Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Кафедра информатики и прикладной математики

Дисциплина «Алгоритмы и структуры данных»
Лабораторная работа №4

Вариант 3

Выполнил:
Съестов Дмитрий Вячеславович
Группа P3217

Преподаватель:
Зинчик Александр Адольфович

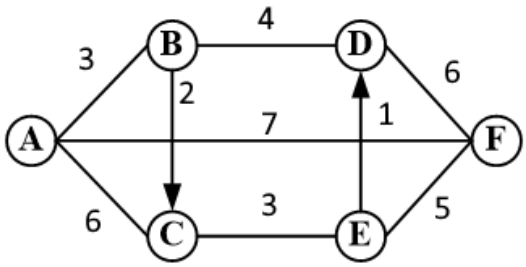
Санкт-Петербург
2018

Задание

Написать программу для заданного варианта.

1. Самостоятельно задать пропускные способности дуг и построить максимальный поток в транспортной сети.

2. Найти минимальный разрез сети и проверить справедливость теоремы Форда – Фалкерсона.



Типы данных

```
newtype Node = Node { nodeLabel :: Char } deriving Eq

instance Show Node where
      show = show . nodeLabel

data Arc = Arc { arcSrc :: Node, arcDest :: Node, arcCap :: Int } deriving Eq

instance Show Arc where
    show (Arc src dest cap) = nodeLabel src : nodeLabel dest : " " ++ show cap

data Network = Network { netSrc :: Node, netSink :: Node, netArcs :: [Arc] } deriving Show

type ArcFlow = (Arc, Int)
type AugmentingPath = [(Arc, Bool)]
```

Поиск максимального потока

```
--Наша транспортная сеть
network :: Network
network = let arcData = [('A', 'B', 3), ('A', 'C', 6), ('A', 'F', 7), ('B', 'D', 4),
                         ('B', 'C', 2), ('C', 'E', 3), ('D', 'B', 4), ('D', 'F', 6),
                         ('E', 'C', 3), ('E', 'D', 1), ('E', 'F', 5)]
             e = [Arc (Node s) (Node d) c | (s, d, c) <- arcData]
          in Network (Node 'A') (Node 'F') e

--Возвращает всех соседей узла, считая граф неориентированным.
neighbours :: Network -> Node -> [Node]
neighbours g v = let dests  = [arcDest e | e@(Arc s _ _) <- netArcs g, s == v]
                     sources = [arcSrc e | e@(Arc _ t _) <- netArcs g, t == v]
                 in dests `union` sources
```

```
--Находит все пути из истока в сток, считая граф неориентированным.
nodePaths :: Network -> [[Node]]
nodePaths g = let src = netSrc g
              in pathsFrom src [src]
    where pathsFrom v p | v == netSink g = [p]
                        | null nextNodes = []
                        | otherwise      = concatMap (\w -> pathsFrom w $ p ++ [w])
                                                     nextNodes
                        where nextNodes = neighbours g v \\ p

--Находит увеличивающие цепи, помечая направление каждой дуги.
augmentingPaths :: Network -> [AugmentingPath]
augmentingPaths g = concatMap mark $ nodePaths g
    where mark p = let arcPaths = cartesian $ zipWith arcsBetween p (tail p)
                   in map (\path -> zipWith markDirection p path) arcPaths
          markDirection v arc@(Arc s _ _) = (arc, v == s)
          arcsBetween u v = filter (\(Arc s t _) -> (s, t) == (u, v) ||
(s, t) == (v, u)) $ netArcs g

--Находит максимальный поток транспортной сети.
maximumFlow :: Network -> Int
maximumFlow g = let arcFlows = map (\x -> (x, 0)) $ netArcs g
                in maximumFlow' arcFlows (augmentingPaths g) 0
    where maximumFlow' flows paths acc =
            case find (all allowed) paths of
                Just path ->
                    let inPath x = any (fstEq x) path
                        sigma = minimum [arcCap arc - flow | (arc, flow) <- flows, inPath arc]
                    in maximumFlow' (update flows path sigma) paths (acc + sigma)
                Nothing ->
                    let minCut = minimumCut g flows
                    in assert (minCut == acc) acc
            where getFlow arc = snd . fromJust $ find (fstEq arc) flows
                  allowed (arc, True)  = getFlow arc < arcCap arc
                  allowed (arc, False) = getFlow arc > 0
          update flows path sigma = let (plusArcs, minusArcs) = mapPair fst $ partition snd path
                                        update' pair@(arc, flow) | arc `elem` plusArcs  = (arc,
                                                                   flow + sigma)
                                                                 | arc `elem` minusArcs = (arc,
                                                                   flow - sigma)
                                                                 | otherwise = pair
                                    in map update' flows

--Находит минимальный разрез транспортной сети.
minimumCut :: Network -> [ArcFlow] -> Int
minimumCut (Network src _ _) flows = sum [flow | (arc, flow) <- flows, shouldCut arc, arc
`notElem` unsaturated]
    where unsaturated = map fst . filter (\((Arc _ _ c), f) -> f < c) $ flows
          marked = findMarkedNodes src [src]
          findMarkedNodes v m | null nextNodes = [v]
                              | otherwise = v : concatMap (\w -> findMarkedNodes w $ w:m)
                                nextNodes
                              where nextNodes = [t | (Arc s t _) <- unsaturated, s == v] \\ m
          shouldCut (Arc s t _) = (s `elem` marked) /= (t `elem` marked)
```

Результат

```
*Main> maximumFlow network

13
```

Максимальный поток оказался равен минимальному разрезу (проверка в программе выделена жирным шрифтом), а следовательно, теорема Форда-Фалкерсона доказана.