Algoritmos y Estructuras de Datos II – 7 de Julio de 2021

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. La no observación de estas recomendaciones resta puntaje.

1. (Backtracking) No es posible correr una carrera de 800 vueltas sin reemplazar cada tanto las cubiertas (las ruedas) del auto. Como los mecánicos trabajan en equipo, cuando se cambian las cubiertas se reemplazan simultáneamente las cuatro. Reemplazar el set de cuatro cubiertas insume un tiempo T fijo, totalmente independiente de cuál sea la calidad de las cubiertas involucradas. Hay diferentes sets de cubiertas: algunas permiten mayor velocidad que otras, y algunas tienen mayor vida útil que otras, es decir, permiten realizar un mayor número de vueltas. Sabiendo que se cuenta con n sets de cubiertas, que $t_1, t_2, \ldots, t_n$ son los **tiempos por vuelta** que pueden obtenerse con cada uno de ellos, y que $v_1, v_2, \ldots, v_n$ es la vida útil medida en **cantidad de vueltas** de cada uno de ellos, se pide encontrar el tiempo de carrera mínimo cuando la misma consta de m vueltas.

   (a) (Backtracking) 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.

   (b) (Programación dinámica) Implementá un algoritmo que utilice Programación Dinámica para resolver el problema.
   - ¿Qué dimensiones tiene la tabla que el algoritmo debe llenar?
   - ¿En qué orden se llena la misma?
   - ¿Se podría llenar de otra forma? En caso afirmativo indique cuál.

2. (Algoritmos voraces) Llega el final del cuatrimestre y tenés la posibilidad de rendir algunas de las materias $1, ..., n$. Para cada materia i conocés la fecha de examen f_i, y la cantidad de días inmediatamente previos d_i que necesitás estudiarla de manera exclusiva (o sea, no podés estudiar dos materias al mismo tiempo). Dar un algoritmo voraz que obtenga la mayor cantidad de materias que podés rendir.

   Se pide lo siguiente:

   (a) Indicar de manera simple y concreta, cuál es el criterio de selección voraz para construir la solución?

   (b) Indicar qué estructuras de datos utilizarás para resolver el problema.

   (c) Explicar en palabras cómo resolverá el problema el algoritmo.

   (d) Implementar el algoritmo en el lenguaje de la materia de manera precisa.

3. (Comprensión de algoritmos) Para cada uno de los siguientes algoritmos determinar **por separado** cada uno de los siguientes incisos.

   (a) ¿Qué hace?

   (b) ¿Cómo lo hace?

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

   (d) Proponer nombres más adecuados para los identificadores (de variables y procedimientos).

```
proc p(a: array[1..n] of nat)
    var d: nat
    for i:= 1 to n do
        d:= i
        for j:= i+1 to n do
            if a[j] < a[d] then d:= j fi
        od
        swap(a, i, d)
    od
end proc
```

```
fun f(a: array[1..n] of nat) ret b : array[1..n] of nat
    var d: nat
    for i:= 1 to n do b[i] := i od
    for i:= 1 to n do
        d:= i
        for j:= i+1 to n do
            if a[b[j]] < a[b[d]] then d:= j fi
        od
        swap(b, i, d)
    od
end fun
```

4. (TADs) Considerá la siguiente especificación del tipo Conjunto de elementos de algún tipo T.

```
spec Set of T where

constructors
    fun empty_set() ret s : Set of T
    {- crea un conjunto vacío. -}

    proc add (in e : T, in/out s : Set of T)
    {- agrega el elemento e al conjunto s. -}

destroy
    proc destroy (in/out s : Set of T)
    {- Libera memoria en caso que sea necesario. -}

operations
    fun is_empty_set(s : Set of T) ret b : bool
    {- Devuelve True si s es vacío. -}

    fun cardinal(s : Set of T) ret n : nat
    {- Devuelve la cantidad de elementos que tiene s -}

    fun member(e : T, s : Set of T) ret b : bool
    {- Devuelve True si el elemento e pertenece al conjunto s -}

    proc inters (in/out s : Set of T,in s0 : Set of T)
    {- Elimina de s todos los elementos que NO pertenecen a s0 -}

    {- PRE: not is_empty_set(s) -}
    fun get(s : Set of T) ret e : T
    {- Devuelve algún elemento (cualquiera) de s -}

    proc elim (in/out s : Set of T,in e : T)
    {- Elimina el elemento e del conjunto s en caso que esté. -}

    proc union (in/out s : Set of T,in s0 : Set of T)
    {- Agrega a s todos los elementos de s0 -}

    proc diff (in/out s : Set of T,in s0 : Set of T)
    {- Elimina de s todos los elementos de s0 -}
```

fun copy_set(s1 : Set **of** T) **ret** s2 : Set **of** T
{- *Crea un nuevo conjunto* s2 *con todos los elementos de* s1 -}

(a) Implementá los constructores del TAD Conjunto de elementos de tipo T, y las operaciones member, elim e inters, utilizando la siguiente representación:

implement Set **of** T **where**

type Set **of** T = **tuple**
elems : array[0..N-1] **of** T
size : nat
end tuple

¿Existe alguna limitación con esta representación de conjuntos? En caso afirmativo indicá si algunas de las operaciones o constructores tendrán alguna precondición adicional.
NOTA: Si necesitás alguna operación extra para implementar lo que se pide, debes implementarla también.

(b) Utilizando el tipo **abstracto** Conjunto de elementos de tipo T, implementá una función que reciba un conjunto de enteros s, un número entero i, y obtenga el entero perteneciente a s que está *más cerca* de i, es decir, un $j \in s$ tal que para todo $k \in s$, $|j-i| <= |k-i|$. Por ejemplo si el conjunto es $1, 5, 9$, y el entero 7, el resultado puede ser 5 o 9.

5. (Para alumnos libres) Sea T un árbol (no necesariamente binario) y supongamos que deseamos encontrar la hoja que se encuentra más cerca de la raíz. ¿Cuáles son las distintas maneras de recorrer T? ¿Cuál de ellas elegirías para encontrar esa hoja y por qué?