Constructivo.

Índice.

1. Descripción del programa.

Considerar el siguiente problema de optimización, denominado problema de la máxima intersección de k conjuntos (k-MIC). Dada una colección de m conjuntos C={S 1 , ..., S m } sobre un conjunto finito de enteros ({1,...,n}), y un entero positivo k, el objetivo de este problema es seleccionar exactamente k conjuntos de C cuya intersección tiene el máximo cardinal. La práctica consiste en desarrollar un programa que realice las siguientes acciones:

1. Leer m conjuntos de un fichero de entrada, donde cada línea (terminada en '\n') contiene los enteros asociados a cada conjunto.
2. Resolver el problema k-MIC utilizando una técnica constructiva (método greedy) que consiste en los siguientes pasos:
   - Partir de una solución vacía, Sol.
   - Sea S' el conjunto con mayor cardinal de C.
   - Incluir S' en Sol y eliminarlo de C.
   - Mientras |Sol| ≠ k hacer:
     - Buscar el conjunto S" en C cuya intersección con todos los conjuntos en Sol tiene el mayor cardinal.
     - Incluir S" en Sol y eliminarlo de C.
   - Devolver Sol.
3. Guardar los conjuntos de la solución encontrada, Sol, en un fichero de salida (que tenga el mismo formato que el fichero de entrada).

NOTA: El ejecutable se deberá llamar constructivo y se invocará de la siguiente forma:
> constructivo k fichero_entrada fichero_salida

2. Interfaz de las clases.

Conjunto
Celda *l; Celda *ultimo; int elementos;

Conjunto();
Conjunto(TIPOBASE valor);
Conjunto(const Conjunto& otroconjunto);
~Conjunto();
bool Vacio() const;
int Tamanio() const;
void Destruir();
void CopiarConjunto(const Conjunto& otroconjunto);
bool Buscar(const TIPOBASE& valor) const;
void Inserta(const TIPOBASE& valor);
void BorrarElemento(const TIPOBASE& valor);
const Conjunto& operator=(const Conjunto& otroconjunto);
friend ostream& operator<<(ostream &os, const Conjunto &conjunto);
friend istream& operator>>(istream &is, Conjunto &conjunto);
friend bool operator==(const Conjunto &un_conjunto1, const Conjunto &un_conjunto2);
friend bool operator!=(const Conjunto &un_conjunto1, const Conjunto &un_conjunto2);
friend Conjunto operator+(const Conjunto &un_conjunto1, const Conjunto &un_conjunto2);
friend Conjunto operator-(const Conjunto &un_conjunto1, const Conjunto &un_conjunto2);
friend Conjunto operator*(const Conjunto &un_conjunto1, const Conjunto &un_conjunto2);

ConjuntoDeConjuntos
Conjunto *conjuntodeconjuntos; int elementos;

ConjuntoDeConjuntos();
ConjuntoDeConjuntos(const ConjuntoDeConjuntos &otro_cdc);
void DestruirCDC();
~ConjuntoDeConjuntos();
int Tamanio() const;
void Aniadir(const Conjunto &conjunto);
void BorrarConjunto(int posicion);
Conjunto Get(int i) const;
Conjunto Interseccion() const;
int Aleatorio(int i) const;
int ConjuntoAniadir(ConjuntoDeConjuntos &solucion) const;
void Editar(ConjuntoDeConjuntos &solucion);
ConjuntoDeConjuntos Solucion(int k);
const ConjuntoDeConjuntos& operator=(const ConjuntoDeConjuntos& otro_cdc);
friend ostream& operator<<(ostream &os, const ConjuntoDeConjuntos &otro_cdc);

3. Implementación.

Para poder resolver el problema de la máxima intersección de k conjuntos he decidido implementar dos clases, la clase Conjunto y la clase ConjuntoDeConjuntos.
El dato miembro de la clase Conjunto es un puntero a Celda (un struct), esto me permite crear un objeto Conjunto que sea una lista enlazada. Como me he basado en el ejercicio 8 de la relación 4 de problemas que indica que implementes la clase Conjunto, mis objetos Conjunto no van a tener elementos repetidos y van a estar siempre ordenador, tal y como indica el enunciado de dicho problema. Para controlar que esté siempre ordenado y que no haya elementos repetidos lo hago en el método Insertar de la clase Conjunto.
El dato miembro de la clase ConjuntoDeConjuntos es un puntero a un Conjunto, esto me permite crear un objeto ConjuntoDeConjuntos que sea un vector de objetos Conjunto.
En ambas clases tengo un dato miembro privado llamado “elementos” que sirve para contabilizar el número de elementos totales de cada objeto (ya sea el numero de elementos si es un objeto Conjunto o bien el numero de conjuntos si es un objeto ConjuntoDeConjuntos).

4. Métodos de la clase Conjunto.

• Conjunto();
Constructor por defecto. Crea una lista con 0 elementos.

• Conjunto(TIPOBASE valor);
Constructor por defecto. Crea una lista con 1 elemento, es decir, una celda. La información de dicha celda es “valor”.

• Conjunto(const Conjunto& otroconjunto);
Constructor de copia. Llama al método CopiarConjunto.

• ~Conjunto();
Destructor. Llama al método Destruir.

• bool Vacio() const;
Método sin argumentos que devuelve un bool, indicando true si el conjunto está vacío (0 elementos) y false en caso contrario. El método es constante ya que no modifica el objeto Conjunto sobre el que se aplica.

• int Tamanio() const;
Método sin argumentos que devuelve un entero indicando el numero de elementos del conjunto sobre el que se ha llamado a dicho método. El método es constante ya que no modifica el objeto Conjunto sobre el que se aplica.

• void Destruir();
Método que libera la memoria reservada por cada elemento del objeto Conjunto uno a uno para destruir el objeto.

• void CopiarConjunto(const Conjunto& otroconjunto);
Método que copia un otroconjunto en un conjunto. Método auxiliar para el constructor de copia y la sobrecarga del operador de asignación.

• bool Buscar(const TIPOBASE& valor) const;
Método sin argumentos que devuelve un bool, indicando true si el conjunto contiene a “valor” y false en caso contrario. El método es constante ya que no modifica el objeto Conjunto sobre el que se aplica.

• void Inserta(const TIPOBASE& valor);
Método para insertar “valor” en el conjunto. Sólo lo añade si no está ya. Si va a ser añadido, lo inserta en la posición correcta para que el conjunto permanezca siempre ordenado.

• void BorrarElemento(const TIPOBASE& valor);
Método que borra “valor” de un conjunto si dicho valor se encuentra en el conjunto. Si no está, no hace nada. Este método es usado por la sobrecarga del operador resta.

• const Conjunto& operator=(const Conjunto& otroconjunto);
Sobrecarga del operador de asignación. Copia otroconjunto (parte derecha de la asignación) en el objeto Conjunto de la izquierda de la asignación.

• friend ostream& operator<<(ostream &os, const Conjunto &conjunto);
Sobrecarga del operador de extracción. Muestra todos los elementos del conjunto.

• friend istream& operator>>(istream &is, Conjunto &conjunto);
Sobrecarga del operador de inserción. Lee elementos y los inserta (llamando al método insertar) en el conjunto hasta llegar al terminador.

• friend bool operator==(const Conjunto &un_conjunto1, const Conjunto &un_conjunto2);
Sobrecarga del operador de igualdad. Comprueba si dos conjuntos son iguales, es decir, tienen el mismo número de elementos y los elementos son iguales.

• friend bool operator!=(const Conjunto &un_conjunto1, const Conjunto &un_conjunto2);
Sobrecarga del operador distinto de. Comprueba si dos conjuntos son distintos, es decir, no tienen el mismo número de elementos o los elementos son distintos.

• friend Conjunto operator+(const Conjunto &un_conjunto1, const Conjunto &un_conjunto2);
Sobrecarga del operador suma. Se comporta como la unión de conjuntos. Calcula la unión de un_conjunto1 y un_conjunto2 y devuelve el resultado. Si un_conjunto1 o un_conjunto2 no son un objeto Conjunto, por ejemplo un entero, se llama al constructor con un parámetro, el cual crea un objeto Conjunto con dicho entero y se hace la unión de este nuevo conjunto con el que ya teníamos. Funciona cuando el entero va a la izquierda del operador + y cuando va a la derecha.

• friend Conjunto operator-(const Conjunto &un_conjunto1, const Conjunto &un_conjunto2);
Sobrecarga del operador resta. Los elementos que están en un_conjunto2 son eliminados de un_conjunto1 si se encuentran en el. Si un_conjunto1 o un_conjunto2 no son un objeto Conjunto, por ejemplo un entero, se llama al constructor con un parámetro, el cual crea un objeto Conjunto con dicho entero y se hace la resta de este nuevo conjunto con el que ya teníamos.

• friend Conjunto operator*(const Conjunto &un_conjunto1, const Conjunto &un_conjunto2);
Sobrecarga del operador multiplicación. Se comporta como la intersección de conjuntos. Calcula la intersección de un_conjunto1 y un_conjunto2 y devuelve el resultado. Si un_conjunto1 o un_conjunto2 no son un objeto Conjunto, por ejemplo un entero, se llama al constructor con un parámetro, el cual crea un objeto Conjunto con dicho entero y se hace la intersección de este nuevo conjunto con el que ya teníamos. Funciona cuando el entero va a la izquierda del operador * y cuando va a la derecha.

5. Métodos de la clase ConjuntoDeConjuntos.

• ConjuntoDeConjuntos();
Constructor por defecto. Crea un objeto ConjuntoDeConjuntos con 0 elementos, es decir, 0 conjuntos.

• ConjuntoDeConjuntos(const ConjuntoDeConjuntos &otro_cdc);
Constructor de copia.

• void DestruirCDC();
Destructor. Destruye el objeto ConuntoDeConjuntos recorriendo el vector y destruyendo cada objeto Conjunto (llamando al método Destruir) uno a uno. Luego libera la memoria reservada para crear el vector de objetos Conjunto.

• ~ConjuntoDeConjuntos();
Destructor. Llama al método DestruirCDC.

• int Tamanio() const;
Método que devuelve el número de elementos del objeto ConjuntoDeConjuntos sobre el que se aplica el método. El método es constante ya que no modifica el objeto ConjuntoDeConjuntos sobre el que se llama dicho método.

• void Aniadir(const Conjunto &conjunto);
Método que añade al objeto ConjuntoDeConjuntos el conjunto que se le pasa como argumento.

• void BorrarConjunto(int posicion);
Método que borra un objeto Conjunto de nuestro ConjuntoDeConjuntos dada su posición.

• Conjunto Get(int i) const;
Método que devuelve el conjunto que ocupa la posición i en el objeto ConjuntoDeConjuntos. El método es constante ya que no modifica el objeto ConjuntoDeConjuntos sobre el que se llama dicho método.

• Conjunto Interseccion();
Método que devuelve la intersección de todos los conjuntos que conforman un objeto ConjuntoDeConjuntos.

• int Aleatorio(int i) const;
Método que calcula un número aleatorio entre 0 e i-1. El método es constante ya que no modifica el objeto ConjuntoDeConjuntos sobre el que se llama dicho método.

• int ConjuntoAniadir(ConjuntoDeConjuntos &solucion) const;
Método que calcula el siguiente conjunto que hay que añadir a la solución y borrar del conjunto de conjuntos original. Tiene como argumento el objeto "solucion" para poder hacer la intersección de la solución y así calcular el conjunto correcto adecuadamente en función del cardinal de la intersección entre los conjuntos de solución y el conjunto que el método está buscando. El método es constante ya que no modifica el objeto ConjuntoDeConjuntos sobre el que se llama dicho método.

• void Editar(ConjuntoDeConjuntos &solucion);
Método que, una vez calculado el siguiente conjunto a añadir con el método ConjuntoAniadir, elimina dicho conjunto del conjunto de conjuntos original y lo añade a la solución.

• ConjuntoDeConjuntos Solucion(int k);
Método que resuelve el problema planteado. Devuelve un objeto ConjuntoDeConjuntos con el número de conjuntos deseados, k, que se le pasa como argumento a dicho método.

• const ConjuntoDeConjuntos& operator=(const ConjuntoDeConjuntos& otro_cdc);
Sobrecarga del operador de asignación. Copia otro_cdc) (parte derecha de la asignación) en el objeto ConjuntoDeConjuntos de la izquierda de la asignación.

• friend ostream& operator<<(ostream &os, const ConjuntoDeConjuntos &otro_cdc);
Sobrecarga del operador de extracción. Muestra los conjuntos del objeto ConjuntoDeConjuntos.

6. Método que resuelve el problema, Solución.

```
1   ConjuntoDeConjuntos Solucion(int k){
2         ConjuntoDeConjuntos solucion;
3     int cardinal_c=(*this).Tamanio();
4     //Si no queremos ningun conjunto (k=0), aniadir un conjunto vacio
5     if(k==0){
6       Conjunto auxiliar;
7       auxiliar.Inserta(0);
8       solucion.Aniadir(auxiliar);
9     }
10    //Si k!=0, aniadir el conjunto de mayor cardinal
11    else{
12      Editar(solucion);
```

```
13        //Si k>1, aniadir el resto de conjuntos conforme a los requisitos del problema
14        if(k>1)
15          //Mientras no se han aniadido suficientes (solucion.Tamanio()<k) y mientas
16          //no hemos aniadido todos los conjuntos de c (solucion.Tamanio()<cardinal_c)
17          while(solucion.Tamanio()<k && solucion.Tamanio()<cardinal_c)
18            Editar(solucion);
19      }
20      return solucion;
21  }
```

El método Solución es un método de la clase ConjuntoDeConjuntos. Tiene un único parámetro, k, que indica el número de conjuntos que el usuario quiere calcular. El método devuelve un objeto de la clase ConjuntoDeConjuntos, el objeto solución, con los conjuntos que mi método ha calculado.
Lo primero que hago para implementar mi método Solución es ver si k es mayor que cero (línea 5) para ver si tengo que realizar todos los cálculos y procesos siguientes. Si k=0, devuelvo el conjunto vacío. Si k es mayor que cero, pero k=1 (línea 11), sólo tengo que añadir el de mayor cardinal. Si k>1, resuelvo el problema.
En la línea 12 llamo al método Editar. Este método llama al método ConjuntoAniadir que me calcula el siguiente conjunto que tengo que añadir y devuelve su posición. Una vez que ConjuntoAniadir ha devuelto la posición del conjunto a añadir, el método Editar añade dicho conjunto a la solución (que por eso paso el objeto solucion de la clase ConjuntoDeConjuntos por referencia) y borrar dicho conjunto del conjunto original, c. De esta manera ya tengo añadido el primer conjunto, el de mayor cardinal.
Una vez hecho esto, si k>1, sigo añadiendo conjuntos a mi solución mientras que el cardinal de la solución sea menor que k o ya haya añadido todos los conjuntos existentes en c (línea 17).
Cuando ya estén todos los conjuntos añadidos (bien por haber llegado al número de conjuntos pedidos, k, o por haber añadido todos los conjuntos de c) devuelvo la solución.

7. Solución.

Fich. Prueba	k=2	k=3	k=4	k=5
instancia1.txt	8	5	3	3-2, depende del aleatorio
instancia2.txt	9-10, depende del aleatorio	5-6, depende del aleatorio	3-4, depende del aleatorio	3-2, depende del aleatorio
instancia3.txt	12	7-8, depende del aleatorio	5-6, depende del aleatorio	3-4, depende del aleatorio