Dynamic programming in probabilistic problems

I. Recap on DP

II. DP in probabilistic problems

III. An example: frog feeding

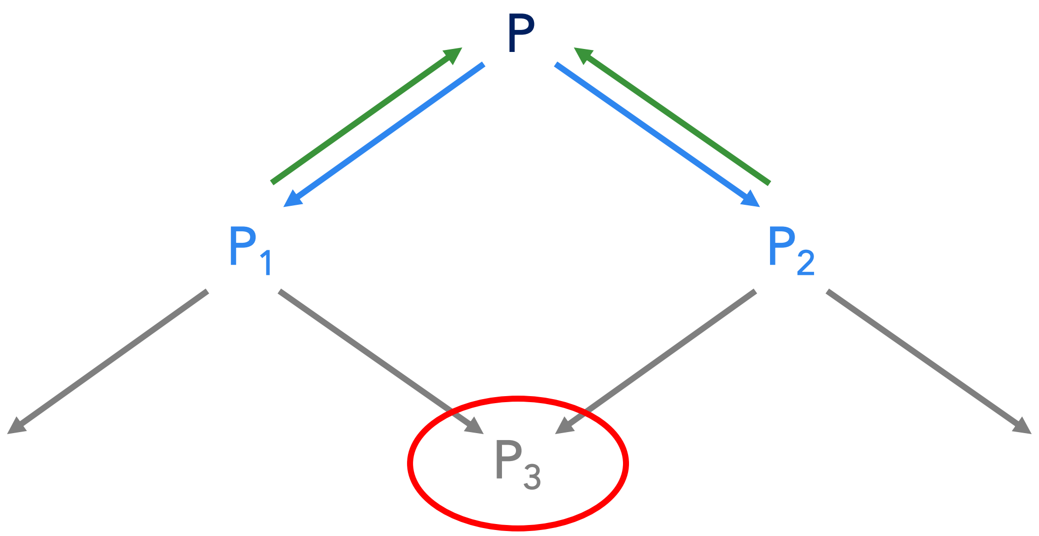
Recap on dynamic programming

Decomposition in subproblems

- The problem involves a "decision" which leads to subproblems
- The solution of the main problem can be obtained using the solutions of the subproblems

Overlapping subproblems

- While making choices, we end up with the same subproblem multiple times
- Key part of DP: store and reuse previously computed solutions



DP in probabilistic problems

Decomposition in subproblems

- The "decision" naturally arises in probabilistic problems
- For example:
  - With probability p the event E happens
  - With probability $(1-p)$ the complement event $\bar{E}$ happens

Useful tools to come up with a recurrence relation

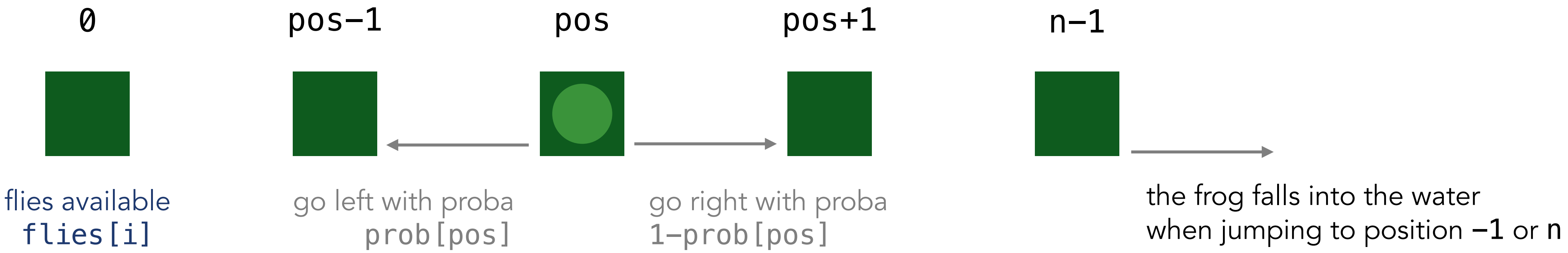
- Law of total probability
- Law of total expectation

  Next: we look at an example using it!

An example: frog feeding

Problem

A frog randomly jumps from leaf to leaf and eat the flies available there



Question

Starting from position x, what is the expected number of flies the frog will eat in m jumps?

An example: frog feeding

Starting from position x, what is the expected number of flies the frog eats in m jumps?

1. Introduce two random variables:
   - X_i: position of the frog with i jumps left.
   - F_i: number of flies that the frog eats in the last i jumps.

   Goal: compute $\mathbb{E}[F_m | X_m = x]$

2. Use law of total expectation (marginalize over the next position):

$$\mathbb{E}[F_i | X_i = p] = \sum_q \mathbb{E}[F_i | X_{i-1} = q, X_i = p] \mathbb{P}[X_{i-1} = q | X_i = p]$$

$$= \underbrace{\text{flies[p]}}_{\substack{\text{eat the flies} \\ \text{available here}}} + \underbrace{\text{prob[p]}\mathbb{E}[F_{i-1} | X_{i-1} = p-1]}_{\substack{\text{go to the left} \\ \text{and continue eating flies}}} + \underbrace{(1 - \text{prob[p]})\mathbb{E}[F_{i-1} | X_{i-1} = p+1]}_{\substack{\text{go to the right} \\ \text{and continue eating flies}}}$$

3. Base cases

$\mathbb{E}[F_0 | X_0 = p] = \text{flies[p]}$ eat and stop there

$\mathbb{E}[F_i | X_i = -1 \text{ or } n] = 0$ cannot eat flies after falling in the water

An example: frog feeding

$$\mathbb{E}[F_i | X_i = p] = \text{ flies[p]} + \text{prob[p]}\mathbb{E}[F_{i-1} | X_{i-1} = p - 1] + (1 - \text{prob[p]})\mathbb{E}[F_{i-1} | X_{i-1} = p + 1]$$

$$\mathbb{E}[F_m | X_m = x]$$

$$\mathbb{E}[F_{m-1} | X_{m-1} = x - 1] \qquad \mathbb{E}[F_{m-1} | X_{m-1} = x + 1]$$

$$\mathbb{E}[F_{m-2} | X_{m-2} = x - 2] \qquad \mathbb{E}[F_{m-2} | X_{m-2} = x] \qquad \mathbb{E}[F_{m-2} | X_{m-2} = x + 2]$$

Reuses the same computations!
→ **use DP**

An example: frog feeding

Pseudo-code

```
DP[pos][jump] = -1.  for all pos, jump              // default value for 𝔼[F_jump|X_jump = pos]

solve(pos, jump) {
    if (jump == 0) return flies[pos]                 // no jumps remaining
    if (pos < 0 || pos > n-1) return 0.              // frog falls in the water
    if (DP[pos][jump] != -1.) return DP[pos][jump]   // reuse previously done computation
    else {
       DP[pos][jump] = flies[pos]
            + prob[pos] * solve(pos-1, jump-1)
            + (1-prob[pos]) * solve(pos+1, jump-1)   // use recurrence formula
       return DP[pos][jump]
    }
}

solve(x, m)
```

An example: frog feeding

Pseudo-code

```
DP[pos][jump] = -1.  for all pos, jump

solve(pos, jump) {
    if (jump == 0) return flies[pos]
    if (pos < 0 || pos > n-1) return 0.
    if (DP[pos][jump] != -1.) return DP[pos][jump]
    else {
        DP[pos][jump] = flies[pos]
             + prob[pos] * solve(pos-1, jump-1)
             + (1-prob[pos]) * solve(pos+1, jump-1)
        return DP[pos][jump]
    }
}

solve(x, m)
```

Init DP table

Base cases

Check for memory

Otherwise compute value and store it

General structure for DP algorithms

Complexity: O(mn)

Take-home messages

- DP naturally arises in probablistic problems
  - Substructure naturally arises + overlapping subproblems
- Law of total probability / expectation are powerful tools
- Use `double` when implementing (to avoid overflows and underflows)
- Be careful with initial values of the DP table
- DP can be equally solved in bottom-up or top-down way