Algorithms 2020

~~Last flow example~~
Intro to Complexity

Recap

- HW - due Sunday now
- Next HW: oral grading (over flows) during week of Nov. 9 (?) (likely Nov. 11 + 12, but stay tuned)
- Final HW: before break? after break?
  - NP-Hardness
  - graded

Then final exam during our assigned time (plus 1-2 hours), due on Canvas

→ (Thursday Dec. 3? 8-10pm)

↳ conflicts? Concerns?

tell me soon

Crazier "word problem" examples (last flow problem)

A company sells k products, + keeps records on n customers.

Goal: Design a survey to send to n customers, to get feedback.

- Each customer's survey shouldn't be too long, + should ask only about products they purchased
- Each product needs some # of reviews from different customers

Input: - k products
- n customers

$A[k][n]$ - records of who bought what: $\boxed{A[i][j]}$ for $i \le k$, $j \le n$ $= 0$ or 1

- For each customer,

$C[1..n]$ C_j is max # of products to ask customer j ~~them~~ about

- for each product, P_i is minimum # of reviews needed for product i

$P[1..k]$

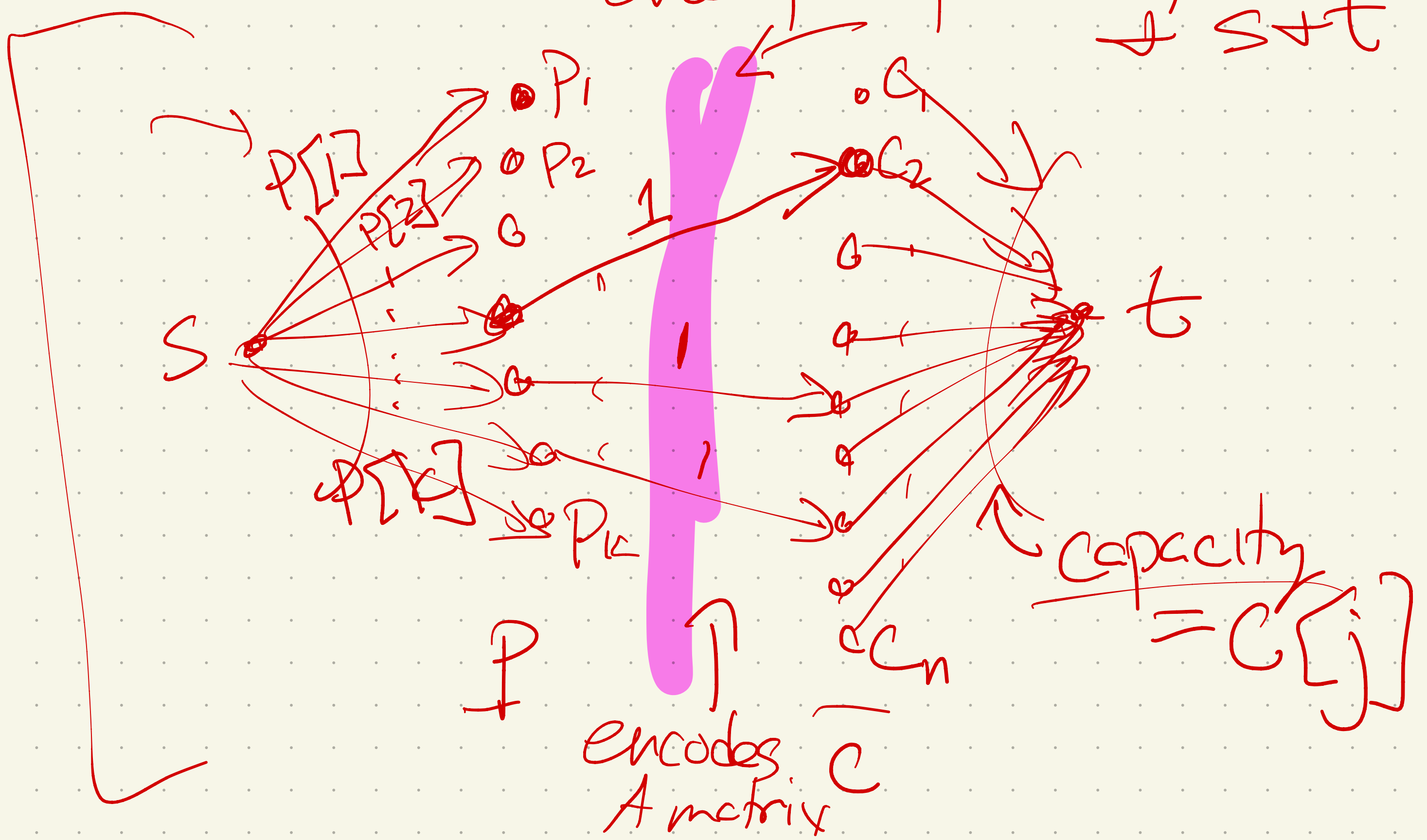
Can we design a survey?

use flow!

Algorithm — Reduce survey design to flow:

Build a graph G: use A

add $n+k+2$ vertices: one per product, customer, + s + t



Add edges:

$s \to p_i$ edges w/ capacity $= P[i]$

$c_j \to t$ edges w/ capacity $= C[j]$

cap $= 1$ $p_i \to c_j$ edge if $A[i][j] = T$ (or 1)

(add all purchased edges)

//build G as on prev page

run flow on G

↳ (Orlin $O(VE)$)

use flow to build survey;

decompose flow paths in G for max flow;

each "middle" edge, if $f(e) = 1$ $p_i \longrightarrow c_j$

then add that product p_i to c_j's survey

↳ output: survey allocation

Runtime: $O((n+k)nk)$

Build the G:

$V = n+k+2 = O(n+k)$

$E = \underset{\text{to } s?}{n} + \underset{\text{to } t?}{k} + \underline{n \cdot k} = \underline{O(nk)}$

Run Orlin: $O(VE) = O((n+k)(nk))$

use flow + get survey: check each flow value: $\underline{O(nk)}$

Correctness

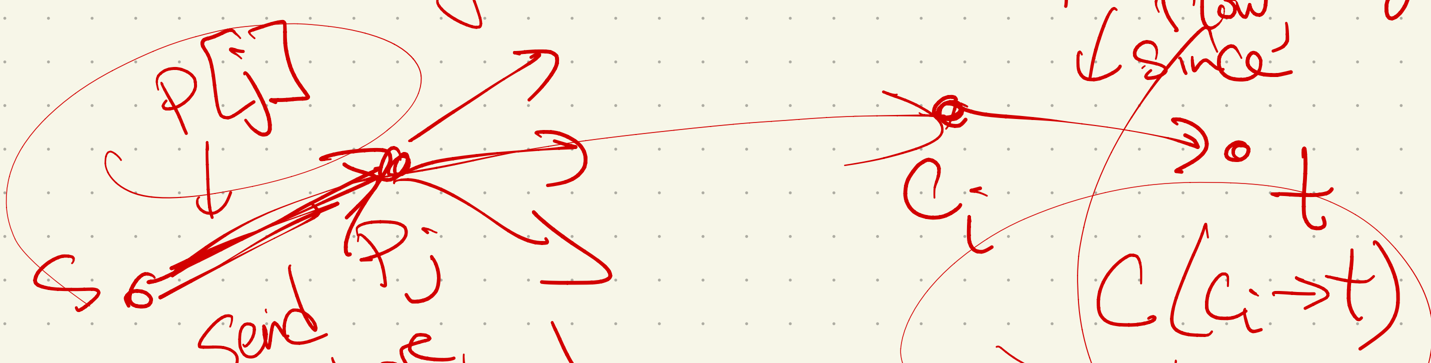
Take valid survey:

$\Rightarrow$ build flow $f(P_j \to c_i) = 1$

put 1 flow on edge if c_i review P_j

Claim: get valid flow

put necessary flow (since



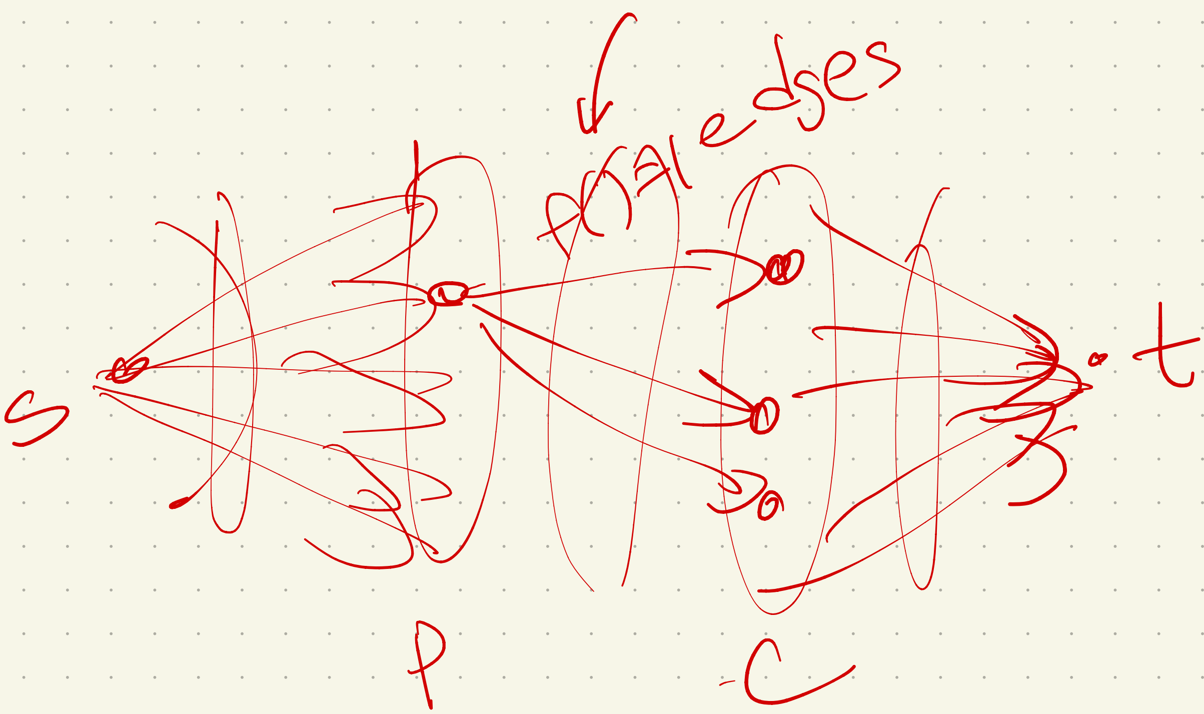
send flow here ✓ no exceed

to be no more than max amount C[i]

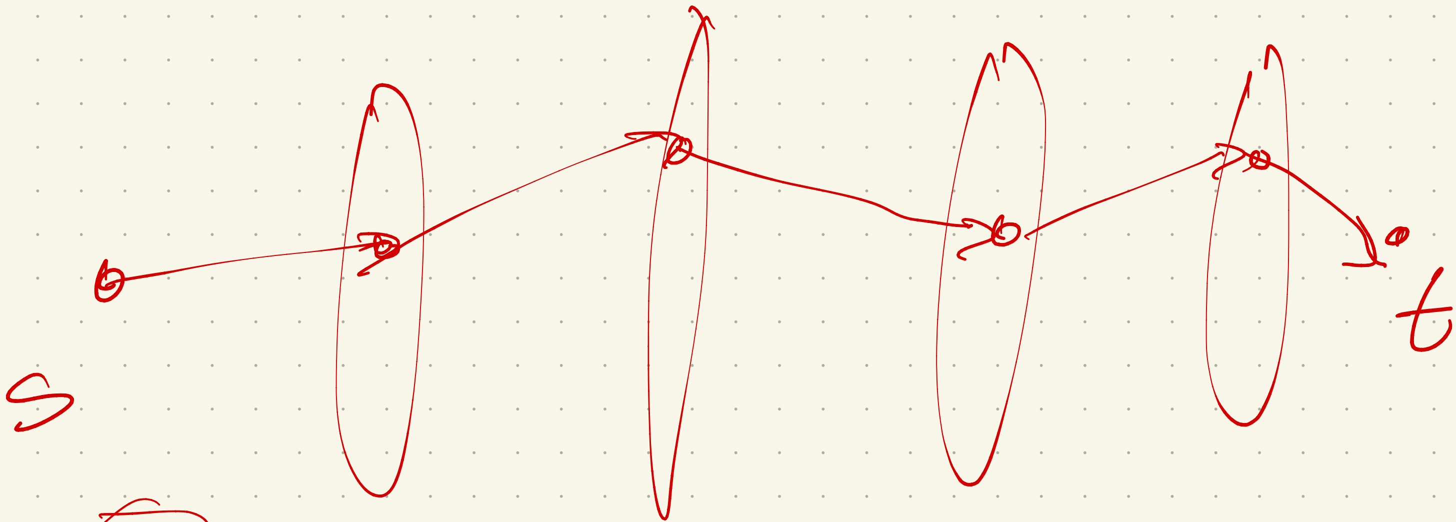
Valid flow must give valid survey.

Why? $c[i] = c(c_i \to t)$ can't be exceeded.

Get flow of value = $\sum_i P[i]$ if all product constants are meet



in book!



☆ Key: flow paths give some assignment

Practice: build such a graph

Correctness (cont)

Quantifying Hardness:

Fundamental question:

Are there "harder" problems?

How do we rank? Polynomial Hierarchy

↳ runtimes:

- linear
- n log n
- quadratic

⋮

- ?? → subexponential → Ex: factoring
- → exponential (backtracking chapter)
- → worse? yes.

} polynomials (in input)

Undecidabily:

Some problems are impossible to solve!

The Halting Problem:

(Turing + Church - 30's?)

code: Python code

Given a program P and input I does P halt or run forever if given I?

infinite loop

Output: True / False

(Utility should be obvious!)

Note: Can't just simulate P on I. Why?

If it goes forever, won't stop + let me answer

↳ I don't know when to output true.

Thm [Turing 1936]:

The halting problem is undecidable.

(That is, no such algorithm can exist.)

Proof: (by contradiction) suppose we have such a program h:

$$h(P, I) = \begin{cases} \text{True} & \text{if } P \text{ halts on } I \\ \text{False} & \text{otherwise (infinite loop)} \end{cases}$$

program input

Need a contradiction now...

Now define a program g that uses h:

g $X(X)$ so loops

uses h

$$g(X) := \begin{cases} \text{if } h(X, X) = \text{False} \\ \quad \text{return False} \\ \text{else } (X \text{ halts on input } X) \\ \quad \text{loop forever} \\ \quad \text{while}(1) \end{cases}$$

$h(X,X)=T$

The contraction: What does g(g) do?

$X = g$

Calls h(g, g):

If h(g,g) = true, that means g halts on input g

But then g(g) should run ~~forever~~ forever!

If $\boxed{h(g,g)}$ = False, then g infinitely loops ~~on~~ on input g.

But then g should return false ~~on~~ on input g!

$\Longrightarrow$ h can't exist.

So... what next?

Clearly, many things are solvable in polynomial ~~time~~.

Some things are impossible.

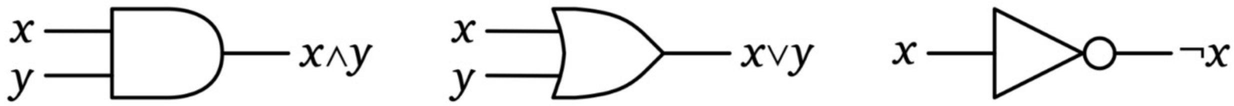
But – what is in between?

?? what can we do?

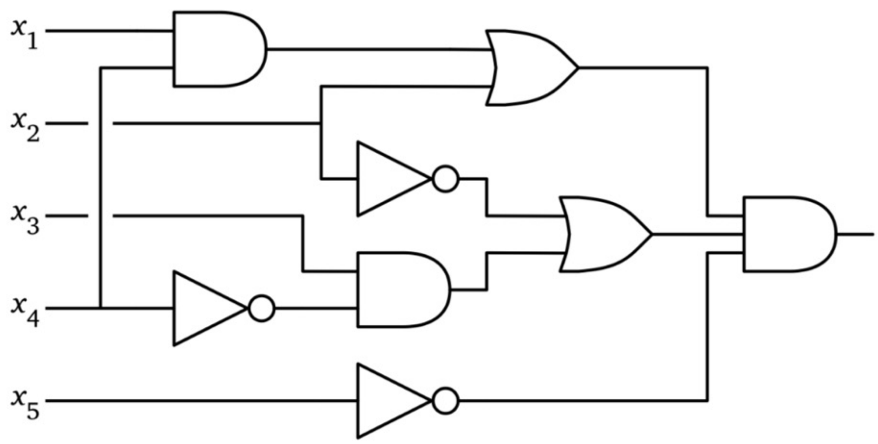
Idea:

Set the idea of what are limits of (practicle) computing.

The first problem found:
Boolean circuits



An AND gate, an OR gate, and a NOT gate.



A boolean circuit. inputs enter from the left, and the output leaves to the right.

Given a set of inputs, can clearly calculate output in linear time (in # inputs + # gates).

How?