Recursive cities

Description

Recursion simply means *return*. So a recursive city could be a city that you return to, or that encourages or requires you to keep coming back â?? like your home town, or a site of pilgrimage. The metaphor of *excursion and return* applies in many city contexts. See blog post on that theme.

Where thereâ??s repetition, itâ??s likely thereâ??s also *recursion* â?? a series of returns. The return may be to a different component that you treat the same as the whole, or the same but at a different scale, e.g. a neighbourhood grid sits within a district grid that is within a city grid.

Recursion applies to geometry, shapes, forms and arrangements in space, as well as rhythms, sounds and of course music. So the physical aspects of a city fall under the influence not only of repetition, but of *recursion*. An article by Michael Batty and Andrew Hudson-Smith affirms as much: *Imagining the recursive city.*

Tautologies

Douglas Hofstadter and Daniel Dennettâ??s famous book *GÃ¶del, Escher, Bach* provided a source of fascination for early generations of coders, design theorists and composers. The authors made much of the cognitive significance, if not paradoxes, of *recursion*. Think of the M.C. Escher drawing of a hand drawing a hand drawing itself.

> â??The concept is very general. (Stories inside stories, movies inside movies, paintings inside paintings, Russian dolls inside Russian dolls (even parenthetical comments inside parenthetical comments!) â??these are just a few of the charms of recursion.)â?• (135).

Recursion has a precise meaning to mathematicians and computer programmers. There are computer programs that repeat a procedure over and over, perhaps with variation. Thatâ??s *iteration*. Hence, a looped routine will draw a series of rectangles to replicate a city grid on a display screen.

Recursion is more cunning than iteration. It allows the programmer to define a procedure in terms of the procedure itself. Recursion is particularly useful in algorithms that do navigation and search. Thanks to

navigation apps like Google maps, we take for granted the algorithms that find the best route from Arbroath to Hampton Court.

Do what you do by doing what you do

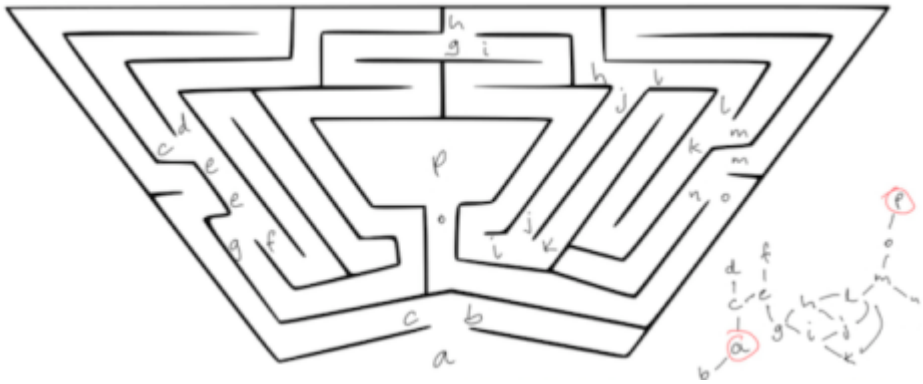
Hereâ??s a human example of recursion in navigation. Sometimes when someone asks for directions to the railway station, itâ??s helpful to say â??go to the end of this street, then ask someone there for directions.â?• You are saying: â??go to point A, which is on the network of possible points on the way, and apply the same procedure you did here, which was to ask someone.â?• To get to where you are going, you need to do what you are doing now.

More precisely, to go from A to B, select from a set of nodes adjacent to A in the network and go from there. A computer program canâ??t ask for directions, but will have a coded form of the city network at its disposal and the recursive definition will search all the possible connections to find the route.

Coding platforms that support recursion, such as Prolog, have to keep track of nodes and pathways, and be able to backtrack on dead ends or abortive loops. Hofstadter and Dennett philosophize around that process in a section of their book they call â??Pushing, Popping, and Stacksâ?• (136).

A worked example

Hereâ??s a way of simplifying and coding the Hampton Court Maze as a network of decision points.



Iâ??m pleased to see that itâ??s still possible to run programs written in the logic programming language Prolog (https://swish.swi-prolog.org). Hereâ??s the Hampton Court Maze reduced via Prolog â??assertionsâ?• to a series of decision points.

```
decide(a, b).
decide(a, c).
decide(c, d).
decide(c, e).
decide(e, g).
decide(e, f).
decide(g, h).
decide(g, i).
decide(i, j).
decide(i, k).
decide(h, j).
decide(j, l).
decide(k, l).
decide(l, m).
decide(m, n).
decide(m, o).
decide(o, p).
```


These logic statements simply record what nodes are adjacent to each other at each decision point in the maze. When I leaned Prolog it took me months to interpret, understand, write and debug the kind of logic code listed below. So, Iâ??m not expecting this to be clear. But it is worth noting that the â??:-â?• symbol means â??ifâ?•.

Just a cursory glance at the following code shows that you can **go** from one place to another if you can **go** from one place to another place. This sounds like a tautology, i.e. like recursion. The term â??goâ?• appears on both sides of an *if* statement. Itâ??s a recursive definition. The other stuff involving â??memberâ?• is to ensure that you donâ??t go to the same place more than once.

```
go(X, X, T, T).
go(X, Y, T, L) :- (decide(X, Z); decide(Z, X)), not(member(Z, T)), go(Z, Y, [Z|T], L).

member(A, [A|_]).
member(A, [_|B]) :- member(A, B).
```

Thatâ??s the program, and the â??decideâ?• assertions are the database, though for Prolog itâ??s all just logic. The way you interact with Prolog is with a text query, a bit like interrogating an SQL database.

```
?- go(p, a, [], L).
```

That means: how do you get to p from a? You donâ??t need to avoid any nodes []. Please *instantiate* L. If the program finds a value for L then thatâ??s the route. Prolog displays a text response indicating that the query statement is true, you *can* go to p from a in the Hampton Court Maze, i.e. as everyone knows, there is a solution. And here is the route. L is

```
[a, c, e, g, i, j, l, m, o].
```

In fact the program returns 3 other routes as well.

[a, c, e, g, h, j, l, m, o]
[a, c, e, g, h, j, i, k, l, m, o]
[a, c, e, g, i, k, l, m, o]

Thatâ??s the power of recursion.

References

- Batty, Michael, and Andrew Hudson-Smith. 2007. Imagining the recursive city: Explorations in urban simulacra. In Harvey Miller, J. (ed.), *Societies and Cities in the Age of Instant Access*. Dordrecht, The Netherlands: Springer.
- Clocksin, W.F., and C.S. Mellish. 1981. *Programming in Prolog*. Berlin: Springer-Verlag
- Coyne, Richard. 1988. *Logic Models of Design*. London: Pitman
- Hofstadter, Douglas R., and Daniel C. Dennett. 1979. *GÃ¶del, Escher, Bach: An Eternal Golden Braid*. New York: Basic Books

Notes

- I adapted the Prolog routine from Clocksin and Mellish, pp.133-136, which to my frustration at the time didnâ??t yield the actual route. I explain this version in *Logic Models of Design*, p.46.
- Iâ??ve made the program look as simple as possible. There are smarter and more efficient ways to implement depth-first and breadth-first search in Prolog, and to instantiate the journey as a list of nodes. Here I have harvested the list that prevents journeys revisiting the same node as the final route.
- A challenge for recursive algorithms is to establish a stopping condition. The stopping condition for the membership rule is that an element is a member of a list if it is the first element in the list. If that isnâ??t the case, then an element will be a member of a list if it is a member of the rest of the list, i.e. the list stripped of its first element. The symbol â??|â?• is used to separate the first element of a list (the head), from the rest of the list (the tail). The square brackets of course contain an ordered list with elements (words) separated by commas.
- Number series are typically recursive, e.g. the next number in the series will be equal to the sum of the two numbers that go before it. So, starting with 1, the next number will be 0+1, then follows 1+1=2, then 1+2=3, then 2+3=5. Thatâ??s the start of the fibonacci series: 1, 1, 2, 3, 5, 8, 13, 21 â?¦

Category

1. Architecture

Tags

1. labyrinth
2. logic
3. maze
4. Prolog
5. search

Date Created

October 19, 2019

Author
rcoyne99