

DP 풀기위한 4가지 STEP

1. 문제가 DP로 쓰면 간단해지는지 확인하기
2. 최소 매개변수는 이용해 어떤 상태를 표현할지 정하기
3. 상태 관계를 수식화하기.
4. 도표 작성 (tabulation) 하기 / 메모리제이션 (Memorization) 하기

STEP 1. 문제 구별하기

- 일반적으로 특정 수량의 최대, 최소 구하는 문제들.
- 특정 조건을 만족하는 배열 찾기를 하는 문제
- 특정 확률문제 (동전 고환 문제 등)

* 모든 DP 문제들은 충분되는 정보를 이용해 성질을 만족합니다.
(피보나치 수를 구할 때 트리에서 충분되는 항목을 생략)

STEP 2. 상태(State) 정하기

- DP 문제들은 상태과 이전 (transition)에 대한 문제.
- $S_{\text{state}} = \text{주어진 문제에서 특정 위치나 현 상태를 나타낼 수 있는 것들을 개별적으로 (uniquely) 표현 할 수 있는 어떤 특성} \text{ 있는 매개변수}$
- 같은 조건을 위해서 이 매개변수들은 최대한 중복을 피해야 한다
- ex) Knapsack 문제
상태 : 2개의 매개변수 사용 (index, weight)
i.e. $DP[\text{index}][\text{weight}]$
 $DP[i][w]$ 는 w kg 0부터 index 까지의 물건을
갖출 때 가방에 차게 되는 무게 weight 중 가장
최대의 가치를 얻을 수 있는 Value 가 아닌 경우 얻지
않았다는.

weak_happiness

파이썬 수열 - 2계 정화식.

- Memoization Technique 사용하여 더 효율적인 코드 작성 가능; 여기서는 먼저 계산한 fib(x) 값 저장하기

1 1 2 3 5 8 13

```
int fib (int n) {
    if (n == 1 || n == 2)
        return 1
```

```
else if (f[n] > -1)
    return f[n]
```

Cache							
1	1	2	3	5	8	13	24
f[1]	f[2]	f[3]	f[4]	f[5]	f[6]	f[7]	f[8]

① Bottom up 방식 계산 (중복 계산 피한다)

1 1 2 3 5 8 13 24 ...
→

② 이항 계수 계산하기 - 14:00

$$\alpha_{ik} = \begin{pmatrix} a \\ b \end{pmatrix} \quad \text{if } a \leq k \text{ or } b \leq i;$$

$$\binom{n}{k} = \begin{cases} 1 & (\binom{n-1}{k}) + (\binom{n-1}{k-1}) \text{ otherwise} \end{cases}$$

int fib (int n)

$$f[1] = f[2] = 1$$

for (int i = 3; i <= n; i++)

$$f[i] = f[i-1] + f[i-2];$$

return f[n];

어떤 f[i]를 계산
하는 시점에
그 때 (f[i-1], f[i-2])
는 이미 계산되어 있다

- Loop 사용하여 가장 작은 문제

배낭 문제

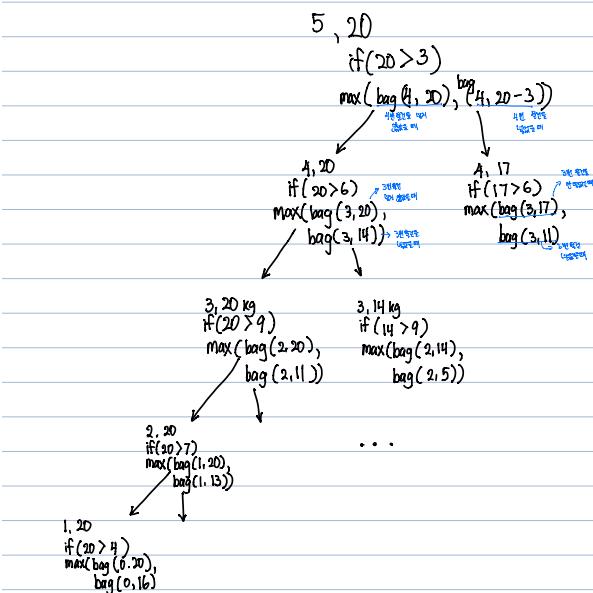
- 배낭에 넣을 수 있는 물건의 가치? 단 배낭의 흡재 용량 제한 (20kg)

번호	무게 (kg)	가치	$1 \cdot 3 \cdot 4 \Rightarrow 120$ 가치, 19 kg
1	4	30	
2	7	25	
3	9	50	
4	6	40	
5	3	10	

Recursive Way

```
int bag (현재 물건의 번호, 배낭의 남은 무게)
if (현재 물건에 들어갈 수 있으면)
    return max (bag (물건을 넣었을 때) + 현재 물건의 가치,
                bag (물건을 안 넣었을 때))
else (현재 물건이 들어갈 수 없으면)
    return bag (다음 물건)
```

↓



```
int weights [5] // 5개 물건의 무게
int value [5] // 5개 물건의 가치
// 물건을 선택할 수 있는 경우
int bag (int index, int leftbag)
    if (index > 4, || leftbag <= 0) ] 종료 조건, index 가 0 일 때 leftbag (선택하지 않은 경우)가 0 이면
        return 0
    if (leftbag > weights [index]) // 만약 남은 가치가 물건의 가치보다 크다면
        return max (bag (index-1, leftbag), bag (index-1, leftbag - weights [index])) // 선택하지 않은 경우
    else
        return bag (index-1, leftbag) // 남은 가치가 물건의 가치보다 크다면
```

⇒ 이와 같은 재귀 방식은 탐색을 험악화하는데 최대 2^n 가지로 해야 한다

∴ Time Complexity = $O(2^n)$

⇒ Memoization 기법으로 시간 줄일 수 있다

$m [index]$ [방문 여부]

<Dynamic Programming>

int bag (int i, int bag_weight)

if ($i < 0$ || $bag_weight < 0$)

return 0

if ($m[i][bag_weight] != -1$)

return $m[i][bag_weight]$

if ($bag_weight > weight[i]$) // 물건이 물건에 대해서 고려해 보았을 때 남은 무게

$m[i][bag_weight] = \max (bag (i-1, bag_weight), bag (i-1, bag_weight - weight[i]) + value[i])$

} else

$m[i][bag_weight] = bag (i-1, bag_weight)$

} return $m[0][bag_weight]$

Tip

* 가방문제와 유사한 문제들

- 배에 실을 차운 무게 배분하기

- 천을 재단할 때, 낭비되는 천의 양 최소화하기

- 강의실 편성할 때

⇒ 자원이 일정한 때 그 자원을 어디에 배치시켜야 하는지 결정하기 등...

$$m[i][j] = \begin{cases} 0 & \text{if } j < 0 \text{ or } i < 0 \\ \max (m[i-1][j], m[i-1][j - weight[i]] + value[i]) & \text{if } j \geq 0 \text{ and } i \geq 0 \\ m[i-1][j] & \text{if } j < weight[i] \end{cases}$$

Recurrence

〈 배낭문제 ② - Unbounded knapsack problem 〉

물건의 갯수 제한이 없는 배낭 문제

문제) 배낭에 담을 수 있는 무게 = 8 kg

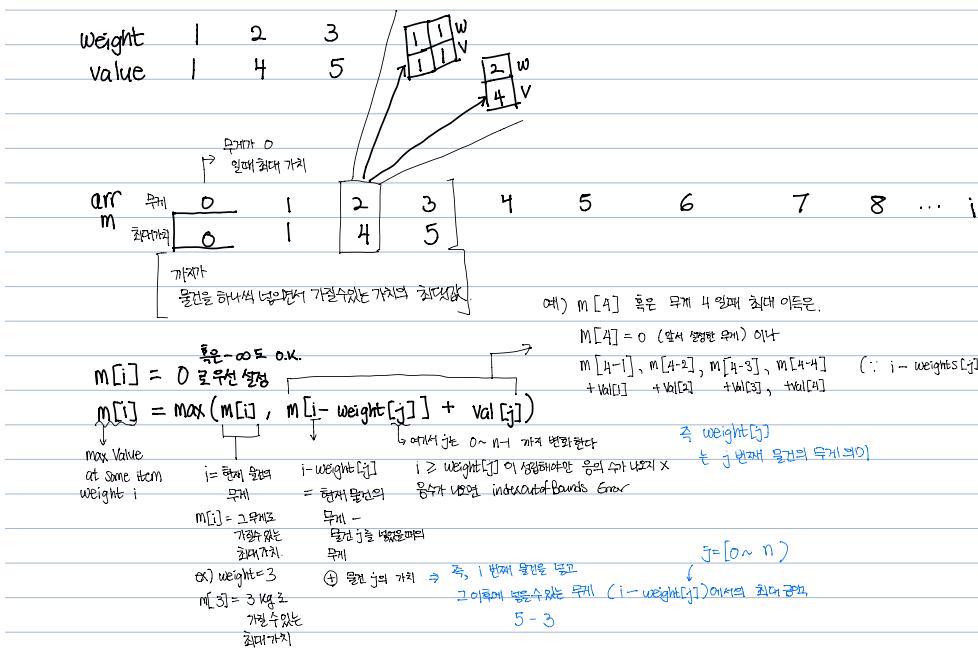
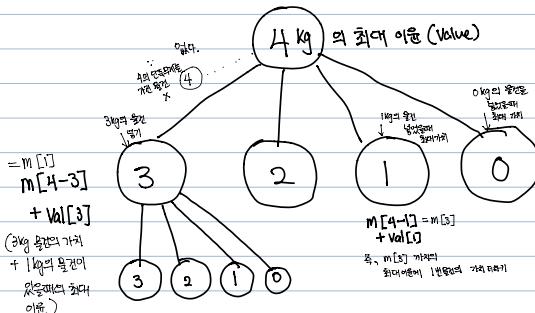
물건은 다음 조건과 같다. 이때 담을 수 있는 물건의 최대 가치는?

물건	무게	가치
1	1	1
2	2	1
3	3	5

Variable은 이 흔나부에 같다. 물건의 개수가 iteration 단위에 바뀌지 않는다.

- 1 차원 배열 사용 (원래의 original knapsack 문제에서는 2d 배열 사용 : 물건의 갯수는 반복되 않는다)
 - 단 배열의 크기는 배낭 무게 + 1 (\because 배낭무게와 0 까지 무게로 가능할 수 있다)

- Repetition == 물건의 종류의 갯수 (all possible options of k)



$m[\text{weight of Bag}]$ = 가방에 담을 수 있는 물건의
최대가치

<소스코드>

```

int m = new int [weight+1]           // 0부터 가방용량까지.
for (int i=0 ; i < m.length ; i++) {
    m[i] = 0                         // initializing array elements to 0.
}

for (int i=0 ; i < weight ; i++) {
    for (int j=0 ; j < Val.length ; j++) {
        if (weight[j] <= i)          // - index (원본 index) 를 증가
            m[i] = max (m[i], m[i-weight[j]] + Val[j]) // 이전 가능한 모든 경우의 합과 비교 후 최대.
    }
}
return m[weight]

```

예

i	v_i	w_i	
1	1	1	
2	6	2	$OPT(i, w) = \begin{cases} 0 & \text{if } i=0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{OPT(i-1, w), v_i + OPT(i-1, w-w_i)\} & \text{otherwise} \end{cases}$
3	18	5	
4	22	6	
5	28	7	

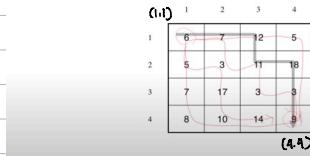
w	0	1	2	3	4	5	6	7	8	9	10	11
(1)	0	0	0	0	0	0	0	0	0	0	0	0
(1, 1)	0	1	1	1	1	1	1	1	1	1	1	1
(1, 2)	0	1	6	7	7	7	7	7	7	7	7	7
(1, 2, 3)	0	1	6	7	7	18	19	24	25	25	25	25
(1, 2, 3, 4)	0	1	6	7	7	18	22	24	28	29	29	40
(1, 2, 3, 4, 5)	0	1	6	7	7	18	22	28	29	34	34	40

$OPT(0, w) = \max \text{ profit subset of items } 1, \dots, i \text{ with weight limit } w.$

최단 경로 문제

- 정수들이 저장된 $n \times n$ 행렬의 좌상단에서 우하단까
이나 아래쪽 방향으로만 이동할 수 있다
 - 방문한 칸에 있는 정수들의 합이 최소화되도록 하리

행렬 경로 문제



순환식 세우기

- 일반화 시키기

i, j)에 도달하기 위해서는 $(i, j-1)$ 혹은 $(i-1, j)$ 을 거쳐야 한다.
 또는 $(i, j-1)$ 혹은 $(i-1, j)$ 까지는 최선의 방법으로 이동해야 한다.



$$L[i, j] := \begin{cases} M_{i,j} & \text{if } i=1, j=1 \\ L[i-1, j] + M_{i,j} & \text{if } j > 1 \\ L[i, j-1] + M_{i,j} & \text{if } i > 1 \end{cases}$$

\$\rightarrow\$ 최소합

\$\rightarrow\$ \$O(n^2)\$

\$M_{i,j} = \min(L[i-1, j], L[i, j-1]) + M_{i,j}\$ otherwise

\$\rightarrow\$ 순회하는 방식으로 풀기

② 순환식 계산하기

① Recursion 사용 - 순환식을 recursion 으로 옮긴 것

```

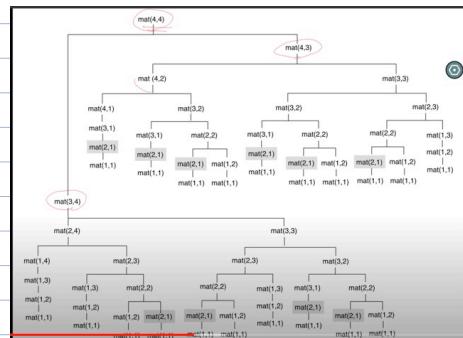
int mat(int i, int j)
{
    if (i==1 && j==1) // m[1][1]
        return m[1][1]

    else if (i==1) // 1 row 일 때
        return mat(1, j-1) + m[1][j] // 행을 하나씩 줄이고 칸을 따라 계산

    else if (j==1) // 1 column 일 때
        return mat(i-1, 1) + m[i][1] // column == 1 일 때 재귀하여
                                         // (i-1, 1)의 최소값 찾은 뒤 본인 값이 더하기

    else // otherwise
        return min(mat(i-1, j)), mat(i, j-1)) + m[i][j]
}

```



② Recursion + Memoization

```

int mat (int i, int j) {
    if (L[i][j] != -1) return L[i][j];
    if (i == 1 && j == 1)
        L[i][j] = m[i][j];
    else if (i == 1)
        L[i][j] = mat(1, j-1) + m[i][j];
    else
        L[i][j] = mat(i-1, j) + mat(i, j-1);
}

```

③ Bottom up 방식 - 개별 방식의 반대

- 특정 순서가 따로 있는 것이 아니라.

$$x_1 = \text{_____}$$

$$L(i,j) = \min(L(i-1, j), L(i, j-1)) + m_{ij}$$

else if ($g == 1$)
 $L[i][j] = \text{mat}(i-1, 1) + m[i][j]$
 else
 $L[i][j] = \min(\text{mat}(i-1, j), \text{mat}(i, j-1)) + m[i][j]$
 Return $L[i][j]$ // Recursion 끝나면 ③ 짧은 경우 ④ 긴 경우

m	6	7	12	5
$i = 2$	5	3	$14, \frac{1}{4}$	18
$i = 3$	7	17	3	3
$i = 4$	8	10	14	9
	$j=1$	$j=2$	$j=3$	$j=4$

1	2	3	4
4	19	25	20
1	11	16	3
31	28	31	
26	36	42	40

int mat() {
 for (int i=1; i <= n; i++) // Rows
 for (int j=1; j <= n; j++) // n columns
 if (i == 1 && j == 1)
 L[i][j] = m[i][0];
 else if (i == 1)
 L[i][j] = m[0][j] + L[i][j-1];
 else if (j == 1)
 L[0][j] = L[0][j-1] + m[i][j];
 else
 L[0][j] = L[0][j-1] + min(L[i-1][j], L[i-1][j-1]);
 return L[0][n];
 }

$O(n^2)$
 시간 복잡도 = $O(n^2)$

이 문제는 행렬 곱셈 문제에서
 $L[0][0]$ 은 $L[0][1]$, $L[0][0] = \infty$ for all

3. 3x3
(0,0) 1 2 3
(0,1) 2 3 4
(0,2) 3 4 5
(1,0) 1 2 3
(1,1) 2 3 4
(1,2) 3 4 5
(2,0) 1 2 3
(2,1) 2 3 4
(2,2) 3 4 5

〈최단 경로 구하기〉

경로 구하기

m	6	7	12	5
	5	3	11	18
	7	17	3	3
	8	10	14	9

L	1	2	3	4
i=1	1	6	13	25 (30)
	2	11	14	43 (3)
	3	18	31 (28)	31
	4	26	36 (42)	40

$\uparrow = 1$

$\leftarrow = 2$

	-	\leftarrow	\leftarrow	\leftarrow
	\uparrow	\leftarrow	\leftarrow	\leftarrow
	\uparrow	\uparrow	\uparrow	\leftarrow
	\uparrow	\leftarrow	\uparrow	\uparrow

2층은 3이에 있는
앞으로 끌려온다
 하늘로 떠나가면
축구장으로는 찾을 수가 нет
 (21:09)

NX N 행렬

int Mat() {

 for (int i = 0; i < n; i++) {

 for (int j = 0; j < n; j++) {

 if (i == 1 && j == 1) {

 L[i][j] = m[0][0];

 P[i][j] = '-';

 } else if (L[i-1][j] < L[i][j-1]) {

 L[i][j] = m[i][j] + L[i-1][j];

 P[i][j] = 'N';

 } else {

 L[i][j] = m[i][j] + L[i][j-1];

 P[i][j] = 'L';

 }

 }

 }

 return P[n][n];

⑩ 예상 결제 출금 - ① 목적지에서 출금으로 인쇄

Void PrintPath()

int i = n - i;

```
while (pr[pr-1] == '-')
```

D의 경로 축복 - ② 축복지에서 물적지 까지

Recursive 한 번째 .

Void PrintRecursivelyStartToDest(i, i)

```

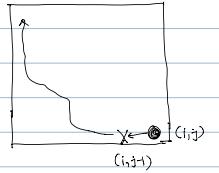
print(i + " " + j)
if(p[i][j] == '<')
    j = j - 1
else if(p[i][j] == '>')
    i = i - 1
}
print(i + " " + j)

```

```

if (p[i][j] == '<')
    print(i + " " + j)
else
    if (p[i][j] == '>') // p[i][j] = '>
        printRSTD(i, j - 1) ←
    else
        printRSTD(i - 1, j)
        print(i + " " + j)

```



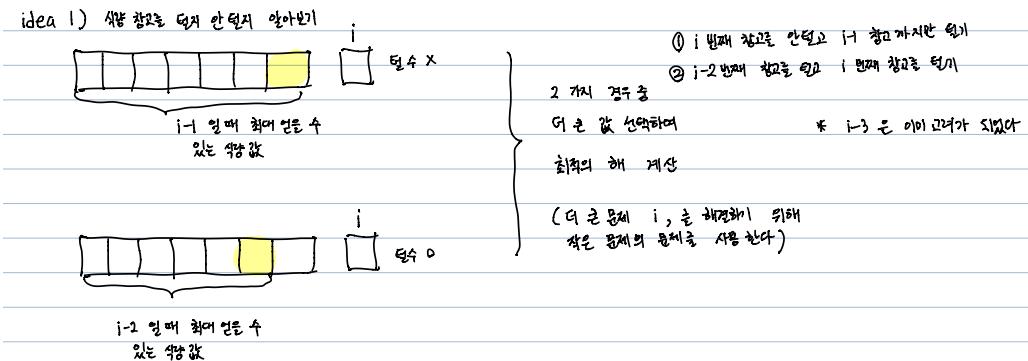
<개미 전사>

식堂 칭고 N 개 중 일정 수 있는 식당의 최댓값?

단, 연속해서 있는 칭고는 갈 수 없다

칭고 0	칭고 1	칭고 2	칭고 3	칭고 4	a ₀	a ₁	a ₂	a ₃
1	3	1	5	—	1	3	3	8

최적의 해를 계산하여
DP 테이블로 나타낸다면



정화식 표현

$$a_i = i \text{ 번째 식당 칭고까지 최적의 해}$$

$$k_i = i \text{ 번째 식당 칭고에 있는 식당의 양}$$

$$a_i = \max(a_{i-1}, a_{i-2} + k_i)$$

* 한칸 이상 떨어진 식당 칭고는 항상 텰 수 있으므로

(i-3) 번째 이하는 고려해 필요 X

<여러가지 경우의 경우의 경우 계산>

정수 x에 사용할 수 있는 연산 = 4개

- if $x \% 5 == 0$, 5로 나눈다
- if $x \% 3 == 0$, 3으로 나눈다
- if $x \% 2 == 0$, 2로 나눈다

정수 x가 주어졌을 때

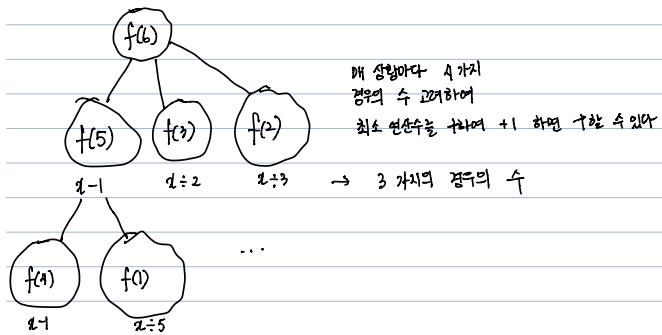
- 연산 4개를 사용하여 1을 만들기.
- 연산을 최대 사용할 수 있는 방법 찾기

$$\text{ex) } \# x = 26$$

otherwise $x-1$

$$26 \Rightarrow 25 \Rightarrow 5 \Rightarrow 1 ; 3 \text{ 번의 연산}$$

- 트리로 표현하면



<정화식>

$$a_i = i \text{ 를 } 1\text{ }2\text{ }3\text{ }5 \text{ 만들기 위한 최소 연산 횟수}$$

$$a_i = \min(a_{i-1}, a_{i/2}, a_{i/3}, a_{i/5}) + 1$$

$$a_{i-1}$$

$$a_{i/2}$$

$$a_{i/3}$$

$$a_{i/5}$$

$$\text{optimal solution}$$

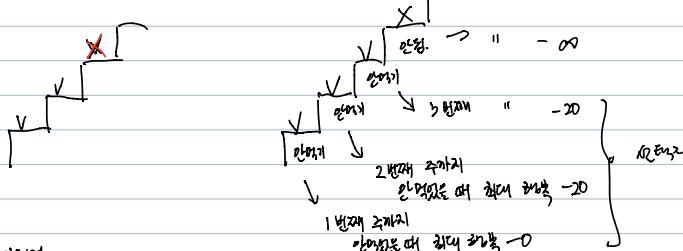
$$a_i = \begin{cases} \min(a_i, a_{i/2} + 1) & \text{if } i \% 2 == 0 \\ \min(a_i, a_{i/3} + 1) & \text{if } i \% 3 == 0 \\ \min(a_i, a_{i/5} + 1) & \text{if } i \% 5 == 0 \\ a_i - 1 + 1 & \text{otherwise} \end{cases}$$

제일 작은 수를 선택하는 것은 제일 좋은 계산

<계단 놓기 문제>

얻을 수 있는 정수의 최댓값 구하기

1. 계단은 후행에 허여 2개를 놓을 수 있다
2. 연속된 3개의 계단 모두 놓기 X
3. 마지막 놓은 계단은 반드시 놓아야 한다



DP의 조건 만족

① Overlapping Subproblem

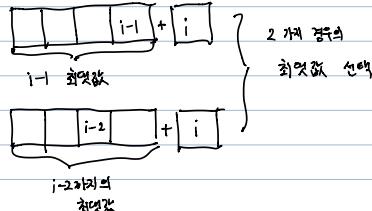
i 번째 계단까지 가려면

$\rightarrow i-1$ or $i-2$ 번째 계단 놓아야 함

② Optimal Substructure

i 번째 까지의 최댓값?

$\rightarrow i-1$ 번째 or $i-2$ 번째 계단까지의 최댓값
+ i 번째 계단 횟수



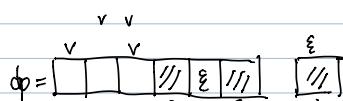
③ 2차원 배열

$dp[i][0]$: $i-1$ 번째 계단을 놓지 않고 i 번째 까지 계단까지의 최댓값

$dp[i][1]$: $i-1$ 번째 계단을 놓은 i 번째 계단까지의 최댓값

④ 1 차원 배열

$$\left\{ \begin{array}{l} dp[0] = arr[0] \\ dp[1] = \max(arr[0] + arr[1], arr[1]) \\ dp[2] = \max(arr[0] + arr[2], arr[1] + arr[2]) \end{array} \right.$$



$$arr = [2, 5, 6, 7, 8, 4]$$

$$dp[i] = \max(dp[i-2] + arr[i], dp[i-3] + arr[i-1] + arr[i])$$

$$\frac{2+5, 5}{2+6, 5+6}$$



<정화식 사용법>

<증가부분수열>

- Sequence 속의 가장 긴 증가하는 부분 수열.

- ex) 2 7 10 5 20 30 50 \rightarrow array

Sub 1 : {2, 7}

Sub 2 : {2, 7, 10}

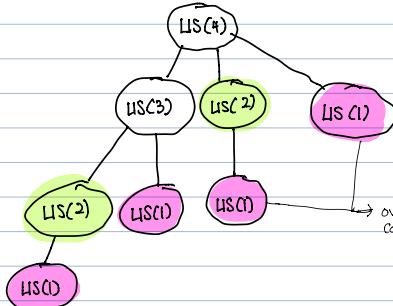
Sub 3 : {5, 20, 30}

Sub 4 : {5, 20, 30, 40} ✓

- 증가하는 부분 수열에서
- * 같은 원소는 연속으로 있음.
- 길이가 가장 긴 부분 수열
- 증가. (증가가 가장 많다)
- 포함된 Subsequence)

\Rightarrow Why DP?

1) Overlapping substructure property



Approach.

- 기존의 배열 array에 주어진: Length, Subsequence 배열 추가

- Base condition: 각각의 원소는 각자 자신을 최장 증가 부분 수열로 가지고 있다.

(Length 배열을 1로 초기화 한다.)

Initialize HS Value

Base Case: i=1 setting arr[0] = arr[i]

For i=1 (i는 두번의 loop. j=0에서부터 시작하여 j+1까지 증가하는 수를 찾는다.)

arr[] 0 4 12 2 10 6 9 13 3 11 7 15

length 1 2 2 1 1 1 1 1 1 1 1 1
3

Subsequence
↳ stores the previous element to be HS

i=2. j=0 ~ j=1

arr[] 0 4 12 2 10 6 9 13 3 11 7 15

length 1 2 1 \rightarrow 2
3
Subsequence X 0 1

i=3. j=0 ~ j=2

arr[] 0 4 12 2 10 6 9 13 3 11 7 15

length 1 2 3 2 1

Subsequence X 0 1 0
↳ last index that satisfies arr[i] > arr[j].
j이 x.
i.e. (4, 2) \rightarrow (12, 2)

i=4. j=0 ~ j=3

arr[] 0 4 12 2 10 6 9 13 3 11 7 15

length 1 2 3 2 1

Subsequence X 0 1 0 0
 $0 \rightarrow 1 \rightarrow 3$
(증가하는)
(1은 3을 찾았지만 12를 찾는 x)

$i=5 \quad j=0 \sim j=4$

arr [] 0 4 12 2 10 6 9 13 3 11 7 15

Length 1 2 3 2 1
 $| \rightarrow 2 (i+1)$ $\text{arr}[5] > \text{arr}[4]$
 $3 (i+2)$ $\Rightarrow X \text{ include}$

Subsequence X 0 1 0 0
 $0 \rightarrow 1 \rightarrow 3$ $(\text{arr}[0] < \text{arr}[1] & \text{arr}[1] < \text{arr}[2])$

Continue until $i=11$, $j=10$

arr [] 0 4 12 2 1 6 9 13 3 11 7 15

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

Subsequence X 0 1 0 3 3 5 6 3 6 8 9
Y
 $\hookrightarrow \text{Subsequence}[i] = 9$
 $\text{Subsequence}[i] \geq 4 \text{ th}$.

$[0, 3, 5, 6, 9] \Rightarrow \text{array의 원소 번호.}$

0, 2, 4, 9, 11 이 해당하는 원소 번호.

<가장 긴 바이토닉 수열 - LIS>

- 바이토닉 수열.

$$s_1 < s_2 < s_3 < \dots < s_k > s_{k+1} > \dots > s_n$$

- s_k 기준으로 증가했다가 감소하는 수열

Approach.

⇒ 나누어서 생각해보기!

1) $s_1 < s_2 < s_3 < \dots < s_k \Rightarrow LIS$ 와 같다

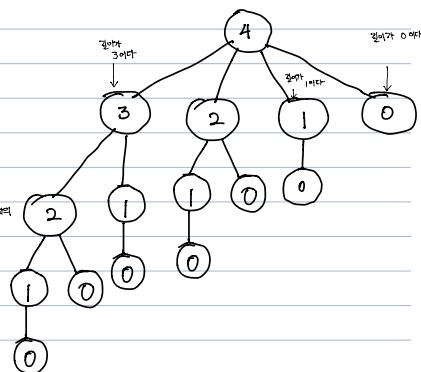
2) $s_k > s_{k+1} > \dots > s_n \Rightarrow$ n 번째 생각하면 LIS와 같다

⇒ 모든 s_i 로 끝나는 최장 증가 수열을 구해주고 전체 수열을 두번에서 한번 더 하면 OK.

① Recursion 사용한 풀이

```
Cutrod(p, n)
if n == 0
    return 0
q = -∞ // 가능한 최대의 이익은 -∞으로 일단 설정한다.
for i = 1 to n
    q = max(q, p[i] + Cutrod(p, n-i)) // 최대 되는 이득 선택.
    현재의 이득이 아니면 현재 끝번의 이득 + 현재 조각에서 더 최대선택
    하는가?
return q
```

$n=4$ 일 때,



② Memoization + Recursion

```
let r[0..n] be new array
for i = 0 to n
    r[i] = -∞
return memoized-Cut-Rod-Max

Memoized-Cut-Rod-Max(p, n, r) // Recursion method 와 같다
if r[n] ≥ 0
    return r[n]
if n == 0
    q = 0
else q = -∞
for i = 1 to n
    q = max(q, p[i] + Memoized-Cut-Rod-Max(p, n-i, r))
r[n] = q
return q
```

→ 배열 r의 구성

- 특정 길이 n' 에 대한 최적의 해를 $r[n']$ 에 저장
- 특정 길이 n' 에 대하여 $r[n']$ 의 값이 ≥ 0 이면 값을 update 된다

예시)	길이	1	2	3	4
	p_j	/	5	8	9

③ Bottom up approach

BUCutRod(p, n)

```
let r[0..n] new array // 초기화
r[0] = 0 // 길이가 0인 막대는 가치가 0
for j=1 to n
    q = -∞
    for i=1 to j // i의 부분문제 쪼개는 j의 상용을 해마다
        q = max(q, p[i] + r[j-i]) // 각자
    r[j] = q // 현재 막대의 최댓값
    r[j] = q // 현재 막대의 최댓값
    r[j] = q // 현재 막대의 최댓값
return q
```

Ex) $n=4$

j	1	2	3	4
i	1	2	3	4

(길이)

$$\begin{aligned} \text{전체문제} & \quad \text{부분문제} & q = \\ j=1, i=1 & \quad \max(-\infty, p[1]+r[0]) & \\ \max(-\infty, 1+0) & & \\ j=2, i=1 & \quad \max(1, p[2]+r[1]) & \\ j=3, i=1, 2, 3 & & \end{aligned}$$

⇒ 나는 불규칙의 길이가 1이다면

1밖에 조개질 수 밖에 없고

(1로 길이가 정해졌을 때,

나눌수있는 최대 횟수는 자기자신 밖에

없다)

$j=3, i=2$ 일 때
전체 길이는 3이지만
부분의 길이는 2.



⇒ 나는 불규칙의 길이가 2이면

그것을 1x2 개나

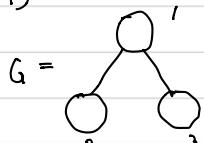
2x1 개로 더 나눌수있다

<정점커버란?>

- 주어진 그래프 $G = (V, E)$ 에서 각 선분 중 양쪽의 끝점
을 중 적어도 한 곳정을 포함하는 점들의 집합 중에서 최소의
집합 찾는 것

- 그래프의 모든 선분들이 집합 내 어느 하나라도 연결이 되어 있어야
한다

예시)



정점 커버는 다음과 같다
 $\{1, 2, 3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{3, 1\}$

* 정점 $\{2\}, \{3\}$ 은 정점 커버가 아니다
 \because 선분 $(1, 3), (1, 2)$ 를 커버하지
못하기 때문이다

\Rightarrow 정점 커버 문제는 최소 크기의 집합을

찾는 것으로 해 = 1

<접근법>

① 주어진 그래프의 모든 선분을 커버하기 위해서는 어떤 점을
선택해야 하는지 고려해야 한다

i) 차수가 높은 점을 선택하면 많은 수의 선분이 커버될 수 있다

ref. 정점 커버 문제의 근사 알고리즘에서 사용 된 것.

이때의 근사 비율은 $\log(n)$ 이다

ii) 점 선택 X 선분 선택 O

- 선분을 선택하면 선택된 선분의 양 끝점을 연결한 선이
모두 커버된다

- 이때의 정점 커버는 선택된 각 선분의 양 끝점들로
이루어진 집합이다

Study Title _____ Date _____

TREE AND DYNAMIC PROGRAMMING

- 트리도 graph의 일종

- 서로 다른 두 노드 있는 길이 하나만 존재
- X cycle
- 그래프처럼 DP 적용 가능.

<https://chanhuiseok.github.io/posts/algo-56/>

- 트리에서 DP 구현하기

a) 어떤 특정 i 번째 노드를 root로 하는 Subtree에 대해서

- i 번째 root node 포함할 때와 } 조건에 있는
- i 번째 root node 포함하지 않을 때 } 경우의

- 방향 (Top down vs Bottom up)

- Top down - pre-order
- Memoization

Bottom up - post order traversal problem

b) 트리는 비선형 구조.

- 트리에서 dp를 하기 전에 탐색순서 미리 정해야 한다
- 주로 dfs tree 가 기준이 된다

c) 트리 dp에서도 상태 dp가 등여야 함.

- $dp[i][j] = i$ 를 주트로 하는 Subtree에서
 i 의 상태가 j 일 때 ...

< 예제 1. 15681 트리와 쿼리 >

트리와 쿼리

시간 제한	메모리 제한	점수	정답
1초	128 MB	15/11	8/5

문제

간선에 가중치와 방향성이 없는 임의의 루트 있는 트리가 주어졌을 때, 아래의 쿼리에 답해보도록 하자.

- 정점 N을 루트로 하는 서브트리에 속한 정점의 수를 출력한다.

만약 이 문제를 해결하는 데에 어려움이 있다면, 하단의 힌트에 첨부한 문서를 참고하자.

입력

트리의 정점의 수 N과 루트의 번호 R, 쿼리의 수 Q가 주어진다. ($2 \leq N \leq 10^5$, $1 \leq R \leq N$, $1 \leq Q \leq 10^3$)

이어 N-1줄에 걸쳐, U V의 형태로 트리에 속한 간선의 정보가 주어진다. ($1 \leq U, V \leq N$, $U \neq V$)

이는 UV와 VU 양 끝점으로 하는 간선이 트리에 속함을 의미한다.

이어 Q줄에 걸쳐, 문제에 설명한 U가 하나씩 주어진다. ($1 \leq U \leq N$)

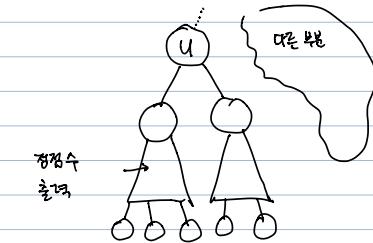
입력으로 주어지는 트리는 항상 유효한 트리임이 보장된다.

출력

Q줄에 걸쳐 각 쿼리의 답을 정수 하나로 출력한다.

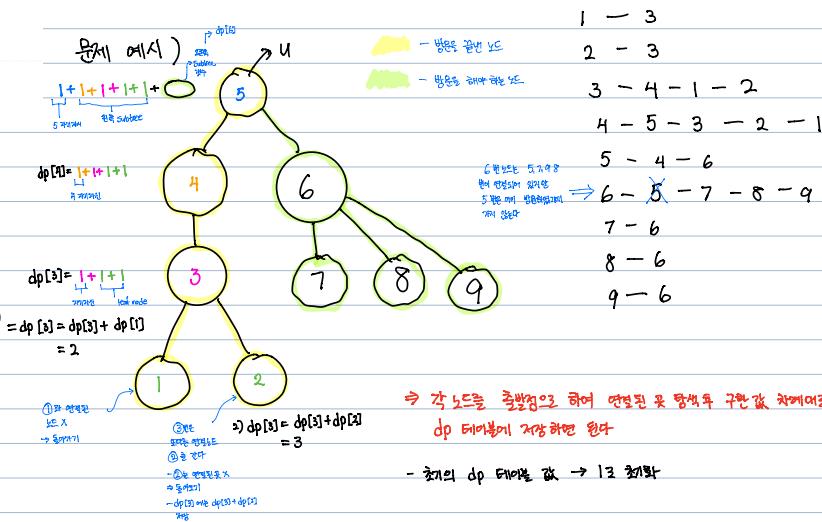
- Root r 값으로 고정

- Tree와 DP를 어떻게 구현하면 좋을지 생각해보기



- DFS 써서 면 마지막 leaf node (base case) 까지 가지 칠까? 아니면 어떤가?

- visit 활용 적용용 X (visit을 초기화하는 X)



< Source Code >

```
int N, R, Q;
int dp[100001];
bool visit[100001];

int dfs(int cur) {
    if (visit[cur]) return dp[cur]; // 이미 방문한 것 재방문 X

    for (nextnode of node[cur])
        if (!visit[nextnode]) continue; // 이미 방문한 것 재방문 X
        dp[cur] = dp[cur] + dfs(nextnode); // 가장자리의 neighboring / child node 브루
    return dp[cur];
}
```

< 예제 . 트리의 둑집집합 - 2213 >

- 일반적인 그래프 X . tree (연결되고 사이클 X)

Tip.

- 가정은 :

- 1) 포함되거나
 - 2) 포함되지 않는
- } 두 가지 상태중 하나.

⇒ $dp[i][0] = i$ 를 주트로 하는 서브트리에서 i 를 포함하지 않을 때
 $dp[i][1] = i$ 를 주트로 하는 서브트리에서 i 를 포함할 때

< 예제 . 티켓 세팅하기 1693 >

Tip

< Stmt >

$dp[i][j] = i$ 를 주트로 하는 subtree에서
 i 를 j 로 칠할 때 정답.

< pseudoCode >

$dp[10000][20]$ // n 번째 정점을 0 색으로 칠했을 때 최소 비용.
ic) 3번의 정점을 그린으로 칠했을 때 최소 비용.

< Time Complexity >

$O(n \log(n))$

```
int OPT (int curr, int color)
res = dp[curr][color] // 현재노드를 color로 칠했을 때의 결과
if (res != -1) // -1 이 아님
    return res // 반환.
```

prev = 0 // 초기값 = 0.

```
each
for child of curr { // 현재노드의 모든 자식에 대해
    temp = infinity
    for every color
        for (i=1 ; i<20 ; i++) {
            if (color == i) continue // 같은 노드에 같은 색깔은 Continue
            temp = min(temp, OPT(child, i)) // 자식 노드에 대한 최소값
        }
    }
    prev += temp // 이전의 값에 temp 더해
}
return res = prev + color // 초기값 + 이전의 합
```

이 예자

이 진주

이 진우

이 예자

이 진주

이 진우

< 예제 2. 우수 마을 19H9번 >

우수 마을

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2 초	128 MB	3242	1401	938	46.85%

문제

N개의 마을로 이루어진 나라가 있다. 편의상 마을에는 1부터 N까지 번호가 붙어 있다고 하자. 이 나라는 트리(Tree) 구조로 이루어져 있다. 즉 마을과 마을 사이에 직접 있는 N-1개의 길이 있으며, 각 길은 방향성이 있어서 A번 마을에서 B번 마을로 갈 수 있다면 B번 마을에서 A번 마을로 갈 수 있다. 또, 모든 마을은 연결되어 있다. 두 마을 사이에 직접 있는 길이 있을 때, 두 마을이 인접해 있다고 한다.

이 나라의 주민들에게 성취감을 높여 주기 위해, 다음 세 가지 조건을 만족하면서 N개의 마을 중 몇 개의 마을을 '우수 마을'로 선정하려고 한다.

- '우수 마을'로 선정된 마을 주민 수의 총 합을 최대로 해야 한다.
- 마을 사이의 충돌을 방지하기 위해서, 만일 두 마을이 인접해 있으면 두 마을을 모두 '우수 마을'로 선정할 수는 없다. 즉 '우수 마을'끼리는 서로 인접해 있을 수 없다.
- 선정되지 못한 마을에 경각심을 불러일으키기 위해서, '우수 마을'로 선정되지 못한 마을은 적어도 하나의 '우수 마을'과는 인접해 있어야 한다.

각 마을 주민 수와 마을 사이의 길에 대한 정보가 주어졌을 때, 주어진 조건을 만족하도록 '우수 마을'을 선정하는 프로그램을 작성하시오.

입력

첫째 줄에 정수 N이 주어진다. ($1 \leq N \leq 10,000$) 둘째 줄에는 마을 주민 수를 나타내는 N개의 자연수가 번갈아 사이에 두고 주어진다. 1번 마을부터 N번 마을까지 순서대로 주어지며, 주민 수는 10,000 이하이다.셋째 줄부터 N-1개 줄에 걸쳐서 인접한 두 마을의 번호가 번갈아 사이에 두고 주어진다.

출력

첫째 줄에 '우수 마을'의 주민 수의 총 합을 출력한다.

- Tip

- 탐색 후 다중과 같이 dp table 생성

// 일반 마을은 자식마을이 ①우수이거나 ②일반마을이다
// 더 문제를 선택하겠다. → 문제의 3번 조건 해결

$$dp[cur][0] = dp[cur][0] + \max(dp[next][0], dp[next][1])$$

현재 마을 인접
마을의 인접
마을의 우수
인접인 마을

$$dp[cur][1] = dp[cur][1] + dp[next][0]$$

현재 마을
인접인 마을
인접인 마을

- 트리구조 구현 없이 트리 탐색을 위해 한번 방문했던 node 재방문 X.

⇒ visit 배열 사용해 관리

⇒ 최종적으로는 root로 선정한 마을의 $dp[cur][0]$ 값과 $dp[cur][1]$ 값 중 더 큰 값이 정답

< Source Code >

```
int dp[100001][2]; bool visit[100001];
```

2개의 값의 수

```
int dfs(int cur)
if (visited[cur]) return dp[cur]; // 이미 방문하였다면 dp에 있는 cur 값 반환
visited[cur] = true;
dp[cur][0] = 0;
if (cur == 1) dp[cur][1] = W[cur];
else dp[cur][1] = W[cur];
```

우수 마을은 우수 마을로 만든 설정한다 (최종화)

```
for (next : node[cur])
if (visit[next]) continue; // 방문하지 않은 노드에서
dfs(next); // 끝까지 탐색 종료
dp[cur][0] = dp[cur][0] + max(dp[next][0], dp[next][1]);
dp[cur][1] = dp[cur][1] + dp[next][0];
```

현재 마을이 우수이면
우수
증 더 큰값 선택 ; 3번 조건 해결
마을이다

// 현재 마을이 우수마을이면 자식마을을 반드시 있는
마을이다

- 마을 하나 = node, 연결관계 = edge

조건 1) 우수 마을로 선정된 주민수 총합 최대

조건 2) 우수마을은 인접 불가

조건 3) 우수마을이 아닌 일반 마을은 적어도 하나의 우수 마을과 인접해야 한다

⇒ 우수마을이 보인 경우 자식에는 우수 마을이 있으면 X

⇒ 일반 마을이 보인 경우 자식에는 우수마을 O

⇒ dp table을 바꾸어 보면

$dp[cur][0] = \text{Cur 마을이 일반일 때}$

Cur 마을을 주로 하는 서브 트리의 우수 마을들의 인접
최댓값

$dp[cur][1] = \text{Cur 마을이 우수일 때}$

Cur 마을을 주로 하는 Subtree의 우수 마을들의
인접 최댓값

- 여기서 시작 node의 순서는 관계 X

...



거짓말이나
우수일 때
(그냥 정수로 생각해라)

$$dp[i][0] = \sum_{j \in \text{neigh}} \max(dp[j][0], dp[j][1])$$

$$dp[i][1] = \text{people}[i] + \sum_{j \in \text{neigh}} dp[j][0]$$

< Source Code - Initialization >

```
for (i=1; i <= N; i++)
```

```
W[i] = input(people)
```

```
}
```

```
:
```

< 헛간 채칠하기 - 15458 >

- 헛간을 3 가지 색으로 채칠 ; 인접한 두 헛간은 항상 다른 색으로 채칠
- K 개 헛간은 이미 채칠되어 있다
- 헛간 그래프와 채칠된 헛간, 색 주어질 때 모든 헛간을 칠하는 경우의 수?

Tip.

- 트리이지만 루트가 따로 X. 모든 경우에 등장하는 1 번 헛간 = root

< Source Code - ① >

```
dp[1000][4] ; color[1000]
```

사용되는 색의 수

1~3개

```
possible (int u, int pc, int p) {
```

```
    ColVal = dp[u][pc]
```

```
    if (ColVal != -1) return ColVal
```

```
    if (Col[u] == 0)
```

```
        ColVal = 0;
```

```
        for (C=1; i<3; i++)
```

```
            if (C==pc) Continue;
```

```
            temp = 1
```

```
            for (int v: adj[u])
```

```
                if (v == p) Continue
```

```
                temp *= possible(v, C, u)
```

```
}
```

```
        ColVal += temp
```

```
    else { // parentcolor != 선택한 경우
```

```
        if (pc == Col[u])
```

```
            return ColVal = 0
```

```
    ColVal = 1
```

```
    for (v: adj[u])
```

```
        if (v == p) Continue
```

```
        ret *= possible(v, Col[u], u)
```

```
}
```

```
return ColVal
```

< Soln >

- 부모 노드에서 칠한색 PC 일 때

- u 정점은 루트로 하는 서브트리를 칠하는 경우의 수

= $dp[u][pc]$ 에서

- u 번 정점이 색이 칠해지지 않은 상태와

색이 칠해진 상태로 나누어 생각하라

1) 색이 X 일 경우

PC 와 다른 색상 C₁, C₂에 대해

$$\prod_{v \in \text{adj}(u)} dp[v][C_1] + \prod_{v \in \text{adj}(u)} dp[v][C_2]$$

상태는 놓쳤을 때 계산 + 이전에는 놓쳤을 때 계산

2) 색이 칠해진 경우

PC 와 색깔이 같다면 불가능. 0 반환

PC 와 색깔이 다르다면

$\prod_{v \in \text{adj}(u)} dp[v][\text{color}[v]]$

