Data Structure Assignment 2 (Paper Homework)

Paper homework

(Textbook p.66-69)

請閱讀課本 2.4.2 Polynomial Representation,試著去比較兩種不同的資料結構,並且分析為何在係數(coef)不為零的情況下,第二種方法所需要的空間會是第一種方法的兩倍

(Please read section 2.4.2 Polynomial Representation in your textbook. Try to compare both types of data structures, and analyze that why the second method need's twice the space than the first one. (In the condition that there are no zero coefficients).)

General Information

- **Deadline：2015/10/23 (Please submit to TA after class)**
- **Late homework will not be accepted.**
- **Please write on A4 papers.**
- **Notice：You won't get any point if you only write the answer, please list your process and reason.**

Data Structure Assignment 2 (ProgrammingHomework)

Programming homework1

(Textbook p.84.EXERCISE2,Program2.9)

此次作業希望同學能夠以”一個陣列”來記錄 rowTerms 和 startingPos,完成矩陣的轉置,因此希望同學能夠改寫這個 function,並且能夠輸入 n*m 的矩陣,利用此 function 將此矩陣轉置成為 m*n 的矩陣(空格及換行必須遵守下列公佈形式)

(In this program, you should only use “one” array to record the “rowTerms” and “startingPos”, and rewrite the transpose matrix function in the example.)
(The input will be a n*m matrix, and you should use the function that you just rewired to transpose the matrix into a m*n matrix!)
(Spaces and line breaks should comply with the example below.)

Input:
```
3
2
5 23
-6 8
95 0
```

Output:
```
5 -6 95
23 8 0
```

Programming homework2

(Textbook p.104-106)

請同學必須要依照 ktmove1 和 ktmove2 的順序,這樣答案才會唯一,方便我們做批改
(Please follow the order of ktmove1 and ktmove2, so the answers will be unique.)

10. § [***Programming project***] Chess provides the setting for many fascinating diversions that are quite independent of the game itself. Many of these are based on the strange "L-shaped" move of the knight. A classic example is the problem of the "knight's tour," which has captured the attention of mathematicians and puzzle enthusiasts since the beginning of the eighteenth century. Briefly stated, the problem requires us to move the knight, beginning from any given square on the chessboard, successively to all 64 squares, touching each square once and only once. Usually we represent a solution by placing the numbers 0, 1, $\cdots$, 63 in the squares of the chess board to indicate the order in which the squares are reached. One of the more ingenious methods for solving the problem of the knight's tour was given by J. C. Warnsdorff in 1823. His rule stated that the knight must always move to one of the squares from which there are the fewest exits to squares not already traversed.

The goal of this programming project is to implement Warnsdorff's rule. The ensuing discussion will be easier to follow, however, if you try to construct a solution to the problem by hand, before reading any further.

The crucial decision in solving this problem concerns the data representation. Figure 2.16 shows the chess board represented as a two-dimensional array.

The eight possible moves of a knight on square (4, 2) are also shown in this figure. In general, a knight may move to one of the squares $(i-2, j+1)$, $(i-1, j+2)$, $(i+1, j+2)$, $(i+2, j+1)$, $(i+2, j-1)$, $(i+1, j-2)$, $(i-1, j-2)$, $(i-2, j-1)$. However, notice that if (i, j) is located near one of the board's edges, some of these possibilities could move the knight off the board, and, of course, this is not permitted. We can represent easily the eight possible knight moves by two arrays *ktmove* 1 and *ktmove* 2 as:

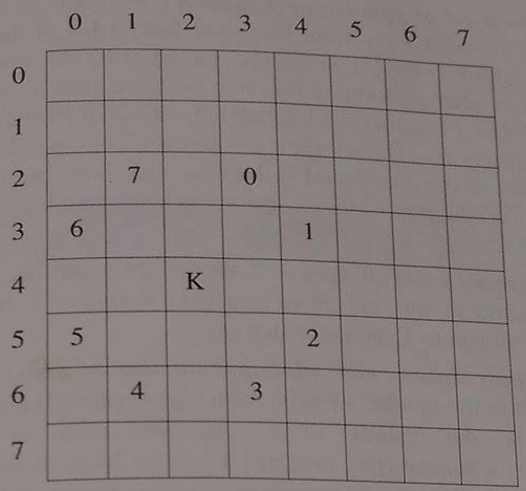

Figure 2.16: Legal moves for a knight

ktmove 1	*ktmove* 2
−2	1
−1	2
1	2
2	1
2	−1
1	−2
−1	−2
−2	−1

Then a knight at (i, j) may move to $(i + ktmove1[k], j + ktmove2[k])$, where k is some value between 0 and 7, provided that the new square lies on the chess board. Below is a description of an algorithm for solving the knight's tour problem using Warnsdorff's rule. The data representation discussed in the previous section is assumed.

(a) [***Initialize chessboard***] For $0 \le i, j \le 7$ set $board[i][j]$ to 0.

(b) [*Set starting position*] Read and print (i, j) and then set $board[i][j]$ to 0.

(c) [*Loop*] For $1 \leq m \leq 63$, do steps (d) through (g).

(d) [***Form a set of possible next squares***] Test each of the eight squares one knight's move away from (i, j) and form a list of the possibilities for the next square $(nexti[l], nextj[l])$. Let *npos* be the number of possibilities. (That is, after performing this step we have $nexti[l] = i + ktmove1[k]$ and $nextj[l] = j + ktmove2[k]$, for certain values of k between 0 and 7. Some of the squares $(i + ktmove1[k], j + ktmove2[k])$ may be impossible because they lie off the chessboard or because they have been occupied previously by the knight, that is, they contain a nonzero number. In every case we will have $0 \leq npos \leq 8$.)

(e) [***Test special cases***] If $npos = 0$, the knight's tour has come to a premature end; report failure and go to step (h). If $npos = 1$, there is only one next move; set *min* to 1 and go to step (g).

(f) [***Find next square with minimum number of exits***] For $1 \leq l \leq npos$, set $exits[l]$ to the number of exits from square $(nexti[l], nextj[l])$. That is, for each of the values of l, examine each of the next squares $(nexti[l] + ktmove1[k], nextj[l] + ktmove2[k])$ to see if it is an exit from $(nexti[l], nextj[l])$, and count the number of such exits in $exits[l]$. (Recall that a square is an exit if it lies on the chessboard and has not been occupied previously by the knight.) Finally, set *min* to the location of the minimum value of exits. (If there is more than one occurrence of the minimum value, let *min* denote the first such occurrence. Although this does not guarantee a solution, the chances of completing the tour are very good.)

(g) [***Move knight***] Set $i = nexti[min]$, $j = nextj[min]$, and $board[i][j] = m$. Thus, (i, j) denotes the new position of the knight, and $board[i][j]$ records the move in proper sequence.

(h) [***Print***] Print out the board showing the solution to the knight's tour, and then terminate the algorithm.

Write a C program that corresponds to the algorithm. This exercise was contributed by Legenhausen and Rebman.

10. §【程式計畫】西洋棋本身有許多種與本身下法無關的玩法，其中有很多玩法是根據騎士的 L 型移動。騎士巡邏（knight's tour）的問題便是一個最典型的例子。在十八世紀初，這個問題引起了眾多數學家與拼圖迷的關注。簡單來說，這個問題描述的是一個騎士從棋盤上的任何一個格子出發，陸續地走遍棋盤中的所有 64 個格子，每一個格子並且剛好走過一遍。為了方便起見，我們可以將棋盤上的 64 個格子依照騎士經過的順序依序編號為 0、1、…、63。騎士巡邏問題最巧妙的解法之一由 J. C. Warnsdorff 在 1823 年提出。他的規則是，騎士得永遠選擇移動到出口數最少的格子，其中所謂的一個格子的出口數表示從該格子所能進入的未被巡邏過之格子數。

這個程式計畫的目標是實行 Warnsdorff 的規則。然而，在繼續下面的閱讀前，你可以試著自己完成這個問題的答案。接下來的討論會比較容易領會。

解決這個問題的一個重要決定，在於資料的表示法。圖 2.16 把棋盤當成一個二維陣列顯示。

從這一張圖可以看到位於格子 $(4, 2)$ 內的騎士的八種可能移動方向。一般來說，一個位於 (i, j) 的騎士可以移動到 $(i-2, j+1)$、$(i-1, j+2)$、$(i+1, j+2)$、$(i+2, j+1)$、$(i+2, j-1)$、$(i+1, j-2)$、$(i-1, j-2)$、$(i-2, j-1)$ 這八個格子中的一個。但是要注意的是，如果 (i, j) 的位置很靠近棋盤的邊緣，有一些方向的移動可能會使得騎士超出棋盤外，當然這是不允許的。我們將騎士可能移動的八種方向用兩個陣列來表示，分別是 *ktmove*1 與 *ktmove*2：

ktmoev1	*ktmove2*
−2	1
−1	2
1	2
2	1
2	−1
1	−2
−1	−2
−2	−1

這麼一來，一個在位置 (i, j) 的騎士可以移動到 $(i + ktmove1[k], j + ktmove2[k])$，其中 k 是一個介於 0 到 7 的整數。當然，新格子必須在棋盤上才算數。以下描述的是利用 Warnsdoff 規則來解騎士巡邏問題的演算法。假設使用前面章節討論的資料表示法。

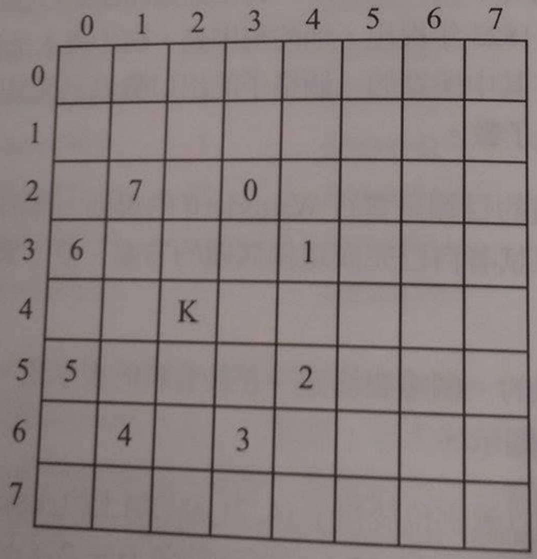

圖 2.16：騎士合法移動表示

(a)【棋盤之初始化】設定 $board[i][j] = 0, 0 \le i, j \le 7$。

(b)【設定起始位置】讀取並且輸出 (i, j)，然後設定 $board[i][j] = 0$。

(c)【迴圈】當 $1 \le m \le 63$ 時，執行步驟 (d) 到步驟 (g)。

(d)【組成下一步可能格子的集合】測試從 (i, j) 出發可能到達的八個格子並且組成下一步可巡邏的格子清單 ($nexti[l]$, $nextj[l]$)。令 *npos* 表示可以巡邏的格子數。(換句話說，在執行完這一個步驟後，我們得到 $nexti[l] = i + ktmove1[k]$ 與 $nextj[l] = j + ktmove2[k]$，其中 k 是介於 0 到 7 的整數。有些新格子 $(i + ktmove1[k], j + ktmove2[k])$ 可能因為它們超出了棋盤範圍或者是先前已經被騎士巡邏過了（即，這些格子的數字不為零），因此不予以列入可巡邏的格子清單內。在每一種情況下，我們總是可以得到 $0 \le npos \le 8$。)

(e)【測試特殊個案】如果 $npos = 0$，騎士巡邏將會提早結束；回報失敗並且跳到步驟 (h)。如果 $npos = 1$，那麼表示只有一個位置可以巡邏；設定 $min = 1$ 並且跳到步驟 (g)。

(f)【找出下一個出口數最少的格子】當 $1 \le l \le npos$，設定 $exits[l]$ 為格子 ($nexti[l]$, $nextj[l]$) 的出口數。換句話說，針對每一個 l 值檢查每一個可能的下一步巡邏點 ($nexti[l] + ktmove1[k]$, $nextj[l] + ktmove2[k]$) 看它是否是 ($nexti[l]$, $nextj[l]$) 的一個出口，並且計算這些出口的數目 $exits[l]$。（提醒一下讀者：如果一個新格子沒有跑出棋盤外，之前也沒被巡邏過，那麼它就是一個出口。）最後，把 *min* 的值設為出口數最小的位置。(出口數最小的可能不只一個。如果這種情形真的發生了，我們可以將 *min* 的值直接選為第一個出口數最小的位置，雖然這麼做我們將無法保證一定找得到解答。但是，用這個方法找到完整的騎士路徑的機率是相當高的。)

(g)【移動騎士】設定 $i = nexti[min]$，$j = nextj[min]$，以及 $board[i][j] = m$。因此，(i, j) 表示騎士的新位置，而陣列 $board[i][j]$ 則記錄移動的順序。

(h)【輸出】輸出陣列 *board* 以顯示騎士路徑的解答並且結束演算法。

寫一個對應到此演算法的 C 程式。這個習題是由 Legenhausen 和 Rebman 提供的。

General Information:

- Deadline：2015/10/30 23:59.
- Upload your assignment to Moodle system.
- Upload file format: Student-Id_Name.rar , Ex.P76991094_王小明.rar
- Your file should consist of the following items:
- Source Code
- Readme file (Program description)
- Late homework will not be accepted.   Any copies will be scored as zero.

Do not plagiarize!!!

大家加油