Functional Programming Final Review

CS16: Introduction to Data Structures & Algorithms

Spring 2020

Functional Programming Paradigm

- A style of building the structure and elements of computer programs that treats computation as the evaluation of *mathematical functions*.
- Programs written in this paradigm rely on smaller methods that do one part of a larger task. The results of these methods are combined using function compositions to accomplish the overall task.

Approaches

- How do we decide to use map vs. reduce?
  - Map creates a one to one mapping - we use it for (sub)-problems that involve doing the same thing to multiple elements.
    - Length will stay the same!
  - Reduce can be used to "summarize" a list, or create a new (smaller or larger) list
    - Map can be implemented with reduce, but not vice-versa!

Using Map

- How to choose the function?
  - What do you want to have happen to *each* element in the input list?
  - Other variables needed for the function can be created outside of the map call if needed!
- Quick Tip
  - Built-ins/existing functions do not need to have their arguments written out.

```
map(lambda x: f(x), input_list) => map(f, input_list)
```

Using Reduce 1/2

- How to choose the binary function?
  - Takes in the acc and each successive element in the input list.
  - Think about how to break down your task!
    - the max of an entire list -> the max of two integers
    - remove all successive duplicates -> check if 2 elements are equal
  - Remember ternary syntax!

```
a if condition else b
```

Using Reduce 2/2

- How to choose the accumulator?
  - Needs to be of the type that you are returning
  - What should your operation return on the empty list?

List Syntax

- [x]
  - makes a list out of element x
- my_list[-1]
  - returns the last element in my_list
- my_list + [x]
  - returns a new list with x at the end, and does not modify the original list.
  - don’t use append! this modifies the original list and returns nothing.

Practice Problems

- Write a function that will turn a list of nouns into adverbs. (ex: loud -> loudly)
- Write a function that sums the total length of a list of strings. (ex: [“hi”, “cs16”] -> 6)
- Write a function that counts the number of times the string “dog” appears in a list of strings.
- Write a function that removes numbers less than 10 from a list of ints.

Practice Problem Answers

- map(lambda el: el+’’ly’’, input_list)
- reduce(lambda acc, el: acc+el,
  map(len, input_list), 0)
- reduce(lambda acc, el: acc+1 if el == "dog" else acc, input_list, 0))
- reduce(lambda acc, el: acc+[el] if el > 10 else acc, input_list, [])

Dynamic Programming Final Review

CS16: Introduction to Data Structures & Algorithms
Spring 2020

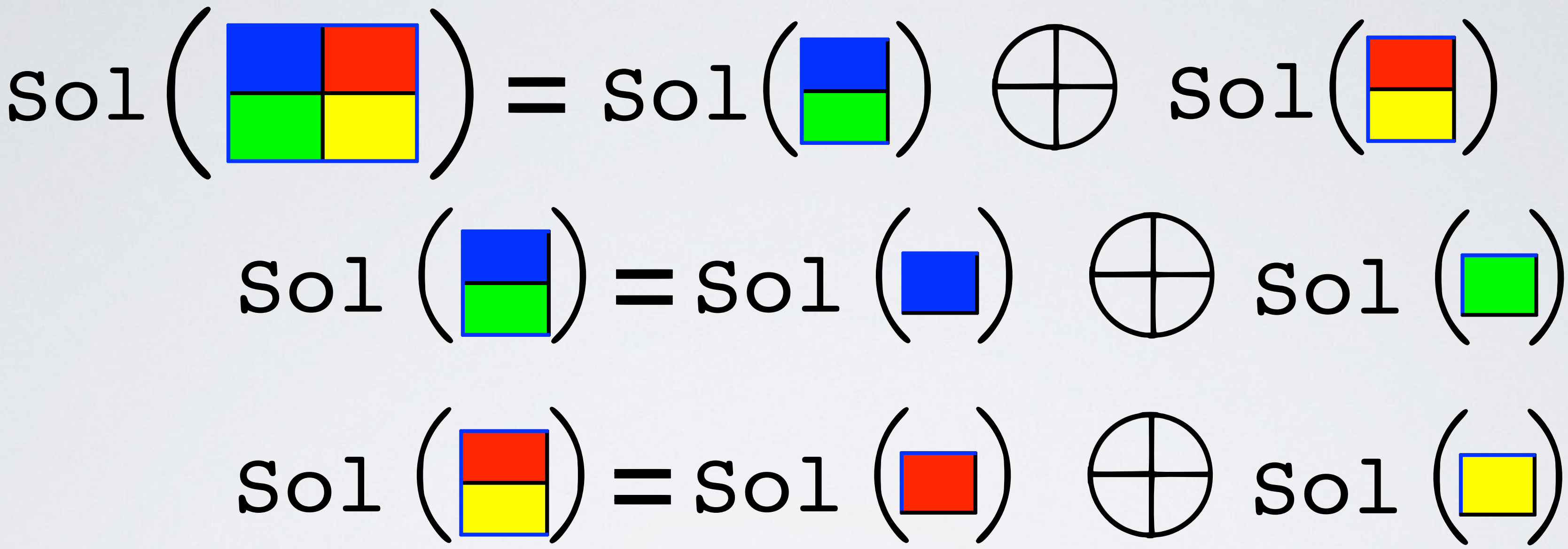
What is Dynamic Programming?

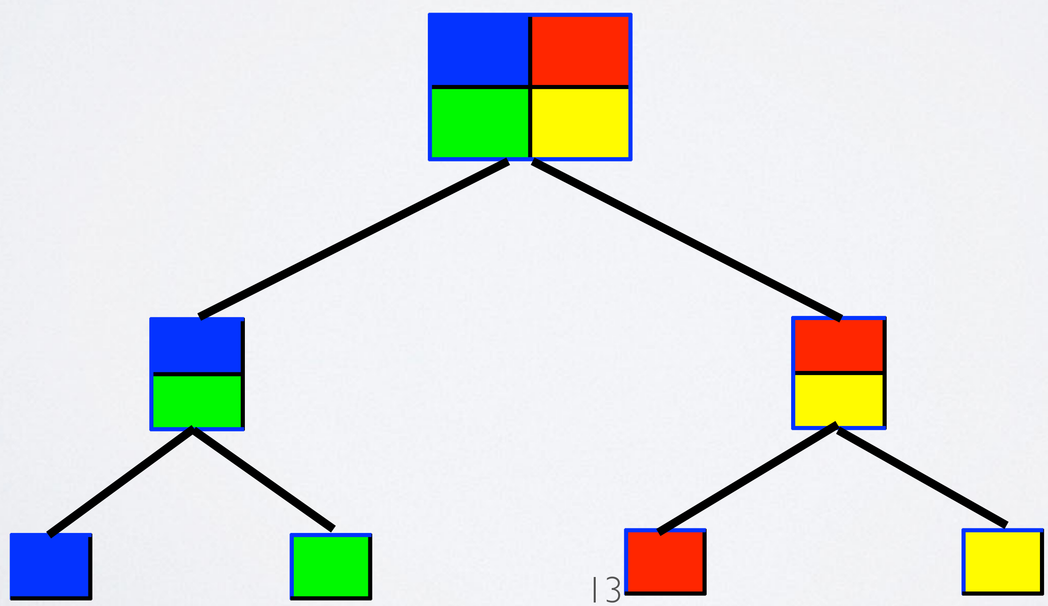
- Algorithm design paradigm/framework
  - Design efficient algorithms for optimization problems
- Optimization problems
  - "find the **best** solution to problem `X`"
  - "what is the **shortest** path between `u` and `v` in `G`"
  - "what is the **minimum** spanning tree in `G`"
- Can also be used for non-optimization problems

When is Dynamic Programming Applicable?

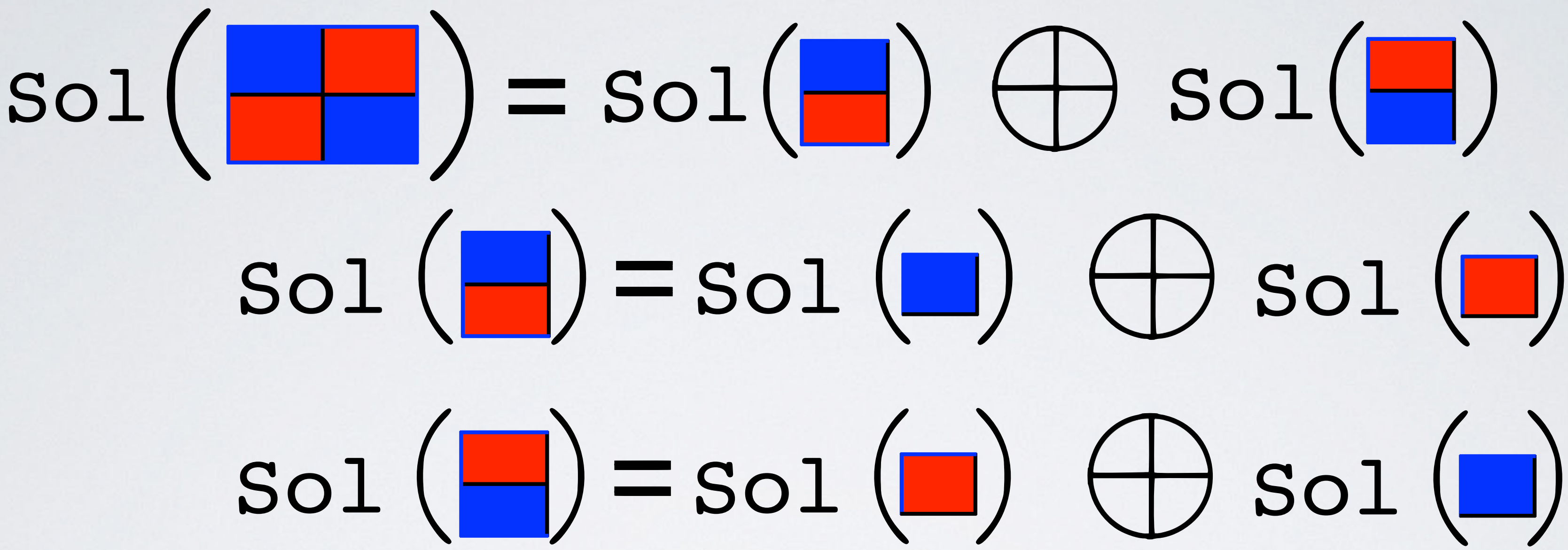
- Condition **#1**: sub-problems
  - The problem can be solved recursively
  - Can be solved by solving sub-problems
- Condition **#2**: overlapping sub-problems
  - Same sub-problems need to be solved many times

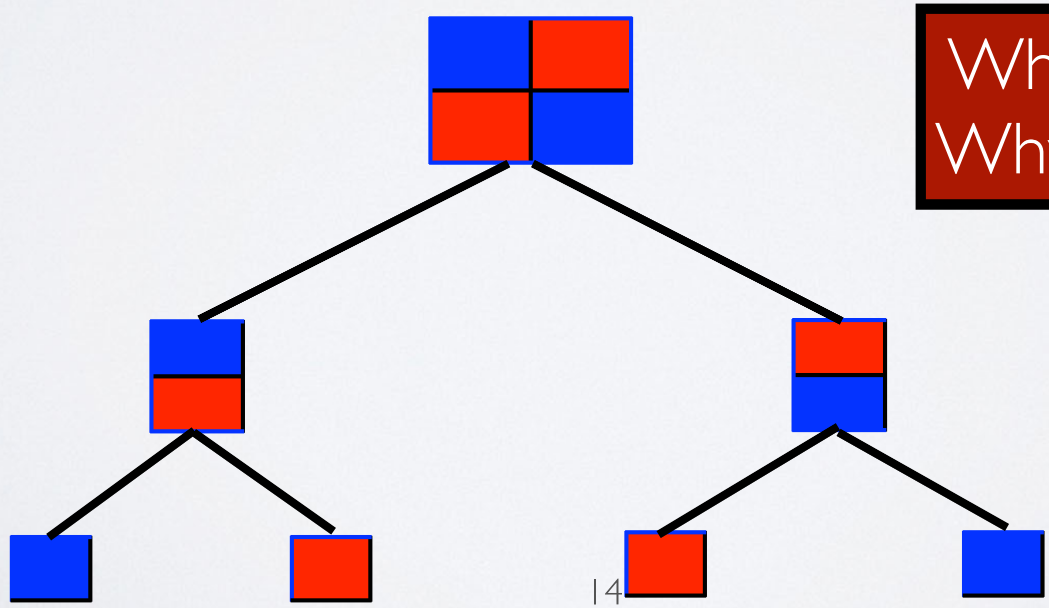
Sub-Problems





Overlapping Sub-Problems





Why solve red twice?
Why solve blue twice?

When is Dynamic Programming Applicable?

- Core idea
  - Decompose problem into its sub-problems
  - and if sub-problems are overlapping then
  - solve each sub-problem once and store the solution
  - use stored solution when you need to solve sub-problem again

Steps to Solving a Problem w/ DP

- What are the **sub-problems**?
- What is the "**magic**" step?
  - Given solution to a sub-problem…
  - …how do I *combine* them to get solution to the problem?
- Which **(topological) order** on sub-problems can I use?
  - so that solutions to sub-problems available before I need them
- Design iterative **algorithm**
  - that solves sub-problems in order and stores their solution

Shortest Path in Layered Directed Graph

- Layered
  - edge **`(x,y)`** only if **`x<y`**
- Negative & positive weights