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

IIC2133 – Estructuras de Datos y Algoritmos

2024 - 1

Tarea 2

Fecha de entrega código: 29 de mayo

Link a generador de repos: CREAR REPOSITORIO AQUÍ

Objetivos

- Emplear optimizaciones de búsqueda usando hashing.
- Diseñar una estrategia de backtracking para solucionar un problema de restricciones.

Introducción

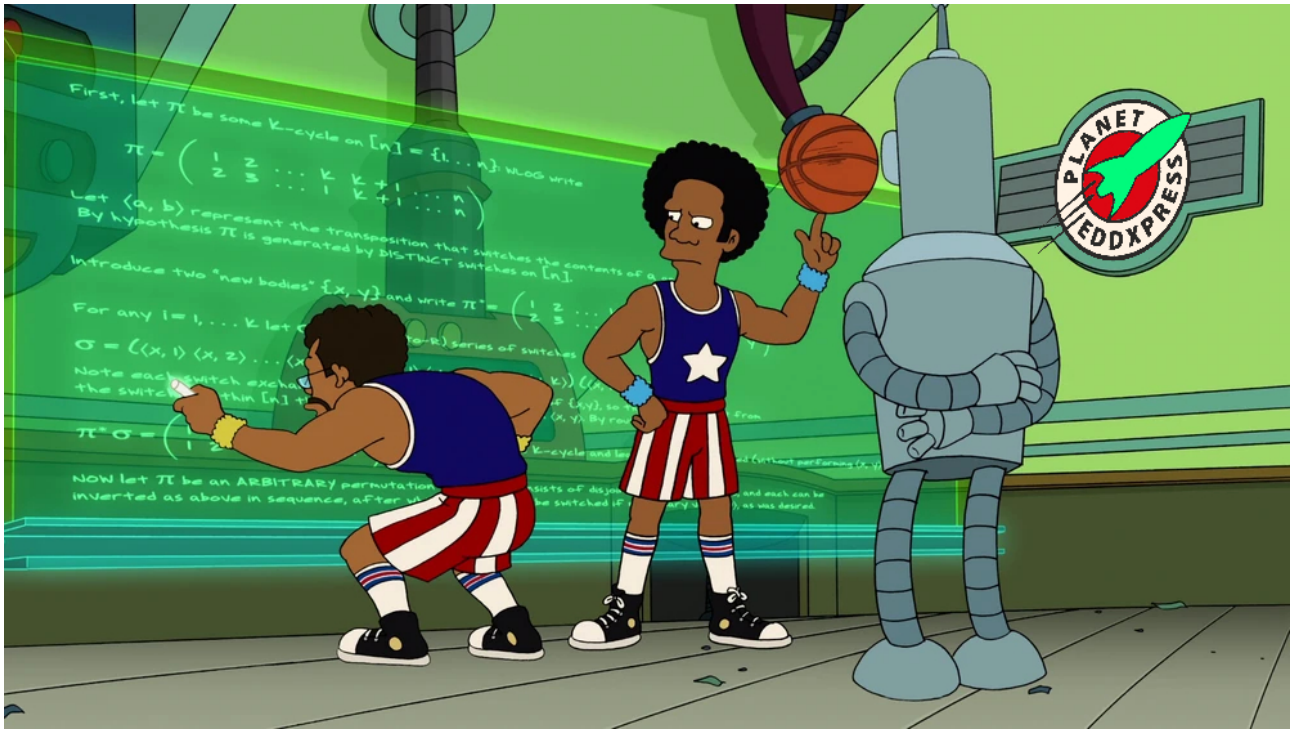


Figura 1: Profesores ayudando a Planet EDDXPRESS solucionar problemas y llegar a nuevos horizontes

El éxito de EDDXPRESS le ha permitido convertirse en el mejor ~~ramo~~ servicio de entregas de todo el universo. Luego de duros y complejos desafíos, ahora les toca solucionar nuevos y diversos desafíos, que los desafiarán en conocimientos HASH y modelamiento de problemas de restriciones. En esta ocación, te han asignado que te encargues de los problemas de identificación de secuencias de ADN de Seymour, la optimización de espacio de la Nave, y ayudar a frenar las duplicaciones de Bender identificando patrones en árboles.

Parte 1: *Rolling DNA Codes*

Figura 2: Profesor Farnsworth tratando de identificar ADN

En un intento de reencontrarse con Seymour Diera, Fry le pide al profesor que use su nueva tecnología de clonación genética para traer de vuelta a su amado perro.

Para poder realizar la clonación, el Profesor Farnsworth necesita encontrar las ubicaciones de diferentes secuencias génicas claves para la correcta clonación genética.

Problema

Se entregarán una secuencia de tamaño N y Q cadenas de tamaño M. Deberás encontrar en qué posiciones se encuentran las cadenas en la secuencia entregada. Pueden asumir que $N > M$. El alfabeto de la secuencia y consultas es `AGTC`. Se recomienda usar rolling hashing, como el algoritmo Rabin-Karp.

Ejecución

Tu programa se debe compilar con `make` y debe generar el ejecutable `find_sequences`, que se ejecuta con:

```
./find_sequences input.txt output.txt
```

Se leerá de un archivo de input y se escribirá en el archivo de output, entregados como argumentos del programa. En caso de querer correr el programa para ver los leaks de memoria utilizando deberás utilizar:

```
valgrind ./find_sequences input.txt output.txt
```

Input y Output

Siempre se entregarán al menos 4 líneas:

1. El tamaño N del documento
2. El documento de tamaño N, en una línea, con caracteres `AGTC`
3. El largo M de cada cadena a buscar
4. La cantidad Q de cadenas a buscar

Luego se entregarán Q líneas de largo M con caracteres `AGTC`, que tendrán los contenidos de la consulta.

input

```
8
ATCGGATC
3
2
ATC
GCT
```

En cada consulta, deberás imprimir `Consulta q` con q el índice de la consulta. Luego, separado en líneas distintas, las posiciones de cada cadena en el documento. Si no hay, igual se imprime `Consulta q`.

output

```
Consulta 0
0
5
Consulta 1
```

Parte 2: Rompecabezas Cósmico de Entregas

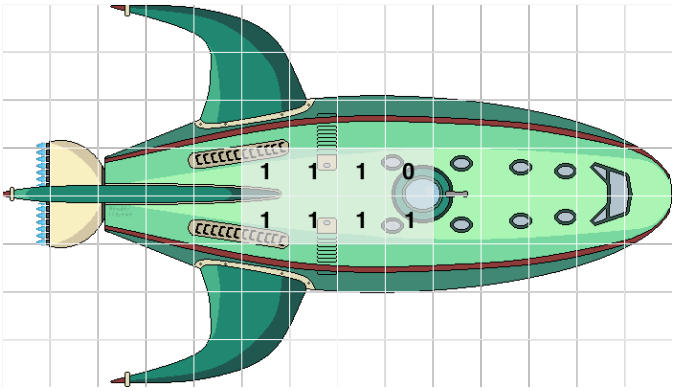


Figura 3: Ejemplo de bodega de nave.

Mientras revisas los paquetes que debes repartir por toda la galaxia, te das cuenta de que ocupan mal el espacio disponible en tu nave, la famosa Planet Express. Para poder realizar todas las entregas a tiempo y sin contratiempos, necesitas organizarlos de manera eficiente, aprovechando cada centímetro de la bodega.

Problema

Tu misión es lograr encontrar la forma de organizar los diversos paquetes en la bodega de la nave, la cual puede tener forma irregular. Para esto, representaremos la bodega como una grilla donde cada espacio es un lugar que puede ocupar parte de un paquete.

Las **bodegas** se representarán mediante una matriz rectangular compuesta por unos y ceros, donde los ceros (0) indican lugares inicialmente no disponibles, y los unos (1) los lugares disponibles. Primero se te entregará un número A, que representa el ancho de la matriz, y luego un número L, que denota el largo. A continuación, recibirás A líneas, cada una con L dígitos que serán 0 o 1, formando así la representación matricial de la bodega.

Bodega

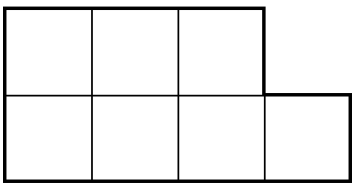
```
2
4
1 1 1 0
1 1 1 1
```

Para representar los **paquetes**, usaremos una matriz rectangular, con unos y ceros. Las posiciones con valor uno (1) indicarán partes que ocupan los paquetes, mientras que cero (0) los que no ocupa.

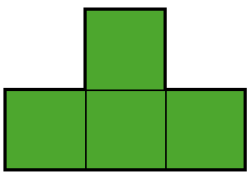
A diferencia de la bodega, la matriz que representa los paquetes siempre será siempre de 3x3. Antes de presentar cada matriz, se proporcionará un identificador, que es una letra **mayúscula o minúscula** (un solo carácter). A continuación se muestra un ejemplo de paquete con el identificador 'A':

Paquete

```
A
0 0 0
0 1 0
1 1 1
```



(a) Ejemplo de bodega



(b) Ejemplo de paquete

A partir de varios paquetes y una bodega, debes encontrar una forma de posicionar los paquetes dentro de la bodega de una forma válida, es decir:

- Solo se pueden colocar paquetes en el área establecida por la bodega.
- Ningún paquete se puede sobreponer a otro.
- Todos los paquetes fueron posicionados.

Ejecución

Tu programa se debe compilar con `make` y debe generar el ejecutable `organizer`, que se ejecuta con:

```
./organizer input.txt output.txt
```

Se leerá de un archivo de input y se escribirá en el archivo de output, entregados como argumentos del programa. En caso de querer correr el programa para ver los leaks de memoria utilizando deberás utilizar:

```
valgrind ./organizer input.txt output.txt
```

Input y Output

Se te entregarán dos enteros, que representan el número de filas y de columnas respectivamente, luego una matriz de ceros y unos donde los unos representan los lugares válidos del compartimiento. Finalmente, se te entregará un número N de paquetes, donde para cada uno se entrega un identificador (letra) seguido de una matriz 3x3 de unos y ceros, donde los unos representan la forma del paquete.

input

```
4
3
0 1 1
1 1 1
1 1 1
1 1 1
2
A
0 1 0
1 1 1
0 1 0
B
1 0 1
1 1 1
0 0 0
```

Para el output, se debe entregar la matriz de la bodega con los paquetes. Es decir, se deben reemplazar los espacios que fueron ocupados (inicialmente 1) con el identificador del paquete que se encuentra en dicha posición. A continuación se presenta un ejemplo:

output

```
0 A 1
A A A
B A B
B B B
```

Consideraciones

- Solo les entregaremos input que permitan un posible resultado de orden.
- Los paquetes tendrán un identificador único.
- Los paquetes no siempre ocuparán todo el espacio disponible de la bodega.
- Los paquetes no se pueden rotar.

Parte 3: Patrones de duplicación de Bender

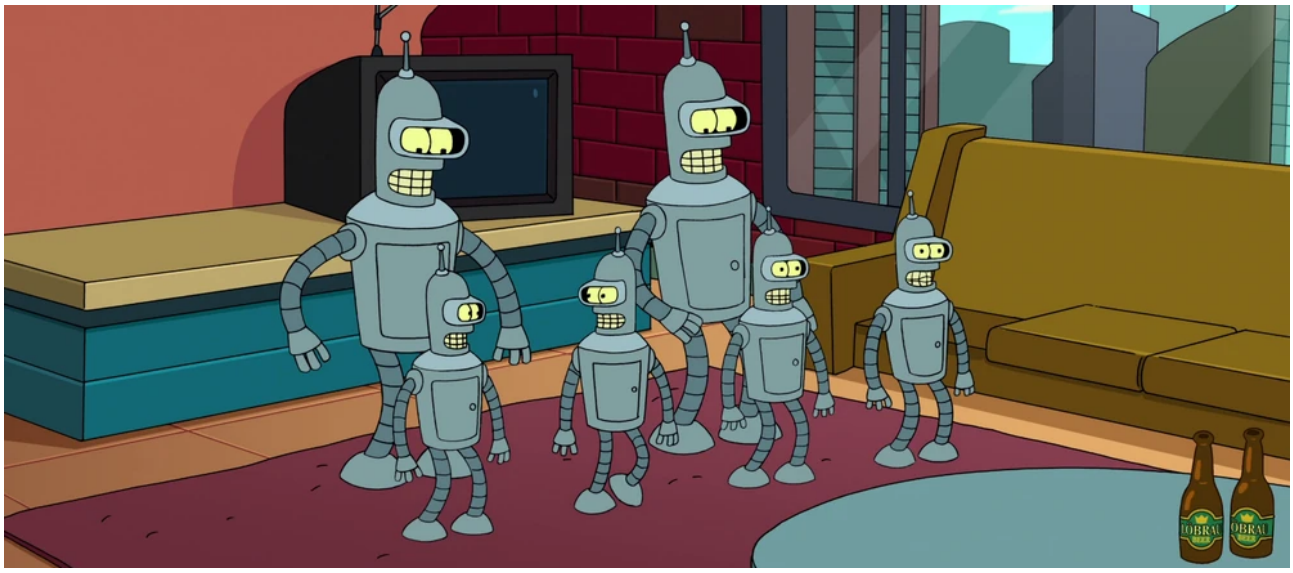


Figura 5: Subárboles de Bender

El profesor Farnsworth inventó una máquina novedosa que permite dividir y duplicar el elemento que se inserta. Bender, que solo quería doblar y olvidar, decide crear copias de él mismo para dividirse el trabajo pendiente que estaba procrastinando, asignando a sus dos copias, un tipo de trabajo a hacer.

Lamentablemente, la duplicación y la procrastinación de los robots se va fuera de control. Por lo que es tu trabajo identificar que robots generaron jerarquías claves de trabajos pendientes.

Problema

Deberás identificar donde se encuentran ciertos patrones de trabajos en el **árbol binario perfecto** de divisiones de Bender. Esto se representa como subárboles, donde cada nodo corresponde a una letra que identifica el tipo de trabajo.

Para esto recibirás de entrada un árbol con sus N letras que le corresponden. Un ejemplo de árbol, es representado por el siguiente *string* `'AABABBACBACAAAB'`, el cual tiene la siguiente representación.

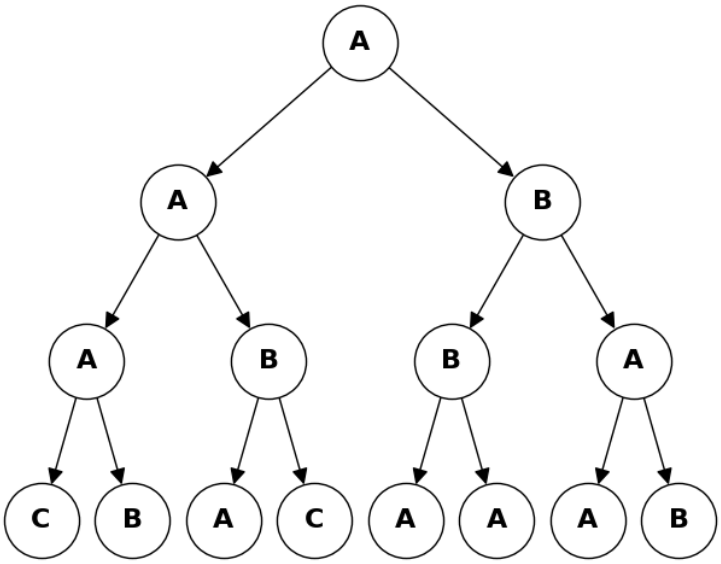


Figura 6: Imagen del árbol `AABABBACBACAAAB`.
Los valores se dan ordenados por niveles,
de izquierda a derecha (BFS)

Luego, se darán K subárboles binarios perfectos, y se deberá imprimir las posiciones en la que esta está presente en el primer árbol dado. Por ejemplo, si se entrega el árbol `AAB`, se deberá imprimir 0, 1, y 6:

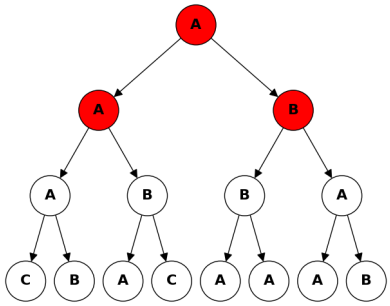


Figura 7: Match en nodo 0

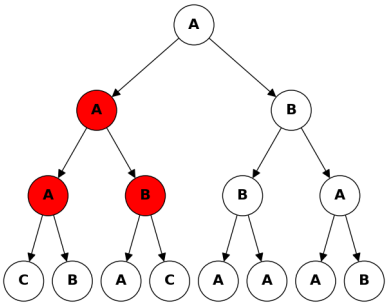


Figura 8: Match en nodo 1

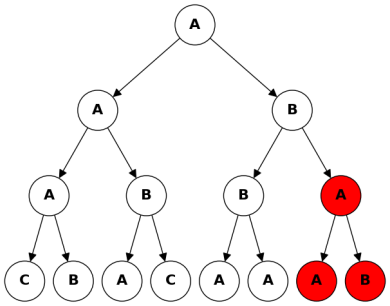


Figura 9: Match en nodo 6

Deberás emplear una estrategia para encontrar estos patrones de forma eficiente.

Ejecución

Tu programa se debe compilar con `make` y debe generar el ejecutable `patterns`, que se ejecuta con:

```
./patterns input.txt output.txt
```

Se leerá de un archivo de input y se escribirá en el archivo de output, entregados como argumentos del programa. En caso de querer correr el programa para ver los leaks de memoria utilizando deberás utilizar:

```
valgrind ./patterns input.txt output.txt
```

Input y Output

El input se estructura de la siguiente forma

- Un número N de nodos del árbol principal
- Una línea con N caracteres de la `A` a `Z`, que describe el principal, ordenando por nivel.
- Una línea indicando el número K de consultas.
- $2 * K$ líneas, donde en una se entrega el tamaño del árbol a buscar y luego los caracteres de la `A` a `Z`, del subárbol γ a buscar.

Por ejemplo, un posible input es el siguiente:

input

```
15
ABBBABAAEEDBAFE
2
3
BBA
3
ZWA
```

La primera línea indica que es un árbol de 15 nodos. Luego le sigue la descripción del árbol. Luego se indica el número de consultas. Finalmente cada una de las consultas.

El output debe consistir de K líneas, cada una con los ID de las ocurrencias del árbol γ correspondiente en el input. Estos ID se imprimen ordenados de menor a mayor. Por ejemplo, dado el input anterior, el resultado es el siguiente:

output

```
1 2 5
-1 # cuando no existe el subarbol dentro del inventario
```

Bonus

Para esta tarea, se otorgará un bonus en la **parte 1 y la parte 3**. Además de cumplir con los requisitos solicitados, se espera que se busquen y apliquen métodos para aumentar la eficiencia de los algoritmos. Esto debe reflejarse en una reducción del tiempo de ejecución y una mejor utilización de los recursos.

Los bonus se entregará según su posición de tiempo de ejecución en el ranking de las personas que participarán en este bonus. Para participar, deben completar este formulario.

Cuadro 1: Bonificaciones por Lugar

Lugar	**Décimas**
1° a 3°	4
4° y 7°	3
8° al 15°	2

Notar que cada parte tendrá su propio ranking, por lo que podrán optar hasta 8 décimas máximo.

Informe

Además del código de la tarea, se debe redactar un **breve** informe que explique como se pueden implementar algunas estructuras y algoritmos del enunciado. No es estrictamente lo que implementaste, **puedes hablar de posibles mejoras no realizadas**. Les recomendamos **comenzar la tarea escribiendo parte de 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.

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

1. Sobre la parte 1:
   - Explicación precisa de la complejidad, considerando N, M, Q. Mencionar mejoras si aplica.
   - Consideraciones importantes de memoria y propias de hashing.
2. Sobre la parte 2:
   - Estrategia empleada y sus posibles optimizaciones y extensiones.
   - Como se logra recorrer el árbol de posibles opciones. Considerar trabajo de memoria.
3. Sobre la parte 3:
   - La técnica empleada para optimizar la búsqueda. A grandes rasgos, como se ejecuta.
   - Que se realiza para evitar colisiones, y para entregar los valores ordenados.

Los puntos anteriores **NO** son las estructura que debe seguir el informe. Te recomendamos separar el informe en cada parte de la tarea, y seguir un hilo coherente que incluya los puntos anteriores. Se encuentra un template en este link para el uso en la tarea. En la documentación de Overleaft se indica como clonar un template. No se aceptarán informes de más de **3 planas** (excluyendo la página de referencias), informes ilegibles, o generados con inteligencia artificial.

Nota: Es importante destacar que este informe no necesita seguir una estructura convencional de responder pregunta por pregunta. Se brinda total libertad en el formato de presentación, pero es crucial que incluyan los puntos mencionados anteriormente para obtener el puntaje completo. La evaluación se centrará en la comprensión y la aplicación efectiva de los conceptos de estructuras de datos y búsqueda eficiente en el contexto de la tarea más que en la conformidad con un formato específico de informe.

Evaluación

La nota de tu tarea es calculada a partir de testcases de Input/Output y además de un breve informe escrito en LaTeX que explique un modelamiento sobre como abarcar el problema, las estructuras de datos a utilizar, etc. La ponderación se descompone de la siguiente forma

- 20 % Por el informe de modelamiento, que debe incluir motivación, modelamiento y conclusión.
  - 5 % Por coherencia y seguir el formato dado.
  - 15 % Por el informe dado, siguiendo lo mínimo pedido previamente.
    En esta tarea, el puntaje es **equitativamente distribuido en cada parte**.
- 70 % a la nota entre las partes 1 y 2, donde se evalúa por % de tests pasados.
  - 15 % a la nota de los tests de la parte 1
  - 30 % a la nota de los tests de la parte 2
  - 25 % a la nota de los tests de la parte 3
- 10 % Por manejo de memoria
  - 5 % Por no poseer leaks de memoria
  - 5 % Por no poseer errores de memoria

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[1]. Ademas, en esta tarea cada sección indica algún tipo de complejidad algorítmica la cual deberas respetar. De lo contrario, recibirás 0 puntos en ese test.

Recomendación de tus ayudantes

Las tareas generalmente requieren de mucha dedicación, 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, con anticipación, el enunciado detenidamente 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, modelación y elaboración de tu informe, para posteriormente poder programar de manera más eficiente. 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 política de atraso y de 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.

[1] Puedes revisarlo con `valgrind --tool=massif --massif-out-file=massif.out` y luego `ms_print massif.out`