Last week: for QuickSort we decided to use median as pivot, and reduced to Selection.

I more general than median!

Imagine (recursively) looking for the median:

\[
\begin{align*}
& \leq A[p] \quad A[p] \geq A[0] \\
\text{if } p > \frac{n}{2} \rightarrow \\
& \text{recurse left} \quad \text{"median" of } A[1..p-1] \\
& \text{is not median of } A[1..p-1].
\end{align*}
\]

Immediately realize that "recursing" requires thinking of the more general Selection problem.

\[\rightarrow \text{Sometimes, to do recursion, sometimes first generalize the problem.} \]

Recursive Backtracking

\[
\begin{array}{c}
\text{n x n chessboard} \\
\text{place n "queens" on the board s.t. none of them are attacking each other (or report impossible).}
\end{array}
\]
Generalize:

Given an n x n chessboard & placements of r queens on the first r rows, 
Place n-r queens on the remaining rows 
so that no queens attack each other.

```
PlaceQueens(Q[1..r])
if r = n
    return Q
else
    for j = 1 to n
        brute force check if column j is valid for n-r+1
```

Here, which (valid) position to pick?
Recurse me backtracking:
Try all possible choices at current step
    ⇒ recurse
Figure out based on result of recurse all which choice was correct.

(doing DFS through dependency tree of the recursive function)
brute force check if column \( j \) is valid for \( w = r + 1 \)
if so,
    recurse \( w \rightarrow Q(1, \cdots, r) \) appended \( w \rightarrow j \).
if recurse call did not return fail
    return output.

return fail.

X

learn to live

\( \text{学生活} \)

student job

name of a café/bookstore on green st?

Preprocessing problem: given a string \( w \), a dictionary (to some linguistic) \( A \) (e.g. et al.)
can we split the string into valid sequence of words in the dictionary?

**BOTHEARTLANDSATURNSPIN**

(assume black box access to \( \text{IsWord}(w) \))

**BOTHEARTLANDSATURNSPIN**

\( \Rightarrow \begin{cases} \text{true} & \text{yes} \ \\
\text{false} & \text{no} \end{cases} \)

\( \Rightarrow \begin{cases} \text{true} & \text{yes} \ \\
\text{false} & \text{no} \end{cases} \)
Is Splittable? \( (w \[1 \ldots n]) \) = \( \begin{cases} \text{True} & \text{if } n = 3 \\ \bigvee_{i=1}^{n} \left( \text{isWord}(w \[1 \ldots i]) \land \text{IsSplittable}(w \[i+1 \ldots n]) \right) & \text{o.w.} \end{cases} \)

equivalently:

fix input \( w \[1 \ldots n] \)

Is Splittable? \( (i) \) = \( \begin{cases} \text{True} & \text{if } i > n \lor w \[1 \ldots n] = 3 \\ \bigvee_{j=i+1}^{n} \left( \text{isWord}(w \[j \ldots n]) \land \text{IsSplittable}(j + 1) \right) & \text{o.w.} \end{cases} \)

What problem are you really solving recursively?

- what are the recursive subproblems?

what do you need to remember about past decisions?

(optional) Encode subproblem more efficiently?

---

Longest Increasing Subsequence

Input sequence \( A \[1 \ldots n] \) of numbers (array)

Looking for the length of the longest \( w \) subseq of \( A \).

\( B \[i] > B \[i-1] \) for all \( i > 2 \).

Q: What is a (local) decision we care about?

is \( A \[i] \) in a LIS?

3 1 4 1 5 9 2 6

(String) Sing is a subsequence

a sequence \( B \[1 \ldots n] \) is increasing.
Q: What is the actual problem I'm solving recursively?

LIS greater \((A[1:n], k)\) =

\[
\begin{cases}
0 & \text{if } A[0] = 0 \\
\max \{1 + \text{LIS greater } (A[i+1:n], k) \} & \text{if } A[i] > k \\
\text{LIS greater } (A[1:i-1:n], k) & \text{o.w.}
\end{cases}
\]

Q: What to remember? \(k\) for solving LIS for \(k \geq 2\).

Q: Can we create more efficiently? Yes!

How to use LIS greater to solve original problem?

\[\text{LIS}(A[1:n]) = \text{LIS greater}(1, -\infty)\]