Tarea 4
TADs Tabla y Cola de Prioridad
Curso 2023

Esta tarea tiene como principales objetivos:

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

La fecha límite de entrega es el **jueves 22 de junio a las 16:00 horas**. El mecanismo específico de entrega se explica en la Sección 5. 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.
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 cada función se especifica cuál debe ser el orden del tiempo de ejecución o en el **peor caso** o en el **caso promedio**.

1. Partiendo de la Tarea 3

1. Para comenzar con la Tarea 4 se deben **descargar** los materiales de la sección correspondiente del EVA. Observe que en los archivos *fecha.cpp*, *evento.cpp*, *agendaLS.cpp*, *persona.cpp* y *personasLDE.cpp* hay una sección demarcada con el título **PEGAR CÓDIGO TAREA 3**. **Reemplace** estas funciones con el código de la Tarea 3. **Compile** el código de la tarea ejecutando *make*, y verifique que los módulos previos funcionan correctamente **ejecutando** el test tarea3-combinado. Foro de dudas.

2. **Implemente** las nuevas funciones primerEventoAgendaLS y primerEventoDeTPersona de los módulos *agendaLS* y *persona* respectivamente. Ejecute el test persona1-primerEvento para verificar que funcionan correctamente. Foro de dudas.

3. **Implemente** las nuevas funciones del módulo *personasLDE*: eliminarPersonaConNombreTPersonasLDE, estaPersonaConNombreEnTPersonasLDE y obtenerPersonaConNombreTPersonasLDE. Ejecute el test personasLDE1-nuevasFunciones para verificar que funcionan correctamente. Foro de dudas.

2. TAD Tabla

En esta sección se implementará en *tablaPersonas.cpp* el TAD TablaPersonas a partir de la especificación dada en *tablaPersonas.h*. Este TAD consiste en una Tabla no acotada cuyo dominio son los *strings* que representan nombres de personas, y el codominio son elementos del tipo *TPersona*. La tabla debe ser implementada mediante una **tabla de dispersión abierta**.

1. **Implemente** la estructura rep_tablaPersona. Para esto, recomendamos revisar las operaciones del TAD solicitadas y realizar un diseño de la/s estructura/s necesaria/s. La representación debe utilizar listas del tipo TPersonasLDE. Foro de dudas.

2. **Implemente** las funciones crearTablaPersonas y liberarTablaPersonas. **Ejecute** el test tabla1-crearLiberar para verificar el funcionamiento de la funciones. Foro de dudas.

3. **Implemente** las funciones insertarPersonaEnTabla e imprimirTTablaPersonas. La función de inserción recibe una TTablaPersonas y una TPersona, y asocia en la tabla a la TPersona con su nombre. Para calcular la posición en la que se debe insertar a la persona en la tabla de dispersión abierta se debe utilizar la función brindada funcionDeDispersion (la función está declarada al comienzo de *tablaPersonas.cpp*). Esta función recibe el nombre de una persona y devuelve un natural calculado como la suma de los valores ascii de los caracteres que conforman el nombre. La persona debe ser insertada

en la posición $p = funcionDeDispersion(nombrePersona)\ \%tamanio$, donde $\%$ es la operación de módulo y $tamanio$ es el tamaño con el que se creó la tabla de dispersión. Por convención se deberá **insertar** la persona **al comienzo** de la lista asociada a esa posición. La función imprimirTTablaPersonas debe imprimir cada persona de la tabla, en orden creciente de posiciones asociadas en la tabla. En caso de que haya más de una persona en la misma posición, se deben imprimir por orden de ingreso a la tabla, desde la más reciente a la menos. **Ejecute** el test tabla2-insertarImprimir para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones eliminarPersonaDeTTablaPersonas, perteneceATTablaPersonas y obtenerPersonaDeTabla. **Ejecute** el test tabla3-eliminarPerteneceObtener para verificar el correcto funcionamiento de las funciones. Foro de dudas.

5. **Ejecute** el test tabla4-combinado. Foro de dudas.

3. TAD Cola de Prioridad

En esta sección se implementará en *colaDePrioridadPersona.cpp* el TAD ColaDePrioridadPersona a partir de la especificación dada en *colaDePrioridadPersona.h*. Este TAD consiste en una cola de prioridad cuyos elementos son de tipo TPersona, es decir, es una cola de prioridad de personas y cuya prioridad se define por la fecha del primer evento de la agenda de la persona.
Las personas son únicas, por lo que no es posible que haya dos personas con el mismo ID. **Se asume que los IDs de las personas están acotados entre 1 y N, siendo N el tamaño máximo de la cola.**
El criterio para establecer la prioridad entre personas es, **de manera predeterminada**, que una persona es prioritaria ante otra si la fecha de su primer evento en la agenda es menor que la fecha del primer evento de la otra persona. Este criterio se puede modificar, haciendo que la persona prioritaria sea la del evento con fecha mayor (siempre considerando el primer evento de la agenda).
Si la cola no es vacía hay una persona considerada la prioritaria, según el criterio de prioridad. Si hay más de una persona con la misma fecha cualquiera de ellas es la prioritaria.
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_colaDePrioridadPersona. Para esto, recomendamos revisar las operaciones del TAD solicitadas y realizar un diseño de la/s estructura/s necesaria/s. Foro de dudas.

2. **Implemente** las funciones crearCP, liberarCP, estaVaciaCP, estaEnCP y insertarEnCP. Recomendamos que declare e implemente las funciones auxiliares void filtrado_ascendente(nat pos, TColaDePrioridadPersona &cp) que realiza el filtrado ascendente en el heap y TFecha obtenerFechaPrioridad(TPersona persona) que devuelve la fecha que se usará como prioridad de la persona. Además, recuerde que puede comparar fechas utilizando la función compararTFechas del módulo Fecha. **Ejecute** el test CP1-crear-liberar-vacia-esta-insertar para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** las funciones prioritaria y prioridad. **Ejecute** el test CP2-prioritaria-prioridad-insertar para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** la función eliminarPrioritaria. Recomendamos que implemente la función auxiliar void filtrado_descendente(nat pos, TColaDePrioridadPersona &cp) que realiza el filtrado descendente en el heap. **Ejecute** el test CP3-eliminar. Foro de dudas.

5. **Implemente** la función invertirPrioridad y **ejecute** el test CP4-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 cp. Sin embargo, existe una solución que lo hace en $O(n)$.Foro de dudas.

6. Ejecute el test CP5-tiempo para verificar que su implementación cumple con los tiempos de ejecución solicitados. Recuerde que los tests que evalúan el tiempo de ejecución se deben ejecutar sin valgrind. Para esto puede hacer **$ make tt-NN**, donde NN es el nombre del caso de prueba. Por ejemplo, para este caso **pruebe ejecutar**: `make tt-CP5-tiempo`. Foro de dudas

4. Módulo de aplicaciones

En esta sección se implementarán las funciones del módulo *aplicaciones.cpp*. Estas funciones hacen uso de los TADs implementados anteriormente como parte de su implementación. **Recordar que no se debe acceder a la representación de los TADs, solamente se deben usar las funciones definidas de cada TAD.** Para esto puede ser necesario incluir las especificaciones de los TADs a utilizar en el archivo *.cpp*.

1. **Implemente** la función listarEnOrden que recibe una Tabla de personas y una lista de nombres (en un arreglo de strings) e imprime en pantalla los datos de las personas de la lista que están en la tabla, en orden de fecha de su evento más próximo. Se asume que los IDs de las personas están acotados entre 1 y MAX_ID (definido en aplicaciones.h). El tiempo de ejecución en promedio debe ser $O(n*\log n+n*m)$, donde n es la cantidad de nombres en la lista, y m es la cantidad de eventos de la agenda con mas eventos entre todas las personas de la tabla. Se **debe** utilizar una cola de prioridad auxiliar en la implementación de la función. **Ejecute** el test aplicaciones1-listarEnOrden para verificar el funcionamiento de las funciones. Foro de dudas.

5. 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 --
   ```

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

   ```
   tarea3-combinado
   persona1-primerEvento
   personasLDE1-nuevasFunciones
   tabla1-crearLiberar
   tabla2-insertarImprimir
   tabla3-eliminarPerteneceObtener
   tabla4-combinado
   CP1-crear-liberar-vacia-esta-insertar
   CP2-prioritaria-prioridad-insertar
   CP3-eliminar
   CP4-invertirPrioridad
   CP5-tiempo
   aplicaciones1-listarEnOrden
   ```

   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 y no coincide con** `EntregaTarea4.tar.gz`. Foro de dudas.

5. **Subir la entrega al receptor**. Se debe entregar el archivo **EntregaTarea4.tar.gz**, que contiene los módulos a implementar **fecha.cpp**, **evento.cpp**, **persona.cpp**, **agendaLS.cpp**, **personasLDE.cpp**, **tabla.cpp**, **colaDePrioridadPersona.cpp** y **aplicaciones.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.