Contents

1 Stable Maching problema

1.1 Algoritmo Gale-Shapley

Este algoritmo al terminar de ejecutarse se encuentra un matching prefecto si:

- Si existen n solicitantes con diferentes preferencias.
- Si existen n requeridos con diferentes preferencias.

Eligiendo las estructuras correctamente se puede plantear en $O(n)$.

```
    Inicialmente M=Vacio

    Mientras existe un solicitante sin pareja que no aun se haya postulado a todas las
     parejas

        Sea s un solicitante sin pareja
        Sea r el requerido de su mayor preferencia al que no le
                    solicito previamente

        if r esta desocupado
            M = M U (s,r)
            s esta ocupado
        else
            Sea s' tal que (s', r) pertenece a M

            si r prefiere a s sobres s'
                M = M - {(s', r)} U (s,r)
                s esta ocupado
                s' esta libre
    Retornar M
```

Listing 1: Algoritmo de Gale-Shapley

1.2 Alternativas

1.2.1 Diferentes cantidades de oferentes que requeridos

Dado n oferentes y m requeridos, con $m <> n$, no se puede encontrar un matching stable.

Entonces, tenemos que redefinir el concepto de estable. Una pareja (s,r) es **estable** si:

- No existe requerido r' sin pareja al que s prefiera a su actual pareja.
- No existe un requerido r' en pareja, tal que s y r' se prefieran sobre sus respectivas parejas.
- No existe solicitante s' sin pareja al que r prefiera a su actual pareja.
- No existe un solicitante s' en pareja tal que r y s' se prefieran sobre sus respectivas parejas.

Por lo tanto un matching es estable si:

- No tienen parejas inestables bajo la condicion anterior.
- Que no queden requeridos y solicitantes sin pareja.

Soluciones para ajustar al modelo de Gale-Shapley:

1. Inventar $|n - m|$ elementos ficticios
   - Los elementos ficticios se pondran en las listas de preferencias con menos elementos.
   - Estos elementos ficticios se agregan al final y deben ser los menos preferidos.

- Luego ejecutar Gale-Shapley
- Por ultimo, eliminar las parejas con elementos ficticios. Estos seran los requeridos que quedan sin pareja.

2. Adecuar el Algoritmo

   - Si hay mas **solicitantes** que requeridos, quitar de la *lista de solicitantes* sin parejas a aquellos que agotaron sus propuestas.
   - Si hay mas **requeridos** que solicitantes, quitar de la *lista de parejas* a aquellas donde el requerido quedo sin pareja.

1.2.2 Preferencias incompletas

Las listas de preferencias de los oferentes y los requeridos son un subset de las contrapartes.

Son parejas **aceptables** de un elemento a aquellas contrapartes que figuran en su lista de preferencias.

Una pareja (s,r) es **estable** si:

- Son *aceptables* entre ellos.
- No existe requerido *aceptable* r' sin pareja al que s prefiera a su actual pareja.
- No existe un requerido *aceptable* r' en pareja, tal que s y r' se prefieran sobre sus respectivas parejas.
- No existe solicitante *aceptable* s' sin pareja al que r prefiera a su actual pareja.
- No existe un solicitante *aceptable* s' en pareja tal que r y s' se prefieran sobre sus respectivas parejas.

Un matching es estable si no tiene parejas inestables bajo la condicion anterios.

```
Inicialmente M=Vacio

#Iterea mientras no haya acotado su sublista de preferencias
Mientras existe un solicitante sin pareja
                'que no aun se haya postulado a todas las parejas'

    Sea s un solicitante sin pareja
    Sea r el requerido de su mayor preferencia al que no le
                solicito previamente

    # se condiera si es aceptable
    if r considera 'aceptable' a s

        if r esta desocupado
            M = M U (s,r)
            s esta ocupado
        else
            Sea s' tal que (s', r) pertenece a M
            si r prefiere a s sobres s'
                M = M - {(s', r)} U (s,r)
                s esta ocupado
                s' esta libre

# Retornar solo parejas aceptables
Retornar M
```

Listing 2: Algoritmo para parejas incompletas

1.2.3 Preferencias con empates

INDIFERENCIA Y PREFERENCIA ESTRICTA

1. X es **indiferente** a "y" y a "z" si en su lista de preferencias estan el la misma posicion.
2. X es **prefefiere estrictamente** a "y" sobre "z" si en su lista de preferencias no le son indiferentes y "y" se encuentra antes que "z" en la misma.

ESTABILIDAD DEBIL

Una pareja (s,r) es debilmente estable si no existe una pareja (s' y r') talque:

- s prefiere estrictamente a r' sobre r *(pareja actual de s)*
- r' prefiere estrictamente a s sobre s' *(pareja actual de r')*

```
Inicialmente M=Vacio

#Iterea mientras no haya acotado su sublista de preferencias
Mientras existe un solicitante sin pareja
                'que no aun se haya postulado a todas las parejas'

    Sea s un solicitante sin pareja
    Sea r el requerido de su mayor preferencia al que no le
                solicito previamente

    if r esta desocupado
        M = M U (s,r)
        s esta ocupado
    else
        Sea s' tal que (s', r) pertenece a M

        # prefiere estrictamente
        si r prefiere estrictamente a s sobres s'
            M = M - {(s', r)} U (s,r)
            s esta ocupado
            s' esta libre

Retornar M
```

Listing 3: Algoritmo para parejas incompletas

En caso de que sea empate, se mantendra con su pareja actual.

ESTABILIDAD FUERTE

Una pareja (s,r) es debilmente estable si no existe una pareja (s' y r') talque:

- s prefiere estrictamente o le es indiferente a r' sobre r *(pareja actual de s)*
- r' prefiere estrictamente o le es indiferente a s sobre s' *(pareja actual de r')*

Puede no existir un matching perfecto.

```
Inicialmente M=Vacio

Mientras existe un solicitante sin pareja y no exista solicitante que agoto sus
parejas

    Sea s un solicitante sin pareja
    Sea r el requerido de su mayor preferencia al que pueda proponer
    Por cada sucesor s' a s en la lista de preferencias de r
        if (s',r) pertence a M
            M = M - {(s',r)}
```

```
                s' esta libre
            quitar s' de la lista de preferencias de r
            quitar r de la lista de preferncias de s'

        Por cada requerido r' que tiene multiples parejas
            Por cada pareja s' en pareja con r'
                M = M - {(s',r')}
                quitar s' de la lista de preferencias de r'
                quitar r' de la lista de preferencias de s'

    if estan todos en pareja
        Retornar M
    else
        No existe ningun matching super estable
```

Listing 4: Algoritmo para parejas super estables

En caso de que sea empate, se mantendra con su pareja actual.

1.2.4 Agrupacion de 1 a muchos

El solicitante puede tener varios cupos por lo tanto:

- Exiten m requeridos, donde un requerido puede estar unicamente con 1 pareja.
- Exiten n solicitantes, donde cada solicitante puede tener c cupos para armar parejas.

Existe un matching estable si la cantidad de requeridos es igual a la cantidad de solicitantes por la cantidad de cupos.

$$m = n * c \tag{1}$$

No cambia la definición de Gale Shampey para **matching estable**

```
    Inicialmente M=Vacio

    Mientras exista un solicitante con cupo disponible

        Sea s un solicitante sin pareja
        Sea r el requerido de su mayor preferencia al que no le
                    solicito previamente

        if r esta desocupado
            M = M U (s,r)
            s decremente su disponibilidad de parejas
        else
            Sea s' tal que (s', r) pertenece a M

            si r prefiere a s sobres s'
                M = M - {(s', r)} U (s,r)
                s decremente su disponibilidad de parejas
                s' incrementa su disponibilidad de parejas
    Retornar M
```

Listing 5: Algoritmo de solicitantes con cupos

La complejidad algoritmica no se modifica porque solo se agrega un contador.

1.2.5 Agrupacion de muchos a 1

El requerido puede tener varios cupos por lo tanto:

- Exiten m requeridos, donde cada solicitante puede tener z cupos para armar parejas.
- Exiten n solicitantes, donde un requerido puede estar unicamente con 1 pareja.

Existe un matching estable si la cantidad de solicitantes es igual a la cantidad de requeridos por la cantidad de cupos.

$$n = m * z \tag{2}$$

No cambia la definición de Gale Shampey para **matching estable**

```
Inicialmente M=Vacio

Mientras exista un solicitante con cupo disponible

    Sea s un solicitante sin pareja
    Sea r el requerido de su mayor preferencia al que no le
                solicito previamente

    if r tiene cupo
        M = M U (s,r)
        s esta ocupado
        r decrementa su disponibilidad de parejas
    else
        Sea s' tal que (s', r) pertenece a M y
                s' es el menos preferidos de las parejas r

        si r prefiere a s sobres s'
            M = M - {(s', r)} U (s,r)
            s esta ocupado
            s' esta libre
Retornar M
```

Listing 6: Algoritmo de requeridos con cupos

La complejidad algoritmica si se modifica.

Para conocer el solicitante de menor preferencia podemos utilizar un heap de minimos. Como el cupo es de z, la complejidad algoritmica para actualizar el heap es $log(z)$.

1.2.6 Agrupacion de y a x

- Exiten n solicitantes, donde cada solicitante puede tener c cupos para armar parejas.
- Exiten m requeridos, donde cada requerido puede tener z cupos para armar parejas.

Existe un matching estable si:

$$n * c = m * z \tag{3}$$

No cambia la definición de Gale Shampey para **matching estable**
Para implementar se requieren las siguientes estructuras:

- Un heap de minimos para los requeridos.
- Un contador de cupos para los solicitantes.

La complejidad algoritmica es igual a la de los requeridos con cupos

1.2.7 Conjuntos no bipartios - Stable Roommate Problem

Pendiente

2 Analisis amortizado

2.0.1 Metodo de agregacion

2.0.2 Metodo del banquero

2.0.3 Metodo del potencial

2.0.4 Heap binomial y fibonacci

Revisar capitulo 19 del Corven.
Para el **heap binomial** se utilizan bosques de arboles binarios. Existe un proceso donde se van ordenando los arboles.

Al insertar, se parece al ejemplo de contador binario y la amortizacion es O(1)
Decrementar en un log binomial, es log(n) porque no es posible amortizar
Eliminar el minimo, es el el peor caso es log(n)

Para el **heap fibonacci** ...

3 Algoritmos Greedy

Utiliza heurisica de seleccion para encontrar una solución global optima despues de muchos pasos.

3.0.1 Mochila fraccionaria

Dado un contener de capacidad W, y un conjunto de elementos n fraccionables de valor v_i y peso w_i

El objetivo es seleccionar un subconjunto de elemento o fracciones de ellos de modo de maximizar el valor almacenado y sin superar la capacidad de la mochila.

La complejidad es $O(nlog(n))$

3.0.2 Cambio de moneda

Es una solución es conocido como solución de cajero. Contamos con un conjunto de diferentes monedas de diferentes denominación sin restricción de cantidad.

$$\$ = (C_1, C_2, C_3, \cdots, C_n)$$

El objetivo es entregar la menor cantidad posible de monedas como cambio.
Tiene una complejidad de $O(n)$.
El sistema \$ se conoce como **canonico** a aquel en el que para todo x, $greedy(\$, x) = optimo(\$, x)$.
Para saber si una base es canonica:

1. Basta con buscar un contraejemplo. Estaria entre la 3ra denomininacion y la suma de las ultimas dos doniminaciones.

2. Utilizar un algoritmo Polinimico para determinar si es un sistema canonico.

Si el problema no es greddy, se puede construir un algoritmo utilizando programación dinamica.