Curso de Programación en C++.
TPL3. Trabajo Práctico de Laboratorio. [2017-11-27]

PASSWD PARA EL ZIP: `DW2N 11XD LS3B`

Ejercicios

[Ej. 1] **[splitmono]** Dado un `vector<int> v` escribir una función
`void splitmono(vector<int> &v,vector< vector<int> > &vmono);`
que separa a `v` en subsecuencias monótonas crecientes. Ejemplos:

```
v:[1,2,3,1,2,1]}, vmono:[[1,2,3],[1,2],[1]]
v:[0,4,3,4,1,4,1,1,2]}, vmono:[[0,4],[3,4],[1,4],[1,1,2]]
v:[1,4,1,3,3,0,3,3,4,0,0,0]}, vmono:[[1,4],[1,3,3],[0,3,3,4],[0,0,0]]
v:[2,3,3,2,4,1,0,4,0,4]}, vmono:[[2,3,3],[2,4],[1],[0,4],[0,4]]
```

[Ej. 2] **[filtersets]** Escribir una función
`void filtersets(vector< set<int> > &vset,vector< set<int> > &vsout,pred_t pred);`
que dado un vector de conjuntos y un predicado **`pred`**, deja en **`vsout`** sólo los conjuntos que satisfacen el predicado **`pred`**.

```
pred:sumzero  vset:[[1,-1],[1,2],[2,3]]           => vsout:[[-1,1]]
pred:hassize2 vset:[[1,-1,5],[1,2],[2,3]]         => vsout:[[1,2],[2,3]]
pred:hassize2 vset:[[-1,1,2],[-1],[-2,3],[-2,-1,0]]=> vsout:[[-2,3]]
pred:hassize2 vset:[[2,3],[0,3],[-2,-1,1],[-2,1]] => vsout:[[2,3],[0,3],[-2,1]]
pred:sumzero  vset:[[-2,-1,3],[2],[1,3]]          => vsout:[[-2,-1,3]]
```

Adicionalmente: escribir los predicados:

- **`sumzero`**: La suma de los elementos da 0.
- **`hassize2`**: El tamaño del conjunto es 2.
- **`hasodd`**: Tiene al menos un elemento impar
- **`allpositive`**: Todos los elementos son positivos.

[Ej. 3] **[inside]** Escribir una función
`bool inside(MatrixXd &xvrtx, VectorXd &xprobe);` que determina si el punto **`xprobe`** está dentro del **símplice** definido por sus vértices **`xvrtx`**. (Nota: Los símplices son los polígonos con menor número de vértices (**`ndim+1`**) en dimensión **`ndim`**, es decir en 1D son segementos, en 2d son triángulos y en 3D son tetrahedros). Los veŕtices del símplice están en las columnas de **`xvrtx`** que por lo tanto es de tamaño **`ndim x (ndim+1)`**. **`xprobe`** es de dimensión **`ndim`**.

La función debe funcionar en dimensión 2 o 3. Se puede determinar la dimensión del espacio a partir de la longitud de **`xprobe`**.

Ayuda: basta con determinar si existen **ndim+1** escalares $0 \leq \alpha_j \leq 1$ tales que

$$\mathbf{x}_p = \sum_j \alpha_j \mathbf{x}_j \tag{1}$$

Con la restricción que $\sum_{j=1}^{n_d+1} \alpha_j = 1$. O sea que debemos resolver un sistema de $(n_d+1) \times (n_d+1)$ de la siguiente manera:

$$\begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_p \\ 1 \end{bmatrix} \tag{2}$$

$$\mathbf{C}\boldsymbol{\alpha} = \mathbf{b}$$

Una vez resuelto el sistema, debe verificarse que todos los α_j estén en el rango $[0, 1]$.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Hay una función de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

```
ev.eval1(f,vrbs);
```

`ev.eval1(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

mstorti@galileo/fsiconf.0-9-g042426cb/Sat Jul 21 10:42:49 2018 -0300