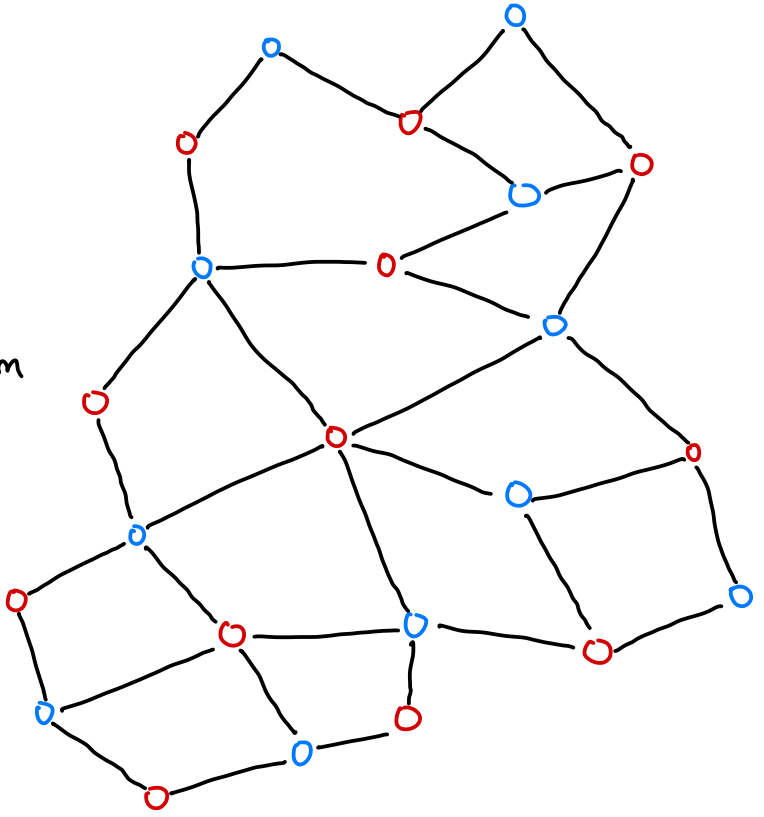
Welcome to Distributed Algorithms!!

Spring 2022

Outline

1. Course introduction and structure
2. Motivating example: stable matchings
3. Gale-Shapley algorithm
4. Stable matchings as a distributed problem



Course Introduction + Structure

Meetings: 80 min 2× per week

"Guided discussion" format

Readings/activities to be completed before class

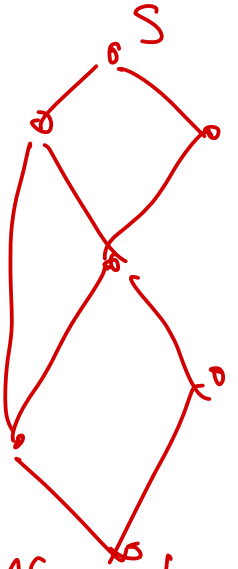
Active participation expected

Coursework:

- Written homework every ~2 weeks
  - groups of up to 3
- take home quizzes every ~2 weeks
- 1 "exam" take-home, ~ week 8
- final group project (any topic relevant to distributed computing)
  - class presentation
  - written submission

Your Background

- Experience with algorithms and algorithmic techniques
  - graph algorithms: shortest paths, spanning trees, etc
  - greedy algorithms, divide-and-conquer, dynamic programming
  - analysis of algorithms, big O notation



Course Philosophy

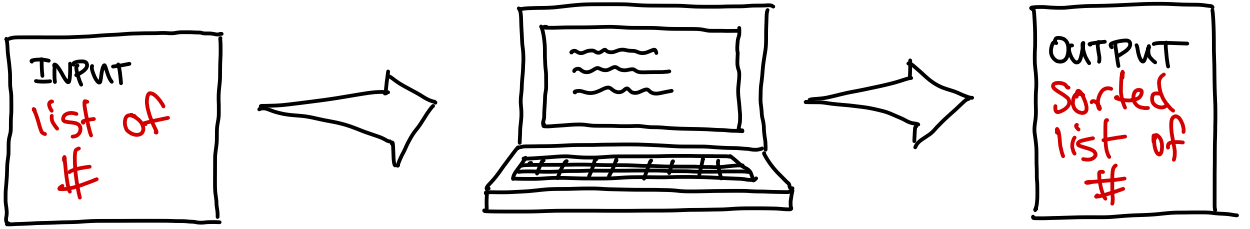
conceptual, formal

- Emphasize **principles** of distributed computing
- Examine fundamental tasks for distributed systems
- Compare **models** of distributed computation

assumptions about systems

- Understand power & limitations of distributed systems
  - What problems can and cannot be solved efficiently in distributed systems?

Centralized Computing :



Problem instance is given to program as input

Program accesses (entire) input, performs computations

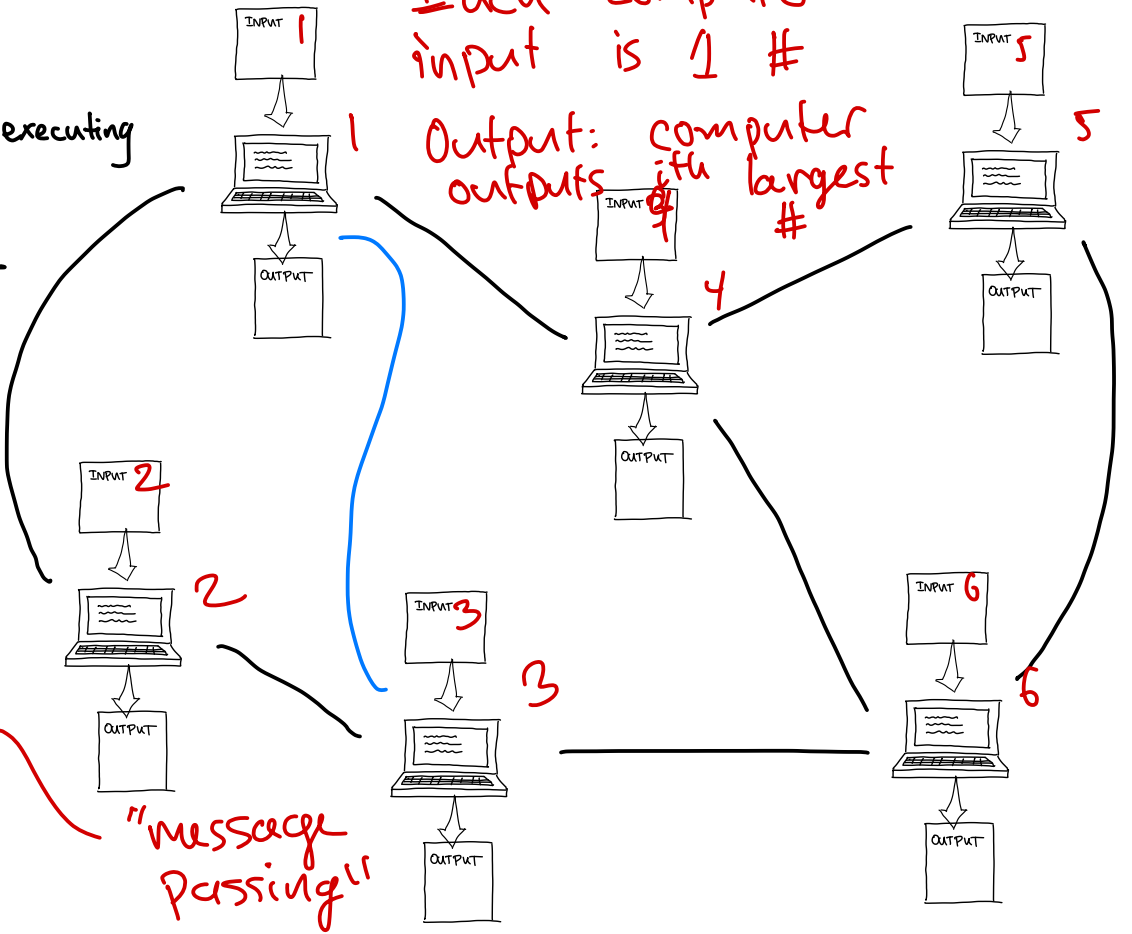
Program produces some output

Assumptions

1. Single process is performing computations
2. That process can access entire input

Distributed Computing

- Many "computers," each executing same program
- Each has its own local input
- computes own local output
- correct output depends on other computers' local inputs
- computers communicate by sending messages
  - fixed comm. links
- Problem instance
  + entire network
  + local inputs of each node



Complexity / Efficiency Measures

Centralized

- Time
  - program running time "empirical"
  - asymptotic growth of # of elementary operations "big O"
- Space / memory
  - empirical physical memory usage "RAM"

Distributed

- Time — same as centralized setting for empirical
- Space / Memory
  - space usage per process?
  - total space usage?
- Communication
  - ↪ info access is 1000's slower from another computer
  - how much data from elsewhere?
- Locality
  - who has info I need?

Issues w/ Distributed Systems

1. Distributed systems are <u>everywhere</u>
   - not just computer networks/systems
   - communication/locality concepts relevant to all distributed systems

2. Reasoning about distributed systems is <u>subtle</u>.
   - small changes in computational model can have large impact on system capabilities
   - getting the model "right" is a major challenge

This course will emphasize a (more) formal treatment

- make our assumptions about computation explicit
- as much as possible, derive results from first principles

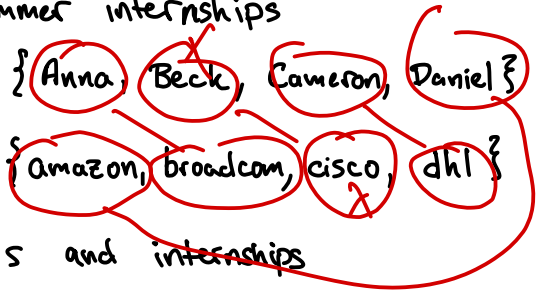
- computer networks
- social network
- Ant colonies
- neurons in brain

Motivating Example: Stable Matchings

Problem Setup Summer internships

S = students = {Anna, Beck, Cameron, Daniel}

U = internships = {amazon, broadcom, cisco, dhl}



Goal: match students and internships

- each student assigned at most 1 internship
- each internship assigned at most 1 student

So far, easy to match students to internships ... How?

↪ randomly (arbitrarily)

Greedy Strategy

More realism: preferences and constraints

- b requires systems experience — C can't go to b
- B wants to stay in US — B can't intern @ d
- A prefers a to b to c but doesn't want d
- A is rock star: a, b, c all rank her first

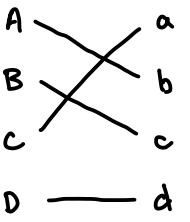
Stable matching instance consists of

1. Set S of students
2. Set U of internships

} "agents"

3. Preference list for each agent
   - ranked list of acceptable partners

An okay matching?



Example.

	1	2	3	4
Anna:	a	b	c	—
Beck:	c	b	a	
Cameron:	a	d	c	
Daniel:	c	d	b	—

	1	2	3	4
a:	A	C	B	
b:	A	B	D	
c:	A	C	D	B
d:	D	C		

Red non-matches are "blocking pairs"

Def. A student–internship pair (s, u) are a **blocking pair** if they both prefer each other to their assigned matches

- unmatched agent prefers all acceptable partners to not being matched

A matching is **stable** if there are no blocking pairs

Note Previous example was **not** stable: (A, a) was blocking

Example.

	1	2	3	4
Anna:	a	b	c	—
Beck:	c	b	a	—
Cameson:	a	d	c	—
Daniel:	c	d	b	—

	1	2	3	4
a:	A	C	B	
b:	A	B	D	
c:	A	C	D	B
d:	D	C		

What about matching:

A — a

B — b

C — c

D — d

Questions

1. Do stable matchings always exist?
   i.e. for all possible preferences

2. How could we determine if an instance has a stable matching and find one?

↪ Check every matching, check for stability

if n students, n internships all apply to all ⟹ check $n!$ $n \cdot (n-1) \cdot (n-2) \cdots$ matchings

Gale - Shapley Algorithm 1962

Main Ideas.

1. Students only apply to one internship at a time — apply to favorite first, then sequentially in order of pref.

2. Internships reject all but favorite app. so far

How to turn this into an algorithm?

Terminology. Call a student *active* if

(1) not currently matched

(2) has not been rejected by all acceptable internships

Gale Shapley Algorithm

While some student is active

1. All active students apply to next most favored internship
2. Each internship defers best app so far, rejects others

Return set of deferred pairs

Theorem. (GS, 1962). On all inputs, the GS algorithm terminates and returns a stable matching.

Example.

Students

1	2	3	4	
a	b	c		A
c	b	a		B
a	d	c		C
c	d	b		D

Internships

	1	2	3	4
a :	A	C	B	
b :	A	B	D	
c :	A	C	D	B
d :	D	C		

Analysis of Gale-Shapley Algorithm.

Terminology. Call a single iteration of the while-loop a **round**.

Observations:

- **O1** students apply to internships sequentially in decreasing order of preference
- **O2** If student s applies to u in round r, s was rejected by all preferred internships in previous rounds
- **O3** If internship u is matched in round r_1 then u remains matched in all subseq. rounds
- **O4.** If u rejects s in round r, then u prefers its match to s in all subsequent rounds.

Gale Shapley Algorithm

While some student is active

1. All active students apply to next most favored internship
2. Each internship defers best app so far, rejects others

Return set of deferred pairs

Termination. Suppose $S = \{s_1, s_2, \ldots, s_n\}$ and s_i has m_i acceptable partners. Then the GS algorithm terminates after at most $m = m_1 + m_2 + \cdots + m_n = \sum_{i=1}^{n} m_i$ rounds.

Proof.

Stability. Suppose M is the matching of deferred applicants. Then M is a stable matching.

Proof.

Interlude. Why start w/ stable matchings?

- Algorithmic problem that has nothing to do w/ computers
- math problem w/ out numbers
- economic problem w/ out money
- applications
  - matching medical residents to hospitals
  - kidney exchange
- academic significance
  - GS paper has 6,000+ citations
  - 2012 Nobel prize in economics to Shapley & Roth for work on SM
- endless variations

Personal history:

- subject of my PhD dissertation
- entry to distributed computing
- started from offhand comment

Finally. Stable matching problem is naturally a distributed problem

Stable Matchings as a Distributed Problem

Distributed Features

1. Each agent (student or internship) has own local input (pref. list)
   - preferences are only initially known to self
2. finding solution requires coordination and communication
3. Communication is (naturally) restricted
   - only communicate w/ acceptable partners
   - locality

Gale-Shapley as Distributed Algorithm

- Students communicate directly w/ internships
  - → send message "apply"
- internships respond
  - → send message "reject" or silence
- all decisions based on
  1. local information / input
  2. messages received

For next time

1. read Gale-Shapley paper
2. think about: What complications arise in implementing GS in a decentralized manner?

Gale Shapley Algorithm

While some student is active

1. All active students apply to next most favored internship
2. Each internship defers best app so far, rejects others

Return set of deferred pairs