Thinking Recursively

Part III

A Quick Word of Thanks!

Subsets

- Given set S, a **subset** of S is a set formed by choosing some number of elements from S.
- Examples:
  - {0, 1, 2} is a subset of {0, 1, 2, 3, 4, 5}
  - {dikdik, ibex} is a subset of {dikdik, ibex}
  - { A, G, C, T } is a subset of { A, B, C, D, …, Z }
  - $\{ \} \subseteq \{a, b, c\}$
  - $\{ \} \subseteq \{ \}$

Generating Subsets

- **Base Case:**
  - The only subset of the empty set is the empty set.
- **Recursive Step:**
  - Fix some element x of the set.
  - Generate all subsets of the set formed by removing x from the main set.
  - These subsets are subsets of the original set.
  - All of the sets formed by adding x into those subsets are subsets of the original set.

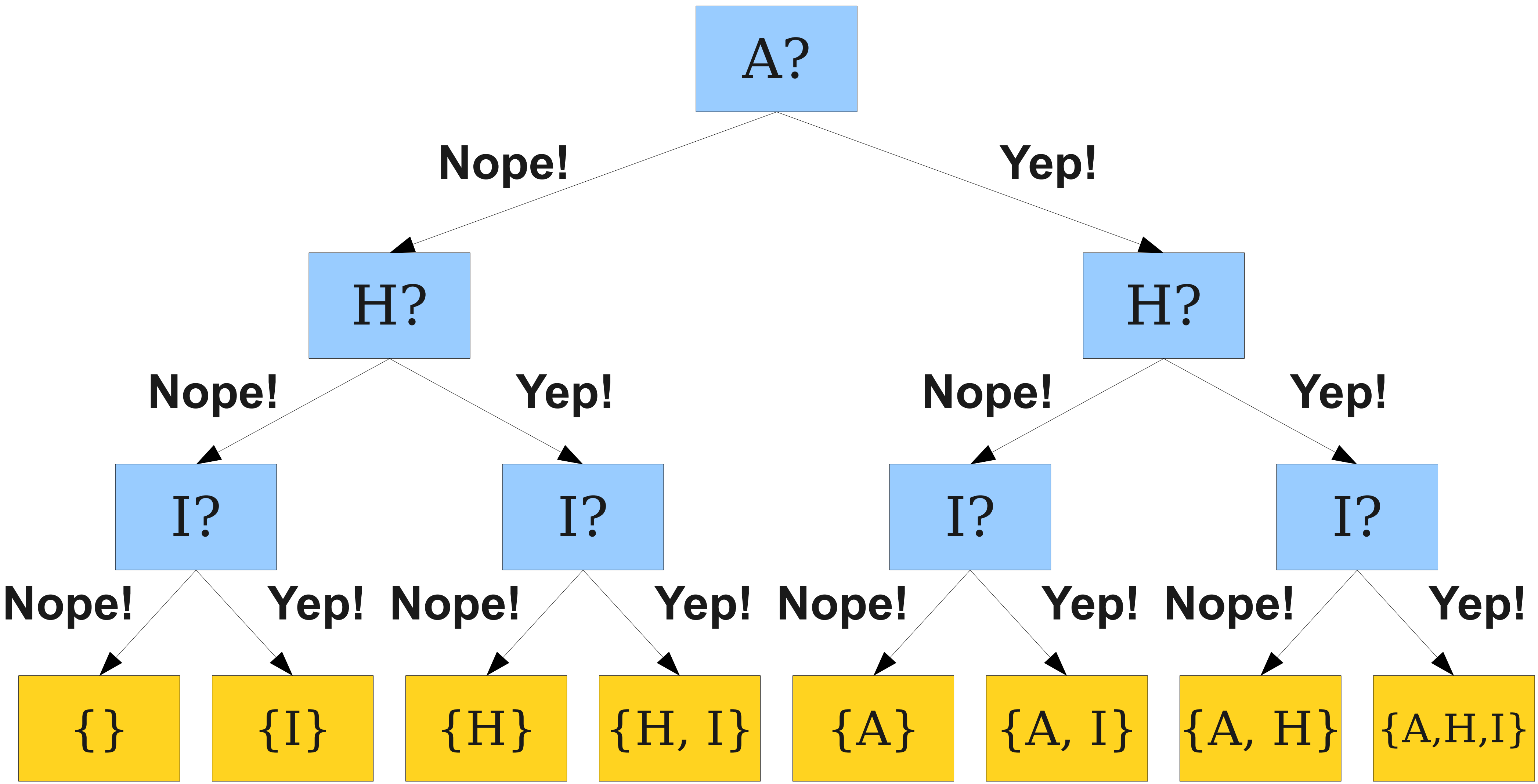
Reducing Memory Usage

- In many cases, we need to perform some operation on each subset, but don't need to actually store those subsets.
- **Idea:** Generate each subset, process it, and then discard it.
- **Question:** How do we do this?

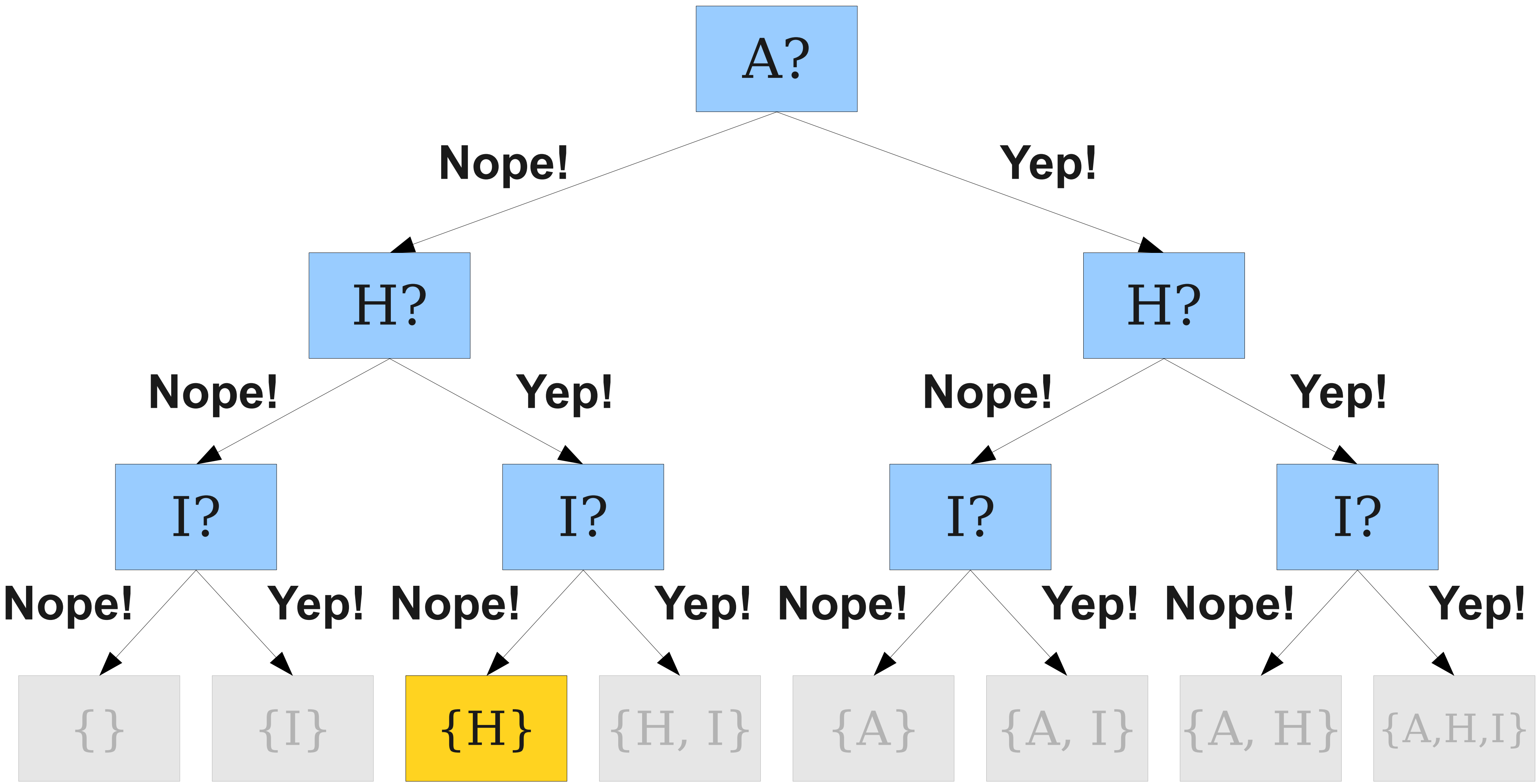
A Decision Tree

{ }	{I}	{H}	{H, I}	{A}	{A, I}	{A, H}	{A,H,I}

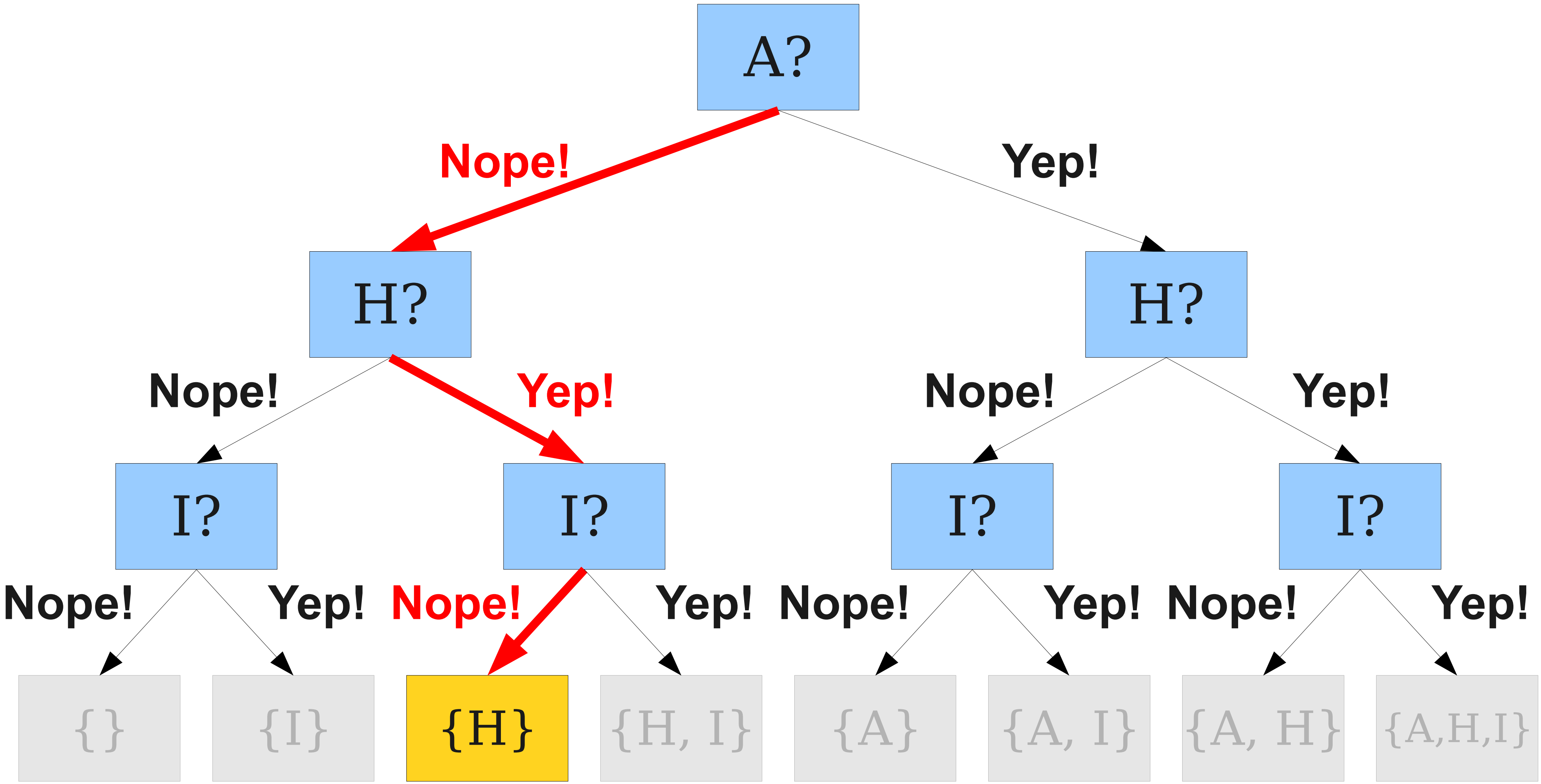
A Decision Tree



A Decision Tree



A Decision Tree



Recursively Exploring Options

- Our recursive function needs to keep track of
  - What choices we've made so far, and
  - What choices we still need to make.
- **Base Case:**
  - If there are no choices left, output the set we formed from the choices we made.
- **Recursive Step:**
  - Find the next choice to make.
  - For each possible choice, recursively explore all options formed from making that choice.

Permutations

- A **permutation** of a sequence is a sequence with the same elements, though possibly in a different order.

Permutations

- A **permutation** of a sequence is a sequence with the same elements, though possibly in a different order.

Permutations

- A **permutation** of a sequence is a sequence with the same elements, though possibly in a different order.
- For example:
  - E Pluribus Unum
  - E Unum Pluribus
  - Pluribus E Unum
  - Pluribus Unum E
  - Unum E Pluribus
  - Unum Pluribus E

Listing all Permutations

- Like subsets, permutations are an important structure in programming.
- Listing all permutations is useful for answering questions like these:
    - What is the best order in which to perform a series of tasks?
    - What possible DNA strands can be made by assembling smaller fragments together?

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4	x_2	x_1	x_3	x_4	x_3	x_1	x_2	x_4	x_4	x_1	x_2	x_3
x_1	x_2	x_4	x_3	x_2	x_1	x_4	x_3	x_3	x_1	x_4	x_2	x_4	x_1	x_3	x_2
x_1	x_3	x_2	x_4	x_2	x_3	x_1	x_4	x_3	x_2	x_1	x_4	x_4	x_2	x_1	x_3
x_1	x_3	x_4	x_2	x_2	x_3	x_4	x_1	x_3	x_2	x_4	x_1	x_4	x_2	x_3	x_1
x_1	x_4	x_2	x_3	x_2	x_4	x_1	x_3	x_3	x_4	x_1	x_2	x_4	x_3	x_1	x_2
x_1	x_4	x_3	x_2	x_2	x_4	x_3	x_1	x_3	x_4	x_2	x_1	x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

- **Base Case**:
  - If the string is empty, there is just one permutation – that string itself.
- **Recursive Step**:
  - For each character in the string:
    - Remove that character.
    - Permute the rest of the string.
    - Add that character back in.

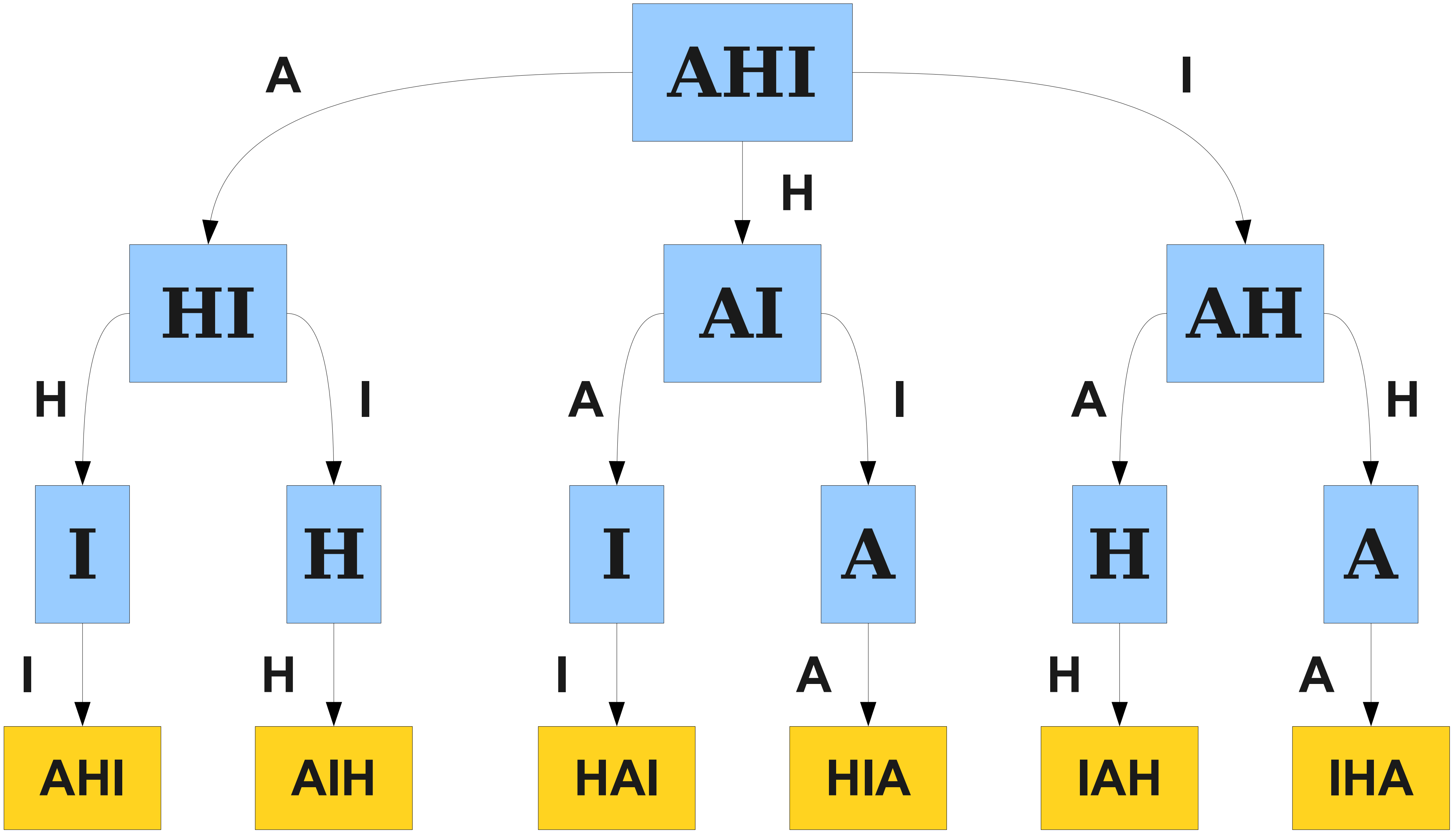
Memory Usage… Again

- How many permutations are there of an n-element sequence?
- **Answer**: $n \times (n - 1) \times \ldots \times 2 \times 1 = \boldsymbol{n!}$
- Storing all permutations of n elements uses at least $n!$ memory.
- If $n = 13$, $n! = 6{,}227{,}020{,}800$. We would almost certainly run out of memory trying to store all permutations of a 13-element sequence in memory.

Reducing Memory Usage

- As before, what if we just need to perform some operation on each permutation, rather than storing all of them?
- **Idea:** Generate each permutation, process it, then discard it.

A Decision Tree



A Second Recursive Function

- Our recursive function needs to keep track of
  - What choices we've made so far, and
  - What choices we still need to make.
- **Base Case:**
  - If there are no choices left, output the permutation we formed from the choices made.
- **Recursive Step:**
  - Find the next choice to make.
  - For each possible choice, recursively explore all options formed from making that choice.