알고리즘
Dynamic Programming

이영석

동적프로그래밍

- Dynamic programming (DP)
  - 문제해결 패러다임
  - 문제를 해결하기 위해 더 작은 문제를 해결하고 해를 재활용하는 방식
  - “기억하며 풀기“
- 분할 정복 기법과 유사

Fibonacci with DP

- memoization
  - f(n) = f(n-1) + f(n-2)
    - if f(n) > 0 (n != 0, 1)
      - 계산값 저장

```python
def fibo(n):
    if n < 2:
        return n
    cache = [0 for _ in range(n+1)]
    cache[1] = 1
    for i in range(2, n+1):
        cache[i] = cache[i-1] + cache[i-2]
    return cache[n]

for n in range(0, 51):
    print(n, fibo(n))
```

배낭(Knapsack) 채우기 문제

- 문제
  - 값(value)를 최대로!
- 제한조건
  - 배낭의 무게는 W보다 작아야함

$$\max \sum_{i \in T} v_i \ \text{subjectto} \sum_{i \in T} w_i \leq W$$

Item #	Weight	Value
1	1	8
2	3	6
3	5	5

Recursive Formula

$$V[k,w] = \begin{cases} V[k-1,w] & \text{if } w_k > w \\ \max\{V[k-1,w], V[k-1,w-w_k]+b_k\} & \text{else} \end{cases}$$

- *V[k, w]*: 1, … , k개 아이템, weight w의 의한 가치 V
  - 방법: k-1개 있는 집합에 k번째 item 포함할 것인지? 아닌지?
- First case: $w_k > w$
  - item *k* 넣을 수 없음
- Second case: $w_k \leq w$
  - item *k* <u>포함할 수 있음, 대신 w_k 무게만큼 뺀 상태에서</u>

Knapsack Algorithm

```
for w = 0 to W
    V[0,w] = 0
for i = 1 to n
    V[i,0] = 0
for i = 1 to n
    for w = 0 to W
            if w_i <= w // item i can be part of the solution
                    if b_i + V[i-1,w-w_i] > V[i-1,w]
                            V[i,w] = b_i + V[i-1,w- w_i]
                    else
                            V[i,w] = V[i-1,w]
            else V[i,w] = V[i-1,w]  // w_i > w
```

Example

무게 최대 5 가방
n = 4 (# of elements)
W = 5 (max weight)

4개 아이템 (weight, benefit):
(2,3), (3,4), (4,5), (5,6)

4개 아이템 (weight, benefit):
(2,3), (3,4), (4,5), (5,6)

Example (2): V(k, w) 행렬 채우기

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						

for w = 0 to W
    V[0,w] = 0

4개 아이템 (weight, benefit):
(2,3), (3,4), (4,5), (5,6)

Example (3)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

for i = 1 to n

$V[i,0] = 0$

Example (4)

무게 제한 1인 경우
1번 물건 무게 2이기때문에 x
0번 물건 넣었을 경우 그대로

Items:

$\boxed{1: (2,3)}$
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0				
2	0					
3	0					
4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=1$

$w-w_i=-1$

if w_i <= w // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w- w_i]$
    else
        $V[i,w] = V[i-1,w]$
else **V[i,w] = V[i-1,w]** // $w_i > w$

Example (5)

무게 제한 2인 경우 1번 물건 무게 2이기때문에 0
1번 넣으면(1번 무게 빼고 0번까지 가치)과
0번만 있을 경우 가치 비교

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	**3**			
2	0					
3	0					
4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=2$

$w-w_i=0$

if $w_i <= w$ // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w- w_i]$
    else
        $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (6)

무게 제한 3인 경우 1번 물건 무게 2이기때문에 0
1번 넣으면(1번 무게 빼고 0번까지 가치)과
0번만 있을 경우 가치 비교

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	**3**		
2	0					
3	0					
4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=3$

$w-w_i=1$

if $w_i <= w$ // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w- w_i]$
    else
        $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (7)

무게 제한 4인 경우 1번 물건 무게 2이기때문에 0
1번 넣으면(1번 무게 빼고 0번까지 가치)과
0번만 있을 경우 가치 비교

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	**3**	
2	0					
3	0					
4	0					

i=1

b_i=3

w_i=2

w=4

w-w_i =2

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // w_i > w

Example (8)

무게 제한 5인 경우 1번 물건 무게 2이기때문에 0
1번 넣으면(1번 무게 빼고 0번까지 가치)과
0번만 있을 경우 가치 비교

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	**3**
2	0					
3	0					
4	0					

$i=1$
$b_i=3$
$w_i=2$
$w=5$
$w-w_i=3$

if $w_i <= w$ // item i can be part of the solution
  if $b_i + V[i-1,w-w_i] > V[i-1,w]$
    $V[i,w] = b_i + V[i-1,w-w_i]$
  else
    $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

무게 제한 1인 경우 2번 물건 무게 3이기때문에 x

Example (9)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	**0**				
3	0					
4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=1$

$w-w_i=-2$

if $w_i <= w$ // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w-w_i]$
    else
        $V[i,w] = V[i-1,w]$
else **V[i,w] = V[i-1,w]** // $w_i > w$

Example (10)

무게 제한 2인 경우 2번 물건 무게 3이기때문에 x

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	**3**			
3	0					
4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=2$

$w-w_i=-1$

if w_i <= w // item i can be part of the solution
  if $b_i + V[i-1,w-w_i] > V[i-1,w]$
    $V[i,w] = b_i + V[i-1,w- w_i]$
  else
    $V[i,w] = V[i-1,w]$
else **V[i,w] = V[i-1,w]** // $w_i > w$

Example (11)

무게 제한 3인 경우 2번 물건 무게 3이기때문에 0
2번 넣고, 2번 무게(3) 뺀 1번까지의 가치와
1번까지의 가치 비교

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	**4**		
3	0					
4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=3$

$w-w_i=0$

if $w_i <= w$ // item i can be part of the solution
  if $b_i + V[i-1,w-w_i] > V[i-1,w]$
    $V[i,w] = b_i + V[i-1,w- w_i]$
  else
    $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (12)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	**4**	
3	0					
4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=4$

$w-w_i=1$

```
if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w]  // w_i > w
```

Example (13)

무게 제한 5인 경우 2번 물건 무게 3이기때문에
o
2번 넣고, 2번 무게(3) 뺀 1번까지의 가치와
1번까지의 가치 비교

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0					
4	0					

$i=2$
$b_i=4$
$w_i=3$
$w=5$
$w-w_i=2$

if w_i <= w // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w- w_i]$
    else
        $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (14)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	**0**	**3**	**4**		
4	0					

$i=3$

$b_i=5$

$w_i=4$

$w= 1..3$

if w_i <= w // item i can be part of the solution
  if b_i + V[i-1,w-w_i] > V[i-1,w]
    V[i,w] = b_i + V[i-1,w- w_i]
  else
    V[i,w] = V[i-1,w]
else **V[i,w] = V[i-1,w]** // w_i > w

Example (15)

Items:

1: (2,3)
2: (3,4)
3: (4,5)

4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	
4	0					

$i=3$

$b_i=5$

$w_i=4$

$w= 4$

$w- w_i=0$

```
if wi <= w // item i can be part of the solution
    if bi + V[i-1,w-wi] > V[i-1,w]
        V[i,w] = bi + V[i-1,w- wi]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w]  // wi > w
```

Example (16)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0					

$i=3$

$b_i=5$

$w_i=4$

$w= 5$

$w- w_i=1$

if w_i <= w // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w- w_i]$
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // $w_i > w$

Example (17)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	**0**	**3**	**4**	**5**	

$i=4$

$b_i=6$

$w_i=5$

$w= 1..4$

if w_i <= w // item i can be part of the solution

  if $b_i + V[i-1,w-w_i] > V[i-1,w]$

    $V[i,w] = b_i + V[i-1,w- w_i]$

  else

    $V[i,w] = V[i-1,w]$

else **V[i,w] = V[i-1,w]** // $w_i > w$

Example (18)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	**7**

$i=4$

$b_i=6$

$w_i=5$

$w= 5$

$w- w_i=0$

if w_i <= w // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w- w_i]$
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // $w_i > w$

Finding the Items

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=4
k= 5
b_i=6
w_i=5
$V[i,k] = 7$
$V[i-1,k] = 7$

i=n, k=W
while i,k > 0
    if $V[i,k] \neq V[i-1,k]$ then
        mark the i^{th} item as in the knapsack
        $i = i-1$, $k = k-w_i$
    else
        $i = i-1$

Finding the Items (2)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=4
k= 5
b_i=6
w_i=5
$V[i,k] = 7$
$V[i-1,k] = 7$

i=n, k=W
while i,k > 0
    if $V[i,k] \neq V[i-1,k]$ then
        mark the i^{th} item as in the knapsack
        $i = i-1,\ k = k-w_i$
    else
        $\boldsymbol{i = i-1}$

Finding the Items (3)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=3
k= 5
b_i=5
w_i=4
$V[i,k] = 7$
$V[i-1,k] = 7$

i=n, k=W
while i,k > 0
  if $V[i,k] \neq V[i-1,k]$ then
    mark the i^{th} item as in the knapsack
    $i = i-1$, $k = k-w_i$
  else
    $\boldsymbol{i = i-1}$

Finding the Items (4)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=2
k= 5
b_i=4
w_i=3
$V[i,k] = 7$
$V[i-1,k] = 3$
$k - w_i = 2$

i=n, k=W
while i,k > 0
  if $V[i,k] \neq V[i-1,k]$ then
    mark the i^{th} item as in the knapsack
    $i = i-1$, $k = k-w_i$
  else
    ***i* = *i*−1**

Finding the Items (5)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=1
k= 2
b_i=3
w_i=2
$V[i,k] = 3$
$V[i-1,k] = 0$
$k - w_i$=0

i=n, k=W
while i,k > 0
if $V[i,k] \neq V[i-1,k]$ then
mark the i^{th} item as in the knapsack
$i = i-1$, $k = k-w_i$
else
$\boldsymbol{i = i-1}$

Finding the Items (6)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=0
k= 0

The optimal knapsack should contain {1, 2}

i=n, k=W
while i,k > 0
    if $V[i,k] \neq V[i-1,k]$ then
        mark the n^{th} item as in the knapsack
        $i = i-1$, $k = k-w_i$
    else
        $i = i-1$

Finding the Items (7)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

The optimal knapsack should contain {1, 2}

i=n, k=W
while i,k > 0
    if $V[i,k] \neq V[i-1,k]$ then
        mark the n^{th} item as in the knapsack
        $i = i-1$, $k = k-w_i$
    else
        $i = i-1$

알고리즘
Dynamic Programming

이영석

동적프로그래밍

- Dynamic programming (DP)
  - 문제해결 패러다임
  - 문제를 해결하기 위해 더 작은 문제를 해결하고 해를 재활용하는 방식
  - “기억하며 풀기“
- 분할 정복 기법과 유사

최장 공통 부분 수열
Longest Common Subsequence (LCS)

AGCAT
GAC

LCS ? →

AC
GC
GA

최장 공통 부분수열 문제는 LCS라고도 불린다. 이는 주어진 여러 개의 수열 모두의 부분수열이 되는 수열들 중에 가장 긴 것을 찾는 문제다.(종종 단 두 개중 하나가 되기도 한다.) 컴퓨터 과학에서 고전으로 통하는 문제이며, diff 유틸리티의 근간이 되며, 생물정보학에서도 많이 응용되고 있다.

이 문제는 연속되어 있는 공통 문자열을 찾는 최장 공통 부분문자열(longest common substring) 문제와 혼동해서는 안 된다.

두 개의 수열에 대한 해 [편집]

LCS 문제는 최적의 부분구조를 가진다. 이 문제는 더 작은, "부분문제"로 쪼개질 수 있고, 이것은 반복해서 자명한 부분문제가 될 때 까지 더 간단한 부분문제로 쪼개질 수 있다. LCS는 또한 겹치는 부분문제를 가진다. 더 높은 부분문제에 대한 풀이는 몇몇의 하위 부분문제의 풀이에 의존한다. "최적의 부분구조"와 "겹치는 부분문제"는 동적 프로그래밍이라는 가장 간단한 부분문제에서 출발하는 문제 풀이 기법으로 접근될 수 있다. 이 과정은 부분문제의 해답을 표에 저장하는 방식인 메모이제이션을 통하여 상위 단계의 부분문제에서 해답을 접근할 수 있도록 하는 과정을 필요로 한다. 이 방법은 다음과 같이 묘사된다. 두 수열 $X_{1...m}$ and $Y_{1...n}$이 주어졌을 때, 주어진 두 수열의 최장 공통 부분수열(longest common subsequence)은 다음과 같이 표현된다.

접두사 [편집]

부분문제는 수열이 짧아질 수록 간단해진다. 짧은 수열은 접두사라는 용어로 간단히 묘사된다. 어떤 수열의 접두사는 말단이 잘려나간 수열이다. S를 수열 (AGCA)라 둔다. 그러면 S의 접두사는 수열 (AG)이다. 접두사는 그 수열의 이름과 그 접두사가 포함하는 문자의 수로 정의된다.[3] 따라서 접두사 (AG)는 S_2로 명명된다. S의 가능한 접두사들은

S_1 = (A)
S_2 = (AG)
S_3 = (AGC)
S_4 = (AGCA)

이다.

임의의 두 수열 *X*와 *Y*에서, LCS문제의 해법, 즉 최장 공통 부분 수열, *LCS*(*X*, *Y*)는 다음의 두 속성에 의존한다

첫 속성 [편집]

두 수열이 같은 원소로 끝난다고 가정해보자. 그들의 LCS를 찾기 위해 마지막 원소를 지움으로써 수열의 길이를 줄이고, 짧아진 수열에 대한 LCS를 찾은 후 삭제한 원소를 붙여준다.

> 예를 들어, 같은 마지막 원소를 가진 두 수열 (BANANA)와 (ATANA)가 존재한다.
> 마지막 원소를 삭제한다. 이 과정을 공통된 마지막 원소가 존재하지 않을때까지 반복한다. 삭제된 수열은 ANA이다.
> 이제 연산해야 하는 수열은 다음과 같다. (BAN)와 (AT)
> 이 두 수열의 LCS는 (A)가 된다.
> 삭제했던 부분수열 (ANA)를 다시 결합시키면 (AANA)가 되고, 이것이 원 수열의 LCS가 된다.

접두사에서,

$$\boxed{LCS(X_n, Y_m) = (LCS(X_{n-1}, Y_{m-1}), x_n)}$$

반점은 원소 x_n가 이 수열에 붙게되는 부분을 말한다.

X_n 과 Y_m의 LCS를 계산하려면 더 짧은 수열 X_{n-1} 와 Y_{m-1}의 LCS를 계산해야 하는 점에 유의한다.

두 번째 속성 [편집]

두 수열 X, Y가 같은 기호로 끝나지 않는다고 가정한다. 그러면 X와 Y의 LCS는 LCS(Xn,Ym-1)와 LCS(Xn-1,Ym)중 더 긴 수열이다. 이 특징을 이해하기위해 다음 두 수열을 보도록 한다. 수열 X: ABCDEFG (n개의 원소) 수열 Y: BCDGK (m개의 원소) 이 두 수열의 LCS의 마지막 문자는 수열 X의 마지막 원소인 G로 끝나거나, 그렇지 않을것이다.

첫 번째 경우: LCS가 G로 끝나는 경우

이 경우 LCS는 K로 끝날 수 없다. 따라서 수열 Y에서 K를 제거하여도 손실이 일어나지 않는다. 만약 K가 LCS에 있었다면 결과적으로 K는 LCS에 존재하지 않으므로 마지막 문자였을것이다. 따라서 이렇게 표기할 수 있다. $\boxed{LCS(X_n,Y_m) = LCS(X_n, Y_{m-1})}$.

두 번째 경우: LCS가 G로 끝나지 않는 경우

이 경우 위와 같은 이유로 수열 X에서 G를 제거하여도 손실이 일어나지 않는다. 즉 이렇게 쓸 수 있다. $\boxed{LCS(X_n,Y_m) = LCS(X_{n-1}, Y_m)}$.

어떤 경우에서든지 우리가 찾는 LCS는 $LCS(X_n, Y_{m-1})$이거나 $LCS(X_{n-1}, Y_m)$이다. 이 두 LCS는 둘다 X와 Y의 공통 부분수열이다. LCS(X,Y)는 최장이다. 따라서 그 값은 $LCS(X_n, Y_{m-1})$와 $LCS(X_{n-1}, Y_m)$중의 최장 수열이다.

ABCDEFG → *ABCDEFG*

BCDGK → *BCDGK*

LCS 함수의 정의 [편집]

두 수열을 다음과 같이 정의한다. $X = (x_1, x_2 ... x_m)$, $Y = (y_1, y_2 ... y_n)$. X의 접두사는 $X_{1, 2, ... m}$이고, Y의 접두사는 $Y_{1, 2, ... n}$이다. $LCS(X_i, Y_j)$를 접두사 X_i와 Y_j의 최장 공통 부분수열을 대표한다고 둔다. 이 수열의 집합은 다음과 같이 주어진다.

$$LCS\left(X_i, Y_j\right) = \begin{cases} \emptyset & \text{if } \ i = 0 \text{ or } j = 0 \\ LCS\left(X_{i-1}, Y_{j-1}\right) + 1 & \text{if } x_i = y_j \\ \text{longest}\left(LCS\left(X_i, Y_{j-1}\right), LCS\left(X_{i-1}, Y_j\right)\right) & \text{if } x_i \neq y_j \end{cases}$$

X_i 와 Y_j의 최장 공통 부분 수열을 찾기 위해서, 두 원소 x_i와 y_j를 비교한다. 만약 그들이 같다면 수열 $LCS(X_{i-1}, Y_{j-1})$는 x_i원소로 확장된다. 만약 그들이 같지 않다면 두 수열 $LCS(X_i, Y_{j-1})$, 와 $LCS(X_{i-1}, Y_j)$중 더 긴 것이 얻어진다. (만약 그 둘이 길이가 같지만 동일하지 않다면 둘다 얻어진다.) 이 공식들에서 첨자가 1씩 감소했음을 주목하라. 이것은 첨자가 0이 되는 상황을 만들 수 있다. 수열의 원소들은 1부터 시작하는 것으로 정의되어 있으므로, 첨자가 0일때 LCS는 비어있다는 필요조건을 추가할 필요가 있다.

예시 [편집]

C = (AGCAT)와 R = (GAC)의 최장 공통 부분순열을 찾을것이다. LCS 함수는 "0번째"원소를 이용하기 때문에, 이 수열에서 비어있는 0번째 접두사를 정의하는 것이 편리하다. $C_0 = \varnothing$, 그리고 $R_0 = \varnothing$이다. 모든 접두사들은 C를 첫 번째 행에 위치하고, R을 첫 열에 위치시킨 표에 자리잡는다.

LCS Strings

	0	A	G	C	A	T
0	∅	∅	∅	∅	∅	∅
G	∅					
A	∅					
C	∅					

이 표는 연산의 각 단계에서 LCS 수열을 저장하는데 이용된다. 두 번째 행과 두 번째 열은 ∅로 채워지는데, 빈 수열이 비어있지 않은 수열과 비교될때 가장 긴 공통 부분 수열이 항상 빈 수열이 되기 때문이다.

"G" Row Completed

	Ø	A	G	C	A	T
Ø	Ø	Ø	Ø	Ø	Ø	Ø
G	Ø	←↑Ø	↖(G)	←(G)	←(G)	←(G)
A	Ø					
C	Ø					

$LCS(R_1, C_1)$는 각 수열의 첫 원소를 비교함으로써 결정된다. G와 A는 일치하지 않기 때문에, 이것의 LCS는 두 번째 속성에 의해 두 수열 $LCS(R_1, C_0)$ 와 $LCS(R_0, C_1)$중 긴 것을 갖게 된다.

표에서 보면, 이 둘 모두 비어있기 때문에 $LCS(R_1, C_1)$ 도 마찬가지로 아래쪽 표에서 볼 수 있듯이 비어있게 된다. 화살표는 수열이 위쪽의 두 셀 $LCS(R_0, C_1)$ 과 그 왼쪽 셀인 $LCS(R_1, C_0)$에서 온다는 것을 가리킨다.

$LCS(R_1, C_2)$는 G와 G를 비교함으로써 결정된다. 그들은 동일하므로, 왼쪽 위의 (Ø)의 수열$LCS(R_0, C_1)$뒤에 붙어서 (ØG)가 되므로 결과적으로 (G)가 된다.

$LCS(R_1, C_3)$에서, G와 C는 일치하지 않는다. 그 위의 수열은 비어있고, 그 왼쪽의 것은 G라는 하나의 원소를 포함한다. 이들중 가장 긴 것을 선택하면 $LCS(R_1, C_3)$는 (G)가 된다. 화살표는 왼쪽을 가리키는데, 그것이 둘 중 가장 긴 것이기 때문이다.

$LCS(R_1, C_4)$는 같은 방법으로 (G)이다.

$LCS(R_1, C_5)$또한 같은 방법으로 (G)이다.

"G" & "A" Rows Completed

	Ø	A	G	C	A	T
Ø	Ø	Ø	Ø	Ø	Ø	Ø
G	Ø	↲↑Ø	↖(G)	←(G)	←(G)	←(G)
A	Ø	↖(A)	↲↑(A) & (G)	↲↑(A) & (G)	↖(GA)	←(GA)
C	Ø					

$LCS(R_2, C_1)$에서, A는 A와 비교된다. 두 원소가 동일하므로, A는 Ø에 첨가되어 (A)가 된다.

$LCS(R_2, C_2)$에서, A와 G는 같지 않다. 따라서 두 수열$LCS(R_1, C_2)$와 $LCS(R_2, C_1)$중 가장 긴 것, 즉 (G)와 (A)중 가장 긴 것이 사용된다. 이 예시에서 그들은 하나의 원소만들 포함하므로, LCS는 두 부분 수열 (A)와 (G)로 주어진다.

$LCS(R_2, C_3)$에서, A는 C와 동일하지 않다. $LCS(R_2, C_2)$는 수열 (A) 와 (G)를 포함한다. LCS(R_1, C_3)는 (G)로, $LCS(R_2, C_2)$에 이미 포함되어있다. 결과적으로 $LCS(R_2, C_3)$ 또한 두 수열 (A) 와 (G)를 포함한다.

$LCS(R_2, C_4)$의 경우, A는 A와 같으므로, 왼쪽 위 셀에 붙어, (GA)가 된다.

$LCS(R_2, C_5)$의 경우에서, A는 T와 같지 않다. 두 수열 (GA) 와 (G)를 비교했을 때, 가장 긴것은 (GA)이므로$LCS(R_2, C_5)$ 는 (GA)이다.

Completed LCS Table

	Ø	A	G	C	A	T
Ø	Ø	Ø	Ø	Ø	Ø	Ø
G	Ø	←↑Ø	↖(G)	←(G)	←(G)	←(G)
A	Ø	↖(A)	←↑(A) & (G)	←↑(A) & (G)	↖(GA)	←(GA)
C	Ø	↑(A)	←↑(A) & (G)	↖(AC) & (GC)	←↑(AC) & (GC) & (GA)	←↑(AC) & (GC) & (GA)

$LCS(R_3, C_1)$에서, C 와 A 는 같지 않으므로, $LCS(R_3, C_1)$ 는 가장 긴 수열 (A)를 갖는다.

$LCS(R_3, C_2)$에서, C 와 G는 같지 않다. $LCS(R_3, C_1)$ 와 $LCS(R_2, C_2)$ 모두 단 하나의 원소를 가지므로 $LCS(R_3, C_2)$ 는 두 원소 (A) 와 (G)를 가지게 된다.

$LCS(R_3, C_3)$에서, C 와 C는 동일하므로, C는 두 부분수열 (A)와 (C)를 포함하는 $LCS(R_2, C_2)$에 붙어 (AC) 와 (GC)가 된다.

$LCS(R_3, C_4)$에서, C와 A는 같지 않다. (AC)와 (GC)를 포함하는 $LCS(R_3, C_3)$와 (GA)를 포함하는 $LCS(R_2, C_4)$를 조합하면 총 세 개의 수열 (AC), (GC), 그리고 (GA)를 준다.

마지막으로, $LCS(R_3, C_5)$에 대해서, C와 T는 일치하지 않는다. 결과적으로 $LCS(R_3, C_5)$ 또한 세 수열 (AC), (GC), 그리고 (GA)를 갖는다.

AGCA
T
GAC

LCS ?

AC
GC
GA

역추적 접근 [편집]

LCS 표의 한 행의 LCS를 계산하는 데에는 현재 행의

Storing length, rather than sequences

	Ø	A	G	C	A	T
Ø	0	0	0	0	0	0
G	0	←↑0	↖1	←1	←1	←1
A	0	↖1	←↑1	←↑1	↖2	←2
C	0	↑1	←↑1	↖2	←↑2	←↑2

실제 부분수열들은 표의 마지막 셀로부터 시작하여 화살표들을 거슬러 "역추적"함으로써 추론할 수 있다. 길이가 줄어들 때, 각 수열들은 반드시 공통 원소를 가진다. 두 화살표가 한 셀 안에서 있으면 여러 경로가 가능하다. 아래는 길이가 감소하는 셀에 대해 색이 칠해진 수들이 나타난 분석 과정을 나타낸 표이다. 굵은 숫자는 (GA) 수열을 찾아내는 경로이다.[4]

Traceback example

	Ø	A	G	C	A	T
Ø	0	**0**	0	0	0	0
G	0	←↑0	↖1	←**1**	←1	←1
A	0	↖1	←↑1	←↑1	↖2	←**2**
C	0	↑1	←↑1	↖2	←↑2	←↑**2**

AC
GC
GA

알고리즘
Dynamic Programming

이영석

동적프로그래밍

- Dynamic programming (DP)
  - 문제해결 패러다임
  - 문제를 해결하기 위해 더 작은 문제를 해결하고 해를 재활용하는 방식
  - "기억하며 풀기"
- 분할 정복 기법과 유사

최대증가부분수열(LIS: Longest Increasing Subsequence)

어떤 임의의 수열이 주어질 때, 몇 개의 수들을 제거해서 부분수열을 만들 수 있다.
부분수열 중 오름차순으로 정렬된 최대 증가 부분 수열 찾기

3 5 7 9 2 1 4 8

위 수열에서 몇 수를 제거해 부분수열 만들기

3 5 7 9 2 1 4 8 (5, 2 제거): LIS No
3 5 7 9 2 1 4 8 (3, 5, 2, 4 제거): LIS No
3 5 7 9 2 1 4 8 (9, 2, 1, 4 제거): LIS OK
3 5 7 9 2 1 4 8 (3, 5, 7, 9, 2 제거): LIS OK

세번째, 네번째 수열은 오름차순으로 정렬
'증가 부분 수열'

증가 부분 수열 중 가장 긴 수열을 '최대 증가 부분 수열 (LIS)'이라 한다.
부분수열 **3 5 7 8**은 LIS

한 수열에서 여러 개의 LIS가 나올 수도 있다.

5 1 6 2 7 3 8

에서 부분수열

5 1 6 2 7 3 8: 1 2 3 8
5 1 6 2 7 3 8: 5 6 7 8

은 모두 길이가 4인 LIS이다.

LIS해결방법: DP

- DP (dynamic programming: 동적계획법)
  - 복잡도 O(n^2)
- 주어진 배열
  - input[n] : n개의 문자열
- 답
  - L[x]: x 번째 수를 마지막 원소로 가지는 LIS 길이
  - L[x] 를 찾았다면, 다음에 찾아야할 것
    - x보다 큰 위치 y의 배열값 input[y] > input[x]보다 크다면 LIS 에 포함됨!!!
      - 가장 긴 이전 LIS 찾기!
      - L[y] = max(L[x]) +1

LIS Example

- input: [0] 3 5 7 9 2 1 4 8
  - 0 3 -> L[0] = 1
  - 0 3 5 -> L[1] = 2
  - 0 3 5 7 -> L[2] = 3
  - 0 3 5 7 9 -> L[3] = 4
  - 0 3 5 7 9 2 -> L[4] = 1
    - 0 다음에
  - 0 3 5 7 9 2 1 -> L[5] = 1
    - 0 다음에
  - 0 3 5 7 9 2 1 4 -> L[6] = 2
    - 0, 3 다음에
  - 0 3 5 7 9 2 1 4 8 -> L[7] = 4
    - 0, 3, 5, 7 다음에

LIS해결방법: Lower Bound

- Lower bound를 이용한 O(n log n)
  - lower bound는 정렬된 배열에 어떤 값이 삽입될 수 있는 가장 작은 인덱스
  - 현재값이 배열의 마지막 원소보다 크면 추가
  - 작으면, lower bound 위치의 값을 대체
- 예) 3 5 7 9 2 1 4 8

  3

  3 5

  3 5 7 (크면 추가)

  3 5 7 9

  2 5 7 9

  1 5 7 9

  1 4 7 9

  1 4 7 8 (8이 9를 대체)