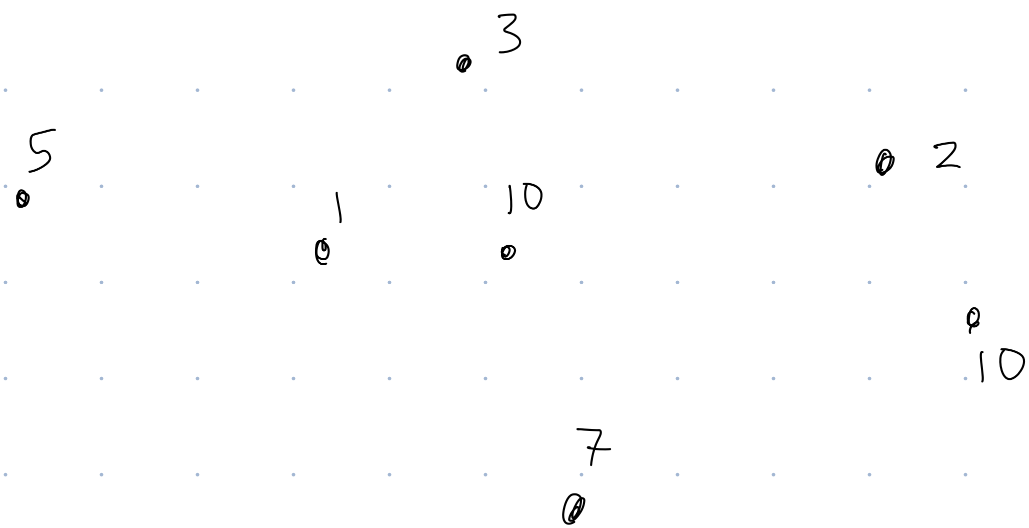
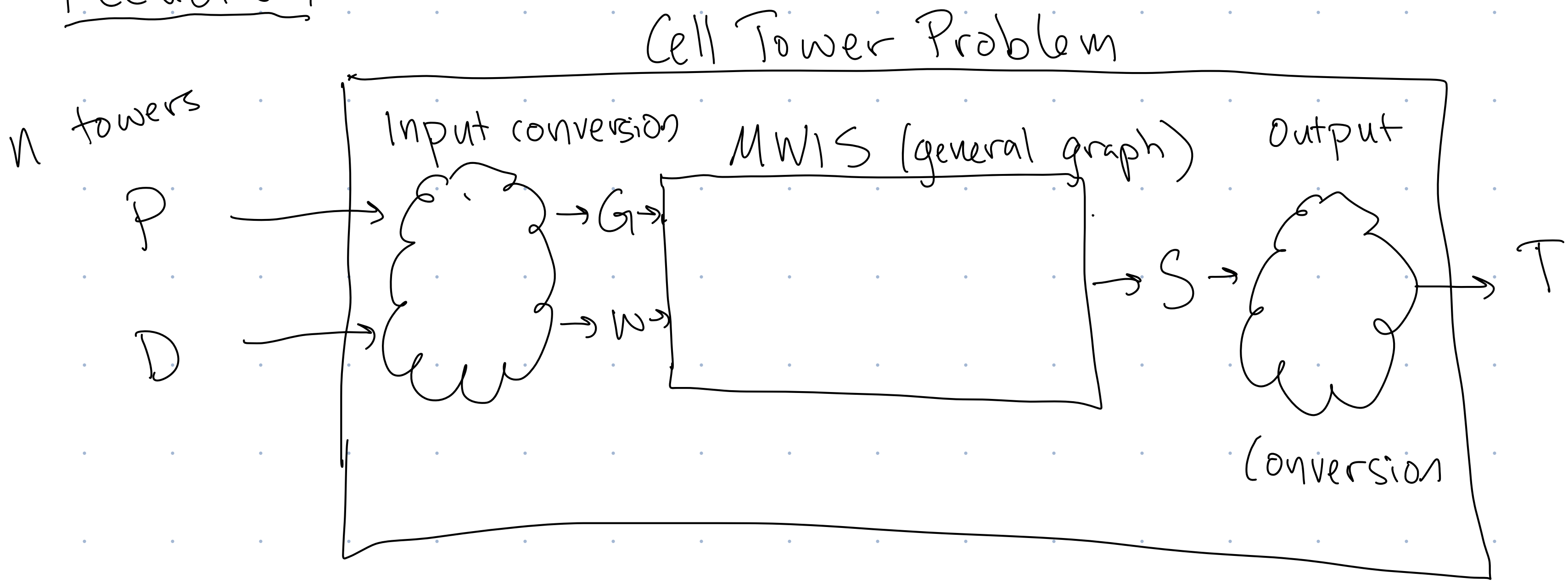
Cell Tower Scheduling

Input: Array P s.t. P[i] is location of i^{th} cell tower
Array D s.t. D[i] is # of data packets to send from i^{th} tower



Output: Set of towers T to broadcast in the next time step.
(If 2 towers within 2 miles of each other broadcast at the same time → interference)

Reduction



I.C. (P, D)

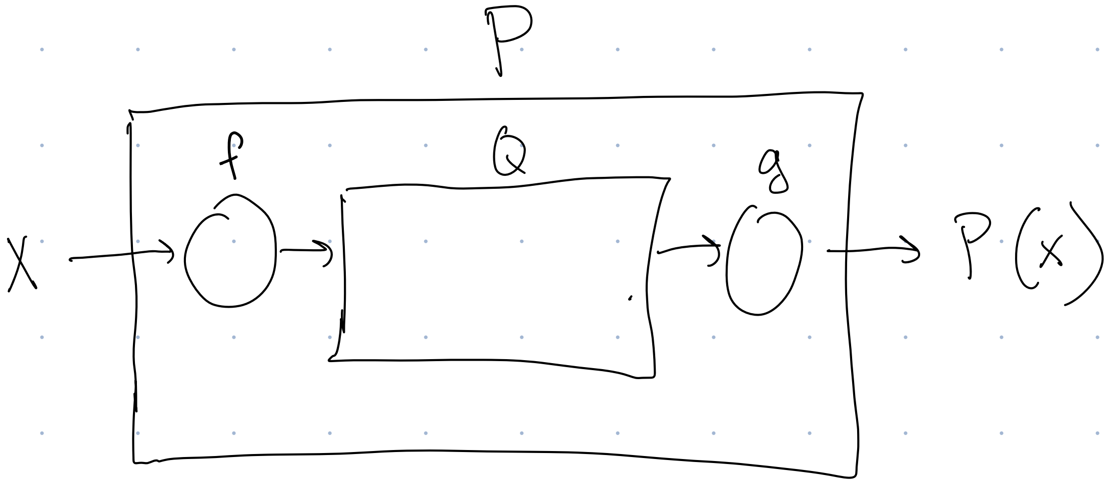
- $V = \{1, 2, \ldots, n\}$,
- For $i \in [n]$: $[n] = \{1, 2, 3, \ldots n\}$
  - $w(i) = D[i]$
- For each pair $i, j \in n$:
  - If $d(t_i, t_j) \leq 2$, add $\{i, j\}$ to E

O.C. (S)

Return S

1. Ethical concerns (stakeholders, who benefits, who is harmed, cycle of reinforcement)
2. What should input/output conversion be?
3. Runtime of conversion in terms of n?

Generic Reduction

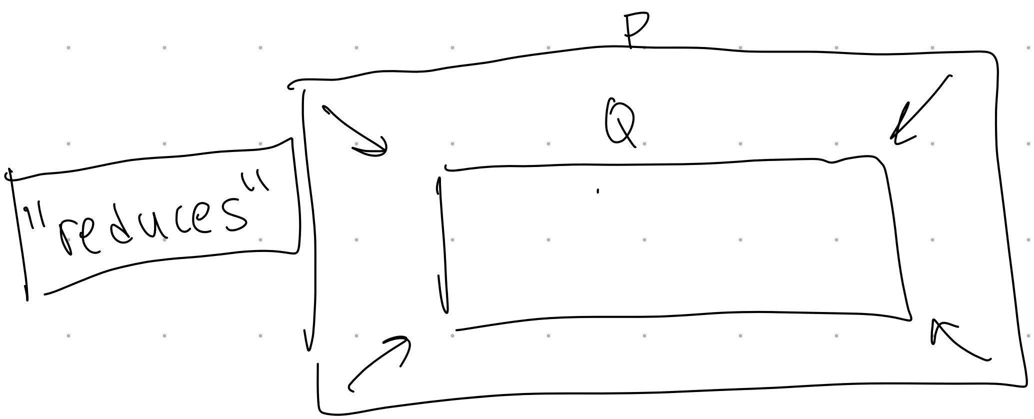


def: If runtime of f, g are $\boxed{O(poly(n))}$ then we say "P is polynomial time reducible to Q", denoted $\boxed{P \le_p Q}$

"P reduces to Q"

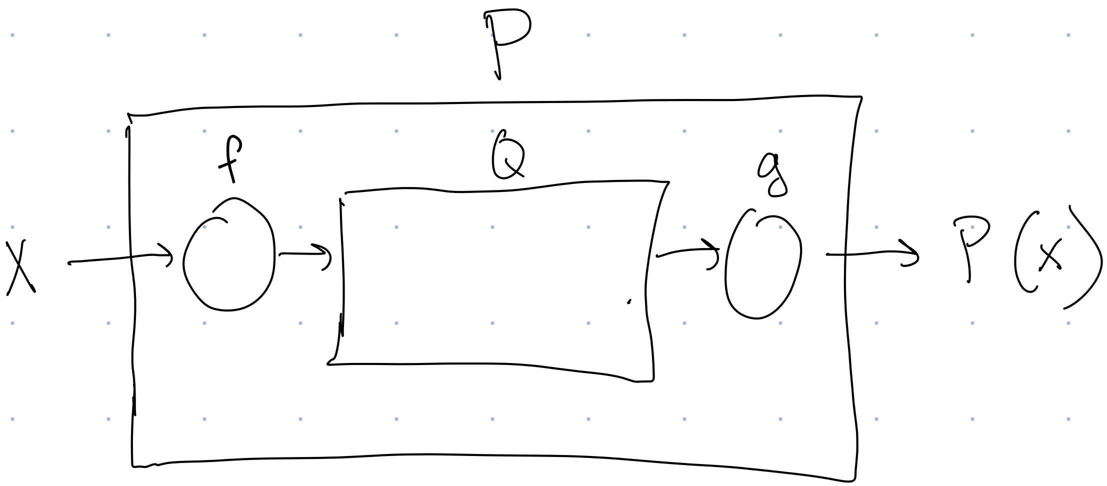
$O(n^d)$ for d a constant
ex: $O(n)$, $O(1)$, $O(n \log n)$
trivial (relative to $O(2^n)$)

- Q is so powerful, it can not only solve Q it can also solve P (with only a trivial amount of extra work)
- Q must be harder than P, because we need more resources to solve

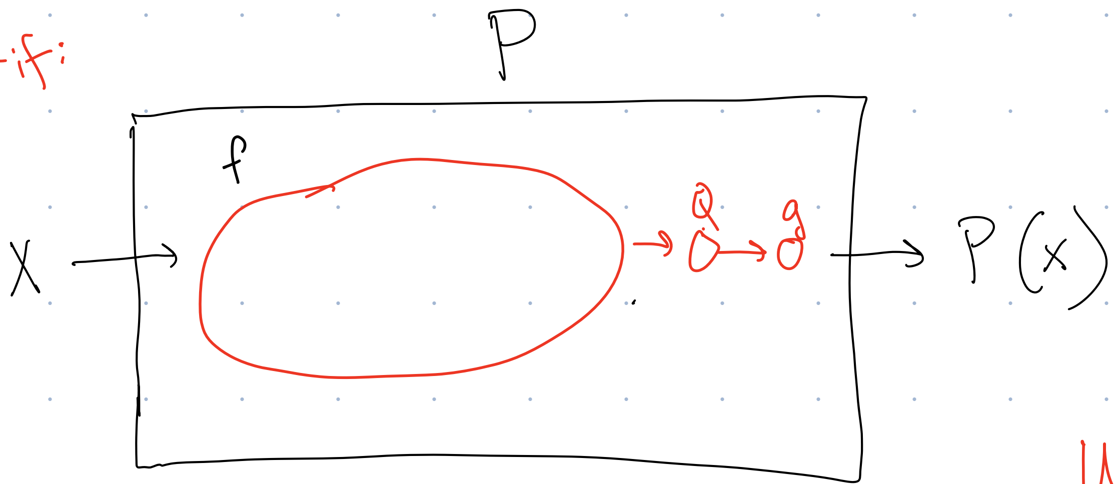


Generic Reduction

Want



What if:



f is doing all the work!

We will put upper bounds on how much work f, g can do, to ensure Q solver does most of the work

Why think about reductions?

- Practical: If have an alg for Q, can use it for P
- Conceptual: Gives us a way to compare the difficulty of problems