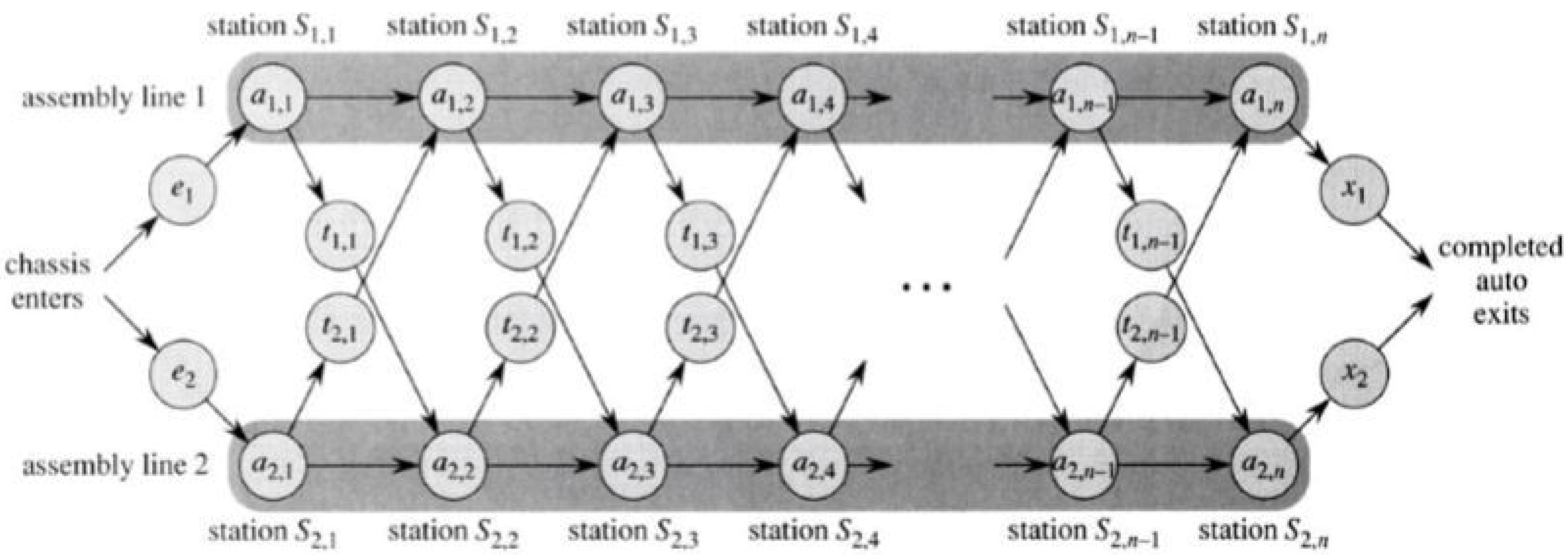
동적계획법
(Dynamic Programming)

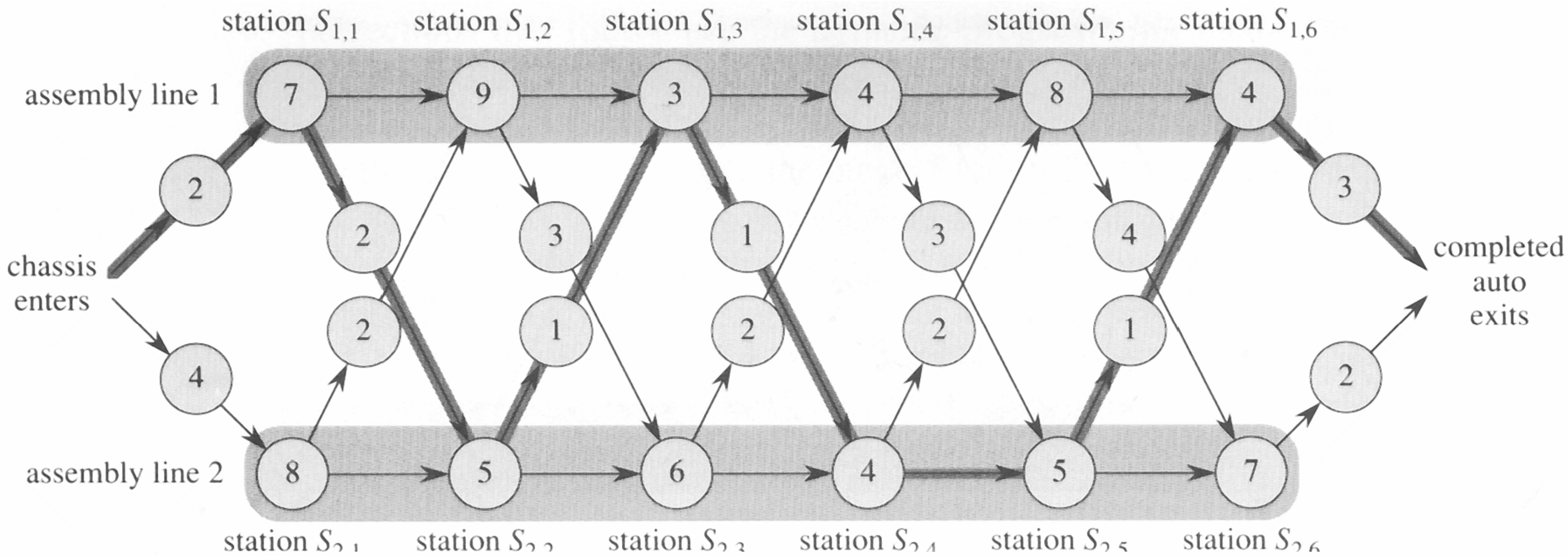
- 개요
- 피보나치 수열; 이항계수; 양의 정수 n의 합 표현 방법; 거스름돈 나누어주는 방법
- Weighted Interval Scheduling;
- 연속하는 수들의 최대 합 구하기
- 그리드에서 경로 찾기
- LCS(Longest Common Subsequence) 문제
- Assembly line scheduling 문제

9. Assembly line scheduling



- Car chassis 를 조립하기 위한 특정 공장에 두 개의 라인이 있다.
- 그림에서 위의 라인이 1, 아래의 라인이 2이다.
- 각 라인에서 진행되는 1부터 n까지의 조립공정(station에서 수행)이 있다.
- 같은 열에 있는 공정은 같은 공정이지만 라인에 따른 시간차이가 있다.
- 위 또는 아래의 라인에서 다른 라인으로 넘어 가는데 소요되는 시간이 있다. 같은 라인에서 넘어가는 시간은 0이라고 가정한다.
- 문제: Car chassis를 완성하는데 걸리는 최소시간을 구하라.

예



j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

$f_1(j)$: 시작 위치에서부터 스테이션 $S_{1,j}$까지 조립하는데 걸리는 최소시간
$f_2(j)$: 시작 위치에서부터 스테이션 $S_{2,j}$까지 조립하는데 걸리는 최소시간
f*: 조립을 완료하는데 걸리는 최소시간

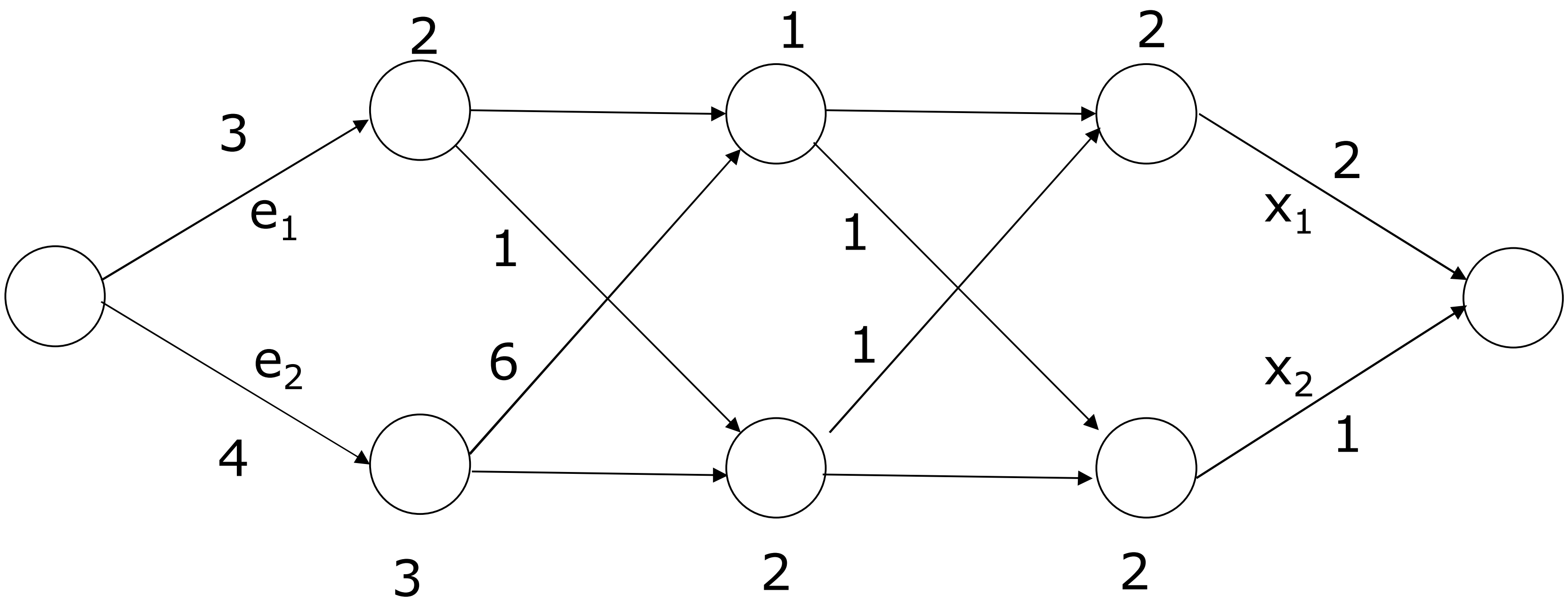
- $f_1(i)$: 시작 위치에서부터 스테이션 $S_{1,i}$까지 조립하는데 걸리는 최소시간
- $f_2(i)$: 시작 위치에서부터 스테이션 $S_{2,i}$까지 조립하는데 걸리는 최소시간

- f^*: 시작위치에서부터 조립을 끝나는데 걸리는 최소시간: $\min(f_1(n)+x_1, f_2(n)+x_2)$ (최종적으로 가장 빠른 시간을 저장하는 변수)

$$f_1(i) = \begin{cases} e_1 + a_{1,1} & , i = 1 \\ \min(f_1(i-1)+a_{1,i}, f_2(i-1)+t_{2,i-1} + a_{1,i}) & , i >= 2 \end{cases}$$

$$f_2(i) = \begin{cases} e_2 + a_{2,1} & , i = 1 \\ \min(f_2(i-1)+a_{2,i}, f_1(i-1)+t_{1,i-1} + a_{2,i}) & , i >= 2 \end{cases}$$

a	**2**	**1**	**2**	



b	**3**	**2**	**2**	
t_1	**1**	**1**		
t_2	**6**	**1**		

```
def f(a,b,n, e1,e2,x1,x2,t1,t2):
# a[i] : a_{1,i}
# b[i] : a_{2,i}
# t1[i]: t_{1,i}
# t2[i]: t_{2,i}

    f1 = [0 for x in range(n)]
    f2 = [0 for x in range(n)]

    f1[0] = e1+a[0]
    f2[0] = e2+b[0]

    for i in range(1,n):
        f1[i] = min(f1[i-1],f2[i-1]+t2[i-1])+a[i]
        f2[i] = min(f2[i-1],f1[i-1]+t1[i-1])+b[i]

    return min(f1[n-1]+x1, f2[n-1]+x2)
```

Brute - force

- 최적의 답을 찾기 위해 전체 경우의 수를 고려하면, n개의 공정이 있다고 가정할때, 2개의 라인이 존재하므로 2^n 개의 경우의 수를 고려해야 한다.
- S1, n의 공정을 수행하는 데 오는 경로는 S1, (n-1)과 S2, (n-1)이 있으므로. 특정 위치의 공정을 위한 경로을 확인하면, 각각 2개씩 존재함을 알 수있고, 공정은 총 n개이기 때문에 2^n 개를 확인해야하는 것이다.
- 즉, 이러한 모든 경우의 수를 탐색하는 Brute - force Approach로는 다음과 같은 시간 복잡도를 가진다. $T(n) = \Omega(2^n)$
- 동적 프로그래밍의 기본 핵심은 "테이블" 사용
-

(1) Pseudo-code

```
Fastest-Way(a, t, e, x, n)
   f1[1] <- e1+a1,1
   f2[1] <- e2+a2,1

   for j = 2 to n
      do if f1[j-1] + a1,j ≤ f2[j-1] + t2,j-1 + a1,j
         then f1[j] <- f1[j-1] + a1,j
              l1[j] <- 1
         else f1[j] <- f2[j-1] + t2,j-1 + a1,j
              l1[j] <- 2
      if f2[j-1] + a2,j ≤ f1[j-1] + t1,j-1 + a2,j
         then f2[j] <- f1[j-1] + a1,j
              l2[j] <- 2
         else f2[j] <- f1[j-1] + t1,j-1 + a2,j
              l2[j] <- 1

   if f1[n] + x1 ≤ f2[n] + x2
       then f* = f1[n] + x1
            l* = 1
```

(2) Python-code

```python
def f(a, b, n, e1, e2, x1, x2, t1, t2):
   f1 = [0 for x in range(n)]
   f2 = [0 for x in range(n)]

   f1[0] = e1 + a[0]
   f2[0] = e2 + b[0]

   for i in range(1, n):
      f1[i] = min(f1[i-1], f2[i-1] + t2[i-1]) + a[i]
      f2[i] = min(f2[i-1], f1[i-1] + t1[i-1]) + b[i]

   # print(f1)
   # print(f2)

   return min(f1[n-1]+x1, f2[n-1]+x2)
```