Tarea 4
TADs Tabla y Cola de Prioridad
Curso 2024

1. Introducción

Esta tarea tiene como principales objetivos:

- Continuar trabajando sobre el manejo dinámico de memoria.
- Continuar trabajando con el concepto de tipo abstracto de datos (TADs).
- Continuar trabajando en el uso de TADs como auxiliares para la resolución de problemas.
- Trabajar en la implementación de TADs a partir de su especificación y utilizando nuevas estructuras de datos (Hash y Heap).

La fecha límite de entrega es el **lunes 1ero de julio a las 16:00 horas**. El mecanismo específico de entrega se explica en la Sección 7. Por otro lado, para plantear **dudas específicas de cada paso** de la tarea, se deja un link a un **foro de dudas** al final de cada parte.

2. Descripción de las funcionalidades a implementar

En esta tarea se construirá sobre las bases asentadas en las anteriores tareas, completando las funcionalidades de la galería. El mayor agregado se relaciona con el manejo de la galería de los grupos que ingresan y su procesamiento histórico.
La galería manejará entonces una referencia a visitaDia (de la tarea anterior) para almacenar los grupos que ingresan en la fecha actual, así como un hash (5) de visitasDia para almacenar los registros históricos.
Adicionalmente, ya que los administradores de la galería desean fomentar el arte entre las nuevas generaciones, el módulo visitaDia (4) será modificado para brindar funciones asociadas al TAD Cola de Prioridad, con el objetivo de encontrar eficientemente aquellos grupos con menor promedio de edad y poder premiarlos.
Por último, se modificará el módulo visitante (3) para agregar las piezas favoritas de cada persona. Con esta información, la galería podrá sacar estadísticas de cuáles piezas favoritas son vistas por cada visitante a fin de mejorar la satisfacción en las visitas.
A continuación se presenta una **guía** que deberá **seguir paso a paso** para resolver la tarea. Tenga en cuenta que la especificación de cada función se encuentra en el **.h** respectivo, y para la mayoría de las funciones se especifica cuál debe ser el orden del tiempo de ejecución en el **peor caso** o el **caso promedio**.

3. Partiendo de la tarea 3

En esta sección se describen una serie de cambios que se han realizado en varios módulos de la tarea anterior.

1. En el módulo *Visitante*, **Modifique** la estructura rep_visitante para incorporar las piezas favoritas del visitante. Esto se debe representar con un campo de tipo TConjuntoPiezas.

   Modifique las funciones crearTVisitante y liberarTVisitante para que creen y liberen el conjunto de piezas respectivamente. Utilice la constante MAX_PIEZAS definida en conjuntoPiezas.h como parámetro en el constructor de TConjuntoPiezas.

   Ejecute el test regresion1-visitante para verificar que los cambios no afectan el código anterior en el módulo.

   Foro de dudas.

2. En el módulo *ConjuntoPiezas*, **Copie** las implementaciones de las funciones de la tarea anterior (no varían en esta tarea). Ejecute el test regresion2-conjuntoPiezas para verificar que las funciones de la tarea anterior continúan funcionando correctamente. Foro de dudas.

3. En el módulo *Visitante*, **Implemente** las funciones agregarPiezaFavoritaTVisitante y obtenerPiezasFavoritasTVisitante. Verifique el funcionamiento de las funciones ejecutando el test visitante-agregar-obtener-pieza para verificar el correcto funcionamiento de las funciones. Foro de dudas.

4. En este ítem y en el siguiente se trabajará en el módulo **Exposición**. Agregue sus funciones de la tarea anterior y ejecute el test regresion3-exposicion para verificar que las funciones de la tarea anterior continúan funcionando correctamente. Foro de dudas.

5. **Implemente** la función obtenerPiezasTExposicion. Verifique el funcionamiento de la función ejecutando el test exposicion-obtenerPiezas-compatibles. Nota: este test también verifica de forma adicional la función sonExposicionesCompatibles. Foro de dudas.

6. En este ítem y en el siguiente se trabajará sobre el módulo **ListaExposición**. Agregue sus funciones de la tarea anterior ejecute el test regresion4-listaExposicion para verificar que las funciones de la tarea anterior continúan funcionando correctamente. Foro de dudas.

7. **Implemente** las funciones cantidadExposicionesTListaExposiciones y obtenerNesimaExposicionTListaExposiciones. **Ejecute** el caso de prueba listaExposicion-cantidad-enesima para verificar el correcto funcionamiento de las funciones. Foro de dudas.

4. Módulo Visita Día (TAD Cola de Prioridad)

En esta sección se implementará en visitaDia el TAD Cola de Prioridad a partir de la especificación dada en visitaDia.h. Las funciones provistas son en su mayoría similares o las mismas de la tarea anterior, **pero la estructura interna del módulo deberá cambiar drásticamente** para poder satisfacer las nuevas funciones así como los requerimientos de tiempo solicitados.
La visitaDia sigue manteniento elementos de tipo grupoABB, pero el módulo provisto grupoABB ahora incluye un id, por lo cual es posible identificar de forma sencilla a cada grupo. No existirán dos grupos en la visitaDia con el mismo ID. Se asume que los IDs de los grupos están acotados entre 1 y N, siendo N el tamaño máximo de la cola.
Como se mencionó anteriormente, se implementará una prioridad para obtener eficientemente el grupo más prioritario. El criterio para establecer la prioridad entre grupos es, de manera predeterminada, que un grupo es prioritario ante otro si la edad promedio del grupo es menor que la del otro grupo. Este criterio se puede modificar (con la función invertirPrioridad), logrando que el grupo prioritario sea el de mayor edad promedio. Asuma que no hay grupos con la misma edad promedio.
Si la visitaDia no es vacía hay un grupo considerado el prioritario, según el criterio de prioridad. Para esta implementación se recomienda utilizar la estructura de Heap (montículo binario) vista en el curso. Además, podrá ser necesario considerar estructuras auxiliares para cumplir con los ordenes de tiempo de ejecución de algunas operaciones.

1. **Implemente** la estructura *rep_visitadia*. Para esto, recomendamos revisar las operaciones del TAD solicitadas y realizar un diseño de la/s estructura/s necesaria/s. Recuerde que no es necesario mantener los campos de la tarea anterior. Foro de dudas.

2. **Implemente** las funciones crearTVisitaDia, liberarTVisitaDia, estaEnTVisitaDia, fechaTVisitaDia y encolarGrupoTVisitaDia. Recomendamos que declare e implemente la función auxiliar *void filtradoAscendente(int pos, TVisitaDia &visita)* que realiza el filtrado ascendente en el heap. **Ejecute** el test visitaDia1-crear-liberar-esta-encolar para verificar las funciones. Foro de dudas.

3. **Implemente** las funciones prioridadTVisitaDia y masPrioritarioTVisitaDia. **Ejecute** el test visitaDia2-prioritario-prioridad para verificar las funciones. Foro de dudas.

4. **Implemente** la función desencolarGrupoTVisitaDia. Recomendamos que implemente la función auxiliar *void filtradoDescendente(int pos, TVisitaDia &visita)* que realiza el filtrado descendente en el heap. **Ejecute** el test visitaDia3-desencolar para verificar las funciones. Foro de dudas.

5. **Implemente** la función invertirPrioridad y ejecute el test visitaDia4-invertirPrioridad. La función debe modificar la cola de forma de que se respete el nuevo criterio de prioridad. Se pide que el tiempo de ejecución en el peor caso sea $O(n \log n)$, siendo n la cantidad de elementos de la visitaDia. Sin embargo, existe una solución que lo hace en $O(n)$. Foro de dudas.

6. **Ejecute** el test visitaDia5-combinado Foro de dudas.

7. **Ejecute** el test visitaDia6-tiempo para verificar que su implementación cumple con los tiempos de ejecución solicitados. Foro de dudas.

5. Módulo Hash Visita Día (TAD Tabla)

En esta sección se implementará el módulo *hashVisitaDia.cpp*. Este TAD se utilizará para almacenar históricos de visitaDia, y consiste en una tabla no acotada cuyo dominio son las *fechas* de las visitaDia y el codominio son elementos del tipo *TVisitaDia*. La tabla debe ser implementada mediante una **tabla de dispersión abierta**.

1. **Implemente** la estructura *rep_hashvisitadia*. Para esto, recomendamos revisar las operaciones del TAD solicitadas y realizar un diseño de la/s estructura/s necesaria/s. La representación debe implementar listas del tipo *TVisitaDia*, a definirse internamente en el módulo. Foro de dudas.

2. **Implemente** las funciones crearTHashVisitaDia y liberarTHashVisitaDia. Ejecute el test hash1- crear-liberar para verificar el funcionamiento de la funciones. Foro de dudas.

3. **Implemente** las funciones agregarVisitaDiaTHashVisitaDia e imprimirTHashVisitaDia. La función de inserción recibe un THashVisitaDia y un TVisitaDia, y asocia en la tabla al TVisitaDia con su fecha. Para calcular la posición en la que se debe insertar a la TVisitaDia en la tabla de dispersión abierta se debe utilizar la función brindada **funcionHash** (la función está declarada al comienzo de hashVisitaDia.cpp). La visitaDia debe ser ubicada en la posición de la tabla indicada por dicha función. Por convención, se deberá insertar la visitaDia al inicio de la lista definida para dicha posición de la tabla. La función imprimirTHashVisitaDia debe imprimir cada visitaDia de la tabla, en orden creciente de posiciones asociadas en la tabla. En caso de que haya más de un jugador en la misma posición, se deben imprimir en el orden inverso al que fueron agregados (que será el orden natural en que se recorrerá la lista). En el archivo hashVisitaDia.h se encuentra una descripción más detallada del formato de impresión. Ejecute el test hash2-insertar-imprimir para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones perteneceVisitaDiaTHashVisitaDia y obtenerVisitaDiaTHashVisitaDia. Ejecute el test hash3-pertenece-obtener. Foro de dudas.

5. **Ejecute** el test hash4-combinado. Foro de dudas.

6. **Ejecute** el test hash5-tiempo.
   Foro de dudas.

6. Módulo Galería

En esta sección se extenderá el módulo *galería* con funciones adicionales, para esto, primero se deberá extender la estructura *rep_galeria*. La Galería almacenará, además de los campos incluídos en la tarea 3 (Fecha actual, colección de piezas disponibles en la galería y exposiciones finalizadas, activas y futuras), nuevos campos para manejar las visitas de cada día. La galería deberá mantener una referencia a un elemento de tipo TVisitaDia para almacenar los grupos que visitan la galería *en el día actual*, así como una referencia a un elemento de tipo THashVisitaDia para almacenar las visitasDia para fechas pasadas.

1. **Modifique** la representación de galeria *rep_galeria* según la descripción brindada en la sección anterior. Modifique las funciones crearTGaleria y liberarTGaleria para que inicialicen y liberen las estructuras adicionales. Ejecute el test regresion5-galeria para verificar que el funcionamiento de la galería permanence incambiado. Foro de dudas.

2. **Modifique** la función avanzarAFechaTGaleria, agregando la funcionalidad de que al avanzar de fecha, la visitaDia actual se guarde en el histórico de visitas día (hashVisitaDia). Ejecute nuevamente el caso de prueba regresion5-galeria. Foro de dudas.

3. **Implemente** las funciones llegaGrupoTGaleria y obtenerVisitaDiaTGaleria. Ejecute el caso de prueba galeria1-llegaGrupo-obtenerVisita para verificar el funcionamiento de las funciones. Foro de dudas.

4. En las siguientes secciones se trabajará sobre las piezas de la galería. **Implemente** la función piezasEnExposicion que devuelve un TConjuntoPiezas indicando aquellas piezas que están en exposición (es decir, que están incluídas en una exposición activa). Recuerde que puede asumir que el máximo de piezas se mantiene para todas las exposiciones de la galería y para las piezas que se agregan a la galería y que este coincidirá con MAX_PIEZAS. Ejecute el caso de prueba galeria2-piezasEnExposicion. Foro de dudas.

5. **Implemente** la función piezasEnReserva, que devuelve un TConjuntoPiezas indicando aquellas piezas que NO están en exposición. Sugerencia: mantenga un conjunto de piezas a nivel de galería, el cual será actualizado cuando se agregue una nueva pieza, y utilícelo apropiadamente. Ejecute el caso de prueba galeria3-piezasEnReserva. Foro de dudas.

6. **Implemente** la función indiceFelicidadVisitanteTGaleria, que calcula el índice de felicidad de un visitante. El índice de felicidad se calcula como un el porcentaje de piezas favoritas de un visitante que logra ver al entrar a la galería. Consulte la descripción de la función en *galeria.h* para obtener una descripción detallada de cómo realizar el cálculo. Ejecute el caso de prueba galeria4-indiceFelicidad Foro de dudas.

7. Test final y entrega de la Tarea

Para finalizar con la prueba del programa utilice la regla *testing* del Makefile y verifique que no hay errores en los tests públicos. Esta regla se debe utilizar **únicamente luego de realizados todos los pasos anteriores (instructivo especial para PCUNIX en paso 3)**.

1. **Ejecute**:

```
$ make testing
```

Si la salida no tiene errores, al final se imprime lo siguiente:

```
-- RESULTADO DE CADA CASO --
11111111111111111111
```

Donde un 1 simboliza que no hay error y un 0 simboliza un error en un caso de prueba, en este orden:

```
regresion1-visitante
regresion2-conjuntoPiezas
regresion3-exposicion
regresion4-listaExposicion
regresion5-galeria
visitante-agregar-obtener-pieza
exposicion-obtenerPiezas-compatibles
listaExposicion-cantidad-enesima
visitaDia1-crear-liberar-esta-encolar
visitaDia2-prioritario-prioridad
visitaDia3-desencolar
visitaDia4-invertirPrioridad
visitaDia5-combinado
visitaDia6-tiempo
hash1-crear-liberar
hash2-insertar-imprimir
hash3-pertenece-obtener
hash4-combinado
hash5-tiempo
galeria1-llegaGrupo-obtenerVisita
```

```
galeria2-piezasEnExposicion
galeria3-piezasEnReserva
galeria4-indiceFelicidad
```

Foro de dudas.

2. **Prueba de nuevos tests**. Si se siguieron todos los pasos anteriores el programa creado debería ser capaz de ejecutar todos los casos de uso presentados en los tests públicos. Para asegurar que el programa es capaz de ejecutar correctamente ante nuevos casos de uso es importante realizar tests propios, además de los públicos. Para esto **cree un nuevo archivo en la carpeta test**, con el nombre *test_propio.in*, y **escriba una serie de comandos** que permitan probar casos de uso que no fueron contemplados en los casos públicos. **Ejecute el test** mediante el comando:

   ```
   $ ./principal < test/test_propio.in
   ```

   y verifique que la salida en la terminal es consistente con los comandos ingresados. La creación y utilización de casos de prueba propios, es una forma de robustecer el programa para la prueba de los casos de test privados. Foro de dudas.

3. **Prueba en pcunix**. Es importante probar su resolución de la tarea con los materiales más recientes y en una pcunix, que es el ambiente en el que se realizarán las correcciones. Para esto siga el procedimiento explicado en Sugerencias al entregar.

   IMPORTANTE: Debido a un problema en los *pcunix*, al correrlo en esas máquinas se debe iniciar valgrind **ANTES** de correr *make testing* como se indica a continuación:

   Ejecutar los comandos:

   ```
   $ make
   $ valgrind ./principal
   ```

   Aquí se debe **ESPERAR** hasta que aparezca:

   ```
   $ valgrind ./principal
   ==102508== Memcheck, a memory error detector
   ==102508== Copyright (C) 2002-2022, and GNU GPL'd, by Julian $ Seward et al.
   ==102508== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
   ==102508== Command: ./principal
   ==102508==
   $ 1>
   ```

   Luego se debe ingresar el comando **Fin** y recién luego ejecutar:

   ```
   $ make testing
   ```

   Foro de dudas.

4. **Armado del entregable**. El archivo entregable final debe generarse mediante el comando:

   ```
   $ make entrega
   ```

   Con esto se empaquetan los módulos implementados y se los comprime generando el archivo `EntregaTarea4.tar.gz`.

   El archivo a entregar **DEBE** ser generado mediante este procedimiento. Si se lo genera mediante alguna otra herramienta (por ejemplo, usando un entorno gráfico) **la tarea no será corregida**, independientemente de la calidad del contenido. Tampoco será corregida si el nombre del archivo se modifica en el proceso de entrega. Foro de dudas.

5. **Subir la entrega al receptor**. Se debe entregar el archivo **EntregaTarea4.tar.gz**, que contiene los módulos a implementar **visitaDia.cpp**, **hashVisitaDia.cpp**, el módulo de la tarea anterior **conjuntoPiezas.cpp**, así como los cambios a los módulos **visitante.cpp**, **exposicion.cpp**, **listaExposiciones.cpp** y **galeria.cpp**. Una vez generado el entregable según el paso anterior, es necesario subirlo al receptor ubicado en la sección Laboratorio del EVA del curso. **Recordar que no se debe modificar el nombre del archivo generado mediante** `make entrega`. Para verificar que el archivo entregado es el correcto se debe acceder al receptor de entregas y hacer click sobre lo que se entregó para que automáticamente se descargue la entrega.
   IMPORTANTE: Se puede entregar **todas las veces que quieran** hasta la fecha final de entrega. La última entrega **reemplaza a la anterior** y es la que será tenida en cuenta. Foro de dudas.