Optimization Problems, Lecture 2, Segment 3

John Guttag

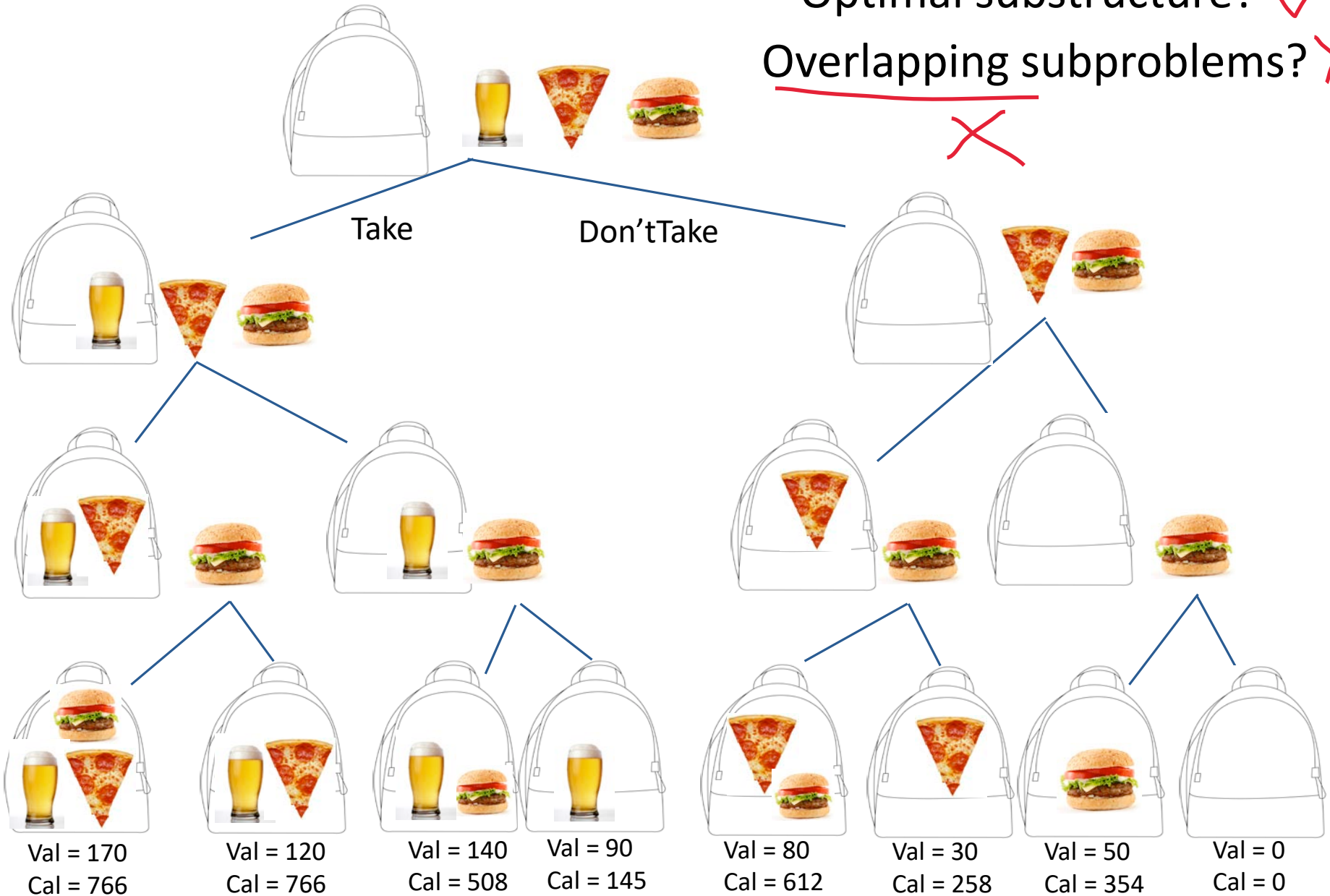MIT Department of Electrical Engineering and Computer Science

Dynamic Programming

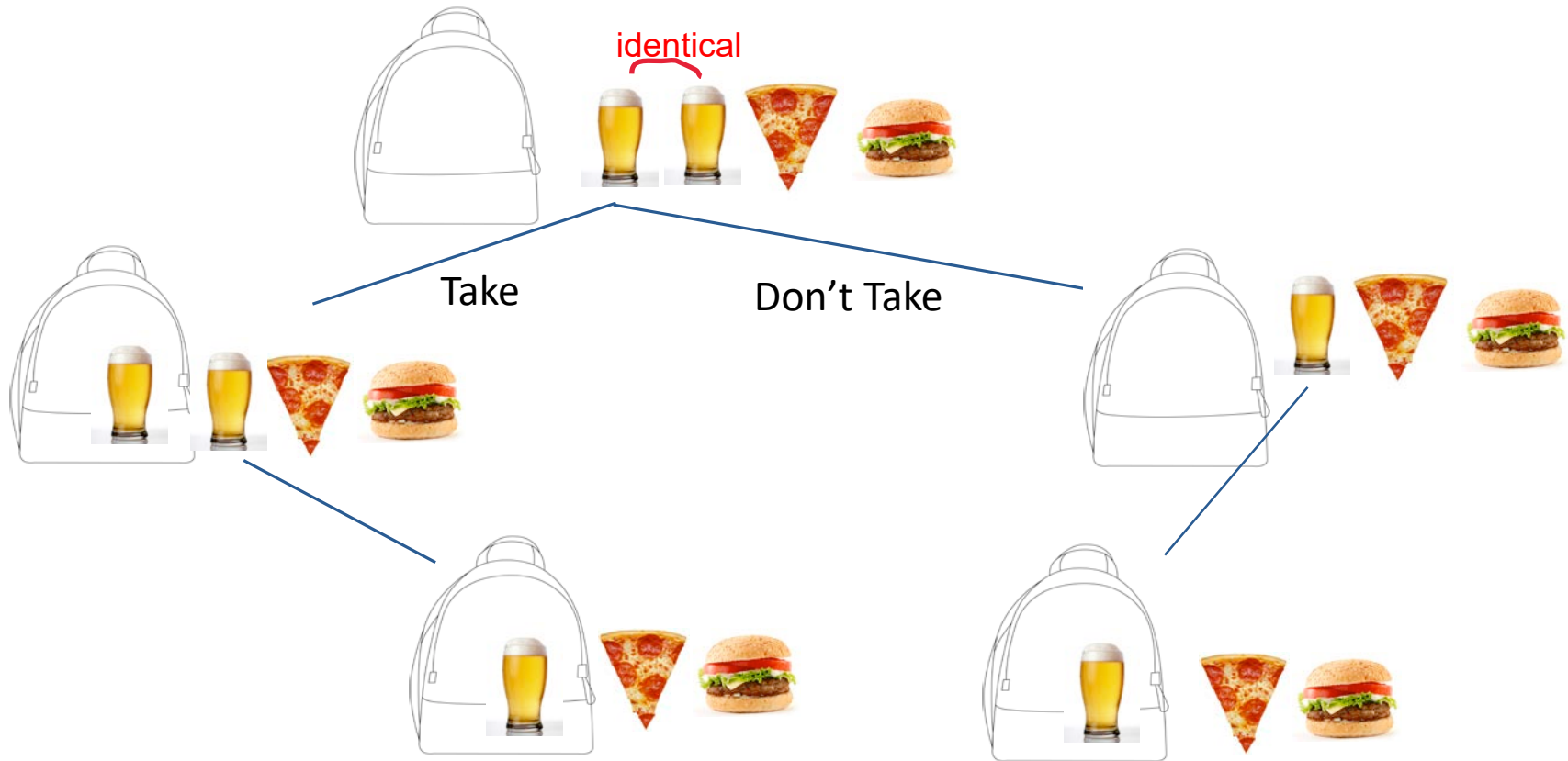- Optimal substructure: a globally optimal solution can be found by combining optimal solutions to local subproblems
  - For x > 1, fib(x) = fib(x - 1) + fib(x − 2)

- Overlapping subproblems: finding an optimal solution involves solving the same problem multiple times
  - Compute fib(x) or many times

Search Tree

Optimal substructure? ✓

Overlapping subproblems? ✗

Take   Don'tTake

Val = 170	Val = 120	Val = 140	Val = 90	Val = 80	Val = 30	Val = 50	Val = 0
Cal = 766	Cal = 766	Cal = 508	Cal = 145	Cal = 612	Cal = 258	Cal = 354	Cal = 0

A Different Menu

identical

Take

Don't Take

Need Not Have Copies of Items

Item	Value	Calories
a	6	3
b	7	3
c	8	2
d	9	5

Search Tree

- Each node = <taken, left, value, remaining calories>

0: {}, [a,b,c,d], 0, 5

1: {a}, [b,c,d], 6, 2

6: {}, [b,c,d], 0, 5

2: {a}, [c,d], 6, 2

7: {b}, [c,d], 7, 2

11: {}, [c,d], 0, 5

3: {a,c}, [d], 14, 0

4: {a}, [d], 6, 2

8: {b,c}, [d], 15, 0

9: {b}, [d], 7, 2

12: {c}, [d], 8, 3

14: {}, [d], 0, 5

5: {a}, [ ], 6, 2

10: {b}, [ ], 7, 2

13: {c}, [ ], 8, 3

15: {d}, [ ], 9, 0

16: {}, [ ], 0, 5
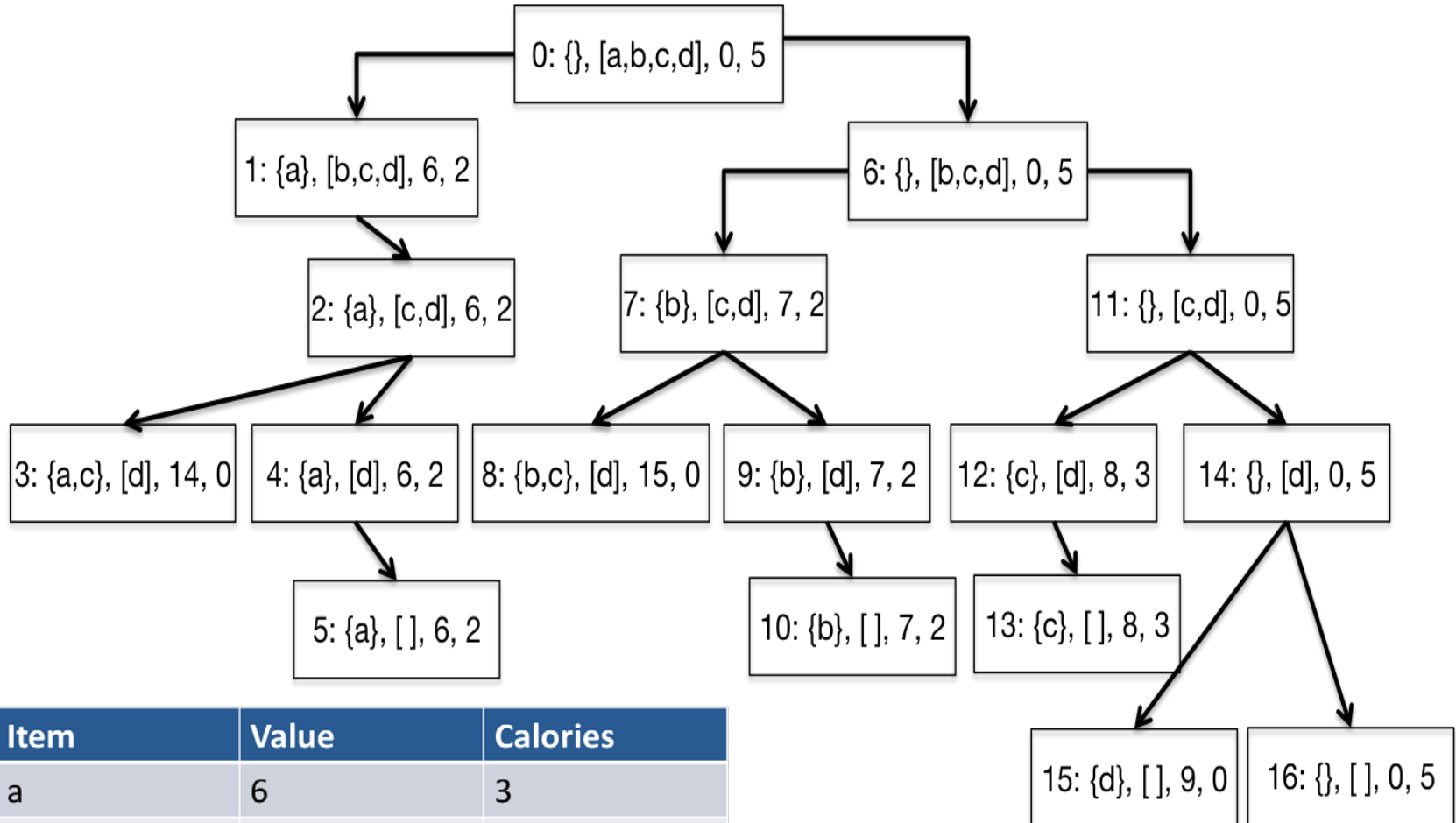
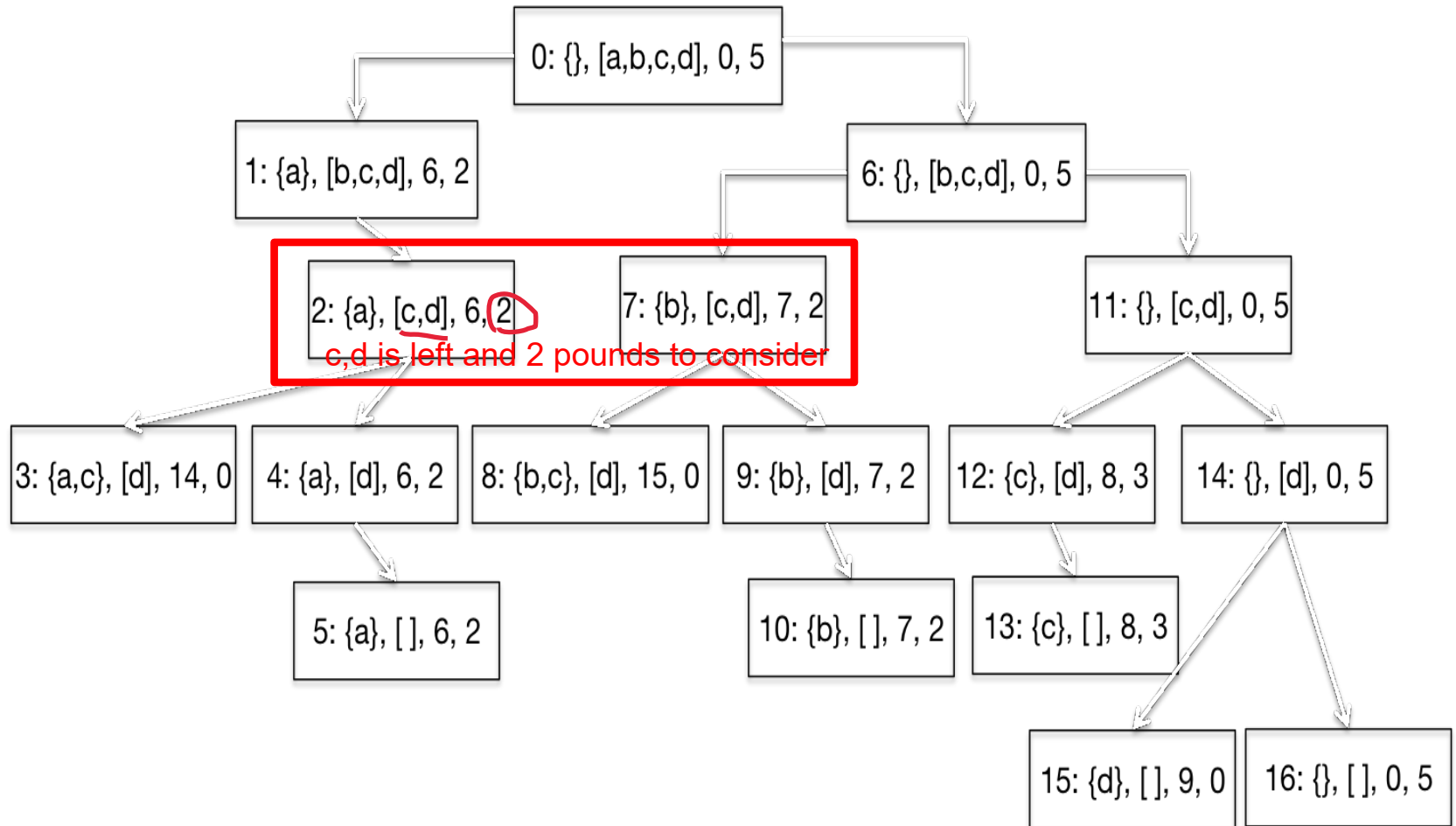| Item	Value	Calories
a	6	3
b	7	3
c	8	2
d	9	5

What Problem is Solved at Each Node?

- Given remaining weight, maximize value by choosing among remaining items

- Set of previously chosen items, or even value of that set, doesn't matter!

Overlapping Subproblems

0: {}, [a,b,c,d], 0, 5

1: {a}, [b,c,d], 6, 2

6: {}, [b,c,d], 0, 5

2: {a}, [c,d], 6, 2
c,d is left and 2 pounds to consider

7: {b}, [c,d], 7, 2

11: {}, [c,d], 0, 5

3: {a,c}, [d], 14, 0

4: {a}, [d], 6, 2

8: {b,c}, [d], 15, 0

9: {b}, [d], 7, 2

12: {c}, [d], 8, 3

14: {}, [d], 0, 5

5: {a}, [ ], 6, 2

10: {b}, [ ], 7, 2

13: {c}, [ ], 8, 3

15: {d}, [ ], 9, 0

16: {}, [ ], 0, 5
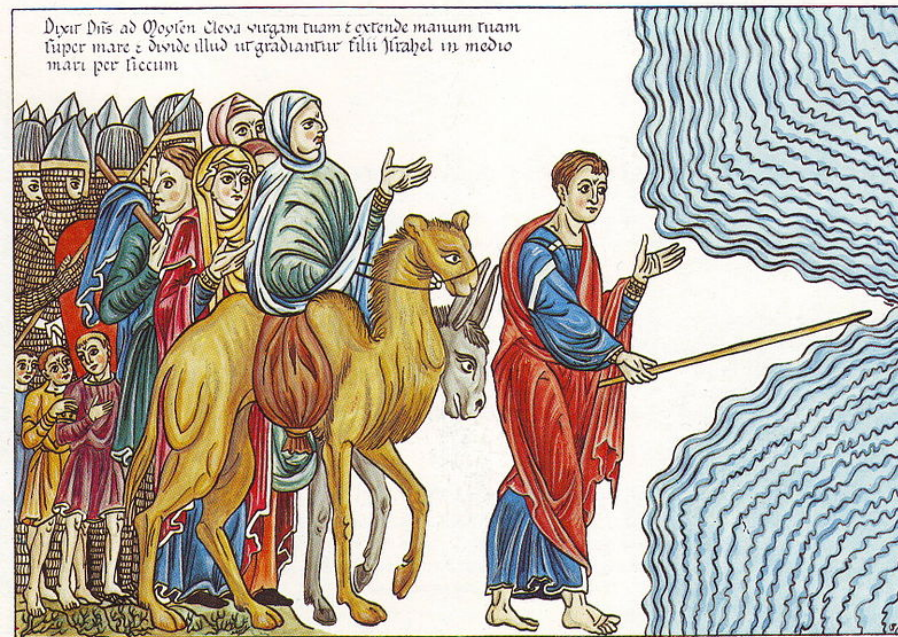
Modify maxVal to Use a Memo

- Add memo as a third argument
  - `def fastMaxVal(toConsider, avail, memo = {}):`

- Key of memo is a tuple
  - (items left to be considered, available weight)
  - Items left to be considered represented by `len(toConsider)` items are taken from left, so know the remain length = know the items remaining

- First thing body of function does is check whether the optimal choice of items given the the available weight is already in the memo

- Last thing body of function does is update the memo

Performance

len(items)	2**len(items)	Number of calls
2	4	7
4	16	25
8	256	427
16	65,536	5,191
32	4,294,967,296	22,701
64	18,446,744,073,709,551,616	42,569
128	Big	83,319
256	Really Big	176,614
512	Ridiculously big	351,230
1024	Absurdly big	703,802

How Can This Be?

- Problem is exponential

- Have we overturned the laws of the universe?

- Is dynamic programming a miracle?

Dixit Dñs ad Moysen Eleva virgam tuam & extende manum tuam super mare & divide illud ut gradiantur filii Israhel in medio mari per siccum

How Can This Be?

- Problem is exponential

- Have we overturned the laws of the universe?

- Is dynamic programming a miracle?

- No, but computational complexity can be subtle

- Running time of `fastMaxVal` is governed by number of distinct pairs, `<toConsider, avail>`
  - Number of possible values of `toConsider` bounded by `len(items)`
  - Possible values of `avail` a bit harder to characterize
    - Bounded by number of distinct sums of weights
  - Covered in more detail in assigned reading

Summary of Lectures 1-2

- Many problems of practical importance can be formulated as optimization problems

- Greedy algorithms often provide adequate (though not necessarily optimal) solutions

- Finding an optimal solution is usually exponentially hard

  reduce identical

- But dynamic programming often yields good performance for a subclass of optimization problems—those with optimal substructure and overlapping subproblems
  - Solution always correct
  - Fast under the right circumstances