Algoritmos y Estructuras de Datos II – 3 de Diciembre de 2024

Examen Final Teórico-Práctico

Alumno: ..... Email: .....

Siempre se debe explicar la solución. Una respuesta correcta no es suficiente si no viene acompañada de una justificación lo más clara y completa posible. Los algoritmos no deben escribirse utilizando código c o de bajo nivel, sino el código de la materia y evitando la utilización innecesaria de punteros. Realizar cada ejercicio en HOJAS SEPARADAS y NOMBRADAS. La no observación de estas recomendaciones resta puntaje.

1. (TADs) Una Liquidity Pool es una especie de alcancía comunitaria donde las personas ponen dos tipos de monedas distintas, por ejemplo dólares y pesos, o dólares y euros, o cualquier par de monedas. En una de sus variantes, existe una constante que define el valor relativo entre las dos monedas: por ejemplo 1 dólar equivale a 1000 pesos. A esta constante le podemos llamar **tasa de cambio**. En todo momento dado, la Liquidity Pool (abreviaremos LP) debe contener una cantidad de monedas que respeten la tasa. Por ejemplo, considerando la tasa dólar-peso igual a 1000, si hay 20 dólares, debe también haber exactamente 20000 pesos. Una persona puede **agregar liquidez**, que consiste en agregar las dos monedas respetando su tasa de cambio. En el ejemplo anterior, se podrían agregar 5 dólares y 5000 pesos. Además de agregar liquidez, la LP permite **intercambiar** monedas. Por ejemplo, si la LP en un momento dado contiene 500 dólares y 500000 pesos, alguien puede querer cambiar 5 dólares por pesos, para lo cual agrega esa cantidad de dólares, y recibe a cambio 5000 pesos. Al finalizar la operación la LP tendrá 505 dólares y 495000 pesos. **No** podría intercambiar 600 dólares, ya que no se dispone de los 600000 pesos que se necesitan para respetar la tasa de cambio.

   (a) Se debe **especificar** el TAD **LiquidityPool** el cual tendrá un único constructor que recibe el **nombre de la moneda 1** que contendrá (en el ejemplo anterior, "dólar"), el **nombre de la moneda 2** (en el ejemplo anterior, "peso"), y un número racional indicando la **tasa de cambio**.

   Sus operaciones serán:

   - **Agregar liquidez**: que deberá recibir un racional indicando la cantidad de la moneda 1 que se está agregando, y otro racional indicando la cantidad de la moneda 2 que se está agregando. Esta operación **modifica** la LP (no devuelve nada). Solo podrá ejecutarse si las cantidades que se están agregando **respetan la tasa de cambio** (o sea hay pre-condición).
   - **Consultar nombre de la moneda 1**: que dada una LP, devuelve un string indicando el nombre de la moneda 1.
   - **Consultar nombre de la moneda 2**: que dada una LP, devuelve un string indicando el nombre de la moneda 2.
   - **Consultar cuántas monedas 1** hay en la LP: que dada una LP, devuelve un número racional indicando cuántas monedas 1 contiene en ese momento.
   - **Consultar cuántas monedas 2** hay en la LP: que dada una LP, devuelve un número racional indicando cuántas monedas 2 contiene en ese momento.
   - **Consultar la tasa de cambio** que hay en la LP: que dada una LP, devuelve un número racional indicando la tasa de cambio entre las monedas.
   - **Intercambiar moneda 1 por moneda 2**: que deberá recibir un número racional indicando la cantidad de la moneda 1 que se quiere agregar a la LP, y devolverá un número racional indicando la cantidad de de la moneda 2 que se está obteniendo. Notar que esta operación debe ser un procedimiento que también devuelve un valor (o sea, hay un argumento que es **solamente out**).
   - **Intercambiar moneda 2 por moneda 1**: Análoga a la anterior, solo que se agrega moneda 2, y se recibe moneda 1.

   (b) Se debe **implementar** el TAD **LiquidityPool** utilizando una tupla con 2 strings que indican el nombre de la moneda 1 y la moneda 2 respectivamente, y 3 floats que indican cuántas monedas hay de cada una, y la tasa de cambio. **NO** se deben utilizar punteros.

   (c) Utilizando el tipo **abstracto LiquidityPool**, implementar una operación que dadas dos LP, y un número racional, permite intercambiar una cantidad de la moneda 1 de la primera LP por una cantidad de la moneda 2 de la segunda LP. Esta operación tendrá como pre-condición que el nombre de la moneda 2 de la primera LP coincida con el nombre de la moneda 1 de la segunda LP, además de la precondición que asegure que hay liquidez para realizar el intercambio.

   Por ejemplo, podemos tener una LP con moneda 1 "euro", moneda 2 "dólar", tasa de cambio 1, 2, conteniendo 10 euros y 12 dólares. Y una LP con moneda 1 "dólar", moneda 2 "peso", tasa de cambio 1000, conteniendo 100 dólares y 100000 pesos. Podemos llamar a la operación para cambiar 5 euros y obtener pesos.

   Esta operación también será un procedimiento que debe modificar las dos LP, y a la vez devolverá un racional correspondiente a la cantidad de monedas 2 de la segunda LP. El prototipo será entonces:

   proc *dobleIntercambio*(**in/out** $lp_1, lp_2 : LiquidityPool$, **in** $q_1 : float$, **out** $q_2 : float$)

(Backtracking) Ya pasaste los 35 años y al fin te ponés las pilas para hacer ejercicio físico. En el gimnasio tenés un plan con n ejercicios donde cada ejercicio i, con $1 \leq i \leq n$ tiene asociado un "valor de entrenamiento general" e_i. Además, cada ejercicio i requiere un esfuerzo de brazos b_i, un esfuerzo de zona media z_i y un esfuerzo de piernas p_i.

Se debe encontrar el máximo valor de entrenamiento general obtenible al elegir ejercicios sin que el esfuerzo total en brazos supere el monto B, el esfuerzo total en zona media supere el monto Z y el esfuerzo total en piernas supere el monto P.

Resolvé el problema utilizando la técnica de backtracking dando una función recursiva. Para ello:

- Especificá precisamente qué calcula la función recursiva que resolverá el problema, indicando qué argumentos toma y la utilidad de cada uno.
- Da la llamada o la expresión principal que resuelve el problema.
- Definí la función en notación matemática.

Para cada uno de los siguientes algoritmos determinar **por separado** cada uno de los siguientes incisos.

(a) ¿Qué hace? ¿Cuáles son las precondiciones necesarias para haga eso?

(b) Si le aplicamos el procedimiento a la lista que tiene los elementos $[1, 2, 3, 4, 5]$ ¿cómo es la lista resultante?

(c) ¿Cómo lo hace?

(d) El orden del algoritmo, analizando los distintos casos posibles.

```
proc p (in/out l: list of T)
    var a, b: pointer to (node of T)
    a := l
    if a ≠ null then
      if a→next ≠ null then
        b := a→next
        while b→next ≠ null do
            a := a→next
            b := b→next
        od
        b→next := l
        a→next := null
        l := b
      fi
    fi
end proc
```

donde los tipos node y list se definen como sigue

```
type node of T = tuple
                   value: T
                   next: pointer to (node of T)
                 end
type list of T = pointer to (node of T)
```

(Para alumnos libres) Dar la forma general de los algoritmos divide y vencerás, identificar sus características, explicarlas y mencionar ejemplos de uso conocido de esa técnica.