Faculdade de Computação
Programação Funcional (BCC/BSI) - 1° Período
Aula Prática: Conjuntos, Relações e Grafos

Conjuntos

Definição para conjunto:

```
newtype Set a = SetI [a]
```

O novo tipo *Set* usa um literal, neste caso de nome *SetI*, para definir um conjunto a partir de uma lista de elementos de tipo 'a'. O construtor **newtype** tem o mesmo efeito de uma declaração **data** (para tipos algébricos), porém com construtor unário e implementada de forma mais eficiente.

A definição acima de novo tipo pode ser implementada para representar conjuntos como listas ordenadas e sem elementos repetidos. Algumas das principais funções são:

```
newtype Set a = SetI [a] deriving Show

-- Interseção entre conjuntos
inter:: Ord a => Set a ->Set a ->Set a
inter (SetI xs) (SetI ys) = SetI (ints xs ys)

ints:: Ord a =>[a]->[a]->[a]
ints [] ys = []
ints xs [] = []
ints (x:xs) (y:ys)
      | x < y  = ints xs (y:ys)
      | x == y = x: ints xs ys
      | otherwise = ints (x:xs) ys


-- Mapeamento e Filtragem para conjuntos
mapSet::Ord b=> (a->b)-> Set a-> Set b
mapSet f (SetI xs) = makeSet (map f xs)

filterSet::(a->Bool)-> Set a-> Set a
filterSet p (SetI xs) = SetI (filter p xs)

makeSet::Ord a=> [a]->Set a
makeSet xs = SetI (remDup (qsort xs))

remDup::Ord a=>[a]->[a]
remDup [] = []
remDup [x] = [x]
remDup (x:y:xs)
   | x<y = x: remDup (y:xs)
   | otherwise = remDup (y:xs)

-- Função Quicksort
qsort [] = []
qsort (x:xs) = qsort menores ++ [x] ++ qsort maiores
   where menores = [h|h<-xs, h<x]
         maiores = [h|h<-xs, h>=x]
```

```
-- Função diferença entre conjuntos
diff::Ord a => Set a -> Set a -> Set a
diff (SetI xs) (SetI ys) = SetI (remove (ints xs ys) xs)

remove::Ord a=>[a]->[a]->[a]
remove xs [] = []
remove [] ys = ys
remove (x:xs) (y:ys)
   | x==y = remove xs ys
   | otherwise = y:remove (x:xs) ys
```

Grafos:

Grafos podem ser definidos como um conjunto de pares relacionados, sendo cada par um caminho direcionado num grafo:

```
newtype Set a = SetI [a] deriving Show
type Relation a = Set (a,a)

graph1 = SetI [(1,2),(1,3),(2,4),(3,2),(3,4),(4,2)]
graph2 = SetI [(1,2),(1,3),(2,4),(3,2),(3,4),(4,2),(4,3)]
graph3 = SetI [(1,2),(2,3),(3,4),(4,5),(5,2)]
```

Algumas funções para busca em grafo:

```
-- Imagem de um elemento numa relação
image::Ord a=>Relation a-> a -> Set a
image rel val = mapSet snd (filterSet ((==val).fst) rel)

-- Busca em Profundidade
depthFirst::Ord a => Relation a -> a -> [a]
depthFirst rel v = depthSearch rel v []

depthSearch::Ord a => Relation a-> a-> [a] -> [a]
depthSearch rel v used =
   v: depthList rel (findDescs rel usedx v) usedx
   where
   usedx = v:used

depthList::Ord a=>Relation a-> [a] -> [a] -> [a]
depthList rel [] used = []
depthList rel (val:rest) used =
    next ++ depthList rel rest (used++next)
    where
    next = if (elem val used) then [] else depthSearch rel val used

findDescs::Ord a=> Relation a-> [a]->a->[a]
findDescs rel xs v = flatten (newDescs rel (makeSet xs) v)

flatten::Set a -> [a]
flatten (SetI xs) = xs

newDescs::Ord a=> Relation a -> Set a -> a -> Set a
newDescs rel st v = diff (image rel v) st
```

Faculdade de Computação
Programação Funcional (BCC/BSI) – 1° Período
Aula Prática: Conjuntos, Relações e Grafos

Exercícios:

1) Forneça os resultados para as chamadas:

```
> depthFirst grafo1 1
> depthFirst grafo2 2
```

Desenhe em seguida cada passo para a obtenção dos resultados acima.

2) Usando a função de busca como modelo, faça uma função que retorna o tamanho do menor caminho entre dois nós:

```
distancia grafo1 4 1 = 0
distancia grafo1 1 4 = 2
```