1. INTRODUCTION

Serge Gaspers
serge.gaspers@unsw.edu.au

Katie Clinch
k.clinch@unsw.edu.au

Course admin: Song Fang
cs3121@cse.unsw.edu.au

School of Computer Science and Engineering, UNSW Sydney

Term 3, 2023

Table of Contents

1. Admin

2. Solving problems using algorithms

3. Proofs

4. An example of the role of proofs

5. Puzzles

Required knowledge and skills

- Understanding of fundamental data structures and algorithms
  - Arrays, trees, heaps, sorting, searching, asymptotic notation, etc.
  - See review booklet on Moodle
- Written communication skills
  - No programming involved in COMP3121/9101. Emphasis on problem solving, correctness, and running time analysis
  - Subsequent courses involve programming
    - COMP4121 Advanced Algorithms
    - COMP4128 Programming Challenges
    - COMP6741 Algorithms for intractable problems
  - Smarthinking for writing help

Prerequisite courses

- Only prerequisite: COMP2521/9024 - Data structures and Algorithms
- Desirable (but not officially required)
    - For undergrads: MATH1081 Discrete Mathematics (proofs, graphs)
    - For postgrads: COMP9020 Foundations of Computer Science

Extended courses

- The extended courses COMP3821/9801 run in T1 only
- Differences in content and assessment
- Can transfer until at least week 2 in term 1
- Marks will be adjusted in both courses so as not to disadvantage the extended students

Lectures

- Tuesday 14:00 - 16:00 and Thursday 16:00 - 18:00, weeks 1–5 and 7–10
- No lecture in week 6 (flexibility week)
- Face to face at Keith Burrows Theatre (J14-G5)
- Live streams in Moodle/Echo 360
- Slides and recordings on Moodle
- Ask questions in the Ed forum

Tutorials

- Weeks 1–5, 7–10 (no tutorials in flex week)
- 17 face to face, 6 online
- Will demonstrate problem solving and reasoning
- Prepare by reading and practicing the tutorial sheet before attending
- One separately recorded tutorial session will be made available

Practice Problems

- Six sets of practice problems will be released on Moodle, with written solutions (approx weeks 1, 2, 3, 5, 7, 9).
- Discuss these with your friends and on the https://edstem.org/ and during drop-in help sessions.
- You can also bring these to tutorials, drop-in help sessions, and consultation.

Drop-in help sessions

- Tuesday 10:00 - 12:00 (Newton 307) and Friday 14:00 - 16:00 (Squarehouse 215, 218)

- Weeks 2–5 and 7–10

- Collaborate with other students, work on practice problems, and consult with tutors in attendance.

Consultation

- Friday 16:00 - 17:00, weeks 1–5 and 7–10
- Join live on MS Teams
- Recordings on MS Teams

Assessment

- Take-home quiz
    - Released week 1, attempt before your second tutorial
    - Incorporate tutor's feedback until task complete
    - Weighted 5% of course mark
- Assignments
    - Three assignments, released in weeks 1, 4 and 7
    - Each consists of 3 questions
    - Each weighted 15% of course mark

Assessment

- Final Exam
    - Section I: five MCQ
    - Section II: three algorithm design problems
    - Hurdle: must get at least 50% in at least one question of Section II
    - INSPERA (on campus): more info here
    - Weighted 50% of course mark

- Forum participation
    - Up to 5 bonus marks

Textbooks

Recommended textbook

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at UNSW Bookshop

- excellent: very readable textbook (and very pleasant to read!);
- not so good: as a reference manual for later use.

An alternative textbook

Cormen, Leiserson, Rivest and Stein: *Introduction to Algorithms*
4th edition now available at UNSW Bookshop, 3rd edition also useful

- excellent: to be used later as a reference manual;
- not so good: somewhat formalistic and written in a rather dry style.

Feedback

- Changes from last year:
  - weekly tutorials
  - drop-in help sessions
  - fewer assignments and fewer questions per assignment
  - FAQ on Moodle and Ed
  - earlier feedback via take-home quiz

- Feedback is always welcome, e.g.
  - myExperience survey
  - feedback post on Ed (can post anonymously)
  - email

Table of Contents

1. Admin
2. Solving problems using algorithms
3. Proofs
4. An example of the role of proofs
5. Puzzles

Introduction

What is this course about?

It is about **designing algorithms** for solving practical problems.

What is an algorithm?

- An algorithm is a sequence of precisely defined steps that can be executed *mechanically*, i.e. without intelligent decision-making.
- Designing a recipe involves creativity, executing it does not; the same is true of algorithms.
- The word "algorithm" comes by corruption of the name of Muhammad ibn Musa al-Khwarizmi, a Persian scientist 780–850, who wrote an important book on algebra, *"al-Kitāb al-Mukhtaṣar fī Ḥisāb al-Jabr wa'l-Muqābala"*.

Introduction

In this course we will deal only with sequential deterministic algorithms, which means that:

- they are given as sequences of steps, thus assuming that only one step can be executed at a time;
- the action of each step is not random, i.e., it always gives the same result for the same input.

Introduction

Our goal:

To learn **techniques** which can be used to solve **new, unfamiliar** problems that arise in a rapidly changing field.

Course content:

- a survey of algorithm **design techniques**
- particular algorithms will be mostly used to illustrate design techniques
- emphasis on development of your algorithm design **skills**

Why should you study algorithms?

- Real world problems are bespoke, just like the problems you'll encounter in this course.
- AI tools and many programmers do not have the skills to solve new problems.
- These problems also pose interesting intellectual challenges in and of themselves.
- Interview questions at good companies are often similar to questions you encounter in this course.
- It is vital to be able to communicate your ideas clearly and persuasively.

Example: Two Thieves

Problem

Alice and Bob have robbed a warehouse and have to split a pile of divisible items. There is no objective valuation of the items (e.g. price tags); instead, Alice and Bob each have a valuation in mind, and their valuations might be different.

Design an algorithm to split the pile so that each thief values their own pile as at least half the loot.

Solution

Alice splits the pile in two parts, so that she believes that both parts are of equal value. Bob then chooses the part that he believes is no worse than the other.

An indivisible detour

Note

We are assuming that it's always possible to split up the loot into whatever fraction we like. With indivisible items, new complications appear!

No Solution

For some instances (e.g., a single item that both thieves want), it is not possible to split the items proportionally.
Can we find a solution if one exists?

An indivisible detour II

Question

Can Alice efficiently split the loot of n items into two equal piles, if this is possible?

Answer

The fastest known algorithm for this needs exponential time, $O(n \cdot 2^{n/2}) \subseteq O(1.4145^n)$.

No faster solution

If Bob has the exact same valuations for the items as Alice, then an equal split is the only valid solution.

Can we achieve a slightly suboptimal solution in polynomial time?

An indivisible detour III

Weaker property

Proportionality up to one item: each thieve's share is worth at least half the loot with the most valuable item removed.

Algorithm

The thieves alternatively pick their most valuable item, starting with Alice.

Correctness

Divide the picking sequence into rounds os size 2.
Alice always picks a more valuable item than Bob in each round.
So, she receives at least half her value of the loot.
In each round, Bob picks a more valuable item than Alice picks in the next round. So, if we remove Alice's first picked item from the loot, Bob gets at least half the value.

Example: Three Thieves

Problem

Alice, Bob and Carol have robbed a warehouse and have to split a pile of divisible items. Each thief has their own valuation of the items.

Design an algorithm to split the pile so that each thief values their own pile as at least one third of the loot.

Example: Three Thieves

- Let us try do the same trick as in the case of two thieves. Say Alice splits the loot into three piles which she thinks are of equal value; then Bob and Carol each choose which pile they want to take.

- If they choose different piles, they can each take the piles they have chosen and Alice gets the remaining pile; in this case clearly each thief thinks that they got at least one third of the loot.

Example: Three Thieves

- But what if Bob and Carol choose the same pile?
- One might think that in this case, Alice can pick either of the other two piles, after which the remaining two piles are put together for Bob and Carol to split them as in the earlier problem with only two thieves.
- Unfortunately this does not work!

Example: Three Thieves

- Suppose that Alice splits the loot into three piles X, Y, Z, and that Bob thinks that

  $$X = 50\%, Y = 40\%, Z = 10\%$$

  of the total value, while Carol thinks that

  $$X = 50\%, Y = 10\%, Z = 40\%.$$

Example: Three Thieves

- Clearly both Bob and Carol choose pile X, so Alice can choose pile Y or Z.

- However, if Alice picks pile Y, then Bob will object that (in his eyes) only 60% of the loot remains, so he is not guaranteed to get at least one-third of the total.

- If instead Alice picks pile Z, then Carol will object for the same reason.

Example: Three Thieves

- Why did this fail?
- We need to ensure that both Bob and Carol think that two-thirds of the loot remains after Alice takes her pile.
- If Alice wants to take a pile and Bob *doesn't* think that two-thirds of the loot is left, what can we do?
- Give less than the whole pile to Bob instead, and try sharing the rest between Alice and Carol!

Example: Three Thieves

Algorithm

- Alice makes a pile X which she believes is $1/3$ of the whole loot.
- Alice proceeds to ask Bob whether he agrees that $X \leq 1/3$.
- If Bob says YES, then he would be happy to split the remainder of the loot (worth $\geq 2/3$) with one other thief.
  - Alice then asks Carol whether she thinks that $X \leq 1/3$.
  - If Carol says NO, then Carol takes X, and Alice and Bob split the rest.
  - If Carol says YES, then Alice takes X, and Bob and Carol split the rest.

Example: Three Thieves

Algorithm (continued)

- What if Bob says NO? Then Alice values pile X at $1/3$ of the total, but Bob believes it to be $> 1/3$.
- Now we ask Bob to reduce the pile X until he believes it to be $1/3$ of the total. Alice values the new pile as $< 1/3$, so she is happy to split the remainder of the loot (worth $> 2/3$) with one other thief.
- This is exactly the situation we had before, but with Alice and Bob's roles reversed!
  - Bob asks Carol whether she thinks that $X \leq 1/3$.
  - If Carol says NO, then Carol takes X, and Alice and Bob split the rest.
  - If Carol says YES, then Bob takes X, and Alice and Carol split the rest.

Example: *n* Thieves

Exercise

Try generalising this to *n* thieves.

Hint

There is a *nested recursion* happening even with 3 thieves!

Table of Contents

1. Admin
2. Solving problems using algorithms
3. Proofs
4. An example of the role of proofs
5. Puzzles

The role of proofs in algorithm design

Question

When do we need to give a **mathematical proof** that an algorithm we have just designed terminates and returns a solution to the problem at hand?

Answer

When this is not obvious by inspecting the algorithm using common sense!

Mathematical proofs are **NOT** academic embellishments; we use them to justify things which are not obvious to common sense!

Example: MERGE-SORT

Algorithm

We recursively apply the following algorithm to the subarray $A[\ell..r]$.

If $\ell = r$, i.e. the subarray has only one element, we simply exit. This is the base case of our recursion.

Otherwise:

- first define $m = \left\lfloor \frac{\ell+r}{2} \right\rfloor$, the midpoint of the subarray,
- then apply the algorithm recursively to $A[\ell..m]$ and $A[m+1..r]$, and
- finally merge the subarrays $A[\ell..m]$ and $A[m+1..r]$.

Example: Merge-Sort

- The depth of recursion in Merge-Sort is $\log_2 n$.
- On each level of recursion, merging all the intermediate arrays takes $O(n)$ steps in total.
- Thus, Merge-Sort always terminates, and in fact it terminates in $O(n \log_2 n)$ steps.
- Merging two sorted arrays always produces a sorted array, thus, the output of Merge-Sort will be a sorted array.
- The above is essentially a proof by induction, but we will generally not formalise proofs of (essentially) obvious facts.

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate.

- Sometimes it is **NOT** clear that an algorithm terminates in a reasonable number of steps.

- Sometimes it is **NOT** clear from a description of an algorithm why such an algorithm, after it terminates, produces a desired solution.

The role of proofs in algorithm design

- Proofs are needed for such circumstances; in a lot of cases they are **the only way** to know that the algorithm does the job.

- However, we will **not** prove the obvious (the CLRS textbook sometimes does just that, by sometimes formulating and proving trivial little lemmas). We will prove only what is genuinely nontrivial.

- What is the definition of trivial? Immediately obvious to a typical student in this class.

Table of Contents

1. Admin
2. Solving problems using algorithms
3. Proofs
4. An example of the role of proofs
5. Puzzles

Stable Matching Problem

- Suppose there are *n* hospitals in the state, and *n* new doctors have graduated from university. Each hospital wants to employ exactly one new doctor.

- Every hospital submits a list of preferences, which ranks all the doctors, **and** every doctor submits a list of preferences, which ranks all the hospitals.

- We'd like to assign all *n* doctors to different hospitals in order to "make everyone happy" in some sense.

Stable Matching Problem

Definition

A *matching* is an assignment of doctors to hospitals so that no doctor has more than one job and no hospital has more than one employee.

Definition

A *perfect matching* is a matching involving *all* doctors and *all* hospitals.

Our task is to design a perfect matching that somehow satisfies the hospitals and doctors, with regards to their preferences.

Stable Matching Problem

Question

Can we assign every doctor and every hospital their first preference?

Answer

Not necessarily!

- We'll have to lower our expectations. Let's aim for an allocation that doesn't make anyone too unhappy.
- If a hospital and doctor were both very unhappy with their allocations, they might leave our system and arrange hiring directly.

Stable Matching Problem

Definition

A *stable matching* is a perfect matching in which there are no two pairs (h, d) and (h', d') such that:

- hospital h prefers doctor d' to doctor d, **and**
- doctor d' prefers hospital h to hospital h'.

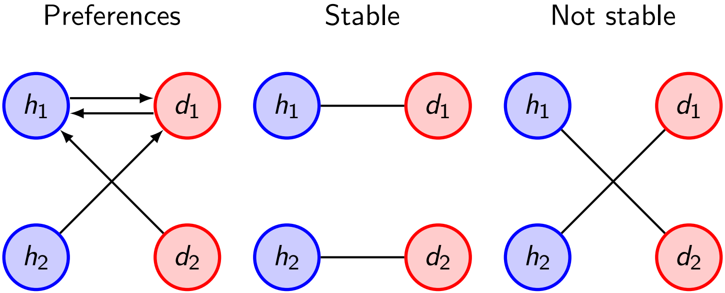
Stable matchings are self-enforcing; no *unmatched* pair of hospital h and doctor d' will leave.

We will design an algorithm which produces a stable matching.

Stable Matching Problem: Example 1

$h_1 : d_1, d_2$ $\qquad$ $d_1 : h_1, h_2$

$h_2 : d_1, d_2$ $\qquad$ $d_2 : h_1, h_2$



Preferences

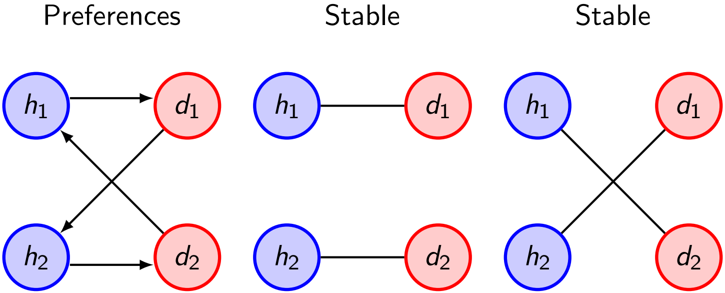
Stable

Not stable

Stable Matching Problem: Example 2

$h_1 : d_1, d_2$ $\qquad$ $d_1 : h_2, h_1$

$h_2 : d_2, d_1$ $\qquad$ $d_2 : h_1, h_2$



Stable Matching Problem: $n = 2$

Exercise

Up to symmetry, there are two more cases to consider. Solve them.

The following rules cover all situations with two hospitals and two doctors.

- If the hospitals prefer different doctors, assign each hospital their preferred doctor. Neither hospital wants to swap, so the matching is stable.
- The same applies vice versa, i.e. if the doctors prefer different hospitals.
- However, if both hospitals prefer the same doctor d **and** both doctors prefer the same hospital h, pair h with d and the other hospital with the other doctor. Neither h nor d wants to swap, so the matching is stable.

Stable Matching Problem: Gale - Shapley algorithm

Question

Given n hospitals and n doctors, how many ways are there to match them, without regard for preferences?

Answer

$n! \approx (n/e)^n$ – exponentially many in n ($e \approx 2.71$).

Question

Is it true that for every possible collection of n lists of preferences provided by all hospitals, and n lists of preferences provided by all doctors, a stable matching always exists?

Answer

YES, but this is **NOT** obvious!

Stable Matching Problem: Gale - Shapley algorithm

Question

Can we find a stable matching in a reasonable amount of time?

Answer

YES, using the **Gale - Shapley algorithm**.

- Produces pairs in stages, with possible revisions
- A hospital which is not currently employing a doctor will be called *vacant*.
- Hospitals will be offering jobs to doctors. Doctors will decide whether they accept a job offer or not.
- Start with all hospitals vacant.

Stable Matching Problem: Gale - Shapley algorithm

- While there is a vacant hospital which has not offered jobs to all doctors, pick any such vacant hospital and call it h.
- Hospital h offers a job to the highest ranking doctor d on its list, ignoring any doctors to whom it has already offered a job.
    - If d is not yet employed, she accepts the job (at least tentatively).
    - Otherwise, if d is already employed at a different hospital h':
        - if d prefers the new hospital h to her current hospital h', she quits h' and joins h (making h' vacant)
        - otherwise, since d prefers her current hospital h', she rejects the offer from h and stays at h'.

Stable Matching Problem: Gale - Shapley algorithm

Claim 1

The algorithm terminates after $\leq n^2$ rounds.

Proof

- In every round of the algorithm, one hospital offers a job to one doctor.
- Every hospital can make an offer to a doctor at most once.
- Thus, every hospital can make at most n offers.
- There are n hospitals, so in total they can make $\leq n^2$ offers.
- Thus there can be no more than n^2 rounds.

Stable Matching Problem: Gale - Shapley algorithm

Claim 2

The algorithm produces a perfect matching, i.e., every hospital is eventually paired with a doctor (and thus also every doctor is paired to a hospital).

Proof

- Assume that the algorithm has terminated, but hospital h is still vacant.
- This means that h has already offered a job to every doctor.
- A doctor is unemployed only if no hospital has offered them a job, so all doctors must have a job.
- But this would mean that n doctors are paired with all of n hospitals, so h cannot be vacant.
- This is a contradiction, completing the proof.

Stable Matching Problem: Gale - Shapley algorithm

Claim 3

The matching produced by the algorithm is stable.

- Recall that each hospital makes offers to doctors in order from its most preferred to its least preferred.
- Therefore, the sequence of doctors employed at a particular hospital is in this order also.
- Each doctor is initially unemployed, then takes the first offer made to them, and only ever moves to a hospital they prefer to their current employer.
- Thus, each doctor is paired with hospitals in order from their least preferred to their most preferred.

Stable Matching Problem: Gale - Shapley algorithm

Proof

We will prove that the matching is stable using *proof by contradiction*.

Assume that the matching is not stable. Thus, there are two pairs (h, d) and (h', d') such that:

- h prefers d' over d and
- d' prefers h over h'.

Stable Matching Problem: Gale - Shapley algorithm

Proof (continued)

- Since h prefers d' over d, it must have made an offer to d' before offering the job to d.
- Since h is paired with d, doctor d' must have either:
    - rejected h because they were already at a hospital they prefer to h, or
    - accepted h only to later rescind this and accept an offer from a hospital they prefer to h.
- In both cases d' would now be at a hospital which they prefer over h.
- This is a contradiction, completing the proof.

Table of Contents

1. Admin

2. Solving problems using algorithms

3. Proofs

4. An example of the role of proofs

5. Puzzles

Puzzle

Why puzzles? It is a fun way to practice problem solving!

Problem

Tom and his wife Mary went to a party where nine more couples were present.

- Not every one knew everyone else, so people who did not know each other introduced themselves and shook hands.
- People who knew each other from before did not shake hands.
- Later that evening Tom got bored, so he walked around and asked all other guests (including his wife) how many hands they had shaken that evening, and got 19 different answers.
- How many hands did Mary shake?
- How many hands did Tom shake?

That's All, Folks!!