UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Práctica final: Cifras y letras

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Estructuras de Datos

Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

1. Introducción

La idea de esta práctica es resolver un problema eligiendo la mejor estructura de datos para las operaciones que se solicitan.

Los requisitos para poder realizar esta práctica son los siguientes:

1. Haber estudiado el Tema 1: Introducción a la eficiencia de los algoritmos.
2. Haber estudiado el Tema 2: Abstracción de datos, templates.
3. Haber estudiado el Tema 3: TDAlLineales.
4. Haber estudiado el Tema 4: STL e iteradores.

2. Objetivo.

El objetivo de esta práctica es llevar a cabo el análisis, diseño e implementación de un proyecto. Con tal fin el estudiante abordará un problema donde se requiere estructuras de datos que permiten almacenar grandes volúmenes de datos y poder acceder a ellos de la forma más eficiente.

2.1. Tareas a realizar

El estudiante deberá implementar varios programas simulando el juego “Cifras y letras”. Este juego tiene dos partes independientes:

- La primera se refiere a la simulación del juego de las cifras. Se trata de, a partir de un conjunto de cifras específicas, y usando operaciones elementales, conseguir un número de 3 cifras concreto.
- La segunda se refiere a la simulación del juego de las letras. Consiste en, dada una serie de letras escogidas de forma aleatoria, obtener la palabra existente en un diccionario formada a partir de ellas de mayor longitud o que tenga la mayor puntuación.

Se deberán llevar a cabo las siguientes tareas:

1. Definir los TDA que se consideren necesarios para la solución del problema propuesto.
2. Probar los módulos con programas de test.

Se puede usar la STL en todos los módulos.

A continuación se detallan ambas partes y los programas que se deberán desarrollar.

2.1.1. El problema de las cifras

Esta parte del juego trata de, dado un conjunto de 6 enteros sacados aleatoriamente del conjunto:

$$C = \{1,2,3,4,5,6,7,8,9, 10, 25, 50, 75, 100\}$$

conseguir otro entero aleatorio de 3 cifras usando para ello solo las operaciones de suma, resta, producto y división entera, y teniendo en cuenta que solo se puede usar cada número (de los 6) como mucho una vez, aunque es posible que no todos se usen para conseguir el número de 3 cifras.

Ejemplo:

Se sacan aleatoriamente del conjunto C los números:

6, 8, 10, 9, 4, 75

y se pide que con ellos se consiga el número (también aleatoriamente generado):

835

Una posible solución (que no tiene por qué ser única) es la siguiente:

- 8/4 = 2
- 9+2 = 11
- 75*11= 825
- 825+10 = 835

En este caso, se han usado solo 5 de los 6 números y sin usar ninguno más de una vez (el dígito 6 no ha hecho falta en esta solución) y solo operaciones de +, * y / (la resta en este caso tampoco ha hecho falta) para conseguir llegar al número exacto

No pueden tenerse resultados temporales negativos, es decir, pasos intermedios como 4-8=-4 y usar ese -4 no están permitidos, como tampoco está permitido hacer una división no exacta, es decir, no puede hacerse 75/11 y redondear.

Obviamente puede que sea imposible conseguir el número aleatorio de 3 cifras con 6 números aleatorios. En ese caso hay dos opciones:

1. La más simple: que el algoritmo diga que no hay solución.
2. La más interesante: que la salida del algoritmo sea conseguir el número más próximo posible al que nos piden.

Hay que tener en cuenta que en la generación aleatoria podría haber repeticiones y salir p.ej. estos 6 números:

{1, 3, 5, 3, 100, 5}

donde el 3 y el 5 aparecen 2 veces. No pasa nada, esto es válido, simplemente que para alcanzar la solución se cuenta con dos 3 y dos 5, nada más, pero sigue estando presente la restricción de la no repetición, es decir que se cuenta con 6 números, un 1, dos 3, dos 5 y un 100, es decir se puede usar cada número de la serie SOLO UNA VEZ (como mucho, una vez el 1, una vez el primer 3, una vez el primer 5, una vez el segundo 3, una vez el 100 y una vez el segundo 5).

Como curiosidad, indicar que existen combinaciones de 6 números de C con las que puede obtenerse cualquier número de 3 cifras (y sería interesante pensar en cómo podrían conseguirse tales combinaciones). P.ej. a partir del conjunto C={2,6,7,9,50,75} se puede conseguir cualquier número de 3 cifras). Se denominan *combinaciones mágicas*.

2.1.2. El problema de las letras

Esta parte del juego consiste en, dada una serie de letras escogidas de forma aleatoria, obtener la palabra existente de mayor longitud de la lista de palabras formada a partir de ellas.

Ejemplo: dadas las siguientes letras:

O D Y R M E T

una de las mejores soluciones sería METRO.

El tamaño del conjunto de letras de partida lo decide el usuario y dichas letras pueden repetirse. En la sección siguiente se analiza con más detalle, incluyendo ejemplos.

3. TDA y programas

Como se ha comentado antes, las dos partes son independientes. En esta sección se explican cómo llevar a cabo la implementación de cada una de ellas.

3.1. Cifras

Basándose en estructuras lineales y en contenedores `set`, resolver el juego de las cifras, incluyendo la búsqueda de combinaciones mágicas. Se han de definir adecuadamente los TDA utilizados.

3.2. Letras

3.2.1. Test diccionario

Este programa permitirá comprobar el buen funcionamiento del `TDA Diccionario` (ver sección 3.2.6), representado como un `set`. Para ello, el código fuente `testdiccionario.cpp` (mostrado a continuación) deberá funcionar con el `TDA Diccionario` que se desarrolle.

```
1.  #include <fstream>
2.  #include <iostream>
3.  #include <string>
4.  #include <vector>
5.  #include <set>
6.  int main(int argc, char * argv[]){
7.   if (argc!=2){
8.            cout<<"Los parametros son:"<<endl;
9.            cout<<"1.- El fichero con las palabras";
10.           return 0;
11.  }
12.  ifstream f(argv[1]);
13.  if (!f){
14.           cout<<"No puedo abrir el fichero "<<argv[1]<<endl;
15.   return 0;
16.  }
17.  Diccionario D;
18.  cout<<"Cargando diccionario...."<<endl;
19.  f>>D;
20.  cout<<"Leido el diccionario..."<<endl;
21.  cout<<D;
22.
23.  int longitud;
24.   cout<<"Dime la longitud de las palabras que quieres ver";
25.  cin>>longitud;
26.  vector<string> v=D.PalabrasLongitud(longitud);
27.
28.  cout<<"Palabras de Longitud "<<longitud<<endl;
29.  for (unsigned int i=0;i<v.size();i++)
30.   cout<<v[i]<<endl;
31.
32.  string p;
33.  cout<<"Dime una palabra: ";
34.  cin>>p;
35.  if (D.Esta(p))  cout<<"Sí esa palabra existe";
36.  else   cout<<"Esa palabra no existe";
37. }
```

En este código, se carga el diccionario en memoria y luego se imprime por la salida estándar. A continuación se muestran todas las palabras del diccionario de una longitud dada. Y finalmente, dada una palabra por el usuario, el programa indica si existe tal palabra en el diccionario o no.

El estudiante creará, por tanto, el programa `testdiccionario` que se deberá ejecutar en la línea de órdenes de la siguiente manera:

```
./bin/testdiccionario diccionario.txt
```

El único parámetro que se da al programa es el nombre del fichero donde se almacena el diccionario.

3.2.2. Letras

Este programa construye las palabras de longitud mayor (o de puntuación mayor) de un diccionario a partir de una serie de letras seleccionadas de forma aleatoria. Los parámetros de entrada son los siguientes:

1. El nombre del fichero con el diccionario.
2. El nombre del fichero con las letras.
3. El número de letras que se deben generar de forma aleatoria.
4. Modalidad de juego:
   - Longitud: Si el parámetro es `L` se buscará la palabra más larga.
   - Puntuación: Si el parámetro es `P` se obtendrá la palabra de mayor puntuación.

El programa `letras` se deberá ejecutar en la línea de órdenes de la siguiente manera:

```
./letras diccionario.txt letras.txt 8 L
```

Tras la ejecución de la orden anterior, aparecerá en pantalla lo siguiente:

```
Las letras son: T       I       E       O       I       T       U       S
Dime tu solucion:tieso
tieso   Puntuacion: 5

Mis soluciones son:
otitis  Puntuacion: 6
tiesto  Puntuacion: 6
Mejor Solucion:tiesto
¿Quieres seguir jugando[S/N]?N
```

En primer lugar el programa genera 8 letras. Estas letras se escogen, de forma aleatoria, entre las dadas en el fichero `letras.txt` (ver sección 3.1.5). Una vez generadas las letras, el programa pide al usuario su solución; en el ejemplo anterior, la solución dada es “tieso”. A continuación, se muestra la solución del usuario junto con su puntuación. Y finalmente se muestran las soluciones dadas por el programa.

Para generar de forma aleatoria las letras con la que construir la palabra, el estudiante creará una *Bolsa de letras* (contenedor de letras que se disponen de forma aleatoria) en la que el número de veces que aparece cada letra en dicha bolsa, viene dado por la columna `Cantidad` del fichero `letras.txt`.

En el caso de que el usuario haya escogido jugar en modo *Puntuación*, como resultado se obtendrán las palabras que acumulen una mayor puntuación. Para obtener la puntuación de la palabra simplemente tenemos que sumar las puntuaciones de las letras en la palabra (en el fichero `letras.txt` la puntuación de cada letra viene dada en la columna `Puntos`). En este otro caso, un ejemplo de salida sería el siguiente:

```
******Puntuaciones Letras*******       Z       10
A       1
B       3                             Las letras son:
C       3                             N       S       A       O       T       O       A       I
D       2                             Dime tu solucion:sonata
E       1
F       4
G       2                             sonata  Puntuacion: 6
H       4                             Mis Soluciones son:
I       1                             asiano  Puntuacion: 6
J       8                             atonia  Puntuacion: 6
L       1                             ostion  Puntuacion: 6
M       3                             sonata  Puntuacion: 6
N       1                             sotana  Puntuacion: 6
O       1                             sotani  Puntuacion: 6
P       3                             sotano  Puntuacion: 6
Q       5                             tisana  Puntuacion: 6
R       1                             toison  Puntuacion: 6
S       1                             Mejor Solucion:toison
T       1                             ¿Quieres seguir jugando[S/N]?N
U       1
V       4
Y       4
```

En ambas versiones, se le preguntará al usuario si quiere seguir jugando. En caso afirmativo se generará una nueva secuencia de letras aleatorias para jugar de nuevo. En otro caso el programa termina.

En las secciones 3.2.4 y 3.2.5 se detallan los formatos de los ficheros de entrada al programa.

3.2.3. Cantidad de letras

El programa `cantidad_letras` obtiene la cantidad de instancias de cada letra (ver fichero `letras.txt` en la sección 3.2.5). Los parámetros de entrada son los siguientes:

1. El nombre del fichero con el diccionario.
2. El nombre del fichero con las letras.
3. El fichero donde escribir el conjunto de letras con la cantidad de apariciones calculada.

El programa se deberá ejecutar en la línea de órdenes de la siguiente manera:

```
./bien/cantidad_letras diccionario.txt letras.txt salida.txt
```

Este programa, una vez haya cargado en memoria el fichero del diccionario y el conjunto de letras, obtiene para cada letra en el conjunto el número de veces que aparece en el diccionario, es decir encuentra la frecuencia de aparición. Finalmente se obtiene el tanto por ciento, sobre el total de las

frecuencias, del número de veces que aparece cada letra y se asigna a dicha letra una puntuación de 1 a 10 de acuerdo a su frecuencia de aparición, de forma que cuanto menos aparezca, mayor puntuación tendrá. Este valor será la columna `Puntos` en el fichero de salida.

Ejemplo: tras la ejecución de la orden anterior, el fichero `salida.txt` obtenido es:

```
#Letra Cantidad Puntos
A       16      1
B       2       3
C       6       3
D       5       2
E       10      1
F       2       4
G       2       2
H       1       4
I       9       1
J       1       8
L       5       1
M       4       3
N       7       1
O       10      1
P       3       3
Q       1       5
R       10      1
S       5       1
T       6       1
U       4       1
V       2       4
X       1       8
Y       1       4
Z       1       10
```

3.2.4. Fichero diccionario

El fichero diccionario se compone del conjunto de palabras que se consideren como válidas, cada una en una línea. Un ejemplo de fichero diccionario es el siguiente:

```
a
aaronica
aaronico
ab
abab
ababillarse
ababol
abaca
abacera
abaceria
abacero
abacial
abaco
abad
abada
abadejo
abadenga
abadengo
```

3.2.5. Fichero letras

Un ejemplo de fichero de letras es el que se muestra a la derecha.

```
#Letra Cantidad Puntos
A       12      1
E       12      1
O       9       1
I       6       1
S       6       1
N       5       1
L       1       1
R       6       1
U       5       1
T       4       1
D       5       2
G       2       2
C       5       3
B       2       3
M       2       3
P       2       3
H       2       4
F       1       4
V       1       4
Y       1       4
Q       1       5
J       1       8
X       1       8
Z       1       10
```

El formato del fichero es el siguiente:

1. En primer lugar aparece una línea encabezada con el carácter # donde se describen las columnas del fichero que son: `Letra`, `Cantidad` y `Puntos`.
2. A continuación, cada línea se corresponde con la información de una letra, es decir:
    - Valor de la letra.
    - Número de veces que aparece la letra en la bolsa de letras.
    - Puntos asignados a la letra.

3.2.6. Módulos a desarrollar

Módulo Diccionario

Será necesario construir el `TDA Diccionario`. Un objeto del `TDA Diccionario` almacena palabras de un lenguaje y será representado como un `set` instanciado a `string`. Así, en líneas generales el módulo `Diccionario` se detalla a continuación:

```
#ifndef __Diccionario_h__
#define __Diccionario_h__
#include <set>

class Diccionario{
 private:
   set<string> datos;
 public:
      /**
        @brief Construye un diccionario vacío.
      **/

      Diccionario()
      /**
        @brief Devuelve el numero de palabras en el diccionario
      **/
      int size() const ;

      /**
        @brief Obtiene todas las palabras en el diccionario de un longitud dada
        @param longitud: la longitud de las palabras de salida
        @return un vector con las palabras de  longitud especifica en el parametro de entrada
      **/
      vector<string> PalabrasLongitud(int longitud);

       /**
        @brief Indica si una palabra está en el diccionario o no
        @param palabra: la palabra que se quiere buscar
        @return true si la palabra esta en el diccionario. False en caso contrario
      **/
      bool Esta(string palabra);
       /**
        @brief Lee de un flujo de entrada un diccionario
        @param is:flujo de entrada
        @param D: el objeto donde se realiza la lectura.
        @return el flujo de entrada
      **/
      friend   istream & operator>>(istream & is,Diccionario  &D);
       /**
        @brief Escribe en un flujo de salida un diccionario
        @param os:flujo de salida
        @param D: el objeto diccionario que se escribe
        @return el flujo de salida
      **/
      friend ostream & operator<<(ostream & os, const Diccionario  &D);
};
```

```
#endif
```

Podríamos generar un iterador sobre el `TDA Diccionario` que nos permita recorrer de manera ordenada todas las palabras del diccionario. Una posible especificación y representación de este iterador es la siguiente:

```
#ifndef __Diccionario_h__
#define __Diccionario_h__
#include <set>

class Diccionario{
 private:
   set<string> datos;
 public:
  …

 class iterator{
    private:
       set<string>::iterator it;
    public:
       iterator ();
       string operator *();
       iterator & operator ++();
       bool operator ==(const iterator &i)
       bool operator !=(const iterator &i)
       friend class Diccionario;
 };
 iterator begin();

 iterator end();


};

#endif
```

Módulos Letra y Conjunto de Letras

El `TDA Letra` almacena una letra que se especifica con tres valores:

1. El carácter de la propia letra.
2. La cantidad de veces que puede aparecer.
3. La puntuación de una letra.

El `TDA Conjunto_Letras` permitirá tener en memoria un fichero de letras. Este TDA se define como una colección de letras, en las que no hay letras repetidas.

Módulo Bolsa de Letras

Este módulo será útil para el programa `letras`. El `TDA Bolsa_letras` almacena caracteres correspondientes a una letra de un conjunto de letras. Este carácter aparece en la `Bolsa_letras`

repetido tantas veces como diga el campo cantidad de la letra. Por lo tanto, en la bolsa de letras aparecen las letras de forma aleatoria. En el programa `letras` la secuencia de letras con las que se juega se cogen de la bolsa de letras.

4. Práctica a entregar

Empaquetar todos los archivos relacionados en el proyecto en un archivo comprimido y entregarlo antes de la fecha que se publicará en la página web de la asignatura. No se incluirán ficheros objeto, ni ejecutables, ni la carpeta datos. Es recomendable hacer una limpieza para eliminar los archivos temporales o que se puedan generar a partir de los fuentes.

Se deben incluir los archivos **CMakeLists.txt y Doxyfile.in** para realizar la compilación y preparar la documentación. La estructura de directorios es la siguiente:

/letras

/data

/doc

/include

/src

La práctica se debe realizar obligatoriamente (salvo excepciones justificadas) por parejas. Ambos miembros de la pareja deberán subir los ficheros a prado a través del enlace correspondiente a la entrega de la práctica final. Se deberá rellenar el fichero mainpage.dox (incluido en la carpeta doc) explicando exactamente qué se ha hecho y cómo se ha hecho, y si hay algo que no funciona.

Se **penalizarán** aquellas prácticas que no mantengan los 4 directorios y los dos archivos de compilación o no realicen y documenten el mainpage.dox.

5. Valoración

En esta práctica se podrá alcanzar una puntuación máxima de 1 punto, que dependerá no solo del material entregado sino también de la defensa de la misma que se hará ante el profesor. Se podrán entregar de manera individual las CIFRAS y se obtendrá una nota máxima de 0.5 puntos, o las LETRAS y se obtendrá una nota máxima de 0.75.

Para el desarrollo y realización de la práctica se podrán usar Inteligencias Artificiales siempre y cuando se comente que se han usado y SOLO se podrá obtener un 0.25 de nota máxima. Si alguien no avisa del uso de IA’s y se demuestra su uso o no consigue defender la práctica, supondrá un 0 en la práctica con las correspondientes sanciones.

6. Referencias

- Rosa Rodriguez-Sánchez, J. Fdez-Valdivia, J.A. García, Estructuras de Datos en C++. Un enfoque práctico, 2020
- Garrido, A., Fdez-Valdivia, J. “Abstracción y estructuras de datos en C++”. Delta publicaciones, 2006.
- Garrido, A. Estructuras de datos avanzadas: con soluciones en C++. Editorial Universidad de Granada. 2025