Thinking Recursively

Part III

Announcements

- Assignment 2 Due right now
- Assignment 3: Recursion!
  - Requires writing *very* little code
  - Forces you to develop a good understanding of recursion
  - Takes a lot of time for many people, so please start early!
- I'm frequently available outside of my office hours in the afternoon. Email me if you'd like to chat about the course and/or assignments.

From Last Time...

Subsets

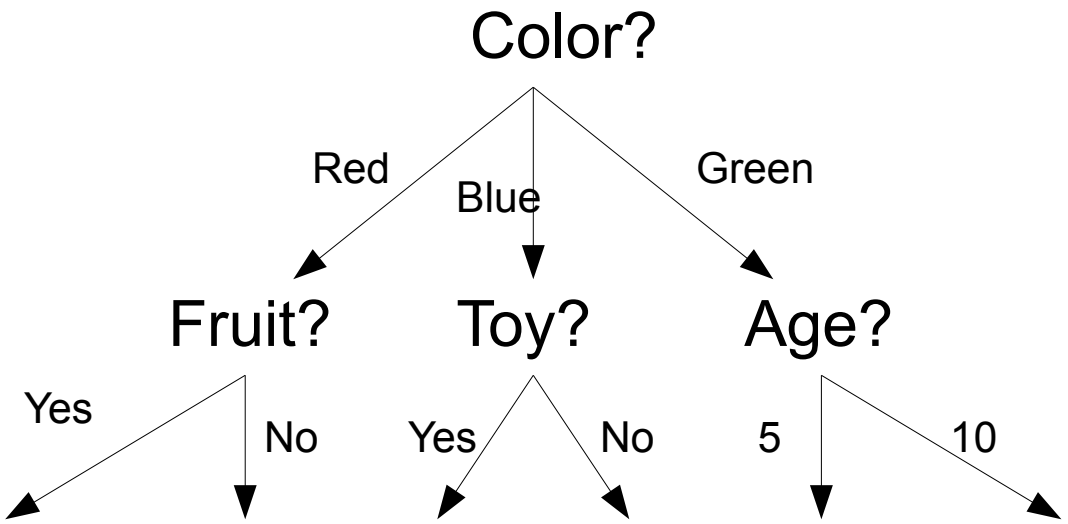
- Given set S, a **subset** of S is a set formed by choosing some number of elements from S.
  - Note: A set can only hold a single “copy” of an element.
    - e.g The set {0, 1, 1} is identical to the set {0, 1}
- Examples:
  - {0, 1, 4} is a subset of {0, 1, 2, 3, 4, 5}
  - {dikdik, ibex} is a subset of {dikdik, ibex}
  - { } ⊆ {a, b, c}
  - { } ⊆ { }

Generating Subsets

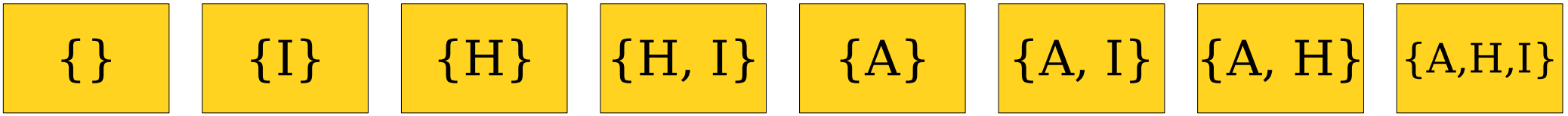
- **Base Case:**
  - The only subset of the empty set is the empty set.
- **Recursive Step:**
  - Fix some element x of the set.
  - Generate all subsets of the set formed by removing x from the main set.
  - These subsets are subsets of the original set.
  - All of the sets formed by adding x into those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

Decision Problems

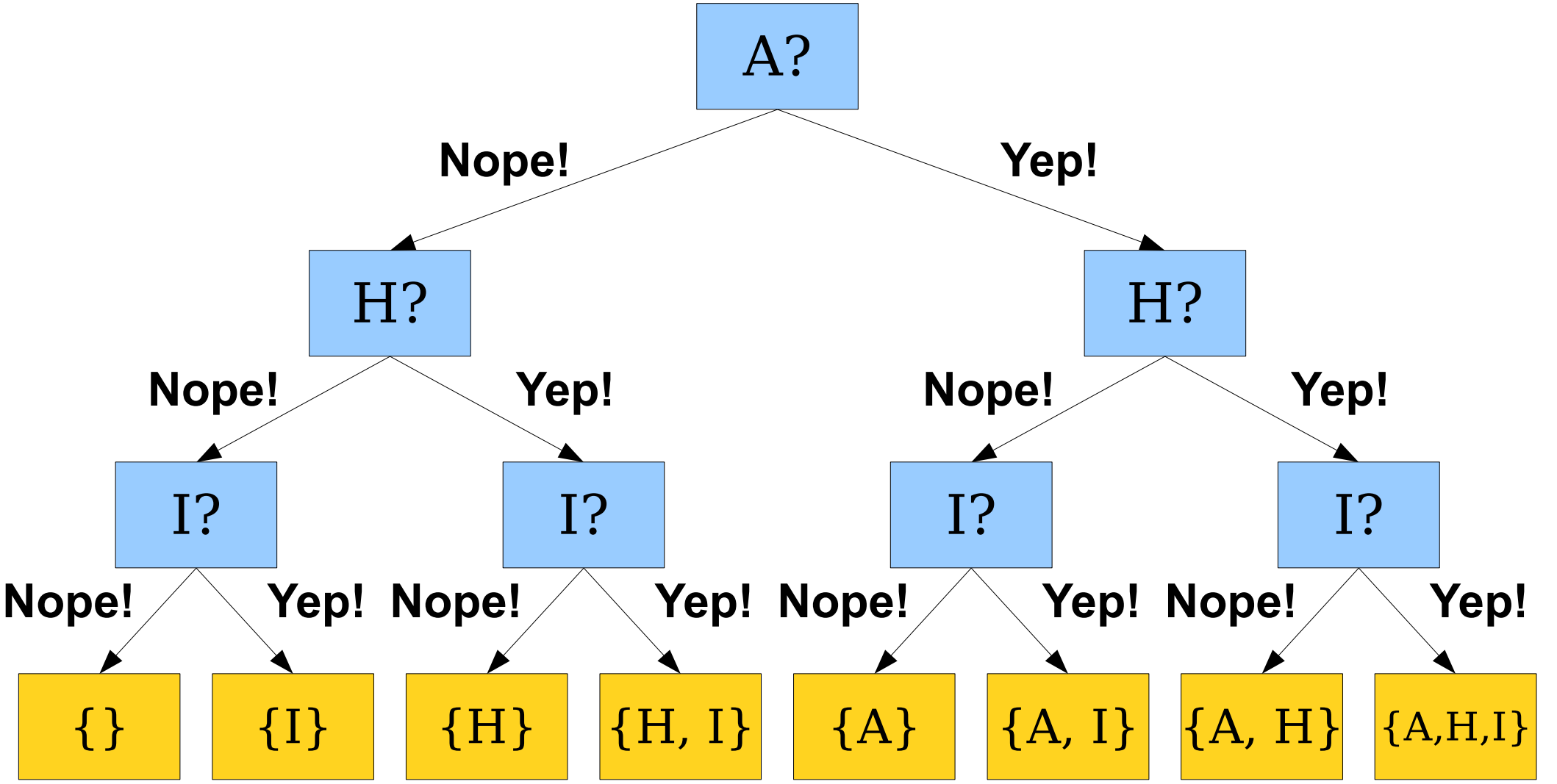
- It is useful to think of the process of generating permutations as a **decision tree**.
  - A decision tree is a structure which models a series of choices



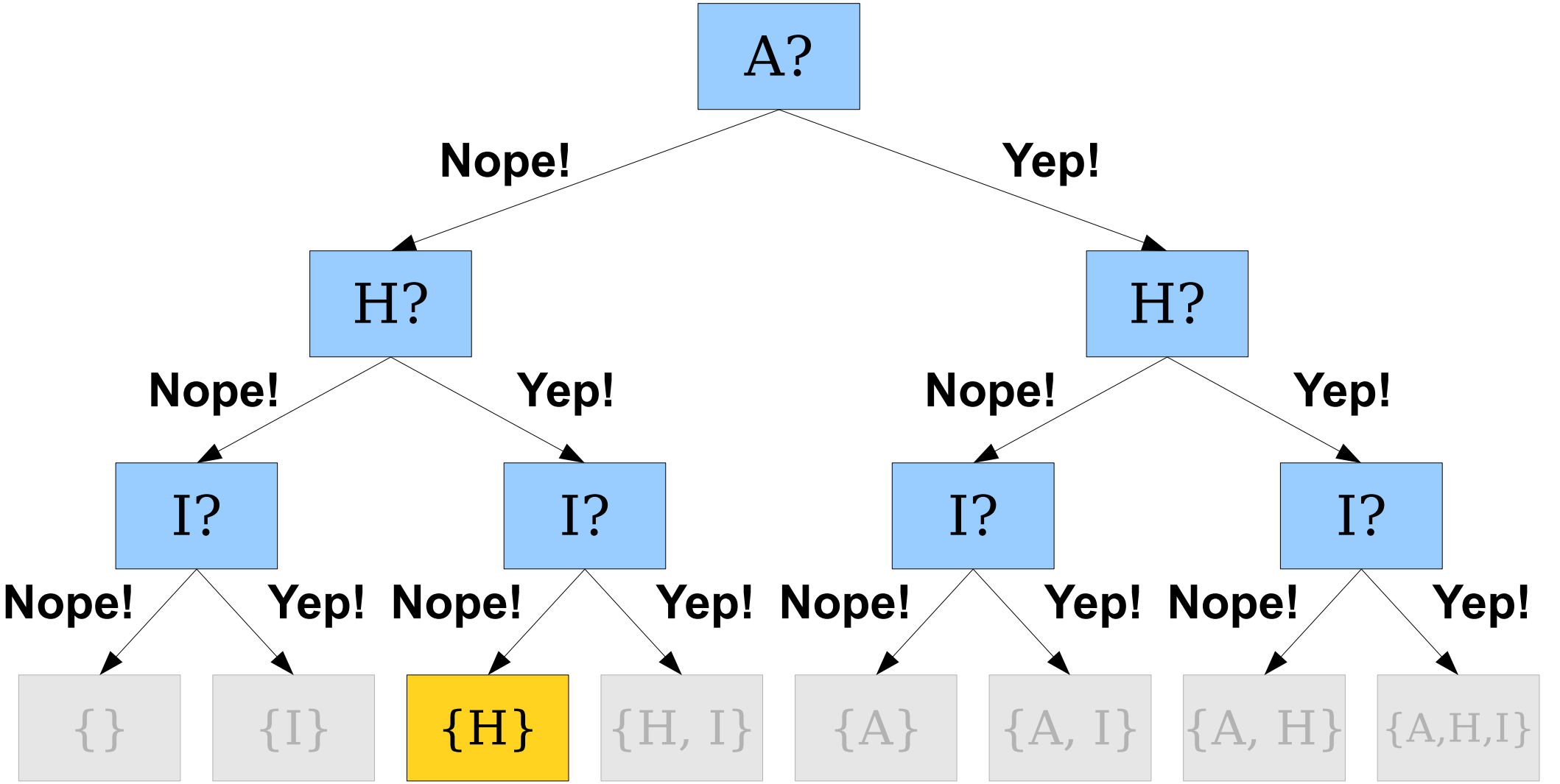
A Decision Tree



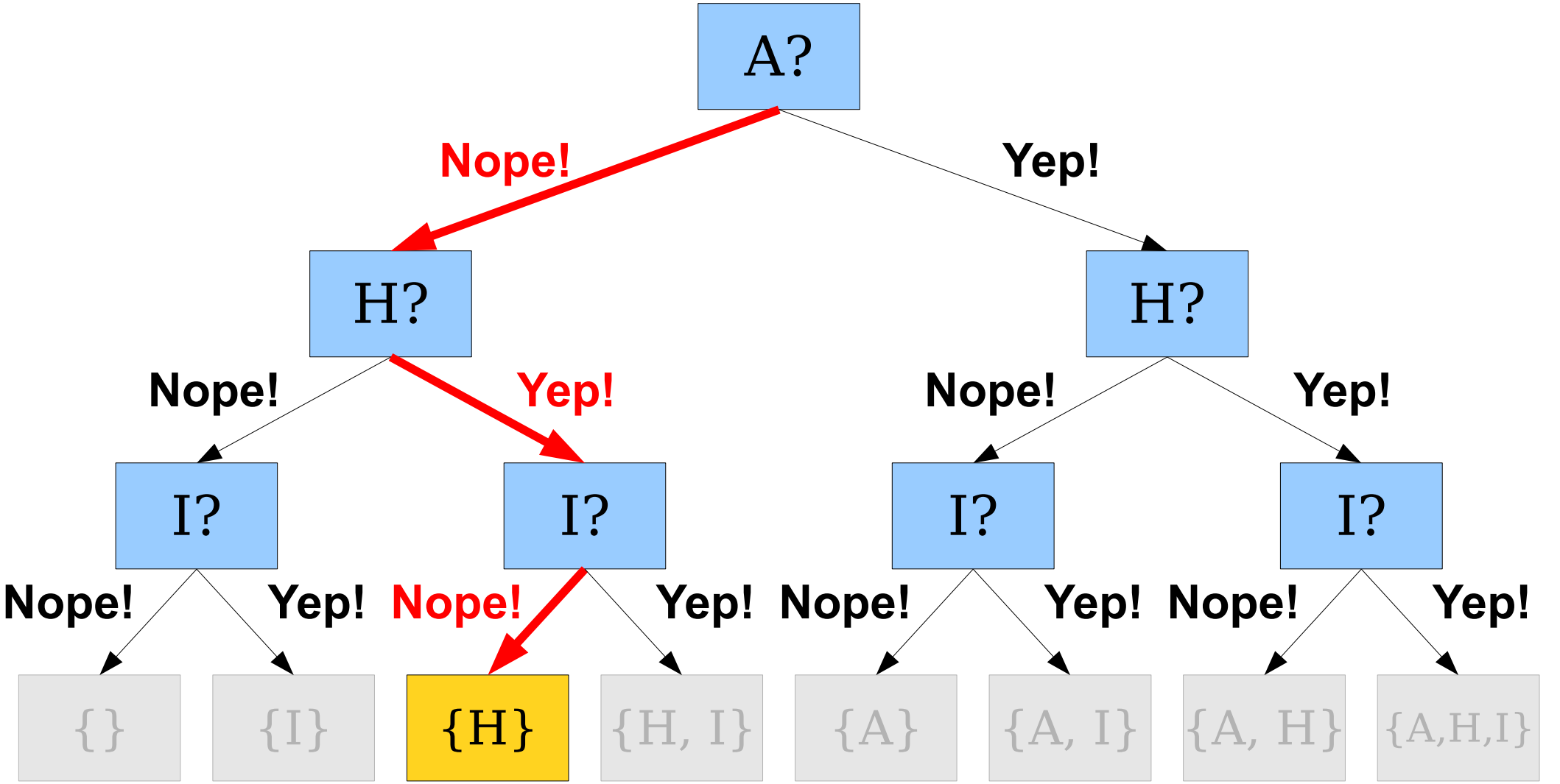
A Decision Tree



A Decision Tree



A Decision Tree



Reducing Memory Usage

- Problem: Saving every subset in a Vector will use a ton of memory.
- Solution: 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?

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.

Visiting Subsets
(Decision Tree)

New Stuff...

Traveling Salesperson

- Let's say you're a traveling salesperson and you have a set of cities you need to visit.
  - You need to visit each city exactly once.
- To do this you need to schedule a series of flights throughout the world, where each flight has a cost.
  - Assume that there is a flight from each city to every other city
- You get to pick the order in which you visit cities.
- Goal: Construct an itinerary that minimizes how much you spend on flights.

Traveling Salesperson

https://www.google.com/maps/vt/data=VLHX1wd2Cgu8wR6jwyh-km8JBWAkEzU4,2bUCUBVs3YYr-KB4ccFl-1Q1nWYcyKzmW0Ggf8ar4OOyEuuN9txRnTiKzIvmH6qy6B4vSoZvopndG7VjMIsOIDayhdkqKblOykP1wZYm9RcF8-Y6pkecPwDi3xc98B3gNGLchfR7xnPKzCGEmRocrv9OczmELzORvRseZHLyjWOvL0GzUeg0WFJGA4Y

Traveling Salesperson

https://www.google.com/maps/vt/data=VLHX1wd2Cgu8wR6jwyh-km8JBWAkEzU4,2bUCUBVs3YYr-KB4ccFl-1Q1nWYcyKzmW0Ggf8ar4OOyEuuN9txRnTiKzIvmH6qy6B4vSoZvopndG7VjMIsOIDayhdkqKblOykP1wZYm9RcF8-Y6pkecPwDi3xc98B3gNGLchfR7xnPKzCGEmRocrv9OczmELzORvRseZHLyjWOvL0GzUeg0WFJGA4Y

Traveling Salesperson

https://www.google.com/maps/vt/data=VLHX1wd2Cgu8wR6jwyh-km8JBWAkEzU4,2bUCUBVs3YYr-KB4ccFl-1Q1nWYcyKzmW0Ggf8ar4OOyEuuN9txRnTiKzIvmH6qy6B4vSoZvopndG7VjMIsOIDayhdkqKblOykP1wZYm9RcF8-Y6pkecPwDi3xc98B3gNGLchfR7xnPKzCGEmRocrv9OczmELzORvRseZHLyjWOvL0GzUeg0WFJGA4Y

Traveling Salesperson

https://www.google.com/maps/vt/data=VLHX1wd2Cgu8wR6jwyh-km8JBWAkEzU4,2bUCUBVs3YYr-KB4ccFl-1Q1nWYcyKzmW0Ggf8ar4OOyEuuN9txRnTiKzIvmH6qy6B4vSoZvopndG7VjMIsOIDayhdkqKblOykP1wZYm9RcF8-Y6pkecPwDi3xc98B3gNGLchfR7xnPKzCGEmRocrv9OczmELzORvRseZHLyjWOvL0GzUeg0WFJGA4Y

Traveling Salesperson

- No known efficient algorithms for solving this.
  - MANY approximation algorithms
  - Dynamic programming algorithm (beyond the scope of this course)
- Algorithm: Consider every possible ordering of cities and select the ordering that is the cheapest.

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

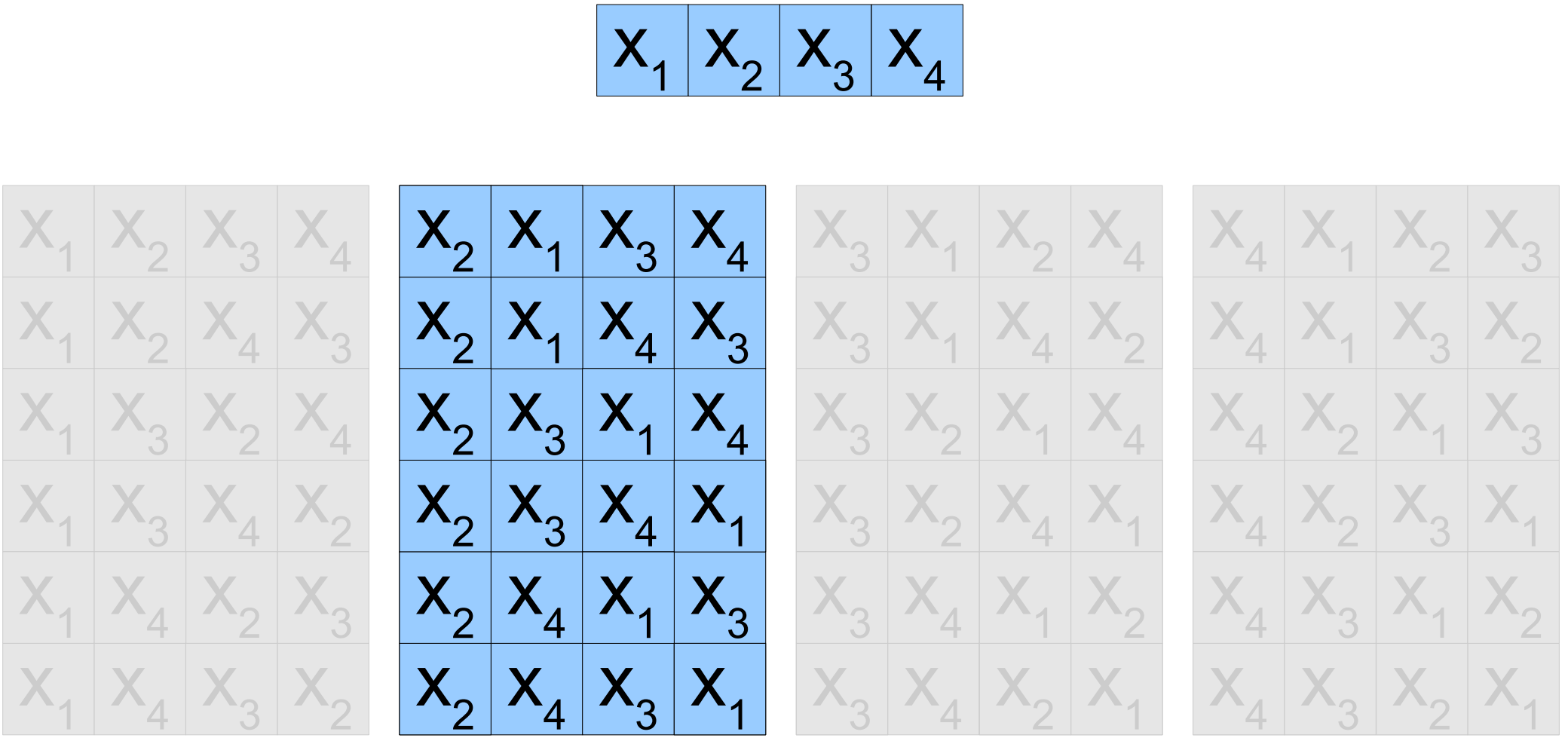
x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

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



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_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

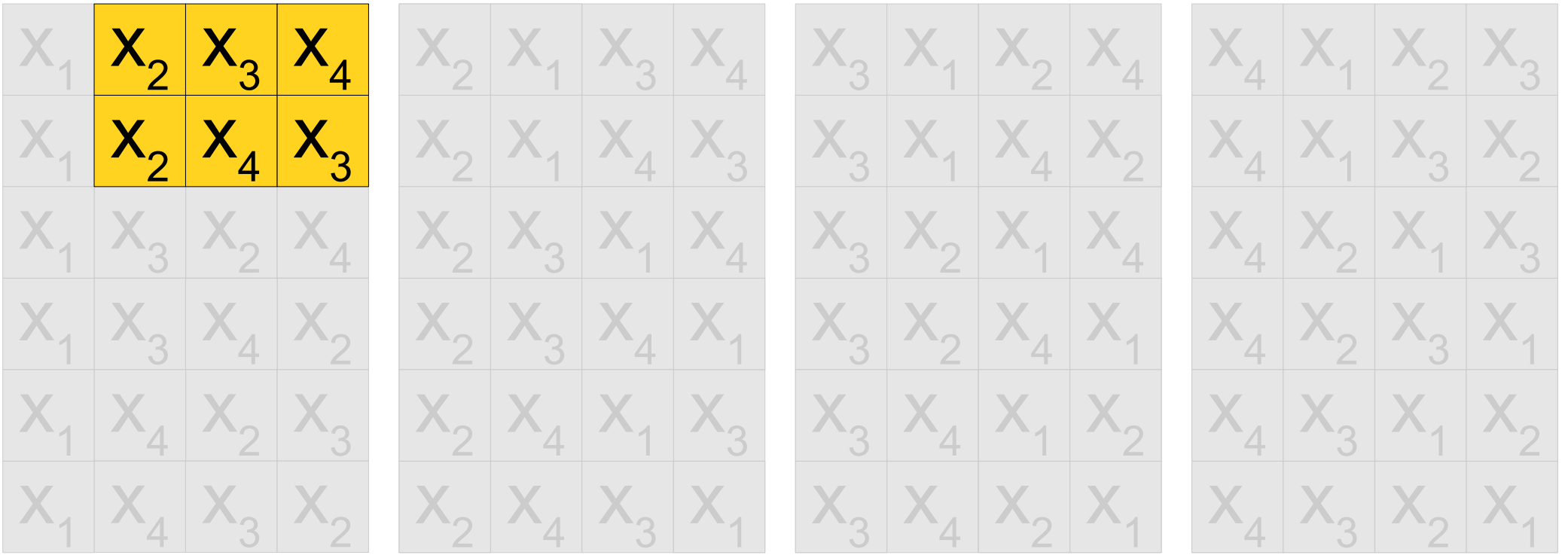
x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

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

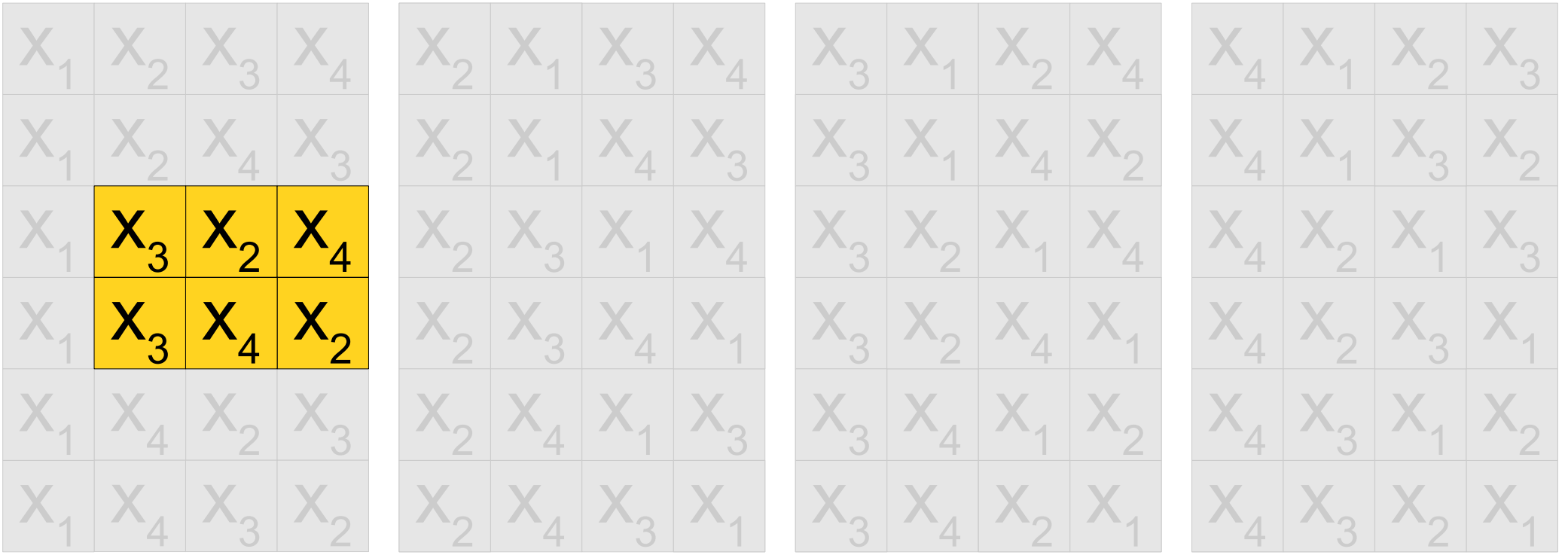
Generating Permutations

x_1	x_2	x_3	x_4



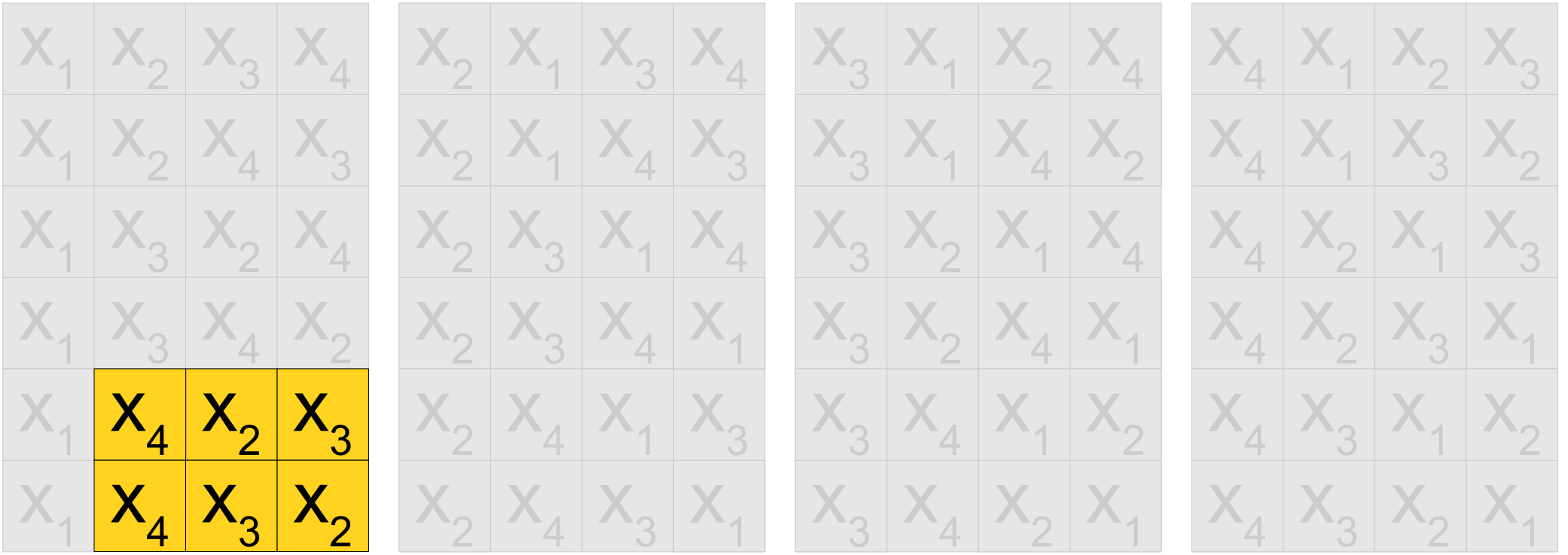
Generating Permutations

x_1	x_2	x_3	x_4



Generating Permutations

x_1	x_2	x_3	x_4



`permutations`
(Pseudocode)

`permutations.cpp`
(Computer)

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

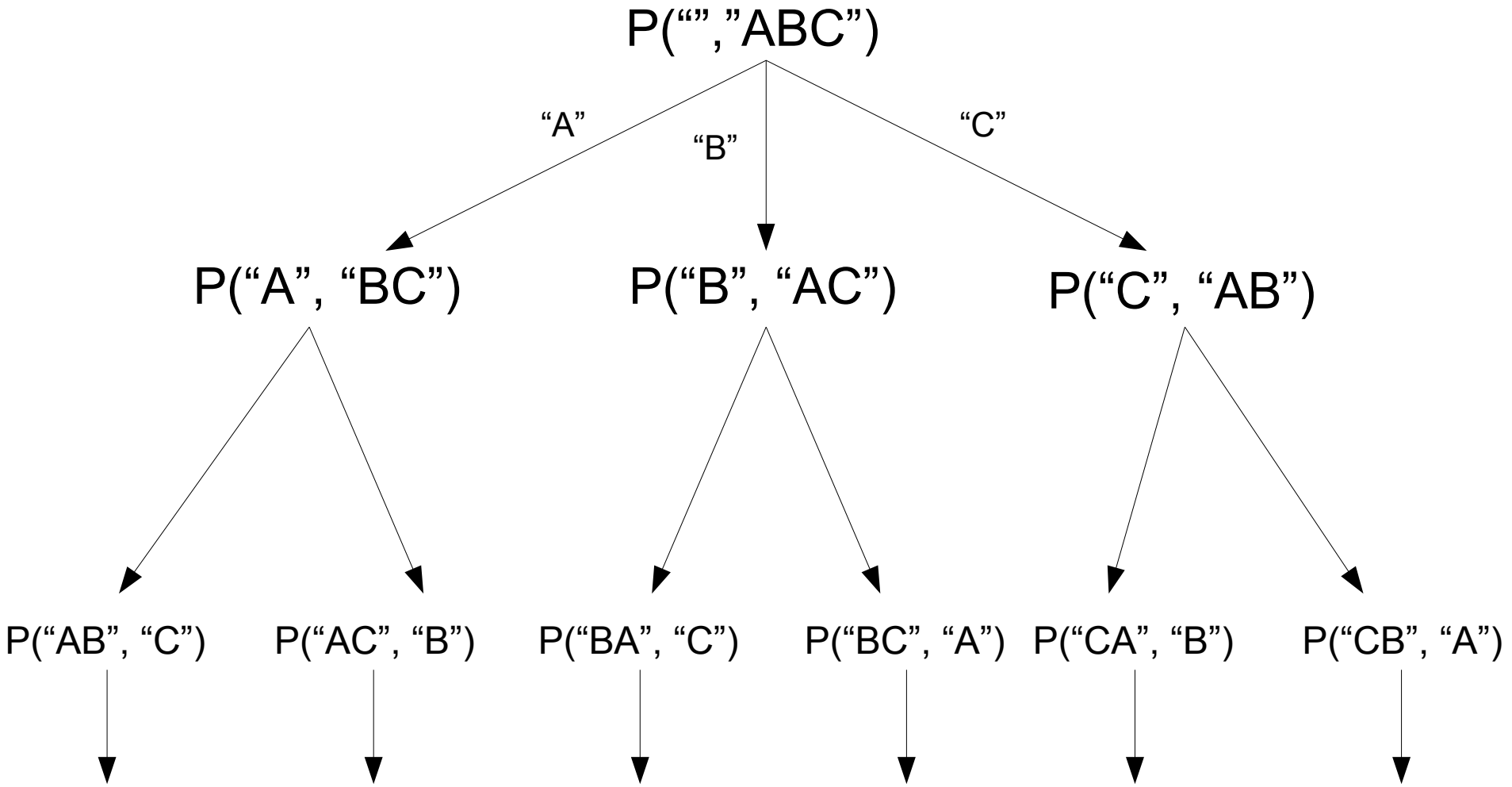
- As before, what if we just need to perform some operation on each permutation, rather than storing all of them?
- **Idea:** Generate each permutation, process it, then discard it.

Permutations Decision Tree

(Board)

Permutations: Decision Tree

P(`string soFar`, `string remaining`)



`visitPermutations` (Pseudocode)

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.

`visitPermutations` (Computer)

Subsets vs. Permutations

- Notice that the codes for generating subsets and permutations are *very* similar.
- The only difference is a small change in the recursive decomposition.
  - **Subsets:** Pick an element, recurse with and without that element.
  - **Permutations:** For each element, recurse starting with that element

Some exciting news...

Most of the “interesting” exhaustive recursive programs can be reduced to either generating subsets or permutations

14 22 13 25 30 11 9

Maximize what's left in here.

14 22 13 25 30 11 9

Maximize what's left in here.

Subsets and Permutations

- Optimizing over cell phone towers can be thought of as generating all subsets with the constraint and no two towers can be adjacent.
- Similar reductions can be made for many problems.
  - Including problems in assignment 3 =)

This means if you can generate subsets and permutations, then you can do most of the “interesting” stuff you can with exhaustive recursion!

A good first step to solving an exhaustive recursive problem is first determine if it's related to generating subsets or permutations.

Next Time

- **Exhaustive Recursion III**
  - One last recursive structure
- **Recursive Backtracking**