Práctica Entregable
Algoritmo de *clustering K-means*
OpenMP

1. Algoritmo de clasificación *K-means*

El *clustering* es una técnica para dividir un conjunto de datos en grupos o *clusters* datos y descubrir patrones ocultos en ellos. Consiste en agrupar objetos de datos similares en *clusters* separados. Esta técnica es ampliamente utilizada en áreas como la inteligencia artificial, la biología, la compresión de datos o la minería de datos, entre otras.

El algoritmo de clasificación *K-means* es un método de agrupación no supervisado que se utiliza para clasificar un conjunto de datos en K *clusters* diferentes. Es decir, se conocen a priori el número de *clusters*.

Dada una nube de m puntos $P = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ donde cada punto tiene n dimensiones $\mathbf{x_i} = (x_{i1}, x_{i2}, \ldots, x_{in})$, el algoritmo *K-means* asigna cada punto de la nube a un determinado *cluster* $\{c_1, c_2, \ldots, c_K\}$, $K < n$. El algoritmo asigna cada punto al *cluster* cuyo centroide esté más próximo.

El algoritmo funciona de la siguiente manera:

1. Selecciona K centroides aleatorios $\mathbf{ce} = (ce_1, ce_2, \ldots, ce_K)$, un para cada *cluster*, de la nube de puntos P.

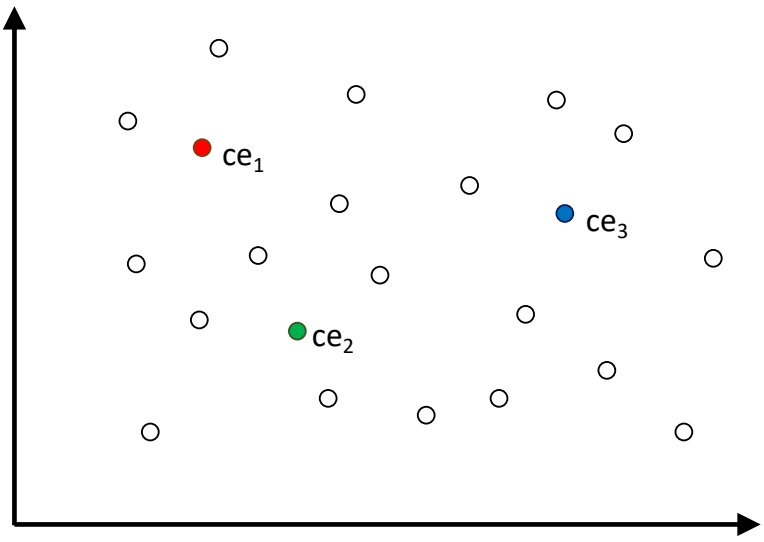

Figura 1: Selección aleatoria de centroides.

2. Asigna cada punto de datos al centroide más cercano (distancia euclídea más pequeña).

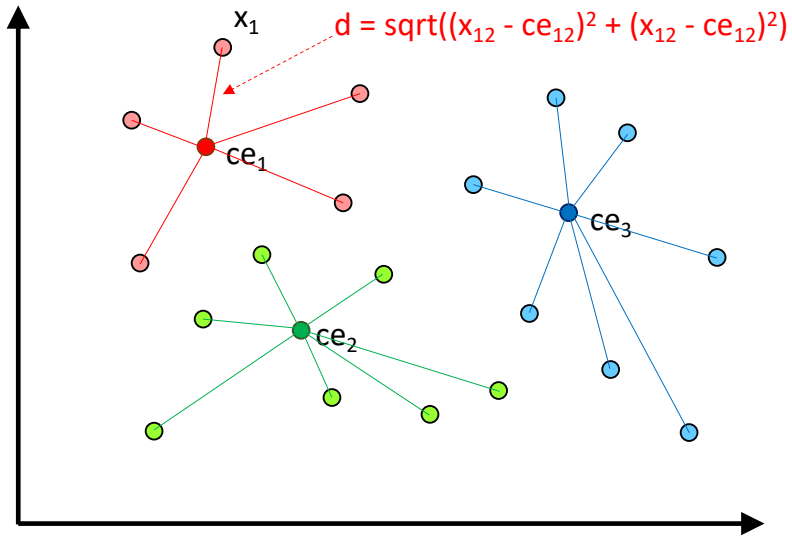

Figura 2: Asignación de puntos a los *clusters*.

3. Recalcula los centroides de cada *cluster* como la media de los puntos de datos asignados a ese *cluster*.

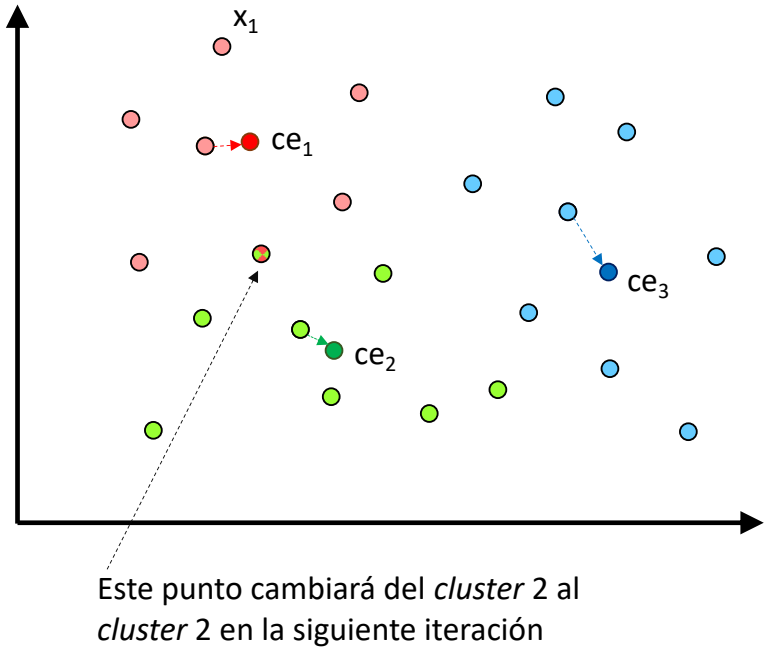

Figura 3: Cálculo de los nuevos centroides.

4. Repite los pasos 2 y 3 hasta que:

   a) o los centroides no cambien significativamente, es decir entre una iteración y la anterior el máximo movimiento de entre todos los centroides de cada *cluster* sea menor que una cota dada,

   b) o el número de cambios de puntos de un *cluster* a otro entre una iteración y la anterior sea menor que una cota dada,

c) o se alcance un número máximo de iteraciones.

1.1. Código secuencial

Se va a proporcional en el campus virtual de la asignatura un programa en C que implementa esta versión del algoritmo *K-means*. Este programa recibe por línea de comandos los siguiente argumentos:

1. `argv[1]`: Fichero de datos de entrada.
2. `argv[2]`: Número de *clusters*.
3. `argv[3]`: Número máximo de iteraciones.
4. `argv[4]`: Cota del número de cambios de puntos de un *cluster* a otro entre una iteración y la anterior.
5. `argv[5]`: Cota de la longitud del movimiento máximo entre centroides entre una iteración y la anterior.
6. `argv[6]`: Fichero de salida. *Cluster* asignado a cada linea del fichero.

Los ficheros de entrada tienen una línea por cada punto de la nube de puntos a clasificar y una columna separada por tabuladores para cada dimension de los puntos. En el campus virtual habrá ficheros de entrada de prueba de diferentes tamaños y dimensiones.

1.2. Entrega

Cada grupo de prácticas deberá entregar un una versión paralela del fichero fuente C proporcionado. Para ello, se pueden usar todas las estrategias OpenMP estudiadas en la asignatura. El objetivo de la práctica es minimizar el tiempo de cómputo lo máximo posible sin **alterar los resultados obtenidos en la versión secuencial**.

No es necesario paralelizar la lectura del fichero de entrada que puede dejarse en ejecución secuencial.

La entrega se hará en una actividad del campus virtual **antes del 10/04/2023 a las 23:59 h**. Los profesores pueden citar a los alumnos para que defiendan el código entregado.

2. Algunas funciones que se usan en el código

1. `strtok`: rompe la cadena en una serie de *tokens* utilizando un delimitador.

```
char *strtok(char *str, const char *delim);
```

Parámetros:

- `str`: El contenido de este string **se modifica** y se divide en cadenas más pequeñas (*tokens*).
- `delim`: Es el string que contiene el delimitador. Estos pueden variar de una llamada a otra.

Retorno:

- `char *`: Esta función devuelve un puntero al siguiente *token* encontrado en el string. Devuelve un puntero nulo `NULL` si no quedan *tokens* por recuperar.

Se suele usar en un bucle `while` que irá invocando a `strtok` en cada iteración hasta que la función devuelva `NULL` (no hay más `tokens`). ¡Recuerda! la cadena `str` es modificada por la función y se puede perder su contenido.

Ejemplo de uso:

```
#include <string.h>
#include <stdio.h>

int main () {
   char cadena[100] = "Esto es un token;esto otro;y esto otro mas";
   const char delim[2] = ";";
   char *token; //no requiere reserva de memoria

   //lectura del primer token
   token = strtok(cadena, delim);

   //Bucle de busqueda de tokens
   while( token != NULL ) {
      printf( "%s\n", token );
      token = strtok(NULL, delim);
   }
   printf("%s\n",cadena);

   return(0);
}
```

2. `mencpy`: copia `n` bytes del área de memoria apuntada por `src` al área de memoria apuntada por `dest`.

```
  void *memcpy(void *dest, const void *src, size_t n);
```

Parámetros:

- `dest`: Puntero al array de destino donde se copiará el contenido, convertido a un puntero de tipo `void*`.
- `src`: Puntero al array origen de los datos a copiar, convertido a un puntero de tipo `void*`.
- `n`: Es el número de bytes a copiar.

Ejemplo de uso:

```
#include <string.h>
#include <stdio.h>

int main () {
  int *vector, *vector2;

  vector = (int*) calloc(10,sizeof(int));
  vector2 = (int*) calloc(10,sizeof(int));
  if (vector==NULL || vector2==NULL) {
```

```
        printf("Error alojando memoria\n");
        exit(-1);
    }

    for(i=0; i<K; i++) {
      vector[i]=rand()%5;
    }

    //Copia los 10 enteros apuntados por vector a la
    //zona de memoria apuntada por vector2
    memcpy(vector2,vector,10*sizeof(int));
    return 0;
}
```

3. Uso de estructuras lineales como bidimesionales

Cuando se reserva memoria con `malloc` o `calloc` en C, aunque se trate de estructuras bidimensionales (o de orden superior), los datos se agrupan en la memoria de manera unidimensional o lineal y sólo se pude usar un índice.

Por ejemplo, el siguiente código C representa la reserva de una matriz de 3 filas y 4 columnas y su recorrido para asignar valores aleatorios.

```
int main()
{
    int fil=3, col=4, i;
    int *matriz = NULL;

    //Reserva de la matriz
    matriz = (int *)malloc(fil*col*sizeof(int));
    if (matriz == NULL) {
        fprintf(stderr,"Error alojando memoria\n");
        exit(-1);
    }

    //Solo se puede usar un indice para acceder a las posiciones de la matriz
    for (i=0; i<fil*col; i++) {
        matriz[i] = rand() % 10;
        printf("matriz[%d]: %d\n",i,matriz[i]);
    }
    free(matriz);
}
```

Los datos en la memoria se almacenan de manera lineal y mediante el índice se puede acceder a cada uno de ellos, como se muestra en la parte izquierda de la Figura 4.

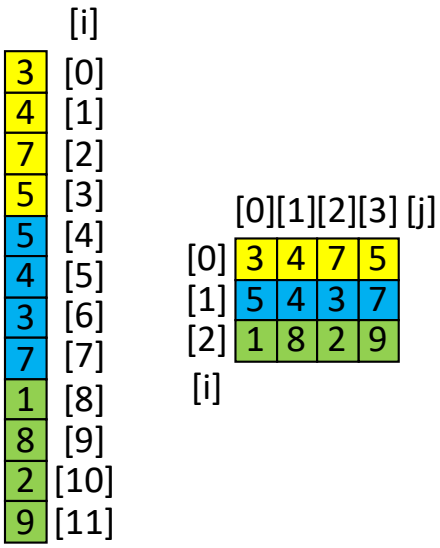

Figura 4: Representación de la matriz en memoria.

Si se quiere recorrer la matriz por filas y columnas con dos índices, uno para indexar la fila y otro la columna, y por lo tanto, con dos bucles anidados, se puede recurrir a alguna de las siguientes soluciones:

1. Uso de una expresión que operando con los índices de fila y columna obtenga el índice del recorrido lineal (depende del número de columnas):

```c
int main()
{
    int fil=3, col=4, i, j;
    int *matriz = NULL;
    //Reserva de la matriz
    matriz = (int *)malloc(fil*col*sizeof(int));
    if (matriz == NULL)     {
        fprintf(stderr,"Error alojando memoria\n");
        exit(-1);
    }
    for (i=0; i<fil; i++) { //recorrido por filas
        for (j=0; j<col; j++) { //recorrido por columnas
            matriz[i*col + j] = rand() % 10;
            printf("matriz[%d, %d]: %d\n",i,j,matriz[i*col + j]);
        }
    }
    free(matriz);
}
```

2. Uso de una macro de C que utilice la expresión anterior.

```c
//Macro para el acceso a matrices bidimensionales
#define accesMat(m,i,j,col) m[i*col+j]
int main()
{
    int fil=3, col=4, i, j;
    int *matriz = NULL;

    //Reserva de la matriz
```

```
    matriz = (int *)malloc(fil*col*sizeof(int));
    if (matriz == NULL)
    {
        fprintf(stderr,"Error alojando memoria\n");
        exit(-1);
    }

    for (i=0; i<fil; i++) { //recorrido por filas
        for (j=0; j<col; j++) { //recorrido por columnas
            accesMat(matriz,i,j,col) = rand() % 10;
            printf("matriz[%d, %d]: %d\n",i,j,accesMat(matriz,i,j,col));
        }
    }
}
```