Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2133 - Estructuras de Datos y Algoritmos

Actividad 0

10 de agosto de 2018

Figura 1: El juego tradicional de la rana.

Entrega código: domingo 19 de agosto
Entrega informe: lunes 20 de agosto

Introducción

¡Estamos calentando motores! Sólo quedan 38 días para comenzar a celebrar las fiestas patrias. Se está organizando desde ya para este 18 la fonda DCC (Donde Cantan los Cóndores), donde se diseñó un novedoso juego llamado EDD (Enuncia Donde Disparamos) creado y operado por el ingenioso señor Yadrán. El juego, basado en el clásico juego de la rana de las fondas chilenas, consiste de un cañón que dispara pelotas de colores a una matriz de tubos opacos de altura muy grande. El operador del juego tiene programadas las coordenadas y el color de las pelotas que el cañón va a disparar. A continuación se muestra una visualización del juego en una grilla de 3x3 con los tubos transparentes. Durante el juego no es posible ver

qué pelotas están adentro de los tubos, la importancia de esto será explicada en la sección de consultas.

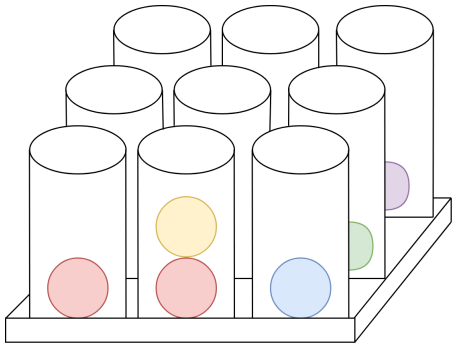

Figura 2: Ejemplo de una matriz de 3x3 con varias pelotas de distintos colores (los tubos se muestran transparentes, no opacos).

Para la figura 2 podemos pensar en que una secuencia de instrucciones, con la coordenada 0,0 correspondiendo a la fila de más atrás y la columna de más a la izquierda en la imagen puede generarse por

- push(2, 0, rojo)
- push(2, 1, rojo)
- push(2, 1, amarillo)
- push(2, 2, azul)
- push(1, 2, verde)
- push(0, 2, morado)

Pero también podría corresponder a la siguiente secuencia de acciones. Noten que el orden relativo de la pelota amarilla y roja del tubo (2,1) no cambia

- push(0, 2, morado)
- push(2, 1, rojo)
- push(1, 2, verde)
- push(2, 1, amarillo)
- push(2, 2, azul)
- push(2, 0, rojo)

Aunque en el dibujo los tubos se ven como que tienen una altura de 3, en la práctica estos son muy profundos por lo que **pueden asumir que nunca se llenaran con pelotas**.

Consultas

Durante el juego se te van a hacer ciertas consultas sobre los colores de las pelotas, los cuales tienes que responder y seguir jugando. Cada consulta consiste en lo siguiente: para un tubo en las coordenadas (R,C) y un color K debes recitar de arriba hacia abajo todos los colores de las pelotas que se encuentran por encima de la primera pelota del color K en el tubo (R,C), para luego sacar dichas pelotas, incluyendo la pelota del color K. Ejemplo:
Digamos que tenemos el siguiente caso, ilustrado en la figura 3.

Si se nos hace la consulta:

- pop(2, 0, azul)

Entonces debemos esperar como respuesta simplemente “azul“, ya que no fue necesario remover pelotas adicionales para poder llegar a la primera pelota de color azul en el tubo (2, 0). En cambio, si recibiéramos la siguiente consulta

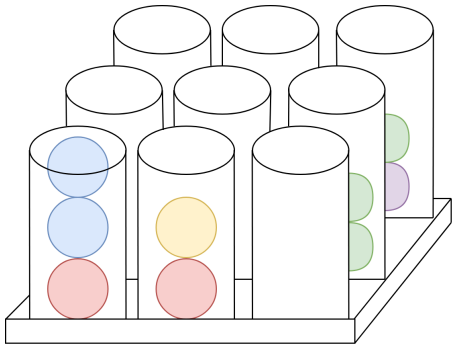

Figura 3: Otro ejemplo de una matriz de 3x3 con varias pelotas de distintos colores (los tubos se muestran transparentes, no opacos)

- pop(2, 0, rojo)

La respuesta sería "azul, azul, rojo", ya que para llegar a la primera pelota roja en el tubo (2, 0), tenemos que primero sacar las 2 pelotas azules que se encuentran por encima de ella. Ahora, cuando se hace una consulta y después de contestarla correctamente, el operador procede a sacar las pelotas nombradas del tubo correspondiente y puede hacerte otra consulta o continuar disparando su cañón con más pelotas. Por ejemplo, si partimos con nuestra matriz como se muestra en la figura 3 y se realizan las siguientes operaciones

- pop(2, 0, rojo)
- push(2, 0, amarillo)
- push(2, 0, morado)
- pop(2, 0, amarillo)

Debiésemos entregar el siguiente resultado: "azul azul rojo morado amarillo". Primero, al igual que antes, se pide sacar la pelota roja en el tubo (2, 0) por lo que nuestra respuesta parte con "azul azul rojo". Luego se añade en el mismo tubo una pelota amarilla y después una morada, por lo que la pelota morada queda arriba de la amarilla. Luego se pide sacar la pelota amarilla, por lo que primero tenemos que sacar la morada para luego sacar la amarilla, generando como respuesta "morado amarillo".

Casos borde

1. El programa del operador puede pedirte sacar una pelota de un tubo que se encuentra vacío, en ese caso debes entregar simplemente como respuesta "vacio".
2. Ya se menciono antes pero en caso de que hayan dos pelotas del mismo color en el tubo, debes detenerte cuando encuentres **la primera pelota**.

3. En caso de que el programa te pida sacar una pelota de un tubo que no se encuentre, debes ir nombrando de todas formas las pelotas en orden terminando en "vacio". Por ejemplo, si para la figura 3 se pidiera sacar la pelota de color *negro* del tubo (2, 0), debes entregar como respuesta "azul azul rojo vacio"

Como hay muchos tubos en el juego y por su opacidad no puedes ver las pelotas que están dentro, debes dar las respuestas acordándote en tu cabeza dónde están las pelotas en todo momento. Pero la lista de instrucciones parece infinita, y la matriz tiene un tamaño muy grande, por lo que no eres capaz de ganar, ya que siempre se te olvida el orden de las pelotas. Ahí es cuando, ayudado de tu conocimiento de programación, decides crear un programa eficiente que te vaya diciendo las respuestas que tienes que entregar, para un programa dado.

Para esta primera actividad, cuyo objetivo es que te familiarices con C, esperamos que implementes dos estructuras sencillas, un stack por medio de una lista ligada y una matriz. Se espera que logres manejar las estructuras usando punteros y que aprendas a manejar la memoria de tu programa de manera responsable y eficiente. Para cada una de estas estructuras vas a tener que implementar un par de operaciones sencillas las cuales vamos a detallar en la sección de operaciones, y analizar el tiempo de ejecución de tus estructuras al realizar las distintas operaciones generando un informe con tus conclusiones.

Stack

Un ***Stack*** implementado por medio de una lista ligada va a almacenar los datos en nodos, los cuales tienen referencias al elemento siguiente. Esto permite a la estructura crecer un nodo cada vez que se inserta algo. Cuando se inserta un nodo, este queda a la cabeza y pasa a ser el primer elemento. Cada vez que sacas el primer elemento, el segundo elemento queda a la cabeza de la estructura.

Matriz

La matriz a implementar no es más que un arreglo de arreglos de ***Stack*** (o arreglo de ***Stacks***, dependiendo de su implementación). La matriz debe permitir seleccionar un ***Stack*** dentro de ella, para poder realizar operaciones sobre este.

Operaciones

Tu programa deberá implementar las siguientes operaciones para el correcto funcionamiento del ***Stack***.

- stack_init() : Inicializa un ***Stack*** vacío, retornando un puntero a este mismo.

- push(stack, color) : Inserta un elemento al ***Stack*** ubicado en la posición (row, col) de la matriz del color especificado.
- pop(stack) : Retorna el elemento superior del ***Stack*** ubicado en la posición (row, col) de la matriz y lo saca del stack.
- destroy(stack) : Recibe un puntero al ***Stack***. Libera toda la memoria que fue guardada durante la ejecución.

Memoria

Para esta tarea se espera que puedan realizar un correcto manejo de memoria. Por correcto manejo de memoria, nos referimos a que sean capaces de usar ***malloc*** o ***calloc***, además de realizar los ***free()*** correspondientes, evitando generar ***leaks*** de memoria luego de la ejecución. Para poder revisar que estén liberando la memoria correctamente durante la ejecución de su programa, deben utilizar ***valgrind***. Este punto será de mucha importancia para las próximas tareas del curso, por lo que se recomienda que aprovechen esta actividad para aprender y tener claro cómo escribir código libre de errores de memoria. Si no lo has hecho, se recomienda que revises las guías de punteros y memoria del repositorio.

Input

Cada secuencia de acciones a ejecutar estará cargada en un archivo el que se les entregará en la carpeta de tests en el Github del curso. La estructura del archivo, mostrado será la siguiente:

```
3 3 12
0 2 0 0
0 2 1 0
0 2 1 1
0 2 2 2
0 1 2 3
0 0 2 4
1 2 0 0
0 2 0 1
0 2 0 4
1 2 0 1
1 2 0 5
1 0 2 5
```

Donde la primera linea tendrá tres dígitos, indicando el N, M y L, donde N es el numero de filas y M el numero de columnas y L es el numero de operaciones siguiente. Las L próximas lineas contienen 4 dígitos cada una, O, R, C y K donde O es la operación a realizar (0 para push y 1 para pop), R y C son la fila y la columna respectiva donde se va a realizar la operación y K es el color de la pelota en esa operación.

Output

Se espera que entregues tu output en la forma de un archivo llamado *output.txt* con una linea conteniendo un numero para cada vez que tengas que anunciar un color. Para el ejemplo mostrado en el input, se esperaría el siguiente output:

```
0
4
1
vacio
4
vacio
```

Informe

En esta parte de su trabajo deberán explicar **brevemente** las complejidades teóricas de las siguientes operaciones en listas ligadas:

1. insert: inserta un elemento en una posición especifica.
2. delete: elimina un elemento en una posición especifica.
3. append: agrega un elemento al final de la lista.
4. pop: elimina el elemento del final de la lista.
5. concatenate: agrega los elementos de una lista al final de otra.
6. destroy: borra la lista, sin perdida de memoria.

Debes ser claro, y no deberás usar más de 3 líneas por método.

Entrega

Cada alumno tendrá acceso a un repositorio personal donde tendrán que subir todos los archivos de su código en la carpeta *Programa*, además de subir un archivo *PDF* con nombre *Informe.pdf* dentro de la carpeta *Informe*. La fecha límite de entrega del código es el domingo 19 de agosto y del informe es el lunes 20 de agosto ambos a las 23:59. No se aceptarán entregas atrasadas.

Cómo correr tu programa

Para compilar tu programa, será necesario que lo realices con ***make***, y que el ***output*** de tu programa se guarde en un archivo llamado *resultado.txt*. Para esto, debes primero desde la Terminal usar ***make*** y luego con *./nombre_programa* se ejecutará tu programa.

Para realizar las pruebas con el manejo de memoria se debe ejecutar *valgrind ./nombre_programa* y de la misma forma para poder saber el tiempo de ejecución se usa *time ./nombre_programa*.

Evaluación

La nota final de la Actividad 0 será calculada de la siguiente forma

- Código (**70 %**)
  - Programa (**50 %**) : Se espera que tu programa ejecute bien las operaciones del operador, entregando el output correcto en el archivo solicitado.
  - Memoria (**20 %**) : Un correcto manejo de memoria es muy recomendado para el desarrollo de esta actividad, dado que les servirá para las próximas tareas. Si arroja 0 errores, es un 7, en cualquier otro caso se calificará con un 1.
- Informe (**30 %**)
  - Explicar de manera clara y concisa las complejidades de las estructuras usadas. Se entregará 1 punto por cada una de las operaciones si cumple lo pedido.