Proyecto de Matemática Discreta II-2014

1 Introducción

Deberán implementar una API en C (de hecho, C99, i.e., pueden por ejemplo usar comentarios // u otras cosas de C99). [1] , la cual servirá para desarrollar un programa que cargará un network y calculará su flujo maximal utilizando DINIC. Este año el proyecto llevará una nota entre 0 y 10, la cual se promediara con las notas del práctico y del teórico del final para obtener la nota final de la materia. No pueden rendir el final mientras el proyecto no este aprobado por la cátedra, o si la nota del proyecto es menor a 4. El proyecto sera revisado por la catedra, y si no se lo considera aprobado será devuelto para ser corregido. Cada devolución implica un descuento de puntos de la nota final máxima que puede tener el proyecto. Cuando el total de puntos descontado sume mas de 6, ya no pueden volver a entregar el proyecto y no pueden rendir, debiendo recursar la materia. En la mayoria de los casos, cada devolución implica una perdida de 2 puntos, pero en algunos casos de errores menores puede ser menos, y en casos de errores graves, la perdida puede ser de hasta 6 puntos.

Deben hacer la API según las especificaciones indicadas abajo.

La API sera usada por uno o varios programas de la cátedra al cual uds. NO TENDRAN ACCESO, asi que deben asegurarse que su API cumpla las especificaciones.

Ademas de la API deben entregar un main.c que use la API para calcular el flujo maximal. Se evaluará por separado la API y el main.c.

Deben entregar un archivo API.h en el cual se hagan las llamadas que uds. consideren convenientes a librerias generales, ademas de las especificaciones de los tipos de datos y funciones descriptas abajo. El programa que la cátedra usará para testear su programa solo tendrá includes de stdio, la stdlib, stdint.h y API.h por lo cual, por ejemplo, si van a usar string.h el include lo deben escribir uds. en la API. No es necesario (ni conveniente) que todo este definido en API.h, pero si definen algo en otros archivos auxiliares, ademas de (obviamente) entregarlos, la llamada a esos archivos auxiliares debe estar dentro de API.h. En API.h deben figurar (comentados, obvio) los nombres de todos los integrantes del grupo, junto con sus mails.

A continuación se detalla la especificación de la API, y la documentación que se debe entregar junto al código. Su programa DEBE RESPONDER a estas especificaciones.

2 Especificación de la API

2.1 Tipos de datos

La API deberá proveer los siguientes tipos de datos.

2.1.1 u64:

se utilizará el tipo de dato `u64` para especificar un entero de 64 bits sin signo. Todos los enteros sin signo de 64 bits que aparezcan en la implementación deberán usar este tipo de dato. Uds deben definir u64 en API.h.

Los networks seran tales que el valor del flujo maximal siempre cabrá en un u64 y todas las capacidades serán u64.

[1] En palabras del gran Chun: "La elección de C para el proyecto se debe a que al ser un lenguaje de bajo nivel es más evidente la complejidad del algoritmo y su implementación. En un lenguaje de más alto nivel, como Python, mucha de esta complejidad se haría invisible y se perderían gran cantidad de detalles que son importantes tener en cuenta."

2.1.2 DovahkiinSt:

Es una estructura, la cual debe contener toda la información necesaria para correr Dinic. La definición interna de la esta estructura es a elección de ustedes y deberá soportar los métodos que se describirán más adelante, más los métodos que ustedes consideren necesarios para implementar Dinic. Entre los parametros debe haber como mínimo los necesarios para guardar los datos de un network pero ademas los necesarios para guardar el flujo que se tiene hasta ese momento en el network y cualquier información requerida en las busquedas de caminos aumentantes, como por ejemplo las etiquetas de distancias. Tambien debe haber un parametro que indique cuantos caminos aumentantes se han usado hasta el momento y si el flujo es maximal o no. Puede o no haber un parametro interno que guarde el valor del flujo, o este lo pueden calcular cuando lo necesiten. Como quieran. Basicamente, debe tener lo que uds. consideren necesario para poder implementar las funciones descriptas abajo.

Una forma posible de definir la estructura es por ejemplo:

```
typedef struct NetworkYFlujo{
    //aquí van los parámetros específicos de cada equipo
} DovahkiinSt;
```

2.1.3 DovahkiinP:

es un puntero a una estructura de datos *DovahkiinSt*. Esto se define de la siguiente forma:

```
typedef DovahkiinSt *DovahkiinP;
```

2.1.4 Lado

Será una estructura definida por ustedes que representará un lado. La información mínima que esta estructura debe guardar es al menos los nombres de los dos vertices y la capacidad, pero tambien pueden elegir guardar en esta estructura el flujo por el lado, o cualquier otra cosa que crean útil.

2.1.5 LadoNulo

LadoNulo sera un elemento de tipo Lado, que representa el hecho que en realidad no es un lado. Pueden representarlo como quieran, pero debe ser una estructura de tipo Lado, y debe ser tal que nunca esa estructura particular pueda estar representando un lado real.

2.2 Funciones

Se deben implementar las siguientes funciones.

2.2.1 NuevoDovahkiin()

Prototipo de función:

```
DovahkiinP NuevoDovahkiin();
```

La función aloca memoria e inicializa una estructura DovahkiinSt y devuelve un puntero a ésta. En caso de error, la función devolverá un puntero a `null`.

2.2.2 DestruirDovahkiin()

Prototipo de función:

```
int DestruirDovahkiin(DovahkiinP D);
```

Destruye D y libera la memoria alocada. Retorna 1 si todo anduvo bien y 0 si no.

2.2.3 FijarFuente()

Prototipo de función:

```
void FijarFuente(DovahkiinP D,u64 x);
```

Fija internamente que el número x es la fuente s.

2.2.4 FijarResumidero()

Prototipo de función:

`void FijarResumidero(DovahkiinP D,u64 x);`

Fija internamente que el número x es el resumidero t.

2.2.5 ImprimirFuente()

Prototipo de función:

`int ImprimirFuente(DovahkiinP D);`

Si la fuente esta fijada, returna 0 e imprime en standard output lo siguiente:

Fuente: x

donde x es el número (vertice) que estamos considerando como fuente.

Si la fuente no esta fijada devuelve -1.

2.2.6 ImprimirResumidero()

Prototipo de función:

`int ImprimirResumidero(DovahkiinP D);`

Si el resumidero esta fijado, retorna 0 e imprime en standard output lo siguiente:

Resumidero: x

donde x es el número (vertice) que estamos considerando como resumidero.

Si el resumidro no esta fijado devuelve -1.

Nota: en estas funciones se imprime el "nombre" real de s y t. En todas las otras impresiones se imprimirá "s" y "t" en vez de sus valores reales.

2.2.7 LeerUnLado()

Prototipo de función:

`Lado LeerUnLado();`

La función lee una linea desde **standard input** que representará un lado y devuelve el elemento de tipo Lado que representa ese lado, si la linea es válida, o bien LadoNulo si la linea no es válida.

Cada linea válida del formato de entrada será de la siguiente forma:

`x y c`

lo cual representa un lado $\overrightarrow{xy}$ con capacidad c.

x,y,c serán u64.

Como dijimos arriba, si la linea no es válida, la función debe retornar LadoNulo.

(nota: la cátedra no testeará lineas no válidas cuya no validez provenga de tener mas de un espacio entre x,y,c. Es decir, toda linea que proveamos será, o bien válida, o bien no válida por ser por ejemplo una linea sin números, o bien EOF, o bien una sucesión de carácteres no númericos, o bien números que no serán u64 (por ejemplo, negativos), etc, pero no testearemos con lineas de la forma, por ejemplo x y c. Uds pueden, o no, tomar ventaja de este conocimiento).

x e y serán los nombres "reales" de los vertices dados de alguna forma en el input. Internamente uds. los pueden representar como quieran, pero cuando impriman las salidas deben imprimir con el nombre real.

EXCEPCION: En las salidas la fuente s y el resumidero t siempre deben decir s y t independiente de que número sean. (salvo, obviamente, en las funciones ImprimirFuente e ImprimirResumidero, donde se imprimen los valores reales).

2.2.8 CargarUnLado()

Prototipo de función:

`int CargarUnLado(DovahkiinP D,Lado L);`

La función carga el lado L en `D`.

Debe inicializar el flujo en el lado cargado como cero, si es que esa información no esta contenida en la estructura Lado. La función retorna 1 si no hubo problemas y 0 en caso contrario.

2.2.9 Prepararse()

Prototipo de función:

```
int Prepararse(DovahkiinP D);
```

La función será llamada una vez que todos los lados del network hayan sido cargados, y debe realizar cualquier procesamiento de los mismos que sean necesarios para dejar a `D` listo para empezar a buscar caminos aumentantes. Dependiendo de la estructura que usen y de como definan CargarUnLado, esta función puede hacer casi nada, pero algunos grupos necesitaran un preprocesamiento de la estructura antes de estar listos para empezar las busquedas, de ahi la inclusión de esta función en la API.

Una cosa que si debe hacer aun cuando no deba hacer nada es siempre verificar que todo este listo, en particular, antes de empezar la busqueda se debe saber quien es la fuente y quien el resumidero.

Devuelve 1 si termina de preparase y 0 si no puede (por ejemplo, porque la fuente no fue fijada).

2.2.10 ActualizarDistancias()

Prototipo de función:

```
int ActualizarDistancias(DovahkiinP D);
```

La función actualiza (usando una busqueda BFS-FF, como en el teórico) las etiquetas de distancias de todos los vertices. Devuelve 1 si existe un camino aumentante entre s y t (i.e. la distancia de s a t es menor a $n+1$) y 0 si no. (i.e., si el flujo es maximal).

2.2.11 BusquedaCaminoAumentante()

Prototipo de función:

```
int BusquedaCaminoAumentante(DovahkiinP D);
```

La función hace una busqueda FF-DFS de un camino aumentante de menor longitud usando las etiquetas dadas por la función anterior. Debe actualizar en D cualquier dato que uds. hayan decidido crear asociadas a la busqueda.

Devuelve 1 si se llega a t, 0 si no.

Observar que esta función NO aumenta el flujo, solo busca un camino aumentante y actualiza cualquier dato interno necesario para poder usarlo luego. Por ejemplo, pueden tener en la estructura un buffer donde guardarán el camino encontrado, (memory inefficient) o bien pueden tener etiquetas asociadas a los vertices que permitan reconstruir el camino cuando se lo necesite.

2.2.12 AumentarFlujo()

Prototipo de función:

```
u64 AumentarFlujo(DovahkiinP D);
```

Precondición: se debe haber efectuado una BusquedaCaminoAumentante que llegó a t pero que todavia no se ha usado para actualizar el flujo. Esta condición debe ser chequeada por la función. (con lo cual deberán tener en la estructura `D` algun campo que guarde esta información).

La función actualiza el flujo en el network `D`, para lo cual lee los datos internos de `D` que le permitan reconstruir el camino aumentante y cambia el flujo a lo largo de ese camino como se vio en clase. (Si hay un parametro interno que guarde el valor del flujo, tambien lo actualiza)

Valor de retorno: el valor por el cual se aumenta el flujo, o 0 si hubo algún error, en particular si no se cumple la precondición.

2.2.13 AumentarFlujoYtambienImprimirCamino()

Prototipo de función:

```
u64 AumentarFlujoYtambienImprimirCamino(DovahkiinP N);
```

La función hace lo mismo que AumentarFlujo y tiene la misma precondición, pero ademas imprime el camino aumentante y el aumento a lo largo de el.

La impresión debe ser siempre por **standard output**.

Se imprime el camino aumentante y su aumento en el siguiente formato:

```
camino aumentante #:
t;x_r;...;x_1;s:  <cantDelIncremento>
```

donde # será el número del camino aumentante que se esta procesando (1,2,etc). El punto y coma (;) entre vertices va cuando el lado es forward. Si el lado es backward en vez de ";" deben escribir `>`. Por ejemplo: `t;11>100>2;8;7>4;3;s` es un camino con lados forward (s,3),(3,4), (7,8), (8,2) y (11,t) y lados backward (4,7), (2,100), (100,11).

Presten atención que si bien en el input s y t serán dos números, la salida debe decir s y t y no esos números.

Valor de retorno: el valor del aumento o 0 si hubo algún error, en particular si no se cumple la precondición.

2.2.14 ImprimirFlujo()

Prototipo de función:

```
void imprimirFlujo(DovahkiinP D);
```

Imprime el **flujo** que se tiene en ese momento **La impresión debe ser siempre por standard output**.

Debe imprimir:

```
Flujo †:
Lado x_1,y_1:  <FlujoDelLado>
...
Lado x_m,y_m:  <FlujoDelLado>
```

donde † es `Maximal` si el flujo es maximal y `(no maximal)` si no lo es.

El orden de los lados es a elección de uds.

2.2.15 ImprimirValorFlujo()

Prototipo de función:

```
void ImprimirValorFlujo(DovahkiinP D);
```

Imprime el **valor** del flujo. **La impresión debe ser por standard output**. Debe imprimir:

```
Valor del flujo †:  <valorDelFlujo>
```

donde como antes, † es `Maximal` si el flujo es maximal y `(no maximal)` si no lo es.

2.2.16 ImprimirCorte()

Prototipo de Función:

```
void ImprimirCorte(DovahkiinP D);
```

Debe imprimir por standard output un corte minimal del network y su capacidad en el siguiente formato:

```
Corte Minimal:  S = {s,x_1,...}
Capacidad:  <capacidad>
```

(el orden en que quieran poner los vertices de S es arbitrario, no es necesario que empiece en s).

2.3 Main

Ademas de la API, deben entregar un main.c en donde se use la API para leer un network desde standard input, donde la entrada será como es descripta arriba (y donde la carga debe finalizar apenas se encuentre una linea no válida) y que calcule un flujo maximal y su valor. main.c debe siempre imprimir al menos el valor del flujo maximal, y debe tener opciones de entrada que permitan como mínimo decidir al que llama la función si quiere ademas imprimir los caminos aumentantes o no, imprimir el flujo maximal o no, imprimir el corte minimal o no, y declarar quien es la fuente y quien el resumidero. Sin embargo NO debe ser interactivo. (i.e., no debe ser tal que escriba cosas como "escriba si quiere imprimir o no", "escriba la fuente", etc.) Simplemente debe tener ciertos argumentos con los cuales se pueda especificar todo.

Es decir, debemos poder correrlo de la siguiente forma, asumiendo que lo compilamos y lo pusimos en ejecdinic:

```
./ejecdinic [parametros en el formato que ustedes especifiquen] <archivoconunnetwork
```

El main.c y la API serán testeados por separado, i.e., usaremos su API con un main nuestro, y su main con una API nuestra para ver si son compatibles con las especificaciones.

2.4 Suposiciones que pueden o no hacer sobre el network

Se asegura (por nosotros) que en los tests no habra lados repetidos ni loops.

Que se agregue el vértice v al network no implica que se agregue el vértice v' tal que $v' < v$. Por ejemplo, los vertices pueden ser solo cinco, y ser $0, 1, 2, 15768, 21234567$.

3 Cosas para tener en cuenta

- El uso de macros esta permitido pero como siempre, sean cuidadosos si los usan.
- Debe haber MUCHO COMENTARIO, las cosas deben entenderse. En particular, en cada funcion o macro que usen debe haber una buena explicación, entendible, de lo que hace. Ademas cada variable que se use debe tener un comentario al lado indicando su significado y para que se usará. Tambien seria bueno que los nombres sean mas o menos significativos. (por ejemplo:

  ```
  int i,j; //indices
  u64 vflujo; //guarda el valor del flujo
  bool Stop; //una booleana que sera 0 si...  y 1 si...
  ```

  etc;
- El código debe estar bien modularizado. Por ejemplo, el siguiente pseudo-código es una mala modularización de la función `FuncionEjemplo()`:

  ```
  int FuncionEjemplo(DovahkiinP D){
      if (D->camp1 = 0xff){
          MUCHO CODIGO PARA ESTE CASO
      } else {
          MUCHO OTRO CODIGO PARA ESTE OTRO CASO
      }
      return resultado;
  }
  ```

  El "`MUCHO CODIGO...`", en ambos casos, debería ser reemplazado por llamadas a funciones o macros.
- Presten atencion al formato especificado, no solo al de entrada sino al de salida: uds. deben asumir que otro grupo de personas usará la salida de su programa como input de algun programa propio, de ahi las especificaciones de salida.
- Como se dijo varias veces, la entrada y la salida son estandards. Imprimir a un archivo implica devolución automática del proyecto.
- No se pueden usar librerias externas que no sean las básicas de C. (eg, stdlib.h, stdio.h, strings.h, stdbool.h, assert.h etc, si, pero otras no). Pueden tomar ideas de ellas, pero su código debe ser lo mas portable posible, algunas librerias externas no siempre lo son, o son demasiado grandes. La experiencia de años anteriores indica que nos crea problemas.
- Su programa no debe usar nunca mas de 256 MB de memoria RAM.
- El network de entrada puede tener miles de vertices y lados. Testeen para casos grandes. (pero nunca va a haber millones de vertices)
- Su programa debe ser razonablemente rápido. Algunos lo harán mas rápido que otros, pero en particular no puede demorar horas (y menos aun dias) en hallar un flujo maximal.
- El algoritmo debe ser Dinic. En particular, no puede pasar que un camino de longitud 5 este luego de un camino de longitud 7, o usar el MISMO camino dos veces seguidas.
- Test, test, test. Inventen sus propios networks para testear.

4 Entrega

4.1 Protocolo

Deben entregar vía e-mail los archivos que implementan la API dada arriba y al menos un main.c. No debe haber ningun ejecutable. Tambien se deberá entregar un informe donde se explique y fundamente el desarrollo realizado y tambien se detalle, mas o menos, que hizo cada integrante del grupo.

El informe puede ser un .pdf o un README que sea leible con cualquier lector de textos ASCII (asi que no usen Word o FreeOffice o cualquier editor no ASCII). Pueden hacer tanto un .pdf como un README, por ejemplo, en el .pdf poner un informe detallado, y en el README algo mas corto.

El informe debe ser claro, y explicar las decisiones de diseño y las dificultades que encontraron y como las resolvieron. Nos interesa muy poco si usaron Valgrind o su programa favorito para esto o aquello, lo que interesa es que expliquen bien la estructura que usaron, como implementaron la busqueda, como implementaron $\Gamma^+(x)$, $\Gamma^-(x)$, $\Gamma_{f,d}(x)$ (si es que lo hicieron, y si no, como hicieron para buscar vecinos), porque usaron la estructura de datos que usaron, etc. Nos debe quedar claro que ustedes entienden su propio proyecto y no lo copiaron de algun lado (esta permitido tomar ideas de otros lados, pero debe quedar claro que las entienden y no que solo hicieron cut and paste). Algunos detalles del código nos pueden hacer sospechar que lo copiaron, en ese caso podemos requerir una exposición oral intensiva y si no nos convencen el proyecto se devuelve con 5 puntos de descuento.

En definitiva, el informe debe proveer **una especificación** del proyecto, tal que alguien, leyendo el informe, pudiera hacer una black room implementation de su proyecto sin tener que leer el código.

Deben empaquetar todo de la siguiente forma: Debe haber un directorio de primer nivel cuyo nombre consiste en las iniciales de los miembros del equipo.

En ese directorio deben poner el README o el .pdf (o ambos, si hacen los dos). Ademas, en ese directorio habrá dos directorios de segundo nivel, uno llamado apifiles y otro llamado dirmain. En apifiles deben estar todos los archivos de la API, incluidos archivos auxiliares, pero no debe haber ningun main. **Todo .c que este en apifiles se considera que debe formar parte de la compilación.** En dirmain debe estar su archivo main.c.
No hace falta un make. Nosotros compilaremos de la siguiente forma: si un grupo formado por Alvarez, Garcia, Gonzales, Rodriguez y Torres manda un proyecto, nos pararemos en AGGRT/dirmain y compilaremos asi:

```
gcc -Wall -Wextra -O3 -std=c99 main.c ../apifiles/*.c -o nombreejecutable
```

Empaqueten todo en formato .tgz.

Luego de enviado, se les responderá con un "recibido" y se les enviarán 5 networks (que cambiarán de grupo a grupo). Uds. deben encontrar el valor del flujo maximal para cada uno de esos networks usando su API y main.c y devolver esos 5 valores. Si los valores son correctos, corregiremos su proyecto, caso contrario el proyecto es devuelto automaticamente con descuento de 2 puntos.

4.2 Entrega

El 26 de Mayo deben entregar el proyecto.

Una vez entregado el proyecto, lo podemos aprobar o no. Si no lo aprobamos, se devolverá para que lo corrigan, con descuento de puntos. Pueden entregarlo tantas veces como quieran, pero cada vez suma puntos de descuento. Una vez que se hayan descontado 6 puntos, no pueden volver a entregar y el proyecto queda rechazado definitivamente.

4.2.1 Entrega Alternativa

Si no lo entregan en esa fecha o lo entregan y se los devolvemos, quedan libres.

Pueden reentregar cuantas veces quieran, pero no prometemos corregirlos a tiempo para la primera fecha de examenes si las entregas son despues del 26 de Mayo.

Para rendir en la segunda fecha de examen deben entregar antes del 12 de Junio. Para rendir en la tercera fecha de examen deben entregar para el primer lunes despues de las vacaciones de julio. (en las vacaciones de Julio no revisaremos proyectos).

Para rendir en Noviembre-Diciembre-Febrero-Marzo tienen que entregarlo como máximo el 30 de Septiembre. Luego de esa fecha no se aceptan mas proyectos. Proyectos entregados en Septiembre

que sean devueltos en principio pierden la posibilidad de rendir en Diciembre, salvo que el error sea menor.

5 Devolución del proyecto y Descuento de Puntos

En general la experiencia demuestra que algunos alumnos son muy creativos a la hora de cometer errores, asi que es imposible para nosotros listar todos los errores posibles que puedan cometer.

La nota por default del proyecto es 10, es decir, no jusgaremos calidades distintas de proyectos. Sin embargo, a partir del 10, descontaremos puntos por fallas en el proyecto, en algunos casos devolviendo el proyecto.

Si juzgamos que los comentarios no son los suficientes o no lo suficientemente claros, les descontaremos entre 0,5 puntos y 1 punto, y podemos devolver el proyecto requiriendo que agreguen mas comentarios.

Idem si consideramos que el informe no es claro. Si mandan el informe en un formato que no es el que especificamos les descontaremos 4 puntos por idiotas.

La siguiente lista de errores de programación, que han ocurrido previamente, implica una devolución automatica del proyecto, con descuento de puntos.

1. No compila. Aca hay dos casos. Si no compila con su main y su API, les descontaremos 2 puntos.

   Si no compila cuando lo intentamos con nuestro main y su API (o al reves) entonces es un error de compilación por incompatibilidad entre su API y nuestro main. Esto solo ocurre si ustedes no respetan las especificaciones. Cosas que pueden ocurrir (y ocurrieron en otros años) es que no tipeen bien las funciones de la API (descontaremos 0,5 puntos), que sus funciones usen mas argumentos o argumentos de distinto tipo que los especificados o retornen valores que no son los especificados (descontaremos entre 2 a 3 puntos) o bien que hagan cosas completamente ridiculas, que es dificil listar aqui pero han ocurrido. (descontaremos entre 3 y 4 puntos).

2. Valor del flujo maximal no es el correcto. -2 puntos si el valor calculado es incorrecto, -1 punto si es sólo un error de impresión.

3. El valor del flujo maximal no es igual a la capacidad del corte minimal. Si el valor del flujo maximal es correcto, -1 punto. Si no, se aplica lo enunciado arriba.

4. Los caminos no cumplen la propiedad de que la longitud del camino $k+1$ debe ser mayor o igual que la del camino k: -1 punto.

5. Se usa un mismo camino mas de una vez. -1 punto.

6. Hay memory leaks. -1 punto.

7. Runtime error. (por ejemplo, segmentation fault, o peor aún, clava la maquina). Dependiendo del error, entre -1 y -2 puntos.

8. Error de impresión de la salida. entre -0,5 puntos si es un error muy menor, a -2 puntos.

9. No acepta inputs válidos. (por ejemplo, es incapaz de leer desde standard input, o bien espera un formato que no es el que especificamos) -2 puntos.

10. Errores de carga en los datos: continua cargando mas alla de una linea no válida, o carga mal algunos lados, o directamente no los carga, o carga mal las capacidades o inicializa mal algo. Entre -1 y -2 puntos.

11. El flujo maximal esta bien pero los caminos aumentantes estan mal. Por ejemplo, en otros años han pasado cosas tales como que se imprimen caminos aumentantes con vertices que no existen o caminos aumentantes que no son tales o caminos aumentantes que ninguna implementación de Dinic podría elegir.

    (-1 punto si es un error tonto del formateo de la impresión, -2 si es algo mas grave)

12. El flujo maximal devuelto NO ES FLUJO. -3 puntos.

13. Implementan un algoritmo que no es Dinic. -4 a -6 puntos.