프로그래밍 실습 (스트링 탐색)

1. 텍스트 ababacabcbababcacacbcaababca에 대하여 패턴 ababca를 탐색한다고 할 때, 알고리즘 4.1 직선적 알고리즘을 파이썬으로 구현하라. 실행결과로 패턴이 나타난 위치를 출력한다. 마지막 패턴을 인식하지 못하면 텍스트의 마지막에 널 문자('\0')를 추가한다. 텍스트와 패턴에 있는 문자에 대한 총 비교횟수를 구하라.

알고리즘 4.1 직선적 알고리즘

```
bruteForce(p[], t[])
  M ← 패턴의 길이; N ← 텍스트의 길이;
  for (i ← 0, j ← 0; j  < M and i < N; i ← i + 1, j ← j + 1) do
    if (t[i] ≠ p[j]) then {
      i ← i − j;
      j ← −1;
    }
  if (j = M) then return i − M;
  else return i;
end bruteForce()
```

---

2. 알고리즘 4.3을 사용하여 다음과 같은 패턴에 대해 재시작 위치를 구하라.

(1) ababca
(2) abababca
(3) abcbabcbabc
(3) abracadabra

알고리즘 4.3 재시작 위치를 구하는 알고리즘

```
initNext(p[])
  M ← 패턴의 길이;
  next[0] ← -1;
  for (i ← 0, j ← -1; i  < M; i ← i + 1, j ← j + 1) do {
    next[i] ← j;
    while ((j ≥ 0) and (p[i] ≠ p[j])) do
      j ← next[j];
  }
end initNext()
```

---

3. 알고리즘 4.3을 수정하여 2번에서 구한 패턴에 대해 개선된 재시작 위치를 구하라. 알고리즘에서 next[i] ← j; 를 다음과 같이 변경한다.

```
-------------------------
if (j ≠ -1 and p[i] = p[j]) then next[i] ← next[j];
else next[i] ← j;
-------------------------
```

4. 텍스트 ababacabcbababcacacaababca에 대하여 패턴 ababca를 탐색한다고 할 때, 알고리즘 4.2 KMP 알고리즘을 파이썬으로 구현하라. 실행 결과로 패턴이 나타난 위치를 출력한다. 마지막 패턴을 인식하지 못하면 텍스트의 마지막에 널 문자('\0')를 추가한다. 텍스트와 패턴에 있는 문자에 대한 총 비교횟수를 구하라.

알고리즘 4.2 KMP 알고리즘

```
KMP(p[], t[])
  M ← 패턴의 길이; N ← 텍스트의 길이;
  initNext(p);
  for (i ← 0, j ← 0; j  < M and i < N; i ← i + 1, j ← j + 1) do
    while ((j ≥ 0) and (t[i] ≠ p[j])) do
      j ← next[j];
  if (j = M) then return i - M;
  else return i;
end KMP()
```

---

5. 텍스트 STRING STARTING CONSISTING에 대하여 패턴 STING을 탐색한다고 할 때, 알고리즘 4.5 보이어-무어 알고리즘을 파이썬으로 구현하라. 실행 결과로 패턴이 나타난 위치를 출력한다. 텍스트와 패턴에 있는 문자에 대한 총 비교횟수를 구하라.

이 알고리즘에서 initSkip() 함수는 패턴에 대한 skip 배열을 만들어주는 함수이고, index() 함수는 문자가 띄어쓰기(space)이면 0을 반환하고, 나머지 경우에는 영문 대문자의 i번째 문자에 대해 정수 i를 반환하는 함수이다. 예를 들어, index(A)를 호출하면 1을 반환하고, index(E)는 5를 반환한다.

알고리즘 4.5 보이어-무어 알고리즘

```
BM(p[], t[])
  M ← 패턴의 길이; N ← 텍스트의 길이;
  initSkip(p);
  for (i ← M-1, j ← M-1; j  ≥ 0; i ← i - 1, j ← j - 1) do
    while (t[i] ≠ p[j]) do {
      s ← skip[index(t[i])];
      if (M-j > s) then i ← i + M - j;
      else i ← i + s;
      if (i ≥ N ) then return N;
      j ← M - 1;
    }
  return i+1;
end BM()
```

---

skip 배열을 구하는 알고리즘

```
initSkip(p[])
  M ← 패턴의 길이;
  for (i ← 0; i < NUM; i ← i+1) do skip[i] ← M;
  for (i ←  0; i < M; i ← i+1) do skip[index(p[i])] ← M - i - 1;
end initSkip()
```

※ 위 알고리즘에서 영문 대문자를 사용할 경우 NUM의 값은 27이 됨.

---

index() 함수의 알고리즘

```
index(c)
  if (c의 아스키코드 = 32) then return 0;
  else return c의 아스키코드 - 64;
end index()
```

---

6. 텍스트 STRING STARTING CONSISTING에 대하여 패턴 STING을 탐색한다고 할 때, 알고리즘 4.6 라빈-카프 알고리즘을 파이썬으로 구현하라. 실행 결과로 패턴이 나타난 위치를 출력한다.
이 프로그램에서 사용되는 변수 중 q = 33554393, d = 32이다. index() 함수는 보이어-무어 알고리즘에서 사용된 것과 동일한 함수를 사용한다.

알고리즘 4.6 라빈-카프 알고리즘

```
RK(p[], t[])
  dM ← 1; h1 ← 0; h2 ← 0;
  M ← 패턴의 길이; N ← 텍스트의 길이;
  for (i ← 1; i  < M; i ← i + 1) do
    dM ← (d*dM) mod q;
  for (i ← 0; i  < M; i ← i + 1) do {
    h1 ← (h1 * d + index(p[i])) mod q;
    h2 ← (h2 * d + index(t[i])) mod q;
  }
  for (i ← 0; h1 ≠ h2; i ← i + 1) do {
    h2 ← (h2 + d * q - index(t[i]) * dM) mod q;
    h2 ← (h2 * d + index(t[i+M])) mod q;
    if (i > N-M) then return N;
  }
  return i;
end RK()
```

이 프로그램의 실행 결과로 출력되는 h1과 h2의 값은 다음과 같다.

```
-------------------------
h1 :  20587975
h2 :  20597038
h2 :  21571015
h2 :  19183840
h2 :  9903123
h2 :  14910068
h2 :  7360129
h2 :  643122
h2 :  20579924
h2 :  21023369
h2 :  1659182
h2 :  19539399
h2 :  21280992
h2 :  9903107
h2 :  14909551
h2 :  7343598
h2 :  114131
h2 :  3652201
h2 :  16207155
h2 :  15312500
h2 :  20237961
h2 :  10080558
h2 :  20587975
패턴이 나타난 위치 :  21
스트링 탐색 종료
-------------------------
```