Tema 22: Algoritmos sobre grafos

Informática (2010–11)

José A. Alonso Jiménez

Grupo de Lógica Computacional
Departamento de Ciencias de la Computación e I.A.
Universidad de Sevilla

Tema 22: Algoritmos sobre grafos

Tema 22: Algoritmos sobre grafos

Definiciones y terminología sobre grafos

- Un **grafo G** es un par (V, A) donde V es el conjunto de los **vértices** (o nodos) y A el de las **aristas**.
- Una **arista** del grafo es un par de vértices.
- Un **arco** es una arista dirigida.
- $|\mathbf{V}|$ es el número de vértices.
- $|\mathbf{A}|$ es el número de aristas.
- Un **camino** de v_1 a v_n es una sucesión de vértices $v_1, v_2, \dots, v_n$ tal que para todo i, $v_{i-1}v_i$ es una arista del grafo.
- Un **camino simple** es un camino tal que todos sus vértices son distintos.
- Un **ciclo** es un camino tal que $v_1 = v_n$ y todos los restantes vértices son distintos.

Definiciones y terminología sobre grafos

- Un **grafo acíclico** es un grafo sin ciclos.
- Un **grafo conexo** es un grafo tal que para cualquier par de vértices existe un camino del primero al segundo.
- Un **árbol** es un grafo acíclico conexo.
- Un vértice v es **adjacente** a v' si vv' es una arista del grafo.
- En un grafo dirigido, el **grado positivo** de un vértice es el número de aristas que salen de él y el **grado negativo** es el número de aristas que llegan a él.
- Un **grafo ponderado** es un grafo cuyas aristas tienen un peso.

Tema 22: Algoritmos sobre grafos

Signatura del TAD de los grafos

```
creaGrafo  :: (Ix v,Num p) => Bool -> (v,v) -> [(v,v,p)]
                              -> Grafo v p
adyacentes :: (Ix v,Num p) => (Grafo v p) -> v -> [v]
nodos      :: (Ix v,Num p) => (Grafo v p) -> [v]
aristasND  :: (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
aristasD   :: (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
aristaEn   :: (Ix v,Num p) => (Grafo v p) -> (v,v) -> Bool
peso       :: (Ix v,Num p) => v -> v -> (Grafo v p) -> p
```

Descripción de la signatura del TAD de grafos

- `(creaGrafo d cs as)` es un grafo (dirigido si `d` es True y no dirigido en caso contrario), con el par de cotas `cs` y listas de aristas as (cada arista es un trío formado por los dos vértices y su peso).
  Ver un ejemplo en la siguiente transparencia.
- `(adyacentes g v)` es la lista de los vértices adyacentes al nodo `v` en el grafo g.
- `(nodos g)` es la lista de todos los nodos del grafo g.
- `(aristasND g)` es la lista de las aristas del grafo no dirigido g.
- `(aristasD g)` es la lista de las aristas del grafo dirigido g.
- `(aristaEn g a)` se verifica si a es una arista del grafo g.
- `(peso v1 v2 g)` es el peso de la arista que une los vértices `v1` y `v2` en el grafo g.

Ejemplo de creación de grafos.

```
creaGrafo False (1,5) [(1,2,12),(1,3,34),(1,5,78),
                        (2,4,55),(2,5,32),
                        (3,4,61),(3,5,44),
                        (4,5,93)]
```

crea el grafo

```
       12
  1 -------- 2
  | \78     /|
  |  \   32/ |
  |   \   /  |
34|     5    |55
  |   /   \  |
  |  /44   \ |
  | /     93\|
  3 -------- 4
       61
```

Tema 22: Algoritmos sobre grafos

Los grafos como vectores de adyacencia

- Cabecera del módulo:

```
module GrafoConVectorDeAdyacencia
    (Grafo,
     creaGrafo,   -- (Ix v,Num p) => Bool -> (v,v) -> [(v,v,p)]
                  --                 -> Grafo v p
     adyacentes, -- (Ix v,Num p) => (Grafo v p) -> v -> [v]
     nodos,       -- (Ix v,Num p) => (Grafo v p) -> [v]
     aristasND,   -- (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
     aristasD,    -- (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
     aristaEn,    -- (Ix v,Num p) => (Grafo v p) -> (v,v) -> Bool
     peso         -- (Ix v,Num p) => v -> v -> (Grafo v p) -> p
    ) where
    where
```

- Librerías auxiliares.

```
import Data.Array
```

Los grafos como vectores de adyacencia

- `(Grafo v p)` es un grafo con vértices de tipo `v` y pesos de tipo `p`.

```
type Grafo v p = Array v [(v,p)]
```

- `grafoVA` es la representación del grafo del ejemplo de la página 9 mediante un vector de adyacencia.

```
grafoVA = array (1,5) [(1,[(2,12),(3,34),(5,78)]),
                       (2,[(1,12),(4,55),(5,32)]),
                       (3,[(1,34),(4,61),(5,44)]),
                       (4,[(2,55),(3,61),(5,93)]),
                       (5,[(1,78),(2,32),(3,44),(4,93)])]
```

Los grafos como vectores de adyacencia

- (creaGrafo d cs as) es un grafo (dirigido si d es True y no dirigido en caso contrario), con el par de cotas cs y listas de aristas as (cada arista es un trío formado por los dos vértices y su peso). Ver un ejemplo a continuación.

```
creaGrafo :: (Ix v, Num p) => Bool -> (v,v) -> [(v,v,p)]
                              -> Grafo v p
creaGrafo d cs vs =
    accumArray
     (\xs x -> xs++[x])
     []
     cs
     ((if d then []
       else [(x2,(x1,p))|(x1,x2,p) <- vs, x1 /= x2]) ++
      [(x1,(x2,p)) | (x1,x2,p) <- vs])
```

Los grafos como vectores de adyacencia

- (creaGrafo d cs as) es un grafo (dirigido si d es True y no dirigido en caso contrario), con el par de cotas cs y listas de aristas as (cada arista es un trío formado por los dos vértices y su peso). Ver un ejemplo a continuación.

```
creaGrafo :: (Ix v, Num p) => Bool -> (v,v) -> [(v,v,p)]
                                -> Grafo v p
creaGrafo d cs vs =
    accumArray
     (\xs x -> xs++[x])
     []
     cs
     ((if d then []
       else [(x2,(x1,p))|(x1,x2,p) <- vs, x1 /= x2]) ++
      [(x1,(x2,p)) | (x1,x2,p) <- vs])
```

Los grafos como vectores de adyacencia

- `grafoVA'` es el mismo grafo que `grafoVA` pero creado con `creaGrafo`. Por ejemplo,

```
ghci> grafoVA'
array (1,5) [(1,[(2,12),(3,34),(5,78)]),
             (2,[(1,12),(4,55),(5,32)]),
             (3,[(1,34),(4,61),(5,44)]),
             (4,[(2,55),(3,61),(5,93)]),
             (5,[(1,78),(2,32),(3,44),(4,93)])]
```

```
grafoVA' = creaGrafo False (1,5) [(1,2,12),(1,3,34),(1,5,78),
                                  (2,4,55),(2,5,32),
                                  (3,4,61),(3,5,44),
                                  (4,5,93)]
```

Los grafos como vectores de adyacencia

- (adyacentes g v) es la lista de los vértices adyacentes al nodo v en el grafo g. Por ejemplo,

```
adyacentes grafoVA' 4  ~>  [2,3,5]
```

```
adyacentes :: (Ix v,Num p) => (Grafo v p) -> v -> [v]
adyacentes g v = map fst (g!v)
```

- (nodos g) es la lista de todos los nodos del grafo g. Por ejemplo,

```
nodos grafoVA'  ~>  [1,2,3,4,5]
```

```
nodos :: (Ix v,Num p) => (Grafo v p) -> [v]
nodos g = indices g
```

Los grafos como vectores de adyacencia

- (adyacentes g v) es la lista de los vértices adyacentes al nodo v en el grafo g. Por ejemplo,

  ```
  adyacentes grafoVA' 4  ⇝  [2,3,5]
  ```

  ```
  adyacentes :: (Ix v,Num p) => (Grafo v p) -> v -> [v]
  adyacentes g v = map fst (g!v)
  ```

- (nodos g) es la lista de todos los nodos del grafo g. Por ejemplo,

  ```
  nodos grafoVA'  ⇝  [1,2,3,4,5]
  ```

  ```
  nodos :: (Ix v,Num p) => (Grafo v p) -> [v]
  nodos g = indices g
  ```

Los grafos como vectores de adyacencia

- (adyacentes g v) es la lista de los vértices adyacentes al nodo v en el grafo g. Por ejemplo,

  ```
  adyacentes grafoVA' 4  ⇝  [2,3,5]
  ```

```haskell
adyacentes :: (Ix v,Num p) => (Grafo v p) -> v -> [v]
adyacentes g v = map fst (g!v)
```

- (nodos g) es la lista de todos los nodos del grafo g. Por ejemplo,

  ```
  nodos grafoVA'  ⇝  [1,2,3,4,5]
  ```

```haskell
nodos :: (Ix v,Num p) => (Grafo v p) -> [v]
nodos g = indices g
```

Los grafos como vectores de adyacencia

- `(aristaEn g a)` se verifica si a es una arista del grafo g. Por ejemplo,

```
aristaEn grafoVA' (5,1)  ⇝  True
aristaEn grafoVA' (4,1)  ⇝  False
```

```
aristaEn :: (Ix v,Num p) => (Grafo v p) -> (v,v) -> Bool
aristaEn g (x,y) = elem y (adyacentes g x)
```

- `(peso v1 v2 g)` es el peso de la arista que une los vértices v1 y v2 en el grafo g. Por ejemplo,

```
peso 1 5 grafoVA'  ⇝  78
```

```
peso :: (Ix v,Num p) => v -> v -> (Grafo v p) -> p
peso x y g = head [c | (a,c) <- g!x , a == y]
```

Los grafos como vectores de adyacencia

- `(aristaEn g a)` se verifica si a es una arista del grafo g. Por ejemplo,

```
aristaEn grafoVA' (5,1)  ⇝  True
aristaEn grafoVA' (4,1)  ⇝  False
```

```
aristaEn :: (Ix v,Num p) => (Grafo v p) -> (v,v) -> Bool
aristaEn g (x,y) = elem y (adyacentes g x)
```

- `(peso v1 v2 g)` es el peso de la arista que une los vértices v1 y v2 en el grafo g. Por ejemplo,

```
peso 1 5 grafoVA'  ⇝  78
```

```
peso :: (Ix v,Num p) => v -> v -> (Grafo v p) -> p
peso x y g = head [c | (a,c) <- g!x , a == y]
```

Los grafos como vectores de adyacencia

- (aristaEn g a) se verifica si a es una arista del grafo g. Por ejemplo,

  ```
  aristaEn grafoVA' (5,1)  ⇝  True
  aristaEn grafoVA' (4,1)  ⇝  False
  ```

  ```haskell
  aristaEn :: (Ix v,Num p) => (Grafo v p) -> (v,v) -> Bool
  aristaEn g (x,y) = elem y (adyacentes g x)
  ```

- (peso v1 v2 g) es el peso de la arista que une los vértices v1 y v2 en el grafo g. Por ejemplo,

  ```
  peso 1 5 grafoVA'  ⇝  78
  ```

  ```haskell
  peso :: (Ix v,Num p) => v -> v -> (Grafo v p) -> p
  peso x y g = head [c | (a,c) <- g!x , a == y]
  ```

Los grafos como vectores de adyacencia

- (aristasD g) es la lista de las aristas del grafo dirigido g. Por ejemplo,

```
ghci> aristasD grafoVA'
[(1,2,12),(1,3,34),(1,5,78),
 (2,1,12),(2,4,55),(2,5,32),
 (3,1,34),(3,4,61),(3,5,44),
 (4,2,55),(4,3,61),(4,5,93),
 (5,1,78),(5,2,32),(5,3,44),(5,4,93)]
```

```
aristasD :: (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
aristasD g =
    [(v1,v2,w) | v1 <- nodos g , (v2,w) <- g!v1]
```

Los grafos como vectores de adyacencia

- (aristasD g) es la lista de las aristas del grafo dirigido g. Por ejemplo,

```
ghci> aristasD grafoVA'
[(1,2,12),(1,3,34),(1,5,78),
 (2,1,12),(2,4,55),(2,5,32),
 (3,1,34),(3,4,61),(3,5,44),
 (4,2,55),(4,3,61),(4,5,93),
 (5,1,78),(5,2,32),(5,3,44),(5,4,93)]
```

```
aristasD :: (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
aristasD g =
   [(v1,v2,w) | v1 <- nodos g , (v2,w) <- g!v1]
```

Los grafos como vectores de adyacencia

- `(aristasND g)` es la lista de las aristas del grafo no dirigido g. Por ejemplo,

```
ghci> aristasND grafoVA'
[(1,2,12),(1,3,34),(1,5,78),
 (2,4,55),(2,5,32),
 (3,4,61),(3,5,44),
 (4,5,93)]
```

```haskell
aristasND :: (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
aristasND g =
    [(v1,v2,w) | v1 <- nodos g, (v2,w) <- g!v1, v1 < v2]
```

Los grafos como vectores de adyacencia

- (aristasND g) es la lista de las aristas del grafo no dirigido g. Por ejemplo,

```
ghci> aristasND grafoVA'
[(1,2,12),(1,3,34),(1,5,78),
 (2,4,55),(2,5,32),
 (3,4,61),(3,5,44),
 (4,5,93)]
```

```haskell
aristasND :: (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
aristasND g =
    [(v1,v2,w) | v1 <- nodos g, (v2,w) <- g!v1, v1 < v2]
```

Tema 22: Algoritmos sobre grafos

Los grafos como matrices de adyacencia

- Cabecera del módulo.

```
module GrafoConMatrizDeAdyacencia
    (Grafo,
     creaGrafo,  -- (Ix v,Num p) => Bool -> (v,v) -> [(v,v,p)]
                 --                 -> Grafo v p
     adyacentes, -- (Ix v,Num p) => (Grafo v p) -> v -> [v]
     nodos,      -- (Ix v,Num p) => (Grafo v p) -> [v]
     aristasND,  -- (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
     aristasD,   -- (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
     aristaEn,   -- (Ix v,Num p) => (Grafo v p) -> (v,v) -> Bool
     peso        -- (Ix v,Num p) => v -> v -> (Grafo v p) -> p
    ) where
```

- Librerías auxiliares

```
import Data.Array
```

Los grafos como matrices de adyacencia

- `(Grafo v p)` es un grafo con vértices de tipo `v` y pesos de tipo `p`.

```
type Grafo v p = Array (v,v) (Maybe p)
```

- `grafoMA` es la representación del grafo del ejemplo de la página 9 mediante una matriz de adyacencia.

```
grafoMA = array ((1,1),(5,5))
                [((1,1),Nothing),((1,2),Just 10),((1,3),Just 20),
                 ((1,4),Nothing),((1,5),Nothing),((2,1),Nothing),
                 ((2,2),Nothing),((2,3),Nothing),((2,4),Just 30),
                 ((2,5),Nothing),((3,1),Nothing),((3,2),Nothing),
                 ((3,3),Nothing),((3,4),Just 40),((3,5),Nothing),
                 ((4,1),Nothing),((4,2),Nothing),((4,3),Nothing),
                 ((4,4),Nothing),((4,5),Just 50),((5,1),Nothing),
                 ((5,2),Nothing),((5,3),Nothing),((5,4),Nothing),
                 ((5,5),Nothing)]
```

Los grafos como matrices de adyacencia

- `(creaGrafo d cs as)` es un grafo (dirigido si `d` es True y no dirigido en caso contrario), con el par de cotas `cs` y listas de aristas `as` (cada arista es un trío formado por los dos vértices y su peso). Ver un ejemplo a continuación.

```
creaGrafo :: (Ix v, Num p) => Bool -> (v,v) -> [(v,v,p)]
                                -> Grafo v p
creaGrafo dir cs@(l,u) as
    = matrizVacia //
      ([((x1,x2),Just w) | (x1,x2,w) <- as] ++
       if dir then []
       else [((x2,x1),Just w) | (x1,x2,w) <- as, x1 /= x2])
    where
      matrizVacia = array ((l,l),(u,u))
                          [((x1,x2),Nothing) | x1 <- range cs,
                                               x2 <- range cs]
```

Los grafos como matrices de adyacencia

- `(creaGrafo d cs as)` es un grafo (dirigido si `d` es True y no dirigido en caso contrario), con el par de cotas `cs` y listas de aristas `as` (cada arista es un trío formado por los dos vértices y su peso). Ver un ejemplo a continuación.

```
creaGrafo :: (Ix v, Num p) => Bool -> (v,v) -> [(v,v,p)]
                              -> Grafo v p
creaGrafo dir cs@(l,u) as
    = matrizVacia //
      ([((x1,x2),Just w) | (x1,x2,w) <- as] ++
       if dir then []
       else [((x2,x1),Just w) | (x1,x2,w) <- as, x1 /= x2])
    where
      matrizVacia = array ((l,l),(u,u))
                          [((x1,x2),Nothing) | x1 <- range cs,
                                               x2 <- range cs]
```

Los grafos como matrices de adyacencia

- grafoMA' es el mismo grafo que grafoMA pero creado con creaGrafo. Por ejemplo,

```
ghci> grafoMA'
array ((1,1),(5,5))
      [((1,1),Nothing),((1,2),Just 12),((1,3),Just 34),
       ((1,4),Nothing),((1,5),Just 78),((2,1),Just 12),
       ((2,2),Nothing),((2,3),Nothing),((2,4),Just 55),
       ((2,5),Just 32),((3,1),Just 34),((3,2),Nothing),
       ((3,3),Nothing),((3,4),Just 61),((3,5),Just 44),
       ((4,1),Nothing),((4,2),Just 55),((4,3),Just 61),
       ((4,4),Nothing),((4,5),Just 93),((5,1),Just 78),
       ((5,2),Just 32),((5,3),Just 44),((5,4),Just 93),
       ((5,5),Nothing)]
```

```
grafoMA' = creaGrafo False (1,5) [(1,2,12),(1,3,34),(1,5,78),
                                   (2,4,55),(2,5,32),
                                   (3,4,61),(3,5,44),
                                   (4,5,93)]
```

Los grafos como matrices de adyacencia

- (adyacentes g v) es la lista de los vértices adyacentes al nodo v en el grafo g. Por ejemplo,

  ```
  adyacentes grafoMA' 4  ⇝  [2,3,5]
  ```

```
adyacentes :: (Ix v,Num p) => (Grafo v p) -> v -> [v]
adyacentes g v1 =
    [v2 | v2 <- nodos g, (g!(v1,v2)) /= Nothing]
```

- (nodos g) es la lista de todos los nodos del grafo g. Por ejemplo,

  ```
  nodos grafoMA'  ⇝  [1,2,3,4,5]
  ```

```
nodos :: (Ix v,Num p) => (Grafo v p) -> [v]
nodos g = range (l,u)
    where ((l,_),(u,_)) = bounds g
```

Los grafos como matrices de adyacencia

- (adyacentes g v) es la lista de los vértices adyacentes al nodo v en el grafo g. Por ejemplo,

  adyacentes grafoMA' 4 ⇝ [2,3,5]

```
adyacentes :: (Ix v,Num p) => (Grafo v p) -> v -> [v]
adyacentes g v1 =
    [v2 | v2 <- nodos g, (g!(v1,v2)) /= Nothing]
```

- (nodos g) es la lista de todos los nodos del grafo g. Por ejemplo,

  nodos grafoMA' ⇝ [1,2,3,4,5]

```
nodos :: (Ix v,Num p) => (Grafo v p) -> [v]
nodos g = range (l,u)
    where ((l,_),(u,_)) = bounds g
```

Los grafos como matrices de adyacencia

- `(adyacentes g v)` es la lista de los vértices adyacentes al nodo v en el grafo g. Por ejemplo,

  ```
  adyacentes grafoMA' 4  ~>  [2,3,5]
  ```

```haskell
adyacentes :: (Ix v,Num p) => (Grafo v p) -> v -> [v]
adyacentes g v1 =
    [v2 | v2 <- nodos g, (g!(v1,v2)) /= Nothing]
```

- `(nodos g)` es la lista de todos los nodos del grafo g. Por ejemplo,

  ```
  nodos grafoMA'  ~>  [1,2,3,4,5]
  ```

```haskell
nodos :: (Ix v,Num p) => (Grafo v p) -> [v]
nodos g = range (l,u)
    where ((l,_),(u,_)) = bounds g
```

Los grafos como matrices de adyacencia

- (aristaEn g a) se verifica si a es una arista del grafo g. Por ejemplo,

  ```
  aristaEn grafoMA' (5,1)  ⇝  True
  aristaEn grafoMA' (4,1)  ⇝  False
  ```

  ```
  aristaEn :: (Ix v,Num p) => (Grafo v p) -> (v,v) -> Bool
  aristaEn g (x,y)= (g!(x,y)) /= Nothing
  ```

- (peso v1 v2 g) es el peso de la arista que une los vértices v1 y v2 en el grafo g. Por ejemplo,

  ```
  peso 1 5 grafoMA'  ⇝  78
  ```

  ```
  peso :: (Ix v,Num p) => v -> v -> (Grafo v p) -> p
  peso x y g  = w where (Just w) = g!(x,y)
  ```

Los grafos como matrices de adyacencia

- (aristaEn g a) se verifica si a es una arista del grafo g. Por ejemplo,

```
aristaEn grafoMA' (5,1)  ⇝  True
aristaEn grafoMA' (4,1)  ⇝  False
```

```
aristaEn :: (Ix v,Num p) => (Grafo v p) -> (v,v) -> Bool
aristaEn g (x,y)= (g!(x,y)) /= Nothing
```

- (peso v1 v2 g) es el peso de la arista que une los vértices v1 y v2 en el grafo g. Por ejemplo,

```
peso 1 5 grafoMA'  ⇝  78
```

```
peso :: (Ix v,Num p) => v -> v -> (Grafo v p) -> p
peso x y g  = w where (Just w) = g!(x,y)
```

Los grafos como matrices de adyacencia

- (aristaEn g a) se verifica si a es una arista del grafo g. Por ejemplo,

```
aristaEn grafoMA' (5,1)  ⇝  True
aristaEn grafoMA' (4,1)  ⇝  False
```

```
aristaEn :: (Ix v,Num p) => (Grafo v p) -> (v,v) -> Bool
aristaEn g (x,y)= (g!(x,y)) /= Nothing
```

- (peso v1 v2 g) es el peso de la arista que une los vértices v1 y v2 en el grafo g. Por ejemplo,

```
peso 1 5 grafoMA'  ⇝  78
```

```
peso :: (Ix v,Num p) => v -> v -> (Grafo v p) -> p
peso x y g  = w where (Just w) = g!(x,y)
```

Los grafos como matrices de adyacencia

- (aristasD g) es la lista de las aristas del grafo dirigido g. Por ejemplo,

```
ghci> aristasD grafoMA'
[(1,2,12),(1,3,34),(1,5,78),
 (2,1,12),(2,4,55),(2,5,32),
 (3,1,34),(3,4,61),(3,5,44),
 (4,2,55),(4,3,61),(4,5,93),
 (5,1,78),(5,2,32),(5,3,44),(5,4,93)]
```

```
aristasD :: (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
aristasD g = [(v1,v2,extrae(g!(v1,v2)))
              | v1 <- nodos g,
                v2 <- nodos g,
                aristaEn g (v1,v2)]
    where extrae (Just w) = w
```

Los grafos como matrices de adyacencia

- (aristasD g) es la lista de las aristas del grafo dirigido g. Por ejemplo,

```
ghci> aristasD grafoMA'
[(1,2,12),(1,3,34),(1,5,78),
 (2,1,12),(2,4,55),(2,5,32),
 (3,1,34),(3,4,61),(3,5,44),
 (4,2,55),(4,3,61),(4,5,93),
 (5,1,78),(5,2,32),(5,3,44),(5,4,93)]
```

```
aristasD :: (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
aristasD g = [(v1,v2,extrae(g!(v1,v2)))
              | v1 <- nodos g,
                v2 <- nodos g,
                aristaEn g (v1,v2)]
    where extrae (Just w) = w
```

Los grafos como matrices de adyacencia

- (aristasND g) es la lista de las aristas del grafo no dirigido g. Por ejemplo,

```
ghci> aristasND grafoMA'
[(1,2,12),(1,3,34),(1,5,78),
 (2,4,55),(2,5,32),
 (3,4,61),(3,5,44),
 (4,5,93)]
```

```
aristasND :: (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
aristasND g = [(v1,v2,extrae(g!(v1,v2)))
              | v1 <- nodos g,
                v2 <- range (v1,u),
                aristaEn g (v1,v2)]
    where (_,(u,_)) = bounds g
          extrae (Just w) = w
```

Los grafos como matrices de adyacencia

- (aristasND g) es la lista de las aristas del grafo no dirigido g. Por ejemplo,

```
ghci> aristasND grafoMA'
[(1,2,12),(1,3,34),(1,5,78),
 (2,4,55),(2,5,32),
 (3,4,61),(3,5,44),
 (4,5,93)]
```

```haskell
aristasND :: (Ix v,Num p) => (Grafo v p) -> [(v,v,p)]
aristasND g = [(v1,v2,extrae(g!(v1,v2)))
               | v1 <- nodos g,
                 v2 <- range (v1,u),
                 aristaEn g (v1,v2)]
    where (_,(u,_)) = bounds g
          extrae (Just w) = w
```

Tema 22: Algoritmos sobre grafos

Recorrido en profundidad

- Importaciones de librerías auxiliares.

```
-- Nota: Elegir una implementación de los grafos.
import GrafoConVectorDeAdyacencia
-- import GrafoConMatrizDeAdyacencia

-- Nota: Elegir una implementación de las pilas.
import PilaConListas
-- import PilaConTipoDeDatoAlgebraico
```

Recorrido en profundidad

- En los ejemplos se usará el grafo g

```
+---> 2 <---+
|           |
|           |
1 --> 3 --> 6 --> 5
|                 |
|                 |
+---> 4 <---------+
```

que se define por

```
g = creaGrafo True (1,6)
              [(1,2,0),(1,3,0),(1,4,0),(3,6,0),
               (5,4,0),(6,2,0),(6,5,0)]
```

Procedimiento elemental de recorrido en profundidad

- `(recorridoEnProfundidad i g)` es el recorrido en profundidad del grafo `g` desde el vértice `i`. Por ejemplo,

```
recorridoEnProfundidad 1 g  ⇝  [1,2,3,6,5,4]
```

```
recorridoEnProfundidad i g = rp [i] []
    where
      rp [] vis = vis
      rp (c:cs) vis
          | elem c vis = rp cs vis
          | otherwise  = rp ((adyacentes g c)++cs)
                            (vis++[c])
```

Procedimiento elemental de recorrido en profundidad

- `(recorridoEnProfundidad i g)` es el recorrido en profundidad del grafo g desde el vértice `i`. Por ejemplo,

  ```
  recorridoEnProfundidad 1 g  ⇝  [1,2,3,6,5,4]
  ```

```haskell
recorridoEnProfundidad i g = rp [i] []
    where
      rp [] vis = vis
      rp (c:cs) vis
          | elem c vis = rp cs vis
          | otherwise  = rp ((adyacentes g c)++cs)
                            (vis++[c])
```

Procedimiento elemental de recorrido en profundidad

- Traza del cálculo de (recorridoEnProfundidad 1 g)

```
recorridoEnProfundidad 1 g
= rp [1]      []
= rp [2,3,4]  [1]
= rp [3,4]    [1,2]
= rp [6,4]    [1,2,3]
= rp [2,5,4]  [1,2,3,6]
= rp [5,4]    [1,2,3,6]
= rp [4,4]    [1,2,3,6,5]
= rp [4]      [1,2,3,6,5,4]
= rp []       [1,2,3,6,5,4]
= [1,2,3,6,5,4]
```

Recorrido en profundidad con acumuladores

- (recorridoEnProfundidad' i g) es el recorrido en profundidad del grafo, usando la lista de los visitados como acumulador. Por ejemplo,

```
recorridoEnProfundidad' 1 g  ⇝  [1,2,3,6,5,4]
```

```
recorridoEnProfundidad' i g = reverse (rp [i] [])
    where
      rp [] vis      = vis
      rp (c:cs) vis
          | elem c vis = rp cs vis
          | otherwise  = rp ((adyacentes g c)++cs)
                            (c:vis)
```

Recorrido en profundidad con acumuladores

- (recorridoEnProfundidad' i g) es el recorrido en profundidad del grafo, usando la lista de los visitados como acumulador. Por ejemplo,

  ```
  recorridoEnProfundidad' 1 g  ⇝  [1,2,3,6,5,4]
  ```

```
recorridoEnProfundidad' i g = reverse (rp [i] [])
    where
      rp [] vis     = vis
      rp (c:cs) vis
          | elem c vis = rp cs vis
          | otherwise  = rp ((adyacentes g c)++cs)
                            (c:vis)
```

Recorrido en profundidad con acumuladores

- Traza del cálculo de (`recorridoEnProfundidad' 1 g`)

```
recorridoEnProfundidad' 1 g
= reverse (rp [1]     [])
= reverse (rp [2,3,4] [1])
= reverse (rp [3,4]   [2,1])
= reverse (rp [6,4]   [3,2,1])
= reverse (rp [2,5,4] [6,3,2,1])
= reverse (rp [5,4]   [6,3,2,1])
= reverse (rp [4,4]   [5,6,3,2,1])
= reverse (rp [4]     [4,5,6,3,2,1])
= reverse (rp []      [4,5,6,3,2,1])
= reverse [4,5,6,3,2,1]
= [1,2,3,6,5,4]
```

Recorrido en profundidad con pilas

- `(recorridoEnProfundidad" i g)` es el recorrido en profundidad del grafo `g` desde el vértice `i`, usando pilas. Por ejemplo,

```
recorridoEnProfundidad'' 1 g  ⇝  [1,2,3,6,5,4]
```

```
recorridoEnProfundidad'' i g = reverse (rp (apila i vacia)
 where
   rp s vis
    | esVacia s          = vis
    | elem (cima s) vis = rp (desapila s) vis
    | otherwise          = rp (foldr apila (desapila s)
                                            (adyacentes g c))
                                   (c:vis)
                              where c = cima s
```

Recorrido en profundidad con pilas

- `(recorridoEnProfundidad'' i g)` es el recorrido en profundidad del grafo g desde el vértice i, usando pilas. Por ejemplo,

```
recorridoEnProfundidad'' 1 g  ~>  [1,2,3,6,5,4]
```

```haskell
recorridoEnProfundidad'' i g = reverse (rp (apila i vacia) [])
 where
   rp s vis
    | esVacia s          = vis
    | elem (cima s) vis = rp (desapila s) vis
    | otherwise          = rp (foldr apila (desapila s)
                                              (adyacentes g c))
                                  (c:vis)
                               where c = cima s
```

Tema 22: Algoritmos sobre grafos

Recorrido en anchura

- Importaciones de librerías auxiliares.

```
-- Nota: Elegir una implementación de los grafos.
import GrafoConVectorDeAdyacencia
-- import GrafoConMatrizDeAdyacencia

-- Nota: Elegir una implementación de las colas
import ColaConListas
-- import ColaConDosListas
```

Recorrido en anchura

- `(recorridoEnAnchura i g)` es el recorrido en anchura del grafo `g` desde el vértice `i`, usando colas. Por ejemplo,

```
recorridoEnAnchura 1 g  ⇝  [1,4,3,2,6,5]
```

```haskell
recorridoEnAnchura i g = reverse (ra (inserta i vacia) [])
 where
  ra q vis
    | esVacia q = vis
    | elem (primero q) vis = ra (resto q) vis
    | otherwise  = ra (foldr inserta (resto q)
                                   (adyacentes g c))
                         (c:vis)
                       where c = primero q
```

Recorrido en anchura

- `(recorridoEnAnchura i g)` es el recorrido en anchura del grafo g desde el vértice `i`, usando colas. Por ejemplo,

  ```
  recorridoEnAnchura 1 g  ⇝  [1,4,3,2,6,5]
  ```

  ```
  recorridoEnAnchura i g = reverse (ra (inserta i vacia) [])
   where
    ra q vis
     | esVacia q = vis
     | elem (primero q) vis = ra (resto q) vis
     | otherwise  = ra (foldr inserta (resto q)
                                   (adyacentes g c))
                         (c:vis)
                     where c = primero q
  ```

Tema 22: Algoritmos sobre grafos

Ordenación topológica

- Dado un grafo dirigido acíclico, una ordenación topológica es una ordenación de los vértices del grafo tal que si existe un camino de `v` a `v'`, entonces `v'` aparece después que `v` en el orden.
- Librerías auxiliares.

```
-- Nota: Elegir una implementación de los grafos.
import GrafoConVectorDeAdyacencia
-- import GrafoConMatrizDeAdyacencia

import Data.Array
```

- Se usará para los ejemplos el grafo g de la página 44.

Ordenación topológica

- `(gradoEnt g n)` es el grado de entrada del nodo `n` en el grafo `g`; es decir, el número de aristas de `g` que llegan a `n`. Por ejemplo,

```
gradoEnt g 1  ⇝  0
gradoEnt g 2  ⇝  2
gradoEnt g 3  ⇝  1
```

```
gradoEnt :: (Ix a, Num p) => Grafo a p -> a -> Int
gradoEnt g n =
   length [t | v <- nodos g, t <- adyacentes g v, n==t]
```

Ordenación topológica

- `(gradoEnt g n)` es el grado de entrada del nodo `n` en el grafo `g`; es decir, el número de aristas de `g` que llegan a `n`. Por ejemplo,

```
gradoEnt g 1  ⇝  0
gradoEnt g 2  ⇝  2
gradoEnt g 3  ⇝  1
```

```
gradoEnt :: (Ix a, Num p) => Grafo a p -> a -> Int
gradoEnt g n =
   length [t | v <- nodos g, t <- adyacentes g v, n==t]
```

Ordenación topológica

- `(ordenacionTopologica g)` es una ordenación topológica del grafo g. Por ejemplo,

```
ordenacionTopologica g  ⇝  [1,3,6,5,4,2]
```

```
ordenacionTopologica g =
  ordTop [n | n <- nodos g , gradoEnt g n == 0] []
  where
    ordTop [] r     = r
    ordTop (c:cs) vis
      | elem c vis = ordTop cs vis
      | otherwise  = ordTop cs (c:(ordTop (adyacentes g c) vis))
```

Ordenación topológica

- `(ordenacionTopologica g)` es una ordenación topológica del grafo g. Por ejemplo,

```
ordenacionTopologica g  ⇝  [1,3,6,5,4,2]
```

```haskell
ordenacionTopologica g =
  ordTop [n | n <- nodos g , gradoEnt g n == 0] []
  where
    ordTop [] r     = r
    ordTop (c:cs) vis
     | elem c vis = ordTop cs vis
     | otherwise  = ordTop cs (c:(ordTop (adyacentes g c) vis))
```

Ordenación topológica

- Ejemplo de ordenación topológica de cursos.

```
data Cursos = Matematicas   | Computabilidad | Lenguajes | Programacion |
              Concurrencia | Arquitectura    | Paralelismo
    deriving (Eq,Ord,Enum,Ix,Show)

gc = creaGrafo True (Matematicas,Paralelismo)
                   [(Matematicas,Computabilidad,1),
                    (Lenguajes,Computabilidad,1),
                    (Programacion,Lenguajes,1),
                    (Programacion,Concurrencia,1),
                    (Concurrencia,Paralelismo,1),
                    (Arquitectura,Paralelismo,1)]
```

La ordenación topológica es

```
ghci> ordenacionTopologica gc
[Arquitectura,Programacion,Concurrencia,Paralelismo,Lenguajes,
 Matematicas,Computabilidad]
```

Tema 22: Algoritmos sobre grafos

Árboles de expansión mínimos

- Sea $G = (V, A)$ un grafo conexo no orientado en el que cada arista tiene un peso no negativo. Un **árbol de expansión mínimo** de G es un subgrafo $G' = (V, A')$ que conecta todos los vértices de G y tal que la suma de sus pesos es mínima.
- **Aplicación:** Si los vértices representan ciudades y el coste de una arista $\{a, b\}$ es el construir una carretera de a a b, entonces un árbol de expansión mínimo representa el modo de enlazar todas las ciudades mediante una red de carreteras de coste mínimo.

Árboles de expansión mínimos

- Terminología de algoritmos voraces: Sea $G = (V, A)$ un grafo y T un conjunto de aristas de G.
  - T es una **solución** si es un grafo de expansión.
  - T es **completable** si no tiene ciclos.
  - T es **prometedor** si es completable y puede ser completado hasta llegar a una solución óptima.
  - Una arista **toca** un conjunto de vértices B si exactamente uno de sus extremos pertenece a B.
- **Teorema:** Sea $G = (V, A)$ un grafo conexo no orientado cuyas aristas tienen un peso asociado. Sea B un subjconjunto propio del conjunto de vértices V y T un conjunto prometedor de aristas tal que ninguna arista de T toca a B. Sea e una arista de peso mínimo de entre todas las que tocan a B. Entonces $(T \cup \{e\})$ es prometedor.

Tema 22: Algoritmos sobre grafos

El algoritmo de Kruskal

Para los ejemplos se considera el siguiente grafo:

```
     1       2
 1 ----- 2 ----- 3
 |      /|      /|
 |     / |     / |
 |    /  |    /  |
4|   /6  |4  /5  |6
 |  /    |  /    |
 | /     | /     |
 |/      |/      |
 4 ----- 5 ----- 6
  \  3   |   8  /
   \     |     /
    \    |    /
    4\   |7  /3
      \  |  /
       \ | /
        \|/
         7
```

El algoritmo de Kruskal

- Aplicación del algoritmo de Kruskal al grafo anterior:

```
Etapa Arista Componentes conexas
0            {1} {2} {3} {4} {5} {6} {7}
1     {1,2}  {1,2} {3} {4} {5} {6} {7}
2     {2,3}  {1,2,3} {4} {5} {6} {7}
3     {4,5}  {1,2,3} {4,5} {6} {7}
4     {6,7}  {1,2,3} {4,5} {6,7}
5     {1,4}  {1,2,3,4,5} {6,7}
6     {2,5}  arista rechazada
7     {4,7}  {1,2,3,4,5,6,7}
```

- El árbol de expansión mínimo contiene las aristas no rechazadas:

```
{1,2}, {2,3}, {4,5}, {6,7}, {1,4} y {4,7}.
```

El algoritmo de Kruskal

- Librerías auxiliares.

```
-- Nota: Seleccionar una implementación del TAD grafo.
-- import GrafoConVectorDeAdyacencia
import GrafoConMatrizDeAdyacencia

-- Nota: Seleccionar una implementación del TAD cola
-- de prioridad.
import ColaDePrioridadConListas
-- import ColaDePrioridadConMonticulos

-- Nota: Seleccionar una implementación del TAD tabla.
-- import TablaConFunciones
import TablaConListasDeAsociacion
-- import TablaConMatrices

import Data.List
import Data.Ix
```

El algoritmo de Kruskal

- Grafos usados en los ejemplos.

```
g1 :: Grafo Int Int
g1 = creaGrafo True (1,5) [(1,2,12),(1,3,34),(1,5,78),
                           (2,4,55),(2,5,32),
                           (3,4,61),(3,5,44),
                           (4,5,93)]

g2 :: Grafo Int Int
g2 = creaGrafo True (1,5) [(1,2,13),(1,3,11),(1,5,78),
                           (2,4,12),(2,5,32),
                           (3,4,14),(3,5,44),
                           (4,5,93)]
```

El algoritmo de Kruskal

- (kruskal g) es el árbol de expansión mínimo del grafo g calculado mediante el algoritmo de Kruskal. Por ejemplo,

```
kruskal g1  ⇝  [(55,2,4),(34,1,3),(32,2,5),(12,1,2)]
kruskal g2  ⇝  [(32,2,5),(13,1,2),(12,2,4),(11,1,3)]
```

```
kruskal :: (Num p, Ix n, Ord p) => Grafo n p -> [(p,n,n)]
kruskal g =
   kruskal' (llenaCP (aristasND g) vacia)  -- Cola de prioridad
            (tabla [(x,x) | x <- nodos g]) -- Tabla de raices
            []                             -- Árbol de expansión
            ((length (nodos g)) - 1)       -- Aristas por colocar

kruskal' cp t ae n
    | n==0        = ae
    | actualizado = kruskal' cp' t' (a:ae) (n-1)
    | otherwise   = kruskal' cp' t  ae     n
    where a@(_,x,y)         = primero cp
          cp'               = resto cp
          (actualizado,t')  = buscaActualiza (x,y) t
```

El algoritmo de Kruskal

- (kruskal g) es el árbol de expansión mínimo del grafo g calculado mediante el algoritmo de Kruskal. Por ejemplo,

```
kruskal g1  ⇝  [(55,2,4),(34,1,3),(32,2,5),(12,1,2)]
kruskal g2  ⇝  [(32,2,5),(13,1,2),(12,2,4),(11,1,3)]
```

```haskell
kruskal :: (Num p, Ix n, Ord p) => Grafo n p -> [(p,n,n)]
kruskal g =
   kruskal' (llenaCP (aristasND g) vacia)  -- Cola de prioridad
            (tabla [(x,x) | x <- nodos g]) -- Tabla de raices
            []                             -- Árbol de expansión
            ((length (nodos g)) - 1)       -- Aristas por colocar

kruskal' cp t ae n
    | n==0        = ae
    | actualizado = kruskal' cp' t' (a:ae) (n-1)
    | otherwise   = kruskal' cp' t  ae     n
    where a@(_,x,y)          = primero cp
          cp'                = resto cp
          (actualizado,t')   = buscaActualiza (x,y) t
```

El algoritmo de Kruskal

- `(llenaCP xs cp)` es la cola de prioridad obtenida añadiéndole a la cola de prioridad cp (cuyos elementos son ternas formadas por los dos nodos de una arista y su peso) la lista xs (cuyos elementos son ternas formadas por un nodo de una arista, su peso y el otro nodo de la arista). Por ejemplo, con `ColaDePrioridadConListas`

```
ghci> llenaCP [(3,7,5),(4,2,6),(9,3,0)] vacia
CP [(0,9,3),(5,3,7),(6,4,2)]
```

  y con `ColaDePrioridadConMonticulos`

```
ghci> llenaCP [(3,7,5),(4,2,6),(9,3,0)] vacia
CP (M (0,9,3) 1
      (M (5,3,7) 1 (M (6,4,2) 1 VacioM VacioM) VacioM) VacioM)
```

```
llenaCP :: (Ord n, Ord p, Ord c) =>
           [(n,p,c)] -> CPrioridad (c,n,p) -> CPrioridad (c,n,p)
llenaCP [] cp           = cp
llenaCP ((x,y,p):es) cp = llenaCP es (inserta (p,x,y) cp)
```

El algoritmo de Kruskal

- `(llenaCP xs cp)` es la cola de prioridad obtenida añadiéndole a la cola de prioridad `cp` (cuyos elementos son ternas formadas por los dos nodos de una arista y su peso) la lista `xs` (cuyos elementos son ternas formadas por un nodo de una arista, su peso y el otro nodo de la arista). Por ejemplo, con `ColaDePrioridadConListas`

```
ghci> llenaCP [(3,7,5),(4,2,6),(9,3,0)] vacia
CP [(0,9,3),(5,3,7),(6,4,2)]
```

  y con `ColaDePrioridadConMonticulos`

```
ghci> llenaCP [(3,7,5),(4,2,6),(9,3,0)] vacia
CP (M (0,9,3) 1
      (M (5,3,7) 1 (M (6,4,2) 1 VacioM VacioM) VacioM) VacioM)
```

```
llenaCP :: (Ord n, Ord p, Ord c) =>
           [(n,p,c)] -> CPrioridad (c,n,p) -> CPrioridad (c,n,p)
llenaCP [] cp            = cp
llenaCP ((x,y,p):es) cp = llenaCP es (inserta (p,x,y) cp)
```

El algoritmo de Kruskal

- `(raiz t n)` es la raíz de `n` en la tabla `t`. Por ejemplo,

```
> raiz (crea [(1,1),(3,1),(4,3),(5,4),(2,6),(6,6)]) 5
1
> raiz (crea [(1,1),(3,1),(4,3),(5,4),(2,6),(6,6)]) 2
6
```

```
raiz:: Eq n => Tabla n n -> n -> n
raiz t x | v == x     = v
         | otherwise = raiz t v
         where v = valor t x
```

El algoritmo de Kruskal

- (raiz t n) es la raíz de n en la tabla t. Por ejemplo,

```
> raiz (crea [(1,1),(3,1),(4,3),(5,4),(2,6),(6,6)]) 5
1
> raiz (crea [(1,1),(3,1),(4,3),(5,4),(2,6),(6,6)]) 2
6
```

```
raiz:: Eq n => Tabla n n -> n -> n
raiz t x | v == x     = v
         | otherwise = raiz t v
         where v = valor t x
```

El algoritmo de Kruskal

- (buscaActualiza a t) es el par formado por False y la tabla t, si los dos vértices de la arista a tienen la misma raíz en t y el par formado por True y la tabla obtenida añadiéndole a t la arista formada por el vértice de a de mayor raíz y la raíz del vértice de a de menor raíz. Por ejemplo,

```
ghci> let t = crea [(1,1),(2,2),(3,1),(4,1)]
ghci> buscaActualiza (2,3) t
(True,Tbl [(1,1),(2,1),(3,1),(4,1)])
ghci> buscaActualiza (3,4) t
(False,Tbl [(1,1),(2,2),(3,1),(4,1)])
```

```
buscaActualiza :: (Eq n, Ord n) => (n,n) -> Tabla n n
                                -> (Bool,Tabla n n)
buscaActualiza (x,y) t
    | x' == y'  = (False, t)
    | y' <  x'  = (True, modifica (x,y') t)
    | otherwise = (True, modifica (y,x') t)
    where x' = raiz t x
          y' = raiz t y
```

El algoritmo de Kruskal

- (buscaActualiza a t) es el par formado por False y la tabla t, si los dos vértices de la arista a tienen la misma raíz en t y el par formado por True y la tabla obtenida añadiéndole a t la arista formada por el vértice de a de mayor raíz y la raíz del vértice de a de menor raíz. Por ejemplo,

```
ghci> let t = crea [(1,1),(2,2),(3,1),(4,1)]
ghci> buscaActualiza (2,3) t
(True,Tbl [(1,1),(2,1),(3,1),(4,1)])
ghci> buscaActualiza (3,4) t
(False,Tbl [(1,1),(2,2),(3,1),(4,1)])
```

```haskell
buscaActualiza :: (Eq n, Ord n) => (n,n) -> Tabla n n
                               -> (Bool,Tabla n n)
buscaActualiza (x,y) t
    | x' == y'  = (False, t)
    | y' <  x'  = (True, modifica (x,y') t)
    | otherwise = (True, modifica (y,x') t)
    where x' = raiz t x
          y' = raiz t y
```

Tema 22: Algoritmos sobre grafos

El algoritmo de Prim

- (prim g) es el árbol de expansión mínimo del grafo g calculado mediante el algoritmo de Prim. Por ejemplo,

```
prim g1  ⇝  [(55,2,4),(34,1,3),(32,2,5),(12,1,2)]
prim g2  ⇝  [(32,2,5),(12,2,4),(13,1,2),(11,1,3)]
```

```haskell
prim :: (Num p, Ix n, Ord p) => Grafo n p -> [(p,n,n)]
prim g = prim' [n]              -- Nodos colocados
               ns               -- Nodos por colocar
               []               -- Árbol de expansión
               (aristasND g)    -- Aristas del grafo
         where (n:ns) = nodos g

prim' t [] ae as = ae
prim' t r  ae as = prim' (v':t) (delete v' r) (e:ae) as
    where e@(c,u', v') = minimum [(c,u,v)| (u,v,c) <- as,
                                            elem u t,
                                            elem v r]
```

El algoritmo de Prim

- (prim g) es el árbol de expansión mínimo del grafo g calculado mediante el algoritmo de Prim. Por ejemplo,

```
prim g1  ⇝  [(55,2,4),(34,1,3),(32,2,5),(12,1,2)]
prim g2  ⇝  [(32,2,5),(12,2,4),(13,1,2),(11,1,3)]
```

```haskell
prim :: (Num p, Ix n, Ord p) => Grafo n p -> [(p,n,n)]
prim g = prim' [n]              -- Nodos colocados
               ns               -- Nodos por colocar
               []               -- Árbol de expansión
               (aristasND g)    -- Aristas del grafo
         where (n:ns) = nodos g

prim' t [] ae as = ae
prim' t r  ae as = prim' (v':t) (delete v' r) (e:ae) as
    where e@(c,u', v') = minimum [(c,u,v)| (u,v,c) <- as,
                                           elem u t,
                                           elem v r]
```