MSSC 6000

Feb 28 2022 - Day 16

Topic #6 - Divide and Conquer (continued)

Ex #3 - Counting Inversions.

Consider a list of distinct #.

$L = 3 \quad 19 \quad -7 \quad 2 \quad 1 \quad 6 \quad 0 \quad -10$

$5 + 6 + 1 + 3 + 2 + 2 + 1 + 0 = 20$

An inversion is a pair (L_i, L_j) where $i < j$ but $L_i > L_j$ (an out-of-order pair)

20 inversions.

Goal: compute the # of inversions in a list of n elements

Obvious Algorithm: Check every pair: $O(n^2)$

Divide + Conquer: $O(n \cdot \log(n))$

Divide + Conquer:

$$\underbrace{3 \quad 19 \quad -7 \quad 2}_{} \quad \underbrace{1 \quad 6 \quad 0 \quad -10}_{}$$

recursively count 4 inversions

recursively count 5 inversions

So we count 9 inversions within a half but we're missing the 11 between halves.

One solution: Check all pairs (A, B) where A is in the first half and B in second half. $\frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4} = O(n^2)$. So this would defeat the purpose of D+C.

Here's the trick: While we're counting inversions, we'll <u>also</u> sort the list with merge sort (which takes $O(n \cdot \log(n))$ time).

$$\underbrace{3 \quad 19 \quad -7 \quad 2}_{} \quad \underbrace{1 \quad 6 \quad 0 \quad -10}_{}$$

recursively count 4 inversions

recursively count 5 inversions

[-7 2 3 19] [-10 0 1 6]

Now, we recombine the lists just like with mergesort, and when do we detect an inversion?

Any time we take from the red list, there is an inversion for everything left in the blue list.

$\cancel{-7}$ $\cancel{2}$ $\cancel{3}$ $\cancel{19}$ | $\cancel{-10}$ $\cancel{0}$ $\cancel{1}$ $\cancel{6}$

[-10 -7 0 1 2 3 6 19]

$+4$ $+3$ $+3$ $+1$

$= 11$

$4 + 5 + 11 = 20$ inversions

Time: $T(n) = 2\,T(\frac{n}{2}) + 2n$

$\rightsquigarrow T(n) = O(n \cdot \log(n))$

<u>Ex #4</u>: Closest Pair of Points:

Brute Force: $O(n^2)$

70s - D+C: $O(n \cdot \log(n))$

2 more famous D+C algos.

Integer Multiplication

Input: Two n-digit #s x and y

Output: $x \cdot y$

Brute Force:

$$
\begin{array}{r}
\overset{2}{1}72 \\
\times\ 424 \\
\hline
688 \\
3440 \\
68800 \\
\hline
72928
\end{array}
\qquad O(n^2)
$$

Divide + Conquer: $T(n) \leq 3T\left(\frac{n}{2}\right) + n$

$$\Rightarrow T(n) = O\left(n^{\log_2(3)}\right)$$

$$= O\left(n^{1.59\ldots}\right)$$

Matrix Multiplication: n = dimension

$$
\begin{bmatrix} - & - & - \\ - & - & - \\ - & - & - \end{bmatrix}
\begin{bmatrix} - & - & - \\ - & - & - \\ - & - & - \end{bmatrix}
=
\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}
$$

$\overset{n^2}{9}$ (row, col) pairs, n operations per pair

$O(n^3)$ Determinants

1969 Strassen's algo, $O(n^{\log_2(7)}) = O(n^{2.807})$

1978 $n^{2.796}$

1979 $n^{2.780}$

1981 $n^{2.522}$

1986 $n^{2.479}$

1990 $n^{2.3755}$

2010 $n^{2.3737}$

2013 $n^{2.3729}$

2014 $n^{2.3728639}$

2020 $n^{2.3728596}$

No one knows how fast matrix multiplication can get!

Lower bound: $O(n^2)$

Topic #7 - Backtracking

Like D+C: Backtracking is an algorithmic paradigm to find an 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 partially built 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:
$2^7 = 128$

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

way too heavy

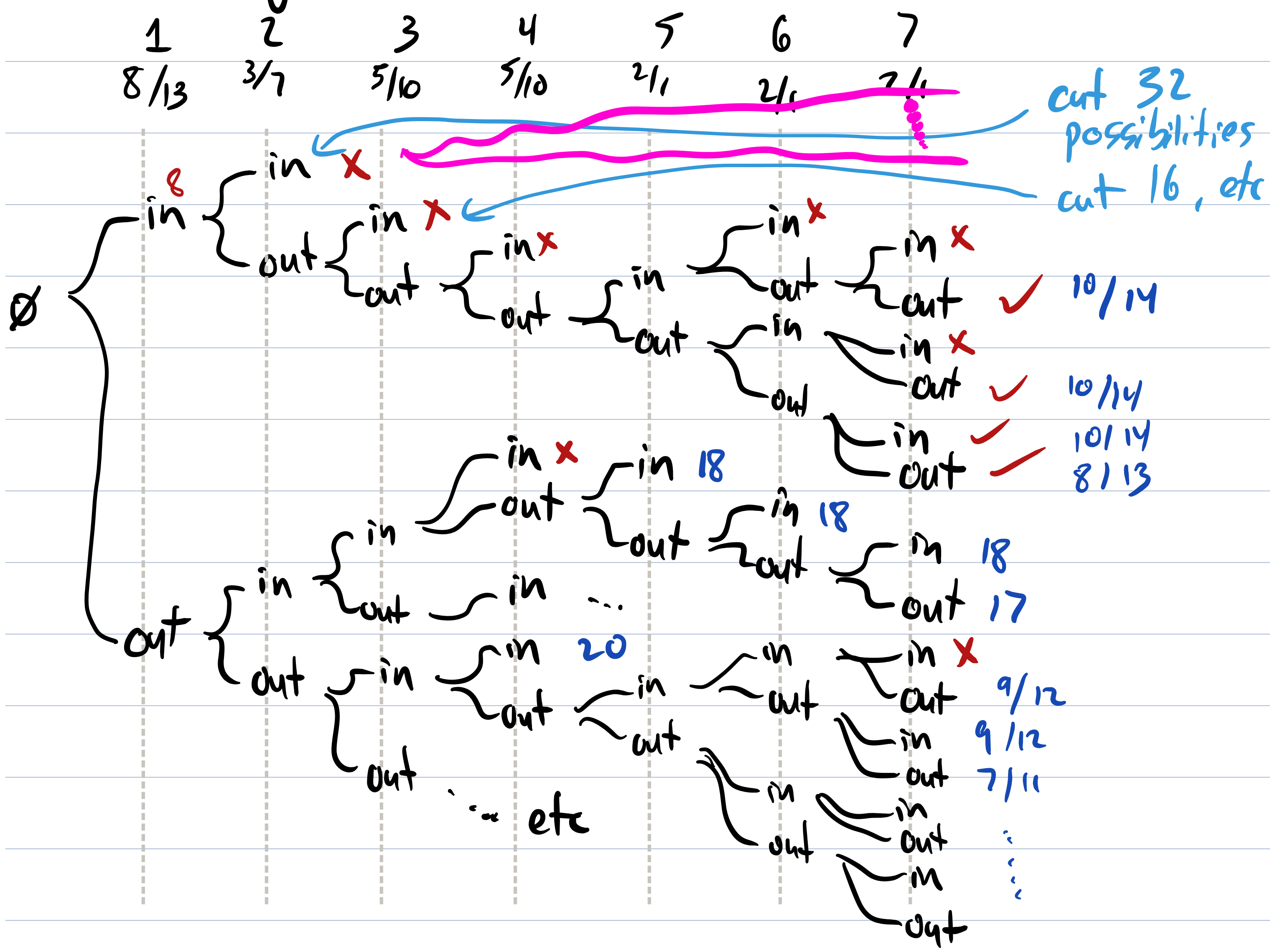
$\{1,3\}$ is too heavy, so checking $\{1,3,4,5,7\}$ was silly

Build up possible solutions by deciding in steps:
- Item #1 is in or out
- Item #2 is in or out

....

Backtracking:

weights / values



Way better than brute force.

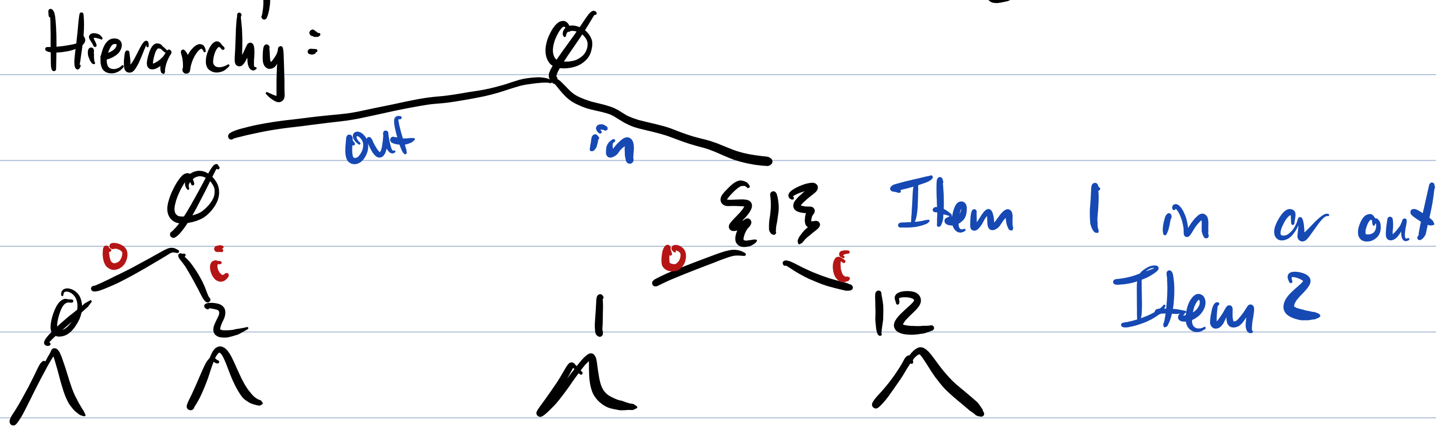
What are we doing?

- Putting a hierarchy on decisions that builds the whole search space, with the critical property that if a partially built solution is bad, then every way of completing it must also be bad.

Knapsack with 7 items:

Search space: all subsets of $\{1,2,3,4,5,6,7\}$

Hierarchy:



Traverse this tree, and whenever you reach a candidate that is bad, prune the branch (stop exploring downward) go back upward, and explore a different branch.