Algoritmos y Estructuras de Datos II – 12 de Agosto 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. (Algoritmos voraces) Es principio de mes y tenés que ayudar a tu abuelo a pagar n facturas de servicios. El viejo es medio desconfiado y solo paga él mismo por ventanilla en efectivo, nada de transferencias o homebanking.

   Para cada factura i sabés qué día d_i va a llegar al domicilio y el día de vencimiento v_i. Obviamente no podés ir a pagar si no te ha llegado aún la factura al domicilio, y tenés que pagarlas todas antes del vencimiento (se puede pagar también el mismo día que vence). Como sos un excelente estudiante de Algoritmos 2, vas a diseñar un algoritmo que obtenga qué facturas se pagarán cada día, de manera tal que el abuelo vaya la menor cantidad de veces posible a la ventanilla de pago.

   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. Sos la única programadora de una flamante empresa que provee desarrollo en distintos proyectos. Tenés n proyectos posibles a los cuales ofrecer servicio y la posibilidad de trabajar H horas como máximo. Para cada proyecto $i \in \{1..n\}$ ya calculaste la cantidad de horas h_i que requiere de trabajo, y la paga p_i que recibirás si lo hacés. Tenés la posibilidad de pedirle a un amigo que te ayude con algunos proyectos, en cuyo caso te va a tomar la mitad de las horas (división entera) realizarlo, pero vas a cobrar la mitad del dinero (ya que la otra mitad se la darás a tu amigo). Tu tarea es calcular la máxima ganancia que podés obtener eligiendo qué proyectos tomar y cuándo recurrir a la ayuda de tu amigo.

   (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.

3. 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).

```
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(i, n-i+1));
         swap(p, n-i+1, t(i+1, n-i+1));
     od
end fun
```

4. Considere la siguiente especificación del tipo Pila de elementos de tipo T:

```
spec Stack of T where

constructors
      fun empty_stack() ret s : Stack of T
      {- crea una pila vacía. -}

      proc push (in e : T, in/out s : Stack of T)
      {- agrega el elemento e al tope de la pila s. -}

destructors
     proc destroy(in/out s : Stack of T)

copy
     fun copy(s : Stack of T) ret s : Stack of T

operations
      fun is_empty_stack(s : Stack of T) ret b : Bool
      {- Devuelve True si la pila es vacía -}

      fun top(s : Stack of T) ret e : T
      {- Devuelve el elemento que se encuentra en el tope de s. -}
      {- PRE:  not is_empty_stack(s) -}

      proc pop (in/out s : Stack of T)
      {- Elimina el elemento que se encuentra en el tope de s. -}
      {- PRE:  not is_empty_stack(s) -}
```

   (a) Utilizando como representación un arreglo de N elementos y un número natural, implementá los constructores y todas las operaciones indicando el orden de complejidad de cada una.

   (b) ¿La representación elegida tiene alguna limitación? Si es así, ¿cuál? Justificá la respuesta.

   (c) ¿Podría implementarse el tipo Pila utilizando como representación interna un conjunto de elementos? Justificá la respuesta.

(d) Utilizando el tipo **abstracto**, implementá un procedimiento *invert* que invierta los elementos de una pila de elementos de tipo T. ¿Qué orden de complejidad tiene la implementación?

5. (Para alumnos libres) Dar la forma general de los algoritmos divide y vencerás, identificar sus características, explicarlas y mencionar ejemplos de uso conocido de esa técnica.