2008 Spring: Programming Methodologies

4th Project Description

Suffix Tree Full Text Indexing and Searching

1. 목표

Suffix tree를 이용한 풀 텍스트 (full text) 검색엔진을 구현한다. 풀 텍스트 검색이란 문서에 포함된 모든 부분 문자열에 대한 검색을 뜻한다. 본 프로젝트는 풀 텍스트 검색이 가능하도록 문서를 색인하고 검색하는 기능을 가진 프로그램 작성을 목표로 한다.

2. 설명

1) Suffix Tree

Suffix Tree란 문자열의 모든 Suffix를 색인한 트리 구조를 말한다. Suffix의 개념부터 살펴보자.

문자열
String
Suffix
g, ng, ing, ring, tring, String

위와 같이 문자열이 주어졌을 때 모든 위치에서 문자열의 끝까지 해당하는 모든 가능한 부분문자열을 주어진 문자열의 suffix라고 말한다. 그렇다면 이를 이용하여 어떻게 suffix tree를 구성하는지 알아보자. (첨부 PPT의 1페이지 애니메이션 참조)

2) 검색

검색이란 찾고자 하는 문자열(쿼리)가 나타나는 모든 문서를 돌려주는 기능을 말한다. 예제와 같이 “String”이란 문자열을 포함하는 문서가 있다면 “ring”으로 검색했을 때 그 문서를 대답해야 한다.

2) 색인

본 프로젝트에서는 모든 문서에 대해 suffix tree를 만드는 과정을 색인이라 한다.

3. 구현 내용

1) 색인 기능

문서를 읽어 단어 단위로 suffix tree를 작성한다. 첨부 PPT파일의 2~3페이지를 참조.

각 노드는 자식노드를 가리키는 포인터와 그 suffix가 나타난 문서의 번호를 저장한 곳을 가리키는 포인터를 갖고 있어야 할 것이다. 그러나 만약 자식노드를 가리키는 포인터를 모두 갖고 있는다면 모든 알파벳에 대해 포인터를 갖고 있어야 하기 때문에 노드 하나에 26개의 포인터가 들어가야 할 것이다. 이처럼 구현을 한다면 메모리가 매우 많이 필요할 것이므로 효율적이지 않다. 따라서 슬라이드와 같이 자식 노드는 리스트를 작성해 항상 26개의 포인터를 저장하지 않고 동적으로 그 크기를 늘리는 방법을 사용해야 한다. 리스트를 작성하는 방법은 직접 구현하여도 되고 STL에 있는 클래스를 사용해도 좋다. 실습 수업시간에 배운 vector 템플릿을 사용하기를 권장한다. (STL의 list 템플릿을 사용하여도 좋다)

첨부 PPT의 2~3페이지 예와 같이 suffix를 suffix tree에 넣은 후에 해당 suffix가 어느 문서에서 나왔는지 그 리스트를 저장해야 한다. 앞의 프로젝트(Q-Gram search engine)에서 말한 역색인 리스트와 같은 것이다. 3페이지의 그림에서는 문서 1만 역색인 리스트에 들어가 있으나 만약 one이란 단어가 문서 4에서도 나왔다면 one, ne, e 의 서픽스 끝의 노드에 달린 역색인 리스트에 4도 추가를 해야 할 것이다.

본 프로젝트는 색인 결과를 파일에 저장하지 않아도 된다.

2) 검색 기능

쿼리로 들어온 문자열이 나타나는 모든 문서를 찾아 그 문서 번호를 출력한다. 쿼리로 들어온 문자열에 따라 노드를 따라간다. 만약 그 문자열로 끝나는 path가 존재하지 않는다면 쿼리가 나타나는 문서가 없는 것이다. 예를 들어 첨부 파일 2~3의 예에서 oni를 찾아보자 on까지의 path는 존재하지만 i로 가는 path가 없으므로 결과는 없다. 만약 쿼리 문자열로 끝나는 path가 존재한다면 그 하위 모든 path의 suffix가 나타나는 모든 문서가 쿼리의 답이 된다. 예를 들어 f를 찾아보자. f로 시작하는 path(f를 prefix로 갖는 suffix)는 ffix, fix 두 가지가 있다. 따라서 f의 결과는 위 두 path를 모두 따라가 역색인 리스트를 합집합한 것이 된다.

3) 입출력

실행파일은 색인을 한 후 검색을 실행하는 하나의 프로그램을 작성한다.

색인 실행파일에게 주어질 입력은 다음과 같다.

```
$ search inputfilelist.txt querylist.txt
```

텍스트형식 파일 inputfilelist.txt는 다음과 같이 한 라인에 색인 할 문서 파일명 하나를 갖는다.

```
1.txt
3.txt
10.txt
```

```
4.txt
9.txt
7.txt
```

확장자(.txt)를 제외한 파일명을 문서번호로 사용하며 위 예와 같이 파일명은 정수형 숫자이다.

텍스트형식의 파일 query.txt는 검색할 문자열을 한 라인에 하나씩 포함한다.

```
one
suffix
tree
test
shim
```

출력은 각 query를 포함하는 모든 문서번호를 다음과 같이 한 라인에 하나씩 출력한다.

```
one 1 4 9 10
suffix 2 5 9 11
tree 1 2 3 4
test 3 8 10 11 12
shim 5 6 7 8 9
```

실행 파일은 search로 생성하여야 한다.

4) Abstract 클래스

지난 Q-Gram 검색과 이번 suffix tree 검색은 몇 가지 제약조건 (Q-gram의 경우 3글자 이상)을 제외하면 동일한 인터페이스를 갖는다. 따라서 최 상위 클래스를 적절히 정의하면 q-gram이든 suffix tree이든 어떤 방식의 색인/검색 방법을 사용해도 동일한 상위 클래스를 상속받아 구현이 가능할 것이다. 본 프로젝트에서는 이와 같은 abstract 클래스를 정의하고 (적절한 멤버 변수와 멤버 함수, virtual 함수 등을 사용: search, index 함수 포함) 이를 상속받아 suffix tree를 이용해 검색을 수행하는 클래스를 작성하도록 한다. (클래스의 상속개념을 잘 이해하고 디자인을 해야 한다. 예를 들어 suffix tree를 구현한 클래스는 위에서 정의한 abstract 클래스를 상속받는 클래스가 아닐 것이다)

4. 기타

1) 영문 문서를 색인, 검색한다고 가정한다. (한글 등 multi-byte 제외)

2) 구현언어는 C++만 허용한다.

3) 보고서 제출은 월 일까지 제출하며 소스 제출은 6월 17일 23시59분까지 이메일로 제출한다. 이메일의 제목은 반드시 [PM2008] PRJ3 2000-00000과 같은 형식을 지켜 제출하도록 한다.