"For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing." - Francis Sullivan
Welcome back!!

We hope you are all alright!

This class is completely on-line. Lectures are in Canvas via echo360. TAs will do office hours in Teams.

If you have issues (illness, uncertainties, timing, anything really) please don’t hesitate to let me (Wim Bohm) know (e-mail, office hours, ...)

This is a 3 credit course with no recitations. We have 2 GTAs:

Anju Gopinath
William Scarbro

Who will help you with assignments
Course Objectives

**Algorithms:**
- Design – strategies for algorithmic problem solving
- Reasoning about algorithm **correctness**
- Analysis of **time** and **space complexity**
- Implementation – create an implementation that respects the **runtime analysis**. In this class a program has to be correct and has to have the optimal complexity

**Algorithmic Approaches / Classes:**
- Greedy
- Divide and Conquer
- Dynamic programming

**Parallel Algorithms:**
- Dynamic Multi-threading

*(if time permits) Problem Classes:*
- Reduction, P: Polynomial, NP: Non deterministic Polynomial
Grading

Programming Assignments  15%
Written Assignments       15%
Quizzes                   20%
Exams                     50%

See  CS320 web site:
https://www.cs.colostate.edu/~cs320
Implementation

Programs will be written in Python:
- Powerful **data structures**
  - tuples, dictionaries, (array)lists
- Simple, easy to learn syntax
- Highly readable, compact code
- An extensive standard library
- Strong support for integration with other languages (C, C++, Java) and libraries (numpy, jupyter, CUDA)

We assume you are familiar with Python (CS220)!
Python vs. e.g. Java

What makes Python different from Java?
- Java is statically typed, i.e. variables are bound to types at compile time. This avoids run time errors, but makes java programs more rigid.
- Python is dynamically typed, i.e. a variable takes on some type at run time, and its type can change. A variable can be of one type somewhere in the code and of another type somewhere else.
  ```python
  f = open(filename)
  for line in f:
    # line is a String here, split it using " " as delimiter
    line = line.strip().split(" ")
    # line is an (Array)List of Strings here
  ```
- This makes python programs more flexible, but can cause strange run time errors, e.g. when a caller expects a return value but the called function does not return one.
Our approach to problem solving

- Formulate it with precision (usually using mathematical concepts, such as sets, relations, and graphs)
- Design an algorithm and its main data structures
- Prove its correctness
- Analyze its complexity (time, space)
  - Improve the initial algorithm (in terms of complexity), preserving correctness

- Implement it, preserving the analyzed complexity! In the lab PAs we will test for that. So in this course we check for correctness and complexity of your PAs.
Our first problem: matching

Two parties e.g., companies and applicants

- Each applicant has a **preference list** of companies
- Each company has a **preference list** of applicants
- A possible scenario:
  
  cA offers job to aA
  
  aA accepts, but now gets offer from cX
  
  aA likes cX more, retracts offer from cA

We would like a systematic method for assigning applicants to companies— **stable matching**

- A system like this is e.g. in use for matching medical residents with hospitals
Stable Matching

**Goal.** Given a set of preferences among companies and applicants, design a *stable matching algorithm*.

**Unstable pair:** applicant $x$ and company $y$ are an unstable pair *(not in the current matching)* if:
- Both $x$ prefers $y$ to its assigned company
- And $y$ prefers $x$ to one of its selected applicants.

**Stable assignment.** Assignment without unstable pairs.
- Natural and desirable condition.
Is some control possible?

Given the preference lists of applicants $A$ and companies $C$, can we assign $A$s to $C$s such that

for each $C$
  for each $A$ not scheduled to work for $C$
    either $C$ prefers all its students to $A$
    or $A$ prefers current company to $C$

If this holds, then what?
Stable state

Given the preference lists of applicants $A$ and companies $C$, can we assign $A$s to $C$s such that

for each $C$
  for each $A$ not scheduled to work for $C$
    $C$ prefers all its students to $A$
    or $A$ prefers current company to $C$

If this holds, there is no unstable pair, and therefore individual self interest will prevent changes in student / company matches:

Stable state
Simplifying the problem

Matching students/companies problem messy:

- Company may look for **multiple** applicants, students looking for a **single** internship
- Maybe there are **more jobs** than applicants, or **fewer jobs** than applicants
- Maybe some applicants/jobs are **equally** liked by companies/applicants (partial orders)

Formulate a "bare-bones" version of the problem: match n men and n women
Stable Matching Problem: n women and n men

**Perfect matching:** Each man matched with exactly one woman, and each woman matched with exactly one man.

**Stability:** no incentive for some pair to undermine the assignment.

- A pair \((m,w)\) NOT IN THE CURRENT MATCHING is an **instability** if BOTH \(m\) and \(w\) prefer each other to current partners in the matching, i.e.:
  - BOTH \(m\) and \(w\) can improve their situation

**Stable matching:** perfect matching with no unstable pairs. **Stable matching problem (Gale, Shapley 1962):**
Given the preference lists of \(n\) men and \(n\) women, find a stable matching if one exists.
The Stable Matching Problem

Problem: Given n men and n women where
- Each man lists women in total order of preference
- Each woman lists men in total order of preference

A total order (remember CS220?) allows the elements of the set to be linearly ordered. Do you know an example? Do you know a counter example?

Men's Preference Profile

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xavier</td>
<td>Amy</td>
<td>Bertha</td>
</tr>
<tr>
<td>Yancey</td>
<td>Bertha</td>
<td>Amy</td>
</tr>
<tr>
<td>Zeus</td>
<td>Amy</td>
<td>Bertha</td>
</tr>
</tbody>
</table>

Women's Preference Profile

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>Yancey</td>
<td>Xavier</td>
</tr>
<tr>
<td>Bertha</td>
<td>Xavier</td>
<td>Yancey</td>
</tr>
<tr>
<td>Clare</td>
<td>Xavier</td>
<td>Yancey</td>
</tr>
</tbody>
</table>

find a stable matching of all men and women
Create all possible perfect matchings and check (in)stability

\[
\begin{align*}
\{ (X,A), (Y,B), (Z,C) \} & \quad \text{Stable (neither Z nor C can improve)} \\
\{ (X,A), (Y,C), (Z,B) \} & \quad \text{Instability: } (Y,B) \quad Y \text{ prefers } B \quad \text{and } B \text{ prefers } Y \\
\{ (X,B), (Y,A), (Z,C) \} & \quad \text{Stable} \\
\{ (X,B), (Y,C), (Z,A) \} & \quad \text{Instability: } (X,A) \\
\{ (X,C), (Y,A), (Z,B) \} & \quad \text{Instability: } (X,B) \\
\{ (X,C), (Y,B), (Z,A) \} & \quad \text{Instability: } (X,A)
\end{align*}
\]
Men: \( M = \{ m_1, \ldots, m_n \} \)  
Women: \( W = \{ w_1, \ldots, w_n \} \)

The Cartesian Product \( M \times W \) is the set of all possible ordered pairs.

A matching \( S \) is a set of pairs (subset of \( M \times W \)) such that each \( m \) and \( w \) occurs in at most one pair.

A perfect matching \( S \) is a set of pairs (subset of \( M \times W \)) such that each individual occurs in exactly one pair.

How many perfect matchings are there?

\[
\begin{array}{cccc}
   n & n-1 & n-2 & 1 \\
   m1 & m2 & m3 & \ldots & mn \\
\end{array}
\]
Instability

Given a perfect match, eg

\[ S = \{ (m_1,w_1), (m_2,w_2) \} \]

But \( m_1 \) prefers \( w_2 \) and \( w_2 \) prefers \( m_1 \)

\((m_1,w_2)\) is an instability for \( S \)

(notice that \((m_1,w_2)\) is not in \( S \))

\( S \) is a stable matching if:
- \( S \) is perfect
- and there is no instability in \( S \)
Example 1

$m_1$: $w_1, w_2$  \quad $m_2$: $w_1, w_2$  \\
$w_1$: $m_1, m_2$  \quad $w_2$: $m_1, m_2$

What are the perfect matchings?
Example 1

$m_1$: $w_1, w_2$  \quad $m_2$: $w_1, w_2$

$w_1$: $m_1, m_2$  \quad $w_2$: $m_1, m_2$

1. $\{ (m_1, w_1), (m_2, w_2) \}$
2. $\{ (m_1, w_2), (m_2, w_1) \}$

which is stable/instable?
Example 1

$m_1$: $w_1, w_2$  \hspace{1cm} $m_2$: $w_1, w_2$

$w_1$: $m_1, m_2$  \hspace{1cm} $w_2$: $m_1, m_2$

1. $\{(m_1, w_1), (m_2, w_2)\}$ stable, WHY?
2. $\{(m_1, w_2), (m_2, w_1)\}$ unstable, WHY?
Example 2

$m_1$: $w_1, w_2$  
$m_2$: $w_2, w_1$

$w_1$: $m_2, m_1$  
$w_2$: $m_1, m_2$

1. $\{ (m_1,w_1), (m_2,w_2) \}$
2. $\{ (m_1,w_2), (m_2,w_1) \}$

which is / are instable/stable?

both are stable!

1: $w_1$ prefers $m_2$ but $m_2$ prefers $w_2$, $w_2$ prefers $m_1$ but $m_1$ prefers $w_1$
2: $m_1$ prefers $w_1$ but $w_1$ prefers $m_2$, $m_2$ prefers $w_2$ but $w_2$ prefers $m_1$

Conclusion?

Sometimes there is more than 1 stable matching
Example 3

\[ m_1: w_1, w_2, w_3 \quad m_2: w_2, w_3, w_1 \quad m_3: w_3, w_1, w_2 \]
\[ w_1: m_2, m_1, m_3 \quad w_2: m_1, m_2, m_3 \quad w_3: m_1, m_2, m_3 \]

Is \( \{ (m_1,w_1), (m_2,w_2), (m_3,w_3) \} \) stable?

Is \( \{ (m_1,w_2), (m_2,w_1), (m_3,w_3) \} \) stable?

Do this one yourself.
Questions…

- Given a preference list, does a stable matching exist?
- Can we efficiently construct a stable matching if there is one?
- A naive algorithm:

  for S in the set of all perfect matchings :
  	if S is stable : return S
  return None

Is this algorithm correct?
What is its running time?
Towards an algorithm

initially: no match

An unmatched man $m$ proposes to the woman $w$ highest on his list.
Will this be part of a stable matching?
Towards an algorithm

initially: no match

An unmatched man m proposes to the woman w highest on his list.
Will this be part of a stable matching?
   Not necessarily: w may like some m’ better, AND?

So w and m will be in a temporary state of engagement.

w is prepared to change her mind when a man higher on her list proposes.
While not everyone is matched...

An unmatched man $m$ proposes to the woman $w$ highest on his list that he hasn't proposed to yet.

Why is that important?

If $w$ is free, they become engaged

If $w$ is engaged to $m'$:
  - If $w$ prefers $m'$ over $m$, $m$ stays free
  - If $w$ prefers $m$ over $m'$, $(m, w)$ become engaged
The Gayle-Shapley algorithm\(^1\)

Initialize each person to be free.

\begin{verbatim}
while (some man is free and hasn't proposed to every woman)
  Choose such a man \( m \)
  \( w = \) highest-ranked woman on \( m \)'s list to whom \( m \) has not yet proposed
  if (\( w \) is free)
    (\( m, w \)) become engaged
  else if (\( w \) prefers \( m \) to her fiancé \( m' \))
    (\( m, w \)) become engaged, \( m' \) becomes free
  else
    \( m \) remains free
\end{verbatim}

A few non-obvious questions:

How long does it take?

Does the algorithm return a stable matching?

Does it even return a perfect matching?

Each woman w remains engaged from the first proposal
and the sequence of w-s partners gets better
Each man proposes to less and less preferred women and will not propose to the same woman twice

Initialize each person to be free.
while (some man is free and hasn't proposed to every woman)
  Choose such a man m
  w = highest-ranked woman on m's list to whom m has not yet proposed
  if (w is free)
    (m,w) become engaged
  else if (w prefers m to her fiancé m')
    (m,w) become engaged, m' becomes free
  else
    m remains free
Observations

Initialize each person to be free.

while (some man is free and hasn't proposed to every woman)
  Choose such a man m
  w = highest-ranked woman on m's list to whom m has not yet proposed
  if (w is free)
    (m,w) become engaged
  else if (w prefers m to her fiancé m')
    (m,w) become engaged, m' becomes free
  else
    m remains free

Claim. The algorithm terminates after at most $n^2$ iterations of the while loop.
Observations

Initialize each person to be free.

while (some man is free and hasn't proposed to every woman)
    Choose such a man m
    w = highest-ranked woman on m's list to whom m has not yet proposed
    if (w is free)
        (m,w) become engaged
    else if (w prefers m to her fiancé m')
        (m,w) become engaged, m' becomes free
    else
        m remains free

Claim. The algorithm terminates after at most n² iterations of the while loop.

At each iteration a man proposes (only once) to a woman he has never proposed to, and there are only n² possible pairs (m,w)

WHY ONLY n²?

only n choices for each of the n men
Initialize each person to be free.

while (some man is free and hasn't proposed to every woman)

    Choose such a man \( m \)

    \( w = \) highest-ranked woman on \( m \)'s list to whom \( m \) has not yet proposed

    if \( (w \) is free)
        \( (m,w) \) become engaged
    else if \( (w \) prefers \( m \) to her fiancé \( m' \))
        \( (m,w) \) become engaged, \( m' \) becomes free
    else
        \( m \) remains free

When the loop terminates, the matching is **perfect**

Proof: **By contradiction.** Assume there is a free man, \( m \).

Because the loop terminates, \( m \) proposed to all women
But then all women are engaged, hence there is no free man

\( \Rightarrow \) **Contradiction**
Proof of Correctness: Stability

**Claim.** No instable pairs. **Proof.** (by contradiction)

- Suppose \((m, w)\) is an instable pair: each prefers each other to partner in Gale-Shapley matching \(S^*\).

- **Case 1:** \(m\) never proposed to \(w\).
  - \(\Rightarrow m\) prefers his GS partner \(w’\) to \(w\)
  - \(\Rightarrow (m, w)\) is not instable.

- **Case 2:** \(m\) proposed to \(w\).
  - \(\Rightarrow w\) rejected \(m\) (right away or later)
  - \(\Rightarrow w\) prefers her \(S^*\) partner \(m’\) to \(m\).
  - \(\Rightarrow (m, w)\) is not instable.

- In either case \((m, w)\) is not instable, a contradiction. □
Which solution?

$m_1$: $w_1, w_2$  \hspace{1cm}  $m_2$: $w_2, w_1$

$w_1$: $m_2, m_1$  \hspace{1cm}  $w_2$: $m_1, m_2$

Two stable solutions

1: \{(m_1,w_1), (m_2,w_2)\}

2: \{(m_1,w_2), (m_2,w_1)\}

GS will always find one of them (which?)

When will the other be found?
Stable matching problem. Given n men and n women and their preferences, find a stable matching if one exists.

Gale-Shapley algorithm. Guaranteed to find a stable matching for any problem instance.
The stable matching problem is symmetric w.r.t. to men and women, but the GS algorithm is asymmetric.

There is a certain unfairness in the algorithm: If all men list different women as their first choice, they will end up with their first choice, regardless of the women's preferences (see example 3).
Non-determinism

Notice the following line in the GS algorithm:
while (some man is free and hasn't proposed to every woman)
Choose such a man m

The algorithm does not specify WHICH

Still, it can be shown that all executions of the algorithm find the same stable matching.

This ends our discussion of stable matching.
Representative Problems
Remember the problem solving paradigm

1. **Formulate the problem with precision** (usually using mathematical concepts, such as sets, relations, and graphs, costs, benefits, optimization criteria)

2. (Re)design an algorithm
3. Prove its correctness
4. Analyze its complexity
5. Implement respecting the derived complexity

Often, steps 2-5 are repeated, to improve efficiency

Our first algorithm for Stable Matching was exponential,
Our second was polynomial (quadratic)
Interval Scheduling

You have a resource (hotel room, printer, lecture room, telescope, manufacturing facility, professor...)

There are requests to use the resource in the form of start time $s_i$ and finish time $f_i$, such that $s_i < f_i$

Objective: grant as many requests as possible.
Two requests $i$ and $j$ are compatible if they don't overlap, i.e.,

$$f_i \leq s_j \text{ or } f_j \leq s_i$$
Interval Scheduling

**Input.** Set of jobs with start times and finish times.

**Goal.** Find **maximum cardinality** subset of compatible jobs.

What happens if you pick the first starting (a)?, the smallest (c)? What is the optimum?
Algorithmic Approach

The interval scheduling problem is amenable to a very simple solution.

Now that you know this, can you think of it?

Hint: Think how to pick a first interval while preserving the longest possible free time...
Weighted Interval Scheduling

**Input.** Set of jobs with start times, finish times, and profits.

**Goal.** Find maximum profit subset of compatible jobs.
Stable matching was defined as matching elements of two disjoint sets.

We can express this in terms of graphs.

A graph is **bipartite** if its nodes can be partitioned in two sets $X$ and $Y$, such that the edges go from an $x$ in $X$ to a $y$ in $Y$. 
Bipartite Matching

**Input.** Bipartite graph.

**Goal.** Find **maximum cardinality** matching.

Matching in bipartite graphs can model **assignment** problems, e.g., assigning jobs to machines, where an edge between a job $j$ and a machine $m$ indicates that $m$ can do job $j$, or professors and courses.

How is this different from the stable matching problem?
- Not perfect, $|X| \neq |Y|$
- No preferences, less information
Independent Set

**Input.** Graph.
**Goal.** Find maximum cardinality independent set: subset of nodes such that no two are joined by an edge

Can you formulate interval scheduling as an independent set problem?

Yes, interval = node, edge if two intervals overlap

If so, how could you solve the interval scheduling problem?

Pose it as an independent set problem, we call this reduction
Independent set problem

- There is no known efficient way to solve the independent set problem.

- But we just said: we can formulate interval scheduling as independent set problem..... ???

- What does "no efficient way" mean?
  The only solution we have so far is trying all subsets and finding the largest independent one.

- How many subsets of a set of n nodes are there? (CS220: $2^n$) WHY?
Representative Problems / Complexities

Looking ahead...

- **Interval scheduling:** $n \log(n)$ greedy algorithm.

- **Weighted interval scheduling:** $n \log(n)$ dynamic programming algorithm.

- **Independent set:** NP (no known polynomial algorithm exists).
Algorithm

Algorithm: effective procedure

- mapping input to output

**effective: unambiguous, executable**

- Turing defined it as: "like a Turing machine"
- program = effective procedure

Is there an algorithm for every possible problem?

No, the problem must be effectively specified: "how many angels can dance on the head of a pin?" not effective. **Even if** it is effectively specified, there is not always an algorithm to provide an answer. This occurs often for programs analyzing programs (examples?)
Ulam's problem

```python
def f(n):
    if (n==1) return 1
    elif (odd(n)) return f(3*n+1)
    else return f(n/2)
```
Ulam's problem

def f(n):
    if (n==1) return 1
    elif (odd(n)) return f(3*n+1)
    else return f(n/2)

Steps in running f(n) for a few values of n:
1
2, 1
3, 10, 5, 16, 8, 4, 2, 1
4, 2, 1
5, 16, 8, 4, 2, 1
6, 3, 10, 5, 16, 8, 4, 2, 1
7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
8, 4, 2, 1
9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
10, 5, 16, 8, 4, 2, 1

Does f(n) always stop?
Ulam's problem

def f(n):
    if (n==1) return 1
    elif (odd(n)) return f(3*n+1)
    else return f(n/2)

Nobody has found an n for which f does not stop
Nobody has found a proof that f stops for all n
(so there can be no algorithm deciding this)

A generalization of this problem has been proven to be undecidable. It is called the Halting Problem.
A problem P is undecidable if there is no algorithm that produces P(x) for every possible input x
The Halting Problem is undecidable

Given a program $P$ and input $x$
will $P$ stop on $x$?

We can prove (cs420):
the halting problem is undecidable

i.e. there is no algorithm $\text{Halt}(P, x)$ that for any program $P$ and input $x$ decides whether $P$ stops on $x$.

But for some “nice” programs, we can prove they halt, e.g.:

    for i in range(100): print(i)
Intractability

Suppose we have a program,
- does it execute a in a reasonable time?
- E.g., towers of Hanoi (cs200).

Three pegs, one with \( n \) smaller and smaller disks, move (1 disk at the time) to another peg without ever placing a larger disk on a smaller peg.

Monk: before a tower of Hanoi of size 100 is moved, the world will have vanished.
def hanoi(n, from, to):
    if (n>0):
        via = 6 - from - to
        hanoi(n-1,from, via)
        print "move disk", n, " from", from, " to ", to
        hanoi(n-1,via,to);
f(n): #moves in hanoi

f(n) = # moves for tower of size n

f(n) = 2f(n-1) + 1, f(1)=1
f(1) = 1, f(2) = 3, f(3) = 7, f(4) = 15

f(n) = 2^{n-1}

How can you show that?
By induction (cs220)

Was the monk right?
2^{100} moves, say 1 per second.....
How many years?

2^{100} \sim 10^{30} \sim 10^{25} \text{ days} \sim 3.10^{22} \text{ years}

more than the age of the universe
Is there a better algorithm?

THE ONE MILLION DOLLAR QUESTION IN THIS CLASS
Is there a better algorithm?

Pile(n-1) must be

off peg1

and

completely on one other peg

before disk n can be moved to its destination

so all moves are necessary
**Algorithm complexity**

Measures in units of **time** and **space**

Linear Search X in dictionary D

\[
i = 1 \\
\text{while not at end and } X \neq D[i]: \\
i = i + 1
\]

CS220: We don't know if X is in D, and we don't know where it is, so we can only give **worst** or **average** time bounds

We don't know the time for atomic actions, so we only determine **Orders of Magnitude**
Linear Search: time and space complexity

Space: n locations in D plus some local variables

Time:
In the worst case we search all of D, so the loop body is executed n times

In average case analysis we compute the expected number of steps: i.e., we sum the products of the probability of each option and the time cost of that option. In the average case the loop body is executed about n/2 times

\[ \sum_{i=1}^{n} \frac{1}{n} \times i = \frac{1}{n} \sum_{i=1}^{n} i = \frac{n(n+1)/2}{n} \approx \frac{n}{2} \]