UNIVERSIDAD DE GRANADA

ESTRUCTURA DE DATOS PROYECTO FINAL DE CIFRAS Y LETRAS

Descripción del proyecto

Se ha realizado la creación del juego de cifras y letras en código C++, el seguimiento del proyecto ha sido con un repositorio de git para facilitar la implementación del código sin que haya ningún problema de solapamiento del código.

El juego de las cifras recibe un conjunto de números de 6 elementos aleatorios seleccionados de una lista y un número aleatorio de 3 cifras, el objetivo es obtener dicho número realizando operaciones elementales con el conjunto de números. En el caso de que no encuentre solución para dicho número devuelve la solución del número más cercano al introducido.

Por el lado del juego de las letras, se obtiene un conjunto de ocho letras a partir de un fichero que contiene las letras disponibles. Dicho fichero presenta la siguiente estructura: letra, número de ocurrencias de la letra y puntuación asociada a la letra. Se crea un juego que muestra un conjunto de letras, el objetivo es formar, en función del juego seleccionado, la palabra con mayor puntuación posible. El juego se caracteriza por tener dos modalidades: una modalidad más clásica, cuyo objetivo es formar la palabra más larga y otra modalidad más alternativa cuyo objetivo es formar la palabra cuya puntuación sea mayor, por ejemplo, le asignamos una puntuación a una letra en función del número de palabras que se pueden formar con esa letra y el número de ocurrencias que tiene en el fichero letras.txt. Una vez introducida la palabra el juego te genera, si las hay, las palabras cuya puntuación sea mayor a la proporcionada.

Desarrollo del proyecto

Juego de las Cifras

Para el desarrollo del juego de las cifras hemos decidido crear una clase auxiliar llamada “ArbolCifras”, se trata de una clase con un TDA árbol que genera todas las soluciones posibles realizando operaciones elementales con los números introducidos.

Dicha clase tiene un struct llamado nodo para ayudarnos en la implementación del TDA árbol. Hemos elegido esta estructura para el nodo por simplicidad a la hora de recorrer el árbol. La etiqueta es el resultado de la operación realizada a la hora de crear el nodo, por ejemplo, si la operación es 2+2 el valor de la etiqueta será 4.

```
struct nodo
{
    int etiqueta = 0;
    string expresion;
    multiset<int> numeros;
    vector<nodo *> hijos;
    nodo *padre = nullptr;
    nodo() {}
};
```

Para facilitar la obtención de la ruta que hay que seguir para obtener el número objetivo hemos decidido crear un parámetro expresión cuyo valor es, según el ejemplo anterior, “2+2=4”, de esta manera solo tenemos que ir recorriendo los hijos, con el puntero al padre del nodo, y obtener el valor de la expresión.

Para permitir valores repetidos en los números hemos usado un multiset para así también tener los valores ordenados.

Siguiendo con el TDA ArbolCifras hemos decidido que, cuando se cree un nodo, validar si la etiqueta es un número de 3 cifras, ya que de esta manera puede tratarse de una posible solución objetivo y nos interesa guardar todas las soluciones para saber si el conjunto de números aportado es una combinación mágica, un conjunto es combinación mágica si se puede obtener cualquier número de 3 cifras realizando operaciones elementales con ese conjunto.

Para poder almacenar todos los nodos hemos creado un set (para no tener valores repetidos) de punteros de nodo “set<nodo*>”, un valor entra en ese set con la condición anterior, si su etiqueta es un número de 3 cifras. Gracias a este set cuando queramos obtener la ruta de operaciones que hay que seguir solamente buscamos el nodo en la lista cuya etiqueta sea el valor objetivo.

El algoritmo escogido para mostrar la solución ha sido: 1) buscar el nodo cuya etiqueta tenga el numero que queremos calcular, 2) guardar en una pila la expresión del nodo, 3) obtener el nodo padre y repetir el proceso 2 y 3 hasta que al guardar el nodo padre nos devuelva nullptr.

Juego de las Letras

Para el desarrollo del juego de las letras se ha decidido seguir la estructura dada en la práctica de varias clases auxiliares que permiten organizar el problema en distintos TDAs. Cada uno de ellos son:

TDA Dictionary, que almacena todas las palabras válidas del diccionario en un set de strings. Lee desde un archivo .txt (en el caso del proyecto es diccionario.txt) dichas palabras y tiene distintas funciones que opera con ese contenedor, a parte de un iterador que permite recorrer el mismo. Se puede filtrar por longitud de palabras, por palabras que contengan una secuencia de caracteres especifica, entre otros.

TDA LettersSet es el conjunto de letras disponibles en el juego, que también guarda sus repeticiones y su puntuación. Se administra con un map de parejas de caracteres y una estructura externa (LetterInfo) el cual almacena la información de repeticiones y puntuación de cada una de ellas. Tiene funciones como calcular la puntuación de una palabra evaluando cada una de sus letras, ver cuantas repeticiones tiene una letra o cuál es el total de letras del conjunto.

TDA LettersBag se pedía para simular la bolsa de letras del juego, esta se forma por un vector de caracteres que incluye todas las repeticiones de cada letra (de igual manera que se calcula el total de letras del conjunto de LettersSet) y se utiliza para extraer letras aleatorias respetando las cantidades que se definen en el LettersSet. Para extraer, genera un índice aleatorio, saca el carácter y luego lo borra del conjunto.

TDA Solver se utiliza para resolver el juego, utiliza los TDA anteriores para formar esas posibles soluciones. Su funcionamiento es: primero saca un conjunto de letras de la bolsa, luego con un iterador sobre el TDA Dictionary se van revisando todas las palabras que hay para calcular, si la palabra vale como posible solución (este paso también comprueba que la palabra se puede formar con las letras dadas en el juego) teniendo en cuenta el modo de juego de la partida en la que esté, de forma que

si esta en modo longitud la puntuación de la palabra viene de la longitud de la misma y si es en modo puntuación se evalúa cuanto vale cada uno de los caracteres y la suma será la puntuación final. Con todo esto, si ve que la mejor solución tiene la MISMA puntuación que la palabra evaluada la añade a un vector de strings que va guardando las soluciones con mayor puntuación, pero si la palabra evaluada tiene MAYOR puntuación que la anterior más grande entonces limpia ese vector y almacena la nueva palabra, así hasta recorrer el diccionario entero.

Ya por último hemos hecho un ejecutable letras.cpp que permite interactuar con el juego de las letras, permite al jugador introducir una palabra que contenga las letras sacadas por pantalla, comprueba que este en el diccionario y si está te muestra tu puntuación. Tras esto, hace uso del TDA Solver para mostrar por pantalla todas las mejores soluciones posibles encontradas, y antes de acabar permite elegir si se quiere seguir jugando o no (Y/N). Con esto, la prueba de las letras estaría completa y completamente funcional, se pensó en la idea de sumar la puntuación entre partidas si se juega varias de ellas, y también optimizar algo la búsqueda en el diccionario limitando el rango de búsqueda conforme se iban encontrando soluciones, pero no se ha podido por falta de tiempo principalmente. Aun con esto, el programa es funcional y cumple su función correctamente.