CS5800 – ALGORITHMS

MODULE 6. GREEDY ALGORITHMS - I

Lesson 1: Change Making

Ravi Sundaram

Topics

- Greedy Algorithms – what, why & examples
- Change-making
    - Problem
    - Algorithm
    - Proof
    - Efficiency
    - Counter-example
- Greedy Strategy – some tips
- Summary

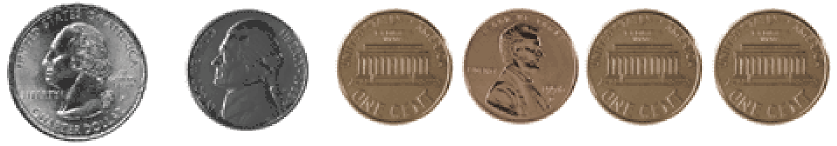
Greedy – what, why & examples

- A greedy algorithm is one that tries to produce an optimal solution by choosing whatever piece of a solution **looks best** in the moment, **without calculating consequences.** The algorithm is myopic – it does not look at the long term.
- The "greed" may be trying to **maximize *or* minimize something** with each decision
- Example greedy rules that we'll talk about in more depth:
  - Schedule the request that **finishes earliest** (Interval Scheduling)
  - Add the **cheapest** edge to our tree (Prim's minimal spanning tree algorithm)
  - Explore the **closest** unexplored vertex (Dijkstra's shortest paths algorithm)

The change-making problem

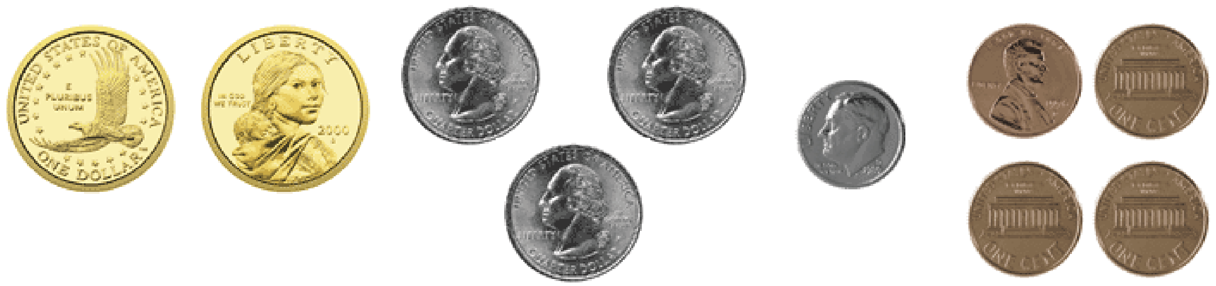
- Goal. Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest number of coins.

- Ex: 34¢.



- Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

- Ex: $2.89.



Greedy Algorithm aka Cashier's Algorithm

- At each iteration, add coin of the largest value that does not take us past the amount to be paid.

```
Sort coins denominations by value: $c_1 < c_2 < … < c_n$.

  coins selected
$S \leftarrow \phi$
while ($x \neq 0$) {
   let k be largest integer such that $c_k \leq x$
   if ($k = 0$)
      return "no solution found"
   $x \leftarrow x - c_k$
   $S \leftarrow S \cup \{k\}$
}
return S
```

Change-making: Analysis of Greedy Algorithm

- Theorem. Greed is optimal for U.S. coinage: 1, 5, 10, 25, 100.
- Pf. (by induction on x)
  - Consider optimal way to change $c_k \leq x < c_{k+1}$: greedy takes coin k.
  - We claim that any optimal solution must also take coin k.
    - if not, it needs enough coins of type $c_1, \ldots, c_{k-1}$ to add up to x
    - table below indicates no optimal solution can do this
  - Problem reduces to coin-changing x - c_k cents, which, by induction, is optimally solved by greedy algorithm. ▪

k	c_k	All optimal solutions must satisfy	Max value of coins 1, 2, …, k-1 in any OPT
1	1	$P \leq 4$	-
2	5	$N \leq 1$	4
3	10	$N + D \leq 2$	4 + 5 = 9
4	25	$Q \leq 3$	20 + 4 = 24
5	100	no limit	75 + 24 = 99

Change-making algorithm: complexity

- Assumption: the denominations are fixed, and the amount is specified in binary using n bits.

- The complexity is O(2n) since the amount can be exponentially large in the number of bits.

Change-making: counterexample

- Observation. Greedy algorithm is sub-optimal for US postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

- Counterexample. 140¢.
  - Greedy: 100, 34, 1, 1, 1, 1, 1, 1.
  - Optimal: 70, 70.

Greedy Strategy: some tips

- Figure out some metric to be greedy about, where it seems like it's *always* best to choose that option.
- Consider plugging in some corner cases to see if it always works
- Your algorithm can consist of **sorting by this metric** ahead of time *or* **putting things in a priority queue** where the keys are the greedy metric values.
    - The latter is better if you would otherwise need to re-sort.
- If there's no obvious **lower bound** your algorithm matches, consider these proof techniques:
    - **Greedy-stays-ahead**: prove by induction that you start "ahead" and remain "ahead" (we will see this argument in Interval Scheduling)
    - **Exchange argument**: Why is it never a bad idea to make a solution look more like your solution?

Summary

- Greedy – algorithmic design paradigm, involves choosing one by one myopically to construct solution
- Typically, many greedy choices; art to select the right one
- Need to prove correctness. Not always guaranteed to work
- Main Takeaway: for some problems building the solution incrementally in myopic fashion works. Short-term greedy is, sometimes, long-term optimal