Dynamic Programming Activities

Practice

by Marina Barsky

1. From idea to pseudocode

Subset sum

Subset sum problem

given a set S of integers is there a subset which sums up to k?

Sample instance: S = {None,3, 2, 1, 4, 1, 5}, k = 8

	Total →	0	1	2	3	4	5	6	7	8
0	-	T	F	F	F	F	F	F	F	F
1	**3**	T	F	F	T	F	F	F	F	F
2	**2**	T	F	T	T	F	T	F	F	F
3	**1**	T	T	T	T	T	T	T	F	F
4	**4**	T	T	T	T	T	T	T	T	T
5	**1**	T								
6	**5**	T								

Note that we also need row 0 as a base case

3 + 1 + 4 = 2 + 1 + 5

Recurrence relation

- Give a recurrence relation to compute subset sum:

Let $T(i,j)$ be the answer to the following question:

Is it possible to obtain sum j using only first i integers: $\{1,\ldots,i\}$?

Recurrence relation: solution

Base case:

T(i,j) = True if j = 0

T(i,j) = False if i=0, j>0

Recurrence:

T(i,j) = True if T(i-1,j) is True or T(i – 1, j – S[i]) is True

Pseudocode

$T(i,j)$ = True if $j = 0$, $T(i,j)$ = False if $i=0$, $j>0$

$T(i,j)$ = True if $T(i-1,j)$ is True or $T(i-1, j-S[i])$ is True

Pseudocode: solution

T(i,j) = True if j = 0, T(i,j) = False if i=0, j>0

T(i,j) = True if T(i-1,j) is True or T(i – 1, j – S[i]) is True

Algorithm subset_sum(array S of size n, integer k)

```
create Table [(n+1)x(k+1)]        Zero-based 2D array
for i from 0 to n:
   Table[i][0] : = True
for j from 1 to n:
   Table[0][j]: = False
for i from 1 to n:
   for j from 1 to k:
      Table[i][j]: = Table[i – 1][j]        # looking at cell above - by default
      if not Table[i][j]:                   # trying to fit item i with value S[i]
         Table[i][j]: = Table[i-1][j-S[i]]
   if Table[i][k] :
      return True
return False
```

Also think how you would recover items that sum up to k

Exercise 2. Improving recursive solution with memorization and DP (simple)

From recurrence relation to algorithm

Given the following recurrence relation:

T(0) = 1, T(1) = 2
T(n) = T(n-1)*T(n-2), for n > 1

Convert this relation into a recursive algorithm for computing T given n.

Recursive Solution

Given the following recurrence relation:

$T(0) = 1, T(1) = 2$
$T(n) = T(n-1)*T(n-2)$, for $n > 1$

Convert this relation into a recursive algorithm for computing T given n.

Algorithm T(n)
if n=0: return 1 if n=1: return 2 return T(n -1)*T(n-2)

Running time of the recursive algorithm?

Given the following recurrence relation:

$T(0) = 1, T(1) = 2$
$T(n) = T(n-1)*T(n-2)$, for $n > 1$

Convert this relation into a recursive algorithm for computing T given n.

Algorithm T(n)

```
if n=0: return 1
if n=1: return 2
return T(n -1)*T(n-2)
```

What is the running time of this algorithm?

Running time solution

Given the following recurrence relation:

T(0) = 1, T(1) = 2
T(n) = T(n-1)*T(n-2), for n > 1

Convert this relation into a recursive algorithm for computing T given n.

Algorithm T(n)
if n=0: return 1 if n=1: return 2 return T(n -1)*T(n-2)

Running time $O(2^n)$

Memoization/DP

Algorithm T(n)

```
if n=0: return 1
if n=1: return 2
return T(n -1)*T(n-2)
```

Can we avoid repeating computations applying memorization/DP?

Memoization: solution

Algorithm T(n, A)

```
if n=0 or n=1: return A[n]

if A[n-1] is Null:
    A[n-1]: = T(n -1, A)
if A[n-2] is Null:
    A[n-2]: = T(n -2, A)

return A[n-1] * A[n-2]
```

Algorithm T_memoization(n)

```
create array A of size n+1
                    filled with Nulls

A[0]: = 1
A[1]: = 2
return T(n, A)
```

zero-based

Dynamic Programming: solution

Algorithm T_DP(n)

```
create array A of size n+1 filled with Nulls
A[0]: = 1
A[1]: = 2
for i from 2 to n:
   A[i]: = A[i-1]*A[i-2]
return A[n]
```

Exercise 3. Improving recursive solution with DP (more complex)

From recurrence relation to pseudocode

Given the following recurrence relation:

T(0) = T(1) = 2

T(n) = $\sum_{i=1}^{n-1}(2 \times T(i) \times T(i-1))$ for n > 1

Convert this relation into a recursive algorithm for computing T given n.

Recursive solution

Given the following recurrence relation:

T(0) = T(1) = 2

T(n) = $\sum_{i=1}^{n-1}(2 \times T(i) \times T(i-1))$ for n > 1

Algorithm T(n)

```
if n=0 or n=1: return 2
sum: = 0
for i from 1 to n – 1:
  sum: += 2*T(i)*T(i-1)
return sum
```

Improve recursion with DP

Given the following recurrence relation:

T(0) = T(1) = 2

T(n) = $\sum_{i=1}^{n-1}(2 \times T(i) \times T(i-1))$ for n > 1

Algorithm T(n)

```
if n=0 or n=1: return 2
sum: = 0
for i from 1 to n – 1:
  sum: += 2*T(i)*T(i-1)
return sum
```

Improving recursion with DP: solution

T(0) = T(1) = 2

T(n) = $\sum_{i=1}^{n-1}(2 \times T(i) \times T(i-1))$

To see the solution – run through examples:

T(0) = T(1) = 2

T(2) = 2*T(1)*T(0)

T(3) = 2*T(1)*T(0) + 2*T(2)*T(1)

T(4) = 2*T(1)*T(0) + 2*T(2)*T(1) + 2*T(3)*T(2)

Improving recursion with DP: solution

To see the solution – run through examples:

T(0) = T(1) = 2

T(2) = 2*T(1)*T(0)

T(3) = 2*T(1)*T(0) + 2*T(2)*T(1)

T(4) = 2*T(1)*T(0) + 2*T(2)*T(1) + 2*T(3)*T(2)

Algorithm T_DP(n)

```
create array A of size n+1
A[0]: = A[1]: =2
for i from 2 to n:
    A[i]: = 0
    for j from 1 to i-1:
       A[i]: += 2*A[i]*A[i-1]
return A[n]
```

Improving recursion with DP: solution

To see the solution – run through examples:

T(0) = T(1) = 2

T(2) = 2*T(1)*T(0)

T(3) = 2*T(1)*T(0) + 2*T(2)*T(1)

T(4) = 2*T(1)*T(0) + 2*T(2)*T(1) + 2*T(3)*T(2)

Algorithm T_DP(n)

```
create array A of size n+1
A[0]: = A[1]: =2
for i from 2 to n:
    A[i]: = 0
    for j from 1 to i-1:
       A[i]: += 2*A[i]*A[i-1]
return A[n]
```

Complexity: $O(n^2)$

Can we do better?

Improving recursion with DP: solution

To see the solution – run through examples:

T(0) = T(1) = 2

T(2) = 2*T(1)*T(0)

T(3) = 2*T(1)*T(0) + 2*T(2)*T(1)

$T(4) = \underbrace{2*T(1)*T(0) + 2*T(2)*T(1)}_{T(3)} + 2*T(3)*T(2)$

Algorithm T_DP_fast(n)

```
create array A of size n+1
A[0]: = A[1]: =2
A[2]: = 2*A[0]*A[1]
for i from 3 to n:
    A[i]: = A[i-1] + 2*A[i-1]*A[i-2]
return A[n]
```

Complexity: O(n)