Gale-Shapley

```
public static Marriages GaleShapley(Preference pref) {
    int n = pref.getN();
    Marriages marriages = new Marriages(n);
    int[] next = new int[n];
    while (marriages.HasFreeMen()){
        int m = marriages.getFreeMan();
        int w = pref.getManPreference(m, next[m]);
        next[m]++;
        if (marriages.isWomanAvailable(w)){
            marriages.addMarriage(w, m);
        } else {
            int m1 = marriages.husband(w);
            if (pref.womanPrefers(w, m, m1)) {
                marriages.addMarriage(w, m);
            }
        }
    }
    return marriages;
}
```

```
public class Marriages{
...
}
public class Preference{
...
}
```

Representing Marriages

```
public static Marriages GaleShapley(Preference pref) {
    int n = pref.getN();
    Marriages marriages = new Marriages(n);
    int[] next = new int[n];
    while (marriages.HasFreeMen()){
        int m = marriages.getFreeMan();
        int w = pref.getManPreference(m, next[m]);
        next[m]++;
        if (marriages.isWomanAvailable(w)){
            marriages.addMarriage(w, m);
        } else {
            int m1 = marriages.husband(w);
            if (pref.womanPrefers(w, m, m1)) {
                marriages.addMarriage(w, m);
            }
        }
    }
    return marriages;
}
```

- A (mathematical) relation - a set of pairs.
- Set<Integer,Integer > marriages = ...

Choosing a Representation (Data Structure)

- Criteria:
  - Prevent spurious states.
    - i.e. marriages = {(1,2),(1,2),(1,3), (2,3)}
  - Fast to answer queries / perform updates
  - Add redundant information.
  - Maintain consistency.

Evaluating the List of Pairs Representation

Operations on Marriages	List of Pairs
`new Marriages(n)`	
`marriages.hasFreeMen()`	
`marriages.getFreeMan()`	
`marriages.isAvailable(w)`	
`marriages.husbandOf(w)`	
`marriages.addMarriage(w, m)`	
Spurious Info	

Evaluating the List of Pairs Representation

Operations on Marriages	List of Pairs
`new Marriages(n)`	O(1)
`marriages.hasFreeMen()`	O(n)
`marriages.getFreeMan()`	O(n)
`marriages.isAvailable(w)`	O(n)
`marriages.husbandOf(w)`	O(n)
`marriages.addMarriage(w, m)`	O(n), find if w is married
Spurious Info	Yes

Representing Marriages: Preventing Spurious States

```
public class Marriages{
    //free woman has -1 as husband
     public static final int NOT_ENGAGED = -1;
    int[] husband;
}
```

- 1 to 1
- Woman, Men are taken from the domain 0..n-1

2
-1
-1

Woman 0 married to man 2. Other women unmarried.

Evaluating the Husband of Array Representation

Operations on Marriages	Husband Array
`new Marriages(n)`	
`marriages.hasFreeMen()`	
`marriages.getFreeMan()`	
`marriages.isAvailable(w)`	
`marriages.husbandOf(w)`	
`marriages.addMarriage(w, m)`	
Spurious Info	

Evaluating the Husband of Array Representation

Operations on Marriages	List Of Pairs	Husband Array
`new Marriages(n)`	O(1)	O(n), init array
`marriages.hasFreeMen()`	O(n)	O(n)
`marriages.getFreeMan()`	O(n)	O(n)
`marriages.isAvailable(w)`	O(n)	O(1)
`marriages.husbandOf(w)`	O(n)	O(1)
`marriages.addMarriage(w, m)`	O(n)	O(1)
Spurious Info	Yes	No

Representing Marriages: Adding Redundant Info

- Add a set of currently free men.
- Maintain the set so that it is consistent with the `husband` array as we add/revoke marriages.

```
public class Marriages{
    //free woman has -1 as husband
    public static final int NOT_ENGAGED = -1;
int[] husband;
Set<Integer> freeMen;
}
```

Evaluation

Operations on Marriages	Husband Array + FreeMen Set
`new Marriages(n)`	
`marriages.hasFreeMen()`	
`marriages.getFreeMan()`	
`marriages.isAvailable(w)`	
`marriages.husbandOf(w)`	
`marriages.addMarriage(w, m)`	
Spurious Info	

Evaluation

Operations on Marriages	LOP	HA	Husband Array + FreeMen Set
`new Marriages(n)`	O(1)	O(n)	O(n), init array and free men
`marriages.hasFreeMen()`	O(n)	O(n)	O(1)
`marriages.getFreeMan()`	O(n)	O(n)	O(1)
`marriages.isAvailable(w)`	O(n)	O(1)	O(1)
`marriages.husbandOf(w)`	O(n)	O(1)	O(1)
`marriages.addMarriage(w, m)`	O(n)	O(1)	O(n), remove an element from a set
Spurious Info	Yes	No	No

Representing Marriages: Adding Redundant Info

- Turn free men to a list where order matters.
- It is no longer arbitrary at which location we get, add, remove from the list. The first element.

```
public class Marriages{
    //free woman has -1 as husband
     public static final int NOT_ENGAGED = -1;
    int[] husbandOf;
    List<Integer> freeMen = new LinkedList;
}
```

Evaluation

Operations on Marriages	LOP	HA	Husband Array + FreeMen List
`new Marriages(n)`	O(1)	O(n)	O(n), init array and free men
`marriages.hasFreeMen()`	O(n)	O(n)	O(1)
`marriages.getFreeMan()`	O(n)	O(n)	O(1)
`marriages.isAvailable(w)`	O(n)	O(1)	O(1)
`marriages.husbandOf(w)`	O(n)	O(1)	O(1)
`marriages.addMarriage(w, m)`	O(n)	O(1)	O(1)
Spurious Info	Yes	No	No

womanPrefers(w,m1,m2)

```
public class Preference {
    private int n ;
    private int[][] manPref;
    private int[][] womanPref;

    public boolean womanPrefers(int w, int m1,
    int m2) {
        int i1 = indexOf(womanPref[w], m1);
        int i2 = indexOf(womanPref[w], m2);
        return i1 < i2;
    }

    public static int indexOf(int arr[], int v){
        for (int i = 0; i < arr.length; i++) {
            if(arr[i] == v) return i;
        }
        throw new RuntimeException("...");
    }

}
```

- womanPrefers (w,m1,m2) is O(n).
- Should we speed up `indexOf` ?Or `womanPrefers`?

indexOfMan(w,m)

```
public class Preference {
    private int n ;
    private int[][] manPref;
    private int[][] womanPref;

    private int[][] indexOfMan;

    public Preference(...){
          ...
        indexOfMan = new int[n][];
        for(int w=0;w<n;w++){
            indexOfMan[w] = new int[n];
            for(int i=0;i<n;i++){
                indexOfMan[w][womanPref[w,i]] = i;
            }
        }
    }

    public boolean womanPrefers(int w, int m1,
    int m2) {
        int i1 = indexOfMan[w][m1];
        int i2 = indexOfMan[w][m1];
        return i1 < i2;
    }
```

- O(1)