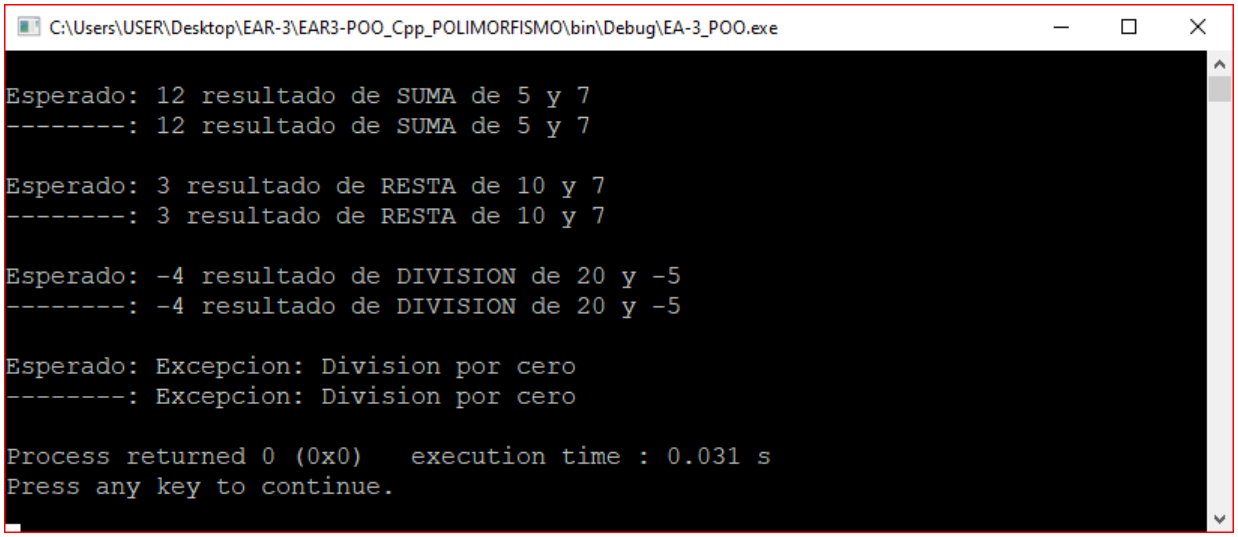
UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

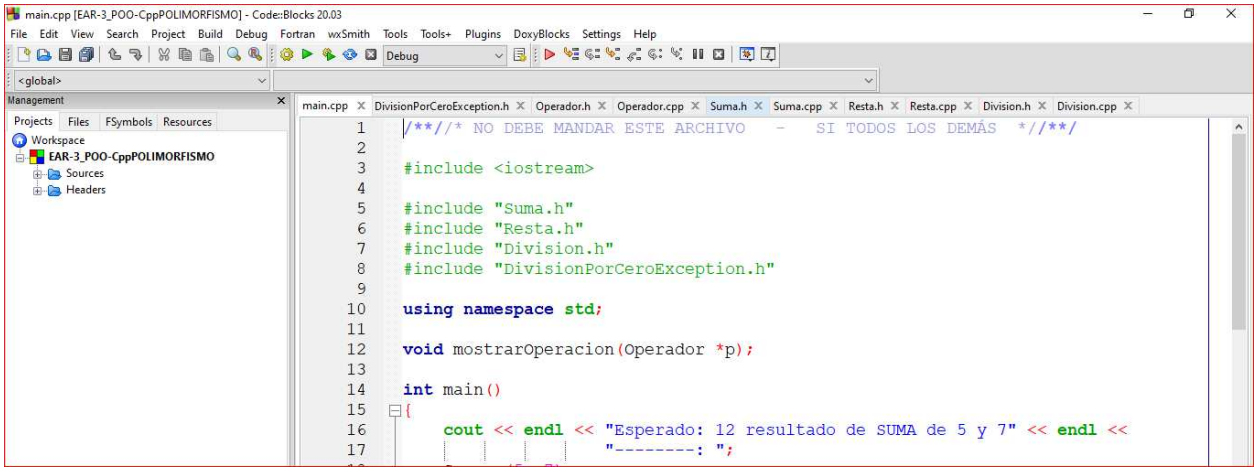
E. A. R. Nro. 3– Rehacer - Tercera Evaluación de Aprendizaje.

Parte 1,-.POO.- Un muy simple y muy conceptual ejercicio de Programación Orientada a Objetos en Lenguaje C++, en el que demuestre los conocimientos adquiridos de herencia, excepciones, polimorfismo, etcétera.

El proyecto que se le entrega contiene solo el archivo main.cpp y debe agregar los archivos con la declaración de las clases y sus desarrollos.

La salida por pantalla será





```
18      Suma s(5, 7);
19      mostrarOperacion(&s);
20
21      cout << endl << "Esperado: 3 resultado de RESTA de 10 y 7" << endl <<
22                      "--------: ";
23      Resta *r = new Resta(10, 7);
24      mostrarOperacion(r);
25
26      cout << endl << "Esperado: -4 resultado de DIVISION de 20 y -5" << endl <<
27                      "--------: ";
28      Operador *d = new Division(20, -5);
29      mostrarOperacion(d);
30
31      // La siguiente linea no deberia compilar
32      //Operador o;
33
34      cout << endl << "Esperado: Excepcion: Division por cero" << endl <<
35                      "--------: ";
36      try
37      {
38          Operador *malD = new Division (1, 0);
39          mostrarOperacion(malD);
40          delete malD;
41      }
42      catch(DivisionPorCeroException &dpc)
43      {
44          cout << "Excepcion: " << dpc.what() << endl;
45      }
46
47      delete r;
48      delete d;
49
50      return 0;
51  }
52
53  void mostrarOperacion(Operador *p)
54  {
55      float r = p->calcular();
56      cout << r << " resultado de "
57           << p->operacion() << " de "
58           << p->getOperandoA() << " y "
59           << p->getOperandoB() << endl;
60  }
61
```

C:\Users\USER\Desktop\EAR-3\EAR3-POO_Cpp_Resuelto\main.cpp | C/C++ | Windows (CR+LF) | WINDOWS-1252 | Line 1, Col 1, Pos 0 | Insert | Read/Write | default

Parte 2.-.Java.- Se pretende desarrollar una aplicación que permita calcular los precios de un Taller mecánico que repara vehículos. Para esto hay mecánicos que realizan diversos trabajos.

Cada mecánico tiene nombre y apellido.

Cada trabajo se identifica unívocamente por medio de su idTrabajo, además posee una descripción del trabajo a desarrollar y una cantidad de horas que le demandó o le va a

demandar. Cada trabajo tiene un costo de trabajo que consta de las horas trabajadas x $800 fijos.

Los trabajos pueden ser de Reparación o Revisión.

Los trabajos de reparación tienen un costo de reparación y los trabajos de revisión tienen un costo de revisión.

Si un trabajo es de Reparación de chapa del vehículo, para calcular el costo total del trabajo se debe hacer la cantidad de horas trabajadas x $800 + el costo de reparación x 2.

Si un trabajo es de Revisión, el costo total del trabajo se debe hacer la cantidad de horas a trabajar + costo de revisión x 1.5 .

- A realizar:

- Class Mecanico: Atributos -> nombre y apellido. Constructor parametrizado y sobrecarga del metodo tostring.

- Class Trabajo: Atributos -> idTrabajo, contadorTrabajos, descripcion, cantHoras. Constructor por defecto, parametrizado, toString, getters correspondientes y metodo costoTrabajo.

- Class Reparacion: Atributos -> costoreparacion . Constructor parametrizado, toString y metodo costoTrabajo.

- Class Revision: Atributos -> costorevision. Constructor parametrizado, toString y metodo costoTrabajo.

- Class ReparacionChapa: Constructor parametrizado, metodo costoTrabajo y toString.

- Main:

- Se debe pedir por teclado el nombre y apellido de un mecánico para crear un objeto mecánico y mostrar el mismo una vez ingresado con el método toString.

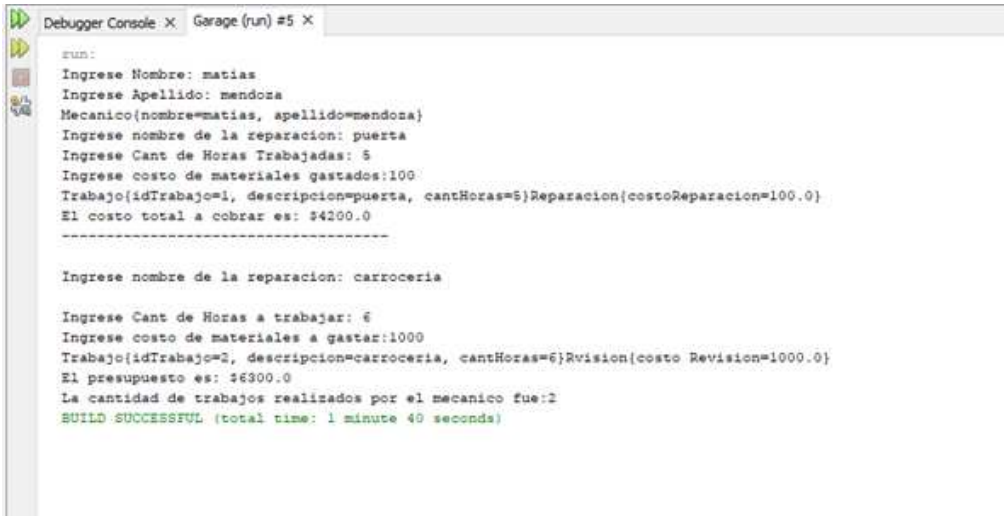
- Se debe pedir por teclado descripción, canthoras y costomateriales para crear un objeto reparacionChapa y luego mostrarlo con el método toString y finalmente calcular el costoTotal del trabajo.

- Se debe pedir por teclado descripción, canthoras y costomateriales para crear un objeto revision y luego mostrarlo con el método toString y finalmente calcular el costoTotal del trabajo presupuestado.

Recordar no duplicar código sino será considerado mal el ejercicio.

Recordar sobrecargar métodos y constructores.

Ejemplo de una salida por consola:



Con el material disponible en [Semana 14] puede resolver todo lo solicitado en este ejercicio.

Parte 3.-.TDA.- Se dispone de una función que permite cargar información sintética (simulada) de los exámenes finales de alumnos en un TDA LISTA implementado en una lista dinámica doblemente enlazada. Esta información está compuesta de: legajo (que

es el DNI), apellido(s) y nombre(s), código de la materia y calificación obtenida, almacenados en orden cronológico en la lista.

Se requiere:

- La primitiva que muestra la lista del primero al último, similar a las dadas, mostrando los títulos (sólo si hay datos a mostrar).

- La primitiva que ordene la lista según la frecuencia de aparición de los nodos (al comienzo quedarán aquellos cuya clave está más veces pero respetando el orden cronológico -ver al final explicación más detallada-).

- La primitiva que elimina todos los nodos, generando un listado en que para cada alumno se muestre al final la cantidad de materias rendidas y su promedio en el mismo orden en que estaban cargados en la lista y que devuelva la cantidad de nodos eliminados.

Para que las primitivas anteriores puedan cumplir su tarea, se deben resolver las funciones de manejo de la información (o datos) cargados en la lista.

Para el ordenamiento pedido, bajo ningún concepto debe utilizar otras primitivas. Pero sí puede utilizar, si lo necesita, otras funciones auxiliares. No debe crear ni eliminar nodos ni espacio para información auxiliar.

La salida de su programa debe coincidir exactamente con la que hace el proyecto que se le entrega. No debe alterar los archivos **`funciones.h`** ni **`main.h`**. Las únicas modificaciones en **`main.c`** son para que invoque sus primitivas. Cualquier "**`include`**" de biblioteca estándar que necesite, debe hacerlo en **`funciones.c`**.

Dispone de un proyecto que al ejecutarlo hace todo lo pedido, con lo que puede comparar los resultados que obtiene con los resultados esperados.

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

Consideraciones Generales (idénticas a las evaluaciones anteriores)

Descargue el proyecto provisto, genere una *carpeta* en su computadora y en ella descomprímalo. Verá que se generan dos *subcarpetas*, una para plataforma (o compilador) de 32 bits y otro para plataforma de 64 bits. Además tendrá el proyecto de C++ sobre el que podrá trabajar. El ejercicio de POO-Java lo deberá desarrollar desde "*cero*".

En la *carpeta* que corresponde a su IDE (**`EA3-Proyecto64`** o **`EA3-Proyecto32`**) abra el proyecto que corresponde a su plataforma (habitualmente con doble click en el archivo **`EA-3.cbp`** o en su defecto una vez abierto el IDE, "*arrastrándolo*" en el mismo).

Proceda a compilarlo.

Si en el cuadro de diálogo inferior le aparecen errores:

. . .**¡¡¡es porque se equivocó de proyecto!!! Vaya al otro y compile:**

Cuando compile el proyecto, asegúrese que el IDE genera el ejecutable con las opciones como se ve en la figura anterior.

```
gcc.exe -Wall -g  -c C:(...etcétera)
gcc.exe -Wall -g  -c C:(...etcétera)
gcc.exe  -o bin\Debug\(...etcétera)    .\lib-EA_3_Release_bib.a
```

De otro modo, podrá tener algunos problemas en los que no le podremos ayudar.

DEBE DEVOLVER TAN SOLO el archivo "**`funciones.c`**", además de los archivos "**`.cpp`**" y "**`.h`**" correspondientes al ejercicio de POO-C++ y al comienzo de cada uno de estos archivos, debe poner en una línea de comentario:

DNI-APELLIDO,Nombre-(curso-comisión), por ejemplo:

```
/**//* 22.333.444-PEREZ_DEL_RIO,JuanManuel-(07-2299) *//**/
```

Estos archivos deben estar comprimidos únicamente en formato **`.zip`** (*) . Para ello copie los tres archivos en una misma ubicación, selecciónelos y con botón derecho elija de la ventana emergente la opción: [**`Enviar a`**] / [**`Carpeta comprimida (en zip)`**] (o [**`Send to`**] / [**`Compressed (zipped) folder`**])

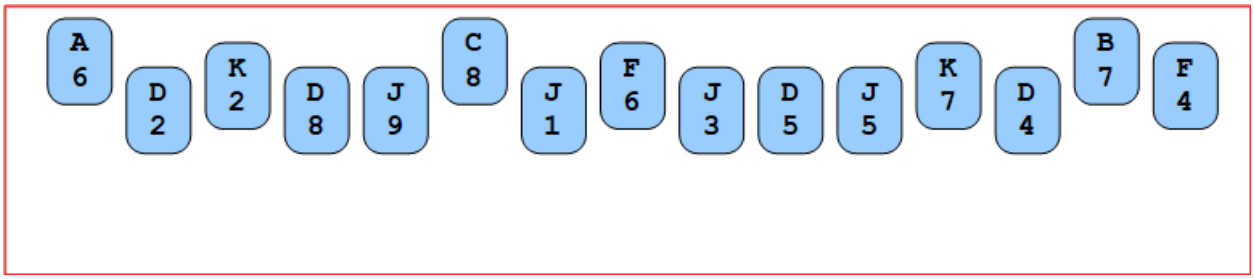
El archivo comprimido debe estar renombrado con su número de dni sin puntos, por ejemplo: **`22333444.zip`**

Aclaración sobre qué significa y qué se espera al ordenar de mayor a menor un TDA-LISTA por la frecuencia de aparición de la clave, respetando el orden cronológico con que fue cargado en la lista.

Este ordenamiento no es un caso trivial, por esto vale que lo orientemos en la primera etapa: entender el problema.

Graficaremos una de las posibles estrategias para el ordenamiento pedido tomando como ejemplo lo siguiente: La información (o datos) a ordenar tiene una clave (que se grafica con una letra) además de otros miembros de esa información (o de ese dato, que graficaremos con el número).

Partiendo por ejemplo de . . .



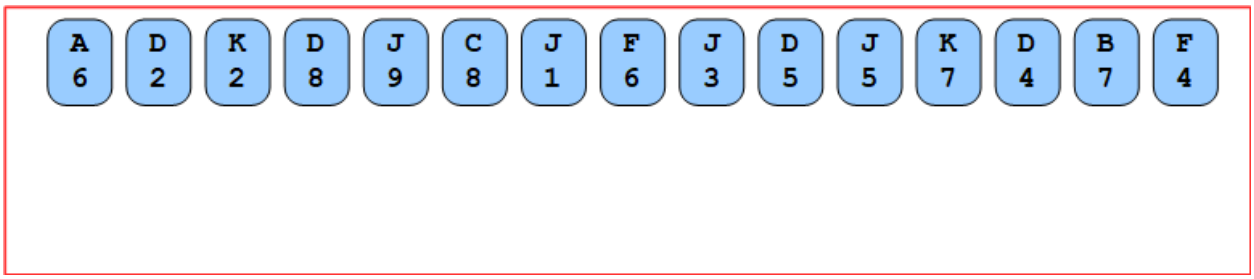
NOTE que este lote de prueba se muestra así para poder visualizar con facilidad que con las claves "**A**", "**C**" y "**B**" solo hay un nodo para cada clave.

En cambio, con las claves "**K**" y "**F**" hay dos nodos con cada clave.

Con tres nodos con la misma clave no hay ningún caso.

Finalmente, con cuatro nodos para cada clave, esto sucede con "**D**" y "**J**".

En síntesis, nuestro lote de prueba es . . .



. . . y la estrategia que se va a implementar es la de buscar en primer lugar los nodos que se encuentran una sola vez, quitarlos de las posiciones en que están para que queden al final de la lista, una vez se la haya ordenado.

Como se quiere respetar el orden cronológico, primero se buscará el último de la lista que está una sola vez (o sea que se debe buscar desde el final el primero que se encuentre y esté una sola vez). Una vez "desenganchado" (o desvinculado) de la lista, se repite la búsqueda, y si hay otro que sólo esté una vez se procede a desvincularlo de la lista y vincularlo antes del que se sacó, y así sucesivamente hasta que no haya más nodos cuya clave esté sólo una vez.

Luego se buscará la primera aparición de derecha a izquierda del que esté dos veces en la lista, desvinculando esos nodos y poniéndolos antes de los que están una vez. Se continúa buscando el próximo que esté dos veces procediendo igual que antes. Se continúa buscando el próximo que aparezca dos veces y como no hay más se pasará a buscar con tres ocurrencias.

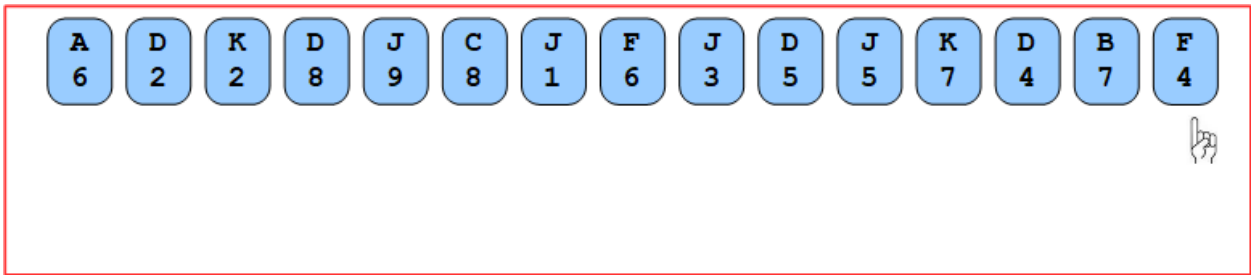
En nuestro lote de prueba, no hay nodos con tres ocurrencias, por lo que se procederá de modo análogo con los que tengan cuatro ocurrencias, y así sucesivamente.

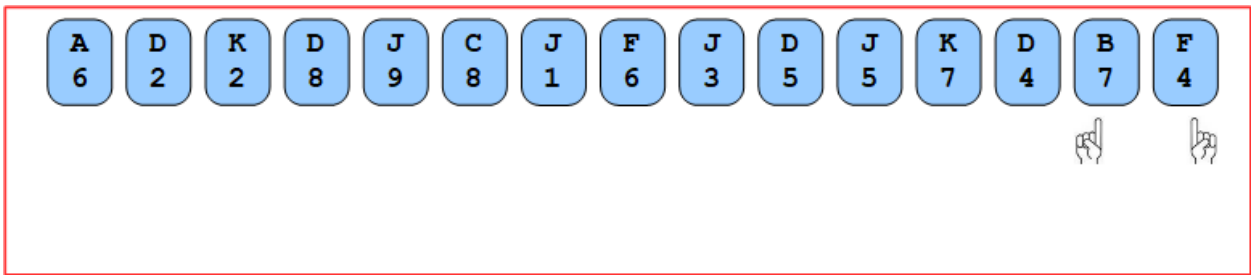
ATENCIÓN: en ningún momento, al ordenar la lista, debe modificar la variable lista (qué nodo direcciona el TDA-LISTA -es una lista dinámica doblemente enlazada- y no necesariamente será ni al primero ni al último), si originalmente apuntaba al último insertado (en nuestro ejemplo "F-4") debe quedar apuntando al nodo al que apuntaba sin

modificación. Considere que puede estar apuntando a cualquier otro nodo (de hecho, será así).

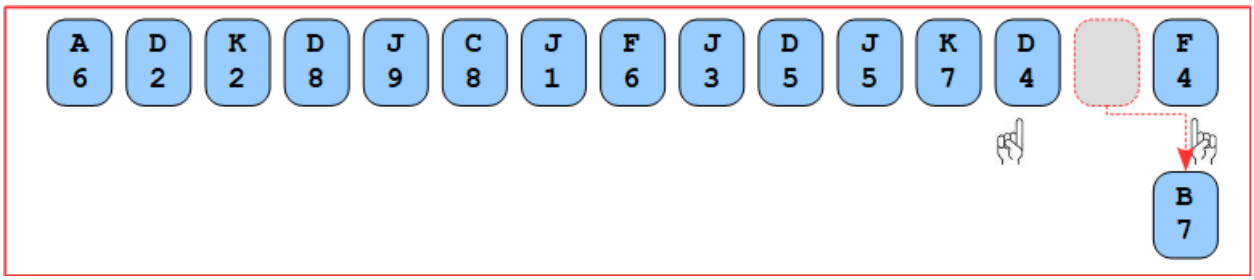
A continuación, se detallan los distintos pasos.

Desde el final y hacia el comienzo se busca el primero que tenga una sola ocurrencia:

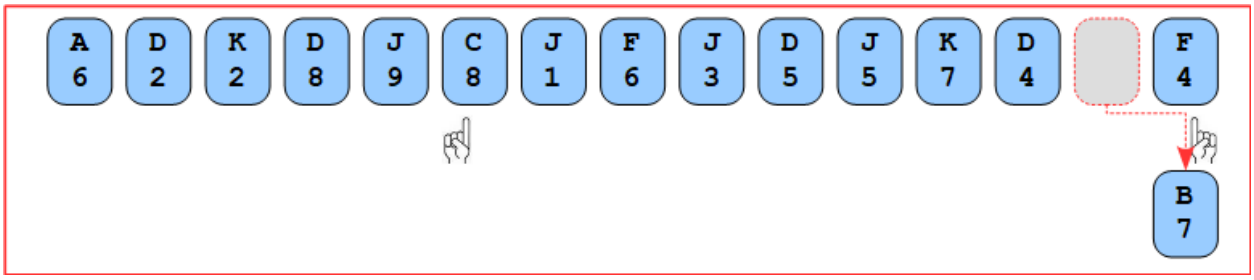




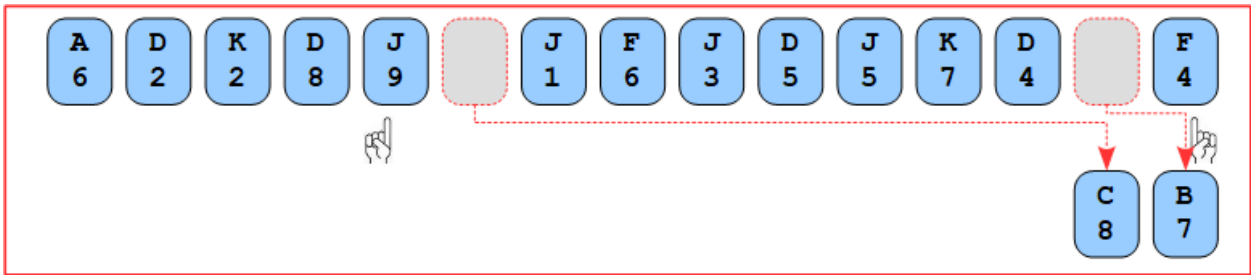
. . . que resultará ser el que tiene la clave "**B**", se lo desvincula de la lista y . . .



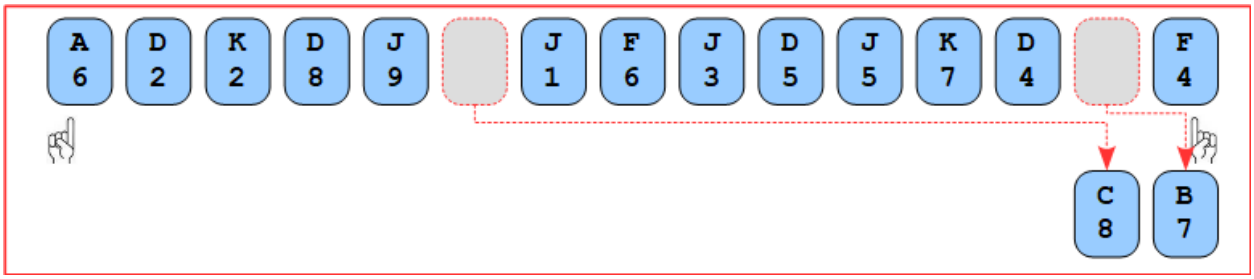
Una vez desvinculado, se continúa buscado hacia la izquierda el primero que ocurra una vez . . .



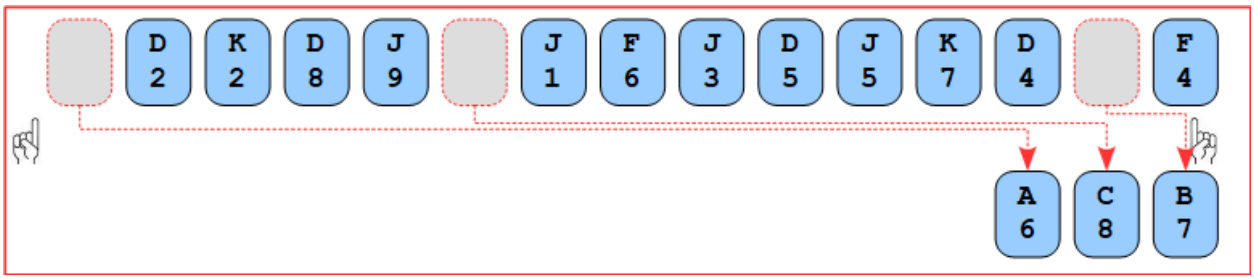
. . . encontrando el que tiene la clave "**c**", con lo que . . .



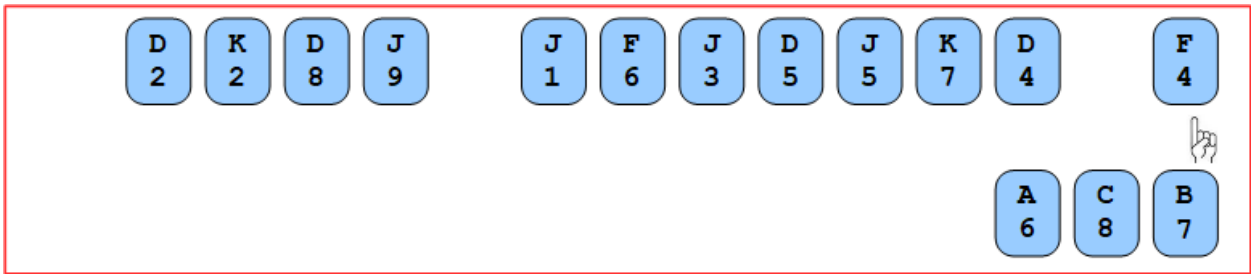
Una vez desvinculado, se continúa buscado hacia la izquierda el primero que ocurra una vez . . .



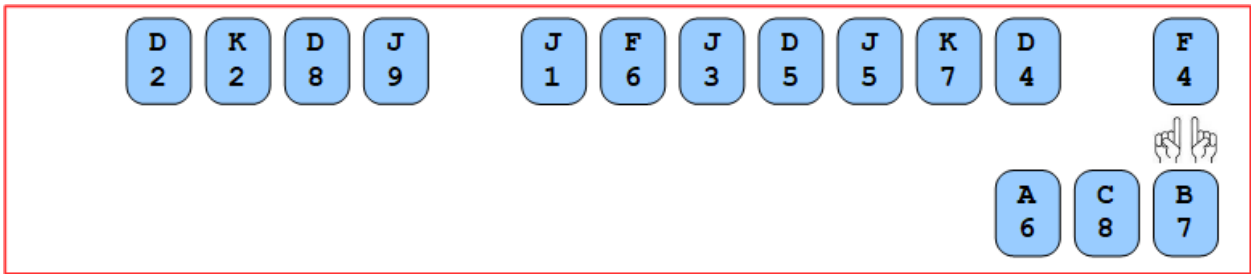
. . . encontrando el que tiene la clave "**A**", . . .



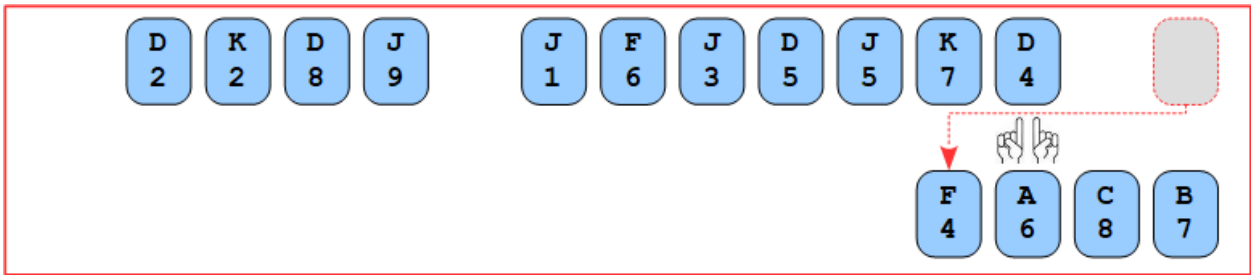
Al volver a buscar como no hay más a la izquierda o no hay más que tengan una sola ocurrencia, la lista queda . . .



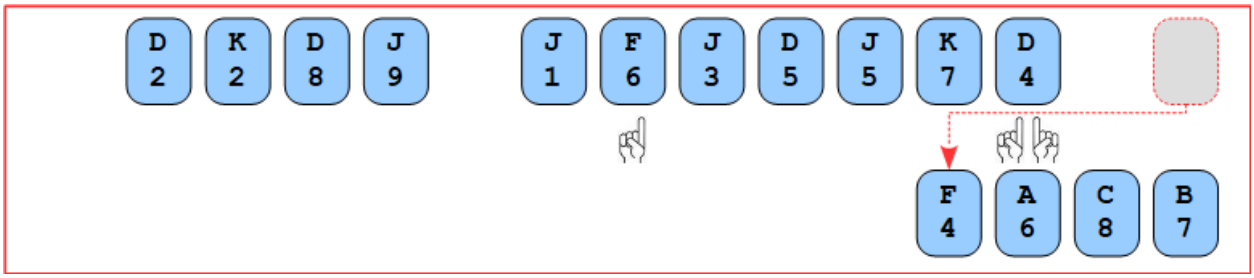
Y se procede a buscar desde el último de la derecha hacia la izquierda el primero que se encuentre hacia la derecha que esté dos veces:



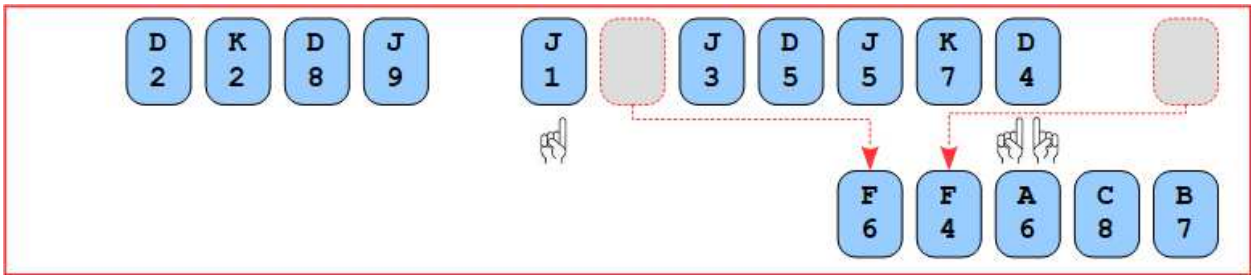
. . . encontrando el que tiene la clave "**F**", . . .



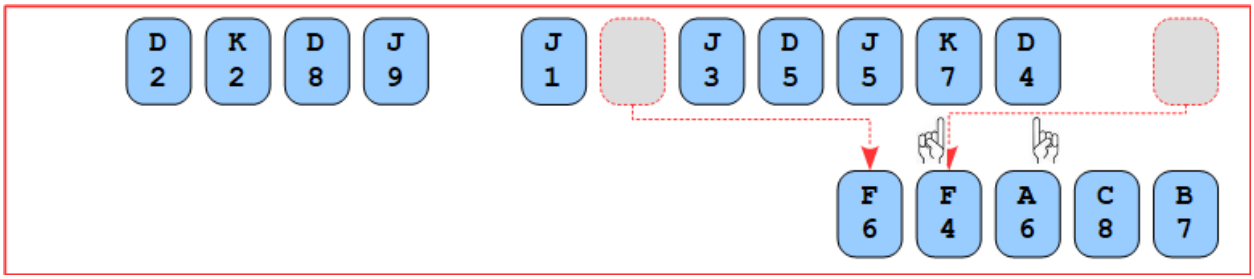
Y se procede a buscar el restante con esa clave, . . .



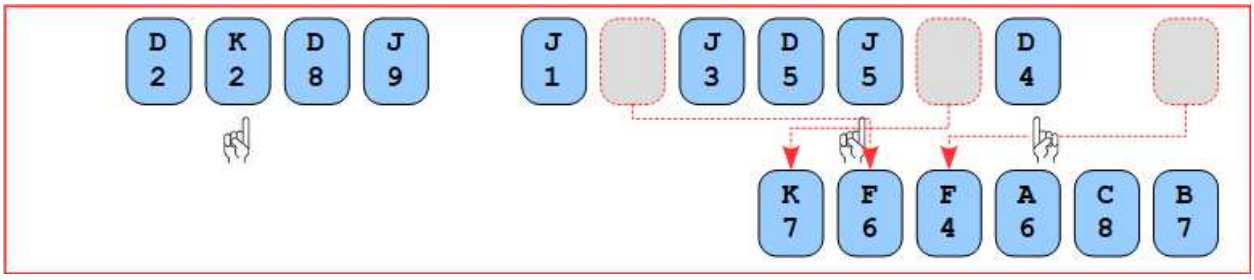
Procediendo a desvincularlo y ponerlo al comienzo de los ordenados.



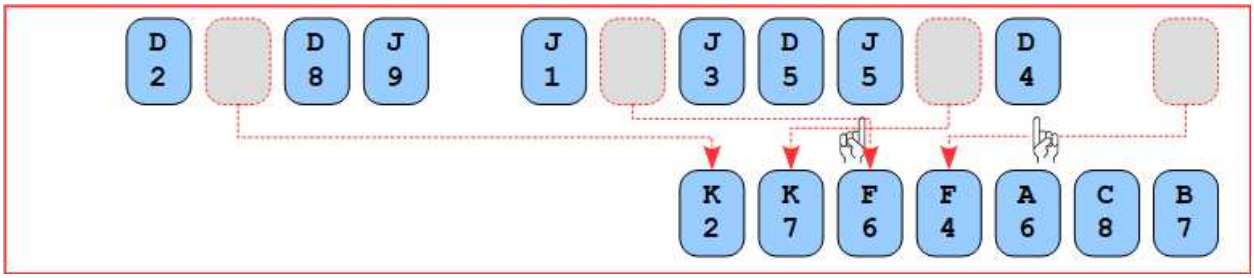
Y como ya se trataron las dos ocurrencias procede a buscar el último de la lista con dos ocurrencias, encontrando el que tiene la clave "**K**"



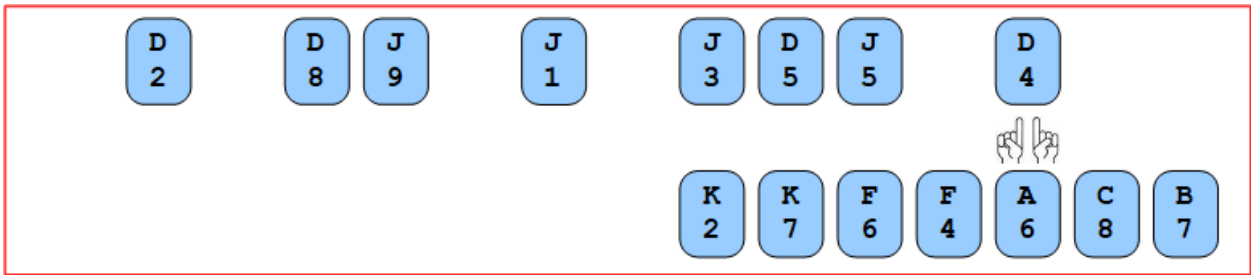
Procediendo a desvincularlo y ponerlo al comienzo de los ordenados. Sigue buscando el restante, encontrándolo . . .



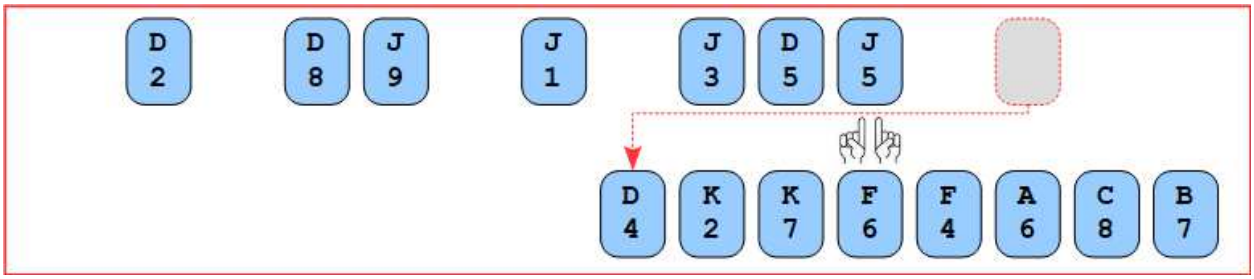
Lo desvincula de la lista poniéndolo al comienzo de los ordenados . . .



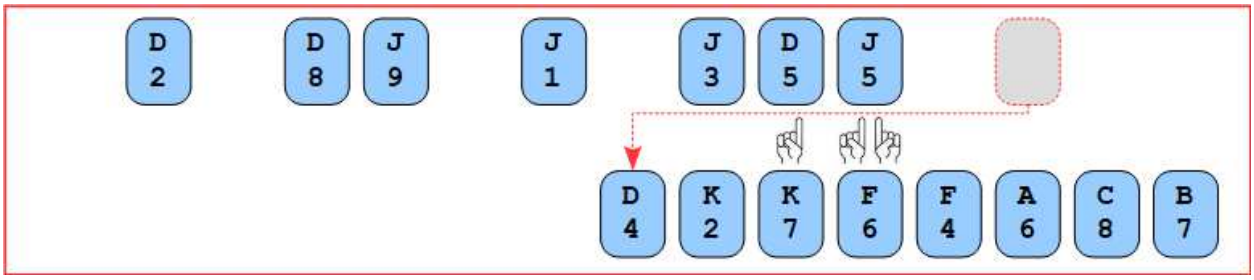
. . . y como al volver a buscar más nodos con dos ocurrencias, no hay más, comienza a buscar con tres ocurrencias. Al no haber con tres ocurrencias, procede a buscar con cuatro ocurrencias . . .



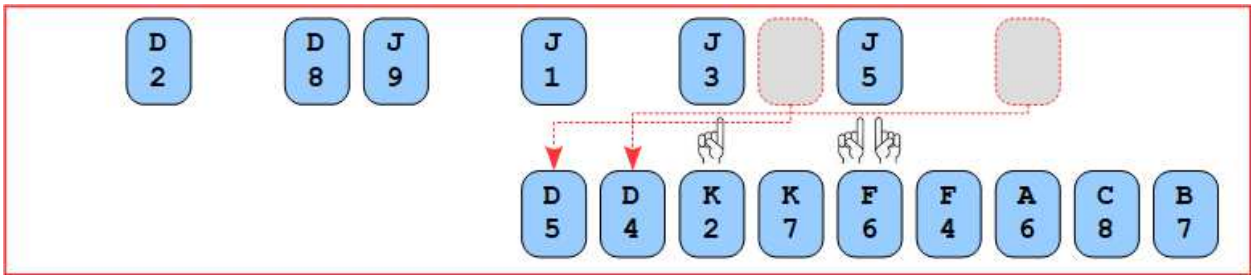
Al buscar desde el último de la derecha hacia la izquierda el primero (más a la derecha) que está cuatro veces, en este caso lo encuentra ahí, con lo que el resultado será:



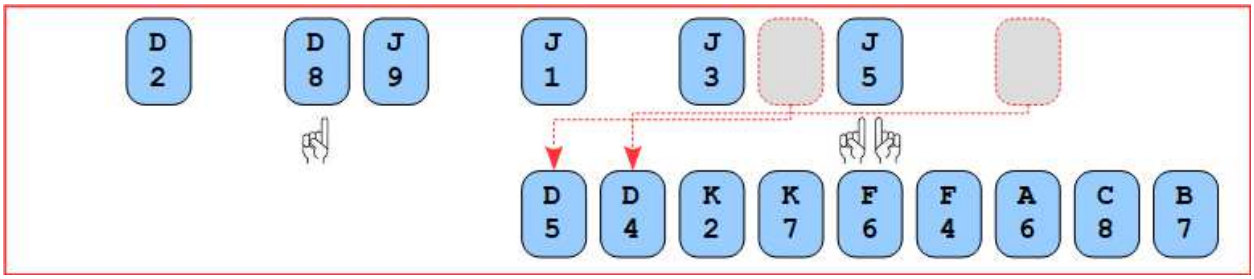
Al continuar buscando el segundo con clave "D" . . .



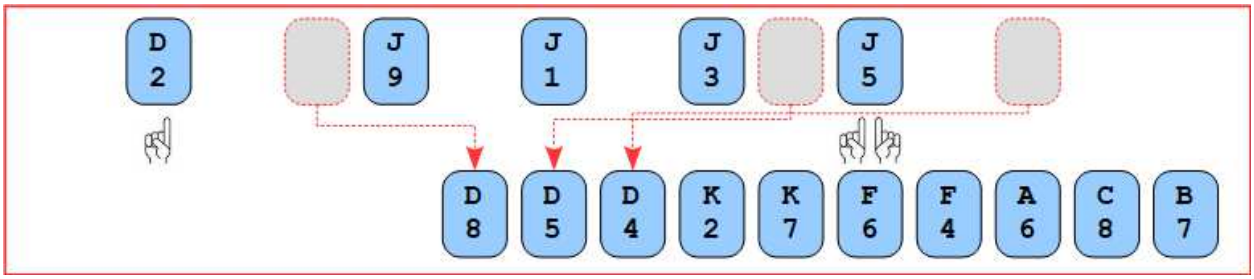
. . . resulta . . .



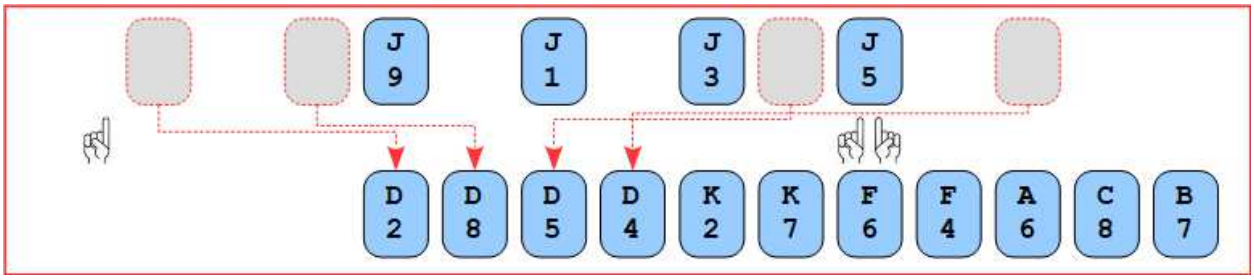
Continua buscando el tercero con clave "D" . . .



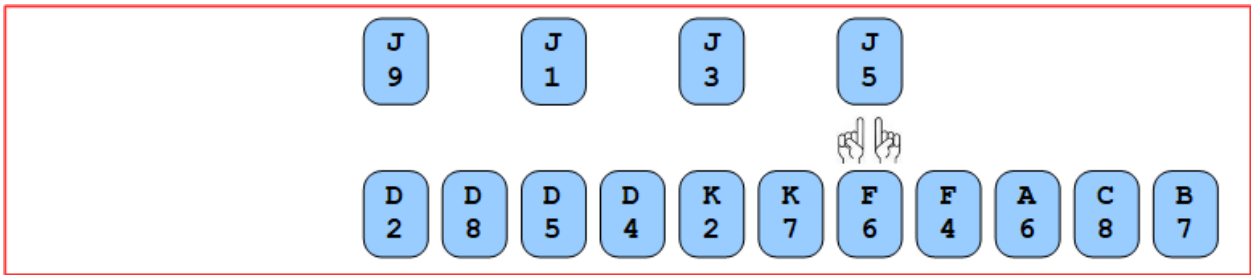
Lo desvincula y continúa buscando el cuarto . . .



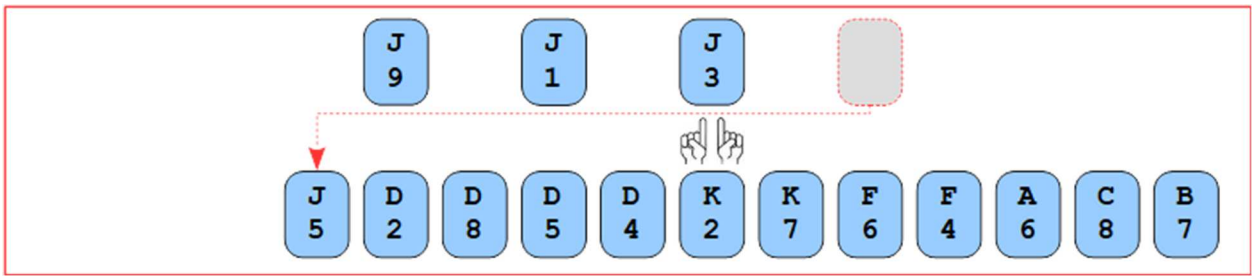
Lo desvincula . . .

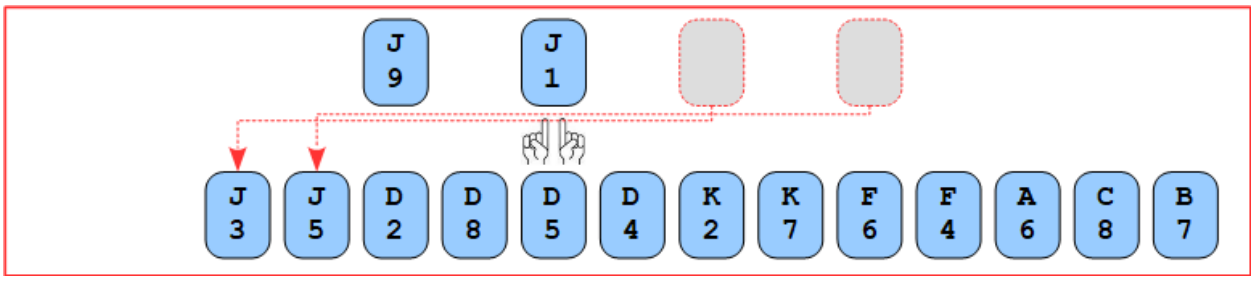


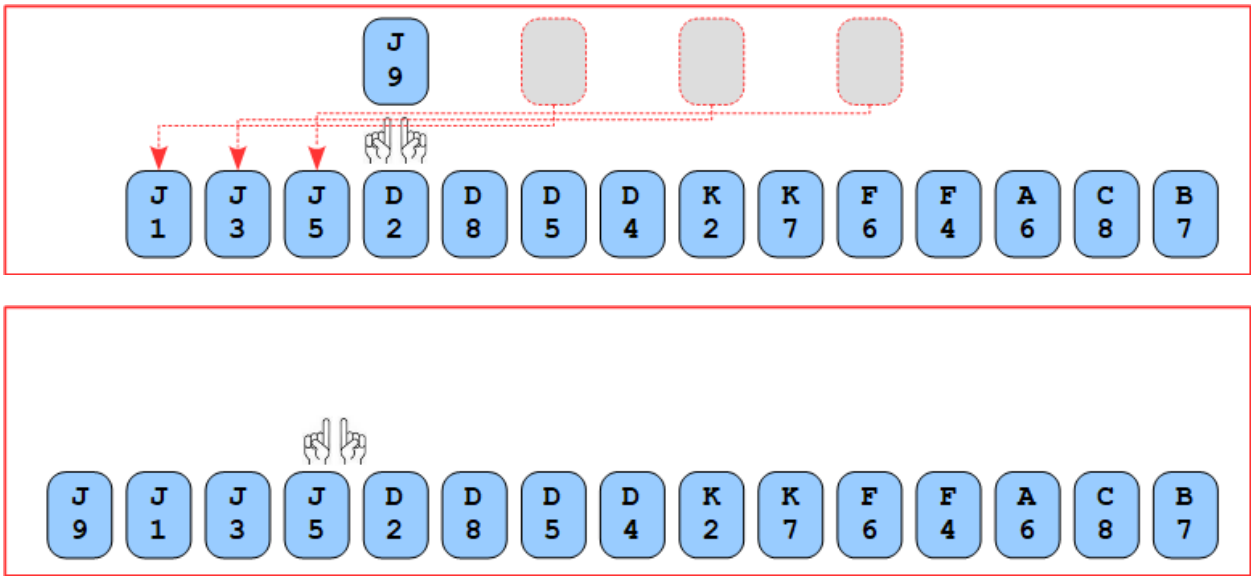
Resultando en:



Y se prepara a continuar buscando nuevamente con cuatro ocurrencias, resultando en la siguiente secuencia . . .







Si hubiera más nodos a la izquierda (con cuatro o más ocurrencias) continuaría del mismo modo.

La primera y la segunda secuencia de cuatro ocurrencias, le tendría que dar la clave de cómo resolver el algoritmo (teniendo en cuenta que contemple los casos de una sola ocurrencia y de que haya más nodos al comienzo, con lo que se pueden evitar un exceso de "desenganchar" nodos).

Lo hemos ayudado a entender el problema y además hemos esbozado una de las estrategias posibles con que lo debe resolver.

Deberá resolverlo respondiendo a esta estrategia.

No debe eliminar ni crear nodos, tan sólo debe reordenarlos.

En la función "**main**" sólo debe invocar a sus primitivas en los tres lugares indicados.

Ya sabe que el proyecto se entrega funcionando con las primitivas y funciones solicitadas totalmente operativas.

Su salida por pantalla (o en el archivo que se genera) deberá coincidir exactamente con la salida que produce el programa que le entregamos.

Su solución debe compilar sin advertencias ("**warnings**") y ejecutar correctamente.