**Data structures:** arrays, linked lists, doubly linked lists, priority queue

**Example:** Gale-Shapley algorithm

* \( \leq n^2 \) iterations while loop, how efficient can we make each iteration? *

- need to encode preferences of each \( \text{men} \) \( \text{men} \) and \( \text{men} \) \( \text{women} \) \( \text{women} \)
- need to know at each step which \( \text{men} \) \( \text{men} \) \( \text{women} \) \( \text{women} \) are free.

**Arrays**

* "query \( A_{i,j} \) in \( O(1) \) time (direct access)"

* "check if \( e \) is in \( A \)" in \( O(n) \) time
  (check one by one)

* If \( A \) is sorted, then "check if \( e \) is in \( A \)" in \( O(\log n) \) time
  (binary search)

* (drawback) dynamically maintain list (add/delete element list)

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\end{array}
\]

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\end{array}
\]

**Linked list** (good for dynamically maintaining list)

* Each element has a pointer to the next element (null if last element)
  8 have pointer to first element.

* "query \( A_{i,j} \)" in \( O(1) \) time

* "check if \( e \) is in list" in \( O(n) \) time

* In a doubly linked list you also have pointers to previous element on list

* Deletion: splice list \( O(1) \) operations

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\end{array}
\]

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\end{array}
\]

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\end{array}
\]

**Priority Queue**

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\end{array}
\]

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\end{array}
\]

**Set \( e \).prev and \( e \).next point at each other**

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\end{array}
\]

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\end{array}
\]

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\end{array}
\]

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\end{array}
\]

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \bullet \bullet \\
\end{array}
\]
* insertion: splice and extend list in O(1) operations

insert e between e' and e'':

```
| e' | e | e'' |
```

```
| e' | e | e'' |
```

set e'.next to e e.prev to e e''.prev to e e.next to e''

**Example** Gale-Shapley algorithm

**Algorithm**

```plaintext
start S = φ
while ( m is unmatched and has not proposed to all women)
    w = first woman in m's list not proposed yet w proposes to m
    if w is free
        add (m,w) to S
    else if (m',w) ∈ S and w prefers m to m'
        add (m',w) to S, m' becomes free
    else if (m',w) ∈ S and w prefers m' to m
        m remains free
return S
```

**Goal:** Give implementation of GS algorithm with O(n^2) running time. (we know at most n^2 iterations of while loop, need to show runtime of each iteration is O(1)).

**Input** M = \([1, 2, \ldots, n]\) / W = \([1, 2, \ldots, n]\)

- n×n array Pref M, Pref(M[i, j]) = w if woman w is ith preference for m
- n×n array Pref W, Pref(W[i, j]) = m if man m is ith preference for w

ex n=3

```
<table>
<thead>
<tr>
<th>Pref_W</th>
<th>Pref_M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 3 2</td>
<td>1 2 3</td>
</tr>
<tr>
<td>2 1 3</td>
<td>1 3 2</td>
</tr>
<tr>
<td>3 2 1</td>
<td>2 1 3</td>
</tr>
</tbody>
</table>
```
Tasks in each iteration

1. identify a free man
   - use linked list firstM of free men
     - pick first element m of list
     - if m becomes engaged delete from list
     - if new m' becomes free insert beginning list
   \[ \text{runtime} \quad O(1) \]
   \[ O(1) \]
   \[ O(1) \]

2. for m identify highest ranked w not proposed yet
   - use array nextj, next[m] = j position jth woman on preference list
   - m proposes to prefM[m, next[m]]
   - then next[m] = next[m] + 1
   \[ \text{runtime} \quad O(1) \]

3. for w need to check if engaged or if so with who?
   - use array PartnerW
     \[ \text{PartnerW}[w] = \begin{cases} \text{m'} & \text{if current partner of m'} \\ \text{null} & \text{if m' is single} \end{cases} \]
   \[ \text{runtime} \quad O(1) \]

4. for w has partner m' and m proposes need to decide
   - which of m or m' is preferred by w
   - option 1 find i, j such that prefW[w, i] = m
     \[ \text{runtime} \quad O(n) \]
   - each iteration
   - option 2 before loop compute n x n array of inverse preferences for each w
     \[ \text{runtime} \quad O(n^2) \]
     \[ \text{before while loop} \]
   \[ \text{InvPrefW}[w, m] = i \text{ if m is ranked i by w.} \]
   \[ \text{compare if InvPrefW}[w, m] = i \text{ and InvPrefW}[w, m'] = j \text{ runtime O(1) each iteration} \]

Conclusion
- We complete tasks 1, 2, 3, 4 each in runtime \( O(n) \).
- So runtime of this implementation of GS algorithm has runtime \( O(n^2) + O(n^3) = O(n^3) \)

preprocessing while loop