알고리즘_HW06_결과보고서

201802088_류지현

1. 구현 아이디어

주어진 문자열 “pattern“에 대해서 생각해보자. 자르는 위치가 {010101}로 주어진다. 처음에는 자르는 위치에 대해서 문자열을 자른 후 길이를 비교하는 방법을 생각해보았다. 하지만 이 방법은 구현하기 힘들어서 자르는 위치에 대해서 주어진 문자열을 모두 자른 후에 합치는 과정으로 생각했다.

즉 ”pattern“을 ”pa“, “tt“, “er“, “n“으로 자른다. 이를 이웃한 문자열끼리 합하면서 비교를 한다. 이웃한 문자열끼리 합하는 이유는 ”pa“ + “er“은 애초에 존재할 수 없는 경우이다.

2. 구현

dp[i][j] = {i 번째 부분문자열부터 **j** 번째 부분문자열까지의 합하는 최소 비용}이라 하자. 이때 **dp[i][i]**에 대해서 생각해보면 크게 의미가 없다. 이 경우 처음 자른 문자열의 상태이기 때문이다. **dp[i+1][i+1]**의 결과와 더한 **dp[i][i+1]**일 때 **dp[i][i+1] = cost[i] + cost[i+1]** 이다. 이후 **dp[i][j]**에 대한 recurrence equation은 다음과 같다. **dp[i][j] = min{dp[i][j], dp[i][k] + dp[k+1][j] + sum[j] - sum[i-1]} (i <= k < j, sum[j] - sum[i-1]은 이전 결과를 더해준다.)**

Algorithm 1: cutString

Data: $cost[]$
Result: $dp[1][n-1]$

```
1  n = length[cost]
2  dp[n][n], sum[n]
3  for i = 1 to n − 1 do
4  |  sum[i] = sum[i − 1] + cost[i];
5  end
6  for l = 1 to n − 1 do
7  |  for i = 1 to n − l − 1 do
8  |  |  j = i + l
9  |  |  dp[i][j] = MAX
10 |  |  for k = i to j − 1 do
11 |  |  |  dp[i][j] =
   |  |  |    min(dp[i][j], dp[i][k] + dp[k + 1][j] + sum[j] − sum[i − 1])
12 |  |  end
13 |  end
14 end
15 return dp[1][n − 1]
```

0	3	9
0	0	4
0	0	0

“pattern” dp배열 예시

3. 시간복잡도

이 문제는 matrix chain order 알고리즘과 구현방식이 똑같으므로 시간복잡도를 구해보면, dp배열을채우는데 for문을 3번 반복하게 되므로 $T(n) = O(n^3)$이 된다.

4. 결과

```
<terminated> Mainjava (6) [Java Application] C:₩Program Files₩Java₩jre1.8.0_301₩bin₩javaw.exe (2021. 10. 29. 오후 8:05:53)
9
24
11
0
7
133
14
```