T1-BRC-Reporte

August 27, 2020

1 Tarea 1

1.1 Gale-Shapley algorithm para apareamientos estables

Instrucciones: > Implementar en Python el algoritmo de Gale-Shapley. Probar con varias entradas. Entregarlo como notebook de Jupyter. Recuerden documentar su código.

[1]:
```
from copy import deepcopy

# Caso de prueba robado de 1.2-3
preferencias_h = {
 'abe':  ['abi', 'eve', 'cath', 'ivy', 'jan', 'dee', 'fay', 'bea', 'hope',␣
 ↪'gay'],
 'bob':  ['cath', 'hope', 'abi', 'dee', 'eve', 'fay', 'bea', 'jan', 'ivy',␣
 ↪'gay'],
 'col':  ['hope', 'eve', 'abi', 'dee', 'bea', 'fay', 'ivy', 'gay', 'cath',␣
 ↪'jan'],
 'dan':  ['ivy', 'fay', 'dee', 'gay', 'hope', 'eve', 'jan', 'bea', 'cath',␣
 ↪'abi'],
 'ed':   ['jan', 'dee', 'bea', 'cath', 'fay', 'eve', 'abi', 'ivy', 'hope',␣
 ↪'gay'],
 'fred': ['bea', 'abi', 'dee', 'gay', 'eve', 'ivy', 'cath', 'jan', 'hope',␣
 ↪'fay'],
 'gav':  ['gay', 'eve', 'ivy', 'bea', 'cath', 'abi', 'dee', 'hope', 'jan',␣
 ↪'fay'],
 'hal':  ['abi', 'eve', 'hope', 'fay', 'ivy', 'cath', 'jan', 'bea', 'gay',␣
 ↪'dee'],
 'ian':  ['hope', 'cath', 'dee', 'gay', 'bea', 'abi', 'fay', 'ivy', 'jan',␣
 ↪'eve'],
 'jon':  ['abi', 'fay', 'jan', 'gay', 'eve', 'bea', 'dee', 'cath', 'ivy',␣
 ↪'hope']}

preferencias_m = {
 'abi':  ['bob', 'fred', 'jon', 'gav', 'ian', 'abe', 'dan', 'ed', 'col', 'hal'],
 'bea':  ['bob', 'abe', 'col', 'fred', 'gav', 'dan', 'ian', 'ed', 'jon', 'hal'],
 'cath': ['fred', 'bob', 'ed', 'gav', 'hal', 'col', 'ian', 'abe', 'dan', 'jon'],
 'dee':  ['fred', 'jon', 'col', 'abe', 'ian', 'hal', 'gav', 'dan', 'bob', 'ed'],
 'eve':  ['jon', 'hal', 'fred', 'dan', 'abe', 'gav', 'col', 'ed', 'ian', 'bob'],
```

```python
 'fay':  ['bob', 'abe', 'ed', 'ian', 'jon', 'dan', 'fred', 'gav', 'col', 'hal'],
 'gay':  ['jon', 'gav', 'hal', 'fred', 'bob', 'abe', 'col', 'ed', 'dan', 'ian'],
 'hope': ['gav', 'jon', 'bob', 'abe', 'ian', 'dan', 'hal', 'ed', 'col', 'fred'],
 'ivy':  ['ian', 'col', 'hal', 'gav', 'fred', 'bob', 'abe', 'ed', 'jon', 'dan'],
 'jan':  ['ed', 'hal', 'gav', 'abe', 'bob', 'jon', 'col', 'ian', 'fred', 'dan']}

# Funcion copiada de 1.2-3 para verificar los resultados
def verificar(engaged, debug=False):
    inverseengaged = dict((v,k) for k,v in engaged.items())
    for she, he in engaged.items():
        shelikes = preferencias_m[she]
        shelikesbetter = shelikes[:shelikes.index(he)]
        helikes = preferencias_h[he]
        helikesbetter = helikes[:helikes.index(she)]
        for guy in shelikesbetter:
            guysgirl = inverseengaged[guy]
            guylikes = preferencias_h[guy]
            if guylikes.index(guysgirl) > guylikes.index(she):
                if debug:
                    print("%s and %s like each other better than "
                          "their present partners: %s and %s, respectively"
                          % (she, guy, he, guysgirl))
                return False
        for gal in helikesbetter:
            girlsguy = engaged[gal]
            gallikes = preferencias_m[gal]
            if gallikes.index(girlsguy) > gallikes.index(he):
                if debug:
                    print("%s and %s like each other better than "
                          "their present partners: %s and %s, respectively"
                          % (he, gal, she, girlsguy))
                return False
    return True
```

Pseudocodigo propuesto por la profesora en clase.

```
[90]: """  _               _                 _                            _
| \ | | ___     ___ (_) ___  ___ _   _| |_ __ _ _ __    ___  ___| |_ __ _
|  \| |/ _ \   / _ \| |/ _ \/ __| | | | __/ _` | '__|  / _ \/ __| __/ _` |
| |\  | (_) | |  __/| |  __/ (__| |_| | || (_| | |    |  __/\__ \ || (_| |
|_| \_|\___/   \___|/ |\___|\___|\__,_|\__\__,_|_|     \___||___/\__\__,_|
          _     _ |__/
  ___ ___| | __| | __ _
 / __/ _ \ |/ _` |/ _` |
| (_|  __/ | (_| | (_| |
 \___\___|_|\__,_|\__,_|
```

```
    No pude hacer funcionar esta funcion, obtiene una lista de
matrimonios pero no es estable.
"""

# Funcion principal
def stable_marriage(p_h, p_m):
    """Funcion principal para tratar de resolver el pro_
    blema del matrimonio estable. Trata de imitar el al_
    goritmo propuesto por el pseudocodigo propuesto  en
    clase.
        Esta funcion recibe:
            p_h := Lista de preferencias para hombres.
            p_m := Lista de preferencias para mujeres.

        Esta funcion regresa:
            parejas := Un diccionario con las parejas
                tal que el nombre de las mujers estan
                como clave y los de los hombres  como
                valores de las claves correspondientes.
                    'parejas[n_mujer] = n_hombre'
    """

    # Dadas las siguientes personas ...
    hombres = list(p_h.keys())
    mujeres = list(p_m.keys())
    # ... y sus respectivas preferencias.
    preferencias_h = deepcopy( p_h )
    preferencias_m = deepcopy( p_m )
    """ En esta seccion debi usar deepcopy (inspirado por
    1.2-3 )  porque dentro de la funcion estaba alterando
    los valores internos de las variables externas.
    """

    # Iniciamos con todes hombres y mujeres no comprometidos
    h_libres = hombres[:]
    parejas = {}

    # Mientras haya hombres libres ...
    while len(h_libres) > 0:
        # Tomamos a uno de los hombres
        h = h_libres.pop()

        # ... y queden mujeres para proponer
        for m in preferencias_h[h]:

            # Tomamos a la siguiente mujer ideal
            preferencias_h[h].remove(m)    # Solo proponemos una vez x m
```

```
            # Si ella ya esta comprometida
            if (m in parejas):
                # Si ella prefiere a h sobre parejas[m]
                if ((preferencias_m[m]).index(h) <
                    (preferencias_m[m]).index(parejas[m])):

                    h_libres.append(parejas[m])   # El dejado es soltero
                    parejas[m] = h        # Actualizamos la nueva pareja
                    break

            # Si ella no esta comprometida
            else:
                parejas[m] = h                # Creamos una nueva pareja
                break


        # Parejas parciales por iteracion
        #print(f'{parejas=}')

    #print(preferencias_h)
    #print(preferencias_m, end="\n\n\n")
    # Regresamos el grupo de las parejas comprometidas
    return parejas
```

[2]:
```
"""___     _                          _         _
| ____|___| |_ __ _     ___  ___   | | __ _   | |__   _   _  ___ _ __   __ _
|  _| / __| __/ _` |   / _ \/ __|  | |/ _` |  | '_ \ | | | |/ _ \ '_ \ / _` |
| |___\__ \ || (_| |  |  __/\__ \  | | (_| |  | |_) | |_| |  __/ | | | (_| |
|_____|___/\__\__,_|   \___||___/ |_|\__,_|  |_.__/ \__,_|\___|_| |_|\__,_|

    En esta funcion decidi empezar escribiendo los comentarios basados
en el pseudocodigo propuesto por la profesora. Todos los comentarios con
comillas(") corresponden al pseudocodigo y los que solo usan almohadilla
(#) son respecto a la implementacion del mismo.
"""

# Funcion principal
def stable_marriage(p_h, p_m):
                                # Hacemos una copia de las preferencias
    preferencias_h = deepcopy(p_h)
    preferencias_m = deepcopy(p_m)
                            # Conseguimos los nombres de los participantes
    hombres = list(preferencias_h.keys())
    mujeres = list(preferencias_m.keys())

    """ Inicializar todos los hombres y mujeres como no
```

```
    comprometidos. (disponibles)
    """
    parejas = {}                        # Al inicio no hay parejas formadas
    h_libres =hombres[:] # Al inicio todos los hombres estan disponibles

    """ Mientras haya un hombre soltero que no se haya
    propuesto a todas las mujeres
    """
    while len(h_libres) > 0:
        """ Toma a dicho hombre h """
                    # Tomamos al primer hombre de la cola de disponibles
            h = h_libres.pop(0)
        """ Sea m la mujer mas preferida por el hombre y
        a quien h no se haya propuesto """
               # Tomamos a la mujer mas preferida, aun disponible, por h
        m = (preferencias_h[h]).pop(0)

        """ Si m esta disponible """
        if m not in parejas:           # Si no esta en una pareja lo esta
            """ Haz que (m,h) se comprometan """
            parejas[m] = h

        # """ Si, por otro lado, m ya se comprometio con h' """
        else:
            """ Si m prefiere a h' sobre h """
             # Mientras menor sea el indice de alguien, mas preferido es
            if (preferencias_m[m].index(parejas[m]) <
                preferencias_m[m].index(h) ):
                """ h permanece disponible """
      # Agregamos el hombre al final para que escoja hasta la proxima ronda
                h_libres.append(h)

            # """ Si, por otro lado, m prefiere a h sobre h' """
            else:              # Si no prefiere a uno, entonces al otro si
                """ h' se vuelve soltero """
           # Ponemos a la antigua opcion de ella en la cola de disponibles
                h_libres.append(parejas[m])
                """ (m, h) se comprometen """
                parejas[m] = h    # El actual se vuelve la opcion de ella
      # Estos ultimos dos pasos los inverti para evitar usar una variable
    temporal
    """ EndWhile """
    """ Regresamos el conjunto de todas las parejas creadas """
    return parejas
```

[3]:
```
parejas = stable_marriage(preferencias_h, preferencias_m)

print( parejas if verificar(parejas, True) else "Error")
```

```
{'abi': 'jon', 'cath': 'bob', 'hope': 'ian', 'ivy': 'abe', 'jan': 'ed', 'bea':
'fred', 'gay': 'gav', 'eve': 'hal', 'dee': 'col', 'fay': 'dan'}
```

1.2 Referencias

1. Video profesora 1
2. Video profesora 2
3. Inspiracion y caso de prueba