Dynamic Programming

* Text-segmentation

Given a string $S[1, 2, .., n]$ & access to a function $\text{IsWord}(i, j)$ that checks if $S[i, .., j]$ is a valid word

Check if S can be partitioned into valid words

Recursion 1:

$$\text{split}(i) = \begin{cases} 1 & \text{if } S[1, .., i] \text{ can be split} \\ 0 & \text{o/w} \end{cases}$$

$$\text{Split}(i) = \bigvee_{j=1}^{i} \left(\text{IsWord}(1, j) \wedge \text{split}(?) \right) ?$$

$$\text{Split}(i) = \bigvee_{j=1}^{i} \left(\text{split}(j) \wedge \text{IsWord}(j+1, i) \right)$$

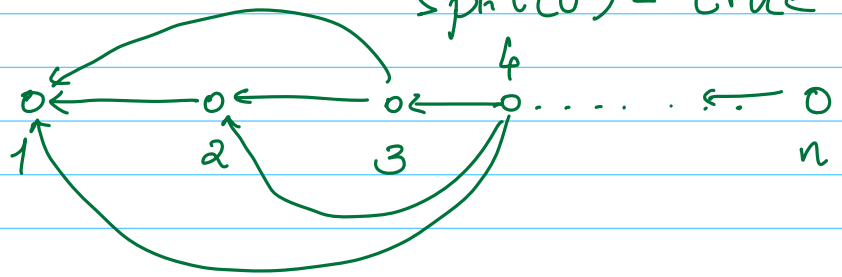
* Compute $\text{Split}(n)$

① How does the dependency digraph look?

② What are the base cases?

↳ split(1) = IsWord(1,1)

split(0) = true



Memoized Version

```
Segment(j)
  if j=0 return true
  if split(j) defined
     return split(j)
  for 1 ≤ k ≤ j
     if (Segment(k-1) ∧ IsWord(k, j))
        split(j) = true  → Memoization
        return
  split(j) = false
  return
```

Bottom-up version

```
Segment(n)
    split(0) = true
    for 1 ≤ i ≤ n
        split(i) = false
        for 1 ≤ k ≤ i
            if ( split(k-1) ∧ IsWord(k,i) )
                split(i) = true   (store split(i)=k)
```

$O(n^2)$ DFS on the digraph

- Understand the recursive structure and the dependencies to create the bottom-u version of the DP.

- You can construct the actual split of the string into segments as well. (How ?)