Міністерство освіти і науки України

Національний технічний університет України

«Київський Політехнічний Інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра «Системного програмування і спеціалізованих комп’ютерних систем»

Лабораторна робота №8

з дисципліни «Вступ до функціонального програмування»   
Варіант 21

Виконав:

студент III курсу,

група КВ-41

Яковенко Максим

Перевірила:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ-2017

**Завдання**

1. Написать функцию depth-first, которая для заданного взвешенного ориентированного графа c петлями находит путь наименьшей стоимости между двумя его произвольными вершинами с использованием поиска в глубину.
2. Написать функцию breadth-first, которая для заданного взвешенного ориентированного графа c петлями находит путь наименьшей стоимости между двумя его произвольными вершинами с использованием поиска в ширину.
3. Написать функции

**a)** degree(Graph Node) определяющую степень заданной вершины графа.

**b)** nodes-list(Graph)генерирующую список всех вершин графа, отсортированных в порядке убывания их степеней.

Форма задания графа.

1. *Arc-clause form*

( (a b 7) (b c 9) (c a 5);; loop

(k k 0)) ;;standalone vertex

Т.е., граф задается перечислением его взвешенных дуг.

**Код програми:**

(defvar \*graph\*

'((a b 7) (b c 9) (c a 5) (k k 0)))

(defun get-current-node(graph node &aux res (to nil) (cost nil))

(setq res (list node))

(dolist (x graph)

(when (equalp (car x) node)

(setq to (append to (list(cadr x))))

(setq cost (append cost (list (caddr x))))

))

(append res (list to) (list cost)))

(defun get-all-node(graph &aux (res nil))

(dolist (x graph)

(progn

( if (not (numberp (position (car x) res))) (setq res (cons (car x) res)))

(if (not (numberp (position (cadr x) res))) (setq res (cons (cadr x) res)))

)) res)

(defun nodes-list(graph &aux (res nil))

(dolist(x (get-all-node graph))

(setq res (cons (list x (degree graph x)) res))

)(sort res #'(lambda (x y) (> (cadr x) (cadr y)))))

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

(dolist (x graph)

(progn

(if (equalp node (car x)) (setq ct (1+ ct)))

(if (equalp node (cadr x)) (setq ct (1+ ct)))))

ct)

(defun depth-first (graph from to &optional path (acc 0) min-cost res &aux curr-node)

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

(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-current-node graph from)) nil 0))) min-cost res

&aux curr-node)

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

(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))

Перевірка:

A 🡪 C



C 🡪 A



C 🡪 C

