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

IIC2133 – Estructuras de Datos y Algoritmos

2025 - 1

Tarea 3

Fecha de entrega código: 25 de Junio, 23:59 Chile continental

Link a generador de repos: CLICK AQUÍ

Objetivos

- Diseñar una estrategia de backtracking para solucionar un problema de restricciones.
- Identificar e implementar la técnica algorítmica adecuada para encontrar la solución óptima.
- Explorar el desarrollo de algoritmos que se emplean sobre grafos.

Introducción

¡Últimos tres días de verano en Danville! Phineas y Ferb están decididos a exprimir cada momento, pero sus planes vienen acompañados de desafíos nada sencillos. Candace, como siempre, intenta arruinarles la fiesta, aunque las cosas no le resultan tan fáciles. Mientras tanto, Doofenshmirtz se ha involucrado en una situación inesperada que también requiere atención. ¡Tendrás que usar todo tu ingenio para ayudar a que el verano termine en paz y sin contratiempos!

Tres misiones, una mente brillante. ¿Listo para ayudar a salvar el verano?

Parte 1: Agenda del Verano de Phineas y Ferb

Quedan sólo tres días de verano, y Phineas y Ferb están decididos a exprimir hasta el último minuto de diversión. Para lograrlo, consiguieron 9 lugares vacíos en el vecindario (agrupados en 3 zonas: Zona A: L1-L2-L3, Zona B: L4-L5-L6, Zona C: L7-L8-L9), donde podrán realizar 9 actividades distintas. Además, cada día se divide en 3 franjas horarias: mañana, tarde y noche. Esto implica que existen 9 momentos distintos, considerando los 3 días y las 3 franjas horarias de cada uno.

Resumiendo lo anterior, considera que los elementos disponibles son los siguientes:

- 9 actividades, enumeradas del 1 al 9.
- 3 días, enumerados del 1 al 3. Además, cada día cuenta con 3 franjas horarias: mañana, tarde y noche. Esto implica que existen 9 momentos diferentes en total.

Día 1			Día 2			Día 3		
Mañana	Tarde	Noche	Mañana	Tarde	Noche	Mañana	Tarde	Noche

- 9 lugares, enumerados del 1 al 9. Además los lugares se agrupan en 3 zonas, tal y como se indica en siguiente la tabla:

Zonas	A	B	C
Lugares	L1, L2, L3	L4, L5, L6	L7, L8, L9

Tu objetivo es llenar la agenda, es decir, todo lugar debe tener alguna actividad en todo momento. Sin embargo, para evitar confusiones y garantizar que todo funcione sin problemas, debes seguir estas tres reglas clave:

1. Actividades únicas por momento: En cada franja horaria (mañana, tarde o noche) de cada día, los 9 lugares deben tener actividades diferentes. Por ejemplo, si ya se realiza la Actividad 3 el Día 1 en la mañana, entonces esa misma actividad no podrá realizarse en ningún otro lugar el mismo Día 1 por la mañana.
2. Actividades únicas por lugar: Cada lugar debe albergar las 9 actividades sin repeticiones a lo largo de los tres días. Es decir, si en L4 ya se hizo la Actividad 5, no puede repetirse la misma actividad en L4.
3. Sin repeticiones en la misma zona y día: En cada día, los 3 lugares de una misma zona (ej: L1, L2, L3) deben tener actividades diferentes en las 3 franjas (mañana, tarde, noche). Por ejemplo, si en L2 se hace la Actividad 8 el Día 3 por la mañana, ni L1 ni L3 pueden hacer esa misma actividad el Día 3 en ninguna franja horaria.

Eventos

Primero recibirás un entero E, que indica la cantidad de eventos. Luego, recibirás los E eventos correspondientes.

`VALIDATE {agenda}`

Recibirás una agenda completa. Esto se hará en 9 líneas donde cada línea indica las actividades programadas para un lugar. Además, cada línea contiene 9 actividades representadas por: Actividad, Día y Franja horaria.

input

```
VALIDATE
9 1 tarde 3 3 tarde ... 4 1 manana
.
.
.
8 2 noche 7 3 manana ... 1 2 noche
```

En este ejemplo, en L1 se realizará la actividad 9 el día 1 por la tarde, la actividad 3 el día 3 por la tarde, y así sucesivamente hasta completar 9 actividades. De esta forma, recibirás 9 líneas indicando las actividades agendadas para cada lugar.

Luego, deberás verificar si la agenda recibida cumple con todas las reglas y entregar el siguiente output según el caso:

Si cumple con todas las reglas:

output

```
La agenda es valida.
```

En caso contrario:

output

```
La agenda no es valida.
```

SOLVE-PARTIAL {k} {agenda}

Recibirás una agenda parcial válida[1] y debes indicar de cuántas formas es posible completar la agenda siguiendo las reglas indicadas previamente. Para ello, primero recibirás un entero `k`, que representa la cantidad de actividades ya agendadas. Luego recibirás `k` líneas con las actividades que componen la agenda parcial. Cada actividad será representada por: Actividad, Lugar, Día y Franja horaria. El input se ve de la siguiente forma:

input

```
SOLVE-PARTIAL k
9 4 2 tarde
4 2 1 manana
.
.
.
3 1 3 noche   # la k-esima actividad agendada.
```

El output esperado es el siguiente:

output

```
Soluciones encontradas: {cantidad_de_soluciones}.
```

[1] Una agenda parcial válida quiere decir que ya hay actividades agendadas y todas cumplen con las reglas. Sin embargo, aún falta agendar más actividades para completar la agenda.

Ejecución

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

```
./schedule input output
```

donde `input` será un archivo con los eventos a simular y `output` el archivo donde se guardarán los resultados.

En caso de querer correr el programa para ver los leaks de memoria utilizando valgrind deberás utilizar el siguiente comando:

```
valgrind ./schedule input output
```

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 y output deben seguir el formato indicado en el apartado de eventos.

Parte 2: ¡Mamáaa! Debes ver lo que Phineas y Ferb están haciendo

¡La ciudad está llena de inventos de Phineas y Ferb! Candace, por supuesto, los ha vigilado y está decidida a que esta vez su mamá sí los vea. Pero hay un problema... Algo extraño ocurre cada vez que intenta mostrar un invento: misteriosamente, otros inventos desaparecen justo antes de que llegue su mamá. Candace debe elegir con mucho cuidado qué inventos mostrar, para asegurarse de mostrar los inventos más peligrosos que han hecho sus hermanos y finalmente convencer a su mamá.

Objetivo

Candace ha registrado en una lista qué tan peligrosos son los inventos que sus hermanos han hecho. Pero si elige mostrar un invento de peligro X, los inventos de peligro $X-1$ y $X+1$ desaparecen misteriosamente. Esto la ha obligado a ser extremadamente estratégica al elegir qué inventos presentar como evidencia. Tu misión es ayudarla a generar un reporte que indique:

- El máximo total de peligro que puede mostrarle a su mamá.
- La cantidad mínima de inventos necesarios para alcanzar ese total de peligro.
- La cantidad máxima de inventos posibles para alcanzar ese total de peligro.

Problema

Se te entregará la siguiente información:

- En la primera línea un entero K, que indica el rango de peligro posible de un invento (entre 1 y K inclusive).
- Luego un entero N indicando el número total de inventos registrados por Candace.
- Finalmente, se te entregarán N líneas. Cada una representa un invento y contiene un entero entre 1 y K que indica el nivel de peligro del invento.

Debes indicar el total de peligro más alto que Candace puede mostrarle a su mamá, junto a la menor y mayor cantidad de inventos con que es posible alcanzar ese total de peligro.

Ejecución

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

```
./reportinator input output
```

donde `input` será un archivo con los eventos a simular y `output` el archivo donde se guardarán los resultados.

En caso de querer correr el programa para ver los leaks de memoria utilizando valgrind deberás utilizar el siguiente comando:

```
valgrind ./reportinator input output
```

Tu tarea será ejecutada con archivos de creciente dificultad, asignando puntaje a cada una de estas ejecuciones que tenga un output igual al esperado. A continuación detallaremos los archivos de input y output.

Input

input

```
5
4
2
2
1
3
```

Output

output

```
Total: 4
Minimo: 2
Maximo: 2
```

Explicación

Recibimos primero el valor 5, que establece el rango de peligro (de 1 a 5 inclusive). Luego, el número 4 indica la cantidad total de inventos registrados, seguido por los niveles de peligro de cada uno.

Existen dos estrategias posibles para seleccionar los inventos:

- Si seleccionamos los inventos de peligro 2, automáticamente se descartan los de peligro 1 y 3. Esto nos deja con [2, 2], obteniendo un total de 4 usando exactamente 2 inventos.
- Si comenzamos por los inventos de peligro 1, se eliminan los de peligro 2. Los inventos restantes serían [1, 3]. Al seleccionar el de peligro 3, obtenemos igualmente un total de 4 con 2 inventos.

En ambos casos:

- El peligro máximo alcanzable es 4.
- La cantidad mínima de inventos necesarios es 2.
- La cantidad máxima de inventos posibles es 2.

Parte 3: En busca de Globito

Doofenshmirtz está marcando lugares en el mundo donde puede estar Globito y necesita saber donde empezar a buscarlo. Por suerte, puedes salvar el día con tu conocimiento de estructuras de datos y profundamente poderosos algoritmos para modelar este problema.

Problema

Se te entregará la representación de distintos lugares conectados entre sí de la siguiente forma:

- La primerá línea será un número V que indica la cantidad de lugares. Se identificará a cada lugar con un ID del 0 al $V - 1$.
- La segunda línea será un número E que indica la cantidad de caminos entre lugares.
- Luego, las siguientes E líneas seran pares de IDs de lugares separados por un espacio. Cada una de estas líneas representa un camino entre dos lugares presentes.
- Finalmente, recibirás una última linea que será el ID de un lugar, donde se cree que globito podrías encontrar a globito.

Los caminos son simétricos. Es decir, si existe un camino desde A a B entonces tambien puedes ir desde B a A.

Debes determinar la cantidad de lugares desde donde puedes alcanzar el lugar en que sospechas que se encuentra globito.

Ejecución

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

```
./balloony input output
```

donde `input` será un archivo con los eventos a simular y `output` el archivo donde se guardarán los resultados.

En caso de querer correr el programa para ver los leaks de memoria utilizando valgrind deberás utilizar el siguiente comando:

```
valgrind ./balloony input output
```

Tu tarea será ejecutada con archivos de creciente dificultad, asignando puntaje a cada una de estas ejecuciones que tenga un output igual al esperado. A continuación detallaremos los archivos de input y output.

Input

input

```
10
6
0 1
2 4
5 7
2 3
4 0
9 6
0
```

Output

output

```
4
```

Explicación

Recibimos primero el valor 10, que indica la cantidad total de lugares (numerados del 0 al 9). Luego, el número 6 especifica la cantidad de conexiones entre estos lugares, seguido por los pares de lugares conectados. Finalmente, el último valor 0 señala el lugar donde se sospecha que podemos encontrar a Globito. Luego, analizamos las conexiones posibles:

- Conexiones directas:
  - El lugar 1 está conectado directamente con 0.
  - El lugar 4 está conectado directamente con 0.
- Conexiones indirectas:
  - El lugar 2 llega a 0 a través de 4.
  - El lugar 3 llega a 0 a través de 2 y 4.

Por lo tanto, encontramos 4 conexiones al lugar 0.

Informe

Además del código de la tarea, se debe redactar un **breve** informe que explique cómo se pueden implementar algunas estructuras y algoritmos del enunciado. Les recomendamos **comenzar la tarea escribiendo este informe**, ya que pensar y armar un esquema antes de pasar al código ayuda a estructurarse al momento de programar y adelantar posibles errores que su solución pueda tener.

Este informe se debe entregar en un archivo PDF escrito usando LaTeX junto a la tarea (por lo cual tienen la misma fecha de entrega), con nombre `informe.pdf` en la raíz del repositorio. En este se debe explicar al menos los siguientes puntos:

- Parte 1
  1. Indique cómo modeló el problema para resolverlo. Explique brevemente las decisiones tomadas en cuanto a variables, dominios y restricciones.
  2. ¿Cómo contribuyen las restricciones del problema a reducir el espacio de búsqueda? Dé un ejemplo concreto.
- Parte 2
  1. Explique qué técnica utilizó para resolver el problema e indique la complejidad temporal y de memoria de su solución, en términos de K y N.
  2. Describa qué modificaciones serían necesarias para recuperar el conjunto de inventos seleccionados que componen el total de peligro máximo. ¿Qué estructuras o estrategia utilizaría para encontrar una solución óptima?
- Parte 3
  1. Explique cómo representó y almacenó los lugares y sus caminos.
  2. Indique y justifique la complejidad temporal de su algoritmo expresado en función de V y E.
- Bibliografía: Citar código de fuentes externas.

Cómo citar código: Para asegurar una mejor comprensión y evaluación del informe, es **esencial** que cites el código al que haces referencia. Cada vez que menciones un fragmento de código, incluye el nombre del archivo y el número de línea correspondiente y su *hipervínculo* correspondiente. Puedes aprender cómo crear un *permalink* a tu código consultando la documentación de GitHub. Esto facilitará la revisión y garantizará que el informe sea coherente con la implementación.

Ejemplo: Si en la Parte 1 del informe estás explicando una función que implementa una determinada operación, la referencia al código podría ser así:

> En nuestra implementación, utilizamos una lista enlazada para manejar la estructura de datos, como se muestra en la función `insertar_elemento` (`src/estructuras.c`, línea 45). Esta elección permite...

De esta manera, aseguras que los revisores puedan localizar rápidamente el fragmento de código relevante.

Nota: Si por alguna razón no lograste completar alguna parte del código mencionado en el informe, **aún puedes obtener puntaje** en la sección correspondiente. Para ello, debes describir claramente cuál era tu idea de implementación, los pasos que planeabas seguir, y explicar por qué no pudiste completarlo. Esto demuestra tu comprensión del problema y del proceso, lo cual también es valioso para la evaluación.

Finalmente, **puedes** referenciar código visto en clases sin necesidad de incluir el código o pseudocódigo en el informe. Puedes encontrar un template en Overleaf disponible en este enlace para su uso en la tarea. Para aprender a clonar un template, consulta la documentación de Overleaf.

No se aceptarán informes de más de tres páginas (sin incluir bibliografía), informes ilegibles, o generados con inteligencia artificial. Además, en caso de no cumplir con el formato de entrega indicado previamente, habrá un descuento de 10 décimas sobre la nota del informe.

Evaluación

La nota de tu tarea es calculada a partir de testcases de input/output, así como un informe escrito en LaTeX que explique un modelamiento sobre cómo abarcar el problema, las estructuras de datos a utilizar, etc.
La ponderación se descompone de la siguiente forma:

Informe (20 %)	**Nota entre partes (70 %)**	**Manejo de memoria (10 %)**
7 % Parte 1	25 % Tests Parte 1	5 % Sin leaks de memoria
7 % Parte 2	25 % Tests Parte 2	5 % Sin errores de memoria
6 % Parte 3	20 % Tests Parte 3	

Para cada test de evaluación, tu programa deberá entregar el output correcto en **menos de 5 segundos** y utilizar menos de 1 GB de RAM[2]. De lo contrario, recibirás 0 puntos en ese test.

Recomendación de tus ayudantes

Estas tareas generalmente requieren de mucha dedicación, por lo que desde ya te recomendamos distribuir tu tiempo considerando los plazos definidos (Recuerda que la entrega es el 25 de Junio!). Asimismo, te recomendamos fuertemente que antes de empezar a programar tu tarea, **leas el enunciado detenidamente y con anticipación para que te dediques a entender de manera profunda lo que te pedimos**. Una vez hayas comprendido el enunciado, dedica el tiempo que sea necesario para la planificación, modelación y elaboración de tu informe, para posteriormente poder programar de manera más eficiente. No olvides compilar el código como se indica en la tarea y de preguntar cualquier duda o por problemas que te surjan en Discussions. Estos son consejos de tus ayudantes que te pueden ayudar a pasar el ramo.

Entrega

Código: GIT - Rama principal del repositorio asignado, que debes generar con el link dado al principio de este enunciado. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Atraso: Esta tarea considera la política de atraso y cupones especificada en el repositorio del curso.

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.

Uso de IA

Se prohíbe el uso de herramientas de inteligencia artificial para la realización de esta tarea. Esto incluye, pero no se limita a, el uso de modelos de lenguaje como ChatGPT, Copilot, u otras tecnologías similares para la generación automática de código, soluciones, o cualquier parte del trabajo requerido. Nos reservamos el derecho de revisar todas las tareas entregadas con el fin de detectar el uso de inteligencia artificial en la creación de las soluciones. Las tareas en las que se determine que se ha utilizado IA serán consideradas como una infracción a la política de honestidad académica y serán tratadas como casos de copia.

[2]Puedes revisarlo con el comando `htop` u ocupando `valgrind --tool=massif`