6.006	Lecture 20	Nov. 27, 2018

TODAY: Dynamic Programming V
- review of past DPs
- pro tips for DP
  - subproblem choices
  - recurrences via guessing
  - adding subproblems when stuck
  - accounting running time
- Subset Sum & Coin Changing DPs
- pseudopolynomial time

Recall: 5 steps to DP/recursive algorithm:
1. Subproblems definition
2. Recurrence relating subproblems acyclicly
3. Base cases for recurrence
4. Solve original problem using subproblem(s)
   (and recursion + memoization OR bottom-up loops)
5. Time analysis

"SR. BST"

Generic example: (DP for all problems!)

① $x(a,b)$

② $x(a,b) = R(a,b, x(a',b') \text{ for } (a',b') < (a,b))$ — some function; some topo. order

③ $x(a,b) = B(a,b)$ for (a,b) where $R(a,b)$ fails

④

TOP DOWN:

```
memo = {}
def x(a,b):
    if (a,b) not in memo:
        if base case:
            memo[a,b] = B(a,b)
        else:
            memo[a,b] = R(a,b,x)   # calls
    return memo[a,b]
S(x)
```

OR

BOTTOM UP:

```
x = {}
for (a,b) in
      topo. order:
    if base case:
        x[a,b] = B(a,b)
    else:
        x[a,b] = R(a,b,x)   # looks up
S(x)
```

⑤ $\text{time} = \sum_{(a,b)} \{$ time for $B(a,b)$ if base case
time for $R(a,b,x)$ $\}$

EXCLUDING TIME TO RECURSE
(treating calls to x as $O(1)$ time!)

$\leq$ # choices for (a,b)
$\cdot \max \{$ time for $g(a,b)$,
time for $h(a,b,x) \mid (a,b) \}$

Pro tips for DP:

① Subproblem choices:

input < sequence < prefixes/suffixes / substrings
input < integer – smaller vals.

- if input is a sequence S (or string/array)
  - try prefixes $S[:i]$ or suffixes $S[i:]$ (mostly personal preference which)
  - if you seem to need both, probably need:
    - try substrings $S[i:j]$ (more subprobs. ⇒ slower but can then split in middle)
- if input is multiple sequences
  - try subproblems defined by prefixes/suffixes/substrings of each
- if input includes an integer
  - try smaller values of that integer

alt. coin game, arith. parens.
edit dist.
Fib, rod cut, TODAY

①' – if fail to come up with recurrence:

- seem to need multiple answers to one subproblem (not just one "best")
- subproblem doesn't capture all you need to know
- relation has cycles

arith. parens, L.I.S., TODAY
SSSP

then you probably need to add more subproblems

- similar to "graph duplication" trick
  → need to know vertex AND some extra information

② Recurrence via "guessing":

- try to identify a feature of the subproblem's solution that, if you knew, lets you reduce to one or more "smaller" subproblems (recursion)
  - for prefix $S[:i]$, feature is typically "what to do with/at $S[i-1]$", then reduce to / recurse on $S[:i-1]$
  - for suffix $S[i:]$, feature is typically "what to do with/at $S[i]$", then reduce to / recurse on $S[i+1:]$
  - for substring $S[i:j]$, recursion is typically on $S[i:k(-1)]$ & $S[k(+1):j]$ and feature includes k $(i \le k \le j)$
- try all options for this feature ← via for loop & take the best resulting solution
  - best: typically min/max / OR/AND ↑ TODAY
  - resulting solution: ↳ combining recursive solutions with this choice of feature

Alternate view: "lucky algorithms" [Teela Brown]

- guess the feature of the solution
  - imagine algorithm makes lucky (correct) guesses, always
- then recurse & combine into solution

↳ simulate this with real algorithm via for loop over choices & taking best result

Past DPs:	① subprobs.	② guess/choice
[L16] Fibonacci	smaller n	nothing
[L16] DAG SSSP	vertex v	last edge on $s \to v$ path
[L17] SSSP (B-F)	+ # edges	last edge on $s \to v$ path
[L17] Rod cutting	smaller rod	last/first piece
[R16] Text justification	suffixes	first linebreak
[R16] L.I.S.	suffixes + use A[i]	next A[j] used
[R17] L.C.S.	A prefix, B prefix	drop from A or B?
[L18] Edit distance	A prefix, B prefix	rightmost edit
[R17] Alt. coin game	substrings	coin to take
[L18/19] Arith. parens.	substrings + min/max	root operation
[L19] Egg drop	smaller floors/eggs	first drop height
[L20] Subset sum	{ suffixes + smaller t	include A[i]?
[L20] Change making	{ suffixes + smaller t	# uses of A[i]

Subset sum: given n positive integers $A = \{a_0, a_1, \ldots, a_{n-1}\}$ and target sum t

is there a subset $S \subseteq A$ with $\sum_i(S) = t$?

- e.g. given rods of various lengths & want to span a given length exactly
  → Lego → batteries
- e.g. pack $1 \times a_i$ rectangles into $2 \times \frac{\sum_i A}{2}$ rect.
  $\Rightarrow t = \frac{\sum_i A}{2}$ ~ special case called Partition

First attempt: suffixes

① ~~subproblem~~ $x(i)$ = ~~is there a subset $S \subseteq A[i:]$ with $\sum(S) = t$?~~ WRONG

$\Rightarrow$ # subproblems $= n+1$

② guess whether $A[i] \in S$

recurrence: $x(i) = \text{OR}(x(i+1),$ ← $A[i] \notin S$

~~$x(i+1)$~~ if $a_i \leq t$) ← $A[i] \in S$

↳ need smaller t

Second attempt: + smaller t

① subproblem $x(i, s)$ = is there a subset $S \subseteq A[i:]$ with $\sum(S) = s$?

$\Rightarrow$ # subproblems $= (n+1) \cdot (t+1)$

② guess whether $A[i] \in S$ ← 2 choices

recurrence: $x(i, s) = \text{OR}(x(i+1, s),$ ← $A[i] \notin S$
$x(i+1, s-a_i)$ if $a_i \leq s)$ ← $A[i] \in S$

$\Rightarrow$ time / subproblem $= O(1)$

acyclic: decreasing i is a topological order

③ base cases: $x(n, 0) = \text{TRUE}$

$x(n, >0) = \text{FALSE}$

④ solve problem: $x(0, t)$

⑤ time = # subproblems $\cdot$ time/subprob.

$O(nt)$ $O(1)$

$= O(nt)$

Submultiset sum: given n positive integers $A = \{a_0, a_1, \ldots, a_{n-1}\}$ and target sum t (coin denom^n's.)

(change making)

find $m_0, m_1, \ldots, m_{n-1} \geq 0$ with $\sum_{i=1}^{n-1} m_i \cdot a_i = t$ (m_i: # of each coin)

minimizing $\sum_{i=1}^{n-1} m_i$ (total # coins)

- greedy algorithm is optimal for e.g. U.S./Euro coins but not e.g. for $A = \{1, 5, 18, 25\}$, $t = 36$

["What This Country Needs is an 18¢ Piece" by Shallit '03]

① **subproblem** $x(i, s) = \min \sum_{j=i}^{n-1} m_j$ s.t. $\sum_{j=i}^{n-1} m_i \cdot a_i = s$

$\Rightarrow$ # subproblems $= (n+1) \cdot (t+1)$

② **guess** $m_i \in \{0, 1, \ldots, \text{largest } m \text{ with } m \cdot a_i \leq s\}$

recurrence:

$$x(i, s) = \min \left\{ m_i + x(i+1, s - m_i \cdot a_i) \mid 0 \leq m_i \leq \frac{s}{a_i} \right\}$$

$\Rightarrow$ time/subproblem $= O(t)$

acyclic: decreasing i

③ **base cases:** $x(n, 0) = 0$

$x(n, >0) = \infty$ (invalid solution)

④ **solve problem:** $x(0, t)$

⑤ **time** $= O(nt) \cdot O(t)$

$= O(nt^2)$

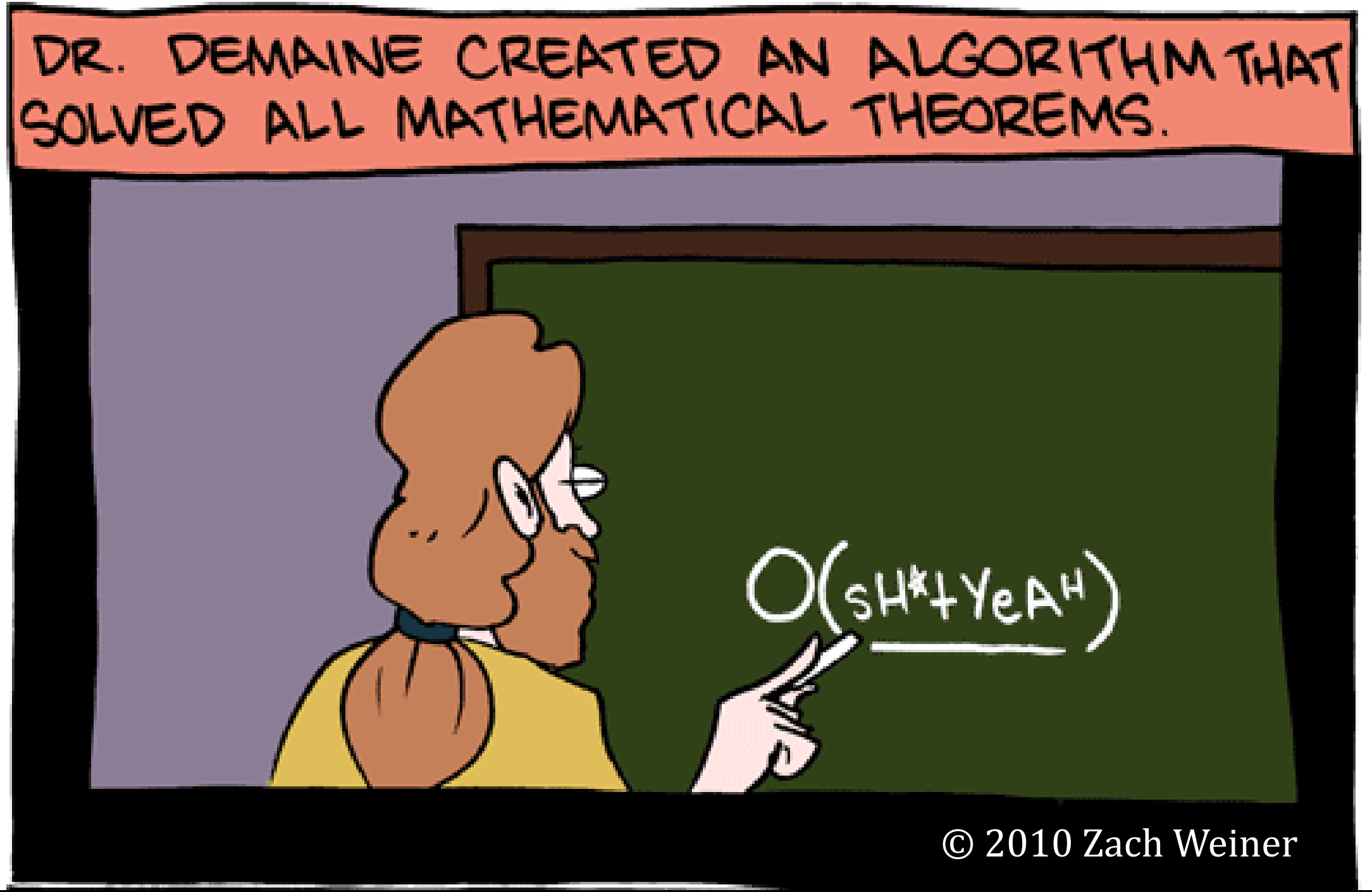
AMAZING: effectively trying all possible subsets!
... but is this actually fast?

Polynomial time = polynomial in input size
- here $\Theta(n)$ if number t fits in a word
  $O(n \lg t)$ in general (can assume $a_i \leq t$)
- t is exponential in $\lg t$ (not polynomial)

Pseudopolynomial time = polynomial in the problem size AND the numbers in input
here: t & a_i's
- $\Theta(nt)$ is pseudopolynomial

$\Longrightarrow$

Remember: polynomial – GOOD
exponential – BAD
pseudopoly. – SO SO


DR. DEMAINE CREATED AN ALGORITHM THAT SOLVED ALL MATHEMATICAL THEOREMS.
O(SH*TYEAH)
© 2010 Zach Weiner

```
# top-down, memoized
memo = {}
def x(a, b):
    if (a, b) not in memo:
        if base case:
            memo[a, b] = B(a, b)
        else:
            memo[a, b] = R(a, b, x)
    return memo[a, b]
S(x)
```

B = Base Case
R = Recurrence
S = Solution

```
# bottom-up
    x = {}

    for (a, b) in topological order:
          if base case:
                    x[a, b] = B(a, b)
          else:
                    x[a, b] = R(a, b, x)

    S(x)
```

B = Base Case
R = Recurrence
S = Solution

```
def R(a, b, x):
    best = math.inf
    for choice in choices:
        subcost = x(a′, b′)
        cost = c(subcost, choice)
        if cost < best:
            best = cost
    return best
```

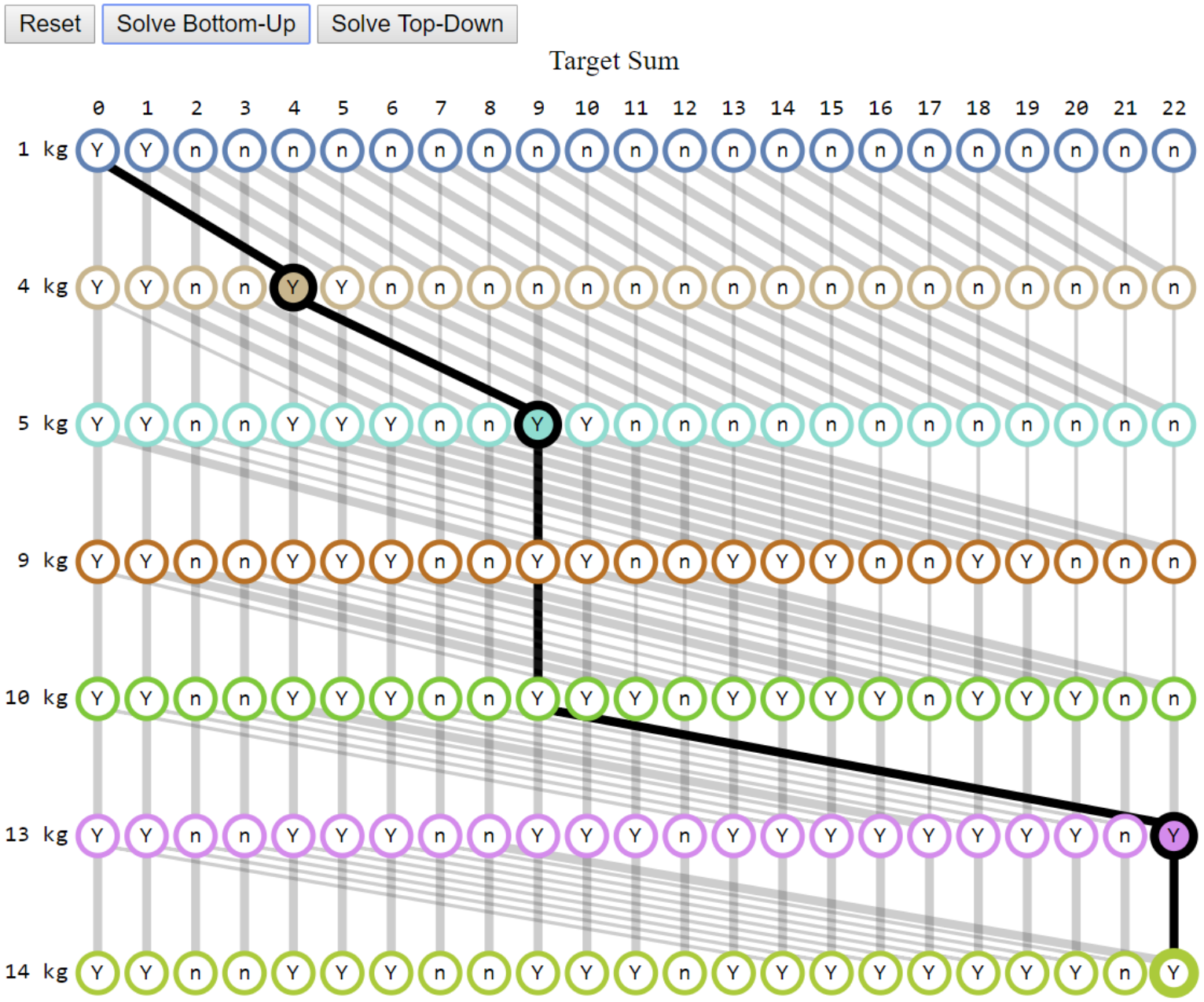
B = Base Case
R = Recurrence
S = Solution

```
def R(a, b, x):
        guess choice from choices
        subcost = x(a′, b′)
        return c(subcost, choice)
```

B = Base Case
R = Recurrence
S = Solution

Past DPs:	① subprobs.	② guess/choice
[L16] Fibonacci	smaller n	nothing
[L16] DAG SSSP	vertex v	last edge on $s \to v$ path
[L17] SSSP (B-F)	+ # edges	last edge on $s \to v$ path
[L17] Rod cutting	smaller rod	last/first piece
[R16] Text justification	suffixes	first linebreak
[R16] L.I.S.	suffixes + use $A[i]$	next $A[j]$ used
[R17] L.C.S.	A prefix, B prefix	drop from A or B?
[L18] Edit distance	A prefix, B prefix	rightmost edit
[R17] Alt. coin game	substrings	coin to take
[L18/19] Arith. parens.	substrings + min/max	root operation
[L19] Egg drop	smaller floors/eggs	first drop height

```
def subset_sum(A, t): # bottom up
    x = [[None]*(t+1) for i in range(len(A)+1)]
    x[n][0] = True
    for s in range(1, t+1):
        x[n][s] = False
    for i in reversed(range(len(A))):
        for s in range(t+1):
            if A[i] <= s:
                x[i][s] = x[i+1][s] or \
                          x[i+1][s-A[i]]
            else:
                x[i][s] = x[i+1][s]
    return x[0][t]
```



Take?	Largest Subset	Sum to 22 kg?
No	1 kg, 4 kg, 5 kg, 9 kg, 10 kg, 13 kg, 14 kg	Yes
Yes	1 kg, 4 kg, 5 kg, 9 kg, 10 kg, 13 kg, 14 kg	No, only 20 kg

https://codepen.io/mit6006/pen/JeBvKe

```
def submultiset_sum(A, t): # b-up change making
    x = [[None]*(t+1) for i in range(len(A)+1)]
    x[n][0] = True
    for s in range(1, t+1):
        x[n][s] = False
    for i in reversed(range(len(A))):
        for s in range(t+1):
            x[i][s] = min(
                m_i + x[i+1][s - m_i*A[i]]
                for m_i in range(s // A[i] + 1))
    return x[0][t]
```

Past DPs:	① subprobs.	② guess/choice
[L16] Fibonacci	smaller n	nothing
[L16] DAG SSSP	vertex v	last edge on $s \to v$ path
[L17] SSSP (B-F)	+ # edges	last edge on $s \to v$ path
[L17] Rod cutting	smaller rod	last/first piece
[R16] Text justification	suffixes	first linebreak
[R16] L.I.S.	suffixes + use A[i]	next A[j] used
[R17] L.C.S.	A prefix, B prefix	drop from A or B?
[L18] Edit distance	A prefix, B prefix	rightmost edit
[R17] Alt. coin game	substrings	coin to take
[L18, L19] Arith. parens.	substrings + min/max	root operation
[L19] Egg drop	smaller floors/eggs	first drop height
[L20] Subset sum	{ suffixes + smaller t	include A[i]?
[L20] Change making	{ suffixes + smaller t	# uses of A[i]