Hamiltonian Cycle
using Backtracking

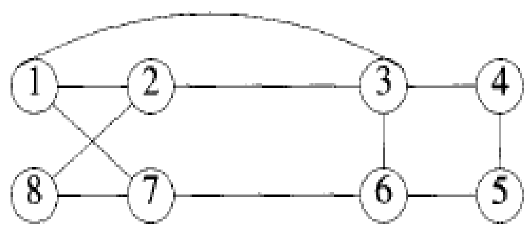
Hamiltonian Cycle Problem

Let G = (V, E) be a connected graph with n vertices. A Hamiltonian cycle
a round-trip path along ‘ *n’* edges of G that visits every vertex once and returns to its starting position

In other words if a Hamiltonian cycle begins at some vertex ‘v_1’ ϵ G and the vertices of G are visited in the order $v_1, v_2, \ldots, v_{n+1}$ then the edges (v_i, v_{i+1})

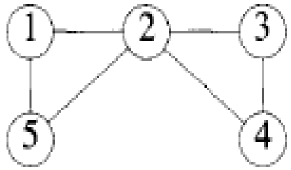
are in E, $1 \leq i \leq n$ and the v_i are distinct except for v_1 and v_{n+1} which are equal.

G1:



For G1:
Hamiltonian cycle 1,2,8, 7, 6,5, 4, 3,1

G2:



For G2:
No Hamiltonian cycle

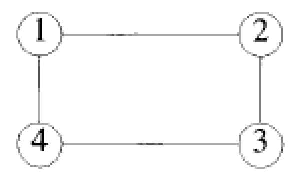
Hamiltonian Cycle Problem

The backtracking solution vector $(x_1, \ldots, x_n)$ is defined so that x_i represents the ith visited vertex of the proposed cycle.

determine how to compute the set of possible vertices for x_k if $x_1, \ldots, x_{k-1}$ have already been chosen.

If $k = 1$, then x_1 can be any of the n vertices. To avoid printing the same cycle n times, we require that $x_1 = 1$.

1-2-3-4-1
2-3-4-1-2 all are same cycles



If $1 < k < n$, then x_k can be any vertex v that is distinct from $x_1, x_2, \ldots, x_{k-1}$ and v is connected by an edge to x_{k-1}.

The vertex x_n can only be the one remaining vertex and it must be connected to both x_{n-1} and x_1.

Hamiltonian Cycle Problem

This algorithm is started by first initializing the adjacency matrix $G[1:n,1:n]$, then setting $x[2:n]$ to zero and $x[1]$ to 1, and then executing Hamiltonian(2).

```
Algorithm Hamiltonian(k)
// This algorithm uses the recursive formulation of
// backtracking to find all the Hamiltonian cycles
// of a graph. The graph is stored as an adjacency
// matrix G[1 : n, 1 : n]. All cycles begin at node 1.
{
    repeat
    { // Generate values for x[k].
        NextValue(k); // Assign a legal next value to x[k].
        if (x[k] = 0) then return;
        if (k = n) then write (x[1 : n]);
        else Hamiltonian(k + 1);
    } until (false);
}
```

Hamiltonian Cycle Problem

```
Algorithm NextValue(k)
// x[1 : k − 1] is a path of k − 1 distinct vertices. If x[k] = 0, then
// no vertex has as yet been assigned to x[k]. After execution,
// x[k] is assigned to the next highest numbered vertex which
// does not already appear in x[1 : k − 1] and is connected by
// an edge to x[k − 1]. Otherwise x[k] = 0. If k = n, then
// in addition x[k] is connected to x[1].
{
    repeat
    {
        x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
        if (x[k] = 0) then return;
        if (G[x[k − 1], x[k]] ≠ 0) then
        { // Is there an edge?
            for j := 1 to k − 1 do if (x[j] = x[k]) then break;
                                // Check for distinctness.
            if (j = k) then // If true, then the vertex is distinct.
                if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
                    then  return;
        }
    } until (false);
}
```