고급계산이론 HW1 Report

공과대학 컴퓨터공학부 석사과정
2017-20870 오평석

0. 컴파일 방법

auto 키워드를 사용하였으므로 c++11 플래그를 주고 컴파일하여야 합니다. Makefile을 같이 넣어두었으니 make를 입력하여도 됩니다.

1. Baker-Bird 알고리즘 구현

2D string pattern matching 알고리즘인 baker-bird 알고리즘을 구현하기 위해서는, KMP 알고리즘과 Aho-Corasick 알고리즘을 구현하여야 합니다. 따라서 이를 따로 구현한 다음 class로 선언해 두었습니다. main 함수에서는 KMP와 Aho-Corasick을 이용하여 Baker-Bird 알고리즘을 수행합니다. 또한 강의 노트에 있는 방법을 사용하여 Baker-Bird 알고리즘의 extra space를 $O(|\sum|m^2 + n)$ 으로 줄였습니다. 이는 R값을 저장하는 r 배열이 1차원으로만 선언되어있는 것을 통해 확인할 수 있습니다.

(1) KMP

Text를 입력받아 pattern을 find하는 대신, text에 한 글자를 push한 다음 제일 마지막 m개의 글자에 대해서만 KMP를 수행하는 push 함수를 구현하였습니다. 이는 KMP 함수에서 제일 마지막 iteration만 가져오면 됩니다.

(2) Aho-Corasick

입력으로 주어지는 글자가 모두 알파벳 소문자이므로, 트라이를 구성할 때 각 노드마다 26개의 배열을 가지도록 구현하였습니다. 이 값은 상수이므로 공간복잡도에 영향을 주지 않습니다. 그리고 root로부터 bfs를 돌면서 failure function과 output function을 생성합니다. 이 때 output function에 pattern의 index를 넣어 location과 어떤 pattern과 매칭이 일어났는지 기록할 수 있도록 하였습니다. 이는 일반적인 Aho-Corasick 알고리즘에서는 성립하지 않으나 (substring이 존재하는 경우 matching의 가능성이 여러 개이므로), Baker-Bird 알고리즘에서는 Aho-Corasick에 들어가는 pattern들이 모두 같은 길이를 가지는 문자열이므로 성립합니다.

2. Checker 프로그램 구현

Checker 프로그램은 naïve한 2d pattern matching 알고리즘을 구현하여 작성하였습니다. Input을 입력받아 따로 답을 구한 다음, 이를 입력받은 답과 비교하여 yes 또는 no를 출력합니다.

3. Example running 결과

두 개의 example을 작성하여 프로그램을 검증하였습니다. 이 파일들은 sample/ps1_in.txt 와 sample/ps2_in.txt 에서 확인할 수 있습니다. 전자는 pattern을 정해두고 랜덤한 string과 함께 섞어 넣은 것이며, 후자는 matching이 자주 일어나는 극단적인 경우를 잘 해결하는지 테스트합니다. 둘 다 checker를 무사히 통과하는 것을 확인할 수 있었습니다.

```
[pyeongseok@login0 hw1]$ ./hw1 sample/ps1_in.txt ps1_out.txt
[pyeongseok@login0 hw1]$ ./hw1 sample/ps2_in.txt ps2_out.txt
[pyeongseok@login0 hw1]$ ./checker sample/ps1_in.txt ps1_out.txt ps1_check.txt
[pyeongseok@login0 hw1]$ ./checker sample/ps2_in.txt ps2_out.txt ps2_check.txt
[pyeongseok@login0 hw1]$ cat *_check.txt
yes
yes
```