07/20 DP

2020년 7월 20일 월요일 오후 7:25

DP (Dynamic programming), 동적 프로그래밍에 대해 알아보자!!!!!!!

https://www.zerocho.com/category/Algorithm/post/584b979a580277001862f182 를 참고하였다!!!

동적 프로그래밍의 프로그래밍은 놀랍게도 컴퓨터 프로그래밍의 프로그래밍이 아니라 테이블을 만든다는 뜻이다!!!!
어떤 교수님인지는 모르겠지만 어떤 교수님은 동적 프로그래밍 대신 기억하기 프로그래밍이라는 용어를 쓴다!!

메모이제이션을 아는가!
재귀 호출 시, 반복적으로 계산되는것들의 계산 횟수를 줄이기 위해 이전에 계산했던 값을 저장해두었다가 나중에 재사용하는 방법이다.
메모이제이션이 동적 프로그래밍 중 하나이다!!

분할정복에도 적용된다~!!!!!!
알고리즘 짤 때 분할정복 기법을 사용하는 경우가 많은데,
큰 문제를 한 번에 해결하기 힘들 때 작은 여러개의 문제로 나누어서 푸는 기법이다.
작은 문제들을 풀다보면 같은 문제들을 반.복.해서 푸는 경우가 생기는데
그 문제들을 매번 재계산하지 않고 값을 저장해두었다가 재사용하는 기법이 동적 프로그래밍이다!!!

동적 프로그래밍의 대표적인 세가지 문제가 있다!!!!!

1. 막대기 자르기
   문제. 하나의 긴 막대기가 있는데 막대기 조각마다 가격이 다르다. 막대기를 적절히 잘라 가장 높은 가격을 만들어라.
   길이(i) 0 1 2 3 4 5 6 7 8 9 10
   가격(Pi) 0 1 5 8 9 10 17 17 20 24 30

   예시)
   길이가 4인 막대기의 최대 가격은 길이 2인 막대기 두개로 5+5=10을 만드는 것이다!
   길이가 6인거는 그냥 안자르고 17에 파는 게 최대의 가격!

   풀이)
   길이가 n인 막대기의 최대 가격을 Rn이라고 했을 때, Rn = max(Pi + Rn-i) 으로 나타낼 수 있다.(i는 1부터 n까지)
   예시의 길이가 4인 막대기의 최대가격은 R4= max(P1+R3, P2+R2, P3+R1, P4+R0)이다
   P1, P2, P3 값은 이미 알고 있다. R1, R2, R3를 구해야한다.
   R1은 Rn = max(Pi + Rn-i) 식에서 max(Pi+R0)이므로 1이다.
   R2는 R2 = max(P1+ R1, P2+R0) = max(1+1, 5+0)이므로 5이다.
   R3는 R3 = max(P1+ R2, P2+R1, P3+R0) = max(1+5, 5+1, 8+0) = 8이다.
   R4는 R4 = max(P1 + R3, P2 + R2, P3 + R1, P4 + R0) = max(1+8, 5+5, 8+1, 9) = 10

   이러한 계산과정에서 R1, R2, R3 값들을 저장할 수 있다.
   Top-down으로 불리는 메모이제이션을 사용할 수도 있고
   Bottom-up이라 불리는 상향식 계산법을 사용할 수도 있다.

   상향식 계산법 풀이

```
public static void main(final String[] args) throws Exception {

    int[] p = {0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 30};

    System.out.println(getMaxVal(p, 7));
}

static int getMaxVal(int[] p, int n){
```

```
        int[] r = new int[n+1];

        for(int i = 1; i <= n; i++){
            int max = 0;
            for(int j = 1; j <= i; j++){
                max = Math.max(max, p[j]+r[i-j]);
            }
            r[i] = max;
        }

        return r[n];
    }
```

2. 최장 공통 부분 수열 문제

   최장 공통 부분 수열 문제.. 이름부터 어렵다.

   문제. 최장 공통 부분 수열(LCS) : 두 개의 문자열에서 순서대로 겹치는 문자가 최대 몇 개인가?

   예시)

   ABCBDAB와 BDCABA에서 LCS는 BCAB나 BDAB나 BCBA이다.

   앞에서부터 겹치는 것들을 찾아 문자열을 만들때 그것이 가장 길다면 LCS이다.

   풀이)

   i라는 문자열과 j 라는 문자열이 있다.

   lcs(i, j)는 이 두 문자열의 LCS 길이이다.

   만약 문자열의 마지막 문자가 같다면 lcs(i, j) = lcs(i-1, j-1)+1과 같다.

   lcs(ABCBDAB, BDCAB)는 lcs(ABCBDA, BDCA) + 1과 같다.

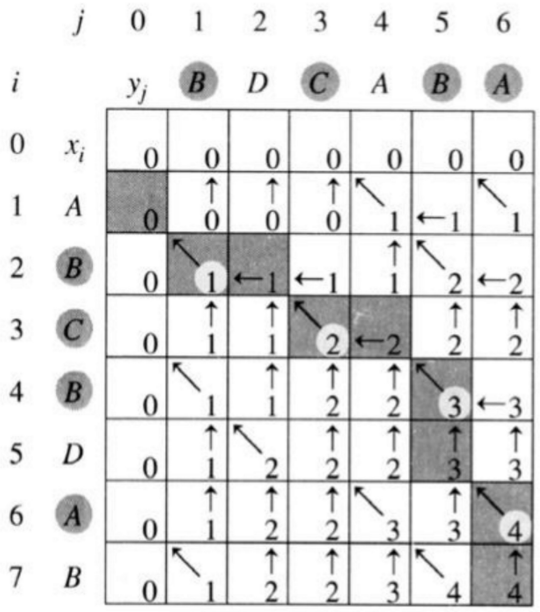
   만약 마지막 문자가 다르다면 lcs(i, j) = max(lcs(i, j-1), lcs(i -1, j))가 된다.

   lcs(ABCBDAB, BDCABA)는 lcs(ABCBDAB, BDCAB)와 c(ABCBDA, DBCABA) 중 더 큰 값이 된다.

   이렇게 끝부분을 비교해 하나씩 줄여나가면 LCS의 길이가 나오게 된다.

ㄱ) 두 문자가 같을 때, 좌측위에 있는 숫자에 1을 더한 값을 가지며 화살표는 좌측위를 가리킵니다.
ㄴ) 두 문자가 다를 때, 좌측이나 위에 있는 숫자중 큰 값을 가리키며 같은 값을 가집니다. 동일한 값일 때는 위의 값에 대해 행동합니다.

먼저, i가 1일때 1부터 6까지의 j에 대해 확인합니다.
X1는 A이며 Y1은 B이기 때문에 두 문자가 서로 다르고, 좌측과 위에 있는 숫자가 같은 값을 가지기에 그 값을 가지며 위를 가리킵니다.
X1과 Y2도 같은 방식이고, X1과 Y3 또한 같은 방식입니다.
X1과 Y4를 보시면 서로 같은 문자, A를 가지고 있습니다. 따라서 대각(좌측상단) 방향의 값보다 1큰 값인 1을 값으로 가지며 대각을 화살표로 가리킵니다.
이러한 방법으로 모든 칸을 채우면 아래와 같이 채워지게 됩니다.



$$c[i,j] = c[0..m, 0..n]$$
$$= \text{length of LCS}$$
$$b[i,j] = b[1..m, 1..n]$$
$$= \text{table entry indicator}$$

O(mn) time

https://doorbw.tistory.com/59

3. 0/1 배낭문제
배낭 문제는 무게 제한이 50인 배낭에 다음과 같은 세 개의 물건을 넣는 문제이다.
넣은 물건들의 가치(v) 합이 최대가 되면 된다.
문제는 세 물건의 무게(w)를 합치면 60이라 다 넣지는 못한다.

문제의 이름이 0/1 배낭문제인 이유는 물건을 쪼개서 넣지는 못하고,
선택지가 통째로 d넣거나 아예 안넣거나 두개뿐이기 때문이다.

https://gsmesie692.tistory.com/113

$$P[i,w] = \begin{cases} P[i-1,w] & \text{if } w_i > w \\ \max\{v_i + P[i-1, w-w_k], P[i-1,w]\} & \text{else} \end{cases}$$

P[i, w] 란 i개의 보석이 있고 배낭의 무게 한도가 w일 때 최적의 이익을 의미한다. 식을 좀 문장으로 풀어보면 이렇다.

- i번째 보석이 배낭의 무게 한도보다 무거우면 넣을 수 없으므로 i번째 보석을 뺀 i-1개의 보석들을 가지고 구한 전 단계의 최적값을 그대로 가져온다
- 그렇지 않은 경우, i번째 보석을 위해 i번째 보석만큼의 무게를 비웠을 때의 최적값에 i번째 보석의 가격을 더한 값 or i-1개의 보석들을 가지고 구한 전 단계의 최적값 중 큰 것을 선택한다