UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL

Departamento Electrónica, Automática e Informática Industrial

APELLIDOS

NOMBRE

Nº Mat.

Calificación

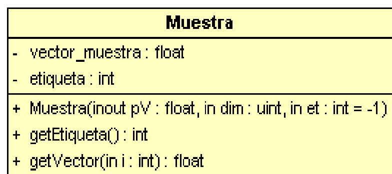
ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º GRUPO Julio 2014

2. Problema de ADOO (10 puntos - 40 minutos)

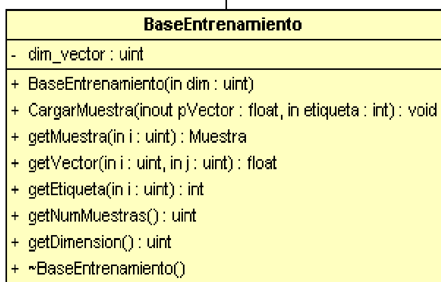
Los clasificadores son parte esencial de muchos programas de Ingeniería. Normalmente, hay un conjunto de muestras de entrenamiento, las cuales tienen definido tanto el vector de características como la etiqueta que se asociada a la clase que le corresponde. Hay muchos tipos de clasificadores: Bayes, redes neuronales, lógica borrosa,… De los clasificadores más simples están los denominados kNN (*k-Nearest Neighbours*). En esta primera versión se va a implementar un clasificador de tipo kNN: ante una nueva muestra, ésta queda clasificada con la etiqueta de la muestra de entrenamiento con menor distancia Euclídea entre vectores. Se pide;

1. Ingeniería Inversa de las clases *Muestra* y *BaseEntrenamiento* (3 puntos).
2. Diagrama de clase de diseño DCD en UML de las clases *Clasificador*, *kVecinos* y *Factoria*. Indique los patrones empleados (3 puntos).
3. Implementación de estas últimas clases en C++ (4 puntos).

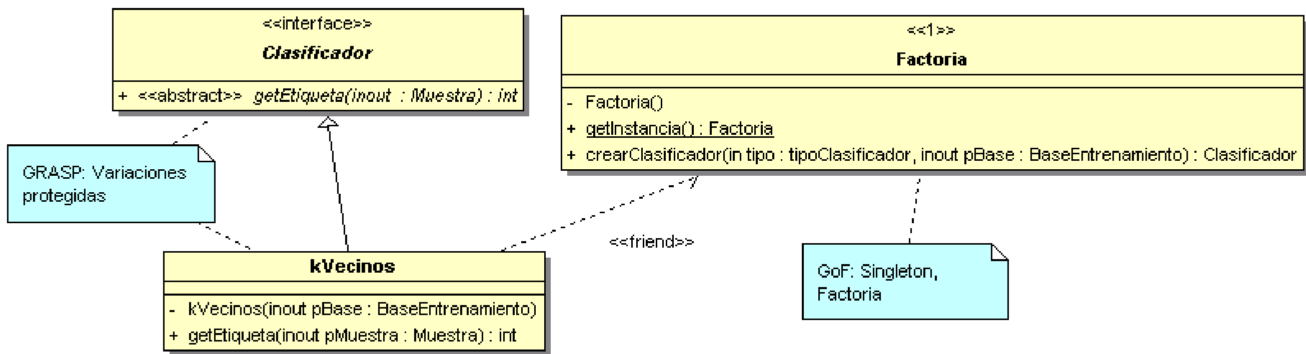


listaMuestras

<<vector>>




```cpp
#include <vector>
#include <iostream>
using namespace std;

class Muestra {
  vector<float> vector_muestra;
  int etiqueta;
  public:
  Muestra(float *pV,unsigned dim,int et=-1): etiqueta(et){
       for(unsigned i=0;i<dim;i++)
        vector_muestra.push_back(pV[i]);
       }
  int getEtiqueta(){return etiqueta;}
  float getVector(int i) {return vector_muestra[i];}
};

class BaseEntrenamiento{
  vector<Muestra *> listaMuestras;
  unsigned dim_vector;
  public:
  BaseEntrenamiento(unsigned dim):dim_vector(dim){}
  void CargarMuestra(float *pVector,int etiqueta){
       listaMuestras.push_back(new
       Muestra(pVector,dim_vector,etiqueta));
       }
  Muestra * getMuestra(unsigned i) {return listaMuestras[i];}
  float getVector(unsigned i, unsigned j) {return
       listaMuestras[i]->getVector(j);}
  int getEtiqueta(unsigned i) {return
       listaMuestras[i]->getEtiqueta();}
  unsigned getNumMuestras() { return listaMuestras.size(); }
  unsigned getDimension() {return dim_vector;}
  ~BaseEntrenamiento() {
    for(unsigned i=0; i<listaMuestras.size();i++)
       delete listaMuestras[i];
     }
};

typedef enum{kNN,Bayes,RandomForest} tipoClasificador;
class Clasificador{…};
class kVecinos{…};
class Factoria {…};

int main() {
 unsigned dim = 2;
 unsigned numMuestras = 5;
 float vectores_muestra[5][2] = { {1.0f,1.0f},{0.0f,0.0f},
                      {2.0f,2.0f}, {2.0f,3.0f}, {3.0f,2.0f} };
 int etiquetas_muestra[] = {1,1,2,2,2};
 BaseEntrenamiento laBase(dim);

 for(unsigned i=0;i<numMuestras;i++)
  laBase.CargarMuestra(vectores_muestra[i],etiquetas_muestra[i]);

 float vector_nuevo[]={1.0,2.0};
 Muestra laMuestra(vector_nuevo,dim);

 // Clasificar la muestra
 Factoria laFactoria = Factoria::getInstancia();
 Clasificador *pClasificador =
      laFactoria.crearClasificador(kNN,&laBase);
 cout <<"La muestra con vector: "<< laMuestra.getVector(0)<<" "
 << laMuestra.getVector(1) << " tiene la etiqueta: "
 << pClasificador->getEtiqueta(&laMuestra) << endl;

return 0;
}
```

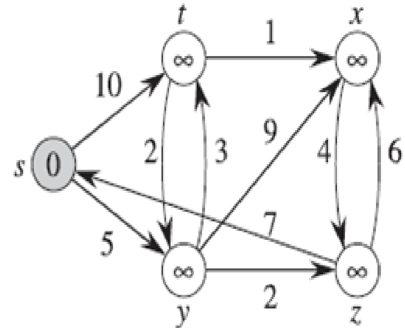



```
class Clasificador{
public:
        virtual int getEtiqueta(Muestra *) = 0;
};

float distancia_muestras(Muestra *pM1, Muestra *pM2,unsigned dim)
{
        float resultado=0.0f;
        for(unsigned i=0; i< dim;i++)
                resultado += (pM1->getVector(i)-pM2->getVector(i))*
                                (pM1->getVector(i)-pM2->getVector(i));
        return resultado;
}

class kVecinos: public Clasificador {
        friend class Factoria;
        BaseEntrenamiento *pBaseDatos;
        kVecinos(BaseEntrenamiento *pBase):pBaseDatos(pBase) {}
public:
        virtual int getEtiqueta(Muestra *pMuestra) {
                int etiqueta = -1;
                unsigned numMuestras = pBaseDatos->getNumMuestras();
                unsigned dim = pBaseDatos->getDimension();
                float min_distancia = 1e10;
                for(unsigned i=0;i<numMuestras;i++){
                        float dist = distancia_muestras(pMuestra,pBaseDatos->getMuestra(i),dim);
                        if(dist < min_distancia){
                                min_distancia = dist;
                                etiqueta= pBaseDatos->getEtiqueta(i);
                        }
                }
                return etiqueta;
        }
};


class Factoria
{
        Factoria(){}
public:
    static Factoria& getInstancia()
        {
            static Factoria unicaInstancia;
            return unicaInstancia;
        }
    Clasificador* crearClasificador(tipoClasificador tipo, BaseEntrenamiento *pBase )
        {
         if(tipo == kNN) return new kVecinos(pBase);
         else return NULL;
        }

};
```

	APELLIDOS		
escuela técnica superior de ingeniería y diseño industrial	NOMBRE		Nº Mat.
UNIVERSIDAD POLITÉCNICA DE MADRID ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL			Calificación
Departamento Electrónica, Automática e Informática Industrial	ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES		
	CURSO 4º GRUPO Julio 2014		

3. Problema de algoritmia (5 puntos - 30 minutos)

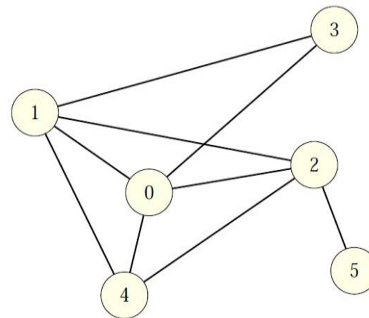
1. El grafo de la figura representa una configuración de trayectorias en una red local de transporte de una planta; las etiquetas en los vértices representan distancias entre nodos. Se desea encontrar las distancias más cortas desde el nodo origen *s* a cualesquiera de los otros vértices (t, x, y, z). Los pesos en los vértices representan las distancias más cortas desde *s*. Al empezar la búsqueda el peso de *s* es 0 (y no cambia, puesto que es el punto de partida) y el resto de pesos es ∞ (ver figura). Al terminar la búsqueda los pesos de t, x, y, z corresponderán con las distancias más cortas desde *s*.



Se pide: Escriba la traza de ejecución del algoritmo de Dijkstra que resuelve este problema. Indique, en cada paso, la lista de vértices abiertos, los pesos de los vértices y el candidato elegido para continuar.

2. Para el grafo de la figura indique:

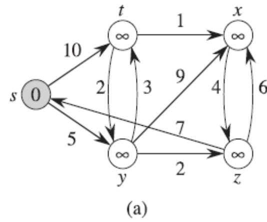
a) El número de clique
b) El número cromático
c) El grado máximo del grafo
d) Grado de conectividad



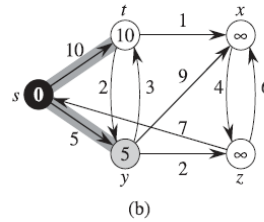
3. Para la implementación del algoritmo de *Dijkstra* del ejercicio 1 es necesario elegir de entre los vértices candidatos el que tiene menor distancia al origen. El código que figura a continuación propone una posible forma solucionar este problema. Complételo **convenientemente** e indique el resultado que aparecerá en pantalla para la solución propuesta.

```
#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>

using namespace std;
#define EMPTY -1
#define NUM_ELEM 5

struct elem_t{
        int vertex, weight;
        elem_t():vertex(EMPTY), weight(EMPTY){}
        elem_t(int v, int w);
        friend ostream& operator <<(ostream& o, const elem_t& e);
};
struct min_weight{      //functor
        bool operator()(elem_t e1, elem_t e2);
};
void main(){
        vector<elem_t> vec(NUM_ELEM);

        //carga del vector (caso de uso)
        for(int v=0; v<NUM_ELEM; v++){
                elem_t e(v,2*NUM_ELEM-v); vec[v]=e;
        }

        sort(  ,     ,     );
        copy(   ,     , ostream_iterator<elem_t>(cout, " "));
}
```

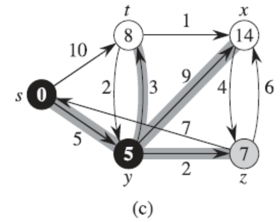
2.
a) número de clique = 4
b) número cromático = 4
c) grado máximo = 4
d) conectividad = 1



(a)

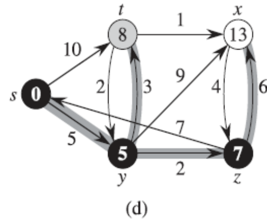


(b)

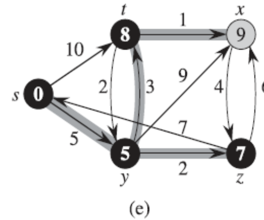


(c)

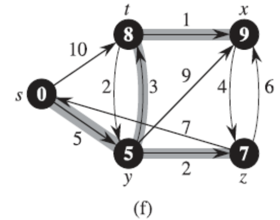
3.



(d)



(e)



(f)

```
#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>

using namespace std;
#define EMPTY -1
#define NUM_ELEM 5

struct elem_t{
	int vertex, weight;
	elem_t():vertex(EMPTY), weight(EMPTY){}
	elem_t(int v, int w):vertex(v), weight(w){}
	friend ostream& operator <<(ostream& o, const elem_t& e){
		o<<"("<<e.vertex<<","<<e.weight<<")"<<endl; return o;
	}
};

struct min_weight{                                                  //functor
	bool operator()(elem_t e1, elem_t e2){
		return(e1.weight<e2.weight);
	}
};

void main(){
	vector<elem_t> vec(NUM_ELEM);

	//carga del vector (caso de uso)
	for(int v=0; v<NUM_ELEM; v++){
		elem_t e(v,2*NUM_ELEM-v);
		vec[v]=e;
	}

	//ordenar
	sort(vec.begin(), vec.end(), min_weight());

	//salida en pantalla
	copy(vec.begin(), vec.end(), ostream_iterator<elem_t>(cout, " "));
}
```

escuela técnica superior de ingeniería y diseño industrial
UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL

Departamento Electrónica, Automática e Informática Industrial

APELLIDOS

NOMBRE Nº Mat.

Calificación

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º GRUPO Julio 2014

4. Problema de Sistemas Operativos (10 puntos - 40 minutos)

PARTE A (5 puntos). Dado el siguiente programa, responde a las preguntas:

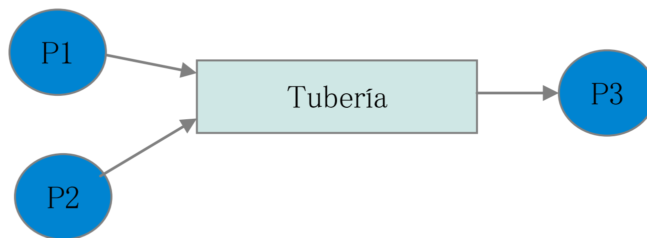
```
01  #include <stdio.h>
02
03  int main(void) {
04    struct elemento dato;
05    int tuberia[2];
06    int pid1, pid2;
07
08    if (pipe(tuberia) < 0) {
09          perror("No se puede crear la tuberia");
10          exit(0);
11    }
12
13    switch ((pid1=fork()) {
14          case -1:
15                perror("Error");
16                close(tuberia[0]);
17                close(tuberia[1]);
18                exit(0);
19          case 0:
20                close(tuberia[0]);
21                break;
22          default:
23              switch((pid2 = fork()) {
24                case -1:
25                      perror("Error");
26                      close(tuberia[0]);
27                      close(tuberia[1]);
28                      kill(pid1, SIGKILL);
29                      exit(0);
30                case 0:
31                      close(tuberia[1]));
32                      break;
33
34                default:
35                      close(tuberia[0]);
36          }
37    }
38  }
```

1. Indica el número de cuántos procesos, incluyendo al primer padre, se producen al invocar el programa:

   3

2. Dibuja el diagrama de comunicaciones entre los procesos a través de la tubería, con las correspondientes flechas que indiquen la apertura de los descriptores:

El Proceso 1 es el padre. P2 y P3 son los procesos hijos.



3. Marca la opción verdadera.
   a) Si no hay ningún proceso escritor y la tubería está vacía, la operación de lectura devuelve fin de fichero.
   b) La operación de lectura sobre una tubería no es atómica, por lo que hay que contar con algún mecanismo de sincronización adicional.
   c) Si la tubería almacena M bytes y se quieren leer n bytes, si M < n, entonces la operación de lectura devuelve error pero no le bloquea.
   d) La lectura de una tubería obtiene los datos almacenados en la misma en el orden contrario al que fueron introducidos.

   Opción a

4. Indica **detrás** de qué línea o líneas del código se debería incluir la siguiente llamada para que se produzca la lectura y escritura entre procesos:

```
read(tuberia[0], (char *) &dato, sizeof(struct elemento));
```

   31

5. Indica **detrás** de qué línea o líneas del código se debería incluir la siguiente llamada para que se produzca la lectura y escritura entre procesos:

```
write(tuberia[1], (char *) &dato, sizeof(struct elemento));
```

   20 y 35

escuela técnica superior de ingeniería y diseño industrial
UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL

Departamento Electrónica, Automática e Informática Industrial

APELLIDOS

NOMBRE

Nº Mat.

Calificación

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º GRUPO Julio 2014

PARTE B (5 puntos). Dado el siguiente programa, a falta de completar algunas líneas de código, responde a las preguntas:

```
01 #define MAX_BUFFER 1024
02 #define DATOS_A_PRODUCIR 100000

03 pthread_mutex_t mutex;
04 pthread_cond_t lleno;
05 pthread_cond_t vacio;
06 int n_elementos;
07 int buffer[MAX_BUFFER];

08 void Productor(void) {
09     int dato, i ,pos = 0;
10     for(i=0; i < DATOS_A_PRODUCIR; i++ ) {
11         dato = i;
12         while (CODIGO 1){
13                 pthread_cond_wait(CODIGO 3, &mutex);
14         }
15         buffer[pos] = i;
16         pos = (pos + 1) % MAX_BUFFER;
17         n_elementos ++;
18         pthread_cond_signal(&vacio);
19    }
20    pthread_exit(0);
21 }

22 void Consumidor(void){
23     int dato, i ,pos = 0;
24     for(i=0; i < DATOS_A_PRODUCIR; i++ ) {
25          while ( CODIGO 2){
26                  pthread_cond_wait(CODIGO 4, &mutex);
27           }
28           dato = buffer[pos];
29           pos = (pos + 1) % MAX_BUFFER;
30           n_elementos --;
31           pthread_cond_signal(&lleno);
32           printf("Consume %d \n", dato);
33    }
34    pthread_exit(0);
35 }

36 int main(int argc, char *argv[]){
37    pthread_t th1, th2;
38    pthread_mutex_init(&mutex, NULL);
39    pthread_cond_init(&lleno, NULL);
40    pthread_cond_init(&vacio, NULL);
41    CODIGO 5
42    pthread_mutex_destroy(&mutex);
43    pthread_cond_destroy(&lleno);
44    pthread_cond_destroy(&vacio);
45    exit(0);
46 }
```

6. Indica el código por el que hay que sustituir **CODIGO 1 y CODIGO 2**:

   CODIGO 1: `n_elementos == MAX_BUFFER`
   CODIGO 2: `n_elementos == 0`

7. Indica el código por el que hay que sustituir **CODIGO 3 y CODIGO 4**:

   CODIGO 3: `&lleno`
   CODIGO 4: `&vacio`

8. Indica **detrás** de qué línea o líneas incluirías el código de bloqueo del mutex:
   `pthread_mutex_lock(&mutex);`

   Detrás de las líneas 11 (ó 12) y 24

9. Indica **detrás** de qué línea o líneas incluirías el código de desbloqueo del mutex:
   `pthread_mutex_unlock(&mutex);`

   Detrás de las líneas 18 y 31 (ó 32)

10. Escribe el código (pueden ser una o varias líneas) marcado por **CODIGO 5** correspondiente a la creación y terminación de los threads. Ten en cuenta que el programa debe esperar a la terminación de los dos hilos.

```
pthread_create(&th1, NULL, (void *)&Productor, NULL);
pthread_create(&th2, NULL, (void *)&Consumidor, NULL);
pthread_join(th1, NULL);
pthread_join(th2, NULL);
```

A continuación se incluye el código comentado de la PARTE A:

```
int main(void){
        struct elemento dato;
        int tuberia[2];
        int pid1, pid2;

        /* el proceso padre, que crea el pipe, será el proceso p1 */
        if (pipe(tuberia) < 0) {
                perror("No se puede crear la tuberia");
                exit(0);
        }

        switch ((pid1=fork()) { /* se crea el proceso p2 */
                case -1:
                        perror("Error al crear el proceso");
                        /* se cierra el pipe */
                        close(tuberia[0]);
                        close(tuberia[1]);
                        exit(0);
                case 0: /* proceso hijo, proceso P2 */
                        /* cierra el descriptor de lectura del pipe */
                        close(tuberia[0]);

                                /* en esta sección de código el proceso P2 */
                        /* escribiría en la tubería */
                                /* utilizando el descriptor tubería[1] */

                                break;
                        default:
                            /* el proceso padre crea ahora el proceso P3 */
                            switch((pid2 = fork()) {
                                case -1:
                                        perror("Error al crear el proceso ");
                                        close(tuberia[1]);
                                        close(tuberia[1]);

                                        /* se mata al proceso anterior */
                                        kill(pid1, SIGKILL);
                                        exit(0);
                                case 0:
                                        /* el proceso hijo, el proceso P3, */
                                        /* lee de la tubería */
                                        /* cierra el descriptor de escritura */
                                        close(tuberia[1]));

                                        /* en esta sección de código el proceso P3 */
                                        /* lee de la tubería */
                                        /* utilizando el descriptor tubería[0] */
                                        break;
                                default:
                                        /* el proceso padre, proceso P1, */
                                        /* escribe en la tubería */
                                        / * cierra el descriptor de lectura */
                                        close(tuberia[0]);

                                        /* en esta sección de código el proceso */
                                        /* padre escribe en la tubería */
                                        /* utilizando el descriptor tubería[1] */
                        }
        }
}
```

A continuación se incluye el código de la PARTE B:

```
#define MAX_BUFFER 1024
#define DATOS_A_PRODUCIR 100000

pthread_mutex_t mutex;
pthread_cond_t lleno;
pthread_cond_t vacio;
int n_elementos;
int buffer[MAX_BUFFER];

void Productor(void) {
 int dato, i ,pos = 0;
 for(i=0; i < DATOS_A_PRODUCIR; i++ ) {
  dato = i;
  printf ("Produce %d \n", dato);
  pthread_mutex_lock(&mutex);

  while (n_elementos == MAX_BUFFER){
   pthread_cond_wait(&lleno, &mutex); }

  buffer[pos] = i;
  pos = (pos + 1) % MAX_BUFFER;
  n_elementos ++;
  pthread_cond_signal(&vacio);
  pthread_mutex_unlock(&mutex);
 }
 pthread_exit(0);
}

void Consumidor(void){
 int dato, i ,pos = 0;
 for(i=0; i < DATOS_A_PRODUCIR; i++ ) {
  pthread_mutex_lock(&mutex);

  while (n_elementos == 0){
   pthread_cond_wait(&vacio, &mutex);
  }

  dato = buffer[pos];
  pos = (pos + 1) % MAX_BUFFER;
  n_elementos --;
  pthread_cond_signal(&lleno);
  pthread_mutex_unlock(&mutex);
  printf("Consume %d \n", dato);
 }
 pthread_exit(0);
}

int main(int argc, char *argv[]){
 pthread_t th1, th2;
 pthread_mutex_init(&mutex, NULL);
 pthread_cond_init(&lleno, NULL);
 pthread_cond_init(&vacio, NULL);
 pthread_create(&th1, NULL, (void *)&Productor, NULL);
 pthread_create(&th2, NULL, (void *)&Consumidor, NULL);
 pthread_join(th1, NULL);
 pthread_join(th2, NULL);
 pthread_mutex_destroy(&mutex);
 pthread_cond_destroy(&lleno);
 pthread_cond_destroy(&vacio);
 exit(0); }
```