Step 8: Recursion and Iteration - Dynamic Programming

Instructor: Eunil Park (eunilpark@skku.edu)

Data eXperience Lab

SUNG KYUN KWAN UNIVERSITY

Contents

SUNG KYUN KWAN UNIVERSITY

Today’s Schedule

- 동적프로그래밍 (Dynamic Programming)?

1. 피보나치 수열
2. 파스칼 삼각형

0. Dynamic Programming?

Dynamic Programming

- 복잡한 문제를 간단한 여러 개의 문제로 나누어 푸는 방법을 말한다.
- 부분 문제 반복과 최적 부분 구조를 가지고 있는 알고리즘을 일반적인 방법에 비해 더욱 적은 시간 내에 풀 때 사용한다.

1. 먼저 입력 크기가 작은 부분 문제들을 모두 해결
2. 그 해들을 이용하여 보다 큰 크기의 부분 문제들을 해결
3. 최종적으로 원래 주어진 입력의 문제를 해결하는 알고리즘을 의미

0. Dynamic Programming?

Dynamic Programming

- 어떤 문제가 여러 단계의 반복되는 부분 문제로 이루어질 때, 각 단계의 부분 문제의 답을 기반으로 전체 문제의 답을 구하는 방법

- 동적 계획법으로 문제를 풀기 위해서는 그 문제가 "<u>최적 부분구조(**Optimal Substructure**)</u>"를 갖추고 있다는 전제 필요

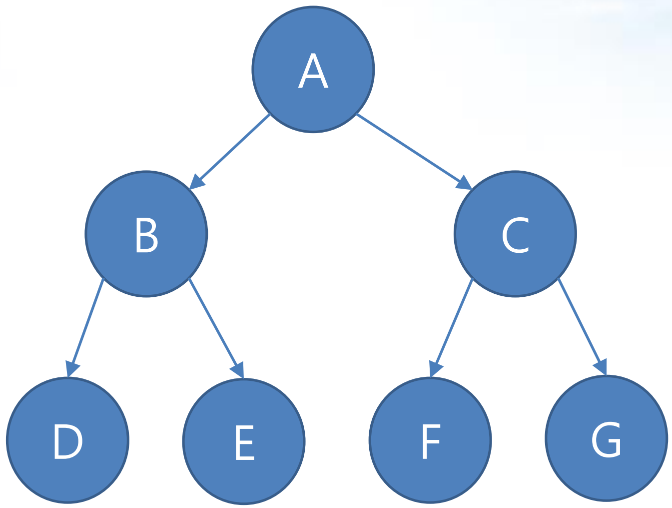
- 부분 구조?
  - <u>전체 문제의 최적해가 부분문제의 최적해로부터 만들어지는 구조</u>

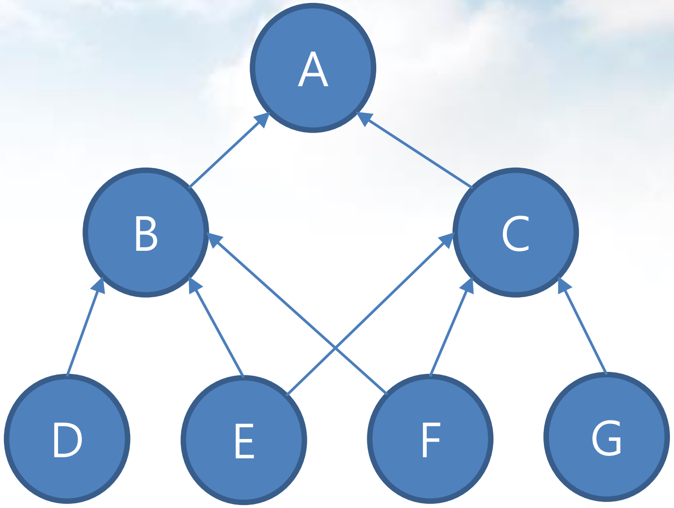
- 예: 5개의 작은 문제로 쪼갤 수 있는 어떤 문제가 있을 경우,
  - 쪼개진 문제의 해 5개를 모두 얻어야 이 문제의 해를 구할 수 있다 → 최적 부분 구조를 갖춤.

0. Dynamic Programming?

분할정복 알고리즘과의 차이

- 분할 정복 알고리즘과 동적 계획 알고리즘의 차이





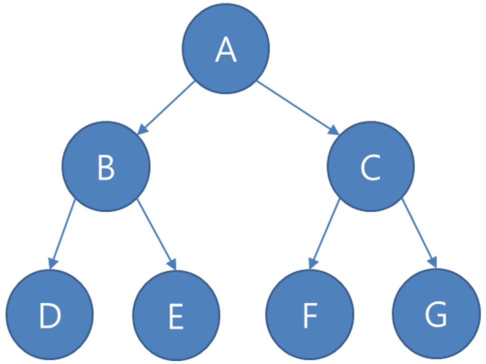
- 분할 정복 알고리즘의 부분문제들 사이의 관계: A는 B와 C로 분할되고, B는 D와 E로 분할되는데, D와 E의 해를 취합하여 B의 해를 구함.
  (단, D, E, F, G는 각각 더 이상 분할할 수 없는 (또는 가장 작은 크기의) 부분문제들임.
- 마찬가지로 F와 G의 해를 취합하여 C의 해를 구한 후, 마지막으로 B와 C의 해를 취합하여 A의 해를 구함.

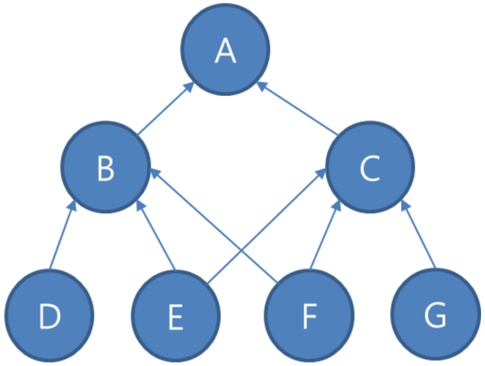
0. Dynamic Programming?

SUNG KYUN KWAN UNIVERSITY

분할정복 알고리즘과의 차이

- 동적 계획 알고리즘은 먼저 최소 단위의 부분 문제 D, E, F, G의 해를 각각 구한다.
- 그 다음에 D, E, F의 해를 이용하여 B의 해를 구한다.
- E, F, G의 해를 이용하여 C의 해를 구한다.
- B와 C의 해를 구하는데 E와 F의 해 모두를 이용한다.
- **분할 정복 알고리즘은 부분문제의 해를 중복 사용하지 않는다(이를 disjoint하다고 함).**



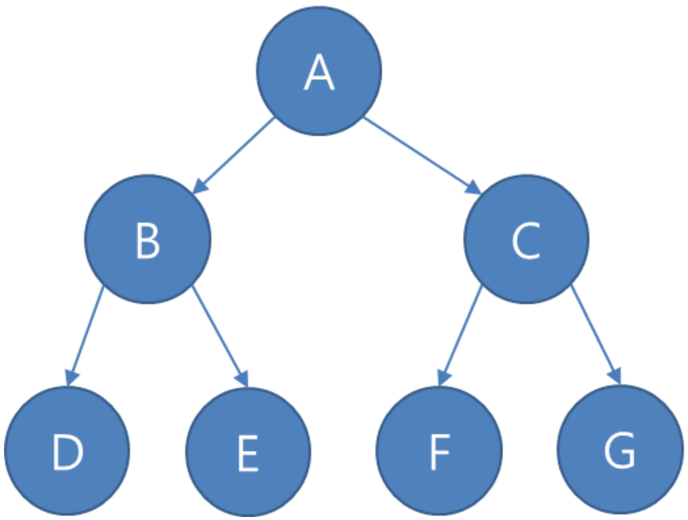


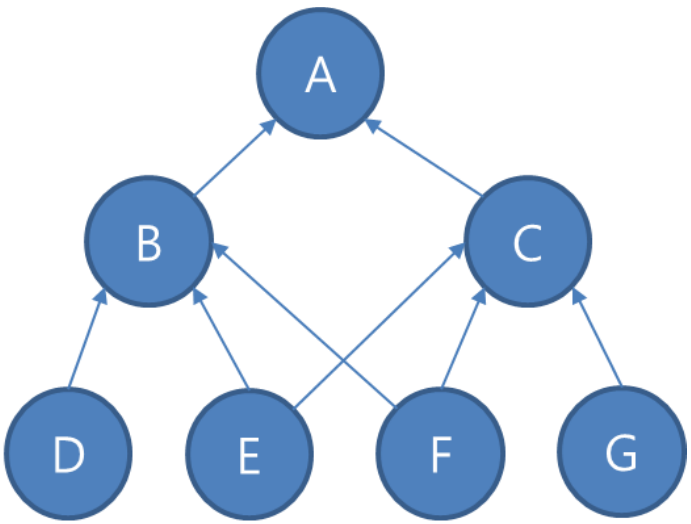
0. Dynamic Programming?

SUNG KYUN KWAN UNIVERSITY

동적계획법

- 동적 계획 알고리즘에는 부분문제들 사이에 의존적 관계(dependency)가 존재.
- 예를 들면, D, E, F의 해가 B를 해결하는데 사용되는 관계가 있음.
- 이러한 관계는 문제 또는 입력에 따라 다르고, 대부분의 경우 뚜렷이 보이지 않아서 **'함축적인 순서' (implicit order)**라고 한다.





01. 피보나치 수열

피보나치?

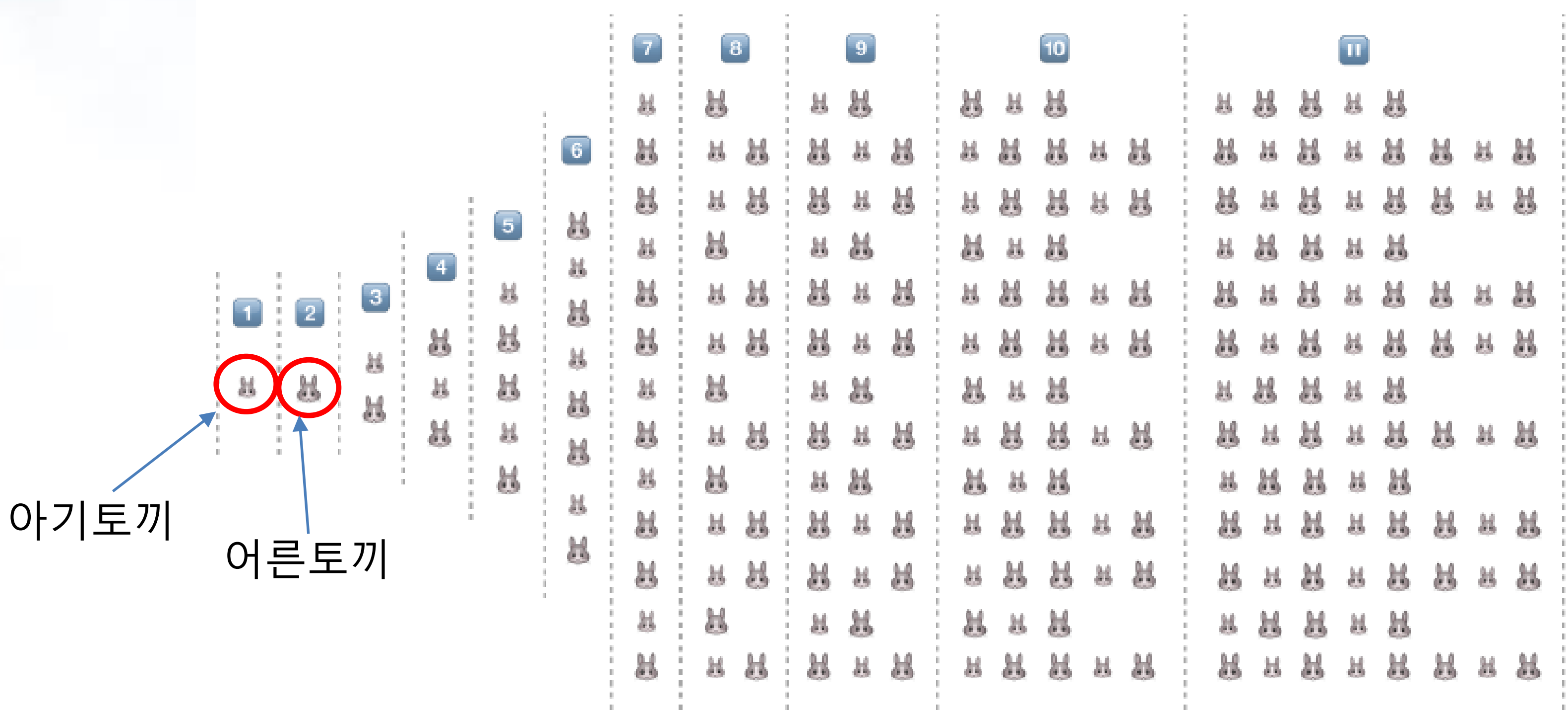
- 레오나르도 피보나치
  (이탈리아어: Leonardo Fibonacci, 1170년 ~ 1250년) 또는
  레오나르도피사노 (Leonardo da Pisa, Leonardo Pisano)
  - 이탈리아의 수학자로 **피보나치 수에 대한 연구**로 유명.
  - 유럽에 아라비아 수 체계를 소개.

01. 피보나치 수열

피보나치수

- 피보나치(Leonardo Fibonacci, 1170~1250, 이탈리아 수학자)가 1202년에 제시
- 토끼 번식 규칙 예
  - 어른 토끼 1쌍은 매달 아기 토끼 1쌍을 낳음
  - 아기 토끼는 태어난지 1달이 지나면 어른 토끼가 되어 번식 가능해짐
  - 토끼는 죽지 않음

01. 피보나치 수열



토끼 가족의 개체수 역사(단위: 쌍)

월	0	1	2	3	4	5	6	7	8	9	10	11	12
아기 토끼	0	1	0	1	1	2	3	5	8	13	21	34	55
어른 토끼	0	0	1	1	2	3	5	8	13	21	34	55	89
총	0	1	1	2	3	5	8	13	21	34	55	89	144

01. 피보나치 수열

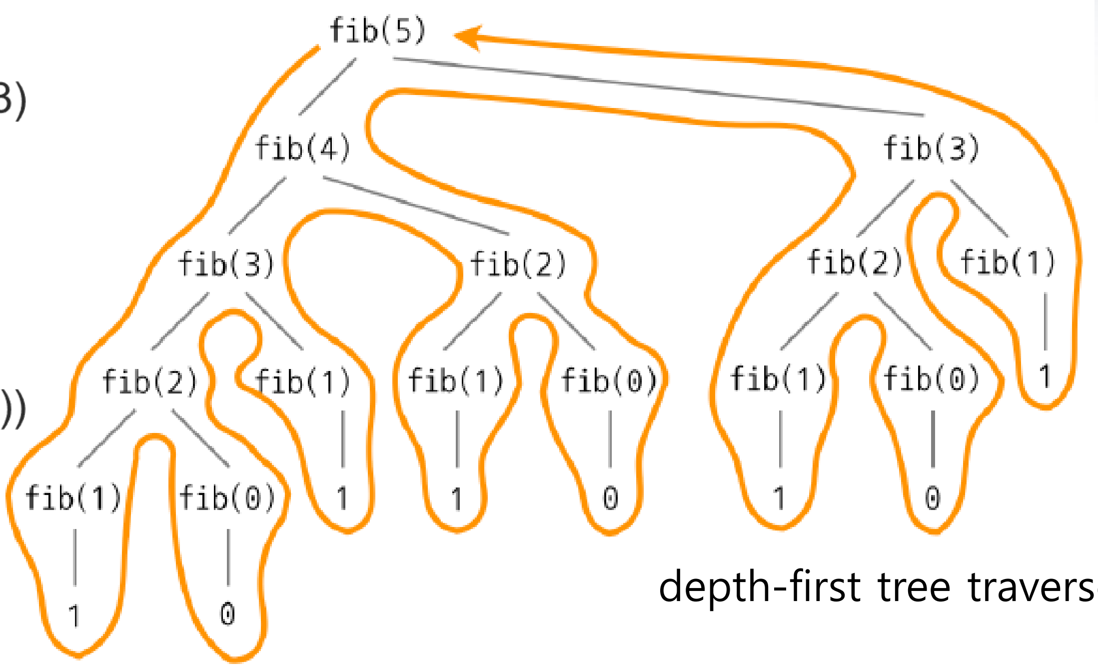
피보나치수의 재귀 해법

- 토끼 개체수(쌍)의 증가
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, …
- 피보나치수열(Fibonacci sequence)
  - 이전 두개의 수를 더해서 다음 수를 정하는 수열
- n째 피보나치수의 귀납구조
  - fib(n) = fib(n-1) + fib(n-2), n>1
  - fib(1) = 1
  - fib(0) = 0

```
def fib(n):
  if n > 1:
    return fib(n - 1) + fib(n - 2)
  else:
    return n
```

SUNG KYUN KWAN UNIVERSITY

- fib(5)
  → fib(4) + fib(3)
  → (fib(3) + fib(2)) + fib(3)
  → ((fib(2) +fib(1)) + fib(2)) + fib(3)
  → (((fib(1) + fib(0)) + fib(1)) + fib(2)) + fib(3)
  → (((1+ fib(0)) + fib(1)) + fib(2)) + fib(3)
  → (((1 + 0) + fib(1)) + fib(2)) + fib(3)
  → ((1 + fib(1)) + fib(2)) + fib(3)
  → ((1 + 1) + fib(2)) + fib(3)
  → (2 + fib(2)) + fib(3)
  → (2 + (fib(1) + fib(0))) + fib(3)
  → (2 + (1 + fib(0))) + fib(3)
  → (2 + (1 + 0)) + fib(3)
  → (2 + 1) + fib(3)
  → 3 + fib(3)
  → 3 + (fib(2) + fib(1))
  → 3 + ((fib(1) + fib(0)) + fib(1))
  → 3 + ((1 + fib(0)) + fib(1))
  → 3 + ((1 + 0) + fib(1))
  → 3 + (1 + fib(1))
  → 3 + (1 + 1)
  → 3 + 2
  → 5



depth-first tree traverse

01. 피보나치 수열

피보나치수의 재귀 해법

- fib(12)
  - 144
- fib(24)
  - 46368
- fib(30)
  - 832040
- fib(36)
  - 14930352
- fib(42)
  - 267914296
- fib(48)
  - 4807526976

계산 시간이 급격히 길어짐!

SUNG KYUN KWAN UNIVERSITY

계산복잡도

- fib(n) 호출 -> 재귀호출 2번
    - 각 재귀호출이 재귀호출을 2번씩
        - 각 재귀호출이 재귀호출을 2번씩
            - 각 재귀호출이 재귀호출을 2번씩
                - 각 재귀호출이 재귀호출을 2번씩
                    - …

2
2^2
2^3
2^4
…

- 48번째 재귀호출의 경우 총 호출 횟수가 2^{48}, 280조 이상 넘음
- 즉, n이 증가함에 따라 호출 회수가 2^n 비례하여 기하급수적으로 증가함

- 동일한 재귀호출을 중복 호출 -> 중복계산의 시간 낭비가 큼
- 하향식(top-down) 방법이 아닌 상향식(bottom-up) 해법이 필요함
    - <u>**동적 계획법(dynamic programming)**</u> 풀이 방식

01. 피보나치 수열

동적계획법

- 전략
    - fib(2)계산을 먼저하고, 다음에 fib(3), 그 다음에 fib(4), … 이런 식으로 계산해 감
- 토끼 번식 규칙
    - 어른 토끼 1쌍은 아기 토끼를 한달에 한 쌍 낳으니까, 이달의 아기 토끼 쌍의 수는 지난달의 어른 토끼 쌍의 수
    - 아기 토끼는 한달이 지나면 어른 토끼가 되므로, 이달의 어른 토끼 쌍의 수는 지난달의 어른 토끼 쌍의 수 + 지난달의 아기 토끼 쌍의 수

i	0	1	2	3	4	5	6	7	8	9	10	11	12
아기 토끼 bunny	0	1	0	1	1	2	3	5	8	13	21	34	55
어른 토끼 rabby	0	0	1	1	2	3	5	8	13	21	34	55	89
fib(i)	0	1	1	2	3	5	8	13	21	34	55	89	144

01. 피보나치 수열

동적계획법

i	0	1	2	3	4	5	6	7	8	9	10	11	12
아기 토끼 bunny	0	1	0	1	1	2	3	5	8	13	21	34	55
어른 토끼 rabby	0	0	1	1	2	3	5	8	13	21	34	55	89
fib(i)	0	1	1	2	3	5	8	13	21	34	55	89	144

- 식으로 표현 (이달 i, 지난달 i-1)
  - 전체 토끼 쌍의 수: fib_i ,
    아기 토끼 쌍의 수: $bunny_i$,어른 토끼 쌍의 수: $rabby_i$
  - $bunny_i = rabby_{i-1}$
  - $rabbiy_i = rabby_{i-1} + bunny_{i-1}$
  - $fib_i = bunny_i + rabby_i$

$bunny_9 = rabby_8$

$rabby_9 = rabby_8 + bunny_8$

- 구현

i	bunny	rabby
1	1	0
2	0	1
3	1	1
4	1	2
5	2	3
6	3	5
7	5	8
…	…	…

```
def fibo2(n):
    i = 1
    bunny, rabby = 1, 0
    while i < n:
        i = i + 1
        bunny, rabby = rabby, bunny + rabby
    return bunny + rabby
```

계산 횟수: n-1

계산 시간은 입력값 n에 비례

동적계획법

```python
def fibo2(n):
    i = 1
    bunny, rabby = 1, 0
    while i < n:
        i = i + 1
        bunny, rabby = rabby, bunny + rabby
    return bunny + rabby
```

Python에서는 동시지정 허용

X

```python
def fibo2(n):
    i = 1
    bunny, rabby = 1, 0
    while i < n:
        i = i + 1
        bunny = rabby
        rabby = bunny + rabby
    return bunny + rabby
```

OK

```python
def fibo2(n):
    i = 1
    bunny, rabby = 1, 0
    while i < n:
        i = i + 1
        temp = bunny
        bunny = rabby
        rabby = temp + rabby
    return bunny + rabby
```

임시 변수 사용
skill 필수!

SUNG KYUN KWAN UNIVERSITY

동적계획법

```
def fibo2(n):
    i = 1
    bunny, rabby = 1, 0
    while i < n:
        i = i + 1
        bunny, rabby = rabby, bunny + rabby
    return bunny + rabby
```

For 반복문 버전

```
def fibo2(n):
    bunny, rabby = 1, 0
    for i in range(2, n + 1):
        bunny, rabby = rabby, bunny + rabby
    return bunny + rabby
```

i를 쓰지 않기 때문에,
와일드카드(밑줄)로 쓰는 것이 좋음

```
def fibo2(n):
    bunny, rabby = 1, 0
    for _ in range(2, n + 1):
        bunny, rabby = rabby, bunny + rabby
    return bunny + rabby
```

01. 피보나치 수열

피보나치 수열 전체

```
def fibseq(n):
  fibs = [0, 1]
  for k in range(2, n+1):
    fibs.append(fibs[k - 1] + fibs[k - 2])
  return fibs
```

print(fibseq(10)) ➔ [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

```
def fib(n):
  return fibseq(n)[-1]
```

print(fib(10)) ➔ 55

조합계산

- 조합(combination): n개에서 순서에 상관없이 r개를 뽑는 가지수
- 조합의 재귀 정의

$$ {}_{n}C_{r} = {}_{n-1}C_{r-1} + {}_{n-1}C_{r}, \ r \neq 0 \ \text{and} \ r \neq n \qquad {}_{n}C_{0} = 1 \qquad {}_{n}C_{n} = 1 $$

- Python 코드

```
def comb(n, r):
    if not (r == 0 or r == n):
        return comb(n - 1, r - 1) + comb(n - 1, r)
    else:
        return 1
```

- 계산 복잡도
  - 재귀 호출 횟수가 지수에 비례하여 증가
  - 동일한 재귀호출을 엄청나게 많이 중복호출
- 프랑스 수학자 블레즈 파스칼(Blaise Pascal, 1623 ~ 1662)이 고안한 삼각형을 이용하면 이 문제의 해답을 빨리 (효율적으로) 구할 수 있음

SUNG KYUN KWAN UNIVERSITY

파스칼 삼각형

- 파스칼 삼각형
  - 삼각형의 빗면은 모두 1
  - 내부의 수는 바로 위의 두 수를 더함
- 파스칼 삼각형 = 조합의 계산 결과

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1

$_0C_0$
$_1C_0$ $_1C_1$
$_2C_0$ $_2C_1$ $_2C_2$
$_3C_0$ $_3C_1$ $_3C_2$ $_3C_3$
$_4C_0$ $_4C_1$ $_4C_2$ $_4C_3$ $_4C_4$
$_5C_0$ $_5C_1$ $_5C_2$ $_5C_3$ $_5C_4$ $_5C_5$
$_6C_0$ $_6C_1$ $_6C_2$ $_6C_3$ $_6C_4$ $_6C_5$ $_6C_6$

02. 파스칼 삼각형

상향식 동적계획

- $_6C_4$ 계산: 위에서 아래로 계산하면서 평형사변형만 채우면 구할 수 있음

1

1 1

1 2 1

3 3 1

6 4 1

10 5

15

$_0C_0$

$_1C_0$ $_1C_1$

$_2C_0$ $_2C_1$ $_2C_2$

$_3C_1$ $_3C_2$ $_3C_3$

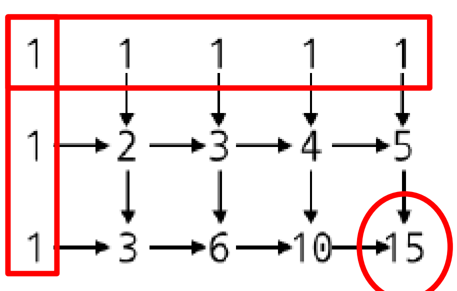
$_4C_2$ $_4C_3$ $_4C_4$

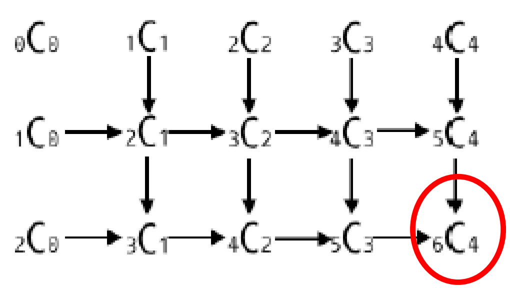
$_5C_3$ $_5C_4$

$_6C_4$

- 이 평행사변형은 왼쪽 아래로 비스듬히 눌러 직사각형으로 눕혀보면, 화살표와 같은 방향인 <u>왼쪽에서 오른쪽으로 위에서 아래로</u> 답을 구하는 것 과 같음

총 8회 계산





상향식 동적계획

- ${}_nC_r$ 계산: (n-r) * r회의 덧셈 수행

"1"로 셋팅

${}_0C_0$	${}_1C_1$	${}_2C_2$	${}_3C_3$	…	${}_{r-2}C_{r-2}$	${}_{r-1}C_{r-1}$	${}_rC_r$
${}_1C_0$	${}_2C_1$	${}_3C_2$	${}_4C_3$	…	${}_{r-1}C_{r-2}$	${}_rC_{r-1}$	${}_{r+1}C_r$
${}_2C_0$	${}_3C_1$	${}_4C_2$	${}_5C_3$	…	${}_rC_{r-2}$	${}_{r+1}C_{r-1}$	${}_{r+2}C_r$
${}_3C_0$	${}_4C_1$	${}_5C_2$	${}_6C_3$	…	${}_{r+1}C_{r-2}$	${}_{r+2}C_{r-1}$	${}_{r+3}C_r$
…	…	…	…	…	…	…	…
${}_{n-r-2}C_0$	${}_{n-r-1}C_1$	${}_{n-r}C_2$	${}_{n-r+1}C_3$	…	${}_{n-4}C_{r-2}$	${}_{n-3}C_{r-1}$	${}_{n-2}C_r$
${}_{n-r-1}C_0$	${}_{n-r}C_1$	${}_{n-r+1}C_2$	${}_{n-r+2}C_3$	…	${}_{n-3}C_{r-2}$	${}_{n-2}C_{r-1}$	${}_{n-1}C_r$
${}_{n-r}C_0$	${}_{n-r+1}C_1$	${}_{n-r+2}C_2$	${}_{n-r+3}C_3$	…	${}_{n-2}C_{r-2}$	${}_{n-1}C_{r-1}$	${}_nC_r$

n-r+1

r+1

02. 파스칼 삼각형

행렬의 표현

- 중첩 리스트로 표현
  - [[1, 1, 1, 1, 1], [1, 2, 3, 4, 5], [1,3, 6, 10, 15]]
  - 행렬의 행과 열 번호가 중첩 리스트의 위치 번호와 일치하기 때문에 편리함

1	1	1	1	1
1	2	3	4	5
1	3	6	10	15

```
>>> matrix = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
>>> matrix[1][2]
7
>>> matrix[1][2] = -7
>>> matrix
[[1, 2, 3, 4], [5, 6, -7, 8], [9, 10, 11, 12]]
>>> matrix[1] = [55, 66, 77]
>>> matrix
[[1, 2, 3, 4], [55, 66, 77], [9, 10, 11, 12]]
>>> matrix[0] = 1234
>>> matrix
[1234, [55, 66, 77], [9, 10, 11, 12]]
>>>
```

SUNG KYUN KWAN UNIVERSITY

조합 계산 함수 구현

```
def comb_pascal(n, r):
    matrix = [[ ]] * (n - r + 1)
    matrix[0] = [1] * (r + 1)
    for i in range(1, n - r + 1):
        matrix[i] = [1]
    for i in range(1, n - r + 1):
        for j in range(1, r + 1):
            newvalue = matrix[i][j - 1] + matrix[i - 1][j]
            matrix[i].append(newvalue)
    return matrix[n - r][r]
```

빈 리스트가 n-r+1개 들어 있는 [[], [], ..., []] 초기화

[[1, 1, 1, ..., 1], [], ..., []]

[[1, 1, 1, ..., 1], [1], ..., [1]]

안쪽 for:
행렬의 하나 채움
바깥 for:
행렬 전체 계산

행렬의 맨 아래 오른쪽 끝 원소 return

Today's Lessons!

Summary

동적프로그래밍

1. 피보나치 수열
2. 파스칼 삼각형

Thanks

Step 8: Recursion and Iteration – Dynamic Programming
Instructor: Eunil Park (eunilpark@skku.edu)