COL1000: Introduction to Programming

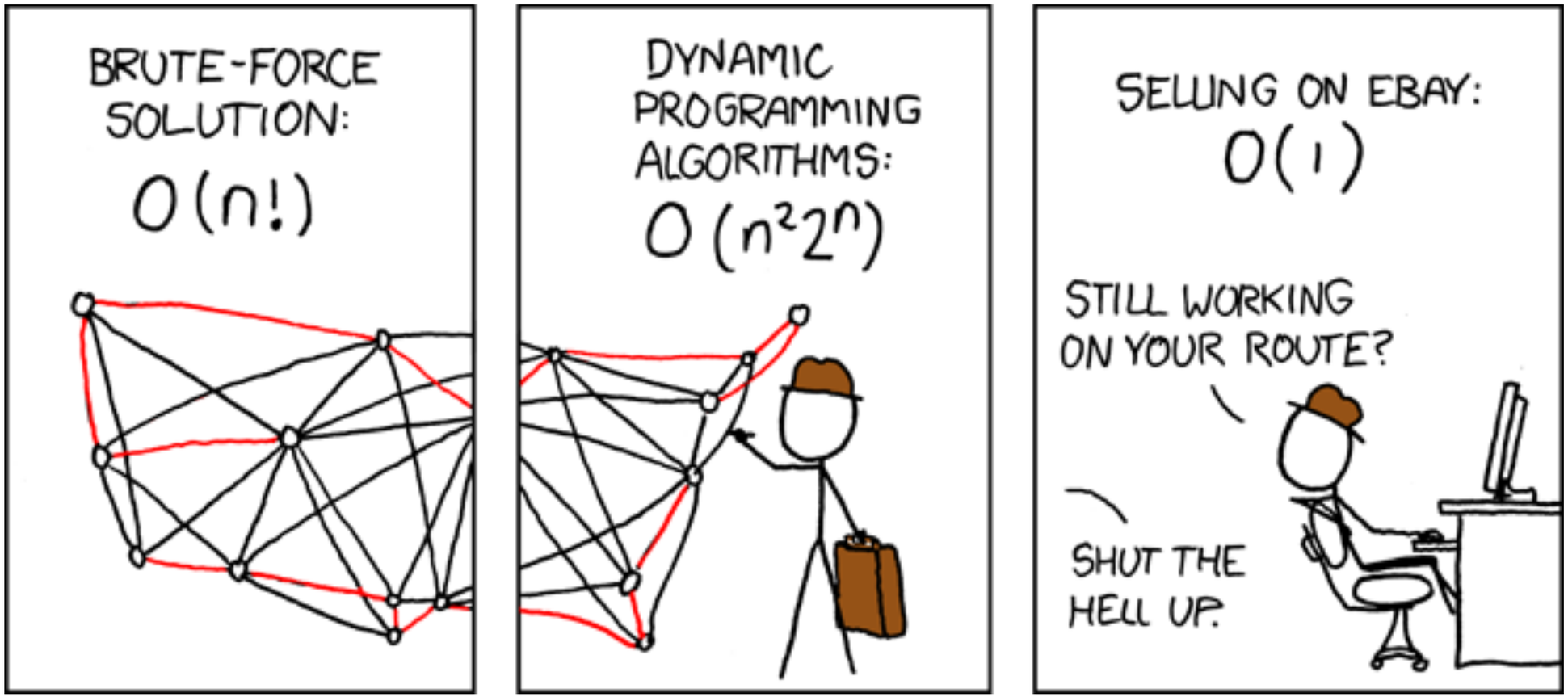
Algorithmic Thinking

Subodh Sharma | Lec 32 | Nov 1

Announcement

- **(Usually) Monday's — 5 pm to 7 pm; Doubt learning sessions in Bharti 419**
  - **Only one student visited me!**
-

Algorithmic Development



Algorithmic Thinking & Development

- **Start with the specification**
  - Preconditions, Postconditions, edge behaviours,
    - Objective: minimize/maximize/counted?/decision?
  - Supply input-output pairs
- **Choose the right data structures**
  - Dictionaries, Lists, Tuples, …
- **Choose the model for computation based on performance goals(must be efficient)**
  - Greedy, Divide and Conquer, Randomisation, Dynamic Programming …

Common Ideas for Algorithm Design

- **Decomposition:** Break problem down to sub-problems
- **Abstraction:** Know **what to do first** before **how to do** them
- **Pattern Recognition:** Identify patterns similar in problems/subproblems
- **Generalisation:** Check to generalise a specialised solution

Common Algorithmic Techniques

- **Brute-force:** explore the entire space of answers and stop when the solution is identified
  - Eg: Given an array of integers, a, and a target integer, t, find indices i, j, such that, $a[i] + a[j] = t$
    - Try all pairs
- **Greedy:** At each step in the decision making take the most optimal decision
  - Eg: Coin change with **canonical denominations, say** $\{1,5,10,25\}$**;** choosing the largest denomination each step will be give optimal
- **Divide and Conquer:** Partition input into two or more subsets, solve each (usually recursively)
  - Eg: Merge Sort [38,27,43,3] —> Split [38,27], [43,3] -> Conquer (merge singletons)

Common Algorithmic Techniques

- **Others: (Not to be covered in this course)**
  - **Dynamic programming:** Store previous results
    - Eg: Pascal's triangle
  - **Randomisation:** Probabilistic correctness
    - **Eg:** Miller-Rabin Primality Testing — Given n, randomly pick several witnesses to check if n is composite
  - **Backtracking:** Constraint satisfaction fails then backtrack and start again
    - **Eg:** Place N queens on a NxN board so that none attack each other
  - **Branch and Bound:**
    - Traveling Salesman: Prune branches that exceed the best cost so far

Case Study: Coin Change Problem

- **Input**: Set of coin denominations $C = \{C_1, \ldots, C_m\}$of positive integers; **reach a target amount** T >=0; unlimited copies of each coin denomination
- **Output: Possibly Minimum** number of coins whose values sum to T; if impossible then report no solution
- **Specs:**
  - **Preconditions:** $\forall i : C_i$ is positive; T is non-negative
  - **Postconditions:** return $min \Sigma x_i$ **s.t.** $\Sigma(x_i \,.\, c_i) = T$
  - **Performance goal:** **As efficient as possible**

Case Study: Coin Change Problem

- **Example:** C = {1, 5, 10, 25}; T = 42
  - Choose the unexhausted largest —> 25, count = 1, remaining_amt = 17
  - Choose the unexhausted largest —> 10, count = 2, remaining_amt = 7
  - Choose the unexhausted largest —> 05, count = 3, remaining_amt = 2
  - Choose the unexhausted largest —> 01, count = 4, remaining_amt = 1
  - Choose the unexhausted largest —> 01, **count = 4**, remaining_amt = 0
  - Solution = {25:1, 10:1, 5:1, 1:2}
- **Greedy Algorithm:**
  - **Select the largest denomination coin that is less than or equal to the remaining amount**
  - **Subtract that coin's value from the amount**
  - **Repeat the above steps until the amount reaches zero.**

Is the spec complete?

Refined Spec & Algorithm

- **Refined Spec:** f you reach amount 0 — done; if the next coins all the way cannot sum up to the target or least coin is exhausted and amount > 0 —> no solution
    - Quicker way? If GCD(C) does not divide T, then no combination of C can reach T
- **Algorithm:**
    - Reverse sort coin Denominations set
    - Loop until coin Denominations set is not empty:
        - Check if the largest coin denomination can be used, ie amount >= coin
            - Calculate the number of such coins can be used, ie count = amount // coin; Add to the result
            - Update the remaining amount, ie amount %= coin; remove the largest coin from the coin Denom set
        - If the amount is 0, then stop with Success
    - If amount > 0, then stop with Failure

Greedy is Suboptimal

- **Suboptimal** — C= {1,3, 4} T = 6 —> 4, 1,1, —> 3 coins; 3 +3 —> 2 coins (optimal)
- Greedy approach is **optimal only for canonical coins**
  - Let $G(x)$ be the # of coins in greedy approach and $Opt(x)$ be the real optimal solutions to meet target x
    - Then a system is canonical $iff\ G(x) = Opt(x)$
    - **(Optional)** Look at **Kozen-Zaks theorem** to establish when Greedy can be optimal

Greedy is Suboptimal

- **Simple Greedy strategy is suboptimal — Why?**
  - **False assumption: that the largest coin always participate in an optimal solution**
- **Alternate strategy — backtrack + greedy**