Título de la Práctica: Método Simplex Light.
Profesor Responsable: Antonio Sedeño Noda.
Práctica número: 1

- **Objetivo.**

En esta práctica se pretende que el alumno implemente la clase PROBLEM asociada a un problema de programación lineal con todas las restricciones del tipo menor o igual. La clase incorporará los métodos necesarios resolver dicho problema de optimización.

- **Diseño**

  El Método Simplex Primal "Light" a implementar resolverá problemas de Programación Lineal (PL) con variables restringidas en signo donde todas las restricciones son del tipo menor igual. La formulación del problema es:

$$\begin{aligned} &\min \ c^t x \\ &\text{s. a. } Ax \leq b \quad (1) \\ &\qquad x \geq 0 \end{aligned}$$

  En la anterior formulación se supondrá que todas las componentes de b son mayores ó iguales que cero. Esto implica chequear si b cumple la condición. En el caso de que alguna componente sea menor que cero, no se resolverá el correspondiente problema. Dado que el simplex únicamente resuelve problemas de PL en forma estándar, las restricciones (1) obligan a la introducción de variables de holgura relacionadas con ellas y a añadir columnas a A, tantas como filas (restricciones) tiene la matriz A. Una vez realizada esta operación, obtenemos una la descomposición de $A = (N, I)$, donde I es la matriz identidad formada por las variables de holgura. Por lo tanto, en este punto se aplica el método del Simplex Primal. En la práctica, se visualizará la tabla del Simplex en cada iteración del método. Finalmente, en el caso que exista solución óptima acotada se mostrará por pantalla.

- **Implementación.**

  El problema se lee desde un fichero de texto con el siguiente formato:

  $t\_p \;\; n \;\; m$
  $c_1 \;\; c_2 \;\; \dots \;\; c_n$
  $a_{11} \;\; a_{12} \;\; \dots \;\; a_{1n} \;\; b_1$

  $a_{m1} \;\; a_{m2} \;\; \dots \;\; a_{mn} \;\; b_m$

  donde t_p es el tipo del problema cuyo valor puede ser (min ó max), n es el número de variables y m el número de restricciones. La siguiente línea en el fichero corresponde a los costes de las n variables naturales y las siguientes líneas corresponden a las restricciones (1).

  El problema se almacena en un objeto perteneciente a la clase `PROBLEM` que es definida del modo siguiente:

```
Typedef vector<double> fila;

class PROBLEM
{
    char     clase[4];  // Clase del problema : max o min
    unsigned m;         // nº de restricciones
    unsigned n;         // nº de variables
    unsigned nh;        // nº de variables de holgura
    unsigned na;        // nº de variables artificiales
    vector<double> c;   // vector de costes unitarios
    vector<fila>   A;   // matriz de coeficientes tecnológicos
    vector<double> b;   // vector de términos independientes

    // Los atributos siguientes son necesarios para el método
    simplex
    vector<unsigned> ivb; //vector de índices de variables básicas
    double Vo;            // valor de la función objetivo
public:
    PROBLEM(char nombrefichero[]); // constructor
    ~PROBLEM();                    // destructor
    void Volcar_problema();  // Método que presenta por pantalla
    el problema de programación lineal

    //métodos propios del simplex
    unsigned entrante();  //devuelve la variable entrante
    unsigned saliente(unsigned s);  //devuelve la fila
    asociada a la variable saliente
    void actualizar_valores(unsigned s, unsigned r); //
    realiza pivoteo y actualiza ivb, A, b, c, vo
    void mostrar_solucion(); // muestra la sol. Óptima si
    existe
    void volcar_tabla();// muestra una tabla del simplex
    void Simplex_Light(); // implementa el método simplex
};
```

  donde `ivb` es un vector de dimensión m tal que `ivb[i]` es el índice de la variable básica asociada a la fila `i`.

El constructor `PROBLEM(char nombrefichero[])` se encarga de leer el problema del fichero de texto `nombrefichero` del tipo `ifstream (#include <fstream>)` y de asignar los atributos de la clase `PROBLEM`. La solicitud de memoria para la matriz `A` ha de ser por filas (dado el tipo definido). La dimensión de estos elementos estará asociada a *n*, *nh* y *m* únicamente.

Si el problema es de tipo `max` se cambia de signo al vector de costes. A continuación se amplía el bloque de memoria para el vector `c` hasta dimensión `n + nh` y se asigna un coste cero a las variables de holgura.

Finalmente, se dimensiona el vector `ivb` a tamaño *m*, asignándole los valores adecuados. De esta manera el constructor queda definido de la siguiente manera:

```
PROBLEM::PROBLEM(char nombrefichero[85])
{
  unsigned i, j;
  double dummy;
  ifstream textfile;

  textfile.open(nombrefichero);
  if (textfile.is_open()) {
       textfile >> (char *) clase >> (unsigned &) n >>
       (unsigned &) m;
       A.resize(m); // m filas
       //leemos la fila de los n costes
       for (i=0;i<n; i++) {
         textfile >> (double &) dummy;
         c.push_back(dummy);
       }
       // si el problema es de máximo cambiamos de signo los
costes
       if (strncmp(clase,"max",3) == 0)
         for(i=0;i<n;i++) c[i] *= -1.0;
       // extendemos el vector c a tamaño n + nh
       nh = m;
       for(i=0;i<nh;i++)c.push_back(0.0);
       // leemos las m restricciones y asignamos A, b, ivb
       for (i=0;i<m;i++) {
         for (j=0;j<n;j++) {
           textfile >> (double &) dummy;
           A[i].push_back(dummy);
         }
         textfile >> (double &) dummy;
         b.push_back(dummy);
         if (dummy < CERONEG){
           cout << "Lado derecho negativo, Modifique problema"
           << endl;
```

```
            exit(0);
          }
          A[i].resize(n+nh); // extendemos la fila a tamaño n
          +nh
          A[i][n+i] = 1.0; // ponemos un 1 en la columna n+i
          fila i correspondiente a la v. de holgura de la fila i
          ivb.push_back(n+i); // la v. básica asociada a la fila
          i es n+i
        }
        textfile.close();
    }
}
```

En la carpeta de la práctica se incluye el fichero prob.h con la definción de las clase PROBLEM. Además como se verá más adelante, debido a la aritmética en punto flotante, este fichero .h es conveniente define siguientes constantes:

```
const double   CEROPOS = 1E-15;  /* Cero positivo */
const double   CERONEG = -1E-15; /* Cero negativo */
const double   INFR    = 1E+10;  /* Infinito real */
const unsigned UERROR  = 65000;  /* error para variables unsigned */
```

Los objetivos de la primera parte de la práctica se alcanzan con la implementación de los métodos: `volcar_tabla()` para la presentación por pantalla de las sucesivas tablas generadas por el método del Simplex, `mostrar_solucion()`y `Volcar_problema()` que una vez leído el problema lo muestra por pontalla.

```
void PROBLEM::volcar_tabla()
void PROBLEM::mostrar_solucion()
void PROBLEM::Volcar_problema()
```

A continuación los métodos necesarios para la implementación del Simplex Primal Light. Los métodos necesarios para la implementación del Simplex Primal Light son:

`unsigned PROBLEM::entrante()` devuelve la variable que entra en la base, es decir, la variable no básica con coste más negativo. Si los costes de todas las variables no básicas son no negativos, entonces la solución actual es óptima y la función devuelve el valor `UERROR`.

`unsigned PROBLEM::saliente(unsigned s)` devuelve el índice de la fila *r* que determina la variable que sale de la base (`ivb[r]`). En el caso de que la solución sea No Acotada ($A^s \leq 0$) devuelve *UERROR*. Este método determinará la variable saliente mediante la regla:

$$\frac{b_r}{A_{rs}} = \text{mínimo}\left\{\frac{b_i}{A_{is}} : A_{is} > 0\right\} i = 1,\ldots,m$$

`void PROBLEM::actualizar_valores(unsigned s, unsigned r)` se encarga de actualizar la matriz *A*, el vector *b*, el vector *ivb*, los costes (reducidos) *c* y *Vo*. Siguiendo el siguiente esquema:

```
{
ivb[r] = s; /* actualización de la base */
temp = A[r][s];
A[r] /= temp;
x[r] /= temp;
Para todo i=1 hasta m con i<>r y A[i][s] <>0 hacemos
   temp = A[i][s];
   A[i] -= A[r] * temp; // la fila entera de i
   b[i] -= b[r] * temp;
Vo   -= b[r] * c[s];  actualizamos Vo
c[i] -= c[s] * A[r][i]; para todo i<>s; cr[s] = 0.0;
}
```

Con los métodos anteriores el método del Simplex Light queda descrito de la manera siguiente:

```
void PROBLEM::Simplex_Ligth()

unsigned s,r;
while ((s = entrante())!= UERROR) {
   volcar_tabla();
   r = saliente(s);
   if (r != UERROR)
      actualizar_valores(s,r);
   else
      PROBLEMA NO ACOTADO;
}
volcar_tabla();
mostrar_solucion();
```

Nota: En el campus virtual de la asignatura se encuentra una carpeta que contiene el ejecutable Lsimplex (./Lsimplex <nombrefichero>), que puede servir de guía en el desarrollo de la práctica, así como ocho ejemplos para la

depuración de la misma. El programa ha de funcionar con todos estos ejemplos.