CS3000: Algorithms & Data
Jonathan Ullman

Lecture 7:

- Dynamic Programming: Segmented Least Squares

Sep 28, 2018

Dynamic Programming Recap

- **Recipe:**
  - (1) identify a set of **subproblems**
  - (2) relate the subproblems via a **recurrence**
  - (3) find an **efficient implementation** of the recurrence
  - (4) **reconstruct the solution** from the DP table

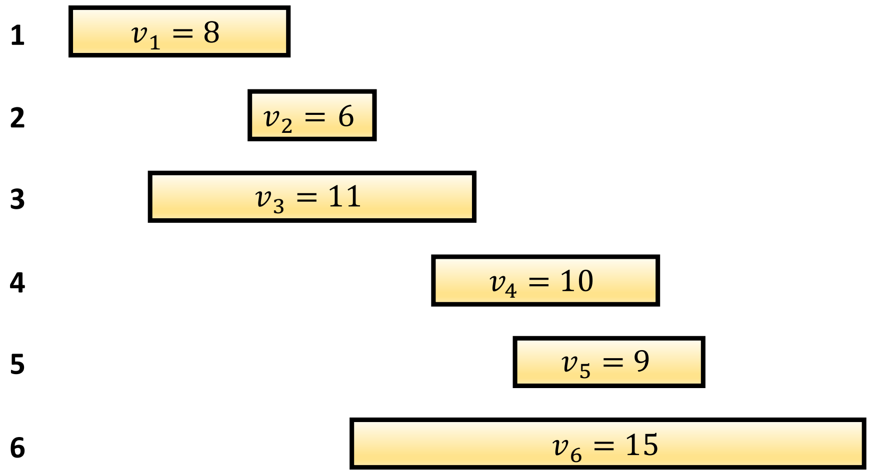
Dynamic Programming Recap

Goal: Find a schedule maximizing total value

$OPT(j)$ = value of opt sched. for $\{1, \ldots, j\}$

$OPT(0) = 0$

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



$p(1) = 0$

$p(2) = 0$

$p(3) = 0$

$p(4) = 2$

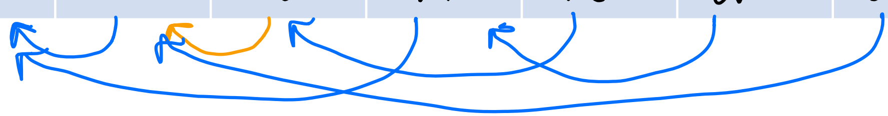
$p(5) = 3$

$p(6) = 1$

$O_6 = \{6, 1\}$

$O_4 = \{4, 1\}$

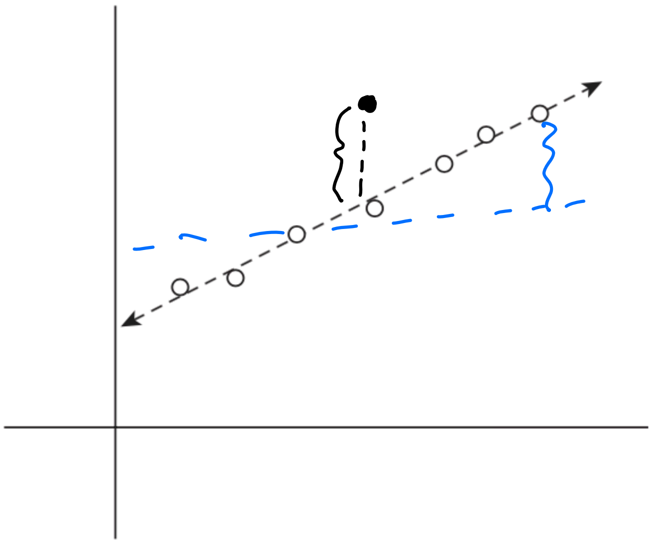
M[0]	M[1]	M[2]	M[3]	M[4]	M[5]	M[6]
0	8	8	11	18	20	23



Segmented Least Squares

Background: Least Squares

- **Input:** n data points $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- **Output:** the line L (i.e. $y = ax + b$) that fits **best**
  - **best** = minimizes $error(L, P) = \sum_i (y_i - ax_i - b)^2$



$$a = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{n\sum x_i^2 - (\sum x_i)^2}$$

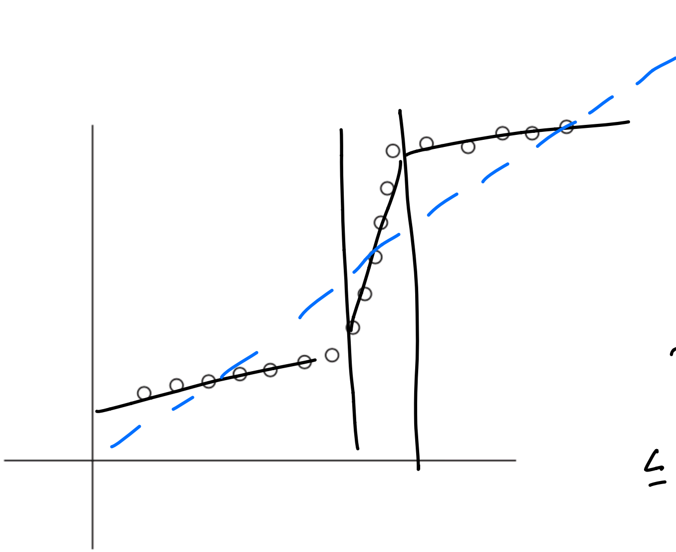
$$b = \frac{\sum y_i - a\sum x_i}{n}$$

Can find least squares estimate in $O(n)$ time

Segmented Least Squares

- **Input:** n data points $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- What if the data does not look like a line?

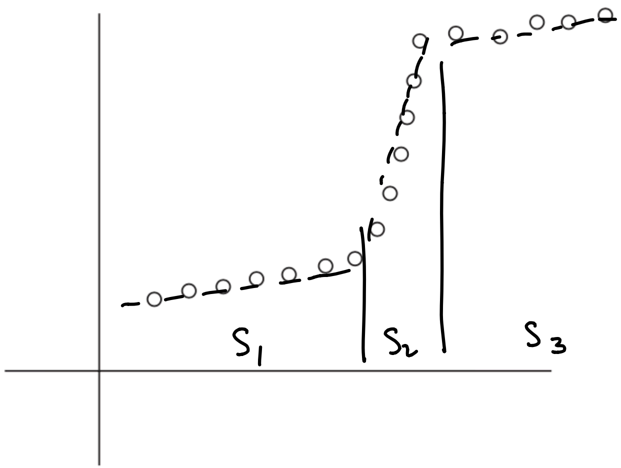
piecewise linear function



Any n points lie on a piecewise linear function with $\leq \frac{n}{2}$ pieces.

Segmented Least Squares

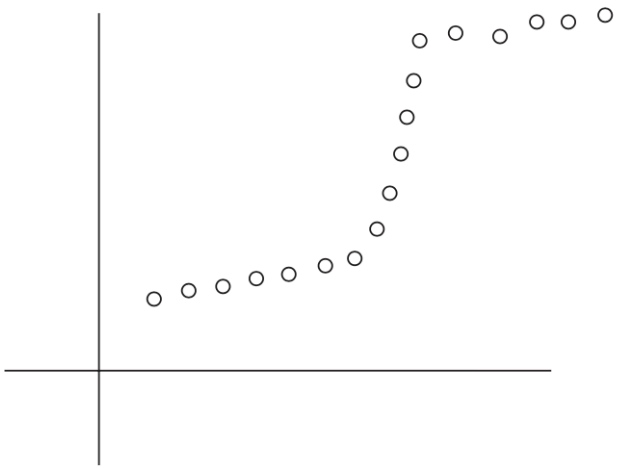
- **Input:** n data points $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, **cost parameter** $C > 0$
  - Assume $x_1 < x_2 < \cdots < x_n$
- **Output:** a partition of P into contiguous segments $S_1, S_2, \ldots, S_m$, lines $L_1, L_2, \ldots, L_m$, minimizing “cost”



$$\text{cost}(S_1, \ldots, S_m, L_1, \ldots, L_m) = Cm + \sum_{j=1}^{m} \text{error}(L_j, S_j)$$

Segmented Least Squares

- **First observation:** for every segment S_j, L_j must be the (single) line of best fit for S_j
  - Let $L^*_{i,j}$ be the optimal line for $\{p_i, \ldots, p_j\}$
  - Let $\varepsilon_{i,j} = error\left(L^*_{i,j}, \{p_i, \ldots, p_j\}\right)$

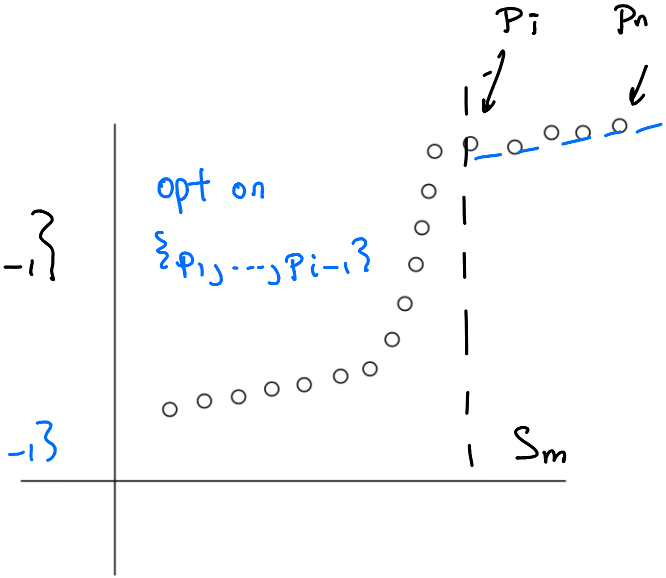


Can compute in $O(n^2)$ time.
(Easy to do in $O(n^3)$)

SLS

Let $L^*_{i,j}$ be the optimal line for $\{p_i, \ldots, p_j\}$
Let $\varepsilon_{i,j} = error(L^*_{i,j}, \{p_i, \ldots, p_j\})$

- Let O be the **optimal** solution
- O contains some final segment $\{p_i, \ldots, p_n\}$ $(1 \le i \le n)$
- If the final segment is $\{p_i, \ldots, p_n\}$ then the optimal segments are
  (1) $\{p_i, \ldots, p_n\}$
  \+ (2) optimal segments for $\{p_1, \ldots, p_{i-1}\}$



- cost would be $\varepsilon_{i,n} + C +$ opt cost on $\{p_1, \ldots, p_{i-1}\}$

SLS

Let $L^*_{i,j}$ be the optimal line for $\{p_i, \ldots, p_j\}$
Let $\varepsilon_{i,j} = error(L^*_{i,j}, \{p_i, \ldots, p_j\})$

- Let $\mathrm{OPT}(j)$ be the **value** of the optimal solution for points $\{p_1, \ldots, p_j\}$ for $0 \le j \le n$
- **Case *i*:** final segment is $\{p_i, \ldots, p_j\}$ for some $1 \le i \le j$
  - optimal solution is $L^*_{i,j} \cup$ optimal sol. for $\{p_1, \ldots, p_{i-1}\}$
  - can use any $i \in \{1, \ldots, j\}$

$$\mathrm{OPT}(j) = \min_{i:\, 1 \le i \le j} \varepsilon_{i,j} + C + \mathrm{OPT}(i-1)$$

$\mathrm{OPT}(0) = 0 \qquad \mathrm{OPT}(1) = C \qquad \mathrm{OPT}(2) = C$

SLS

Let $L^*_{i,j}$ be the optimal line for $\{p_i, \dots, p_j\}$
Let $\varepsilon_{i,j} = error(L^*_{i,j}, \{p_i, \dots, p_j\})$

- Let $\mathrm{OPT}(j)$ be the **value** of the optimal solution for points $\{p_1, \dots, p_j\}$
- **Case i:** final segment is $\{p_i, \dots, p_j\}$
  - optimal solution is $L^*_{i,j}$ ∪ optimal sol. for $\{p_1, \dots, p_{i-1}\}$
  - can use any $i \in \{1, \dots, j\}$

Recurrence: $\mathrm{OPT}(j) = \min_{1 \le i \le j} \varepsilon_{i,j} + C + \mathrm{OPT}(i-1)$

Base cases: $\mathrm{OPT}(0) = 0$
$\mathrm{OPT}(1) = \mathrm{OPT}(2) = C$

SLS: Take I

```
// All inputs are global vars
FindOPT(n):
  if (n = 0): return 0
  elseif (n = 1,2): return C
  else:
    return min_{1≤i≤n} ε_{i,n} + C + FindOPT(i − 1)
```

$T(n) = $ # of recursive calls

$$T(n) = \sum_{i=0}^{n-1} T(i) \geq T(n-1) + T(n-2)$$

$$T(n) = \Omega(1.62^n)$$

SLS: Take II ("Top-Down")

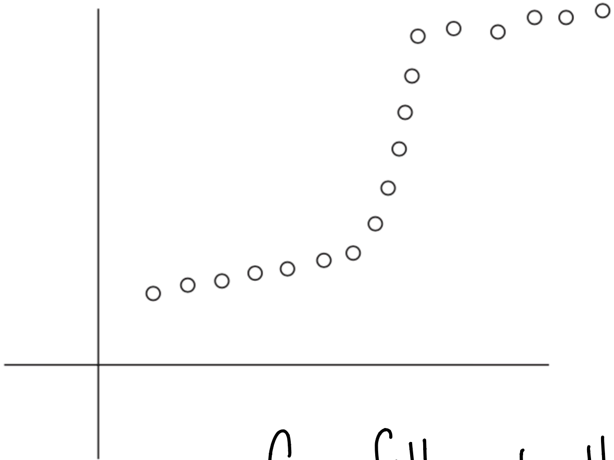
```
// All inputs are global vars
M ← empty array, M[0] ← 0, M[1] ← C, M[2] ← C
FindOPT(n):
  if (M[n] is not empty): return M[n]
  else:
    M[n] ← min_{1≤i≤n} ε_{i,n} + C + FindOPT(i − 1)
    return M[n]
```

$$M[n] \leftarrow \min_{1 \le i \le n} \varepsilon_{i,n} + C + \text{FindOPT}(i-1)$$

# of recursive calls (n+1 arrays elements) × (≤ n calls)

$= O(n^2)$

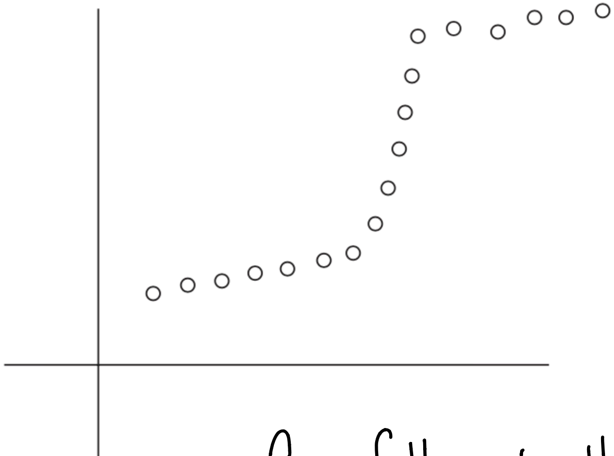
SLS: Take III ("Bottom-Up")



Can fill out the table left to right

M[0]	M[1]	M[2]	...	M[i]	...	M[n]
0	c	c	...		...	716.8

SLS: Take III ("Bottom-Up")



Can fill out the table left to right

M[0]	M[1]	M[2]	…	M[i]	…	M[n]
0	c	c	…		…	716.8

SLS: Take III ("Bottom-Up")

```
// All inputs are global vars
FindOPT(n):
  M[0] ← 0, M[1] ← C, M[2] ← C
  for (j = 3,…,n):
    M[j] ← min_{1≤i≤j} ε_{i,j} + C + M[i − 1]
  return M[n]
```

$n-2$ (loop iterations); $O(n)$ (per iteration)

Running time is $O(n^2)$ + (time to compute $\{\varepsilon_{i,j}\}$)

Rule of Thumb: running time $O(\#$ of dependencies btw problems$)$

Finding Segments

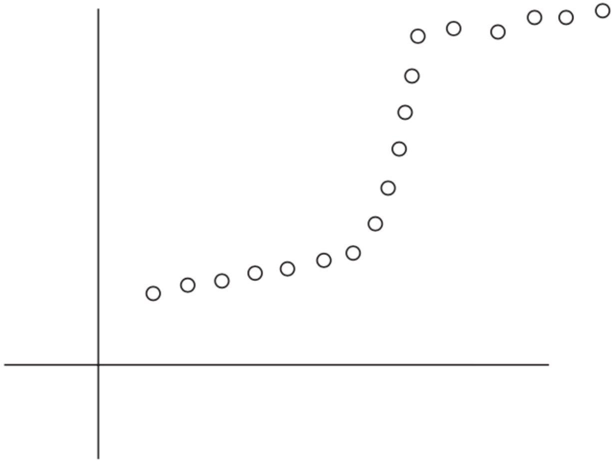
Let $L^*_{i,j}$ be the optimal line for $\{p_i, \dots, p_j\}$
Let $\varepsilon_{i,j} = error(L^*_{i,j}, \{p_i, \dots, p_j\})$

- Let $\mathrm{OPT}(j)$ be the **value** of the optimal solution for points $\{p_1, \dots, p_j\}$
- **Case i:** final segment is $\{p_i, \dots, p_j\}$
  - optimal solution is $L^*_{i,j} \cup$ optimal sol. for $\{p_1, \dots, p_{i-1}\}$
  - can use any $i \in \{1, \dots, j\}$
- If the final segment is $\{p_i, \dots, p_j\}$ then cost is $\varepsilon_{i,j} + C + \mathrm{OPT}(i-1)$
- If $i \in \operatorname{argmin}_{1 \le i \le j} \varepsilon_{i,j} + C + \mathrm{OPT}(i-1)$ then there is an optimal solution uses $\{p_i, \dots, p_j\}$ as a final segment + optimal solution for $\{p_1, \dots, p_{i-1}\}$

Finding Segments

```
// All inputs are global vars
// M[0:n] contains solutions to subproblems
FindSol(M,n):
  if (n = 0): return ∅
  elseif (n = 1): return {1}
  else:
    Let i ← argmax_{1≤i≤n} ε_{i,n} + C + M[i − 1]:
    return {i,…,n} + FindSol(M,i-1)
```

SLS: Take III ("Bottom-Up")



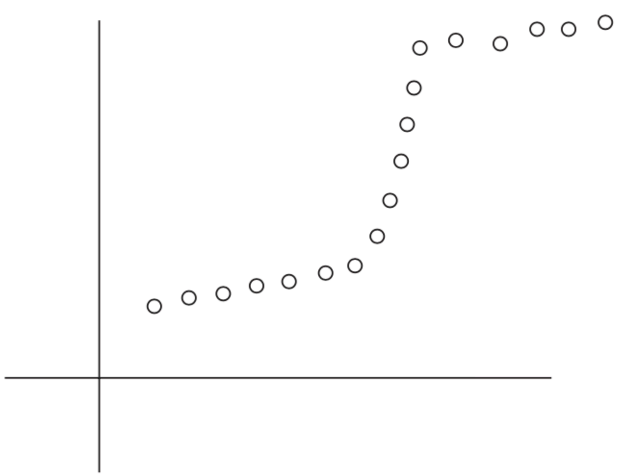
$$M[n] = \varepsilon_{i,n} + C + OPT(i-1)$$

M[0]	M[1]	M[2]	...	M[i]	...	M[n]
0	c	c	...		...	716.8

Segmented Least Squares v.2

Segmented Least Squares v.2

- **Input:** n data points $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, parameter $1 \leq k \leq n$
  - Hard upper bound on the number of segments
- **Output:** a partition of P into $\leq k$ contiguous segments $S_1, S_2, \ldots, S_k$ minimizing "cost"

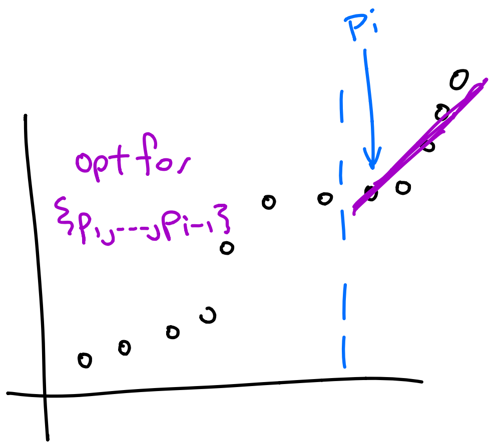


$$\text{cost}(S_1, \ldots, S_k, L_1 \ldots L_k) = \sum_{j=1}^{k} \text{error}(S_j, L_j)$$

SLSv.2

Let $L^*_{i,j}$ be the optimal line for $\{p_i, \ldots, p_j\}$
Let $\varepsilon_{i,j} = error(L^*_{i,j}, \{p_i, \ldots, p_j\})$

- Let O be the optimal solution
- O uses some final segment $\{p_i, \ldots, p_n\}$
- Possible recurrence: $OPT(n) = \min_{1 \le i \le n} \varepsilon_{i,n} + OPT(i-1)$



problem: O_{i-1} uses k segments so we get $k+1$ segments total

idea: find the optimal solution using $k-1$ segments

SLSv.2

Let $L^*_{i,j}$ be the optimal line for $\{p_i, \dots, p_j\}$
Let $\varepsilon_{i,j} = error(L^*_{i,j}, \{p_i, \dots, p_j\})$

- Let $\mathrm{OPT}(j, \ell)$ be the optimal solution for points $\{1, \dots, j\}$ using $\leq \ell$ segments
- **Case $\boldsymbol{i}$:** final segment is $\{p_i, \dots, p_j\}$
  - optimal solution is $L^*_{i,j} \cup$ optimal solution for points $\{p_1, \dots, p_{i-1}\}$ using $\leq \ell - 1$ segments
  - can use any $i \in \{1, \dots, j\}$

Recurrence: $\mathrm{OPT}(j, \ell) = \min_{1 \leq i \leq j} \varepsilon_{i,j} + \mathrm{OPT}(i-1, \ell-1)$

Base cases: $\mathrm{OPT}(0, \ell) = 0 \quad \forall \ell \geq 0$
$\mathrm{OPT}(j, 0) = \infty \quad \forall j \geq 1$

SLSv.2: Take II ("Top-Down")

```
// All inputs are global vars
M ← empty array, M[0,ℓ] ← 0, M[j,0] ← ∞
FindOPT(n,k):
  if (M[n,k] is not empty): return M[n,k]
  else:
    M[n,k] ← min_{1≤i≤n} ε_{i,n} + FindOPT(i − 1, k − 1)
    return M[n,k]
```

$(n+1)(k+1)$ subproblems

$\times$ n calls per subproblem.

$O(n^2k)$

SLSv.2: Take III ("Bottom-Up")

$M[j,\ell] = OPT(j,\ell)$

Fill the table one column at a time left to right

	M[·,0]	M[·,1]	M[·,2]	M[·,3]	…	M[·,k]
M[0, ·]	0	0	0	0	0	0
M[1, ·]	∞					
M[2, ·]	∞					
M[3, ·]	∞					
…	∞				M[j,ℓ]	
M[n, ·]	∞					

Because $OPT(j,\ell)$ depends on $OPT(i,\ell-1)$ for $i<j$ all "arrows" go up and left

SLSv.2: Take III ("Bottom-Up")

```
// All inputs are global vars
FindOPT(n,k):
  M[0,ℓ] ← 0, M[j,0] ← ∞
  for (ℓ = 1,…,k):
    for (j = 1,…,n):
    M[j,ℓ] ← min_{1≤i≤j} ε_{i,j} + FindOPT(j − 1, ℓ − 1)   ] O(n)
  return M[n,k]
```

k iter

n iters

$O(n^2 k)$ time

$O(nk)$ space for M (+ $O(n^2)$ for $\{\varepsilon_{i,j}\}$)

SLSv.2: Finding Segments

```
// All inputs are global vars
// M[0:n,0:k] contains solutions to subproblems
FindSol(M,n,k):
  if (n = 0): return ∅
  elseif (n = 1): return {1}
  else:
    let i ← argmax_{1≤i≤n} ε_{i,n} + M[i − 1, k − 1]:
    return {i,…,n} + FindSol(M,i-1,k-1)
```

SLS Wrapup

- **Version 1:** can solve SLS with a "segment cost" in time $O(n^2)$ space $O(n^2)$
  - New idea: multiway case analysis for the final segment
- **Version 2:** can solve SLS with a "hard cap" of k segments in time $O(n^2 k)$ space $O(n^2 + nk)$
  - New idea: introducing additional variables to expand the set of subproblems
- Correctness follows from the recurrence
- Computational costs:
  - Running time $\approx$ total number of terms in all recurrences
  - Space $\approx$ total number of subproblems