abhi

stable matching
Stable Matching
definition: matchings

- $M = \{ m_1, \ldots, m_n \}$
- $W = \{ w_1, \ldots, w_n \}$

$S = \{ (m_i, w_j) \}$ set of pairs.

- Matching if $m_i$ only occur in one pair in $S$.

- Perfect if $|S| = n$. 
definition: matchings

\[ M = \{m_1, \ldots, m_n\} \]
\[ W = \{w_1, \ldots, w_n\} \]
\[ S = \{(m_{i_1}, w_{j_1}), \ldots, (m_{i_k}, w_{i_k})\} \]

Each \( m_i \) appears only one in a pairing.
A matching is perfect if every \( m_i \) appears.
definition: preferences

\[ M = \{m_1, \ldots, m_n\} \]

Each \( m_i \) has a preference list on the set \( W \)

\("w_i \prec m_i \prec w_2": m_i \text{ prefers } w_2 \text{ to } w_i\)
example: preferences

\[ M = \{ m_1, \ldots, m_n \} \]

\( m_i \) has a preference relation on the set \( W \)

\[ w_1 \prec_{m_i} w_4 \prec_{m_i} w_2 \prec_{m_i} w_8 \cdots w_n \]
\[ S = \left\{ (D, V), (C_1, H) \right\} \]
def: instability

\[ S = \left\{ \left( \begin{array}{c} w' \\ m_1 \end{array} \right), \left( \begin{array}{c} m' \\ w^* \end{array} \right) \right\} \]

\( S \) is unstable if \( \exists (m^*, w^*) \in S, (m^*, w'), (m', w^*) \in S \)

and

1. \( m^* \) prefers \( w^* \) to \( w' \)
2. \( w^* \) prefers \( m^* \) to \( m' \)

\( S \) is a stable matching if there are no such triples.
def: instability

\[ S = \begin{cases} \begin{pmatrix} \text{ bother } & \text{ coat } \\ \text{ human } & \text{ scar } \end{pmatrix} & (m^*, w^*) \not\in S \\ \begin{pmatrix} \text{ bother } & \text{ coat } \\ \text{ human } & \text{ scar } \end{pmatrix} & w' <_{m^*} w^* \\ \begin{pmatrix} \text{ bother } & \text{ coat } \\ \text{ human } & \text{ scar } \end{pmatrix} & m' <_{w^*} m^* \end{cases} \]
prove: for every input there exists a stable matching.
Start with an empty matching $S$
While there is an unmatched $m$ who has not exhausted his preference list
Let $w$ be the first $f$ on $m$'s list who has not asked
If $w$ is unpaired, PAIR($m$, $w$)
If $(m', w) \in S$ and $w$ prefers $m$ to $m'$
Breakup $(m', w)$
PAIR $(m, w)$
StableMatch($M, W, \prec_m, \prec_w$)

1. Initialize all $m, w$ to be FREE
2. while $\exists$FREE($m$) and hasn’t proposed to all $W$
   3. do Pick such an $m$
      4. Let $w \in W$ be highest-ranked to whom $m$ has not yet proposed
      5. if FREE($w$)
         6. then Make a new pair $(m, w)$
      7. elseif $(m', w)$ is paired and $m' \prec_w m$
         8. do Break pair $(m', w)$ and make $m'$ free
            9. Make pair $(m, w)$
   10. return Set of pairs
Each $m$ only asks each $w$ once.

$|W| = n$.

$\Rightarrow \Theta(n^2)$ iterations of the loop.
proposal algorithm ends

\[ O(n^2) \] steps

each m proposes at most once to each w.
each m proposes at most n times.
size of M is n.
output is a matching

1. Each m only paired w/ one w.
2. Why is each w paired w/ only 1 m??
   every time w is paired, she is single.
If $\exists a$ an unmatched $m$, some $\varphi$ has not been asked.
output is perfect

if $\exists m$ who is free, then $\exists w$ who has not been asked
Output is stable = S

Suppose not. That means ∃ (m*, w*) ∈ S and (m*, w'), (m', w*) ∈ S such that m* prefers w* to w' & w* prefers m* to m'.

Consider the moment when (m*, w) are paired in the execution.

m* was single. m* must have already asked w*.

w* must have rejected m* in favor of m'.

⇒ m* ⪯w* m'

⇒ either m' = m' or m' ⪯w' m'

⇒ m* ⪯w* m', which contradicts

(Gale–Shapley)
output is stable

\[ \exists (m^*, w), (m, w^*) \in S \quad w \prec_{m^*} w^* \quad m \prec_{w^*} m^* \]
output is stable

\[ \exists (m^*, w), (m, w^*) \in S \quad w \sim_{m^*} w^* \quad m \sim_{w^*} m^* \]

- \( m^* \) last proposal was to \( w \)
- but \( w \sim_{m^*} w^* \) and so \( m^* \) must have already asked \( w^* \)
- and must have been rejected by \( m^* \sim_{w^*} m' \)
- then either \( m' \sim_{w^*} m \) or \( m' = m \)
- which contradicts assumption \( m \sim_{w^*} m^* \)
Proposer wins

\[ (D, U), (C, H) \]

\[ \Rightarrow \text{ propose} \]

\[ (C, U), (D, H) \]
Proposer wins
Remarkable theorem

w is valid for m: \( \exists \) some stable matching \( S \) s.t. \( (m_i, w) \in S \)

\( \text{best}(m): w \text{ s.t. } (m_i, w) \text{ is valid and every } w^* \stackrel{m}{\succ} w \text{ is not valid.} \)

\( S^* : \{ (m_i, \text{best}(m_i)) \} \) \( \forall m \)

Thm: GS returns \( S^* \) (every execution of it).
GS is man-optimal.

Proof: Consider some execution $E$ of GS that returns $S = S^*$. 

⇒ some $m$ is not matched with their best valid match. 

⇒ some $w$ must have rejected a valid $m$. 

⇒ consider the first time some $w$ rejects a valid $m$. 

@ this point, $(m', w)$ have been matched. 

But $w$ is valid for $m$, so $\exists (m, w) \in S'$, who does $m'$ match with in $S'$? $(m', w') \in S'$. 

⇒ since this is the first rejection in $E$, then 

$w'$ could not have rejected $m'$ in $E$ at this point. 

(Toke fixed).
GS matching vs \( \text{Wopt} \)

\(
\text{max-optimal.}
\)

- The max-optimal matching is the most possible stable matching for \( w \).
a new technique for algorithm design
MergeSort(n)

- **<base case>**
- MergeSort(n/2)  **<left half>**
- MergeSort(n/2)  **<right half>**
- Merge(left, right)  **<combine>**
$T(n) = 2T(n/2) + O(n)$
\[
\text{BEST}_n = \min \left\{ \begin{array}{l}
\text{BEST}_0 + S_{1,n}^2 \\
\text{BEST}_1 + S_{2,n}^2 \\
\text{BEST}_2 + S_{3,n}^2 \\
\vdots \\
\text{BEST}_{\ell - 1} + S_{\ell,n}^2 \\
\vdots \\
\text{BEST}_{n-1} + S_{n,n}^2 
\end{array} \right. 
\]
solving \( \text{BEST}_n \) can be reduced to solving \( n-1 \) \( \text{BEST}_i \) problems and combining the answer in linear time.
HUFFMAN

Finding an optimal code for an X character alphabet

solved by

Finding an optimal code for an X-1 character alphabet
WE HAVE BEEN SOLVING PROBLEM A BY SOLVING SMALLER VERSIONS OF PROBLEM A
GENERAL IDEA:

SOLVE PROBLEM A BY SOLVING PROBLEM B
REDUCTION

\text{PROBLEM}_a \leq_{f(n)} \text{PROBLEM}_b
REDUCTION

\[
\text{PROBLEM}_a \leq f(n) \ \text{PROBLEM}_b
\]

\[\exists c, d \ \ T(\text{PROBLEM}_a(n)) \leq f(n) + cT(\text{PROBLEM}_b(dn))\]
MAXIMUM BIPARTITE MATCHING
EDGE-DISJOINT PATHS
\( \text{maxBIPARTITE} <_{e+v} \text{maxflow} \)

\( \text{maxedgedisj} <_{e+v} \text{maxflow} \)
TRIPLET PROBLEM

given numbers \((x_1, \ldots, x_n)\)

determine whether there is a triplet \((x_i, x_j, x_k)\)
such that \(x_i + x_j + x_k = 0\)
3, -6, 5, 2, 6, 8, -1, 12, 7, -10, -3, 14
EASY TO SOLVE IN $O(n^3)$
EASY TO SOLVE IN $O(n^2)$
COLINEARITY

given points in the plane \((x_1, y_1), \ldots, (x_n, y_n)\)

determine whether any 3 are co-linear but not horizontal.
HOW CAN WE COMPARE 2 PROBLEMS?

\[ \text{PROBLEM}_a \leq f(n) \text{ PROBLEM}_b \]

\[ T(\text{PROBLEM}_a(n)) \leq f(n) + cT(\text{PROBLEM}_b(dn)) \]
3, -6, 5, 2, 6, 8, -1, 12, 7, -10, -3, 14
\[ T = \{3, -6, 5, 2, 6, 8, -1, 12, 7, -10, -3, 14\} \]
$T = \{ 3, -6, 5, 2, 6, 8, -1, 12, 7, -10, -3, 14 \}$

$P =$
T is a TRIPLET-set if and only if P is a COLINEAR set.
SEGMENT PARTITION
SEGMENT PARTITION

Problem: Given a set of line segments in the plane, determine if there exists a line that partitions the segments into two sets.
Consider the following segment splitting problem: Given a collection of line segments in the plane, is there a line that does not hit any segment and splits the segments into two non-empty subsets?

To show that this problem is $\text{NP-hard}$ we start with the collection of points produced by our last reductions. Replace each point by a "hole" between two horizontal line segments. To make sure that the only way to split the segments is by passing through three colinear holes, we build two "gadgets" each consisting of five segments to cap off the left and right ends as shown in the figure below.

Top: 3n points, three on a non-horizontal line.
Bottom: 3n + 13 segments separated by a line through three colinear holes.

This reduction could be performed in linear time if we could make the holes infinitely small, but computers can't really deal with infinitesimal numbers. On the other hand, if we make the holes too big, we might be able to thread a line through three holes that don't quite line up. I won't go into details, but it is possible to compute a working hole size in $O(n \log n)$ time by first computing the distance between the closest pair of points.

Thus, we have a valid reduction from $\text{NP-hard}$ to segment splitting by way of colinearity:

\[
\text{set of } n \rightarrow \text{set of } 3n \text{ points} \rightarrow \text{set of } 3n + 13 \text{ segments}
\]

\[
\text{\texttt{\textbf{\textsl{SPLITTABLE?}}}}
\]

\[
\text{\texttt{\textbf{\textsl{T}}}} \leftrightarrow \text{\texttt{\textbf{\textsl{F}}}}
\]

\[
\text{\texttt{\textbf{\textsl{T}}}} \leftrightarrow \text{\texttt{\textbf{\textsl{F}}}}
\]

\[
\text{\texttt{\textbf{\textsl{T}}}} \leftrightarrow \text{\texttt{\textbf{\textsl{F}}}}
\]

Finally, suppose we want to know whether a robot can move from one position and location to another. To make things simple, we'll assume that the robot is just a line segment, and the environment in which the robot moves is also made up of non-intersecting line segments. Given an initial position and orientation and a final position and orientation, is there a sequence of translations and rotations that moves the robot from start to finish?

To show that this motion planning problem is $\text{NP-hard}$ we do one more reduction, starting from the set of segments output by the previous reduction algorithms. Specifically, we use our earlier set of line segments as a "screen" between two large rooms. The rooms are constructed so that the robot can enter or leave each room only by passing through the screens. We make the robot long enough that the robot can pass from one room to the other if and only if it can pass through three colinear holes in the screens. If the robot isn't long enough, it could get between the 'layers' of the screens. See the figure below:

\[
\text{image: erickson}
\]
WHY DO WE CARE?
ANOTHER EXAMPLE
3SAT PROBLEM

input:

output: “
3SAT EXAMPLE

\((x \lor y \lor z) \land (x \lor \overline{y} \lor y) \land (u \lor y \lor \overline{z}) \land (z \lor \overline{x} \lor u) \land (\overline{x} \lor \overline{y} \lor z)\)
INDEPENDENT SET
A set $S \subseteq V$ is an independent set if no two nodes in $S$ are joined by an edge.
EXAMPLE
GOAL: given a graph G,
3SAT \leq_p \text{ INDSET}

(x \lor y \lor z) \land (x \lor \overline{y} \lor y) \land (u \lor y \lor z) \land (z \lor \overline{x} \lor u) \land (\overline{x} \lor \overline{y} \lor z)

what must we do to?
3SAT \leq_p \text{ INDSET}

(x \lor y \lor z) \land (x \lor \overline{y} \lor y) \land (u \lor y \lor z) \land (z \lor \overline{x} \lor u) \land (\overline{x} \lor \overline{y} \lor z)
3SAT \leq_p \text{INDSET}

(x \lor y \lor z) \land (x \lor \overline{y} \lor y) \land (u \lor y \lor z) \land (z \lor \overline{x} \lor u) \land (\overline{x} \lor \overline{y} \lor z)
\((x \lor y \lor z) \land (x \lor \overline{y} \lor y) \land (u \lor y \lor z) \land (z \lor \overline{x} \lor u) \land (\overline{x} \lor \overline{y} \lor \overline{z})\)
\( \phi \in \text{SAT} \implies \)
\[(G, k) \in \text{INDSET} \implies \]
Theory of NP
A language $L$
a language $L$ belongs to the class NP iff 

$$\exists A, c \text{ such that}$$

$$L = \{x \in \{0, 1\}^* \mid \exists y \in \{0, 1\}^{\lfloor x \rfloor c} \text{ s.t.} A(x, y) = 1\}$$
WHY IS TRIPLETS IN \( \textbf{NP} \)?

\((x_1, x_2, \ldots, x_n)\)
WHY IS INDSET IN NP?
COMPLEXITY CLASSES

NP

P
COOK-LEVIN THEOREM
The implication of this
BASEBALL
a vertex cover of a graph is a
a vertex cover of a graph is a set $C \subseteq V$ such that
$
\forall (x, y) \in E
$
either $x \in C$ or $y \in C$
EXAMPLE
**GOAL:**

given a graph $G$, 
MAXINDSET $\leq_{O(V)}$ MINVERTEXCOVER