동적계획법
(Dynamic Programming)

- 개요
- 피보나치 수열
- 이항계수
- 계단오르기
- 양의 정수 n의 합 표현 방법 – 나열되는 숫자들의 순서는 고려하지 않음
- 거스름돈 나누어주는 방법
- Weighted Interval Scheduling

동적계획법 개요

- 분할과 정복(Divide and Conquer) 방법은 Top-Down Design으로서 문제를 부분문제(subproblem)들로 나누고 이들 부분문제(subproblem)들의 해를 recursive하게 구하여 이들 해들로부터 원래의 문제의 해를 구한다.
  - 작은 문제들은 독립적이다

- 동적계획법(Dynamic Programming)도 주어진 문제의 해를 부분문제(subproblem)들의 해를 이용하여 구한다.
  - 분할과 정복과는 달리, 주어진 문제의 해를 구하는데 필요한 부분문제들은 서로 독립적이지 않다 (부분문제들이 overlap하는 경우)

동적계획법

- 동적계획법은 주로 최적화 (optimization) 문제를 해결하는 데 주로 사용한다. 동적계획법을 이용한 문제 해결은 아래의 4 단계를 거친다.

단계 1. 최적 해의 구조를 파악한다.

단계 2. 최적 해의 (목적함수) 값을 재귀적으로 (recursively) 정의한다.

단계 3. 단계 2의 재귀적 정의로부터 최적 해의 (목적함)수 값을 bottom-up(혹은 memoization)으로 구하면서 테이블에 저장한다.
(bottom-up: 작은 부분문제에서 시작하여 큰 부분문제들까지 최적 해의 목적함수 값을 차례대로 구한다)

단계 4. 단계 3에서 구한 정보를 이용하여 최적 해를 찾는다.

※ 단계 1-3은 동적계획법으로 해를 구할 때 필수적인 과정임
단계 4는 최적 해의 목적함수 값만 구하는 경우는 필요 없고, 최적 해를 찾고자 하는 경우에 필요.

3. 계단 오르기

3.1 계단 오르기 1(양의 정수 n의 합 표현 방법 (1))

- 계단을 오를 때, 한번에 한 개 혹은 두개의 계단씩 올라 갈 수 있다. N개 계단 아래에서 계단 꼭대기까지 올라가는 방법의 수를 구하시오.
  - 4개 계단을 올라갈 수 있는 방법

    1계단 + 1계단 + 1계단 + 1계단
    1계단 + 1계단 + 2계단
    1계단 + 2계단 + 1계단
    2계단 + 1계단 + 1계단
    2계단 + 2계단

  - C(i) : i개의 계단을 아래에서 꼭대기까지 올라가는 방법의 수

    C(i) = ___ if i = 0
    = 1 if i = 1
    = C(i-1) + C(i-2) if i > 1
    => C(n)을 구하면 된다.

```c
// C
const int MAX = 100;
int count(int n)
{
   int  C[MAX];
   C[0] = C[1] = 1;
   for (int i = 2; i <= n; i++) {
        C[i] = C[i-1] + C[i-2];
    }
    return C[n];
}
```

i	0	1	2	3	4	5	6	..
C[i]	1	1	2	3	5	8	13	..

```python
# 파이썬
def count(n):
    C = [0 for i in range(n+1)]
    C[0] = C[1] = 1
    for i  in range(2,n+1):
        C[i] = C[i-1]+ C[i-2]

    return C[n]
```

수행시간: O(n)

3.2 계단 오르기 2 (양의 정수 n의 합 표현 방법 (2))

- 양의 정수 n을 1, 3, 4의 합으로 나타내는 방법의 수를 구하시오. (1, 3, 4을 사용하는 횟수에 대한 제한은 없고, 합에 나타나는 숫자들의 순서를 고려한다.) => 한번에 한 개, 세 개 혹은 네 개의 계단씩 올라갈 수 있을 경우 계단 문제
  - 예: n = 5인 경우 다음의 6가지가 있다.

    1 + 1 + 1 + 1 + 1
    1 + 1 + 3
    1 + 3 + 1
    3 + 1 + 1
    1 + 4
    4 + 1
  - C(i) : i를 1, 3, 4의 합으로 나타내는 방법의 수

    C(i) = ___ if i = 0
    1 if i = 1, i = 2
    2 if i = 3 // = C[2] + C[0]
    = C(i-1) + C(i-3) + C(i-4) if i > 3

    => C(n)을 구하면 된다.

```
// C
const int MAX = 100;
int count(int n)
{
  int  C[MAX];
  C[0] = C[1] = C[2] = 1;
  C[3] = 2;
  for (int i = 4; i <= n; i++) {
      C[i] = C[i-1] + C[i-3] + C[i-4];
   }
   return C[n];
}
```

i	0	1	2	3	4	5	6	..
C[i]	1	1	1	2	4	6		..

```
# 파이썬
def count(n):
   C = [0 for i in range(n+1)]
   C[0] = C[1] = C[2] = 1
   C[3] = 2
   for i  in range(4,n+1):
      C[i] = C[i-1]+ C[i-3] + C[i-4]

   return C[n]
```

수행시간: O(n)

3.3. 계단 오르기 3 (양의 정수 n의 합 표현 방법 (3))

- 양의 정수 n을 $n_1 < n_2 < \ldots < n_k$의 합으로 나타내는 방법의 수를 구하시오. (각 수를 사용하는 횟수에 대한 제한은 없고, 합에 나타나는 숫자들의 순서를 고려한다.)
  - 예: $n_1 = 1, n_2 = 3, n_3 = 4$

    1 + 1 + 1 + 1 + 1

    1 + 1 + 3

    1 + 3 + 1

    3 + 1 + 1

    1 + 4

    4 + 1

i	0	1	2	3	4	5	6	..
C[i]	1	1	1	2	4	6		..

```
# 파이썬
# numList[i]: ni+1
def count(n, numList):
    C = [0 for i in range(n+1)]
    C[0] = 1
    for i in range(numList[0],n+1):
        for x in numList:
            if i >= x:
                C[i] += C[i - x]
            else:
                break
    return C[n]
```

  - C(i) : i를 $n_1, n_2, \ldots, n_k$ 의 합으로 나타내는 방법의 수

    $C(i) = 1 \quad \text{if } i = 0$

    $\quad\; = 0 \quad \text{if } i < n_1$

    $\quad\; = C(i-n_1) + C(i-n_2) + \ldots + C(i-n_j) \quad \text{if } n_j \le i < n_{j+1} \text{ for } 1 \le j \le k$

    => C(n)을 구하면 된다.

수행시간: O(nk)

- 양의 정수 n을 $n_1 < n_2 < \ldots < n_k$의 합으로 나타내는 방법의 수를 구하시오. (각 수를 사용하는 횟수에 대한 제한은 없고, 합에 나타나는 숫자들의 순서를 고려한다.)
  - 예: $n_1 = 3, n_2 = 5, n_3 = 11$

i	0	1	2	3	4	5	6	7	8	..
C[i]	1	0	0	1	0	1	1	0	2	..

- C(i) : i를 $n_1, n_2, \ldots, n_k$ 의 합으로 나타내는 방법의 수

$$C(i) = 1 \quad \text{if } i = 0$$
$$= 0 \quad \text{if } i < n_1$$
$$= C(i-n_1) + C(i-n_2) + \ldots + C(i-n_j) \quad \text{if } n_j \le i < n_{j+1} \text{ for } 1 \le j \le k$$

=> C(n)을 구하면 된다.

```
# 파이썬
# numList[i]: ni+1
def count(n, numList):
    C = [0 for i in range(n+1)]
    C[0] = 1
    for i in range(numList[0],n+1):
        for x in numList:
            if i >= x:
                C[i] += C[i - x]
            else:
                break
    return C[n]
```

수행시간: O(nk)

3.4 계단 오르기 4

n(1이상 10,000이하 정수)개 계단을 바닥에서 위로 올라가려고 한다. 계단을 올라갈 때 한 번에 1개, 3개 혹은 4개의 계단만 오를 수 있으며, 각 계단은 밟을 때 비용이 있다. 바닥에서 가장 위의 계단으로 올라갈 때 밟는 계단의 비용 합이 최소가 되도록 하면서 올라가고자 한다. 이 최소 비용을 구하시오.

예) n = 6

각 계단을 밟을 때 비용: cost = [0, 2, 7, 2, 9, 12, 3]

cost[i]: 계단 i를 밟을 때 비용

costSum(i): 바닥에서 계단 i까지 올라갈 때 밟는 계단의 최소 비용 합

costSum(i) = 0 if i = 0

= cost[1] if i = 1

= cost[1] + cost[2] if i = 2

= cost[i] if i = 3 혹은 4

= min(costSum(i-1),costSum(i-3),costSum(i-4)) + cost[i] if i > 4

수행시간: O(n)

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**..**
costSum[i]	0	2	**9**	2	9	**14**	**5**	..

4. 양의 정수 n의 합 표현 방법 (1) – 나열되는 숫자들의 순서는 고려하지 않음

- 양의 정수 n에 대하여, n을 1부터 n까지 숫자들의 합으로 나타내는 방법의 수를 구하시오. 단 나열되는 숫자들의 순서는 고려하지 않는다. 1+2와 2+1은 동일한 것으로 간주한다.

  – 예: n = 4인 경우 다음의 5가지가 있다.

    1 + 1 + 1 + 1
    1 + 1 + 2
    2 + 2
    1 + 3
    4

  ✓ 숫자들의 합에서 숫자들의 순서는 고려하지 않으므로, 숫자들의 합 표현은 증가하는 숫자들의 합으로 표현

  C(i,j) : i를 1부터 j까지 숫자들의 합으로 나타내는 방법의 수
  (나열되는 숫자의 순서는 고려하지 않음)

  C(i,j) = 1 if i = 0
  = 1 if j = 1
  = C(i,i) if i < j
  = C(i-j,j) + C(i,j-1) if i ≥ j // 숫자들 합의 표현에서 마지막 숫자가 j

  => C(n,n)을 구하면 된다.

- 양의 정수 n에 대하여, n을 1부터 n까지 숫자들의 합으로 나타내는 방법의 수를 구하시오. 단 나열되는 숫자들의 순서는 고려하지 않는다. 1+2와 2+1은 동일한 것으로 간주한다.

  예: n = 4인 경우 다음의 5가지가 있다.

  1 + 1 + 1 + 1, 1 + 1 + 2, 2 + 2, 1 + 3, 4

  C(i,j) : i를 1부터 j이하의 숫자들의 합으로 나타내는 방법의 수   (나열되는 숫자의 순서는 고려하지 않음)

  $$C(i,j) = \begin{cases} 1 & \text{if } j = 1 \\ 1 & \text{if } i = 0 \\ C(i,i) & \text{if } i < j \\ C(i-j,j) + C(i,j-1) & \text{if } i \geq j \end{cases}$$

  => C(n,n)을 구하면 된다.

```
const int MAX = 100;
int count(int n, int k)
{
   int  C[MAX][MAX];
   for (int i = 0; i <= n; i++)
       for (int j = 0; j <= n; j++)
           if (j == 0)
              C[i][j] = 0;
           else if (i == 0)
              C[i][j] = 1;
           else if (i < j)
              C[i][j] = C[i][i];
           else
              C[i][j] = C[i-j][j]+C[i][j-1];
    return C[n][n];
}
```

i \ j	1	2	3	4	…	n
0	1	1	1	1	1	1
1	**1**	**1**	**1**	**1**	**1**	**1**
2	**1**	**2**	**2**	**2**	**…**	**2**
3	**1**	**2**	**3**	**3**	**…**	**3**
4	**1**	**3**	**4**	**5**	**…**	**5**
…						
n						

```
def count(n, k):
   C = [[0 for j in range(n+1)] for i in range(n+1)]
   for i in range(n+1):
        for j in range(n+1):
            if j == 1:
               C[i][j] = 1
            elif i == 0:
               C[i][j] = 1
            elif i < j:
               C[i][j] = C[i][i]
            else:
               C[i][j] = C[i-j][j]+C[i][j-1]
   return C[n][n]
```

수행시간: $O(n^2)$

4. 양의 정수 n의 합 표현 방법 (2) – 나열되는 숫자들의 순서는 고려하지 않음

- 양의 정수 n에 대하여, n을 1부터 k까지 숫자들의 합으로 나타내는 방법의 수를 구하시오. 단 나열되는 숫자들의 순서는 고려하지 않는다. 1+2와 2+1은 동일한 것으로 간주한다.

  – 예: n = 4이고, k = 3인 경우 다음의 4가지가 있다.

    1 + 1 + 1 + 1

    1 + 1 + 2

    2 + 2

    1 + 3

  C[n][k]를 구하면 된다.

5. 거스름돈을 나누어주는 방법 (1)

- 동전들의 단위 c_1, c_2, …,c_n이 있다. 이들 동전들을 이용하여 거스름돈 M을 나누어 주는 방법의 수를 구하시오.
  - 예: n = 3이고, (c_1, c_2, c_3) = (1, 3, 5), M = 10인 경우 다음의 7가지가 있다.

    1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1
    1 + 1 + 1 + 1 + 1 + 1 + 1 + 3
    1 + 1 + 1 + 1 + 3 + 3
    1 + 3 + 3 + 3
    1 + 1 + 1 + 1 + 1 + 5
    1 + 1 + 3 + 5
    5 + 5

    $c_1 < c_2 < c_3 < \ldots < c_n$이라 가정한다.

    C(i,j) : 거스름돈 i를 c_1부터 c_j까지 동전들을 이용하여 나누어주는 방법의 수

    $$C(i,j) = \begin{cases} 0 & \text{if } j = 0 \\ 1 & \text{if } j > 0 \text{ and } i = 0 \\ 0 & \text{if } j = 1 \text{ and } i < c_1 \\ C(i,j-1) & \text{if } i < c_j \\ C(i-c_j,j) + C(i,j-1) & \text{otherwise} \end{cases}$$

    => **C(M,n)**을 구하면 된다.

    수행시간: O(Mn)

i \ j	0	1	2	3(=n)
0	0	1	1	1
1	0	1	1	1
2	0	1	1	1
3	0	1	2	2
4	0	1	2	2
5	0	1	2	3
6	0	1	3	4
7	0	1	3	4
8	0	1	3	5
9	0	1	4	6
10 (=M)	0	1	4	7

5. 거스름 돈을 나누어주는 방법 (2)

- 동전들의 단위 c_1, c_2, …,c_n이 있다. 이들 동전으로 거스름 돈 M을 나누어 줄 때, 동전들의 최소 개수를 구하시오.
  - 예: n = 3이고, (c_1, c_2, c_3) = (1, 4, 5), M = 8인 경우 다음의 4가지가 있다.
    1 + 1 + 1 + 1 + 1 + 1 + 1 + 1
    1 + 1 + 1 + 1 + 4
    4 + 4
    1 + 1 + 1 + 5

$c_1 < c_2 < c_3 < \ldots < c_n$이라 가정한다.

C(i,j) : 거스름 돈 i를 c_1부터 c_j까지 동전들을 이용하여 나누어 줄 때, 동전들의 최소 개수

$C(i,j) = 0$ if $i = 0$
$= \infty$ if $0 < i < c_1$ or $j = 0$
$= C(i,j-1)$ if $i < c_j$
$= \min(C(i-c_j,j) + 1, C(i,j-1))$ otherwise

=> **C(M,n)**을 구하면 된다.

수행시간: O(Mn)

i \ j	0	1	2	3(=n)
0	∞	0	0	0
1	∞	1	1	1
2	∞	2	2	2
3	∞	3	3	3
4	∞	4	1	1
5	∞	5	2	1
6	0	6	3	2
7	0	7	4	3
8 (=M)	0	8	2	2

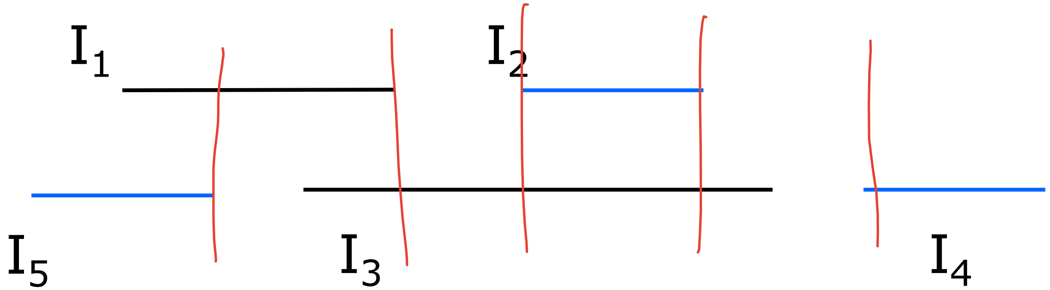
6. Weighted Interval Scheduling

- Interval Scheduling problem (구간 스케줄링 문제):

  n개의 구간들 $I_1, I_2, \ldots, I_n$(구간 I_i 의 시작시간 s_i, 끝나는 시간 f_i)에 대하여, 큰 겹치지 않는 가장 많은 구간들을 구하라.

  activity selection problem 회의실 배정 문제

- 아래의 5개의 구간 I_1, I_2, I_3, I_4, I_5에 대하여, 겹치지 않는 구간은 최대 3개 (I_2, I_4, I_5)이다

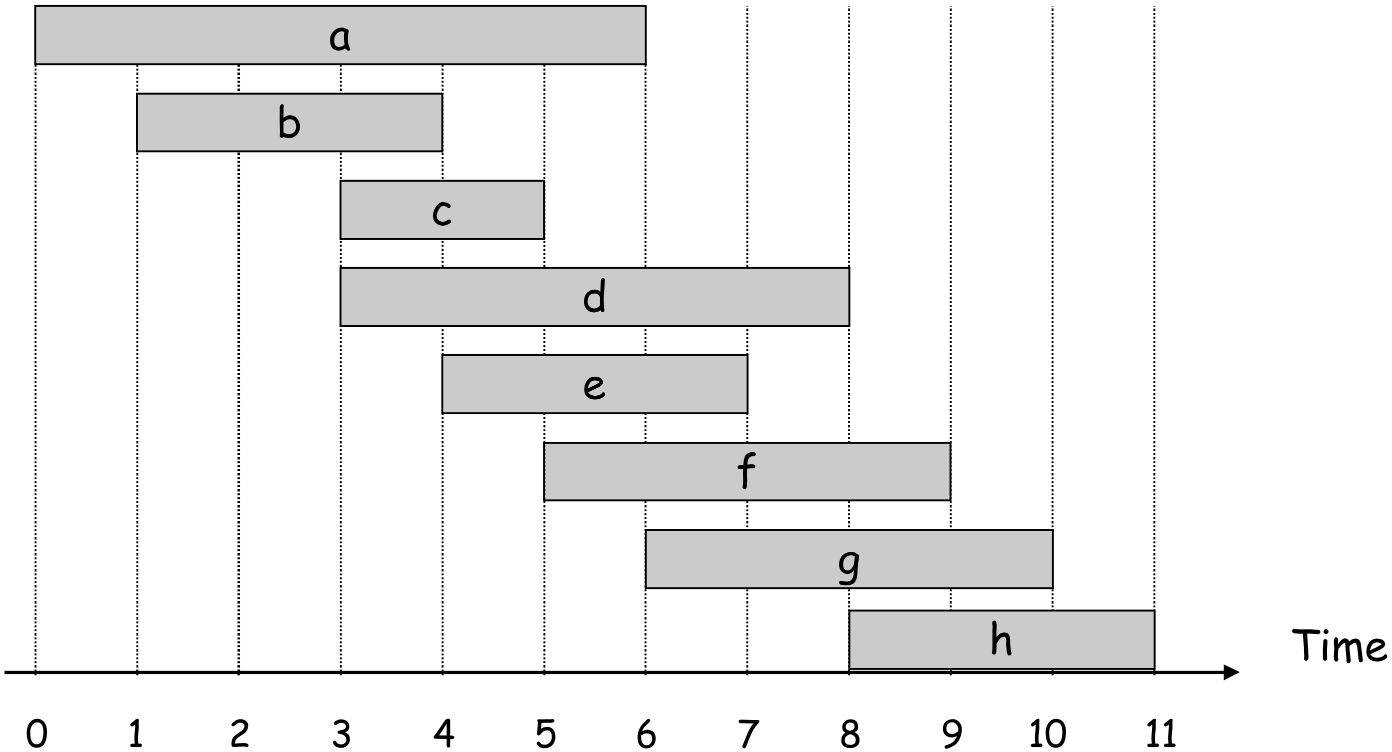


→ interval scheduling

- 남아 있는 구간들 중 끝나는 시간이 가장 빠른 구간을 선택하는 전략 (욕심장이 방법: Greedy method)
  - This gives the blue intervals in the example.

=> 먼저 끝나는 구간을 선택하는 욕심장이(greedy) 방법이 최적 해를 구한다

구간 스케쥴링

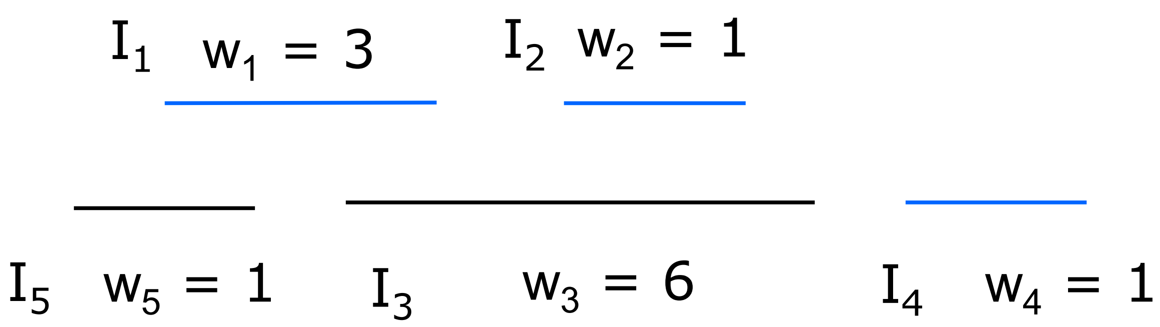


Weighted Interval Scheduling (가중치가 있는 구간 스케쥴링)

- Weighted interval scheduling problem (가중치가 있는 구간 스케쥴링 문제) activity

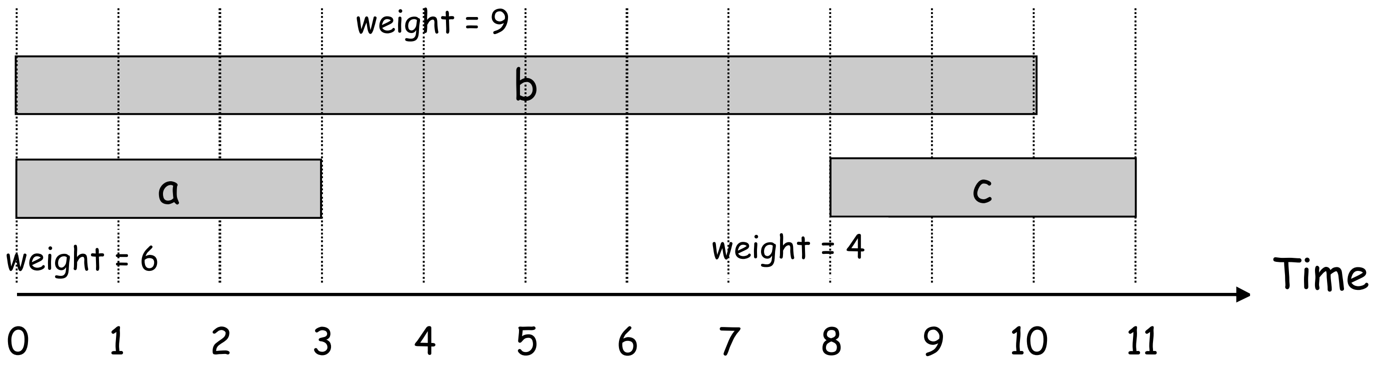
  n개의 구간들 $I_1, I_2, \ldots, I_n$(구간 I_i 의 시작시간 s_i, 끝나는 시간 f_i, 가중치 w_i)에 대하여, 가중치의 합이 가장 큰 겹치지 않는 구간들을 구하라.

- 예: 아래의 5개의 구간 I_1, I_2, I_3, I_4, I_5에 대하여, 가중치의 합이 가장 큰 겹치지 않는 구간은 ($I_{\cancel{1}\,3}$, $I_{\cancel{2}\,4}$, $I_{\cancel{4}\,5}$ ~~예~~)이다



가중치가 있는 구간 스케쥴링(계속)

- Greedy 방법은 최적인 해를 보장하지 못한다.



최적해 a, c

그리디 : b

가중치가 있는 구간 스케쥴링(계속)

- 주어진 n개의 구간들을 finish time에 의하여 정렬한다

  이들 구간들을 $I_1, I_2, \ldots, I_n$이라 하고, 구간 I_i 의 시작시간과 끝나는 시간을 s_i , f_i라 하고, 그 weight(가중치)를 w_i라 하자.

- <u>부분 문제(subproblem) 정의</u>

  $I_1, I_2, \ldots, I_i$에 대하여, 겹치지 않으면 가중치 합이 가장 많은 ~~큰~~ 구간들을 찾아라.

  재귀적 해를 구할 수 있어야됨. 부분문제들을 가지고 본 문제를 구할 수 있어야함.

- <u>부분문제의 최적 해 값(목적함수)</u>

  OPT(i) : $I_1, I_2, \ldots, I_i$에 대하여, 겹치지 않는 구간들의 가중치 합의 최대값

- <u>주어진 문제의 최적 해 값(목적함수)</u>

  OPT(n)

- <u>부분문제 최적 해 값(목적함수)의 재귀적 정의</u>

  $OPT(i) = \max\{OPT(i-1), w_i + OPT(j)\}$,

  여기서 j는 $I_1, I_2, \ldots, I_{i-1}$중 I_i 와 겹치지 않으면서 finish time이 가장 큰 구간의 인덱스

가중치가 있는 구간 스케쥴링(계속)

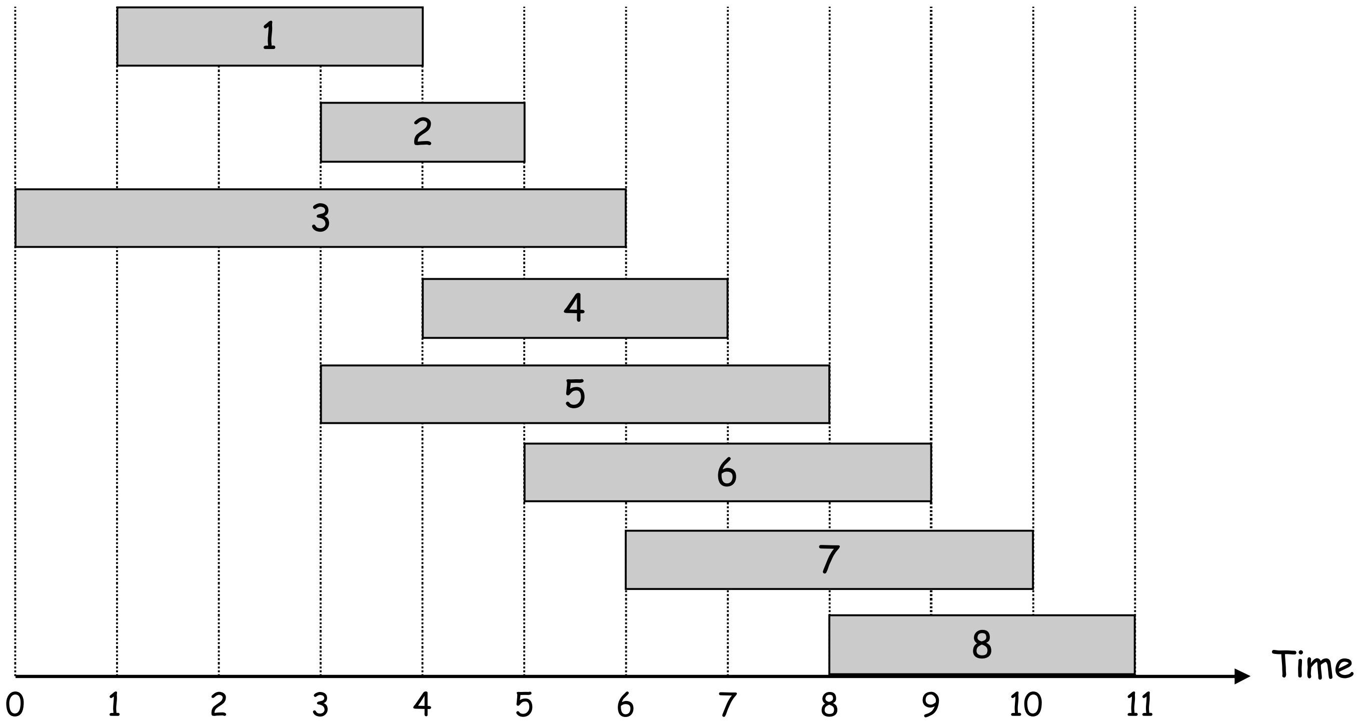
예: n = 8

끝나는 시간으로 정렬

Label intervals by finishing time: $f_1 \le f_2 \le . . . \le f_n$.

w 배열

i	1	2	3	4	5	6	7	8	9
w[i]	3	6	8	6	5	7	8	3	



가중치가 있는 구간 스케쥴링 (계속)

- Bottom-up dynamic programming.

Input: n, $s_1,\ldots,s_n$, $f_1,\ldots,f_n$, $w_1,\ldots,w_n$

1. Sort intervals by finish times so that $f_1 \le f_2 \le \ldots \le f_n$. $O(n\log n)$

// p(i) = largest index j < i such that interval I_j does not overlap interval I_i.

2. Compute p(1), p(2), …, p(n) $O(n\log n)$

3. Algorithm Iterative-Compute-Opt

```
    OPT[0] = 0
    for i = 1 to n
       OPT[i] = max(w_j + OPT[p(i)], OPT[i-1])
```

$O(n)$

수행시간: O(n log n)

가중치가 있는 구간 스케쥴링 (계속)

w 배열

i	1	2	3	4	5	6	7	8	9
w[i]	3	6	8	6	5	7	8	3	

P 배열

i	1	2	3	4	5	6	7	8	9
P[i]	0	0	0	1	0	2	3	5	

OPT 배열

i	0	1	2	3	4	5	6	7	8	9
OPT[i]	0	3	6	8	9	9	13	16	16