Lecture 9 : Dynamic Programming의 기초

이번 장에서 다룰 내용은 프로그래밍 대회, 입사 코딩에서 항상 나오고 또 대부분의 초보 프로그래머들이 까다롭게 생각하는 동적계획법이다. 보통 학생들은 간단히 DP 라고 부르기도 한다. 동적계획법의 핵심은 inductive computing이다. 즉 어떤 문제가 있을 때 이를 어떤 매개변수 n의 단계로 구분한다. 예를 들어 어떤 배열(array)에서 가장 큰 값을 찾는 문제를 생각해보자. array D[N]의 문제는 N을 변수로 이해한다. 즉 우리가 해결해야 할 문제는 D[N]에서 max를 찾는 것이다. 그런데 만일 우리가 D[N-1]에서 max를 안다고 할 때 이를 활용해서 원래 문제는 D[N]에서의 max는 어떻게 알아낼 것인가 생각해보자. 그것은 매우 간단하다. 그 답은 아래와 같다.

```
max { D[N] } = max {  D[N], max { D[N-1] }
```

즉 앞에서 N-1번째의 max와 제일 끝 원소 D[N]을 비교해서 더 큰 것을 선택하면 전체 D[N]에서 가장 큰 원소가 된다. 물론 배열에서 max를 찾는 문제를 이런 식으로 해결하지는 않지만 이것이 바로 Dynamic programming의 핵심이다. 즉 어떤 전체 문제 P의 답을 구할 때, 만일 우리가 P보다 조금 더 적은 크기의 문제, 즉 P의 부분집합의 답을 알 수 있다고 가정했을 때 이것으로부터 전체 문제의 답을 유추하는 것이다. 이것이 말로는 쉽지만, 실제 현실에서는 상당히 까다롭고 많은 경험이 필요한 계산 과정이다.

동적 프로그래밍으로 해결되는 문제는 응시자의 수준을 평가하기에 매우 유용하여 각종 대회, 코딩 테스트에서 반드시 출제되는 문제 형식이다. 이 문제는 해당되는 동적계획법의 식을 구할 수 있으면 만점을 받을 수 있고 아는 경우는 거의 예외없이 0점을 받기 때문에 코딩 테스트에서 고득점을 받기 위해서는 반드시 돌파해야 할 개발 패러다임이다. 그런데 어떤 문제 P를 주고 이것을 Dynamic Programming으로 해결하라[1]-라고 지시를 하면 그나마 쉽지만 이런 가이드 없이 주어지는 현실에서는 매우 당황스러운 상황이 된다.

이것을 극복하는 핵심을 다양한 문제를 풀어보는 것 밖에 없다. 여기에 표시된 사이트에 제시된 문제를 모두 해결할 수 있다면 대부분의 dynamic programming으로 해결되는 문제는 해결할 수 있을 것이다.[2] 하지만 이중 Dynamic programming (double induction)[3]과 같은 문제는 여기에 제시되어 있지 않기 때문에 좀 더 고난이도의 문제를 찾아서 해결해야 할 필요가 있다.

---

1) 우리 알고리즘 강의나 과제물의 경우와 같이. 교수가 친절하게 “이 문제를 동적계획법으로 해결하시오”라고 가이드를 하면 그나마 해결가능하다.

2) CrazyforCode http://www.crazyforcode.com/dynamic-programming/

3) G(n,k)를 n과 모든 k에 대하여 증명하는 것. 이 경우 k를 상수로 두고 모든 n에 대하여 증명한 다음 그 다음에는 n을 상수로 두고 모든 k에 대하여 증명하는 방식.

한 가지 명심해야 할 사실은 동적계획법이 효율적인 알고리즘을 보장해주는 것은 아니라는 점이다. 이 방법은 전혀 감이 잡히지 않는 문제에 접근이 가능한, 즉 효율과 상관없이 하나의 답을 제시해줄 수 있는 방법이라는 것이다. 따라서 어떤 경우 전혀 해결책을 찾지 못하는 것보다 시간은 비록 상당히 소요되지만 답을 보장하는 동적계획법이 실용적인 대책이 될 수 있다.

9.1 동적계획법 (dynamic programming) 방법의 개요

- Dynamic Programming의 유래:

이 방법은 2차대전이 한참인 시절 미국에서 전략계산 과정에서 도출되었다고 알려져 있다. 예를 들어 폭격기나 전투기에 폭탄이나 공격용 무기를 많이 실으면 공격력은 강화되지만 그 무게로 인하여 기동성이 떨어져 적의 공격에도 쉽게 노출된다. 따라서 공격과 방어에 가장 최적인 무게는 어는 정도인지 등을 계산하는 과정이 필요하였다. 이러한 연구를 일반적으로 Operations Research라고 부르고 산업공학[4]의 대표적인 연구분야 중의 하나이다.

그런데 이러한 연구의 연구비를 요청하기 위하여 국회에 프로젝트 제안서를 보내야 하는데, 그 이름의 연구의 취지에 맞게 “귀납적 방법을 사용한 최적화 문제 해결 기법” 이렇게 처음 제안하였다. 그런데 연구팀 일각에서 “그렇게 해서 상하원 의워들이 알아먹겠냐고?” 이런 회의론이 나오게 된것이다. 그리하여 쉽게 멋있게 보이는 이름으로 새로 만들어진 것이 바로 “Dynamic Programming"이다.

- 콘서트 프로그램, C++ Program, 동적 프로그램의 같은 점, 다른 점
- 컴퓨터 이전 시대에서 <프로그래밍>이 의미하는 바는 무엇인가 ?
- 정확한 이름은 “Inductive Algorithmic Solving"(귀납적 해결법)

9.2 동적계획법의 원리

– 주어진 문제 X보다 약간 더 쉬운 문제 Y를 풀 수 있다면
– Y 결과의 답을 모아서
– 주어진 문제 X의 답을 좀 더 쉽게 찾을 수 있다.

e.g.) **부산**에서 **강릉까지** 가는 가장 빠른 길을 알고 싶다고 ?
강릉에 인접한 4개 도시{ a, b, c, d }까지 가는 가장 빠른 길을 우리가 알 수만 있다면 원래 문제를 풀 수 있다.

4) 우리나라에서 번역이 조금 이상하게 되었지만 산업공학은 응용수학을 공학에 적용한 대표적인 수리 과학(mathematical science)의 한 분야이다. 요즘의 컴퓨터 기술의 발달로 이런 취지가 많이 퇴색되었다. 특히 인공지능 기술을 통한 최적화는 산업공학의 근간을 흔드는 큰 위기를 만들었다.

Q) 연 5000만원 이상 벌 수 있는 사람이 되고 싶다면 ?
A) 나에게 연 4500만원 버는 방법을 가르쳐 다오, 그러면

9.3 동적계획법을 활용할 때 얻을 수 있는 계산적인 이점
- 이미 계산된 결과를 저장하여 계산 없이 바로 불러 사용한다.
(Do not re-compute the same problem.)

9.4 동적계획법 알고리즘 설계의 개념적 구조

a. 문제의 단계별 분할
S = [s(1) | s(2) | ... s(n)] , s(i)은 sub-problem

b. 이전 단계의 답들이 모두 구해져 있을 때, { S(1), S(2)... S(n) }
이들을 이용해서 S(n+1)의 해를 구하기

c. 단 그 전 단계의 답을 모두 "꽁꽁"기록해서 잘 관리해야 한다.

9.5 동적계획법 실행을 위한 **3가지** 준비물

a) 동적계획 공식 (Dynamic Programming formula)
- 주로 recursive form으로 제시된다.

b) 초기조건 (Base condition)

c) 적당한 크기의 Table[ ][ ] 또는 몇 개의 Table[ ][ ]

9.6 간단한 동적계획법 알고리즘의 예(1)
- 이항계수(Binomial coefficient)의 두 가지 계산법

a. 대수식을 이용해서 직접 계산하기.
$\binom{n}{k} = \frac{n!}{k!(n-k)!}$, $\binom{120}{3}$을 구하려면 120!, 117!을 구해야 함

b. 재귀(recurrence)적으로 정의하여 단계적으로 구하기 ($n \geq k$)

$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

$\binom{n}{k} = 1$ if k= 0 이거나 k=n 이면

9	9								
8	8	28				$\binom{n}{k}$			
7	7	21			$\binom{n-1}{k-1}$	$\binom{n-1}{k}$			
6	6	15							
5	5	10	10	5					
4	4	6	4	1					
3	3	3	1	0	0	0	0	0	0
2	2	1	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0
n/k	1	2	3	4	5	6	7	8	9

9.7 [동적계획법] 접근법으로 잘 해결되는 작은 문제들

a) N 칸의 사다리를 1칸, 또는 2칸 증가를 이용해서 올라가기

c) 주어진 숫자 N을 1과 2의 순서있는(ordered) 합으로 표시하기

N=5 인 경우 1 + 1 + 1 + 1 + 1,
2 + 2 + 1
1 + 2 + 2
1 + 2 + 1 + 1
2 + 1 + 2

9.8 **실전문제** ["똥개와 똥, 그리고 개장수]

똥개는 "앞"쪽으로만 전진을 한다. 여기서 앞이란 오른쪽, 그리고 위 방향을 말한다. 똥개는 S에서 시작하여 T까지 가는 길에 가능하면 많은 똥(&)을 주워 먹으려고 한다. 어떤 길을 택하면 가장 많은 똥을 먹고 T까지 갈 수 있는지 생각해보자. 그런데 중간에 개장수(dog hunter) X를 만나면 똥개는 다른 세상을 만나게 되므로 이는 절대 피해야 한다.

			&			T
	&	&	X			
					&	
&	X		&		X	
						X
	&				&	
S				&	&	&

9.9 **실전문제 (숫자놀이)** 여러분은 위에서부터 하나의 숫자를 잡아서 밑으로 내려간다. 그리고 지나온 칸에 적힌 해당되는 숫자(만원)를 보상으로 받는다. 단 내려오는 길은 바로 아랫 칸, 혹은 그 좌우 한 칸씩으로만 옮겨갈 수 있다. 이렇게 했을 때 가장 많이 받을 수 있는 길을 구하고 그 값을 구하시오.

14	19	8	-12	5	17	2
-23	6	21	5	-15	-8	11
7	-17	-13	9	9	6	-4
-8	6	25	-14	8	16	20
12	-12	9	7	9	-11	14

9.10 실전문제 (스키장에서 장비 대여하기)

스키장에서 장비를 대여해서 즐기려고 한다. 단 장비를 대여할 때 기준은 1일, 3일, 일주일 단위로 빌려준다. 당연하게 길게 빌리면 하루 당 가격은 내려간다. 그런데 스키장에서 일기에 따라서 항상 스키를 타지는 못한다. 다음 표에서 'O'로 표시된 날만 스키를 탈 수 있다. 이때 가장 경제적으로 원하는 날('O')을 모두 타려면 최소의 비용이 얼마인지 계산하시오.

일수	가격
1	2
3	5
7	9

일수	가격
1	3
3	7
5	10

일수	가격
1	2
3	5
6	9

20일 마지막 날에 7일권, 3일권, 1일권을 사는 경우로 각각 나눠서 생각.

d	1	2	3	4	5	6	7	8	9
		O		**O**	**O**	**O**			**O**
C									

d	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		O		**O**	**O**	**O**			**O**	**O**		**O**		**O**		**O**
C																

d	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		O		**O**	**O**	**O**			**O**	**O**		**O**		**O**		**O**			**O**	**O**
C																				

d	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		O	**O**		**O**	**O**		**O**	**O**	**O**			**O**	**O**		**O**	**O**			**O**
C																				

d	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	O	**O**			**O**	**O**	**O**	**O**		**O**		**O**		**O**		**O**		**O**		**O**
C																				

Lecture 10 : Dynamic Programming의 실전응용

이 장에서는 현실에서 나타나는 전형적인 dynamic programming 문제를 살펴본다. 그리고 코딩 대회나 입사 코딩 test에서는 제시된 문제의 다양한 변형이 문제로 제시되기 때문에 데이터나 설명이 비슷하다고 해서 "옳타꾸나!"하며 이전에 사용하여 "재미를 본" 방식으로 기계적으로 적용하면 안된다. 이러한 함정은 출제자들이 노리는 전형적인 것이다.

10.1 동적계획법 알고리즘의 준비물: Review

- 동적 프로그래밍 계획 구성 식(Dynamic Programming formula);
  이 식을 만드는 일이 제일 중요하다.
- 기저 조건(Base condition)을 손으로 확인해야 한다.
  반드시 손으로 꼼꼼하게 확인해야 나중에 비극을 막을 수 있다.
- 계산 결과를 저장할 수 있는 Table 때로는 2차원 3차원

10.2 동적계획법은 각종 프로그래밍 대회에 단골로 나오는 문제 유형이다.

- 계산 시간이 빠르다. (문제를 DP로 풀 수 있다는 사실을 알면)
- 그러나 이 문제가 동적계획으로 해결되는지 미리 알기는 어렵다.
  (많은 경험이 필요하다.)

10.3 동적계획법 알고리즘들의 시간복잡도, 공간복잡도 분석

- 공간복잡도 = Table의 크기

- 시간 복잡도 = Table의 전체 entry를 채우는데 걸리는 시간
  - 각 Entry를 채우는데 걸리는 시간 × 전체 cell의 갯수

10.5 0/1 Knapsack을 동적계획법으로 해결하기 (완전히 선택하거나 버리거나) (pseudo-polynomial time algorithm)[5)]

6개의 항목이 있다. 각각은 C_1=2, C_2=5, C_3=11, C_4=1$C_5$3, =19, C_6=31 이다. 다음 Table[i][K]는 1, 또는 0을 가지는데 1의 의미는 해당 항목을 선택(포함해서 집어넣는 것을 의미한다.) 0은 사용하지 않는 것을 말한다. C_1.. C_k 의 원소의 일부분을 이용해서 그 합으로 주어진 수 K를 만들 수 있음을 나타내보자.

5) 이 개념은 고차원적이긴 하지만(대학원 수준) 이것을 이해하는 것은 매우 중요하다.

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...	...	31	32	33	34
2																				
5																				
11																				
13																				
19																				
31																				

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...	...	31	32	33	34
2																				
5																				
11																				
13																				
19																				
31																				

예) Table[23][3] = 1 // 숫자 23을 제시된 자료 a_1, a_2, a_3 만을 이용해서 만들 수 있다. dynamic programming 식을 만들어 보자.

Table[K][i] = // 모든 Table[1-.(K-1)][n]와 Table[K][1-(i-1)]을 알고 있다고 가정하고.

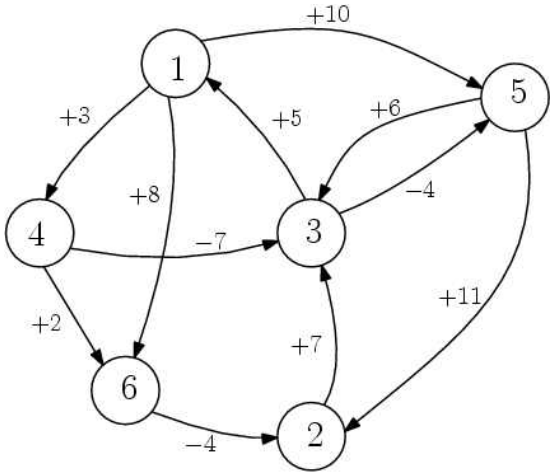
10.4 동적계획법 알고리즘의 예: 최단경로 Floyd–Warshall algorithm

- $W[i][j]$ 는 v_i 와 v_j 를 연결하는 edge의 weight를 나타낸다.
- $D^{(k)}[i][j]$ 는 $v_1, v_2, \ldots v_k$ 까지만의 vertex를 이용해서 v_i에서 v_j 에 가능 가장 짧은 길이는 다음의 식으로 구할 수 있다.

$$D^{(k)}[i][j] = \min(D^{(k-1)}[i][j],\ D^{(k-1)}[i][k] + D^{(k-1)}[k][i])$$

경우 A 경우 B

경우 A: vertex k를 거치지 않고 가는 경우
경우 B: vertex k를 반드시 거쳐서 가는 경우



	1	2	3	4	5	6
1	**0**	∞		**3**	**10**	**8**
2	∞	**0**	**+7**			
3	**+5**		**0**		**-4**	
4			**-7**	**0**		**+2**
5		**11**	**+6**		**0**	
6		**-4**				**0**

$D^{(0)}[i][j]$

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

$D^{(1)}[i][j]$

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

$D^{(2)}[i][j]$

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

$D^{(3)}[i][j]$

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

$D^{(4)}[i][j]$

10.5 동적계획법과 게임(Game)

- 게임의 종류
  - a) 완전 정보 게임(complete information game): 모든 정보가 공개
    { 바둑, 오목, 'IQ puzzle' }
  - b) 확률 게임: { 고스톱, 포커, 야구, "스타_크래프트" }

- 완전 정보게임은 이미 승부가 정해져 있다. 단지 모를 뿐
- Game Tree를 구성해서 해결하는 것이 일반적인 해법

10.6 바둑돌 게임의 예

a) 초기상태 항아리 안에 바둑돌 K개가 담겨 있다.
b) 두 사람, 선수(first mover), 후수(second mover)을 번갈아 가면서 "수"를 둔다.
c) 승패를 결정하는 방법은 명확하며 반드시 결정적(deterministic)이다.
d) 무승부는 나올 수 없으며 일정한 단계 이후에는 반드시 승부가 난다.

예 1) GAME 1

초기) 바둑돌이 K개 있다.
동작) 자신의 차례에 1개, 2개, 혹은 3개의 돌을 가져갈 수 있다.
승부) 자신의 차례에 정해진 동작을 할 수 없으면, 그 사람이 패자(loser)가 된다.

K	1	2	3	4	5	6	7	8	9	10	11	12
Winner	F	F	F									

예 2) GAME 2

초기) 바둑돌이 K개 있다.
동작) 자신의 차례에 이 한번에 1개, 3개, 4개의 돌을 가져갈 수 있다.
승부) 자신의 차례에 동작을 할 수 없으면 그 사람이 패자(loser)가 된다.

K	1	2	3	4	5	6	7	8	9	10	11	12
Winner	F	S	F	F								

예 3) GAME 3

초기) 바둑돌이 K개 있다.
동작) 자신의 차례에 이 한번에 2개, 3개, 5개의 돌을 가져갈 수 있다.

K	1	2	3	4	5	6	7	8	9	10	11	12
Winner	S	F	F	F	F							

예 4) GAME 4

초기) 바둑돌이 K개 있다.

동작) 자신의 차례에 이 한번에 1개, 3개, 4개의 돌을 가져갈 수 있다.

단 앞 사람이 가져간 만큼 다음 사람이 가져갈 수 없다 (반복금지원칙).

만일 앞 사람이 3개를 가져가면 그 다음 사람은 1, 4개만 가져갈 수 있다.

K	1	2	3	4	5	6	7	8	9	10	11	12
1												
3												
4												
종합	F	F	F	F								

예 5) GAME 5

초기) 흰색 바둑돌이 W개, 검은색이 B개 있다.

동작) 자신의 차례에 가져갈 수 있는 돌의 개수는 다음과 같다.

	W	B
경우1	1	1
경우2	3	2
경우3	2	0

W/K	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	S	S	S	S	S	S	S	S	S	S
1	S	F	F							
2	F									
3	S		F							
4	S									
5	S									
6	S									
7	S									
8	S									
9	S									