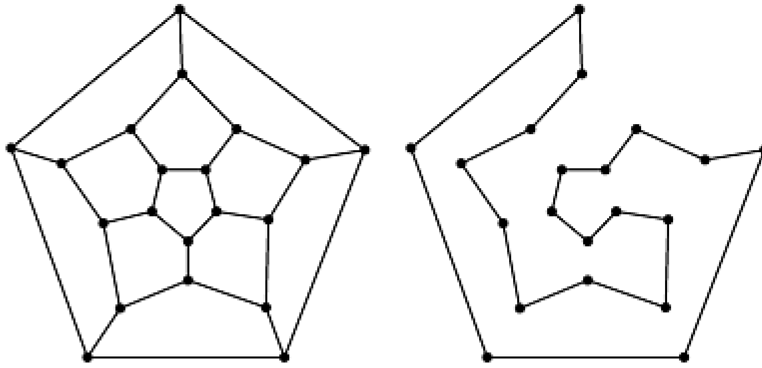
CSE 413
Homework 1 Solution Guide

1)

For this example, the following path is Hamiltonian. Graphic from:
http://www.cs.sunysb.edu/~algorith/files/hamiltonian-cycle.shtml



2)

The Idea:

Consider a graph with N vertices. Note that any Hamiltonian Tour must contain every vertex exactly once. Therefore, to determine if a Hamiltonian tour exists, the different permutations of the set of all vertices of the graph can be considered. Each permutation can then be checked against the adjacency matrix of the graph to ensure that every two consecutive vertices in the tour are indeed linked by an edge in the graph. If a permutation contains a pair of consecutive vertices that are not linked, that permutation cannot be a Hamiltonian tour. If all permutations have been tested and fail the test, then no tour is present in the graph.

Code:

```
boolean adjacency[N][N];

boolean findHamiltonian()
{
        int test[N];
        boolean used[N] = {false};
        recursive_find(0,current,used);
}
boolean recursive_find(int on, int current[N], boolean used[N])
{
        for(int i=0; i<N;i++)//for every vertex
        {
                if(!used[i])//if it is not already in this permutation
                {
                        current[on] = i;//add it
                        used[i] = true;//mark it as added
                        on ++;//move to the next mode
                        if(on==N &&test(current))//if we are at the end, test
                                return true;
                        else if(on<N)//if the list is not full, add another
                                if(recursive_find(on,current,used))
                                        return true;

                        used[i] = false;//this path has failed moved on
                        on --;
                }
        }
        return false;
}
boolean test(int order[])
{
        tf(order.length!=N)
                //invalid
```

```
        for(int i=0; i<N;i++)
        {
                int next = i+1;
                if(i==N-1)
                        next = 0;

                if(!adjacency[i][next])
                        return false;
        }
        return true;
}
```

Complexity

Because the heart of this algorithm takes every permutation of the integers 0 to n-1, the complexity is at least factorial. In fact it is O(n*n!), passing through the recursion n! times with an O(n) loop in each pass. This severely limits the size of input that will be practical for this algorithm. None of the structure of the graph was taken advantage of in this solution. The exhaustive search technique can be optimized to reduce backtracking and increase the practical input limit. The following paper details a method for a more efficient search.

Rubin, F. "A Search Procedure for Hamilton Paths and Circuits." *J. ACM* **21**, 576-580, 1974

3)

There are many ways to approach the coding of this problem, though the simplest ways will be similar to the pseudocode presented in the solution to problem 2. In the data for this project, the solution for the graph of size ten was not specified. There is a Hamiltonian tour in this graph, with {0,1,6,5,7,8,2,3,4,9,0} where 0 is the topmost point and the points are numbered in increasing order clockwise.

The important observation about the graph of program running times is that the increase in graph size does not cause a linear increase in running time. Instead the O(n*n!) running time increase should be seen with a sharp increase as the size goes up.

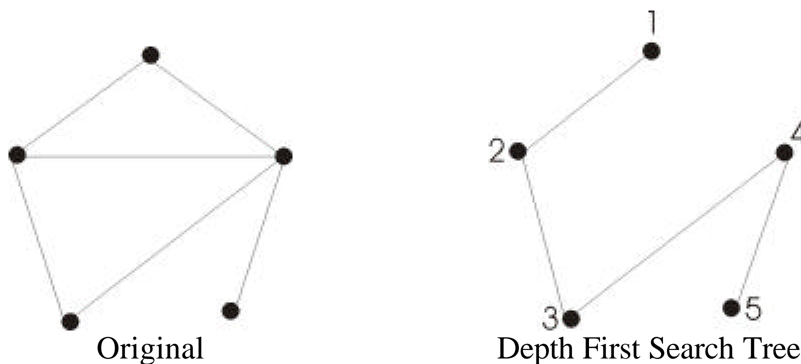
4)

The key relaxation of the Hamiltonian tour is that each vertex may be used more than once. Because of this, the tour found does not need to be a simple cycle and any edge may be used more than once. In order to find an approximate tour, take a depth first search of the graph. As each node is visited, add that node to the tour. It is important to remember to add not only each node that is unvisited, but each node that is visited through backtracking. At the end of the depth first search procedure, the search will have terminated on some node other than the starting node. Backtrack from this node to the original node. This produces an approximate Hamiltonian tour.

To determine the size of this approximation, consider the depth first search tree. This tree must contain every vertex (total N) and N-1 edges, one edge to reach every vertex except the first. Each time an edge is visited in this procedure a vertex is added to the tour. Each edge in the depth first search tree is visited exactly twice (once to enter a node and once to backtrack out of it). If there are at most N edges, this implies that the tour is at most 2N in size. In fact, the final vertex visited will only ever be added once. There is no reason to backtrack to it as it terminates the procedure, and the actual approximation is at most 2N-1.

Notice that this method can be used to find an approximate tour on any connected graph. The graph does not necessarily need to have a Hamiltonian tour for this to work.

Consider the following example.



Original Depth First Search Tree

The tour will add each vertex as it is visited in the depth first search to give {1,2,3,4,5}. However, this tour is incomplete as it does not finish on the final vertex. Therefore this method will backtrack along the search and give {1,2,3,4,5,4,3,2,1}.

This algorithm is a modified depth first search and will run in the same complexity as depth first search, O(|V|+|E|), where V is the set of vertices and E is the set of edges.

This approximation may be optimized further, though this is not required in this assignment. For example, start adding the vertices in the depth first search order. At any time when backtracking is used, look for alternate routes whenever backtracking, and take the first shortcut that is available. In this example when backtracking from 5 to 1, the (4,1) edge not included in the depth first search can be used to give a final tour of {1,2,3,4,5,4,1}. Depending on the extent of optimizations used, doing so may effect the complexity of the algorithm.