Topic 5: Recursion

Matthew Johnson

matthew.johnson2@dur.ac.uk

Recursive Algorithms: Factorials

- The factorial of a positive integer n is the product of all integers from 1 to n, often denoted $n!$.
- For example, $5! = 5 \times 4 \times 3 \times 2 \times 1$.
- Observe that $5! = 5 \times 4!$.
- Hence we can implement the calculation of factorials using recursion.

Iterative Factorial

```
total = 1
for i=1 to n do
    total = total × i
end for
return total
```

Recursive Factorial: factorial(n)

```
if n=1 then
    return 1
else
    return n × factorial(n-1)
end if
```

Recursive Algorithms

- A recursive algorithm must have a base case.
- A recursive algorithm must change its state and move toward the base case.
- A recursive algorithm must call itself, recursively.

Iterative Sum of a list L

```
sum = 0
for i in L do
    sum = sum + i
end for
return sum
```

Recursive Sum of a list L: listsum(L)

```
if len(L) = 1 then
    return L[0]
else
    return L[0] + listsum(L[1:])
end if
return sum
```


We will look at many examples on the "whiteboard", or by looking at code. See the write up on duo after the lectures.

A Recursive Technique: Backtracking

- A technique for problems with many candidate solutions but too many to try.
- For example: there are 6,670,903,752,021,072,936,960 ways to fill in a sudoku grid.
- General idea: build up the solution one step at a time, backtracking when unable to continue.

Generic algorithm (informal)

1. Do I have a solution yet?
2. No. Can I extend my solution by one "step"?
3. If yes, do that.
4. Do I have a solution now? If yes, I'm done.
5. If not, try and extend again.
6. When I can't extend, take one step back and try a different way.
7. If no other extension available, then give up — no solution can be found.

▸ Sudoku demo from Wikimedia Commons

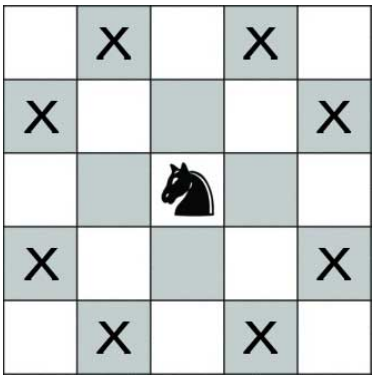
Generic algorithm

```
extend_solution(current solution)
  if current solution is valid then
      if current solution is complete then
          return current solution
      else
          for each extension of the current solution do
              extend_solution(extension)
          end for
      end if
  end if
```

For sudoku, start by calling extend_solution with the partially filled grid that is given to you.

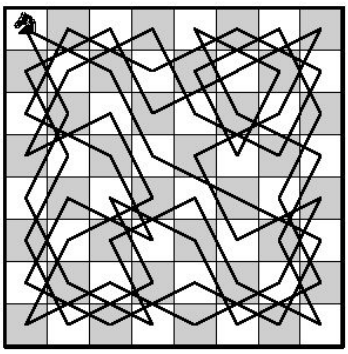
Knights

A knight is a chess piece that can move by moving one square in one direction and two squares in a perpendicular direction.



A Knight's Tour

A Knight's Tour: to move a knight around a chessboard such that each square is visited exactly once.



A Knight's Tour: using the generic algorithm

- What is a (partial) solution?
- When is it valid?
- When is it complete?
- How can the current solution be extended?

Using the generic algorithm

```
extend_solution(current solution)
  if current solution is valid then
      if current solution is complete then
          return current solution
      else
          for each extension of the current solution do
              extend_solution(extension)
          end for
      end if
  end if
```


Using the generic algorithm

```
extend_solution(current solution)
  if new move is to unvisited square on the board then
      if every square has been visited then
          return current solution
      else
          for each of eight possible moves do
              extend_solution(with move added)
          end for
      end if
  end if
```

So we have an algorithm for Knight's Tour . . .

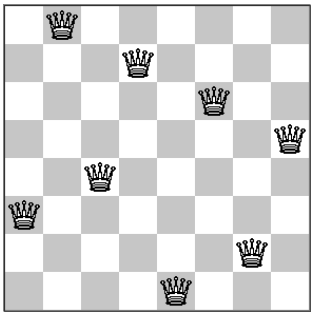
Implementing Knight's Tour

- Rather than having a list of moves made, it is easier to maintain an 8×8 array recording when each square was visited (initially all values are zero).
- Use a counter to record how many squares have been visited.

The algorithm is practical for a 6×6 board, but rather slow for an 8×8 board and impractical for much larger boards. What additional ideas could we add to the algorithm?

8 Queens Problem

A queen is a chess piece that attacks all squares it can reach horizontally, vertically or diagonally. The problem is to find all possible ways of placing 8 queens on a chessboard so that none attacks any other.



8 Queens Problem: using the generic algorithm

- What is a (partial) solution?
- When is it valid?
- When is it complete?
- How can the current solution be extended?

Using the generic algorithm

```
extend_solution(current solution)
  if current solution is valid then
      if current solution is complete then
          return current solution
      else
          for each extension of the current solution do
              extend_solution(extension)
          end for
      end if
  end if
```

Using the generic algorithm

```
extend_solution(current solution)
  if  new queen is not under attack  then
      if   8 queens on the board  then
           return  current solution
      else
           for each of eight squares in the next row do
                extend_solution(with extra queen)
           end for
      end if
  end if
```