Algoritmos y Estructuras de Datos – Recuperatorio M1/M2

Licenciatura en Tecnologías Digitales, UTDT

Segundo 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.

	M1P1	M1P2	M2P1	M2P2	Total M1	Total M2
Problema:						
Nota:						

M1: Especificación de problemas — Corrección

Problema 1. (50 puntos) Especificación de problemas
Una importante cadena de salas de cine gestiona reservas de butacas consecutivas en sus salas. Cada pedido consiste en una cantidad de butacas requeridas, y la empresa desea asignar cada pedido a un bloque de butacas libres de manera que se reduzca el desperdicio máximo de butacas. Es decir, dado un `vector<int>` P con n pedidos de butacas, y un `vector<pair<int,int>>` R de n rangos de butacas libres **que no deben solaparse ni ser consecutivos**, se desea asignar cada reserva i a un bloque j de manera que - el bloque j tenga suficientes butacas para la reserva i - el sobrante máximo de butacas de cada bloque (calculado como `R[j].second-R[j].first+1-P[i]`) sea el mínimo posible. La asignación resultante se debe expresar como un vector de enteros v de n enteros donde `v[i]` es el índice del rango de butacas libres asignado a la reserva i.
Ejemplo:
Para los pedidos `P = {4,5,3,3}` y los rangos de butacas `{{1,5},{6,11},{13,15},{17,19}}`:

- en total hay 24 asignaciones posibles (las permutaciones de `{0,1,2,3}`);
- la asignación `{0,2,1,3}` no es válida pues la segunda reserva (5) es mas grande que la cantidad de butacas del tercer rango (3).
- la asignación `{1,0,2,3}` es válida con un sobrante máximo de 2 butacas;
- la asignación `{0,1,2,3}` es válida con un sobrante máximo de 1 butaca;
- la asignación `{0,1,3,2}` es válida con un sobrante máximo de 1 butaca.

Para este ejemplo, se desea obtener como resultado alguna de las asignaciones (válidas) que tenga el sobrante máximo de 1 butaca, por ejemplo, `0,1,3,2`.

Especificar el problema `obtener_mejor_asignacion`.

Problema 2. (50 puntos) Corrección de ciclos

La función `buscar_butacas` *obtiene, dado un vector de rangos no consecutivos y ordenados crecientemente de butacas libres, el vector correspondiente de rangos de butacas ocupados, sabiendo que el primer y el último rango de butacas siempre son libres. Por ejemplo, para el vector de entrada* `{1,3},{5,6},{9,10}`*, se debería devolver los rangos* `{4,4},{7,8}`.

`vector<pair<int,int>> buscar_butacas(vector<pair<int,int>> R)`
Pre: $|R| > 1 \land R[0].first = 1 \land (\forall j : \texttt{int})\ 0 < j < |R| \implies 0 < R[j].first \leq R[j].second \land (j < |R| - 1 \implies R[j].second < R[j+1].first - 1)$
Post: $|res| = |R| - 1 \land (\forall j : \texttt{int})\ 0 \leq j < |res| \implies res[i].first = R[i].second + 1 \land res[i].second = R[i+1].first - 1$

Se tiene la siguiente implementación de `buscar_butacas`:

```
1  int i = R.size()-1;
2  // res de tamaño |R|-1 lleno de pares {0,0}
3  vector<pair<int,int>> res = vector<pair<int,int>>(i, {0,0});
4  while(i > 0) {
5    i = i-1;
6    res[i].first = R[i].second + 1;
7    res[i].second = R[i+1].first - 1;
8  }
```

(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 valores de `count`, `v`, y `i` que cumpla con $\mathcal{I}$ y un ejemplo que no lo cumpla.

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

M2: Escritura y especificación de ciclos — Recursión

Problema 1. (50 puntos) Escritura y especificación de ciclos
Dados dos vectores de longitud n, uno P de tipo `vector<int>` que representa una secuencia de pedidos de butacas, y otro B de tipo `vector<pair<int,int>>` de rangos de butacas libres, se desea calcular los rangos de butacas asignados para cada una de las n reservas, devolviéndolos como un `vector<pair<int,int>>`. Si el pedido $P[i]$ se puede satisfacer por el rango $B[i]$, se genera la reserva que comprende las butacas desde la primera butaca de B[i] hasta la butaca $B[i] + P[i] - 1$. Si el pedido $P[i]$ no se puede satisfacer por el rango $B[i]$, se genera la reserva `{0,0}`.

Ejemplo: Para los pedidos `P = {1,2,2,3,1}` y los rangos `B = {{1,2},{4,6},{8,11},{15,16},{17,19}}`, se espera obtener la reserva `{{1,1},{4,5},{8,9},{0,0},{17,17}}`.

(a) Escribir una función con un único ciclo que, dado un `vector<pair<inr,int>>` B y un `vector<int>` P, ambos de la misma longitud n, devuelva un `vector<pair<int,int>>` con las n asignaciones de butacas correspondientes. Ambos vectores P y B deben recorrerse una única vez.

Sugerencia: si se quiere crear un `vector<T>` *de tamaño N y con un valor de relleno V, se puede escribir la línea:*

```
vector<T> v = vector<T>(N,V);
```

Por ejemplo, `vector<int> v = vector<int>(5,0);` crea el vector `v = {0,0,0,0,0}`.

(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 2. (50 puntos) Recursión
Implementar la función

```
vector<pair<int,int>> rangos_libres(const vector<bool> & butacas)
```

que, dado un vector de `bool`, donde la posición i indica si la butaca $i + 1$ está o no ocupada, calcula el vector de rangos de butacas libres. Por ejemplo, si `butacas = {true,false,false,true,true,false,false}`, debe devolver `res = {{2,3},{6,7}}`.

(a) Implementar `rangos_libres` con recursión simple, es decir utilizando un sólo llamado recursivo.

(b) Implementar `rangos_libres` 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 se llama a esa función auxiliar desde la función `rangos_libres` original.**.

Sugerencia: se puede usar el método `insert` *de vector para agregar todos los elementos de otro vector. Ejemplo: para concatenar* `v1` *y* `v2`*, se puede escribir*

```
v1.insert(v1.end(), v2.begin(), v2.end());
```

para agregar todos los elementos de `v2` *al final de* `v1`.