

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

| | |
|--|----------|
| 1. Módulo DCNet | 3 |
| 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 |
| 1.3. Algoritmos | 4 |
| 2. Módulo Red | 7 |
| 2.1. Interfaz | 7 |
| 2.2. Representación | 8 |
| 2.2.1. Estructura (????????? esto no esta listo todavia) | 8 |
| 2.2.2. Invariante de Representación | 8 |
| 2.2.3. Función de Abstracción | 8 |

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_{trie}(puntero(compuDCNet)),
 laQueMásEnvió: puntero(compuDCNet))

donde compuDCNet es tupla(*pc*: puntero(compu),
 conjPaquetes: conj(paquete),
 diccPaquetesDCNet: dicc_{avl}(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

1.3. Algoritmos

iIniciarDCNet (*in topo: red*) \rightarrow *res: estr*

```

res.topologia  $\leftarrow$  Copiar(topo)                                O(Fijarse en Copiar de Red)
res.vectorCompusDCNet  $\leftarrow$  Vacía()                            O(1)
res.diccCompusDCNet  $\leftarrow$  CREARDICCTRIE()                      O(1)
res.laQueMasEnvio  $\leftarrow$  NULL                                    O(1)

itConj(compu): it  $\leftarrow$  CrearIt(Computadoras(topo))           O(1)

if(HaySiguiente?(it)) then                                     O(1)
    res.laQueMasEnvio  $\leftarrow$  puntero(Siguiente(it))             O(1)
end if

while HaySiguiente?(it) do                                     O(1)
    ESTAVASINOESPUNTERO paqueteDCNet: paquetedcnet  $\leftarrow$  <CrearIt( $\emptyset$ ), Vacía(> O(1)
    compuDCNet: computdcnet  $\leftarrow$  <puntero(Siguiente(it)),  $\emptyset$ , CREARDICCAVL(),
                                Vacía(), NULL, 0>                O(1)
    AgregarAtras(res.vectorCompusDCNet, computdcnet)            O(n)
    DEFINIR(res.diccCompusDCNet, Siguiente(it).ip, puntero(computdcnet))
                                                                O(L)
    Avanzar(it)                                                  O(1)
end while                                                        O(n * (n + L))

```

Complejidad: $O(n * (n + L))$

iCrearPaquete (*in/out dcn: dcnet, in p: paquete*)

```

puntero(compuDCNet): computdcnet  $\leftarrow$ 
    Significado(dcn.diccCompusDCNet, p.origen.ip)                O(L)
itConj(paquete): itpaquete  $\leftarrow$  AgregarRapido(computdcnet  $\rightarrow$  conjPaquetes, p)
                                                                O(copy(compu))
lista(compu): recorr  $\leftarrow$  AgregarAtras(Vacía(), p.origen)      O(1)
puntero(paqueteDCNet): paquetedcnet  $\leftarrow$  puntero(<itpaquete, recorr>) O(1)
DEFINIR(computdcnet  $\rightarrow$  diccPaquetesDCNet, *paquetedcnet)    O(log(k))
ENCOLAR(computdcnet  $\rightarrow$  colaPaquetesDCNet, paquetedcnet)   O(log(k))

```

Complejidad: $O(\log(k) + L)$

iAvanzarSegundo (*in/out dcn: dcnet*)

```

nat: i  $\leftarrow$  0                                                  O(1)
while i < Longitud(dcn.vectorCompusDCNet) do                    O(1)
    if( $\neg$ EsVacía?(dcn.vectorCompusDCNet[i].colaPaquetesDCNet)) then

```

```

        dcn.vectorCompusDCNet[i].paqueteAEnviar ←
            DESENCOLAR(dcn.vectorCompusDCNet[i].colaPaquetesDCNet)      O(log(k))
    end if
    i++
end while
                                     O(1)
                                     O(n * log(k))

i ← 0
while i < Longitud(dcn.vectorCompusDCNet) do
    if (dcn.vectorCompusDCNet[i].paqueteAEnviar ≠ NULL) then
        dcn.vectorCompusDCNet[i].enviados++
                                     O(1)
                                     O(1)
                                     O(1)
                                     O(1)

        paquete: paqAEnviar ←
            Siguiete(dcn.vectorCompusDCNet[i].paqueteAEnviar→it)      O(copy(compu))
        itConj(lista(compu)): itercaminos ←
            CrearIt(CaminosMinimos(dcn.topologia,
                                   *(dcn.vectorCompusDCNet[i].pc), paqAEnviar.destino))
                                     O(1)
        compu: siguientecompu ← Primero(Siguiete(itercaminos))
                                     O(1)

        if (paqAEnviar.destino ≠ siguientecompu) then
                                     O(1)

            puntero(compuDCNet): siguientecompudcnet ←
                OBTENER(dcn.diccCompusDCNet, siguientecompu.ip)
                                     O(L)

            itConj(paquete): itpaquete ←
                AgregarRapido(siguientecompudcnet→conjPaquetes, paqAEnviar)
                                     O(copy(compu))

            dcn.vectorCompusDCNet[i].paqueteAEnviar→it ← itpaquete
                                     O(1)
            AgregarAtras(dcn.vectorCompusDCNet[i].paqueteAEnviar→recorrido,
                         siguientecompu)
                                     O(copy(compu))

            ENCOLAR(siguientecompudcnet→colaPaquetesDCNet,
                    dcn.vectorCompusDCNet[i].paqueteAEnviar)
                                     O(log(k))
            DEFINIR(siguientecompudcnet→diccPaquetesDCNet,
                    siguientecompu.ip,
                    *(dcn.vectorCompusDCNet[i].paqueteAEnviar))
                                     O(log(k))
        end if
        BORRAR(dcn.vectorCompusDCNet[i].diccPaquetesDCNet,
                Siguiete(dcn.vectorCompusDCNet[i].paqueteAEnviar→it).id)
                                     O(log(k))
        EliminarSiguiete(dcn.vectorCompusDCNet[i].paqueteAEnviar→it)
                                     O(1)
        dcn.vectorCompusDCNet[i].paqueteAEnviar ← NULL
                                     O(1)
    end if
    i++
end while
                                     O(1)
                                     O(n * (L + log(k)))

```

Complejidad : $O(n * (L + \log(k)))$

Red (in dcn: dcnet) → res: red

```

    res ← dcn.topologia
                                     O(1)

```

Complejidad : $O(1)$

CaminoRecorrido (in dcn: dcnet, in p: paquete) → res: lista(compu)

```

nat: i ← 0                                O(1)
while i < Longitud(dcn.vectorCompusDCNet) do  O(1)
  if DEF?(dcn.vectorCompusDCNet[i].diccPaquetesDCNet, p.id) then  O(log(k))
    res ← OBTENER(dcn.vectorCompusDCNet[i].diccPaquetesDCNet,
                  p.id)→recorrido                                O(log(k))
  end if
  i++
end while                                O(1)

```

Complejidad : $O(n * \log(k))$

CantidadEnviados (**in** dcn: dcnet, **in** c: compu) → res: nat

```

res ← OBTENER(dcn.diccCompusDCNet, c.ip)→enviados  O(L)

```

Complejidad : $O(L)$

EnEspera (**in** dcn: dcnet, **in** c: compu) → res: nat

```

res ← OBTENER(dcn.diccCompusDCNet, c.ip)→conjPaquetes  O(L)

```

Complejidad : $O(L)$

PaqueteEnTransito (**in** dcn: dcnet, **in** p: paquete) → res: bool

```

res ← false
nat: i ← 0                                O(1)
while i < Longitud(dcn.vectorCompusDCNet) do  O(1)
  if DEF?(dcn.vectorCompusDCNet[i].diccPaquetesDCNet, p.id) then  O(log(k))
    res ← true                                O(1)
  end if
  i++
end while                                O(1)

```

Complejidad : $O(n * \log(k))$

LaQueMasEnvio (**in** dcn: dcnet) → res: compu

```

res ← *(dcn.laQueMasEnvio→pc)  O(1)

```

Complejidad : $O(1)$

2. Módulo Red

2.1. Interfaz

se explica con: RED.

géneros: red.

INICIARRED() $\rightarrow res : \text{red}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} \text{iniciarRed}\}$

Complejidad: $O(1)$

Descripción: Crea una red nueva

AGREGARCOMPUTADORA(**in/out** $r : \text{red}$, **in** $c : \text{compu}$)

Pre $\equiv \{(r = r_0) \wedge ((\forall c' : \text{compu}) (c' \in \text{computadoras}(r) \Rightarrow \text{ip}(c) \neq \text{ip}(c')))\}$

Post $\equiv \{r =_{\text{obs}} \text{agregarComputadora}(r_0, c)\}$

Complejidad: $O(L + n)$

Descripción: Agrega un computadora a la red

CONECTAR(**in/out** $r : \text{red}$, **in** $c : \text{compu}$, **in** $c' : \text{compu}$, **in** $i : \text{compu}$, **in** $i' : \text{compu}$)

Pre $\equiv \{(r = r_0) \wedge (c \in \text{computadoras}(r)) \wedge (c' \in \text{computadoras}(r)) \wedge (\text{ip}(c) \neq \text{ip}(c'))$

$\wedge (\neg \text{conectadas?}(r, c, c')) \wedge (\neg \text{usaInterfaz?}(r, c, i) \wedge \neg \text{usaInterfaz?}(r, c', i'))\}$

Post $\equiv \{r =_{\text{obs}} \text{conectar}(r_0, c, i, c', i')\}$

Complejidad: $O(L)?$

Descripción: Conecta dos computadoras

COMPUTADORAS(**in** $r : \text{red}$) $\rightarrow res : \text{conj}(\text{compu})$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res = \text{computadoras}(r)\}$

Complejidad: $O(1)$

CONECTADAS?(**in** $r : \text{red}$, **in** $c : \text{compu}$, **in** $c' : \text{compu}$) $\rightarrow res : \text{bool}$

Pre $\equiv \{(c \in \text{computadoras}(r)) \wedge (c' \in \text{computadoras}(r))\}$

Post $\equiv \{res = \text{conectadas?}(r, c, c')\}$

Complejidad: $O(1)$

INTERFAZUSADA(**in** $r : \text{red}$, **in** $c : \text{compu}$, **in** $c' : \text{compu}$) $\rightarrow res : \text{interfaz}$

Pre $\equiv \{\text{conectadas?}(r, c, c')\}$

Post $\equiv \{res = \text{interfazUsada}(r, c, c')\}$

Complejidad: $O(?)$

VECINOS(**in** $r : \text{red}$, **in** $c : \text{compu}$) $\rightarrow res : \text{conj}(\text{compu})$

Pre $\equiv \{c \in \text{computadoras}(r)\}$

Post $\equiv \{res = \text{vecinos}(r, c)\}$

Complejidad: $O(n)$

USAINTERFAZ?(**in** $r : \text{red}$, **in** $c : \text{compu}$, **in** $i : \text{interfaz}$) $\rightarrow res : \text{bool}$

Pre $\equiv \{c \in \text{computadoras}(r)\}$

Post $\equiv \{res = \text{usaInterfaz?}(r, c, i)\}$

Complejidad: $O(?)$

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

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

2.2. Representación

2.2.1. Estructura (?????????? esto no esta listo todavia)

red se representa con estr

donde estr es $\text{tupla}(\text{compus} : \text{lista}(\text{nodoRed}), \text{dns} : \text{dicc}_{Trie}(\text{string}, \text{puntero}(\text{nodoRed})))$
 donde nodoRed es $\text{tupla}(c : \text{compu}, \text{vecinos} : \text{dicc}_{Lineal}(\text{nat}, \text{puntero}(\text{nodoRed})))$

2.2.2. Invariante de Representación

2.2.3. Función de Abstracción