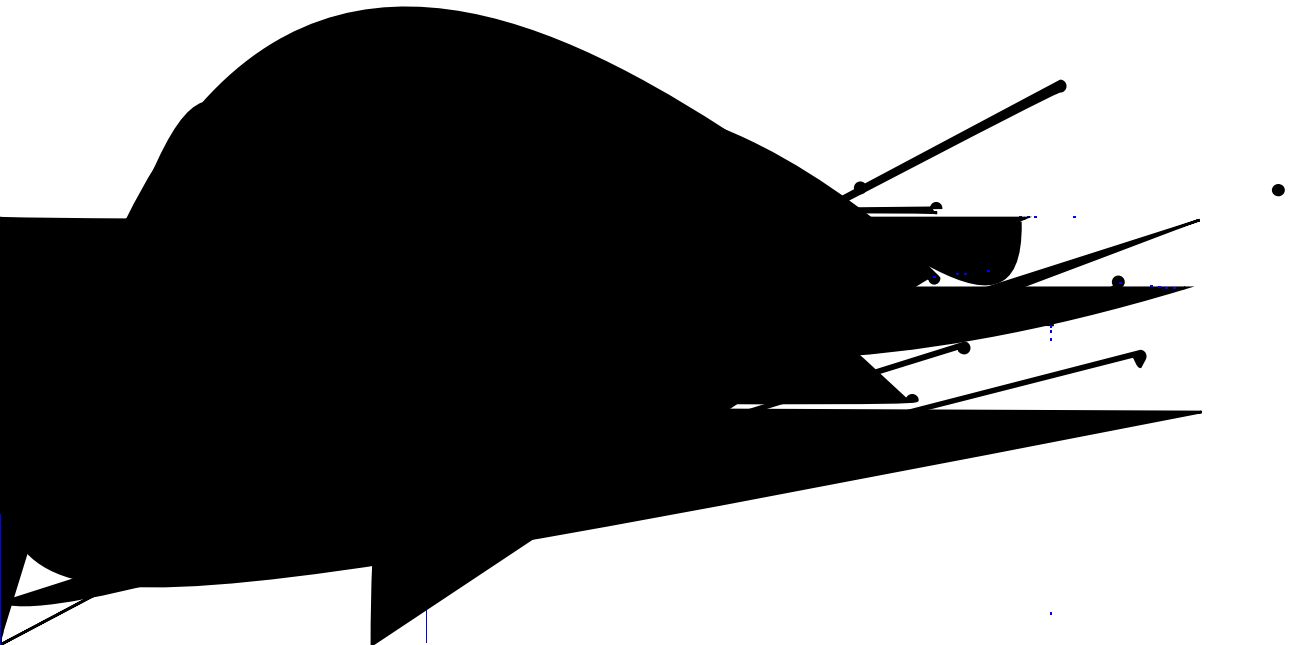
Finding a HamiltWn Path/Circuit.

Ham the

His ‘Theory Wf Systems Wf Rays’ (1827), cWmpleted when he was 23, provides a scientific basis for Optics that is still is use today . HamiltWn is also the creator Wf ‘QuaterniWns’, a lQVear algebra Wf 4-dimensiWnal vectors. The theory

created by Arthur Cayley (1821-1895). By the age Wf 13, HamiltWn had a mastery Wf 13 languages, including Hebrew, Sanskrit and Bengali. He entered Trinity in 1823 and before he graduated he accepted the Professorship Wf

member, to the NatiWnal AcadeUy Wf Sciences Wf the US.



For Euler circuits, a graph Pas an Euler circuit iff every vertex Pas even degree, but there is no such neat characterisation for a Hamilton circuit.

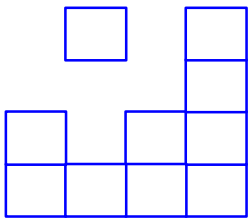
There is no Snown efficient algorithm for fQndQng a Hamilton patP / circuit. In general, fQndQng a Hamilton path / circuit is NP-complete. The fastest Snown algorithms take exponential time. A particular case of a Hamilton circuit is the TraveTlQng Salesman problem where a salesman wants to visit n cities via the shortest route.

There are simple criteria tPat are useful in analysQng whether a graph Pas a Hamilton patP / circuit.

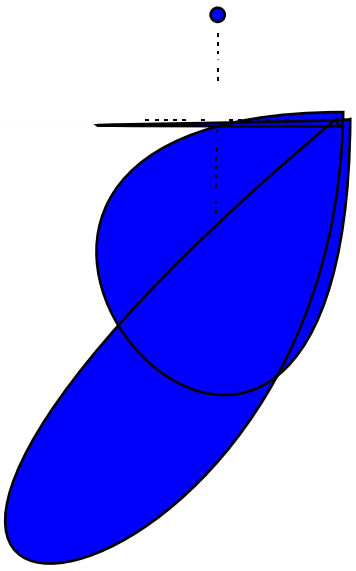
- If G Pas a Hamilton circuit, then aTl vertices Pave degree≥ 2.
- If degree of v = 2 tPen both edges Qncident on v are in the circuit
- circuit, then the other edges incident on v are not Qn tPe Hamilton circuit.

The Knight's Tour and Hamilton Circuit.

We can modeT tPe problem of tPe Knight's journey / tour by a graph (or dQgraph) . Each square on tPe board is a vertext and each possQble Snight's move is an edge.

a) 

tPat have ≥ m edges.



```
find_cycle(j, move : INTEGER) is
        lWcal
                S : INTEGER
        do
                if move = G.count  and then G.item(j,1) then
                        success := true
                else
                        from
                                S := 1
                        until
                                S > G.count or success
                        lWWp

                                        visited.put(true,S)      -- marS S as visited

                                        find_cycle(S, move+1)
                                        visited.put(success, S) --   unmarS S,
```

```
i fG.item(j,S) and not visited.item(S)then

hamilton_cycle.put(k, move+1)
```