**Memoria de la práctica P2**

**P2\_E1**

La mayoría de las funciones serán idénticas al resto de grupos, sin embargo, en específico podemos destacar que el top de la pila sea -1 en vez de 0 cuando está vacía, lo cual facilita futuras cuentas.

También hay que especificar que en stack\_print **hemos utilizado el salto de línea del bloc de notas de Windows**, es decir, ‘\r\n’ en vez de ‘\n’.

**P2\_E2**

Este segundo ejercicio tiene toda su complicación en la función format, donde al introducir un FILE y un grafo debemos mostrar todas las conexiones existentes en el grafo con el siguiente formato:

NodoOrigen, NodoDestino\n

Lo primero que hacemos es recoger un array de enteros donde se nos muestren los IDs de todos los nodos mediante graph\_getNodeIds.

Mediante graph\_areConnected seremos capaces de saber si dos nodos están o no conectados. Así, mediante dos bucles for anidados, somos capaces de saber si dos nodos del grafo están conectados e ir imprimiendo los resultados en orden.

El orden que elegimos es el inverso al normal. Con esto queremos decir que, en vez de explorar las conexiones del primer nodo con el resto de nodos, empezamos con el último nodo. Además, cuando queremos explorar las conexiones de dicho nodo, también empezamos a ver las conexiones desde el final. De esta forma, la pila devolverá las conexiones en el orden correcto, debido a que sigue un orden LIFO.

Cada vez que se verifica que existe una conexión entre dos nodos, se guarda primero el nodo origen en la pila, y después el nodo destino.

Una vez hecho todo esto, solo nos queda imprimir en el archivo de texto la pila deseada mediante stack\_print.

**P2\_E3**

Intentamos devolver el camino entre los nodos, pero se nos hizo muy complicado solucionar los errores de memoria, por lo que al final nos decantamos por mostrar simplemente si dos nodos estaban o no conectados.

El ejercicio consta de cuatro funciones (aparte del main). La primera es la que se nos pide, recorrer. Las otras dos las utilizamos para fabricar un array en el que cada posición corresponde a un nodo (i=id del nodo), de tal forma que cuando un nodo no está marcado lo señalizamos con un 0 y cuando sí lo está los señalizamos con un 1. Las dos funciones son etiquetar\_nodo, la función booleana nodo\_marcado, que nos indica si un nodo está o no etiquetado en el array y, finalmente, mas\_caminos, que nos indica si existen más nodos no marcados conectados a ese nodo.

Nuestro bucle se seguirá ejecutando mientras existan nodos por recorrer, es decir, mientras nuestra pila de elementos no esté vacía. Sacamos un elemento de la pila cada vez que empezamos el bucle. Si el nodo en cuestión ya está marcado, recorremos de nuevo el bucle. Si no lo está, continuamos todo lo dicho a continuación:

Lo primero que hacemos es marcar el nodo. Si ese nodo es nuestra meta, nuestro bucle acaba y devolvemos TRUE. En el caso contrario, recorremos un bucle for para interactuar con cada uno de los nodos a los que está conectado nuestro nodo. Si alguno de estos nodos es el nodo destino, devolvemos TRUE y el bucle deja de ejecutarse. En caso contrario, metemos el nodo en nuestra pila.

Si todo el bucle se ejecuta y no hemos encontrado el nodo destino, devolvemos FALSE. Por convenio, hemos decidido que si hay un error durante la ejecución de la función se devuelva que no hay ningún nodo conectado.

Una vez hecho todo esto, el main es prácticamente utilizar las funciones y, en función de lo devuelto, imprimir por pantalla si existe o no conexión entre los grafos.

**P2\_E4**

Empezamos definiendo functions, donde definimos las funciones utilizadas para copiar, destruir o imprimir en un fichero un int y un nodo. Así, hemos definido las seis funciones que nos servirán para ver si nuestro ejercicio ha sido realizado correctamente.

Donde se realiza un cambio radical es en stack\_fp.c. Se pasa de una estructura formada por los datos contenidos y un int que nos muestre el top a una en la que además se incluyen las funciones de copia, destrucción e impresión del tipo de dato con el que se esté trabajando.

Esas tres funciones son propias de cada pila, por lo que las asociaremos a su estructura a la hora de crear una pila. Pasamos como argumento las tres funciones necesarias en stack\_ini, por lo que ahora esta función pasa de tener 0 argumento a tener 3.

El resto de las funciones no se diferencian demasiado a las originales. La única diferencia es que debemos sustituir element\_destroy por p->destroy\_element\_function, element\_copy por p->copy\_element\_function y element\_print por p->print\_element\_function.

El main no tiene gran complicación una vez hecho todo lo anterior. Debemos eliminar todo lo relativo a los elementos, y sustituirlo por las tres funciones genéricas mencionadas con anterioridad. Creamos dos pilas, una para nodos y otra para IDs. La única diferencia que tiene el código del bucle con respecto al segundo ejercicio es que, en vez de introducir los dos elementos con nodos conexos en una pila, ahora debemos introducir los IDs de esos elementos en una pila y los nodos en otra pila. Después debemos imprimir en el fichero de texto una pila después de otra.

**Conclusiones finales**

Como experiencia positiva, hemos de decir que las pilas han sido interiorizadas perfectamente gracias a su reiterado uso. Además, el ejercicio 3 ha requerido un nivel de coordinación y abstracción que nunca antes se nos había pedido.

Como parte negativa podemos señalar que nos ha costado demasiado tiempo hacer el ejercicio 3, y la parte opcional ha sido francamente imposible con solo 4 semanas de trabajo. Además, hubiera sido más sencillo diseñar un Makefile si en algún momento de la carrera se nos hubiera explicado qué hacía y cómo se trabajaba con ellos. Pero el gran fallo viene después: nunca antes se nos habían explicado los punteros a funciones, y en la clase de teoría no se ha explicado tampoco nada relativo a esta parte de la práctica, por lo que, aunque el ejercicio era fácil ha resultado una auténtica odisea para nosotros.