

# Algoritmos y Estructuras de Datos II

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

## Trabajo Práctico II

### Grupo: 12

| Integrante            | LU     | Correo electrónico            |
|-----------------------|--------|-------------------------------|
| Pondal, Iván          | 078/14 | ivan.pondal@gmail.com         |
| Paz, Maximiliano León | 251/14 | m4xileon@gmail.com            |
| Mena, Manuel          | 313/14 | manuelmena1993@gmail.com      |
| Demartino, Francisco  | 348/14 | demartino.francisco@gmail.com |

### Reservado para la cátedra

| Instancia       | Docente | Nota |
|-----------------|---------|------|
| Primera entrega |         |      |
| Segunda entrega |         |      |

# Índice

|   |          |
|---|----------|
| <b>1. Módulo DCNet</b>                        | <b>3</b> |
| 1.1. Interfaz . . . . .                       | 3        |
| 1.1.1. Operaciones básicas de mapa . . . . .  | 3        |
| 1.2. Representación . . . . .                 | 3        |
| 1.2.1. Representación de dcnet . . . . .      | 3        |
| 1.2.2. Invariante de Representación . . . . . | 3        |
| <b>2. Módulo Red</b>                          | <b>5</b> |
| 2.1. Interfaz . . . . .                       | 5        |
| 2.2. Representación . . . . .                 | 6        |
| 2.2.1. Estructura . . . . .                   | 6        |
| 2.2.2. Invariante de Representación . . . . . | 6        |
| 2.2.3. Función de Abstracción . . . . .       | 6        |

# 1. Módulo DCNet

## 1.1. Interfaz

se explica con: DCNET.

géneros: dcnet.

### 1.1.1. Operaciones básicas de mapa

**CREAR()**  $\rightarrow res : dcnet$

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} vacio()\}$

**Complejidad:**  $O(1)$

**Descripción:** crea un mapa nuevo

## 1.2. Representación

### 1.2.1. Representación de dcnet

dcnet se representa con estr

donde estr es tupla(*topología*: red,  
                           *vectorCompusDCNet*: vector(compuDCNet),  
                           *diccCompusDCNet*: dicc<sub>trie</sub>(puntero(compuDCNet)),  
                           *laQueMásEnvió*: puntero(compuDCNet))

donde compuDCNet es tupla(*pc*: puntero(compu),  
                               *conjPaquetes*: conj(paquete),  
                               *diccPaquetesDCNet*: dicc<sub>avl</sub>(nat, paqueteDCNet),  
                               *colaPaquetesDCNet*: colaPrioridad(nat, paqueteDCNet),  
                               *paqueteAEnviar*: paqueteDCNet, *enviados*: nat)

donde paqueteDCNet es tupla(*it*: itConj(paquete), *recorrido*: lista(compu))

donde paquete es tupla(*id*: nat, *prioridad*: nat, *origen*: compu, *destino*: compu)

donde compu es tupla(*ip*: string, *interfaces*: conj(nat))

### 1.2.2. Invariante de Representación

- (I) Los elementos de vectorCompusDCNet son punteros a todas las compus de la topología
- (II) Las claves de diccCompusDCNet son todos los hostnames de la topología
- (III) Los significados de diccCompusDCNet son punteros que apuntan a las compuDCNet cuyo hostname equivale a su clave en vectorCompusDCNet
- (IV) laQueMásEnvió es un puntero a la compuDCNet en vectorCompusDCNet que más paquetes enviados tiene. Si no hay compus es NULL
- (V) Todos los paquetes en conjPaquetes de cada compuDCNet tienen id único
- (VI) El paquete en conjPaquetes tiene que tener en su recorrido a la compuDCNet en la que se encuentra y no puede ser igual a su destino

- (VII) Las claves de `diccPaquetesDCNet` son los `id` de los paquetes en `conjPaquetes`
- (VIII) Los significados de `diccPaquetesDCNet` contienen un `itConj` que apunta al paquete con el `id` equivalente a su clave y en recorrido, un camino mínimo válido para el origen del paquete y la compu en la que se encuentra
- (IX) Si `colaPaquetesDCNet` no es vacía, su próximo es un `paqueteDCNet` que contiene un `itConj` apuntando a uno de los paquetes de `conjPaquetes` con mayor prioridad y un recorrido, que es un camino mínimo válido para el origen del paquete y la compu en la que se encuentra

## 2. Módulo Red

### 2.1. Interfaz

se explica con: RED.

géneros: red.

INICIARRED()  $\rightarrow res : red$   
**Pre**  $\equiv \{true\}$   
**Post**  $\equiv \{res =_{obs} iniciarRed\}$   
**Complejidad:**  $O(1)$   
**Descripción:** Crea una red nueva

AGREGARCOMPUTADORA(in/out  $r : red$ , in  $c : compu$ )  
**Pre**  $\equiv \{(r = r_0) \wedge ((\forall c' : compu) (c' \in computadoras(r) \Rightarrow ip(c) \neq ip(c')))\}$   
**Post**  $\equiv \{r =_{obs} agregarComputadora(r_0, c)\}$   
**Complejidad:**  $O(L + n)$   
**Descripción:** Agrega un computadora a la red

CONECTAR(in/out  $r : red$ , in  $c : compu$ , in  $c' : compu$ , in  $i : compu$ , in  $i' : compu$ )  
**Pre**  $\equiv \{(r = r_0) \wedge (c \in computadoras(r)) \wedge (c' \in computadoras(r)) \wedge (ip(c) \neq ip(c')) \wedge (\neg conectadas?(r, c, c')) \wedge (\neg usaInterfaz?(r, c, i) \wedge \neg usaInterfaz?(r, c', i'))\}$   
**Post**  $\equiv \{r =_{obs} conectar(r_0, c, i, c', i')\}$   
**Complejidad:**  $O(L)?$   
**Descripción:** Conecta dos computadoras

COMPUTADORAS(in  $r : red$ )  $\rightarrow res : conj(compu)$   
**Pre**  $\equiv \{true\}$   
**Post**  $\equiv \{res = computadoras(r)\}$   
**Complejidad:**  $O(1)$

CONECTADAS?(in  $r : red$ , in  $c : compu$ , in  $c' : compu$ )  $\rightarrow res : bool$   
**Pre**  $\equiv \{(c \in computadoras(r)) \wedge (c' \in computadoras(r))\}$   
**Post**  $\equiv \{res = conectadas?(r, c, c')\}$   
**Complejidad:**  $O(1)$

INTERFAZUSADA(in  $r : red$ , in  $c : compu$ , in  $c' : compu$ )  $\rightarrow res : interfaz$   
**Pre**  $\equiv \{conectadas?(r, c, c')\}$   
**Post**  $\equiv \{res = interfazUsada(r, c, c')\}$   
**Complejidad:**  $O(?)$

VECINOS(in  $r : red$ , in  $c : compu$ )  $\rightarrow res : conj(compu)$   
**Pre**  $\equiv \{c \in computadoras(r)\}$   
**Post**  $\equiv \{res = vecinos(r, c)\}$   
**Complejidad:**  $O(n)$

USAIINTERFAZ?(in  $r : red$ , in  $c : compu$ , in  $i : interfaz$ )  $\rightarrow res : bool$   
**Pre**  $\equiv \{c \in computadoras(r)\}$   
**Post**  $\equiv \{res = usaInterfaz?(r, c, i)\}$   
**Complejidad:**  $O(?)$

CAMINOSMINIMOS(**in**  $r$ : red, **in**  $c$ : compu, **in**  $c'$ : compu)  $\rightarrow res$  : conj(secu(compu))  
**Pre**  $\equiv \{(c \in computadoras(r)) \wedge (c' \in computadoras(r))\}$   
**Post**  $\equiv \{res = caminosMinimos(r, c, i)\}$   
**Complejidad:**  $O(L)$

HAYCAMINO?(**in**  $r$ : red, **in**  $c$ : compu, **in**  $c'$ : compu)  $\rightarrow res$  : bool  
**Pre**  $\equiv \{(c \in computadoras(r)) \wedge (c' \in computadoras(r))\}$   
**Post**  $\equiv \{res = hayCamino?(r, c, i)\}$   
**Complejidad:**  $O(L)$

## 2.2. Representación

### 2.2.1. Estructura

red se representa con **estr**

donde **estr** es  $\text{tupla}(\text{compus: conj}(\text{compu}),$   
 $\text{dns: dicc}_{Trie}(\text{ip}, \text{nodoRed}) )$

donde **nodoRed** es  $\text{tupla}(c: \text{puntero}(\text{compu}),$   
 $\text{caminos: dicc}_{Trie}(\text{ip}, \text{conj}(\text{lista}(\text{compu}))) ,$   
 $\text{conexiones: dicc}_{Lineal}(\text{interfaz}, \text{compu}) )$

### 2.2.2. Invariante de Representación

- (I) Todas las compus deben tener IPs distintas.
- (II) Ninguna compu se conecta con si misma.
- (III) Ninguna compu se conecta a otra a traves de dos interfaces distintas.
- (IV) El trie **estr.dns** apunta a un **nodoRed** por cada elemento de **compus**.
- (V) En cada **nodoRed**,  $c$  tiene que apuntar a un elemento de **estr.compus**.
- (VI) Para cada **nodoRed**, **caminos** tiene como claves todas las IPs de las compus de la red, y los significados corresponden a todos los caminos mínimos desde la compu  $c$  hacia la compu cuya IP es clave.
- (VII) **nodoRed.conexiones** contiene como claves todas las **interfaz** usaconedas de la compu  $c$  (que tienen que estar en  $c.interfaces$ )

$\text{Rep} : \text{estr} \rightarrow \text{bool}$

$\text{Rep}(e) \equiv \text{true} \iff$

### 2.2.3. Función de Abstracción

$\text{Abs} : \text{estr } e \rightarrow \text{red}$

$\{\text{Rep}(e)\}$

$\text{Abs}(e) =_{\text{obs}} r : \text{red} \mid$