НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ ТА СПЕЦІАЛІЗОВАНИХ КОМП’ЮТЕРНИХ СИСТЕМ

**Лабораторна робота №8  
з дисципліни «Вступ до функціонального програмування»**

**Варіант 11**

Виконав  
студент 3-го курсу  
групи КВ-41  
Курач Віктор

Київ – 2017

**Постановка задачі**

Написати функцію *depth-first*, яка для заданого зваженого орієнтованого графа з петлями знаходить шлях найменшої вартості між двома його довільними вершинами з використанням пошуку в глибину.

Написати функцію *breadth-first*, яка для заданого зваженого орієнтованого графа з петлями знаходить шлях найменшої вартості між двома його довільними вершинами з використанням пошуку в ширину.

Написати функції:

a) *(degree graph node)*, що визначає степінь заданої вершини графа;

b) *(nodes-list graph)*, що генерує список усіх вершин графа, відсортованих у порядку спадання їх степенів.

Граф задається у формі *((k () (0)) (m (q) (7)) (p (m q) (5 9)))*. Граф є список списків виду *(p (m q) (5 9))*, де *p* – вершина графа, *(m q)* – список вершин, у які є дуги з *p*, *(5 9)* – список ваг відповідних дуг.

**Розв’язок**

(defparameter \*graph\* '((a (a b d e) (0 1 1 3)) (b (a c d) (2 1 3))

(c (c e f) (0 1 1)) (d (c d f) (2 0 1))(e () ())(f (a c e) (2 1 1))))

(defun get-curr-node (node graph &aux res)

(dolist (x graph)

(when (equalp (car x) node)

(setq res x)))

res)

(defun depth-first (graph from to

&optional path (acc 0) min-cost res

&aux curr-node)

(setq curr-node (get-curr-node from graph))

(cond

((and (equalp (car curr-node) to)

(or (not min-cost) (< acc min-cost)))

(setq res (list (append path (list (car curr-node))) acc)))

((or (equalp (car curr-node) to) (not (cadr curr-node))

(member (car curr-node) path))

res)

(t

(mapc (lambda (node cost)

(setq res (depth-first graph node to

(append path (list (car curr-node)))

(+ acc cost) min-cost res))

(when (or (not min-cost) (< min-cost (cadr res)))

(setq min-cost (cadr res))))

(cadr curr-node) (caddr curr-node))))

res)

(defun breadth-first (graph from to

&optional (queue (list (list (car (get-curr-node from graph)) nil 0))) min-cost res

&aux curr-node)

(setq curr-node (get-curr-node from graph))

(cond

((and (equalp (car curr-node) to)

(or (not min-cost) (< (caddar queue) min-cost)))

(setq min-cost (caddar queue))

(setq res (list (append (cadar queue)

(list (car curr-node)))

(caddar queue))))

((or (equalp (car curr-node) to) (not (cadr curr-node))

(member (car curr-node) (cadar queue)))

res)

(t

(mapc (lambda (node cost)

(setq queue (append queue (list

(list node (append (cadar queue) (list from))

(+ (caddar queue) cost))))))

(cadr curr-node) (caddr curr-node))))

(setq queue (cdr queue))

(if queue

(breadth-first graph (caar queue) to queue min-cost res)

res))

(defun degree (graph node &aux (acc 0))

(setq node (get-curr-node node graph))

(dolist (x (cadr node))

(if (equalp (car node) x)

(setq acc (+ acc 2))

(setq acc (1+ acc))))

(dolist (n graph)

(when (not (equalp node n))

(dolist (x (cadr n))

(when (equalp x (car node))

(setq acc (1+ acc))))))

acc)

(defun nodes-list (graph &optional res

&aux (min-node (car graph)) (min-degree (degree graph (car min-node))))

(if (= (list-length res) (list-length graph))

res

(progn

(dolist (x graph)

(when (and (not (member (car x) res))

(<= (degree graph (car x)) min-degree))

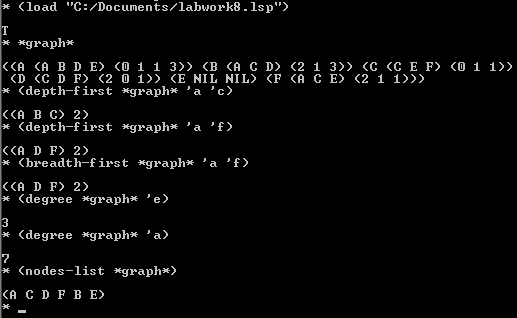
(setq min-degree (degree graph (car x)))

(setq min-node x)))

(setq res (cons (car min-node) res))

(nodes-list graph res))))

**Приклади**

****