Práctica Final MP k-MIC

Índice:

Índice:

1. Interface de las clases

1.1 Clase VectorDinamico

1.2 Clase Conjunto

1.3 Clase SetConjuntos

2. Resultados obtenidos

2.1 Tabla

2.2 Valgrind

Antonio de la Vega Jiménez 1ºA

1. Interface de las clases

En este apartado se verá cuál es la funcionalidad de cada clase, además se explicará para qué sirve cada uno de los atributos de la clase.

He implementado el algoritmo k-MIC en el mismo archivo que el main, ya que he pensado que este algoritmo no debía pertenecer a ninguna de las clases implementadas, ya que aunque hace uso de la clase SetConjuntos, no es una operación propia de los conjuntos.

1.1 Clase VectorDinamico

Esta clase sirve como base a la clase Conjunto, su funcionalidad es almacenar un vector de tipo TBase. El vector almacenado es un vector dinámico, que mediante del método resize implementado en la clase, puede variar su tamaño, además de este método, contiene otros, que nos servirán de ayuda en la clase Conjunto.

```
class VectorDinamico {
private:
	int util, tam;
	TBase *vect;
/*.........*/
}
```

Los atributos de la clase son:

util -> Indica cuántos elementos hay en el vector.
tam -> Indica el tamaño del vector
vect -> Es el vector donde se guardan los elementos.

1.2 Clase Conjunto

Esta clase está basada en la clase VectorDinamico, su objetivo es representar un conjunto de elementos de tipo TBase. Esta clase es muy similar a un vector, pero tiene algunas propiedades que la convierte en un conjunto, como por ejemplo no poder almacenar elementos repetidos. Además proporciona las funciones necesarias para hacer operaciones con conjuntos, como la intersección, la unión o la resta.

```
class Conjunto {
private:
	VectorDinamico datos;
/*.........*/
}
```

La parte privada de esta clase es muy simple, ya que hace uso de un objeto de la clase VectorDinamico:

datos -> Es un vector dinámico en el que se guardan los elementos del conjunto.

1.3 Clase SetConjuntos

Esta clase es la usada para almacenar los ficheros , obtener la solución y guardar los datos de la solución. SetConjuntos es muy similar a la clase VectorDinamico, representa varios conjuntos unidos en un mismo objeto, además incorpora algunas operaciones sobre los conjuntos almacenados. Para implementar esta clase se ha hecho uso de la clase Conjunto.

```
class SetConjuntos{
private:
    Conjunto *conjuntos;
    int util, tam;
/*..........*/
}
```

Los atributos de la clase son:

util -> Indica cuantos conjuntos hay almacenados.
tam -> Indica el tamaño del vector en el que se almacenan los conjuntos.
conjuntos -> Es un vector de conjuntos.

2. Resultados obtenidos

2.1 Tabla

Fich. Prueba	k=2	k=3	k=4	k=5
instancia1.txt	8	5	3	3
instancia2.txt	9	5	3	3
instancia3.txt	12	7	5	3

Nota sobre los resultados:

Aunque en los archivos proporcionados hay elementos repetidos, estos no se han tenido en cuenta. Esto se debe a que en un conjunto no puede haber elementos repetidos, por ello al insertar los elementos desde los ficheros, los repetidos han sido eliminados (la función insertar() de la clase Conjunto, no permite guardar elementos repetidos).

2.2 Valgrind

Para comprobar que el programa no tiene fallos de memoria, he usado la herramienta valgrind, obteniendo el siguiente resultado:

```
$ valgrind --leak-check=full --track-origins=yes ./bin/constructivo 3
datos/instancia3.txt salida.txt

==4873== Memcheck, a memory error detector
==4873== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==4873== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==4873== Command: ./bin/constructivo 3 datos/instancia3.txt salida.txt
==4873==
Número total de conjuntos.............50
Número conjuntos en la solución......3
Cardinal de la solución..............7
==4873==


==4873== HEAP SUMMARY:
==4873==     in use at exit: 0 bytes in 0 blocks
==4873==     total heap usage: 1,036 allocs, 1,036 frees, 80,955 bytes allocated
==4873==
==4873== All heap blocks were freed -- no leaks are possible
==4873==
==4873== For counts of detected and suppressed errors, rerun with: -v
==4873== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Viendo la salida de valgrind, se puede observar que toda la memoria queda liberada, por lo tanto, los destructores y todas la funciones que usan memoria dinámica están funcionando correctamente.