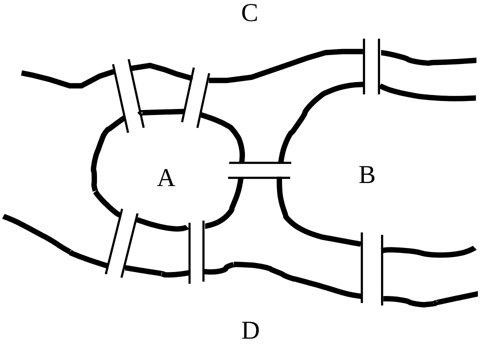
Optimal Graph Traversals

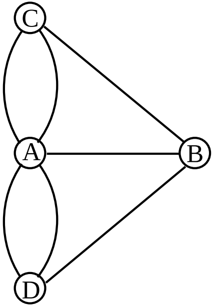
Stanislav Palúch

Fakulta riadenia a informatiky, Žilinská univerzita

5. mája 2016

Problem of 7 bridges in Kaliningrad – K oningsberg
Leonhard Euler – 1736





Eulerian Trails and Tours

Definition

An **Eulerian walk** $s(u, v)$ *in a connected graph* $G = (V, H)$ *is a walk that contains every edge of that graph.*

An Eulerian tour *is a closed Eulerian trail.*

An Eulerian graph is a graph that contains an Eulerian tour.

Remark

A trail is a special case of a walk, therefore the above definiton fully specifies **an Eulerian trail** *in graph* G *as a trail that contains every edge of that graph.*

Remark

An Eulerian trail $t(u, v)$ *contains every edge of graph* G *exactly once therefore the sequence of vertices and edges of that trail represents a way how to draw the diagram of* G *by one pencil stroke.*

Euler's Theorem

Theorem

(Euler, 1736.) *A connected graf $G = (V, H)$ is Eulerian if and only if it has all vertices of even degree.*
A connected graf $G = (V, H)$ has an open Eulerian tour if and only if it has exactly two vertices of odd degree.

PROOF.

1. If there exists a closed Eulerian tour $\mathcal{T}$ in graph G then the the degree of every vertex i is even since the number of edges of $\mathcal{T}$ incomming into i is equal to the number of edges outgoing from i.

2. Constructing an Eulerian tour in a connected graph is described by the following Algorithm:

Algorithm

- **Step 1.** *Start at any vertex s, set $\mathcal{T} := (s)$ and step by step extend the trail $\mathcal{T}$ by an unused edge till possible.*
  The last vertex of $\mathcal{T}$ is s – trail $\mathcal{T}$ is closed.
- **Step 2.** *Choose the first vertex v of $\mathcal{T}$ that is incident with an unused edge in trail $\mathcal{T}$.*

  If such vertex v does not exist then STOP.

  Trail $\mathcal{T}$ is the desired Eulerian tour.
- **Step 3.** *Create a trail $\mathcal{S}$ as follows:*

  Set $\mathcal{S} := (v)$ and step by step extend the trail $\mathcal{S}$ by an unused edge till possible. Last vertex of $\mathcal{S}$ is v – $\mathcal{S}$ is closed trail.
- **Step 4.** *Split trail $\mathcal{T}$ into s–v trail $\mathcal{T}_1$ and v–s trail $\mathcal{T}_2$, i. e. $\mathcal{T} = \mathcal{T}_1 \oplus \mathcal{T}_2$.*

  Set $\mathcal{T} = \mathcal{T}_1 \oplus \mathcal{S} \oplus \mathcal{T}_2$.

  The new trail $\mathcal{T}$ is concatenation of trails $\mathcal{T}_1$, $\mathcal{S}$ and $\mathcal{T}_2$.

  GOTO Step 2.

Algorithm

- **Step 1.** *Start at any vertex s, set $\mathcal{T} := (s)$ and step by step extend the trail $\mathcal{T}$ by an unused edge till possible.*
  The last vertex of $\mathcal{T}$ is s – trail $\mathcal{T}$ is closed.
- **Step 2.** *Choose the first vertex v of $\mathcal{T}$ that is incident with an unused edge in trail $\mathcal{T}$.*

  If such vertex v does not exist then STOP.

  Trail $\mathcal{T}$ is the desired Eulerian tour.
- **Step 3.** *Create a trail $\mathcal{S}$ as follows:*

  Set $\mathcal{S} := (v)$ and step by step extend the trail $\mathcal{S}$ by an unused edge till possible. Last vertex of $\mathcal{S}$ is v – $\mathcal{S}$ is closed trail.
- **Step 4.** *Split trail $\mathcal{T}$ into s–v trail $\mathcal{T}_1$ and v–s trail $\mathcal{T}_2$, i. e. $\mathcal{T} = \mathcal{T}_1 \oplus \mathcal{T}_2$.*

  Set $\mathcal{T} = \mathcal{T}_1 \oplus \mathcal{S} \oplus \mathcal{T}_2$.

  The new trail $\mathcal{T}$ is concatenation of trails $\mathcal{T}_1$, $\mathcal{S}$ and $\mathcal{T}_2$.

  GOTO Step 2.

Algorithm

- **Step 1.** *Start at any vertex s, set $\mathcal{T} := (s)$ and step by step extend the trail $\mathcal{T}$ by an unused edge till possible.*
  The last vertex of $\mathcal{T}$ is s – trail $\mathcal{T}$ is closed.
- **Step 2.** *Choose the first vertex v of $\mathcal{T}$ that is incident with an unused edge in trail $\mathcal{T}$.*

  If such vertex v does not exist then STOP.

  Trail $\mathcal{T}$ is the desired Eulerian tour.
- **Step 3.** *Create a trail $\mathcal{S}$ as follows:*

  Set $\mathcal{S} := (v)$ and step by step extend the trail $\mathcal{S}$ by an unused edge till possible. Last vertex of $\mathcal{S}$ is v – $\mathcal{S}$ is closed trail.
- **Step 4.** *Split trail $\mathcal{T}$ into s–v trail $\mathcal{T}_1$ and v–s trail $\mathcal{T}_2$, i. e. $\mathcal{T} = \mathcal{T}_1 \oplus \mathcal{T}_2$.*

  Set $\mathcal{T} = \mathcal{T}_1 \oplus \mathcal{S} \oplus \mathcal{T}_2$.

  The new trail $\mathcal{T}$ is concatenation of trails $\mathcal{T}_1$, $\mathcal{S}$ and $\mathcal{T}_2$.

  GOTO Step 2.

Algorithm

- **Step 1.** *Start at any vertex* s*, set* $\mathcal{T} := (s)$ *and step by step extend the trail* $\mathcal{T}$ *by an unused edge till possible.*
  The last vertex of $\mathcal{T}$ *is* s *– trail* $\mathcal{T}$ *is closed.*
- **Step 2.** *Choose the first vertex* v *of* $\mathcal{T}$ *that is incident with an unused edge in trail* $\mathcal{T}$.

  If such vertex v *does not exist then STOP.*

  Trail $\mathcal{T}$ *is the desired Eulerian tour.*
- **Step 3.** *Create a trail* $\mathcal{S}$ *as follows:*

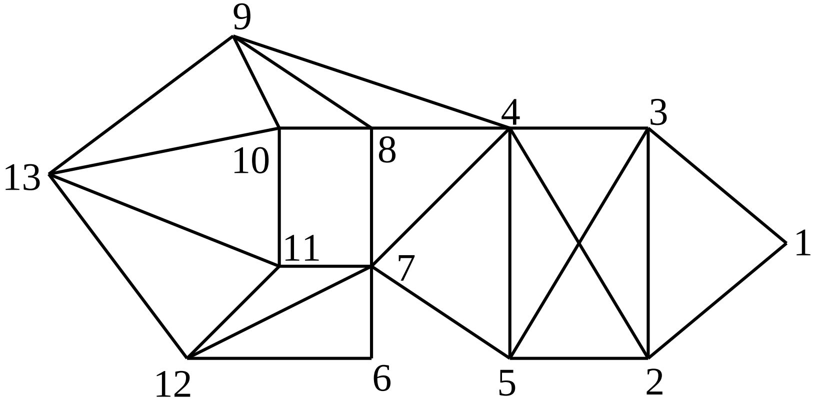
  Set $\mathcal{S} := (v)$ *and step by step extend the trail* $\mathcal{S}$ *by an unused edge till possible. Last vertex of* $\mathcal{S}$ *is* v *–* $\mathcal{S}$ *is closed trail.*
- **Step 4.** *Split trail* $\mathcal{T}$ *into* s*–*v *trail* $\mathcal{T}_1$ *and* v*–*s *trail* $\mathcal{T}_2$*, i. e.* $\mathcal{T} = \mathcal{T}_1 \oplus \mathcal{T}_2$.

  Set $\mathcal{T} = \mathcal{T}_1 \oplus \mathcal{S} \oplus \mathcal{T}_2$.

  The new trail $\mathcal{T}$ *is concatenation of trails* $\mathcal{T}_1$*,* $\mathcal{S}$ *and* $\mathcal{T}_2$.

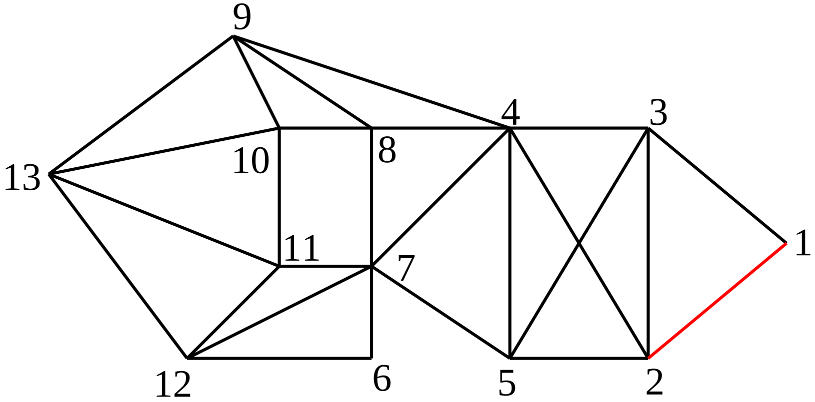
  GOTO Step 2.

Example



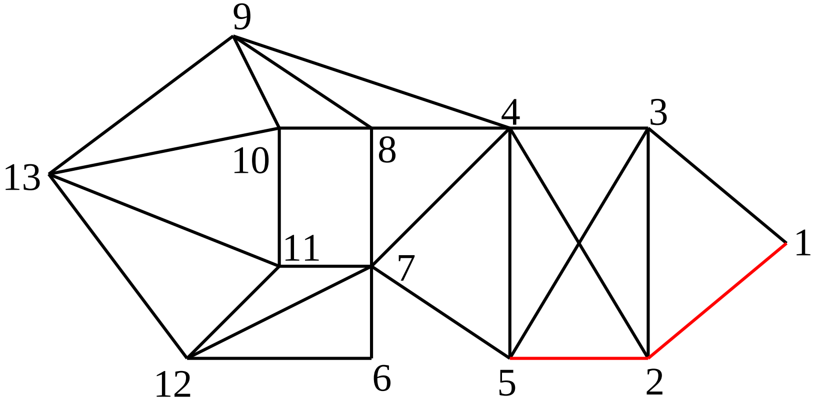
$\mathcal{T} = (1)$

Example



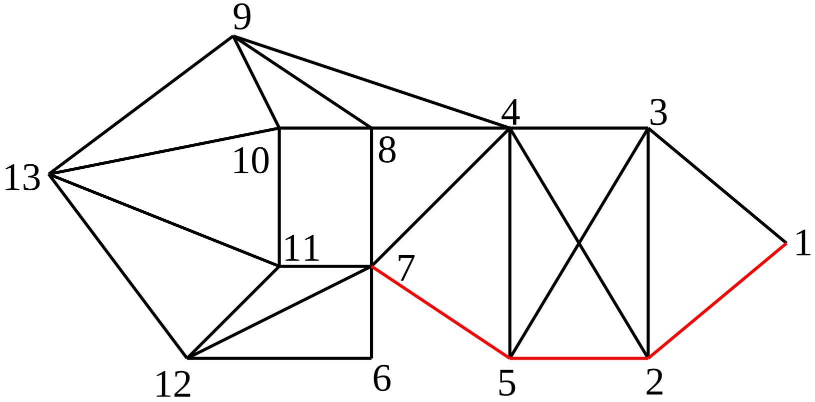
$\mathcal{T} = (1, 2)$

Example



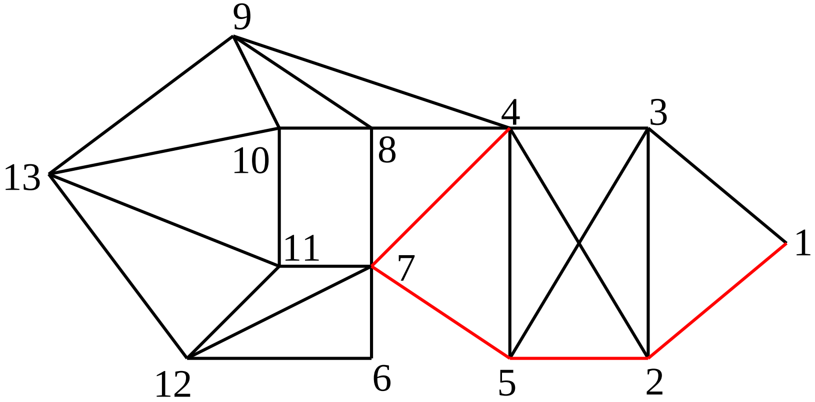
$\mathcal{T} = (1, 2, 5)$

Example



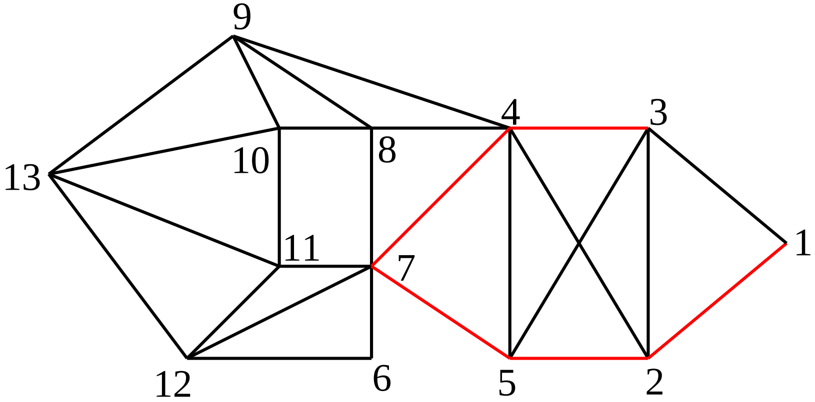
$\mathcal{T} = (1, 2, 5, 7)$

Example



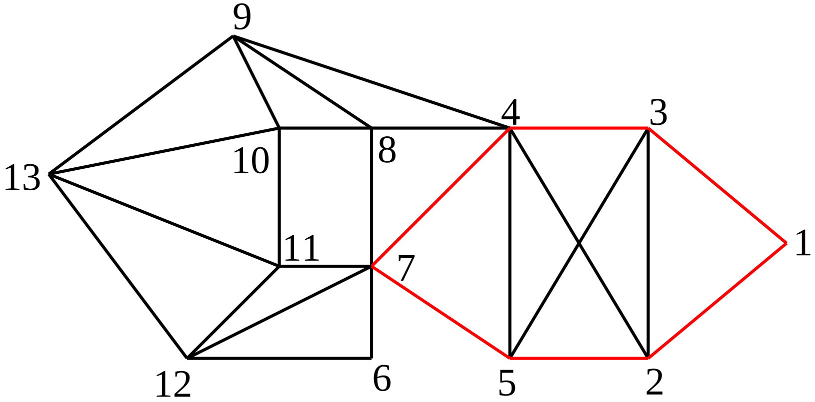
$\mathcal{T} = (1, 2, 5, 7, 4)$

Example



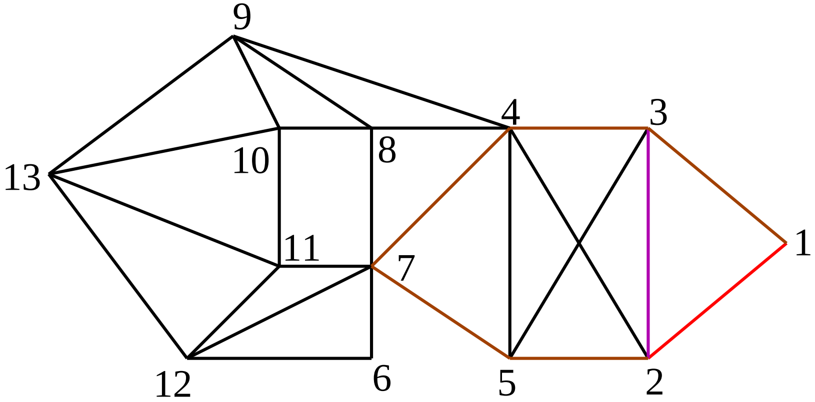
$\mathcal{T} = (1, 2, 5, 7, 4, 3)$

Example



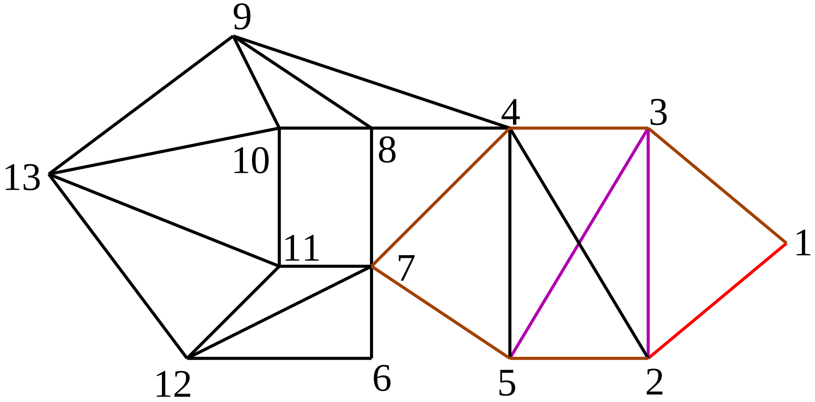
$\mathcal{T} = (1, 2, 5, 7, 4, 3, 1)$

Example



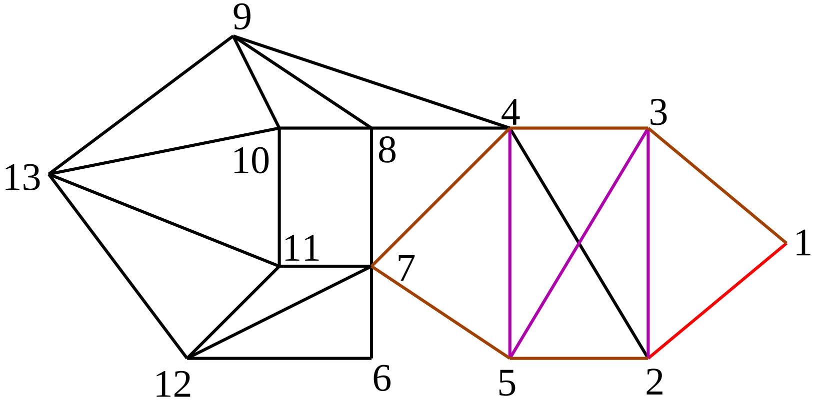
$\mathcal{T} = (1, 2, 5, 7, 4, 3, 1)$
$\mathcal{T}_1 = (1, 2)$, $\mathcal{T}_2 = (2, 5, 7, 4, 3, 1)$
$\mathcal{S} = (2, 3)$

Example



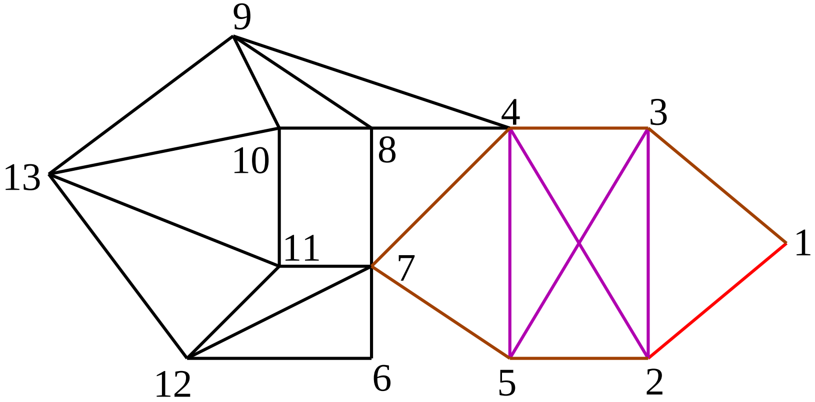
$\mathcal{T} = (1, 2, 5, 7, 4, 3, 1)$
$\mathcal{T}_1 = (1, 2)$, $\mathcal{T}_2 = (2, 5, 7, 4, 3, 1)$
$\mathcal{S} = (2, 3, 5)$

Example



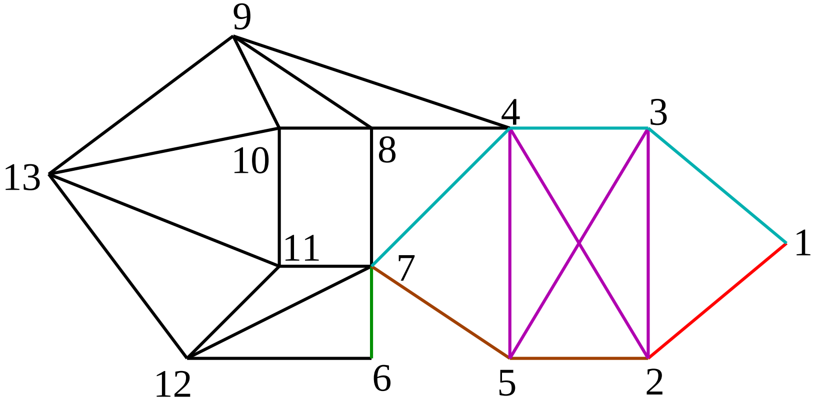
$\mathcal{T} = (1, 2, 5, 7, 4, 3, 1)$
$\mathcal{T}_1 = (1, 2)$, $\mathcal{T}_2 = (2, 5, 7, 4, 3, 1)$
$\mathcal{S} = (2, 3, 5, 4)$

Example



$\mathcal{T} = (1, 2, 5, 7, 4, 3, 1)$
$\mathcal{T}_1 = (1, 2)$, $\mathcal{T}_2 = (2, 5, 7, 4, 3, 1)$
$\mathcal{S} = (2, 3, 5, 4, 2)$

Example

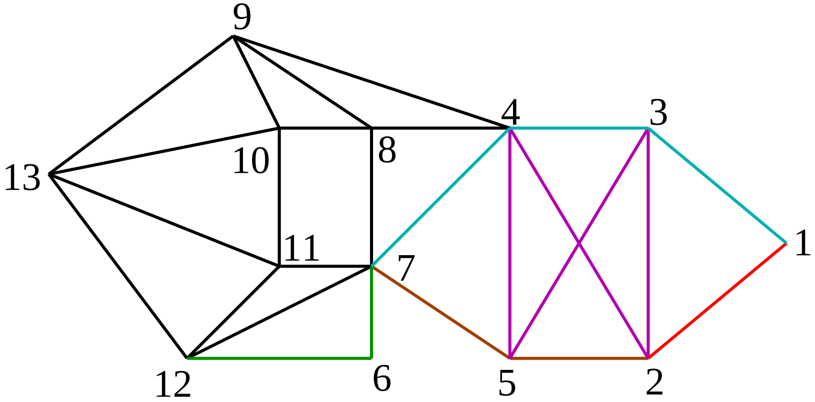


$\mathcal{T} = (1, \underbrace{2, 3, 5, 4, 2}_{\mathcal{S}}, 5, 7, 4, 3, 1)$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7)$, $\mathcal{T}_2 = (7, 4, 3, 1)$

$\mathcal{S} = (7, 6)$

Example

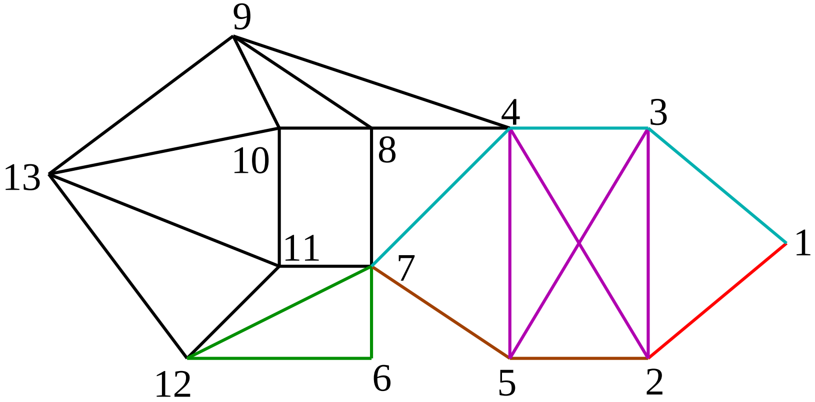


$\mathcal{T} = (1, \underbrace{2, 3, 5, 4, 2}_{\mathcal{S}}, 5, 7, 4, 3, 1)$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7)$, $\mathcal{T}_2 = (7, 4, 3, 1)$

$\mathcal{S} = (7, 6, 12)$

Example

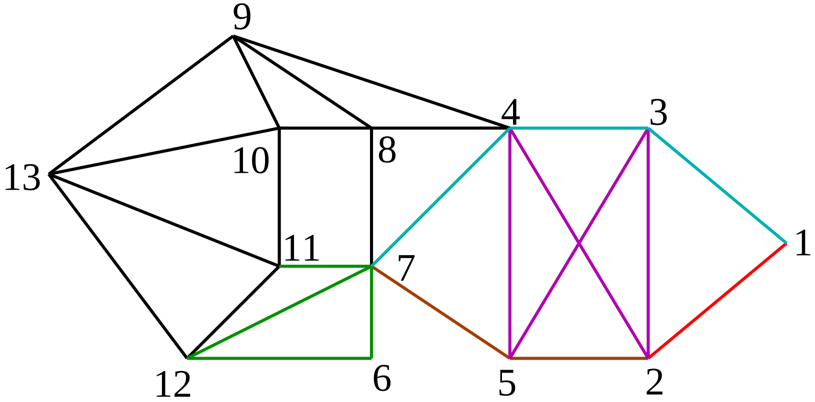


$\mathcal{T} = (1, \underbrace{2, 3, 5, 4, 2}_{\mathcal{S}}, 5, 7, 4, 3, 1)$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7)$, $\mathcal{T}_2 = (7, 4, 3, 1)$

$\mathcal{S} = (7, 6, 12, 7)$

Example

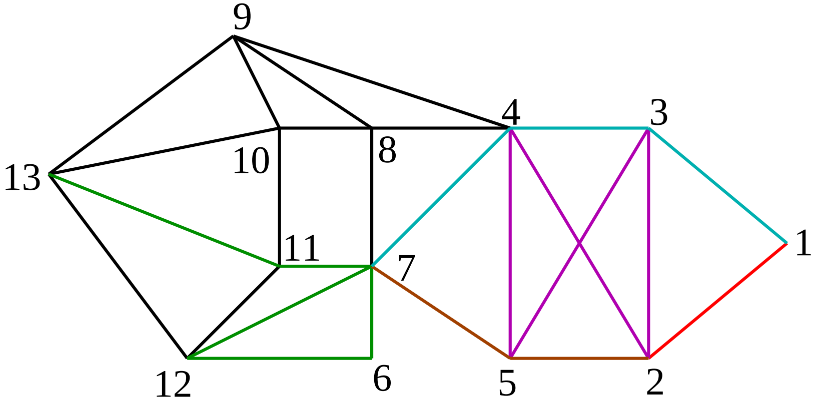


$\mathcal{T} = (1, \underbrace{2, 3, 5, 4, 2}_{\mathcal{S}}, 5, 7, 4, 3, 1)$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7)$, $\mathcal{T}_2 = (7, 4, 3, 1)$

$\mathcal{S} = (7, 6, 12, 7, 11)$

Example

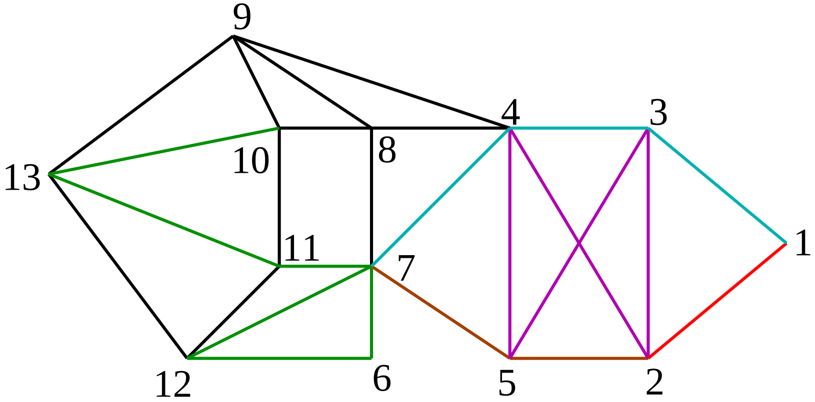


$\mathcal{T} = (1, \underbrace{2, 3, 5, 4, 2}_{\mathcal{S}}, 5, 7, 4, 3, 1)$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7)$, $\mathcal{T}_2 = (7, 4, 3, 1)$

$\mathcal{S} = (7, 6, 12, 7, 11, 13)$

Example

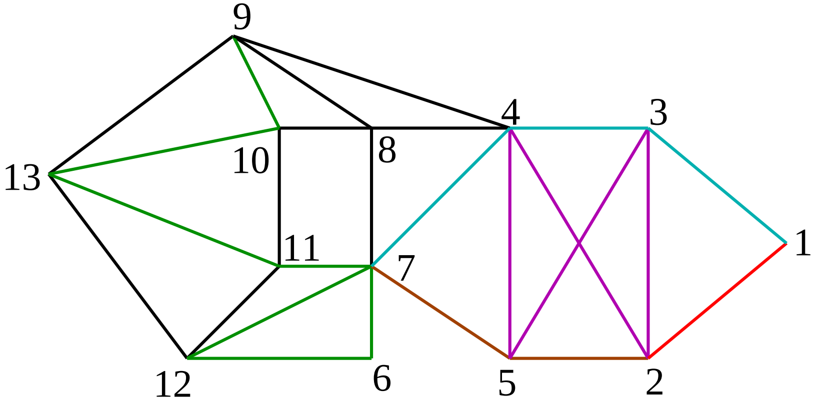


$\mathcal{T} = (1, \underbrace{2, 3, 5, 4, 2}_{\mathcal{S}}, 5, 7, 4, 3, 1)$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7)$, $\mathcal{T}_2 = (7, 4, 3, 1)$

$\mathcal{S} = (7, 6, 12, 7, 11, 13, 10)$

Example

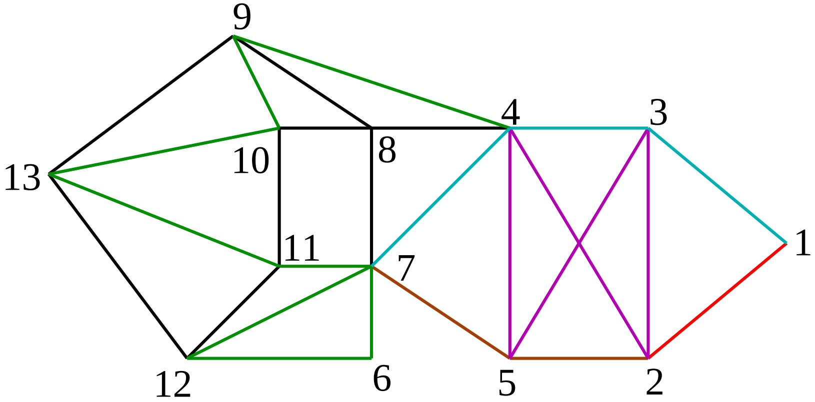


$\mathcal{T} = (1, \underbrace{2, 3, 5, 4, 2}_{\mathcal{S}}, 5, 7, 4, 3, 1)$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7)$, $\mathcal{T}_2 = (7, 4, 3, 1)$

$\mathcal{S} = (7, 6, 12, 7, 11, 13, 10, 9)$

Example

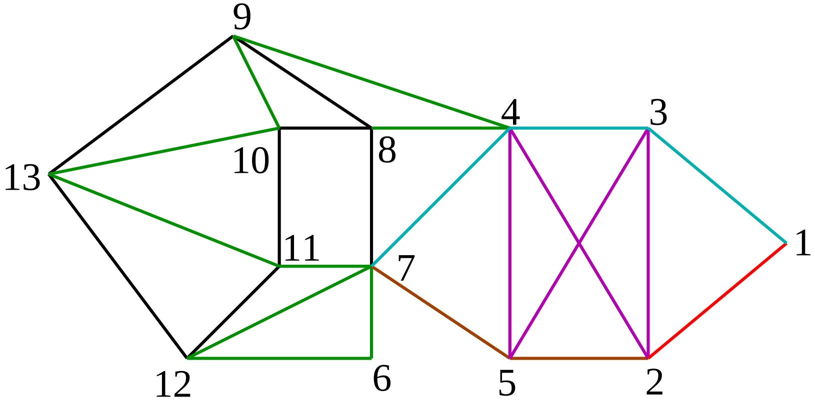


$\mathcal{T} = (1, \underbrace{2, 3, 5, 4, 2}_{\mathcal{S}}, 5, 7, 4, 3, 1)$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7)$, $\mathcal{T}_2 = (7, 4, 3, 1)$

$\mathcal{S} = (7, 6, 12, 7, 11, 13, 10, 9, 4)$

Example

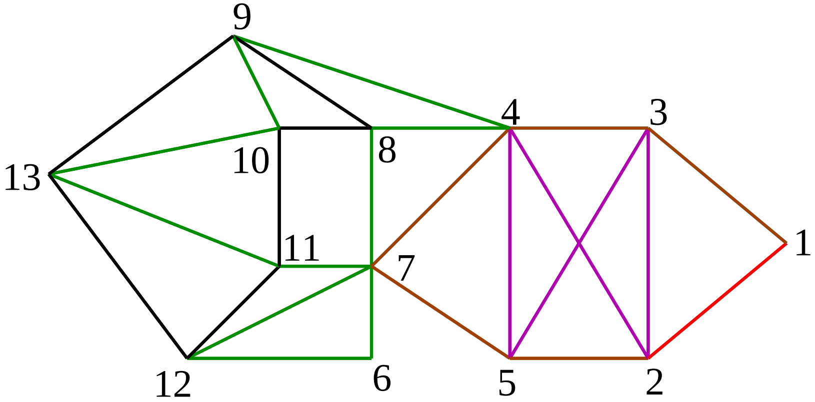


$\mathcal{T} = (1, \underbrace{2, 3, 5, 4, 2}_{\mathcal{S}}, 5, 7, 4, 3, 1)$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7)$, $\mathcal{T}_2 = (7, 4, 3, 1)$

$\mathcal{S} = (7, 6, 12, 7, 11, 13, 10, 9, 4, 8)$

Example

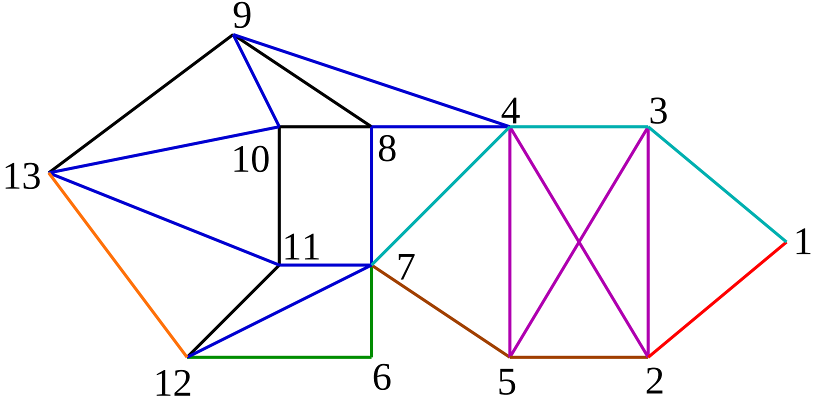


$\mathcal{T} = (1, \underbrace{2, 3, 5, 4, 2}_{\mathcal{S}}, 5, 7, 4, 3, 1)$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7)$, $\mathcal{T}_2 = (7, 4, 3, 1)$

$\mathcal{S} = (7, 6, 12, 7, 11, 13, 10, 9, 4, 8, 7)$

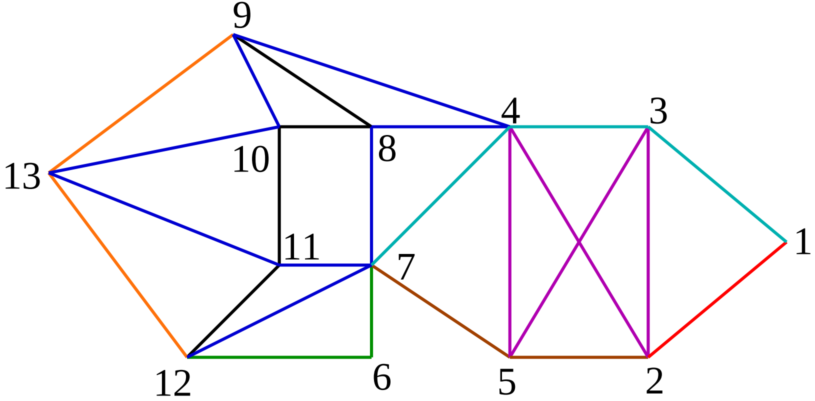
Example



$$\mathcal{T} = (1,2,3,5,4,2,5,\underbrace{7,6,12,7,11,13,10,9,4,8,7}_{\mathcal{S}},4,3,1)$$

$\mathcal{T}_1 = (1,2,3,5,4,2,5,7,6,12)$, $\mathcal{T}_2 = (12,7,11,13,10,9,4,8,7,4,3,1)$
$\mathcal{S} = (12,13)$

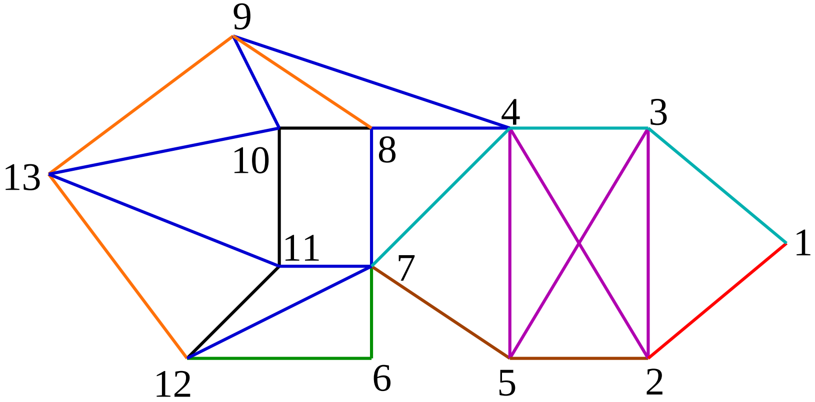
Example



$\mathcal{T} = (1, 2, 3, 5, 4, 2, 5, \underbrace{7, 6, 12, 7, 11, 13, 10, 9, 4, 8, 7}_{\mathcal{S}}, 4, 3, 1)$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7, 6, 12)$, $\mathcal{T}_2 = (12, 7, 11, 13, 10, 9, 4, 8, 7, 4, 3, 1)$
$\mathcal{S} = (12, 13, 9)$

Example

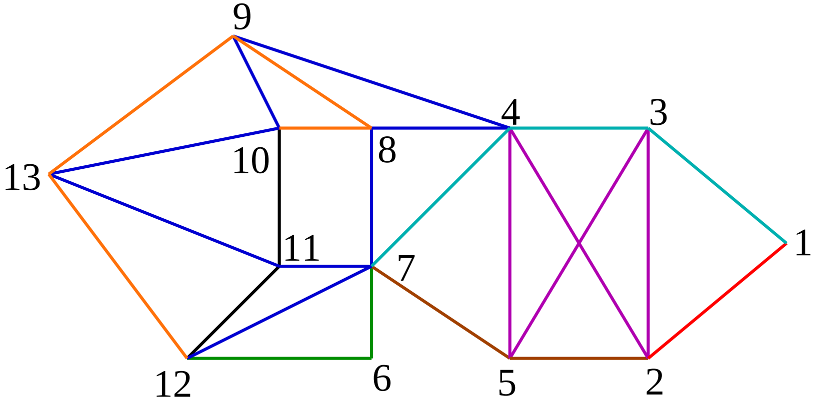


$$\mathcal{T} = (1, 2, 3, 5, 4, 2, 5, \underbrace{7, 6, 12, 7, 11, 13, 10, 9, 4, 8, 7}_{\mathcal{S}}, 4, 3, 1)$$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7, 6, 12)$, $\mathcal{T}_2 = (12, 7, 11, 13, 10, 9, 4, 8, 7, 4, 3, 1)$

$\mathcal{S} = (12, 13, 9, 8)$

Example

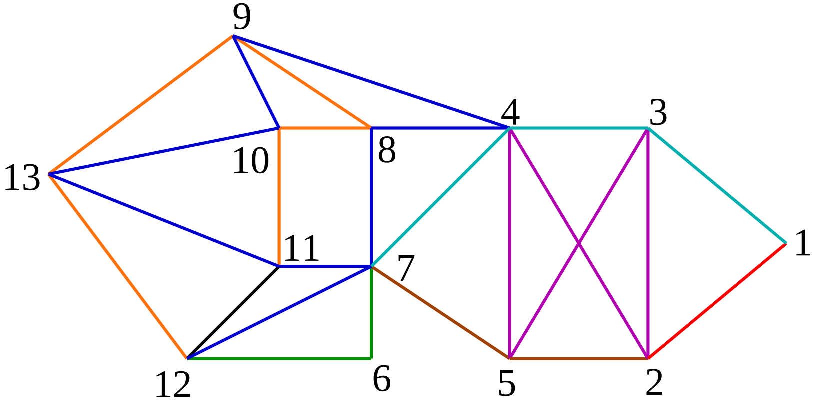


$$\mathcal{T} = (1,2,3,5,4,2,5,\underbrace{7,6,12,7,11,13,10,9,4,8,7}_{\mathcal{S}},4,3,1)$$

$\mathcal{T}_1 = (1,2,3,5,4,2,5,7,6,12)$, $\mathcal{T}_2 = (12,7,11,13,10,9,4,8,7,4,3,1)$

$\mathcal{S} = (12,13,9,8,10)$

Example

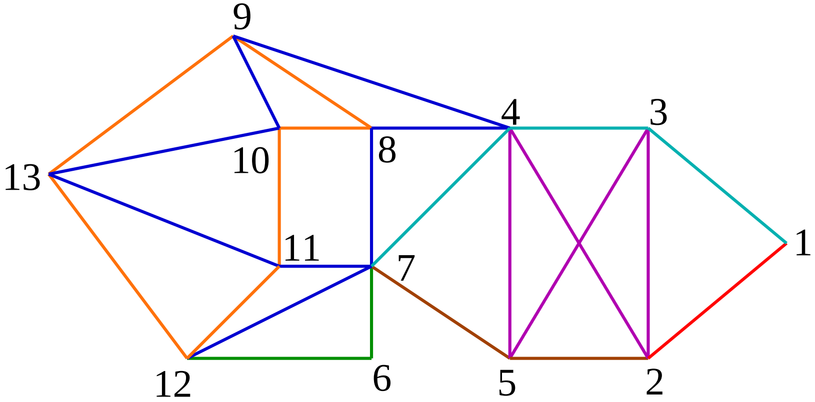


$$\mathcal{T} = (1,2,3,5,4,2,5,\underbrace{7,6,12,7,11,13,10,9,4,8,7}_{\mathcal{S}},4,3,1)$$

$\mathcal{T}_1 = (1,2,3,5,4,2,5,7,6,12)$, $\mathcal{T}_2 = (12,7,11,13,10,9,4,8,7,4,3,1)$

$\mathcal{S} = (12,13,9,8,10,11)$

Example

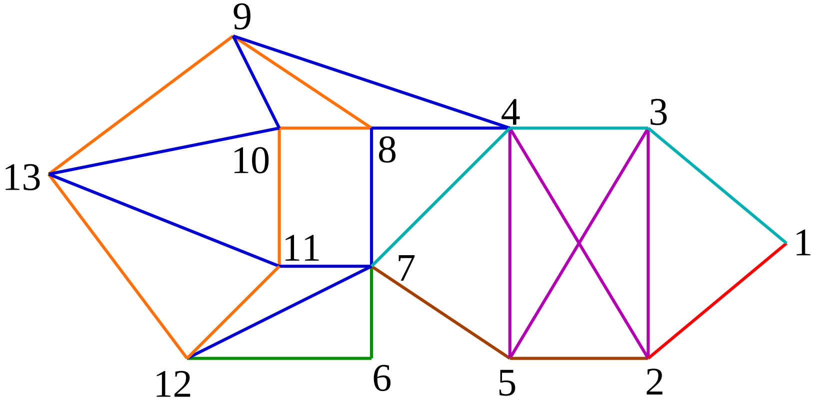


$\mathcal{T} = (1, 2, 3, 5, 4, 2, 5, \underbrace{7, 6, 12, 7, 11, 13, 10, 9, 4, 8, 7}_{\mathcal{S}}, 4, 3, 1)$

$\mathcal{T}_1 = (1, 2, 3, 5, 4, 2, 5, 7, 6, 12)$, $\mathcal{T}_2 = (12, 7, 11, 13, 10, 9, 4, 8, 7, 4, 3, 1)$

$\mathcal{S} = (12, 13, 9, 8, 10, 11, 12)$

Example



$$\mathcal{T} = (1, 2, 3, 5, 4, 2, 5, 7, 6, \underbrace{12, 13, 9, 8, 10, 11, 12}_{\mathcal{S}}, 7, 11, 13, 10, 9, 4, 8, 7, 4, 3, 1)$$

Algorithm

Fleury's Algorithm to find an Eulerian tour in a connected graph $G = (V, H)$ with all vertices of even degree.

- **Step 1.** *Start at arbitrary vertex s and insert into trail $\mathcal{T}$ arbitrary edge incident with v.*
- **Step 2.** *If all edges of G are used in $\mathcal{T}$ then STOP.*
- **Step 3.** *Extend the trail $\mathcal{T}$ by such an edge incident with last vertex of $\mathcal{T}$ after removing it the subgraph $\overline{G}$ consisting of unused edges and corresponding incident vertices[a] does not contain:*
  - *two nontrivial components (i.e is disconnected) or*
  - *nontrivial component which does not contain starting vertex s of trail $\mathcal{T}$.*
- *GOTO Step 2.*

♣

[a] $\overline{G}$ is a subgraph of G induced by the set of unused eges

Algorithm

Fleury's Algorithm to find an Eulerian tour in a connected graph $G = (V, H)$ with all vertices of even degree.

- **Step 1.** *Start at arbitrary vertex s and insert into trail $\mathcal{T}$ arbitrary edge incident with v.*
- **Step 2.** *If all edges of G are used in $\mathcal{T}$ then STOP.*
- **Step 3.** *Extend the trail $\mathcal{T}$ by such an edge incident with last vertex of $\mathcal{T}$ after removing it the subgraph $\overline{G}$ consisting of unused edges and corresponding incident vertices[a] does not contain:*
  - *two nontrivial components (i.e is disconnected) or*
  - *nontrivial component which does not contain starting vertex s of trail $\mathcal{T}$.*
- *GOTO Step 2.*

♣

[a] $\overline{G}$ is a subgraph of G induced by the set of unused eges

Algorithm

Fleury's Algorithm to find an Eulerian tour in a connected graph $G = (V, H)$ with all vertices of even degree.

- **Step 1.** *Start at arbitrary vertex s and insert into trail $\mathcal{T}$ arbitrary edge incident with v.*
- **Step 2.** *If all edges of G are used in $\mathcal{T}$ then STOP.*
- **Step 3.** *Extend the trail $\mathcal{T}$ by such an edge incident with last vertex of $\mathcal{T}$ after removing it the subgraph $\overline{G}$ consisting of unused edges and corresponding incident vertices[a] does not contain:*
  - *two nontrivial components (i.e is disconnected) or*
  - *nontrivial component which does not contain starting vertex s of trail $\mathcal{T}$.*
- *GOTO Step 2.*

♣

[a] $\overline{G}$ is a subgraph of G induced by the set of unused eges

Algorithm

Fleury's Algorithm to find an Eulerian tour in a connected graph $G = (V, H)$ with all vertices of even degree.

- **Step 1.** *Start at arbitrary vertex s and insert into trail $\mathcal{T}$ arbitrary edge incident with v.*
- **Step 2.** *If all edges of G are used in $\mathcal{T}$ then STOP.*
- **Step 3.** *Extend the trail $\mathcal{T}$ by such an edge incident with last vertex of $\mathcal{T}$ after removing it the subgraph $\overline{G}$ consisting of unused edges and corresponding incident vertices*[a] *does not contain:*
  - *two nontrivial components (i.e is disconnected) or*
  - *nontrivial component which does not contain starting vertex s of trail $\mathcal{T}$.*
- *GOTO Step 2.*

♣

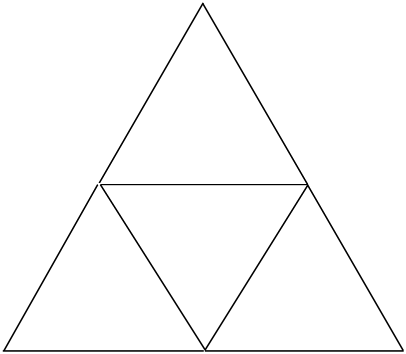
[a] $\overline{G}$ is a subgraph of G induced by the set of unused eges

Fleury's Algorithm

Remark

The idea is **„ don't burn bridges“** *so that we can come back to a starting vertex and traverse remaining edges.*

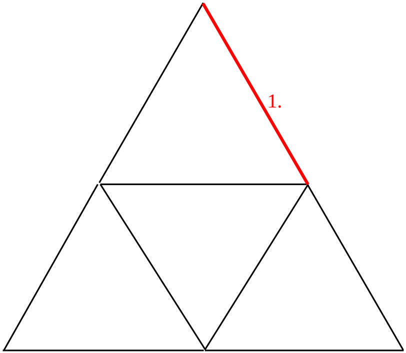
Fleury's Algoritmus – Example



Remark

Checking whether subraph of G induced by the set of unused edges is connected or whether it contains starting vertex of trial $\mathcal{T}$ is intuitive by manual drawing.
An exact algorithm for this checking should be designed for computer implementation of Fleury's algorithm.

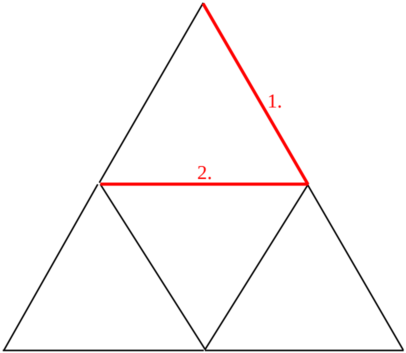
Fleury's Algoritmus – Example



Remark

Checking whether subraph of G induced by the set of unused edges is connected or whether it contains starting vertex of trial $\mathcal{T}$ is intuitive by manual drawing.
An exact algorithm for this checking should be designed for computer implementation of Fleury's algorithm.

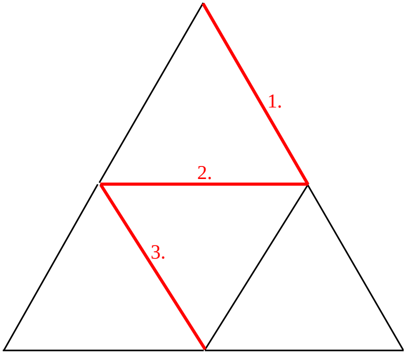
Fleury's Algoritmus – Example



Remark

Checking whether subraph of G induced by the set of unused edges is connected or whether it contains starting vertex of trial $\mathcal{T}$ is intuitive by manual drawing.
An exact algorithm for this checking should be designed for computer implementation of Fleury's algorithm.

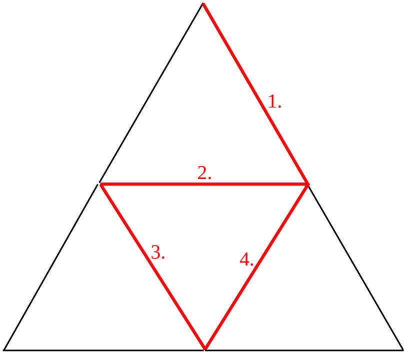
Fleury's Algoritmus – Example



Remark

Checking whether subraph of G induced by the set of unused edges is connected or whether it contains starting vertex of trial $\mathcal{T}$ is intuitive by manual drawing.
An exact algorithm for this checking should be designed for computer implementation of Fleury's algorithm.

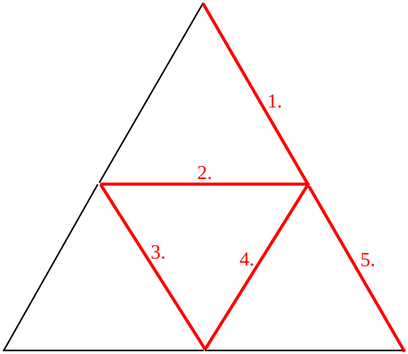
Fleury's Algoritmus – Example



Remark

Checking whether subraph of G induced by the set of unused edges is connected or whether it contains starting vertex of trial $\mathcal{T}$ is intuitive by manual drawing.
An exact algorithm for this checking should be designed for computer implementation of Fleury's algorithm.

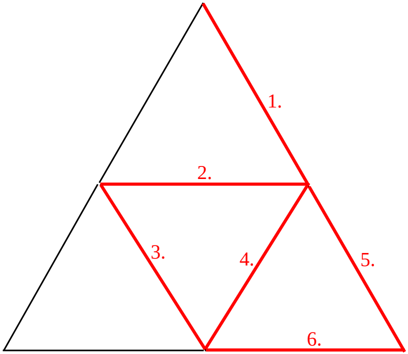
Fleury's Algoritmus – Example



Remark

Checking whether subraph of G induced by the set of unused edges is connected or whether it contains starting vertex of trial $\mathcal{T}$ is intuitive by manual drawing.
An exact algorithm for this checking should be designed for computer implementation of Fleury's algorithm.

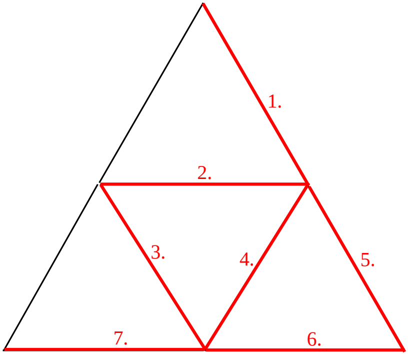
Fleury's Algoritmus – Example



Remark

Checking whether subraph of G induced by the set of unused edges is connected or whether it contains starting vertex of trial $\mathcal{T}$ is intuitive by manual drawing.
An exact algorithm for this checking should be designed for computer implementation of Fleury's algorithm.

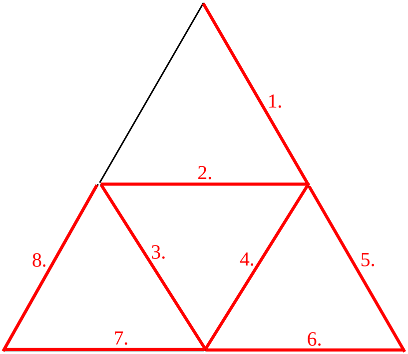
Fleury's Algoritmus – Example



Remark

Checking whether subraph of G induced by the set of unused edges is connected or whether it contains starting vertex of trial $\mathcal{T}$ is intuitive by manual drawing.
An exact algorithm for this checking should be designed for computer implementation of Fleury's algorithm.

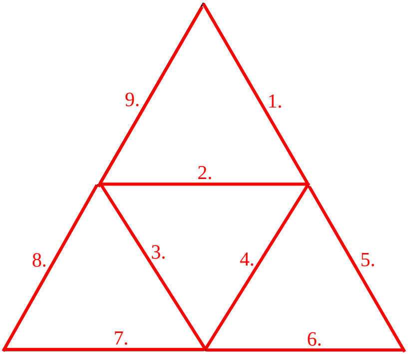
Fleury's Algoritmus – Example



Remark

Checking whether subraph of G induced by the set of unused edges is connected or whether it contains starting vertex of trial $\mathcal{T}$ is intuitive by manual drawing.
An exact algorithm for this checking should be designed for computer implementation of Fleury's algorithm.

Fleury's Algoritmus – Example



Remark

Checking whether subraph of G induced by the set of unused edges is connected or whether it contains starting vertex of trial $\mathcal{T}$ is intuitive by manual drawing.
An exact algorithm for this checking should be designed for computer implementation of Fleury's algorithm.

Algorithm

Labyrint Algorithm to find and Eulerian tour in a connected graph $G = (V, H)$ with all vertices of even degree.

- **Step 1.** *Start at arbitrary vertex* $u \in V$, *set* $\mathcal{S} = \{(s)\}$.

  Walk $\mathcal{S}$ *at the beginning consists from single vertex* s.

  Let vertex w *be the last vertex of the walk* $\mathcal{S}$.

Algorithm (– continuation)

- **Step 2.**

  Let vertex w be the last vertex of the walk $\mathcal{S}$.

  Choose a next edge $\{w, v\}$ fulfilling rules L1, L2 and insert it into walk $\mathcal{S}$. Mark the direction of edge $\{w, v\}$ in S.

  If the vertex v was not used in $\mathcal{S}$ denote the edge $\{w, v\}$ as the first access edge - FAE.

  Moreover, create so called **bacward sequence** *– order of edges in which they appaer in the walk $\mathcal{S}$ for the second time.*

Rules when picking subsequent edge:

(L1): *Every edge can be used in one direction only once.*

(L2): *Precedence of edges:*
– till now not used edges
– edges used once
– first access edge (only if there is no other possibility)

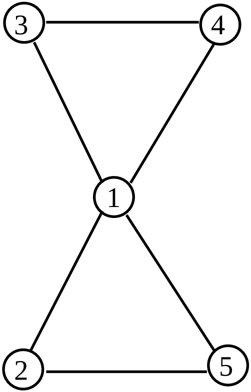
Labyrint Algorithm – continuation

Algorithm (– continuation)

- **Step 3.** *If all edges are used in $\mathcal{S}$ – STOP.*
  Backward sequence defines searched Eulerian tour.
- **Step 4.** *Otherwise set $w := v$.*
  Vertex w is last vertex of actual walk $\mathcal{S}$
  GOTO Step 2.

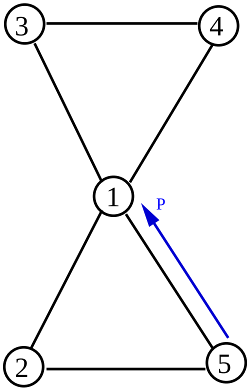
Labyrint algorithm – Example

Edges of $\mathcal{S}$	direction of edge in $\mathcal{S}$ {1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	visited vertex 1	2	3	4	5
-											•
$\{5,1\}$				$\Leftarrow$			•				
$\{1,2\}$	$\Rightarrow$							•			
$\{2,5\}$					$\rightarrow$						
$\boxed{\{5,2\}}$					$\leftarrow$						
$\boxed{\{2,1\}}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							•		
$\{3,4\}$						$\Rightarrow$				•	
$\{4,1\}$			$\leftarrow$								
$\boxed{\{1,4\}}$			$\rightarrow$								
$\boxed{\{4,3\}}$						$\leftarrow$					
$\boxed{\{3,1\}}$		$\leftarrow$									
$\boxed{\{1,5\}}$				$\rightarrow$							



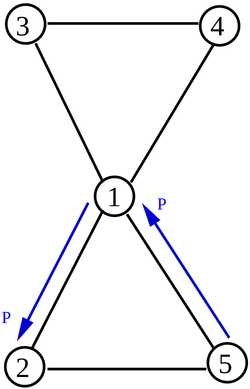
Labyrint algorithm – Example

Edges of $\mathcal{S}$	direction of edge in $\mathcal{S}$ {1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	visited vertex 1	2	3	4	5
-											•
$\{5,1\}$				$\Leftarrow$			•				
$\{1,2\}$	$\Rightarrow$							•			
$\{2,5\}$					$\rightarrow$						
$\boxed{\{5,2\}}$					$\leftarrow$						
$\boxed{\{2,1\}}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							•		
$\{3,4\}$						$\Rightarrow$				•	
$\{4,1\}$			$\leftarrow$								
$\boxed{\{1,4\}}$			$\rightarrow$								
$\boxed{\{4,3\}}$						$\leftarrow$					
$\boxed{\{3,1\}}$		$\leftarrow$									
$\boxed{\{1,5\}}$				$\rightarrow$							



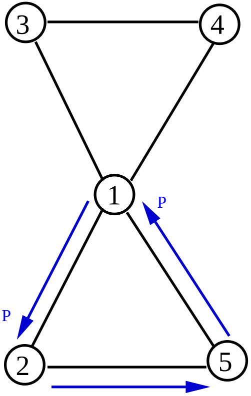
Labyrint algorithm – Example

Edges of $\mathcal{S}$	direction of edge in $\mathcal{S}$ {1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	visited vertex 1	2	3	4	5
-											$\bullet$
$\{5,1\}$				$\Leftarrow$			$\bullet$				
$\{1,2\}$	$\Rightarrow$							$\bullet$			
$\{2,5\}$					$\rightarrow$						
$\{5,2\}$					$\leftarrow$						
$\{2,1\}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							$\bullet$		
$\{3,4\}$						$\Rightarrow$				$\bullet$	
$\{4,1\}$			$\leftarrow$								
$\{1,4\}$			$\rightarrow$								
$\{4,3\}$						$\leftarrow$					
$\{3,1\}$		$\leftarrow$									
$\{1,5\}$				$\rightarrow$							



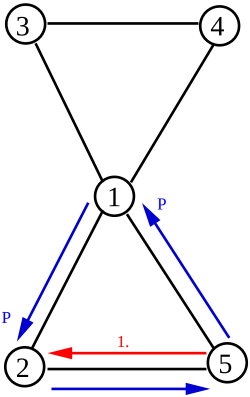
Labyrint algorithm – Example

Edges of $\mathcal{S}$	direction of edge in $\mathcal{S}$ {1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	visited vertex 1	2	3	4	5
-											$\bullet$
$\{5,1\}$				$\Leftarrow$			$\bullet$				
$\{1,2\}$	$\Rightarrow$							$\bullet$			
$\{2,5\}$					$\rightarrow$						
$\{5,2\}$					$\leftarrow$						
$\{2,1\}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							$\bullet$		
$\{3,4\}$						$\Rightarrow$				$\bullet$	
$\{4,1\}$			$\leftarrow$								
$\{1,4\}$			$\rightarrow$								
$\{4,3\}$						$\leftarrow$					
$\{3,1\}$		$\leftarrow$									
$\{1,5\}$				$\rightarrow$							



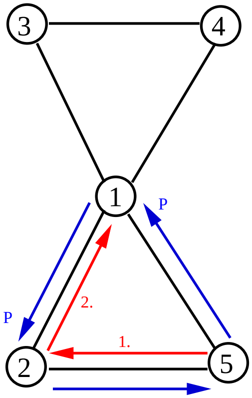
Labyrint algorithm – Example

Edges of $\mathcal{S}$	{1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	1	2	3	4	5
-											•
$\{5,1\}$				$\Leftarrow$			•				
$\{1,2\}$	$\Rightarrow$							•			
$\{2,5\}$					$\rightarrow$						
$\boxed{\{5,2\}}$					$\leftarrow$						
$\boxed{\{2,1\}}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							•		
$\{3,4\}$						$\Rightarrow$				•	
$\{4,1\}$			$\leftarrow$								
$\boxed{\{1,4\}}$			$\rightarrow$								
$\boxed{\{4,3\}}$						$\leftarrow$					
$\boxed{\{3,1\}}$		$\leftarrow$									
$\boxed{\{1,5\}}$				$\rightarrow$							



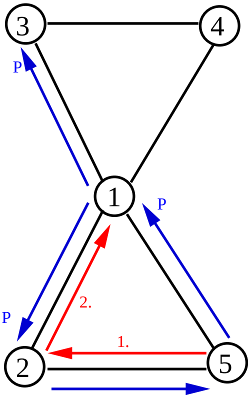
Labyrint algorithm – Example

Edges of $\mathcal{S}$	direction of edge in $\mathcal{S}$						visited vertex				
	{1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	1	2	3	4	5
-											$\bullet$
$\{5,1\}$				$\Leftarrow$			$\bullet$				
$\{1,2\}$	$\Rightarrow$							$\bullet$			
$\{2,5\}$					$\rightarrow$						
$\boxed{\{5,2\}}$					$\leftarrow$						
$\boxed{\{2,1\}}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							$\bullet$		
$\{3,4\}$						$\Rightarrow$				$\bullet$	
$\{4,1\}$			$\leftarrow$								
$\boxed{\{1,4\}}$			$\rightarrow$								
$\boxed{\{4,3\}}$						$\leftarrow$					
$\boxed{\{3,1\}}$		$\leftarrow$									
$\boxed{\{1,5\}}$				$\rightarrow$							



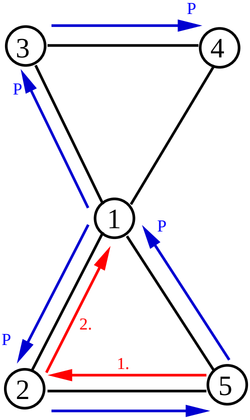
Labyrint algorithm – Example

Edges of $\mathcal{S}$	{1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	1	2	3	4	5
	direction of edge in $\mathcal{S}$						visited vertex				
-											$\bullet$
$\{5,1\}$				$\Leftarrow$			$\bullet$				
$\{1,2\}$	$\Rightarrow$							$\bullet$			
$\{2,5\}$					$\rightarrow$						
$\boxed{\mathbf{\{5,2\}}}$					$\leftarrow$						
$\boxed{\mathbf{\{2,1\}}}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							$\bullet$		
$\{3,4\}$						$\Rightarrow$				$\bullet$	
$\{4,1\}$			$\leftarrow$								
$\boxed{\mathbf{\{1,4\}}}$			$\rightarrow$								
$\boxed{\mathbf{\{4,3\}}}$						$\leftarrow$					
$\boxed{\mathbf{\{3,1\}}}$		$\leftarrow$									
$\boxed{\mathbf{\{1,5\}}}$				$\rightarrow$							



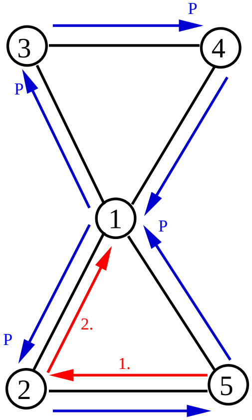
Labyrint algorithm – Example

Edges of $\mathcal{S}$	{1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	1	2	3	4	5
-											$\bullet$
$\{5,1\}$				$\Leftarrow$			$\bullet$				
$\{1,2\}$	$\Rightarrow$							$\bullet$			
$\{2,5\}$					$\rightarrow$						
$\boxed{\mathbf{\{5,2\}}}$					$\leftarrow$						
$\boxed{\mathbf{\{2,1\}}}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							$\bullet$		
$\{3,4\}$						$\Rightarrow$				$\bullet$	
$\{4,1\}$			$\leftarrow$								
$\boxed{\mathbf{\{1,4\}}}$			$\rightarrow$								
$\boxed{\mathbf{\{4,3\}}}$						$\leftarrow$					
$\boxed{\mathbf{\{3,1\}}}$		$\leftarrow$									
$\boxed{\mathbf{\{1,5\}}}$				$\rightarrow$							



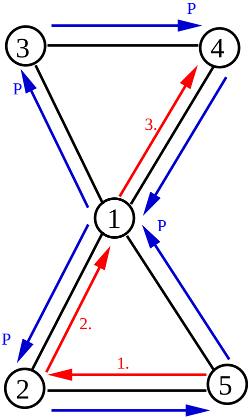
Labyrint algorithm – Example

Edges of $\mathcal{S}$	{1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	1	2	3	4	5
	direction of edge in $\mathcal{S}$						visited vertex				
-											$\bullet$
$\{5,1\}$				$\Leftarrow$			$\bullet$				
$\{1,2\}$	$\Rightarrow$							$\bullet$			
$\{2,5\}$					$\rightarrow$						
$\boxed{\mathbf{\{5,2\}}}$					$\leftarrow$						
$\boxed{\mathbf{\{2,1\}}}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							$\bullet$		
$\{3,4\}$						$\Rightarrow$				$\bullet$	
$\{4,1\}$			$\leftarrow$								
$\boxed{\mathbf{\{1,4\}}}$			$\rightarrow$								
$\boxed{\mathbf{\{4,3\}}}$						$\leftarrow$					
$\boxed{\mathbf{\{3,1\}}}$		$\leftarrow$									
$\boxed{\mathbf{\{1,5\}}}$				$\rightarrow$							



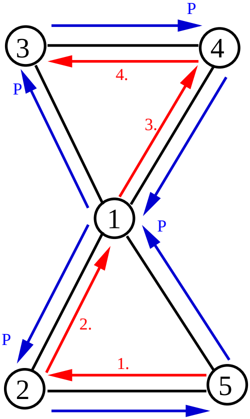
Labyrint algorithm – Example

Edges of $\mathcal{S}$	direction of edge in $\mathcal{S}$ {1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	visited vertex 1	2	3	4	5
-											$\bullet$
$\{5,1\}$				$\Leftarrow$			$\bullet$				
$\{1,2\}$	$\Rightarrow$							$\bullet$			
$\{2,5\}$					$\rightarrow$						
$\boxed{\{5,2\}}$					$\leftarrow$						
$\boxed{\{2,1\}}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							$\bullet$		
$\{3,4\}$						$\Rightarrow$				$\bullet$	
$\{4,1\}$			$\leftarrow$								
$\boxed{\{1,4\}}$			$\rightarrow$								
$\boxed{\{4,3\}}$						$\leftarrow$					
$\boxed{\{3,1\}}$		$\leftarrow$									
$\boxed{\{1,5\}}$				$\rightarrow$							



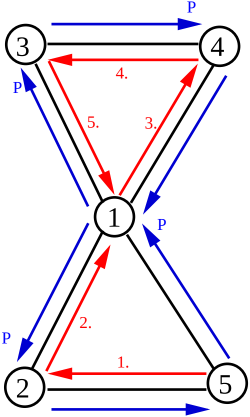
Labyrint algorithm – Example

Edges of $\mathcal{S}$	direction of edge in $\mathcal{S}$ {1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	visited vertex 1	2	3	4	5
-											$\bullet$
$\{5,1\}$				$\Leftarrow$			$\bullet$				
$\{1,2\}$	$\Rightarrow$							$\bullet$			
$\{2,5\}$					$\rightarrow$						
$\boxed{\{5,2\}}$					$\leftarrow$						
$\boxed{\{2,1\}}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							$\bullet$		
$\{3,4\}$						$\Rightarrow$				$\bullet$	
$\{4,1\}$			$\leftarrow$								
$\boxed{\{1,4\}}$			$\rightarrow$								
$\boxed{\{4,3\}}$						$\leftarrow$					
$\boxed{\{3,1\}}$		$\leftarrow$									
$\boxed{\{1,5\}}$				$\rightarrow$							



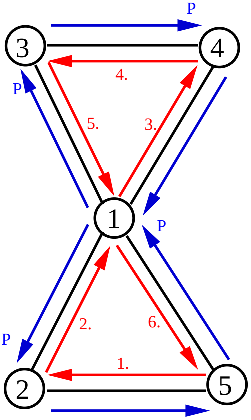
Labyrint algorithm – Example

Edges of $\mathcal{S}$	direction of edge in $\mathcal{S}$ {1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	visited vertex 1	2	3	4	5
-											$\bullet$
$\{5,1\}$				$\Leftarrow$			$\bullet$				
$\{1,2\}$	$\Rightarrow$							$\bullet$			
$\{2,5\}$					$\rightarrow$						
$\boxed{\mathbf{\{5,2\}}}$					$\leftarrow$						
$\boxed{\mathbf{\{2,1\}}}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							$\bullet$		
$\{3,4\}$						$\Rightarrow$				$\bullet$	
$\{4,1\}$			$\leftarrow$								
$\boxed{\mathbf{\{1,4\}}}$			$\rightarrow$								
$\boxed{\mathbf{\{4,3\}}}$						$\leftarrow$					
$\boxed{\mathbf{\{3,1\}}}$		$\leftarrow$									
$\boxed{\mathbf{\{1,5\}}}$				$\rightarrow$							



Labyrint algorithm – Example

Edges of $\mathcal{S}$	{1,2}	{1,3}	{1,4}	{1,5}	{2,5}	{3,4}	visited vertex 1	2	3	4	5
-											$\bullet$
$\{5,1\}$				$\Leftarrow$			$\bullet$				
$\{1,2\}$	$\Rightarrow$							$\bullet$			
$\{2,5\}$					$\rightarrow$						
$\boxed{\mathbf{\{5,2\}}}$					$\leftarrow$						
$\boxed{\mathbf{\{2,1\}}}$	$\leftarrow$										
$\{1,3\}$		$\Rightarrow$							$\bullet$		
$\{3,4\}$						$\Rightarrow$				$\bullet$	
$\{4,1\}$			$\leftarrow$								
$\boxed{\mathbf{\{1,4\}}}$			$\rightarrow$								
$\boxed{\mathbf{\{4,3\}}}$						$\leftarrow$					
$\boxed{\mathbf{\{3,1\}}}$		$\leftarrow$									
$\boxed{\mathbf{\{1,5\}}}$				$\rightarrow$							



Chinese Postman Problem

Verbal formulation of the chinese postman problem:
Postman shall go out from his post-office, to go along all streets of his district and return to the post office in such a way that his travelled distance is minimal.

Matemathematical formulation of Chinese Postman Problem.
To find the shortest Eulerian walk in a connected edge weighted graph.

Remark

- *Model of street network of a postman – a connected edge weighted graph $G = (V, H, c)$.*
- *If all vertices of graf G are of even degree then it would suffice to find an Eulerian tour in G.*
- *If there are vertices of odd degree in G then their number is $2t$ (even number).*
- *We can make an Eulerian (multi)graph $\overline{G}$ from G by adding fictive edges of the type $\{odd, odd\}$. The weight of every such edge will be set to the distance of corresponding vertices in original graph G.*
- *An Eulerian tour in graph $\overline{G}$ represents a route of a postman. Fictive edges represent shortest paths between their endpoints – postman will traverse these paths idly – without delivering mail.*
- **The less total sum of wieghts of added fictive edges the better solution.**

Definition

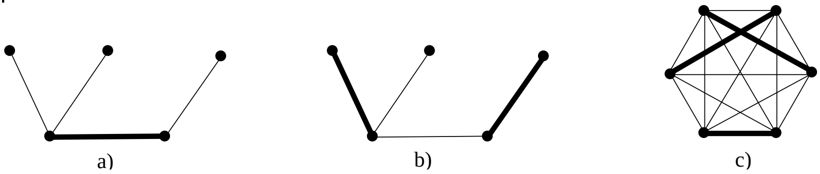
Let $G = (V, H, c)$ be an edge weighted graph.
A matching in a graph *G is a subgraph P of G with all vertices of degree 1.*
The cost of a matching *P is the total weight of it's edges.*

A matching P in a graph G **is a maximum matching in** *G if there does not exist another matching $\overline{P}$ in G such that $P \subset \overline{P}$ and $P \neq \overline{P}$.*

A matching P is **the most numerous matching in** *G if P has the largest number of edges of all matchings in G.*

A perfect matching *P* **in a graph** *G is a matching that is a spanning subgraph of G (P contains all vertices of G).*

Matchings



a) Maximal matching which is neither most numerous nor perfect.
b) Most numerous matching that is not perfect.
c) Perfect matching in K_6.

Algorithm

Edmonds's Algorithm to constructing optimal postman tour in a connected edge weighted graph $G = (V, H, c)$.

- **Step 1.** *Find all vertices of odd degree in graph G. The number of such vertices is even – equal to $2t$.*
  Create a complete graph K_{2t} containing all vertices of G having odd degree. Assign the weight of every edge of K_{2t} **equal to the distance** *of its endpoints in original graph G.*
- **Step 2.** *Find a perfect matching in K_{2t} with minimal cost.*
- **Step 3.** *Edges of that matching add to edge set of original graph G. The result is (multi)graph $\overline{G}$ with all vertices of even degree. Create an Eulerian tour $\mathcal{T}$ in (multi)graph $\overline{G}$.*
- **Step 4.** *Replace the matching edges in Eulerian tour $\mathcal{T}$ by corresponing shortest paths in G and mark them as traversed idly (without delivering mail).*

  The result is a shortest Eulerian walk in graph G.

♣

Algorithm

Edmonds's Algorithm to constructing optimal postman tour in a connected edge weighted graph $G = (V, H, c)$.

- **Step 1.** *Find all vertices of odd degree in graph G. The number of such vertices is even – equal to $2t$.*
  Create a complete graph K_{2t} containing all vertices of G having odd degree. Assign the weight of every edge of K_{2t} **equal to the distance** *of its endpoints in original graph G.*
- **Step 2.** *Find a perfect matching in K_{2t} with minimal cost.*
- **Step 3.** *Edges of that matching add to edge set of original graph G. The result is (multi)graph $\overline{G}$ with all vertices of even degree.*
  Create an Eulerian tour $\mathcal{T}$ in (multi)graph $\overline{G}$.
- **Step 4.** *Replace the matching edges in Eulerian tour $\mathcal{T}$ by corresponing shortest paths in G and mark them as traversed idly (without delivering mail).*

  The result is a shortest Eulerian walk in graph G.

♣

Algorithm

Edmonds's Algorithm to constructing optimal postman tour in a connected edge weighted graph $G = (V, H, c)$.

- **Step 1.** *Find all vertices of odd degree in graph G. The number of such vertices is even – equal to $2t$.*
  Create a complete graph K_{2t} containing all vertices of G having odd degree. Assign the weight of every edge of K_{2t} **equal to the distance** *of its endpoints in original graph G.*
- **Step 2.** *Find a perfect matching in K_{2t} with minimal cost.*
- **Step 3.** *Edges of that matching add to edge set of original graph G. The result is (multi)graph $\overline{G}$ with all vertices of even degree. Create an Eulerian tour $\mathcal{T}$ in (multi)graph $\overline{G}$.*
- **Step 4.** *Replace the matching edges in Eulerian tour $\mathcal{T}$ by corresponing shortest paths in G and mark them as traversed idly (without delivering mail).*

  The result is a shortest Eulerian walk in graph G.

♣

Algorithm

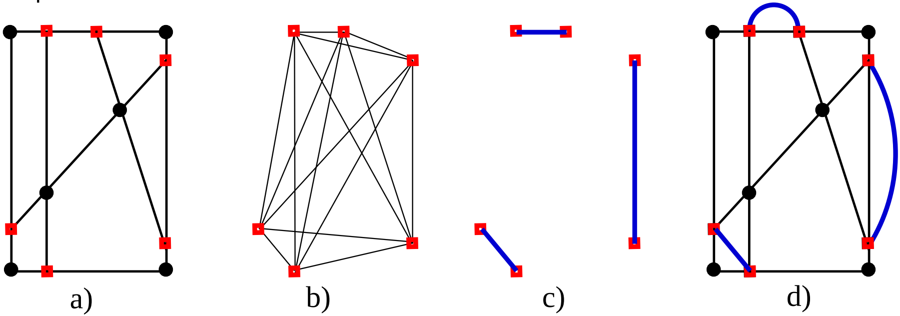
Edmonds's Algorithm to constructing optimal postman tour in a connected edge weighted graph $G = (V, H, c)$.

- **Step 1.** *Find all vertices of odd degree in graph G. The number of such vertices is even – equal to $2t$.*
  Create a complete graph K_{2t} containing all vertices of G having odd degree. Assign the weight of every edge of K_{2t} **equal to the distance** *of its endpoints in original graph G.*
- **Step 2.** *Find a perfect matching in K_{2t} with minimal cost.*
- **Step 3.** *Edges of that matching add to edge set of original graph G. The result is (multi)graph $\overline{G}$ with all vertices of even degree. Create an Eulerian tour $\mathcal{T}$ in (multi)graph $\overline{G}$.*
- **Step 4.** *Replace the matching edges in Eulerian tour $\mathcal{T}$ by corresponing shortest paths in G and mark them as traversed idly (without delivering mail).*

  The result is a shortest Eulerian walk in graph G.

♣

Edmonds's Algorithm



Operation of Edmonds's Algorithm.
a) original graph, vertices of odd degree are illustrated by little squares.
b) Complete graph K_{2t} constructed according to the Step 1. of algorithm
c) Perfect matching in v K_{2t} with minimal cost.
d) Multigraph $\overline{G}$ created according to the Step 3. of algorithm where the existence of Eulerian tour is guarranteed.

Hamiltonian walk, Hamiltonian cycle

Definition

A Hamiltonian walk *in graph G is a walk in G that contains all vertices of graph G.*

Remark

The last definition specifies also Hamiltonian path and Hamiltonian cycle since they both are special case of Hamiltonian walk.

Definition

A Hamiltonian graph *is a graph that contains a Hamiltonian cycle.*

There does not exist a simple criterion for determining that a graph G is a Hamiltonian graph.
We have several rough sufficent conditions as:

Theorem

Let $G = (V, H)$ be a graph with at least 3 vertices. Let

$$\deg(u) + \deg(v) \geq |V|$$

holds for every two no adjacent vertices u, v.
Then G is a Hamiltonian graph.

Theorem

Let $G = (V, H)$ be a graph with at least 3 vertices. Let

$$\deg(v) \geq \frac{1}{2}.|V|$$

holds for every vertex $v \in V$.
Then G is a Hamiltonian graph.

Travelling Salesman Problem - TSP

Travelling Salesman Problem – TSP

Verbal definition of TSP is:

Travelling salesman shall visit all his customers and return home in such a way that his travelled distance is minimal.

Mathematical formulaton of TSP

If the salesman is allowed to visit the same place more times mathematical formulation of TSP is as follows:

To find the shortest closed Hamiltonian walk in a connected edge weighted graph $G = (V, H, c)$**.**

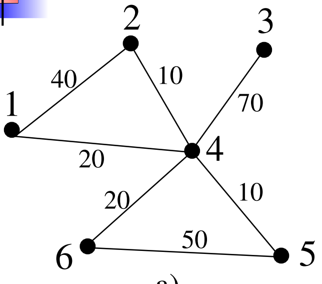
If visiting of the same place is prohibited we have the following formulation of TSP:

To find the shortest Hamiltonian cycle in a connected edge weighted graph $G = (V, H, c)$**.**

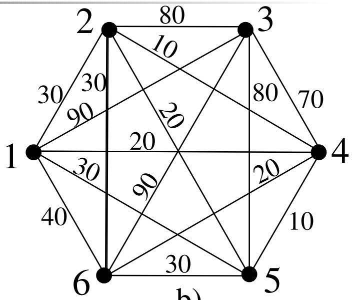
Remark

In practice there is no reason to prohibit manifold visits of customers. Moreover, in many real world situations a Hamiltonian cycle does not even exist. That is why we will focus our attention to constructing a shortest Hamiltonian walk.

Shortest Hamiltonian Walk



a)



b)

There is no Hamiltonian cycle in graph $G = (V, H, c)$ a).
Since it suffices to find the shortest Hamiltonian walk, we will search for it like a Hamiltonian cycle in complete auxiliary graph $\overline{G} = (G, E, d))$ (fig. b), whose edge weight of every edge is equal to the distance of its end point in original graph G.

Triangular inequality holds in complete graph $\overline{G}$ i. e.:
$\forall u, v, w \in V \quad u, v, w$ it holds:

$$d(u, v) \le d(u, w) + d(w, v).$$

Shortest Hamiltonian Cyklus in a Complete Graph with $\triangle$ inequ

Every permutation of vertices defines a Hamiltonian cycle in a complete graph $\overline{G}$ adn vice versa.

If we fix the first vertices we have $(n-1)!$ different Hamiltonian cycles in graph $\overline{G} = (V, H, C)$ with $n = |V|$ vertices.

There is no signicantly better way for exact determination of the shortest Hamiltonian cycle as systematic search of all $(n-1)!$ permutations.

Computation Time Provided That Search Speed is 10^9 permutations/sec.

n	$(n-1)!$	seconds	minutes	days	yars
10	3,6E+05	0,36 ms	-	-	-
15	8,7E+10	87,17	1,45	-	-
20	1,2E+17	1,2E+08	2000000	1400	3,9
25	6,2E+23	6,2E+14	1,0E+13	7,2E+09	2,0E+07
30	8,8E+30	8,8E+21	1,5E+20	1,0E+17	2,8E+14
35	3,0E+38	3,0E+29	4,9E+27	3,4E+24	9,4E+21
40	2,0E+46	2,0E+37	3,4E+35	2,4E+32	6,5E+29
45	2,7E+54	2,7E+45	4,4E+43	3,1E+40	8,4E+37
50	6,1E+62	6,1E+53	1,0E+52	7,0E+48	1,9E+46

Greedy Algorithm

Corolarry: We have to use algorithms that offer sufficiently good but not surely exact results – heuristics, suboptimal algorithms.

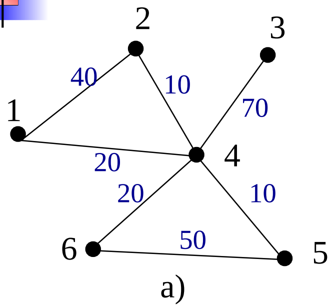
Algorithm

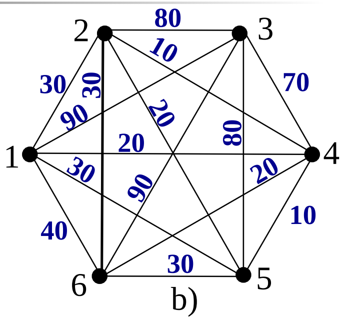
Greedy Algorithm. Heuristics to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ with triangular inequality and with at least 3 vertices.

- **Step 1.** *Start at arbitrary vertex and insert the cheapest edge incident with this vertex together with its second endpoint into chosen sequence – (future Hamiltonian cycle).*
- **Step 2.** *If the number of chosen edges is equal to $n-1$ then close cycle. STOP*
- **Step 3.** *Otherwise choose the cheapest unchosen edgee incident with the last vertex of till now chosen sequence, which is not incident with any other vertex of chosen sequence.*

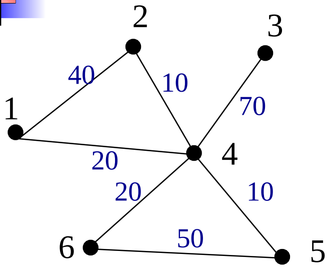
  Insert this edge together with its second endpoint into chosen sequence.

  GOTO Step 2

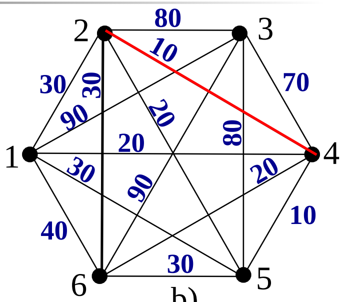

2
3
40
10
1
70
20
4
20
10
6
50
5
a)


2
80
3
10
30
30
70
90
20
80
1
20
4
30
20
90
40
10
30
6
5
b)

$\mathcal{C} = (2)$



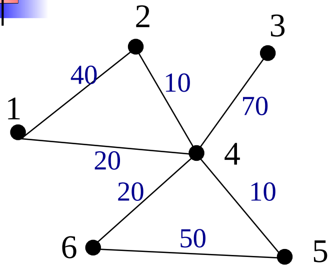
a)



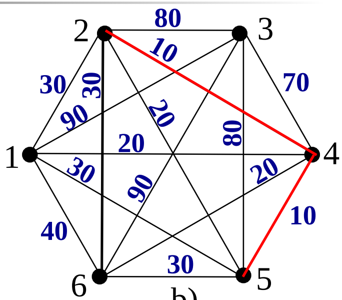
b)

$\mathcal{C} = (2, \{2, 4\}, 4)$

Greedy algoritmus pre TSP



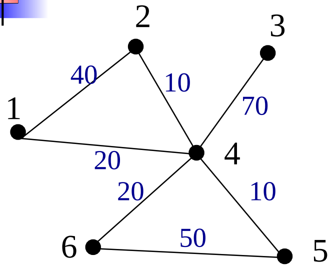
a)



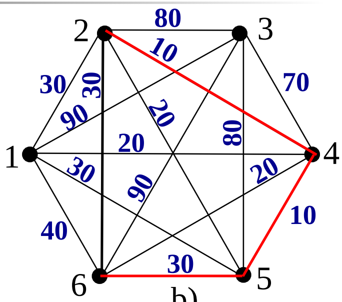
b)

$\mathcal{C} = (2, \{2,4\}, 4, \{4,5\}, 5)$

Greedy algoritmus pre TSP

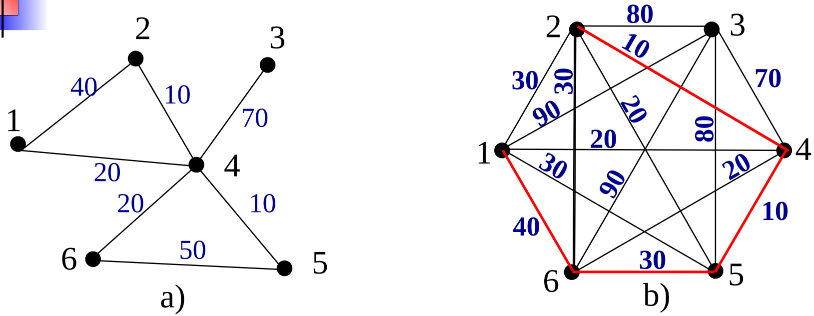


a)

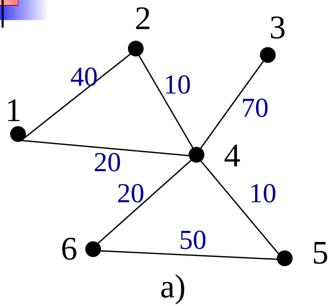


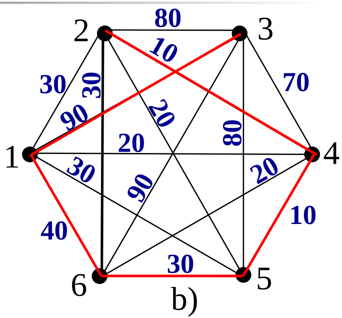
b)

$\mathcal{C} = (2, \{2, 4\}, 4, \{4, 5\}, 5, \{5, 6\}, 6)$



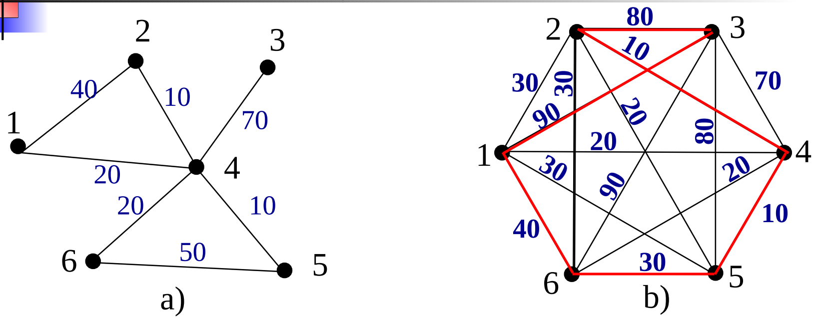
$\mathcal{C} = (2, \{2,4\}, 4, \{4,5\}, 5, \{5,6\}, 6, \{6,1\}, 1)$



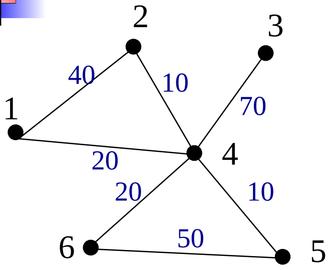


$\mathcal{C} = (2, \{2, 4\}, 4, \{4, 5\}, 5, \{5, 6\}, 6, \{6, 1\}, 1, \{1, 3\}, 3)$

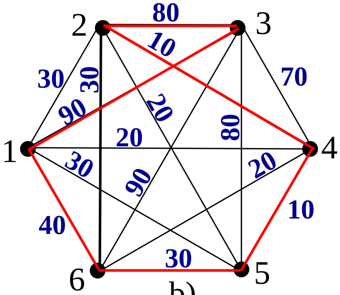
Greedy algoritmus pre TSP



$\mathcal{C} = (2, \{2, 4\}, 4, \{4, 5\}, 5, \{5, 6\}, 6, \{6, 1\}, 1, \{1, 3\}, 3, \{3, 2\}, 2)$



a)



b)

$\mathcal{C} = (2, \{2,4\}, 4, \{4,5\}, 5, \{5,6\}, 6, \{6,1\}, 1, \{1,3\}, 3, \{3,2\}, 2)$

Now we replace every edge of cycle $\mathcal{C}$ by corresponding shortest path in original graph G.

$(2, \{2,4\}, 4) \rightarrow (2, \{2,4\}, 4)$
$(4, \{4,5\}, 5) \rightarrow (4, \{4,5\}, 5)$
$(5, \{5,6\}, 6) \rightarrow (5, \{5,4\}, 4, \{4,6\}, 6)$
$(6, \{6,1\}, 1) \rightarrow (6, \{6,4\}, 4, \{4,1\}, 1)$
$(1, \{1,3\}, 3) \rightarrow (1, \{1,4\}, 4, \{4,3\}, 3)$
$(3, \{3,2\}, 2) \rightarrow (3, \{3,4\}, 4, \{4,2\}, 2)$

Algorithm

Double the Spanning Tree Algorithm. (Kim – 1975). Heuristic to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ with triangular inequality.

- **Step 1.** *Find a minimum spanning tree K in graph G.*
- **Step 2.** *Create a closed walk $\mathcal{S}$ in spanning tree K containing every edge of K exactly two times. (Use Tarry's algorithm).*
- **Step 3.** *Create a Hamiltonian cycle from walk $\mathcal{S}$ as follows: Follow the sequence of vertices of $\mathcal{S}$ and if you find a visited vertex skip this vertex to the next unvisited vertex (or to the last vertex of $\mathcal{S}$ if any) and replace the skipped segment by direct edge.*

♣

Algorithm

Double the Spanning Tree Algorithm. (Kim – 1975). Heuristic to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ with triangular inequality.

- **Step 1.** *Find a minimum spanning tree K in graph G.*
- **Step 2.** *Create a closed walk $\mathcal{S}$ in spanning tree K containing every edge of K exactly two times. (Use Tarry's algorithm).*
- **Step 3.** *Create a Hamiltonian cycle from walk $\mathcal{S}$ as follows: Follow the sequence of vertices of $\mathcal{S}$ and if you find a visited vertex skip this vertex to the next unvisited vertex (or to the last vertex of $\mathcal{S}$ if any) and replace the skipped segment by direct edge.*

♣

Algorithm

Double the Spanning Tree Algorithm. (Kim – 1975). Heuristic to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ **with triangular inequality.**

- **Step 1.** *Find a minimum spanning tree K in graph G.*
- **Step 2.** *Create a closed walk $\mathcal{S}$ in spanning tree K containing every edge of K exactly two times. (Use Tarry's algorithm).*
- **Step 3.** *Create a Hamiltonian cycle from walk $\mathcal{S}$ as follows: Follow the sequence of vertices of $\mathcal{S}$ and if you find a visited vertex skip this vertex to the next unvisited vertex (or to the last vertex of $\mathcal{S}$ if any) and replace the skipped segment by direct edge.*

♣

Double the Spanning Tree Algorithm

Theorem

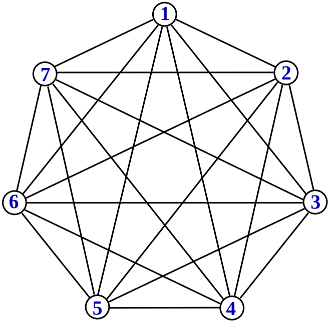
Let $G = (V, H, c)$ be a complete graph with triangular inequality. Let $c(DST)$ be the length of Hamiltonian cycle obtained by Double the Spanning Tree Algorithm, leth $c(OPT)$ be the exact length of shortest Hamiltonian cycle in graph G.
Then

$$\frac{c(DST)}{c(OPT)} < 2.$$

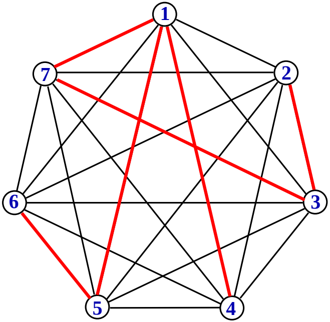
Moreover, the last estimation cannot be improved – for every $\varepsilon > 0$ there exists a graph G_ε for which it holds

$$\frac{c(DST)}{c(OPT)} > 2 - \varepsilon.$$

Double the Spanning Tree Algorithm – Example

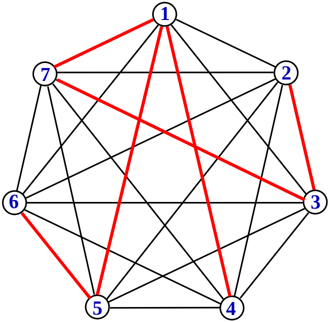


Double the Spanning Tree Algorithm – Example



$$\mathcal{T} = (1, \{1,7\}, 7, \{7,3\}, 3, \{3,2\}, 2, \{2,3\}, 3, \{3,7\}, 7, \{7,1\}, 1,$$
$$\{4,1\}, 4, \{4,1,\}, 1, \{1,5\}, 5, \{5,6\}, 6, \{6,5\}, 5, \{5,1\}, 1)$$

Double the Spanning Tree Algorithm – Example

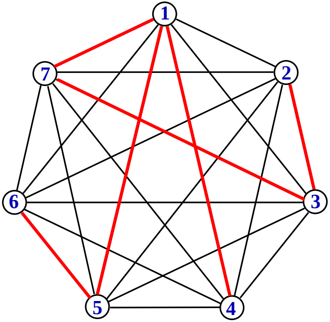


$$\mathcal{T} = (1, \{1,7\}, 7, \{7,3\}, 3, \{3,2\}, 2, \{2,3\}, 3, \{3,7\}, 7, \{7,1\}, 1,$$
$$\{4,1\}, 4, \{4,1,\}, 1, \{1,5\}, 5, \{5,6\}, 6, \{6,5\}, 5, \{5,1\}, 1)$$

Shortly

$$\mathcal{T} = (1, 7, 3, 2, \underbrace{\mathbf{3, 7, 1}}_{\text{to replace by edge } \{2,4\}}, 4, 1, 5, 6, 5, 1)$$

Double the Spanning Tree Algorithm – Example



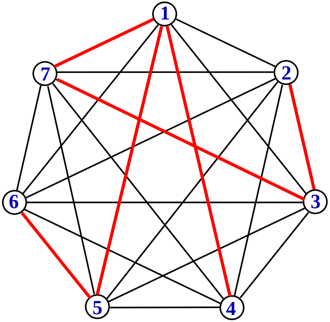
$$\mathcal{T} = (1, \{1,7\}, 7, \{7,3\}, 3, \{3,2\}, 2, \{2,3\}, 3, \{3,7\}, 7, \{7,1\}, 1,$$
$$\{4,1\}, 4, \{4,1,\}, 1, \{1,5\}, 5, \{5,6\}, 6, \{6,5\}, 5, \{5,1\}, 1)$$

Shortly

$$\mathcal{T} = (1, 7, 3, 2, \underbrace{\mathbf{3}, \mathbf{7}, \mathbf{1}}_{\text{to replace by edge } \{2,4\}}, 4, 1, 5, 6, 5, 1)$$

$$\mathcal{T} = (1, 7, 3, 2, 4, \mathbf{1}, 5, 6, 5, 1)$$

Double the Spanning Tree Algorithm – Example



$$\mathcal{T} = (1, \{1,7\}, 7, \{7,3\}, 3, \{3,2\}, 2, \{2,3\}, 3, \{3,7\}, 7, \{7,1\}, 1,$$
$$\{4,1\}, 4, \{4,1,\}, 1, \{1,5\}, 5, \{5,6\}, 6, \{6,5\}, 5, \{5,1\}, 1)$$

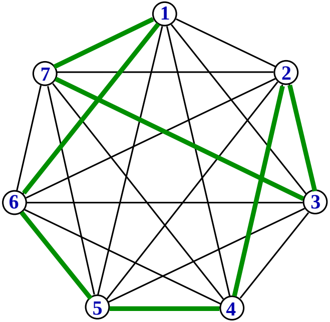
Shortly

$$\mathcal{T} = (1, 7, 3, 2, \underbrace{\mathbf{3, 7, 1}}_{\text{to replace by edge } \{2,4\}}, 4, 1, 5, 6, 5, 1)$$

$$\mathcal{T} = (1, 7, 3, 2, 4, \mathbf{1}, 5, 6, 5, 1)$$

$$\mathcal{T} = (1, 7, 3, 2, 4, 5, 6, \mathbf{5}, 1)$$

Double the Spanning Tree Algorithm – Example



$$\mathcal{T} = (1, \{1,7\}, 7, \{7,3\}, 3, \{3,2\}, 2, \{2,3\}, 3, \{3,7\}, 7, \{7,1\}, 1,$$

$$\{4,1\}, 4, \{4,1,\}, 1, \{1,5\}, 5, \{5,6\}, 6, \{6,5\}, 5, \{5,1\}, 1)$$

Shortly

$$\mathcal{T} = (1, 7, 3, 2, \underbrace{\mathbf{3, 7, 1}}_{\text{to replace by edge } \{2,4\}}, 4, 1, 5, 6, 5, 1)$$

$$\mathcal{T} = (1, 7, 3, 2, 4, \mathbf{1}, 5, 6, 5, 1)$$

$$\mathcal{T} = (1, 7, 3, 2, 4, 5, 6, \mathbf{5}, 1)$$

$$\mathcal{T} = (1, 7, 3, 2, 4, 5, 6, 1)$$

Algorithm

Spanning Tree and Matching Algorithm. (Christofides – 1976.) *Heuristic to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ with triangular inequality.*

- **Step 1.** *Find a minimum spanning tree K in graph G.*
- **Step 2.** *Find all vertices of odd degree in spanning tree K. The number of such vertices is even – equal to $2t$.*
- **Step 3.** *Create a complete graph K_{2t} with vertex set equal to the set of all odd degree vertices of K. Set edge weight of every edge of K_{2t} equal to the weight of this edge in graph G.*
- **Step 4.** *Find a minimum cost perfect matching in K_{2t}.*
- **Step 5.** *Add matching edges obtained in Step 4. into edge set of spanning tree K. The result is (multi)graphf $\overline{K}$ having all vertices of even degree.*
- **Step 6.** *Create a Eulerian tour $\mathcal{T}$ in (multi)graph $\overline{K}$.*
- **Step 7.** *Create a Hamiltonian cycle from Eulerian tour $\mathcal{T}$ as follows: Follow the sequence of vertices of $\mathcal{T}$ and if you find a visited vertex skip this vertex to the next unvisited vertex (or to the last vertex of $\mathcal{T}$ if any) and replace the skipped segment by direct edge.*

Algorithm

Spanning Tree and Matching Algorithm. (Christofides – 1976.) *Heuristic to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ with triangular inequality.*

- **Step 1.** *Find a minimum spanning tree K in graph G.*
- **Step 2.** *Find all vertices of odd degree in spanning tree K. The number of such vertices is even – equal to $2t$.*
- **Step 3.** *Create a complete graph K_{2t} with vertex set equal to the set of all odd degree vertices of K. Set edge weight of every edge of K_{2t} equal to the weight of this edge in graph G.*
- **Step 4.** *Find a minimum cost perfect matching in K_{2t}.*
- **Step 5.** *Add matching edges obtained in Step 4. into edge set of spanning tree K. The result is (multi)graphf $\overline{K}$ having all vertices of even degree.*
- **Step 6.** *Create a Eulerian tour $\mathcal{T}$ in (multi)graph $\overline{K}$.*
- **Step 7.** *Create a Hamiltonian cycle from Eulerian tour $\mathcal{T}$ as follows: Follow the sequence of vertices of $\mathcal{T}$ and if you find a visited vertex skip this vertex to the next unvisited vertex (or to the last vertex of $\mathcal{T}$ if any) and replace the skipped segment by direct edge.*

Algorithm

Spanning Tree and Matching Algorithm. (Christofides – 1976.) *Heuristic to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ with triangular inequality.*

- **Step 1.** *Find a minimum spanning tree K in graph G.*
- **Step 2.** *Find all vertices of odd degree in spanning tree K. The number of such vertices is even – equal to $2t$.*
- **Step 3.** *Create a complete graph K_{2t} with vertex set equal to the set of all odd degree vertices of K. Set edge weight of every edge of K_{2t} equal to the weight of this edge in graph G.*
- **Step 4.** *Find a minimum cost perfect matching in K_{2t}.*
- **Step 5.** *Add matching edges obtained in Step 4. into edge set of spanning tree K. The result is (multi)graphf $\overline{K}$ having all vertices of even degree.*
- **Step 6.** *Create a Eulerian tour $\mathcal{T}$ in (multi)graph $\overline{K}$.*
- **Step 7.** *Create a Hamiltonian cycle from Eulerian tour $\mathcal{T}$ as follows: Follow the sequence of vertices of $\mathcal{T}$ and if you find a visited vertex skip this vertex to the next unvisited vertex (or to the last vertex of $\mathcal{T}$ if any) and replace the skipped segment by direct edge.*

Spanning Tree and Matching Algorithm

Algorithm

Spanning Tree and Matching Algorithm. (Christofides – 1976.) *Heuristic to find a suboptimal solution of TSP in a complete graph* $G = (V, H, c)$ *with triangular inequality.*

- **Step 1.** *Find a minimum spanning tree* K *in graph* G*.*
- **Step 2.** *Find all vertices of odd degree in spanning tree* K*. The number of such vertices is even – equal to* $2t$*.*
- **Step 3.** *Create a complete graph* K_{2t} *with vertex set equal to the set of all odd degree vertices of* K*. Set edge weight of every edge of* K_{2t} *equal to the weight of this edge in graph* G*.*
- **Step 4.** *Find a minimum cost perfect matching in* K_{2t}*.*
- **Step 5.** *Add matching edges obtained in Step 4. into edge set of spanning tree* K*. The result is (multi)graphf* $\overline{K}$ *having all vertices of even degree.*
- **Step 6.** *Create a Eulerian tour* $\mathcal{T}$ *in (multi)graph* $\overline{K}$*.*
- **Step 7.** *Create a Hamiltonian cycle from Eulerian tour* $\mathcal{T}$ *as follows: Follow the sequence of vertices of* $\mathcal{T}$ *and if you find a visited vertex skip this vertex to the next unvisited vertex (or to the last vertex of* $\mathcal{T}$ *if any) and replace the skipped segment by direct edge.*

Algorithm

Spanning Tree and Matching Algorithm. (Christofides – 1976.) *Heuristic to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ with triangular inequality.*

- **Step 1.** *Find a minimum spanning tree K in graph G.*
- **Step 2.** *Find all vertices of odd degree in spanning tree K. The number of such vertices is even – equal to $2t$.*
- **Step 3.** *Create a complete graph K_{2t} with vertex set equal to the set of all odd degree vertices of K. Set edge weight of every edge of K_{2t} equal to the weight of this edge in graph G.*
- **Step 4.** *Find a minimum cost perfect matching in K_{2t}.*
- **Step 5.** *Add matching edges obtained in Step 4. into edge set of spanning tree K. The result is (multi)graphf $\overline{K}$ having all vertices of even degree.*
- **Step 6.** *Create a Eulerian tour $\mathcal{T}$ in (multi)graph $\overline{K}$.*
- **Step 7.** *Create a Hamiltonian cycle from Eulerian tour $\mathcal{T}$ as follows: Follow the sequence of vertices of $\mathcal{T}$ and if you find a visited vertex skip this vertex to the next unvisited vertex (or to the last vertex of $\mathcal{T}$ if any) and replace the skipped segment by direct edge.*

Algorithm

Spanning Tree and Matching Algorithm. (Christofides – 1976.) *Heuristic to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ with triangular inequality.*

- **Step 1.** *Find a minimum spanning tree K in graph G.*
- **Step 2.** *Find all vertices of odd degree in spanning tree K. The number of such vertices is even – equal to $2t$.*
- **Step 3.** *Create a complete graph K_{2t} with vertex set equal to the set of all odd degree vertices of K. Set edge weight of every edge of K_{2t} equal to the weight of this edge in graph G.*
- **Step 4.** *Find a minimum cost perfect matching in K_{2t}.*
- **Step 5.** *Add matching edges obtained in Step 4. into edge set of spanning tree K. The result is (multi)graphf $\overline{K}$ having all vertices of even degree.*
- **Step 6.** *Create a Eulerian tour $\mathcal{T}$ in (multi)graph $\overline{K}$.*
- **Step 7.** *Create a Hamiltonian cycle from Eulerian tour $\mathcal{T}$ as follows: Follow the sequence of vertices of $\mathcal{T}$ and if you find a visited vertex skip this vertex to the next unvisited vertex (or to the last vertex of $\mathcal{T}$ if any) and replace the skipped segment by direct edge.*

Algorithm

Spanning Tree and Matching Algorithm. (Christofides – 1976.) *Heuristic to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ with triangular inequality.*

- **Step 1.** *Find a minimum spanning tree K in graph G.*
- **Step 2.** *Find all vertices of odd degree in spanning tree K. The number of such vertices is even – equal to $2t$.*
- **Step 3.** *Create a complete graph K_{2t} with vertex set equal to the set of all odd degree vertices of K. Set edge weight of every edge of K_{2t} equal to the weight of this edge in graph G.*
- **Step 4.** *Find a minimum cost perfect matching in K_{2t}.*
- **Step 5.** *Add matching edges obtained in Step 4. into edge set of spanning tree K. The result is (multi)graphf $\overline{K}$ having all vertices of even degree.*
- **Step 6.** *Create a Eulerian tour $\mathcal{T}$ in (multi)graph $\overline{K}$.*
- **Step 7.** *Create a Hamiltonian cycle from Eulerian tour $\mathcal{T}$ as follows: Follow the sequence of vertices of $\mathcal{T}$ and if you find a visited vertex skip this vertex to the next unvisited vertex (or to the last vertex of $\mathcal{T}$ if any) and replace the skipped segment by direct edge.*

Theorem

Let $G = (V, H, c)$ be a complete graph with triangular inequality. Let $c(STMA)$ be the length of Hamiltonian cycle obtained by Spanning Tree and Matching Algorithm, let $c(OPT)$ be the exact length of shortest Hamiltonian cycle in graph G.
Then

$$\frac{c(STMA)}{c(OPT)} < \frac{3}{2}.$$

Moreover, the last estimation cannot be improved – for every $\varepsilon > 0$ there exists a graph G_ε for which it holds

$$\frac{c(STMA)}{c(OPT)} > \frac{3}{2} - \varepsilon.$$

Remark

We do not know a polynomial algorithm ALG which would guarantee better ratio $c(ALG)/c(OPT)$ then $3/2$.

Algorithm

Inserting Heuristics to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ with triangular inequality.

- **Step 1.** *Choose an edge $h = \{u, v\}$ with least weight.*
  Find vertex $w \in V$, for which is the sum $c\{u, w\} + c\{w, v\}$ minimal.
  Create cycle $C = (u, \{u, w\}, w, \{w, v\}, v, \{v, u\}, u)$.
- **Step 2.** *If cycle C contains all vertices of graph G then STOP.*
  Otherwise continue in Step 3.
- **Step 3.** *Calculate*

$$z(h) = \min\{c\{u, w\} + c\{w, v\} - c\{u, v\} \mid w \in V - C\}. \quad (1)$$

  for every edge $h = \{u, v\}$ of cycle C.

  Choose the edge $h = \{u, v\}$ with minimal value $z(h)$ and vertex w, for which occured minimum in (1).

  Create acycle C' by replacing the edgge $\{u, v\}$ by two edges $\{u, w\}$, $\{w, v\}$.

  Set $C := C'$.

  GOTO Step 2.

Algorithm

Inserting Heuristics to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ with triangular inequality.

- **Step 1.** *Choose an edge $h = \{u, v\}$ with least weight.*
  Find vertex $w \in V$, for which is the sum $c\{u, w\} + c\{w, v\}$ minimal.
  Create cycle $C = (u, \{u, w\}, w, \{w, v\}, v, \{v, u\}, u)$.
- **Step 2.** *If cycle C contains all vertices of graph G then STOP.*
  Otherwise continue in Step 3.
- **Step 3.** *Calculate*

$$z(h) = \min\{c\{u, w\} + c\{w, v\} - c\{u, v\} \mid w \in V - C\}. \tag{1}$$

  for every edge $h = \{u, v\}$ of cycle C.

  Choose the edge $h = \{u, v\}$ with minimal value $z(h)$ and vertex w, for which occured minimum in (1).

  Create acycle C' by replacing the edgge $\{u, v\}$ by two edges $\{u, w\}$, $\{w, v\}$.

  Set $C := C'$.

  GOTO Step 2.

Algorithm

Inserting Heuristics to find a suboptimal solution of TSP in a complete graph $G = (V, H, c)$ with triangular inequality.

- **Step 1.** *Choose an edge $h = \{u, v\}$ with least weight.*
  Find vertex $w \in V$, for which is the sum $c\{u, w\} + c\{w, v\}$ minimal.
  Create cycle $C = (u, \{u, w\}, w, \{w, v\}, v, \{v, u\}, u)$.
- **Step 2.** *If cycle C contains all vertices of graph G then STOP.*
  Otherwise continue in Step 3.
- **Step 3.** *Calculate*

$$z(h) = \min\{c\{u, w\} + c\{w, v\} - c\{u, v\} \mid w \in V - C\}. \qquad (1)$$

for every edge $h = \{u, v\}$ of cycle C.

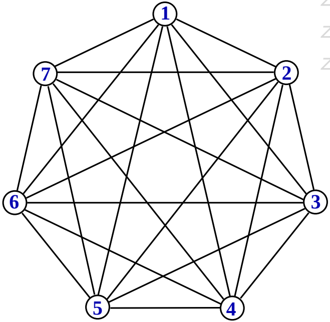
Choose the edge $h = \{u, v\}$ with minimal value $z(h)$ and vertex w, for which occured minimum in (1).

Create acycle C' by replacing the edgge $\{u, v\}$ by two edges $\{u, w\}$, $\{w, v\}$.

Set $C := C'$.

GOTO Step 2.

Inserting Heuristics for TSP



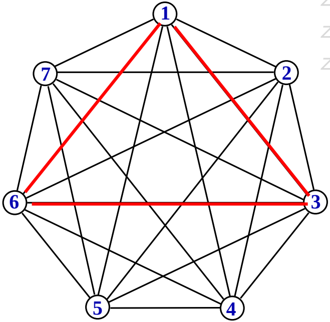
$$
\begin{aligned}
z(h) &= \quad c\{6,5\}+c\{5,3\}-c\{6,3\}\\
z(h) &= \min\{z(h),\ c\{6,4\}+c\{4,3\}-c\{6,3\}\}\\
z(h) &= \min\{z(h),\ c\{6,2\}+c\{2,3\}-c\{6,3\}\}\\
z(h) &= \min\{z(h),\ c\{6,7\}+c\{7,3\}-c\{6,3\}\}
\end{aligned}
$$

$$
z(h)=\min\left\{\begin{array}{l} c\{6,5\}+c\{5,3\}-c\{6,3\},\\ c\{6,4\}+c\{4,3\}-c\{6,3\},\\ c\{6,2\}+c\{2,3\}-c\{6,3\},\\ c\{6,7\}+c\{7,3\}-c\{6,3\} \end{array}\right\}
$$

Remark

Algorithm Inserting Heuristics for TSP creates cycles step by step so that it inserts that vertex into contemporary cycle which extends the length of cycle as little as possible.

Inserting Heuristics for TSP



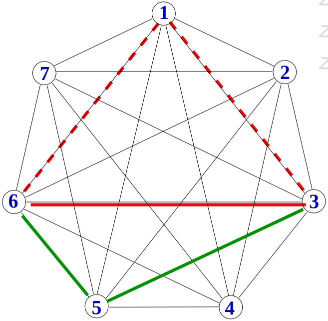
$$z(h) = \qquad c\{6,5\} + c\{5,3\} - c\{6,3\}$$
$$z(h) = \min\{z(h),\ c\{6,4\} + c\{4,3\} - c\{6,3\}\}$$
$$z(h) = \min\{z(h),\ c\{6,2\} + c\{2,3\} - c\{6,3\}\}$$
$$z(h) = \min\{z(h),\ c\{6,7\} + c\{7,3\} - c\{6,3\}\}$$

$$z(h) = \min\left\{\begin{array}{l} c\{6,5\} + c\{5,3\} - c\{6,3\}, \\ c\{6,4\} + c\{4,3\} - c\{6,3\}, \\ c\{6,2\} + c\{2,3\} - c\{6,3\}, \\ c\{6,7\} + c\{7,3\} - c\{6,3\} \end{array}\right\}$$

Remark

Algorithm Inserting Heuristics for TSP creates cycles step by step so that it inserts that vertex into contemporary cycle which extends the length of cycle as little as possible.

Inserting Heuristics for TSP



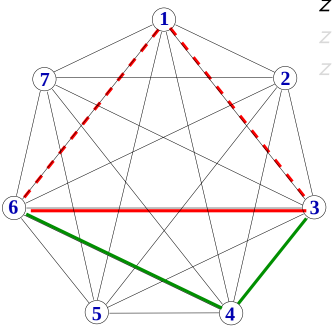
$$z(h) = \quad c\{6,5\} + c\{5,3\} - c\{6,3\}$$
$$z(h) = \quad \min\{z(h),\ c\{6,4\} + c\{4,3\} - c\{6,3\}\}$$
$$z(h) = \quad \min\{z(h),\ c\{6,2\} + c\{2,3\} - c\{6,3\}\}$$
$$z(h) = \quad \min\{z(h),\ c\{6,7\} + c\{7,3\} - c\{6,3\}\}$$

$$z(h) = \min \left\{ \begin{array}{l} c\{6,5\} + c\{5,3\} - c\{6,3\}, \\ c\{6,4\} + c\{4,3\} - c\{6,3\}, \\ c\{6,2\} + c\{2,3\} - c\{6,3\}, \\ c\{6,7\} + c\{7,3\} - c\{6,3\} \end{array} \right\}$$

Remark

Algorithm Inserting Heuristics for TSP creates cycles step by step so that it inserts that vertex into contemporary cycle which extends the length of cycle as little as possible.

Inserting Heuristics for TSP



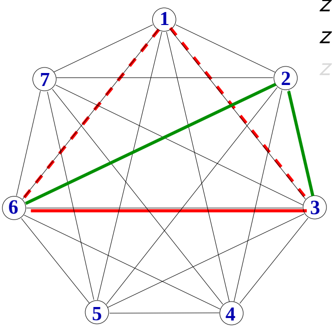
$$z(h) = c\{6,5\} + c\{5,3\} - c\{6,3\}$$
$$z(h) = \min\{z(h),\ c\{6,4\} + c\{4,3\} - c\{6,3\}\}$$
$$z(h) = \min\{z(h),\ c\{6,2\} + c\{2,3\} - c\{6,3\}\}$$
$$z(h) = \min\{z(h),\ c\{6,7\} + c\{7,3\} - c\{6,3\}\}$$

$$z(h) = \min \begin{Bmatrix} c\{6,5\} + c\{5,3\} - c\{6,3\}, \\ c\{6,4\} + c\{4,3\} - c\{6,3\}, \\ c\{6,2\} + c\{2,3\} - c\{6,3\}, \\ c\{6,7\} + c\{7,3\} - c\{6,3\} \end{Bmatrix}$$

Remark

Algorithm Inserting Heuristics for TSP creates cycles step by step so that it inserts that vertex into contemporary cycle which extends the length of cycle as little as possible.

Inserting Heuristics for TSP



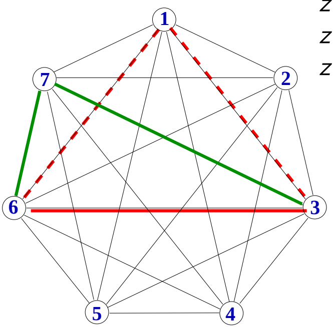
$$z(h) = c\{6,5\} + c\{5,3\} - c\{6,3\}$$
$$z(h) = \min\{z(h),\ c\{6,4\} + c\{4,3\} - c\{6,3\}\}$$
$$z(h) = \min\{z(h),\ c\{6,2\} + c\{2,3\} - c\{6,3\}\}$$
$$z(h) = \min\{z(h),\ c\{6,7\} + c\{7,3\} - c\{6,3\}\}$$

$$z(h) = \min \left\{ \begin{array}{l} c\{6,5\} + c\{5,3\} - c\{6,3\}, \\ c\{6,4\} + c\{4,3\} - c\{6,3\}, \\ c\{6,2\} + c\{2,3\} - c\{6,3\}, \\ c\{6,7\} + c\{7,3\} - c\{6,3\} \end{array} \right\}$$

Remark

Algorithm Inserting Heuristics for TSP creates cycles step by step so that it inserts that vertex into contemporary cycle which extends the length of cycle as little as possible.

Inserting Heuristics for TSP



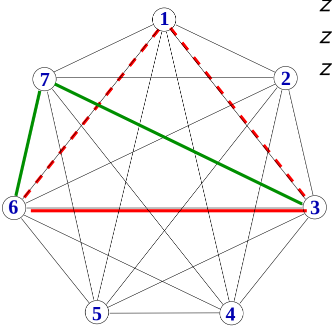
$$z(h) = \qquad c\{6,5\} + c\{5,3\} - c\{6,3\}$$
$$z(h) = \min\{z(h),\ c\{6,4\} + c\{4,3\} - c\{6,3\}\}$$
$$z(h) = \min\{z(h),\ c\{6,2\} + c\{2,3\} - c\{6,3\}\}$$
$$z(h) = \min\{z(h),\ c\{6,7\} + c\{7,3\} - c\{6,3\}\}$$

$$z(h) = \min \begin{Bmatrix} c\{6,5\} + c\{5,3\} - c\{6,3\}, \\ c\{6,4\} + c\{4,3\} - c\{6,3\}, \\ c\{6,2\} + c\{2,3\} - c\{6,3\}, \\ c\{6,7\} + c\{7,3\} - c\{6,3\} \end{Bmatrix}$$

Remark

Algorithm Inserting Heuristics for TSP creates cycles step by step so that it inserts that vertex into contemporary cycle which extends the length of cycle as little as possible.

Inserting Heuristics for TSP



$$
\begin{aligned}
z(h) &= & &c\{6,5\} + c\{5,3\} - c\{6,3\} \\
z(h) &= &\min\big\{z(h),\ &c\{6,4\} + c\{4,3\} - c\{6,3\}\big\} \\
z(h) &= &\min\big\{z(h),\ &c\{6,2\} + c\{2,3\} - c\{6,3\}\big\} \\
z(h) &= &\min\big\{z(h),\ &c\{6,7\} + c\{7,3\} - c\{6,3\}\big\}
\end{aligned}
$$

$$
z(h) = \min \left\{ \begin{array}{l} c\{6,5\} + c\{5,3\} - c\{6,3\}, \\ c\{6,4\} + c\{4,3\} - c\{6,3\}, \\ c\{6,2\} + c\{2,3\} - c\{6,3\}, \\ c\{6,7\} + c\{7,3\} - c\{6,3\} \end{array} \right\}
$$

Remark

Algorithm Inserting Heuristics for TSP creates cycles step by step so that it inserts that vertex into contemporary cycle which extends the length of cycle as little as possible.

Algorithm

Neighbourhood Search Algorithm.

We can define a neighourhood to every solution of TSP.
The neighbourhood $\mathcal{O}(C)$ of a Hamiltonian cycle C can be defined as the set of Hamiltonian cycles obtained from cycle C by means of several simple operations.
Denote by $c(C)$ the length of Hamiltonian cycle C.

- **Step 1.** *Take arbitrary Hamiltonian cycle C as starting solution. (Starting cycle can be obtained as a result of a heuristics or can be generated by a pseudorandom generator).*
- **Step 2.** *Search for $C' \in \mathcal{O}(C)$ with $c(C') < c(C)$.*
  If for all $C' \in \mathcal{O}(C)$ $c(C') \geq c(C)$, STOP, C is suboptimal Hamiltonian cycle.
  Otherwise continye by Step 3.
- **Step 3.** *Take $C' \in \mathcal{O}(C)$ such that $c(C') < c(C)$ and set $C := C'$. Goto Step 2.*

Algorithm

Neighbourhood Search Algorithm.
We can define a neighourhood to every solution of TSP.
The neighbourhood $\mathcal{O}(C)$ of a Hamiltonian cycle C can be defined as the set of Hamiltonian cycles obtained from cycle C by means of several simple operations.
Denote by $c(C)$ the length of Hamiltonian cycle C.

- **Step 1.** *Take arbitrary Hamiltonian cycle C as starting solution. (Starting cycle can be obtained as a result of a heuristics or can be generated by a pseudorandom generator).*
- **Step 2.** *Search for $C' \in \mathcal{O}(C)$ with $c(C') < c(C)$.*

  If for all $C' \in \mathcal{O}(C)$ $c(C') \geq c(C)$, STOP, C is suboptimal Hamiltonian cycle.

  Otherwise continye by Step 3.
- **Step 3.** *Take $C' \in \mathcal{O}(C)$ such that $c(C') < c(C)$ and set $C := C'$.*

  Goto Step 2.

Algorithm

Neighbourhood Search Algorithm.

We can define a neighourhood to every solution of TSP.

The neighbourhood $\mathcal{O}(C)$ of a Hamiltonian cycle C can be defined as the set of Hamiltonian cycles obtained from cycle C by means of several simple operations.

Denote by $c(C)$ the length of Hamiltonian cycle C.

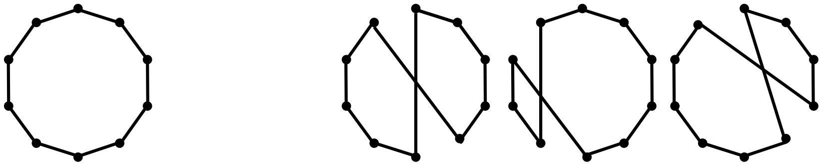
- **Step 1.** *Take arbitrary Hamiltonian cycle C as starting solution. (Starting cycle can be obtained as a result of a heuristics or can be generated by a pseudorandom generator).*
- **Step 2.** *Search for $C' \in \mathcal{O}(C)$ with $c(C') < c(C)$.*

  If for all $C' \in \mathcal{O}(C)$ $c(C') \geq c(C)$, STOP, C is suboptimal Hamiltonian cycle.

  Otherwise continye by Step 3.
- **Step 3.** *Take $C' \in \mathcal{O}(C)$ such that $c(C') < c(C)$ and set $C := C'$.*

  Goto Step 2.

Neighbourhood Search Algorithm

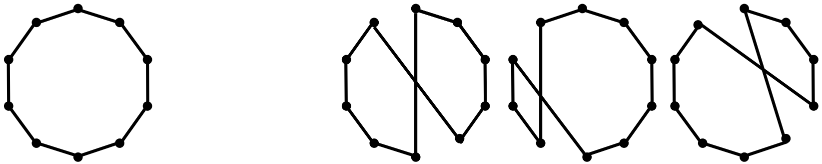


Cycle C and several elemenmts of its neighbourhood.

Danger of Neighbourhood Search Algorithm:
Algorithm gets stuck in a local minimum – in such solution which has no better solution in its neighbourhood.

Treatment:
Multifold runs of algorithm with different initial solutions.
Sophisticated heuristic algorithms.

Neighbourhood Search Algorithm



Cycle C and several elemenmts of its neighbourhood.

Danger of Neighbourhood Search Algorithm:
Algorithm gets stuck in a local minimum – in such solution which has no better solution in its neighbourhood.

Treatment:
Multifold runs of algorithm with different initial solutions.

Sophisticated heuristic algorithms.