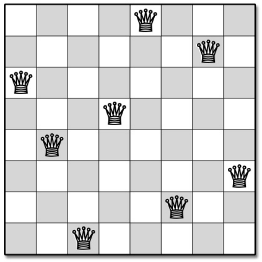
HW 5 today
HW 6 next Tuesday
GPS 6 next Monday

Backtracking => ???

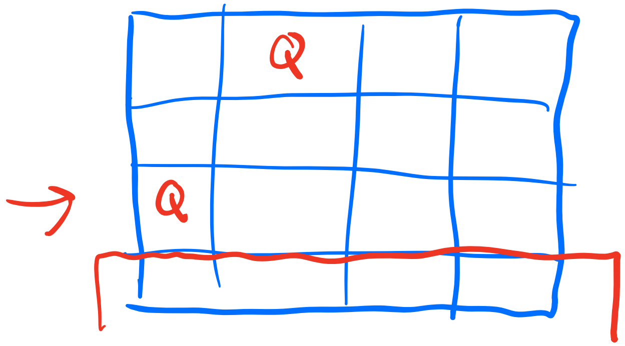
FIRST MAKE IT WORK,
THEN MAKE IT FAST

n Queens : Given n×n chess board, what are all the ways to place n queens so none are attacking another?

No two on same row, column, or diagonal.



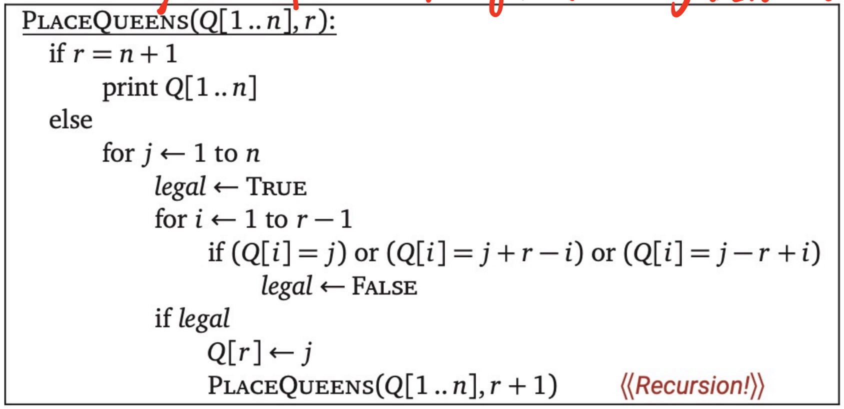
Where do we put the current row's queen (considering rows in top-down order).



Need to remember placement of higher queens..

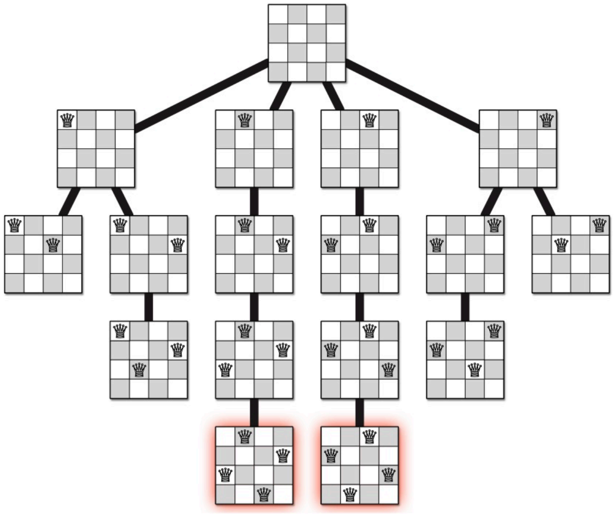
Try each position in row that's not attacked, For each, Recurse on the later rows.

All way to place n queens given for all $i < r$, we placed on in row i, column $Q[i]$.



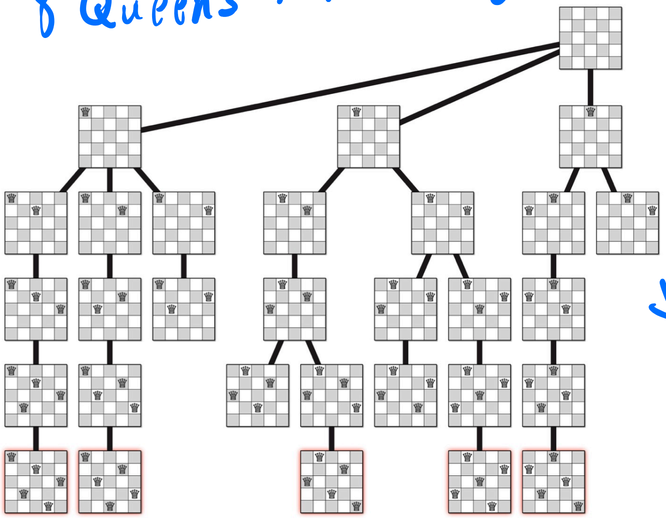
```
PlaceQueens(Q[1..n], r):
    if $r = n + 1$
        print Q[1..n]
    else
        for $j \leftarrow 1$ to $n$
            legal ← True
            for $i \leftarrow 1$ to $r - 1$
                if $(Q[i] = j)$ or $(Q[i] = j + r - i)$ or $(Q[i] = j - r + i)$
                    legal ← False
            if legal
                $Q[r] \leftarrow j$
                PlaceQueens(Q[1..n], r + 1)        ⟨⟨Recursion!⟩⟩
```

4 Queens : 2 solutions



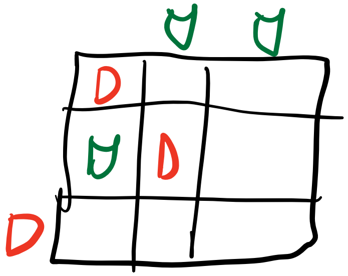
5 Queens : 10 solutions

8 Queens : 92 solutions



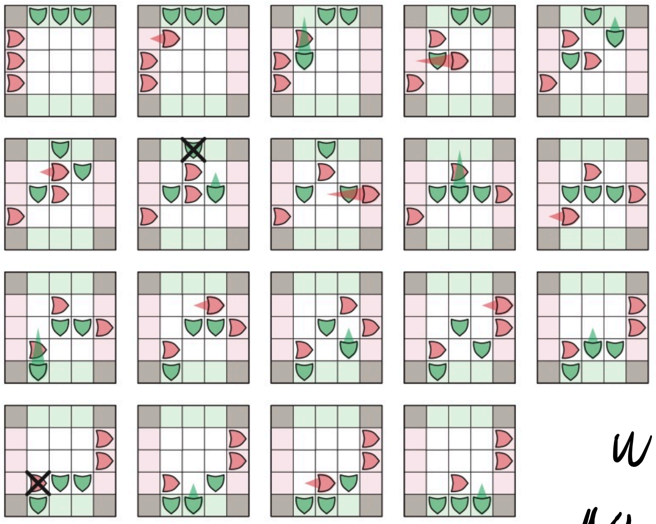
↓ mirror

A game between two players



n x n grid

Take turns moving one step foward to a blank or jumping opponent's packet to land in a blank.



Skip turn if no legal moves.

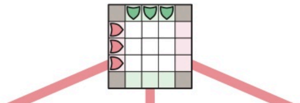
Want a strategy to win <u>any</u> two player game with no hidden info and no randomness.

state: all current info (sugar packet placement)

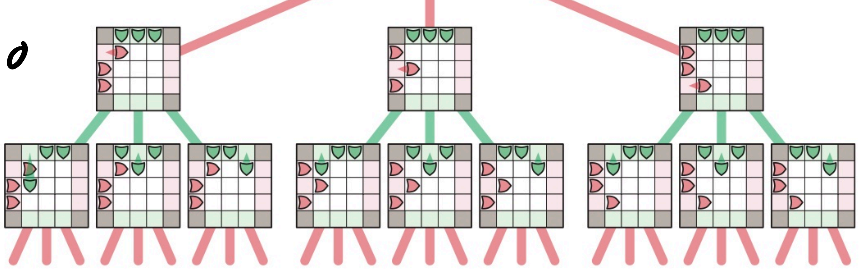
+ current player

game tree:

edge from x to y

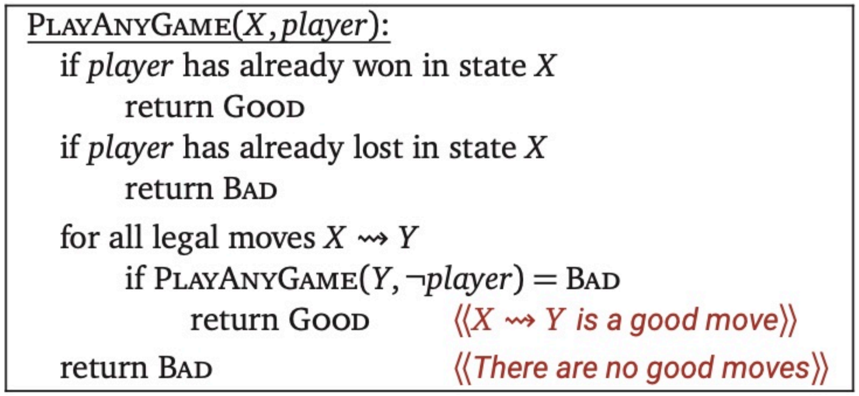


if state can go from x to y in one move.



call a state good: if current player has won or there is a move to put opponent in a bad state

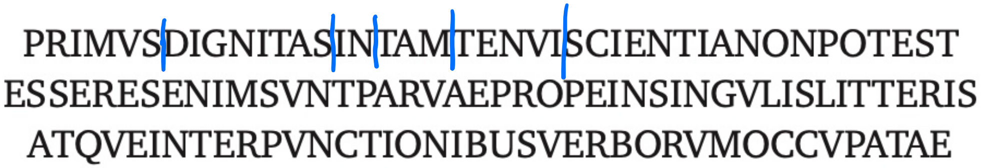
bad: already lost or all moves take opponent to a good state



```
PLAYANYGAME(X, player):
    if player has already won in state X
        return GOOD
    if player has already lost in state X
        return BAD
    for all legal moves $X \rightsquigarrow Y$
        if PLAYANYGAME(Y, ¬player) = BAD
            return GOOD            ⟨⟨$X \rightsquigarrow Y$ is a good move⟩⟩
    return BAD                     ⟨⟨There are no good moves⟩⟩
```

Is state X good or bad?
(current player is.. player)

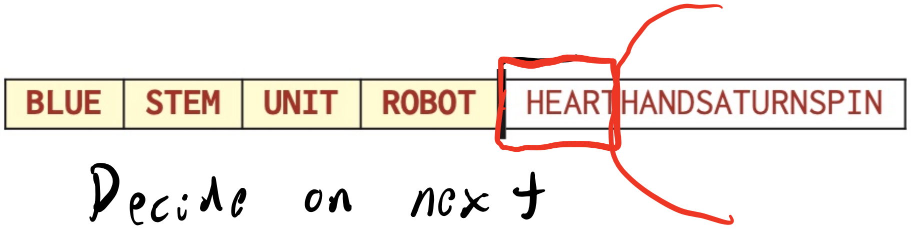
Text segmentation:



PRIMVSDIGNITASINTAMTENVISCIENTIANONPOTEST
ESSERESENIMSVNTPARVAEPROPEINSINGVLISLITTERIS
ATQVEINTERPVNCTIONIBVSVERBORVMOCCVPATAE

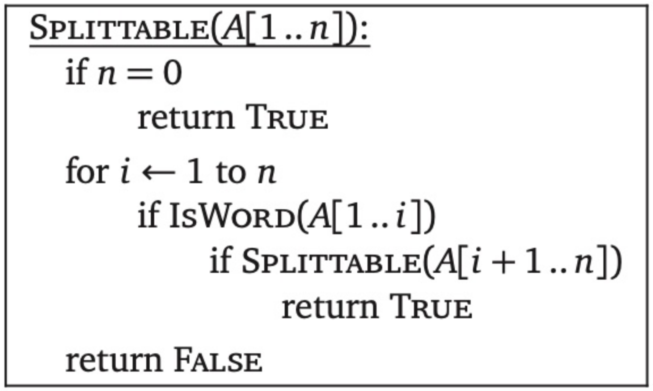
Given string $A[1..n]$ to segment, and a function IsWord where $\text{IsWord}(w) =$ True iff w is a "word".

Can we segment A into words?



Decide on next word, recurse on segmenting the rest...

Recursion just needs remaining substring.



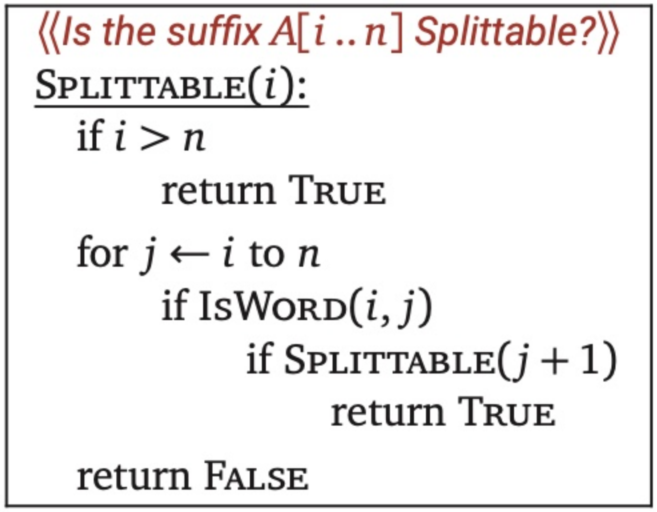
```
SPLITTABLE(A[1..n]):
  if n = 0
      return TRUE
  for i ← 1 to n
      if ISWORD(A[1..i])
          if SPLITTABLE(A[i+1..n])
              return TRUE
  return FALSE
```

Is $A[1..n]$ the concatenation of words?

$\leftarrow O(2^n)$

For "practice": Use $A[1..n]$ as a "global".

Pass the start index i of a suffix for recursion.



```
⟨⟨Is the suffix A[i..n] Splittable?⟩⟩
SPLITTABLE(i):
    if i > n
        return TRUE
    for j ← i to n
        if ISWORD(i, j)
            if SPLITTABLE(j + 1)
                return TRUE
    return FALSE
```

Splittable(i) = True iff $A[i...n]$ is the concat of words.

IsWord(i, j) = True iff $A[i..j]$ is a word.

$$\text{Splittable}(i) = \begin{cases} \text{True} & \text{if } i > n \\ \bigvee_{j=i}^{n} \Big(\text{IsWord}(i,j) \wedge \text{Splittable}(j+1)\Big) & \end{cases}$$

index formulation

useful for dynamic programming

or