

동적계획법



1. 동적계획법 소개



동적계획법의 역사

- ✓ Richard E. Bellman이 1950년 RAND Corporation에서 제시함.
- ✓ 당시의 단어 "Programming"은 최적 스케줄을 찾는 방안을 의미.
- ✓ Bellman의 회고 ...
- ✓ "The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was [Secretary of Defense](#), and he actually had a pathological fear and hatred of the word, research... His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical... I thought **dynamic programming was a good name**. It was something not even a Congressman could object to."



1. 동적계획법 소개



동적계획법의 주요 전략

- ✓ 기본은 inductive reasoning 이다.
- ✓ 만일 주어진 문제보다 조금 작은 문제들의 답을 안다면 그것을 이용해서 좀 더 큰 문제의 답을 구할 수 있다.
 - ✦ 두 문제 P 와 P' 에 대해 P 의 입력이 P' 의 입력의 부분 집합 (혹은 크기/범위가 작은 입력) 이라면 P 는 P' 의 부분 문제이다.
 - ✦ 문제 P' 를 해결하기 위해 필요한 부분 문제들의 답을 먼저 구한 후, 이들을 이용하여 P' 의 답을 구한다.
- ✓ 기본 구조
 - ✦ 재귀식 예) $P(k) = \max\{ P(k-1), P(k-2) \}$
 - ✦ 기저조건 (base condition).



2.1 동적계획법의 구조



문제 표현과 매개변수

- ✓ $P(k)$
- ✓ $P(n, m)$: n 개에서 서로 다른 m 개 선택하기



기저조건 (base condition)

- ✓ 확정된 정답
- ✓ $N=1, 2$ 일 때의 정답



재귀식

- ✓ $N \leq k-1$ 일 때의 모든 정답을 알고 있다고 가정
- ✓ 그 다음 단계의 답을 구한다. 예) $P(k) = \max\{ P(k-1), P(k-2) \}$



Computing Resource

- ✓ 부분문제들의 답을 저장해 두어, 다시 계산하지 않도록 함.



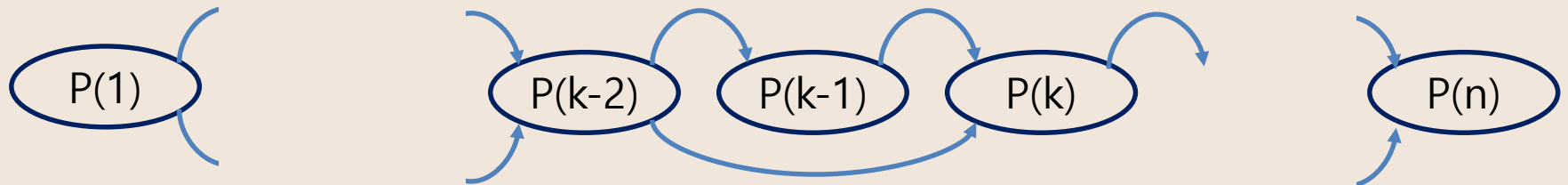
2.1 동적계획법의 구조



동적계획법의 구조적 특징

- ✓ 전체 문제를 여러 개의 부분 문제들(Subproblems)로 분할하여, 부분 문제들 사이의 관계가 DAG(Directed Acyclic Graph)를 이루도록 한다.

✱ 예) $P(k) = \max\{ P(k-1), P(k-2) \}$



- ✓ 따라서, 부분문제들을 이 관계에 따라 순서대로 정렬할 수 있다.
 - ✱ 그래프(DAG)에서 입력 선분이 없는 문제가 가장 먼저 온다.
 - ✱ 입력 선분이 없는 문제를 그래프에서 삭제한 후, 남은 그래프에서 입력 선분이 없는 문제가 그 다음 순서이다. 이 문제를 삭제하고 이 과정을 반복한다.



2.1 동적계획법의 구조



동적계획법의 해결 방법

- ✓ 부분 문제들을 하나씩 순서대로 해결한다.
 - ✧ 가장 작은 부분 문제를 먼저 해결한다.
 - ✧ 큰 부분 문제를 해결하려고 할 때, 작은 부분 문제들의 답을 이용한다.
 - ✧ 모든 부분 문제들을 다 해결할 때까지 지속한다.
- ✓ 배열 등을 이용하여 부분문제들의 답들을 저장해 두고, 더 큰 부분 문제를 해결할 때 이 답들을 이용하여 효율적으로 계산한다.
 - ✧ 예) $P(k) = \max\{ P(k-1), P(k-2) \}$
 - ✧ 일차원 배열 T에 두 부분 문제 $P(k-1)$ 과 $P(k-2)$ 의 답을 $T[k-1]$ 과 $T[k-2]$ 에 각각 저장해 두면, 나중에 더 큰 부분 문제 $P(k)$ 를 해결할 때, 이 답들을 이용하여 $P(k)$ 의 답을 단시간에 계산할 수 있다. 이 답을 $T[k]$ 에 저장한다.



2.1 동적계획법의 구조

예제: n 개의 가로막대가 있는 사다리를 한번에 한 칸 또는 두 칸씩 올라갈 수 있다. 이 사다리를 올라가는 방법의 수를 구하시오.

(올라가는 순서가 다르면 다른 방법으로 간주한다.)

부분 문제 $L(k)$: 가로막대가 k 개가 있을 때의 경우의 수

✓ $L(1)=1, \{ 1 \}$

✓ $L(2)=2, \{ 1+1, 2 \}$

✓ $L(3)=3, \{ 1+1+1, 2+1, 1+2 \}$

부분 문제들 사이의 관계 : $L(k) = L(k-1) + L(k-2)$

✓ k 번째 칸에 도달하려면 1칸 아래 혹은 2칸 아래의 칸에서 한번에 이동

해결 방법 : 작은 부분 문제부터 순서대로 해결.

✓ 가장 작은 문제 $L(1)$ 부터 해결

✓ $L(k-2)$ 와 $L(k-1)$ 을 해결한 후 $L(k)$ 를 해결

✓ $L(n)$ 을 맨 마지막에 해결하면, $L(n)$ 의 값이 전체 문제의 답을 나타낸다.

최적 부분문제 구조(Optimal substructure)

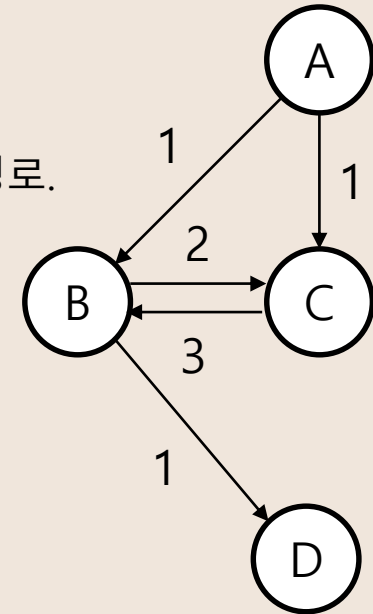
- ✓ 주어진 문제가 최적화의 원칙(Principle of Optimality)을 만족해야만 동적 계획법을 효율적으로 적용할 수 있다.
- ✓ 최적화의 원칙이란 어떤 문제에 대한 해가 최적일 때 그 해를 구성하는 부분 문제들의 해 역시 최적이어야 한다는 것이다.

✍️ 최적의 원칙이 적용되지 않는 예: 최장경로(Longest Path) 문제

- ✓ 최장 경로란 그래프에서 두 정점을 가장 긴 길이로 연결하는 단순 경로를 말한다.

★ 단순 경로: 그래프의 정점을 최대 한번씩만 지나는 경로.

- ✓ A에서 D로의 최장 경로는 [A, C, B, D]가 된다.
- ✓ 그러나, 이 경로의 부분 경로인 A에서 C로의 최장 경로는 [A, C]가 아니라 [A, B, C]이다.
- ✓ 최적의 원칙이 적용되지 않는다.
- ✓ 따라서 최장경로문제는 동적계획법으로 효율적으로 해결하기 어렵다.





2.2 동적 계획법의 예제



예제 1: 금화 모으기



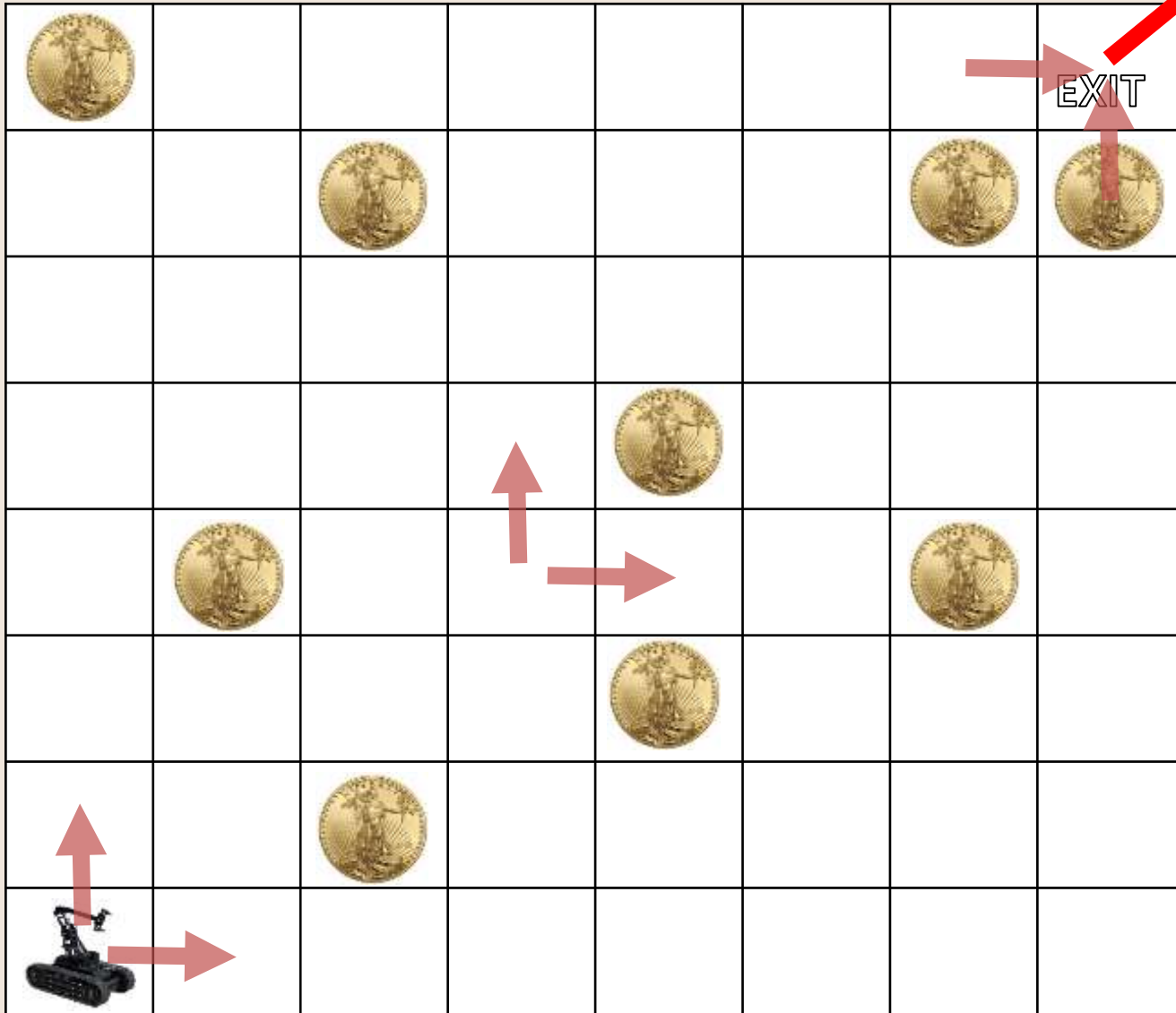
예제 2: 동전 바꾸기



예제 3: 프리랜서의 일정 정하기

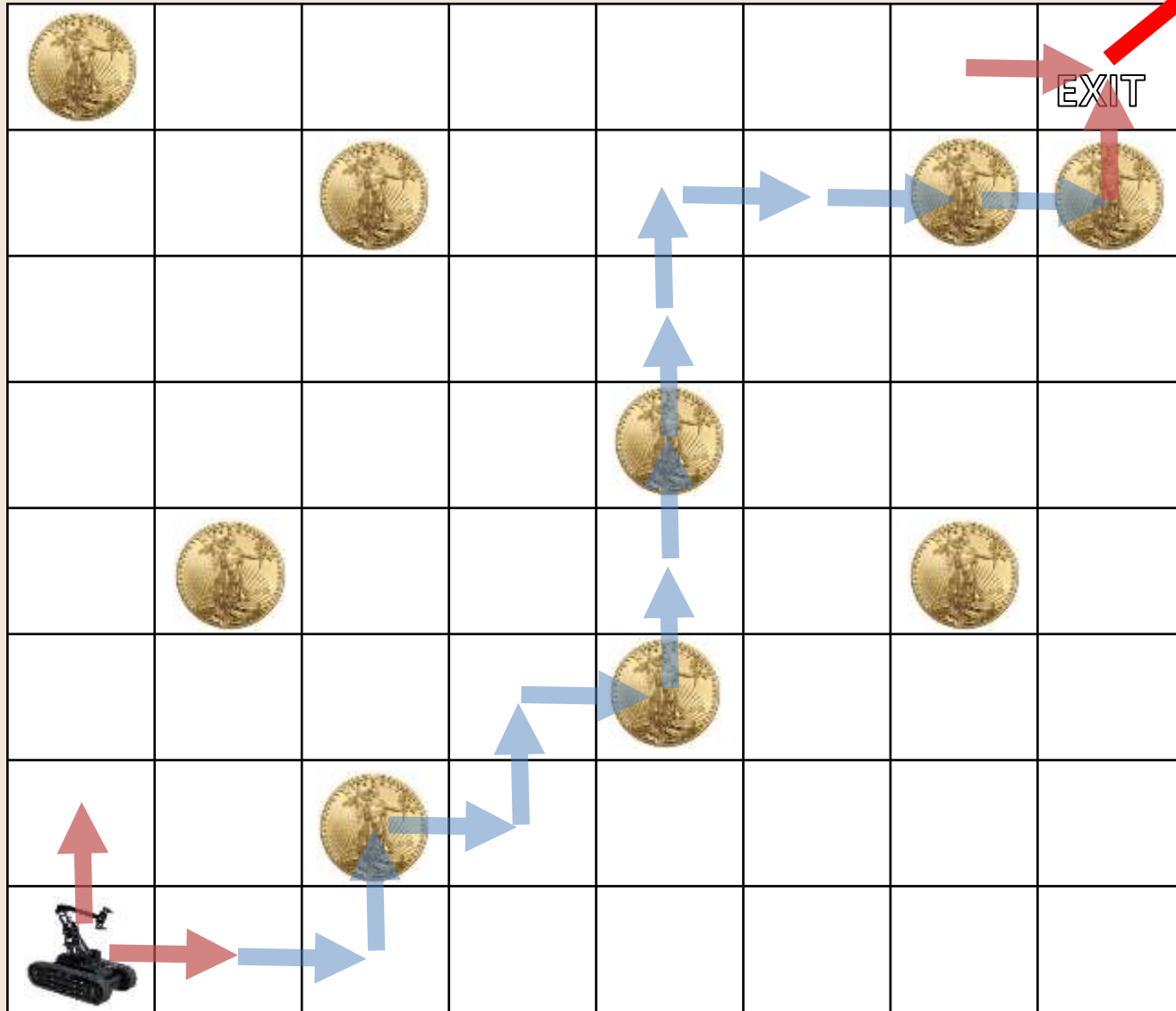


예제 1: 금화 모으기





예제 1: 금화 모으기

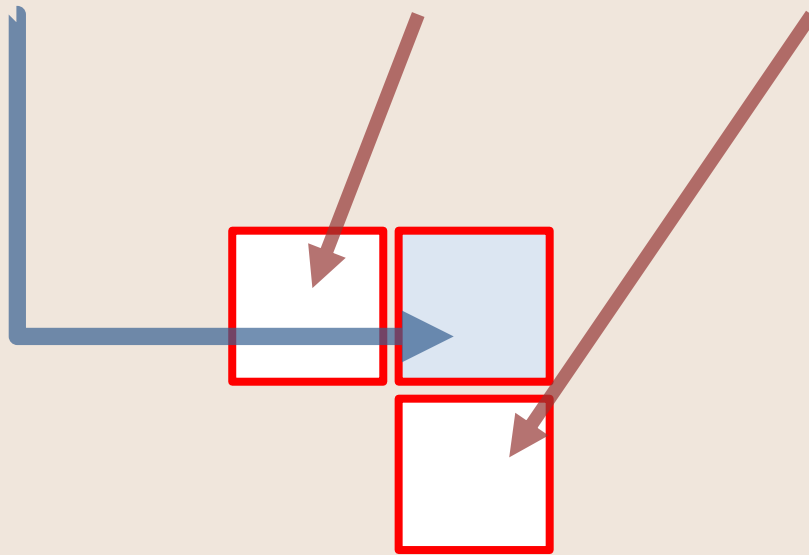




예제 1: 금화 모으기

✎ 부분문제 $\text{Cell}(i, j)$: 로봇이 (i, j) 에 도착할 때까지 모을 수 있는 최대 금화 개수.

✎ $\text{Cell}(i, j) = \max \{ \text{Cell}(i-1, j), \text{Cell}(i, j-1) \} + \text{금화}[i][j]$













✎ $\text{금화}[i][j] = 1$, if 금화

✎ $\text{금화}[i][j] = 0$, if no 금화



예제 1: 금화 모으기

						A	EXIT
							B 
							
0	1 	1	1	2	2	3 	3
0	0	1	1	2 	2	2	2
0	0	1 	1	1	1	1	1
	0	0	0	0	0	0	0





예제 2: 동전 거스름 돈

✎ 문제: 거스름 돈을 돌려 줄 때, 최소 갯수의 동전을 사용하시오.

✎ 동전의 종류

✓ 1원, 4원, 6원

✎ 8원을 거슬러주려 한다. 몇 개의 동전을 사용하여 거슬러 주면 되나?

✎ 그리디 방법(가장 큰 단위 동전 사용)을 이용한 접근

✓ 6원, 1원, 1원

✎ 최적은?

✓ 4원, 4원

✎ 동적 계획법으로 해결하자.



8원 잔돈에 대한 재귀 알고리즘 (하향식)

✓ 3가지 동전 각각을 선택해서 재귀적으로 해결

1원 동전 한 개 + 7원에 대한 최적해

4원 동전 한 개 + 4원에 대한 최적해

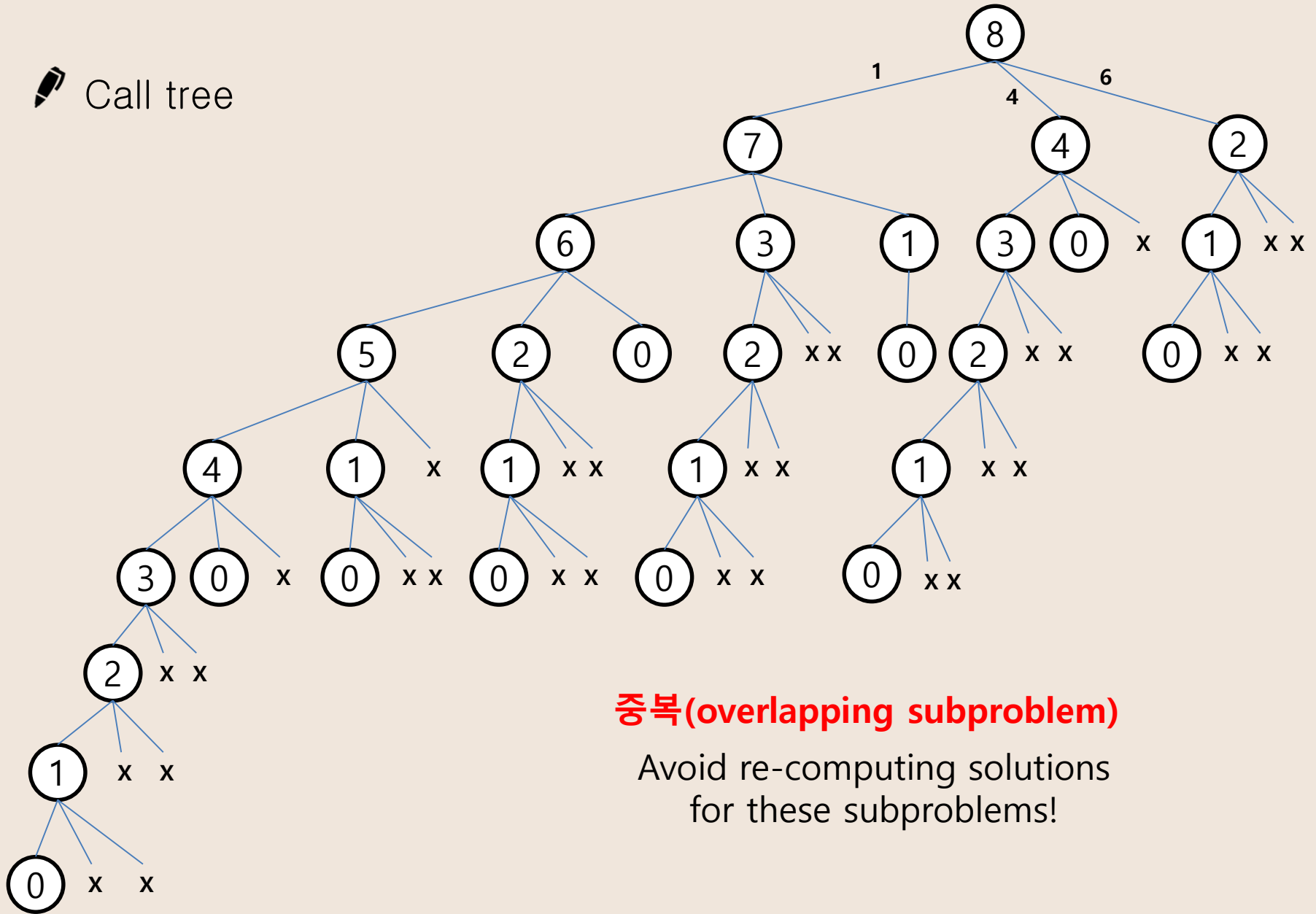
6원 동전 한 개 + 2원에 대한 최적해

위의 3가지 해 중 최적해를 선택

7원에 대한 최적해는 다시 1원, 4원, 6원 동전을 선택하고 나머지 액수에 대한 최적해



Call tree



DP 접근 : 상향식

- ✓ 1원에 대한 최적해 \rightarrow (선택) 2원에 대한 최적해 \rightarrow (선택) 3원에 대한 최적해 \rightarrow (선택) 4원에 대한 최적해 \rightarrow (선택) ...
- ✓ $C(j) = j$ 원을 거슬러 줄 때의 최적

n	sol	choice
8	2	$C[n - 1] + 1 = 3, C[n - 4] + 1 = 2, C[n - 6] + 1 = 3 \rightarrow 2$
7	2	$C[n - 1] + 1 = 2, C[n - 4] + 1 = 4, C[n - 6] + 1 = 2 \rightarrow 2$
6	1	$C[n - 1] + 1 = 3, C[n - 4] + 1 = 3, C[n - 6] + 1 = 1 \rightarrow 1$
5	2	$C[n - 1] + 1 = 2, C[n - 4] + 1 = 2 \rightarrow 2$
4	1	$C[n - 1] + 1 = 4, C[n - 4] + 1 = 1 \rightarrow 1$
3	3	$C[n - 1] + 1 \rightarrow 3$
2	2	$C[n - 1] + 1 \rightarrow 2$
1	1	$C[n - 1] + 1 \rightarrow 1$
0	0	0

부분 문제 정의

✓ $C(i, j)$ = 최대크기 D_i 동전으로 j 원을 거슬러 줄 때의 최적

★ (D_1 는 1원, D_2 는 4원, D_3 는 6원)

★ 거스름돈에 (1) D_i 동전을 사용한 경우, (2) 사용하지 않은 경우.

최적화 원리

✓ 만약 최대크기 D_i 동전으로 j 원을 거슬러 줄 때 최적으로 거슬러 주지 않는다면, 전체 문제의 거스름돈의 개수도 최적이지 않다.

점화식과 테이블

✓ $C(i, j) = \min\{ C(i, j-D_i) + 1, C(i-1, j) \} \rightarrow O(i * j)$

거스름돈	0원	1원	2원	3원	4원	5원	6원	7원	8원	9원
$D_1 \Rightarrow 1$ 원	0	1	2	3	4	5	6	7	8	9
$D_2 \Rightarrow 4$ 원	0	1	2	3	1	2	3	4	2	3
$D_3 \Rightarrow 6$ 원	0	1	2	3	1	2	1	2	2	3



예제 3: 프리랜서의 일정 정하기

- 자유직업인인 인테리어 업자가 있다. 10일간의 일정에 대한 요청이 있다. 어떤 기간은 작업 기간은 길지만 많은 돈을 받는다.
- 직업에 특성상 한번 작업을 하면 그 동안 다른 작업은 할 수 없다. 주어진 일정에서 가장 많은 돈을 버는 일정을 잡아라.

1	2	3	4	5	6	7	8	9	10
T1: 30만원							T4: 9만원		
			T2: 25만원						
T8: 15만원				T6: 27만원					
	T3 : 42만원								
T5: 20만원				T9: 7만원		T7: 16만원			



예제 3: 프리랜서의 일정 정하기

- ✎ T3을 선택하는 경우 다른 모든 일정과 겹치기 때문에 다른 일정은 선택 할 수 없다.
- ✎ 만일 Greedy하게 한다면 제일 돈이 많은 T3를 선택해야 하지만 그 보다 T1 + T7을 선택하면 더 많이 벌 수 있다.
- ✎ 이 문제는 가장 흔히 접할 수 있는 전형적인 문제이다.

1	2	3	4	5	6	7	8	9	10
T1: 30만원							T4: 9만원		
			T2: 25만원						
T8: 15만원				T6: 27만원					
	T3 : 42만원								
T5: 20만원				T9: 7만원		T7: 16만원			



예제 3: 프리랜서의 일정 정하기

✎ 신청 일정이 1개일 경우, 선택

✎ 신청 일정이 2개일 경우

✓ 겹친다면 더 많은 돈을 지불하는 일정, 안 겹친다면 두 일정 모두 선택

✎ 부분 문제들 $OPT(j)$ 를 정의한다.

✓ 각 일정이 마치는 시간을 key로 하여 전체 일정을 다시 정렬하고 순서대로 번호를 붙인다. 즉, 일정 j 는 전체 일정 중 j 번째로 마치는 일정이다.

✓ $OPT(j)$: 1부터 j 까지의 일정에 대한 최적의 해

✎ OPT 가 일정 j 를 선택하지 않음: $OPT(j) = OPT(j-1)$.

✎ OPT 가 일정 j 를 선택함: $OPT(j) = w(j) + OPT(p(j))$.

$p(j)$ 는 일정 j 와 겹치지 않고 $p(j) < j$ 인 일정들 가운데 맨 마지막 일정.


$w(j)$ 는 일정 j 를 수행했을 때 받은 돈 액수

✎ $OPT(j) = \max \{ OPT(j-1), w(j) + OPT(p(j)) \}$

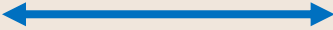

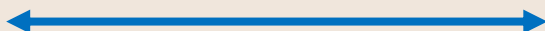


예제 3: 프리랜서의 일정 정하기



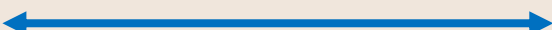
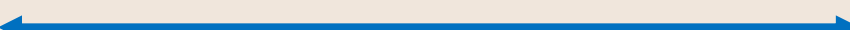
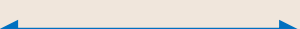

 $OPT(j) = \max \{ OPT(j-1), w(j) + OPT(p(j)) \}$

 Goal: $OPT(n)$

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

T1, 15만원 
 T2, 20만원 
 T3, 30만원 

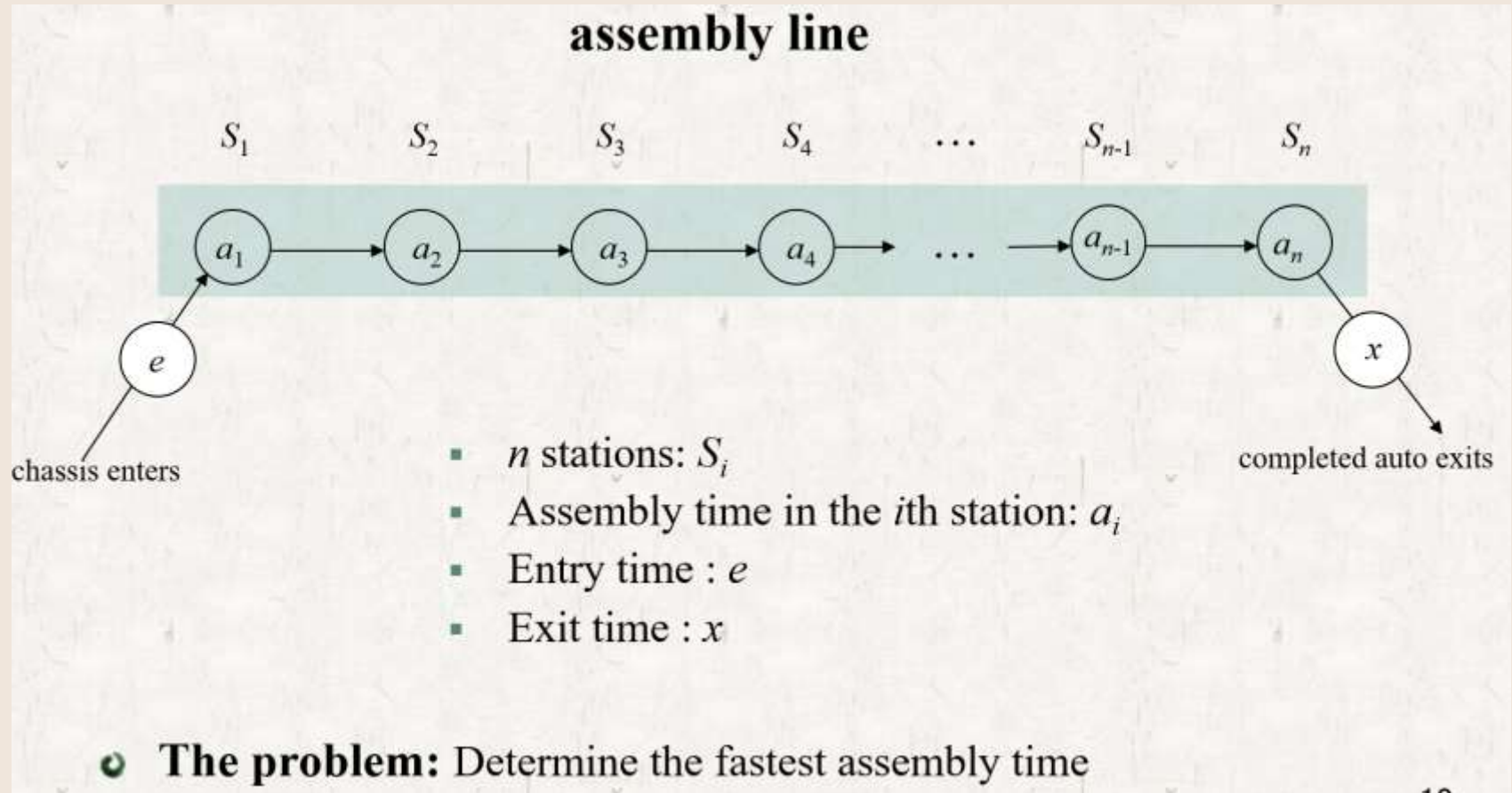
1	2	3	4	5	6	7	8	9	10
T1: 30만원							T4: 9만원		
			T2: 25만원						
T8: 15만원				T6: 27만원					
T3: 42만원									
T5: 20만원					T9: 7만원		T7: 16만원		

T4, 7만원 
 T5, 25만원 
 T6, 27만원 
 T7, 42만원 
 T8, 9만원 
 T9, 16만원 

OPT	1	2	3	4	5	6	7	8	9
	15	20	30	30	40	47	47	49	49



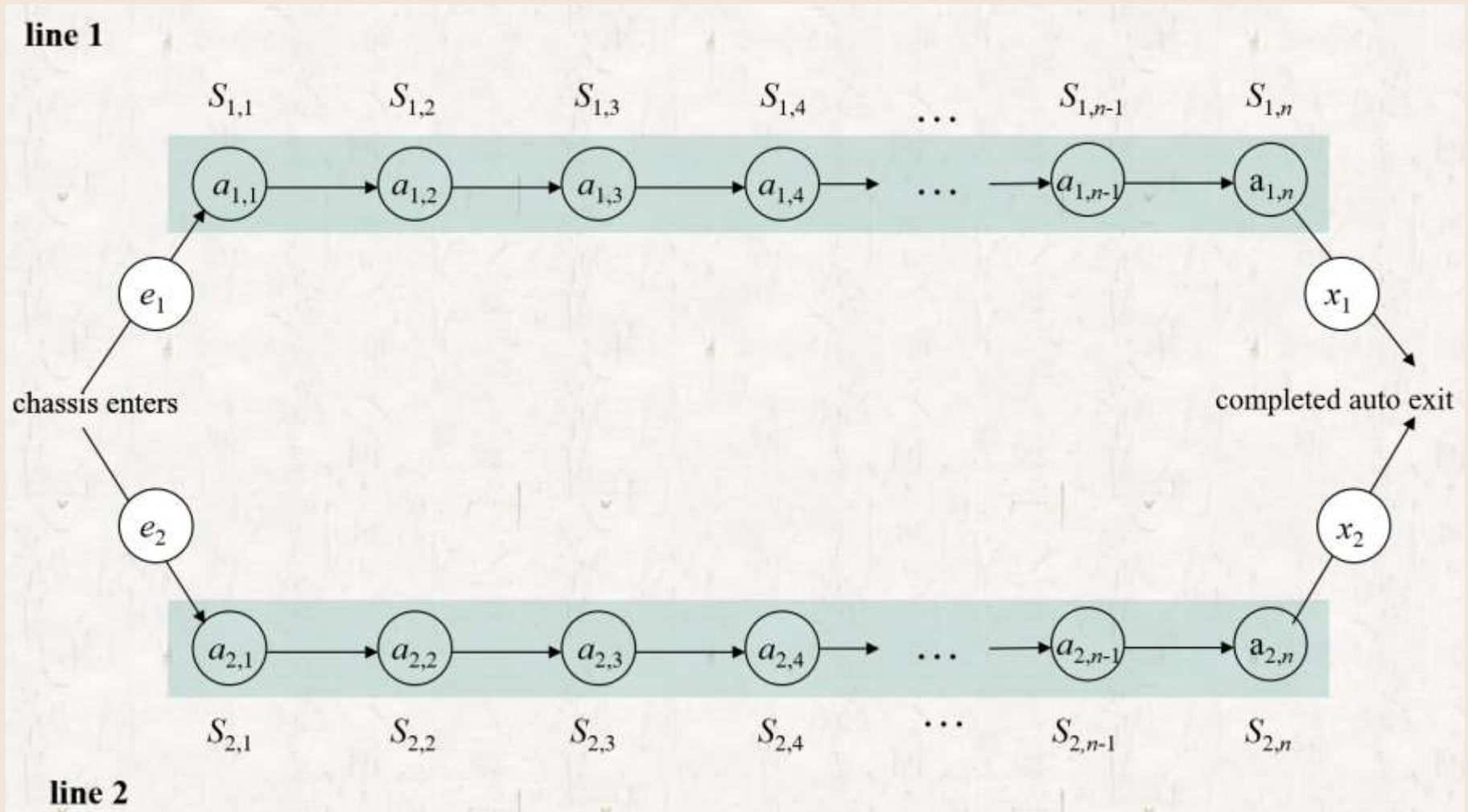
예제 4: Assembly-line scheduling



10

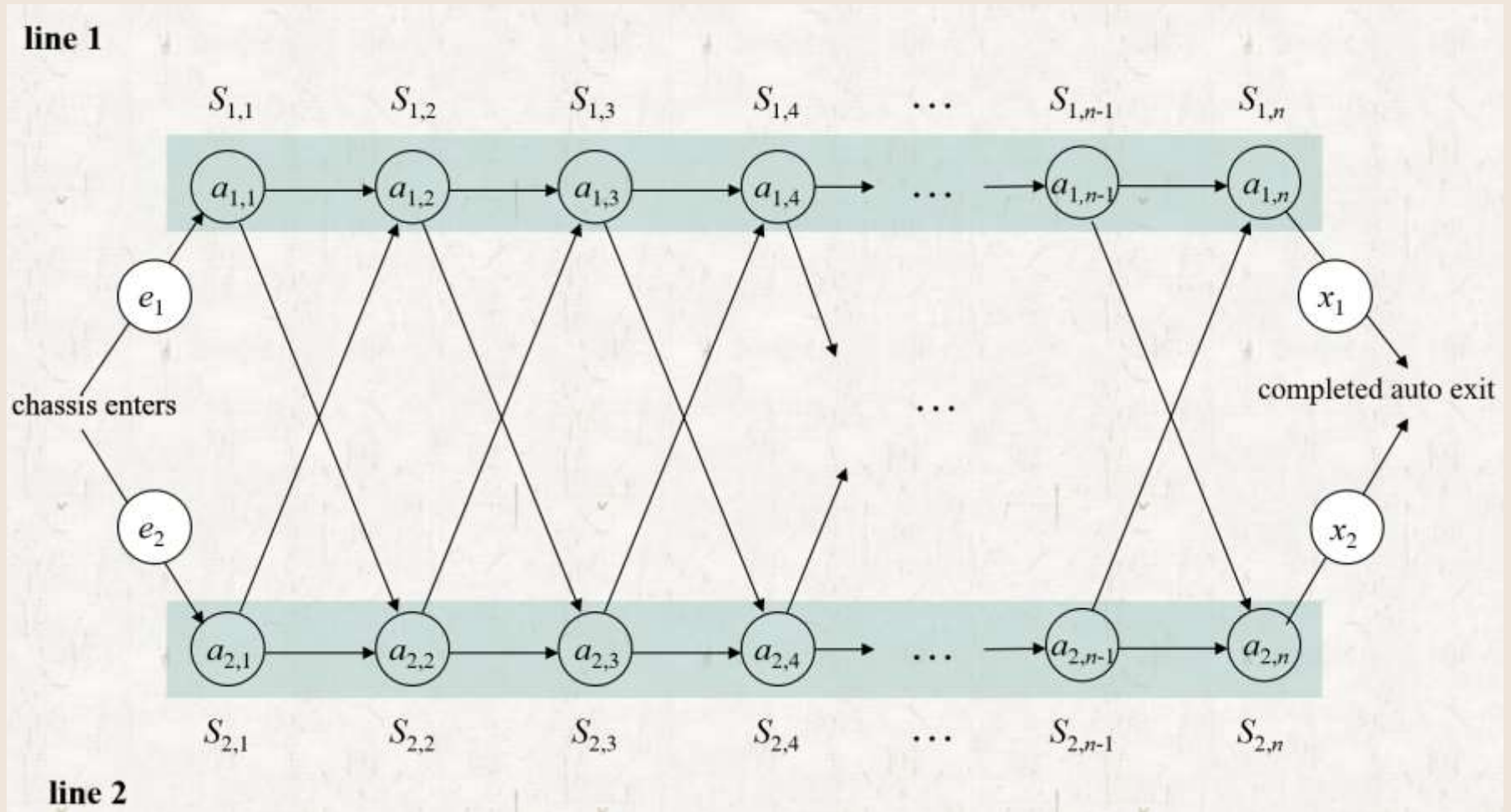


예제 4: Assembly-line scheduling



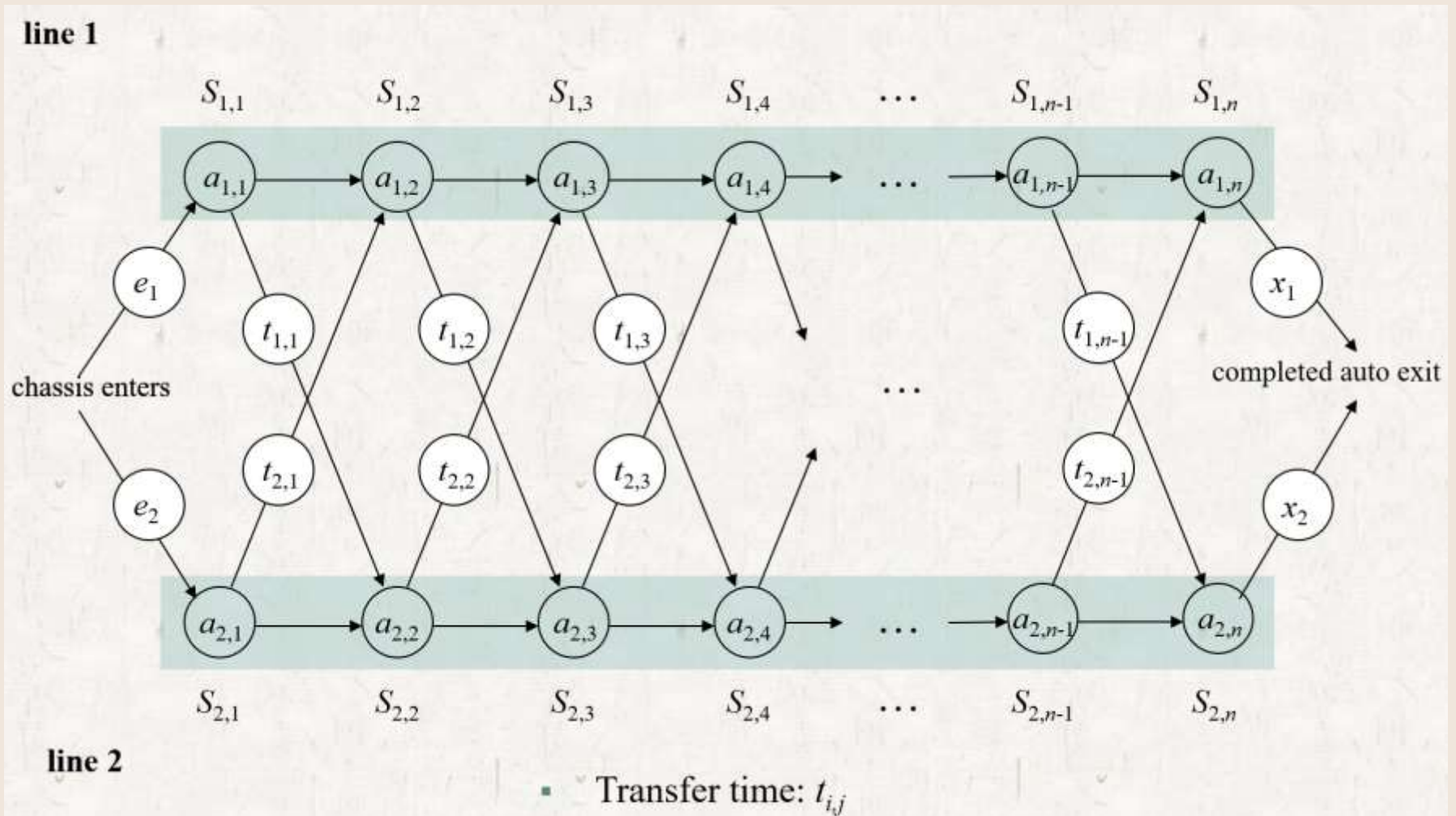


예제 4: Assembly-line scheduling





예제 4: Assembly-line scheduling





예제 4: Assembly-line scheduling



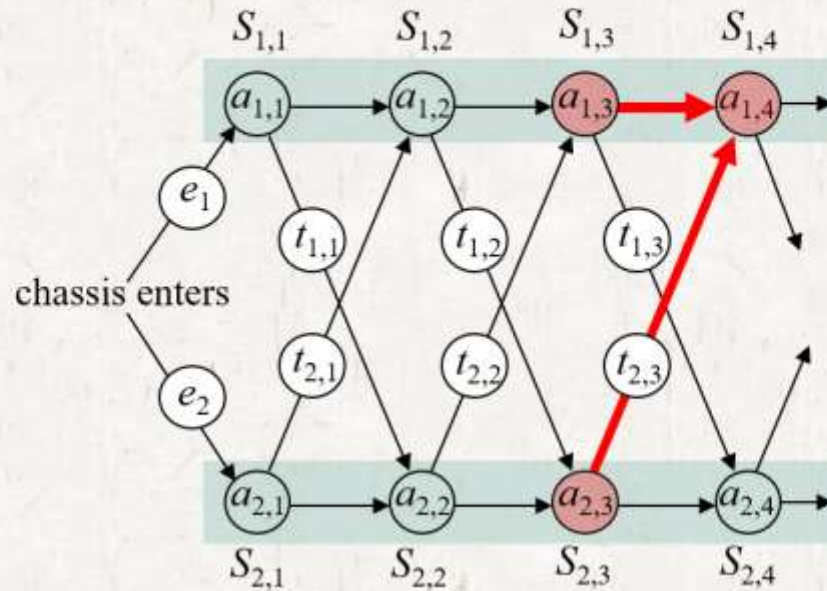
Brute-force approach

- ✓ Enumerate all possible ways and find a fastest way.
- ✓ There are 2^n possible ways: Too many.



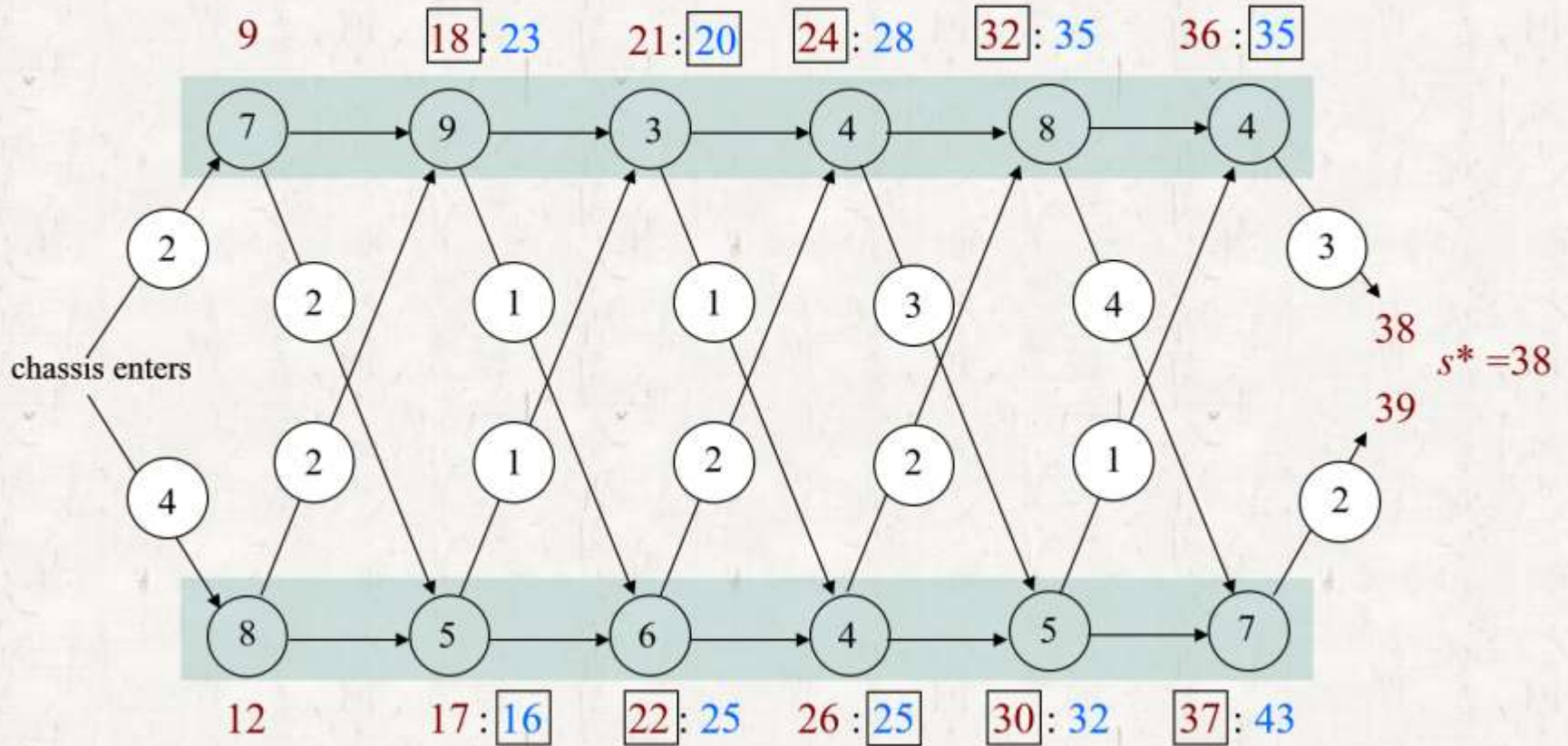
예제 4: Assembly-line scheduling

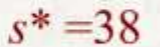
The fastest way to $S_{i,j}$ goes through $S_{1,j-1}$ or $S_{2,j-1}$.





● *Dynamic programming*



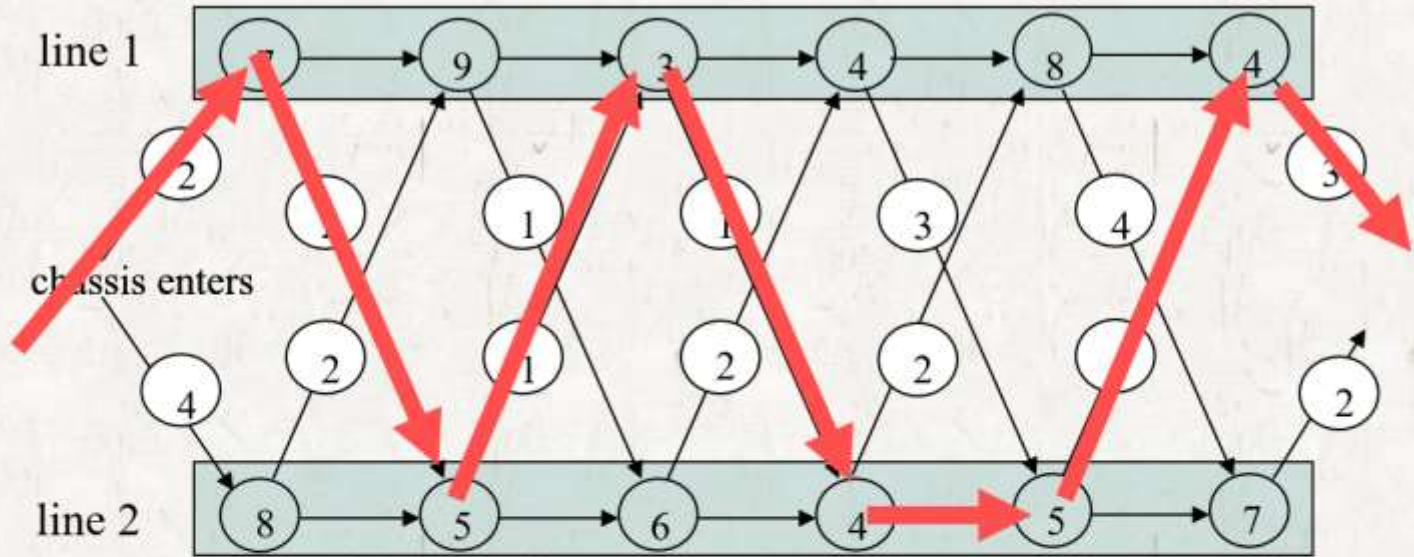


● *Fastest time*

$$s^* = 38$$



예제 4: Assembly-line scheduling



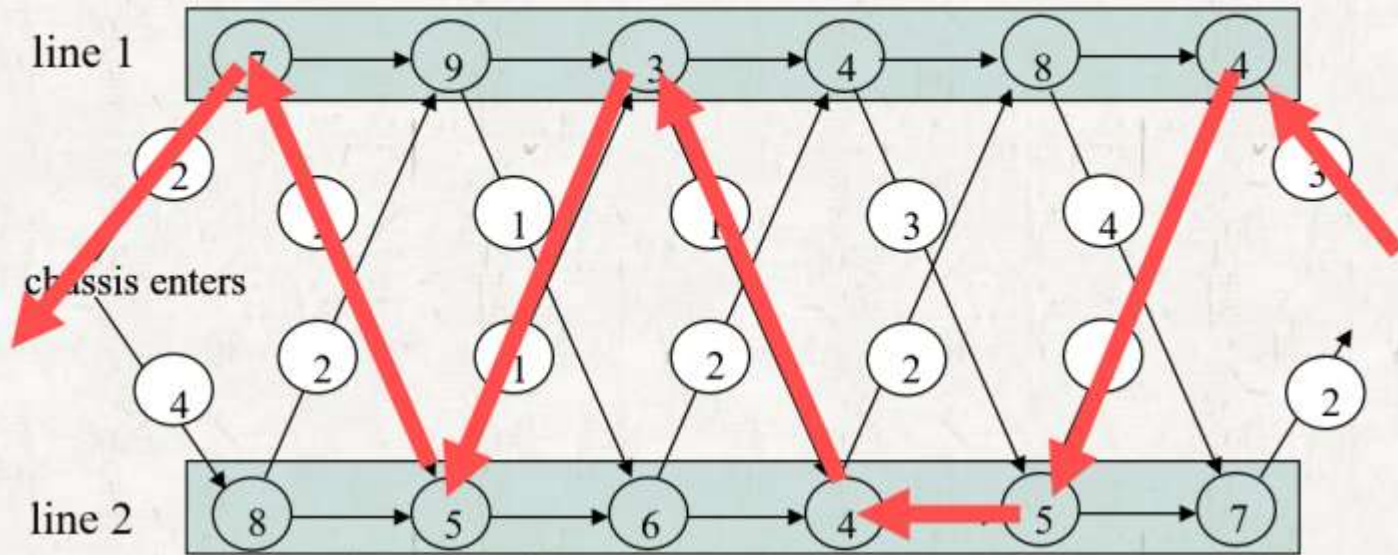
S	1	2	3	4	5	6
1	9	18	20	24	32	35
2	12	16	22	25	30	37

$$s^* = 38$$

● *Fastest way*



예제 4: Assembly-line scheduling



S	1	2	3	4	5	6
1	9	18	20	24	32	35
2	12	16	22	25	30	37

$$s^* = 38$$

L	1	2	3	4	5	6
1	-	1	2	1	1	2
2	-	1	2	1	2	2

$$l^* = 1$$



예제 4: Assembly-line scheduling



Space consumption

- ✓ Table s : $2n$
- ✓ Table l : $2n$
- ✓ $\Theta(n)$ elements in total.



Running time

- ✓ Computing each element requires $\Theta(1)$ time.
- ✓ $\Theta(n)$ time in total



예제 5: 최대 구간 합

🌀 n 개의 정수가 저장된 배열 $\mathbf{a[1]}, \dots, \mathbf{a[n]}$

🌀 연속된 부분 구간 중 합이 최대인 구간은?

	1	2	3	4	5	6	7	8	9	10
a	4	-7	12	-6	5	8	-2	-3	5	6



예제 5: 최대 구간 합



Brute force algorithm

- 모든 가능한 구간에 대해 합을 구한 후 최대 선택
- Time complexity: $\theta(n^3)$



예제 5: 최대 구간 합

개선된 **brute force algorithm**

- $a[1]$ 부터 $a[i]$ 까지의 누적 합을 $S[i]$ 에 저장
- 구간 합: $a[i] + \dots + a[j] = S[j] - S[i-1]$
- Time complexity: $\theta(n^2)$

	1	2	3	4	5	6	7	8	9	10
a	4	-7	12	-6	5	8	-2	-3	5	6
S	4	-3	9	3	8	16	14	11	16	22



예제 5: 최대 구간 합

DP approach

- dp[i]: a[i]를 오른쪽 끝으로 하는 구간의 최대 합
- $dp[i] = \max(dp[i-1] + a[i], a[i])$

		1	2	3	4	5	6	7	8	9	10
a		4	-7	12	-6	5	8	-2	-13	5	6
dp	0	4	-3	12	6	11	19	17	4	9	15



예제 5: 최대 구간 합

DP approach

- dp[i]: a[i]를 오른쪽 끝으로 하는 구간의 최대 합
- $dp[i] = \max(dp[i-1] + a[i], a[i])$

		1	2	3	4	5	6	7	8	9	10
a		4	-7	12	-6	5	8	-2	-13	5	6
dp	0	4	-3	12	6	11	19	17	4	9	15



예제 5: 최대 구간 합

DP approach

- dp[i]: a[i]를 오른쪽 끝으로 하는 구간의 최대 합
- dp[i] = max(dp[i-1] + a[i], a[i])

		1	2	3	4	5	6	7	8	9	10
a		4	-7	12	-6	5	8	-2	-13	5	6
dp	0	4	-3	12	6	11	19	17	4	9	15

- dp[i]를 전체적으로 scan하면서 최대 값을 찾으면 됨
- 최대 합은 찾은 후, 구간은 어떻게 아나?



예제 5: 최대 구간 합

DP approach

- dp[i]: a[i]를 오른쪽 끝으로 하는 구간의 최대 합
- $dp[i] = \max(dp[i-1] + a[i], a[i])$

		1	2	3	4	5	6	7	8	9	10
a		4	-7	12	-6	5	8	-21	13	-5	6
dp	0	4	-3	12	6	11	19	-2	13	8	14
fr		1	1	3	3	3	3	3	8	8	8

Time Complexity: $\theta(n)$



예제 6: RGB Street

- **RGB** 거리에 n 개의 집이 일렬로 있다.
- 각 집은 **Red, Green, Blue** 중 하나의 색으로 칠하려고 한다.
- 단, 이웃한 집과는 같은 색을 칠할 수 없다.
- i 번째 집을 어떤 색으로 칠하는가에 따른 비용이 주어진다.
즉, R_i, G_i, B_i 가 주어진다.
- 조건을 만족하면서 모든 집을 색칠하는 최소 비용은?

집	1	2	3	4	5	6
R	5	4	7	6	7	9
G	6	5	3	4	5	5
B	7	3	6	8	4	4



예제 6: RGB Street

DP approach

- 첫번째 집부터 차례대로 색을 칠해 간다고 가정
- $dp[i][R]$: i 번째 집을 R로 칠하는 데 필요한 최소비용
- $dp[i][G]$, $dp[i][B]$: 유사하게 정의
- $$dp[i][R] = R_i + \min(dp[i-1][B], dp[i-1][G])$$



예제 6: RGB Street

DP approach

- $dp[i][R] = R_i + \min(dp[i-1][B], dp[i-1][G])$
- $dp[i][G] = G_i + \min(dp[i-1][B], dp[i-1][R])$
- $dp[i][B] = B_i + \min(dp[i-1][G], dp[i-1][R])$

집	1	2	3	4	5	6
R	5	4	7	6	7	9
G	6	2	3	5	5	5
B	7	3	6	8	4	5
$dp[i][R]$	5	$4+6=10$	$7+7=14$	$6+11=17$	$7+18=25$	$9+21=30$
$dp[i][G]$	6	$2+5=7$	$3+8=11$	$5+13=18$	$5+17=22$	$5+21=26$
$dp[i][B]$	7	$3+5=8$	$6+7=13$	$8+11=19$	$4+17=21$	$5+22=27$



예제 6: RGB Street

DP approach

- $dp[i][R] = R_i + \min(dp[i-1][B], dp[i-1][G])$
- $dp[i][G] = G_i + \min(dp[i-1][B], dp[i-1][R])$
- $dp[i][B] = B_i + \min(dp[i-1][G], dp[i-1][R])$

집	1	2	3	4	5	6
R	5	4	7	6	7	9
G	6	2	3	5	5	5
B	7	3	6	8	4	5
$dp[i][R]$	5	$4+6=10$	$7+7=14$	$6+11=17$	$7+18=25$	$9+21=30$
$dp[i][G]$	6	$2+5=7$	$3+8=11$	$5+13=18$	$5+17=22$	$5+21=26$
$dp[i][B]$	7	$3+5=8$	$6+7=13$	$8+11=19$	$4+17=21$	$5+22=27$

- Time Complexity: $\theta(n)$



3. 동적 계획법 실전 문제



문제 1: 숫자판 놀이



문제 2: 완전정보 게임



문제 3: 제한된 비트 스트링의 개수



문제 4: 최대 공백 정사각형 (Largest Empty Square)



문제 1: 숫자판 놀이



문제: 각 칸마다 숫자가 적힌 배열이 주어져 있다.

- ✓ 위에서 아래로 내려가면서 거치는 숫자만큼의 상금을 받는다.
- ✓ 단 움직이는 방향은 양쪽으로 최대 1칸씩만 허용한다.
- ✓ 가장 돈을 많이 받을 수 있는 이동 방법은 ?

1	6	5	2	10
4	1	4	-8	-9
-7	2	-9	3	5
3	-6	1	-12	-2
9	8	-2	5	5



문제 1: 숫자판 놀이



top-down path를 고려하는 방법

✓ 모든 가능한 이동 경로의 개수는 3^d

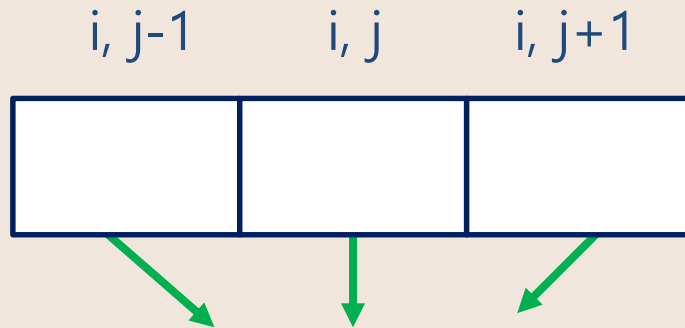


어떤 Cell에 도착하는 방법은 최대 3가지

1	6	5	2	10
10 ₍₄₎	7 ₍₁₎	4	-8	-9
-7	2	-9	3	5



문제 1: 숫자판 놀이



$i+1, j$



$$\text{Sum}(i+1, j) = \max \{ \text{Sum}(i, j-1), \text{Sum}(i, j), \text{Sum}(i, j+1) \} + \text{Cell}[i+1, j]$$



문제 2: 완전 정보 게임



완전 정보 게임

- ✓ 게임의 정보가 참가자 모두에게 공개된 게임
- ✓ 숨어있는 정보는 없다.
- ✓ 누가 먼저 하는가에 따라서 승부는 이미 결정되어 있다.
- ✓ 예) 바둑, 오목, 체스



불완전 정보 게임

- ✓ 확률적으로 정보가 주어진다.
- ✓ 모든 정보가 공개되어 있지는 않다.
- ✓ 예) 화투 놀이, 포커, 스포츠배팅



문제 2.1: 바둑돌 가져가기 1

- ✎ 초기) 바둑돌이 K 개 있다.
- ✎ 동작) 자신의 차례에 1개, 3개, 4개의 돌을 가져갈 수 있다.
- ✎ 승부) 자신의 차례에 동작을 할 수 없으면 그 사람이 패자(loser)가 된다.
- ✎ 부분 문제 $W(i)$: 돌의 갯수가 i 개 일 때, 이기는 방법이 있는 사람.
 - ✓ 기저조건: 돌의 갯수가 1, 3, 또는 4개라면 선수(F)가 항상 이길 수 있다.
 - ✓ 주어진 i 개의 돌에 대해 F가 돌을 1, 3, 또는 4개를 가지고 간 후, 남은 바둑돌에 대한 바둑돌 게임 $W(i-1)$, $W(i-3)$, $W(i-4)$ 에 대해서 S(후수)가 이기는 방법이 하나라도 있다면 $W(i) = F$ 이다.

0	1	2	3	4	5	6	7	8	9	10	11	12
	F	S	F	F	S	F	F	S	F	F	S	F
	W	L	W	W	W	W	L	W	L	W	W	W



문제 2.2: 바둑돌 가져가기 2

- 초기) 바둑돌이 K 개 있다.
- 동작) 자신의 차례에 1개, 3개, 5개의 돌을 가져갈 수 있다.
- 단 앞 사람이 가져간 갯수를 그 다음 사람이 그대로 가져갈 수 없다.
(따라하기 금지 규칙)
- 만일 앞 사람이 3개를 가져가면 그 다음 사람은 1, 5개만 가져갈 수 있다.
- 승부) 자신의 차례에 동작을 할 수 없으면 그 사람이 패자(loser)가 된다.



문제 2.2: 바둑돌 가져가기 2



이 게임을 위한 dp table은 어떻게 만들 것인가?

상대방

가져간 바둑돌

상대방 가져간 후 남은 바둑돌

	0	1	2	3	4	5	6	7	8	9	10	11	12
1		L	L	W	L	W	L	W	L	W	L	W	L
3		W	W	W	L	W	L	W	L	W	L	W	L
5		W	W	W	L	W	L	W	L	W	L	W	L



가져갈 수 있는 돌의 개수가 바뀌면?

- ✓ 1,2,3
- ✓ 1,2,4
- ✓ 1,4,5

Nim Game

두 개의 바둑돌

1, 2, 3개

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		W	W	W	L	W	W	W	L	W	W	W	L	W	W	W
1	W	L	W	W	W	L	W	W	W	L	W	W	W	L	W	W
2	W	W	L	W	W	W	L	W	W	W	L	W	W	W	L	W
3	W	W	W	L				L				L				L
4	L				L				L				L			
5	W	L				L				L				L		
6	W		L				
7	W			L												
8	L				...											
9	W	L														
10	W		L													
11	W															
12	L															

$$dp[n] = !(dp[n-1] \&\& dp[n-2] \&\& dp[n-3])$$



문제 3: 제한된 비트 스트링의 개수

✎ 스트링의 길이가 K 인 이진수 중에서 1이 연속하지 않는 스트링의 개수를 모두 구해보자.

✓ 예) 0001, 0101010, 0001000, 00001000, 1000

✎ 접근방법

✓ 길이 $k-1$ 의 비트 스트링에 대하여 답을 이미 알고 있다고 가정한다.

✓ 길이 k 의 스트링은 2가지로 나뉜다. $K=4$ 일 때를 생각해보자.

✦ 첫 자리가 0으로 시작하는 스트링 0000, 0101, 0010, 0001,...

✦ 첫 자리가 1로 시작하는 스트링 1000, 1010, 1001...








문제 3: 제한된 비트 스트링의 개수

- ✎ $k-1$ 비트의 스트링을 이용해서 k 비트 스트링을 만드는 방법을 생각해 보자.
- ✎ 길이 k 인 비트 스트링 중에서 성질을 만족하는 스트링의 개수를 $S(k)$ 라고 하자.
 - ✓ 두가지 경우가 있다: 첫 자리가 0인 스트링과 첫 자리가 1인 스트링.
 - ✓ 첫 자리가 0인 스트링의 두번째 자리에는 0 또는 1이 올 수 있다.
 - ✓ 첫 자리가 1인 스트링의 두번째 자리에는 반드시 0이 와야 한다.
- ✎ 따라서,
 - ✓ 답을 만족한 모든 $k-1$ 비트 스트링의 맨 앞에 0을 추가하는 방법과 답을 만족한 모든 $k-2$ 비트 스트링의 맨 앞에 10을 추가하는 방법으로 길이 k 인 비트 스트링을 중복되지 않게 모두 표현할 수 있다.




문제 3: 제한된 비트 스트링의 개수

 $k=1$	0	1	null					
 $k=2$	00	01	10					
 $k=3$	000	001	010	100	101			
 $k=4$	0000	0001	0010	0100	0101	1000	1001	1010

 $S(1) = 2$

 $S(2) = 3$

 $S(3) = 5$

 $S(k) = S(k-1) + S(k-2)$

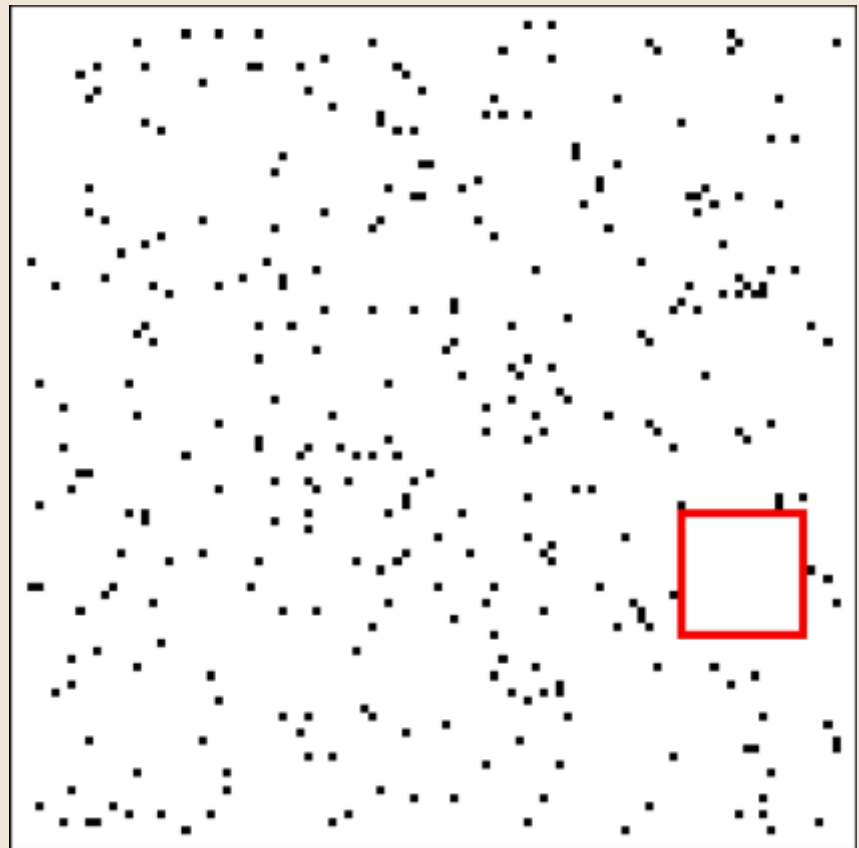


문제 4: 최대 공백 정사각형



문제 정의

- ✓ 주어진 $n \times n$ 흑백 이미지에서 검은 점을 포함하지 않는 가장 큰 빈 (empty) 정사각형 찾기.
- ✓ 최소 비용 운동장 건설
- ✓ 창고 세우기
- ✓ 목 좋은 곳에 상점 열기



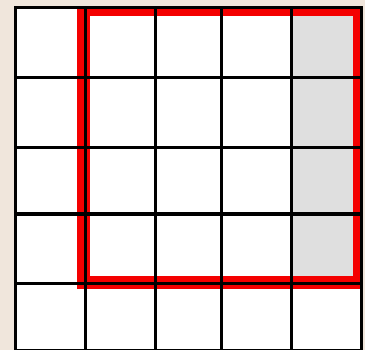
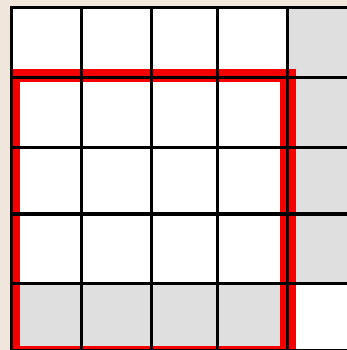
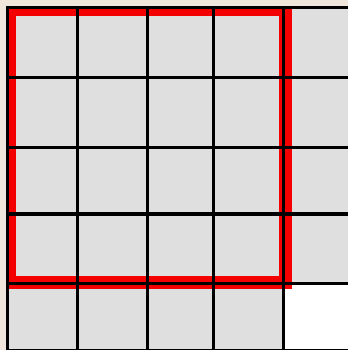
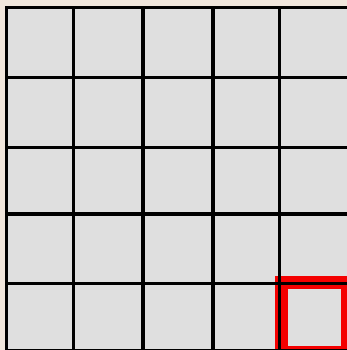


문제 4: 최대 공백 정사각형



재귀적 서술

- ✓ $m \times m$ 의 픽셀로 이루어진 정사각형 S 이 비어있다면 다음이 성립
 - ✦ S 의 가장 우측 하단의 픽셀이 비어있다.
 - ✦ 좌측상단, 좌측하단,우측상단의 크기 $(m - 1) \times (m - 1)$ 정사각형이 비어있다.
 - ✦ 역도 성립.





문제 4: 최대 공백 정사각형



동적계획법으로 해결하기

- ✓ $LES(x,y)$: (x,y) 를 우측하단 꼭지점으로 하는 최대정사각형의 크기 (Largest Empty Square)

- ✓ 다음이 성립:

- ✱ 픽셀 (x,y) 가 비어있지 않다면 $LES(x,y)=0$

- ✱ 첫번째 행 또는 열의 픽셀 (x,y) 가 비어있다면 $LES(x,y)=1$

- ✱ 나머지 픽셀 (x,y) 가 비어있다면

$$LES(x,y) = \min \{ LES(x-1,y-1), LES(x,y-1), LES(x-1,y) \} + 1$$

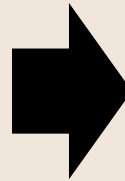


문제 4: 최대 공백 정사각형



예제

✓ 4x4 pixel image



1	1	1	1
1	0	1	2
1	1	1	0
1	2	2	1



문제 4: 최대 공백 정사각형



동적계획법으로 해결하기

✓ 알고리즘 복잡도: $\Theta(n^2)$

IterLES(n)

1. for $x = 1$ to n
2. for $y = 1$ to n
3. if (x, y) not empty
4. LES[x, y] = 0
5. else if $x = 1$ or $y = 1$
6. LES[x, y] = 1
7. else
8. LES[x, y] = $\min(\text{LES}[x-1, y-1], \text{LES}[x, y-1], \text{LES}[x-1, y]) + 1$



4. 서열 데이터

✎ 4.1 최장 공통 부분 서열 문제 (Longest Common Subsequence Problem)

✎ 4.2 Sequence Alignment (global, local, semi-global)



4.1 가장 공통 부분 서열 (LCS)

두 개의 DNA 순서열 (sequence)이 있을 때,
이 두 개가 얼마나 비슷한가를 측정하는 일이 자주 발생한다.

예를 들어,
 $X = \text{ACCGGTCGACGTAAGCCGGCCAA}$,
 $Y = \text{TTTCCCACCTCGTGTCGACGTGTAAGCCTTAAGGCCAA}$ 가 있다.

이 때 이 둘이 얼마나 유사한가를 측정할 때,
둘의 LCS를 구하여 이것이 길면 길수록 둘은 더 유사하다고 할 수 있다.



4.1.1 최장 공통 부분 서열의 정의

- ✎ LCS를 정의하기 위해, 먼저 부분 서열 (subsequence)을 정의한다.
- ✎ 예를 들어, $X = \text{ACCGGT}$ 가 있을 때, ACGT , CCGG , AT , CGT 등은 X 의 부분 서열이 된다.
(즉, 부분 서열에 나오는 문자들이 반드시 순서대로 원래 순서열에 나와야 한다.)
그러나, GCT , CA 등은 아니다.
- ✎ X 과 Y 가 주어질 때, 공통 부분 서열은 X 의 부분 서열이면서, 동시에 Y 의 부분 서열이 되는 것을 말한다.
- ✎ 공통 부분 서열들 가운데 가장 긴 것이 LCS(Longest Common Subsequence)이다.



4.1.2 LCS 문제의 단순 해결방법

- ✎ LCS 문제 : $X = (x_1, \dots, x_m)$ 과 $Y = (y_1, \dots, y_n)$ 이 주어질 때 이들의 LCS를 구하시오.
- ✎ 단순한 방법 : X 의 모든 부분 서열 각각에 대해서, 이것이 Y 의 부분 서열 인가를 조사한다.
- ✎ X 의 부분 서열 총 수는 2^m 이므로 이 방법은 적어도 $\Omega(2^m)$ 시간이 걸린다.



4.1.3 LCS의 최적화 재귀식

<동적계획법으로 해결하기 위해서>

✎ 단계 1: 최적해의 구조 파악

✎ 모든 $i (0 \leq i \leq n)$ 에 대해, X 의 prefix를 $X_i = (x_1, \dots, x_i)$ 로 정의하고 비슷하게 Y_i 도 정의한다. ($X = X_m, Y = Y_n$)

$Z = (z_1, \dots, z_k)$ 를 X 와 Y 의 LCS라 하자. 그러면,

1. If $x_m = y_n$,
then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1} .
2. If $x_m \neq y_n$,
then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y_n .
3. If $x_m \neq y_n$,
then $z_k \neq y_n$ implies that Z is an LCS of X_m and Y_{n-1} .



4.1.3 LCS의 최적화 재귀식

<동적계획법으로 해결하기 위해서>

 단계 2: 최적 해에 대한 순환 정의

- 전 장에서 설명한 것을 쉽게 설명하면,

만약, $z_k = x_m = y_n$ 이면

X_{m-1} 와 Y_{n-1} 의 LCS를 구하고, 뒤에 x_m 를 덧붙이면 되고,

만약, $x_m \neq y_n$ 이면

(1) X_{m-1} 과 Y_n 의 LCS, (2) X_m 와 Y_{n-1} 의 LCS를
모두 구하여 그 중에 긴 것을 선택하면 된다.



4.1.3 LCS의 최적화 재귀식

<동적계획법으로 풀기 위해서>

 단계 2: 최적 해에 대한 순환 정의

- $LCS(i, j)$ 를 X_i 와 Y_j 의 LCS의 길이로 정의하면,
우리가 최종적으로 구해야 하는 값은 $LCS(m, n)$ 이다.
- $LCS(i, j) = 0,$ if $i = 0$ or $j = 0$
- $LCS(i, j) = LCS(i - 1, j - 1) + 1,$ if $x_i = y_j$
- $LCS(i, j) = \max \{ LCS(i, j - 1), LCS(i - 1, j) \}$ if $x_i \neq y_j$



4.2 Sequence Alignment



Alignment 란 무엇인가 ?



다양한 Alignment 문제들

- ✓ Global, Local, Semi-local Alignment Problems
- ✓ Global alignment는 두 개의 문자열을 gap symbol ("-")을 포함하도록 확장하여 같은 길이의 문자열로 만드는 문제이다.
- ✓ 대응되는 문자들의 쌍에 대하여 score가 주어진다.
- ✓ 대응되는 문자들의 쌍들의 score 합이 최대가 되는 alignment를 구하는 것이 목적이다.
- ✓ Alignment 문제들은 대부분 동적계획법으로 해결된다.



4.2 Sequence Alignment



Global Alignment

두 개의 스트링 acbcbd와 cadbd가 있을 때 두 스트링은 얼마나 비슷한가?

- ✓ 두 스트링의 유사도를 측정하는 기본적인 방법은 두 스트링을 확장하여 같은 길이로 만든 후, 같은 위치에 있는 두 문자들을 서로 비교하는 것이다.
여기서, 확장한다는 것은 필요한 곳에 적당한 수의 빈칸 (-로 표시)을 삽입하는 것을 말한다.
- ✓ 아래 그림과 같이, acbcbd에 빈칸을 하나 넣어 acbcbd-로 확장하고, cadbd에 빈칸을 두 개 넣어 -c-adbd로 확장을 하여 두 확장된 스트링을 비교한다.

a	c	b	c	d	b	-
-	c	-	a	d	b	d



4.2 Sequence Alignment



Global Alignment

- ✓ 이제, 같은 위치에 있는 문자 쌍에 대한 유사도 점수를 하나씩 계산하여, 이들을 모두 합하면 전체의 유사도가 된다.
- ✓ $\sigma(x, y)$ 를 문자 x 와 문자 y 에 대한 유사도라 하자. $\sigma(x, y)$ 는 함수로 주어질 수도 있고, 표로 주어질 수도 있다.
- ✓ 일반적으로, $\sigma(x, y)$ 는 $x = y$ 이면 점수가 많고, $x \neq y$ 이면 점수가 적다.
좀 더 자세히, $x \neq y$ 일 때 x, y 둘 다 빈칸이 아닌 경우가 둘 중 하나라도 빈칸인 경우 보다 점수가 더 많다.
- ✓ 특히, 둘 다 빈칸일 경우는 두 스트링의 같은 자리를 빈칸으로 확장한 것이므로 유사도의 값을 증가시켜서는 안 된다. 따라서, $\sigma(-, -)$ 는 0점이나 음수 점수이어야 한다.



4.2 Sequence Alignment



Global Alignment

- ✓ 두 스트링 acbcdb와 cadbd를 예제처럼 확장한 경우라면, 전체 유사도는 $\sigma(a, -) + \sigma(c, c) + \sigma(b, -) + \sigma(c, a) + \sigma(d, d) + \sigma(b, b) + \sigma(-, d)$ 가 된다.
- ✓ 물론, 같은 스트링들에 대해서, 빈칸을 다른 곳에 넣어서 다르게 확장을 하면, 유사도가 달라질 수도 있다.
- ✓ 우리가 원하는 것은 두 스트링의 유사도가 가장 커지도록 확장을 하는 것이 목적이고, 이때의 유사도 점수가 두 스트링의 유사도가 된다.

a	c	b	c	d	b	-
-	c	-	a	d	b	d



4.2 Sequence Alignment



Global Alignment

- ✓ 기저조건 :

$$V(0,0) = 0$$

$$V(i, 0) = V(i - 1, 0) + \sigma(a_i, -)$$

$$V(0, j) = V(0, j - 1) + \sigma(-, b_j)$$

- ✓ 부분 문제들의 관계 :

$$V(i, j) = \max \{ \begin{aligned} &V(i - 1, j - 1) + \sigma(a_i, b_j), \\ &V(i - 1, j) + \sigma(a_i, -), \\ &V(i, j - 1) + \sigma(-, b_j), \\ &\} \end{aligned}$$

- ✓ $V(m, n)$ 는 이 두 스트링의 유사도이며, 실제 확장된 스트링들은 위의 식에 의해서 역추적하여 구할 수 있다.



4.2 Sequence Alignment

✎ Q) P1과 같은 matrix일 때 다음 두 Sequence의 optimal global alignment를 구하시오.

✓ A: GTTCA

✓ B: ATTAGC

	A	C	G	T	-
A	5	-1	-2	-1	-3
C	-1	5	-3	1	-4
G	-2	-3	5	-2	-2
T	-1	1	-2	5	-1
-	-3	-4	-2	-1	0

표 P1 : Scoring Matrix



4.2 Sequence Alignment

	A	C	G	T	-
A	5	-1	-2	-1	-3
C	-1	5	-3	1	-4
G	-2	-3	5	-2	-2
T	-1	1	-2	5	-1
-	-3	-4	-2	-1	0

	-	A	T	T	A	G	C
-	0	-3	-4	-5	-8	-10	-14
G	-2						
T	-3						
T	-4						
C	-8						
A	-11						

$$V(0,0) = 0$$

$$V(i,0) = V(i-1,0) + \sigma(a_i, -)$$

$$V(0,j) = V(0,j-1) + \sigma(-, b_j)$$

$$\max \{ \begin{aligned} &V(i-1, j-1) + \sigma(a_i, b_j), \\ &V(i-1, j) + \sigma(a_i, -), \\ &V(i, j-1) + \sigma(-, b_j), \\ &\} \end{aligned}$$



4.2 Sequence Alignment

	A	C	G	T	-
A	5	-1	-2	-1	-3
C	-1	5	-3	1	-4
G	-2	-3	5	-2	-2
T	-1	1	-2	5	-1
-	-3	-4	-2	-1	0

	-	A	T	T	A	G	C
-	0	-3	-4	-5	-8	-10	-14
G	-2	-2	-5	-6	-7	-3	-13
		-5	-3	-4	-7	-9	-7
T	-3	-3	3	2	-5	-9	-2
		-6	-4	-4	-1	-3	-7
T	-4	-4					
		-7					
C	-8	-5					
		-8					
		-11					
A	-11	-3					
		-8					
		-14					

$$\max\{ V(i-1, j-1) + \sigma(a_i, b_j), \\ V(i-1, j) + \sigma(a_i, -), \\ V(i, j-1) + \sigma(-, b_j), \\ \}$$



6. 동적 계획법 구현상의 주의사항



6.1 부분문제의 답



6.2 Recursion의 효율성



6.3 Memoization



6.4 의사-다항 시간 알고리즘



6.1 부분문제의 답

- ✎ 부분 문제에 대해 하나의 답이 아니라 다수의 답을 저장해야 할 경우가 있다.
- ✎ 예를 들어, 다음 식에서 이진 연산자들의 계산 순서를 마음대로 조정해서 최대값이 나오도록 하려고 한다.
- ✎ 이 때, 각 부분 문제에 해당하는 배열의 원소는 최대값과 최소값을 동시에 저장하고 있어야 한다.

-3	*	-9	+	2	-	8	*	5	
----	---	----	---	---	---	---	---	---	--

- ✎ 왼쪽 * 연산자를 마지막에 적용할 경우, 연산자의 좌측이 음수 -3이기 때문에 우측에는 최소값을 갖도록 연산의 순서를 정하여야 한다.
+ 연산자를 마지막에 적용할 경우, 양쪽 모두 최대값을 갖도록 연산의 순서를 정하여야 한다.



6.2 Recursion의 효율성

- 재귀식이 구성되었다고 바로 그대로 구현하면 같은 함수를 중복으로 호출하여 너무 많은 자원(메모리,CPU)을 사용한다.
- $T(n)$: fibo(n)을 계산하기 위하여 fibo() 함수를 호출하는 횟수

$$T(0)=1;$$

$$T(1)=1;$$

$$T(n)=T(n-1)+T(n-2)+1 \quad \text{for } n \geq 2$$

$$> 2 \times T(n-2) \quad \text{since } T(n-1) > T(n-2)$$

$$> 2^2 \times T(n-4)$$

$$> 2^3 \times T(n-6)$$

...

$$> 2^{n/2} \times T(0)$$

$$= 2^{n/2}$$



6.3 메모이제이션

- 메모이제이션(memoization)은 이전에 계산한 값을 메모리에 저장해서 매번 다시 계산하지 않도록 하여 전체적인 실행속도를 빠르게 하는 방법으로 동적 계획법의 핵심이 되는 기술이다.
- 피보나치 수를 구하는 알고리즘에서 $\text{fibonacci}(i)$ 의 값을 계산하자마자 저장하면, 실행시간을 $O(n)$ 으로 줄일 수 있다.

memo를 위한 배열을 할당하고, 모두 0으로 초기화 한다

$\text{memo}[0]$ 을 0으로 $\text{memo}[1]$ 는 1로 초기화 한다

fibonacci(n)

IF $n > 2$ AND $\text{memo}[n] = 0$

$\text{memo}[n] \leftarrow \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$

RETURN $\text{memo}[n]$



6.4 의사-다항 시간 알고리즘

- 어떤 계산 문제의 시간복잡도가 데이터의 개수가 아니라 데이터의 값(Value)에 따라서 결정될 때 이 문제를 의사-다항식 문제라고 한다.
- 예를 들어, Sum of Subset 문제를 동적계획법으로 해결해 보자.
 - ✓ $S = \{ 3, 4, 9, 10, 11, 31, 40 \}$ 이 주어질 때 이 중 일부를 선택해서 그 합이 $k=100$ 이 되는 부분 집합을 찾으시오.
 - ✓ $S = \{ 13, 43, 59, 110, 311, 431, 540 \}$ 이 주어질 때 이 중 일부를 선택해서 그 합이 $k=1000$ 이 되는 부분 집합을 찾으시오.
- 동적계획법을 사용할 때, 위 두 문제는 모두 $N=7$ 개의 데이터를 갖지만 시간복잡도는 $O(kN)$ 이다.



6.4 의사-다항 시간 알고리즘

- ✎ $T(i, k)$: 첫 원소부터 i 번째까지의 원소들 중에서 몇 개를 선택하여 그 합이 k 가 되는 부분집합이 존재하는지의 여부 (Y/N)
- ✎ 두가지 경우가 있다: i 번째 원소를 포함하거나, 포함하지 않거나.
- ✎ $T(i, k) = T(i-1, k) \text{ or } T(i-1, k - E[i])$
 - ✓ $E[i]$ 는 i 번째 원소의 값
 - ✓ $T(7, 67)$ 은 $T(6, 67)$ 또는 $T(6, 67-40)=T(6,27)$ 이 가능한지 여부에 의해 결정된다.
 - ✓ 따라서, 문제 해결을 위해 사용하는 테이블의 크기는 $O(kN)$ 이다.



동적계획법 설계법의 결론

- ✎ 최적 부분 문제 구조를 만족하는지를 확인해야 한다.
- ✎ 부분 문제를 정의하여 부분 문제들에 대한 답이 모두 있을 때 전체의 답을 구할 수 있는지 생각한다.
- ✎ 답을 몇 가지 disjoint한 경우로 나눌 수 있는지 생각한다.
 - ✓ 답에서 x 가 포함된 경우, 포함되지 않은 경우
- ✎ 답이 discrete한 경우 다차원 테이블로 표현할 수 있는지를 생각한다.
- ✎ 가능한 많은 문제를 해결해 본다.