利用 Lisp 教導電腦寫作

這篇文章最早以中文撰寫並在 CSDN 上發表，網址為 https://blog.csdn.net/lzw_java/article/details/11599993

大部分的程式碼和概念均基於《Ansi Common Lisp》P138~P141。

問題：給定一段英文文本，計算機如何能夠根據它生成隨機但可讀的文本？例如：

> 國家創業投資協會估計，與交易相關的財富是由規範的大量開支決定的，這些規範將花費彼此之間的主要原因是這些項目。

這是由計算機根據學習一些保羅・格雷厄姆的文章後生成的隨機文字。它從單詞「Venture」延伸到一個句子。驚人的是，這些文字通常是可以讀懂的。

演算法：記錄每個單詞後出現的單詞以及它們出現的次數。例如，如果「I leave」在原始文本中出現 5 次，「I want」出現 3 次，且「I」在任何其他單詞之前都沒有出現，然後在生成隨機文本時，當遇到「I」時，選擇「leave」作為下一個單詞的概率為 5/8。如果選擇了「leave」，則檢查在「leave」後出現的單詞並重複該過程。

現在，讓我們用 Lisp 來解決這個問題。

Lisp 的符號型別可以很好地記錄各種字串和標點符號，因此我們將使用它來記錄。我們將使用內建的 hashtable 來創建一個清單：

```
(defparameter *words* (make-hash-table :size 10000))
```

我們如何創建這個清單？

```
(let ((prev '|.|))
  (defun see (sym)
    (let ((pair (assoc sym (gethash prev *words*))))
      (if pair
      (incf (cdr pair))
      (push (cons sym 1) (gethash prev *words*))))
    (setf prev sym)))
```

當前的單詞是關鍵字，assoc 清單是該關鍵字下的值。例如，在「I」下我們有（(|leave| . 5) (|want| . 3)）。如果單詞不存在，則 push（單詞. 1）。

我們如何隨機選擇一個單詞？

```
(defun random-word (word ht)
  (let* ((choices (gethash word ht))
     (x (random (reduce #'+ choices :key #'cdr))))
```

```
    (dolist (pair choices)
      (decf x (cdr pair))
      (if (minusp x)
      (return (car pair))))))
```

這裡，reduce 函數被巧妙地使用。

現在，讓我們思考如何將給定的單詞在兩邊擴展成一個句子？

1) 反轉文本以獲取反轉的清單，即「I leave, I want」變為「leave I, want I」。
2) 反轉 hashtable 以獲取另一個 hashtable，其中後面的單詞是鍵，可能的前面單詞及其計數形成一個 assoc 清單。
3) 試試你的運氣，從標點符號開始擴展句子，直到出現給定的單詞。

我使用了第二種方法：

```
(defparameter *r-words* (make-hash-table :size 10000))

(defun push-words (w1 w2 n)
  (push (cons w2 n) (gethash w1 *r-words*)))

(defun get-reversed-words () ; a cat -> cat a
  (maphash #'(lambda (k lst)
           (dolist (pair lst)
         (push-words (car pair) k (cdr pair))))
       *words*))
```

遍歷原始 hashtable，然後將每對單詞插入到另一個 hashtable 中，反轉它們的順序。以下是自動生成雙向擴展句子的程式碼：

```
(defparameter *words* (make-hash-table :size 10000))
(defconstant maxword 100)
(defparameter nwords 0)
(defconstant debug nil)
(let ((prev '|.|))
  (defun see (sym)
    (incf nwords)
    (let ((pair (assoc sym (gethash prev *words*))))
      (if pair
      (incf (cdr pair))
      (push (cons sym 1) (gethash prev *words*))))
```

```
    (setf prev sym)))

(defun check-punc (c) ; char to symbol
  (case c
    (#\. '|.|) (#\, '|,|)
    (#\; '|;|) (#\? '|?|)
    (#\: '|:|) (#\! '|!|)))

(defun read-text (pathname)
  (with-open-file (str pathname :direction :input)
    (let ((buf (make-string maxword))
      (pos 0))
      (do ((c (read-char str nil 'eof)
          (read-char str nil 'eof)))
      ((eql c 'eof))
    (if (or (alpha-char-p c)
        (eql c #\'))
        (progn
          (setf (char buf pos) c)
          (incf pos))
        (progn
          (unless (zerop pos)
        (see (intern (subseq buf 0 pos)))
        (setf pos 0))
          (let ((punc (check-punc c)))
        (if punc
            (see punc)))))))))

(defun print-ht (ht)
  (maphash #'(lambda (k v)
        (format t "~A ~A~%" k v))
         ht))

(defparameter *r-words* (make-hash-table :size 10000))

(defun push-words (w1 w2 n)
  (push (cons w2 n) (gethash w1 *r-words*)))

(defun get-reversed-words () ; a cat -> cat a
  (maphash #'(lambda (k lst)
```

```
          (dolist (pair lst)
            (push-words (car pair) k (cdr pair))))
        *words*))

(defun print-a-word (word ht)
  (maphash #'(lambda (k lst)
          (if (eql k word)
          (format t "~A ~A~%" k lst)))
       ht))

(if debug
    (print-a-word '|leave| *r-words*))

(defun punc-p (sym) ; symbol to char, nil when fails.
  (check-punc (char (symbol-name sym) 0)))

(defun random-word (word ht)
  (let* ((choices (gethash word ht))
     (x (random (reduce #'+ choices :key #'cdr))))
    (dolist (pair choices)
      (decf x (cdr pair))
      (if (minusp x)
      (return (car pair))))))

(defun gen-former (word str)
  (let ((last (random-word word *r-words*)))
    (if (not (punc-p last))
    (progn
      (gen-former last str)
      (format str "~A " last)))))

(defun gen-latter (word str)
  (let ((next (random-word word *words*)))
    (format str "~A " next)
    (if (not (punc-p next))
        (gen-latter next str))))

;(gen-latter '|leave| t)

(defun get-a-word (ht) ; get a random word
```

```
  (let ((x (random nwords)))
    (maphash #'(lambda (k v)
         (dolist (pair v)
           (decf x (cdr pair))
           (if (minusp x)
               (return-from get-a-word (car pair)))))
         ht)))
;(get-a-word *words*)
(defun gen-sentence (word str)
  (gen-former word str)
  (format str "~A " word)
  (gen-latter word str))

(defun test ()
  (setf nwords 0)
  (read-text "essay.txt")
  (get-reversed-words)
  (let ((word (get-a-word *words*)))
    (print word)
    (gen-sentence word t)))
(test)
```