Algoritmos y Estructuras de Datos II – 1 de Diciembre 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) Un amigo te recomienda que entres en el mundo del trading de criptomonedas asegurándote que siempre vas a ganar, ya que tiene una bola de cristal que ve el futuro.

   Conocés el valor actual $v_1^0, \ldots, v_n^0$ de n criptomonedas. La bola de cristal indica el valor que tendrá cada una de las criptomonedas durante los m días siguientes. Es decir, los valores $v_1^1, \ldots, v_1^m$ que tendrá la criptomoneda 1 dentro de 1 día, ..., dentro de m días respectivamente; los valores $v_2^1, \ldots, v_2^m$ que tendrá la criptomoneda 2 dentro de 1 día, ..., dentro de m días respectivamente, etcétera. En general, v_i^j es el valor que tendrá la criptomoneda i dentro de j días.

   Con esta preciada información podés diseñar un algoritmo que calcule el máximo dinero posible a obtener al cabo de m días comprando y vendiendo criptomonedas, a partir de una suma inicial de dinero D.

   Se asume que siempre habrá suficiente cantidad de cada criptomoneda para comprar y que no se cobra comisión alguna por la compra y venta. También se asume que se pueden comprar fracciones de criptomonedas. Recordá que no siempre las criptomonedas incrementan su valor.

   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. Finalmente tenés la posibilidad de irte N días (con sus respectivas noches) de vacaciones y en el recorrido que armaste, cada día/noche i estarás en una ciudad C_i. Contás con M pesos en total de presupuesto para gastar en alojamiento y para cada ciudad conocés el costo k_i por noche del único hotel que tiene. Cada noche i podés elegir entre dormir en el hotel de la ciudad, lo que te costará k_i, o dormir en una carpa que llevaste, que te cuesta 0. Además, tenés una tabla que indica para cada ciudad i, la puntuación p_i del hotel.

   Se debe encontrar la máxima puntuación obtenible eligiendo en qué ciudades dormirás en hotel, de manera tal que el presupuesto total gastado no supere el monto M. Notar que si decidís dormir en carpa en alguna ciudad, la puntuación correspondiente para la misma será 0.

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

4. Considere 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) Utilizando como representación un arreglo de N elementos de tipo T y un natural, definí la estructura que representa la lista, implementá los constructores y las operaciones tail, concat, length y drop.

(b) ¿La representación elegida para implementar el tad tiene alguna limitación? En caso afirmativo indicá cuál es. ¿Alguna operación debe tener una precondición extra?

(c) ¿Qué orden tiene la operación tail implementada en el inciso anterior? ¿Se podría modificar la estructura de datos utilizada para que tail sea constante? Explicá cómo.

(d) Implementá una función que reciba dos listas de enteros y devuelva otra lista que contenga en las posiciones pares todos los elementos de la primera lista (en el mismo orden), y en las posiciones impares todos los elementos de la segunda lista (en el mismo orden).

Para ello utilizá el tipo **abstracto**, sin acceder a su representación interna.

5. (Para alumnos libres) Cuando se utiliza backtracking, se recorre un grafo implícito que en sus caminos codifica todas las posibles soluciones, para elegir la más apropiada.

A partir del algoritmo que utiliza backtracking para resolver el problema de la moneda, dibujar el árbol implícito de búsqueda para monedas de denominaciones 3, 5 y 7 y monto a pagar 19.