Algoritmos y Estructuras de Datos II – 7 de Febrero de 2022

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. (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.

2. (Backtracking)

   Se tiene un tablero de 9x9 con números enteros en las casillas. Un jugador se coloca en una casilla a elección de la primera fila y se mueve avanzando en las filas y moviéndose a una columna adyacente o quedándose en la misma columna. En cada movimiento, el jugador suma los puntos correspondientes al número de la casilla, pero nunca puede pisar una casilla de manera tal que el puntaje acumulado, contando esa casilla, dé un valor negativo. El juego termina cuando se llega a la novena fila, y el puntaje total es la suma de los valores de cada casilla por la que el jugador pasó.

   Se debe determinar el máximo puntaje obtenible, si es que es posible llegar a la última fila.

   Se pide lo siguiente:

   (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. (Programación Dinámica) Implementá un algoritmo que utilice Programación Dinámica para resolver el problema del inciso anterior.

   - ¿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.

4. Para cada uno de los siguientes algoritmos determinar **por separado** cada uno de los siguientes incisos.

   (a) ¿Qué hace? ¿Cuáles son las precondiciones necesarias para haga eso?

   (b) ¿Cómo lo hace?

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

   (d) Proponer nombres más adecuados para los identificadores (de variables, funciones 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
```

5. Dada la especificación del tad Cola:

```
spec Queue of T where

constructors
    fun empty_queue() ret q : Queue of T
    {- crea una cola vacía. -}

    proc enqueue (in/out q : Queue of T, in e : T)
    {- agrega el elemento e al final de la cola q. -}

operations
    fun is_empty_queue(q : Queue of T) ret b : Bool
    {- Devuelve True si la cola es vacía -}

    fun first(q : Queue of T) ret e : T
    {- Devuelve el elemento que se encuentra al comienzo de q. -}
    {- PRE: not is_empty_queue(q) -}

    proc dequeue (in/out q : Queue of T)
    {- Elimina el elemento que se encuentra al comienzo de q. -}
    {- PRE: not is_empty_queue(q) -}
```

   Implementá el TAD utilizando punteros (siguiendo la idea de nodos enlazados), de manera que todas las operaciones (y los constructores) sean de orden **constante**.

6. (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?