Administración de E/S

1. Dispositivos.

(a) Los dispositivos, según la forma de transferir los datos, se pueden clasificar en 2 tipos:

- **Orientado a bloques**: estos dispositivos transmiten datos en bloques (paquetes) y por esa razón son usados a menudo para la transmisión paralela de datos utilizando el buffer de datos del S.O. Almacenan la información en bloques de tamaño fijo, cada uno con una dirección propia (permite leer, escribir o buscar un bloque sin dependencia de los demás).
    - Por ejemplo: Disco magnético.
- **Orientado a flujos (de caracteres)**: utilizan transmisión en secuencia de datos sin usar buffer, transmitiendo un bit o un byte a la vez. Aceptan o entregan un flujo de caracteres sin considerar estructuras de bloques. No son direccionables y, por tanto, no permiten operaciones de búsqueda.
    - Por ejemplo: Impresora, terminales, cintas de papel, interfaz de redes.

Describa las diferencias entre ambos tipos.

(b) Cite ejemplos de dispositivos de ambos tipos.

(c) Enuncie las diferencias que existen entre los dispositivos de E/S y que el SO debe considerar.

- Heterogeneidad de dispositivos
- Características de los dispositivos
- Velocidad
- Nuevos tipos de dispositivos
- Diferentes formas de realizar E/S

2. Técnicas de E/S Describa como trabajan las siguientes técnicas de E/S

- **E/S programada**: El procesador envía un mandato de E/S, a petición de un proceso, a un módulo de E/S; a continuación, ese proceso realiza una espera activa hasta que se complete la operación antes de continuar.

- **E/S dirigida por interrupciones**: El procesador emite un mandato de E/S a petición de un proceso y continúa ejecutando las instrucciones siguientes, siendo interrumpido por el módulo de E/S cuando éste ha completado su trabajo. Las siguientes instrucciones pueden ser del mismo proceso, en el caso de que ese proceso no necesite esperar hasta que se complete la E/S. En caso contrario, se suspende el proceso en espera de la interrupción y se realiza otro trabajo.

- **DMA (Acceso Directo a Memoria)**: Un módulo de DMA controla el intercambio de datos entre la memoria principal y un módulo de E/S. El procesador manda una petición de transferencia de un bloque de datos al módulo de DMA y resulta interrumpido sólo cuando se haya transferido el bloque completo.

	Sin interrupciones	Con interrupciones
Transferencia de E/S a memoria a través del procesador	E/S programada	E/S dirigida por interrupciones
Transferencia directa de E/S a memoria		Acceso directo a memoria (DMA)

3. La tecnica de E/S programa puede trabajar de dos formas:

- **E/S mapeada**: Los dispositivos y memoria comparten el espacio de direcciones. I/O es como escribir/leer en la memoria por lo que no hay instrucciones especiales para I/O (ya se dispone de muchas instrucciones para la memoria)
- **E/S aislada**: Hay un espacio separado de direcciones para I/O. Se necesitan líneas de I/O. Puertos de E/S. Instrucciones especiales (conjunto limitado)

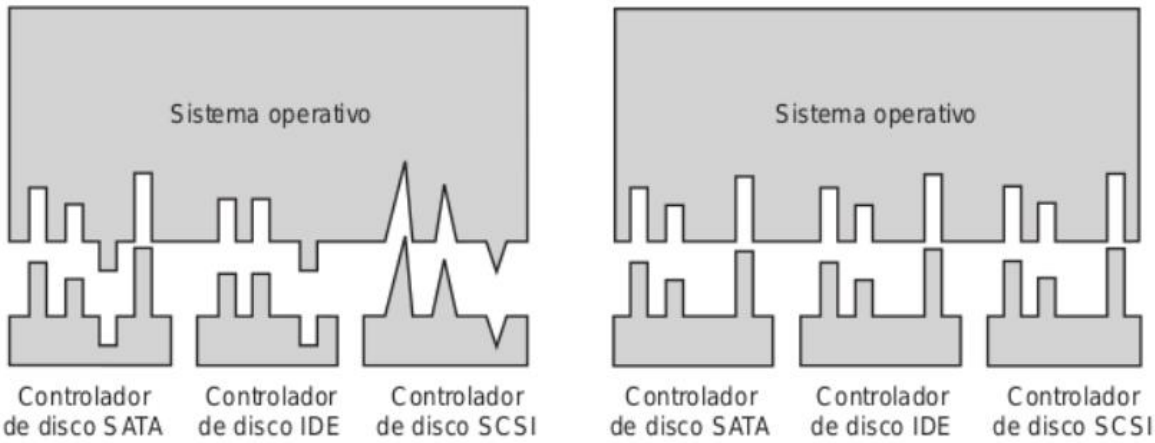
Indique como trabajan estas 2 técnicas.

4. Enuncie las metas que debe perseguir un SO para la administración de la entrada salida.

Generalidad:

- Es deseable manejar todos los dispositivos de I/O de una manera uniforme, estandarizada.
- Ocultar la mayoría de los detalles del dispositivo en las rutinas de niveles más “bajos” para que los procesos vean a los dispositivos, en términos de operaciones comunes como: read, write, open, close, lock, unlock.

Interfaz Uniforme:



Eficiencia:

- Los dispositivos de I/O pueden resultar extremadamente lentos respecto a la memoria y la CPU.
- El uso de la multi-programación permite que un procesos espere por la finalización de su I/O mientras que otro proceso se ejecuta.

Planificación:

- Organización de los requerimientos a los dispositivos
    - Ej: Planificación de requerimientos a disco para minimizar tiempos

5. Drivers

(a) ¿Qué son?

Consiste en una interfaz entre el SO y el HARDWARE cargadas como módulos en el espacio de memoria del kernel conteniendo el código dependiente del dispositivo para manejar el mismo y traducir requerimientos abstractos en los comandos para el dispositivo manejado.

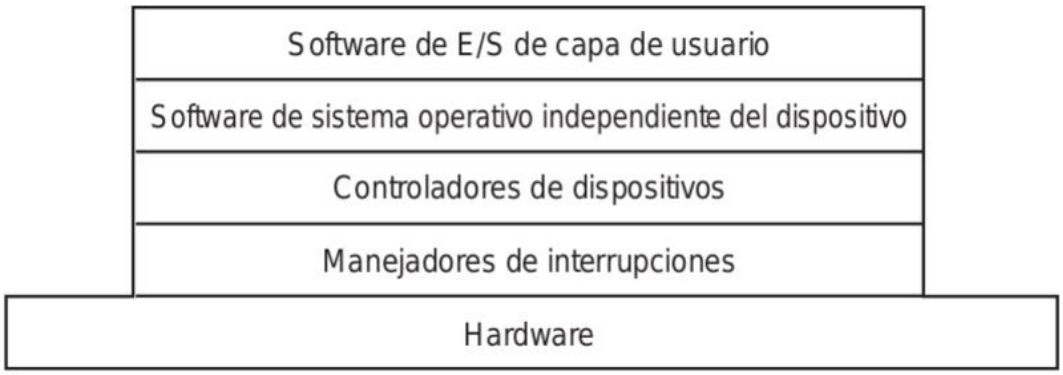
(b) ¿Qué funciones mínimas deben proveer?

- init_module: Para instalarlo
- cleanup_module: Para desinstalarlo.

(c) ¿Quién determina cuales deben ser estas funciones?

El fabricante

6. Realice un grafico que marque la relación entre el Subsistema de E/S, los drivers, los controladores de dispositivos y los dispositivos.



7. Describa mediante un ejemplo los pasos mínimos que se suceden desde que un proceso genera un requerimiento de E/S hasta que el mismo llega al dispositivo

Consideremos la lectura sobre un archivo en un disco:

- Determinar el dispositivo que almacena los datos
    - Traducir el nombre del archivo en la representación del dispositivo.
- Traducir requerimiento abstracto en bloques de disco (Filesystem)
- Realizar la lectura física de los datos (bloques) en la memoria
- Marcar los datos como disponibles al proceso que realizo el requerimiento
- Desbloquearlo

- Retornar el control al proceso

9. Enuncie que servicios provee el SO para la administración de E/S.

Buffering – Almacenamiento de los datos en memoria mientras se transfieren.

- Solucionar problemas de velocidad entre los dispositivos.
- Solucionar problemas de tamaño y/o forma de los datos entre los dispositivos.

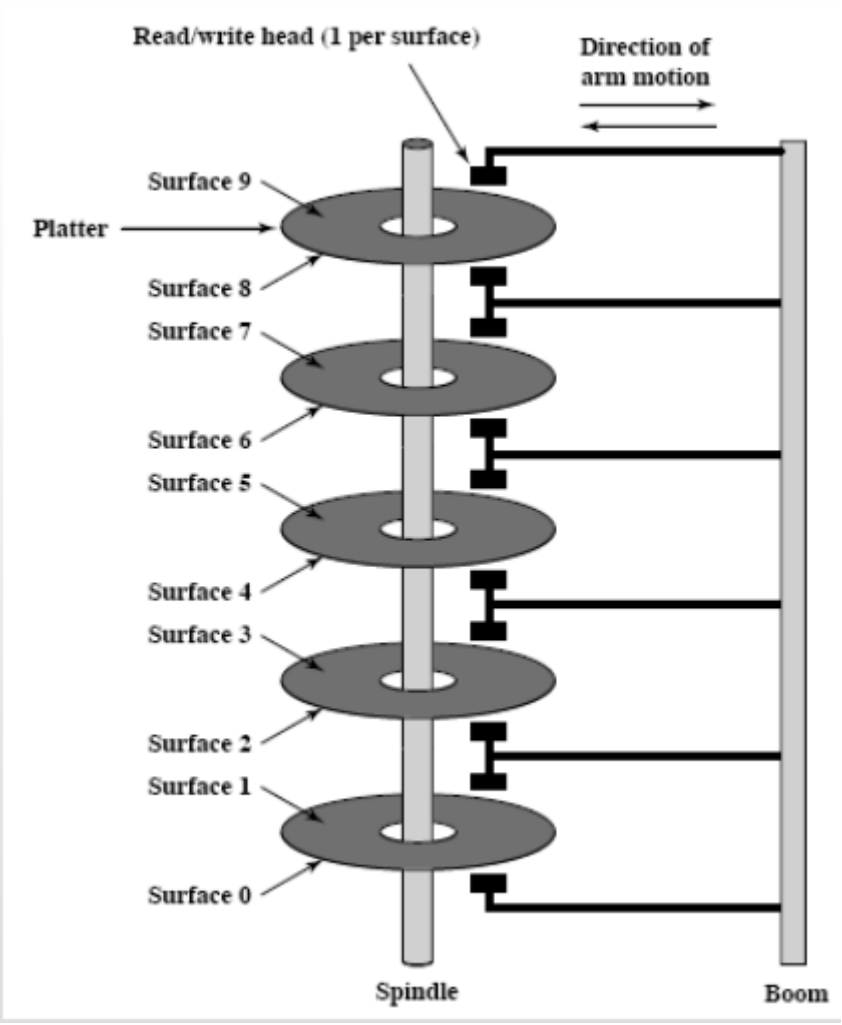
Caching – Mantener en memoria copia de los datos de reciente acceso para mejorar performance.

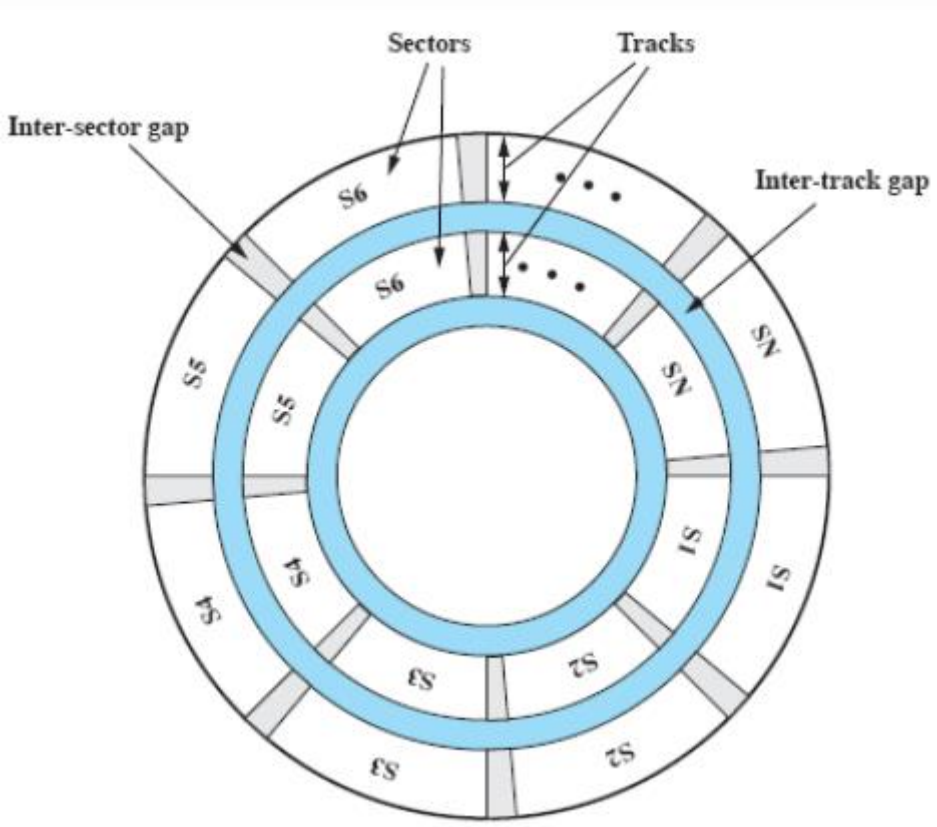
Spooling – Administrar la cola de requerimientos de un dispositivo.

- Algunos dispositivos de acceso exclusivo, no pueden atender distintos requerimientos al mismo tiempo: Por ej. Impresora.
- Spooling es un mecanismo para coordinar el acceso concurrente al dispositivo.

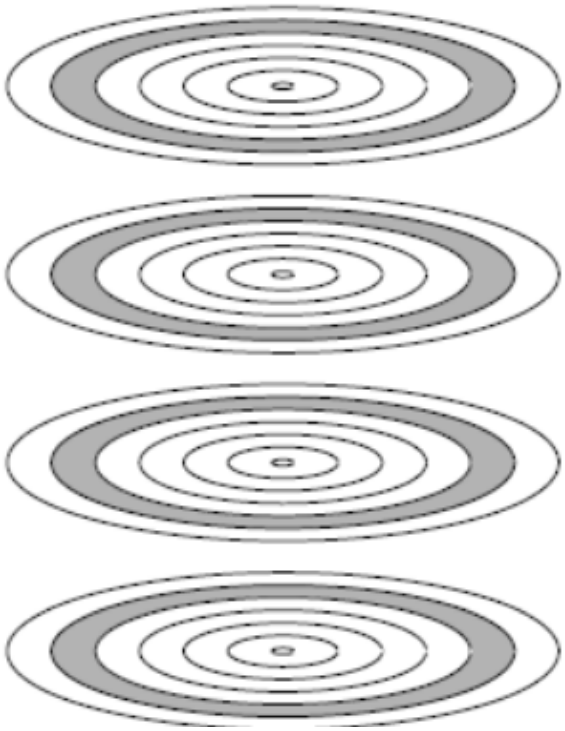
Administración de Discos

10. Describa en forma sintética, cómo es la organización física de un disco, puede utilizar gráficos para mayor claridad.





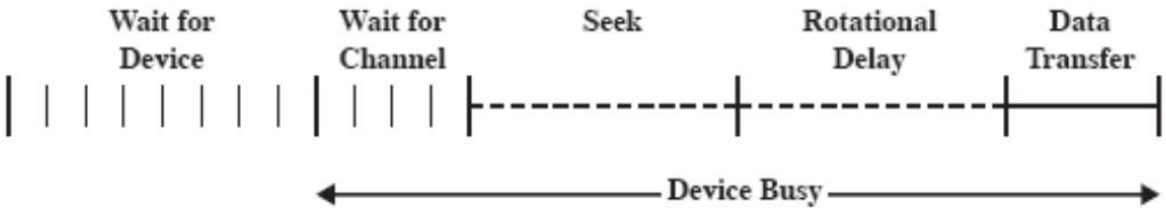
- Cilindro N: todas las n-esimas pistas de todas las caras



11. La velocidad promedio para la obtención de datos de un disco está dada por la suma de los siguientes tiempos:

- **Seek Time** (posicionamiento): tiempo que tarda en posicionarse la cabeza en el cilindro
- **Latency Time** (latencia): tiempo que sucede desde que la cabeza se posiciona en el cilindro hasta que el sector en cuestión pasa por debajo de la misma
- **Transfer Time** (transferencia): tiempo de transferencia del sector (bloque) del disco a la memoria

De una definición para estos tres tiempos.



12.

Suponga un disco con las siguientes características:

- **7 platos con**
- **2 caras utilizables cada uno.**
- **1100 cilindros**
- **300 sectores por pista, donde cada sector de es 512 bytes.**
- **Seek Time de 10 ms**
- **9000 RPM .**
- **Velocidad de Transferencia de 10 MiB/s (Mebibytes por segundos).**

(a) Calcule la capacidad total del disco.

Tamaño_disco = 7 * 2 * 1100 * 300 * 512 bytes

tamaño_disco= 2365440000 bytes = 2310000 KiB = 2255.859375 MiB= 2.2029876709 GiB

(b) ¿Cuantos sectores ocuparía un archivo de tamaño de 3 MiB(Mebibytes)?

- Capacidad de 1 sector:
  512 bytes
- Dividimos el tamaño del archivo por la capacidad de una cara:
  3 MiB = 3145728 bytes
  3145728 / 512 = 6144 sectores

(c) Calcule el tiempo de transferencia real de un archivo de 15 MiB(Mebibytes). grabado en el disco de manera secuencial (todos sus bloques almacenados de manera consecutiva)

Latencia:

9000 vueltas → 60000 ms

0,5 → 3,333 ms

Transferencia:

10 MiB → 1000 ms

512 bytes → x

Unificar unidades

10485760 bytes → 1000 ms

512 bytes → 0,048828125 ms

Bloques:

15728640 bytes / 512 bytes = 30720

Datos obtenidos:

Seek time: 10 ms

Latency time: 3,333 ms

Tiempo transferencia bloque: 0,048828125 ms

#bloques: 30720

Almacenamiento secuencial:

10 ms + 3,333 ms + (0,048828125 ms * 30720) = 1513,33 ms

(d) Calcule el tiempo de transferencia real de un archivo de 16 MiB(Mebibytes). grabado en el disco de manera aleatoria.

Latencia:

9000 vueltas → 60000 ms

0,5 → 3,333 ms

Transferencia:

10 MiB → 1000 ms

512 bytes → x

Unificar unidades

10485760 bytes → 1000 ms

512 bytes → 0,048828125 ms

Bloques:

16777216 bytes / 512 bytes = 32768

Datos obtenidos:

Seek time: 10 ms

Latency time: 3,333 ms

Tiempo transferencia bloque: 0,048828125 ms

#bloques: 32768

Almacenamiento aleatorio:

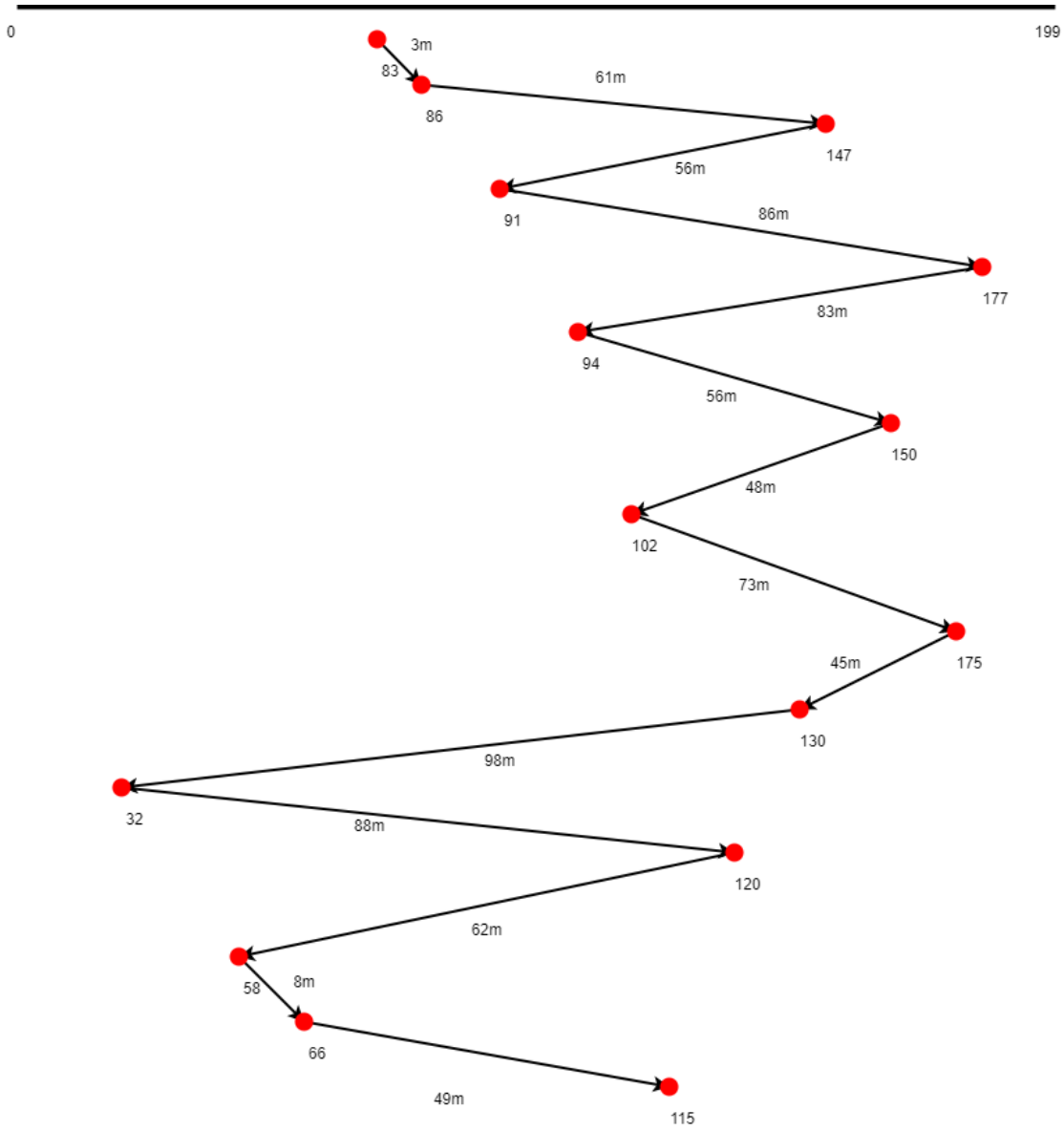
(10 ms + 3,333 ms + 0,048828125 ms) * 32768 = 438495,744 ms

13. El Seek Time es el parámetro que posee mayor influencia en el tiempo real necesario para transferir datos desde o hacia un disco. Es importante que el SO planifique los diferentes requerimientos que al disco para minimizar el movimiento de la cabeza lecto-grabadora.

Analicemos las diferentes políticas de planificación de requerimientos a disco con un ejemplo: Supongamos un Head con movimiento en 200 tracks (numerados de 0 a 199), que está en el track 83 atendiendo un requerimiento y anteriormente atendió un requerimiento en el track 75.

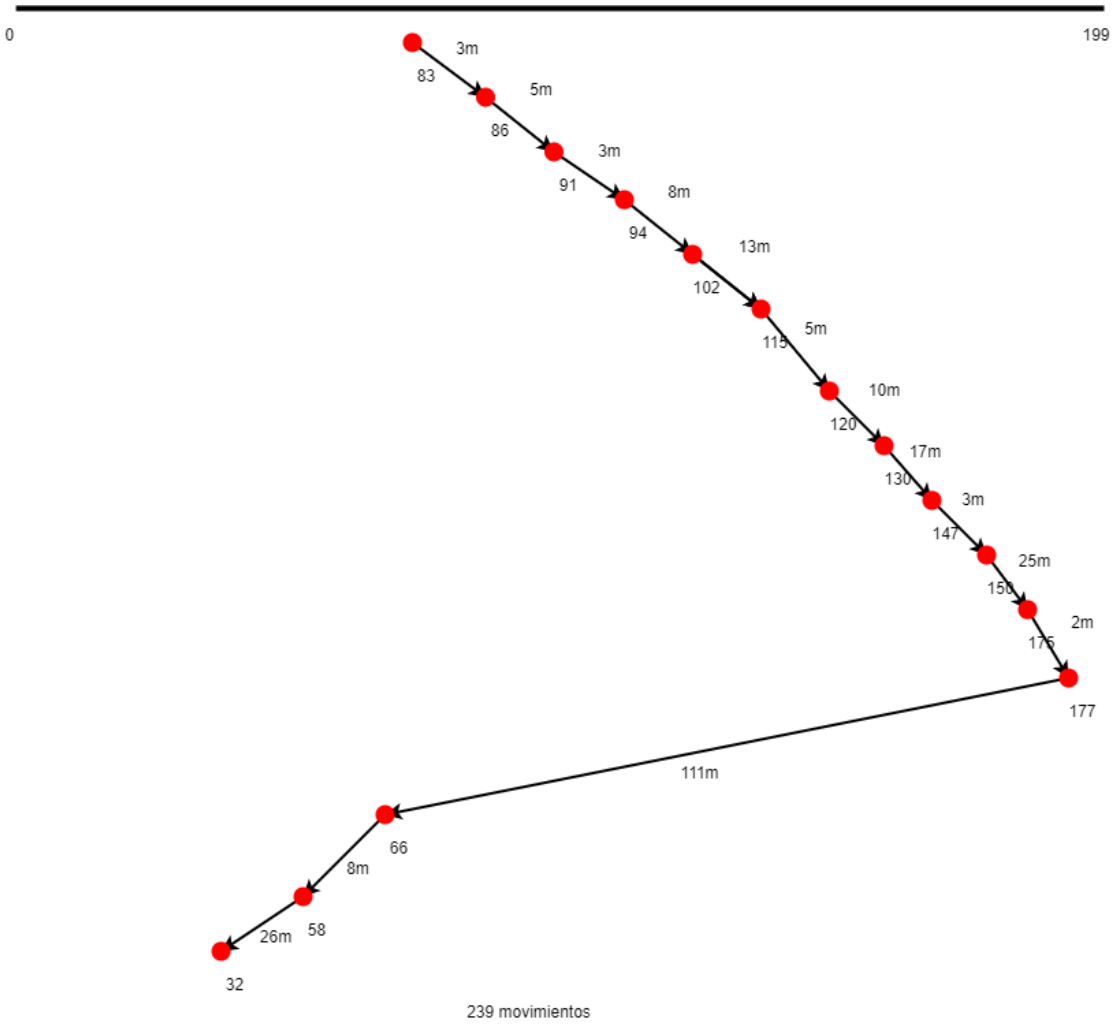
Si la cola de requerimientos es: 86, 147, 91, 177, 94, 150, 102, 175, 130, 32, 120, 58, 66, 115. Realice los diagramas para calcular el total de movimientos de head para satisfacer estos requerimientos de acuerdo a los siguientes algoritmos de scheduling de discos:

(a) FCFS (First Come, First Served)

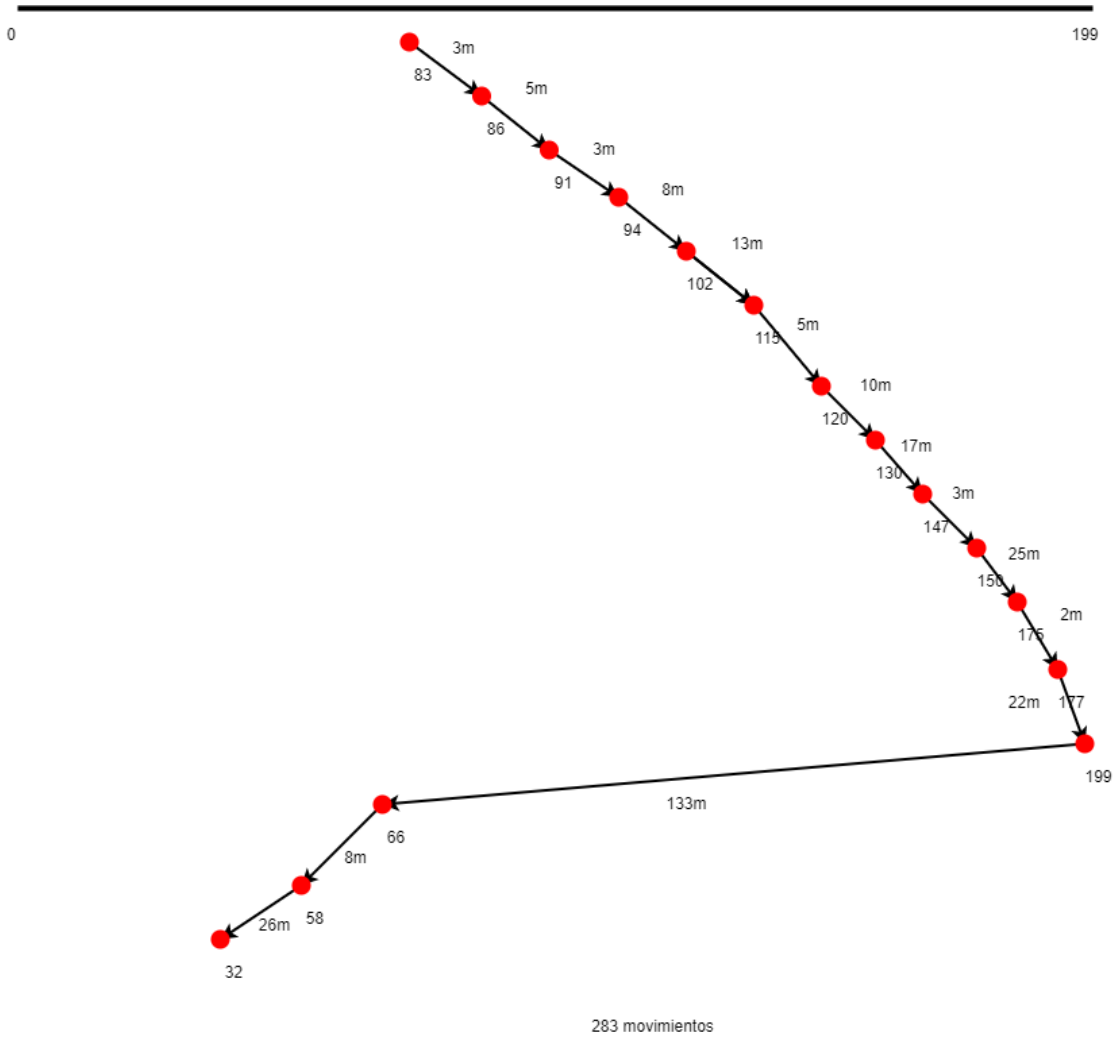


816 movimientos

(b) SSTF (Shortest Seek Time First)

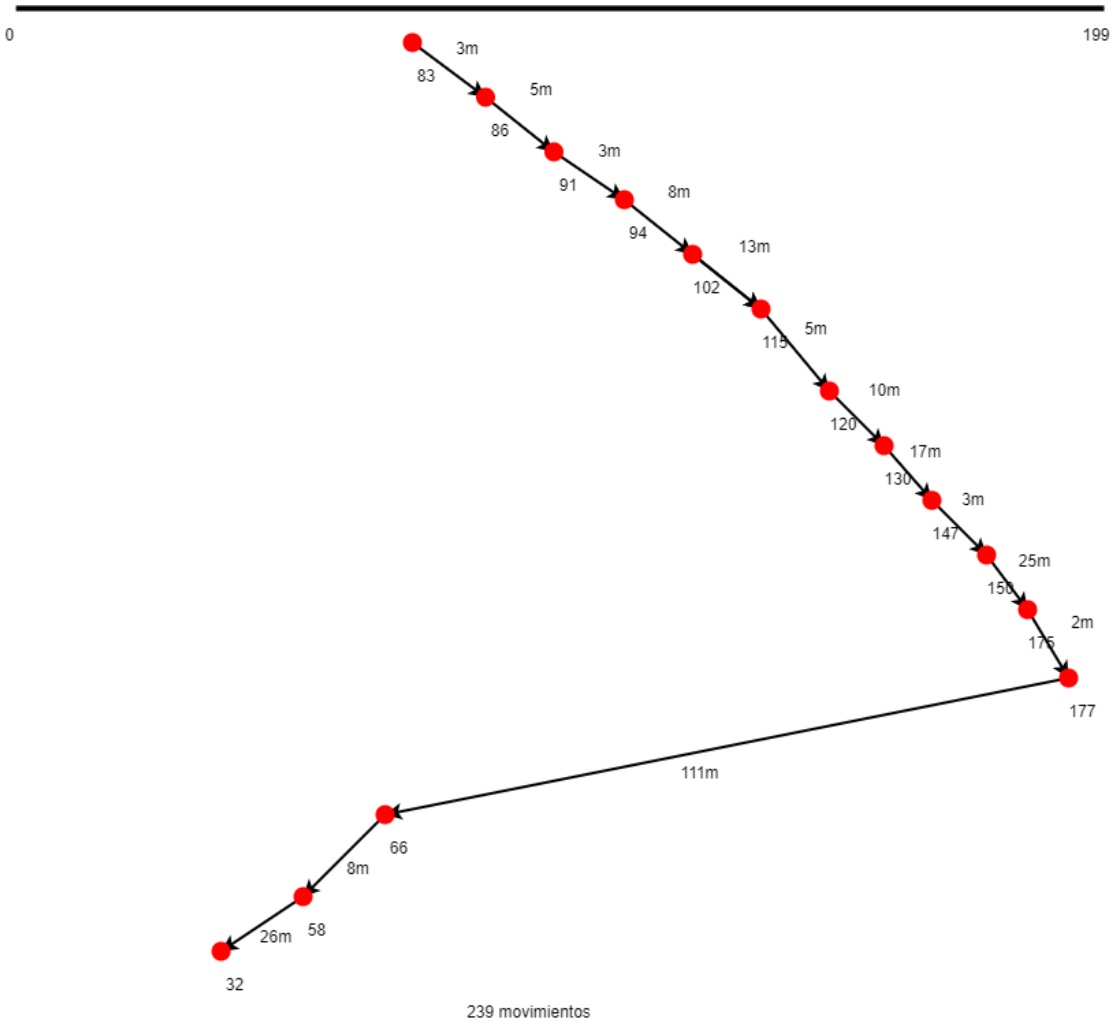


(c) Scan

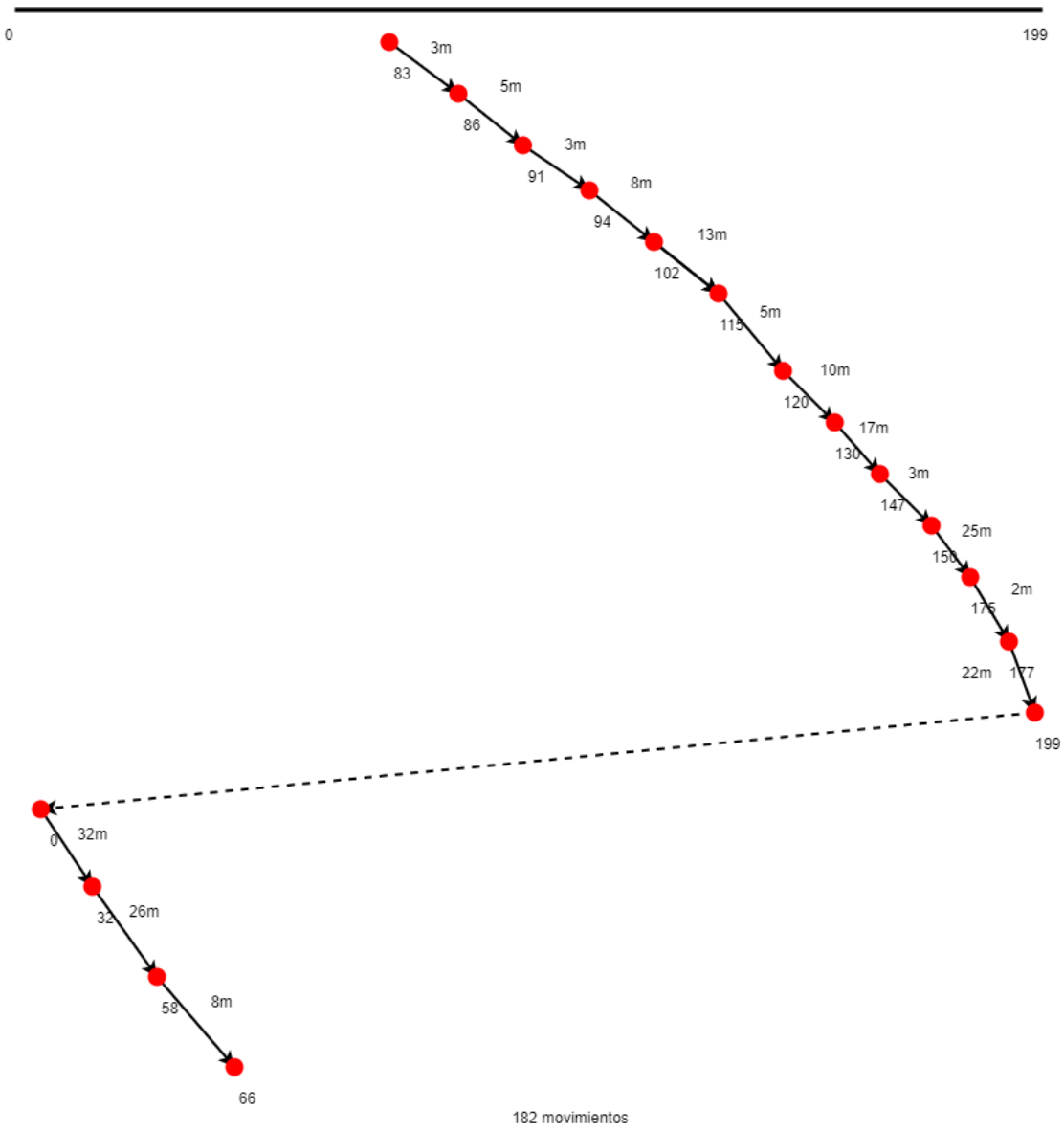


283 movimientos

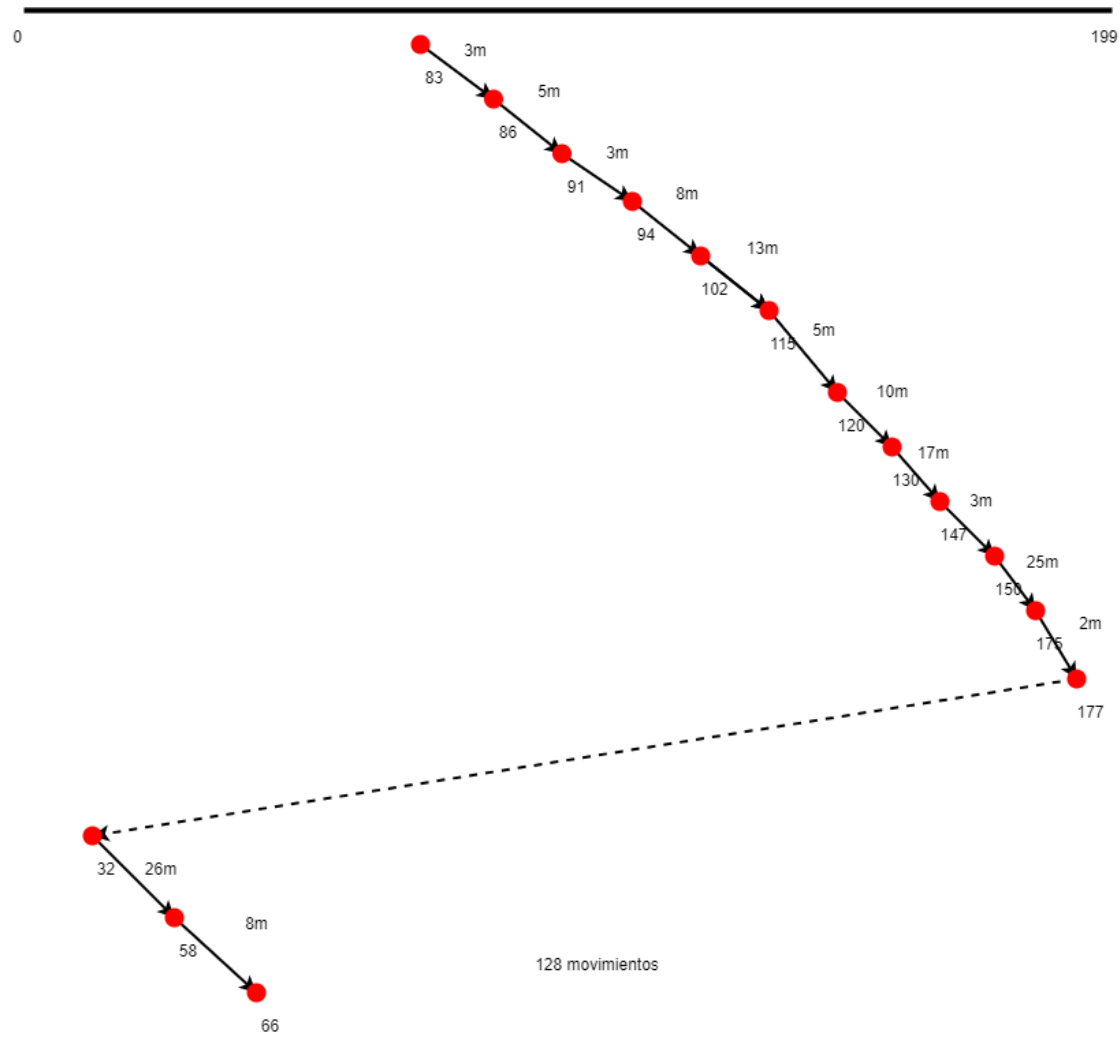
(d) Look


0
199
83
3m
86
5m
91
3m
94
8m
102
13m
115
5m
120
10m
130
17m
147
3m
150
25m
175
2m
177
111m
66
8m
58
26m
32
239 movimientos

(e) C-Scan (Circular Scan)



(f) C-Look (Circular Look)



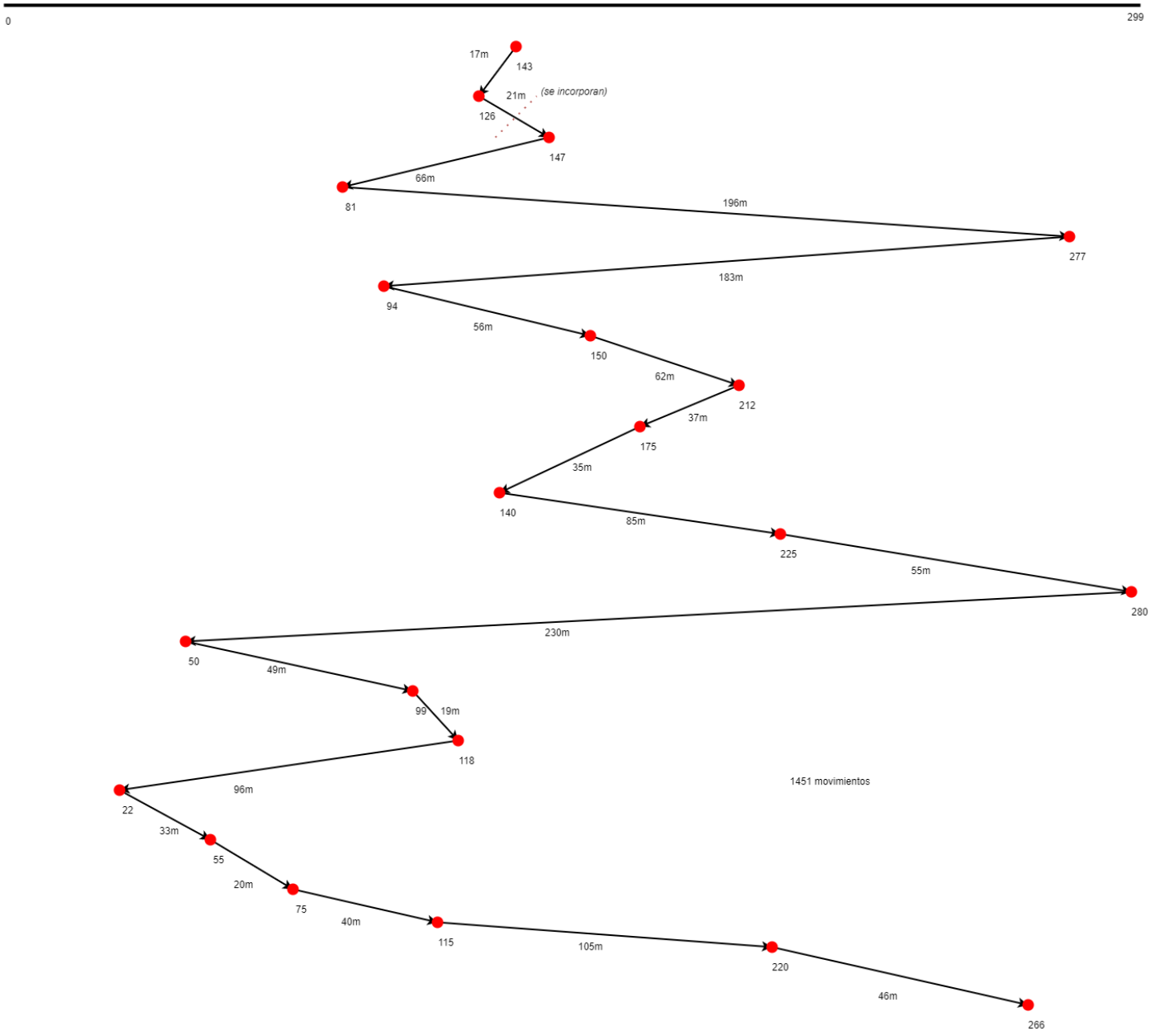
14. ¿Alguno de los algoritmos analizados en el ejercicio anterior pueden causar inanición de requerimientos?

Si, el SSTF

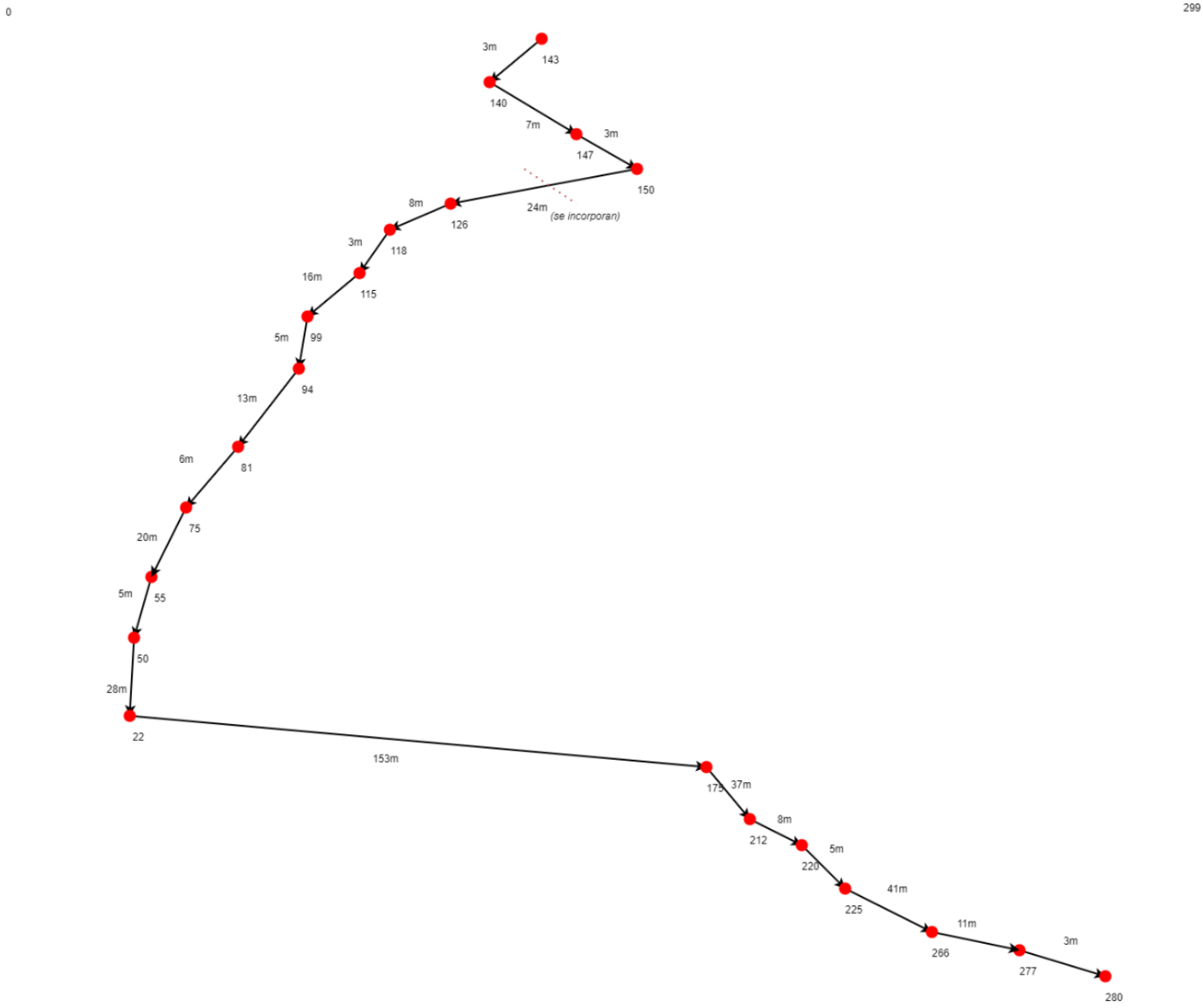
15. Supongamos un Head con movimiento en 300 pistas (numerados de 0 a 299), que esta en la pista 143 atendiendo un requerimiento y anteriormente atendió un requerimiento en la pista 125.

Si la cola de requerimientos es: 126, 147, 81, 277, 94, 150, 212, 175, 140, 225, 280, 50, 99, 118, 22, 55; y después de 30 movimientos se incorporan los requerimientos de las pistas 75, 115, 220 y 266. Realice los diagramas para calcular el total de movimientos de head para satisfacer estos requerimientos de acuerdo a los siguientes algoritmos de scheduling de discos:

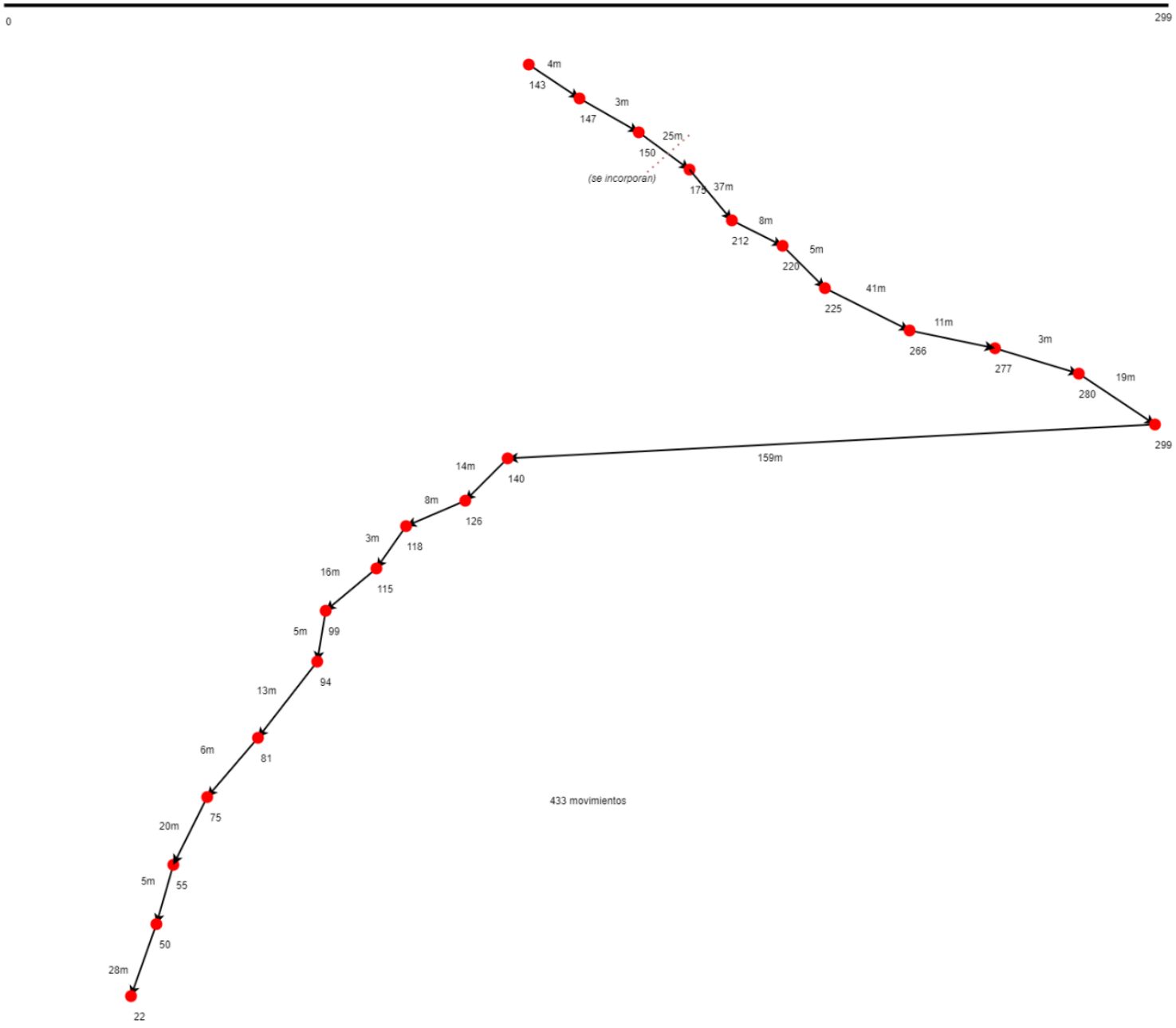
(a) FCFS


0
299
17m
143
21m
(se incorporan)
126
147
66m
81
196m
277
183m
94
56m
150
62m
212
37m
175
35m
140
85m
225
55m
280
230m
50
49m
99
19m
118
96m
22
33m
55
20m
75
40m
115
105m
220
46m
266
1451 movimientos

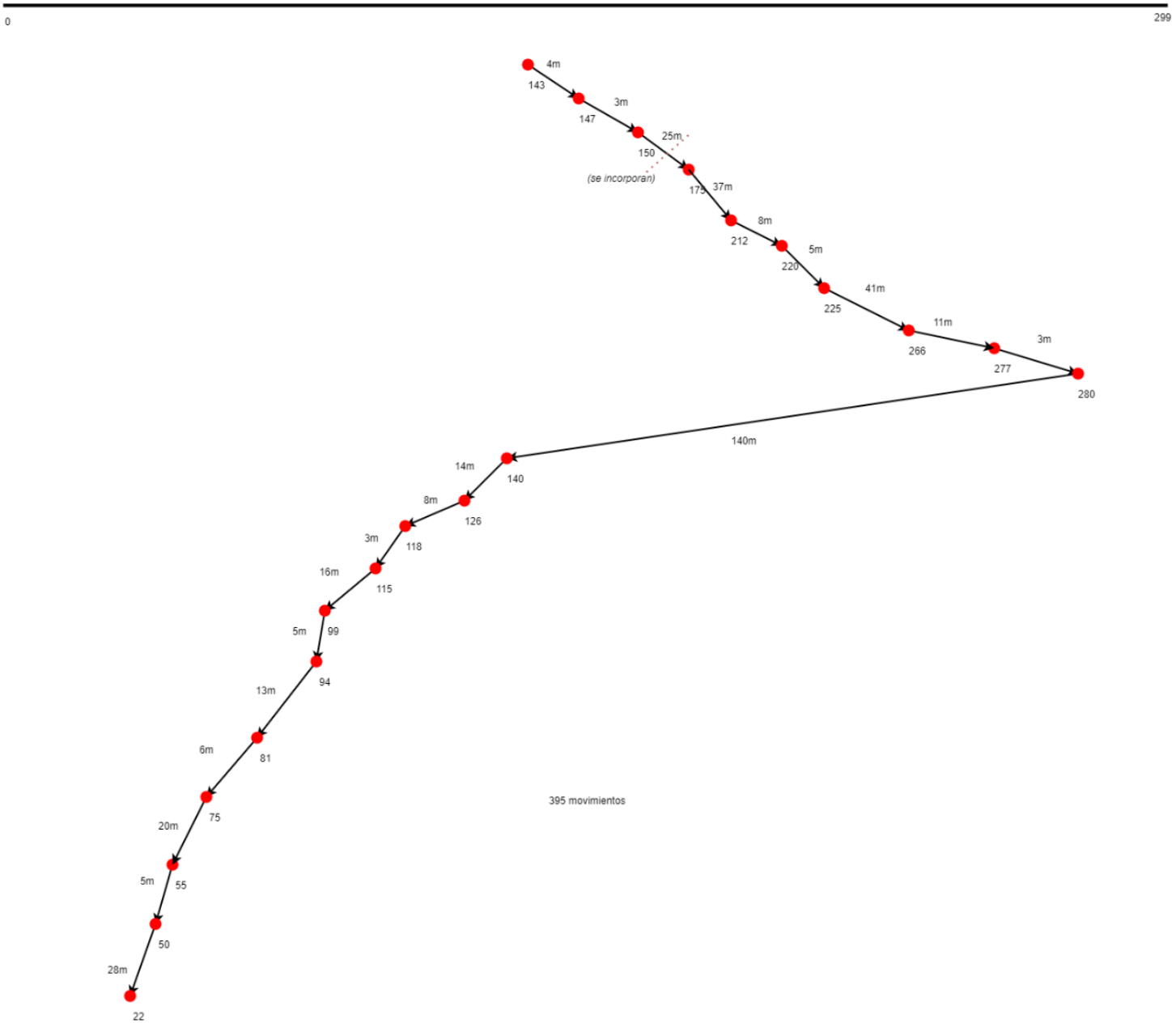
(b) SSTF


0
299
3m
143
140
7m
3m
147
150
24m
(se incorporan)
8m
126
3m
118
16m
115
5m
99
94
13m
81
6m
75
20m
55
5m
50
28m
22
153m
175
37m
8m
212
5m
220
41m
225
11m
266
3m
277
280

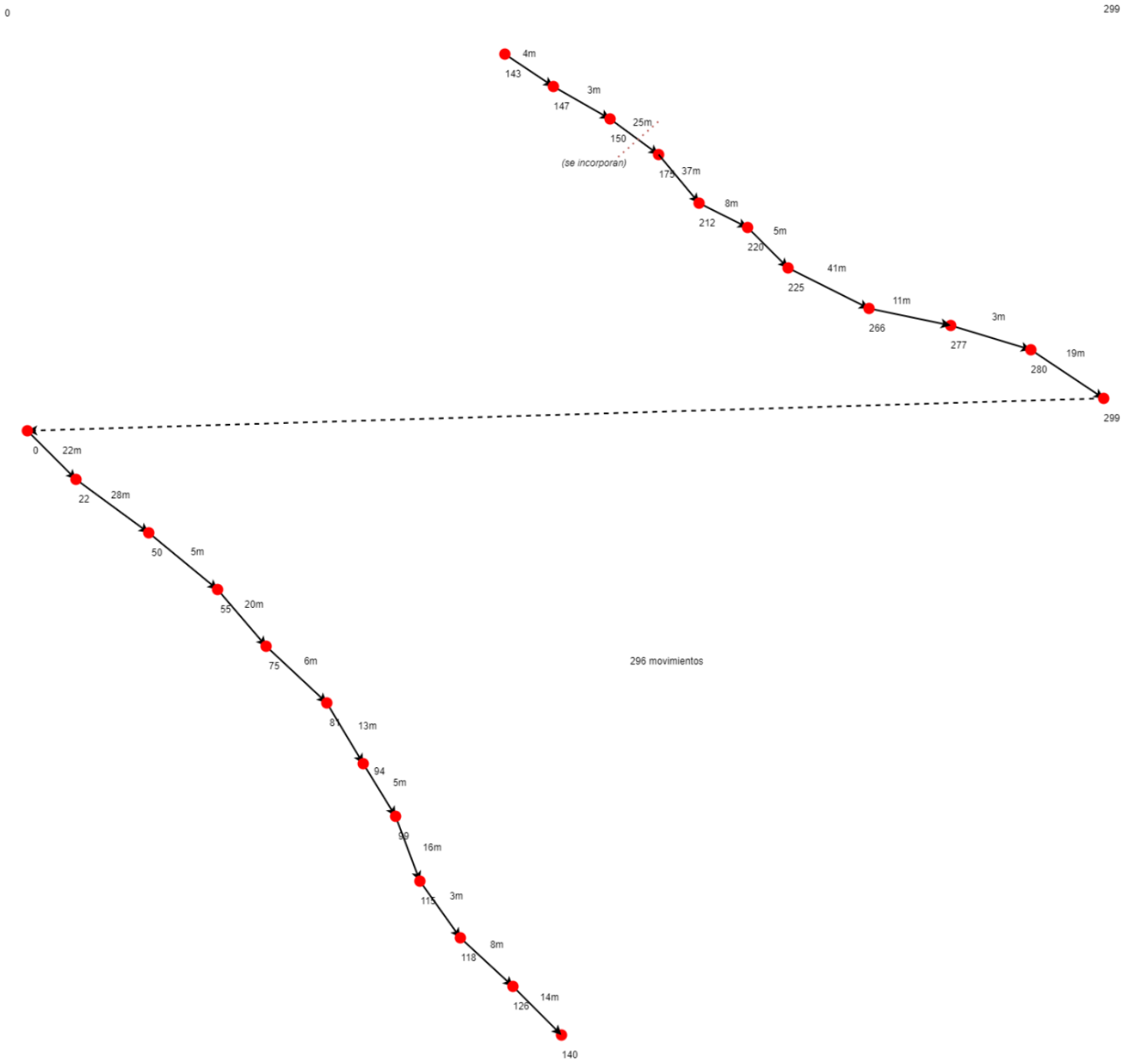
(c) Scan


0
299
4m
143
3m
147
25m
150
(se incorporan)
175
37m
8m
212
5m
220
41m
225
11m
266
3m
277
19m
280
299
159m
14m
140
8m
126
3m
118
16m
115
5m
99
94
13m
81
6m
75
20m
55
5m
50
28m
22
433 movimientos

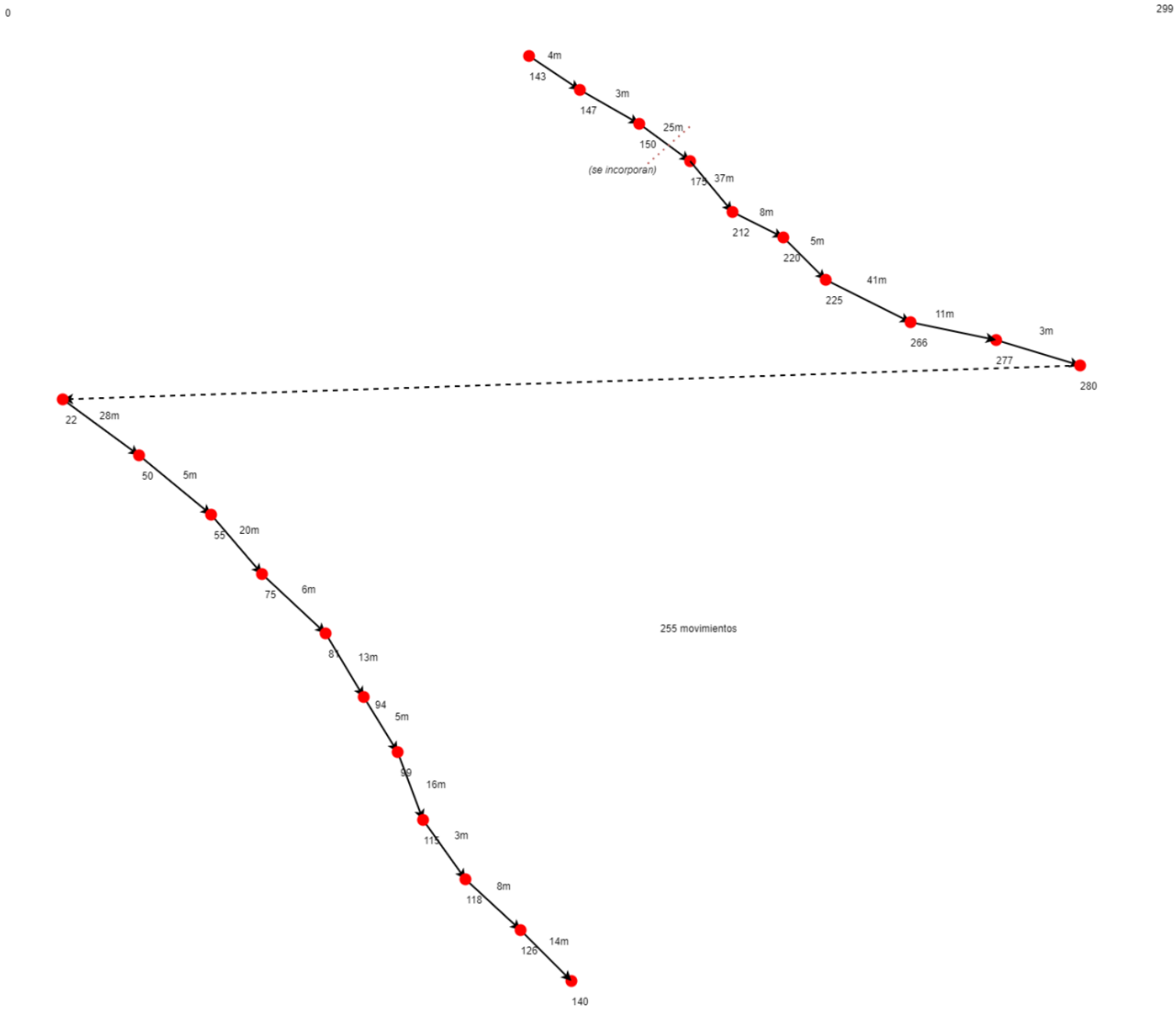
(d) Look


0
299
4m
143
3m
147
25m
150
(se incorporan)
175
37m
8m
212
5m
220
41m
225
11m
266
3m
277
280
140m
14m
140
8m
126
3m
118
16m
115
5m
99
94
13m
81
6m
75
20m
55
5m
50
28m
22
395 movimientos

(e) C-Scan


0
299
4m
143
3m
147
25m
150
(se incorporan)
175
37m
8m
212
5m
220
41m
225
11m
266
3m
277
19m
280
299
0
22m
22
28m
50
5m
55
20m
75
6m
8
13m
94
5m
99
16m
115
3m
118
8m
126
14m
140
296 movimientos

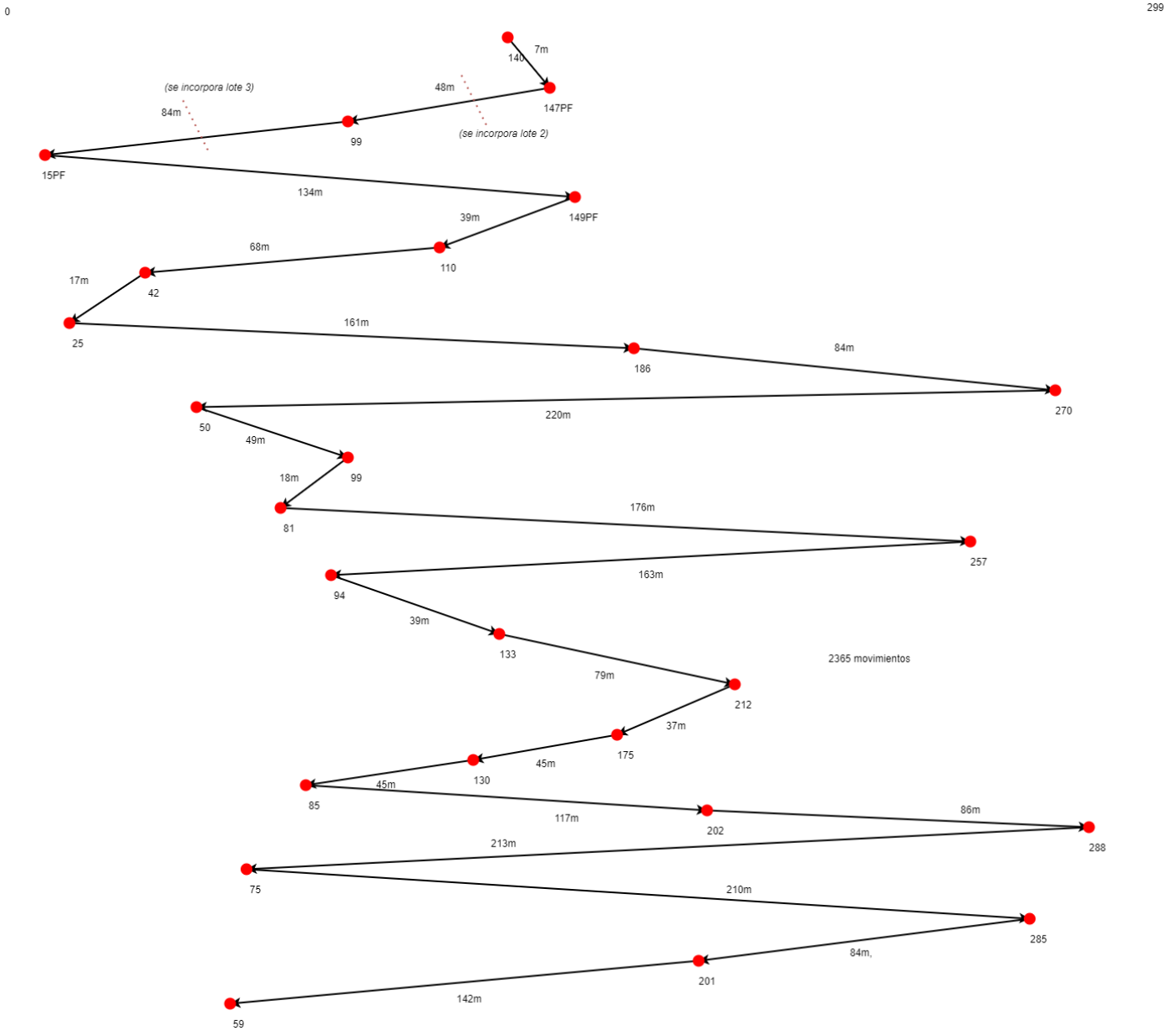
(f) C-Look



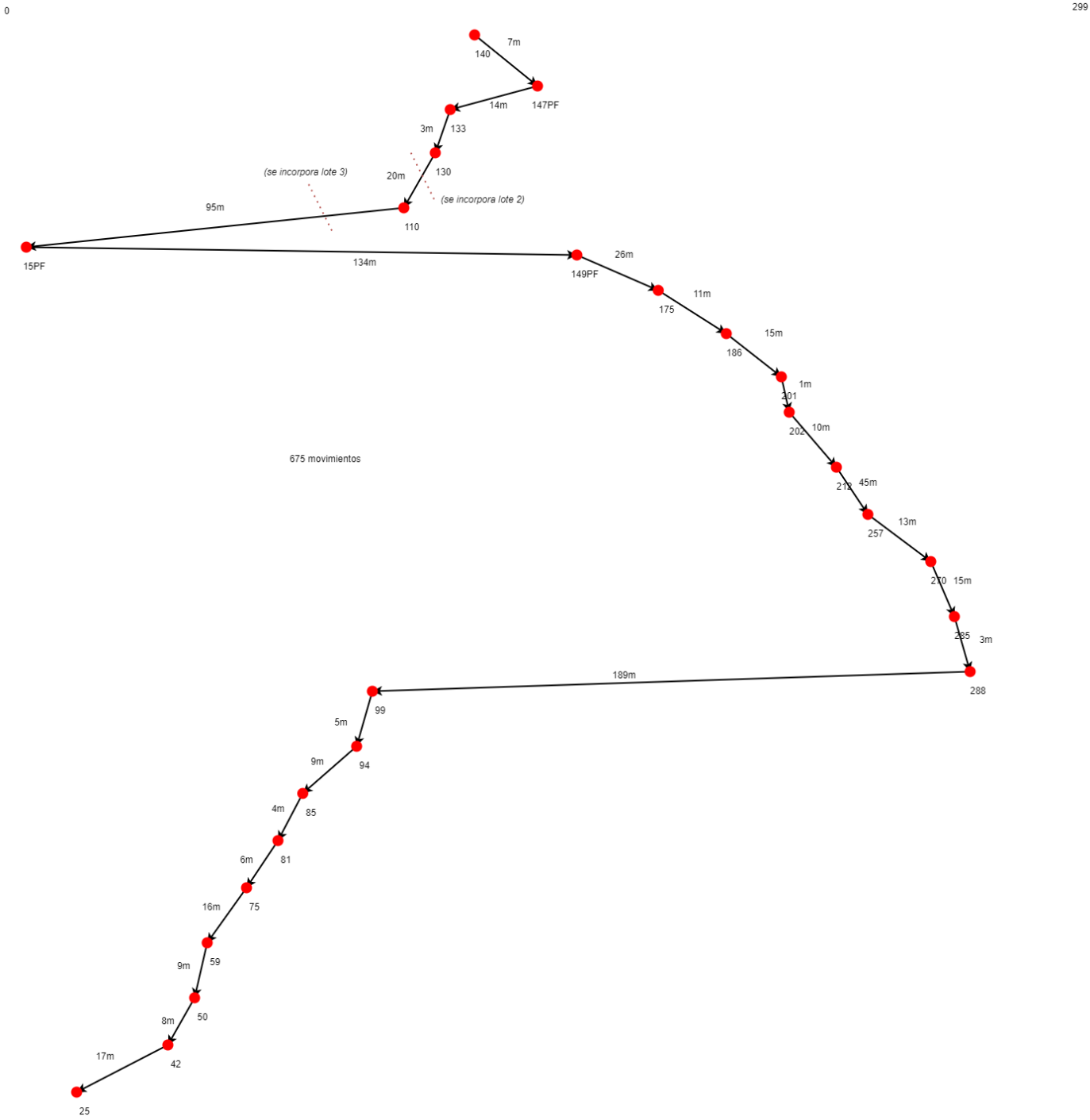
16. Supongamos un Head con movimiento en 300 pistas (numerados de 0 a 299), que esta en la pista 140 atendiendo un requerimiento y anteriormente atendió un requerimiento en la pista 135.

Si la cola de requerimientos es: 99, 110, 42, 25, 186, 270, 50, 99, 147PF , 81, 257, 94, 133, 212, 175, 130; y después de 30 movimientos se incorporan los requerimientos de las pistas 85, 15PF , 202 y 288; y después de otros 40 movimientos más se incorporan los requerimientos de las pistas 75, 149PF , 285, 201 y 59. Realice los diagramas para calcular el total de movimientos de head para satisfacer estos requerimientos de acuerdo a los siguientes algoritmos de scheduling de discos:

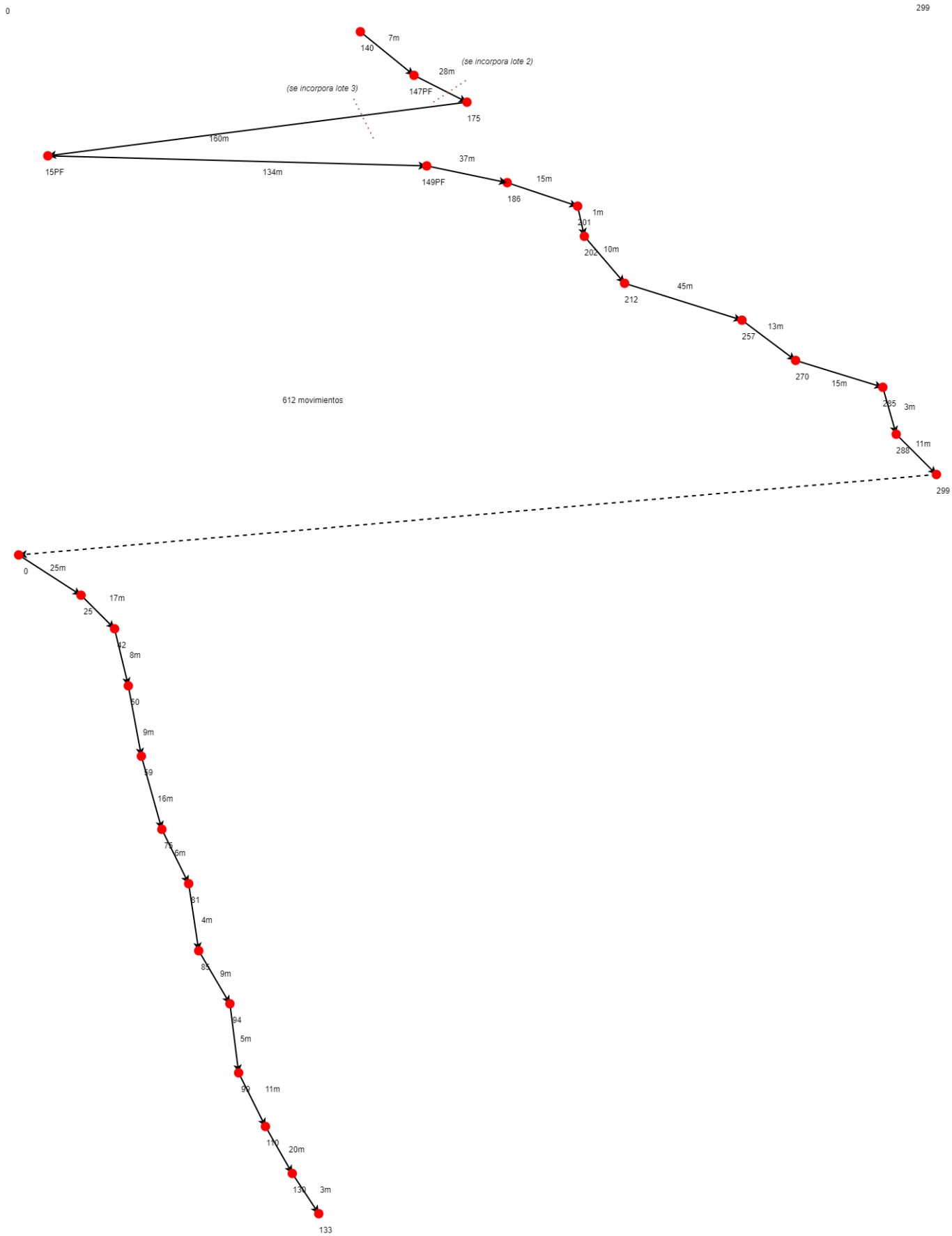
(a) FCFS


0
299
140
7m
147PF
48m
(se incorpora lote 2)
(se incorpora lote 3)
84m
99
15PF
134m
149PF
39m
68m
110
17m
42
25
161m
186
84m
270
220m
50
49m
99
18m
81
176m
257
163m
94
39m
133
2365 movimientos
79m
212
37m
175
45m
130
45m
85
117m
202
86m
288
213m
75
210m
285
84m,
201
142m
59

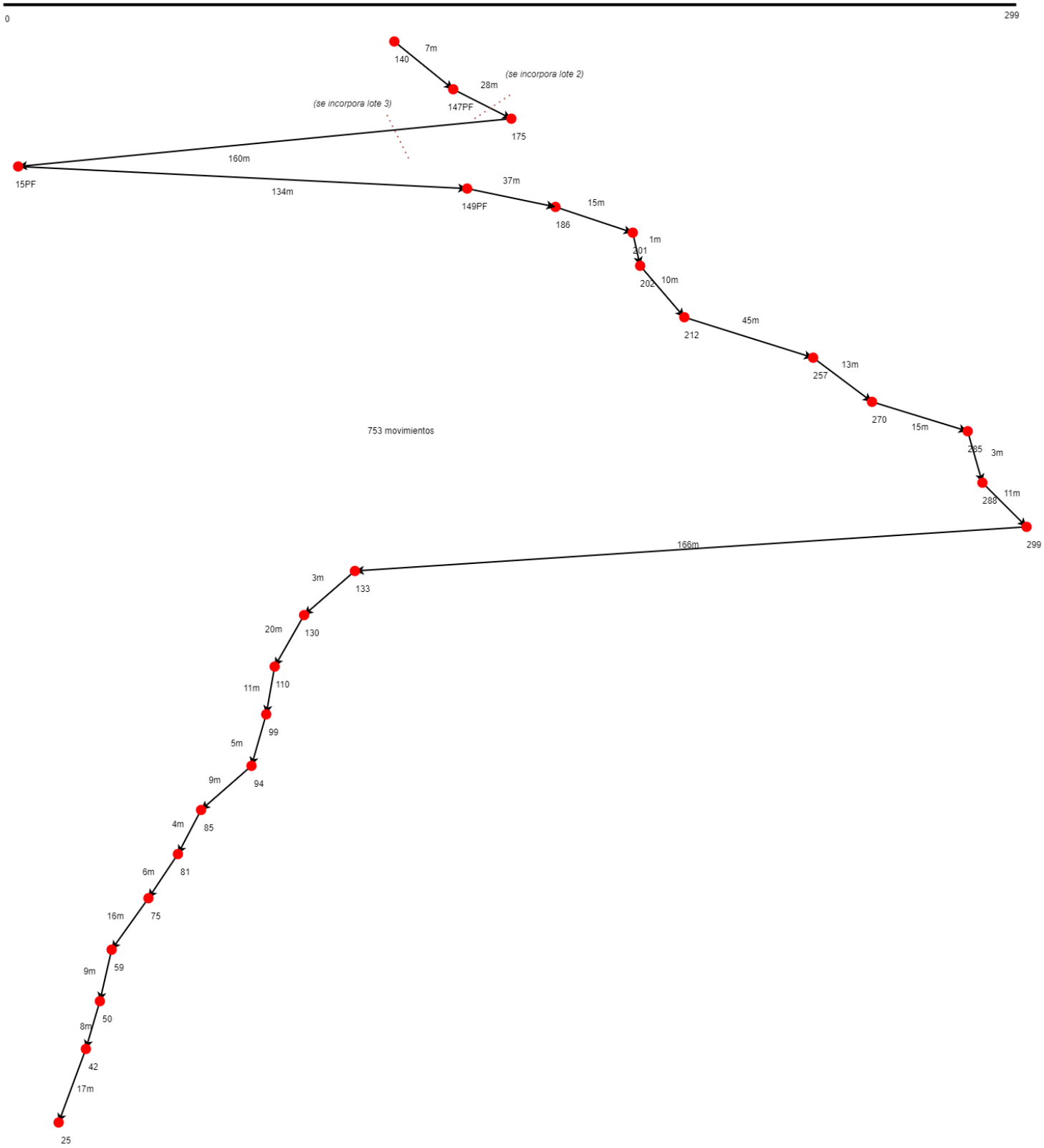
(b) SSTF


0
299
140
7m
147PF
14m
133
3m
130
20m
(se incorpora lote 3)
(se incorpora lote 2)
95m
110
15PF
134m
149PF
26m
175
11m
186
15m
201
1m
202
10m
212
45m
257
13m
270
15m
285
3m
288
189m
99
5m
94
9m
85
4m
81
6m
75
16m
59
9m
50
8m
42
17m
25
675 movimientos

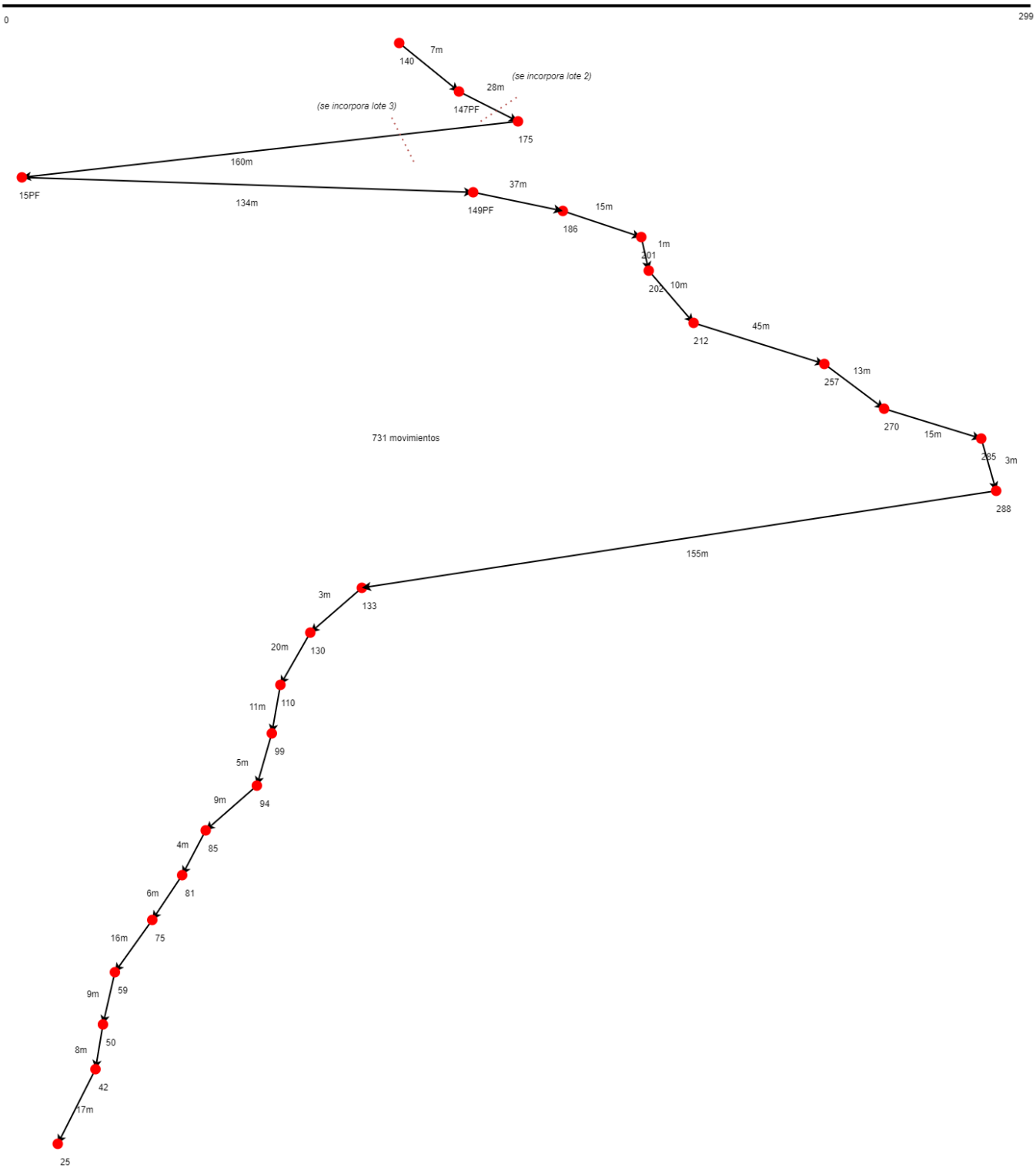
(c) C-Scan


0
299
140
7m
28m
147PF
(se incorpora lote 2)
(se incorpora lote 3)
175
160m
15PF
134m
149PF
37m
15m
186
1m
201
202
10m
212
45m
257
13m
270
15m
285
3m
288
11m
299
612 movimientos
0
25m
25
17m
42
8m
50
9m
59
16m
75
6m
81
4m
85
9m
94
5m
99
11m
110
20m
130
3m
133

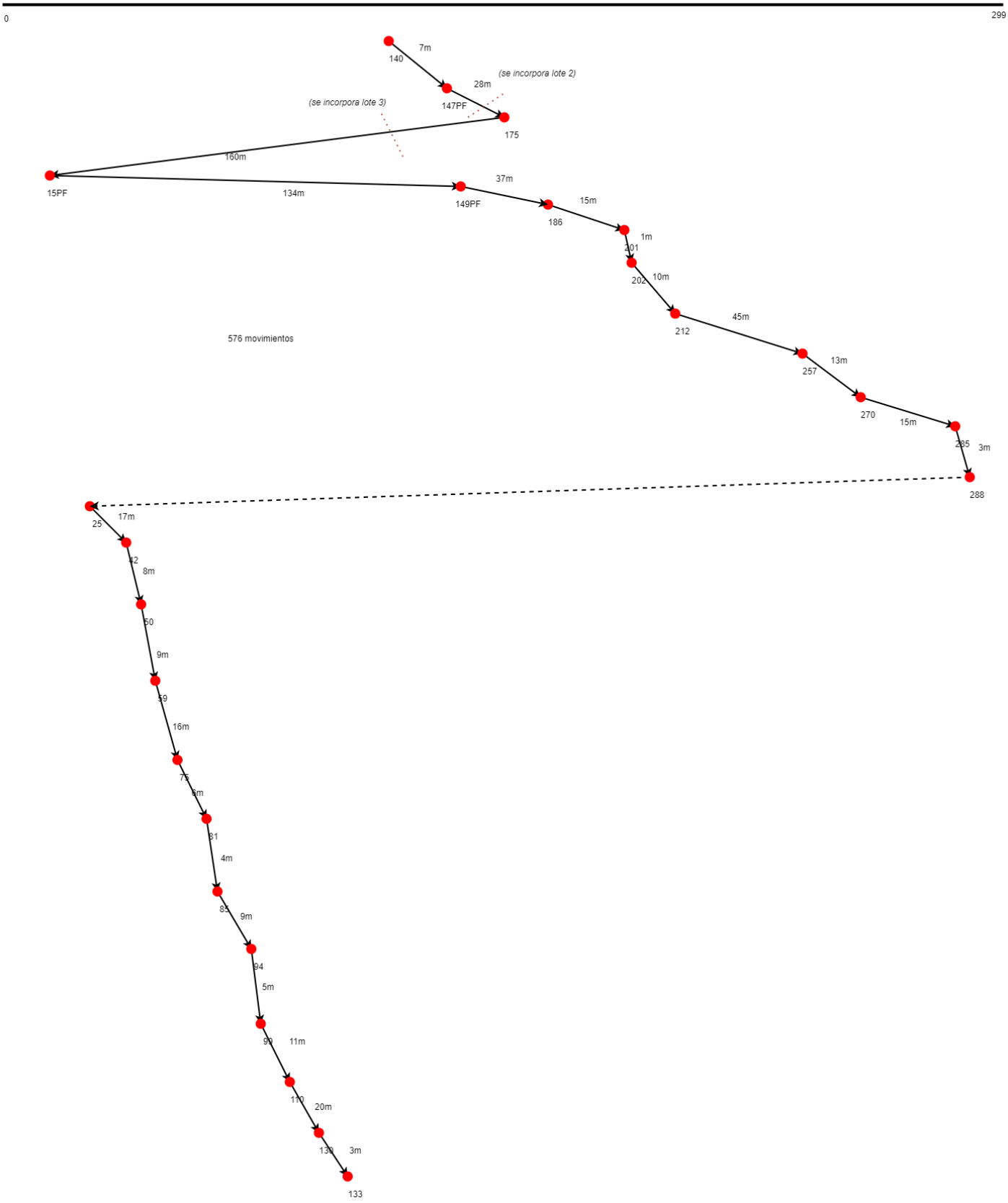
(d) Scan


0
299
140
7m
28m
147PF
175
(se incorpora lote 2)
(se incorpora lote 3)
160m
15PF
134m
37m
149PF
15m
186
1m
201
202
10m
212
45m
257
13m
270
15m
285
3m
288
11m
299
753 movimientos
166m
3m
133
20m
130
11m
110
5m
99
9m
94
4m
85
6m
81
16m
75
9m
59
8m
50
42
17m
25

(e) Look


0
299
140
7m
147PF
28m
(se incorpora lote 2)
(se incorpora lote 3)
175
160m
15PF
134m
149PF
37m
186
15m
201
1m
202
10m
212
45m
257
13m
270
15m
285
3m
288
731 movimientos
155m
133
3m
130
20m
110
11m
99
5m
94
9m
85
4m
81
6m
75
16m
59
9m
50
8m
42
17m
25

(f) C-Look


0
299
140
7m
28m
147PF
(se incorpora lote 2)
(se incorpora lote 3)
175
160m
15PF
134m
37m
149PF
15m
186
1m
201
202
10m
212
45m
576 movimientos
257
13m
270
15m
285
3m
288
25
17m
42
8m
60
9m
69
16m
75
6m
81
4m
85
9m
94
5m
99
11m
110
20m
130
3m
133

Administración de Archivos

17. Dados los siguientes métodos de administración de espació de un archivo:

Asignación contigua: Con asignación contigua, se asigna un único conjunto contiguo de bloques en tiempo de creación de los ficheros.Por tanto, hay una estrategia de preasignación que utiliza porciones de tamaño variable. La tabla de asignación de ficheros necesita sólo una entrada para cada fichero, mostrando el bloque inicial y la longitud del fichero. La asignación contigua es la mejor desde elpunto de vista del fichero secuencial individual. Múltiples bloques se pueden leer de una vez para mejorar el rendimiento de E/S en procesamiento secuencial. Es también fácil obtener un único bloque. Por ejemplo, si un fichero comienza en el bloque b y se quiere acceder al bloque i-ésimo del fichero, su ubicación en almacenamiento secundario es simplemente b + i 1. La asignación contigua presenta algunos problemas. Existirá fragmentación externa, haciendo difícil encontrar bloques contiguos de espacio de suficiente longitud. De vez en cuando, será necesario llevar a cabo un algoritmo de compactación para liberar espacio adicional en el disco. Además, con preasignación, es necesario declarar el tamaño del fichero en el tiempo de creación, con los problemas mencionados anteriormente.

Asignación enlazada: La asignación se realiza a nivel de bloques individuales. Cada bloque contiene un puntero al siguiente bloque en la cadena. De nuevo, la tabla de asignación de ficheros necesita sólo una entrada para cada fichero, mostrando el bloque inicial y la longitud del fichero. Aunque la preasignación es posible, es más común asignar bloques cuando se necesita. La selección de bloques es ahora una cuestión sencilla: cualquier bloque libre se puede añadir a una cadena. No hay fragmentación externa de la que preocuparse porque sólo se necesita un bloque cada vez. Este tipo de organización física se adapta mejor a ficheros secuenciales que se procesan secuencialmente. Seleccionar un bloque individual de un fichero requiere seguir la cadena hasta alcanzar el bloque deseado.

Una consecuencia del encadenamiento, tal como se describe, es que no existe principio de proximidad. Por tanto, si es necesario traer varios bloques de fichero a la vez, como en el procesamiento secuencial, se requiere una serie de accesos a diferentes partes del disco. Esto es tal vez un efecto más significativo en un sistema monousuario, pero también podría ser preocupante en el caso de un sistema compartido. Para resolver este problema, algunos sistemas consolidan ficheros periódicamente.

Asignación indexada: La asignación indexada resuelve muchos de los problemas de la asignación contigua y encadenada. En este caso, la tabla de asignación de ficheros contiene un índice separado de un nivel por cada fichero; el índice tiene una entrada por cada porción asignada al fichero. Típicamente, los índices de fichero no se almacenan físicamente como parte de la tabla de asignación de ficheros. Por el contrario, el índice de ficheros para un fichero se guarda en un bloque separado y la entrada para fichero en la tabla de asignación de ficheros apunta a dicho bloque. La asignación puede realizarse mediante bloques de tamaño fijo (Figura 12.11) o porciones de tamaño variable (Figura 12.12). La asignación por bloques elimina la fragmentación externa, mientras que la asignación por porciones de tamaño variable mejora la proximidad. En cualquier caso, la consolidación de ficheros se puede realizar de vez en cuando. La consolidación de ficheros reduce el tamaño del índice en el caso de porciones de tamaño variable, pero no en el caso de asignación de bloques. La asignación indexada da soporte tanto a acceso secuencial como directo a los ficheros y por tanto es la forma más popular de asignación de ficheros.

(a) Describa como trabaja cada uno.

(b) Cite ventajas y desventajas de cada uno.

18. Gestión de espacio libre. Dados los siguientes métodos de gestión de espacio libre en un disco:

Tabla de bits: Este método utiliza un vector que está formado por un bit por cada bloque en el disco. Cada entrada 0 corresponde a un bloque libre y cada 1 corresponde a un bloque en uso. Una tabla de bits tiene la ventaja de que es relativamente fácil encontrar un bloque libre o un grupo contiguo de bloques libres. Por tanto, una tabla de bits trabaja bien con cualquiera de los métodos de asignación de ficheros descrito. Esta estructura es tan pequeña como sea posible. La cantidad de memoria (en bytes) requerida para un mapa de bits de bloques es:

Tamaño de disco en bytes / 8 x tamaño de bloque del sistema de ficheros

Por tanto, para un disco de 16 Gbytes con bloques de 512 bits, la tabla de bits ocupa 4 Mbytes. Se puede almacenar la tabla de bits de 4 Mbytes en memoria principal? Si es así, entonces a la tabla de bits se puede acceder sin necesidad de acceder a disco. Pero incluso con los tamaños de memoria de hoy, 4 Mbytes es una cantidad considerable de memoria principal para dedicar a una única función (desventaja). La alternativa es poner la tabla de bits en disco. Pero una tabla de bits de 4 Mbytes requeriría alrededor de 8000 bloques de disco. No se puede hacer una búsqueda sobre esta cantidad de espacio de disco cada vez que se necesita un bloque, de tal forma que una tabla de bits residente en memoria es indicada.

Incluso cuando la tabla de bits está en memoria principal, una búsqueda exhaustiva de la tabla puede ralentizar el rendimiento del sistema de ficheros hasta un grado inaceptable. Esto es especialmente cierto cuando el disco está casi lleno y hay pocos bloques libres restantes. De forma análoga, la mayoría de los sistemas de ficheros que utilizan tablas de bits mantienen estructuras de datos auxiliares que resumen los contenidos de subrangos de la tabla de bits. Por ejemplo, la tabla se podía dividir lógicamente en varios subrangos de igual tamaño. Una tabla resumen podría incluir, para cada subrango, el número de bloques libres y el número de bloques libres contiguos de tamaño máximo. Cuando el sistema de ficheros necesita varios bloques contiguos, puede analizar la tabla resumen para encontrar un subrango apropiado y entonces buscar dentro de dicho subrango

Lista Ligada: Las porciones libres se pueden encadenar utilizando un puntero y valor de longitud en cada porción libre. Este método tiene una sobrecarga de espacio insignificante, porque no se necesita una tabla de asignación de disco, sino simplemente un puntero al comienzo de la cadena y la longitud de la primera porción. Este método es apropiado para todos los métodos de asignación de ficheros. Si se asigna un bloque cada vez, simplemente hay que escoger el bloque libre en la cabeza de la cadena y ajustar el primer puntero o el valor de la longitud. Si la asignación se hace de una porción de longitud variable, se debe utilizar un algoritmo de primer ajuste: se cargan las cabeceras de las porciones para determinar la siguiente porción libre apropiada en la cadena. De nuevo, se ajustan el puntero y los valores de longitud.

Este método tiene sus propios problemas. Después de cierto uso, el disco se quedará bastante fragmentado y muchas porciones serán de la longitud de un único bloque. También obsérvese que cada vez que se asigne un bloque, se necesita leer el bloque primero para recuperar el

puntero al nuevo primer bloque libre antes de escribir datos en dicho bloque. Si se necesitan asignar muchos bloques individuales de una vez para una operación sobre ficheros, esto ralentiza en gran medida la creación de ficheros. Similarmente, borrar ficheros altamente fragmentados es una tarea que consume mucho tiempo.

Agrupamiento: La técnica de indexación trata el espacio libre como un fichero y utiliza una tabla de índices tal y como se describió en la asignación de ficheros. Por motivos de eficiencia, el índice se debería utilizar en base a porciones de tamaño variable en lugar de bloques. Por tanto, hay una entrada en la tabla por cada porción libre en el disco. Esta técnica proporciona soporte eficiente a todos los métodos de asignación de ficheros.

Recuento: En este método, a cada bloque se le asigna un número secuencialmente y la lista de los números de todos los bloques libres se mantiene en una porción reservada del disco. Dependiendo del tamaño del disco, se necesitarán 24 o 32 bits para almacenar un único número de bloque, de tal forma que el tamaño de la lista de bloques libres es 24 o 32 veces el tamaño de la correspondiente tabla de bits y por tanto debe almacenarse en disco y no en memoria principal. Sin embargo, este es un método satisfactorio. Considérense los siguientes puntos:

1. El espacio en disco dedicado a la lista de bloques libres es menor que el 1% del espacio total de disco. Si se utiliza un número de bloque de 32 bits, entonces la penalización de espacio es de 4 bytes por cada bloque de 512 bytes.
2. Aunque la lista de bloques libres es demasiado grande para almacenarla en memoria principal, hay dos técnicas efectivas para almacenar una pequeña parte de la lista en memoria principal.
    a. La lista se puede tratar como una pila (Apéndice 1B) con los primeros miles de elementos de la pila residentes en memoria principal. Cuando se asigna un nuevo bloque, se saca de la pila, que está en memoria principal. Similarmente, cuando se desasigna un bloque, se coloca en la pila. Cuando la porción de la pila en memoria se llena o se vacía, hay sólo una transferencia entre disco y memoria principal. Por tanto, esta técnica da acceso muy rápido en la mayoría de las ocasiones
    b. La lista se puede tratar como una cola FIFO, con unos pocos miles de entradas desde la cabeza al final de la cola en memoria principal. Se asigna un bloque tomando la primera entrada de la cabeza de la cola y se desasigna añadiéndolo al final de la cola. Sólo hay una transferencia entre disco y memoria principal cuando la porción en memoria de la cabeza de la cola se vacía o la porción en memoria del final de la cola se llena.

En cualquiera de las estrategias listadas en el punto precedente (pila o cola FIFO), un hilo en segundo plano puede ordenar lentamente la lista en memoria o listas para facilitar la asignación contigua

(a) Describa como trabajan estos métodos.

(b) Cite ventajas y desventajas de cada uno

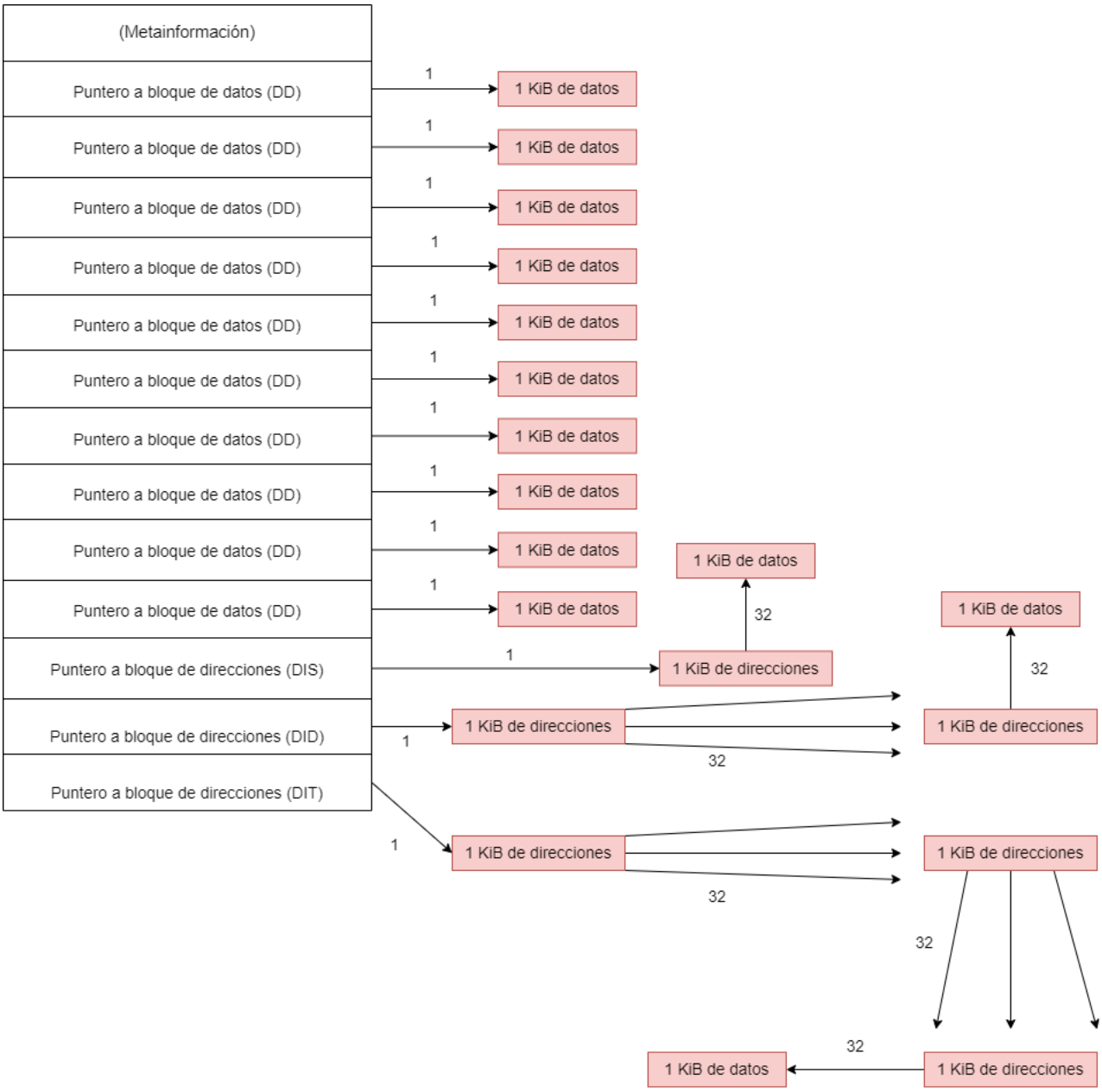
19. Gestión de archivos en UNIX. El sistema de archivos de UNIX utiliza una versión modificada del esquema de Asignación Indexada para la administración de espacio de los archivos.

Cada archivo o directorio esta representado por una estructura que mantiene, entre otra información, las direcciones de lo bloques que contienen los datos del archivo: el I-NODO.

Cada I-NODO contiene 13 direcciones a los bloques de datos, organizadas de la siguiente manera:

- 10 de direccionamiento directo.
- 1 de direccionamiento indirecto simple.
- 1 de direccionamiento indirecto doble.
- 1 de direccionamiento indirecto triple.

(a) Realice un gráfico que describa la estructura del I-NODO y de los bloques de datos. Cada bloque es de 1 Kib(Kibibits). Si cada dirección para referenciar un bloque es de 32 bits:



i. ¿Cuántas referencias (direcciones) a bloque pueden contener un bloque de disco?

1Kib (1024 bits) / 32 bits = 32 direcciones por bloque

ii. ¿Cuál sería el tamaño máximo de un archivo?

1 Kib * 10 + 32 * 1 KiB + 32^2 * 1 KiB + 32^3 * 1 KiB = 33834 Kib = 4229.25 KiB

20. Analice las siguientes fórmulas necesarias para localizar un I-NODO en la lista de inodos:

nro bloque = ((nro de inodo -1)/nro. de inodos por bloque) + bloque de comienzo de la lista de inodos.

Desplazamiento del inodo en el bloque = ((nro de inodo - 1) módulo (número de inodos por bloque)) * medida de inodo del disco.

(a) Según la primer fórmula, asumiendo que en el bloque 2 está en el comienzo de la lista de inodos y que hay 8 inodos por bloque: calcule donde se encuentra el inodo 8 y el 9. ¿Dónde estarían para bloque de disco de 16 inodos?

I-NODO 8:

8 inodos por bloque → **nro bloque** = (7/8) + 2 = 2,87

16 inodos por bloque → **nro bloque** = (7/16) + 2 = 2,4375

I-NODO 9:

8 inodos por bloque → **nro bloque** = (8/8) + 2 = 3

16 inodos por bloque → **nro bloque** = (8/16) + 2 = 2,5

(b) De acuerdo a la segunda fórmula, si cada inodo del disco ocupa 64 bytes y hay 8 inodos por bloque de disco, el inodo 8 comienza en el desplazamiento 448 del bloque de disco. ¿Dónde empieza el 6? Si fueran inodos de 128 bytes y 24 inodos por bloque: ¿dónde empezaría el inodo 8?

I-NODO 6 → **Desplazamiento del inodo en el bloque** = 6-1 % 8* 64 bytes = 320

I-NODO 8 → **Desplazamiento del inodo en el bloque** = 8-1 % 24 * 128 bytes = 896

Laboratorio de Entrada-Salida

Se recomienda resolver este laboratorio con el Sistema en modo consola y con la menor cantidad de programas en ejecución posible.

21. Instale e investigue para que sirve el siguiente programa hdparm

```
sudo apt-get install hdparm
```

hdparm (es decir, parámetro del disco duro) es uno de los programas de línea de comandos para Linux que se usa para manejar dispositivos de disco y discos duros. Con la ayuda de este comando, puede obtener estadísticas sobre el disco duro, modificar los intervalos de escritura, la gestión acústica y la configuración de DMA. También puede establecer parámetros relacionados con las memorias caché de la unidad, el modo de suspensión, la administración de energía, la administración acústica y la configuración de DMA.

https://www.geeksforgeeks.org/hdparm-command-in-linux-with-examples/

22. Ahora ejecute el siguiente comando:

```
sudo hdparm -I /dev/sda
```

¿Cuándo cilindros, cabezas y sectores tiene su disco?

Cilindros 16383

Cabezas 16

Sectores 63

¿Qué pasa si ejecuta esto en un disco de estado sólido?

Ni idea

23. Ahora ejecute el siguiente comando varias veces(al menos 5), de manera tal de poder calcular el promedio de resultados obtenidos

```
sudo hdparm -t /dev/sda
```

```
agusnfr@agusnfr-VirtualBox:~$ sudo hdparm -t /dev/sda

/dev/sda:
 Timing buffered disk reads: 434 MB in  3.01 seconds = 144.23 MB/sec
agusnfr@agusnfr-VirtualBox:~$ sudo hdparm -t /dev/sda

/dev/sda:
 Timing buffered disk reads: 690 MB in  3.01 seconds = 229.25 MB/sec
agusnfr@agusnfr-VirtualBox:~$ sudo hdparm -t /dev/sda

/dev/sda:
 Timing buffered disk reads: 860 MB in  3.01 seconds = 285.39 MB/sec
agusnfr@agusnfr-VirtualBox:~$ sudo hdparm -t /dev/sda

/dev/sda:
 Timing buffered disk reads: 942 MB in  3.01 seconds = 313.34 MB/sec
agusnfr@agusnfr-VirtualBox:~$ sudo hdparm -t /dev/sda

/dev/sda:
 Timing buffered disk reads: 986 MB in  3.01 seconds = 327.98 MB/sec
```

24. Ahora ejecutelo de la siguiente manera:

```
hdparm -t --direct --offset 500 /dev/sda
```

```
agusnfr@agusnfr-VirtualBox:~$ sudo hdparm -t --direct --offset 500 /dev/sda

/dev/sda:
 Timing O_DIRECT disk reads (offset 500 GB): lseek() failed: Invalid argument
agusnfr@agusnfr-VirtualBox:~$ sudo hdparm -t --direct --offset 40 /dev/sda

/dev/sda:
 Timing O_DIRECT disk reads (offset 40 GB): 818 MB in  3.02 seconds = 270.95 MB/sec
```

```
dev/sda:
Timing O_DIRECT disk reads (offset 40 GB): 854 MB in  3.01 seconds = 284.03 MB/sec
gusnfr@agusnfr-VirtualBox:~/Escritorio$ sudo hdparm -t --direct --offset 40 /dev/sda

dev/sda:
Timing O_DIRECT disk reads (offset 40 GB): 810 MB in  3.02 seconds = 267.84 MB/sec
gusnfr@agusnfr-VirtualBox:~/Escritorio$ sudo hdparm -t --direct --offset 40 /dev/sda

dev/sda:
Timing O_DIRECT disk reads (offset 40 GB): 818 MB in  3.00 seconds = 272.35 MB/sec
gusnfr@agusnfr-VirtualBox:~/Escritorio$ sudo hdparm -t --direct --offset 40 /dev/sda

dev/sda:
Timing O_DIRECT disk reads (offset 40 GB): 842 MB in  3.02 seconds = 279.21 MB/sec
gusnfr@agusnfr-VirtualBox:~/Escritorio$ sudo hdparm -t --direct --offset 40 /dev/sda

dev/sda:
Timing O_DIRECT disk reads (offset 40 GB): 834 MB in  3.00 seconds = 277.65 MB/sec
```

25. ¿Para que sirven los parámetros?:

- **direct**

```
--direct
       Use the kernel's "O_DIRECT" flag when performing a -t timing test.  This  bypasses  the
       page  cache, causing the reads to go directly from the drive into hdparm's buffers, us-
       ing so-called "raw" I/O.  In many cases, this can  produce  results  that  appear  much
       faster  than  the usual page cache method, giving a better indication of raw device and
       driver performance.
```

- **offset**

```
--offset
       Offsets  to  given  number of GiB (1024*1024*1024) when performing -t timings of device
       reads.  Speed changes (about twice) along many mechanical drives.  Usually the  maximum
       is  at  the  beginning,  but not always.  Solid-state drives (SSDs) should show similar
       timings regardless of offset.
```

26. Compare los tiempos promedios obtenidos con los parámetros direct y offset y sin ellos(Recuerde comparar tiempos promedio y no ejecuciones aisladas).

260,038 sin parámetros ante 276,27 con parámetros. El segundo es más lento ya que lee los datos directamente desde el disco pero al menos se puede ver la velocidad de transmisión pura del disco

27. ¿Con que concepto de la teoría asocia el parámetro direct?

Agregar la opción –direct a nuestra sintaxis de comando realizará otra velocidad de transferencia, pero esta vez sin pasar por la memoria caché del búfer del disco duro y, por lo tanto, leyendo directamente desde el disco. NI IDEA

28. Ejecute el siguiente comando:

```
ls /sys/block/
```

```
agusnfr@agusnfr-VirtualBox:~/Escritorio$ ls /sys/block
loop0  loop1  loop2  loop3  loop4  loop5  loop6  loop7  sda  sr0
```

29. A que le parece que corresponde cada entrada del directorio anterior?

/sys/block: This subdirectory contains one symbolic link for each block device that has been discovered on the system. The symbolic links point to corresponding directories under /sys/devices.

30. Ejecute el siguiente comando(ajuste el dispositivo de disco según su equipo):

```
cat /sys/block/sda/queue/scheduler
```

```
agusnfr@agusnfr-VirtualBox:~/Escritorio$ cat /sys/block/sda/queue/scheduler
[mq-deadline] none
```

31. Investigue el resultado del comando anterior. ¿Que quiere decir cada item del resultado?, investigue cada uno de ellos y asocielo con conceptos de la teoría y de esta práctica. ¿Cual es la diferencia entre los siguientes conceptos?

noop: El programador NOOP inserta todas las solicitudes de E/S entrantes en una cola FIFO simple e implementa la combinación de solicitudes. Este programador es útil cuando se ha determinado que el host no debe intentar reordenar las solicitudes en función de los números de sector que contiene. En otras palabras, el programador asume que el host no sabe cómo reordenar las solicitudes de manera productiva.

deadline:

Garantizar una hora de inicio de servicio para una solicitud. Lo hace imponiendo una fecha límite en todas las operaciones de E/S para evitar la inanición de solicitudes. También mantiene dos colas de fecha límite, además de las colas ordenadas (tanto de lectura como de escritura). Las colas de fecha límite se ordenan básicamente por su fecha límite (el tiempo de vencimiento), mientras que las colas ordenadas se ordenan por número de sector.

Antes de atender la siguiente solicitud, el programador de fecha límite decide qué cola usar. Las colas de lectura tienen mayor prioridad porque los procesos generalmente se bloquean en las operaciones de lectura. A continuación, el planificador de fecha límite comprueba si la primera solicitud de la cola de fecha límite ha caducado. De lo contrario, el planificador atiende un lote de solicitudes de la cola ordenada. En ambos casos, el programador también sirve un lote de solicitudes después de la solicitud elegida en la cola ordenada.

cfq: CFQ coloca solicitudes sincrónicas enviadas por procesos en una serie de colas por proceso y luego asigna intervalos de tiempo para que cada una de las colas acceda al disco . La duración de la porción de tiempo y la cantidad de solicitudes que una cola puede enviar depende de la prioridad de E/S del proceso dado. Las solicitudes asíncronas para todos los procesos se agrupan en lotes en menos colas, una por prioridad. Si bien CFQ no realiza explícitamente "Anticipatory I/O scheduling", logra el mismo efecto de tener un buen rendimiento agregado para el sistema en su conjunto, al permitir que una cola de proceso esté inactiva al final de la E/S síncrona, por lo que "anticipa" más E/S cercanas de ese proceso. Puede considerarse una extensión natural de la concesión de intervalos de tiempo de E/S a un proceso.

Anticipatory IO: Es un algoritmo para programar la entrada/salida del disco duro (programación de E/S). Busca aumentar la eficiencia de la utilización del disco al "anticipar" futuras operaciones de lectura síncrona.

La "inactividad engañosa" es una situación en la que parece que un proceso ha terminado de leer del disco cuando en realidad está procesando datos en preparación de la siguiente operación de lectura. Esto hará que Work-conserving scheduler normal cambie a E/S de servicio desde un proceso no relacionado. Esta situación es perjudicial para el rendimiento de las lecturas sincrónicas, ya que degenera en una carga de trabajo de búsqueda. La programación anticipada supera la ociosidad engañosa al hacer una pausa breve (unos pocos milisegundos) después de una operación de lectura en previsión de otras solicitudes de lectura cercanas.

---

deadline: el programador de E/S Deadline ordena las solicitudes de E/S y las gestiona en el orden más eficiente. Garantiza una hora de inicio para cada solicitud de E/S. También otorga mayor prioridad a las solicitudes de E/S que han estado pendientes durante demasiado tiempo.

cfq: el programador de E/S Completely Fair Queueing (CFQ) intenta asignar de forma justa los recursos de E/S entre procesos. Ordena e inserta solicitudes de E/S en colas por proceso.

noop: el programador de E/S No Operation (noop) inserta todas las solicitudes de E/S en una cola FIFO y, a continuación, las fusiona en una única solicitud. Este programador no ordena ninguna solicitud.

Noop

Es el planificador de Entrada/Salida más simple que existe para el núcleo de Android. Funciona insertando todas las peticiones de Entrada/Salida, dentro de una cola de procesamiento tipo

FIFO (first in, first out, que se traduce como primero en entrar, primero en salir), e implementando fusión de peticiones y reduciendo el tiempo de petición, y la variabilidad del tiempo de servicio de Entrada/Salida.

Deadline

Este planificador de Entrada/Salida funciona de modo similar al tiempo real, utilizando una política de asignación en circuito (round robin), para intentar distribuir equitativamente las peticiones de Entrada/Salida, evitando se agote la capacidad de procesamiento básicamente impone tiempos de ejecucion (deadline) a todas las operaciones de Entrada/Salida, con la finalidad de impedir que se agote la capacidad de recibir peticiones. Utiliza cinco colas de procesamiento, dos de las cuales son ordenadas de acuerdo a los tiempos de ejecución, al mismo tiempo que las colas de procesamiento son ordenadas de acuerdo a su número de sector.

De modo predeterminado, los tiempos de caducidad son de 500 ms para las peticiones de lectura, y de 5 segundos para las peticiones de escritura.

CFQ

CFQ, que es el acrónimo de Completely Fair Queuing, que podría traducirse como encolado de procesamiento completamente justo, es el planificador de Entrada/Salida su objetivo es mantener una cola de procesamiento de Entrada/Salida escalable por proceso, e intentar distribuir equitativamente el uso del procesador disponible para los procesos de Entrada/Salida, entre todas las peticiones de Entrada/Salida manteniendo una buena capacidad de procesamiento, al permitir que las colas de procesamiento puedan pausar al finalizar un procesos de Entrada/Salida, anticipando el procesos de Entrada/Salida más cercano de ese mismo proceso.

32. Como root, ejecute el siguiente comando

```
sudo echo "noop" > /sys/block/sda/queue/scheduler
```

¿Cuál es el efecto de esto?. Ayuda: vuelva a ejecutar

```
cat /sys/block/sda/queue/scheduler
```

Cambia el scheduler (aunque a mi no me deja)

33. Ahora ejecute el siguiente programa al menos 5 veces, de manera tal de poder calcular el promedio del resultado obtenido

```
hdparm -t --direct --offset 500 /dev/sda
```

34. Ahora del mismo modo repita el paso con las demás opciones obtenidas en el ejercicio 29 y compare los resultados

- **¿Cual le parece que debería ser más óptimo?**
- **¿Por qué?**

Esta es mi respuesta.

https://www.ibm.com/docs/en/linux-on-systems?topic=linuxonibm/performance/tuneforsybase/ioschedulers.html

https://www.cloudbees.com/blog/linux-io-scheduler-tuning