Heathrow, czyli ominąć korki

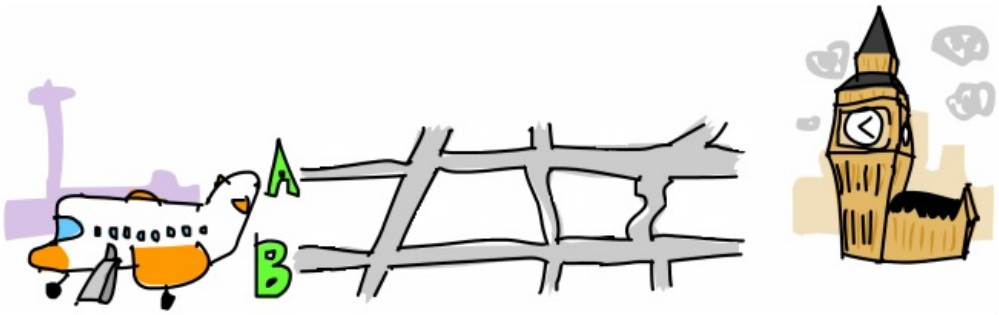
Krzysztof Konieczny, Wojciech Kura

grudzień 2020,
Politechnika Krakowska im. Tadeusza Kościuszki w Krakowie

Spis treści

1 Wstęp

Przypuśćmy, że jesteśmy w podróży biznesowej. Nasz samolot właśnie wylądował w Anglii i wypożyczamy samochód. Wkrótce mamy spotkanie i musimy jak najszybciej dostać się z lotniska Heathrow do Londynu, do którego prowadzą dwie drogi główne i przecina je kilka dróg pobocznych. Podróż z jednego skrzyżowania do drugiego zajmuje określoną ilość czasu. Zaczynamy po lewej stronie i możemy przejść na drugą drogę główną lub iść prosto.

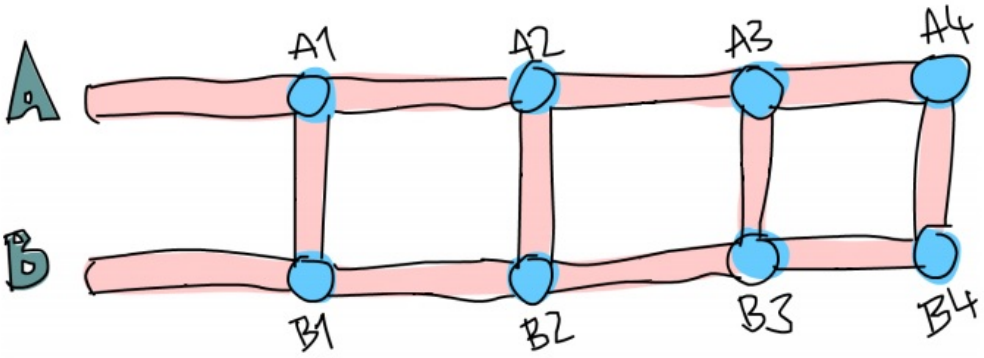


Z rysunku możemy stwierdzić że, najszybszą ścieżką z Heathrow do Londynu jest rozpoczęcie od głównej drogi A lub B jadąc cały czas prosto, lecz nie wiemy ile czasu zajmie nam przebycie każdego z kolei odcinka (liczącego od skrzyżowania do skrzyżowania) naszej drogi co mogłoby skutkować tym, że jechalibyśmy znacznie dłużej niż może się to na pierwszy rzut oka wydawać.

2 Cel pracy

Głównym zadaniem naszej pracy jest pokazanie ścieżki poprzez zaimplementowany w Haskellu program, który pobiera dane wejściowe reprezentujące system drogowy i ukazuje nam najbardziej optymalną trasę do celu w najszybszym możliwym czasie.

3 Jakie operacje będzie wykonywać program



Czy patrząc na powyższy rysunek, można "ręcznie" znaleźć najszybszą drogę z pierwszego skrzyżowania na drodze A? Od razu widzimy, że możemy jechać bezpośrednio prosto, tak samo jak w przypadku drogi B, lecz nie wiemy jak długa jest w rzeczywistości ta trasa i tylko z pozoru może być ona krótsza. Na przykład jadąc drogą B na skrzyżowaniu B1 możemy skręcić do A1 bądź jechać prosto. Problem pojawi się gdy okaże się, że droga do A1 była o wiele szybsza niż po prostu jazda bezpośrednio przez B. Stracimy wówczas sporo cennego czasu. Załóżmy więc, że skręciliśmy z B1 do A1, następnie do A2 i skręciliśmy do B2 po czym pojechaliśmy prosto drogą główną B co mogłoby się być okazać naszą optymalną trasą, lecz pamiętajmy jednak, iż nie znamy od początku czasu przejazdu przez każdy z odcinków i może być on różny, z czego jeden z nich ma 0 minut.

Zatem, nasz program:

- Szuka najlepszej drogi z punktów A i B. Jedyne opcje to jazda bezpośrednio do przodu, rozpoczynanie na przeciwnej drodze i skręt w prawo bądź lewo.
- Jeżeli natrafi na odcinek który zajmuje więcej czasu niż powinien szuka następnego, krótszego i robi to dla każdej sekcji odcinków, aż do końca.
- Otrzymamy jedną najszybszą ścieżkę na drodze A i jedną najszybszą na drodze B. Kiedy dotrzemy do końca, najszybsza z tych dwóch jest naszą szukaną, optymalną drogą.

4 Implementacja danych

System dróg przedstawimy za pomocą typów danych Haskella. Program sprawdza jednocześnie czas trwania trzech części drogi: część drogi na drodze A, jej przeciwna część na drodze B i część C, która styka się z tymi dwiema częściami i je łączy. Kiedy szukaliśmy ręcznie najszybszej drogi do B1 lub A1, skupialiśmy się tylko nad trzema pierwszymi częściami, A $\rightarrow$ A1, B $\rightarrow$ B1, A1 $\longleftrightarrow$ B1. Nazwijmy to "jedną sekcją". Taki system drogowy, użyjmy w przykładzie, który można przedstawić jako cztery sekcje:

- A $\rightarrow$ A1, B $\rightarrow$ B1, A1 $\longleftrightarrow$ B1
- A1 $\rightarrow$ A2, B1 $\rightarrow$ B2, A2 $\longleftrightarrow$ B2
- A2 $\rightarrow$ A3, B2 $\rightarrow$ B3, A3 $\longleftrightarrow$ B3
- A3 $\rightarrow$ A4, B3 $\rightarrow$ B4, A4 $\longleftrightarrow$ B4

Typ danych dla naszego systemu drogowego prezentuje się zatem jako:

```
data Section = Section {getA::Int, getB::Int, getC::Int}
    {deriving (Show)
type Path = [(Label, Int)] type RoadSystem = [Section]
```

Sekcja jest prostym typem danych algebraicznych, która przechowuje trzy liczby całkowite przez czas trwania danych trzech części przypuszczalnej trasy. Wprowadzamy również synonim typu, mówiąc, że "RoadSystem" to lista sekcji.

5 Kod programu

Przejdźmy teraz do stricte samego działania programu poprzez znajdujący się poniżej kod:

```
import Data.List

groupsOf :: Int -> [a] -> [[a]]
groupsOf 0 _ = undefined
groupsOf _ [] = []
groupsOf n xs = take n xs : groupsOf n (drop n xs)

data Section = Section {getA::Int, getB::Int, getC::Int}
    deriving (Show)

type Path = [(Label, Int)]
type RoadSystem = [Section]


data Label = A | B | C deriving (Show)


roadStep::(Path, Path) -> Section -> (Path, Path)
roadStep (pathA, pathB) (Section a b c) =
    let timeA = sum (map snd pathA)
        timeB = sum (map snd pathB)
        forwardTimeToA = timeA + a
        crossTimeToA = timeB + b + c
        forwardTimeToB = timeB + b
        crossTimeToB = timeA + a + c
        newPathToA = if forwardTimeToA <= crossTimeToA
            then (A, a):pathA
            else (C, c):(B, b):pathB
        newPathToB = if forwardTimeToB <= crossTimeToB
            then (B, b):pathB
            else (C, c):(A, a):pathA
    in (newPathToA, newPathToB)
```

```
optimalPath::RoadSystem -> Path
optimalPath roadSystem =
   let (bestAPath, bestBPath) = foldl roadStep ([], []) roadSystem
   in if sum (map snd bestAPath) <= sum (map snd bestBPath)
      then reverse bestAPath
      else reverse bestBPath

main = do
    putStrLn $ "Wpisz nazwe pliku z danymi: "
    plik <- getLine
    file <- readFile plik
    let threes = groupsOf 3 (map read $ lines file)
        roadSystem = map (\[a,b,c] -> Section a b c) threes
        path = optimalPath roadSystem
        pathString = concat $ map (show . fst) path
        pathTime = sum $ map snd path
    putStrLn $ "Najlepsza droga do obrania jest: " ++ pathString
    putStrLn $ "Czas: " ++ show pathTime
```

6 Wynik i podsumowanie

Rezultat działania naszego programu przedstawia się następująco:

```
*Main> main
Wpisz nazwe pliku z danymi:
drogi.txt
Najlepsza droga do obrania jest: BBBBC
Czas: 197

*Main> main
Wpisz nazwe pliku z danymi:
sciezki.txt
Najlepsza droga do obrania jest: BCACBBC
Czas: 75

*Main> main
Wpisz nazwe pliku z danymi:
czasy.txt
Najlepsza droga do obrania jest: AAAAAA
Czas: 53
```

Jak widzimy program znalazł najszybsze trasy biegnące z punktów A i B, są to nasze optymalne trasy spełniając tym samym nasze założenie z samego początku.

7 Bibliografia

- *Miran Lipovača* "Learn You a Haskell for Great Good!: A beginner's guide", No Starch Press, 2011