Práctica Greedy

Introducción

En este documento se va a solucionar el problema 5 de la práctica de greedy. Este es:

Dado un grafo y la definición de matching. Proponer un algoritmo greedy para intentar calcular el matching máximo del grafo.

Para solucionar este problema además se nos provee la definición de matching y matching máximo:

Dado un grafo un matching es un conjunto de aristas que no comparten vértices entre sí. De esta forma en un grafo ponderado podemos encontrar el matching máximo el cual maximiza la suma de los pesos de las aristas que lo componen.

Nosotros vamos a manejar grafos para lo cual es conveniente dar una introducción al modo en que vamos a manipular la información de vértices y aristas dentro de un *Haskell*.

Para representar los vértices utilizaremos por simple conveniencia *Integers* o números y de la misma forma lo haremos para los pesos. Con esto podemos definir los arcos como la unión de tres valores que representan ambos vértices de un arco y el peso del mismo.

De esto nuestra definición de arco se veria asi:

```
type Arco = (Integer, Integer, Integer)
```

Resolución

Para la resolución del problema se va a explicar el algoritmo usado y la técnica que se usó para aproximar una solución al problema y luego se va a presentar el código utilizado.

Al empezar pensamos en que las aristas podían ser ordenadas para así tomar primeramente las aristas que más peso aportaran al matching. Luego de ordenar las aristas necesitamos una forma de chequear si un vértice ya fue tomado y no tomarlo nuevamente para agarrar otra arista ya que esto rompería la definición de matching de un grafo.

Finalmente, la toma de aristas para maximizar el matching seguirá las siguientes reglas:

1. Si los vértices de la arista ya se encuentran incluidos continua.
2. Si los vértices no fueron incluidos, sumarlos al conjunto y sumar el peso de la arista.

Implementación

Para realizar el ordenamiento de las aristas usamos funciones auxiliares declaradas con el tipo de arco para evitar errores de tipado. Estas funciones bien conocidas implementan *mergesort*.

```
merge :: [Arco] -> [Arco] -> [Arco]
merge xs [] = xs
merge [] ys = ys
merge (x:xs) (y:ys) = if peso_x > peso_y
   then x:(merge xs (y:ys))
   else y:(merge (x:xs) ys)
       where
           (_, _, peso_x) = x
           (_, _, peso_y) = y

split :: [Arco] -> ([Arco], [Arco])
split [] = ([], [])
split [x] = ([x], [])
split (x:y:xs) = (x:a, y:b)
   where
       (a, b) = split xs

mergesort :: [Arco] -> [Arco]
mergesort [] = []
mergesort [x] = [x]
mergesort lista = merge (mergesort a) (mergesort b)
   where
       (a, b) = split lista
```

Para chequear si un vértice ya fue incluido en un conjunto, representado como un array, usamos la siguiente función:

```
esta :: Integer -> [Integer] -> Bool
esta elemento [] = False
esta elemento (x:xs) = if x == elemento
   then True
   else esta elemento xs
```

Finalmente la función para calcular el máximo matching se ve como sigue:

```
maximo_matching :: [Arco] -> Integer
maximo_matching arcos = maximizar_puntaje [] arcos_ordenados
   where
       arcos_ordenados = mergesort arcos

       maximizar_puntaje :: [Integer] -> [Arco] -> Integer
```

```
maximizar_puntaje bolsa [] = 0
maximizar_puntaje bolsa (arco:arcos) =
  if esta a bolsa || esta b bolsa
    then maximizar_puntaje bolsa arcos
    else peso + (maximizar_puntaje bolsa_actualizada arcos)
        where
            (a, b, peso) = arco
            bolsa_actualizada = a:b:bolsa
```

Podemos ver en este último *if* la toma de decisiones de la función que elige entre las dos posibilidades que definimos antes para el algoritmo. De esta forma, si una arista puede ser agregada sin que deje de ser un matching, se agrega.

Entonces esta elección de aristas sumado al orden que hace que se tomen las aristas más voluminosas primero parece una solución factible al problema planteado en la *introducción.*

Análisis del algoritmo

En el anterior párrafo dijimos que el algoritmo planteado parecía una solución al problema planteado en la *introducción* aunque esto no es cierto.

El algoritmo que hemos utilizado no se puede demostrar óptimo ya que hay un muy simple caso en el que usar un algoritmo greedy no calcula la matching máximo para el grafo. El caso es el que sigue:

En este caso nuestro algoritmo tomaría primeramente el arco que tiene un peso de 8 anulando así todas las otras posibilidades ya que los arcos restantes salen de uno de los dos vértices de este arco. En cambio, si se tomara primero el arco de peso 6 se podría tomar todavía el arco de peso 5 haciendo así un matching máximo de 11 que supera al matching de nuestro algoritmo.

Por lo tanto un enfoque greedy para este problema no nos genera la solución óptima y se debe buscar una solución que analice todas las posibles combinaciones que sean un matching para encontrar el máximo.