**프로그래밍 실습 #8**

2023년 11월 3주차

□ 다음 스트링 탐색 알고리즘을 파이썬으로 구현해 보라.

**(1) 직선적 알고리즘**

**소스 코드**

def bruteForce(p, t, k):  
 M = len(p)  
 N = len(t)  
  
 # k는 이전 탐색이 끝난 지점부터 이어주는 역할.  
 i, j = k, 0  
  
 while j < M and i < N:  
 if t[i] != p[j]:  
 i = i - j  
 j = -1  
 i += 1  
 j += 1  
 # pattern 길이와 j(새로 탐색한 pattern 의 끝 index)가 같으면  
 if j == M:  
 return i - M  
 else:  
 return i  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 text = 'ababababcababababcaabbabababca' + '\0'  
 pattern = 'abababca'  
 M = len(pattern)  
 N = len(text)  
 K = 0  
 while True:  
 pos = bruteForce(pattern, text, K)  
 K = pos + M  
 if K < N:  
 print('패턴이 나타난 위치 : ', pos)  
 # print(f"재 시작 위치 (k) : {K}")  
 else:  
 break  
 print('스트링 탐색 종료')

**실행 결과**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**(2) 재시작 위치를 구하는 알고리즘**

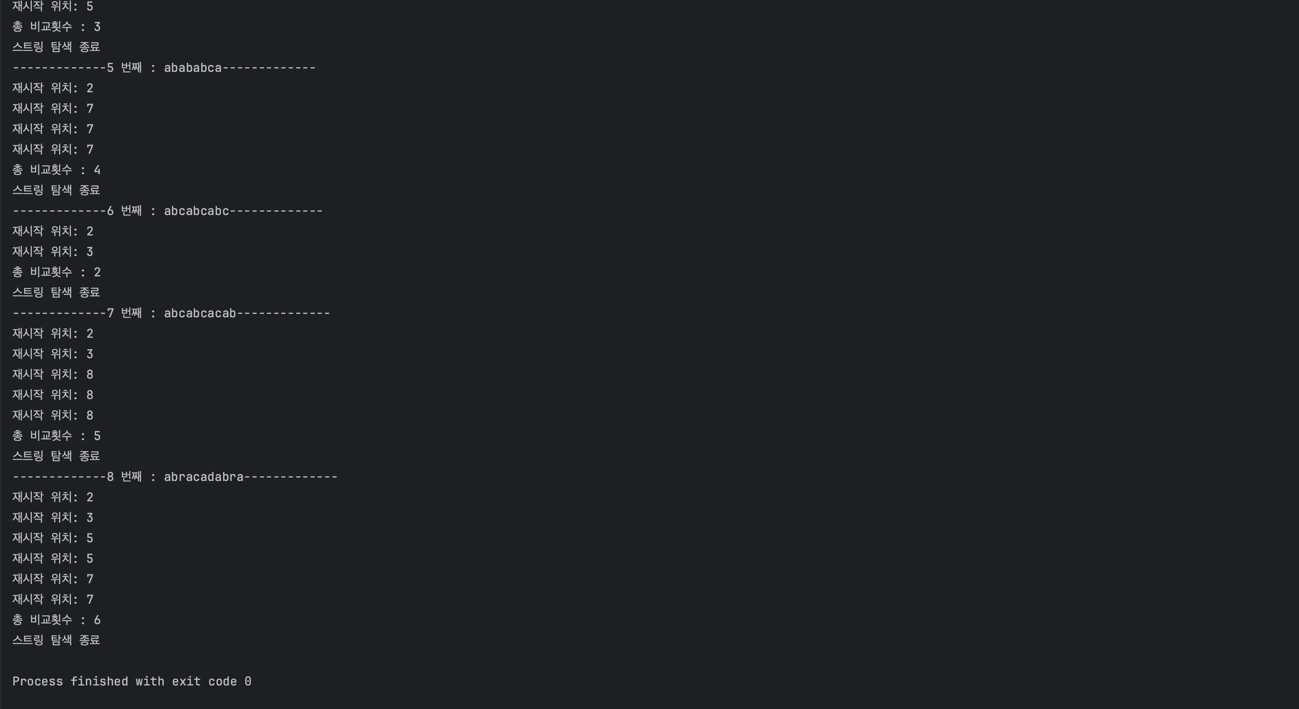
**소스 코드**

def init\_next(p):  
 M = len(p)  
 next[0] = -1  
  
 i, j, comp = 1, 0, 0  
 while i < M:  
 next[i] = j  
 while j >= 0 and p[i] != p[j]:  
 j = next[j]  
 print(f"재시작 위치: {i + 1}")  
 comp += 1  
 i += 1  
 j += 1  
 print(f"총 비교횟수 : {comp}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 pattern\_list = [  
 # 'abababca',  
 # '10100111'  
 'aaaaaaaaa',  
 '00000001',  
 '10100111',  
 'ababca',  
 'abababca',  
 'abcabcabc',  
 'abcabcacab',  
 'abracadabra'  
 ]  
  
 for i in range(len(pattern\_list)):  
 pattern = pattern\_list[i]  
 print(f"-------------{i + 1} 번째 : {pattern}-------------")  
 next = [0] \* len(pattern)  
 init\_next(pattern)  
 print('스트링 탐색 종료')

**실행 결과**

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명



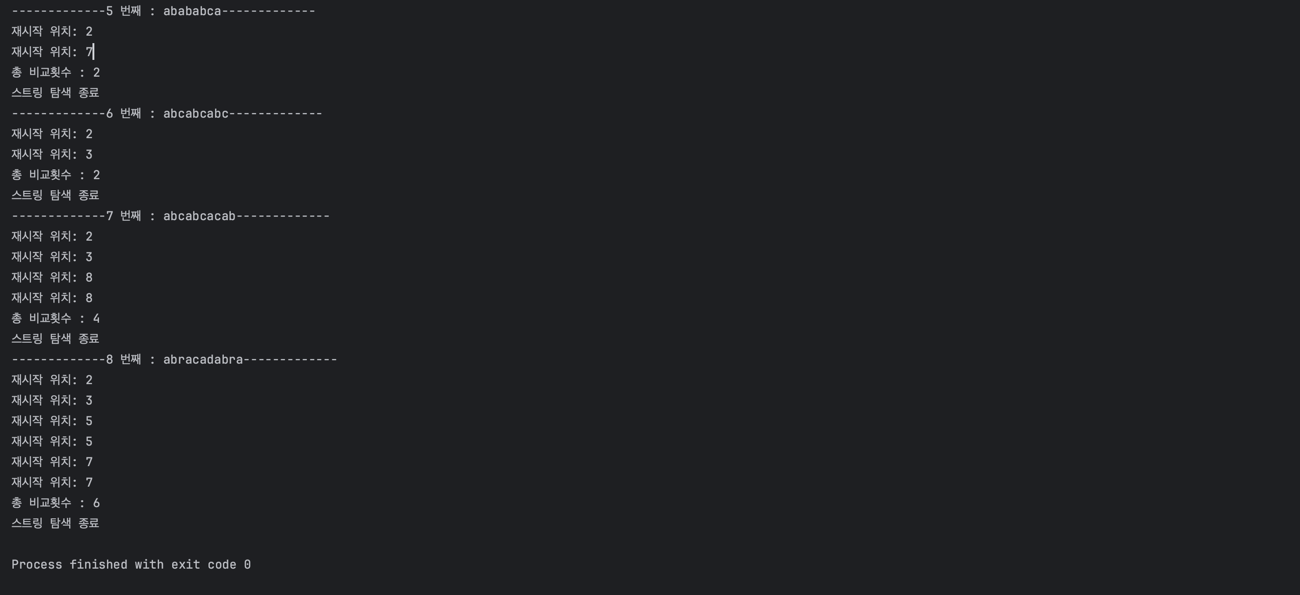
**(3) initNext() 함수 개선**

**소스 코드**

def improved\_init\_next(p):  
 M = len(p)  
 next[0] = -1  
  
 i, j, comp = 1, 0, 0  
 while i < M:  
 if p[i] == p[j]:  
 next[i] = next[j]  
 else:  
 next[i] = j  
 while j >= 0 and p[i] != p[j]:  
 j = next[j]  
 comp += 1  
 # print(f"값(p[i]:{p[i]} != p[j]:{p[j]}")  
 print(f"재시작 위치: {i + 1}")  
 i += 1  
 j += 1  
 print(f"총 비교횟수 : {comp}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 pattern\_list = [  
 # 'abababca',  
 # '10100111'  
 'aaaaaaaaa',  
 '00000001',  
 '10100111',  
 'ababca',  
 'abababca',  
 'abcabcabc',  
 'abcabcacab',  
 'abracadabra'  
 ]  
  
 for i in range(len(pattern\_list)):  
 pattern = pattern\_list[i]  
 print(f"-------------{i + 1} 번째 : {pattern}-------------")  
 next = [0] \* len(pattern)  
 improved\_init\_next(pattern)  
 print('스트링 탐색 종료')

**실행 결과**

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

**(4) KMP 알고리즘**

**소스 코드**

def init\_next(p):  
 M = len(p)  
 next[0] = -1  
  
 i, j, comp = 1, 0, 0  
 while i < M:  
 if p[i] == p[j]:  
 next[i] = next[j]  
 else:  
 next[i] = j  
 while j >= 0 and p[i] != p[j]:  
 j = next[j]  
 comp += 1  
 print(f"값(p[i]:{p[i]} != p[j]:{p[j]}")  
 print(f"재시작 위치: {i + 1}")  
 i += 1  
 j += 1  
 print(f"총 비교횟수 : {comp}")  
  
def KMP(p, t, k):  
 M = len(p)  
 N = len(t)  
 i, j = k, 0  
 init\_next(p)  
 while j < M and i < N:  
 while j >= 0 and t[i] != p[j]:  
 j = next[j]  
 i += 1  
 j += 1  
  
 if j == M:  
 return i - M  
 else:  
 return i  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 next = [0] \* 50  
 text = 'ababababcababababcaabbabababca' + '\0'  
 pattern = 'abababca'  
 M = len(pattern)  
 N = len(text)  
 K = 0  
 while True:  
 pos = KMP(pattern, text, K)  
 K = pos + 1  
 if K <= N - M:  
 print('패턴이 나타난 위치 : ', pos)  
 else:  
 break  
 print('스트링 탐색 종료')

**실행 결과**

텍스트, 스크린샷, 블랙, 폰트이(가) 표시된 사진

자동 생성된 설명

(5) 보이어-무어 알고리즘

**소스 코드**

def index(c):  
 if ord(c) == 32:  
 return 0  
 else:  
 return ord(c)-64  
  
def initSkip(p):  
 M = len(p)  
 for i in range(NUM):  
 skip[i] = M  
 for i in range(M):  
 skip[index(p[i])] = M - i - 1  
  
def BM(p, t, k):  
 M = len(p)  
 N = len(t) - 1  
 initSkip(p)  
 i, j = M - 1 + k, M - 1  
 while j >= 0:  
 while t[i] != p[j]:  
 # 패턴 못 찾은 경우  
 if i >= N:  
 return N  
 s = skip[index(t[i])]  
 # 패턴 길이 - 나쁜 문자가 패턴에서 가진 인덱스 > skip 값  
 if M - j > s:  
 # 텍스트의 인덱스 = 텍스트 인덱스 + 패턴 길이 - 나쁜 문자가 패턴에서 가진 인덱스  
 i = i + M - j  
 else:  
 # 텍스트의 인덱스는 skip 값 만큼 증가  
 i = i + s  
 j = M - 1  
 i -= 1  
 j -= 1  
 return i + 1  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 NUM = 27  
 skip = [0] \* NUM  
 text = 'VISION QUESTION ONION CAPTION GRADUATION EDUCATION' + '\0'  
 pattern = 'ATION'  
 M = len(pattern)  
 N = len(text)  
 K = 0  
 while True:  
 # pos = Pattern 이 나타난 위치  
 pos = BM(pattern, text, K)  
 K = pos + 1  
 if K <= N - M:  
 print('패턴이 나타난 위치 : ', pos)  
 else:  
 break  
 print('스트링 탐색 종료')

**실행 결과**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

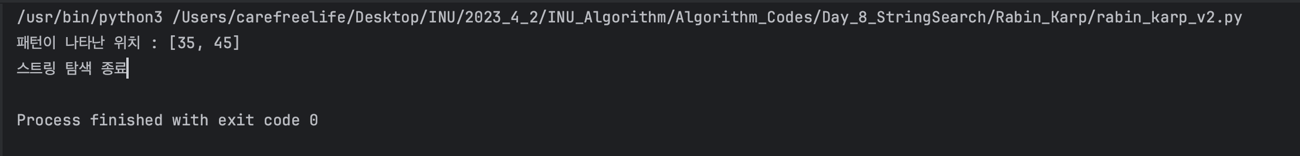
자동 생성된 설명

**(6) 라빈-카프 알고리즘**

**소스 코드**

def index(c):  
 if ord(c) == 32:  
 return 0  
 else:  
 return ord(c) - 64  
  
def RK(p, t, k):  
 q = 33554393  
 d = 32  
 M, N = len(p), len(t)  
  
 d\_m = pow(d, M - 1, q) # d^(M-1) % q 미리 계산  
  
 h1 = 0 # 패턴의 Hash 값  
 h2 = 0 # text 의 첫번째 substring 의 hash 값  
  
 # 패턴 및 text 의 첫번째 Substring 의 hash 값 계산  
 for i in range(M):  
 h1 = (h1 \* d + index(p[i])) % q  
 h2 = (h2 \* d + index(t[i])) % q  
  
 occurrences = []  
  
 while k <= N - M:  
 # hash 값이 맞으면 char 단위로 check  
 if h1 == h2 and p == t[k : k + M]:  
 occurrences.append(k)  
  
 # text 의 다음 substring 탐색을 위해 hash 값 업데이트  
 if k + M < N:  
 h2 = (h2 - index(t[k]) \* d\_m) % q  
 h2 = (h2 \* d + index(t[k + M])) % q  
  
 k += 1  
  
 return occurrences  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 q = 33554393  
 d = 32  
 text = 'VISION QUESTION ONION CAPTION GRADUATION EDUCATION' + '\0'  
 pattern = 'ATION'  
 M = len(pattern)  
 N = len(text)  
 K = 0  
  
 occurrences = RK(pattern, text, K)  
  
 if occurrences:  
 print(f'패턴이 나타난 위치 : {occurrences}')  
 else:  
 print('패턴이 존재하지 않습니다.')  
  
 print('스트링 탐색 종료')

**실행 결과**

****

**이상입니다.**

**감사합니다.**

**201702797 채승민**