**ECE374 Assignment 8**

Due 04/17/2023

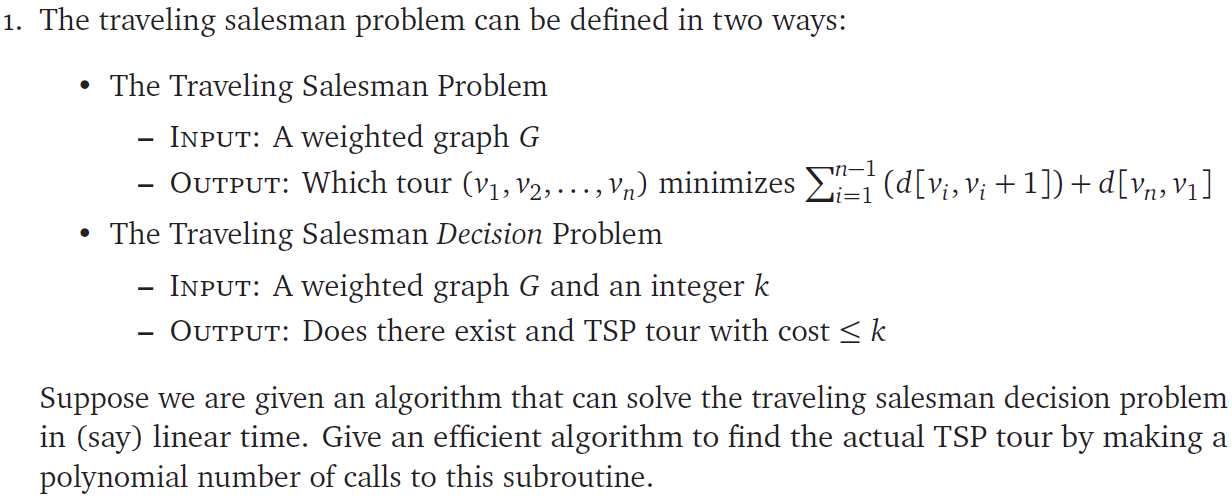
**Group & netid**

*Chen Si chensi3*

*Jie Wang jiew5*

*Shitian Yang sy39*

**Problem 1**



Solution:

Intuition:

We could solve this reduction problem by

(1) find the minimum k that returns True (there exists a TSP tour with cost <= k)

(2) find the tour corresponding to this minimum k:

(a) mark all vertices as unvisited

(b) iterate through all unvisited vertices u:

For each edge e that comes out of u

run TSDP(k)

if TSDP(k) = False 🡪

e is in the minimum TSP tour,

put e back,

mark u as visited

else: 🡪 keep e removed

What remains would be the shortest TSP tour with cost k.

The pseudocode of this algorithm is

**TSP**(G(V,E)):

// find minimum k

k = 0

while (TSDP(G,k) == False):

k++

// mark all vertices as unvisited

unvisited = {}

for v in V:

unvisited.add(v)

for u in unvisited:

for v in Adj(u):

E.remove(u,v) // Remove edge from u to v

if (TSDP(G,k) == False):

E.add(u,v) // Add back edge if on TSP path

Unvisited.remove(u) // Mark u as unvisited

return G // The remaining graph is the TSP tour of cost k

The complexity of this algorithm is [find minimum k] + [get path].

In [find minimum k], denote the largest edge weight in the graph as W, the worst case would loop k from 0 to (n-1)\*W. Therefore, this part is O((n-1)\*W\*TSDP)

In [get path], it takes O(TSDP\*m+n) to loop through all edges and all vertices, while performing TSDP on each edge.

Therefore, the total runtime cost is O(n\*W\*TSDP + m\*TSDP + n).