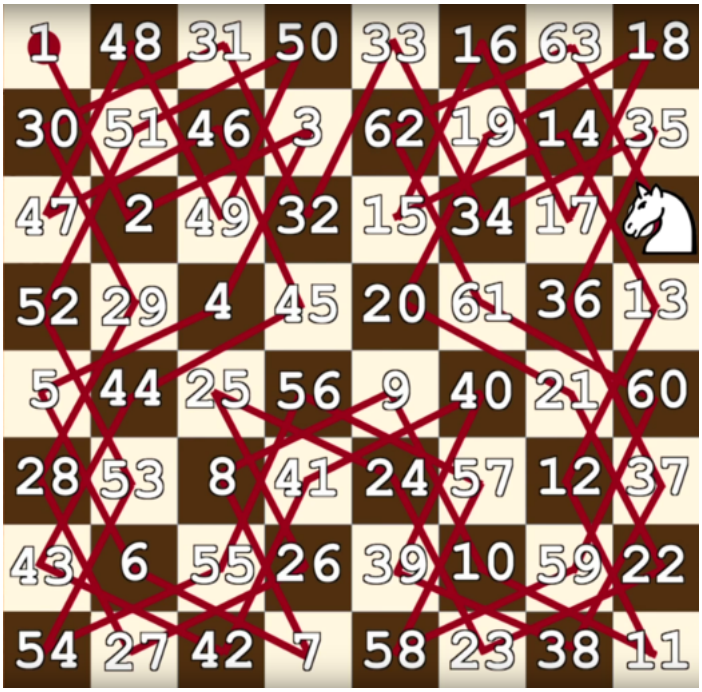
CS 7A - Fall 2015 - CS007A: Knight's Path Problems Part 2. Due 11/25/15

Background Theory

Now that you have (it is hoped) developed working code for exploring knight's tours using a human decision-maker, we want to explore further by automating this process using various heuristics and also examine the donut topology.



1 Problems

Adapt the knight's tour program to

1. Use the accessibility heuristic of going to the least accessible squares first (choosing randomly in the case of a tie) to automate the search for a knight's tour.

```
int access[8][8] = {{2,3,4,4,4,4,3,2},
                    {3,4,6,6,6,6,4,3}
                    {4,6,8,8,8,8,6,4},
                    {4,6,8,8,8,8,6,4},
                    {4,6,8,8,8,8,6,4},
                    {4,6,8,8,8,8,6,4},
                    {3,4,6,6,6,6,4,3},
                    {2,3,4,4,4,4,3,2}};
```

   Allow the user to choose the number of rows and columns of the "chess" board.

   Here is some starter code that may or may not be helpful:

```
1
/// GH working on knights tour 20151015
3
/**
5  1 14  9 22  3
```

```
 18 23  2 15 10
13  8 17  4 21
24 19  6 11 16
 7 12  0 20  5

 1 10 21 16  7
20 15  8 11 22
 9  2 23  6 17
14 19  4  0 12
 3 24 13 18  5

 1 22  3 18 25 30 13 16
 4 19 24 29 14 17 34 31
23  2 21 26 35 32 15 12
20  5 56 49 28 41 36 33
57 50 27 42 61 54 11 40
 6 43 60 55 48 39  0 37
51 58 45  8 53 62 47 10
44  7 52 59 46  9 38 63

 1 20  3 16 23 28 33 14 39
 4 17 22 27 34 15 38 29 32
21  2 19 24 55 36 31 40 13
18  5 76 35 26 41 54 37 30
77 62 25 56 75 72 43 12 47
 6 57 74 79 42 53 46 71 44
63 78 61 52 73 70 67 48 11
58  7 80 65 60  9 50 45 68
 0 64 59  8 51 66 69 10 49

 1 32  3 22 29 48 35 20 39 42
 4 23 30 49 34 21 44 41 36 19
31  2 33 28 47 62 79 38 43 40
24  5 50 63 78 45 60 89 18 37
51 64 27 46 61 90 77 80 59 88
 6 25 66 85 70 81 94 87 92 17
65 52 69 26 95 86 91 76 99 58
10  7 84 67 82 71  0 93 16 75
53 68  9 12 55 96 73 14 57 98
 8 11 54 83 72 13 56 97 74 15

 1 22  3 28  25  20  57  30  35  18  55
 4 27 24 21  58  29  36  19  56  31  34
23  2 61 26  37  74  59  68  33  54  17
62  5 38 79  60  67  72  75  52  69  32
39 80 63 66  73  92  77  70  99  16  53
 6 65 82 91  78  71  98 115  76  51 100
81 40 89 64  97 112  93 108 101 114  15
44  7 96 83  90 107 116 113 120 103  50
41 84 43 88 111  94 109 102 117  14 119
 8 45 86 95  10  47 106   0  12  49 104
85 42  9 46  87 110  11  48 105 118  13
```

```
  1  24   3  30  27  60  21  32  73  36  19  34

  4  29  26  59  22  31  72  63  20  33  76  37

 25   2  23  28  71  64  61 100  77  74  35  18

 56   5  70  65  58 105  78 121  62 101  38  75

 69  66  57 106  79 124 109 104  99 120  17 102

  6  55  68 125 110 107 122 139 130 103  98  39

 67  82 111  80 123 140 129 108 119  40 131  16

 54   7 126 135 128 117 138 133 142  97  92  41

 83 112  81 116 137 134 141 118  93 132  15  90

  8  53 136 127  86  49  94 143  96  91  42  45

113  84  51  10 115   0  87  12  47  44  89  14

 52   9 114  85  50  11  48  95  88  13  46  43


  1  28   3  24  31  36  55  22  39  52  43  20  41

  4  25  30  35  56  23  38  59  54  21  40  51  44

 29   2  27  32  37  68  57  80  65  60  53  42  19

 26   5  74  69  34  79 104  67  58  81  64  45  50

 73  70  33  78 105 120 109  82 103  66  61  18  63

  6  75  72 141 110 107 112 121 118  83 102  49  46

 71 142  77 106 137 140 119 108 113 122  47  62  17

 76   7 144 139 154 111 136 123  94 117  84 101  48

143 148 155 164 145 138 153 116 131 114  95  16  85

  8 165 146 159 152 163 132 135 124  93 100  89  96

147 156 149 162 133 160 125 130 115 128  97  86  15

166   9 158 151 168  11 134 127  92  13  88  99  90

157 150 167  10 161 126   0  12 129  98  91  14  87
```

```
  1  34   3  30  37  42  47  28  45  74  89  26  69  72
  4  31  36  41  48  29  44  75  94  27  70  73  88  25
 35   2  33  38  43  76  95  46  83  90 107 130  71  68
 32   5  52  77  40  49  82  93 106 129  84  87  24 131
 53  78  39  50  81  96 105 128  91 108 135 132  67  86
  6  51  80  97 104 127  92 109 158 133 148  85 136  23
 79  54 103 116  99 110 179 126 173 150 159 134 147  66
102   7  98 111 182 125 172 175 180 157 168 149  22 137
 55 112 117 100 115 190 181 178 171 174 151 160  65 146
  8 101 114 191 124 183 176 185 188 167 156 169 138  21
113  56 123 118 195 186 189 166 177 170 161 152 145  64
 12   9   0 121 192 119 184 187 164 155 144 141  20 139
 57 122  11  14  59 194 165  16  61 162 153  18  63 142
 10  13  58 193 120  15  60 163 154  17  62 143 140  19

  1  30   3  36  33  28  85  38  47  26  75  40  45  24  73
  4  35  32  29  86  37  48  27  84  39  46  25  74  41  44
 31   2  89  34  49 104  87 114 107  76  83  80  43  72  23
 90   5  50 103  88 113 108 105  82 145 140  77  70  79  42
 51 100  91 112 109 102 115 158 139 106  81 146 151  22  71
  6  93 110 101 116 159 138 123 144 195 154 141  78  69 152
 99  52 117  92 111 122 161 196 157 142 147 194 153 150  21
 94   7  98 121 160 137 124 143 198 201 208 155 148 185  68
 53 118  95 136 125 162 197 176 207 156 199 202 193  20 149
  8  97 120 127 134 175 168 215 200 209 206 213 186  67 184
119  54 135  96 163 126 177 210 217 214 219 192 203 188  19
 58   9 128 133 174 169 216 167 220 205 212 187 224 183  66
```

```
 55 130  57 164 171 166 173 178 211 218   0 204 191  18 189

 10  59 132 129  12  61 170 221  14  63 180 223  16  65 182

131  56  11  60 165 172  13  62 179 222  15  64 181 190  17


  1  32   3  38  35  42  29  40  77  94  27  74 111  70  25  72

  4  37  34  43  30  39  78  93  28  75 110 115  26  73 112  69

 33   2  31  36  79  92  41  76 109 132  95 120 113 136  71  24

 46   5  80  89  44  99 108 131  96 121 114 143 116 119  68 137

 81  86  45 100  91  88  97 124 159 144 133 122 135 142  23 118

  6  47  90  87  98 125 160 107 130 123 158 141 146 117 138  67

 85  82 101 126 161 106 129 164 217 204 145 134 157 140 147  22

 48   7  84 105 128 163 216 207 186 165 156 203 148 151  66 139

 83 102 127 162 215 208 187 218 235 250 205 166 155 168  21 150

  8  49 104 209 188 219 214 249 206 185 234 183 202 149 152  65

103 210 189 220 213 238 225 236 251 248 255 176 167 154 169  20

 50   9 212 227 224 231 252 247   0 233 184 201 182 177  64 153

211 190 221 194 239 226 237 232 245 254 243 178 175 170  19  60

 10  51 228 223 230 195 246 253 242 179 198 181 200  61 172  63

191 222  53  12 193 240  55  14 197 244  57  16 171 174  59  18

 52  11 192 229  54  13 196 241  56  15 180 199  58  17  62 173


  1  36   3  32  39  44  49  30  47 104 123  28 101 138 143  26  99

  4  33  38  43  50  31  46 105 122  29 102 137 144  27 100 139 142

 37   2  35  40  45 106 121  48 103 136 145 124 159 154 141  98  25

 34   5  54 107  42  51 112 135 120 125 176 153 146 165 160 157 140

 55 108  41  52 111 128 119 126 177 152 147 166 175 158 155  24  97

  6  53 110 129 118 113 134 151 148 189 174 179 164 167 172 161 156
```

```
 109  56 117 114 133 150 127 190 193 178 183 188 173 180 163  96  23
 116   7 132 197 130 209 194 149 204 191 242 181 184 187 168 171 162
  57 198 115 208 195 206 203 192 241 182 251 186 255 170 227  22  95
   8 213 196 131 210 269 240 205 264 259 254 243 252 185 256 169 226
 199  58 211 234 207 202 275 270 267 250 263 260 257 228 245  94  21
 212   9 214 201 276 239 268 265 274 279 258 253 244 261  92 225  84
  59 200 235 238 233 284 281 278 271 266 249 262 229 246  85  20  93
  10 215  66 285 236 277 232 283 280 273 230 247  86  91 224  83  74
  63  60 237 218  67 282 287 272 231 248  87  90 223  82  75  78  19
 216  11  62  65 286  13 220  69 288  15 222  71  88  17  80  73  76
  61  64 217  12 219  68   0  14 221  70  89  16  81  72  77  18  79


   1  42   3  38  45  50  55  36  53 102 121  34  99 136  85  32  83  94
   4  39  44  49  56  37  52 103 120  35 100 135 142  33  98  93  86  31
  43   2  41  46  51 104 119  54 101 134 143 122 137  92 141  84  95  82
  40   5  60 105  48  57 110 133 118 123 166 151 144 153 138  97  30  87
  61 106  47  58 109 126 117 124 167 150 145 162 183 140  91 154  81  96
   6  59 108 127 116 111 132 149 146 165 182 187 152 161 184 139  88  29
 107  62 115 112 131 148 125 168 181 188 163 208 185 196 155  90 157  80
 114   7 130 177 128 169 180 147 164 209 186 195 200 207 160 197  28  89
  63 176 113 170 179 192 293 210 189 194 265 206 225 198 201 156  79 158
   8 171 178 129 298 211 190 193 300 275 224 199 264 205 252 159 202  27
 175  64 297 212 191 292 299 294 223 266 287 276 269 226 263 204 251  78
 172   9 174 307 296 303 222 319 286 301 274 267 272 279 270 253  26 203
  65 308 213 304 291 320 295 302 283 288 285 280 277 268 227 262  77 250
  10 173 310 315 306 221 318 289 322 239 282 273 246 271 278 249 254  25
 309  66 305 214 317 290 321 220 243 284 245 234 281 248 255 228 261  76
```

```
  14  11 316 311 314 215 242 323 240 219 238 247 236 233 260 257  24 229

  67 312  13  16  69   0 217  18  71 244 235  20  73 256 231  22  75 258

  12  15  68 313 216  17  70 241 218  19  72 237 232  21  74 259 230  23
**/
#include  "..\std_lib_facilities.h"
#include <fstream>
#include <windows.h>  //for colors

///global variable for board width and length
int bWidth{8}, bLength{8};
bool debug = false;

//fixed array of possible moves
int movArr[8][2] = {{-1, 2},{-2, 1},{-2,-1},{-1,-2},
                    { 1,-2},{ 2,-1},{ 2, 1},{ 1, 2}};
/*******************************************/
///prototypes
void ClearScreen();

void displayBoard(vector <int>);

/// choose a legal move and update
void getMove(vector<int>& brd,
             int& mve,
             int& currPos,
             vector<int>& access,
             vector<string> poem,
             vector<string>& sPoem);

///initialize the board and set first position
void start(vector<int>&, int&, vector<int>&);

///set up the access matrix
void setup_access(vector<int>&);

///check to see if stuck
bool stuck(vector<int> brd, int pos);

/// get the index of the first accessible square with the lowest accessibility nu
/// searching counter-clockwise from (-1,2)
int getMinAccess(vector<int> accBrd, vector<int> brd, int pos);

/// update accessibility matrix
void updateAcc(vector<int>& accBrd, int pos);

bool donut{false};

int main() {
    string poemName, word;
    vector<string> poem, sPoem;
    cout << "\nEnter␣the␣file␣name␣for␣your␣poem:␣\n";
```

```
    cin >> poemName;
325 ifstream inFile(poemName);
    int i{0};
327 while(inFile>>word) {
        poem.push_back(word);
329     ++i;
    }
331 sPoem.push_back(poem[0]);
    cout << "\nThere␣are␣" << i << "␣=␣" << poem.size() << "␣words." << endl;
333 cout << "\nEnter␣the␣width␣and␣length␣of␣your␣board:␣";
    cin >> bWidth >> bLength;
335 cout << "\nDo␣you␣want␣to␣use␣donut␣topology?␣";
    cin >> donut;
337 vector<int> board(bWidth*bLength);  // the board of moves
    int mve{1}, currPos{0};
339 if(donut) vector<int> access(bWdith*bLength,8);
    else {
341     vector<int> access(bWidth*bLength); // accessibility board
        setup_access(access);
343 }
    if(debug) {
345     cout << "\nIn␣doubt␣about␣access:␣" << endl;
        displayBoard(access);
347     cin.get();
    }
349
        ///get first position and update accessibility matrix
351 start(board , currPos, access);

353     ///game loop
    displayBoard(board);
355 while(!stuck(board, currPos) && mve << bWidth*bLength-1) {
        ClearScreen();
357     getMove(board, currPos, mve, access, poem, sPoem);
        displayBoard(board);
359     //cout << "\n" << sPoem[sPoem.size()-1];
        //cin.get();
361 }

363 /// print out scrambled poem
    cout << "\n\nThe␣scrambled␣poem␣is\n\n";
365 int j = 0;
    //for(string s: poem) {
367 for(int j = 0; j < sPoem.size(); ++j) {
        cout << sPoem[j] << "␣";
369     if((j+1)%10 == 0) cout << endl;
    }
371
        cout << "\n\nHere␣is␣the␣poem␣unscrambled:␣\n\n";
373 /// to do: write code to unscramble the poem
    return 0;
375 }
```

```
377 ///initialize the board and set first position
    void start(vector<int>& brd, int& cp, vector<int>& access) {
379     int row , col;
        cout  << "\nWhat␣row␣and␣column␣position␣do␣you␣want␣to␣start␣?";
381     cin  >> row  >> col;
        cout << "\n␣That's␣at␣" << row*bWidth+col << endl;
383     cp = row*bWidth+col;
        brd[cp] = 1;
385     updateAcc(access, cp);
        ///debugging
387     if(debug) displayBoard(access);
    }
389
    void displayBoard(vector<int> board)  {
391     for(int i = 0; i < board.size(); ++i) {
            cout << setw(4) << board[i];
393         if((i+1)%bWidth == 0) cout << "\n\n";
        }
395 }

397 void getMove(vector<int>& brd,
                 int& currPos,
399              int& mve,
                 vector<int>& access,
401              vector<string> poem,
                 vector<string>& sPoem) {
403     /// update current position

405     /// update access vector
        updateAcc(access, currPos);
407     ++mve;
        brd[currPos] = mve;
409
        ///debugging
411     if(debug) displayBoard(access);
    }
413
    ///set up the access matrix
415 void setup_access(vector<int>& a) {
        int cntr{0};
417     ///for each row
        for(int i = 0; i < bLength; ++i) {
419         ///for each column
            for(int j = 0; j < bWidth; ++j) {
421             cntr = 0;
                if(i-1>=0       && j+2<bWidth) cntr++;
423             if(i-2>=0       && j+1<bWidth) cntr++;
                if(i-2>=0       && j-1>=0    ) cntr++;
425             if(i-1>=0       && j-2>=0    ) cntr++;
                if(i+1<bLength && j-2>=0    ) cntr++;
427             if(i+2<bLength && j-1>=0    ) cntr++;
                if(i+2<bLength && j+1<bWidth) cntr++;
429             if(i+1<bLength && j+2<bWidth) cntr++;
```

```
            a[i*bWidth+j]=cntr;
        } /// end for loop
    } /// end for loop
    ///debugging for check
    for(int i = 0; i < bWidth*bLength; ++i) {
        cout << a[i];
        if((i+1)%bWidth==0) cout << endl;
    }
}

///check to see if stuck
bool stuck(vector<int> brd, int pos) {
    /// write code that will return true if the knight has nowhere to go
        /// and return false if there is a move possible
}

// return the position number (between 0 and bWidth*bLength-1)
int getMinAccess(vector<int> accBrd, vector<int> brd, int pos) {
    /// find first legal move
    if(!donut) {
                /// write code to find the square with min access with rectangle
                /// and return the index of that square
    }
    else // donut mode
    ///upper left corner
    if(pos==0) {
        mnmm = bWidth*(bLength-1) + 2; //22
        if(accBrd[bWidth*(bLength-2)+1]< accBrd[mnmm])
            mnmm = bWidth*(bLength-2)+1; //16
        if(accBrd[bWdith*(bLength-1)-1]<bWidth[mnmm])
            mnmm = bWidth*bLength-2; //19
        if(accBrd[bWdith*bLength-2]<bWidth[mnmm])
            mnmm = bWidth*bLength-2; //23
        if(accBrd[2*bWdith-2]<bWidth[mnmm])
            mnmm = 2*bWidth-2; //8
        if(accBrd[3*bWidth-1]<bWidth[mnmm])
            mnmm = 3*bWidth-1; //14
        if(accBrd[2*bWidth+1]<bWidth[mnmm])
            mnmm = 2*bWidth+1;  //11
        if(accBrd[bWidth+2]<bWidth[mnmm])
            mnmm = bWidth+2;  //7
        return mnmm;
    }
    ///upper left corner+1
    ///upper right corner
    ///lower right corner
    ///lower left corner
        /// and so on, lots of special cases around the corner
}


/// update accessibility matrix
void updateAcc(vector<int>& access, int currPos) {
```

```
483      /// write code to update access vector
}
485
void ClearScreen() {
487     HANDLE                      hStdOut;
        CONSOLE_SCREEN_BUFFER_INFO  csbi;
489     DWORD                       count;
        DWORD                       cellCount;
491     COORD                       homeCoords = { 0, 0 };

493     hStdOut = GetStdHandle( STD_OUTPUT_HANDLE );
        if (hStdOut == INVALID_HANDLE_VALUE) return;
495
        /* Get the number of cells in the current buffer */
497     if (!GetConsoleScreenBufferInfo( hStdOut, &csbi )) return;
        cellCount = csbi.dwSize.X *csbi.dwSize.Y;
499
        /* Fill the entire buffer with spaces */
501     if (!FillConsoleOutputCharacter(
            hStdOut,
503         (TCHAR) ' ',
            cellCount,
505         homeCoords,
            &count))  return;
507
        /* Fill the entire buffer with the current colors and attributes */
509     if (!FillConsoleOutputAttribute(
            hStdOut,
511         csbi.wAttributes,
            cellCount,
513         homeCoords,
            &count)) return;
515
        /* Move the cursor home */
517     SetConsoleCursorPosition( hStdOut, homeCoords );
}
```

2. Allow the user to play on the donut topology so that, initially, on a board which is at least 5 by 5, each square has accessibility number 8.