

## Curso de Programación en C++.

### TPL2. Trabajo Práctico de Laboratorio. [2017-10-23]

PASSWD PARA EL ZIP: **2VRL XZDN 57Z6**

## Ejercicios

[Ej. 1] **[separate]** Dada una **función de mapeo** `int f(int)` y un conjunto de enteros `S` queremos construir la **función inversa** consideremos el conjunto de los valores `yj=f(xj)` para todos los `xj` en `S`. Queremos construir una estructura `y2x` que nos permita obtener para cada valor de `y` los valores `x` tales que `f(x)=y`. Como `f` no es necesariamente **biyectiva** puede haber varios `x` que cumplen la condición, por eso usaremos para esta estructura un `map<int, set<int>>` es decir, para cada valor de la imagen `y` el conjunto `y2x[y]` es el conjunto de los valores de `x` tales que `y=f(x)`.

**Consigna:** escribir una función

`void separate(set<int> &S, mapfun_t f, map<int, set<int>> &y2x);` que realiza la tarea indicada.

**Ejemplo:**

- `S : [-4,-3,0,1,2,3,4,5,6,7]`, `f=sign(x)`,  
`y2x={-1=>[-4,-3], 0=>[0], 1=>[1,2,3,4,5,6,7]}`
- `S: [-4,-3,0,1,2,3,4,5,6,7]`, `f=abs(x)`,  
`y2x={0=>[0], 1=>[1], 2=>[2], 3=>[-3,3], 4=>[-4,4], 5=>[5], 6=>[6], 7=>[7]}`
- `S: [-4,-3,0,1,2,3,4,5,6,7]`, `f=mod(x,3)`,  
`y2x={0=>[-3,0,3,6], 1=>[1,4,7], 2=>[-4,2,5]}`

[Ej. 2] **[min-path] (Problema del Agente Viajero)** Dado un grafo `graph_t G` escribir una función `int min_path(graph_t &G, vector<int> &pathmin);` en el grafo `G` que determina el camino de longitud mínima `pathmin`. La clase `graph_t` está definida por la siguiente interfaz virtual pura,

```
class graph_t {
public:
    virtual int size()=0; // cantidad de vertices
    virtual int dist(int i,int j)=0; // distancia entre los vertices i,j
};
```

El camino está representado por un `vector<int> path`. Los caminos son cíclicos es decir vuelven al vértice de partida. Es decir que si `path=[0,1,2,3]` entonces hay que sumar las aristas `0-1,1-2,2-3,3-0`

El algoritmo consiste en recorrer todos los caminos posibles (hay  $N!$  posibilidades) se provee la función `next_path(p)` la cual dado un camino `p` lo actualiza al siguiente camino. La función retorna `false` cuando llega al último es decir cuando no lo puede actualizar más. Para recorrer todos los caminos hay que entonces inicializar el camino con `[0 1 2 ... N-1]` y luego aplicar sucesivamente `next_path()` hasta que retorne `false`.

```
vector<int> path(N);
for (int j=0; j<N; j++) path[j] = j;
```

```
while (1) {
    // Analizar 'path'
    if (!next_path(path)) break;
}
```

**Consigna 1:** Escribir la función `int min_path(graph_t &G, vector<int> &pathmin);`

**Ejemplos:** Si el grafo es un círculo `circle_t` donde los `dist(j,k)=1` si `j==(k+1)%N` o viceversa, caso contrario `dist(j,k)=Inf` (usaremos `Inf=10000`). El camino mínimo tiene longitud `N` y debe ser `[0,1,...,N-1]` o cualquier rotación del mismo o inversión, e.g. `[2,...,N-1,0,1]` o `[N-1,N-2,...,1,0]`.

**Consigna 2:** Escribir los siguientes grafos (clases derivadas):

- **full\_t:** Todos los vértices están conectados por la misma distancia `dist(i,j)=1`. El camino mínimo puede ser cualquiera y es de longitud `N`. Calcular el camino mínimo para grafos `full_t` de 4 y 7 vértices.
- **grid\_t:** Consiste en una grilla 2D de `Nx x Ny` vértices. El vértice en la posición 2D `(j,k)` le corresponde el índice `n=k*Nx+j`. La distancia entre los nodos `n1=(j1,k1)` y `n2=(j2,k2)` es la norma  $L_1$  de la distancia: `dist(n1,n2)=|j2-j1|+|k2-k1|`. Calcular el camino mínimo para grafos `grid_t` 3x3 y 4x2 vértices.

[Ej. 3] **[mkorth]** (Eigen) Dada una matriz `Q` buscar una matriz *cercana* `O` tal que `O` es ortogonal, es decir  $OO^T = I$ . Para ello asumir que la corrección es  $\Delta Q$  de manera que queremos que

$$(Q + \Delta Q)(Q + \Delta Q)^T = QQ^T + Q\Delta Q^T + \Delta QQ^T + \Delta Q\Delta Q^T = I \quad (1)$$

Despreciando el último término y asumiendo  $Q\Delta Q^T = \Delta QQ^T$  queda

$$\begin{aligned} \Delta Q &= \frac{1}{2}(I - QQ^T)(Q^T)^{-1}, \\ Q &\leftarrow Q + \Delta Q; \end{aligned} \quad (2)$$

**Consigna:** Escribir una función `void mkorth(MatrixXd &Q);` que realiza la tarea explicada aplicando iterativamente (2) hasta que  $\|\Delta Q\| < \text{tol}=1e-7$

## 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