Curso de Programación en C++.
TPL4-2017. Trabajo Práctico de Laboratorio. [2018-05-09]

PASSWD PARA EL ZIP: WQQR RPF3 V8HM

Ejercicios

[Ej. 1] **[mapvec]** Dado un vector `v`, y una correspondencia `M`, devolver el vectotr `w` que contiene el resultado de aplicar la correspondencia `M` a esos elementos. Es decir si `x` es un elemento de `v`, entonces `w` debe contener el elemento `y=M[x]` (en el mismo orden que están en `v`). Adicionalmente, de todos los elementos de `w` dejar solo los elementos que satisfacen el predicado `pred`, es decir aquellos `y` tales que `pred(y)` retorna verdadero.

Consigna: Escribir la función `void mapvec(map<int,int> &M,bool (*pred)(int),vector<int> &v, vector<int> &w);` que realiza la tarea indicada.

Ejemplos:

```
M: {0=>1,1=>2,2=>3}, pred: even, v:[0,1,2]     ->     output:[2]
M: {0=>1,1=>2,2=>3}, pred: odd,  v:[0,1,2]     ->     output:[1,3]
M: {1=>0,2=>1,3=>2}, pred:odd,   v:[1,2,3]     ->     output:[1]
M: {1=>0,2=>1,3=>2}, pred:even,  v:[1,2,3]     ->     output:[0,2]
```

Ayuda: Recorrer los elementos de `v`, aplicar `M` y verificar si la imagen satisface el predicado o no. Si lo satisface insertar ese elemento imagen en `w`.

[Ej. 2] **[reciprocal]** Dadas dos correspondencias `M1` y `M2` determinar si una es la inversa de la otra, es decir si para cada par de asignación `y=M1[x]` debe haber un par `x=M2[y]` y viceversa (para todo par en `M2` existe el inverso en `M1`).

Consigna: Escribir la función `bool reciprocal(map<int,int> &M1,map<int,int> &M2);` que realiza la tarea indicada.

Ejemplos:

```
M1:{0=>1,1=>2,2=>3},       M2:{1=>0,2=>1,3=>2},       -> true
M1:{0=>1,1=>2,2=>3},       M2:{1=>0,2=>4,3=>2},       -> false
M1:{0=>1,1=>2,2=>2},       M2:{1=>0,2=>1},            -> false
M1:{0=>1,1=>0,2=>3,3=>2},  M2:{0=>1,1=>0,2=>3,3=>2},  -> true
```

[Ej. 3] **[only1]** Dadas dos conjuntos `A` y `B` la differencia simétrica `C` es el conjunto de los elementos que están en `A` o `B` pero no en ambos a la vez. El nombre de diferencia simétrica, proviene por contraposición con la diferencia `A-B` que son los elementos que están en `A` pero no en `B`.

Otra forma de verlo es que si, dado un elemento `x` contamos en cuantos conjuntos está, la diferencia simétrica son aquellos que están en sólo 1 conjunto. Esta definición se puede extender a cualquier número de conjuntos.

Consigna: Escribir una función `void only1(vector< set<int> >&VS,set<int> &DS);` que realiza la tarea indicada.

Ejemplos:

```
VS=[{0,1,2},{1,2,3}]           -> DS={0,3}
VS=[{0,1,2},{1,2,3},{2,4,5}], -> DS=[0,3,4,5]
```

Ayuda: Recorrer todos los elementos `x` de todos los conjuntos y mantener una correspondencia `map<int,int> M` que almacena para cada elemento `x` la cantidad de conjuntos en los que está. Finalmente recorrer las asignaciones de `M` e insertar en `DS` sólo los elementos que tienen un conteo exactamente igual a 1.

[Ej. 4] **[trust-game]** Consideremos el siguiente juego. Hay dos jugadores de ambos lados de una máquina. En cada movida cada uno de los jugadores puede poner 1 moneda en la máquina. Si el jugador

- Si `p1` pone una moneda el otro `p2` recibe 3 monedas,
- Si `p1` no pone la moneda no recibe nada.

Ninguno de los dos sabe cuál es la jugada del otro, pero tienen acceso a toda la historia de jugadas entre ellos.

Por lo tanto, si ambos "colaboran" los dos ganan ambos 2 monedas, pero si uno de ellos "hace trampa" (no pone la moneda) tiene una ganancia adicional ya que no puso la moneda, es decir gana 3 (mientras que el oponente pierde 1). Finalmente si ambos "hacen trampa" (no ponen la moneda) ninguno de los dos gana nada (ganancia 0 para ambos).

La cuestión es cuál es la mejor estrategia de juego, es decir que de la máxima ganancia, en un entorno en el cual los otros jugadores pueden tener diferentes estrategias.

```
enum move_t {NONE, COLLAB, CHEAT};
class player_t {
  public:
  virtual move_t move(vector<move_t> &me,vector<move_t> &you)=0;
  virtual string state() {return string("unknown"); }
};
```

Consigna1: Escribir instancias (clases derivadas) de la clase `player_t` con las siguientes características:

- `collabr_t`: Siempre colabora.
- `cheater_t`: Siempre hace trampa.
- `copycat_t`: (Copión) Copia lo que hizo el otro jugador en la jugada previa.
- `grudger_t`: (Resentido) Inicialmente colabora, pero apenas es traicionado una sola vez, no colabora nunca más.
- `copysmart_t`: (Copión astuto) Mira las dos jugadas previas del oponente. Si fue traicionado las dos veces traiciona, caso contrario colabora.
- `random_t`: (Aleatorio) Tira la moneda y colabora o traiciona con probabilidades 50 %.

Nota: El método **`state()`** retorna un string que describe al jugador, por ejemplo **`cheater_t`** puede retornar **`"siempre traiciona"`**.

Consigna2: Escribir una función
`void game(player_t &p1,player_t &p2,int nrounds,int &score1,int &score2);` que toma dos jugadores **`p1`** y **`p2`** y los hace jugar **`nrounds`** jugadas. Retorna también (score, cantidad de monedas ganadas) de cada jugador.

Referencia: Al hacer jugar los diferentes jugadores entre si debe dar lo siguiente (para 5000 jugadas cada par):

```
collaborator: score 10000 (2.000 per move), | collaborator  score: 10000 (2.000 per move)
collaborator: score -5000 (-1.000 per move), | cheater  score: 15000 (3.000 per move)
collaborator: score 10000 (2.000 per move), | copycat  score: 10000 (2.000 per move)
collaborator: score 10000 (2.000 per move), | copysmart  score: 10000 (2.000 per move)
collaborator: score 10000 (2.000 per move), | grudger  score: 10000 (2.000 per move)
collaborator: score 2356 (0.471 per move), | random  score: 12548 (2.510 per move)
===== Total score collaborator: 37356 (1.245/move)
cheater: score 15000 (3.000 per move), | collaborator  score: -5000 (-1.000 per move)
cheater: score 0 (0.000 per move), | cheater  score: 0 (0.000 per move)
cheater: score 3 (0.001 per move), | copycat  score: -1 (-0.000 per move)
cheater: score 6 (0.001 per move), | copysmart  score: -2 (-0.000 per move)
cheater: score 3 (0.001 per move), | grudger  score: -1 (-0.000 per move)
cheater: score 7371 (1.474 per move), | random  score: -2457 (-0.491 per move)
===== Total score cheater: 22383 (0.746/move)
copycat: score 10000 (2.000 per move), | collaborator  score: 10000 (2.000 per move)
copycat: score -1 (-0.000 per move), | cheater  score: 3 (0.001 per move)
copycat: score 10000 (2.000 per move), | copycat  score: 10000 (2.000 per move)
copycat: score 10000 (2.000 per move), | copysmart  score: 10000 (2.000 per move)
copycat: score 10000 (2.000 per move), | grudger  score: 10000 (2.000 per move)
copycat: score 4943 (0.989 per move), | random  score: 4947 (0.989 per move)
===== Total score copycat: 44942 (1.498/move)
copysmart: score 10000 (2.000 per move), | collaborator  score: 10000 (2.000 per move)
copysmart: score -2 (-0.000 per move), | cheater  score: 6 (0.001 per move)
copysmart: score 10000 (2.000 per move), | copycat  score: 10000 (2.000 per move)
copysmart: score 10000 (2.000 per move), | copysmart  score: 10000 (2.000 per move)
copysmart: score 10000 (2.000 per move), | grudger  score: 10000 (2.000 per move)
copysmart: score 3709 (0.742 per move), | random  score: 8857 (1.771 per move)
===== Total score copysmart: 43707 (1.457/move)
grudger: score 10000 (2.000 per move), | collaborator  score: 10000 (2.000 per move)
grudger: score -1 (-0.000 per move), | cheater  score: 3 (0.001 per move)
grudger: score 10000 (2.000 per move), | copycat  score: 10000 (2.000 per move)
grudger: score 10000 (2.000 per move), | copysmart  score: 10000 (2.000 per move)
grudger: score 10000 (2.000 per move), | grudger  score: 10000 (2.000 per move)
grudger: score 7412 (1.482 per move), | random  score: -2468 (-0.494 per move)
===== Total score grudger: 47411 (1.580/move)
random: score 12444 (2.489 per move), | collaborator  score: 2668 (0.534 per move)
random: score -2479 (-0.496 per move), | cheater  score: 7437 (1.487 per move)
random: score 5085 (1.017 per move), | copycat  score: 5081 (1.016 per move)
random: score 8753 (1.751 per move), | copysmart  score: 3693 (0.739 per move)
random: score -2453 (-0.491 per move), | grudger  score: 7375 (1.475 per move)
random: score 5109 (1.022 per move), | random  score: 5017 (1.003 per move)
===== Total score random: 26459 (0.882/move)
```

Nota de color:

- Otro nombre para este probema es el del **dilema del Prisionero** (goo.gl/iEXtMF, youtu.be/UkXI-zPcDIM).
- Es estudiado en **Teoría de Juegos**, la conclusión es que el juego tiene un **punto de equilibrio de Nash**: a saber **todos traicionan**, es decir que cada jugador si cambia su estrategia (esto es

colabora) pierde y por lo tanto retorna al punto de equilibrio (traicionar). Sin embargo la ganancia en este punto individual y colectiva es baja (nula).
Notablemente, la ganancia total e individual máxima se obtiene si todos colaboran, ya que en promedio ganan 2 unidades por jugada, bastante más que en el punto de equilibrio.

- En el sitio ncase.me/trust este concepto es explicado en detalle, con animaciones muy interesantes e implicancia en temas sociales como la confianza y la cooperación.

[Ej. 5] **[is-inside-cc]** Dados 3 puntos $\mathbf{x}_j \in \mathbb{R}^2$, la **circunferencia circunscripta** (CC) (en inglés *circumcircle*) es el círculo que pasa por los 3 puntos al mismo tiempo, por lo tanto el centro del círculo $\mathbf{x}_c$ es equidistante de los 3 puntos, es decir $\|\mathbf{x}_j - \mathbf{x}_c\| = R$. Dados los 3 puntos, podemos encontrar el centro de la CC planteando que dados dos de los puntos $\mathbf{x}_0, \mathbf{x}_1$ el centro $\mathbf{x}_c$ debe ser equidistante de ambos, por lo tanto debe estar en el plano bisector del segmento que los une, es decir

$$\begin{aligned} (\mathbf{x}_c - \mathbf{x}_0) \cdot (\mathbf{x}_1 - \mathbf{x}_0) &= (\mathbf{x}_1 - \mathbf{x}_c) \cdot (\mathbf{x}_1 - \mathbf{x}_0), \\ \implies \quad (\mathbf{x}_c - \mathbf{x}_{c,01}) \cdot \Delta\mathbf{x}_{01} &= 0, \end{aligned} \tag{1}$$

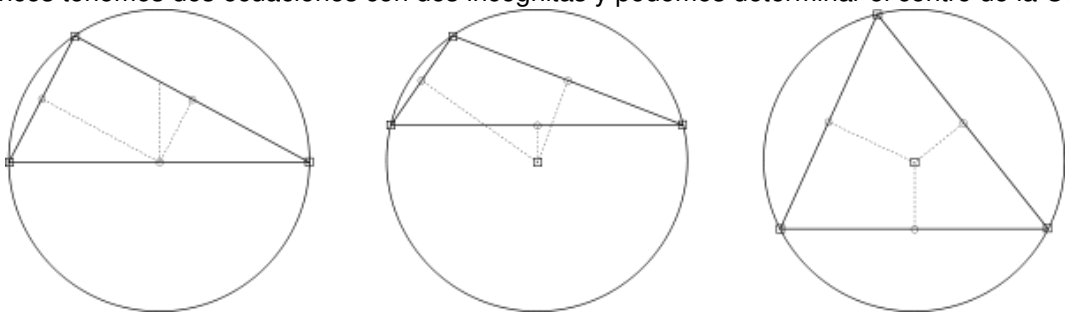
donde

$$\begin{aligned} \mathbf{x}_{c,01} &= (\mathbf{x}_0 + \mathbf{x}_1)/2, \\ \Delta\mathbf{x}_{01} &= (\mathbf{x}_1 - \mathbf{x}_0), \end{aligned} \tag{2}$$

son el punto medio del segmento $\mathbf{x}_0 - \mathbf{x}_1$ y el vector orientado a lo largo del mismo. Notar que esto representa una ecuación lineal para las coordenadas de $\mathbf{x}_c$. Si planteamos lo mismo para el segmento $\mathbf{x}_0 - \mathbf{x}_2$

$$\begin{aligned} \implies \quad (\mathbf{x}_c - \mathbf{x}_{02}) \cdot \Delta\mathbf{x}_{02} &= 0, \\ \mathbf{x}_{c,02} &= (\mathbf{x}_0 + \mathbf{x}_2)/2, \end{aligned} \tag{3}$$

entonces tenemos dos ecuaciones con dos incógnitas y podemos determinar el centro de la CC $\mathbf{x}_c$.



$$\begin{aligned} \mathbf{A}\mathbf{x}_c &= \mathbf{b}, \\ \mathbf{A} &= \begin{bmatrix} \Delta\mathbf{x}_{01}{}^T \\ \Delta\mathbf{x}_{02}{}^T \end{bmatrix}, \\ \mathbf{b} &= \begin{bmatrix} \Delta\mathbf{x}_{01} \cdot \mathbf{x}_{c,01} \\ \Delta\mathbf{x}_{02} \cdot \mathbf{x}_{c,02} \end{bmatrix}, \end{aligned} \tag{4}$$

El concepto se puede extender a 3 dimensiones. En 3D tendremos 4 puntos y planteamos que el centro de la esfera circunscripta CC debe pertenecer a los planos bisectores de los 3 segmentos orientados

desde el primero $\mathbf{x}_0$

$$\begin{aligned} \mathbf{A}\mathbf{x}_c &= \mathbf{b}, \\ \mathbf{A} &= \begin{bmatrix} \Delta\mathbf{x}_{01}{}^T \\ \Delta\mathbf{x}_{02}{}^T \\ \Delta\mathbf{x}_{03}{}^T \end{bmatrix}, \\ \mathbf{b} &= \begin{bmatrix} \Delta\mathbf{x}_{01} \cdot \mathbf{x}_{c,01} \\ \Delta\mathbf{x}_{02} \cdot \mathbf{x}_{c,02} \\ \Delta\mathbf{x}_{03} \cdot \mathbf{x}_{c,03} \end{bmatrix}, \end{aligned} \tag{5}$$

En general, en $\mathbb{R}^n$ tendremos $n+1$ puntos $\mathbf{x}_j$. Tomando uno arbitrario de ellos, por ejemplo el $\mathbf{x}_0$ y considerando los n segmentos $\mathbf{x}_0 - \mathbf{x}_j$ con $j = 1, n$ y para cada uno de ellos podemos plantear una ecuación y por lo tanto podemos construir un sistema lineal con $\mathbf{A} \in \mathbb{R}^{n \times n}$ y $\mathbf{b} \in \mathbb{R}^n$ para determinar el centro de la **Hiper-Esfera Circunscripta** (HEC).

Dado un punto $\mathbf{x} \in \mathbb{R}^n$ y $n+1$ puntos $\mathbf{x}_j \in \mathbb{R}^n$ podemos determinar si $\mathbf{x}$ se encuentra dentro de la HEC correspondiente a los $\{\mathbf{x}_j\}$ con el siguiente algoritmo

- Construir la matriz $\mathbf{A}$, y el miembro derecho $\mathbf{b}$ de acuerdo a (**??**).
- Resolver el sistema y determinar el centro $\mathbf{x}_c$ de la HEC.
- Determinar el radio de la HEC $R = \|\mathbf{x}_0 - \mathbf{x}_c\|$.
- Determinar si el punto está en la HEC $\|\mathbf{x} - \mathbf{x}_c\| < R$

Consigna: Escribir una función **`bool is_insidecc(MatrixXd &X,VectorXd &xprobe);`** que determina si el punto **`xprobe`** está contenido dentro de la HEC determinada por los puntos **`X`**.

Notas:

- La función debe poder utilizarse para dimensiones **`ndim>=2`**.
- **`X`**: es una matriz **`MatrixXd`** de tamaño **`[ndim,(ndim+1)]`** donde **`ndim`** es la dimensión del problema. Cada columna representa uno de los **`xj`**.
- **`xprobe`**: es un **`VectorXd`** de longitud **`ndim`**.

[Ej. 6] **[lsqfit]** Dada un serie de valores $\mathbf{x}_j$ $j \in [0, N)$ y una serie de valores y_j que representan aproximaciones a la aplicación de una función $y_j = f(x_j)$ queremos hallar una aproximación en una base ϕ_k, para una serie de términos m es decir

$$y_j = \sum_k a_k \, \phi(x_k). \tag{6}$$

Los ϕ_k pueden ser potencias ($\phi_k(x) = x^k$), series trigonométricas $\{1, \sin(x), \cos(x), \sin(2x), \cos(2x), ...\}$ Para ello elegimos los a_k por el método de **mínimos cuadrados** (LSQ). Es decir, en forma matricial podemos poner

$$\mathbf{y} = \mathbf{\Phi}\mathbf{a} \tag{7}$$

donde $\mathbf{y} \in \mathbb{R}^N$, $\mathbf{\Phi} \in \mathbb{R}^{N \times m}$, y $\mathbf{a} \in \mathbb{R}^m$. Los elementos de $\mathbf{\Phi}$ son

$$\Phi_{jk} = \phi_k(x_j). \tag{8}$$

Asumiendo que tenemos más puntos que términos de la expansión ($N \geq m$), el sistema está sobredeterminado. En este caso puede demostrarse que LSQ corresponde a premultiplicar el sistema (**??**) por $\mathbf{\Phi}^T$ es decir

$$\begin{aligned} (\mathbf{\Phi}^T\mathbf{\Phi})\,\mathbf{a} &= (\mathbf{\Phi}^T\mathbf{y}), \\ \mathbf{H}\mathbf{a} &= \mathbf{b}, \end{aligned} \tag{9}$$

es decir que se puede obtener $\mathbf{a}$ resolviendo el sistema lineal.

Consigna: Escribir una función
`void lsqfit(VectorXd &X,VectorXd &Y,basis_t basis, int nterms,VectorXd &a);` que toma un vector de abscisas **`X`** de tamaño **`N`** y un vector de valores de la función **`Y`** del mismo tamaño, y determina los coeficientes de la expansión **`a`** en **`nterms`** términos de la base **`basis`**.

La base basis es un puntero a función, cuyo prototipo es
`typedef double (*basis_t)(double x,int j);`

Ejemplos:

```
1) X=[0,1,2,3],Y=[0,1,2,3], basis=poly {1,x,x^2...},nterms=2   =>     a=[0,1]
2) X=[0,1,2,3],Y=[1,2,3,4], basis=poly [1,x,x^2...],nterms=2   =>     a=[1,1]
3) X=[0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1],
      Y=[1,1.094,1.178,1.250,1.310,1.357,1.389,1.409,1.414,1.404,1.381] (y=sin(x)+cos(x)),
      basis=trigo{1,sin(x),cos(x),sin(2*x),cos(2*x),....}, nterms=3 =>  a=[0,1,1]
```

Ayuda:

- Construir la matriz **`Phi`**.
- Construir las matrices **`H`** y **`b`**.
- Resolver el sistema (usar los métodos **`lu()`** y **`solve()`** de Eigen.

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/texstuff-1.4.0-70-g09a0bdc4/Tue Jun 22 10:44:39 2021 -0300