CONECTA 4

V1.0

Generado por Doxygen 1.8.11

Índice

1. Práctica Final: CONECTA4

Versión

v1

Autor

Luis Baca, Óscar Gómez, Carmen Navarro y Carlos Cano

NOTA: este documento contiene el guión de la práctica final CONECTA 4. Para consultar la documentación completa de los T.D.A. proporcionados, consulte los ficheros `doc/latex/refman.pdf` y `doc/html/index.html`.

1.1. Introducción

Los objetivos de este guión de prácticas son los siguientes:

- Resolver un problema eligiendo la mejor estructura de datos para las operaciones que se solicitan.

Los requisitos para poder realizar esta práctica son:

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: T.D.A. Lineales.
4. Haber estudiado el Tema 4: STL e Iteradores.
5. Haber estudiado estructuras de datos jerárquicas: Árboles

1.2. Objetivos

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 requieren estructuras de datos que permiten almacenar grandes volúmenes de datos y poder acceder a ellos de la forma más eficiente.

1.3. Problema

El estudiante debe implementar un programa que simule el juego "Conecta 4" [Conecta4]. El objetivo de Conecta 4 es alinear cuatro fichas sobre un tablero formado por seis filas y siete columnas. Cada jugador dispone de 21 fichas de un color (por lo general, rojas o amarillas). En nuestro caso, las fichas de los jugadores se indicarán con los caracteres "X" y "O". Por turnos, los jugadores deben introducir una ficha en la columna que prefieran (siempre que no esté completa) y ésta caerá a la posición más baja. Gana la partida el primero que consiga alinear cuatro fichas consecutivas de un mismo color en horizontal, vertical o diagonal. Si todas las columnas están llenas pero nadie ha hecho una fila válida, hay empate.

El estudiante debe implementar una versión de este juego en el que los movimientos de uno de los jugadores se efectúan de forma automática. Esta modalidad, que llamaremos "Player VS. Computer" o "1 jugador automático", requiere implementar un mecanismo de decisión automática basado en la disposición actual del tablero. Para implementar este mecanismo de decisión automática, debemos representar todos los posibles movimientos a partir del tablero actual y definir una "métrica" que nos permita "evaluar" cuál de los movimientos posibles deriva un tablero con más posibilidad de éxito para el jugador automático. Cuanto mejor sea esta métrica, más difícil será batir al jugador automático en una partida de Conecta 4.

Por ejemplo, la métrica más sencilla consistiría en evaluar si alguno de los movimientos posibles para el jugador automático produce directamente un tablero donde el jugador automático consigue 4 en línea (y, por tanto, gana la partida) o evita un 4 en línea del rival (y, por tanto, evita perder la partida). Sin embargo, esta métrica sólo nos permitiría evaluar ciertos tableros, dejándonos "a ciegas" para tomar una decisión en tableros que no derivan una victoria o derrota en un sólo movimiento. **Otras métricas más sofisticadas requieren evaluar todos los posibles movimientos a partir de un tablero dado, contemplando varios turnos de los dos jugadores. Una representación natural para almacenar esta información es un Árbol**. El nodo raíz representa el tablero actual y le añadimos un hijo por cada posible tablero que se deriva del padre con un movimiento del jugador que tiene el turno. De este modo, el nivel 1 del árbol representa todos los movimientos posibles de uno de los jugadores, el nivel 2 todos movimientos posibles del otro jugador para cada uno de los tableros de nivel 1, y así sucesivamente. Cada rama del árbol termina en una hoja en la que, o bien se da la victoria de uno de los jugadores, o bien hay empate (no quedan movimientos posibles). Las siguientes figuras ilustran la estructura interna de un TDA Árbol General que almacena los tableros de Conecta 4 con la disposición de la partida en cada momento. La primera figura incide en la estructura interna del árbol y la segunda en posibles tableros almacenados en el árbol.

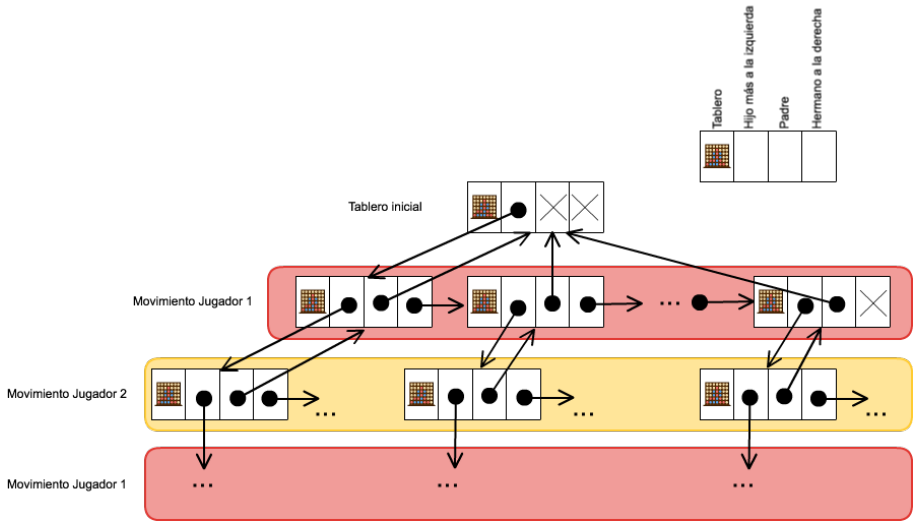

Figura 1 Estructura interna de un TDA Árbol General para representar el espacio de soluciones de Conecta 4

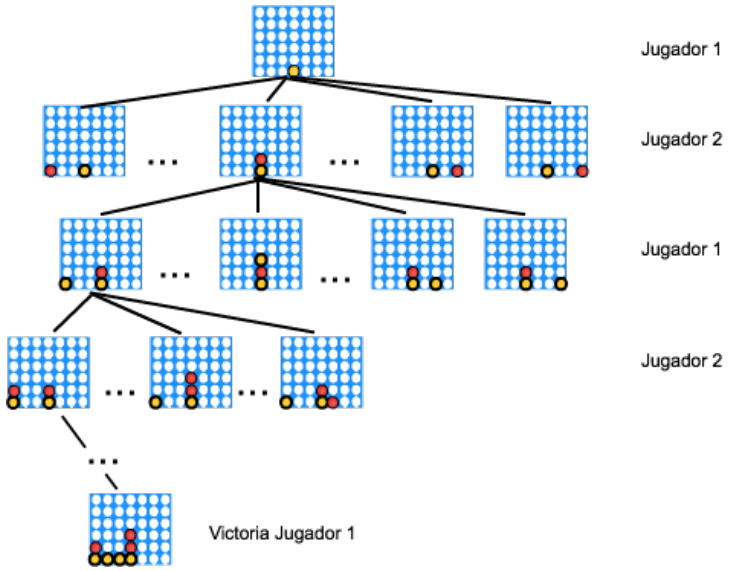

Figura 2 Detalle de posibles movimientos en el espacio de soluciones de Conecta 4

El tamaño real de un tablero de Conecta 4 es de 6 filas x 7 columnas, con lo que cada nodo del árbol tiene hasta 7 hijos. Si una columna del tablero estuviera completa, habría menos movimientos posibles y por tanto menos hijos para ese nodo. Los nodos en los que la partida ya está resuelta (por victoria de uno de los jugadores o por no quedar movimientos posibles) serían los nodos hoja del árbol.

De este modo, en el primer nivel del árbol (considerando 1 turno desde el tablero actual), almacenaríamos hasta 7 jugadas posibles o tableros diferentes. En el segundo nivel (considerando 2 turnos desde el actual), almacenaríamos hasta 7^2 tableros posibles. En el tercero, hasta 7^3. Y así sucesivamente. El estudiante debe ser consciente del coste de almacenar y recorrer este volumen de tableros para implementar métricas de recomendación de movimientos que sean viables (coste razonable en tiempo y memoria). Por ejemplo, se recomienda implementar métricas que utilicen un árbol de movimientos para los próximos N turnos desde el tablero actual, donde N queda a elección del estudiante, para acotar el consumo en tiempo y memoria de la implementación de estas métricas.

Además, a medida que el juego va avanzando, se recomienda reutilizar las ramas del árbol de soluciones que siguen siendo posibles y utilizar mecanismos de poda (ya implementados en el TDA ArbolGeneral) para liberar espacio en memoria eliminando las ramas con movimientos que no se han tomado.

En esta práctica, el estudiante deberá proponer e implementar distintas métricas para guiar la decisión del jugador automático, representando todas las jugadas posibles para los próximos N turnos utilizando un T.D.A. Árbol General.

1.4. Tareas a realizar

Se proprocionan:

1. T.D.A. Tablero: representación del tablero de Conecta 4. Un objeto tablero representa un instante dado de una partida del Conecta 4, es decir, la disposición de las fichas de ambos jugadores sobre el tablero y el turno del jugador al que le corresponde hacer el próximo movimiento.
2. T.D.A. Mando: representación gráfica del tablero de Conecta 4. Incluye la gestión de Entrada/Salida que permite jugar a Conecta 4 de forma interactiva.
3. T.D.A. ArbolGeneral.

Se deberán llevar a cabo las siguientes tareas:

1. Construir el T.D.A. Conecta4, que almacena todos los tableros posibles de Conecta 4 generados en los próximos N turnos a partir de un tablero inicial utilizando las clases T.D.A. ArbolGeneral y T.D.A. Tablero.
2. Definir los métodos del T.D.A. Conecta4 para la solución de los problemas propuestos, en particular, para la implementación de un jugador automático.
3. Probar los módulos con programas test. Se puede usar la STL en todos los módulos excepto en la implementación del TDA ArbolGeneral.

A continuación se detallan los programas que se deberán desarrollar.

1.5. Implementar un jugador automático

Conecta 4 es un juego de estrategia abstracta donde los contrincantes disponen de información perfecta [Conecta4]. Por norma general, el primer jugador tiene más posibilidades de ganar si introduce la primera ficha en la columna central. Si lo hace en las contiguas se puede forzar un empate, mientras que si la mete en las más alejadas del centro su rival puede vencerle con mayor facilidad. Existen libros y webs donde se explican las mejores estrategias para ganar en el Conecta 4, por ejemplo puede consultarse [Allen13].

El estudiante debe implementar distintas "métricas" que permitan al jugador automático decidir qué movimiento realizar dada una disposición del tablero. Las métricas implementadas analizarán el espacio de soluciones posible a partir del tablero actual para evaluar cuál es el movimiento más beneficioso para el jugador automático (esto es, el movimiento que más aumente las expectativas de ganar la partida). La calidad de estas métricas dependerá, en parte, de la cantidad de información de que dispongan. De este modo, una métrica que sólo considere dos niveles en profundidad a partir del tablero actual (esto es, todos los movimientos posibles del jugador y todas las repuestas inmediatas del rival) tendrá más limitaciones que una métrica que disponga de todo el árbol de soluciones a partir del tablero actual. Evidentemente, la contrapartida es que una métrica que dispone de todo el árbol de soluciones requiere de un alto consumo en memoria y en tiempo de cómputo (según el tamaño del tablero, almacenar todo el árbol de soluciones puede ser directamente inviable). De este modo, se propone diseñar distintas métricas

utilizando un árbol de soluciones con N niveles en profundidad a partir del tablero actual, donde N es variable y su elección se deja al estudiante.

El estudiante debe implementar la clase TDA Conecta4 utilizando como tipo rep. el TDA. ArbolGeneral para representar el espacio de soluciones con profundidad N e implementar distintas métricas que, haciendo uso de esta representación del espacio de soluciones, recomienden un movimiento al jugador automático. Anteriormente se ha descrito una posible métrica inicial que sólo requiere explorar un nivel del espacio de soluciones: evaluar si alguno de los movimientos posibles para el jugador automático produce directamente un tablero donde el jugador automático consigue 4 en línea (y, por tanto, gana la partida) o evita un 4 en línea del rival (y, por tanto, evita perder la partida). Otra métrica sencilla que requiere explorar todo el árbol de búsqueda hasta profundidad N consiste en hacer recuento de cuántos nodos a profundidad <=N son favorables/desfavorables (victorias/empates/derrotas) al jugador automático para cada nodo hijo del tablero actual, de modo que el hijo con mejores métricas indicará el mejor movimiento posible. Otras métricas más sofisticadas podrían explorar este espacio de búsqueda para considerar no sólo victorias o derrotas, sino también recontar el número de secuencias de tres/dos fichas del jugador automático no rodeadas por fichas del rival en cada uno de los tableros.

Es importante destacar que, para aliviar el costo de almacenamiento de las métricas basadas en árboles de búsqueda, una vez realizado un movimiento, el árbol debe podarse para eliminar los nodos asociados a movimientos alternativos que finalmente no fueron tomados. El TDA ArbolGeneral incluye métodos de poda facilitan esta tarea.

El estudiante tiene libertad para implementar las métricas sugeridas o cualquier otra para guiar los movimientos del jugador automático. La documentación final a entregar debe recoger una descripción completa de cada métrica implementada y una justificación de sus resultados. Se valorará la dificultad para batir al jugador automático guiado por las métricas implementadas por el estudiante.

En la siguiente sección se detalla cómo implementar este programa principal para jugar a Conecta 4.

1.6. Implementar una partida de Conecta 4

Se adjunta un programa de ejemplo (`src/conecta4.cpp`) que utiliza los TDA Tablero y TDA Mando para implementar una partida de Conecta 4 interactiva entre dos jugadores (no automáticos). Nótese que este programa va pidiendo por teclado sucesivamente cada movimiento de ambos jugadores, y no hace uso del TDA ArbolGeneral (no representa el espacio de soluciones posibles para emitir recomendaciones sobre el mejor movimiento o tomar decisiones automáticas).

```
#include <iostream>
#include <vector>
#include <ctime>
#include <cstdlib>
#include <stdio.h>
#include <unistd.h>
#include <termio.h>         // Linux/Windows users
//#include <termios.h>      // Mac OSX users

#include "ArbolGeneral.h"
#include "tablero.h"
#include "mando.h"

using namespace std;

// Captura el caracter pulsado por teclado (sin necesidad de pulsar, a continuación, Enter).
// Devuelve: Caracter pulsado.

char getch() {
 ...
}

// Imprime en pantalla el tablero completo, con el mando y el jugador.
// t : Tablero que se va a imprimir.
// m : Mando indicando la posición del jugador.

void imprimeTablero(Tablero & t, Mando & m){
    cout << m.GetJugador() << endl;
    cout << t ;
    cout << m.GetBase() << endl;
    cout << m.GetMando() << endl;
```

```
}


// Implementa el desarrollo de una partida de Conecta 4 con dos jugadores humanos
// Devuelve: identificador (int) del jugador que gana la partida
int jugar_partida() {

    Tablero tablero(5, 7);      //Tablero 5x7
    Mando mando(tablero);       //Mando para controlar E/S de tablero
    char c = 1;
    int quienGana = tablero.quienGana();
    //mientras no haya ganador y no se pulse tecla de terminación
    while(c != Mando::KB_ESCAPE && quienGana == 0) {
        system("clear");
        mando.actualizarJuego(c, tablero);  // actualiza tablero según comando c
        imprimeTablero(tablero, mando);     // muestra tablero y mando en pantalla
        quienGana = tablero.quienGana();    // hay ganador?
        if(quienGana==0) c = getch();       // Capturamos la tecla pulsada.
    }

    return tablero.quienGana();
}

int main(int argc, char *argv[]){
    int ganador = jugar_partida();
    cout « "Ha ganado el jugador " « ganador « endl;
}
```

El estudiante debe implementar una partida de Conecta 4 en la que uno de los jugadores realiza sus movimientos automáticamente y el otro jugador introduce sus movimientos por teclado. Utilice el código de `src/conecta4.cpp` como base para programar este módulo. El programa debe permitir elegir cuál es el jugador que inicia la partida: el jugador automático o el humano.

Además, en caso de implementar distintas métricas que guían al jugador automático, el programa contará con un argumento entero que permite identificar la métrica utilizada por el jugador automático (métrica 1, métrica 2, y así sucesivamente). La métrica 1 será, por defecto, la que mejor resultados haya obtenido para el estudiante.

Finalmente, el programa debe también permitir definir el tamaño del tablero.

En resumen, el módulo a desarrollar por el estudiante debe tener los siguientes argumentos:

```
prompt %> conecta4 <filas_tablero> <cols_tablero> <metrica> <turno>
```

Donde:

- `<filas_tablero>, <cols_tablero>`: especifica las dimensiones del tablero (por defecto, 4 en ambas opciones para tablero 4x4)
- `<metrica>`: indica la métrica a utilizar para guiar al jugador automático. Comenzar a numerar con 1 y en adelante, en orden de mejores a peores resultados obtenidos con esa métrica para el jugador automático. 1 es la opción por defecto (la métrica con mejores resultados). 0 para que la partida se desarrolle entre dos jugadores humanos (sin jugador automático).
- `<turno>` indica qué jugador tiene el primer turno:
  - 1: primer turno para el jugador humano
  - 2: primer turno para el jugador automático

1.7. Manejo del T.D.A. ArbolGeneral

Se adjunta un programa de ejemplo (`src/arboltablero_test.cpp`) en el que se exhibe alguna funcionalidad asociada a los TDA Tablero y ArbolGeneral. En este programa se crean distintos objetos Tablero con distintas jugadas posibles de Conecta 4 y una estructura ArbolGeneral que almacena jerárquicamente estos tableros.

En particular, se muestra como:

Para TDA Tablero:

- crear de tableros con las dimensiones deseadas
- colocar ficha para el jugador que dispone del turno
- cambiar turno entre jugadores

Para TDA ArbolGeneral:

- crear de árboles con TDA Tablero como nodos
- obtener el nodo raiz de un ArbolGeneral
- insertar nodos como hijomasizquierda
- insertar nodos como hermanoderecha
- imprimir ArbolGeneral en preorden
- podar una rama del ArbolGeneral

```
#include <iostream>
#include "ArbolGeneral.h"
#include "tablero.h"
#include <string>

using namespace std;


int main(int argc, char *argv[]){

  //Tablero vacío 5x7
    Tablero tablero(5, 7);

    //Manualmente se insertan algunos movimientos:
    tablero.colocarFicha(3);  //Jugador 1 inserta ficha en columna 3
    tablero.cambiarTurno();
    tablero.colocarFicha(1);  //Jugador 2 inserta ficha en columna 1
    tablero.cambiarTurno();
    tablero.colocarFicha(3);  //Jugador 1 inserta ficha en columna 3.
    tablero.cambiarTurno();

    //Se muestra el tablero
    cout « "Tablero obtenido tras tres movimientos: \n"«tablero;

    //A partir de la situación actual del tablero, montamos un árbol para estudiar algunas posibilidades.

    // Éste es el árbol que queremos montar:
    //         tablero
    //           |
    //      |---------------|
    //    tablero1       tablero2
    //                       |
    //                   tablero3


    //Árbol 'partida', con 'tablero' como nodo raíz
    ArbolGeneral<Tablero> partida(tablero);

    //Estudio opciones a partir de tablero: Jugador 2 coloca ficha en columna 1. (tablero1)
    Tablero tablero1(tablero);          //tablero queda sin modificar
    tablero1.colocarFicha(1);
    ArbolGeneral<Tablero> arbol1 (tablero1);  //creo árbol con un nodo (tablero1)
```

```
    //Otra opción: Jugador 2 coloca ficha en columna 2. (tablero2)
    Tablero tablero2(tablero);          //tablero queda sin modificar
    tablero2.colocarFicha(2);
    ArbolGeneral<Tablero> arbol2(tablero2);   //creo árbol con un nodo

    // Sobre la última opción, ahora contemplo la posibilidad de que
    //  Jugador 1 coloque ficha también en columna 2.
    tablero2.cambiarTurno();          //modifico tablero2 (esta modificación sería tablero3)
    tablero2.colocarFicha(2);
    ArbolGeneral<Tablero> arbol3 (tablero2);  //creo árbol con un nodo
    arbol2.insertar_hijomasizquierda(arbol2.raiz(), arbol3);  //añado este árbol como hijo de arbol2

    // Inserto arbol1 y arbol2 como hijos de partida.
    // arbol1 es el hijo más a la izquierda y arbol2 es hermano a la derecha de arbol1

    //  Forma de hacerlo A: inserto varios hijomasizquierda en el orden inverso al deseado
    //  partida.insertar_hijomasizquierda(partida.raiz(), arbol2);
    //  partida.insertar_hijomasizquierda(partida.raiz(), arbol1);  //hijomasizquierda desplaza al anterior
       a la derecha

    // Forma de hacerlo B: inserto un hijomasizquierda y hermanoderecha
    partida.insertar_hijomasizquierda(partida.raiz(), arbol1);              //inserto un hijomasizquierda
    partida.insertar_hermanoderecha(partida.hijomasizquierda(partida.raiz()), arbol2);  //le inserto un
       hermanoderecha


    // Recorremos en preorden para comprobar el arbol 'partida' resultante
    cout « "\nÁrbol en preorden: \n"«endl;
    partida.recorrer_preorden();

    // Podamos el hijomasizquierda y recorremos en preorden:
    ArbolGeneral<Tablero> rama_podada;
    partida.podar_hijomasizquierda(partida.raiz(), rama_podada);

    cout « "\nRecorrido preorden después de podar hijomasizquierda: \n"«endl;
    partida.recorrer_preorden();

    return 0;
}
```

El estudiante deber revisar detenidamente este código y la documentación completa asociada a las clases TDA Tablero y TDA ArbolGeneral para familiarizarse con su manejo y sacarles el máximo partido para la implementación del T.D.A. Conecta4.

1.8. Recomendaciones

- Analice con detenimiento la documentación de las clases proporcionadas: T.D.A. ArbolGeneral, T.D.A. Tablero y T.D.A. Mando. Parte importante del esfuerzo que debe realizar en esta práctica es entender cómo debe utilizar estas clases para su proyecto. Esta práctica es habitual en empresas: para el desarrollo de un proyecto se cuenta con parte del código desarrollado previamente por otro equipo.
- Para acotar el espacio de búsqueda y el tamaño de los objetos ArbolGeneral, comience el desarrollo de su proyecto considerando tableros más pequeños (por ejemplo, 4x4). Cuando el funcionamiento sea correcto, amplíe progresivamente el tamaño de los tableros. Del mismo modo, considere con cautela la profundidad máxima N que va a explorar en el árbol de soluciones. Comience con un valor bajo de N y estudie hasta qué valor de N (depende del tamaño del tablero) es viable el cálculo de sus métricas.

1.9. Entrega

El estudiante deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre "conecta4.tgz" y entregarlo antes de la fecha que se publicará en la página web de la asignatura. Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables, ni la carpeta datos. Es recomendable que haga una "limpieza" para eliminar los archivos temporales o que se puedan generar a partir de los fuentes. El estudiante debe incluir el archivo Makefile para realizar la compilación. Tenga en cuenta que los archivos deben estar distribuidos en directorios:

conecta4

- Makefile
- include – Carpeta con ficheros de cabecera (.h)
- src –Carpeta con código fuente (.cpp)
- doc –Carpeta con Documentación
- obj – Carpeta para código objeto (.o)
- bin – Carpeta para ejecutables
- datos – Carpeta para ficheros de datos

Para realizar la entrega, en primer lugar, realice la limpieza de archivos que no se incluirán en ella, y sitúese en la carpeta superior (en el mismo nivel de la carpeta "conecta4") para ejecutar:

```
prompt% tar zcv conecta4.tgz conecta4
```

tras lo cual, dispondrá de un nuevo archivo conecta4.tgz que contiene la carpeta conecta4 así como todas las carpetas y archivos que cuelgan de ella.

La fecha límite de entrega es el día 22 de Enero de 2017 a las 23:59.

1.10. Referencias

[Allen13] James D. Allen. "Expert Play in Connect-Four" (en inglés). Archivado desde el original el 4 de noviembre de 2015. http://web.archive.org/web/20141125065033/http://homepages.cwi.nl/∼tromp/c4.html

[Conecta4] https://es.wikipedia.org/wiki/Conecta_4

[GAR06b] Garrido, A. Fdez-Valdivia, J. "Abstracción y estructuras de datos en C++". Delta publicaciones, 2006.