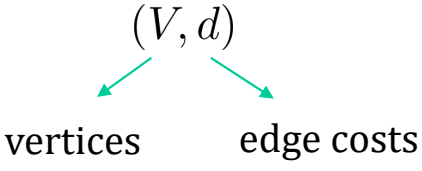
LEC014 Traveling Salesman Problem (TSP)

VG441 SS2020

Cong Shi
Industrial & Operations Engineering
University of Michigan

Traveling Salesman Problem (TSP)

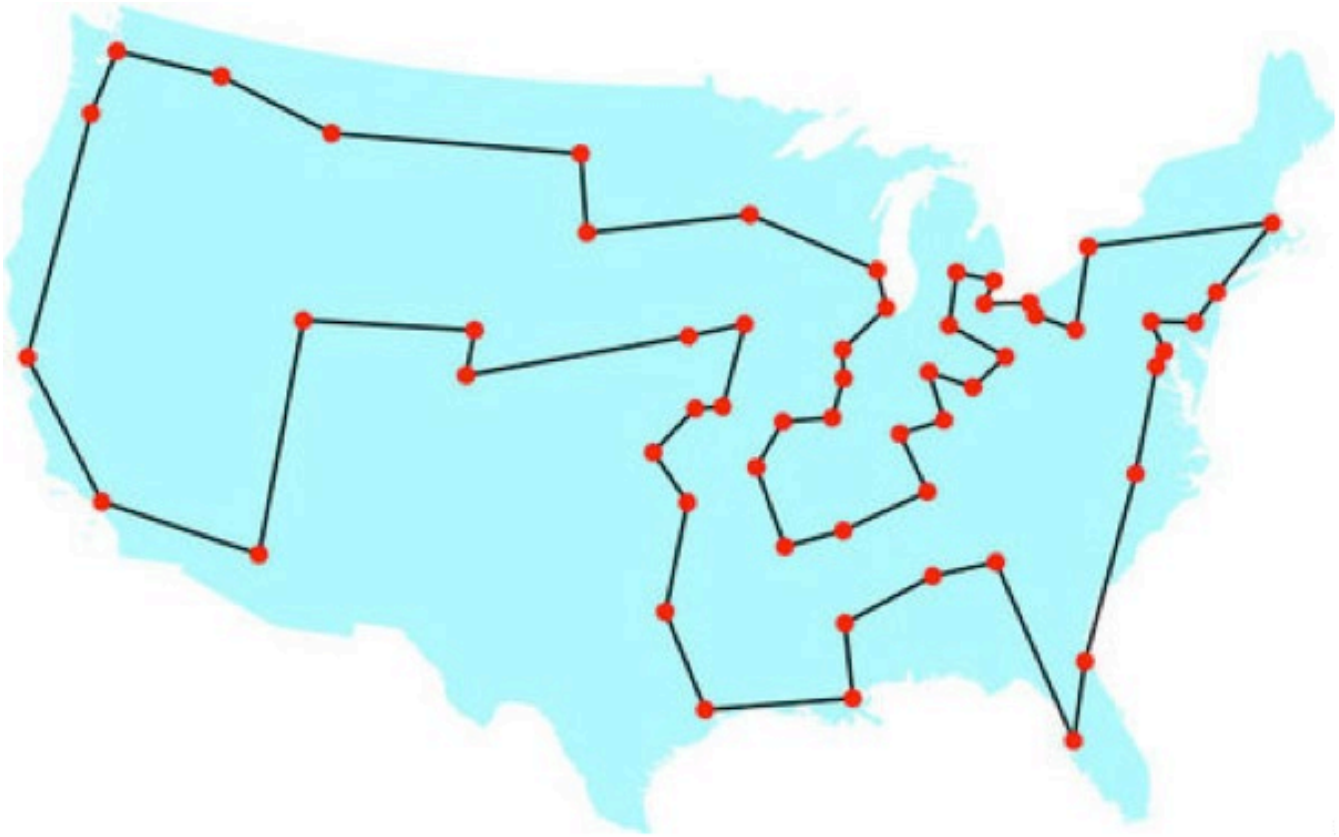
- A set of cities $V = \{1, 2, \ldots, n\}$
- A distance function (*metric*) $d : V \times V \to R_+$
  - Symmetry:
    $$d(u, v) = d(v, u), \forall u, v \in V$$
  - Triangular inequality:
    $$d(u, w) \leq d(u, v) + d(v, w), \forall u, v, w \in V$$
- Consider a complete graph



Objective: find a tour of minimum distance that visits each city exactly once and return to its starting point

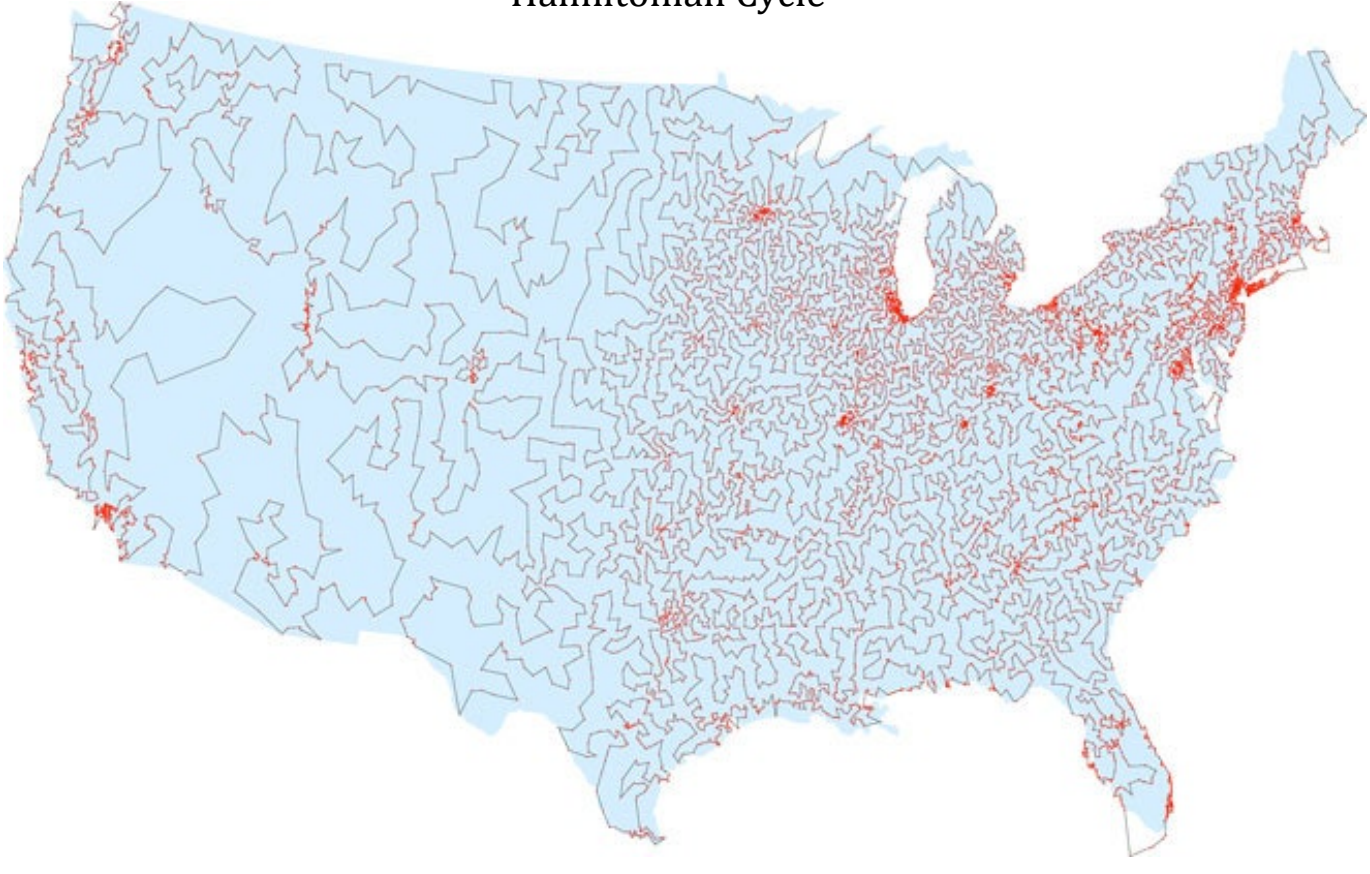
Metric TSP

Hamiltonian Cycle



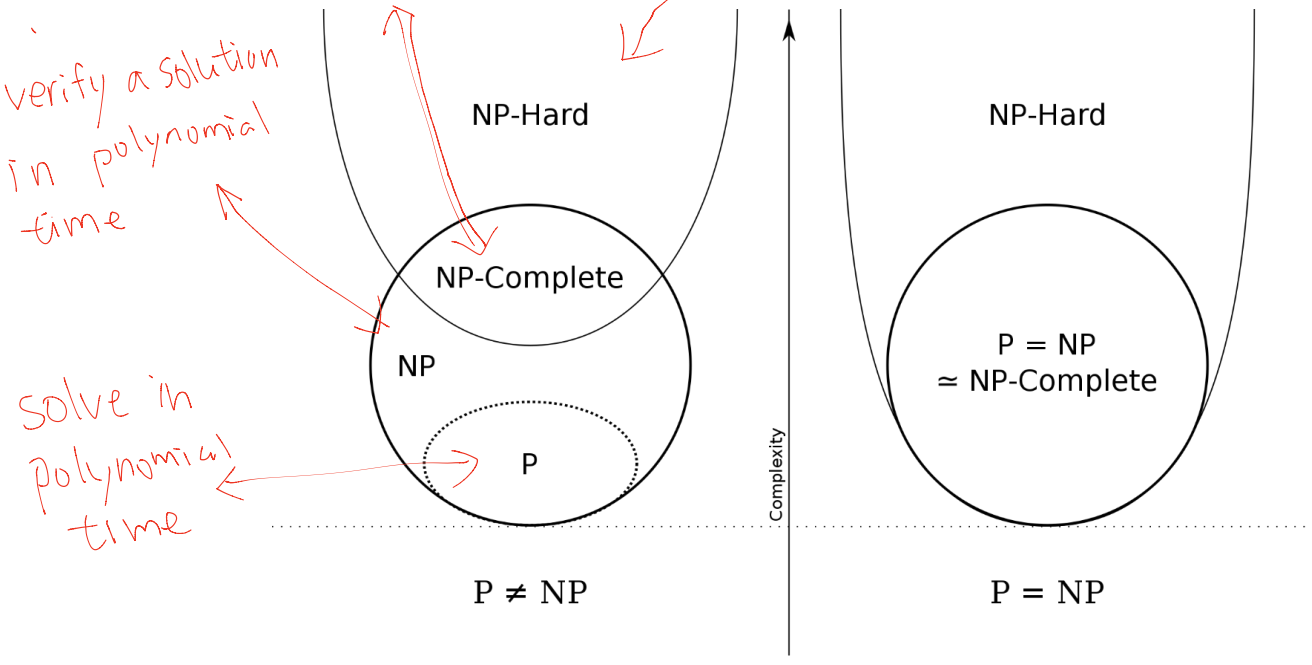
Metric TSP

Hamiltonian Cycle



vertex cover,

Metric TSP



Metric TSP is NP-Hard!

TSP

Lemma: For any instance I to the traveling salesman problem, the cost of optimal tour it at least the cost of the minimum spanning tree on I, i.e., $MST(I) \leq TSP(I)$

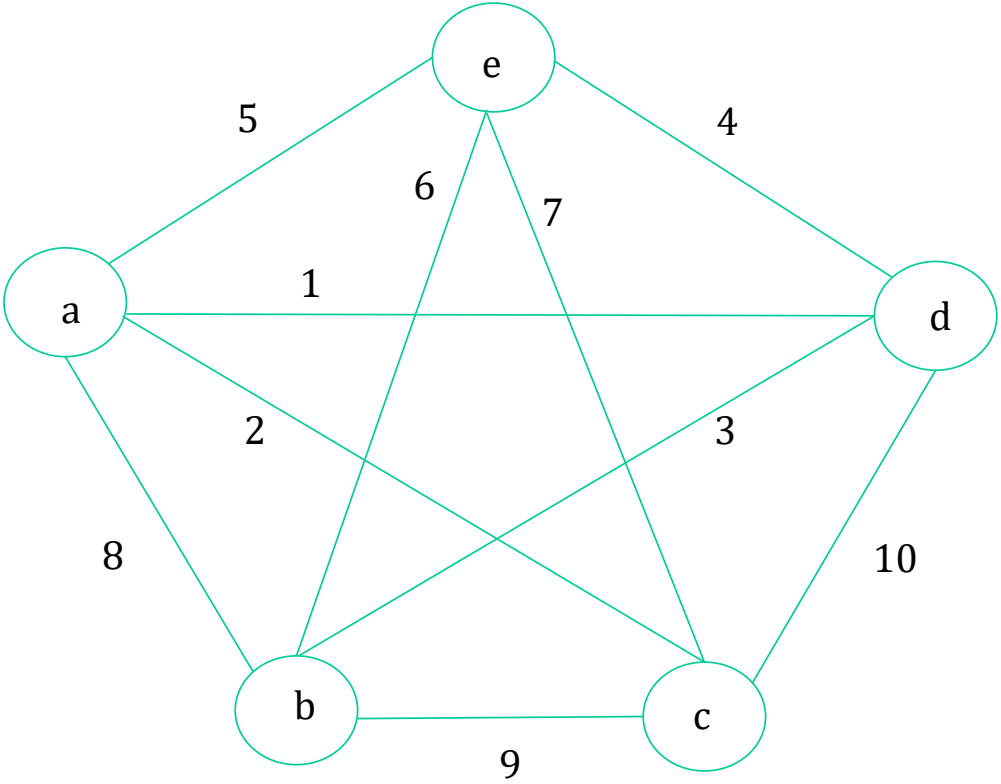
Proof: We assume instance I has $n \geq 2$ cities. Start with the optimal TSP tour of cost $TSP(I)$. If your remove one edge from the tour (break the cycle), the result is a spanning tree $ST(I)$ with a cost at most $TSP(I)$. since the minimum spanning tree (MST) is the one with the minimum cost over all spanning trees, it follows that $MST(I) \leq TSP(I)$.

TSP Double-Tree Algorithm

Algorithm (Double-tree algorithm)

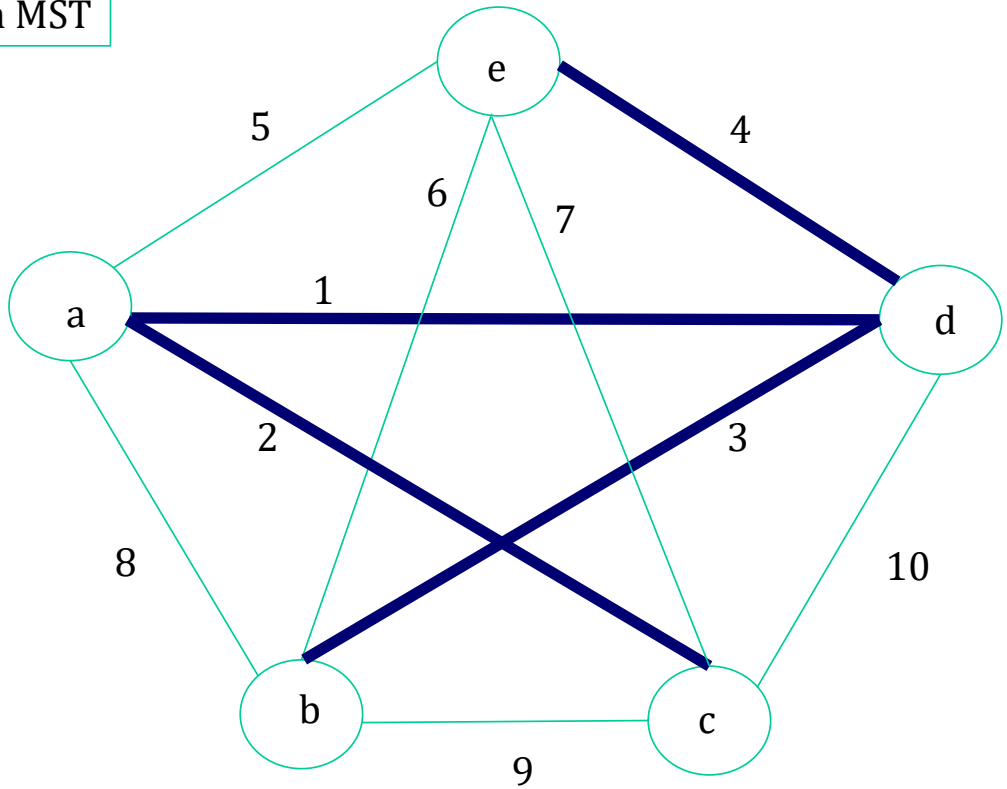
1. Compute the minimum spanning tree M on (V, d).
2. Double all edges of M and call the resulting graph D.
3. Find a walk W that uses each edge of D exactly once.
4. Shortcut W by skipping vertices that are re-visited to get a valid TSP tour T.

Example



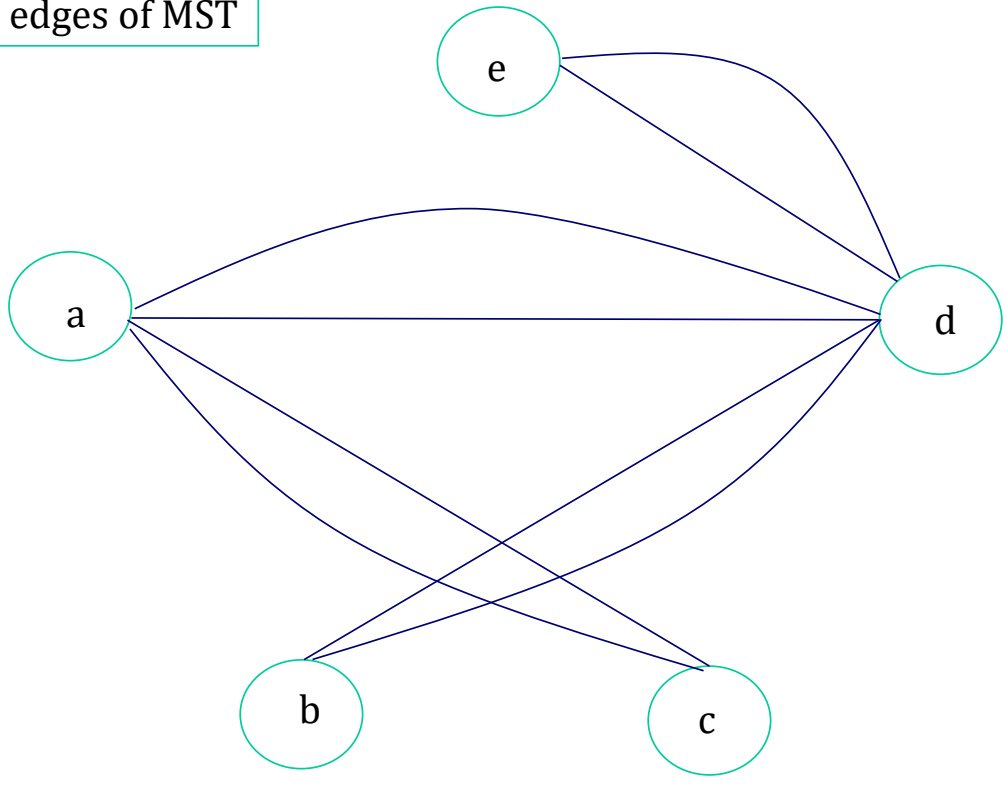
Example

Find an MST



Example

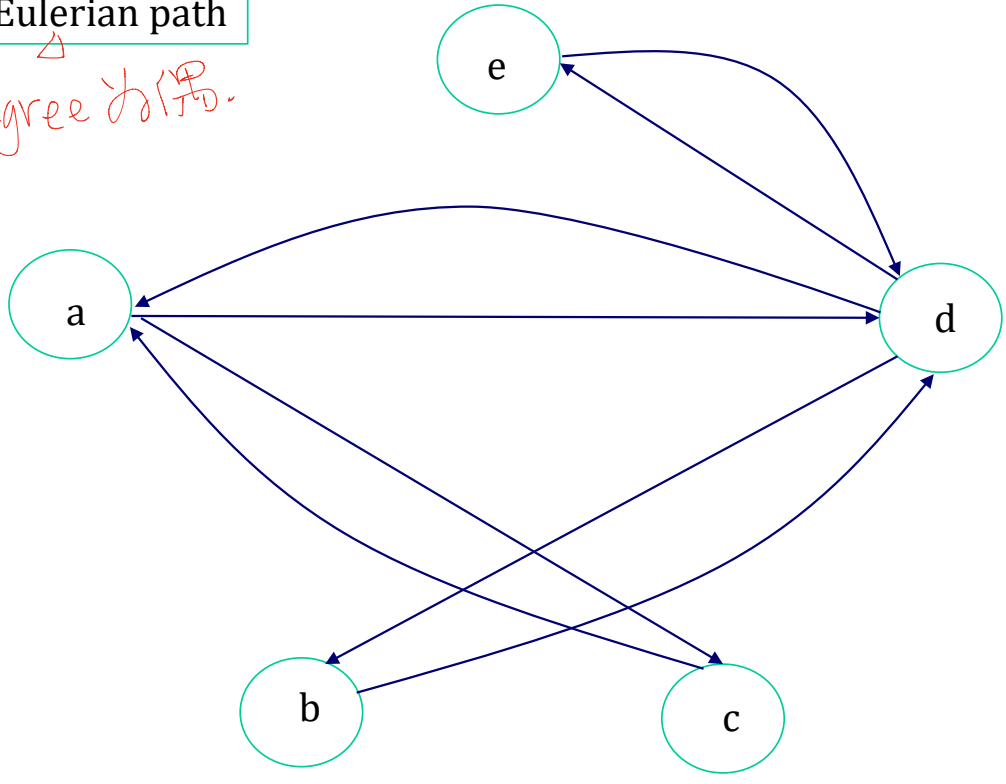
Double edges of MST



Example

Find a Eulerian path

每点 degree 为偶.

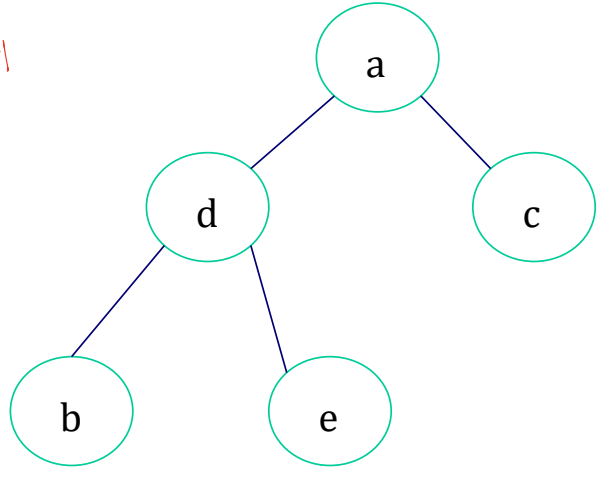


a → d → b → d → e → d → a → c → a

Example

Find a Eulerian path (use DFS traversal)

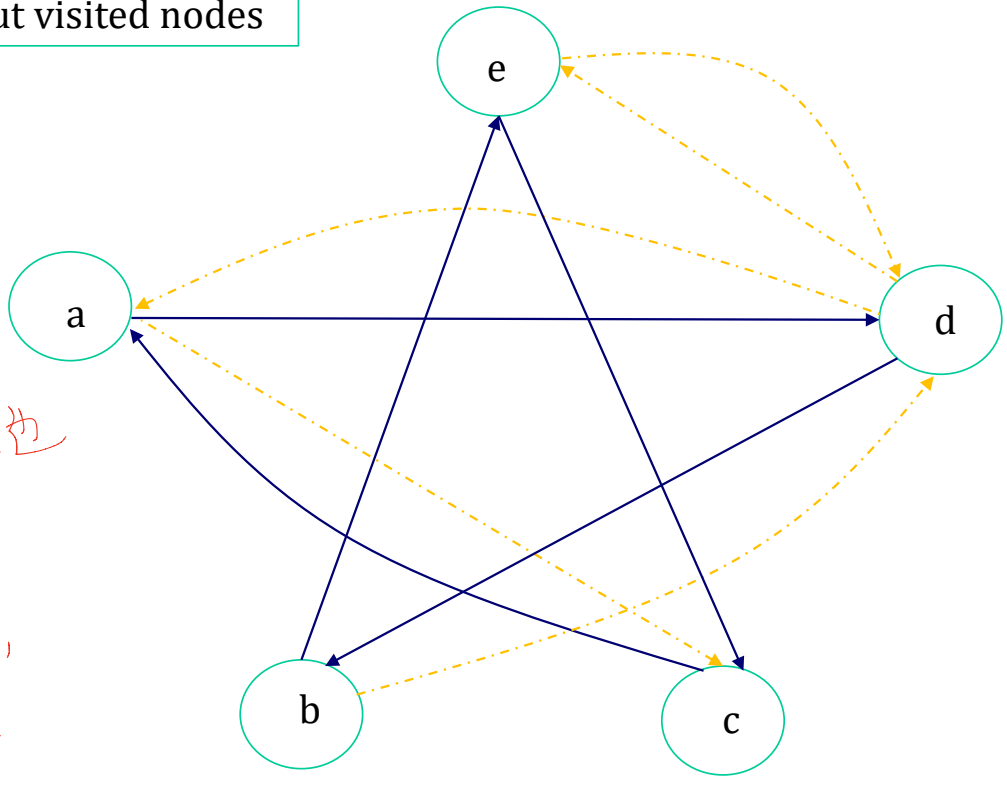
具体找的方法



a → d → b → d → e → d → a → c → a

Example

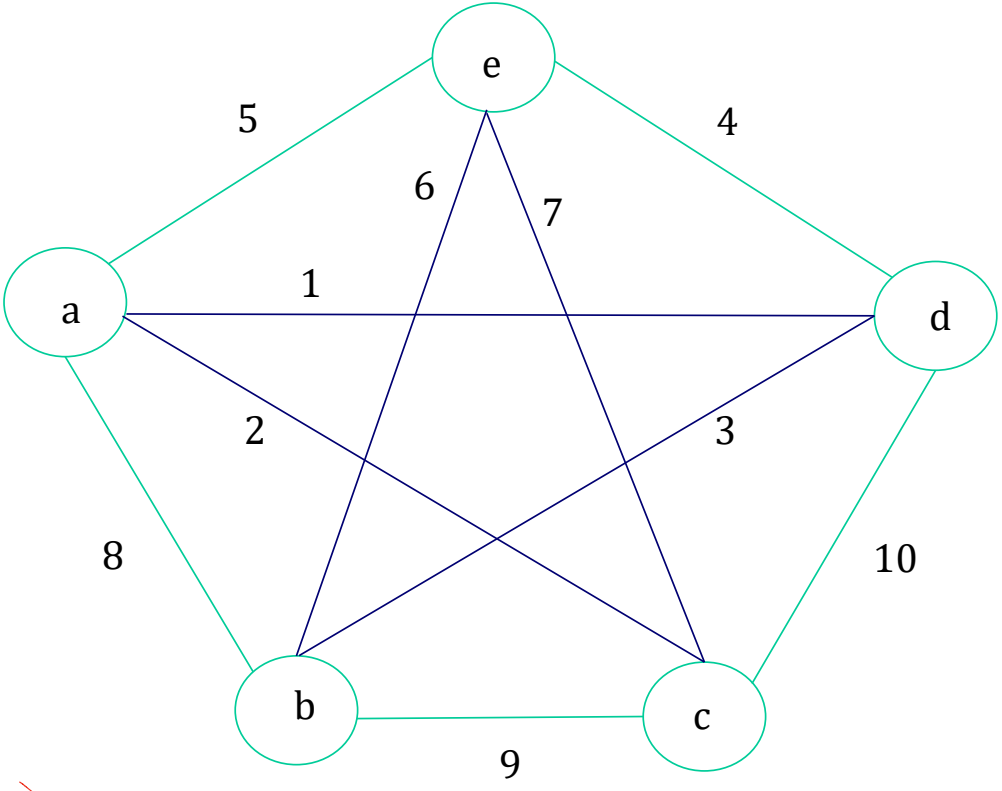
Shortcut visited nodes



Before shortcutting: a → d → b → d → e → d → a → c → a

After shortcutting: a → d → b → e → c → a

Example



We are lucky that this is opt.

Output: a → d → b → e → c → a with cost 19
OPT: a → d → b → e → c → a with cost 19

Double Tree = 2-Approximation

Theorem: The double-tree algorithm for TSP is a 2 -approximation algorithm.

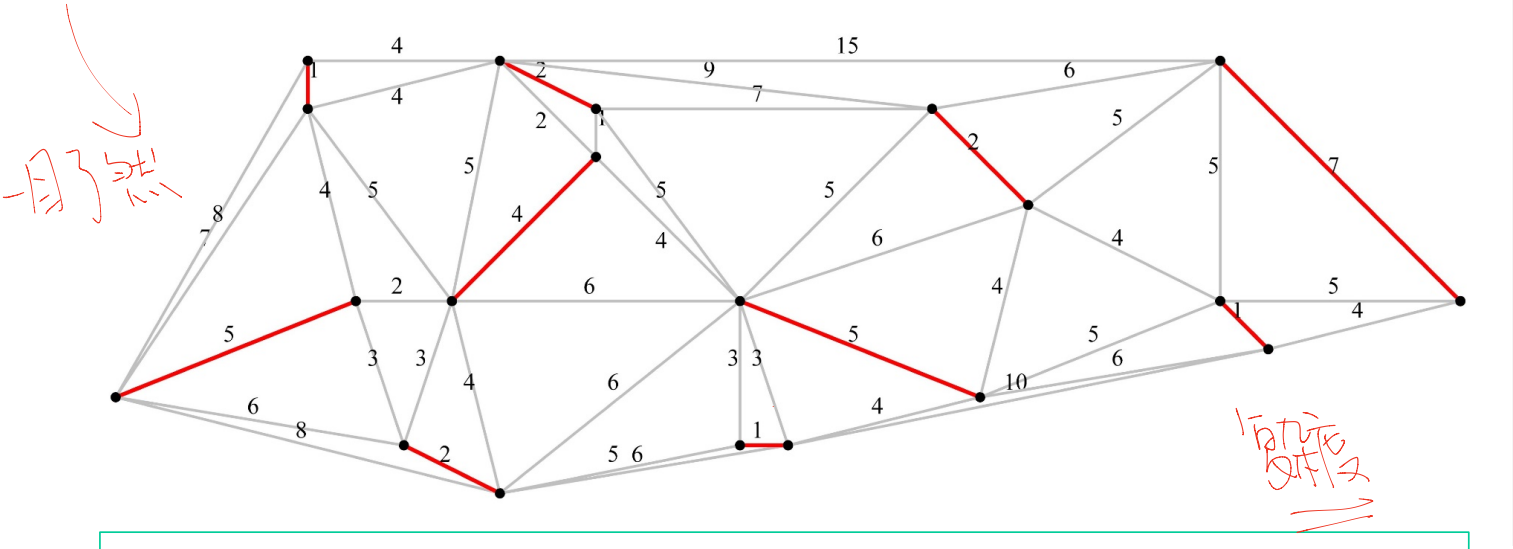
Proof: Let OPT be the cost of the optimal TSP tour.

- The cost of the minimum spanning tree M is at most OPT.
- We then double each edge (replace it with two copies) of M and the cost of the resulting graph D is at most $2OPT$. Also D is Eulerian by construction and a walk W of cost at most $2OPT$.
- Let W be the sequence $i_0, i_1, \ldots, i_k$ of cities where there may be repetitions. To get a tour T, we removing all but the first occurrence of each city in this sequence. This tour T contains each city exactly once (starts at i_0 and returns to i_0). We now show that the cost of T is at most that of W. Consider two consecutive cities in T : i_ℓ and i_m (we omitted $i_{\ell+1}, \ldots, i_{m-1}$ since these cities were already visited earlier in T). It then follows from the triangle inequality (and induction) that the distance d_{i_ℓ, i_m} is upper bounded by the total distance of the edges $(i_\ell, i_{\ell+1}), \ldots, (i_{m-1}, i_m)$ Adding up over all edges in T, the cost of T is at most the cost of W which is at most $2OPT$.

A Better Approximation Algorithm?

- Yes, the celebrated Christofides' algorithm for TSP

Matching: The input is a graph $G = (U, E)$ with even number of vertices U and distance function $d : U \times U \to R_+$. The goal is to find edges $K \subseteq E$ such that each vertex has exactly one end-point in K with minimum cost $\sum_{e \in K} d(e)$



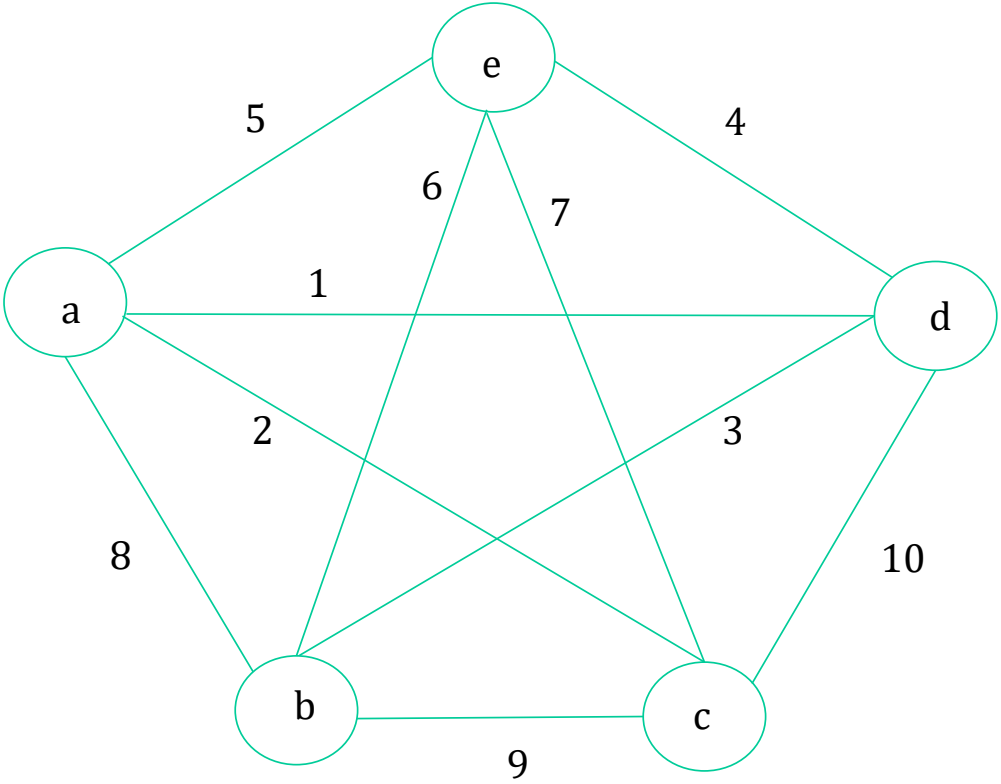
"Minimum-weight-perfect-matching" can be efficiently solved in $O(nm\log n)$
https://www.math.uwaterloo.ca/~bico/papers/match_ijoc.pdf

Christofides' algorithm

Algorithm: (Christofides' algorithm for TSP)

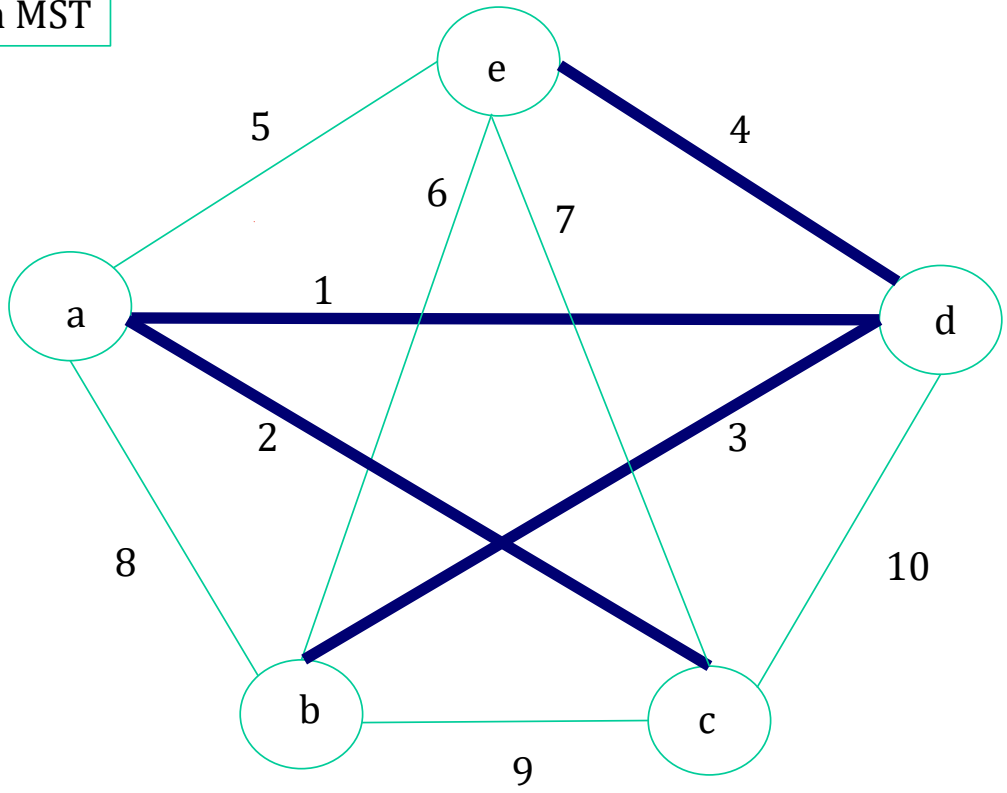
1. Compute the minimum spanning tree M on (V, d)
2. Compute the minimum cost matching K on odd degree vertices of M
3. Add the edges of K to M to obtain an Eulerian graph D'
4. Find a walk W' that uses each edge of D' exactly once.
5. Shortcut W' by skipping vertices that are re-visited to get a valid TSP tour T'

Example



Example

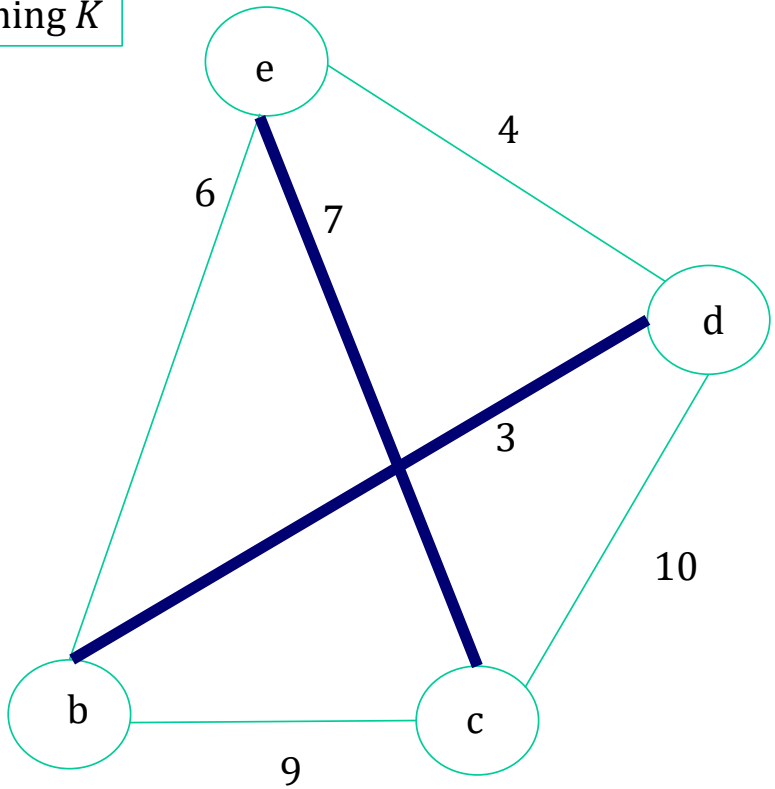
Find an MST



Find the set of odd degree vertices in M:
$U = \{e, d, b, c\}$

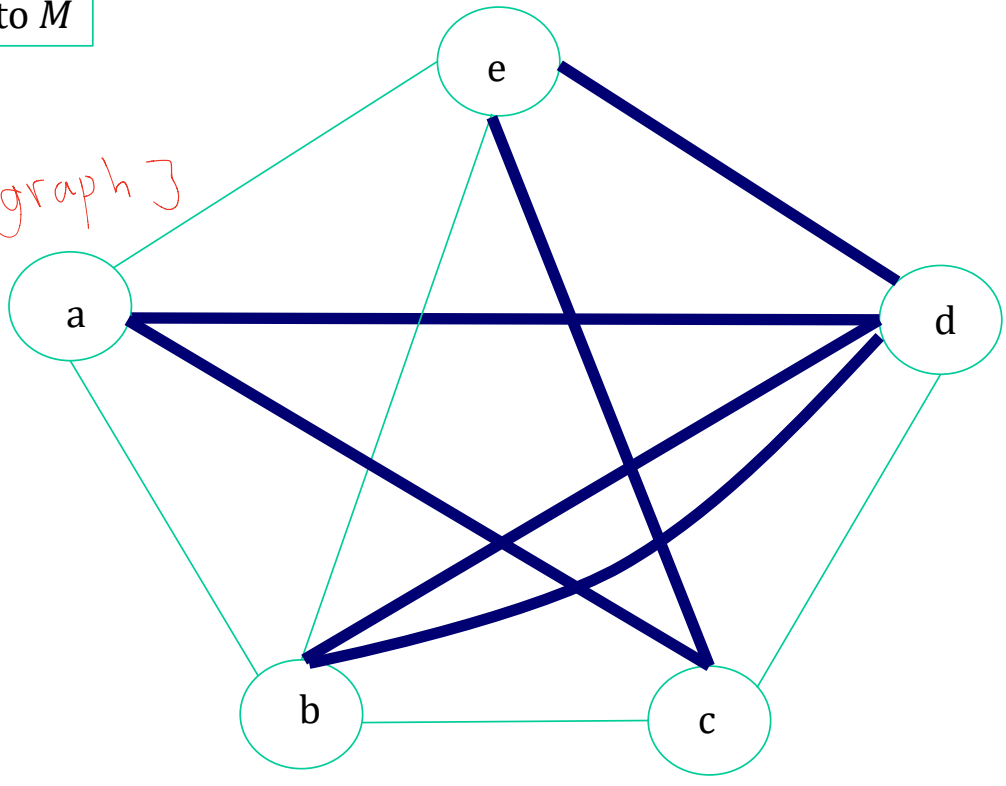
Example

Find min-weight-matching K



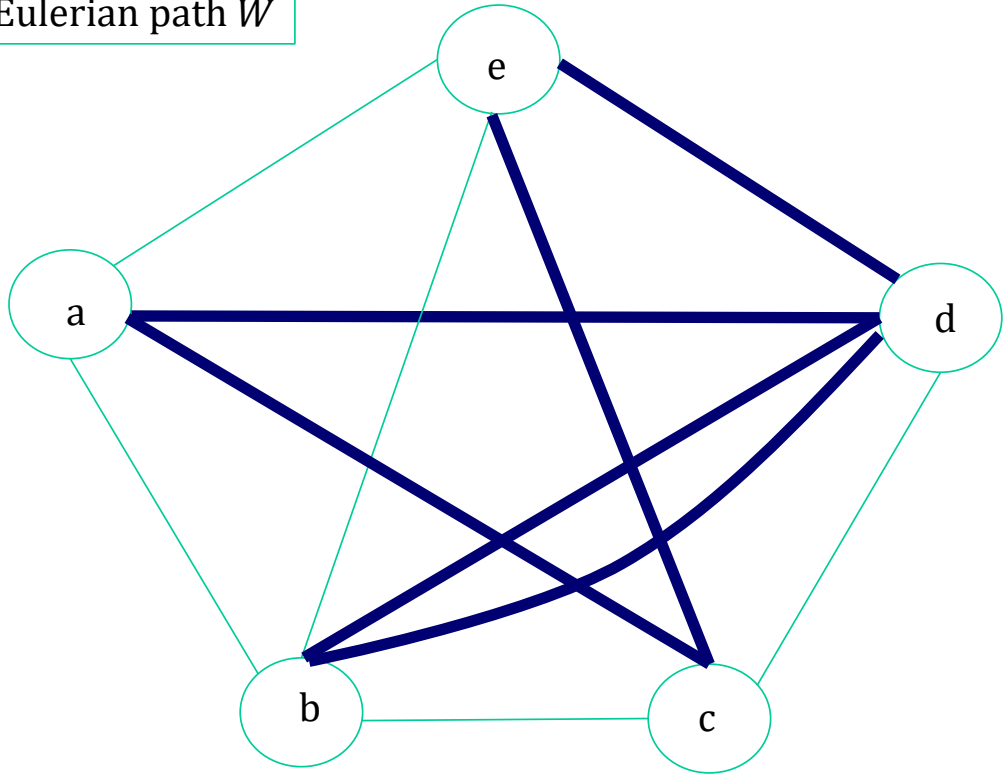
Example

Add K to M



Example

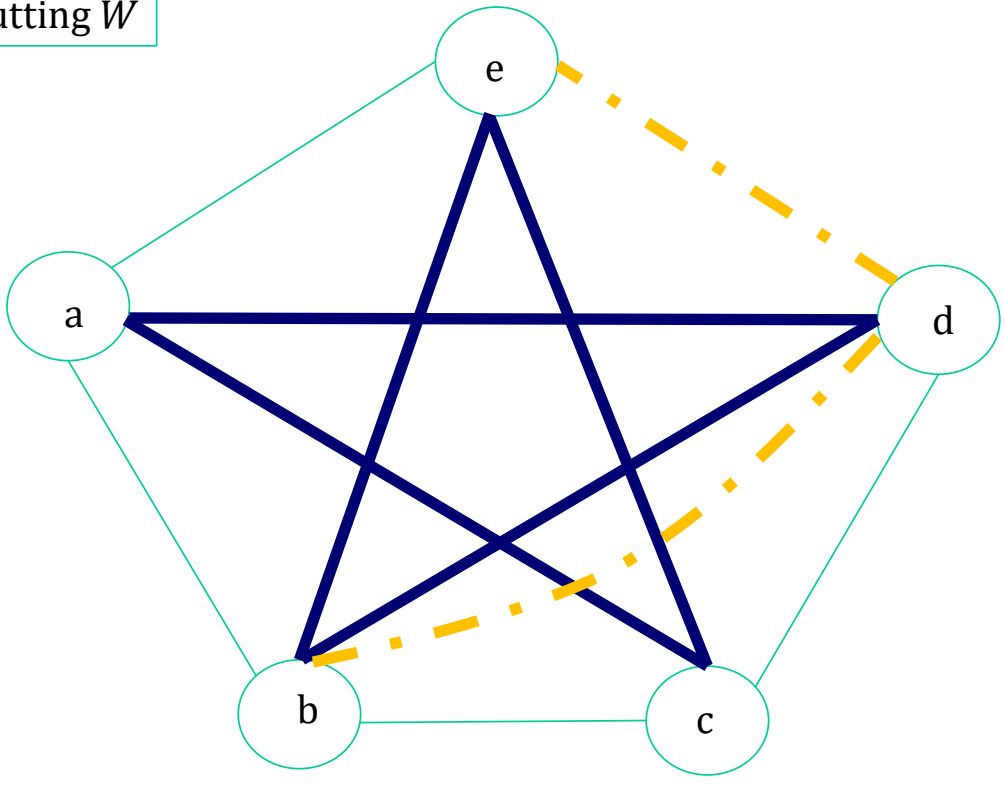
Find a Eulerian path W



$a \rightarrow d \rightarrow b \rightarrow d \rightarrow e \rightarrow c \rightarrow a$

Example

Shortcutting *W*



Before shortcutting: a → d → b → d → e → c → a

After shortcutting: a → d → b → e → c → a

1.5-Approximation

Lemma: The number of odd degree vertices in M is even.

Proof: Let $V_{\text{even}} \subset V$ and $V_{\text{odd}} \subset V$ be the subsets of even and odd degree vertcies in M, respectively

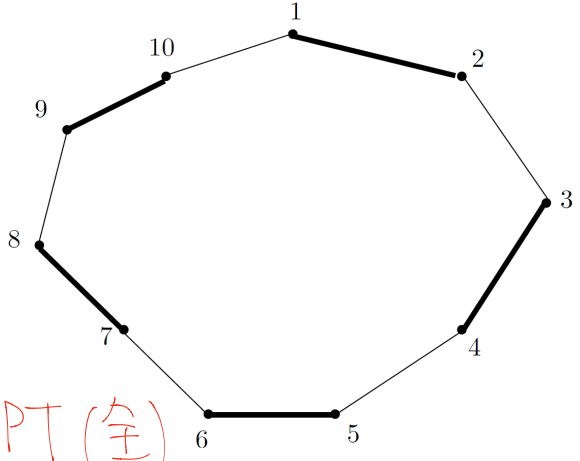
$$2|E| = \sum_{\vartheta \in V} \deg(\vartheta) = \sum_{\vartheta \in V_{odd}} \deg(\vartheta) + \underbrace{\sum_{\vartheta \in V_{even}} \deg(\vartheta)}_{\text{even}} = \text{even}$$

Hence $|V_{\text{odd}}|$ is even.

1.5-Approximation

Lemma: The minimum cost matching on any set U (even number of vertices) is at most $\frac{1}{2}OPT$, where OPT is the cost of the optimal TSP tour.

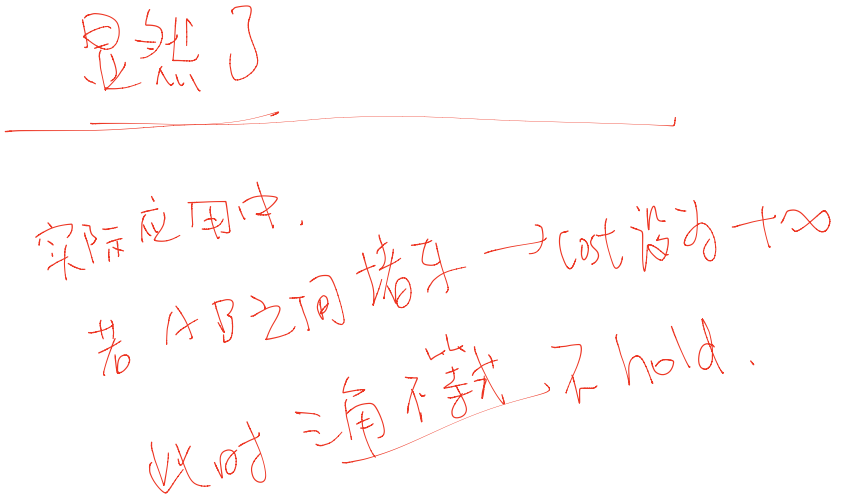
Proof: Consider the optimal TSP tour O and shortcut over all vertices not in U to obtain cycle O' containing vertices U. By triangle inequality, the cost of O' is at most that of O which is OPT. We define two candidate matchings on U using O'. By renumbering vertices let O' be the sequence $1, 2, \cdots, |U|, 1$ of vertices. Let M_1 be the matching that pairs vertices as $(1,2), (3,4) \cdots (|U|-1, |U|)$ and M_2 be $(|U|, 1), (2,3) \cdots (|U|-2, |U|-1)$. Then $\text{cost}(M_1) + \text{cost}(M_2) = \text{cost}(O') \leq OPT$. So $\min(\text{cost}(M_1), \text{cost}(M_2)) \leq OPT/2$.



1.5-Approximation

Theorem: Christofides' algorithm for TSP is a 3/2-approximation algorithm.

Proof: We know that the Cost$(MST) \leq OPT$, and that the min-cost matching has Cost$(K) \leq OPT/2$. So the cost of D' (and hence T') is at most $\frac{3}{2}OPT$.



Python Time

- pip install Christofides

Use the compute() function which takes as input a distance_matrix and returns a Christofides solution as follows:

```
from Christofides import christofides
TSP = christofides.compute(distance_matrix)
```

The Distance Matrix is an upper Triangular matrix with distance from a node on to itself 0, since Christofides algorithm could only be applied for undirected graphs. Also the distance between a node on to itself is practically 0. Example for distance_matrix is as follows, distance_matrix =

```
[[0,45,65,15],
 [0,0,56,12],
 [0,0,0,89],
 [0,0,0,0]]
```