Apalabrados

Ejemplo de PEC

Estructura de la Información / Diseño de Estructuras de Datos

Índice

Ejemplo de Prueba de Evaluación Continua

PRIMERA PARTE

Se quiere diseñar una estructura de datos para almacenar un conjunto de campeonatos del popular juego "Apalabrados". En esta PEC haremos una prueba piloto de una parte del sistema (un conjunto reducido de funcionalidades) y trabajaremos con volúmenes de información pequeños para que os familiaricéis con el problema a resolver y "practiquéis" un poco el diseño y la composición de estructuras de datos para dar forma y solución al problema planteado. Concretamente, para la realización del ejercicio considerad:

1 El número de campeonatos C que guardaremos a la prueba piloto es variable y relativamente pequeño, de unos centenares.

2 El número de partidas P de un campeonato también es variable y relativamente pequeño, de unos centenares.

3 El número de palabras PA que formarán parte del diccionario de palabras válidas de un campeonato será variable y relativamente pequeño.

4 El número de letras diferentes disponibles L será de 29, cada letra tiene una puntuación asociada en función de lo común que sea (por ejemplo, la 'A' dará 1 punto, la 'H' dará 8 puntos, etc.).

5 El número de jugadores de una partida JP será de 2.

6 El número de fichas disponibles F inicialmente en una partida será fijo, 100. No confundáis las letras disponibles en un campeonato con las fichas de una partida, las fichas de las partidas son la suma de todas las ocurrencias de letras disponibles para jugar. Tened en cuenta también que hay una proporción más elevada de fichas para las letras más comunes (por ejemplo, 12 fichas de la letra 'E', 2 fichas de la letra 'B', etc.). Podéis encontrar la relación de fichas y letras en el siguiente enlace: http://es.wikipedia.org/wiki/scrabble

7 El tablero de cada partida tendrá 15 filas y 15 columnas. Considerad que no hay casillas especiales.

8 Cada jugador tendrá un máximo de 7 letras para hacer una jugada.

9 El número de jugadas GP de una partida es variable y relativamente pequeño.

Ejercicio 1

Especificad un TAD *Apalabrados* para guardar los campeonatos que permita:

- Crear un campeonato nuevo, el campeonato tendrá un nombre y éste será único.
- Añadir una palabra al diccionario de palabras que se usará durante todo el campeonato.
- Añadir una letra con su puntuación y el número de fichas de esta letra que habrá disponibles. Esta asociación se usará durante todo el campeonato.
- Crear una nueva partida, la partida tendrá un nombre que la identificará y, en el momento de crearla, se especificarán los dos jugadores que la jugarán y el campeonato en el que participan.
- Añadir una ficha a la partida de un campeonato. No se pueden añadir fichas a una partida empezada ni más fichas de una letra que las máximas indicadas (en la misma letra).
- Empezar la partida. Inicializa una partida de un campeonato y asigna a cada jugador sus fichas iniciales. El reparto de fichas consiste al coger una ficha asociada a la partida (la última insertada) para cada jugador de manera alternativa hasta completar sus 7 fichas.
- Hacer una jugada en una partida. Se tendrá que especificar el jugador que hace la jugada, la fila y columna donde empieza la palabra, la orientación (horizontal –de izquierda a derecha- o vertical –de arriba a abajo-) y la palabra. Hay que comprobar que la palabra sea válida, que el jugador tiene todas las fichas necesarias para hacerla, calcular la puntuación de la jugada y actualizar las letras disponibles del jugador y su puntuación. Considerad también que en esta primera versión los jugadores no pueden cambiar fichas y que para la puntuación de la jugada sólo hay que contar la palabra especificada y no las posibles palabras generadas a partir de esta. NOTA: ver las aclaraciones al final de este documento.
- Averiguar cuántas veces se ha utilizado una determinada palabra en el campeonato.
- Averiguar qué palabra ha obtenido la puntuación más alta en el campeonato.

Apartado a)

Especifica la firma del TAD Apalabrados. Es decir, indica el nombre que darías a las operaciones encargadas de cada funcionalidad requerida. Indica, también, los parámetros de entrada y de salida cuando sean necesarios.

Apartado b)

Haz la especificación contractual de las operaciones del TAD Apalabrados. En la redacción de la especificación puedes usar, si se requiere, cualquiera de las operaciones del TAD. Toma como modelo la especificación del apartado 1.2.3 del Módulo 1 de los materiales docentes. Se valorará especialmente la concisión (ausencia de elementos redundantes o innecesarios), precisión (definición correcta del resultado de las operaciones), completitud (consideración de todos los casos posibles en que se puede ejecutar cada operación) y carencia de ambigüedades (conocimiento exacto de cómo se comporta cada operación en todos los casos posibles) de la solución. Es importante responder este apartado usando una descripción condicional y no procedimental. La experiencia nos demuestra que no siempre resulta fácil distinguir entre las dos descripciones, es por eso que hacemos especial énfasis insistiendo que pongáis mucha atención en vuestras definiciones.

A título de ejemplo indicaremos que la descripción condicional (la correcta a utilizar en el contrato) de llenar un vaso vacío con agua sería::

@pre el vaso se encuentra vacío.

@post el vaso está lleno de agua.

En cambio una descripción procedimental (incorrecta para utilizar en el contrato) tendría una forma parecida a:

@pre el vaso tendría que encontrarse vacío, si no se encontrase vacío se tendría que vaciar.

@post se acerca el vaso al grifo y se echa agua hasta que esté lleno.

También debéis tener en cuenta el invariante si este es necesario para describir el TAD.

Ejercicio 2

En el Ejercicio 1 habéis definido la especificación de un nuevo TAD, el TAD Apalabrados, ahora os pedimos que hagáis el diseño de las estructuras de datos que formarán este TAD. Diseñad, pues, el sistema para que sea el máximo de eficiente posible, tanto a nivel de eficiencia espacial como temporal, teniendo en cuenta los volúmenes de información y las restricciones especificadas en el enunciado. Tened en cuenta sólo las operaciones que se piden en el enunciado al hacer este diseño.

Apartado a)

Dudamos entre utilizar un vector, una lista encadenada o una lista encadenada ordenada para almacenar las jugadas. Justificad cuál creéis que es la mejor opción.

Apartado b)

Dudamos entre utilizar un vector, una cola, una pila o una lista encadenada para almacenar las fichas. Justificad cuál creéis que es la mejor opción.

Apartado c)

Haced un dibujo de la estructura de datos global para el TAD *Apalabrados* donde se vean claramente las estructuras de datos que elegís para representar cada una de las partes y las relaciones entre ellas. Haced el dibujo de la estructura completa, con todas los estructuras que os permitan implementar las operaciones definidas en la especificación.

Apartado d)

Justificad todas las estructuras de datos que habéis elegido a la hora de hacer el diseño del TAD. La justificación de cada una de las estructuras de datos tiene que ser del estilo:

"Para guardar XXX elegimos una lista encadenada ordenada puesto que el número de elementos no es muy grande, necesitamos acceso directo y recorridos ordenados."

Ejercicio 3

En el Ejercicio 1 habéis definido la especificación del TAD *Apalabrados* con sus operaciones y en el Ejercicio 2 habéis elegido las estructuras de datos para cada parte del TAD. En este ejercicio os pedimos que os fijéis en los algoritmos que os servirán para implementar algunas de las operaciones especificadas y en el estudio de eficiencia de las mismas. Tened en cuenta que la implementación de las operaciones va estrechamente ligada a la elección de las estructuras de datos que hayáis hecho.

Apartado a)

Haced el pseudocodigo y el estudio de eficiencia de la operación que hayáis definido para hacer una jugada en una partida. Para hacerlo tenéis que describir brevemente su comportamiento indicando los pasos que la componen (con frases como por ejemplo: "insertar en el árbol AVL / borrar de la tabla de dispersión / consulta del pilón / ordenar el vector..."), especificando la eficiencia asintótica de cada paso y dando la eficiencia total de la operación.

Apartado b)

Haced el pseudocodigo y el estudio de eficiencia de la operación que hayáis definido para averiguar qué palabra ha obtenido la puntuación más alta en el campeonato. Igual que al ejercicio anterior, tenéis que describir brevemente su comportamiento indicando los pasos que la componen (con frases como por ejemplo: "insertar en el árbol AVL / borrar de la tabla de dispersión / consulta del pilón / ordenar el vector..."), especificando la eficiencia asintótica de cada paso y dando la eficiencia total de la operación.

Apartado c)

Suponed ahora que queremos ofrecer una funcionalidad que permita reproducir alguna de las partidas del campeonato, es decir, volver a hacer todas las jugadas de las que se compone una partida. ¿Cambiaríais alguna de las estructuras de datos para que esta operación fuera eficiente? En caso afirmativo, explicad qué cambios haríais y por qué.

Ejercicio 4

Indica cual de los TADs de la biblioteca de TADs de la asignatura te parecen más adecuados para utilizarlos en la implementación de cada una de las estructuras de datos definidas para el TAD *Apalabrados*.

Aclaración: Suponiendo que el tablero se encuentra en la situación indicada a la izquierda, el jugador puede jugar "CRIE" desde la posición 2,3 en vertical, pero solo necesita las letras "C" y "I", al aprovechar las letras "R" y "E" que ya están en el tablero. Es decir, hay que comprobar que "CRIE" (la palabra jugada) es una palabra válida y que el jugador tiene las letras "C" y "I" (las que necesita para completar la palabra que quiere jugar).

		C			
		R	O	S	
		I		A	
I	D	E	A	L	

		C			
		R	O	S	
		I		A	
I	D	E	A	L	

SEGUNDA PARTE

En esta PAC continuamos diseñando una estructura de datos para guardar un conjunto de campeonatos del popular juego "Apalabrados". Ahora ampliaremos las funcionalidades y trabajaremos con un volumen de información grande, que requerirán el uso de estructuras de datos de los módulos 4 al 7. Concretamente, para la realización del ejercicio consideramos::

De la PRIMERA PARTE:

- El número de campeonatos C que guardaremos en la prueba piloto es variable y relativamente pequeño, de unos centenares.
- El número de letras diferentes disponibles L será de 29, cada letra tiene una puntuación asociada en función de lo común que sea (por ejemplo, la 'A' dará 1 punto, la 'H' dará 8 puntos, etc.).
- El número de jugadores de una partida JP será de 2.
- El número de fichas disponibles F inicialmente en una partida será fijo, 100. No confundáis las letras disponibles en un campeonato con las fichas de una partida, las fichas de las partidas son la suma de todas las ocurrencias de letras disponibles para jugar. Tened en cuenta también que hay una proporción más elevada de fichas para las letras más comunes (por ejemplo, 12 fichas de la letra 'E', 2 fichas de la letra 'B', etc.). Podéis encontrar la relación de fichas y letras en el siguiente enlace: http://es.wikipedia.org/wiki/scrabble
- Cada jugador tendrá un máximo de 7 letras para hacer una jugada.
- El número de jugadas GP de una partida es variable y relativamente pequeño.

Cambios respecto la PRIMERA PARTE:

- El número de partidas P de un campeonato también es variable y muy grande.
- El número de palabras PA que formarán parte del diccionario de palabras válidas de un campeonato será muy grande y de tamaño conocido.
- El tablero de cada partida tendrá 15 filas y 15 columnas. Consideramos que sí hay casillas especiales (letra por 2 o por 3, palabra por 2 o por 3).
- Si un jugador coloca las siete fichas en una misma jugada, obtiene 40 puntos extra.

Nuevo en la SEGUNDA PARTE:

- El número de jugadores J dados de alta será muy grande y en constante aumento.
- El número de jugadores inscritos I en un campeonato será variable y, en algunos campeonatos, puede ser muy grande.

Ejercicio 5

Especificad un TAD *Apalabrados* para guardar los campeonatos que permita:

De la PRIMERA PARTE:

- Crear un campeonato nuevo, el campeonato tendrá un nombre y éste será único.
- Añadir una palabra al diccionario de palabras que se usará durante todo el campeonato.
- Añadir una letra con su puntuación y el número de fichas de esta letra que habrá disponibles. Esta asociación se usará durante todo el campeonato.
- Añadir una ficha a la partida de un campeonato. No se pueden añadir fichas a una partida empezada ni más fichas de una letra que las máximas indicadas (en la misma letra).
- Empezar la partida. Inicializa una partida de un campeonato y asigna a cada jugador sus fichas iniciales. El reparto de fichas consiste al coger una ficha asociada a la partida (la última insertada) para cada jugador de manera alternativa hasta completar sus 7 fichas.
- Averiguar cuántas veces se ha utilizado una determinada palabra en el campeonato.
- Averiguar qué palabra ha obtenido la puntuación más alta en el campeonato.

Cambios respecto la PRIMERA PARTE:

- Crear una nueva partida, la partida tendrá un nombre que la identificará y, en el momento de crearla, se especificará el NIF de los dos jugadores que la jugarán y el campeonato en el que participarán. Los jugadores tienen que estar inscritos al campeonato.
- Hacer una jugada en una partida de un campeonato. La partida no puede estar finalizada. Hay que especificar el jugador que hace la jugada, la fila y la columna donde empieza la palabra, la orientación (horizontal- de izquierda a derecha- o vertical – de arriba a abajo-) y las letras necesarias para hacer la palabra. Hay que comprobar que la palabra sea válida, que el jugador tiene todas las fichas necesarias para hacerla, calcular la puntuación de la jugada y actualizar las letras disponibles del jugador y su puntuación. Para actualizar la puntuación hay que tener en cuenta los puntos de las palabras generadas perpendicularmente a la nueva palabra añadida. NOTA: ver las aclaraciones al final de este documento.

Nuevo en la SEGUNDA PARTE:

- Dar de alta un jugador al sistema a partir de su NIF y nombre. En caso de existir otro jugador con el mismo NIF, modificaremos sus datos.
- Inscribir un jugador a un campeonato a partir de su NIF. El jugador tiene que estar dado de alta al sistema. Si el jugador ya estaba inscrito, eliminaremos su inscripción. Las inscripciones del campeonato tienen que estar abiertas.
- Cerrar la inscripción de un campeonato.

- Consultar las fichas de un jugador, en el momento actual de una partida de un campeonato.
- Cambiar una ficha de un jugador de una partida de un campeonato.
- Finalizar una partida de un campeonato. Uno de los dos jugadores tiene que haberse quedado sin fichas.
- Consultar los puntos de los dos jugadores de una partida de un campeonato. Esta consulta se puede hacer en cualquier momento de la partida, finalizada o no.
- Consultar la clasificación de un campeonato ordenando los jugadores inscritos por el número de partidas ganadas y los puntos obtenidos (para desempatar jugadores que hayan ganado el mismo número de partidas). No se tendrán en cuenta los puntos de las partidas no finalizadas. En caso de empate no importa el orden. Esta operación tiene que ser especialmente rápida.

Apartado a)

Especifica la firma del TAD *Apalabrados* del los nuevos métodos de la SEGUNDA PARTE. Es decir, indica el nombre que darías a las operaciones encargadas de cada funcionalidad requerida. Indica, también, los parámetros de entrada y de salida cuando sean necesarios.

Apartado b)

Haz la especificación contractual de las operaciones del TAD *Apalabrados*, modificadas o nuevas en la SEGUNDA PARTE. En la redacción de la especificación puedes usar, si se requiere, cualquiera de las operaciones del TAD. Toma como modelo la especificación del apartado 1.2.3 del Módulo 1 de los materiales docentes. Se valorará especialmente la concisión (ausencia de elementos redundantes o innecesarios), precisión (definición correcta del resultado de las operaciones), completitud (consideración de todos los casos posibles en que se puede ejecutar cada operación) y carencia de ambigüedades (conocimiento exacto de cómo se comporta cada operación en todos los casos posibles) de la solución. Es importante responder este apartado usando una descripción condicional y no procedimental. La experiencia nos demuestra que no siempre resulta fácil distinguir entre las dos descripciones, es por eso que hacemos especial énfasis insistiendo que pongáis mucha atención en vuestras definiciones.

A título de ejemplo indicaremos que la descripción condicional (la correcta a utilizar en el contrato) de llenar un vaso vacío con agua sería:

> @pre el vaso se encuentra vacío.
>
> @post el vaso está lleno de agua.

En cambio una descripción procedimental (incorrecta para utilizar en el contrato) tendría una forma parecida a:

> @pre el vaso tendría que encontrarse vacío, si no se encontrase vacío se tendría que vaciar.
>
> @post se acerca el vaso al grifo y se echa agua hasta que esté lleno.

También debéis tener en cuenta el invariante si éste es necesario para describir el TAD.

Ejercicio 6

En el Ejercicio 5 habéis definido la especificación de un nuevo TAD, el TAD Apalabrados, ahora os pedimos que hagáis el diseño de las estructuras de datos que formarán este TAD. Diseñad, pues, el sistema para que sea el máximo de eficiente posible, tanto a nivel de eficiencia espacial como temporal, teniendo en cuenta los volúmenes de información y las restricciones especificadas en el enunciado.

Tened en cuenta sólo las operaciones que se piden en el enunciado al hacer este diseño.

Apartado a)

Dudamos entre utilizar una lista encadenada, una tabla de dispersión o un AVL para almacenar los jugadores inscritos a un campeonato. Justificad cuál creéis que es la mejor opción.

Apartado b)

Dudamos entre utilizar un vector, una lista encadenada o un AVL para almacenar las partidas que ha hecho un jugador. Justificad cuál creéis que es la mejor opción.

Apartado c)

Dudamos entre utilizar un vector ordenado, una lista encadenada ordenada o una tabla de dispersión para almacenar las palabras del diccionario de un campeonato. Justificad cuál creéis que es la mejor opción.

Apartado d)

Dudamos entre utilizar una cola, una tabla de dispersión o un AVL para almacenar las partidas de un campeonato. Justificad cuál creéis que es la mejor opción.

Apartado e)

Dudamos entre utilizar un vector ordenado, una lista encadenada ordenada o un AVL para almacenar la clasificación de un campeonato. Justificad cuál creéis que es la mejor opción.

Apartado f)

Haced un dibujo de la estructura de datos global para el TAD *Apalabrados* donde se vean claramente las estructuras de datos que elegís para representar cada una de las partes y las relaciones entre ellas. Haced el dibujo de la estructura completa, con todas los estructuras que os permitan implementar las operaciones definidas en la especificación.

Ejercicio 7

En el Ejercicio 5 habéis definido la especificación del TAD *Apalabrados* con sus operaciones y en el Ejercicio 6 habéis elegido las estructuras de datos para cada parte del TAD. En este ejercicio os pedimos que os fijéis en los algoritmos que os servirán para implementar algunas de las operaciones especificadas y en el estudio de eficiencia de las mismas. Tened en cuenta que la implementación de las operaciones va estrechamente ligada a la elección de las estructuras de datos que hayáis hecho.

Apartado a)

Describid y haced el estudio de eficiencia de la operación que hayáis definido para hacer una jugada en una partida. Para hacerlo tenéis que describir brevemente su comportamiento indicando los pasos que la componen (con frases como por ejemplo: "insertar en el árbol AVL / borrar de la tabla de dispersión / consulta del pilón / ordenar el vector..."), especificando la eficiencia asintótica de cada paso y dando la eficiencia total de la operación. No hay que hacer el pseudocódigo, solo describir los pasos.

Apartado b)

Describid y haced el estudio de eficiencia de la operación que hayáis definido para finalizar una partida y obtener la clasificación de un campeonato. Para hacerlo tenéis que describir brevemente su comportamiento indicando los pasos que la componen (con frases como por ejemplo: "insertar en el árbol AVL / borrar de la tabla de dispersión / consulta del pilón / ordenar el vector..."), especificando la eficiencia asintótica de cada paso y dando la eficiencia total de la operación. No hay que hacer el pseudocódigo, solo describir los pasos.

Apartado c)

Describid y haced el estudio de eficiencia de la operación que hayáis definido para cambiar una ficha de un jugador. Para hacerlo tenéis que describir brevemente su comportamiento indicando los pasos que la componen (con frases como por ejemplo: "insertar en el árbol AVL / borrar de la tabla de dispersión / consulta del pilón / ordenar el vector..."), especificando la eficiencia asintótica de cada paso y dando la eficiencia total de la operación. No hay que hacer el pseudocódigo, solo describir los pasos.

Ejercicio 8

Indica cual de los TADs de la biblioteca de TADs de la asignatura te parecen más adecuados para utilizarlos en la implementación de cada una de las estructuras de datos definidas para el TAD *Apalabrados*.

Aclaración: Suponiendo que el tablero se encuentra en la situación indicada a la izquierda, el jugador puede jugar "CRIE" desde la posición 2,3 en vertical, pero solo necesita las letras "C" y "I", al aprovechar las letras "R" y "E" que ya están en el tablero. Es decir, hay que comprobar que "CRIE" (la palabra jugada) es una palabra válida y que el jugador tiene las letras "C" y "I" (las que necesita para completar la palabra que quiere jugar). En este caso el jugador solo hace una palabra.

		R	O	S	A
				A	
I	D	E	A	L	

		C			
		R	O	S	A
		I		A	
I	D	E	A	L	

Si ahora el otro jugador juega "LAS" en la posición 6,4 horizontal, de hecho está generando tres palabras nuevas, la ya mencionada "LAS" pero también "AL" y "SALA", al completar "A" y "SAL" respectivamente. Observar que la palabra "SAL" podía ser una palabra jugada anteriormente, mientras que "A" no era una palabra sino una letra que forma parte de otra palabra jugada (en este caso, "IDEAL").

		C			
		R	O	S	A
		I		A	
I	D	E	A	L	

		C			
		R	O	S	A
		I		A	
I	D	E	A	L	
			L	A	S

Finalmente, recordar que es necesario tener en cuenta si una o más fichas caen en una casilla que multiplica su valor y/o el de las palabras resultantes.

SOLUCIONES PRIMERA PARTE

Ejercicio 1

apartado a)

void crearCampeonato(String idCampeonato)

void anadirPalabra(String idCampeonato, String palabra)

void anadirLletra(String idCampeonato, String letra, int puntos, int numFichas)

void crearPartida(String idCampeonato, String nombre, String primerJugador, String segundoJugador)

void anadirFicha(String idCampeonato, String nombrePartida, String letra)

void empezarPartida(String idCampeonato, String nombrePartida)

void hacerJugada(String idCampeonato, String nombrePartida, int jugador, int fila, int columna, int orientacion, String palabra)

int numeroVeces(String idCampeonato, String palabra)

String palabraConPuntuacionMasAlta(String idCampeonato)

apartado b)

@pre no existe ningún campeonato con el identificador especificado

@post existe un campeonato vacío con el identificador especificado

void crearCampeonato(String idCampeonato)

@pre existe el campeonato y su diccionario de palabras no contiene la palabra especificada

@post el diccionario de palabras del campeonato contiene la palabra especificada

void anadirPalabra(String idCampeonato, String palabra)

@pre existe el campeonato, su lista de letras no contiene la letra especificada y la suma total de fichas no supera las 100

@post la lista de letras contiene la letra especificada con la puntuación y el número de fichas. La nueva ficha se añade detrás

void anadirLletra(String idCampeonato, String letra, int puntos, int numFichas)

@pre existe el campeonato, no existe ninguna partida con el nombre especificado

@post existe una partida no comenzada entre el primer jugador y el segundo jugador

void crearPartida(String idCampeonato, String nombre, String primerJugador, String segundoJugador)

@pre existe la partida, no ha comenzado y el número de fichas añadidas para la letra especificada es

inferior al máximo permitido

@post ha incrementado en una unidad el número de fichas disponibles en la partida para la letra especificada

void anadirFicha(String idCampeonato, String nombrePartida, String letra)

@pre existe el campeonato y la partida y ésta no está empezada

@post la partida está empezada y cada jugador tiene las siete fichas iniciales

void empezarPartida(String idCampeonato, String nombrePartida)

@pre existe el campeonato y la partida y esta empezada.

@post se ha registrado la jugada del jugador en el panel de la partida, se han quitado las fichas de las letras jugadas al jugador, se le ha actualizado la puntuación y se le han repuesto las fichas

void hacerJugada(String idCampeonato, String nombrePartida, int jugador, int fila, int columna, int orientacion, String palabra)

Nota: También sería correcto especificar como precondición las condiciones que hacen que una jugada sea correcta. En nuestro caso hemos considerado que estas condiciones se comprueban en la operación, por tanto, no aparecen en la precondición

@pre existe el campeonato

@post el valor devuelto es el número de veces que la palabra ha salido durante el campeonato

int numeroVeces(String idCampeonato, String palabra)

@pre existe el campeonato

@post el valor devuelto es la palabra con puntuación más alta en el campeonato

String palabraConPuntuacionMasAlta(String idCampeonato)

Nota: En todo el ejercicio hemos especificado como precondiciones algunas comprobaciones que se podrían hacer dentro de las operaciones. También sería correcto considerar que las comprobaciones las hace la misma operación, en este caso, las precondiciones no deberían aparecer y las postcondiciones deberían ampliarse (o debilitarse) para definir el estado cuando las condiciones fallan.

Ejercicio 2

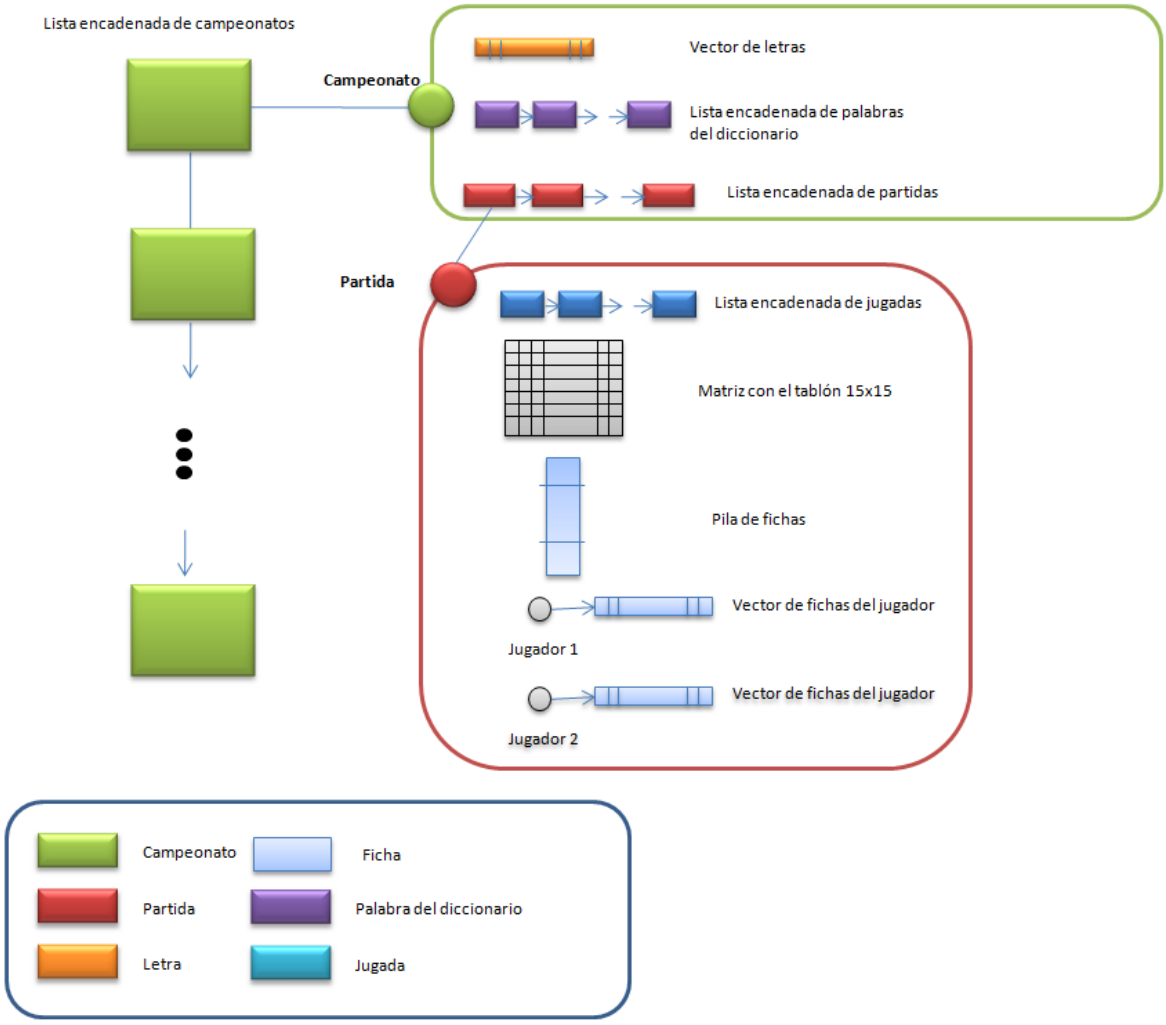
Apartado a)

El enunciado nos dice que el número de jugadas GP de una partida es variable y relativamente pequeño, para volúmenes de datos pequeños y variables, la mejor opción es usar una estrategia basada en memoria dinámica para no desperdiciar espacio. Como en el enunciado no nos piden ninguna operación que requiera la lista de jugadas ordenada elegiremos una lista encadenada para guardar las jugadas de una partida.

Apartado b)

El enunciado nos dice que el número de fichas F disponibles para una partida es fijo y pequeño: 100 fichas. Para volúmenes de datos pequeños y fijos, la mejor opción es usar una estrategia basada en vector para no desperdiciar espacio con los apuntadores. Para el reparto inicial de fichas el enunciado nos dice que utilizamos una estrategia basada en una pila, no se especifica la estrategia que se utilizará para ir repartiendo las fichas a los jugadores después de cada jugada así que podemos elegir una pila implementada con un vector para guardar las fichas de una partida.

Apartado c)



Apartado d)

- Para guardar los campeonatos elegimos una lista encadenada ya que el número de campeonatos es variable y relativamente pequeño, de unos centenares. Aunque necesitamos acceso directo el número de elementos es bastante pequeño como para admitir búsquedas lineales.
- Para las partidas también elegimos una lista encadenada, el razonamiento es el mismo que para los campeonatos.
- Para las palabras del diccionario también elegimos una lista encadenada, el razonamiento es el mismo que para los campeonatos y las partidas.
- Para las letras elegimos un vector ya que el número es pequeño pero no es variable. También necesitaremos acceso directo pero con 29 elementos podemos permitir las búsquedas lineales
- Para las fichas de una partida elegimos una pila implementada con un vector ya que cada

partida tendrá 100 fichas y necesitamos una estructura basada en una pila para repartir las fichas iniciales y las fichas de cada jugada. En este caso no necesitamos acceso directo.

- Para los dos jugadores de una partida no necesitamos ninguna estructura adicional ya que el enunciado no especifica ninguna operación que involucre a todos los jugadores de un campeonato, se guardarán pues como atributos de cada partida.
- Las letras que tiene cada jugador en un momento determinado es de 7, usaremos pues un vector con las letras disponibles de cada jugador. Al igual que antes, aunque se requiere acceso directo, con 7 elementos nos podemos permitir búsquedas lineales.
- Para guardar el tablero elegiremos una matriz de 15x15.
- Para guardar las jugadas de una partida elegimos una lista encadenada ya que el número de jugadas es variable y relativamente pequeño. No necesitamos acceso directo pero si un recorrido para contar el número de veces que aparece una palabra.

Ejercicio 3

Apartado a)

Simplificándolo mucho, la operación *hacerJugada* debería:

- Comprobar que la palabra exista en el diccionario => O (PA)
- Recuperar el campeonato en la lista de campeonatos => O (C)
- Recuperar la partida de la lista de partidas del campeonato => O (P)
- Comprobar cuáles de las letras de la palabra ya están en el tablero => O (1)
- Comprobar que el jugador tiene todas las fichas con las letras que faltan para completar la palabra => O (1)
- Poner las fichas en el tablero => O (1)
- Contar los puntos de la palabra, consultando la puntuación de cada ficha en el vector de letras => O (1)
- Actualizar, si es necesario, la palabra con puntuación más alta del campeonato => O (1)
- Añadir al vector de fichas del jugador tantas fichas como haya puesto en el tablero. Las fichas se toman del la pila de fichas => O (1)

Por tanto, el coste total de la operación sería de O (PA + C + P)

Apartado b)

Si no contemplamos un atributo que guarde la palabra con la puntuación más alta a la hora de hacer una jugada la operación sería muy costosa ya que debería iterar por todas las jugadas de todas las partidas de todos los campeonatos calculando la puntuación de las palabras de cada jugada hasta encontrar la que hubiera proporcionado una puntuación más elevada.

Para la prueba piloto que os plantea la PEC1 aún podría ser admisible pero cuando el volumen de datos creciera un poco deberíais replantear la operación ya que sería muy ineficiente.

Parece mucho más adecuado guardar a nivel de campeonato un atributo con la palabra con más puntuación e ir actualizando este atributo en la operación *hacerJugada*, con esta estrategia el coste de la operación de averiguar la palabra con puntuación más elevada se reduce a consultar este atributo O (1).

Apartado c)

Con el fin de reproducir las partidas se deben guardar las jugadas de forma ordenada, por lo tanto, una posible solución sería cambiar la lista encadenada de jugadas por una lista encadenada ordenada según el número de jugada. Si no teníais el número de jugada especificado en la operación *hacerJugada* también deberíais añadirlo (o guardar un contador interno) para que las jugadas se puedan ordenar según este criterio.

Ejercicio 4

- Para guardar los campeonatos elegimos una lista encadenada, el TAD más adecuado sería una *ListaEncadenada*
- Para las partidas también elegimos una *ListaEncadenada*.
- Por las palabras del diccionario también elegimos una *ListaEncadenada*.
- Para las letras elegimos un array de Java ya que en la biblioteca de TADs no hay ninguna implementación de contenedor secuencial acotado.
- Para las fichas de una partida elegimos una *PilaVectorImpl*.
- Para los dos jugadores de una partida no necesitan ninguna estructura adicional.
- Para las letras que tiene cada jugador elegimos una array de Java.
- Para guardar el tablero elegiremos un array de Java bidimensional.
- Para guardar las jugadas de una partida elegimos una *ListaEncadenada*

SOLUCIONES SEGUNDA PARTE

Ejercicio 5

Apartado a)

void addPlayer(String NIF, String nombre);
void addOrRemoveInscription(String idChampionship, String NIF);
void closeInscriptions(String isChampionship);
Iterator<Card> cards(String idChampionship, String gameId, int playerNumber);
void changeCard(String idChampionship, String gameId, int playerNumber, Card card); void
finishGame(String idChampionship, String gameId);
Points[] getPoints(String idChampionship, String gameId);
Iterator<Player> classification(String idChampionship);

Apartado b)

@pre cierto.
@post los jugadores registrados al sistema serán los mismos que antes de la operación más el jugador nuevo, o el jugador con nif NIF tendrá los datos cambiados.
void addPlayer(String NIF, String nombre);

@pre el campeonato existe, la preinscripción está abierta, y hay un jugador con NIF registrado al sistema.
@post si el jugador con nif NIF no estaba inscrito al campeonato, ahora lo estarà. Por el contrario, si ya estaba inscrito ahora no lo estará.
void addOrRemoveInscription(String idChampionship, String NIF);

@pre el campeonato existe y la inscripción está abierta.
@post el campeonato tiene la inscripción cerrada.
void closeInscriptions(String isChampionship);

@pre el campeonato y el juego existen, y playerNumber vale 0 o 1.
@post devuelve las fichas del jugador de la partida del campeonato indicados.
Iterator<Card> cards(String idChampionship, String gameId, int playerNumber);

@pre el campeonato y el juego existen, playerNumber vale 0 o 1, y card es una de las fichas del

jugador.

@post el jugador tendrá las mismas fichas excepto card y, en su lugar, tendrá una de las fichas pendientes de adjudicar del juego X. Las fichas pendientes de adjudicar del juego serán las mismas exceptuando X que se habrá cambiado per card.

void changeCard(String idChampionship, String gameId, int playerNumber, Card card);

@pre el campeonato y el juego existen y no quedan fichas pendientes de adjudicar y, o bien el jugador 1 o el jugador2, no tienen fichas.

@post el juego está finalizado.

void finishGame(String idChampionship, String gameId);

@pre el campeonato y el juego existen.

@post devuelve los puntos de los dos jugadores.

Points[] getPoints(String idChampionship, String gameId);

@pre el campeonato existe.

@post devuelve un iterador para recorrer los jugadores del campeonato ordenados por los puntos obtenidos. En caso de empate no importa el orden.

Iterator<Player> classification(String idChampionship);

Ejercicio 6

Apartado a)

El número de jugadores inscritos a un campeonato es muy grande y variable. Por eso necesitamos una estructura no acotada y, por tanto, descartamos las tablas de dispersión. Una lista encadenada sería una opción válida, pero comportaría costes lineales cada vez que necesitásemos verificar que un jugador esta inscrito a un campeonato (básicamente a addGame y addOrRemoveInscription). Por eso nos decantamos por un AVL.

Apartado b)

Esta información no es necesaria para poder implementar ningún de los métodos especificados al enunciado. Por eso no guardaremos nada al respecto.

Apartado c)

El número de palabras será grande y conocido. Podemos utilizar cualquiera de las estructuras propuestas, pero una lista encadenada ordenada tendría costes lineales cada vez que necesitásemos hacer una consulta para comprobar la existencia de una palabra. Un vector ordenado tendría costes logarítmicos y una tabla de dispersión costes constantes. La mejor opción es la tabla de dispersión.

Apartado d)

Una cola solo nos permite acceder a la primera partida añadida, y muchos métodos necesitan acceder a las partidas a partir de su id. El número de partidas es grande y variable y esto nos impide utilizar estructuras acotadas como las tablas de dispersión. Por eso, la única opción valida será un AVL.

Apartado e)

El número de jugadores de un campeonato es grande y variable, pero una vez ha empezado el campeonato la inscripción se cierra y, por tanto, el número de jugadores pasa a ser grande y constante. Por tanto, por el volumen de datos a guardar, todas las opciones son válidas. Para poder obtener un iterador de la clasificación hay que tener los jugadores ordenados por sus puntos. Esto no es posible utilizando los AVL implementados en la biblioteca de EI, ja que no permiten claves repetidas.

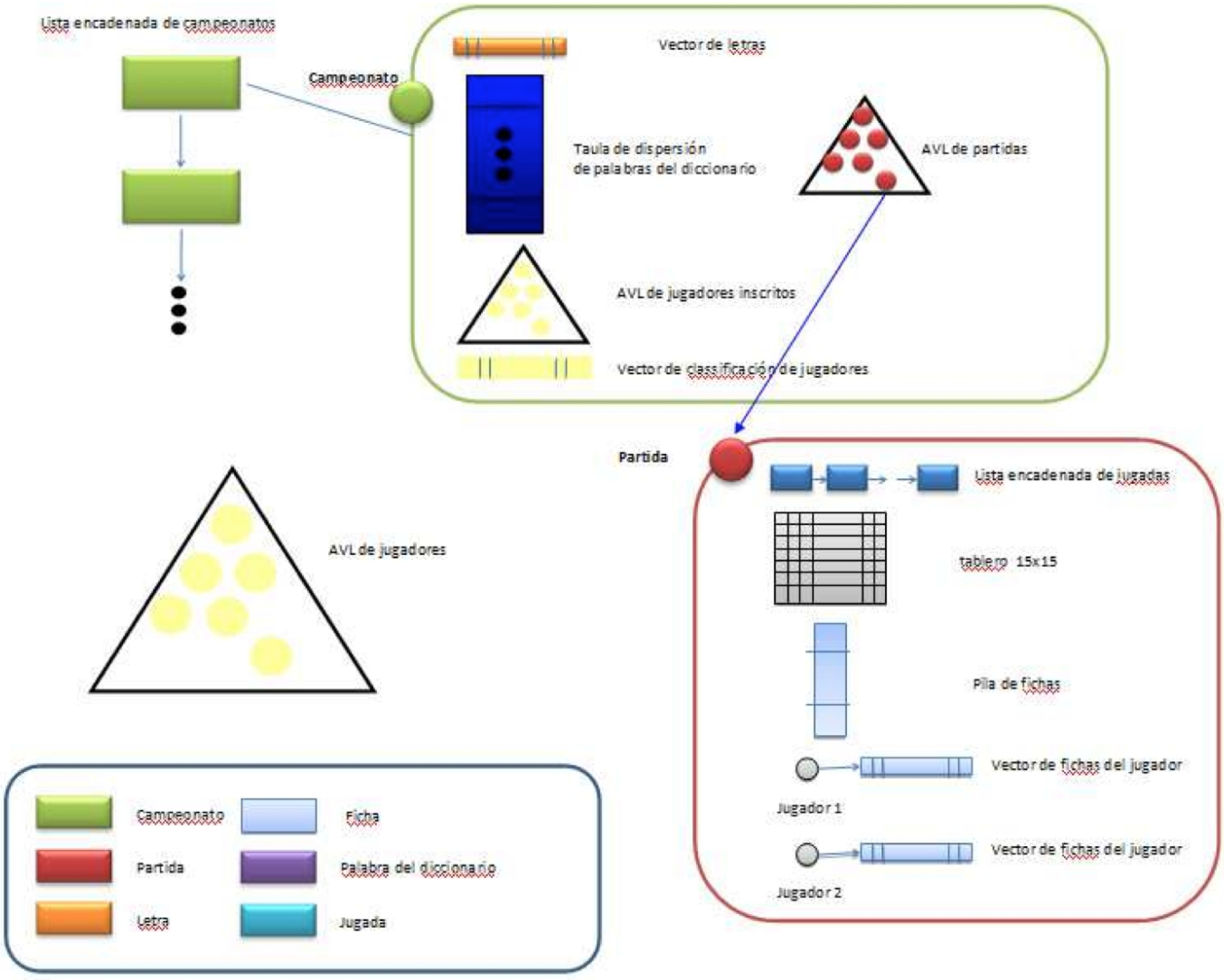
Para evitar esta limitación, podemos definir la clave como los puntos + NIF. De esta manera mantendremos el orden y evitaremos las claves repetidas.

Las listas encadenadas ordenadas tendrían costes lineales para buscar un jugador, mientras que un vector ordenado o un AVL serían logarítmicos.

Para actualizar la posición de un jugador en un AVL hay que eliminarlo, sumar los puntos, y volverlo a

añadir => O(log l). Si utilizamos un vector ordenado, en el peor de los casos hay que mover todos los jugadores 1 celda (cuando el último jugador de la clasificación gane tantos puntos que pase a ser el primero). Pero esta situación será poco habitual, y solo podrá pasar en el inicio de la partida, cuando todos los jugadores tengan 0 puntos. De hecho, normalmente los movimientos que haran los jugadores dentro de la clasificación serán pequeños y, por tanto, el coste de utilizar vectores ordenados será inferior a los AVL.

Apartado f)



Ejercicio 7

Apartado a)

- Comprobar que la palabra exista en el diccionario => O(1)
- Recuperar el campeonato en la lista de campeonatos => O(C)
- Recuperar la partida de la lista de partidas del campeonato => O(log P)
- Comprobar que letras de la palabra ya están al tablero => O(1)
- Comprobar que el jugador tiene todas las fichas con las letras que falten para completar la palabra => O(1)
- Poner las fichas en el tablero => O(1)
- Contar los puntos de la palabra, consultando la puntuación de cada ficha descrita en el vector de letras => O(1)
- Por cada ficha nueva comprobar si se forma una nueva palabra perpendicularmente.
  - Buscar la palabra perpendicular al tablero => O(1)
  - Comprobar si la palabra existe al diccionario => O(1)
  - Contar los puntos => O(1)
  - Total: O(1) * por el número de fichas añadidas => O(1) (consideremos el número de fichas añadidas un número suficientemente pequeño comparado con los otros).
- Actualizar, si cale, la palabra con puntuación más alta del campeonato => O(1)
- Añadir al vector de fichas del jugador tantas fichas como haya añadido al tablero. Las fichas se toman de la pila de fichas => O(1)

Por tanto, el coste total de la operación sería de O(C+log P)

Apartado b)

El enunciado nos pide que la operación para obtener la clasificación sea lo más eficiente posible. Para conseguir coste O(1) siempre tenemos que tener la clasificación ordenada, y que el trabajo lo hagamos en los métodos que modifican la clasificación.

El único método que altera la clasificación es la finalización de una partida. Por tanto, será en este método donde sumaremos los puntos al jugador.

void finishGame(String idChampionship, String gameId);

- Buscar campeonato => O(C)
- Buscar partida =>O(log P)
- Por cada jugador:

- Acceder a los datos del jugador => O(1) (si tenemos una referencia al objeto jugador desde partida)
- Consultar sus puntos (atributodeJugador) => O(1)
- Localizar el jugador en la clasificación (clave: puntos+NIF) => O(logI)
- Actualizar los puntos del jugador => O(1)
- Mover el jugador en la clasificación =>O(1) (ver explicación en el ejercicio 6e)

Por tanto, el coste total de la operación sería de O(C+log P+log I)

Apartado c)

void changeCard(String idChampionship, String gameId, int playerNumber, Card card);

- Buscar campeonato => O(C)
- Buscar partida =>O(log P)
- Buscar card al vector de fichas del jugador => O(7) = O(1)
- Sortear una ficha de la pila de fichas =>O(1) (implica añadir un nuevo método a la Pila!)
- Intercambiar card por la ficha sorteada =>O(1)

Per tanto, el coste total de la operación sería de O(C+log P+log I)

Ejercicio 8

- Para guardar los campeonatos elegimos una lista, el mrjor TAD es una *LlistaEncadenada*.
- Para las partidas y las inscripciones elegimos un *DiccionarioAVLImpl*
- Para las palabras del diccionario elegimos un *TablaDispersion.*
- Para las letras elegimos la clase array de Java, ya que a la biblioteca de TADs no hay ninguna implementación de contenedor secuencial acotado.
- Para la clasificación un *DiccionarioVectorImpl*.
- Para las fichas de una partida elegimos una *PilaVectorImpl* ampliada para poder cambiar las fichas.
- Para los dos jugadores de una partida no necesitamos ninguna estructura adicional.
- Para las letras que tiene cada jugador elegimos un array de Java.
- Para guardar el tablero elegimos un array de Java bidimensional.
- Para guardar las jugadas de una partida elegimos una *ListaEncadenada*.