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

IIC2133 — Estructuras de Datos y Algoritmos
2023 - 1

Tarea 1

Fecha de entrega código: 26 de Abril del 2023 23:59 UTC-4
Link a generador de repos: Generar repo base

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

Parte 1: Matepossible

En los lugares más recónditos de Londres, realizando su última misión antes de volver a casa, se encuentra el auto mas grande, valiente y guapo de tooooodo el mundo, el mismísimo Mate, como To-mate, pero sin el To.

Para esta misión mate necesita una pareja para detener el crimen, efectivamente, Tú, la segunda persona más grande, valiente y guapa de tooooodo el curso IIC2133, decides ofrecerle tu ayuda para garantizar el éxito de la última misión de mate.

Para su misión juntos, deben ir a la ciudad vecina para recolectar información sobre el equipo de carreras rival que está planeando sabotear el próximo campeonato en Radiator Springs. Avanzados en la misión, Mate ha logrado infiltrarse en el equipo rival y ha obtenido un pedazo de papel con una serie de números extraños. Al parecer, estos números son parte de un código secreto que contiene información vital sobre el plan de sabotaje. Pero hay un problema: ¡Mate no sabe cómo ordenarlos para encontrar la información clave para el plan maestro!

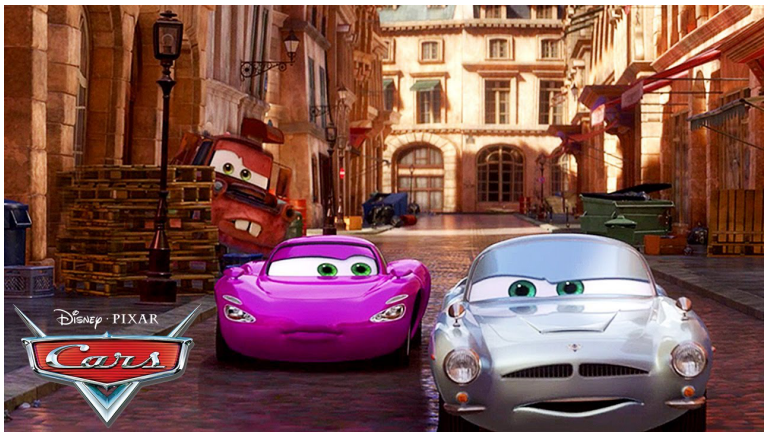

Figura 1: Mate super espia

Por suerte, Mate no está solo, te tiene a ti como su fiel compañero. Y, en base a tus profundos conocimientos sobre programación le recomiendas ordenar los números en un BST, para que luego puedan investigar sobre los planes malévolos de los rivales al encontrar pistas escenciales dentro del árbol.

Problema

Para este problema se te entregará un input con N valores desordenados y lo que tendrás que hacer es colocarlos en una estructura de árbol (BST) para realizarle distintas operaciones.

(Hint: Puedes asumir que todos los números son enteros. No se entregarán inputs con empate).

Lo que deberás hacer es representar el árbol, con respecto a los valores de los nodos. Luego de armado el árbol, se evaluará la construcción de este con las siguientes operaciones.

Además, el equipo de ayudantes preparo una cápsula para implementar BST en C que puede ser de mucha utilidad. Puedes encontrarla en el siguiente enlace.

Operaciones

1. `PATH value`

Esta consulta entrega un `value` que será el valor que debes buscar en el árbol. 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

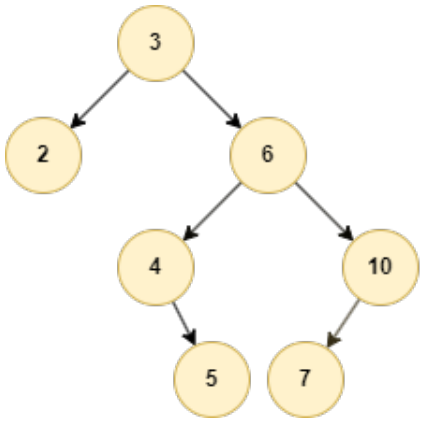

Figura 2: BST de ejemplo

Si la consulta es `PATH 4` entonces tu programa debe escribir (al archivo) el resultado **3 6 4** que es el camino hasta dicho nodo. En caso de no existir el nodo, se debe escribir el camino y una X al final. Por ejemplo, si la búsqueda es `PATH 8` se debería escribir `3 6 10 7 X`

2. `DEEP value`

Esta consulta al igual que la anterior entrega un valor buscado en el árbol. 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 escrito debería ser 2. En caso de no existir el nodo, se debe escribir -1.

3. `ORDER`

Esta operación pide que retornes los `values` ordenados. Ojo, esta consulta no puede tener una complejidad mayor a $\mathcal{O}(n)$. Entonces, en caso del ejemplo, tu programa debería escribir `2 3 4 5 6 7 10`

4. `DEEP-ORDER`

Esta operación pide que escribas los `values` ordenados por niveles. Es decir, debes imprimir cada nivel del árbol presentando los nodos de menor a mayor dentro del nivel. Según el ejemplo anterior se debería escribir `3 2 6 4 10 5 7`, dado que en el primer nivel está solo el nodo 3, en el segundo nivel se encuentra el nodo 2 y 6, y así iterativamente. Notar que esta función no se puede ver influenciada por el comando `INVERT` explicado más abajo. Siempre se debe imprimir en orden ascendente el nivel de profundidad.

5. `INVERT`

Esta consulta pide que inviertas completamente el árbol. Deberás cambiar el árbol original por el nuevo árbol invertido, es decir, los hijos derechos pasan a ser hijos izquierdos, y los izquierdos pasan a ser hijos derechos.

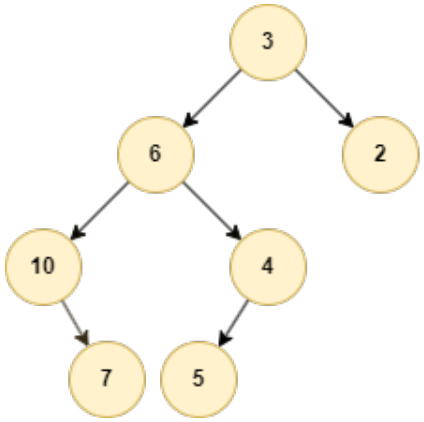

Figura 3: BST INVERT de ejemplo

Para reflejar este cambio, luego de invertir el árbol debes imprimir los `values` por niveles de izquierda a derecha. Según el ejemplo anterior, antes de invertir se debería escribir `3 2 6 4 10 5 7` y luego de invertir sería `3 6 2 10 4 7 5`
Hint: Notar que esta impresión es muy parecida a `DEEP-ORDER`, solo que esta función si cambia la forma de impresión al estar invertido el árbol, es decir, no debe ordenar en orden ascendente los nodos en nivel.

Ejecución

Código Base y Ejecución

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

```
./bstmate input.txt output.txt
```

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

Input y Output

Como input se entregará primero una línea con un número N que indica el número de nodos que se entregarán, seguido de una línea con los valores de los N nodos. Pueden asumir que **no se entregarán nodos repetidos**. Finalmente, se entregará una línea Q que indica el número de consultas a realizar, seguido de Q líneas con cada consulta en el formato descrito anteriormente.
Ejemplo:

```
4
51 414 23 104
12
PATH 10
DEEP 104
PATH 104
DEEP-ORDER
ORDER
INVERT
PATH 10
DEEP 104
PATH 104
DEEP-ORDER
ORDER
INVERT
```

Tu output deberán ser Q líneas con los resultados de cada consulta; debería verse así:

```
51 23 X
2
51 414 104
51 23 414 104
23 51 104 414
51 414 23 104
51 23 X
2
51 414 104
51 23 414 104
23 51 104 414
51 23 414 104
```

Parte 2: DCCodersSort

El mejor equipo de espías del DCC (tu y To-mate) descubre que los enemigos planean viajar a robar una máquina misteriosa ubicada en Radiator Springs. Es por esto que llega el momento de que Mate y tu vuelvan a casa, para contarle el chismecito a sus habitantes y así puedan prepararse para la llegada de los rivales.

Para escapar de la ciudad y pasar desapercibido ante sus enemigos, Mate, finge ser un jefe de carreras, y tu un trabajador de los pits. Sin embargo, la junta directiva se los encuentra antes de escapar, y dado que creen que realmente trabajan allí, les piden analizar los datos de las carreras durante el último mes, para determinar cuáles son los modelos de automóviles más rápidos y cuántas victorias ha logrado cada conductor. Claramente, deben mantener su cuartada, asique recae en tus habilidades de programador hacer el ranking de pilotos dado que mate debe distraer a los empresarios.

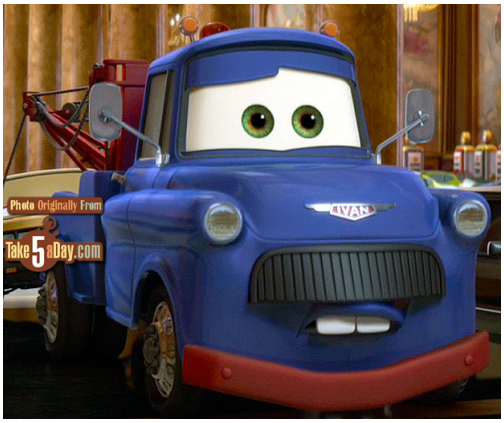

Figura 4: Mate super espia

Problema

En este problema debes implementar QuickSort para ordenar los competidores de mejor a peor en función de sus estadísticas. En la base de datos, la información de cada conductor incluye el `id del conductor, el modelo, tiempo promedio por vuelta y número de victorias`.

El ranking generado por tu programa debe tener la siguiente preferencia de orden:

1. numero de victorias
2. tiempo promedio por vuelta
3. modelo
4. id

Para esta parte de la tarea **NO se puede utilizar qsort**, tu debes ser quien implemente el algoritmo de ordenación. Además, puedes asumir que ningún auto posee el mismo id.

Código Base y Ejecución

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

```
./qsortcars input.txt output.txt
```

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

Input y Output

Como input se entregará primero una línea con un número N que indica el número de autos que se entregarán, seguido de las líneas con los datos de cada auto. En el orden: ID, Modelo, Tiempo promedio de vuelta y cantidad de victorias.
Ejemplo:

```
4
1.0 1.1231 5.2 1.0
2.0 5.3625 5.4 2.0
3.0 6.2901 5.6 3.0
4.0 9.2312 5.3 1.0
5.0 4.3823 5.3 1.0
6.0 9.2312 5.3 1.0
```

Tu output debería ser:

```
3.0 6.2901 5.6 3.0
2.0 5.3625 5.4 2.0
1.0 1.1231 5.2 1.0
5.0 4.3823 5.3 1.0
4.0 9.2312 5.3 1.0
6.0 9.2312 5.3 1.0
```

Parte 3: Mate, eso nunca paso.

Finalmente, tú y mate pueden volver a casa. Sin embargo se enteran que los profesores habían tenido una discusión respecto a cual era el auto favorito de la población chilena, ante lo cual mate se enoja bastante. Pero no porque los profesores casi dejaron de ser BFF's, sino porque no lo incluyeron en la papeleta de votación. Es por ello, que mate decide pasarse al lado obscuro y llama a sus súbditos alienígenas adiestrados en sus diversos viajes por el mundo.
Ahora, el gran dictador To-mate ordena que atrapen a todos los ayudantes, profesores y alumnos en su poderosa DCCapsula de *rashos laser* para castigarlos a todos por dañar su precioso corazón.

Figura 5: Mate enojado

Sabes que Mate tiene la razón al enojarse, sin embargo, tú lo conoces desde chiquito, y le suplicas por tu vida, y la de todos los castigados. Le imploras que recapacite y no los fría a todos.
To-mate, pese a tener un frío corazón vengativo, aun así es un buen líder y adjudica por lo mejor para sus súbditos. Es por ello que les te ofrece un trato para mejorar el bienestar de sus seguidores alienígenas. Mate te dice que si eres capaz de mejorar su cápsula, la cual funciona mucho más lento de lo que cualquier tecnología futurista, liberará y perdonará a los castigados. Así que, ahora, el destino de tus compañeros, ayudantes y profesores está en tus manos.

DCCapsula

El problema que posee Mate, es que sus *rashos* son extremadamente lentos, entonces decides analizar como se estructura dicha arma, luego de pasar unas horas, pides ver la modelacion al problema actual,

- 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.

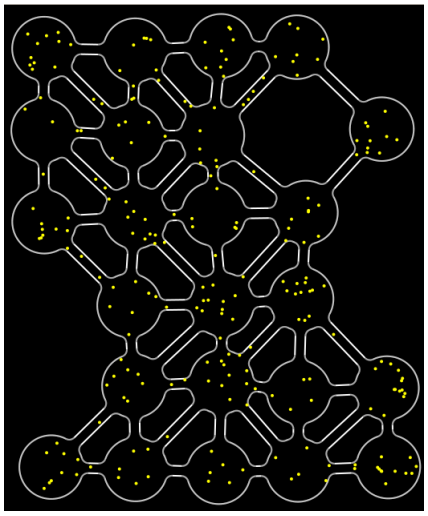

Figura 6: Visualización de la simulación

[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.

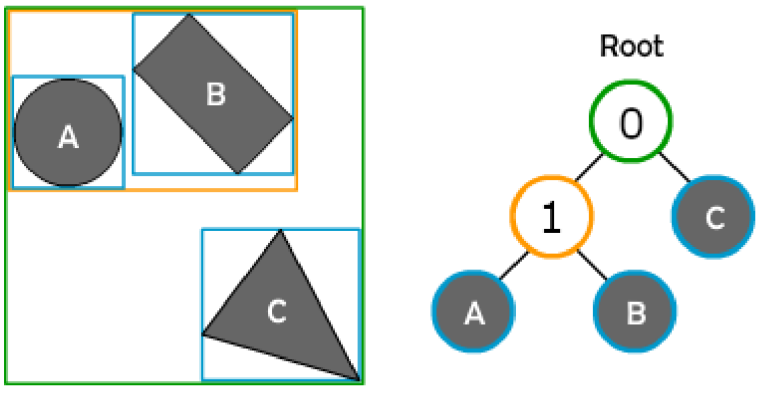

Figura 7: Arbol BVH

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].
Hint: **Puedes reciclar el código utilizado en la parte 2 de esta tarea si lo estimas conveniente.**

Ejecución

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

```
./kdtree 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.

En caso de problemas con el visualizador puedes utilizar:
`./kdtree input.txt output.txt --novis`

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/kdtree`**: 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

Cuestionario

El cuestionario será una mini evaluación de Canvas donde ser realizaran preguntas de alternativas y deberás responder preguntas sobre el enunciado y las estructuras de datos involucradas. Tendrás intentos ilimitados y se utilizará el mejor puntaje.

Evaluación

La nota de tu tarea es calculada a partir de tests. Separando entre la Parte 1 y 2 de la tarea

- Un 95 % de nota asignada por código
  - 40 % Parte 1: BSTMATE
  - 20 % Parte 2: QUICKSORT
  - 40 % Parte 3: KDTree
- Un 5 % correspondiente a manejo de memoria (No Leaks, No errores)
- Hasta 5 décimas de bonus: Cuestionario

Para cada test de evaluación, tu programa deberá entregar el output correcto en menos de 10 segundos y utilizar menos de 1 GB de ram[6]. De lo contrario, recibirás 0 puntos en ese test.

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 :)

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.

Política de atraso: La política de atraso del curso considera dos criterios:

- Utilización de **cupones**[7] de atraso. Donde un cupón te proporciona 24 horas adicionales para entregar tu tarea sin penalización a tu nota
- Entrega atrasada sin cupón donde se aplica un descuento *suave*[8]. Calculada de la forma

$$N_f = min(70 - 7 \cdot d, N_o)$$

  Donde d es la cantidad de días atrasados **con un máximo de 4 días**[9], N_f la nota final y N_o la nota obtenida en la tarea

[6] Puedes revisarlo con el comando htop o con el servidor
[7] Recuerda que solo tienes 2 cupones para todo el semestre
[8] Es un descuento a la nota máxima posible
[9] Luego de eso la nota máxima obtenible es un 1.0

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.