Algoritmos y Estructuras de Datos II – 9 de diciembre de 2020

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. (Backtracking) Debemos llenar con dominós una fila de n casilleros con n par. Cada ficha ocupa 2 casilleros. Contamos con infinitas fichas de todos los tipos. Cada ficha tiene dos números i,j (de 0 a 6) y tiene un puntaje $P_{i,j}$ ($= P_{j,i}$). Como siempre en el dominó, al poner dos fichas juntas los números de los casilleros adyacentes deben coincidir. El último número de la última ficha debe ser un 6. Además, cada casillero tiene un número prohibido $c_1, \ldots, c_n$. Al colocar una ficha en dos casilleros, ésta debe respetar los números prohibidos.

   Escribir un algoritmo que utilice la técnica de backtracking para obtener el máximo puntaje posible colocando las fichas de manera que se repeten todas las restricciones. Antes de dar la solución, especificá con tus palabras qué calcula la función recursiva que resolverá el problema, detallando el rol de los argumentos y la llamada principal.

   Ejemplo: Con n = 6 debemos usar tres fichas. Si los números prohibidos para los casilleros son

3	2	6	5	0	5

   una posible solución es:

4	3	3	1	1	6

2. (Programación dinámica) Escribí un algoritmo que utilice Programación Dinámica para resolver el ejercicio del punto anterior.

   (a) ¿Qué dimensiones tiene la tabla que el algoritmo debe llenar?

   (b) ¿En qué orden se llena la misma?

   (c) ¿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. Completar la especificación del tipo Conjunto de elementos de tipo T, agregando las operaciones

   - **elim**, que elimina de un conjunto un elemento dado.
   - **union**, que agrega a un conjunto todos los elementos que pertenecen a otro conjunto dado.
   - **dif**, que elimina de un conjunto todos los elementos que pertenecen a otro conjunto dado.

spec Set **of** T **where**

constructors

fun empty_set() **ret** s : Set **of** T
{- *crea una pila vacía.* -}

proc add (**in** e : T, **in**/**out** s : Set **of** T)
{- *agrega el elemento* e *al conjunto* s. -}

destroy

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

operations

fun is_empty_set(s : Set **of** T) **ret** b : bool
{- *Devuelve* True *si* s *es vacío.* -}

fun cardinal(s : Set **of** T) **ret** n : nat
{- *Devuelve la cantidad de elementos que tiene* s -}

fun member(e : T, s : Set **of** T) **ret** b : bool
{- *Devuelve True si el elemento* e *pertenece al conjunto* s -}

proc inters (**in**/**out** s : Set **of** T,**in** s0 : Set **of** T)
{- *Elimina de* s *todos los elementos que NO pertenecen a* s0 -}

{- **PRE:** not is_empty_set(s) -}
fun get(s : Set **of** T) **ret** e : T
{- *Devuelve algún elemento (cualquiera) de* s -}

5. A partir de la siguiente implementación de conjuntos utilizando listas ordenadas, implemente el constructor **add**, y las operaciones **member**, **inters** y **cardinal**. La implementación debe mantener el invariante de representación por el cual todo conjunto está representado por una lista ordenada crecientemente. Puede utilizar todas las operaciones especificadas para el tipo lista vistas en el teórico. Para cada operación que utilice, especifique su encabezado, es decir: si es función o procedimiento, cómo se llama, qué argumentos toma y devuelve.

implement Set **of** T **where**

type Set **of** T = List **of** T

fun empty_set() **ret** s : Set **of** T
    s := empty_list()
end fun

6. (Para alumnos libres) ¿Qué hace el algoritmo de Prim? ¿Qué hace el algoritmo de Dijkstra?