Algoritmos y Estructuras de Datos – Recuperatorio Primer Parcial

Licenciatura en Tecnología Digital, UTDT

Primer semestre 2023

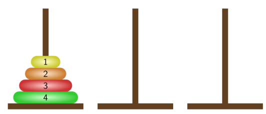
- No está permitido comunicarse por ningún medio con otros estudiantes ni con otras personas durante el examen, excepto con los docentes de la materia.
- Puede consultarse a los docentes solo por aclaraciones específicas del enunciado.
- El examen es a libro abierto: está permitido tener todo el material **impreso** y apuntes personales que deseen traer. **No está permitido el uso de dispositivos electrónicos para este fin.**
- **Cada ejercicio debe resolverse en hoja aparte.**

Por favor, no escribir en este espacio.

Problema:	1	2	3	4	Total (sobre 100)
Nota:					

Problema 1. (30 puntos) :

En el juego de las **Torres de Hanoi** se tienen $N > 2$ discos perforados en el centro y $K > 2$ palos. Cada disco tiene diámetro desde 1 hasta N sin repetir. El juego comienza con todos los discos apilados de mayor (abajo) a menor (arriba) en palo 1, y el objetivo es mover todos los discos hasta el palo K, con la restricción de que no se puede apoyar un disco más grande encima de uno más chico. Cada movida consiste en sacar el disco tope de un palo y moverlo al tope de otro palo.



Torres de Hanoi, $N = 4, K = 3$

Representaremos un juego de *Torres de Hanoi* como un `vector<vector<int>>` v, de longitud K, donde `v[0]` representa al primer palo y `v[K-1]` representa al último palo, y donde los discos presentes en cada palo están almacenados en orden de disco más abajo a disco más arriba como valores `int` que representan a cada disco por su diámetro.

Algunos ejemplos de instancias válidas e inválidas del juego:

(1) $N = 4$, $K = 4$; inicial: `{{4,3,2,1}, {}, {}, {}}` y final: `{{},{},{},{4,3,2,1}}` (con los 4 discos apilados en orden de diametro en los palos 1 y 4 respectivamente)

(2) instancia intermedia: `{{4},{3,1},{},{2}}` (disco 4 en el 1er palo, disco 1 encima del disco 3 en el segundo palo, 3er palo vacío, disco 2 en el 4to palo)

(3) inválida: `{{2,4},{3}}` (problemas: $K < 3$; falta disco 1; disco 4 encima del disco 2).

Se pide especificar lo siguiente:

(a) El predicado $esHanoiValido(v :$ `vector<vector<int>>`$)$ que será $Verdadero$ si y solo si v representa a una partida en un estado válido cualquiera. Será $Verdadero$ para los ejemplos (1) y (2) y falso para (3).

(b) El predicado $esMovimientoValido(v :$ `vector<vector<int>>`$, i :$ `int`$, j :$ `int`$)$ que será verdadero si es legal mover el disco superior del palo i al tope del palo j. Para el ejemplo (2), $esMovimientoValido(v, 1, 3) = Verdadero$, o sea, es legal mover el disco tope del palo 2 (diametro 1) al palo 4 (diametro 2).

(c) El problema `mover_hanoi`, que dado un juego válido y dos palos i, j entre los cuales se puede hacer un movimiento válido, modifica v ejecutando el movimiento pedido.

Nota: Notar que ni N ni K aparecen explícitamente como parámetros de ninguno de los predicados o problemas a especificar.

Problema 2. (20 puntos)

La función `contarDistintos` *toma un vector de strings no vacios* `v` *y devuelve* `int`*. Para cada string, cuenta si su última letra es distinta a la primera. Por ejemplo, para* `vec = {"sad","ara","pip","a*","amlo"}` *se debe devolver* `3` *porque la cantidad de strings que comienzan y terminan con letras distintas es 3.*

`int contarDistintos(const vector<string> & vec)`

Pre: $|vec| \mod 2 = 0 \land (\forall j : \texttt{int})(0 \leq j < |vec| \implies |vec[j]| > 0)$

Post: $res = (\sum_{j=0}^{|vec|-1} \beta(vec[j][0] \neq vec[j][|vec[j]| - 1]))$

```
int i = v.size();
int suma = 0;
while (1 < i){
  i = i - 2;
  if (vec[i][0] == vec[i][vec.size()-1])
    suma = suma + 1;
  if (vec[i+1][0] == vec[i+1][vec.size()-1])
    suma = suma + 1;
}
int res = vec.size() - suma;
```

(a) Dar una precondición P_c y una postcondición Q_c adecuadas para el ciclo.

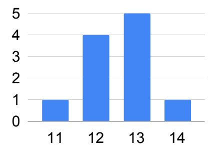
(b) Dar un invariante del ciclo $\mathcal{I}$ que permita demostrar la corrección del programa. Explicar en palabras por qué el invariante es adecuado.

(c) Dar un ejemplo de estado del programa que cumpla con $\mathcal{I}$ y un ejemplo que no lo cumpla.

(d) Demostrar que $\mathcal{I} \land B = \texttt{false} \implies Q_c$, donde B es la guarda del ciclo.

Problema 3. (30 puntos)

El *histograma* de una serie de valores enteros es un gráfico de barras verticales donde cada barra tiene como longitud la cantidad de apariciones de cada valor del rango [a, b]. Representaremos un histograma como un `vector<int>` con la longitud de cada barra en el intervalo [a,b], que incluye a y b, con $a < b$. Por ejemplo, el histograma `{1,4,5,1}` en el rango [11,14] representa a una serie de 11 valores entre 11 y 14, con 1 aparición del valor 11, 4 apariciones del valor 12, 5 apariciones del valor 13, y 1 aparición del valor 14, como muestra la figura adjunta.



Dados dos histogramas h_1 de rango [a,b] y h_2 de rango [c,d], **implementar la función**

```
vector<int> combinar(vector<int> h1, vector<int> h2, int a, int b, int c, int d)
```

que calcula el resultado de combinar ambos histogramas, sabiendo que el rango [c,d] está contenido en el rango [a,b], es decir, que vale que $a \leq c < d \leq b$. Notar que como el rango del histograma h_1 es igual o mayor que el rango de h_2, se cumple que $|h_2| \leq |h_1|$.

Ejemplo: Dado el histograma h_1 = `{0,0,1,1,2,3,0,0}` en el rango [-5,2] y el histograma h_2 = `{4,3,2,1}` en el rango [-4,-1], el resultado de `combinar(h1,h2,-5,2,-4,-1)` será el histograma `{0,4,4,3,3,3,0,0}` (también en rango [-5,2]).

(a) Escribir la función `combinar` con un único ciclo, sabiendo que $a \leq c < d \leq b$, $|h_1| = b - a + 1$ y $|h_2| = d - c + 1$.

(b) Dar la especificación completa del ciclo propuesto $(\mathcal{P}_c, \mathcal{Q}_c, \mathcal{I}, fv)$. Justificar por qué es adecuada para el código escrito.

Problema 4. (20 puntos) Se quiere implementar la función

```
int sumar_picos(const vector<int> & v)
```

que debe calcular la sumatoria de los valores de los *picos* que contiene un vector. Llamamos *pico* a cualquier posición que es estrictamente mayor que sus dos posiciones vecinas. Los extremos de un vector nunca son picos. Por ejemplo, el vector `v = {0,1,0,0,0,10,1,9,8}` tiene tres picos en las posiciones 1, 5 y 7; y `sumar_picos(v) = 1 + 10 + 9 = 20`.

(a) Implementar `sumar_picos` con recursión simple utilizando un sólo llamado recursivo. Se debe recorrer el vector en orden inverso.

(b) Implementar `sumar_picos` con la técnica Divide & Conquer, partiendo el problema en 2 y utilizando 2 llamados recursivos.

En cada caso, si necesita cambiar los parámetros de la función debe hacerlo en una función auxiliar y **mostrar cómo le llama a esa función auxiliar desde la función `sumar_picos` original.**