Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos
2022 - 1

Tarea 1

Fecha de entrega código: Miercoles 04 de Mayo del 2022.

Información General

La siguiente tarea contempla como objetivo principal la aplicación de contenidos del curso (Sorting, árboles y estrategias algorítmicas) para la resolución de un problema relativamente complejo.
Recordar que existe el discord del curso y las issues del repositorio. Donde el equipo de ayudante estará atento para resolver dudas con respecto a la tarea.

Objetivos

- Investigar de manera personal algoritmos y estructuras que permiten resolver un problema
- Usar técnicas de dividir y conquistar. Además de estrategias recursivas para resolver problemas
- Familiarizarse con la importancia de la complejidad computacional en la realidad

Oye DCCerebro, que vamos a hacer esta noche?

Luego del gran éxito que tuviste al manejar los restaurantes en la comunidad de ratones, se corrió la voz de que eres un programador/a excepcional y se puso en contacto contigo Pinky porque su amigo el ratón DCCerebro está teniendo problemas con sus invenciones para tratar de realizar lo que todas las noches hacen, tratar de conquistar el mundo :o
Para dominar al mundo DCCerebro tiene un objetivo muy concreto. Eliminar a Stuart Little. ¿Por qué Stuart Little? Por traumas que DCCerebro posee de su infancia.
Para efectuar esto DCCerebro quiere hacer dos herramientas. Primero un cañón de fotones-inatron. Cuyo objetivo es ser disparado en la casa de espejos donde se encuentra Stuart. Para esto DCCerebro necesita calcular todas las colisiones y rebotes de los fotones con espejos para así finalmente calcular la trayectoria perfecta de los rayos marcadores.

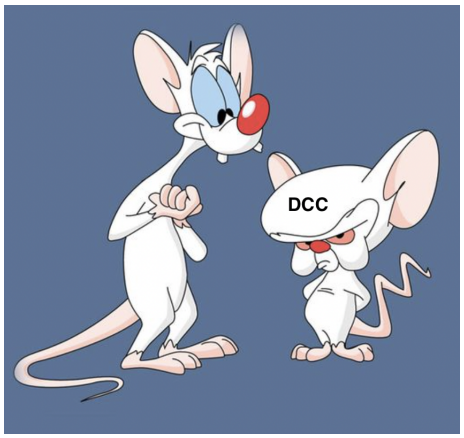

Figura 1: DCCerebro

Por otro lado, Pinky tiene muchos problemas con ordenar las instrucciones que Cerebro le da, ya que que algunas veces le ordena buscar materiales por *indice*, *dureza* u otras propiedades. Muchas veces DCCerebro le entrega la lista de materiales que poseen en la bodega del laboratorio, pero Pinky se demora horas en encontrar lo que se le ordena buscar. Es por esto que también te solicitaron crear un sistema computacional capaz de ayudar a Pinky con la obtención y orden de los materiales.

Problema 1 - DCCRayo de fotones

El primer problema que DCCerebro plantea es simular esta casa de espejos y que dada una configuración inicial de fotones con cierta velocidad y dirección se pueda observar y registrar todas las colisiones con espejos. Pinky ya escribió una solución. Sin embargo, esta es sumamente ineficiente, por lo que DCCerebro te contactó. En la modelación se encuentra lo siguiente:

- Particle: representa un fotón. Posee

```
Particle {
   id
   position
   velocity
}
```

- Segment: representa un espejo. La cual es una recta entre un punto y otro.

```
Segment {
   id
   startPosition
   endPosition
}
```

Tras tener los estados iniciales de *photons* y *segments* se comienza la simulación. Esta es modelada mediante Frames[1]. Al inicio se revisan todas las colisiones entre un fotón y las paredes y, de acuerdo a esto, se calcula y actualiza la nueva dirección de los fotones involucrados. Luego, se actualiza la posición de cada fotón según su velocidad, lo que modificará las colisiones en el siguiente frame.

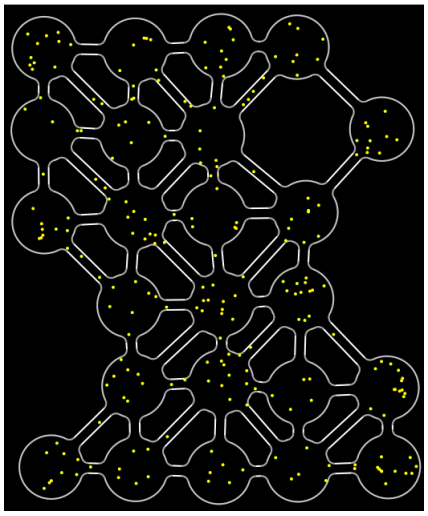

Figura 2: Visualización de la simulación

[1]Son una representación del tiempo que indica cuanto durará la ejecución (en función de cuadros, no segundos), para más información acá, la revisión de colisiones se realiza por cada frame

Solución

En el código base entregado se encuentra una solución directa al problema. Esta consiste en el enfoque directo de revisar que en cada frame, revisar cada fotón y por cada fotón revisar si existe o no colisión con alguna pared. En caso de empates (más de un choque en el mismo frame) la simulación debe seleccionar al segmento con menor id. El pseudocódigo del algoritmo descrito es el siguiente

Algorithm 1 Pseudocódigo del algoritmo directo

```
for frame in F frames do
    for particle in P particles do
        set collision element X to none
        for segments in S segments do
            if particle collides with segment then
                update X to S
            end if
        end for
        Update direction of particle according to X
        Update position of particle
    end for
end for
```

Fácilmente se puede observar que la complejidad de este problema es $\mathcal{O}(FPS)$, con F cantidad de frames, P el número de partículas y S el número de segmentos.

Para esta tarea deberás organizar los segmentos en una estructura de datos que cumpla que su complejidad sea $\mathcal{O}(FP * log(S))$, para poder ejecutar el algoritmo frente a inputs con gran cantidad de segmentos.

Para resolver este problema se recomienda utilizar el algoritmo **B**ounding **V**olume **H**ierarchy (o **BVH** para los amigos). Este algoritmo es esencial en el mundo de los videojuegos ya que permite seccionar el espacio de tal forma que la detección de colisiones sea mucho mas sencilla. **BVH** es un tipo de árbol para objetos geométricos, donde cada uno de los objetos se encuentra contenido por una caja. Esta estructura tiene la característica de que la caja que envuelve un nodo x siempre contiene a las cajas de los hijos de x.

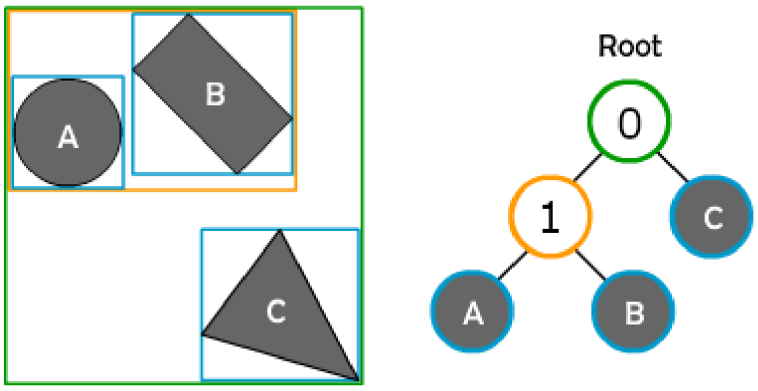

Figura 3: Arbol BVH

En la *Figura 2.* Se observa un ejemplo de **BVH**[2][3] donde la caja del nodo 1 envuelve completamente a los elementos A y B. y a su vez, todos los objetos están envueltos por una caja *raiz*.

Para la solución de este problema se recomienda que implementes un **KD-Tree**[4] de segmentos en dos dimensiones. Sin embargo, si deseas utilizar otra estructura que solucione el problema, eres totalmente libre de hacerlo siempre y cuando no supere la complejidad objetivo[5]

Condiciones/Supuestos

Algunas de las condiciones se podrían tomar como casos borde para los tests

- Las colisiones solamente se realizan entre partículas y pared.

Ejecución

Tu programa se debe compilar con el comando `make` y debe generar un ejecutable de nombre `simulate` que se ejecuta con el siguiente comando:

```
./simulate input.txt output.txt
```

Donde **input** será un archivo con la posición de los segmentos y el estado inicial de los fotones. Por otro lado **output** sera el archivo donde se irán reportando las colisiones en cada frame
Tu tarea será ejecutada con archivos de creciente dificultad, asignando puntaje a cada una de estas ejecuciones que tenga un output igual al esperado.

Input

El input es procesado por una función `simulate_init_from_file(char* input_file, bool visualize)` La cual retorna un objeto `Simulation*` que contiene la cantidad de frames en la simulación, y los arreglos de partículas y segmentos. El booleano, *visualize* indica si se debe abrir o no un visualizador para el problema

Output

El output de tu programa corresponde al mismo output que genera el código base, Por lo que los outputs de tu tarea deberían ser los mismos. Recuerda que el objetivos de esta sección es mejorar el `runtime` del algoritmo, que de por si ya es correcto.

[2] Muy importante para computer graphics Ejemplo
[3] Un muy buen artículo al respecto Link
[4] Wikipedia
[5] $\mathcal{O}(FPlog(S)$

Codigo Base

El problema presentado ya viene totalmente resuelto en el código base. Sin embargo también se agregaron otras carpetas para permitir visualizar las soluciones (Si, visualizar). Encontraras 4 módulos en la tarea:

- `src/simulate`: Solución del problema, aca deben ir tus cambios
- `src/visualizer_core`: modulo a cargo del visualizador. El programa utiliza GTK+3. Asegurate de completar la guia de instalacion
- `src/visualizer`: API de `visualizer_core` para ser usado desde tu programa las funciones para mostrar la simulación ya se muestran en el código base. Para debugear, se dejaron las siguientes funciones
  - `void visualizer_set_color(R, G, B)`: Indica el color que se debera usar para futuras operaciones de dibujo de segmentos. Los valores R, G y B varian entre 0 y 1
  - `bool visualizer_draw_box(BoundingBox boundingbox)`: Dibuja la caja en la interfaz
- `src/engine`: contiene la modelación del problema y sus operaciones. Dentro de este modulo se encuentra `particle.h` que contiene las funciones
  - `bool particle_boundingbox_collision(Particle particle, BoundingBox boundingbox)`; retorna *true* si la particula esta chocando con la caja
  - `bool particle_segment_collision(Particle particle, Segment segment)`; retorna *true* si la particula esta chocando con el segmento
  - `void bounce_against_segment(Particle* particle, Segment segment)`; Cambia la direccion de la *particle* luego de rebotar con un *segment*
  - `void particle_move(Particle* particle)`; Actualiza la posicion de la particula segun su velocidad

Pinky's Managment System

Estuviste ayudando arduamente a DCCerebro con la simulación de fotones, la cual resultó en que el proyecto de de los rayos de destrucción masiva es viable y muy efectivo para eliminar a Stuart Little, por lo que van a proceder a construirlo. Además, como estuviste horas y horas armando el simulador se te olvidó comer y tienes mucha hambre. Pinky y DCCerebro tienen una extensa bodega donde poseen una cantidad absurda de materiales (pues siempre hay que estar listos para construir un arma mortal). El problema es que la bodega está muy pero muy desordenada, con todos los materiales y máquinas (y quien sabe, quizás alguna arma letal por ahí) esparcidos en cualquier parte. Considerando que la construcción de la máquina de destrucción masiva puede llevar horas y horas en ser construido dado que usa mucha cantidad de materiales. Por lo que Pinky se demoraría Mucho mucho tiempo en encontrar los materiales a medida que DCCerebro se los va solicitando.
DCCerebro te pide que construyas un sistema computacional capaz de ser eficiente con la busqueda e insercion de nuevos materiales. Para esto te menciona una estructura en la que tu eres experto un **BST**
Para este problema se te entregarán N inputs desordenados y lo que tendrás que hacer es colocarlos en una estructura de árbol (BST) para realizarle distintas operaciones. Cada línea tendrá K características (todas las líneas con la misma cantidad de características), de la forma $k_0, k_1 ... k_n$ cada línea va a representar a un tipo de material, las características son:

- Material
  - index
  - conductivness
  - hardness
  - inatorness

(Hint: Puedes asumir que todos los numeros son enteros. En caso de empate de valores, hacerlo por orden de input)
Lo que deberas hacer es representar el arbol, con respecto a alguna caracteristica dada por input. Por ejemplo. Si el input dice que las consultas seran con respecto a la `hardness` Entonces el arbol debe ser armado con respecto a eso.
Luego de armado el arbol, se evaluara la construccion de este con las siguientes operaciones

Operaciones

1. `PATH value`

dado una `criteria` dada por input. Esta consulta entrega un `value` que sera el valor que debes buscar en el arbol. El output de esta consulta debe ser el valor de `value` para cada nodo recorrido hasta encontrar el buscado
Por ejemplo considerar el siguiente BST

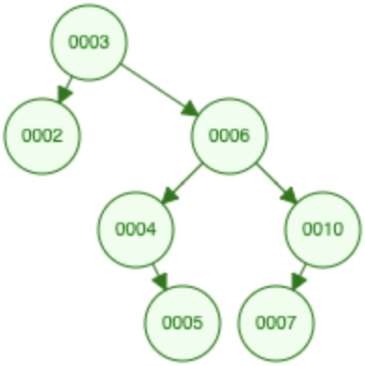

Figura 4: BST de ejemplo

Si la consulta es `PATH 4` Entonces tu programa debe imprimir (al archivo) el resultado **3 6 4** Que es el camino hasta dicho nodo En Caso de no existir el nodo. se debe imprimir el camino y una X al final. Por ejemplo si la busqueda es `PATH 8` se deberia imprimir `3 6 10 7 X`

2. `DEEP value`

Esta consulta al igual que la anterior entrega un valor buscado en el arbol. Como output debes entregar la profundidad a la que se encuentra dicho valor. Considera que el root se encuentra en profundidad 0. En el ejemplo anterior si la consulta es `DEEP 4`, el resultado entregado deberia ser 2

3. `ORDER`

Esta operacion pide que retornes los `values` ordenados. Ojo, esta consulta no puede tener una complejidad mayor a $\mathcal{O}(n)$

4. `DESTROY value`

Sin querer Pinky toco una de las armas de destrucción de DCCerebro, y todo lo que estaba en ese sector de la bodega fue destruido. Se te entrega un `value` y se debe destruir el subárbol de ese nodo, incluyendo al nodo mismo. Por lo que en consecuentes consultas no deberia aparecer. Deberas imprimir el numero de nodos destruidos Por ejemplo si se envia `DESTROY 6` debes entregar 5 como respuesta
Puedes asumir que no se destruira la raiz

Código Base y Ejecución

Tu programa se debe compilar con el comando `make` y debe generar un ejecutable de nombre `pinky` que se ejecuta con el siguiente comando:

`./pinky input.txt output.txt`

El codigo base posee solo un archivo `main.c` encargado de manejar la lectura inicial del archivo. El resto de la logica debe ser creado por ustedes

Input y Output

Como input se entregara primero un valor numerico *criteria* que indica que criterio se utilizara para las consultas. Este criteria significa lo siguiente

```
0: index
1: conductivness
2: hardness
3: inatorness
```

Luego se entrega una linea con un numero N que indica el numero de nodos que se entregaran, seguido de las N lineas que describen cada nodo de la forma

```
index conductivness hardness inatorness
```

Finalmente se entregara una linea S que indica el numero de consultas a realizar seguido de S lineas con cada consulta en el formato descrito anteriormente
Ejemplo

```
3
2
51 414 23 104
24 1 32 52
4
PATH 10
DEEP 104
DESTROY 23
ORDER
```

Tu output deberan ser S lineas con los resultados de cada consultas

Consideracion Importante

Tu repositorio sera creado de la siguiente forma

```
src\
    engine\
    simulate\
    visualizer\
    visualizer_core\
    pinky\
```

Y tu deberas trabajar en las carpetas simulate y pinky. Ademas al hacer make se generaran 3 ejecutables

- simulate: Ejecutable de la Parte 1 (Tu solucion)
- pinky: Ejecutable Parte 2 (tu solucion)
- visualizer: Ejecutable que utilizamos para el visualizador

Recomendación de tus ayudantes

Las tareas requieren de mucha dedicación de tiempo generalmente, por lo que desde ya te recomendamos distribuir tu tiempo considerando los plazos definidos. Así mismo, te recomendamos fuertemente que antes de empezar a programar tu tarea, leas el enunciado y te dediques a entender de manera profunda lo que te pedimos. Una vez que hayas comprendido el enunciado, dedica el tiempo que sea necesario para la planificación y modelación de tu solución, para posteriormente poder programar de manera eficiente. Estos son consejos de tus ayudantes que te pueden ayudar a pasar el ramo :)

Uso de memoria

Parte de los objetivos de esta tarea, es que implementen en la práctica el *trade-off* entre memoria y tiempo, es por esto que independiente del test, tendrán como máximo `1.2 GB` de memoria `RAM` disponible. Pueden revisar la memoria que utiliza su programa con el comando `htop`. Además, el servidor avisará en caso de superar el máximo permitido.

Eficiencia

Se solicita que el programa sea eficiente. No podrá tomar más de 10 segundos `user + sys`, pueden utilizar `time` seguido del comando de ejecución de su programa para revisarlo.

Evaluación

La nota de tu tarea se descompone como se detalla a continuación:

- 95% que el output de tu programa sea correcto y eficiente. Para esto el puntaje se divide en partes iguales para los tests Easy, Medium y Hard.
  - 65% a la nota para el problema 1
  - 35% a de la nota al problema 2
- **Flawless Memory!**: 5% de la nota por no tener errores de memoria ni leaks (se dividen en 2.5 y 2.5)

Para cada test de evaluación, tu programa deberá entregar el output correcto en menos de 10 segundos. De lo contrario, recibirás 0 puntos en ese test.

Bonuses

- **The Tree is a Lie**:Se publicara un cuestionario con respecto a la tarea, este cuestionario es para que estudien el enunciado y lo comprueben. Obtendras 5% adicional a tu nota en caso de responder el cuestionario de la tarea con una aprobacion mayor al 90%
- **You Must Construct Additional Trees!**: Abriremos un concurso de memes atingente al curso. Si, leiste bien, memes, el formulario se encuentra en canvas. Y a los cinco mejores les daremos un dia adicional de atraso para que usen durante el semestre. (No aplicable para esta tarea)

Entrega

Código: GIT - Rama master del repositorio asignado. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.
Importante: No debes modificar el makefile o tu programa no compilara en correccion

Atraso: A lo largo del semestre tendrás 4 días de gracia, los cuales podrás utilizar en caso de que no alcances a entregar alguna tarea en el tiempo indicado. Para esto, puedes usar un máximo de 2 días en una tarea, sin importar si tienes más días disponibles, y tendrás que avisar si quieres que se revise un commit posterior a la fecha de entrega en el formulario que se mandará el día siguiente. Cabe destacar que si entregas a las 00:01 hrs perderás un día en caso de llenar el formulario, y no será revisado ese commit en caso de que decidas no contestarlo. Por otro lado, si se te acaban los 4 días y entregas una tarea atrasada, entonces tendrás la calificación mínima **sin derecho a reclamo.**

Integridad académica

Este curso se adscribe al Código de Honor establecido por la Escuela de Ingeniería. Todo trabajo evaluado en este curso debe ser hecho **individualmente** por el alumno y **sin apoyo de terceros**. Se espera que los alumnos mantengan altos estándares de honestidad académica, acorde al Código de Honor de la Universidad. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Pregrado de la Escuela de Ingeniería.

Figura 5: Funny Meme

¡Éxito! :)