MAT-02660 Algoritmimatematiikka 2

Assignment: Twice-around-the-tree: An approximate solution for TSP

Your goal in this assignment is to write procedures that produce an approximate solution to the traveling salesperson problem (TSP).

TSP is one of the most well-known and highly studied optimization problems in the world. In TSP, one is given a set of cities with the distances between the cities. The problem in TSP is to find a circuit of minimal total distance that visits each city precisely once and returns to the starting city. In graph theory, the simplest TSP corresponds to finding a Hamiltonian circuit in a weighted undirected graph, such that the total weight of the circuit is minimal.

Since TSP is computationally difficult to solve, many methods have been invented to obtain approximate solutions. In this assignment you will write procedures for one of these methods: the twice-around-the-tree method (TATT method). Suppose one is given a weighted undirected graph $G = \langle V, E\rangle$. The weight of edge $\langle a, b\rangle$ is given by $w(\langle a, b\rangle)$. The TATT method consists of three stages:

Stage 1 Construct a minimal spanning tree (MST) for G. Let $T_{MST} = \langle V, E_{MST}\rangle$ be the MST.

Stage 2 Form a *walk* around the edges of T_{MST} where each edge is used twice and the starting vertex and the ending vertex of the walk are the same. We let this walk be represented by a vertex list:

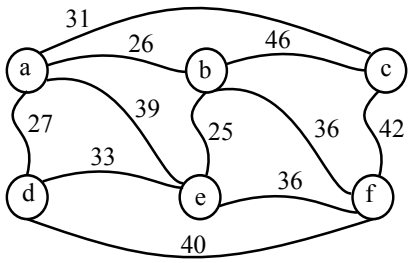
$$L = \langle v_0, v_1, v_2, \ldots v_k, v_0\rangle. \tag{1}$$

Since the walk through T_{MST} uses each edge twice, many vertices may occur several times in L.

Stage 3 Form all possible Hamiltonian circuits from the walk L of stage 2 by appropriately deleting vertices from L. Retain the shortest Hamiltonian circuit found.

In what follows, these three stages will be described using the following weighted graph as an example:

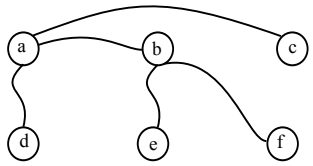
$G_1 =$



Stage 1 For forming an MST there are two standard algorithms: Prim's algorithm and Kruskal's algorithm. Pseudocode for Prim's algorithm is given in the course notes. A description for Kruskal's algorithm can be found from Wikipedia:

`https://en.wikipedia.org/wiki/Kruskal's_algorithm`

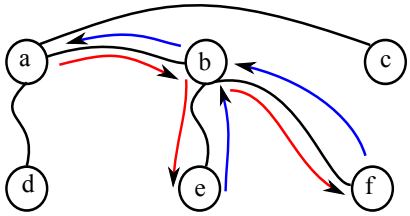
The following is one possible MST for graph G_1:

$T_{MST} =$ 

Stage 2 Given the MST T_{MST} from Stage 1, the job of stage 2 is to produce a walk satisfying two conditions:

- Each edge in T_{MST} is traversed twice.
- The walk starts and ends at the same vertex.

These requirements can be satisfied by performed a depth-first (DF) traversal such that, each edge is recorded both when the traversal moves forward along the edge and when the traversal moves backward along the edge. For example, suppose a DF-traversal of T_{MST} is started at vertex a and the first few movements in the traversal are as shown in the following figure:



In this figure, a red arrow indicates a forward movement and a blue arrow indicates a backward movement. Notice that each edge is traversed twice: once forwards and once backwards. One possible walk corresponding to the above figure is the following:

$$L = \langle a, b, e, b, f, b, a \rangle \ .$$

Of course this walk is not yet complete as vertices c and d have not yet been visited. Assuming vertex c is visited before vertex d, then the walk

$$L = \langle a, b, e, b, f, b, a, c, a, d, a \rangle \ . \tag{2}$$

is a complete walk satisfying the above two conditions. Since the walk produced in this stage is closed, one should not consider it to have a particular 'starting' or 'ending' point. Naturally different walks can be produced depending on the choices made in the DF-traversal or in the starting point of the DF-traversal.

Stage 3 In Stage 3 one starts with the walk from Stage 2 and deletes vertex repetitions, until each vertex appears only once. From the resulting list, one forms the corresponding Hamiltonian circuit. For example, starting with walk (2) and deleting all but the first occurrence of each vertex yields the following:

$$\langle a, b, e, \not{b}, f, \not{b}, \not{a}, c, \not{a}, d, \not{a} \rangle \ .$$

This corresponds to the following (potential!) Hamiltonian circuit:

$$H \ = \ \langle a, b, e, f, c, d, a \rangle \ . \tag{3}$$

To be specific, (3) corresponds to a Hamiltonian circuit with the following edges:

$$\langle a, b\rangle, \langle b, e\rangle, \langle e, f\rangle, \langle f, c\rangle, \langle c, d\rangle, \langle d, a\rangle \ .$$

If the starting graph contains all these edges, then a Hamiltonian circuit has been found. Unfortunately, (3) is not a Hamiltonian circuit for graph G_1, since G_1 does not include edge $\langle c, d\rangle$. If however, one used the following deletions with walk (2),

$$\langle \not{a}, \not{b}, e, b, f, \not{b}, \not{a}, c, a, d, \not{a}\rangle \ ,$$

then one would obtain the following Hamiltonian circuit:

$$H \ = \ \langle e, b, f, c, a, d, e\rangle \ . \tag{4}$$

This circuit does exist and is an approximate solution to the TSP.

If the starting graph contains all possible edges, Stage 3 is trivial. However, in reality one cannot assume there there will always be a direct road between each pair of cities x and y. Hence, it follows that some algorithm is needed to 'search' the walk from Stage 2 for Hamiltonian circuits. One method of performing this search is given here. We refer to this method as the *Cartesian Product Search*.

Let the walk produced from Stage 2 contain r vertices:

$$L \ = \ \langle x_1, x_2, \ \ldots \ x_r\rangle \, , \tag{5}$$

where $x_1 = x_r$. To each vertex in $V = \{\alpha_1, \alpha_2, \ \ldots \ \alpha_n\}$ we can associate a set of the positions in the walk:

$$P(\alpha) \ = \ \{i \,\big|\, x_i = \alpha\} \ . \tag{6}$$

For example, for walk (2), we get the following sets:

α	a	b	c	d	e	f
$P(\alpha)$	$\{1, 7, 9, 11\}$	$\{2, 4, 6\}$	$\{8\}$	$\{10\}$	$\{3\}$	$\{5\}$

(7)

Performing deletions in the walk so that one obtains a list containing only one occurrence of each vertex corresponds to the following three operations:

Cartesian product: Generate one list from the Cartesian product $P(\alpha_1) \times P(\alpha_2) \times \ \ldots \ \times P(\alpha_n)$. Let this list be M, $M = \langle s_1, s_2, \ldots s_n\rangle$.

Sort: Sort the integers in list M. The sorted list now corresponds to positions retained from the walk after deletions are made. Let the sorted list be M_{sort}.

Map: Map the positions in the sorted list to the original vertices of the graph to obtain a potential Hamiltonian path H_{path}.

After the Map operation, a potential Hamiltonian circuit is obtained by appending the first vertex in H_{path} to the end and then one can check whether each edge in the potential Hamiltonian circuit exists. Table 1 contains five cases of applying the Cartesian product search based on the sets of (7). Of the five cases, only the last two correspond to Hamiltonian circuits that do exist.

One can obtain a 'good' approximate solution to the TSP problem by checking all possible lists from the Cartesian product $P(\alpha_1) \times P(\alpha_2) \times \ \ldots \ \times P(\alpha_n)$ and saving the best (shortest) solution. For example, from the last two Hamiltonian circuits of Table 1, circuit $\langle e, b, f, c, a, d, e\rangle$ has length 194 while circuit $\langle e, f, b, c, a, d, e\rangle$ has length 209. For the $P(\alpha)$ sets

Cartesian product M	sorted M_{sort}	mapped H_{path}	potential circuit H	comment
$\langle 1,2,8,10,3,5\rangle$	$\langle 1,2,3,5,8,10\rangle$	$\langle a,b,e,f,c,d\rangle$	$\langle a,b,e,f,c,d,a\rangle$	see (3)
$\langle 1,4,8,10,3,5\rangle$	$\langle 1,3,4,5,8,10\rangle$	$\langle a,e,b,f,c,d\rangle$	$\langle a,e,b,f,c,d,a\rangle$	
$\langle 7,2,8,10,3,5\rangle$	$\langle 2,3,5,7,8,10\rangle$	$\langle b,e,f,a,c,d\rangle$	$\langle b,e,f,a,c,d,b\rangle$	
$\langle 9,4,8,10,3,5\rangle$	$\langle 3,4,5,8,9,10\rangle$	$\langle e,b,f,c,a,d\rangle$	$\langle e,b,f,c,a,d,e\rangle$	see (4)
$\langle 9,6,8,10,3,5\rangle$	$\langle 3,5,6,8,9,10\rangle$	$\langle e,f,b,c,a,d\rangle$	$\langle e,f,b,c,a,d,e\rangle$	

Table 1: Some examples of using the Cartesian product search method when starting from the P sets of (7).

of (7), there are 12 lists in the Cartesian product that should be checked. Many of these lists will correspond to non-feasible Hamiltonian circuits and some feasible Hamiltonian circuits may be repeated.

The goal of this assignment is to produce code that uses the TATT approach to form an approximate solution of TSP. It will be assumed here that the starting graph is given as a a weighted adjacency matrix A. The elements of A are defined as follows:

$$A(i,j) = \begin{cases} w(\langle i,j\rangle) & \text{when there exists an edge } \langle i,j\rangle \text{ in the graph} \\ \infty & \text{otherwise} \end{cases} \tag{8}$$

For example, the weighted adjacency matrix for graph G_1 is the following:

$$A = \begin{bmatrix} \infty & 26 & 31 & 27 & 39 & \infty \\ 26 & \infty & 46 & \infty & 25 & 36 \\ 31 & 46 & \infty & \infty & \infty & 42 \\ 27 & \infty & \infty & \infty & 33 & 40 \\ 39 & 25 & \infty & 33 & \infty & 36 \\ \infty & 36 & 42 & 40 & 36 & \infty \end{bmatrix} \tag{9}$$

If Matlab is being used, a graph object can be formed from a weighted adjacency matrix as follows:

```
G = graph(A);
```

There are many Matlab functions that take a graph object as argument. Here is a short list:

`addedge`, `addnode`, `adjacency`, `dfsearch`, `minspantree`, `plot`

1. In Stage 1, the MST must be produced. Let `makemst` be as procedure that does this. It will be assumed that `makemst` is called as follows:

   $$Amst = \texttt{makemst}(A)$$

   The argument to `makemst` is the weighted adjacency matrix (8). The output from `makemst` must contain sufficient information to form the MST. We give here two alternative outputs

   - an adjacency matrix corresponding to the MST $Amst$
   - a set, $Emst$, containing the edges of the MST

2. Write a procedure called `dftwice`, that will produce the walk specified in Stage 2. The procedure `dftwice` should take as input whatever output was produced by procedure `makemst`. For example, assuming the output from `makemst` is an adjacency matrix $Amst$, then procedure `dftwice` could be called as follows:

$$L = \texttt{dftwice}(Amst, x)$$

The argument x is the starting vertex for the DF-traversal. The output L is a list containing a TATT walk of Stage 2.

3. Write a procedure called `tsptwice` that will produce the best possible TSP solution from the TATT walk of Stage 3 using the Cartesian product search of Stage 3. One possible call to `tsptwice` is the following:

$$H = \texttt{tsptwice}(L, A)$$

The inputs here are the weighted adjacency matrix A of the original graph and a list L containing the TATT walk from procedure `dftwice`. The output H is the best (shortest) Hamiltonian circuit that can be formed from list L. If no feasible Hamiltonian circuit can be formed, then H should be an empty list.

4. Using graph G_1, test procedures `dftwice` and `tsptwice`. Draw the TSP solution produced and give its total weight.

5. Test procedures `mst`, `dftwice` and `tsptwice` using a graph whose weight matrix is the following:

$$W = \begin{bmatrix} - & 107 & - & 163 & - & - & 104 & 140 & 132 & 105 \\ 107 & - & - & - & 120 & - & - & - & 164 & 102 \\ - & - & - & 122 & 145 & - & 126 & 122 & 125 & - \\ 163 & - & 122 & - & 90 & - & 140 & 57 & - & - \\ - & 120 & 145 & 90 & - & - & 135 & 158 & 163 & 170 \\ - & - & - & - & - & - & 192 & 70 & 88 & - \\ 104 & - & 126 & 140 & 135 & 192 & - & - & - & - \\ 140 & - & 122 & 57 & 158 & 70 & - & - & 138 & 95 \\ 132 & 164 & 125 & - & 163 & 88 & - & 138 & - & 59 \\ 105 & 102 & - & - & 170 & - & - & 95 & 59 & - \end{bmatrix}$$

If the element W_{ij} is missing, it means that there is no edge between vertices i and j. Draw the TSP solution produced and give its total weight.