Córdoba 7 de Junio del 2022

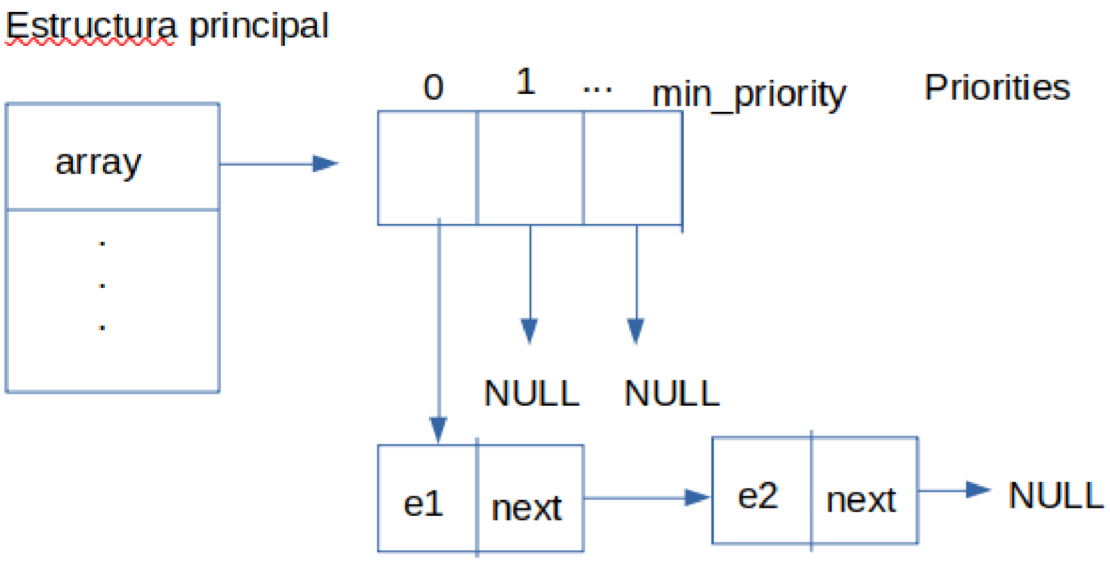
Segundo Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA C

Ejercicio

Implementar el TAD ***pqueue*** que representa una cola de prioridades. Una cola de prioridades (**pqueue**) es un tipo especial de cola en la que cada elemento está asociado con una prioridad asignada. En una cola de prioridades un elemento con mayor prioridad será desencolado antes que un elemento de menor prioridad. Sin embargo, si dos elementos tienen la misma prioridad, se desencolarán siguiendo el orden de la cola. En este examen vamos a implementar **pqueue** usando una estructura principal que contendrá un array dinámico donde cada índice será la queue de una prioridad establecida:



En el gráfico vemos la estructura principal formada por elementos a definir y un elemento llamado “array” que es un array dinámico de punteros a nodo. Cada elemento del array representa una prioridad. En el dibujo vemos una cola de 2 elementos con la misma prioridad “0” e1 es el primer elemento de la cola y e2 el segundo.

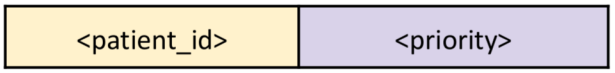
AYUDA: dentro de la estructura principal conviene no sólo guardar el array de prioridades sino además algunos otros datos extra que nos pueden ayudar en las funciones. Ej: conviene guardar min_priority para validar no estar insertando elementos con priority < min_priority. Note que esta relación de orden no es la tradicional pues min_priority 5 representa que un elemento con prioridad 4 es más prioritario que este.

El TAD ***pqueue*** tiene la siguiente interfaz

Función	Descripción
`pqueue pqueue_empty(priority_t min_priority)`	Crea una cola de prioridades vacía para almacenar prioridades <= min_priority
`pqueue pqueue_enqueue(pqueue q, pqueue_elem e, priority_t priority);`	Inserta un elemento a la cola con su correspondiente prioridad.
`bool pqueue_is_empty(pqueue q);`	Indica si la cola de prioridades está vacía
`size_t pqueue_size(pqueue q)`	Obtiene el tamaño de la cola de prioridades
`pqueue_elem pqueue_peek(pqueue q)`	Obtiene el elemento con mayor prioridad
`priority_t pqueue_peek_priority(pqueue q)`	Obtiene el valor de la prioridad del elemento con mayor prioridad.
`pqueue pqueue_dequeue(pqueue q)`	Quita un elemento con mayor prioridad más antiguo de la cola
`pqueue pqueue_destroy(pqueue q)`	Destruye una instancia del TAD Cola de prioridades

En `pqueue.c` se da una implementación incompleta del TAD ***pqueue*** que deben completar **siguiendo la representación explicada anteriormente.** Además deben asegurar que la función `pqueue_size()` sea de orden constante (*O(1)*). Por las dudas se aclara que no es necesario que la función `pqueue_enqueue()` sea de orden constante ya que puede ser muy complicado lograrlo para la representación que se utiliza y no recomendamos intentarlo.

Para verificar que la implementación del TAD funciona correctamente, se provee el programa (**main.c**) que toma como argumento de entrada el nombre del archivo cuyo contenido tiene por encabezado "**min_priority:** <int>" con la minima prioridad permitida en el archivo y el sigue el siguiente formato:



El archivo de entrada representa una lista de pacientes con prioridades de atención médica. Entonces el programa lee el archivo, carga los datos en el TAD ***pqueue*** y finalmente muestra por pantalla la cola de prioridades de los pacientes.

El programa resultante no debe dejar *memory leaks ni lecturas/escrituras inválidas*.

Una vez compilado el programa puede probarse ejecutando:

```
$ ./dispatch_patients inputs/hospital_a.in
```

Obteniendo como resultado:

```
length: 7
[ (454, 1), (456, 2), (345, 3), (686, 4), (234, 5), (234, 6), (789, 8)]
```

Notar que para este archivo el encabezado dice min_priority: 8 ya que la mínima prioridad permitida es 8

Otro ejemplo de ejecución:

```
$ ./dispatch_patients inputs/hospital_b.in
length: 7
[ (100, 0), (200, 0), (300, 0), (400, 0), (500, 0), (600, 0), (700, 0)]
```

En este caso min_priority es 0

Otro ejemplo:

```
$ ./dispatch_patients inputs/hospital_c.in
length: 7
[ (234, 4), (456, 4), (789, 4), (686, 4), (454, 4), (234, 5), (345, 6)]
```

Y en este último min_priority es 6

Consideraciones:

- Se provee el archivo **`Makefile`** para facilitar la compilación.
- Se recomienda usar las herramientas **`valgrind`** y **`gdb`**.
- Si el programa no compila, no se aprueba el parcial.
- Los *memory leaks* bajan puntos
- Entregar código muy improlijo puede restar puntos
- Si **`pqueue_size()`** no es de orden constante baja muchísimos puntos
- Para promocionar **se debe** hacer una invariante que chequee la propiedad fundamental de la representación de la cola de prioridades.