PARTE 2A.

1. En este caso al ser una lista no habría un límite en el número de elementos. Se podrían añadir y extraer elementos de cualquier posición, a diferencia de la pila que se añaden por el final y se extraen del principio.

2. En el caso de la memoria estática, dado que hay una función que nos dice el número de elementos de una lista, podríamos acceder directamente a la posición del medio del array donde se encuentra la dirección de memoria del elemento que se quiere eliminar. (lista[i], siendo i la mitad del número de elementos). Una vez eliminado este elemento habría que mover todos los elementos siguientes a la posición i a una posición anterior.

   En el caso de que fuera con memoria dinámica, tal y como se ha implementado en esta práctica, tendríamos que hacer un bucle for en el que se vaya avanzando i veces al siguiente elemento de la lista. Así lo hemos realizado en la práctica para extraer elementos de una posición i-ésima. Una vez que hemos llegado al elemento anterior que queremos eliminar solo hay que poner que elemento siguiente a este sea el siguiente del siguiente del elemento que se quiere eliminar. Y eliminamos el elemento (para esto nos servimos de una copia que hacemos al principio de la función).

   Aunque en ambos casos es necesario un bucle for o while, si se tuviese que realizar varias veces esta operación sería mucho más eficiente usar la opción con memoria dinámica. La escritura en una memoria siempre será más lenta que la lectura (pues la escritura es síncrona y la lectura no), por tanto será mucho más rápido leer hasta llegar a la mitad de la lista y eliminar el elemento que eliminar el elemento y tener que desplazar todos los siguientes.

3. En el ejercicio 1 tenemos un array (en forma de tabla) de 1 y 0, y en esta práctica son dos arrays de listas donde se indican las conexiones entrantes a un nodo y en el otro las conexiones salientes. Sabiendo esto para saber por ejemplo las conexiones entrantes o salientes a un nodo en el primer ejercicio habría que recorrer una columna o fila (dependiendo del caso) y viendo si en cada posición hay un 1 o 0. En la práctica 3 simplemente hay que ir a la posición del array (de conexiones entrantes o salientes)

correspondiente y ahí ya tenemos una lista de todos los id de los nodos correspondientes.

Antes nos referíamos a la opción A de la práctica 3. Ahora vamos a explicar diferencias respecto a la opción B de la práctica 3. En la opción B se eliminan todos los arrays estáticos, luego para acceder a un nodo ya no se puede de manera directa si no que hay que recorrer toda la lista de nodos hasta llegar a la posición correspondiente. La parte de las conexiones es más parecida a la opción A, en vez de una tabla de 1 y 0 tenemos como en la anterior opción A, con la diferencia que ahora no es un array en el que cada posición señala a una lista, si no que es una lista y cada elemento es otra lista en el que hay el id de un nodo y el resto son id de los nodos a los que el enlace va o viene.

PARTE 2B.

1. En el primer ejercicio no encontramos muchas dificultades ya que fue muy similar al último de la práctica anterior. En el segundo nos ocurre lo mismo, como ya habíamos hecho un ejercicio sobre búsqueda de nodos en la anterior práctica este nos resulto bastante sencillo. En el ejercicio 3 hemos encontrado dificultades en la implementación de las funciones de lista debido a pérdidas de memoria y a que la función de extraer el último elemento nos ha dado bastantes problemas hasta que la hemos conseguido terminar. El ejercicio 4 hemos escogido la opción A ya que nos ha parecido la más sencilla, su implementación ha sido bastante mas fácil de lo que pensábamos y no nos ha costado mucho trabajo, quitando el conseguir resolver unas pérdidas de memoria.

2. Tras solucionar varios errores y pérdidas de memoria, las salidas de valgrind para cada uno de los ejercicios es:

P3_E1

```
valgrind ./a.out nodos.txt

==5480== Memcheck, a memory error detector
```

```
==5480== Copyright (C) 2002-2015, and GNU GPL'd,
by Julian Seward et al.

==5480== Using Valgrind-3.11.0 and LibVEX; rerun
with -h for copyright info

==5480== Command: ./a.out nodos.txt

==5480==

Cola 1: Queue vacia.

Cola 2: Queue vacia.

Cola 3: Queue vacia.

Cola 1: cola con 1 elementos.

[1, uno]

Cola 2: Queue vacia.

Cola 3: Queue vacia.

Cola 1: cola con 2 elementos.

[1, uno]

[2, dos]

Cola 2: Queue vacia.

Cola 3: Queue vacia.

Cola 1: cola con 3 elementos.

[1, uno]

[2, dos]

[3, tres]

Cola 2: Queue vacia.

Cola 3: Queue vacia.

<<<Pasando la primera mitad de Cola 1 a Cola 2

Cola 1: cola con 2 elementos.

[2, dos]

[3, tres]
```

```
Cola 2: cola con 1 elementos.

[1, uno]

Cola 3: Queue vacia.

<<<Pasando la segunda mitad de Cola 1 a Cola 3

Cola 1: cola con 1 elementos.

[3, tres]

Cola 2: cola con 1 elementos.

[1, uno]

Cola 3: cola con 1 elementos.

[2, dos]

Cola 1: Queue vacia.

Cola 2: cola con 1 elementos.

[1, uno]

Cola 3: cola con 2 elementos.

[2, dos]

[3, tres]

==5480==

==5480== HEAP SUMMARY:

==5480==     in use at exit: 0 bytes in 0 blocks

==5480==   total heap usage: 15 allocs, 15 frees,
11,564 bytes allocated

==5480==

==5480== All heap blocks were freed -- no leaks
are possible

==5480==

==5480== For counts of detected and suppressed
errors, rerun with: -v

==5480== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 0 from 0)
```

P3_E2

```
valgrind --leak-check=full ./p3_e2 grafo1.txt

==8004== Memcheck, a memory error detector

==8004== Copyright (C) 2002-2015, and GNU GPL'd,
by Julian Seward et al.

==8004== Using Valgrind-3.11.0 and LibVEX; rerun
with -h for copyright info

==8004== Command: ./p3_e2 grafo1.txt

==8004==

HAY CAMINO!==8004==

==8004== HEAP SUMMARY:

==8004==     in use at exit: 0 bytes in 0 blocks

==8004==   total heap usage: 14 allocs, 14 frees,
67,149,824 bytes allocated

==8004==

==8004== All heap blocks were freed -- no leaks
are possible

==8004==

==8004== For counts of detected and suppressed
errors, rerun with: -v

==8004== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 0 from 0)
```

P3_E3 Recortamos el output para que no ocupe demasiado

```
valgrind ./p3_e3 datos.txt

==5650== Memcheck, a memory error detector

==5650== Copyright (C) 2002-2015, and GNU GPL'd,
by Julian Seward et al.

==5650== Using Valgrind-3.11.0 and LibVEX; rerun
with -h for copyright info
```

```
==5650== Command: ./p3_e3 datos.txt
==5650==
Lista con 1 elementos:
[12]
Lista con 2 elementos:
[12]
[11]

[...]

Lista con 12 elementos:
[2]
[4]
[6]
[8]
[10]
[12]
[11]
[9]
[7]
[5]
[3]
[1]
Lista con 11 elementos:
[4]
[6]
[8]
[10]
```

```
[12]
[11]
[9]
[7]
[5]
[3]
[1]
Lista ordenada con 1 elementos:
[2]

[...]

Lista con 2 elementos:
[9]
[7]
Lista ordenada con 10 elementos:
[1]
[2]
[3]
[4]
[5]
[6]
[8]
[10]
[11]
[12]
Lista con 1 elementos:
[9]
```

```
Lista ordenada con 11 elementos:
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[10]
[11]
[12]
Lista con 0 elementos:
Lista ordenada con 12 elementos:
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
[10]
[11]
[12]
==5650==
==5650== HEAP SUMMARY:
```

```
==5650==     in use at exit: 0 bytes in 0 blocks
==5650==   total heap usage: 77 allocs, 77 frees, 6,328 bytes allocated
==5650==
==5650== All heap blocks were freed -- no leaks are possible
==5650==
==5650== For counts of detected and suppressed errors, rerun with: -v
==5650== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

P3_E4

```
valgrind ./p3_e4 grafo1.txt
==5840== Memcheck, a memory error detector
==5840== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==5840== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5840== Command: ./p3_e4 grafo1.txt
==5840==
Prueba 2
N=3, E=1:
[1 , a]->0 1 0
[2 , b]->0 0 1
[3 , c]->0 0 0
[1, a]
    connected from [2, b]

[2, b]
```

```
    connected to [1, a]
    connected from [3, c]


[3, c]
    connected to [2, b]


==5840==
==5840== HEAP SUMMARY:
==5840==     in use at exit: 0 bytes in 0 blocks
==5840==   total heap usage: 27 allocs, 27 frees,
104,960 bytes allocated
==5840==
==5840== All heap blocks were freed -- no leaks
are possible
==5840==
==5840== For counts of detected and suppressed
errors, rerun with: -v
==5840== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 0 from 0)
```

3. Esta práctica nos ha servido para entender del todo las listas, ya que hemos tenido que pensar una a una las distintas primitivas y como implementarlas. También la primera parte nos ha servido para recordar colas. En general, la mayoría de los problemas han estado relacionados con la memoria.

Pareja 8:

Javier Delgado del Cerro

Victoria Pelayo Alvaredo

Ignacio Rabuñal García