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. La municipalidad de tu pueblo te contrata como experto en problemas de optimización para que diseñes una solución al problema de la combinación de la segunda dosis de la vacuna del covid. Hay n personas, donde cada persona $i \in \{1..n\}$ se ha dado la primera dosis con la vacuna $v_i \in \{AZ, SV, PF\}$, y se han recibido d_k nuevas dosis de cada vacuna para administrar como segunda dosis a las n personas, con $k \in \{AZ, SV, PF\}$.

   Se conoce el porcentaje de inmunidad p_{ij} con $i, j \in \{AZ, SV, PF\}$ que se adquiere al combinar la primera dosis de la vacuna i con la segunda de la vacuna j.

   Se pide maximizar la suma de los porcentajes de inmunidad de las n personas, eligiendo para cada persona i qué vacuna se le aplicará en la segunda dosis, de manera tal que las n personas sean vacunadas.

   (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) Te vas n días de vacaciones al medio de la montaña, lejos de toda civilización. Llevás con vos lo imprescindible: una carpa, ropa, una linterna, un buen libro y comida preparada para m raciones diarias, con $m > n$. Cada ración i tiene una fecha de vencimiento v_i, contada en días desde el momento en que llegás a la montaña. Por ejemplo, una vianda con fecha de vencimiento 4, significa que se puede comer hasta el día número 4 de vacaciones inclusive. Luego ya está fuera de estado y no puede comerse.

   Tenés que encontrar la mejor manera de organizar las viandas diarias, de manera que la cantidad que se vencen sin ser comidas sea mínima. Deberás indicar para cada día j, $1 \leq j \leq n$, qué vianda es la que comerás, asegurando que nunca comas algo vencido.

   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).

```
fun s(p: array[1..n] of nat, v,w: nat) ret y: nat
    y:= v
    for i := v+1 to w do
        if p[i] < p[y] then y:= i fi
    od
end fun

fun t(p: array[1..n] of nat, v,w: nat) ret y: nat
    y:= v
    for i := v+1 to w do
        if p[y] < p[i] then y:= i fi
    od
end fun
```

```
proc r(p: array[1..n] of nat)
    for i := 1 to n div 2 do
        swap(p, i, s(p, i, n-i+1));
        swap(p, n-i+1, t(p, i+1, n-i+1));
    od
end fun
```

4. Considerá la siguiente especificación del tipo Listas de algún tipo T.

```
spec List of T where

constructors
    fun empty() ret l : List of T
    {- crea una lista vacía. -}

    proc addl (in e : T, in/out l : List of T)
    {- agrega el elemento e al comienzo de la lista l. -}

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

operations
    fun is_empty(l : List of T) ret b : bool
    {- Devuelve True si l es vacía. -}

    fun head(l : List of T) ret e : T
    {- Devuelve el primer elemento de la lista l -}
    {- PRE: not is_empty(l) -}

    proc tail(in/out l : List of T)
    {- Elimina el primer elemento de la lista l -}
    {- PRE: not is_empty(l) -}

    proc addr (in/out l : List of T,in e : T)
    {- agrega el elemento e al final de la lista l. -}

    fun length(l : List of T) ret n : nat
    {- Devuelve la cantidad de elementos de la lista l -}

    proc concat(in/out l : List of T,in l0 : List of T)
    {- Agrega al final de l todos los elementos de l0
      en el mismo orden.-}

    fun index(l : List of T,n : nat) ret e : T
    {- Devuelve el n-ésimo elemento de la lista l -}
    {- PRE: length(l) > n -}

    proc take(in/out l : List of T,in n : nat)
    {- Deja en l sólo los primeros n
      elementos, eliminando el resto -}

    proc drop(in/out l : List of T,in n : nat)
    {- Elimina los primeros n elementos de l -}

    fun copy_list(l1 : List of T) ret l2 : List of T
    {- Copia todos los elementos de l1 en la nueva lista l2 -}
```

   (a) A partir de la siguiente implementación de listas mediante punteros, implementá los constructores y las operaciones addr, take, y length.

```
implement List of T where

type Node of T = tuple
```

```
                elem : T
                next : pointer to (Node of T)
            end tuple

type List of T = pointer to (Node of T)

fun empty() ret l : List of T
    l := null
end fun


fun empty ()ret l : List of T
    l := NULL
end fun



proc addl (in e : T, in/out l : List of T)
  var p : pointer to (Node of T)
  alloc(p)
  p->elem := e
  p->next := l
  l := p
end proc
```

(b) Implementá una función que reciba una lista de enteros y encuentre la posición donde se encuentra el máximo. Dicha función debe usar el tipo **abstracto** lista, sin importar cuál es su implementación.

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?