Graphs (and etc.)

Algorithm

1. Understand the problem. Play with small inputs.
2. Don't modify known algorithms (yet)

   Can we "reduce" this problem to an easier one?

   Modify the graph or other inputs so that we can run a known algorithm, e.g. BFS.
3. If not, modify accordingly e.g. best path.

Proof of Correctness

1. Sadly, no set template here :'(
2. If you succeeded in reduction then you may assume the correctness of the black-box algorithm.
3. If not, try induction (or other techniques)

   e.g. imitate graph proofs done in class.

Runtime Trace your algorithm. What data structures did you use?

Greedy

what does that even mean?!

Algorithm

1. Think of a heuristic (greedy rule)

   Run it with small examples and get a ~feel~

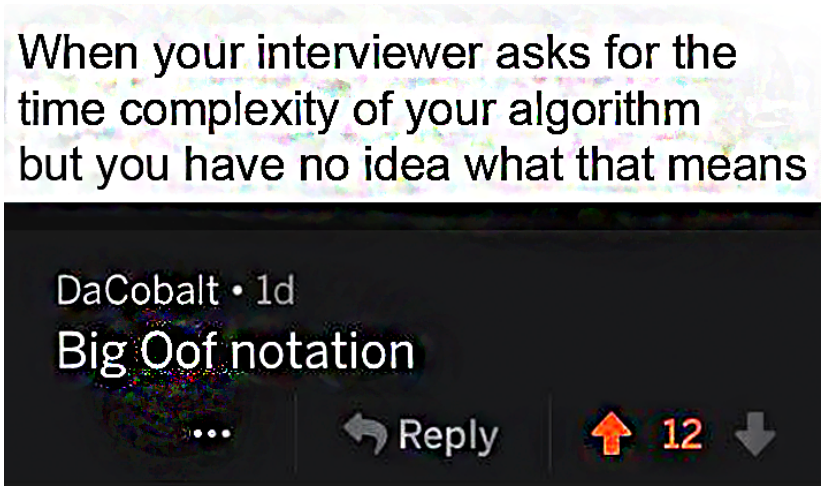
   e.g. highest profit, earliest deadline (or a combo)
2. Add by greedy rule (if feasible)

   how to check feasibility?

Proof of Correctness

1. Various (but rather similar) approaches:
   - Proof by contradiction
   - Exchange argument
   - Stay ahead
2. Prove optimality of greedy.

Runtime Trace your algorithm. What data structures did you use? Did you sort?



* Disclaimer: This is how I approach these problems!

Divide & Conquer

Algorithm

1. Consider small cases. Establish base cases.
2. Divide into subproblems.
   What preprocessing is necessary?
3. Conquer by using recursion on subproblems.
4. How to combine "conquered subproblems"!
   What postprocessing is necessary?

Runtime

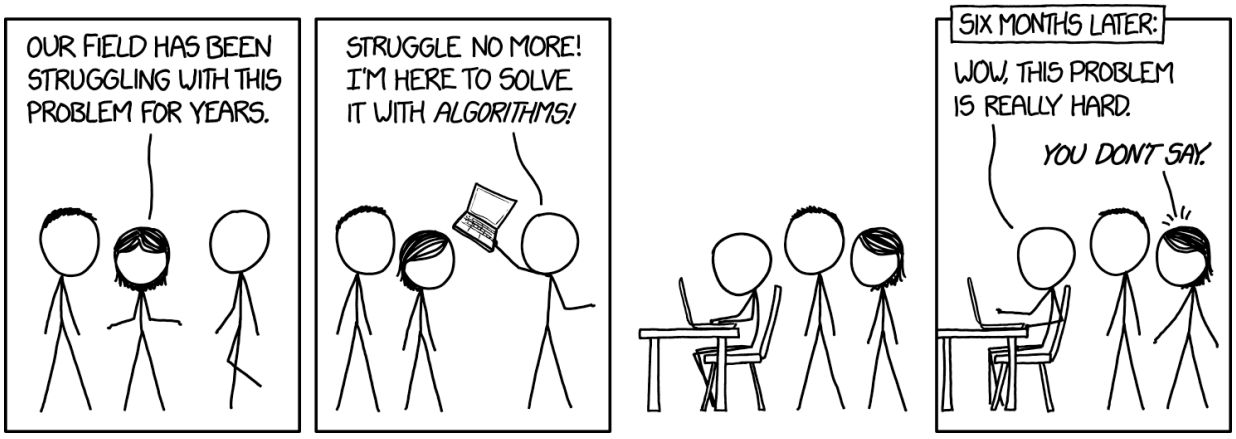
1. Find the recursive relationship
2. Analyze using your favorite method.
   e.g. recursin tree, Master Theorem

Proof of Correctness

1. Induction! Why? When?

* Induction is useful when we do anything repeatedly, e.g. recursion, stay ahead.

2. Trace your algorithm
   - Base case
   - State Inductive Hypothesis. (Strong)
   - Use I.H. for subproblems to assume it returns what we want.
   - How did you transform the subproblem solution to the original problem solution?



Dynamic Programming

Algorithm

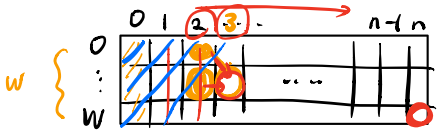
1. Consider small cases. Establish base cases.
2. Divide into multivariate subproblems
   Consider subset of items. Is sorting necessary?
   $\{1, 2, \cdots, i\}$
3. Devise a recursive relationship.
   Consider different options. (e.g. include/exclude)
   Combine via operation such as max/min/AND/OR.
4. Final output e.g. $OPT(n, W)$

Runtime

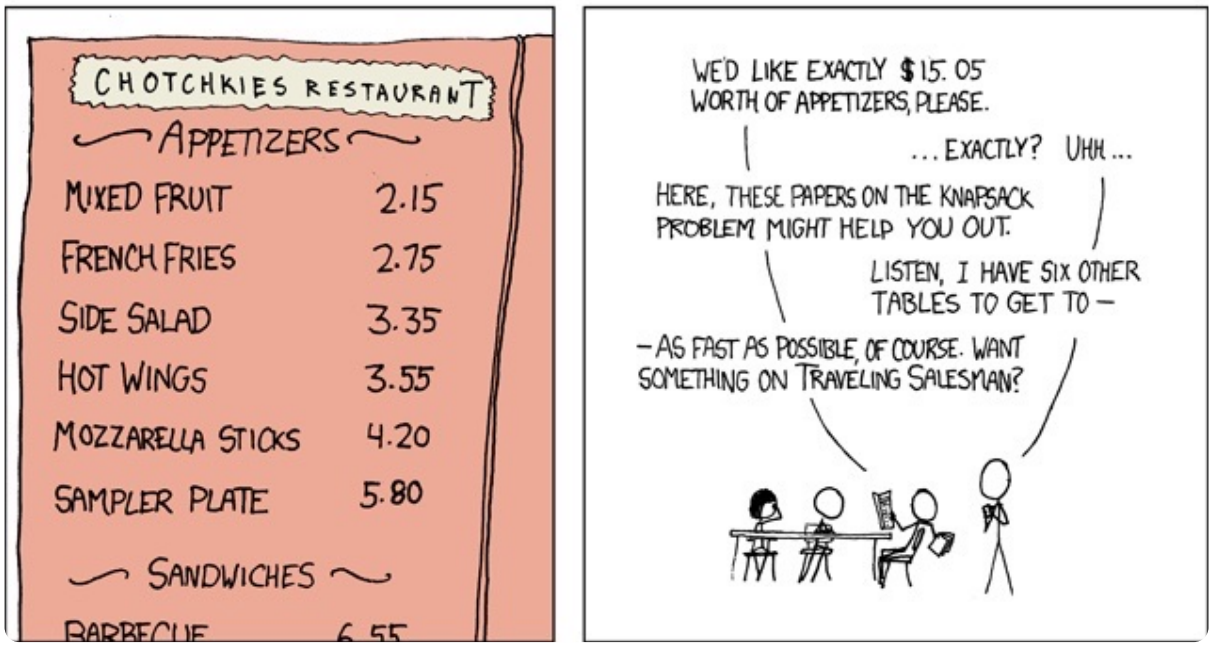
1. [# of subproblems (size of DP table)] x [Recursive relation computation time]

Proof of Correctness

1. Induction!
   Induct on the direction of iteration



2. Trace your algorithm
   - Base case
   - State Inductive Hypothesis. (Strong)
   - Use I.H. for subproblems to assume it returns what we want.
   - Did you consider all cases and why did you take the maximum/minimum?

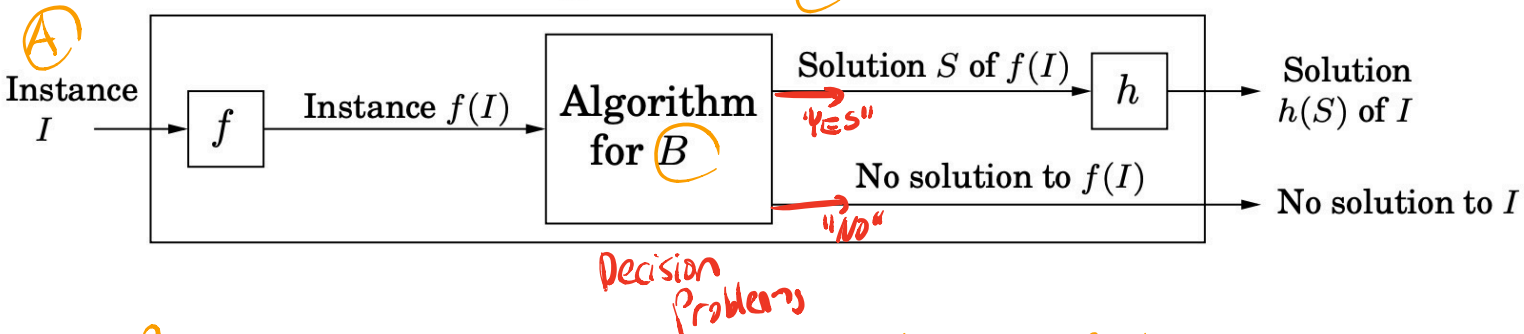


Everything so far we saw was efficiently solvable, but is every problem?

Reduction is the name of the game!

$A \to B$

Algorithm for A



Decision Problems

Polynomial-time Reduction

$A \to B$

Black box

(use polynomial times)

$A \leq_p B$

Mapping Reduction $A \leq_m B$

Map "YES" instance of A to a "YES" instance of B and "NO" to "NO"

★ Black box used only once.

Network Flow

Algorithm

1. Understand Ford-Fulkerson, residual networks, and max-flow min-cut.
2. Is there some kind of "matching" or "grouping"? Can we reduce this problem to one we know?
   Several problems (we know how to solve) at hand.
   Max-flow, Min-cut, Bipartite Matching, Project Selection, Airline Scheduling, and so on ...
3. Reduce accordingly to that problem instance.

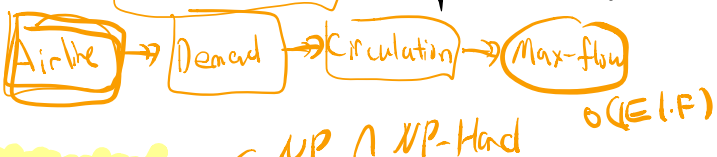
Proof of Correctness

1. If using a reduction, prove a "if and only if" statement.
   For example
   "There exists a perfect matching iff max-flow value is n."

Runtime

How long did it take for each step of the reduction?

Runtime = Preprocess time + Black-box time + Postprocess time.



NP-Completeness

$\in NP \cap NP\text{-Hard}$

Algorithm

(Showing $A \in NP$ and A is NP-hard)

$B \to A$

1. FIRST! Show $A \in NP$.
   Given a candidate solution, how to verify?
2. Think of similar NP-hard problems to A.
3. Fix the problem. Given such instance, how would you solve it using A?
4. Create a 1-1 mapping reduction.

Proof of Correctness

1. Prove a "if and only if" statement.
   For example,
   "There exists a subset that sums to k iff there exists a partition in U'..."

Runtime

Check if your verifier (for checking NP) is poly-time.

We don't really care about specific runtimes for the reduction for NP-hardness as long as the reduction (pre/post processing) is poly-time.

Randomness

- Define the wanted quantity as a random variable, say Y.
- $E[Y] = \sum_y y \cdot Pr[Y = y]$
- SIMPLIFY the problem by introducing indicator R.V.s (only takes values 0 or 1.) if possible.

  Define X_i to be 1 if event for i or 0 otherwise.

  This helps break down Y into smaller manageable chunks.

$$Y = \sum_{i=1}^{n} X_i \Rightarrow E[Y] = E\left[\sum_{i=1}^{n} X_i\right]$$

$$= \sum_{i=1}^{n} E[X_i] \quad \text{(linearity of expectation)}$$

$$= \sum_{i=1}^{n} 1 \cdot Pr[X_i = 1] + 0 \cdot Pr[X_i = 0]$$

$$= \sum_{i=1}^{n} Pr[X_i = 1]$$

- Compute $Pr[X_i = 1]$, i.e. probability that event for i happens.