Section 4.5: Hamiltonian Cycle Problem

Given: a graph G = (V, E) of order n and size m

Find: a **Hamiltonian cycle** in G, that is, find a cycle in G that includes every vertex of V; or indicate that no such Hamiltonian cycle exists.

More formally, find an ordering of **<u>all</u>** n vertices of V, say

$$v_1, v_2, \ldots, v_n, v_1$$

so that

- v_i and v_{i+1} are adjacent for all $i = 1, 2, \ldots, n-1$
- v_n and v_1 are adjacent
- no vertex is repeated in the list $v_1, v_2, \ldots, v_n$
- the list $v_1, v_2, \ldots, v_n$ contains all n vertices of V

Let G be a graph with vertices labeled 1, 2, …, n.
Let `a` be the adjacency matrix of the graph:

`a[i][j] = 1` if vertex `i` and vertex `j` are adjacent
`a[i][j] = 0` if vertex `i` and vertex `j` are not adjacent

The array `x` will store the order of the vertices on a Hamiltonian cycle (if the graph has one).

`x[i]` will be the `i`th vertex on the cycle
`x[1] = 1` i.e., we will assume that the cycle begins at vertex 1

If we successfully find a Hamiltonian cycle, then

for all `i = 1, 2, ..., n-1`, the vertex `x[i]` must be adjacent to `x[i+1]`

`a[          ][          ] = 1`

`x[n]` must be adjacent to `x[1]`

`a[          ][          ] = 1`

Use backtracking.
We will attempt to find a Hamiltonian cycle `x[1], x[2], ..., x[n], x[1]` by creating partial solutions:

- fill the `k`th position in the array `x` with each possible vertex
- to be a partial solution, `x[1],x[2],...,x[k]` must be a sequence of **distinct**, **adjacent** vertices (a path of length `k` that we hope to close out to a cycle of length n)
- to make sure the vertices are distinct, we will keep track of which vertices have been used
  `used[i] = true` if vertex `i` has been used in the array `x`
- check the adjacency matrix `a` to see if vertex `x[k]` is adjacent to vertex `x[k-1]`
- if the array is filled, also check that `x[n]` is adjacent to `x[1]`
- if the array is not yet filled, recursively fill the next position in the array `x`

```
hamiltonian(a, x){
  n = a.last
  x[1] = 1          // can assume any Hamiltonian cycle
  used[1] = true    // begins and ends at vertex 1
  for i = 2 to n
    used[i] = false
  if (rhamiltonian(a, 2 , x))
     // output solution
}

rhamiltonian(a, k, x){
// returns true if x[1],x[2],...,x[k] is a (partial) solution
  n = a.last
  for ________________________________ {
    x[___] = _______
    if (path_ok(a, k, x)){
      used[______] = ____________
      if (k == n) {
         // output x (use a loop since x is an array)
         return true
      }
      else
         rhamiltonian(a, ______, x)
      used[_____] = ________________
    }
  }
  return  _______________
}

path_ok(a, k, x){
// returns true if vertex x[k] is "unused" and x[k-1] is adjacent
// to x[k]; if k = n, it also checks if x[k] is adjacent to x[1]
  n = a.last
  if (used[________])
    return false
  if (k < n)


  else


}
```

Traveling Salesperson Problem

Given a set of n cities and the cost of travel between various cities, determine the cheapest way for a traveling salesperson to visit each city exactly once and return to the starting city.

In graph theoretic terms

Given: a weighted graph G = (V, E) of order n and size m with weight function w, defined on the edges

Find: a Hamiltonian cycle in G of minimum total weight

More formally, find an ordering of **<u>all</u>** n vertices of V, say

$$v_1, v_2, \ldots, v_n, v_1$$

so that

- v_i and v_{i+1} are adjacent for all $i = 1, 2, \ldots, n-1$
- v_n and v_1 are adjacent
- no vertex is repeated in the list $v_1, v_2, \ldots, v_n$
- the list $v_1, v_2, \ldots, v_n$ contains all n vertices of V
- among all possible orderings, $\sum_{i=1}^{n-1} w(v_i v_{i+1}) + w(v_n v_1)$ is a minimum

```
tsp(weight, x){

  // minCost will contain the cost of a minimum tour
  // minTour will contain the ordering (tour) of vertices
  // Assume both minCost and minTour are global variables
  // weight is the weight matrix (data for the weights of edges)
  // x is the array used to backtrack through possible tours
  // used keeps track of which vertices are used - no repeats

  n = adj.last
  x[1] = 1           // can assume any tour of vertices
  used[1] = true     // begins and ends at vertex 1

  minCost = MAXINT
  minTour[1] = 1

  for i = 2 to n
    used[i] = false
  rtsp(weight, 2 , x)
}
```

```
rtsp(w, k, x) {
// returns true if x[1],x[2],...,x[k] is a (partial) tour
  n = adj.last
  for j = 2 to n {
    x[k] = j
    if (path_ok(w, k, x)) {
      used[x[k]] = true
      if (k == n) {
        // found a tour; find total cost and compare to minCost










      }
      else
        rtsp(w, k+1, x)
      used[x[k]] = false
    }
  }
  return false
}

path_ok(w, k, x) {
  n = a.last
  if (used[x[k]])
    return false
  if (k < n) {      // check if x[k-1] and x[k] are adjacent



  }
  else {



  }
  return
}
```