**Programming Assignment1**

**2022310315 이용하**

Main함수 코드는

텍스트이(가) 표시된 사진

자동 생성된 설명

문장의 길이와 패턴의 길이를 입력 받고, 문장과 패턴을 동적 그 길이에 맞게 동적할당 하였다(널문자를 위해 1을 더함). Failure function도 패턴의 길이로 동적할당 하였다. 난수 생성을 통해 1. 문장과 패턴이 모두 ‘a’인 경우 , 2. 문장과 패턴이 ‘a’ 또는 ‘b’가 무작위로 선택되는 경우로 나누어 데이터를 분석했다(위의 코드는 경우2에 해당). <time.h>에 있는 clock를 이용해 각각의 함수가 호출되고 나오는 시간(초)을 분석했다. 마지막으로 동적할당한 메모리들을 해제했다.

<Strstr 함수>

Strstr함수는 가장 원초적인 패턴 찾기 함수로, 문장과 패턴의 앞문자가 일치하면 패턴을 찾기 시작하고 패턴이 다르면 찾기 시작한 문자의 다음 번째로 다시 찾기 시작하는 함수이다. Big-O표기법에 의하면, O(문장의 길이\*패턴의 길이)이다. 아래와 같은 코드로 구현했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

<nfind함수>

nfind함수는 strstr함수보다 시간 복잡도 면에서 약간 개선된 함수로, 문장과 패턴의 뒷문자가 일치하면 패턴을 찾기 시작한다. 나머지는 strstr함수와 같다. 사실 strstr함수와 O(문장의 길이\*패턴의 길이)로 시간 복잡도가 같지만, 확률적으로 연산 횟수가 적다. 아래와 같은 코드로 구현했다.텍스트이(가) 표시된 사진

자동 생성된 설명

<KMP 알고리즘>

KMP 알고리즘은 패턴을 분석해 패턴의 패턴을 찾고 그것을 failure fuction으로 구현한다. 그 함수값을 이용하여 문장과 패턴을 다시 대조할 때, 문장을 이미 비교한 문자 앞에서부터 비교하는 일이 없도록 하는 알고리즘이다. 알고리즘과 failure function은 아래와 같은 코드로 구현했다.텍스트이(가) 표시된 사진

자동 생성된 설명

<데이터 측정> 결과값은 모두 초 단위임

1. 문장과 패턴이 모두 ‘a’인 경우
   1. 문장의 길이는 고정(1MB), 패턴의 길이가 바뀜

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 패턴의 길이 | 1바이트 | 100바이트 | 1KB | 100KB |
| strstr | **0.017811** | **0.211439** | **1.725764** | **161.893015** |
| nfind | **0.007554** | **0.172103** | **1.694328** | **154.438857** |
| KMP Algorithm | **0.005501** | **0.002591** | **0.002697** | **0.002621** |

* 1. 문장의 길이가 바뀜, 패턴의 길이는 고정(1KB)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 문장의 길이 | 1KB | 10KB | 100KB | 1MB | 10MB | 100MB |
| strstr | **0.004586** | **0.043542** | **0.207100** | **1.734966** | **17.134322** | **171.987782** |
| nfind | **0.000007** | **0.024474** | **0.16328** | **1.695691** | **17.186845** | **172.444818** |
| KMP Algorithm | **0.000005** | **0.000037** | **0.000270** | **0.002737** | **0.026578** | **0.268708** |

* 1. 문장의 길이와 패턴의 길이가 모두 바뀜

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 문장/패턴 | 100바이트/10바이트 | 1KB/100바이트 | 10KB/1KB | 100KB/10KB | 1MB/100KB | 1MB/10바이트 | 10MB/10바이트 | 100MB/10바이트 |
| strstr | **0.000024** | **0.001004** | **0.030386** | **1.600118** | **162.303333** | **0.047668** | **0.229472** | **2.105572** |
| nfind | **0.000012** | **0.000765** | **0.020889** | **1.484597** | **154.400997** | **0.027162** | **0.194098** | **1.962027** |
| KMP Algorithm | **0.000004** | **0.000019** | **0.000031** | **0.000256** | **0.002610** | **0.003125** | **0.025942** | **0.262764** |

1. 문장과 패턴이 ‘a’ 또는 ‘b’로 선택되는 경우

2-1. 문장의 길이는 고정되고(1MB), 패턴의 길이가 바뀜

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 패턴의 길이(일치 개수) | | 1바이트(**524231**개) | 2바이트(**262120**개) | 3바이트(**130934**개) | 5바이트(**32688개)** | 10바이트(998개) | 20바이트 이상(약 0개) |
| strstr | **0.017004** | **0.019996** | **0.020215** | **0.021321** | **0.020201** | **0.018354** |
| nfind | **0.011469** | **0.013308** | **0.013352** | **0.013645** | **0.017994** | **0.017448** |
| KMP Algorithm | **0.010364** | **0.011163** | **0.010690** | **0.010909** | **0.009857** | **0.009639** |

20바이트 이상은 패턴이 일치하는 횟수가 높은 확률로 0개라 의미가 없다고 판단.

2-2. 문장의 길이가 바뀌고, 패턴의 길이는 고정됨(3바이트)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 문장의 길이(일치 개수) | 1KB(**126개)** | 10KB(**1315개)** | 100KB(**12889개)** | 1MB(**130933개)** | 10MB(**1310484개**) | 100MB(**13110449개)** |
| strstr | **0.000037** | **0.000428** | **0.002684** | **0.018414** | **0.093234** | **0.925329** |
| nfind | **0.000029** | **0.000327** | **0.002316** | **0.013017** | **0.075187** | **0.753070** |
| KMP Algorithm | **0.000024** | **0.000314** | **0.002611** | **0.010071** | **0.071578** | **0.701932** |

<데이터 분석>

* 1. : strstr과 nfind는 패턴의 길이가 길어질수록, 연산 시간이 늘어난다. 이는 두 함수 모두 패턴의 패턴과 관계없이 항상 이미 비교했던 문장의 문자를 다시 비교하기 때문이다. 하지만, KMP 알고리즘은 이미 비교했던 문장의 문자를 비교하지 않기 때문에 연산 시간이 훨씬 적다. 예를 들어, 문장이 ‘aaaaaa’이고 패턴이 ‘aaa’인 경우, strstr과 nfind는 문장의 1번째 문자와 패턴의 1번째 문자부터 쭉 비교한다. 그 다음 문장의 2번째 문자부터 패턴의 1번째 문자부터 쭉 비교한다. 그런 방식으로 연산을 모두 수행하면, 둘 다 연산 횟수가 3\*4이다. 하지만, KMP 알고리즘은 문장의 1번째 문자와 패턴의 1번째 문자부터 쭉 비교하고 그 다음 문장의 4번째 문자와 패턴의 3번째 문자를 비교한다. 그런 방식으로 연산을 모두 수행하면, 연산 횟수가 6이다. 그러므로, KMP 알고리즘이 나머지 두 함수보다 월등히 빠른 것이다.

또, strstr과 nfind는 문장의 길이가 고정돼 있을 때, 패턴의 길이와 연산 시간이 거의 비례 관계임을 알 수 있다. O(문장의 길이\*패턴의 길이)기 때문에 거의 비례 관계가 성립될 수 있다. KMP알고리즘은 O(문장의 길이)기 때문에 패턴의 길이에 따른 연산 시간이 거의 차이가 없다.

1-2: strstr과 nfind는 O(문장의 길이\*패턴의 길이)기 때문에 패턴의 길이가 고정돼 있을 때, 문장의 길이와 연산 시간이 거의 비례관계임을 알 수 있다. KMP 알고리즘은 O(문장의 길이)기 때문에 문장의 길이가 길어지면서 연산 시간도 조금씩 증가하는 것을 알 수 있다.

1-3: 1-1과 1-2의 내용을 종합하여, Big-O를 연관하여 보면 당연한 결과임

2-1: strstr과 nfind, KMP 알고리즘 모두 거의 연산 시간의 차이가 없다. Strstr과 nfind에서 패턴의 길이가 늘어남에도 패턴과 문장이 일치하는 경우가 희박하기 때문에 내부 for문을 바로 통과하므로 연산 시간이 거의 증가하지 않는다. 아래 코드를 보면,텍스트이(가) 표시된 사진

자동 생성된 설명

외부 for문과 내부 for문이 있는데, 외부 for문은 문장 길이가 고정돼 있기 때문에 연산 횟수는 같다. 하지만, 내부 for문의 조건 중 ‘ s[i] == pat[j] ’이 있는데, 이것 때문에 일치하지 않는 경우 내부 for문을 나오게 된다. 그러므로, 일치하는 경우가 거의 없는 경우 strstr과 nfind는 O(문장의 길이)와 비슷하다.

KMP 알고리즘은 문장의 길이 말고는 일치 비율과 패턴의 길이가 거의 상관없기 때문에 문장의 길이가 같다면 연산 시간이 거의 변하지 않는 것을 알 수 있다.

Nfind는 strstr보다 미세하지만 항상 연산 시간이 짧다. 이 이유는, if문의 차이 때문이다. Nfind는 패턴의 끝 문자가 문장의 문자와 같은 경우를 보고, strstr은 패턴의 시작 문자가 문장의 문자와 같은 경우를 본다. 확률적으로 패턴의 끝 문자가 문장의 문자와 다른 경우가 더 많기 때문에 for문을 돌지 않으므로 nfind가 strstr보다 대게 빠르다.

2-2: 2-1과 다르게 문장의 길이가 증가함에 따라, strstr과 nfind, KMP 알고리즘이 모두 연산 시간이 증가한다. Strstr과 nfind의 코드를 보면, 외부 for문의 연산 횟수가 문장의 길이기 때문이다. Strstr과 nfind는 문장과 패턴이 일치하는 경우가 거의 없기 때문에 2-1에서 이끈 것처럼 O(문장의 길이)와 비슷해지고 그러므로 문장의 길이와 연산 시간이 거의 비례 관계가 나타난다.

1-2와는 다르게 문장의 길이가 증가함에 따라, strstr과 nfind, KMP 알고리즘의 시간 차이가 거의 없다. 이는 2-1에서 이끈 것처럼 strstr과 nfind가 KMP 알고리즘처럼 O(문장의 길이)와 비슷해지기 때문이다. 하지만 일치하는 경우도 있기 때문에 strstr과 nfind가 미세하게 연산 시간이 길다. Nfind가 strstr보다 약간 빠른 이유는 2-1 분석과 같다.

종합 분석: 연산 시간은 일반적으로 KMP 알고리즘, nfind, strstr 순으로 적다. 하지만, 문장의 길이와 패턴의 길이, 문장과 패턴의 일치 비율에 따라 그 차이는 줄어들 수 있다.