Simulador Puestos de Atención

octubre 28

2014

Peratta Franco, LU: 105460; Piersigilli Joaquín E, LU: 105285; Salem Santiago, LU: 105490

Proyecto C

Decisiones de implementación:

Se tomó la decisión de que:

- Los puestos sean un array de tTramite.
- Las colas de espera sean un array de tDeque.
- Las llegadas de los trámites son ingresados en una lista ordenada de menor a mayor, siendo esta lista luego utilizada para ir cargando los clientes.

A su vez, se decidió crear un archivo auxiliar, el cual es una variable global, con el propósito de poder utilizarlo en todas las funciones, sin la necesidad de pasarlo por parámetros. En el archivo antes mencionado se escribe todos los cambios que ocurren con la ejecución del simulador. Luego pueden darse dos situaciones:

1. En el caso que el usuario desee un archivo de salida, entonces la dirección indicada será utilizada para crearlo. Y luego ese será el archivo el que podrá consultar lo sucedido durante la simulación.

2. Sino, directamente se imprime por pantalla.

Métodos del tDeque:

tDeque* crearDeque() Crea un Deque vacío y devuelve un puntero al mismo. El método crea un puntero a Deque y reserva en memoria el espacio necesario para guardar los dos nodos y el entero correspondiente a la cantidad de nodos de la Deque.

void insertarEnDeque(tDeque* D, void * x) Inserta el elemento x al final de la cola. Recibe como parámetro el puntero al Deque donde hay que insertar un nodo que contendrá al elemento x. En el método se reserva espacio en memoria para un nodo nuevo y se le asigna x al puntero a elemento de ese nodo, como se inserta al final de la cola no tiene un nodo siguiente. Si al momento de insertar, la cola está vacía, se inserta el nodo como primer y último elemento de la cola simultáneamente, en cambio si la cola ya tiene elementos, se toma el último nodo existente y se asigna el nuevo nodo como siguiente a ese último y como nuevo nodo último de la cola.

void* eliminarAtras(tDeque* D) Elimina y devuelve el primer elemento de la Deque. Recibe como parámetro el puntero a la Deque de donde se va remover y retornar el primer elemento de la misma. Si la Deque está vacía retorna null, de lo contrario, toma el elemento del primer nodo y luego, si la cola tenía sólo un nodo, le asigna null al mismo y retorna el elemento guardado, si tenía más de un nodo, guarda una referencia al segundo nodo, remueve el primero, y asigna como primer nodo a la referencia del que antes estaba segundo y retorna el elemento guardado. Cada vez que remueve disminuye en uno la cantidad de elementos del nodo.

void* eliminarAdelante(tDeque* D Elimina y devuelve el último elemento de la Deque. Recibe como parámetro el puntero a la Deque de donde se va remover y retornar el último elemento de la misma. Si la Deque está vacía retorna null, de lo contrario, toma el elemento del último nodo y luego, si la cola tenía sólo un nodo, le asigna null al mismo y retorna el elemento guardado, si tenía más de un nodo, guarda una referencia al anteúltimo nodo, remueve el último, y asigna como último nodo a la referencia del que antes estaba anteúltimo y retorna el elemento guardado. Cada vez que remueve disminuye en uno la cantidad de elementos del nodo.

int dequeVacio(tDeque* D) Devuelve verdadero (!= 0) si la Deque está vacía, y falso (0) si contiene al menos un elemento.

int ocupacionDeque(tDeque * D) Devuelve la cantidad de elementos que tiene la Deque.

void * deque2array(tDeque * D) Devuelve un arreglo con todos los elementos de la Deque. La función reserva espacio en memoria para un arreglo con la misma cantidad de elementos que nodos en la cola. Luego para cada nodo de la cola, toma su elemento y lo asigna a su correspondiente lugar en el arreglo.

Métodos de tListaOrdenada:

tListaOrdenada crearLista (int (*f) (void*, void*)) Crea una lista vacía y la devuelve. El orden de los elementos insertados estará dado por la función de comparación int (void*, void*). Se considera que la función f devuelve -1 si el orden del primer argumento es menor que el orden del segundo, 0 si el orden es el mismo, y 1 si el orden del primer argumento es mayor que el orden del segundo. Recibe como parámetro la función con la cual se va a crear la lista y devuelve la variable de la tListaOrdenada. Guarda el espacio correspondiente a dicho nodo.

Void insertar (tListaOrdenada* L, void* x) Inserta el elemento x ordenado en la lista L. Recibe como parámetro el puntero a la tListaOrdenada y el void* que se va a insertar. Crea un nuevo nodo donde guarda el void* y reserva el espacio correspondiente a dicho nodo.

tNodo* siguiente(tNodo* pos) Devuelve la posición siguiente a la posición pos. Devuelve NULL si p es NULL o si no existe posición siguiente. Recibe como parámetro el puntero al tNodo del cual quiere obtenerse su siguiente.

tNodo* primera(tListaordenada L) Devuelve la primera posición de la lista L. Recibe como parámetro la variable a la tListaordenada y retorna el puntero al primer tNodo de la lista.

tNodo* ultima(tListaordenada L) Devuelve la última posición de la lista L. Recibe como parámetro la variable a la tListaordenada y retorna el puntero al último tNodo de la lista.

Int listaVacia(tListaordenada L) Devuelve verdadero (distinto de 0) si la lista está vacía, y falso (=0) si la lista contiene al menos un elemento. Recibe como parámetro la variable a la tListaordenada y retorna alguno de los valores antes mencionados según este o no vacía.

void* elemento(tNodo* pos) Devuelve un puntero al elemento almacenado en el tNodo pasado por parámetro. Si el parámetro es nulo retorna NULL.

Métodos de tTramite:

tTramite* crearTramite(char* c, int d, int l): crea un nuevo trámite con los campos inicializados según los parametros que recibe el método.

char* getNombre(tTramite* tram): retorna el nombre del trámite pasado por parámetro si este no es nulo. De lo contrario devuelve un string vacio.

int getDuracion(tTramite* tram): retorna la duración del trámite pasado por parámetro.

int getLlegada(tTramite* tram): retorna el tiempo de llegada del trámite pasado por parametro.

void setDuracion(tTramite* tram, int nro): Le modifica la duración al trámite pasado por parámetro por un entero que es el pasado por parámetro, siempre y cuando este no sea nulo.

Simulador:

Como fue mencionado antes, el simulador cuenta con un arreglo en la que cada posición del mismo representa un puesto de atención. Una vez que comienza a funcionar el simulador, este toma el archivo con los nombres de los tramites y haciendo uso de la función asignarLlegadas, comienza a insertar en una lista los tramites ordenados (de menor a mayor) según el tiempo de llegada indicado en el correspondiente archivo.

Una vez que la lista está totalmente cargada, comienza el ciclo que se encarga de:

- Liberación de puestos.
- Comienzo de atención.
- Arribo de clientes.
- Robo de clientes.

Este ciclo finaliza cuando se alcanzó el tiempo especificado por el usuario o no hay más trámites por atender.

Métodos del simulador:

void liberacionPuestos(tTramite** puestos, int cantPuestos, int tiempoActual, int *cambios) : Esta función recibe por parámetro un arreglo de Tramites que simulan ser cada uno de los puestos de atención, el tamaño del arreglo, el tiempo actual en el cual se llama a esta función y la variable cambios que notifica si se produce alguna liberación de algún puesto. Cuando comienza la función, recorre los puestos de atención y para los que no están vacios toma el tramite que está atendiendo y le resta un segundo a su duración, si luego de esto su duración llega a 0 se libera el puesto.

void comienzoAtencion(tTramite** puestos, int cantPuestos, tDeque** colasDeEspera, int tiempoActual, int *cambios) Recorre los puestos de atención. Si el puesto no está atendiendo a nadie entonces observa si hay gente esperando en su cola de espera de haber gente, atiende al primero de ellos, siendo removido de la cola de espera. Se utiliza un entero cambios para saber si, al menos un puesto comenzó a atender a un cliente de su cola de espera.

void arriboClientes(int tiempoActual, tListaOrdenada *listaLlegadas, tTramite** puestos, int cantPuestos, tDeque** colasDeEspera, int* cambios): Esta función recibe por parámetro el tiempo en el que es llamada, una lista con los trámites ordenados de menor a mayor según sus tiempos de llegada, un arreglo de todos los puestos de atención, el tamaño de ese

arreglo, las colas de espera de los puestos de atención y una variable cambios que notifica si arribó algún cliente. Cuando comienza la función, toma el primer trámite de la lista de llegadas y se corrobora si llegó en el tiempoActual, en caso de que así lo sea, busca si hay algún puesto libre y si lo hay, inserta el trámite en el mismo, de lo contrario (si no hay ningún puesto libre) busca la cola de espera con menos gente e inserta el tramite al final de la misma. En caso de que ese trámite no haya llegado en el tiempoActual, la función termina.

void roboClientes(tTramite** puestos, int cantPuestos, tDeque** colasDeEspera, int tiempoActual, int *cambios): Si un determinado puesto esta vacio, y en la cola de espera asignada a ese puesto no hay clientes, busca en las colas vecinas y le "roba" un cliente a la que más clientes tenga. Se consideran los casos especiales de que el puesto a evaluar sea el primero o el último, ya que solo tiene una cola vecina.

int puestoLibre(tTramite** puestos, int cantPuestos, tDeque** colasDeEspera): Recorre todos los puestos de atención hasta que encuentre uno que esta vacio y cuya cola de espera también este vacia. Luego retorna el numero de puesto. Caso contrario retorna -1.

void asignarLlegadas(char filename[], int duracion, tListaOrdenada *listaLlegadas): Obtiene a partir de filename el nombre del trámite que debe procesar. Sin embargo lo recibe sin el ".txt". Luego le concatena ".txt" de modo que pueda acceder al archivo del trámite correspondiente. Por último, a medida que lee del archivo, va creando los trámites con la duración indicada por parámetro, el tiempo de llega que es extraído del archivo y el nombre del mismo.

tDeque** generarColasDeEspera(int cantPuestos) Recibe por parámetros la cantidad de puestos existentes. A partir de ello, genera un array en el que cada una de sus celdas es un puntero a un tDeque.

tTramite** generarPuestos(int cantPuestos): Recibe la cantidad de puestos que deben estar presentes en el simulador. Crea un arreglo de tTramites de cantidad cantPuestos. Inicializa todos los componentes del arreglo en NULL, y retorna el arreglo.

void verCola(tDeque* cola, char* visual): Se encarga de almacenar en un String el estado de la cola de espera de cada uno de los puestos.

void mostrar(tTramite** puestos, int cantPuestos, tDeque** colasDeEspera): Se encarga de escribir en el archivo_registro (variable global) el estado actual de los puestos de atención y de sus colas de espera.

void ayuda(): Esta es la función que se encarga de mostrar por pantalla una ayuda para que el usuario entienda el funcionamiento del simulador.

void chequearTiempo(tTramite** puestos, tDeque** colasDeEspera,int cantPuestos, int *TERMINAR): En caso que el cliente no ingrese el tiempo que debe durar el simulador, esta función es que la que se encarga de determinar cuándo debe finalizar el simulador.