Dynamic Programming for Boolean decisions

Lecture 07.03

by Marina Barsky

Partitioning souvenirs

Subset sum

Partitioning souvenirs

- You and your friend have just returned back home after visiting various countries
- During your trip you were buying different souvenirs of various prices, and put them together into a single pile
- Now you would like to evenly split all the souvenirs into two subsets of equal sum

Souvenir Partitioning Problem

Input: The prices of n items $d_1, d_2, \ldots, d_n$ in dollars (integers).

Output: *Yes*, if the items can be divided into two groups, such that each group has the same total cost.
No, otherwise.

Sample problem instance

	1	2	3	4	5	6	7
$$	3	2	3	2	2	5	3

A total of 7 items given with their costs

The output for this instance of the problem is 'yes'. This is because the items can be divided into two groups that both have a total cost of $10

3	2	3	2	2	5	3
1	2	3	4	5	6	7

The blue and red items have the same total cost of $10

DP solution: brainstorming

- What would help us to know if a set of numbers can be divided into 2 subsets with equal sums?
- How can we find out if there is a subset with a given sum?
- What are optimal subproblems?

Think of an optimal solution to a subset sum

- If there is a subset with total cost D, and it contains item i, then there also should be a subset with cost D-d_i

- As always, we can start by checking if all possible costs from 1 to D can be obtained from a current set, and we will reuse this knowledge to obtain an answer for cost D

Example

3	2	1	4	1	5
1	2	3	4	5	6

- First, we compute total cost:

  3+2+1+4+1+5 = 16

- The task becomes to find out if there is a subset that sums up to 16/2 = 8

- We will try methodically to fit each item into the solution, checking if the following total costs are possible: 0,1,2,3,4,5,6,7 and finally 8.

- This check will produce a Boolean value: Y(True) or N(False)

Create DP table

3	2	1	4	1	5
1	2	3	4	5	6

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)									
d_2(**2**)									
d_3(**1**)									
d_4(**4**)									
d_5(**1**)									
d_6(**5**)									

Base condition

Is it possible to create a subset with a total cost **0**?

Yes, just do not take any items.

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T								
d_2(**2**)	T								
d_3(**1**)	T								
d_4(**4**)	T								
d_5(**1**)	T								
d_6(**5**)	T								

What costs are possible with item 1?

Using only the first item with cost 3, is it possible to create a subset with total costs 1,2,3,4…?

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F							
d_2(**2**)	T								
d_3(**1**)	T								
d_4(**4**)	T								
d_5(**1**)	T								
d_6(**5**)	T								

What costs are possible with item 1?

Using only the first item with cost 3, is it possible to create a subset with total costs 1,2,3,4…?

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F						
d_2(**2**)	T								
d_3(**1**)	T								
d_4(**4**)	T								
d_5(**1**)	T								
d_6(**5**)	T								

What costs are possible with item 1?

Using only the first item with cost 3, is it possible to create a subset with total costs 1,2,3,4…?

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T					
d_2(**2**)	T								
d_3(**1**)	T								
d_4(**4**)	T								
d_5(**1**)	T								
d_6(**5**)	T								

What costs are possible with item 1?

Using only the first item with cost 3, is it possible to create a subset with total costs 1,2,3,4…?

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T								
d_3(**1**)	T								
d_4(**4**)	T								
d_5(**1**)	T								
d_6(**5**)	T								

What costs are possible with items 1 and 2?

Using only item 1 (cost 3) and/or item 2(cost 2), it is still not possible to create a subset with total cost 1.

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F							
d_3(**1**)	T								
d_4(**4**)	T								
d_5(**1**)	T								
d_6(**5**)	T								

What costs are possible with items 1 and 2?

Using only item 1 (cost 3) and item 2(cost 2), it is not possible to create a subset with total cost 1, but it is possible to create a subset with total cost 2

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T						
d_3(**1**)	T								
d_4(**4**)	T								
d_5(**1**)	T								
d_6(**5**)	T								

What costs are possible with items 1 and 2?

Using only d1 and d2, can we have a subset with total cost 3?

Yes, we already know that we can do it even without d2

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T					
d_3(**1**)	T								
d_4(**4**)	T								
d_5(**1**)	T								
d_6(**5**)	T								

What costs are possible with items 1 and 2?

How do we check if a subset sum 4 is possible? We know that it was False when we used d1 only, so if we use d2, then we need to check if a subset of (4-2) was possible. It was not.

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T	F				
d_3(**1**)	T								
d_4(**4**)	T								
d_5(**1**)	T								
d_6(**5**)	T								

What costs are possible with items 1 and 2?

To check for d=5, take item d2(2) and see if cost 5-2 was possible with the previous item(s)

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T	F	T	F	F	F
d_3(**1**)	T								
d_4(**4**)	T								
d_5(**1**)	T								
d_6(**5**)	T								

Considering items 1,2, and 3

1,2,3 are possible.

What about 4? Current item d3 has cost 1. Is it possible to have a cost (4-1) with the other 2 items? Yes

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T	F	T	F	F	F
d_3(**1**)	T	T	T	T	T				
d_4(**4**)	T								
d_5(**1**)	T								
d_6(**5**)	T								

Considering items 1,2, and 3

Using only items d1, d2, d3 we get the following boolean answers.

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T	F	T	F	F	F
d_3(**1**)	T	T	T	T	T	T	T	F	F
d_4(**4**)	T								
d_5(**1**)	T								
d_6(**5**)	T								

Considering items 1,2,3,4

For d4(4) we do not even need to consider this item for costs 1,2,3,4,5,6 - we could make these subsets even without item d4(4).

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T	F	T	F	F	F
d_3(**1**)	T	T	T	T	T	T	T	F	F
d_4(**4**)	T	T	T	T	T	T	T		
d_5(**1**)	T								
d_6(**5**)	T								

Considering items 1,2,3,4

What about 7? Fit d4(4) and see if (7-4) was True.

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T	F	T	F	F	F
d_3(**1**)	T	T	T	T	T	T	T	F	F
d_4(**4**)	T	T	T	T	T	T	T	T	
d_5(**1**)	T								
d_6(**5**)	T								

Considering items 1,2,3,4

Same holds for total cost 8.

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T	F	T	F	F	F
d_3(**1**)	T	T	T	T	T	T	T	F	F
d_4(**4**)	T	T	T	T	T	T	T	T	T
d_5(**1**)	T								
d_6(**5**)	T								

Is total cost 8 possible?

At this point we can stop. We know that it is possible to form a subset with a total cost 8 using only the first 4 items.

But what is this subset?

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T	F	T	F	F	F
d_3(**1**)	T	T	T	T	T	T	T	F	F
d_4(**4**)	T	T	T	T	T	T	T	T	T
d_5(**1**)	T								
d_6(**5**)	T								

Recovering the subset with sum 8: trace back

The subset clearly includes item d4 - without it 8 was not possible

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T	F	T	F	F	F
d_3(**1**)	T	T	T	T	T	T	T	F	F
d_4(**4**)	T	T	T	T	T	T	T	T	T
d_5(**1**)	T								
d_6(**5**)	T								

Recovering the subset with sum 8: trace back

If it includes item of cost 4, we need to look at total cost (8-4). This one only became True when we added item d3.

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T	F	T	F	F	F
d_3(**1**)	T	T	T	T	T	T	T	F	F
d_4(**4**)	T	T	T	T	T	T	T	T	T
d_5(**1**)	T								
d_6(**5**)	T								

Recovering the subset with sum 8: trace back

If the solution includes item d3(1), we need to look at total cost (4-1). This one is True because a previous item produced True. This item was d1(3)

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T	F	T	F	F	F
d_3(**1**)	T	T	T	T	T	T	T	F	F
d_4(**4**)	T	T	T	T	T	T	T	T	T
d_5(**1**)	T								
d_6(**5**)	T								

Answer: Yes, it is possible to create 2 subsets with equal total cost

d1(3) + d3(1) + d4(4) = d2(2) + d5(1) + d6(5)

3 + 1 + 4 = 2 + 1 + 5

Total cost→	0	1	2	3	4	5	6	7	8
d_1(**3**)	T	F	F	T	F	F	F	F	F
d_2(**2**)	T	F	T	T	F	T	F	F	F
d_3(**1**)	T	T	T	T	T	T	T	F	F
d_4(**4**)	T	T	T	T	T	T	T	T	T
d_5(**1**)	T								
d_6(**5**)	T								

Game of Rocks

Optimal game strategy

Game: 1-2 rocks

- 2 players
- 2 piles of rocks:

  with *n* and *m* rocks respectively

- Each turn, one player may take either 1 rock (from either pile) or 2 rocks (one from each pile)
- Once the rocks are taken, they are removed from play
- The player that takes the last rock wins

Winning strategy with DP

- To find the winning strategy for the m + n game, we first construct an mxn table *R*.
- If Player 1 can always win the *n + m* game, then we would say *R*(*n, m*) = *W* , but if Player 1 has no winning strategy against a player that always makes the right moves, we would write *R*(*n, m*) = *L*.
- Computing *R*(*n, m*) for arbitrary *n* and *m* seems difficult, but we can build on smaller values.

DP table for game outcomes

- Construct an mxn table *R*.
- Example: let m=n=10.

	0	1	2	3	4	5	6	7	8	9	10
0											
1											
2											
3											
4											
5											
6											
7											
8											
9											
10											

Simple subproblems first

- Notably R(0, 1), R(1, 0), and R(1, 1), are clearly winning propositions for Player 1 since with a single move Player 1 can win.

- Thus, we fill in entries (1, 1) , (0, 1), and (1, 0) as W.

	0	1	2	3	4	5	6	7	8	9	10
0		W									
1	W	W									
2											
3											
4											
5											
6											
7											
8											
9											
10											

Solve larger subproblems based on solutions to the smaller problems

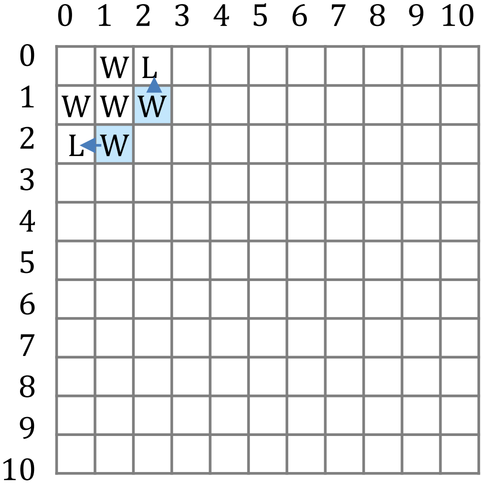
- In the (2, 0) case, the only move that Player 1 can make leads to the (1, 0) case that, as we already know, is a winning position for his opponent.

- A similar analysis applies to the (0, 2) case.

	0	1	2	3	4	5	6	7	8	9	10
0		W	L								
1	W	W									
2	L										
3											
4											
5											
6											
7											
8											
9											
10											

Solve larger subproblems based on solutions to the smaller problems

- In the (2, 1) case, Player 1 can make 3 different moves that lead respectively to the games of (1, 1), (2, 0), or (1, 0).
- One of these cases, (2, 0), leads to a losing position for his opponent and therefore (2, 1) is a winning position.
- The case (1, 2) is symmetric to (2, 1)



Solve larger subproblems based on solutions to the smaller problems

- In the (2, 2) case, Player 1 can make three different moves that lead to entries (2, 1), (1, 2), and (1, 1).

- All of these entries are winning positions for his opponent and therefore R(2, 2) = L.

	0	1	2	3	4	5	6	7	8	9	10
0		W	L								
1	W	W	W								
2	L	W	L								
3											
4											
5											
6											
7											
8											
9											
10											

Fill DP table with game outcomes

- We can proceed filling in *R* in this way by noticing that for the entry (i, j) to be *L*, all the entries above, diagonally to the left, and directly to the left, must be *W*.

- These entries:

((i −1, j), (i −1, j −1), (i, j −1)) correspond to the three possible moves that Player 1 can make.

	0	1	2	3	4	5	6	7	8	9	10
0		W	L	W	L	W	L	W	L	W	L
1	W	W	W	W	W	W	W	W	W	W	W
2	L	W	L	W	L	W	L	W	L	W	L
3	W	W	W	W	W	W	W	W	W	W	W
4	L	W	L	W	L	W	L	W	L	W	L
5	W	W	W	W	W	W	W	W	W	W	W
6	L	W	L	W	L	W	L	W	L	W	L
7	W	W	W	W	W	W	W	W	W	W	W
8	L	W	L	W	L	W	L	W	L	W	L
9	W	W	W	W	W	W	W	W	W	W	W
10	L	W	L	W	L	W	L	W	L	W	L

Rocks: winning strategy

- The *Rocks* algorithm determines if Player 1 wins or loses.

- If Player 1 wins in an *n+m* game, *Rocks* returns W. If Player 1 loses, *Rocks* returns L.

- We introduce an artificial initial condition, R(0, 0) = L to simplify the pseudocode.

Algorithm *Rocks*(*n*, *m*)

```
R[0, 0] ← L
for i from 1 to n:                    # initialize rows
   if R[i − 1, 0] = W :
      R[i, 0] ← L
   else:
      R[i, 0] ← W
for j from 1 to m:                    # initialize columns
   if R[0, j − 1] = W :
      R[0, j] ← L
   else:
      R[0, j] ← W
for i from 1 to n:
   for j from 1 to m:                 # fill DP table
      if R[i − 1, j − 1] = W and R[i, j − 1] = W and R[i − 1, j] = W :
         R[i, j] ← L
      else:
         R[i, j] ← W
return R[n, m]
```

Using DP table for best strategy or game AI

- We can use the DP table to always play the winning strategy.
- If R(n,m) = W, and Player 1 starts first, he can always win: by taking the number of rocks which lead to the losing position of our opponent.
- If R(n,m) = L, then Player 1 can only hope that Player 2 does not use the same table, and makes a mistake.

	0	1	2	3	4	5
0		W	L	W	L	W
1	W	W	W	W	W	W
2	L	W	L	W	L	W
3	W	W	W	W	W	W
4	L	W	L	W	L	W
5	W	W	W	W	W	W

Winning strategy: example

	0	1	2	3	4	5
0		W	L	W	L	W
1	W	W	W	W	W	W
2	L	W	L	W	L	W
3	W	W	W	W	W	W
4	L	W	L	W	L	W
5	W	W	W	W	W	W

Player 1 takes (1,1).

Winning strategy example

	0	1	2	3	4	5
0		W	L	W	L	
1	W	W	W	W	W	
2	L	W	L	W	L	
3	W	W	W	W	W	
4	L	W	L	W	L	
5						

Player 1 takes (1,1).

No matter what Player 2 does, it leads to the winning state of Player 1.

Say, Player 2 takes (1,0)

Winning strategy example

	0	1	2	3	4	5
0		W	L	W	L	
1	W	W	W	W	W	
2	L	W	L	W	L	
3	W	W	W	W	W	
4						
5						

Player 2 takes (1,0)

Winning strategy example

	0	1	2	3	4	5
0		W	L	W	L	
1	W	W	W	W	W	
2	L	W	L	W	L	
3	W	W	W	W	W	
4						
5						

Player 1 should take (1,0).

Winning strategy example

	0	1	2	3	4	5
0		W	L	W	L	
1	W	W	W	W	W	
2	L	W	L	W	L	
3						
4						
5						

Player 1 takes (1,0).

Winning strategy example

	0	1	2	3	4	5
0		W	L	W	L	
1	W	W	W	W	W	
2	L	W	L	W	L	
3						
4						
5						

Player 2 takes (1,1).

Winning strategy example

	0	1	2	3	4	5
0		W	L	W		
1	W	W	W	W		
2						
3						
4						
5						

Player 2 takes (1,1).

Winning strategy example

	0	1	2	3	4	5
0		W	L	W		
1	W	W	W	W		
2						
3						
4						
5						

Player 1 should take (1,1).

Winning strategy example

	0	1	2	3	4	5
0		W	L			
1						
2						
3						
4						
5						

Player 1 takes (1,1).

At this point the victory for Player 1 is guaranteed.

Identifying patterns

- A faster algorithm relies on the simple pattern in R, and checks if *n* and *m* are both even, in which case the player 1 loses.

- However, though *FastRocks* is more efficient than *Rocks*, it may be difficult to modify it for similar games.

	0	1	2	3	4	5	6	7	8	9	10
0		W	L	W	L	W	L	W	L	W	L
1	W	W	W	W	W	W	W	W	W	W	W
2	L	W	L	W	L	W	L	W	L	W	L
3	W	W	W	W	W	W	W	W	W	W	W
4	L	W	L	W	L	W	L	W	L	W	L
5	W	W	W	W	W	W	W	W	W	W	W
6	L	W	L	W	L	W	L	W	L	W	L
7	W	W	W	W	W	W	W	W	W	W	W
8	L	W	L	W	L	W	L	W	L	W	L
9	W	W	W	W	W	W	W	W	W	W	W
10	L	W	L	W	L	W	L	W	L	W	L