Algoritmos y Estructuras de Datos II – 26 de Febrero de 2025

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. Realizar cada ejercicio en HOJAS SEPARADAS y NOMBRADAS. La no observación de estas recomendaciones resta puntaje.

1. (TADs) El TAD PilaReversible tiene los mismos constructores y las mismas operaciones que el TAD Pila, pero además tiene la operación **revertir** que dada una pila, invierte el orden de sus elementos: por ejemplo, si se aplica la operación invertir a la pila que tiene los elementos | 3 | 8 | 2 | (donde 3 es el tope), se obtendrá la pila | 2 | 8 | 3 | (donde 2 es el tope).

   Se pide:

   (a) Especificar el TAD PilaReversible indicando precisamente sus constructores y operaciones con sus respectivos prototipos de funciones/procedimientos.

   (b) Completar la implementación del TAD PilaReversible de manera tal que todos sus constructores y operaciones sean de orden constante. Para ello, se debe utilizar la siguiente representación conocida como "arreglos circulares":

   ```
   type Revstack of T= tuple
                          elems : array[0..N-1] of T
                          bottom : nat
                          top : nat
                          direction : {-1, 1}
                       end tuple
   ```

   donde bottom señala la posición en que se encuentra el elemento más viejo de la pila reversible, top señala la del más nuevo de la pila reversible y direction es 1 si la pila está almacenada "de izquierda a derecha" (orden usual) y -1 si la pila está almacenada "de derecha a izquierda" (por ejemplo, luego de la primera vez que se revierte). La idea es **nunca revertir realmente** los valores de la pila, pues si lo hiciera sería lineal.

   Los constructores se implementan de esta manera:

   ```
   fun empty() ret p : Revstack of T
       p.bottom:= 0
       p.top:= N-1
       p.direction:= 1
   end fun

   {Pre: ¬is_full(p)}
   proc push(in e : T,in/out p : Revstack of T)
         p.top:= (p.top + p.direction) mod N
         p.elems[p.top]:= e
   end proc
   ```

   Notar que esta representación tiene la limitación de que no se pueden almacenar más de N elementos en la pila. Por lo tanto, se debe implementar también una operación "is_full" que reciba una PilaReversible y devuelva True si la pila tiene N elementos almacenados, False caso contrario.

   NOTA: Asumimos que la operación módulo siempre devuelve un número entre 0 y N-1 (cuando su segundo argumento es N), incluso si el primer argumento es un número negativo. Por ejemplo, el resultado de -1 mod N es N-1.

   (c) Utilizando el **tipo abstracto** PilaReversible, implementar una operación que reciba una pila reversible y devuelva el elemento que se encuentra en la "base" de la misma. Es decir, si la pila es | 2 | 7 | 4 | 5 |, donde 2 es el tope, la operación debe devolver 5.

2. (Algoritmos voraces) Sos DJ y tenés una colección de N canciones de 15 artistas diferentes, cada una puntuada con un nivel de intensidad v_i, con i entre 1 y N. Tenés que armar un set de 30 canciones para pasar en una fiesta, de manera que la intensidad vaya creciendo desde la canción 1 a la 30, con la condición de que no podés pasar más de 3 canciones de un mismo autor. Se pide seleccionar las 30 canciones que pasarás, de manera que la suma total de intensidad sea máxima.

   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. 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.

4. Dados el siguiente procedimiento

```
proc p (in/out l: List)
    var a, b: pointer to Node
    a:= l
    while a ≠ null do
        b:= a→next
        if b ≠ null then
            a→next:= b→next
            free(b)
        fi
        a:= a → next
    od
end proc
```

donde los tipos Node y List se definen como sigue

```
type Node = tuple
                value: Int
                next: pointer to Node
            end
type List = pointer to Node
```

   (a) Explicá qué hace el procedimiento p.

   (b) ¿Cuál es el orden del procedimiento p? Justificá la respuesta.

   (c) Si se llama al procedimiento p con una lista que tiene los valores 1, 2, 3, 4, 5, 6 y 7, en ese orden, ¿qué valores tendrá la lista luego de la llamada?

5. (Para alumnos libres) Escribí una variante del algoritmo de ordenación por selección que vaya ordenando desde la última celda del arreglo hacia la primera. El resultado debe ser el mismo, es decir, el arreglo resultante debe estar ordenado en forma creciente, pero el modo de hacerlo debe ser diferente: en cada paso se debe seleccionar el **máximo** elemento aún no ordenado y colocarlo en la posición que corresponda desde el extremo final del arreglo.