Лабораторна робота №2 «Побудова на Lisp двійкового дерева рішень діагностичної бази знань”

1. Двійкові дерева рішень на Lisp

Двійкові дерева рішень можна реалізувати у мові Lisp, використовуючи його потужні можливості роботи зі списками та рекурсією. Ось приклад базової реалізації:

1.1. Визначення структури дерева

У Lisp двійкове дерево можна представити як список, де перший елемент – це вузол, а другий та третій – ліве та праве піддерева відповідно.

```
(defstruct node
  value
  left
  right)
```

1.2. Створення вузла

Функція створення вузла дерева:

```
(defun make-node (value left right)
  (make-node :value value :left left :right right))
```

1.3. Приклад дерева

Створимо просте двійкове дерево:

```
(setq tree
      (make-node 5
                 (make-node 3
                            (make-node 1 nil nil)
                            (make-node 4 nil nil))
                 (make-node 8
                            (make-node 7 nil nil)
                            (make-node 9 nil nil))))
```

1.4. Пошук у дереві

Рекурсивна функція для пошуку значення у дереві:

```
(defun search-tree (tree value)
  (when tree
```

```
    (let ((node-value (node-value tree)))
      (cond ((= value node-value) t)
            ((< value node-value) (search-tree (node-left tree)
value))
            (t (search-tree (node-right tree) value))))))
```

Приклад використання:

```
(search-tree tree 7) ; поверне T (true)
(search-tree tree 10) ; поверне NIL
```

1.5. Вставка у дерево

Функція для вставлення нового значення в дерево:

```
(defun insert-tree (tree value)
  (if (null tree)
      (make-node value nil nil)
      (let ((node-value (node-value tree)))
        (if (< value node-value)
            (setf (node-left tree) (insert-tree (node-left tree)
value))
            (setf (node-right tree) (insert-tree (node-right tree)
value)))
        tree)))
```

Приклад вставки:

```
(setq tree (insert-tree tree 6))
```

1.6. Обхід дерева

Реалізація симетричного обходу (in-order traversal):

```
(defun in-order-traversal (tree)
  (when tree
    (progn
      (in-order-traversal (node-left tree))
      (print (node-value tree))
      (in-order-traversal (node-right tree)))))
```

Приклад виклику:

```
(in-order-traversal tree)
```

Ця реалізація демонструє базові операції із двійковими деревами рішень на Lisp. Ви можете розширити її, додавши, наприклад, візуалізацію або складніші алгоритми обробки даних.

2. На основі опису предметної області діагностики захворювання з лабораторної роботи №1 побудувати двійкове дерево прийняття рішень і реалізувати його на мові Lisp.

3. Звіт з лабораторної роботи повинен мати:

- Номер варіанта та завдання;
- Текст налагодженої програми з коментарями;
- Графічне зображення двійкового дерева прийняття рішень предметної області;
- Результати роботи програми (скріншоти);
- Висновок (перераховуються які основні функції мови Lisp були використані).