Métodos de Modelado y Optimización
Laboratorio 1

Implemente los siguientes métodos solicitados en lenguaje Python.

1. (7.5%) Método `parse_equation(equation)` que recibe una string `equation` que representa una ecuación matemática y retorna un diccionario que mapea las variables de la ecuación con sus constantes asociadas.
    a. Entrada de ejemplo: `"-3.8x1 + 5x2 -2x3"`
    b. Salida de ejemplo: `{'x1': -3.8, 'x2': 5.0, 'x3': -2.0}`
    c. Si una variable no tuviera coeficiente, se debe asumir un 1
    d. Los métodos de strings `replace` y `split` pueden resultar de mucha utilidad
    e. Asuma que las variables siempre serán `xn` donde `n` es su subíndice
    f. Asuma que cada variable aparece una única vez en la ecuación

2. (7.5%) Método `parse_restriction(restriction)` que recibe una string `restriction` que representa una restricción de un problema de programación lineal y retorna una tupla con el diccionario de mapeo de las variables a sus constantes (puede hacer un llamado a `parse_equation`), el valor de la restricción y un valor booleano que representa si la restricción es *upper-bound*.
    a. Entrada de ejemplo: `"-3.8x1 + 5x2 - 2x3 <= 35"`
    b. Salida de ejemplo: `({'x1': -3.8, 'x2': 5.0, 'x3': -2.0}, 35.0, True)`

3. (15%) Método `parse_problem(objective, restrictions, maximize)` que recibe una string `objective` que representa el objetivo de un problema de programación lineal, la lista `restrictions` conformada por strings que representan las restricciones del problema y el parámetro booleano `maximize` que indica si se desea maximizar la función objetivo. La función debe retornar una tupla de tres valores: el primer valor es un arreglo que representa los coeficientes para la función de optimización, incluyendo las variables "*slack*" y artificiales, el segundo valor es la matriz de coeficientes que conforman el cuerpo de la tabla Simplex, incluyendo la columna B (matriz aumentada) y por último un arreglo con los nombres de las variables (en el orden correcto en relación a la matriz de la tabla Simplex y los coeficientes de la función objetivo).
    a. Entrada de ejemplo:
```
"30x1 + 100x2",
["x1 + x2 <= 7",
"4x1 + 10x2 <= 40",
"10x1 >= 30"],
True
```
    b. Salida de ejemplo:
```
([30.0, 100.0, 0, 0, 0, -100000.0],
[     [1, 1, 1, 0, 0, 0, 7.0],
      [4.0, 10.0, 0, 1, 0, 0, 40.0],
      [10.0, 0, 0, 0, -1, 1, 30.0]  ],
['x1', 'x2', 's1', 's2', 's3', 'a3'])
```

4. (40%) Método `simplex(objective, restrictions, variables, maximize)` que recibe un arreglo `objective` con los coeficientes de la función a optimizar (incluyendo las variables *slack* y artificiales), la matriz `restrictions` de coeficientes que conforman el cuerpo de la tabla Simplex, incluyendo la columna B (matriz aumentada), el arreglo `variables` con los nombres de las variables de cada una de las columnas de la matriz y un valor booleano `maximize` que indica si lo que se desea es maximizar la función objetivo. La función debe retornar una tupla con el arreglo solución y el valor de la función objetivo para dicha solución. El arreglo solución está compuesto por tuplas `(var, value)`, donde `var` es el nombre de una variable y `value` el valor de dicha para variable para la solución.
    a. Entrada de ejemplo:
    ```
    [30.0, 100.0, 0, 0, 0, -100000.0],
    [     [1, 1, 1, 0, 0, 0, 7.0],
          [4.0, 10.0, 0, 1, 0, 0, 40.0],
          [10.0, 0, 0, 0, -1, 1, 30.0]  ],
    ['x1', 'x2', 's1', 's2', 's3', 'a3'],
    True
    ```
    b. Salida de ejemplo: `([('x1', 3.0), ('x2', 2.8), ('s2', 1.2)], 370.0)`
    c. Note que los parámetros que recibe el método `simplex` son los mismos que se obtiene al parsear un problema utilizando la función `parse_problem`
    d. Revise las filminas de clase, le permitirán entender paso a paso cómo aplicar Simplex.
    e. La biblioteca `numpy` le permite utilizar arreglos numpy (`numpy.array`) que permiten realizar operaciones sobre "*slices*" de datos, como por ejemplo: `restrictions[i,:] -= restrictions[pivot_row, :]`, que realizan operaciones valor por valor sin utilizar ciclos; pueden ahorrar mucho trabajo.
    f. Si va a utilizar arreglos numpy, recuerde crearlos y especificar su `dtype=numpy.float64`, de lo contrario podría asumir tipo int y generar errores

5. (5%) Método `simplex_solver(objective, restrictions, maximize)` que permita "empacar" el llamado a `parse_problem` y `simplex` en un solo método.
    a. Entrada de ejemplo:
    ```
    "30x1 + 100x2",
    ["x1 + x2 <= 7",
    "4x1 + 10x2 <= 40",
    "10x1 >= 30"],
    True
    ```
    b. Salida de ejemplo: `([('x1', 3.0), ('x2', 2.8), ('s2', 1.2)], 370.0)`
    c. Puede probar el funcionamiento correcto de dicho método con los casos de prueba:
        i. `simplex_solver("0.65x1 + 0.45x2", ["2x1 + 3x2 <= 400", "3x1 + 1.5x2 <= 300", "x1 <= 90"], True)`
        `Solución: ([('x1', 50.0), ('x2', 100.0), ('s3', 40.0)], 77.5)`
        ii. `simplex_solver("30x1 + 100x2", ["x1 + x2 <= 7", "4x1 + 10x2 <= 40", "10x1 >= 30"], True)`
        `Solución: ([('x1', 3.0), ('x2', 2.8), ('s2', 1.2)], 370.0)`

iii. `simplex_solver("3x1 + 8x2", ["x1 + 4x2 >= 3.5" , "x1 + 2x2 >= 2.5"], False)`
`Solución: ([('x1', 1.5), ('x2', 0.5)], 8.5)`

6. (25%) Utilice su método `simplex_solver` para resolver el problema de programación entera siguiente. Para ello utilice el algoritmo de ramificación y acotamiento estudiado en clase. Documente cada uno de los llamados al método `simplex_solver` así como la información adquirida de los mismos. Dibuje un diagrama representando la ramificación del problema hasta llegar a la solución.

$$\max z = x_1 + 4x_2$$

sujeto a:

$$-10x_1 + 20x_2 <= 22$$
$$5x_1 + 10x_2 <= 49$$
$$x_1 <= 5$$

Nota: Se permite utilizar la biblioteca de soporte numpy, pero no se permite utilizar bibliotecas que realicen el trabajo por usted. Cualquier otra biblioteca que se desee usar debe ser consultada primero con el profesor.