Algoritmos y Estructuras de Datos II – 8 de Marzo 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. (Voraz) Malena te pide que le cuides el departamento por N días y te deja la heladera llena con N productos. Cada producto i tiene fecha de vencimiento v_i contada desde el día en que llegás a la casa. Puede haber productos ya vencidos ($v_i \leq 0$). Como no tenés un mango, te vas a alimentar comiendo un producto por día, y no te vas a hacer drama por comer algo vencido. Sin embargo te gustaría que lo que comas lleve la menor cantidad posible de días vencido. Se pide indicar para cada día j con $1 \leq j \leq N$ qué producto vas a comer, minimizando la cantidad de días que llevan vencidos los productos vencidos que comés. Para ello:

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

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

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

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

2. (Backtracking) Luego de que te dan el alta por intoxicación, Malena te pide de nuevo que le cuides el departamento. Esta vez te deja N productos que no vencen pero los tenés que pagar. Cada producto i tiene un precio p_i y un valor nutricional s_i. Tu presupuesto es M. Se pide comer productos para obtener el máximo valor nutricional sin superar el presupuesto M. No hace falta comer todos los días ni vaciar la heladera.

   (a) Especificá precisamente qué calcula la función recursiva que resolverá el problema, indicando qué argumentos toma y la utilidad de cada uno.

   (b) Da la llamada o la expresión principal que resuelve el problema.

   (c) Definí la función en notación matemática.

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 q(in/out a : array[1..N] of int, in x : nat)
    for j:= 1 to x do
        m:= j
        for k:= j+1 to x do
            if a[k] < a[m] then m:= k fi
        od
        swap(a,j,m)
    od
end proc
```

```
proc r(in/out a : array[1..N] of int, in y : nat)
    for j:= y to n do
        m:= j
        while m > y && a[m] < a[m-1] do
            swap(a,m,m-1)
            m:= m-1
        od
    od
end proc
```

```
proc p(in/out a : array[1..N] of int, in i : nat)
    q(a, i-1)
    r(a, i+1)
end proc
```

4. (TADs) Considere 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) Escribí un algoritmo que utilice Programación Dinámica para resolver el ejercicio del punto 2.

   (a) ¿Qué dimensiones tiene la tabla que el algoritmo debe llenar?

   (b) ¿En qué orden se llena la misma?

   (c) ¿Se podría llenar de otra forma? En caso afirmativo indique cuál.