고급 소프트웨어 실습

분반: 1 분반

학번: 20181687

이름: 정지석

과제 9.

1. 실습 3 이 Greedy algorithm 과 Dynamic programming 중 어디에 해당된다고 할 수 있는가? 이유를 설명하시오.

Dynamic programming 에 해당된다. 3 번 문제를 최대한 빠른 시간안에 해결하기 위해서는 주어진 수열을 반복문을 통해 돌면서 이전에 계산한 값들 중 큰 값들만 저장하여 반복 계산을 최소화한 후, 저장된 값들 중 가장 큰 값을 출력하는 방법을 사용해야한다. 위와 같은 해결 방법은 Dynamic programming 의 memoization 에 해당된다고 할 수 있다.

memoization 이란 동일한 계산을 반복해야 할 때, 이전에 계산한 값을 메모리에 저장해 반복 수행을 제거해 속도를 향상시키는 Dynamic programming 의 핵심 기법 중 하나로, 이전에 계산한 값들을 메모리에 저장하는 방식이 3 번 문제 해결에 사용된 것을 통해 실습 3 이 Dynamic programming 에 해당된다는 것을 확인할 수 있다.

2. 앞에서 다루지 않은 문제 중에 Dynamic programming 을 사용했을 때 더 효율적으로풀 수 있는 문제를 2 개 들고, 알고리즘을 설명하시오. brute force 방식에 비하여 DP 를 사용하는 경우 어느 정도 빨라지는지도 설명하시오.

Ex1) 숫자판 문제

숫자들이 2차원 배열에 있다. 가장 윗 행에서 가장 아래 행까지 내려올때 걸쳐지는 길에 있는 숫자의 합이 가장 높은것을 찾아보자. 이 때 어느 한 cell에서 다음 cell로 내려올 경우에는 바로아래, 왼쪽대각선아래, 오른쪽대각선 아래로만 이동이 가능하다.

3	4	9	-2	2	51	-23	2	-1
223	7	8	-11	5	-99	2	3	-4
2	51	-23	-23	6	3	2	4	5
5	-99	2	-1	32	2	5	-99	2
6	3	3	-4	2	-1	6	3	3
32	2	4	5	3	-4	2	-1	4
4	4	23	6	2	-1	3	-4	34
78	32	1	7	3	-4	-23	-23	6

위의 문제를 brute force 방식을 이용해 해결한다고 해보자. 각 칸에서 이동할 수 있는 경우의 수는 양 모서리를 제외하고 3 이 되므로, 총 시간 복잡도는 대략 O(3^column)이 될 것이다. 이렇게 될 경우 배열의 숫자가 커지게 되면 시간 복잡도가 기하 급수적으로 늘어나는 문제가 생기게 된다.

이를 해결하기 위해서는 해당 문제를 Dynamic programming 을 사용하여 해결하면 된다. 이를 위해서는 먼저 점화식을 세울 필요가 있다. 현재 자신의 위치에서 위에 있는 세 개의 칸(오른쪽 위, 바로 위, 왼쪽 위) 중 가장 큰 값과 자신의 값을 더한다고 생각하고 점화식을 만들면 다음과 같다.

array[i][j] = array[i][j] + max(array[i-1][j-1], array[i-1][j], array[i][j-1])

모든 배열 요소에 대해 반복문을 돌리면서 누적 합을 구하면, 가장 큰 값을 찾을 수 있게 된다. 위와 같은 방법을 사용하게 되면 각 행과 열에 대해 반복문을 돈 것이므로 시간 복잡도는 O(row * column)이 된다. Brute force 를 사용하여 문제를 해결하였을 때의 시간 복잡도가 O(3^column)인 것을 생각해보면 알고리즘의 수행 속도가 현저히 빨라진 것을 확인할 수 있다.

Ex2) 오르막 수

오르막 수는 수의 자리가 오름차순을 이루는 수를 말한다. 이때, 인접한 수가 같아도 오름차순으로 친다. 예를 들어, 2234 와 3678, 11119 는 오르막 수이지만, 2232, 3676, 91111 은 오르막 수가 아니다. 수의 길이 N 이 주어졌을 때, 오르막 수의 개수를 구하는 프로그램을 작성하시오. 수는 0 으로 시작할 수 있다.

해당 문제를 brute force 로 해결하게 되면 각 자리 수에 나올 수 있는 모든 경우의 수(0~9)를 모두 확인해 봐야하기 때문에 시간 복잡도는 대략 O(10^N)이 된다. 이렇게 될 경우 N 의 개수가 커질수록 실행 시간이 기하 급수적으로 늘어나게 된다.

이를 해결하기 위해서는 해당 문제를 Dynamic programming 을 사용하여 해결하면 된다. 이를 위해서는 먼저 점화식을 세울 필요가 있다.

	0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1	1
2	1	2	3	4	5	6	7	8	9	10
3	1	3	6	10	15	21	28	36	45	55

위의 표에서 세로축은 N, 가로축은 1 의 자리에 나온 숫자를 의미한다. 예를 들어 N 이 2 이고 1 의 자리에 2 가 들어오게 되면, 나올 수 있는 오르막 수는 02, 12, 22 로 총 3 개가 나오는 것을 확인할 수 있다. 위의 표를 자세히 살펴보면 하나의 점화식을 유도할 수 있는데, N 이 j 이고 1 의 자리가 i 일 때의 경우의 수는 N 이 j 이고 1 의 자리가 i-1 인 경우의 수와 N 이 j-1 이고 1 의 자리가 i 인 경우의 수를 더한 것과 같다. 이를 이해하기 쉽게 식으로 나타내면 아래와 같다.

array[i][j] = array[i-1][j] + array[i][j-1]

이 식을 반복문을 사용하여 계산하면 길이가 N 일 때의 오르막 수의 개수를 구할 수 있게 된다. 반복문을 사용하여 배열의 모든 요소를 한번씩 접근하므로 시간 복잡도는 O(N * 10)이 된다. brute force 를 사용했을 때와 비교하면 실행 시간이 현저히 줄어든 것을 확인할 수 있다.