EECS 281: Data Structures and Algorithms

Brute-force Algorithms &
Greedy Algorithms

Brute-force Algorithms

Def'n: Solves a problem in the most simple, direct, or obvious way

- Not distinguished by structure or form
- Pros
  - Often simple to implement
- Cons
  - May do more work than necessary
  - May be efficient (but typically is not)

Greedy Algorithms

Def'n: Algorithm that makes sequence of decisions, and never reconsiders decisions that have been made

- Pros
  - May run significantly faster than brute-force
- Cons
  - May not lead to correct/optimal solution

Example: Counting Change

Problem Def'n:

- Cashier has collection of 'coins' of various denominations
- Goal is to return a specified sum using the smallest number of coins

Example: Counting Change

Mathematical Def'n:

- n coins:
  $P = \{p_1, p_2, p_3, \ldots, p_n\}$ with value $D = \{d_1, d_2, d_3, \ldots, d_n\}$
  - can have repetition (two dimes, three pennies)
  - S is a subset of P

  $S \subseteq P$, such that $s_i = 1$ if $p_i \in S$, $s_i = 0$ if $p_i \notin S$
- A: sum to be returned
- Goal: minimize Σs_i, such that $\Sigma d_i = A$

Brute-force Approach

- Try all subsets of P
  - since there are n coins, there are 2^n possible subsets
  - enumerate all possible subsets
  - check if a subset equals A
    - called 'feasible solution' set
    - O(n)
  - pick subset that minimizes Σs_i
    - called 'objective function'
    - O(n)

Brute-force Approach

- Best Case
  - $\Omega(n2^n)$
- Worst Case
  - $O(n2^n)$

Greedy Approach

- Go from largest to smallest denomination
  - Return largest coin p_i from P, such that $d_i \leq A$
  - $A = A - d_i$
  - Find next largest coin …

if money is sorted (by value), then algorithm is $O(n)$

Does Greedy Always Work?

Consider $A = 20$
and $D = \{1, 1, 1, 1, 1, 10, 10, 15\}$

Greedy returns 6 coins
Optimal is 2 coins

Text Processing

- Brute-force Pattern Matching
- Improved Pattern Matching
  - Boyer-Moore Algorithm
  - *not really brute-force*
  - *not really greedy either*

Pattern Matching

- T: text string of length *n*
- P: pattern string of length *m*
- Question: Is P a substring of T?
- Answer: starting index of match or indication that P $\notin$ T

Pattern Matching: Pseudocode

```
Algorithm BruteForceMatch (T,P)
   Input: character string T of length n
  and character string P of length m
   Output: integer -1 if P ∉ T, integer i
  (start location of P in T) if P ∈ T
   for i <- 0 to n-m do
      j <- 0
      while (j < m and T[i+j] = P[j]) do
         j <- j + 1
      if j = m then
         return i
   return -1
```

Pattern Matching: Complexity

```
for i <- 0 to n-m do                          O( )
   j <- 0                                       O( )
   while (j < m and T[i+j] = P[j]) do           O( )
      j <- j + 1                                  O( )
   if j = m then                                O( )
      return i                                    O( )
return -1                                     O( )
```

Worst case complexity: O( )

Best case complexity: Ω( )

Better Pattern Matching: Boyer Moore Algorithm

Two Improvements:

- Looking Glass Heuristic
  - When testing P against T, begin at P[m-1]
- Character Jump Heuristic
  - Mismatch T[i] = c with P[j]
    - if c ∉ P, then shift P past T[i]
    - else if last(c) to left of P[j] then
      - shift P to align last(c) with T[i]
    - else shift P to right by one

Summary: Brute & Greedy

- Brute-force:
  - solve problem in simplest way
  - generate entire solution set, pick best
  - will give optimal solution with (typically) poor efficiency
- Greedy:
  - make local, best decision, and don't look back
  - may give optimal solution with (typically) 'better' efficiency
  - depends upon 'greedy-choice property'
    - global optimum found by series of local optimum choices