COMP3711: Design and Analysis of Algorithms

DP Tutorial 1

COMP3711: Design and Analysis of Algorithms

Longest Monotonically Increasing Subsequence

Version of April 1, 2020

Question 1

Give an $O(n^2)$ time dynamic programming algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers, i.e, each successive number in the subsequence is greater than or equal to its predecessor.

For example, if the input sequence is

$$\langle 5, 24, 8, 17, 12, 45 \rangle,$$

the output should be either $\langle 5, 8, 12, 45 \rangle$ or $\langle 5, 8, 17, 45 \rangle$.

Solution

We first give an algorithm which finds the **length** of the longest increasing subsequence; will later modify it to report a subsequence with this length.

Let $X_i = \langle x_1, \cdots, x_i \rangle$ denote the prefix of X consisting of its first i items.

Define

$c[i]$ = the length of the longest increasing subsequence that **ends** at x_i .

The length of the longest increasing subsequence in X is then

$$\max_{1 \le i \le n} c[i].$$

Solution

$c[i]$ = the length of the longest increasing subsequence that **ends** at x_i.

Initial Condition: $\mathrm{c}[1] = 1$

If $i > 1$:

If all items to left of x_i are $>$ than x_i, answer must be 1.

Otherwise, longest increasing subsequence **that ends with x_i** has form $\langle Z, x_i \rangle$, where Z is the longest increasing subsequence **that ends with x_r** for some $r < i$ and $x_r \le x_i$.

This yields the following recurrence relation:

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \max\limits_{\substack{1 \le r < i \\ x_r \le x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

Solution

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \max\limits_{\substack{1 \le r < i \\ x_r \le x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

We do not write the pseudocode, but just note that we store the $c[i]$'s in an array whose entries are computed in order of increasing i .

After computing the c array, we run through all the entries to find the maximum value.

This is the length of the longest increasing subsequence in X.

For every i it takes $O(i)$ time to calculate c_i.

=> the running time is $O(\sum_{i=1}^{n} i) = O(n^2)$.

example

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \max\limits_{\substack{1 \le r < i \\ x_r \le x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

Question:

The input sequence is $X = \{4, 5, 7, 1, 3, 9\}$;

Find the longest monotonically increasing subsequence.

i	1	2	3	4	5	6
X	4	5	7	1	3	9
c[i]	1					

Solution:

$i = 1$: $c[1] = 1$

example

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \max\limits_{\substack{1 \le r < i \\ x_r \le x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

Question:

The input sequence is $X = \{4, 5, 7, 1, 3, 9\}$;
Find the longest monotonically increasing subsequence.

i	1	2	3	4	5	6
X	4	5	7	1	3	9
c[i]	1	2				

Solution:

$i = 1$: $c[1] = 1$

$i = 2$: Since $x_1 \le x_2 \Rightarrow c[2] = \max\{c[1]\} + 1 = 2$

example

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \max\limits_{\substack{1 \le r < i \\ x_r \le x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

Question:

The input sequence is $X = \{4, 5, 7, 1, 3, 9\}$;
Find the longest monotonically increasing subsequence.

i	1	2	3	4	5	6
X	4	5	7	1	3	9
c[i]	1	2	3			

Solution:

$i = 1$: $c[1] = 1$

$i = 2$: Since $x_1 \le x_2 \Rightarrow c[2] = \max\{c[1]\} + 1 = 2$

$i = 3$: Since $x_1, x_2 \le x_3 \Rightarrow \mathrm{c}[3] = \max\{c[1], c[2]\} + 1 = 2 + 1 = 3$

example

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \max\limits_{\substack{1 \le r < i \\ x_r \le x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

Question:

The input sequence is $X = \{4, 5, 7, 1, 3, 9\}$;
Find the longest monotonically increasing subsequence.

i	1	2	3	4	5	6
X	4	5	7	1	3	9
c[i]	1	2	3	1		

Solution:

$i = 1$: $c[1] = 1$

$i = 2$: Since $x_1 \le x_2 \Rightarrow c[2] = \max\{c[1]\} + 1 = 2$

$i = 3$: Since $x_1, x_2 \le x_3 \Rightarrow \mathrm{c}[3] = \max\{c[1], c[\,2]\} + 1 = 2 + 1 = 3$

$i = 4$: Since $x_1, x_2, x_3 > x_4 \Rightarrow \mathrm{c}[4] = 1$

example

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \max\limits_{\substack{1 \le r < i \\ x_r \le x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

Question:

The input sequence is $X = \{4, 5, 7, 1, 3, 9\}$;
Find the longest monotonically increasing subsequence.

i	1	2	3	4	5	6
X	4	5	7	1	3	9
c[i]	1	2	3	1	2	

Solution:

$i = 1$: $c[1] = 1$

$i = 2$: Since $x_1 \le x_2 \Rightarrow c[2] = \max\{c[1]\} + 1 = 2$

$i = 3$: Since $x_1, x_2 \le x_3 \Rightarrow \mathrm{c}[3] = \max\{c[1], c[2]\} + 1 = 2 + 1 = 3$

$i = 4$: Since $x_1, x_2, x_3 > x_4 \Rightarrow \mathrm{c}[4] = 1$

$i = 5$: Since $x_4 \le x_5$ and $x_1, x_2, x_3 > x_5 \Rightarrow \mathrm{c}[5] = \max\{c[4]\} + 1 = 2$

example

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \leq r < i \\ \max\limits_{\substack{1 \leq r < i \\ x_r \leq x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

Question:

The input sequence is $X = \{4, 5, 7, 1, 3, 9\}$;
Find the longest monotonically increasing subsequence.

i	1	2	3	4	5	6
X	4	5	7	1	3	9
c[i]	1	2	3	1	2	4

Solution:

Return: max is $c[6] = 4$

$i = 1$: $c[1] = 1$

$i = 2$: Since $x_1 \leq x_2 \;\Rightarrow\; c[2] = \max\{c[1]\} + 1 = 2$

$i = 3$: Since $x_1, x_2 \leq x_3 \Rightarrow c[3] = \max\{c[1], c[\,2]\} + 1 = 2 + 1 = 3$

$i = 4$: Since $x_1, x_2, x_3 > x_4 \;\Rightarrow\; c[4] = 1$

$i = 5$: Since $x_4 \leq x_5$ and $x_1, x_2, x_3 > x_5 \;\Rightarrow c[5] = \max\{c[4]\} + 1 = 2$

$i = 6$: Since $x_1, x_2, x_3, x_4, x_5 \leq x_6 \Rightarrow c[6] = \max\{c[1], c[2], c[3], c[4], c[5]\} + 1 = 4$

Solution

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \displaystyle\max_{\substack{1 \le r < i \\ x_r \le x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

To report optimal subsequence, we need to store for each i, not only $c[i]$, but also value of r which achieves the maximum in the recurrence relation.

Denote this by $r[i]$. ($\emptyset$ means no predecessor)

i	1	2	3	4	5	6
X	4	5	7	1	3	9
c[i]	1	2	3	1	2	4
r[i]	$\emptyset$	1	2	$\emptyset$	4	3

Suppose $c[k] = \max_{1 \le i \le n} c[i]$. Let *S* be optimal subsequence

x_k is the last item in *S*. the optimal subsequence.

2nd to last item in *S* is $x_{r[k]}$,

3rd to last item in *S* is $x_{r[r[k]]}$, etc.

until we have found all the items in *S*

Running time of this step is $O(n)$, so entire algorithm is still $O(n^2)$.

Solution

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \max\limits_{\substack{1 \le r < i \\ x_r \le x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

To report optimal subsequence, we need to store for each i, not only $c[i]$, but also value of r which achieves the maximum in the recurrence relation.

Denote this by $r[i]$. ($\emptyset$ means no predecessor)

i	1	2	3	4	5	6
X	4	5	7	1	3	9
c[i]	1	2	3	1	2	4
r[i]	∅	1	2	∅	4	3

Return: max is $\mathrm{c}[6] = 4$, so $k = 6$

Solution is

$x_{r[r[r[6]]]} \leftarrow x_{r[r[6]]} \leftarrow x_{r[6]} \leftarrow x_6$
i.e. $x_1 \leftarrow x_2 \leftarrow x_3 \leftarrow x_6$
i.e. {4, 5, 7, 9}

$r[6] = 3$
$r[r[6]] = r[3] = 2$
$r[r[r[6]]] = r[2] = 1$
$r[r[r[r[6]]] = r[1] = \emptyset$

Alternative Solution

This problem can also be solved using the Longest Common Subsequence (LCS) Algorithm

Let $X = \langle x_1, \cdots, x_n \rangle$ be the original input.
Set $Y = \langle y, \cdots, y_m \rangle$ be the items from X sorted.

Example: $\mathrm{X} = \langle 5, 24, 8, 17, 12, 45, 12 \rangle$, $\mathrm{Y} = \langle 5, 8, 12, 12, 17, 24, 45 \rangle$

Then LCS(X, Y) is exactly the Longest Increasing Subsequence of X (why?)

Alternative Solution

This problem can also be solved using the
Longest Common Subsequence (LCS) Algorithm

Let $X = \langle x_1, \cdots, x_n \rangle$ be the original input.
Set $Y = \langle y, \cdots, y_m \rangle$ be the items from X sorted.

Example: $\mathrm{X} = \langle 5, 24, 8, 17, 12, 45, 12 \rangle$, $\mathrm{Y} = \langle 5, 8, 12, 12, 17, 24, 45 \rangle$

Then LCS(X, Y) is exactly the Longest Increasing Subsequence of X (why?)

Since LCS(X, Y) uses $O(n^2)$ time, this new algorithm also uses $O(n^2)$ time.

Surprisingly, there is also an $O(n \log n)$ algorithm for solving the problem. See https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/LongestIncreasingSubsequence.pdf

The Subset Sum Problem

Question

The subset sum problem is: Given a set of n positive integers, $S = \{x_1, x_2, \cdots, x_n\}$ and an integer W, determine whether there is a subset $S' \subseteq S$, such that the sum of the elements in S' is equal to W.

For example, let $S = \{4,2,8,9\}$.

If $W = 11$, then the answer is "yes"
because the elements of $S' = \{2,9\}$ sum to 11.
If $W = 7$, the answer is "no".

Give a dynamic programming solution to the subset sum problem that runs in $O(nW)$ time.
Justify the correctness and running time of your algorithm.

Solution

Define a Boolean array $A[i, j]$, $0 \le i \le n$ and $0 \le j \le W$ as follows:

$A[i, j] = True$, if there is a subset of $\{x_1, x_2, \cdots, x_i\}$ that sums to j,
$A[i, j] = False$, otherwise

Easy Cases:

- For all i, $A[i, 0] = True$ (choosing no items equals 0)
- For all $j > 0$, $A[0, j] = False$.
- If $x_i > j$ $A[i\,, j] = A[i-1, j]$, because item i is too large to use

Otherwise, $A[i, j] = (A[i-1, j-x_i]$ OR $A[i-1, j])$

(i) because either solution uses x_i.

This can only happen if $j - x_i$ can be solved with first $i-1$ items

(ii) or solution does not use x_i

in which case j can be solved with first $i-1$ items

$$\mathrm{A}[i, j] = \begin{cases} \text{True} & \text{if } j = 0 \\ \text{False} & \text{if } i = 0 \text{ and } j > 0 \\ A[i-1, j] & \text{if } x_i > j \\ A[i-1, j-x_i] \text{ OR } A[i-1, j] & \text{Otherwise} \end{cases}$$

Solution

$$A[i,j] = \begin{cases} \text{True} & \text{if } j = 0 \\ \text{False} & \text{if } i = 0 \text{ and } j > 0 \\ A[i-1,j] & \text{if } x_i > j \\ A[i-1,j-x_i] \text{ OR } A[i-1,j] & \text{Otherwise} \end{cases}$$

Dynamic-SubsetSum(x, n, W)

- $A[0,0] = True$
- for $j = 1$ to W do
  - $A[0,j] = False$
- for $i = 1$ to n do
  - $A[i,0] = True$
  - for $j = 1$ to W do
    - if $x_i > j$ then
      - $A[i,j] = A[i-1,j]$
    - else $A[i,j] = (A[i-1,j-x_i] \text{ OR } A[i-1,j])$

It is easy to see that this runs in $O(nW)$ time.

There will be a solution if and only if $A[n,W] = True$.

Example

$$A[i,j] = \begin{cases} \text{True} & \text{if } j = 0 \\ \text{False} & \text{if } i = 0 \text{ and } j > 0 \\ A[i-1,j] & \text{if } x_i > j \\ A[i-1,j-x_i] \text{ OR } A[i-1,j] & \text{Otherwise} \end{cases}$$

Question:

Given a set $S = \{7,4,8,2,5,3\}$, and W=6, determine whether there is a subset, $S' \subseteq S$, such that the sum of the elements in S' is equal to W.

Solution: we use a table to store A

i \ j	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
1	T						
2	T						
3	T						
4	T						
5	T						
6	T						

$\boldsymbol{i = 0,\ j > 0}$

$\boldsymbol{j = 0}$

Example

$$A[i,j] = \begin{cases} \text{True} & \text{if } j = 0 \\ \text{False} & \text{if } i = 0 \text{ and } j > 0 \\ A[i-1,j] & \text{if } x_i > j \\ A[i-1,j-x_i] \text{ OR } A[i-1,j] & \text{Otherwise} \end{cases}$$

Question:

Given a set $S = \{7,4,8,2,5,3\}$, and W=6, determine whether there is a subset, $S' \subseteq S$, such that the sum of the elements in S' is equal to W.

Solution: we use a table to store A

i \ j	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
1	T	F	F	F	F	F	F
2	T						
3	T						
4	T						
5	T						
6	T						

$\boldsymbol{x_1 = 7 > j = 1,..6}$

so, for all j

$A[1,j] = A[0,j]$

Example

$$A[i,j] = \begin{cases} \text{True} & \text{if } j = 0 \\ \text{False} & \text{if } i = 0 \text{ and } j > 0 \\ A[i-1,j] & \text{if } x_i > j \\ A[i-1,j-x_i] \text{ OR } A[i-1,j] & \text{Otherwise} \end{cases}$$

Question:

Given a set $S = \{7,4,8,2,5,3\}$, and W=6, determine whether there is a subset, $S' \subseteq S$, such that the sum of the elements in S' is equal to W.

Solution: we use a table to store A

i \ j	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
1	T	F	F	F	F	F	F
2	T	F	F	F	T	F	F
3	T						
4	T						
5	T						
6	T						

$\boldsymbol{x_2 = 4 \leq j = 4,5,6}$

so, when $j = 4{,}5{,}6$

$A[2,j] = (A[1,j-4] \; OR \; A[1,j])$

e.g. $A[2,4] = (A[1,0] \; OR \; A[1,4])$

$\qquad = \quad T \quad OR \quad F \; = T$

$\boldsymbol{x_2 = 4 > j = 1,2,3}$

so, when $j = 1{,}2{,}3$

$A[2,j] = A[1,j]$

Example

$$A[i,j] = \begin{cases} \text{True} & \text{if } j = 0 \\ \text{False} & \text{if } i = 0 \text{ and } j > 0 \\ A[i-1,j] & \text{if } x_i > j \\ A[i-1,j-x_i] \text{ OR } A[i-1,j] & \text{Otherwise} \end{cases}$$

Question:

Given a set $S = \{7, 4, 8, 2, 5, 3\}$, and W=6, determine whether there is a subset, $S' \subseteq S$, such that the sum of the elements in S' is equal to W.

Solution: we use a table to store A

i \ j	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
1	T	F	F	F	F	F	F
2	T	F	F	F	T	F	F
3	T	F	F	F	T	F	F
4	T						
5	T						
6	T						

$\boldsymbol{x_3 = 8 > j = 1, ..6}$

So, for all j
$A[3,j] = A[2,j]$

Example

$$A[i,j] = \begin{cases} \text{True} & \text{if } j = 0 \\ \text{False} & \text{if } i = 0 \text{ and } j > 0 \\ A[i-1,j] & \text{if } x_i > j \\ A[i-1, j-x_i] \text{ OR } A[i-1,j] & \text{Otherwise} \end{cases}$$

Question:

Given a set $S = \{7,4,8,2,5,3\}$, and W=6, determine whether there is a subset, $S' \subseteq S$, such that the sum of the elements in S' is equal to W.

Solution: we use a table to store A

i \ j	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
1	T	F	F	F	F	F	F
2	T	F	F	F	T	F	F
3	T	F	F	F	T	F	F
4	T	F	T	F	T	F	T
5	T						
6	T						

$\boldsymbol{x_4 = 2 \le j = 2,3,4,5,6}$

So, when j = 2, …, 6

$A[4,j] = (A[3,j-2] \; OR \; A[3,j]\,)$

e.g. $A[4,2] \; = (A[3,0] \; OR \; A[3,2])$

$= T \; OR \; F = T$

$\boldsymbol{x_4 = 2 > j = 1}$

so $A[4,1] = A[3,1] = F$

Example

$$A[i,j] = \begin{cases} \text{True} & \text{if } j = 0 \\ \text{False} & \text{if } i = 0 \text{ and } j > 0 \\ A[i-1,j] & \text{if } x_i > j \\ A[i-1,j-x_i] \text{ OR } A[i-1,j] & \text{Otherwise} \end{cases}$$

Question:

Given a set $S = \{7,4,8,2,5,3\}$, and W=6, determine whether there is a subset, $S' \subseteq S$, such that the sum of the elements in S' is equal to W.

Solution: we use a table to store A

i \ j	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
1	T	F	F	F	F	F	F
2	T	F	F	F	T	F	F
3	T	F	F	F	T	F	F
4	T	F	T	F	T	F	T
5	T	F	T	F	T	T	T
6	T	F	T	T	T	T	$\boxed{\text{T}}$

Fill the table

Return: $A[6,6] = True$

DP Maximum Contiguous Subarray

The Maximum Subarray Problem: A DP solution

Input: Profit history of a company. Money earned/lost each year.

Year	1	2	3	4	5	6	7	8	9
Profit (M$)	3	2	1	-7	5	2	-1	3	-1

Problem: Find the span of years in which the company earned the most

Answer: Year 5-8 , 9 M$

Formal definition:

Input: An array of numbers $A[1 \ldots n]$, both positive and negative

Output: Find the maximum value $V(k,i)$, where $V(k,i) = \sum_{t=k}^{i} A[t]$

Recall

Previously learnt 4 different algorithms for solving this problem

- $\Theta(n^3)$ Brute force Algorithm
- $\Theta(n^2)$ (Reuse of Information) Algorithm
- $\Theta(n \log n)$ Divide-and-Conquer Algorithm
- $\Theta(n)$ Linear Scan Algorithm

- Now: design a $\Theta(n)$ Dynamic Programming Algorithm

Note: previous algorithms solved a slightly different problem than the one defined on the previous page. The problems differ (ONLY) in the case that for all $\boldsymbol{i}$, $\boldsymbol{A}[\boldsymbol{i}] < \mathbf{0}$.

In that case, the old algorithms returned the value 0.
The problem as defined on the previous page returns $\max_i A[i]$.

Easy to transform the solution of one problem to that of the other in $\Theta(n)$ *time.*

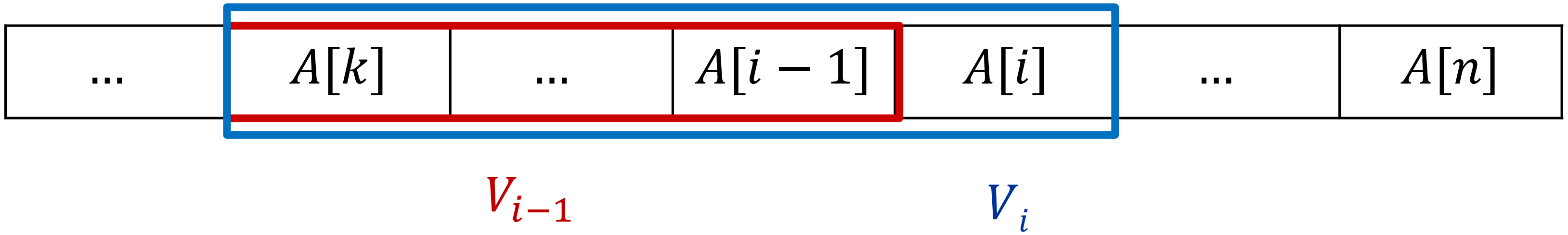
A dynamic programming ($\Theta(n)$) algorithm

Define: V_i to be max value subarray ending at $A[i]$

$$V_i = \max_{1 \le k \le i} V(k, i)$$

The main observation is that if $V_i \neq A[i] = V(i,i)$ then

$$V_i = A[i] + \max_{1 \le k < i} V(k, i-1) = A[i] + V_{i-1}$$



This immediately implies DP Recurrence

$$V_i = \begin{cases} A[1] & \text{if } i = 1 \\ \max\{A[i], A[i] + V_{i-1}\} & \text{if } i > 1 \end{cases}$$

The DP recurrence

Set $V_i = \max_{1 \le k \le i} V(k,i)$. We just saw

$$V_i = \begin{cases} A[1] & \text{if } i = 1 \\ \max\{A[i], A[i] + V_{i-1}\} & \text{if } i > 1 \end{cases}$$

Original problem then becomes finding i' such that

$$V_{i'} = \max_{1 \le i \le n} V_i$$

The DP recurrence permits constructing V_i in O(1) time from V_{i-1}.

$\Rightarrow$ We can construct $V_1, V_2, \ldots, V_n$ in order in O(n) total time while keeping track of the largest V_i found so far

$\Rightarrow$ This finds $V_{i'}$ in O(n) total time, solving the problem.

Note: This algorithm is very similar to the linear scan algorithm we developed in class, but found using DP reasoning

Implementation

Derived recurrence that

$$V_i = \begin{cases} A[1] & \text{if } i = 1 \\ \max\{A[i], A[i] + V_{i-1}\} & \text{if } i > 1 \end{cases}$$

where

$$V_i = \max_{1 \le k \le i} V(k, i)$$

and need to find i' such that

$$V_{i'} = \max_{1 \le i \le n} V_i$$

This is very straightforward.
Next slides give actual code, and a worked example

Version 1

Store V_i in a table $V[1, 2, \ldots, n]$, at each step calculating $V[i]$ from $V[i-1]$

Base condition: $V[1] \leftarrow A[1]$ Recurrence: $V[i] \leftarrow \max(A[i], A[i] + V[i-1])$

```
let V[1,2,...,n] be an array storing V_i
V[1] ← A[1]
V_max ← A[1]
for i ← 2 to n do
        V[i] ← max(A[i], A[i] + V[i − 1])
        if V_max < V[i]
                then V_max ← V[i]
        end if
return V_max
```

Running time:
$\Theta(n)$

i	1	2	3	4	5	6	7	8	9
$A[i]$	3	2	1	-7	5	2	-1	3	-1
$V[i]$	3	5	6	-1	5	7	6	9	8
V_{max}	3	5	6	6	6	7	7	9	9

Solution is $V[8]$

Version 2

Simplified: We only need to remember the last V_i (call it V) and V_{max}

Base condition: $V \leftarrow A[1]$

Recurrence: $V \leftarrow \max(A[i], A[i] + V)$

$V \leftarrow A[1]$
$V_{max} \leftarrow A[1]$
for $i \leftarrow 2$ **to** n **do**
  $V \leftarrow \max(A[i], A[i] + V)$
  if $V_{max} < V$
    then $V_{max} \leftarrow V$
  end if
return V_{max}

Running time:
$\Theta(n)$

This gets same result as Version 1, but is simpler!

Next pages provide a detailed walk-through of how Version 1 fills in the DP table.

Version 1

Store V_i in a table $V[1, 2, \ldots, n]$, at each step calculating $V[i]$ from $V[i-1]$

Base condition: $V[1] \leftarrow A[1]$ Recurrence: $V[i] \leftarrow \max(A[i], A[i] + V[i-1])$

```
let V[1,2,...,n] be an array storing V_i
V[1] ← A[1]
V_max ← A[1]
for i ← 2 to n do
        V[i] ← max(A[i], A[i] + V[i − 1])
        if V_max < V[i]
                then V_max ← V[i]
        end if
return V_max
```

Running time:
$\Theta(n)$

i	1	2	3	4	5	6	7	8	9
$A[i]$	3	2	1	-7	5	2	-1	3	-1
$V[i]$	3								
V_{max}	3								

$V_{max} = V[1] = A[1] = 3$

Version 1

Store V_i in a table $V[1, 2, \ldots, n]$, at each step calculating $V[i]$ from $V[i-1]$

Base condition: $V[1] \leftarrow A[1]$ Recurrence: $V[i] \leftarrow \max(A[i], A[i] + V[i-1])$

```
let V[1,2,...,n] be an array storing V_i
V[1] ← A[1]
V_max ← A[1]
for i ← 2 to n do
    V[i] ← max(A[i], A[i] + V[i − 1])
    if V_max < V[i]
        then V_max ← V[i]
    end if
return V_max
```

Running time:
$\Theta(n)$

i	1	2	3	4	5	6	7	8	9
$A[i]$	3	2	1	-7	5	2	-1	3	-1
$V[i]$	3	5							
V_{max}	3	5							

$V_{max} = \max(A[2], A[2] + V[1]) = \max(2, 2+3) = 5$

Version 1

Store V_i in a table $V[1, 2, \dots, n]$, at each step calculating $V[i]$ from $V[i-1]$

Base condition: $V[1] \leftarrow A[1]$ Recurrence: $V[i] \leftarrow \max(A[i], A[i] + V[i-1])$

```
let V[1,2,...,n] be an array storing V_i
V[1] ← A[1]
V_max ← A[1]
for i ← 2 to n do
        V[i] ← max(A[i], A[i] + V[i − 1])
        if V_max < V[i]
                then V_max ← V[i]
        end if
return V_max
```

Running time: $\Theta(n)$

i	1	2	3	4	5	6	7	8	9
$A[i]$	3	2	1	-7	5	2	-1	3	-1
$V[i]$	3	5	6						
V_{max}	3	5	6						

$V_{max} = \max(A[3], A[3] + V[2]) = \max(1, 1 + 5) = 6$

Version 1

Store V_i in a table $V[1, 2, \ldots, n]$, at each step calculating $V[i]$ from $V[i-1]$

Base condition: $V[1] \leftarrow A[1]$ Recurrence: $V[i] \leftarrow \max(A[i], A[i] + V[i-1])$

```
let V[1,2,...,n] be an array storing V_i
V[1] ← A[1]
V_max ← A[1]
for i ← 2 to n do
        V[i] ← max(A[i], A[i] + V[i − 1])
        if V_max < V[i]
                then V_max ← V[i]
        end if
return V_max
```

Running time:
$\Theta(n)$

i	1	2	3	4	5	6	7	8	9
$A[i]$	3	2	1	-7	5	2	-1	3	-1
$V[i]$	3	5	6	-1					
V_{max}	3	5	6	6					

$V_{max} = 6 > \max(A[4], A[4] + V[3]) = \max(-7, -7 + 6) = -1$

Version 1

Store V_i in a table $V[1,2,\dots,n]$, at each step calculating $V[i]$ from $V[i-1]$

Base condition: $V[1] \leftarrow A[1]$ Recurrence: $V[i] \leftarrow \max(A[i], A[i] + V[i-1])$

```
let V[1,2,...,n] be an array storing V_i
V[1] ← A[1]
V_max ← A[1]
for i ← 2 to n do
        V[i] ← max(A[i], A[i] + V[i − 1])
        if V_max < V[i]
                then V_max ← V[i]
        end if
return V_max
```

Running time:
$\Theta(n)$

i	1	2	3	4	5	6	7	8	9
$A[i]$	3	2	1	-7	5	2	-1	3	-1
$V[i]$	3	5	6	-1	5				
V_{max}	3	5	6	6	6				

$V_{max} = 6 > \max(A[5], A[5] + V[4]) = \max(5, 5-1) = 5$

Version 1

Store V_i in a table $V[1, 2, \ldots, n]$, at each step calculating $V[i]$ from $V[i-1]$

Base condition: $V[1] \leftarrow A[1]$ Recurrence: $V[i] \leftarrow \max(A[i], A[i] + V[i-1])$

```
let V[1,2,...,n] be an array storing V_i
V[1] ← A[1]
V_max ← A[1]
for i ← 2 to n do
        V[i] ← max(A[i], A[i] + V[i − 1])
        if V_max < V[i]
                then V_max ← V[i]
        end if
return V_max
```

Running time:
$\Theta(n)$

i	1	2	3	4	5	6	7	8	9
$A[i]$	3	2	1	-7	5	2	-1	3	-1
$V[i]$	3	5	6	-1	5	7			
V_{max}	3	5	6	6	6	7			

$V_{max} = \max(A[6], A[6] + V[5]) = \max(2, 2+5) = 7$

Version 1

Store V_i in a table $V[1,2,\dots,n]$, at each step calculating $V[i]$ from $V[i-1]$

Base condition: $V[1] \leftarrow A[1]$ Recurrence: $V[i] \leftarrow \max(A[i], A[i] + V[i-1])$

```
let V[1,2,...,n] be an array storing V_i
V[1] ← A[1]
V_max ← A[1]
for i ← 2 to n do
        V[i] ← max(A[i], A[i] + V[i − 1])
        if V_max < V[i]
                then V_max ← V[i]
        end if
return V_max
```

Running time:
$\Theta(n)$

i	1	2	3	4	5	6	7	8	9
$A[i]$	3	2	1	-7	5	2	-1	3	-1
$V[i]$	3	5	6	-1	5	7	6		
V_{max}	3	5	6	6	6	7	7		

$V_{max} = 7 > \max(A[7], A[7] + V[6]) = \max(-1, -1+7) = 6$

Version 1

Store V_i in a table $V[1, 2, \ldots, n]$, at each step calculating $V[i]$ from $V[i-1]$

Base condition: $V[1] \leftarrow A[1]$ Recurrence: $V[i] \leftarrow \max(A[i], A[i] + V[i-1])$

```
let V[1,2,...,n] be an array storing V_i
V[1] ← A[1]
V_max ← A[1]
for i ← 2 to n do
        V[i] ← max(A[i], A[i] + V[i−1])
        if V_max < V[i]
                then V_max ← V[i]
        end if
return V_max
```

Running time:
$\Theta(n)$

i	1	2	3	4	5	6	7	8	9
$A[i]$	3	2	1	-7	5	2	-1	3	-1
$V[i]$	3	5	6	-1	5	7	6	9	
V_{max}	3	5	6	6	6	7	7	9	

$V_{max} = \max(A[8], A[8] + V[7]) = \max(3, 3+6) = 9$

Version 1

Store V_i in a table $V[1, 2, \ldots, n]$, at each step calculating $V[i]$ from $V[i-1]$

Base condition: $V[1] \leftarrow A[1]$ Recurrence: $V[i] \leftarrow \max(A[i], A[i] + V[i-1])$

```
let V[1,2,…,n] be an array storing V_i
V[1] ← A[1]
V_max ← A[1]
for i ← 2 to n do
        V[i] ← max(A[i], A[i] + V[i − 1])
        if V_max < V[i]
                then V_max ← V[i]
        end if
return V_max
```

Running time:
$\Theta(n)$

i	1	2	3	4	5	6	7	8	9
$A[i]$	3	2	1	-7	5	2	-1	3	-1
$V[i]$	3	5	6	-1	5	7	6	9	8
V_{max}	3	5	6	6	6	7	7	9	9

$V_{max} = 9 > \max(A[9], A[9] + V[8]) = \max(-1, -1 + 9) = 8$

Version 1

Store V_i in a table $V[1, 2, \ldots, n]$, at each step calculating $V[i]$ from $V[i-1]$

Base condition: $V[1] \leftarrow A[1]$ Recurrence: $V[i] \leftarrow \max(A[i], A[i] + V[i-1])$

```
let V[1,2,...,n] be an array storing V_i
V[1] ← A[1]
V_max ← A[1]
for i ← 2 to n do
      V[i] ← max(A[i], A[i] + V[i − 1])
      if V_max < V[i]
            then V_max ← V[i]
      end if
return V_max
```

Running time: $\Theta(n)$

i	1	2	3	4	5	6	7	8	9
$A[i]$	3	2	1	-7	5	2	-1	3	-1
$V[i]$	3	5	6	-1	5	7	6	**9**	8
V_{max}	3	5	6	6	6	7	7	9	9

Solution is $V[8]$

$V_{max} = 9 > \max(A[9], A[9] + V[8]) = \max(-1, -1 + 9) = 8$