Scientific Computing

Mon. Feb 16

Announcements

- HW 3 is due Friday, Feb 27
  covers brute force, search spaces,
  divide + conquer
  remember to let yourself struggle!

- Midterm exam
  Monday, March 2
  in class portion + takehome portion
  due Friday, March 6

Office Hours:

Mon, 9:30-10:30
Fri, 2:00-3:00

Cudahy 307

Topic 8 – Backtracking

Like Divide + Conquer, Backtracking is a framework for finding the optimal solution in a search space without checking every candidate one-by-one.

Very simple idea: Build solutions one part at a time, and give up when a partial solution violates the constraints.

Ex #1: Knapsack

Capacity: 10

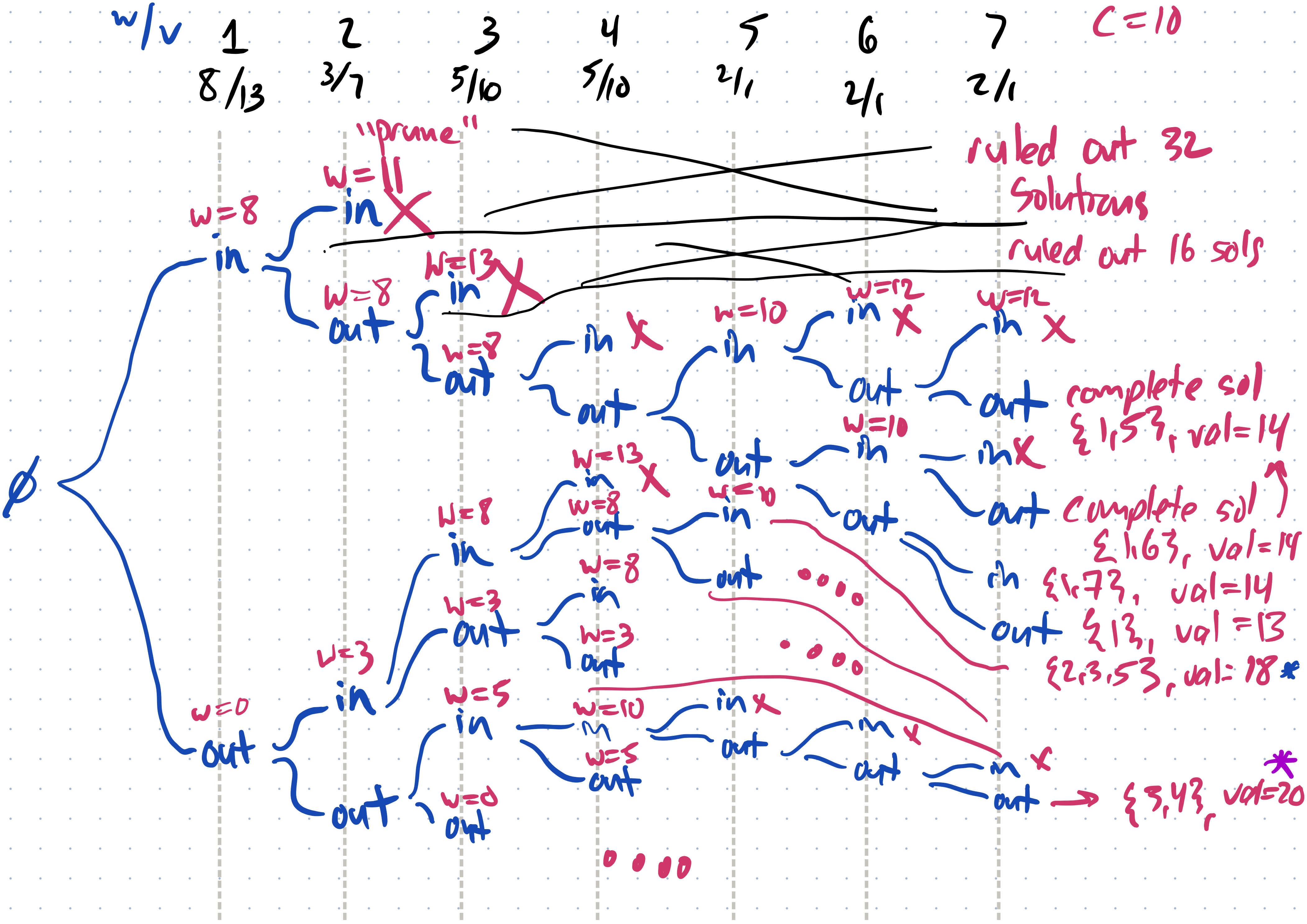
item	weight	value
1	8	13
2	3	7
3	5	10
4	5	10
5	2	1
6	2	1
7	2	1

With brute force:

Possibilities: $\emptyset$, $\{1\}$, $\{2\}$, …
$\{1,3,4,5,7\}$, …

not just too heavy, but still too heavy if you remove any single item, so this is silly to even try!

128 possibilities


w/v
1 8/13
2 3/7
3 5/10
4 5/10
5 2/1
6 2/1
7 2/1
C=10
"prune"
ruled out 32 Solutions
ruled out 16 sols
∅
w=8 in
w=11 in
w=8 out
w=13 in
w=8 out
in
out
w=10 in
out
w=12 in
out
w=12 in
out
complete sol {1,5}, val=14
w=10 in
in
out
complete sol {1,6}, val=14
out
in {1,7}, val=14
out {1}, val=13
{2,3,5}, val=18*
w=0 out
w=3 in
w=8 in
w=13 in
w=8 out
w=10 in
out
w=3 out
w=8 in
w=3 out
w=5 in
w=10 in
w=5 out
in
out
in
out
in
out
→ {3,4}, val=20
*
out
w=0 out
....

Messy picture, but way better than brute force, especially with lots of items!

"partially ordered set"

What are we doing?

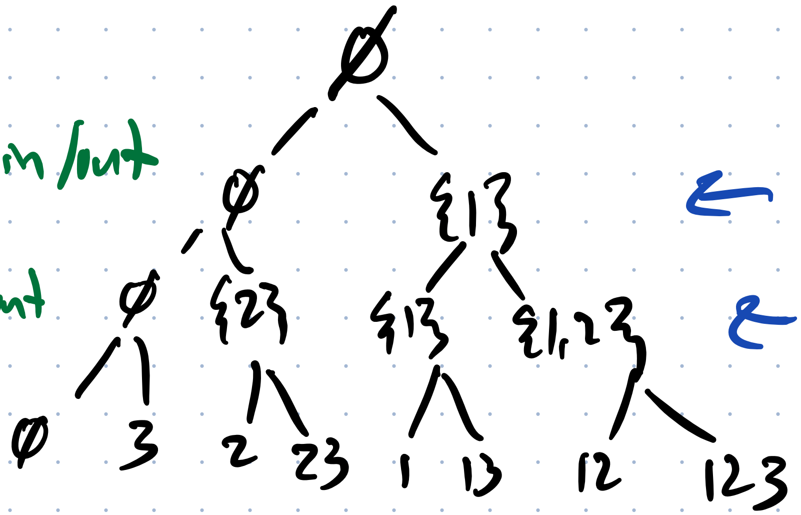
- Putting a hierarchy on decisions that builds the whole space (math: poset) with the critical property that: if a candidate is bad, then the candidates below it must be bad.

partially built solution

Knapsack with 7 items:

Candidates: subsets of $\{1,2,3,4,5,6,7\}$

hierarchy:



item 1 in/out

item 2 is in/out

← all subsets of $\{1\}$

← all subsets of $\{1,2\}$

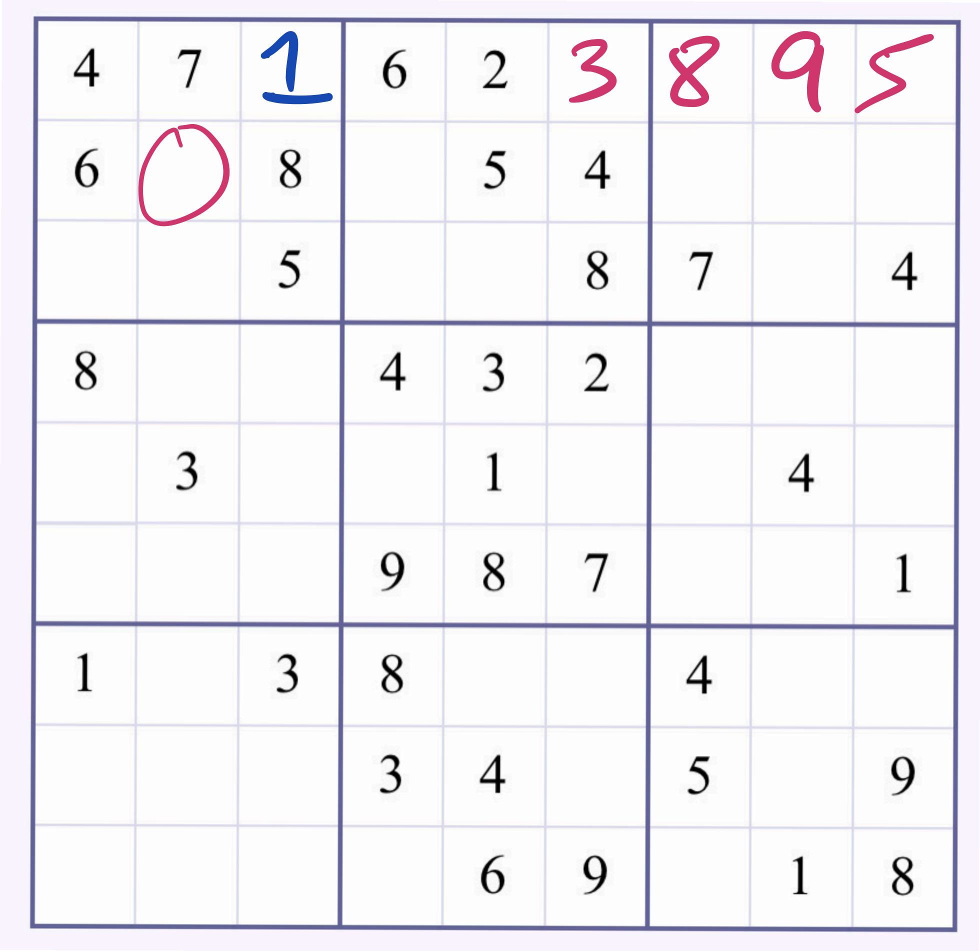
Traverse this tree, and whenever you reach a candidate that is bad, stop traversing that branch.

So, we are checking *or ruling out* every candidate in the search space. In bad cases (high capacity, light items), we might not rule anything out, and so in the worst case this is as bad as brute force.

[demo]

Ex #2: Sudoku

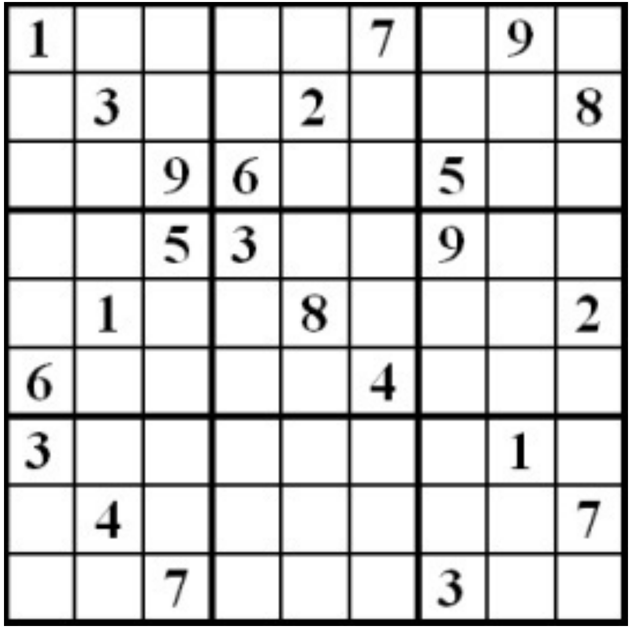
- Start filling in blank cells L-to-R then T-to-bottom.
- Start each cell at 1.
- If the cell doesn't violate a rule, move to the next cell.
- If not, bump up the value.
- If you run out of possibilities, go back to the previous cell.



4	7	1	6	2	3	8	9	5
6	○	8		5	4			
		5			8	7		4
8			4	3	2			
	3			1			4	
			9	8	7			1
1		3	8			4		
			3	4		5		9
				6	9		1	8

* online demo – jaypantone.com/sudoku
/sudoku-slow

"Hardest Sudoku Ever"



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