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

**Example:** Gale-Shapley algorithm

- Need to encode preferences of each man and woman
- Need to know at each step which men & 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(logn) time (binary search)
* (Drawback) dinamically maintain list (add/delete element list)

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

* Each element has a pointer to the next element (null if last element)
* 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

\[ \quad \]
* insertion: splice and extend list in O(1) operations
  insert e between e' and e'':

  ![Diagram showing insertion of e between e' and e'']

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

Example: Gale-Shapely algorithm

**Algorithm**

**Pseudo code**

```
start \( S = \emptyset \)
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 \((m',w) \in S \) and \( w \) prefers \( m \) to \( m' \)
    add \((m,w)\) to \( S \), \( m' \) becomes free
  else if \((m',w) \in S \) 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** \( M = [1, 2, \ldots, n] \), \( W = [1, 2, \ldots, n] \)

- \( n \times n \) array \( \text{pref}_M \), \( \text{pref}_M[m, i] = w \) \( \text{i if woman } w \text{ is } i \text{th preference for } m \)
- \( n \times n \) array \( \text{pref}_W \), \( \text{pref}_W[w, i] = m \) \( \text{i if man } m \text{ is } i \text{th preference for } w \)

**ex** \( n = 3 \)

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

1. **identify a free man**
   - Use linked list `freem` of free men
   - pick first element `m` of list
   - if `m` becomes engaged delete from list
   - if new `m'` becomes free insert beginning list

2. for `m` identify highest ranked `w` not proposed yet
   - use array `next[j]`, `next[w] = j` position jth woman on preference list
   - `m` proposes to `PrefM[m, next[w]]`
   - then `next[w] = next[w] + 1`

3. for `w` need to check if engaged and if so with who?
   - use array `partner[w]`
   
   \[
   \text{Partner}[w] = \begin{cases} 
   m' & \text{if current partner of } m' \\
   \text{null} & \text{if } m' \text{ is single}
   \end{cases}
   \]

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`
     \[
     \text{PrefW}[w, j] = m'
     \]
   - compare `i` and `j`
   - Option 2: before loop, compute `n x n` array of inverse preferences for each `w`
   
   \[
   \begin{array}{c|cccc} 
   \text{w} & 1st & 2nd & 3rd & 4th \\
   \hline 
   1 & 2 & 3 & 4 \\
   2 & 1 & 3 & 4 \\
   3 & 2 & 1 & 4 \\
   4 & 3 & 2 & 1 \\
   \end{array}
   \quad \Rightarrow \quad
   \begin{array}{c|cccc} 
   \text{w} & 1 & 2 & 3 & 4 \\
   \hline 
   1st & 3rd & 1st & 2nd & 4th \\
   \end{array}
   \]

   \[
   \text{InvPrefW}[w, m] = i \quad \text{if } m \text{ is ranked } i \text{ by } w.
   \]

   \[
   \text{Compare if } \text{InvPrefW}[w, m] = i \text{ and } \text{InvPrefW}[w, m'] = j
   \]

   \[
   O(1)
   \]

   each iteration

Conclusion: We complete tasks 1, 2, 3, 4 each in runtime \( O(1) \).

So runtime of this implementation of GS Algorithm has runtime \( O(n^2) \) each iteration.

\[
\text{runtime} \quad O(n^2) + O(n^2) = O(n^2)
\]

preprocessing while loop