍이 발견된 ICPC 2005년도 예선 프로그램을 정리한 것이다.

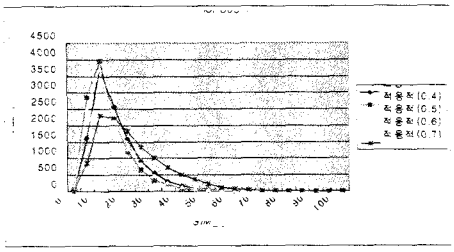
4.2 프로그램간 유사도 분포

유사도 분포는 ICPC 2005 프로그램 그룹 중에서도 가장 순서쌍 수가 많은 프로그램 그룹 1에 대하여 조사하였다. 고정적 유사도 행렬을 채택하였을 경우와 적응적 유사도 행렬을 채택하여 α를 0.4에서 0.7까지 변화시켜 가며 유사도 분포를 조사하였다.

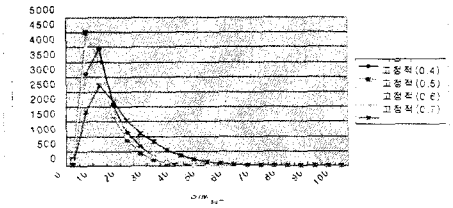
실험결과 고정적 유사도 행렬을 채택하였을 경우와 적응적 유사도 행렬을 채택하였을 경우에 유사도 절대값 SIM_{abs} 분포는 상당히 다른 형태를 나타났다. 그 이유는 SIM_{abs}가 절대적 수치 값을 반환하기 때문이다. 정규화된 유사도를 함께 비교하기 위해 SIM_{abs}와 $SIM_{\min}$, SIM_{sum} 분포를 그래프로 나타내면 그림 2와 같다.



(a) SIM_{abs}



(b) $SIM_{\min}$



(c) SIM_{sum}

그림 2: ICPC05 1 그룹에 대한 유사도 분포.
(a) SIM_{abs}, (b) $SIM_{\min}$, (c) SIM_{sum}

4.3 표절 의심 코드의 비교

유사도 분포 실험에서는 구체적으로 유사한 구간을 확인할 수 없었는데, 여기서는 구체적인 유사 구간의 차이를 살펴본다. 구체적인 유사구간은 각 토큰별 유사도 값에 따라서 결정되므로 유사도 행렬 값에 따라 달라질 수 있다. 유사도 구간은 표절로 의심되는 코드에 대해서만 살펴보았다.

그림 3 및 그림 4에 제시된 두 프로그램은 2004년도 국제 대학생 프로그래밍 대회(ICPC 2004) 예선에서 제출된 프로그램 중에서 표절로 의심되는 프로그램 쌍이다. ICPC 예선은 인터넷을 이용하여 시험을 치른다. 인터넷 예선은 각 대학별 지도교수(ICPC 코치)의 감독 하에 진행되거나 코치가 없는 경우 자율적으로 진행된다. 따라서 참가자가 마음만 먹으면 얼마든지 표절과 같이 부정행위가 가능하다.

설명의 편의를 위해 프로그램 P를 라인별로 나열했을 때, i번째 라인에서 j번째 라인(단, $i < j$) 사이에 있는 코드를 $P[i,j]$로 표시하기로 하자. 그림 3, 4에 나타낸 두 프로그램 P_a, P_b을 고정적 유사도 행렬로 비교했을 때 일치구간은 P_a[1,19]와 P_b[1,20]으로 나왔다. 그런데 적응적 유사도 행렬로 비교했을 때에는 더 넓은 구간인 P_a[1,28] 구간과 P_b[1,37]과 일치한다고 보고가 되었다.

본 연구자들이 두 프로그램을 표절로 보는 이유는 변수 n과 t의 의미 때문이다. 이 문제의 테스트 데이터는 입력 파일에 모두 t개가 있으며 각각의 케이스마다 세부 데이터가 n개가 있다고 문제에서 설명했기 때문에 적어도 문제에서 제시된 변수를 바로 사용한다면 외부에 t번의 루프(loop)가 있어야 하고, 각 케이스마다 n번의 scanf가 있어야 한다. 그런데 P_b에서는 이 의미가 거꾸로 사용되고 있다. 그리고 P_a의 19번째 라인에서만 for 루프에서 j의 제한 조건이 P_b에서는 그 역순(j--)로 표시되었다. 다른 for 루프는 모두 일반적인 형식인 for(X = 0; $X < n$; X++)으로 사용했음에도 불구하고 실제 그 순서에 아무런 의미가 없는 j에 대한 for 루프만 다르게 표시했다는 것은 표절의 의심을 사기에 충분하다.

우리는 본 논문에서 제시한 적응적 방법이 이 차이를 극복해 주었다는 것을 지적하고자 한다. 즉 for 루프 조건을 바꾸는 구간이 다소 길긴 하지만 적응적 방법에서는 그 불일치(mismatch)값이 고정적인 불일치 값인 -1보다 작으므로 그 뒤의 매칭을 포함시킨 부분 정렬을 구성하였다. 통상 사용하지 않는 코드를 강제로 넣어서 표절을 하는 방법의 경우에 보통 표절자는 채점자의 눈에 크게 띄지 않는 코드를 넣게 된다. 왜냐하면 아주 특별한 기능이나 특이한 구문의 문장을 넣을 경우 바로 눈에 띄기 때문에 일반적인 지정문(assignment)나 if-then-else 등을 넣게 된다. 이 경우 고정적 방법에 비해서 그 구간의 매칭 값이 더 작아지기 때문에 적응적 방법에서

```
#include <stdio.h>
int main ()
{
    int t, n;
    int i, j;
    int p[500];
    int ans[500],max;
    scanf("%d", &t);

    while (t--)
    {
        scanf("%d", &n);
        for (i=0; i<n; i++)
        {
            scanf("%d", &p[i]);
            ans[i] = 0;
        }
        for (i=0; i<n; i++)
            for (j=0; j<i; j++)
                if (p[i] >= p[j] && ans[i] < ans[j] + 1)
                    ans[i] = ans[j] + 1;
        max = 0;
        for (i=0; i<n; i++)
            if (ans[i] > max) max = ans[i];
        printf("%d n", max + 1);
    }
    return 0;
}
```

그림 3 ICPC05에서 표절로 의심되는 프로그램 P_a

```
#include<stdio.h>
int main()
{
    int i, j;
    int data[600], cnt[600];
    int n, t;
    int max;
    scanf("%d", &n);
    while(n--)
    {
        scanf("%d", &t);
        for( i=0 ; i<t ; i++ )
        {
            scanf("%d", &data[i]);
            cnt[i] = 0;
        }

        for( i=0 ; i<t ; i++ )
        {
            for( j=i ; j>=0 ; j-- )
            {
                if( data[i] >= data[j] && cnt[j]+1 > cnt[i] )
                {
                    cnt[i] = cnt[j]+1;
                }
            }
        }
        max = 0;
        for( i=0 ; i<t ; i++ )
        {
            if( max < cnt[i] )
                max = cnt[i];
        }
        printf("%d n", max);
    }
    return 0;
}
```

그림 4 P_a의 표절로 의심되는 프로그램 P_b

는 그 구간 밖에 다른 매칭 구간이 있을 경우 그것을 포함하여 표절구간으로 계산하게 된다. 위의 예는 적응적 유사도 행렬이 고정적 유사도 행렬보다 나은 점을 잘 보여주는 경우라고 할 수 있다.

적응적 방식에서는 프로그램을 표절할 때 가장 일반적인 변수 치환, 문장 늘이기 등의 방법을 사용하는 경우 그 상대적 빈도가 높기 때문에 이들의 삽입으로 인한 지역정렬에서의 음수값(벌점)은 상대적으로 작아진다. 따라서 이러한 가장 일반적인 표절 공격에 대해서 적응적 방식은 고정적 방식에 비해서 훨씬 유리함을 알 수 있다.

5. 결론

본 논문에서는 키워드 빈도를 특성 변수로 이용하여 적응적 유사도를 산출하는 방법을 제안하였다. 이를 위해, 특정 목적으로 작성된 프로그램 그룹에 대하여 키워드 빈도를 산출하였고, 빈도의 로그를 취하여 적응적 유사도 행렬을 정의하였다. 결과로 얻어진 유사도 행렬을 바탕으로 세 가지 유사도 SIM_{abs}, $SIM_{\min}$, SIM_{sum}를 정의하였다.

실제 프로그래밍 경시대회 소스코드를 이용한 실험을 통하여 적응적 유사도 행렬에 의한 표절 탐색 방법이 일치/불일치로만 계산하는 이전의 방법에 비하여 상당히 효과적이라는 것을 알 수 있었다.

참고문헌

[1] B. Cheang, A. Kurnia, A. Lim and W. Oon. On automated grading of programming assignments in an academic institution. *Computers and Education*, 41:121-131, 2003.

[2] 조동욱, 소정, 김진용, 최병갑, 김선영, 김지영. 프로그램 표절감정 툴에 대한 비교, 분석 및 개발 툴에 대한 방향제시. In *제20회 한국정보처리학회 추계학술발표대회 논문집*, volume 10, pages 757-760, 2003

[3] 이효섭, 도경구. 프로그램 표절 검출 방법에 대한 조사. In *한국정보과학회 한국컴퓨터종합학술대회 2005 논문집(B)*, volume 32, pages 916-918, 2005.

[4] Kristina L, Verco and Michael J. Wise. Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems. In *Proceedings of the 1st Austrian Conference on Computer Science Education*, pages 130-134, Sydney, Australia, July 1996

[5] Michael J. Wise. Neweyes: A system for comparing biological sequence using the running karp-rabin greedy string-tiling algorithm. In Proceedings of the 3rd International Conference on Intelligent Systems for Modular Biology, pages 393-401, 1995

[6] Lutz Prechelt, Guido Malphol, and Michael Philipsen. Finding plagiarisms among a set of programs with JPlag. Journal of Universal Computer Science, 8(11):1016-1038, 2002

[7] 강은미, 황미녕, 조환규. 유전체 서열의 정렬기법을 이용한 소스코드 표절검사. 정보과학회논문지: 컴퓨팅의 실제, 9(3):352-367, June 2003