The Knight's Tour Pseudo code

Evan Dreher

September 2023

1 Data Structures

1.1 The Square

Class Square
    $rank$: `Int` ▷ `Equivalent to row`
    $file$: `Int` ▷ `Equivalent to column`
end Class

1.2 The Tour

Class Tour
    $Board$: `Int[][]`
    $Steps$: `Int`
    $Path$: `List[Square]`
    $Solved$: `Bool`
    $Ranks$: `Int`
    $Files$: `Int`
    $Size$: `Int` $=$ Rank $\times$ Files
end Class

2 Algorithms

Algorithm 1 Find Legal Moves

This: Tour Object

Input: rank: Int, file: Int

Output: All legal moves that can be made from the square Board[x][y] represented as a list of Square Objects

```
function FindMoves(rank, file)
    Legal_Moves = List[Square]
    if rank + 2 ≤ Ranks then
        if file + 1 ≤ Files then
            add Square(rank + 2, file + 1) to Legal_Moves
        if file − 1 ≥ 1 then
            add Square(rank + 2, file − 1) to Legal_Moves
    if rank − 2 ≥ 1 then
        if file + 1 ≤ Files then
            add Square(rank − 2, file + 1) to Legal_Moves
        if file − 1 ≥ 1 then
            add Square(rank − 2, file − 1) to Legal_Moves
    if rank + 1 ≤ Ranks then
        if file + 2 ≤ Files then
            add Square(rank + 1, file + 2) to Legal_Moves
        if file − 2 ≥ 1 then
            add Square(rank + 1, file − 2) to Legal_Moves
    if rank − 1 ≥ 1 then
        if file + 2 ≤ Files then
            add Square(rank − 1, file + 2) to Legal_Moves
        if file − 2 ≥ 1 then
            add Square(rank − 1, file − 2) to Legal_Moves
    return Legal_Moves
```

Algorithm 2 Sort New Moves

This: Tour Object

Input: rank: Int, file: Int

Output: All legal moves that can be made from the square Board[x][y] represented as a list of Square objects sorted in decreasing order by the number of continuing moves

```
function OrderedMoves(rank, file)
    Moves = FindMoves(rank, file)                                  ▷ See Algorithm 1
    Branches = Int[size of Moves]
    for i → 1 to size of Branches do
        Branches[i] = size of FindMoves(Moves[i].rank, Moves[i].file)
    Counts = Int[8]
    for i → 1 to 8 do
        Counts[i] = 0
    for i → Branches do
        Counts[i] = Counts[i] + 1
    for i → Branches do
        Counts[i] = Counts[i] + 1
    for i → 2 to 8 do
        Counts[i] = Counts[i] + Counts[i − 1]
    Output = Square[size of Moves]
    for i →  size of Moves down to 1 do
        Output[Counts[Branches[i]]] = Moves[i]
        Counts[Branches[i]] = Counts[Branches[i]] − 1
    return Output
```

Algorithm 3 Backtracking Algorithm

This: Tour Object

Input: rank: Int, file: Int

Output: Void

```
procedure Tour(rank, file)
    Board[rank][file] = Steps
    Steps = Steps + 1
    add Square(rank, file) to Path
    Moves = OrderedMoves(rank, file)                               ▷ See Algorithm 2
    for Move → Moves do
        if Board[Move.rank][Move.file] == 0 then
            Tour(Move.rank, Move.file)
    if Steps == Size then
        return null
    Board[rank][file] = 0
    Steps = Steps − 1
    Pop last element from Path
```

Algorithm 4 Solve for a Structured Tour

```
This: Tour Object
Input: rank: Int, file: Int
Output: Void

procedure StructuredKnightTour(rank, file)
    if (rank == 1 and file == 2) or (rank == 2 and file == 1) then
        if steps ≠ 2 and steps ≠ Size − 1 then
            return
    if Board[1][2] ≠ 0 and Board[3][1] ≠ 0 then
        if |Board[1][2] − Board[3][1]| ≠ 1 then
            return
    if Board[1][3] ≠ 0 and Board[2][1] ≠ 0 then
        if |Board[1][3] − Board[2][1]| ≠ 1 then
            return
    if Board[1][Files − 1] ≠ 0 and Board[3][Files] ≠ 0 then
        if |Board[1][Files − 1] − Board[3][Files]| > 1 then
            return
    if Board[1][Files − 2] ≠ 0 and Board[2][Files] ≠ 0 then
        if |Board[1][Files − 2] − Board[2][Files] ≠ 1 then
            return
    if Board[Ranks − 2][1] ≠ 0 and Board[Ranks][2] ≠ 0 then
        if |Board[Ranks − 2][1] − Board[Ranks][2] ≠ 1 then
            return
    if Board[Ranks − 1][1] ≠ 0 and Board[Ranks][3] ≠ 0 then
        if |Board[Ranks − 1][1] − Board[Ranks][3]| ≠ 1 then
            return
    if Board[Ranks − 2][Files] ≠ 0 and Board[Ranks][Files − 1] ≠ 0 then
        if |Board[Ranks − 2][Files] − Board[Ranks][Files − 1]) ≠ 1 then
            return
    if Board[Ranks − 1][Files] ≠ 0 and Board[Ranks][Files − 2] ≠ 0 then
        if |Board[Ranks − 1][Files] − Board[Ranks][Files − 2]| ≠ 1 then
            return
    Board[rank][file] = Steps
    Steps = Steps + 1
    add Move(rank, file) to Path
    Moves = OrderedMoves(rank, file)                    ▷ See Algorithm 2
    for Move → Moves do
        if Board[Move.rank][Move.file] == 0 then
            Tour(Move.rank, Move.file)
    if Steps == Size then
        return null
    Board[rank][file] = 0
    Steps = Steps − 1
    Pop last element from Path
```

Algorithm 5 Merge Solved Tours Into Large Tour

```
This: Static
Input: T1: Tour, T1: Tour, T3: Tour, T4: Tour
Output: A Solved Tour Object

function MergeBoards(T1, T2, T3, T4)
    BigFiles = T1.Files + T2.Files
    BigRanks = T1.Ranks + T3.Ranks
    Merged = Tour(BigRanks, BigFiles)
    add T1.Path.remove() to T1.Path
    Current = T1.Path.pop()
    while Current.Rank ≠ T1.Ranks − 1 or Current.File ≠ T1.Files do
        add Current to Merged.Path
        Current = T1.Path.pop()
    add Current to Merged.Path
    while T2.Path[1].Rank ≠ T2.Ranks − 2 or T2.Path[1].File ≠ 2 do
        add T2.Path.remove() to T2.Path
    if T2.Board[T2.ranks − 2][2] < T2.Board[T2.Ranks][1] then
        add T2.Path.remove() to T2.Path
    while size of T2.Path > 0 do
        Current = T2.Path.pop()
        Current.Files = Current.Files + T1.Files
        add Current to Bigger.Path
    while T3.Path[1].Rank ≠ 1 or T3.Path[1].File ≠ 3 do
        add T3.Path.remove() to T3.Path
    if T2.Board[T2.ranks − 2][2] < T2.Board[T2.Ranks][1] then
        add T2.Path.remove() to T2.Path
    while size of T2.Path > 0 do
        Current = T2.Path.pop()
        Current.Files = Current.Files + T1.Files
        add Current to Bigger.Path
    while T4.Path[1].Rank ≠ 3 or T4.Path[0].Files ≠ T4.Files − 1 do
        add T4.Path.remove() to T4.Path
    if T4.Board[3][T4.Files − 1] < T4.Board[1][T4.Files] then
        add T4.Path.remove() to T4.Path
    while size of T4.Path > 0 do
        Current = T4.Path.pop()
        Current.Rank = Current.Rank + T1.Ranks
        add Current to Bigger.Path
    while size of T1.Path > 0 do
        add T1.Path.pop() to Bigger.Path
    for i → 1...Merged.Path.Size do
        Merged.Board[Merged.Path[i].rank][Merged.Path[i].rank] = i
    return Merged
```

Algorithm 6 Divide and Conquer Solution to the Knight's Tour Problem

```
    This: Tour Object
    Input: none
    Output: A Solved Tour Object

function DivideAndConquerTour(x)
    KT = null
    SmallFiles = This.Files / 2
    SmallRanks = This.Ranks / 2
    if This.Ranks < 10 and This.Files < 10 then
        Tour(1,1)
    else
        K1 = Tour(SmallRanks, SmallFiles)
        K2 = Tour(SmallRanks, SmallFiles)
        k3 = Tour(SmallRanks, SmallFiles)
        K4 = Tour(SmallRanks, SmallFiles)
        K1 = K1.DivideAndConquerTour()
        K2 = K2.DivideAndConquerTour()
        K3 = K3.DivideAndConquerTour()
        K4 = K4.DivideAndConquerTour()
        KT = JoinTours(K1, K2, K3, K4)
    return KT
```