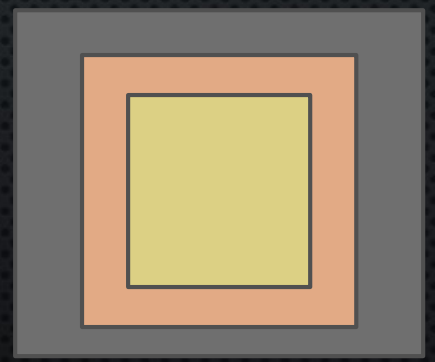
Dynamic programming, Greedy algorithms

Recitation week 8

DYNAMIC PROGRAMMING

- Recall that dynamic programming is just "recursion with memory"
- Dynamic programming techniques work when:
  - The problem exhibits *optimal substructure*
  - The problem has *overlapping subproblems*
- Given a set of *n* different types of boxes (with repetition) each represented as a (ℓ, w, h) triple, find the largest number of boxes you can put in a stack. A box can be stacked on top of another if its footprint is strictly smaller than the one below in both dimensions.

(2,3,3) (2,3,2) (1,1,2)



DYNAMIC PROGRAMMING – BOX STACKING

- Given a set of *n* different types of boxes (with repetition) each represented as a (ℓ, w, h) triple, find the largest number of boxes you can put in a stack. A box can be stacked on top of another if its footprint is strictly smaller than the one below in both dimensions.
- First steps:
  - How would we solve this recursively?
  - Does this relate to other problems we've seen before?
- This problem can be thought as a variation of the *longest increasing subsequence* problem we've seen before. To see this, first generate the *3n* different types of boxes (by considering all orientations of the boxes. Now, sort these *3n* elements by area of footprint. Then, apply the same algorithm we discussed for LIS but this time an element is compatible with a previous one if it is smaller in both dimensions.

GREEDY ALGORITHMS

- The foundation of greedy algorithm design is finding problems which exhibit:
  - Greedy choice property
  - Optimal substructure
- When working with greedy algorithms, the question becomes *myopic* – we can make all of our decisions by blinding picking the best available (similar to local search)
- Depending on the problem, greedy algorithms can give the optimal solution or have some *approximation factor* – how well it does compared to the optimal

GREEDY ALGORITHMS – MAKING CHANGE EFFICIENTLY

- Suppose you are working as a cashier and have to make change using quarters, dimes, nickels, and pennies. You want to be as efficient as possible, so you always try to give out the fewest possible coins. Propose a greedy algorithm which given any *n* outputs a multiset of the coins of minimum cardinality such that the sum of the coins is *n*. Then, argue that your algorithm always produces the optimum result.
- Consider the algorithm which greedily adds one of the coins of largest possible value at every iteration. In other words, if $n' \geq 25$, you add a quarter; if $10 \leq n' \leq 24$, you add a dime; if $5 \leq n' \leq 9$ you add a nickel, and if $n' < 5$, you add a penny and update the value of *n'* accordingly.
- To show a greedy algorithm is correct, we must show the *greedy-choice* and *optimal-substructure* properties of the problem

- Optimal substructure: Suppose an optimal solution to the problem of size *n* contains a coin of value *c*. Let the size of the solution be *k*. The optimal solution to the problem of size *n* must contain the optimal solution to the problem of size *n-c* and this new problem must have size *k-1*. If it were smaller, then we could construct a solution to the problem of size *n* by adding the coin of value *c* to it, contradicting the optimality of our original solution.
- Greedy choice: We must show that the optimal solution to a problem of size *n* always contains at least one of the largest coin with size less or equal to *n*. We'll show this using case analysis and contradiction. Suppose not.
  - n < 5: Then c = 1, but we must use only pennies for any *n* smaller than 5.
  - $5 \leq n < 10$: Then c = 5. By supposition, the optimal solution must not contain a nickel, but then there are at least 5 pennies in the optimal solution, which can be replaced by a nickel, making it smaller.
  - $10 \leq n < 25$: Then c = 10. By supposition, the optimal solution must not contain a dime, but then some set of coins adds to 10 cents problems of this size and the coins are, at largest, 2 nickels. Replacing these with a dime would contradict optimality.
  - $25 \leq n$: Then c = 25. By supposition, the optimal solution must not contain a quarter. Suppose there are 3 dimes in the solution, we may replace this with a quarter and a nickel to make the solution smaller. Suppose there are only 2 dimes, then there is at least 1 nickel or 5 pennies, which we can replace by a quarter, contradicting optimality.

CHANGING COINS

- Greedy will not always produce an optimal solution for coins with arbitrary values (even if there is always a penny) and certain values of *n*. Construct an example (with coin denominations and a value of *n*) such that the previously proposed greedy algorithm is not optimal.
- C = {1, 3, 4}, n = 6
- Propose a dynamic programming solution to this problem which always outputs a multi-set of coins of smallest possible size.
- The idea is to solve the problem bottom-up and declare an array of length *n*. For each *i* from 1 to n we calculate the minimum number of coins required by adding 1 to the minimum number of coins required for each of the smaller subproblems (one for each of the different denominations of coins).