Dynamic Programming (DP)

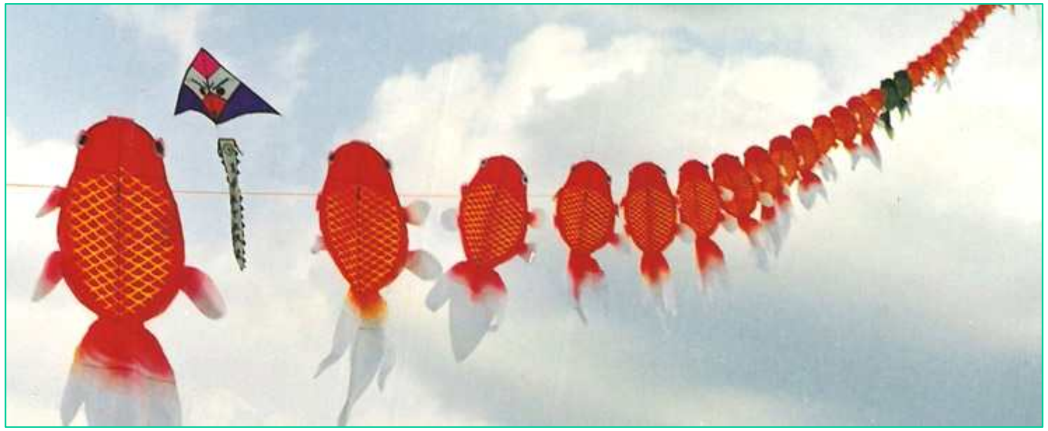
동적 프로그래밍(DP) 활용되는 여러 상황 이해 & 적용 방법에 익숙해 지기

01. 퀴즈 풀이 & 예습 내용 복습

02. Dijkstra's Algorithm 활용해 DP 개념 복습

03. P(N)과 P(N-i)들 간의 관계 찾기 연습

04. 실습 문제 풀이 & 질의 응답



왜 동적 프로그래밍(DP)을 자세히 배우는가?

- DP는 많은 **실제 어플리케이션에서 자주 사용**되며 효율성(수행속도 & 메모리 사용량)을 높이는데 중요한 역할을 함

- 각 DP 문제마다 많이 달라 보여 DP의 **개념이나** DP를 **적용하는 방법에 익숙해지는데** 다른 알고리즘에 비해 **시간이 많이 걸림**

- 여러 DP 문제들을 (기존에 보았던 문제라도 풀이를 외우기보다는) **천천히 잘 이해하며** 함께 **풀어 보기 → DP 개념 & 적용 과정에 익숙**해 지기

Dynamic Programming (DP)

동적 프로그래밍(DP) 활용되는 여러 상황 이해 & 적용 방법에 익숙해 지기

01. 퀴즈 풀이 & 예습 내용 복습

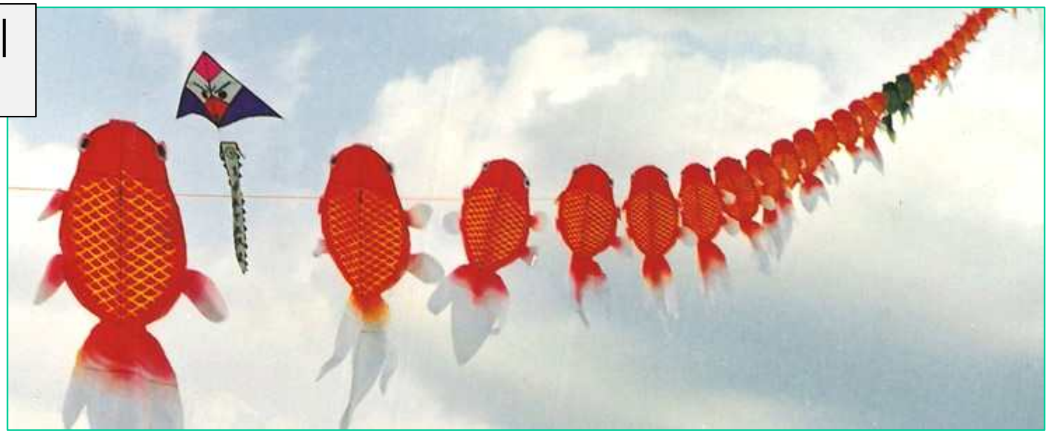
02. Dijkstra's Algorithm 활용해 DP 개념 복습

03. P(N)과 P(N-i)들 간의 관계 찾기 연습

04. 실습 문제 풀이 & 질의 응답

현재 문제에 대한 해답

이전 문제에 대한 해답



수업 전 예습 → 문제풀이/실습/질의응답 (플립 러닝, 거꾸로 학습)

수업 전

문서 형태
개요/이론 학습

수업 전

개요 관련 퀴즈 수행
스스로 학습 & 시행착오

매 수업 전날까지
퀴즈 완료

40~50분

수업

퀴즈 풀이
개요 복습

수업

응용/심화 내용 학습
질의 응답

예습 자료 학습 하였음
가정하고 진행

수업

실습 과제 (프로그래밍)
질의 응답

~3시간

이론 학습 (1.5~2시간) + 실습(1~1.5 시간) = 총 ~3시간

실습과제 수업 중 완료 못한 경우
당일 저녁 or 다음날까지
각자 수행해 제출

수업 진행 관련 유의 사항 #1

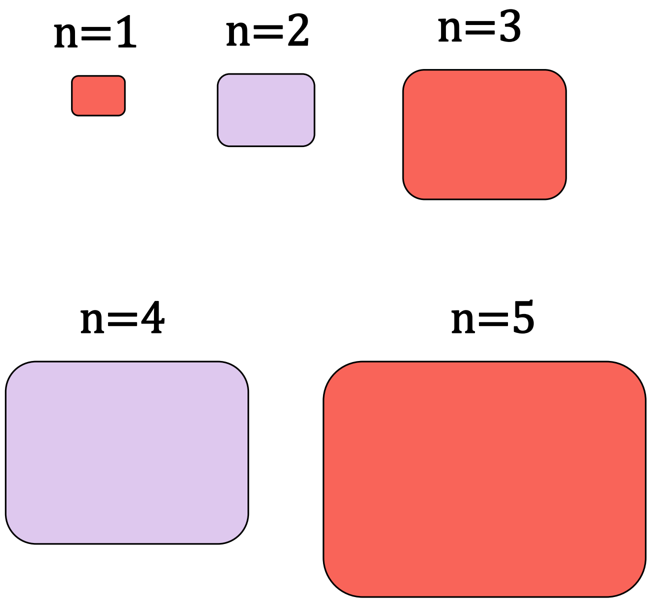
- **질문**을 할 때는 **스스로 답을 생각**해 보는 습관을 들이세요. 내용을 이해하며 배우는데 도움이 됩니다.

- 실습 내용을 보여줄 때는 **따라서 직접 실습**해 보세요. 강의 후 실습 문제 풀이에 도움이 됩니다.

수업 진행 관련 유의 사항 #2

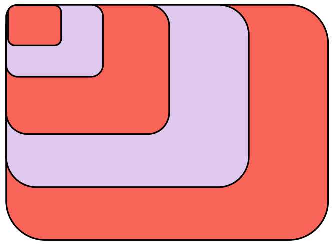
- 한 학기 동안 내용이 계속 쌓여가므로 빠지지 않고 잘 따라오는 것이 중요
- 수업 후 또는 실습 시간에 궁금한 것 꼭 질문해 이해하고 가는 습관 들이기

NON-DYNAMIC PROGRAMMING



DYNAMIC PROGRAMMING

n=1 ⇨ 5



큰 문제(에 대한 해답) 내에서 작은 문제(에 대한 해답) 관찰

DP 사용하면 (**작은 문제와 큰 문제 간 관계** 관찰해 활용하면)
문제에 대한 답을 더 **체계적**으로 정확하게 찾을 수 있음
코드도 더 **빠르고** and/or **간단**해짐

P(n) (size n인 문제의 해답)이 **P(n-1) (직전 문제의 해답)을 활용**하는 경우

n	P(n)=n!	
1	1	1
2	2	1 x 2
3	6	1 x 2 x 3
4	24	1 x 2 x 3 x 4
5	120	1 x 2 x 3 x 4 x 5
6	720	1 x 2 x 3 x 4 x 5 x 6
…		

n	P(n)=n!	
1	1	1
2	2	x 2
3	6	x 3
4	24	x 4
5	120	x 5
6	720	x 6
…		

DYNAMIC PROGRAMMING

P(n) = n x P(n-1)

서로 다른 크기 문제 간 관계 활용해 **답을 더 빠르게** 구할 수 있음 (곱셈 횟수 감소)

P(n) (size n인 문제의 해답)이 P(n-1) (직전 문제의 해답)을 활용하는 경우

S	3^S % 7
0	1
1	3
2	⇐ 3^2 % 7
3	⇐ 3^3 % 7
4	⇐ 3^4 % 7
5	⇐ 3^5 % 7
6	⇐ 3^6 % 7
7	⇐ 3^7 % 7
…	

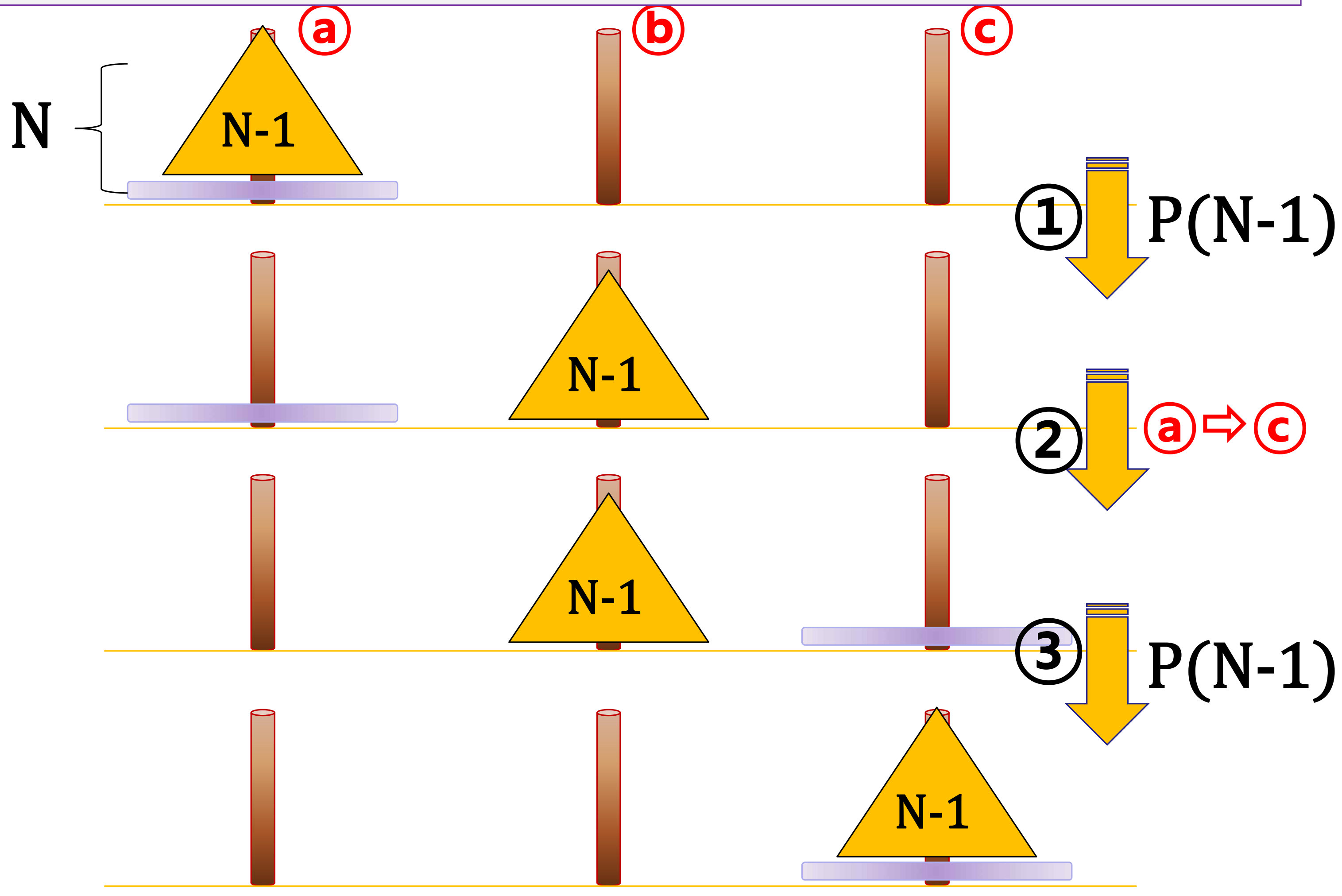
S	$P(s)=3^S$ % 7
0	1
1	3
2	3x3 %7 = 2
3	3x2 %7 = 6
4	3x6 %7 = 4
5	3x4 %7 = 5
6	3x5 %7 = 1
7	3x1 %7 = 3
…	

DYNAMIC PROGRAMMING

P(n) = (3 x **P(n-1)**) % 7

서로 다른 크기 문제 간 관계 활용해 **답을 더 빠르게** 구할 수 있음 (곱셈 횟수 감소)

P(n)이 P(n-1)을 **두 번 이상 사용**하는 경우



P(N) : P(N-1) ⇨ 'ⓐ→ⓒ' ⇨ P(N-1)

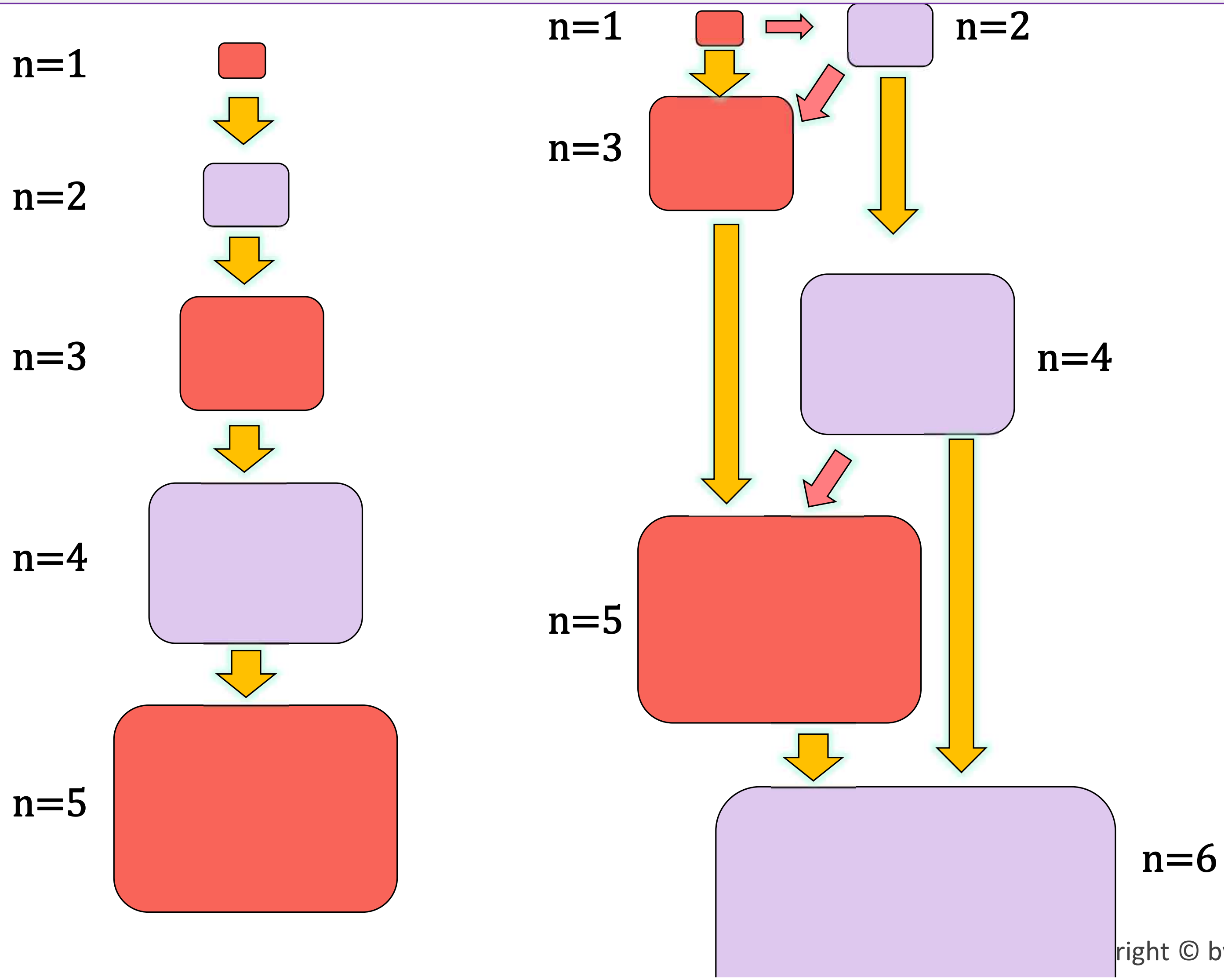
```python
def hanoi(n, fromPeg, viaPeg, toPeg):
    if n==0: pass
    else:
        hanoi(n-1, fromPeg, toPeg, viaPeg) # n-1개 원판을 from -> via로 이동
        print(f"{fromPeg} -> {toPeg}") # 가장 큰 원판을 from -> to로 이동
        hanoi(n-1, viaPeg, fromPeg, toPeg) # n-1개 원판을 via -> to로 이동

if __name__ == "__main__":
    hanoi(6, 'A', 'B', 'C')
```

```
A -> B
A -> C
B -> C
A -> B
C -> A
C -> B
A -> B
A -> C
B -> C
B -> A
C -> A
B -> C
A -> B
A -> C
B -> C
A -> B
C -> A
C -> B
A -> B
C -> A
B -> C
B -> A
C -> A
C -> B
A -> B
A -> C
B -> C
A -> B
C -> A
C -> B
A -> B
A -> C
B -> C
B -> A
C -> A
B -> C
A -> B
A -> C
B -> C
B -> A
C -> A
C -> B
A -> B
C -> A
B -> C
B -> A
C -> A
B -> C
A -> B
A -> C
B -> C
A -> B
C -> A
C -> B
A -> B
A -> C
B -> C
B -> A
C -> A
B -> C
A -> B
A -> C
B -> C
```

여러 다른 n에 대한 복잡한 결과를
간단한 코드로 정확하게 생성 가능

P(n)이 P(n-i) for i>=1 을 활용하며 또한 하나 이상의 P(n-i)을 활용하는 경우



Fibonacci Sequence: P(n) = P(n-1) + P(n-2)

"A baby rabbit pair grows into adults in one month,
then they give birth to a pair of (male and female) rabbits every month"

Month M	Adult Rabbit Pairs (Can produce babies)	Baby Rabbit Pairs	Total # of Pairs P(M)
0			1
1			1
2			2
3			3
4			
5			
	...		

여러 다른 n에 대한 복잡한 결과를 **간단한 코드로** 정확하게 생성 가능

$P(n) = P(n\text{-}3) + \{3\}$ or $P(n\text{-}5) + \{5\}$

N(Price to pay)	P(N) (Coin combinations)
8	3 5
9	3 3 3
10	5 5
11	3 3 5
12	3 3 3 3
13	
14	
15	
…	

+3

+5

P(n) 얻기 위해
P(n-1), P(n-3), P(n-5)와의 관계
모두 활용 가능한데,
P(n-3), P(n-5)과 관계가 좀 더 간단

여러 다른 n에 대한 복잡한 결과를
간단한 코드로 정확하게 생성 가능

DP 정리: 활용할 수 있는 관계 다양함, 체계적으로 문제 풀이 가능

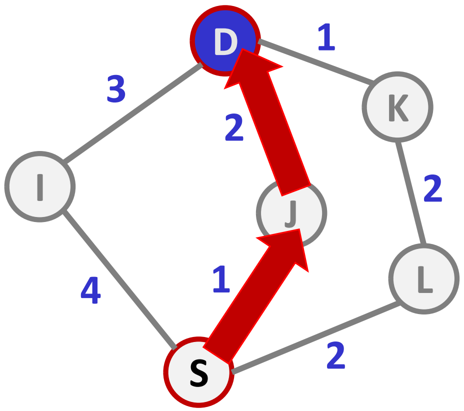
- **활용할 수 있는 관계는 문제마다 다르고 다양함**
  - P(n)이 P(n-1) 활용, 둘 이상의 P(n-1) 활용
  - P(n-1)과 P(n-2) 활용
  - P(n-3), P(n-5) 활용 등

- **이러한 관계 잘 관찰해 활용하면**
  - 문제 풀이가 더 빠르고 체계적이고 정확하게 되며
  - 코드도 간단해짐

P(n) depends on **P(n-i) for arbitrary i>=1**

Dijkstra's Algorithm to Find Shortest Route

- **Shortest route:**
  Minimum-cost route

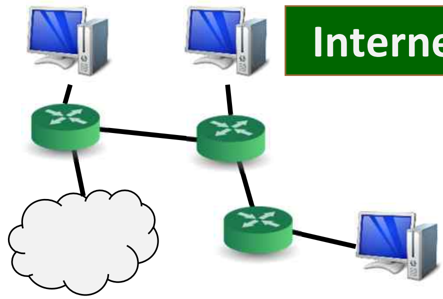


(N) : Node

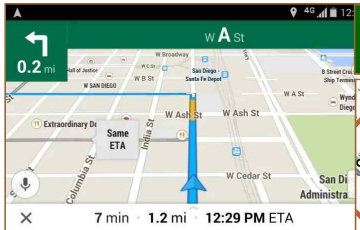
c : Link and its cost (e.g., distance)

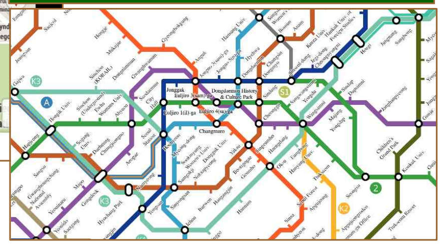
- **Applications**

Internet Routing

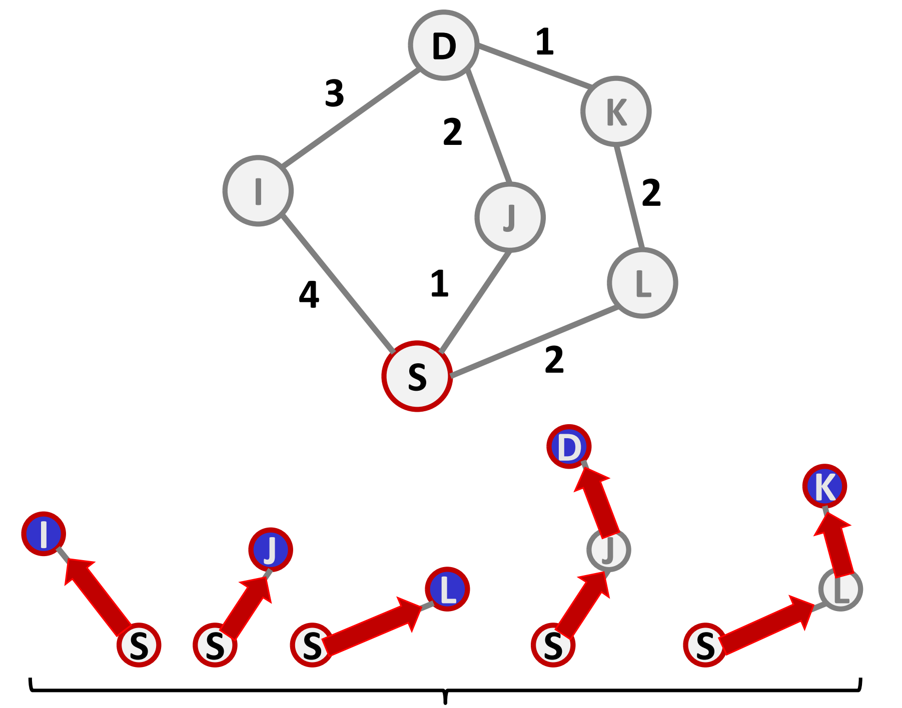


Navigation Systems





Problem Definition: Shortest Paths from Single Source to Multiple Destinations



Shortest Routes from S to I, J, L, D, and K

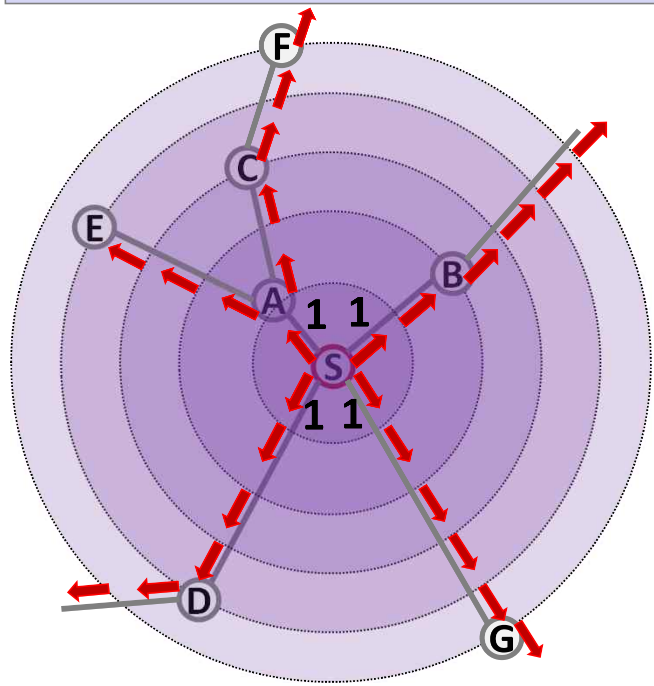
- (Q) WHY compute for ALL destinations? (A) Useful!

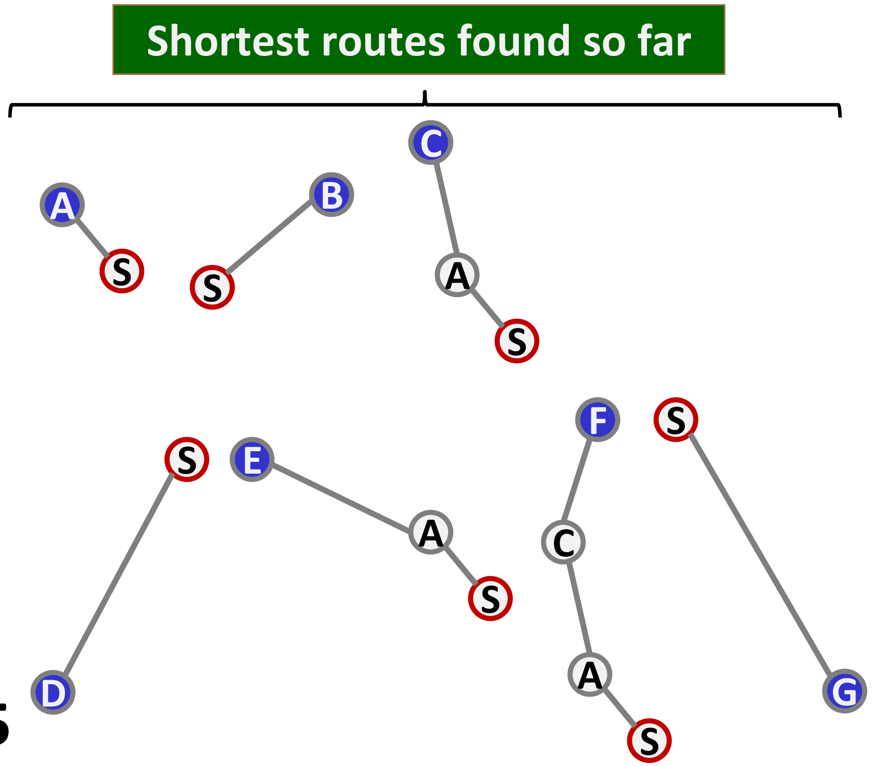
Routing Table	
Destination IP	Output Link #
1.*.*.*	1
2.*.*.*	2
3.*.*.*	3
4.*.*.*	4
5.*.*.*	5
...	...

- Assumption: Link costs are positive integers

Explore ① toward All Directions ② at Same Speed

- If we reach a new destination α, record the route to α as the shortest to α

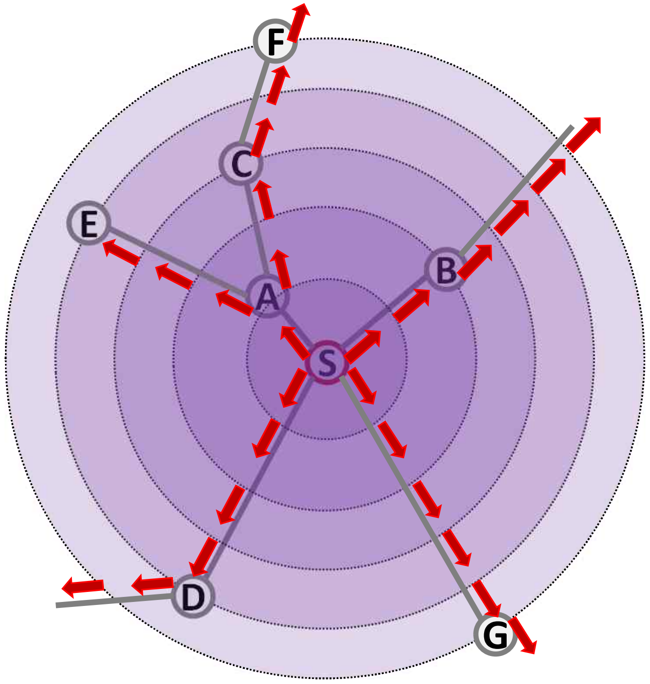




Explored all routes of length **1 2 3 4 5**

Similar to Unrolling Carpet ① to All Directions ② at Same Speed

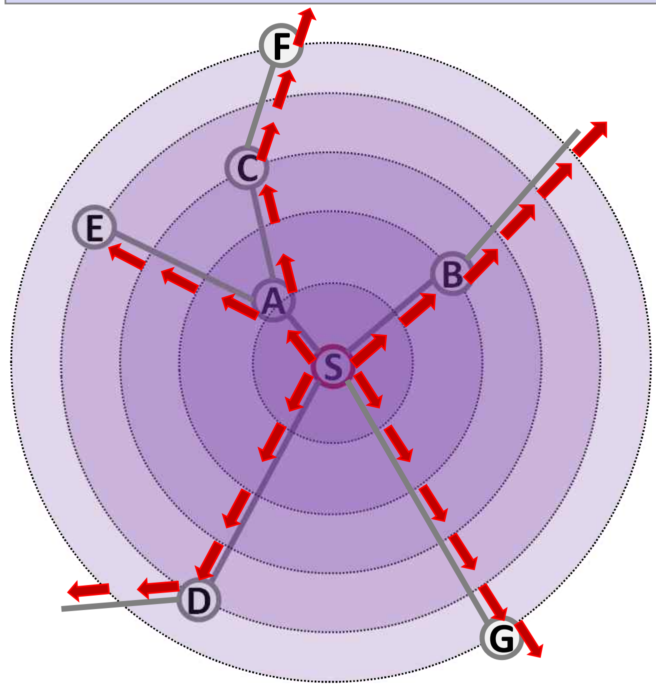
■ **Find shortest route of increasing length (i.e., length 1→2→3→4→...)**



Explored all routes of length 1 2 3 4 5

Dynamic Programming:
Always Go Forward (Outward) and Never Come Back

- We always go further from source and do NOT re-examine same route



e.g., we advance to distance 1,
then we advance outward to distance 2
but we do NOT come back to source

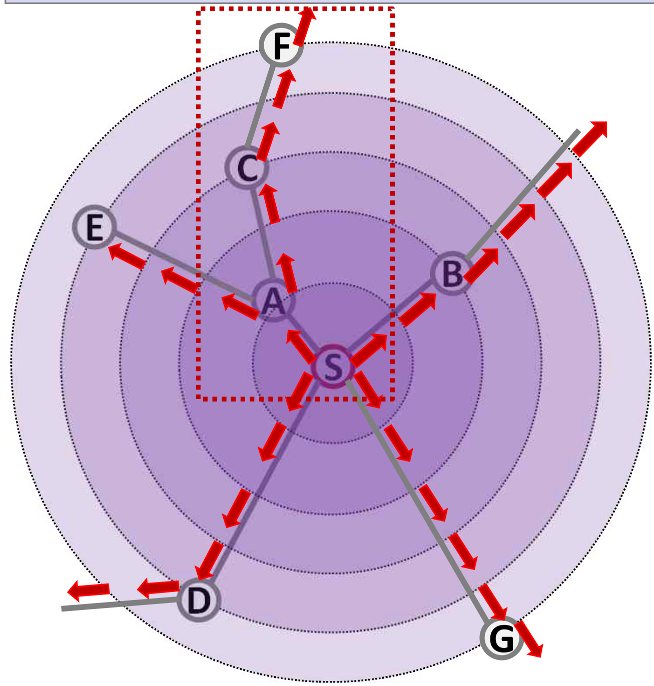
e.g., we advance to distance 2,
then we advance outward to distance 3
but we do NOT come back to distance 1

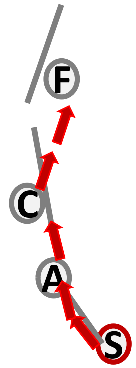
…

Explored all routes of length 1 2 3 4 5

Dynamic Programming:
Newly found Route = Previously found Route + New links

- Take an example on S→A→C→F





Shortest route to F
= S→A→C + new link to F

Shortest route to C
= S→A + new link to C

Shortest route to A
= S + new link to A

- We **REMEMBER** previously found routes and add new information to discover next route

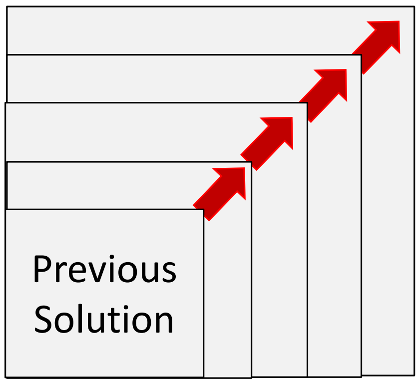
Explored all routes of length 1 2 3 4 5

P(n)이 기존에 찾은 해를 활용하나,
어떤 P(n-i)를 활용하는지는 그래프의 연결 상태에 따라 달라짐

Dijkstra's Algorithm ∈ Dynamic Programming

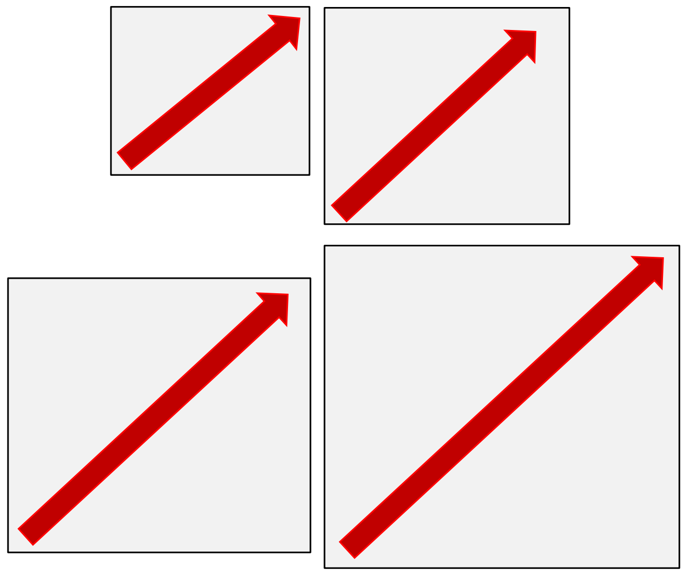
Dynamic Programming

- **REMEMBER and REUSE previous solutions to solve next problem more quickly**



NON-Dynamic Programming

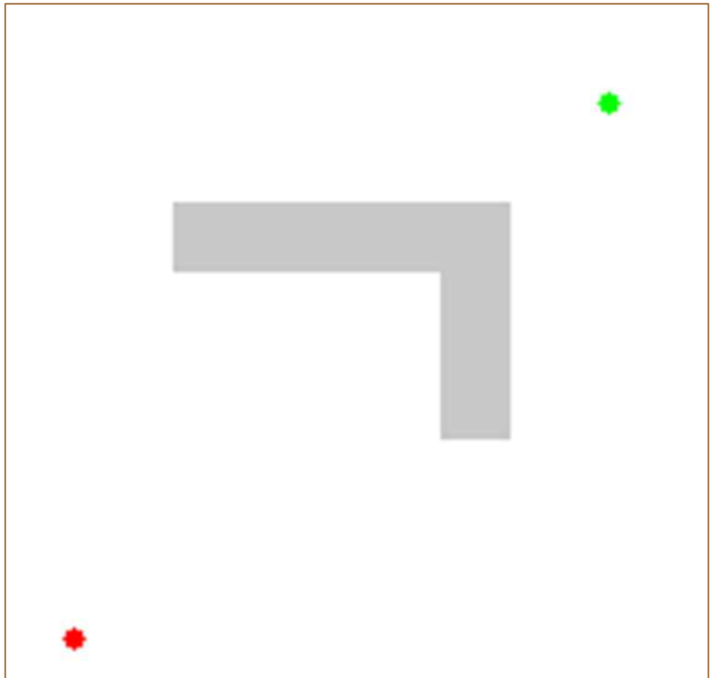
- **Solve each problem from scratch, without using previous solutions**



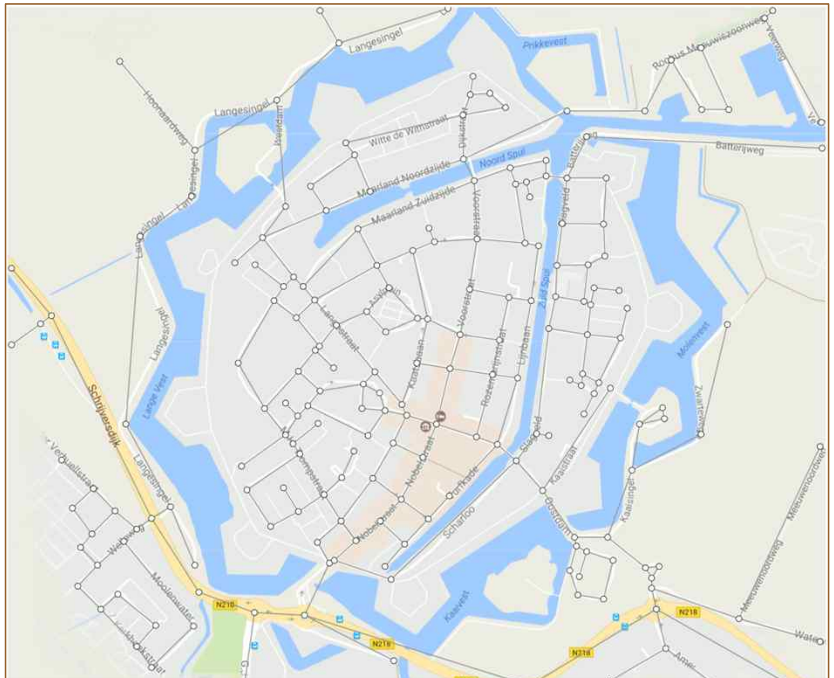
P(n)이 기존에 찾은 해 활용함으로써
각 해를 독립적으로 찾는 것 보다 더 빠르게 여러 해를 찾아냄

Summary: Dijkstra finds Shortest Routes to Multiple Destinations

- **Search All Directions at same speed & Reuse Previous Routes**



Progress of Dijkstra
on Robot's Vision System



Progress of Dijkstra
on Google Map

Dijkstra's algorithm에서는 **각 P(n)을 구하기 위해 무엇을 활용하는가?**

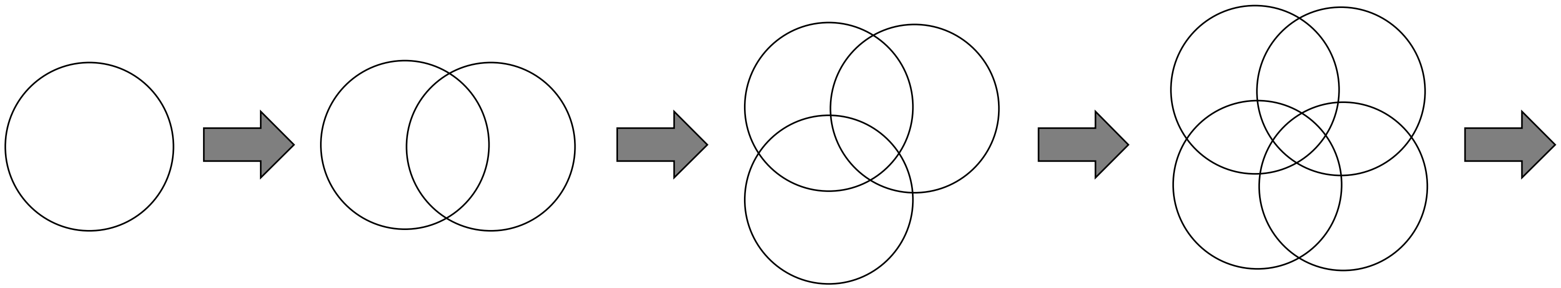
[문제] Dijkstra's algorithm을 사용해 아래와 같은 최소비용 경로를 **왼쪽부터 차례대로** 찾았다. **각 P(i)는 어떤 P(j)를 활용**하여 구한 것인가? **모든 n에 대해 P(n)은 P(n-i), i≥1**를 활용했다고 할 수 있는가? 출발지가 0라고 가정 하시오.

- { 0, 0A, 0AB, 0C, 0D, 0DE, 0AF, 0AFG, 0H, 0CI, 0AK }

[문제] Dijkstra's algorithm을 사용해 아래와 같은 최소비용 경로를 **왼쪽부터 차례대로** 찾았다. **각 P(i)는 어떤 P(j)를 활용**하여 구한 것인가? **모든 n에 대해 P(n)은 P(n-i), i≥1**를 활용했다고 할 수 있는가? 출발지가 0라고 가정 하시오.

- { 0, 0A, 0AB, 0C, 0AF, 0ABE, 0ABD, 0ABEDG }

[문제] N개의 원을 그리되 **각 원이 다른 모든 원과 겹치도록** 그린다.
총 몇 개의 교차점(교점)이 있는가?



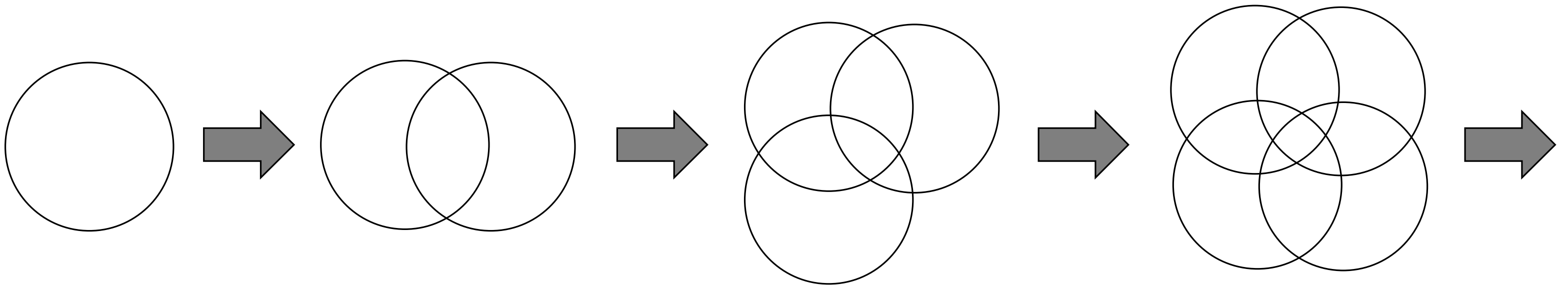
P(1) = 0 **P(2) = 2** **P(3) =** **P(4) =**

[01] N개의 원을 그릴 때 교점의 수를 P(N)이라 하자. 위 그림처럼 P(1)=0 이고 P(2)=2 이다. 그렇다면 P(3)은 무엇인가?

[02] P(4)는 무엇인가?

[01]과 [02]번에 답하면서 **P(N)과 P(N-1) 간의 관계**에 대해 생각해 보시오.

[문제] N개의 원을 그리되 **각 원이 다른 모든 원과 겹치도록** 그린다.
총 몇 개의 교차점(교점)이 있는가?



P(1) = 0　　P(2) = 2　　P(3) =　　P(4) =

[03] P(N)은 P(N-1)과 어떤 관계가 있는가?

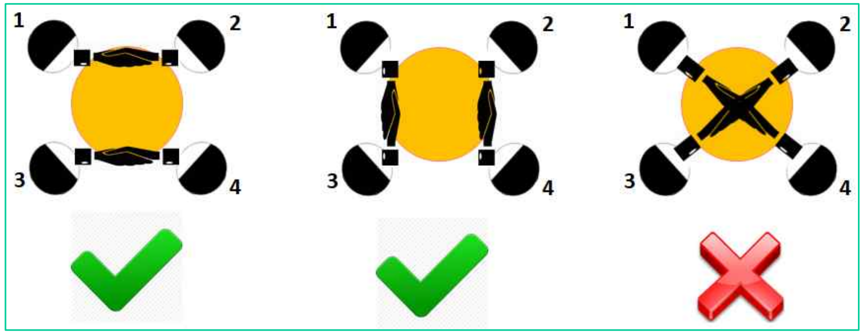
[문제] N개의 원을 그리되 **각 원이 다른 모든 원과 겹치도록** 그린다.
총 몇 개의 교차점(교점)이 있는가?

[04] [03]번 문제에서 발견한 P(N)과 P(N-1) 간의 관계를 활용하여 P(5)에서 P(10) 까지의 값을 구하시오.

[문제] N개의 원을 그리되 **각 원이 다른 모든 원과 겹치도록** 그린다.
총 몇 개의 교차점(교점)이 있는가?

[05] P(N)을 P(N-i)를 사용하지 않고 N만으로 표현할 수 있는가?

[문제] 2N 명의 사람들이 (N≥1) 둥근 탁자 주위에 앉아 있다. 두 사람씩 짝을 지어 악수를 하되 팔이 교차해서는 안 된다. 짝을 지을 수 있는 서로 다른 경우의 수는 몇 가지인가?



[01] 2명(N=1)인 경우 두 사람씩 짝지을 수 있는 모든 경우를 그려 보시오. P(1)은 무엇인가?

[02] 4명(N=2)인 경우 두 사람씩 짝지을 수 있는 모든 경우를 그려 보시오. P(2)는 무엇인가?

[문제] 2N 명의 사람들이 (N≥1) 둥근 탁자 주위에 앉아 있다. 두 사람씩 짝을 지어 악수를 하되 **팔이 교차해서는 안 된다**. 짝을 지을 수 있는 서로 다른 경우의 수는 몇 가지인가?

[03] 6명(N=3)인 경우 두 사람씩 짝지을 수 있는 모든 경우를 그려 보시오. P(3)은 무엇인가? **이전 문제의 해인 P(2)와 P(1)을 활용**해 더 간단하게 P(3)를 구해 보시오.

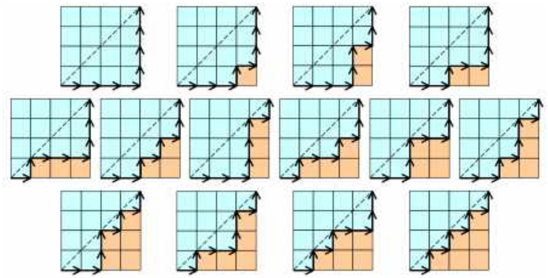
[04] 8명(N=4)인 경우 두 사람씩 짝지을 수 있는 모든 경우를 그려 보시오. P(4)는 무엇인가? 이전 문제의 해인 P(3), P(2)와 P(1)을 활용 하시오.

[05] 10명(N=5)인 경우 두 사람씩 짝지을 수 있는 모든 경우를 그려 보시오. P(5)는 무엇인가? 이전 문제의 해인 P(4), P(3), P(2), P(1)을 활용 하시오.

[06] 지금까지 보았던 P(N)과 P(N-i)의 관계를 일반화하여 임의의 N≥0에 대해 P(N+1)을 P(0)~P(N)의 함수로 나타내 보시오. P(0)=1이라 가정 하시오.

Catalan Numbers(카탈란 수)

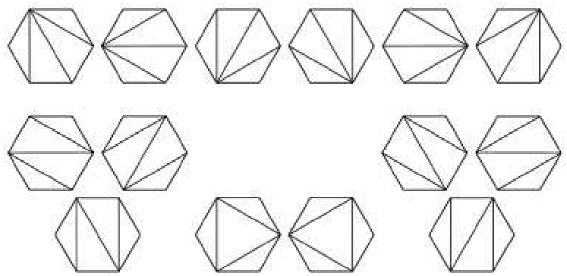
- Catalan numbers appear in in many real-world applications.
- # of ways to move from one corner to another corner in lattice



- # of ways to parenthesize a given set of operands and operators

((ab)c)d (a(bc))d (ab)(cd) a((bc)d) a(b(cd))

- # of ways to divide a polygon



- …
- Reference:
- https://en.wikipedia.org/wiki/Catalan_number

[문제] N cm^2 의 짐을 담을 수 있는 **배낭(knapsack)**이 있다. 아래 표와 같은 짐을 넣을 수 있다면 **총 가치의 합이 최대가 되는 짐의 조합**은 무엇인가? 각 짐은 원하는 개수만큼 넣을 수 있다.

짐의 종류	크기/부피 (cm^2)	가치 (원)
A	6	30
B	4	16
C	3	13
D	2	9

배낭 문제는 DP의 활용 예로 잘 알려진 문제이지만 '결국 DP를 사용하게 되었다'는 것 보다는 **'어떤 과정을 거쳐 DP를 사용하게 되었는가'**를 단계별로 보겠음. 이를 알아야 새로운 문제에 DP를 적용 가능하므로

N (cm^2)	0	1	2	3	4	5	6	7	…
가치	0	0	9	13	18	22	30		…
P(N)	-	-	D	C	D D	D C	A	D D C	…

[01] 위 표에는 N=0~7에 대해 가치의 합을 최대로 하는 짐의 조합 P(N)을 보여준다. N=0과 N=1인 경우에는 왜 해가 없을까?

[02] N=3일 때는 {C} 외에 다른 어떤 조합이 가능한가? 왜 {C}가 최적의 해인가?

[03] N=7일 때의 해인 P(7)={D,D,C}의 경우 가치의 합(테이블 빈 부분의 값)은 무엇인가?

GREEDY 알고리즘을 사용해도 최적의 해를 얻을 수 있나?

cm² 당 가치가 높은 것 부터 넣기

짐의 종류	크기 (cm^2)	가치 (원)	cm^2 당 가치 (원/cm^2)
A	6	30	30/6 = 5
B	4	16	16/4 = 4
C	3	13	13/3 = ~4.3
D	2	9	9/2 = 4.5

Greedy 알고리즘: 전체 가능한 경우를 (혹은 모든 가능한 정보를) 다 고려하지 않고, local한 정보만으로 (일부, 지엽적인 정보만으로) 해를 찾는 방식.
이렇게 얻은 해는 최적의 해일 수도 있고 아닐 수도 있음 (예: 최적에 근사한 해)

N (cm^2)	0	1	2	3	4	5	6	7	···
가치	0	0	9	13	18	22	30		···
P(N)	-	-	D	C	D D	D C	A	D D C	···

[04] 4개의 짐을 cm^2 당 가치가 높은 순으로(내림차순) 정렬 하시오.

GREEDY 알고리즘(cm² 당 가치가 높은 것 부터 넣기) 을 사용해도 최적의 해를 얻을 수 있나?

짐의 종류	크기 (cm²)	가치 (원)	cm² 당 가치 (원/cm²)
A	6	30	30/6 = 5
B	4	16	16/4 = 4
C	3	13	13/3 = ~4.3
D	2	9	9/2 = 4.5

Greedy 알고리즘: 전체 가능한 경우를 (혹은 모든 가능한 정보를) 다 고려하지 않고, local한 정보만으로 (일부, 지엽적인 정보만으로) 해를 찾는 방식.
이렇게 얻은 해는 최적의 해일 수도 있고 아닐 수도 있음 (예: 최적에 근사한 해)

N (cm²)	0	1	2	3	4	5	6	7	…
가치	0	0	9	13	18	22	30		…
P(N)	-	-	D	C	D D	D C	A	D D C	…

[05] 배낭의 남은 공간에 넣을 수 있는 크기의 짐 중 cm² 당 가치가 높은 짐 부터 먼저 넣는다고 하자 (예: N=8이라면 A부터 넣기). 이 방법을 사용한다면 N=2~7인 경우 중 어느 경우에 최적을 해를 얻지 못하는가?

가능성 트리 활용해 모든 가능한 경우 탐색 (완전 탐색, Exhaustive Search)

짐의 종류	크기 (cm^2)	가치 (원)
A	6	30
B	4	16
C	3	13
D	2	9

Greedy가 실패했으므로 **전 범위 탐색**하며 기존 해를 재사용 가능한지(즉 DP 적용 가능한지) 생각해 보자.

모든 가능성 탐색에는 **트리**가 자연스럽고 직관적인 표현 방식

[06] N=0~4인 경우에 대해 전 범위 탐색 위한 가능성 트리 그려보기

정점(vertex): 배낭에 남은 부피(cm^2)

변(edge): 각 단계에서 선정된 짐의 종류

root에서 leaf까지 변에 있는 짐들이 하나의 조합(가능한 해 하나)

가능성 트리 활용해 모든 가능한 경우 탐색 (완전 탐색, Exhaustive Search)

짐의 종류	크기 (cm^2)	가치 (원)
A	6	30
B	4	16
C	3	13
D	2	9

[07] N=4인 경우 가능성 트리를 보고 최적의 해 P(4)를 찾는 방법을 생각해 보시오.

DP: P(N)과 P(N-i)들 간 관계 보며 기존 해 재사용하는 방법 생각 보기

짐의 종류	크기 (cm^2)	가치 (원)
A	6	30
B	4	16
C	3	13
D	2	9

[08] N=6인 경우에 대해 전 범위 탐색 위한 가능성 트리를 그려 보자.

[09] N=6인 경우의 트리를 보며 P(6)을 구하기 위해 **P(0)~P(5)를 재사용 할 수 있는 방법**을 생각해 보자. (hint: 꼭 트리 전체를 다 그려야만 하나?)

DP: P(N)과 P(N-i)들 간 관계 보며 기존 해 재사용하는 방법 생각 보기

짐의 종류	크기 (cm^2)	가치 (원)
A	6	30
B	4	16
C	3	13
D	2	9

[10] 앞 문제에서 찾은 해법을 **임의의 N에 대해 일반화**하시오. 특히 임의이 N에 대해 P(N-i)들 ($i \geq 1$)을 사용하도록 트리를 그려 방법을 제시하시오.

DP: P(N)과 P(N-i)들 간 관계 보며 기존 해 재사용하는 방법 생각 보기

짐의 종류	크기 (cm^2)	가치 (원)
A	6	30
B	4	16
C	3	13
D	2	9

N (cm^2)	0	1	2	3	4	5
가치	0	0	9	13	18	22
P(N)	-	-	D	C	D D	D C

N (cm^2)	6	7	…
가치	30	31	…
P(N)	A	D D C	…

[11] [10]에서 찾은 해를 사용해 P(9)를 구하시오. 위 표의 P(2)~P(7)을 참조하시오.

[문제] 길이 N cm의 막대 사탕(candy bar)이 있다. 이를 길이 k cm(1≤k≤10 정수)인 조각으로 분리해 판매하려 한다. 분리 후 가격의 총 합이 최대가 되려면 어떻게 분리해야 하는가?

길이 (cm)	1	2	3	4	5	6	7	8	9	10
가격(원)	1	5	8	9	10	17	17	20	24	30

[01] 먼저 문제를 이해해 보자. 3 cm의 막대 사탕이 있다면 이를 1~10cm인 조각으로 분리할 수 있는 방법은 아래와 같이 4가지이다. **각 경우에 대해 가격의 총 합을 계산한 후 총합이 최대인 경우를 선택** 하시오.

[02] N=1과 N=2인 경우 각각에 대해 가격의 총합이 최대가 되도록 분리하는 방법을 찾아 보시오.

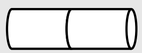
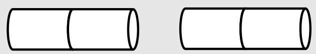
GREEDY 알고리즘을 사용해도 최적의 해를 얻을 수 있나?

cm 당 가치가 가장 높은 길이로 잘라 가기

길이 (cm)	1	2	3	4	5	6	7	8	9	10
가격(원)	1	5	8	9	10	17	17	20	24	30
가격/cm	1	2.5	~2.7	2.25	2	~2.8	~2.4	2.5	~2.7	3

[03] 1~10 cm 길이의 막대 사탕을 cm 당 가치(가치/cm)가 높은 순으로(내림차순) 정렬 하시오.

[04] 막대를 자를 때 cm 당 가치가 가장 높은 길이로 왼쪽에서부터 잘라간다고 가정하자 (예: N=6 이라면 자르지 않음). 이 방법을 사용한다면 N=1~4인 경우 중 어느 경우에 최적을 해를 얻지 못하는가? 최적의 해 P(N)을 보여주는 아래 표를 참조하시오.

N (cm)	1	2	3	4	…
가치 총 합	1	5	8	10	…
P(N)					…

가능성 트리 활용해 모든 가능한 경우 탐색 (완전 탐색, Exhaustive Search)

[05] N=4인 경우에 대해 전 범위 탐색 위한 가능성 트리 그려보기 - 막대의 왼쪽부터 1~10 cm중 하나의 길이로 잘라간다고 생각하고 그려 보시오.

정점(vertex): 남은 막대의 길이(cm)

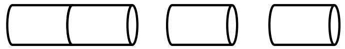
변(edge): 각 단계에서 잘라낸 막대 길이

root에서 leaf까지 변에 있는 길이 하나의 조합(가능한 해 하나)

Greedy가 실패했으므로 **전 범위 탐색**하며 기존 해를 재사용 가능한지(즉 DP 적용 가능한지) 생각해 보자.

가능한 경우 탐색에는 **트리**가 자연스럽고 직관적인 표현 방식

가능성 트리 활용해 모든 가능한 경우 탐색 (완전 탐색, Exhaustive Search)

[06] N=4인 경우의 트리를 보면 8개의 leaf가 있으며, root에서 각 leaf까지의 경로가 하나의 가능한 해를 나타낸다. 각 leaf가 나타내는 해가 무엇인지 그려 보시오 (예: )

[07] 8가지 경우 중 최적의 해(가치의 총 합이 가장 큰 해, P(8))는 무엇인가?

길이 (cm)	1	2	3	4	5	6	7	8	9	10
가격(원)	1	5	8	9	10	17	17	20	24	30

DP: P(N)과 P(N-i)들 간 관계 보며 기존 해 재사용하는 방법 생각 보기

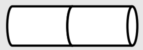
[08] 앞에서 그린 N=4인 경우의 트리를 보며 P(4)을 구하기 위해 P(0)~P(3)을 재사용 할 수 있는 방법을 생각해 보자. (hint: 꼭 트리 전체를 다 그려야만 하나?**)**

[09] 앞 문제에서 찾은 해법을 **임의의 N에 대해 일반화** 하시오. 특히 임의이 N에 대해 P(N-i)들 (i≥1)을 사용하도록 트리를 그려 방법을 제시하시오. 막대는 길이 1~10 cm의 조각으로 분리되어야 함에 유의 하시오.

DP: P(N)과 P(N-i)들 간 관계 보며 기존 해 재사용하는 방법 생각 보기

[10] 앞 문제에 찾은 해법을 사용해 **N=5에 대한 최적의 해 P(5)**를 구하시오. 아래 표의 P(0)~P(4)를 참조하시오.

길이 (cm)	1	2	3	4	5	6	7	8	9	10
가격(원)	1	5	8	9	10	17	17	20	24	30

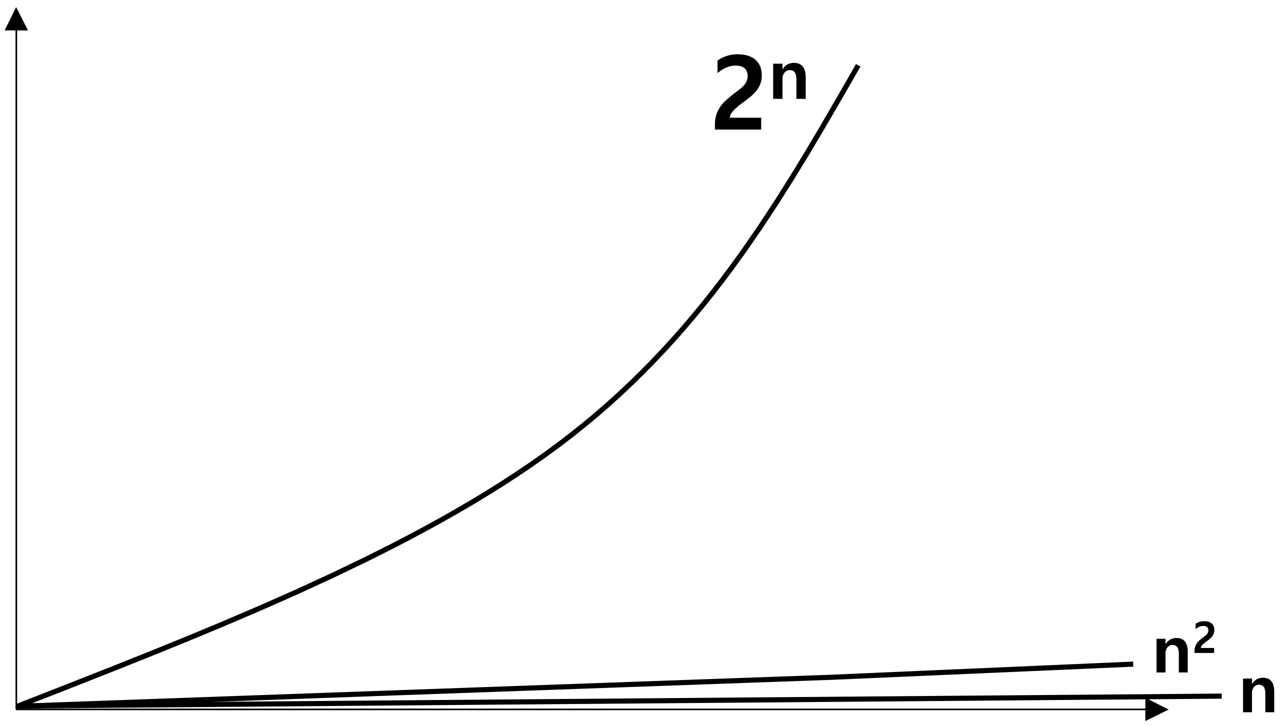
N (cm)	0	1	2	3	4
가치 총 합	0	1	5	8	10
P(N)	-				

가능성 트리 활용해 모든 가능한 경우 탐색 (완전 탐색, Exhaustive Search)

[11] N cm의 막대에 대해 앞에서와 같이 트리를 사용해 완전 탐색을 한다면 얼마나 많은 서로 다른 해를 탐색해 보아야 하는가? (Hint: N=1~4에 대해 각각 1, 2, 4, 8가지 해를 탐색해 보아야 함)

가능성 트리 활용해 모든 가능한 경우 탐색 (완전 탐색, Exhaustive Search)

[12] N이 증가할 때 2^N 값이 얼마나 빠르게 증가하는지 생각해 보자. **두께가 0.1125 mm인 종이**를 생각해 보자. 이를 한 번 반으로 접으면 두께가 2배가 된다 **(x2)**. **25번 반복해서 접으면 두께는** 얼마가 되는가? 실습실 바닥~천장 까지 가득 채울 수 있을까?



정리: Dynamic Programming (DP)

- DP에서는 큰 문제에 대한 해 P(n)를 구할 때 **작은 문제에 대한 해 P(n-i)들을 재활용**함으로써 문제를 보다 체계적으로, 빠르게 풀어냄. 따라서 이들 간의 연관성을 관찰하는 것이 중요함

- **연관성을 관찰**하기 위해서는 (여러 가능성 중 하나를 탐색하는 문제라면) N이 **작은 몇 가지 문제를 완전 탐색으로 풀어보며, 그 과정에서 겹치는 부분을 찾아본 후** 더 큰 문제에 대해서도 이러한 관계가 일반적으로 성립하는지 생각해 보자.

스마트 출결

수업 전

문서 형태
개요/이론 학습

→

수업 전

개요 관련 퀴즈 수행
스스로 학습 & 시행착오

40~50분

수업

퀴즈 풀이
개요 복습

→

수업

응용/심화 내용 학습
질의 응답

→

수업

실습 과제 (프로그래밍)
질의 응답

~3시간

이론 학습 (1.5~2시간) + 실습(1~1.5 시간) = 총 ~3시간

실습과제 수업 중 완료 못한 경우
당일 저녁 or 다음날까지
각자 수행해 제출

04. 실습문제풀이

- **이번 시간에 배운 내용에 대한 실습 문제 풀이 & 질의 응답**
- **채점 방식은 지난 시간 연습 문제 풀이때와 유사함**

- **왜 중요한가?**
- 이번 주 배운 내용을 총괄하는 문제 풀이 통해 배운 내용 활용 & 복습
- 문제 풀이 점수는 이번 주 **과제 점수에 포함**됨

knapsack() 함수 구현 조건: 수업에서 배운 배낭 문제에 대한 코드 작성

- 배낭 크기 **n**과 item 목록 **loads**가 주어졌을 때, 배낭에 들어가는 짐의 최대 가치 합을 출력

  def knapsack (**n**, **loads**):

- 입력 **n**: 0≤**n**≤1000 범위의 정수
  - 위 범위를 벗어나는 값은 입력으로 들어오지 않는다고 가정 (즉 오류 처리 하지 않아도 됨)

- 입력 **loads**: 3-tuple의 list로 각 tuple은 (짐 이름, 짐 하나 크기, 짐 하나 가치)를 나타냄
  - 예: [('A',2,10), ('B',3,7)]은 (1) 크기 2, 가치 10인 짐 'A'와 (2) 크기 3, 가치 7인 짐 'B'가 있음을 의미
  - tuple의 각 값 접근 방법: 배열처럼 index로 접근
  - 예를 들어, e = ('A',2,10) 이라면, e[0] = 'A', e[1] = 2, e[2] = 10

- 반환 값: 배낭에 들어가는 짐의 최대 가치 합 (정수)
  - 배낭에 들어가는 짐의 구성은 반환할 필요 없으며, 짐의 최대 가치 합을 나타내는 정수 하나만 반환

- 이번 시간에 제공한 코드 DynamicProgramming.py의 knapsack() 함수 내부에 코드 작성해 제출

입출력 예 (크기 n인 배낭에 들어가는 짐의 최대 가치 합 반환)

```
print(knapsack(4, [('A',5,10)]))
```

```
0
```

크기 4인 배낭에는 크기 5인 물건은 하나도 넣을 수 없음

```
print(knapsack(9, [('A',5,10)]))
```

```
10
```

크기 9인 배낭에는 크기 5인 물건 하나만 넣을 수 있음

```
print(knapsack(2, [('A',6,30),('B',4,16),('C',3,13),('D',2,9)]))
```

이번 시간에 보았던 짐 목록과 같은 경우

```
9
```

```
print(knapsack(4, [('A',6,30),('B',4,16),('C',3,13),('D',2,9)]))
```

```
18
```

```
print(knapsack(7, [('A',6,30),('B',4,16),('C',3,13),('D',2,9)]))
```

```
31
```

그 외 예제는 __main__ 아래 테스트 코드를 참조하세요.

handshake() 함수 구현 조건: 수업에서 배운 악수 문제에 대한 코드 작성

- 정수 **n**이 입력으로 주어졌을 때, 원탁에 둘러앉은 2**n** 명의 사람들이 팔을 교차하지 않고 악수할 수 있는 서로 다른 경우의 수 반환

  def handshake (**n**):

- 입력 **n**: 0≤**n**≤100 범위의 정수
  - 위 범위를 벗어나는 값은 입력으로 들어오지 않는다고 가정 (즉 오류 처리 하지 않아도 됨)

- 반환 값: 2**n** 명의 사람들이 팔을 교차하지 않고 악수할 수 있는 서로 다른 경우의 수

- 이번 시간에 제공한 코드 DynamicProgramming.py의 handshake() 함수 내부에 코드 작성해 제출

입출력 예 (크기 n인 배낭에 들어가는 짐의 최대 가치 합 반환)

```
print(handshake(0))
```
```
1
```
N=0일 때는 아무도 없으며
짝지을 수 있는 경우의 수가 (아무도 짝짓지 않는) 한가지라 봄

```
print(handshake(1))
```
```
1
```
N=1일 때는 1x2=2명만 있으므로
짝지을 수 있는 경우의 수는 1

```
print(handshake(2))
```
```
2
```
N=2일 때는 2x2=4명이 있으므로
짝지을 수 있는 경우의 수는 2

```
print(handshake(3))
```
```
5
```
N=3일 때는 3x2=6명이 있고
짝지을 수 있는 경우의 수는 5

그 외 예제는 __main__ 아래 테스트 코드를 참조하세요.

디버깅 시 더 많은 n에 대한 답이 궁금하다면 아래와 같은 링크 참조 (or 검색해 보세요)
https://www.mymathtables.com/numbers/first-hundred-catalan-number-table.html

그 외 프로그램 구현 조건

- 최종 결과물로 DynamicProgramming.py 파일 하나만 제출하며, 이 파일만으로 코드가 동작해야 함

- import는 사용할 수 없음

- __main__ 아래의 코드는 작성한 함수가 올바름을 확인하는 코드로
- 코드 작성 후 스스로 채점해 보는데 활용하세요.
- 각 테스트 케이스에 대해 P(or Pass) 혹은 F(or Fail)이 출력됩니다.

- 속도 테스트를 통과하려면 memoization을 사용하세요.
- memoization: 입력 n에 대한 답을 구했다면 저장해 두었다가 이후 다시 필요할 때 계산하지 않고 바로 저장한 값을 꺼내어 재사용

- 코드를 제출할 때는 __main__ 아래 코드는 제거하거나 수정하지 말고 제출합니다. (채점에 사용되기 때문)

이번 시간 제공 코드 함께 보기

- DynamicProgramming.py

실습 문제 풀이 & 질의 응답

- 실습 문제에 대한 코드는 lms 과제함에 **다음 날 23:59까지 제출** 가능합니다.
- 마감 시간까지 작성을 다 못한 경우는 (제출하지 않으면 0점이므로) 그때까지 작성한 코드를 꼭 제출해 부분점수를 받으세요.

- 실습 문제는 **개별 평가**입니다. 제출한 코드에 대한 유사도 검사를 실습 문제마다 진행하며, 코드를 건내 준 사례가 발견되면 0점 처리됩니다.
- 로직에 대해 서로 의견을 나누는 것은 괜찮지만, 코드를 직접 건내 주지는 마세요.
- 본인 실력 향상을 위해서도 **코드는 꼭 각자 직접 작성**해 주세요.