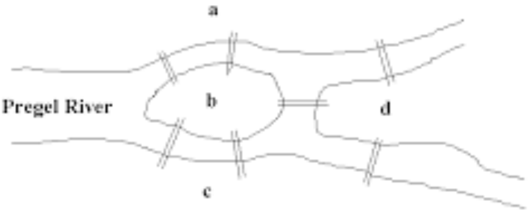
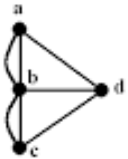
Eulerian and Hamiltonian graphs

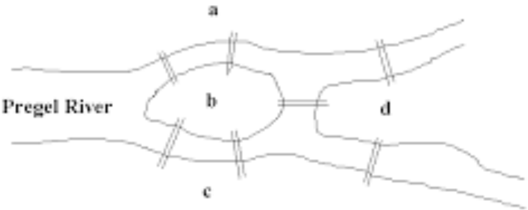
Sylwia Cichacz

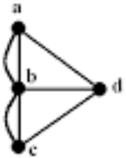
Akademia Górniczo-Hutnicza w Krakowie

December 19, 2021, Kraków


a
Pregel River
b
d
c


a
b
d
c





Problem: Does there exist a closed trail in the graph G that passes through all edges?

Eulerian graph

Definition:

An Eulerian cycle in a graph G is a closed walk that contains each edge exactly once (a closed trail).

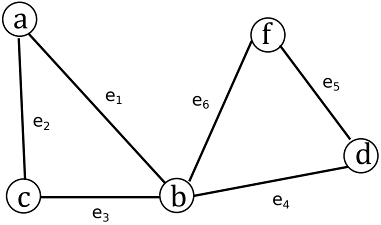
Definition:

An Eulerian cycle in a graph G is a closed walk that contains each edge exactly once (a closed trail).
An Eulerian graph is a connected graph that contains an Eulerian walk.

Eulerian graph

Definition:

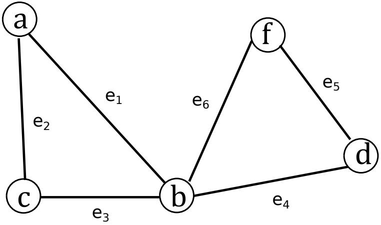
An Eulerian cycle in a graph G is a closed walk that contains each edge exactly once (a closed trail).
An Eulerian graph is a connected graph that contains an Eulerian walk.



Graph G.

Definition:

An Eulerian cycle in a graph G is a closed walk that contains each edge exactly once (a closed trail).
An Eulerian graph is a connected graph that contains an Eulerian walk.



Graph G.

Is G jest Eulerian?

Eulerian graph

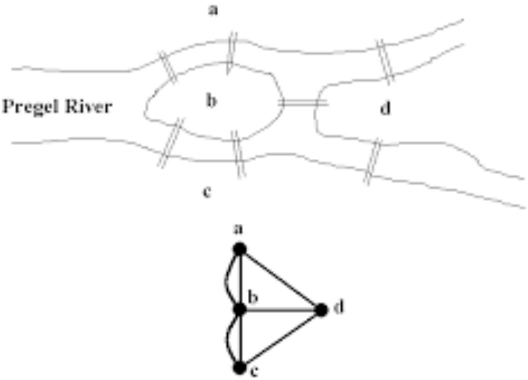
Theorem: L. Euler; 1736

A connected graph G possesses an Eulerian cycle iff the degree of every vertex is even.

Eulerian graph

Theorem: L. Euler; 1736

A connected graph G possesses an Eulerian cycle iff the degree of every vertex is even.

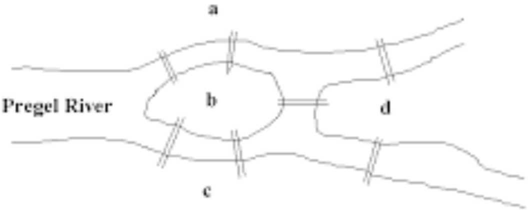


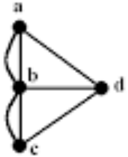
Definition:

An Eulerian walk is a graph G is that uses each edge exaclly once.

Definition:

An Eulerian walk is a graph G is that uses each edge exaclly once. If such a path exists, then the graph G is called semieulerian graph.

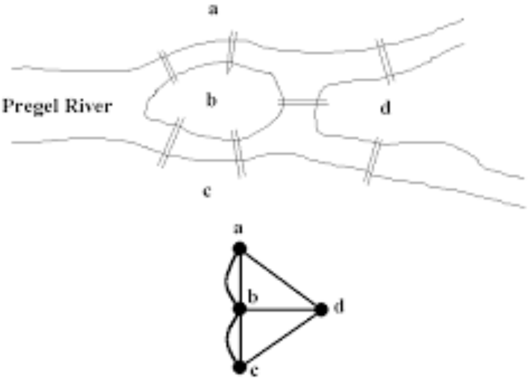




Eulerian graph

Definition:

An Eulerian walk is a graph G is that uses each edge exaclly once. If such a path exists, then the graph G is called semieulerian graph.



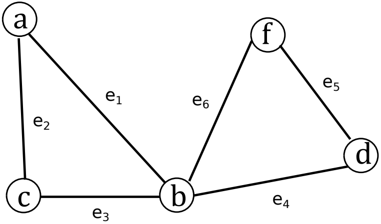
A semieulerian graph is a a graph that exactly two vertices of odd

Definition:

A bridge (cut-edge) is an edge of a graph whose deletion increases its number of connected components.

Definition:

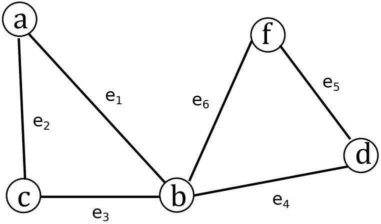
A bridge (cut-edge) is an edge of a graph whose deletion increases its number of connected components.



Graph G.

Definition:

A bridge (cut-edge) is an edge of a graph whose deletion increases its number of connected components.

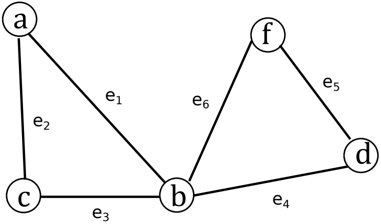


Graph G.

Does G have a bridge?

Definition:

A bridge (cut-edge) is an edge of a graph whose deletion increases its number of connected components.



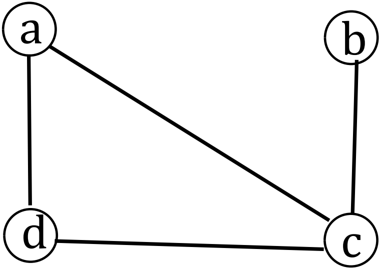
Graph G.

Does G have a bridge?
An edge is a bridge iff it is not contained in any cycle.

Eulerian graph

Definition:

A bridge (cut-edge) is an edge of a graph whose deletion increases its number of connected components.



Graph G.

Does G have a bridge?
An edge is a bridge iff it is not contained in any cycle.

Fleury's algorithm, 1883

Step 0: First make sure that the graph is connected and all vertices have even degree.

Fleury's algorithm, 1883

Step 0: First make sure that the graph is connected and all vertices have even degree.

Step 1: Start at any vertex.

Fleury's algorithm, 1883

Step 0: First make sure that the graph is connected and all vertices have even degree.

Step 1: Start at any vertex.

Step 2: Travel through an edge if

Fleury's algorithm, 1883

Step 0: First make sure that the graph is connected and all vertices have even degree.

Step 1: Start at any vertex.

Step 2: Travel through an edge if

(a) is not a bridge for the untraveled part, or

Fleury's algorithm, 1883

Step 0: First make sure that the graph is connected and all vertices have even degree.

Step 1: Start at any vertex.

Step 2: Travel through an edge if

- **(a)** is not a bridge for the untraveled part, or
- **(b)** there is no other alternative.

Fleury's algorithm, 1883

Step 0: First make sure that the graph is connected and all vertices have even degree.

Step 1: Start at any vertex.

Step 2: Travel through an edge if

- **(a)** is not a bridge for the untraveled part, or
- **(b)** there is no other alternative.

Step 3: Label the edges in the order in which you travel them.

Fleury's algorithm, 1883

Step 0: First make sure that the graph is connected and all vertices have even degree.

Step 1: Start at any vertex.

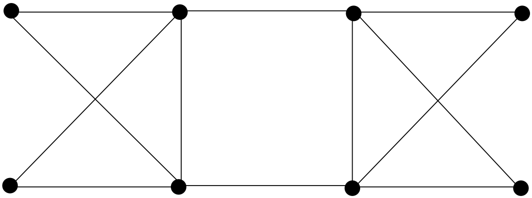
Step 2: Travel through an edge if

- **(a)** is not a bridge for the untraveled part, or
- **(b)** there is no other alternative.

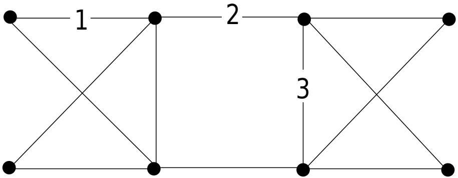
Step 3: Label the edges in the order in which you travel them.

Step 4: When you cannot travel any more, **STOP**. (You are done!)

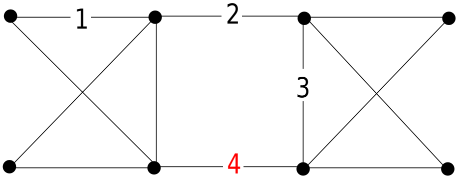
Fleury's algorithm, 1883



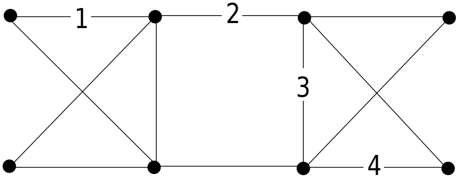
Fleury's algorithm, 1883



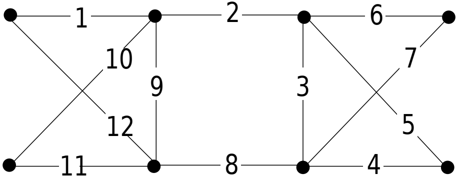
Fleury's algorithm, 1883



Fleury's algorithm, 1883

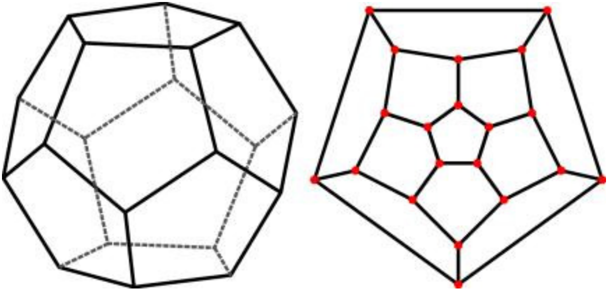


Fleury's algorithm, 1883

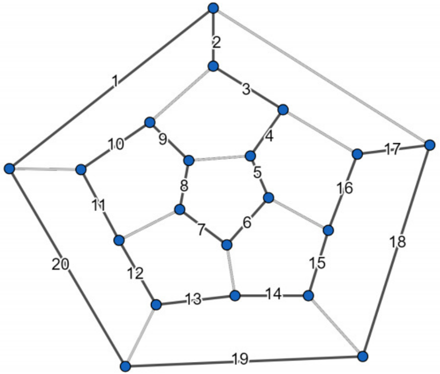


Problem: Does there exist a cycle in the graph G that passes through all vertices?

Problem: Does there exist a cycle in the graph G that passes through all vertices?



Problem: Does there exist a cycle in the graph G that passes through all vertices?



Definicja

A Hamiltonian cycle in a graph G is a cycle which contains all the vertices of G.

Hamiltonian graph

Definicja

A Hamiltonian cycle in a graph G is a cycle which contains all the vertices of G.
A Hamiltonian path in a graph G is a path which contains all the vertices of G.

Definicja

A *Hamiltonian cycle* *in a graph* G *is a cycle which contains all the vertices of* G.

A *Hamiltonian path* *in a graph* G *is a path which contains all the vertices of* G.

A graph G *which possesses a hamiltonian cycle is called the* *Hamiltonian graph.*

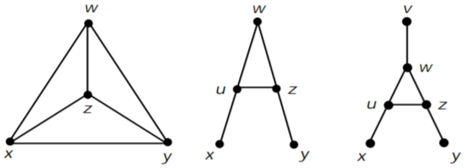
Hamiltonian graph

Definicja

A Hamiltonian cycle in a graph G is a cycle which contains all the vertices of G.
A Hamiltonian path in a graph G is a path which contains all the vertices of G.
A graph G which possesses a hamiltonian cycle is called the Hamiltonian graph.
A graph which possesses a hamiltonian path (but not cycle) is called the semihamiltonian graph.

Definicja

A Hamiltonian cycle in a graph G is a cycle which contains all the vertices of G.
A Hamiltonian path in a graph G is a path which contains all the vertices of G.
A graph G which possesses a hamiltonian cycle is called the Hamiltonian graph.
A graph which possesses a hamiltonian path (but not cycle) is called the semihamiltonian graph.



Hamiltonian graph via eulerian graph

Hamiltonian graph via eulerian graph

- For Eulericity, there was a nice, locally checkable necessary and sufficient condition.

Hamiltonian graph via eulerian graph

- For Eulericity, there was a nice, locally checkable necessary and sufficient condition.
- No such condition is known for Hamiltonicity.

Hamiltonian graph via eulerian graph

- For Eulericity, there was a nice, locally checkable necessary and sufficient condition.
- No such condition is known for Hamiltonicity.
- The question, whether a given graph G is Hamiltonian or not, is NP-complete.

Hamiltonian graph via eulerian graph

- For Eulericity, there was a nice, locally checkable necessary and sufficient condition.
- No such condition is known for Hamiltonicity.
- The question, whether a given graph G is Hamiltonian or not, is NP-complete.
- Hence the existence of a simple algorithm for checking it is unlikely.

Hamiltonian graph via eulerian graph

- For Eulericity, there was a nice, locally checkable necessary and sufficient condition.
- No such condition is known for Hamiltonicity.
- The question, whether a given graph G is Hamiltonian or not, is NP-complete.
- Hence the existence of a simple algorithm for checking it is unlikely.
- There exist easily checkable, sufficient, but not necessary conditions for Hamiltonicity.
  Many of them are are variations of "if a graph has many edges then it is Hamiltonian".

Hamiltonian graph

Theorem: Dirac, 1952

If a simple graph $G = (V, E)$ with $|V| = n > 3$ satisfies

$$\forall v \in V: \quad \deg(v) \geq \frac{n}{2}$$

then G is Hamiltonian.

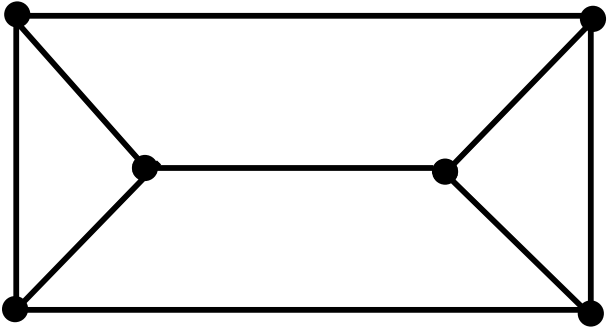
Hamiltonian graph

Theorem: Dirac, 1952

If a simple graph $G = (V, E)$ with $|V| = n > 3$ satisfies

$$\forall v \in V : \quad \deg(v) \geq \frac{n}{2}$$

then G is Hamiltonian.



Theorem: Dirac, 1952

If a simple graph $G = (V, E)$ with $|V| = n > 3$ satisfies

$$\forall v \in V : \quad \deg(v) \geq \frac{n}{2}$$

then G is Hamiltonian.

Theorem: Ore, 1960

If a simple graph $G = (V, E)$ with $|V| = n > 3$ satisfies

$$\forall v, w \in V : \quad (\text{if } \ uv \notin E \quad \Rightarrow \quad \deg(v) + \deg(w) \geq n)$$

then G is Hamiltonian.

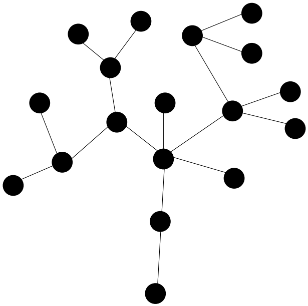
Definition

Definition:

A tree is a connected graph without cycles.

Definition:

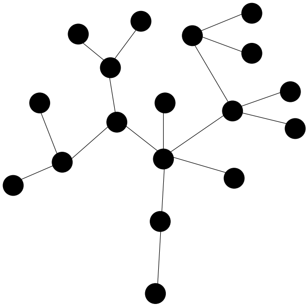
A tree is a connected graph without cycles.



Tree T

Definition:

A tree is a connected graph without cycles.



Tree T

Every tree is a bipartite graph.

Theorem:

If $G = (V, E)$ is a connected graph, then the following are equivalent.

Theorem:

If $G = (V, E)$ is a connected graph, then the following are equivalent.

1. G is a tree.

Theorem:

If $G = (V, E)$ is a connected graph, then the following are equivalent.

1. G is a tree.
2. G has no cycles.

Theorem:

If $G = (V, E)$ is a connected graph, then the following are equivalent.

1. G is a tree.
2. G has no cycles.
3. For every pair of distinct vertices u and v in G, there is exactly one path from u to v.

Theorem:

If $G = (V, E)$ is a connected graph, then the following are equivalent.

1. G is a tree.
2. G has no cycles.
3. For every pair of distinct vertices u and v in G, there is exactly one path from u to v.
4. $G - e$ is disconnected for any edge $e \in E(G)$.

Theorem:

If $G = (V, E)$ is a connected graph, then the following are equivalent.

1. G is a tree.
2. G has no cycles.
3. For every pair of distinct vertices u and v in G, there is exactly one path from u to v.
4. $G - e$ is disconnected for any edge $e \in E(G)$.
5. $|E| = |V| - 1$.

Theorem:

A tree with n vertices has exactly $n - 1$ edges.

Theorem

Theorem:

A tree with n vertices has exactly $n-1$ edges.

Theorem:

A tree with n vertices has exactly $n-1$ edges.

Proof:

Theorem:

A tree with n vertices has exactly $n - 1$ edges.

Proof: by induction.

Theorem:

A tree with n vertices has exactly $n-1$ edges.

Proof: by induction. If $n = 1$, the graph cannot have any edges or there would be a loop, with the vertex connecting to itself, so there must be $n-1 = 0$ edges.

Theorem:

A tree with n vertices has exactly $n - 1$ edges.

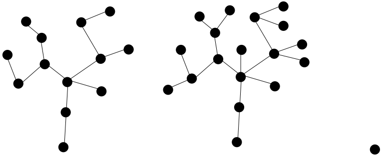
Proof: by induction. If $n = 1$, the graph cannot have any edges or there would be a loop, with the vertex connecting to itself, so there must be $n - 1 = 0$ edges.
Suppose that every tree with k vertices has precisely $k - 1$ edges.

Definition:

A forest is a disjoint union of trees.

Definition:

A forest is a disjoint union of trees.



Forest

Definition:

A Prüfer sequence of length $n - 2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

Definition:

A Prüfer sequence of length $n - 2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:

Definition:

A Prüfer sequence of length $n - 2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n - 2$ times:

Prüfera sequence

Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.

Definition:

A Prüfer sequence of length $n - 2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n - 2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.

Definition:

A Prüfer sequence of length $n - 2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n - 2$ times:

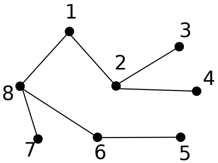
- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree

Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree

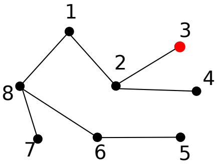


Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree

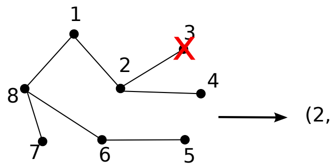


Definition:

A Prüfer sequence of length $n - 2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n - 2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree

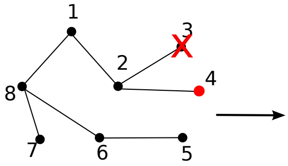


Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree



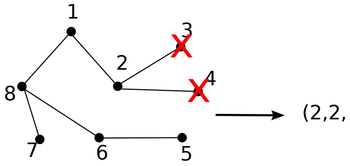
(2,

Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree

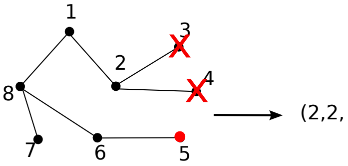


Definition:

A Prüfer sequence of length $n - 2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n - 2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree

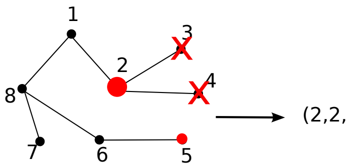


Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree



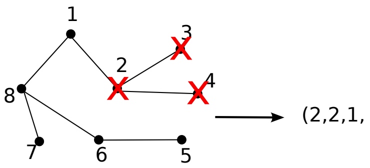
Prüfera sequence

Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree



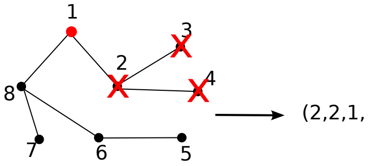
Prüfera sequence

Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree

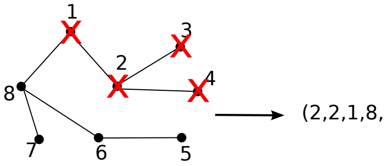


Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree

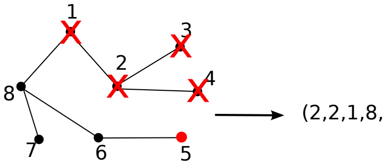


Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree

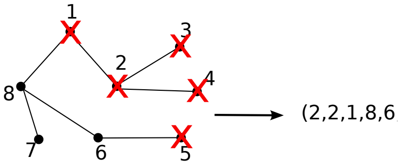


Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree

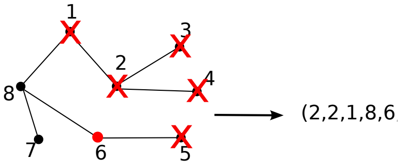


Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree

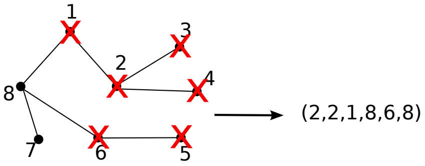


Definition:

A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

A labeled tree gives a Prüfer sequence by:
Repeat $n-2$ times:

- Pick the leaf with the smallest label, call it v.
- Put the label of v's neighbor in the output sequence.
- Remove v from the tree



Prüfer Decoding

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:
L – the ordered list of numbers $1, 2, \ldots, n$.

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

L=(1,2,3,4,5,6,7,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

L=(1,2,3,4,5,6,7,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

L=(1,2,3,4,5,6,7,8)

P=(2,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

L=(1,2,3,4,5,6,7,8)

P=(2,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

L=(1,2,3,4,5,6,7,8)

P=(2,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

L=(1,2,3,4,5,6,7,8)

P=(2,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.

L=(1,2,3,4,5,6,7,8)

P=(2,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.

L=(1,2,3,4,5,6,7,8)

P=(2,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.

L=(1,2,3,4,5,6,7,8)

P=(2,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n-2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.

L=(1,2,3,4,5,6,7,8)

P=(2,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.

L=(1,2,3,4,5,6,7,8)

P=(2,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.

L=(1,2,3,4,5,6,7,8)

P=(2,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.

L=(1,2,3,4,5,6,7,8)

P=(2,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.

L=(1,2,~~3~~,4,5,6,7,8)

P=(~~2~~,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.

L=(1,2,~~3~~,4,5,6,7,8)

P=(~~2~~,2,1,8,6,8)

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.

L=(1,2,~~3~~,4,5,6,7,8)

P=(~~3~~,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

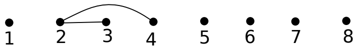
L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=(1,2,~~3~~,4,5,6,7,8)

P=(~~2~~,2,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

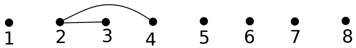
L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=(1,2,~~3~~,~~4~~,5,6,7,8)

P=(~~2~~,~~2~~,1,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n-2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.

L=(1,2,~~3~~,~~4~~,5,6,7,8)

P=(~~2~~,~~2~~,1,8,6,8)

A Prüfer sequence determines a labeled tree by:

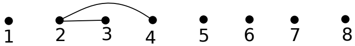
L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=$(1,\boxed{2},\cancel{3},\cancel{4},5,6,7,8)$

P=$(\cancel{2},\cancel{2},\boxed{1},8,6,8)$

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

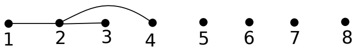
L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=(1,~~2~~,~~3~~,~~4~~,5,6,7,8)

P=(~~2~~,~~2~~,~~1~~,8,6,8)

A Prüfer sequence determines a labeled tree by:

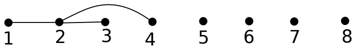
L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=(1,~~2~~,~~3~~,~~4~~,5,6,7,8)

P=(~~2~~,~~2~~,~~1~~,8,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

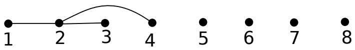
L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=(1,~~2~~,~~3~~,~~4~~,5,6,7,8)

P=(~~2~~,~~2~~,~~1~~,8,6,8)

A Prüfer sequence determines a labeled tree by:

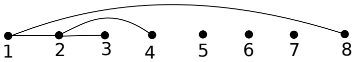
L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=(~~1~~,~~2~~,~~3~~,~~4~~,5,6,7,8)

P=(~~2~~,~~2~~,~~1~~,~~8~~,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

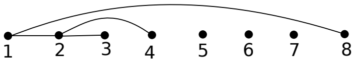
L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n-2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=(~~1~~,~~2~~,~~3~~,~~4~~,5,6,7,8)

P=(~~2~~,~~2~~,~~1~~,~~8~~,6,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

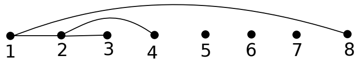
L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=(~~1~~,~~2~~,~~3~~,~~4~~,5,6,7,8)

P=(~~2~~,~~2~~,~~1~~,~~8~~,6,8)

A Prüfer sequence determines a labeled tree by:

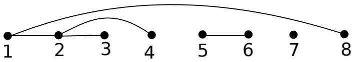
L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=(~~1~~,~~2~~,~~3~~,~~4~~,~~5~~,6,7,8)

P=(~~2~~,~~2~~,~~1~~,~~8~~,~~6~~,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

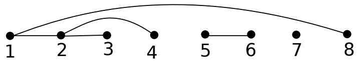
L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n-2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=(~~1~~,~~2~~,~~3~~,~~4~~,~~5~~,6,7,8)

P=(~~2~~,~~2~~,~~1~~,~~8~~,~~6~~,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

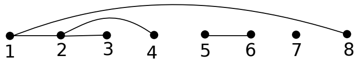
L – the ordered list of numbers $1, 2, \dots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=(~~1~~,~~2~~,~~3~~,~~4~~,~~5~~,6,7,8)

P=(~~2~~,~~2~~,~~1~~,~~8~~,~~6~~,8)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

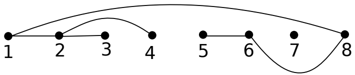
L – the ordered list of numbers $1, 2, \ldots, n$.

P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.



L=(~~1~~,~~2~~,~~3~~,~~4~~,~~5~~,~~6~~,7,8)

P=(~~2~~,~~2~~,~~1~~,~~8~~,~~6~~,~~8~~)

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

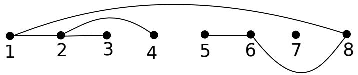
P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n - 2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.

When this is completed there will be two numbers left in L, add the edge corresponding to these two numbers.



$L=(\cancel{1},\cancel{2},\cancel{3},\cancel{4},\cancel{5},\cancel{6},7,8)$

$P=(\cancel{2},\cancel{2},\cancel{1},\cancel{8},\cancel{6},\cancel{8})$

Prüfer Decoding

A Prüfer sequence determines a labeled tree by:

L – the ordered list of numbers $1, 2, \ldots, n$.

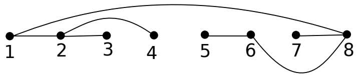
P – Prüfera sequence.

WStart with n labeled isolated vertices.

Repeat $n-2$ times:

- Let k be the smallest number in L which is not in P.
- Let j be the first number in P.
- Add the edge kj to the graph.
- Remove k from L and the first number in P.

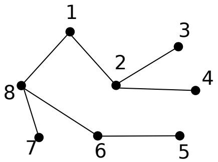
When this is completed there will be two numbers left in L, add the edge corresponding to these two numbers.

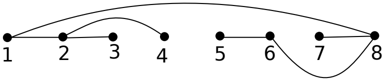


L=(~~1~~,~~2~~,~~3~~,~~4~~,~~5~~,~~6~~,7,8)

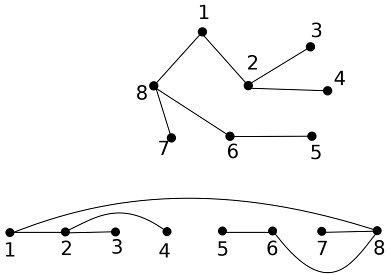
P=(~~2~~,~~2~~,~~1~~,~~8~~,~~6~~,~~8~~)

Prüfer coding and decoding





P=(2,2,1,8,6,8)



P=(2,2,1,8,6,8)

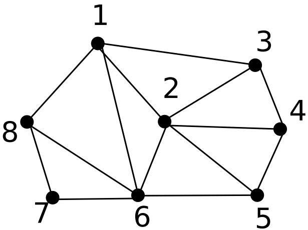
Prüfer coding and decoding are inverse operations, that means that there is a one-to-one correspondence between labeled trees with n vertices and Prüfer sequences of length $n-2$.

Definition:

A spanning tree T of a connected, undirected graph G is a tree composed of all the vertices and some (or perhaps all) of the edges of G.

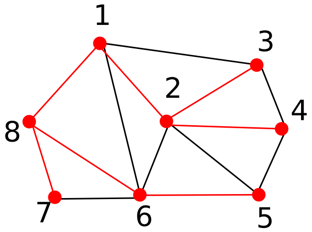
Definition:

A spanning tree T of a connected, undirected graph G is a tree composed of all the vertices and some (or perhaps all) of the edges of G.



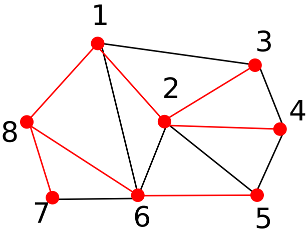
Definition:

A spanning tree T of a connected, undirected graph G is a tree composed of all the vertices and some (or perhaps all) of the edges of G.



Definition:

A spanning tree T of a connected, undirected graph G is a tree composed of all the vertices and some (or perhaps all) of the edges of G.

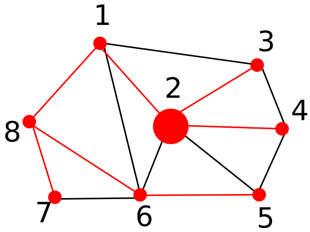


Definition:

A rooted tree is a tree with a vertex designated as root (it can be any vertex).

Definition:

A spanning tree T of a connected, undirected graph G is a tree composed of all the vertices and some (or perhaps all) of the edges of G.



Definition:

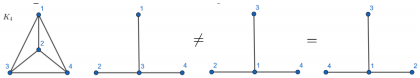
A rooted tree is a tree with a vertex designated as root (it can be any vertex). A rooted tree introduces a parent - child relationship between the vertices and the notion of depth in the tree.

Theorem: Cayley's Formula

The number of spanning labeled trees of K_n is n^{n-2}, for $n \geq 2$.

Theorem: Cayley's Formula

The number of spanning labeled trees of K_n is n^{n-2}, for $n \geq 2$.



Theorem: Cayley's Formula

The number of spanning labeled trees of K_n is n^{n-2}, for $n \geq 2$.

The number of spanning trees of K_n is the same as the number of sequences with $(n-2)$ elements with repetitions from the set $\{1, 2, \ldots, n\}$.

Cayley's Formula

Theorem: Cayley's Formula

The number of spanning labeled trees of K_n is n^{n-2}, for $n \geq 2$.

The number of spanning trees of K_n is the same as the number of sequences with $(n-2)$ elements with repetitions from the set $\{1, 2, \ldots, n\}$. Thus each Prüfer sequence corresponds to exactly one spanning tree of K_n.

Depth-First Search (DFS).

Depth-First Search (DFS).

1. Chose a root r a label it.

Depth-First Search (DFS).

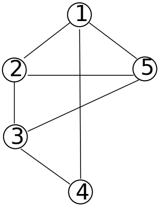
1. Chose a root r a label it.
2. Let $v := r$

Depth-First Search (DFS).

1. Chose a root r a label it.
2. Let $v := r$
3. Label a non-labeled neighbor w of v, $v := r$, otherwise go up.

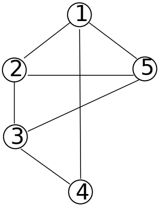
Depth-First Search (DFS).

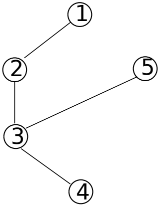
1. Chose a root r a label it.
2. Let $v := r$
3. Label a non-labeled neighbor w of v, $v := r$, otherwise go up.



Depth-First Search (DFS).

1. Chose a root r a label it.
2. Let $v := r$
3. Label a non-labeled neighbor w of v, $v := r$, otherwise go up.





DFS

Breadth-first search (BFS)

Breadth-first search (BFS)

1. Chose a root r a label it.

Breadth-first search (BFS)

1. Chose a root r a label it.
2. Let $v := r$

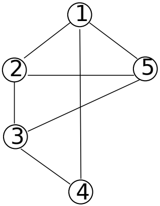
Breadth-first search

Breadth-first search (BFS)

1. Chose a root r a label it.
2. Let $v := r$
3. Label all non-labeled neighbors w of v, otherwise go up.

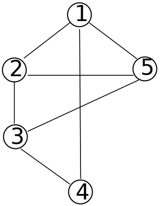
Breadth-first search (BFS)

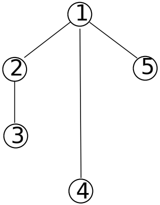
1. Chose a root r a label it.
2. Let $v := r$
3. Label all non-labeled neighbors w of v, otherwise go up.



Breadth-first search (BFS)

1. Chose a root r a label it.
2. Let $v := r$
3. Label all non-labeled neighbors w of v, otherwise go up.





BFS

Definition:

A weighted graph is a graph, in which each edge has a weight (some real number).

Weighted graphs

Definition:

A weighted graph is a graph, in which each edge has a weight (some real number). Formally $G = (V, E, w)$, where $w \colon E \to \mathbb{R}^+$.

Weighted graphs

Definition:

A weighted graph is a graph, in which each edge has a weight (some real number). Formally $G = (V, E, w)$, where $w \colon E \to \mathbb{R}^+$.
The weight of a graph is the sum of the weights of all edges:

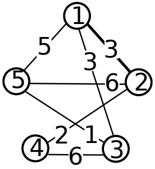
$$w(G) = \sum_{e \in E} w(e).$$

Weighted graphs

Definition:

A weighted graph is a graph, in which each edge has a weight (some real number). Formally $G = (V, E, w)$, where $w\colon E \to \mathbb{R}^+$. The weight of a graph is the sum of the weights of all edges:

$$w(G) = \sum_{e \in E} w(e).$$

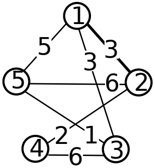


Graph G

Definition:

A weighted graph is a graph, in which each edge has a weight (some real number). Formally $G = (V, E, w)$, where $w\colon E \to \mathbb{R}^+$. The weight of a graph is the sum of the weights of all edges:

$$w(G) = \sum_{e \in E} w(e).$$



Graph G

$$w(G) = \sum_{e \in E} w(e) = 26.$$

Dijkstra's algorithm

Problem:

Dijkstra's algorithm

Problem: Find the shortest trail in the weighted graph from the root S.

Dijkstra's algorithm

Problem: Find the shortest trail in the weighted graph from the root S.

Solution: Dijkstra's algorithm

Problem: Find the shortest trail in the weighted graph from the root S.

Solution: Dijkstra's algorithm

$l(v)$ – distances (the weight of a path from s to v)

Problem: Find the shortest trail in the weighted graph from the root S.

Solution: Dijkstra's algorithm

$l(v)$ – distances (the weight of a path from s to v)

$p(v)$ – a neighbor of v on the path

Dijkstra's algorithm

Problem: Find the shortest trail in the weighted graph from the root S.

Solution: Dijkstra's algorithm

$l(v)$ – distances (the weight of a path from s to v)

$p(v)$ – a neighbor of v on the path

1. Let $l(s) = 0$, $l(v) = \infty$ for $v \neq s$ and $p(v) = 0$ for $v \in V$

Dijkstra's algorithm

Problem: Find the shortest trail in the weighted graph from the root S.

Solution: Dijkstra's algorithm

$l(v)$ – distances (the weight of a path from s to v)

$p(v)$ – a neighbor of v on the path

1. Let $l(s) = 0$, $l(v) = \infty$ for $v \neq s$ and $p(v) = 0$ for $v \in V$
2. Let $S = \emptyset$.
   Unless $S \neq V$ do

Dijkstra's algorithm

Problem: Find the shortest trail in the weighted graph from the root S.

Solution: Dijkstra's algorithm

$l(v)$ – distances (the weight of a path from s to v)

$p(v)$ – a neighbor of v on the path

1. Let $l(s) = 0$, $l(v) = \infty$ for $v \neq s$ and $p(v) = 0$ for $v \in V$
2. Let $S = \emptyset$.

   Unless $S \neq V$ do

   1. pick $u \in V \setminus S$ with minimal $l(u)$ and add it to S

Problem: Find the shortest trail in the weighted graph from the root S.

Solution: Dijkstra's algorithm

$l(v)$ – distances (the weight of a path from s to v)

$p(v)$ – a neighbor of v on the path

1. Let $l(s) = 0$, $l(v) = \infty$ for $v \neq s$ and $p(v) = 0$ for $v \in V$
2. Let $S = \emptyset$.

   Unless $S \neq V$ do

   1. pick $u \in V \setminus S$ with minimal $l(u)$ and add it to S
   2. $\forall v \in V \setminus S$: $uv \in E(G)$ and $l(v) > l(u) + w(vu)$ set $l(v) = l(u) + w(vu)$ and $p(v) = u$ if it is smallest than the previous one, then we put it in the table

Problem: Find the shortest trail in the weighted graph from the root S.

Solution: Dijkstra's algorithm

$l(v)$ – distances (the weight of a path from s to v)

$p(v)$ – a neighbor of v on the path

1. Let $l(s) = 0$, $l(v) = \infty$ for $v \neq s$ and $p(v) = 0$ for $v \in V$
2. Let $S = \emptyset$.
   Unless $S \neq V$ do
   1. pick $u \in V \setminus S$ with minimal $l(u)$ and add it to S
   2. $\forall v \in V \setminus S$: $uv \in E(G)$ and $l(v) > l(u) + w(vu)$ set $l(v) = l(u) + w(vu)$ and $p(v) = u$ if it is smallest than the previous one, then we put it in the table

Problem: Find the shortest trail in the weighted graph from the root S.

Solution: Dijkstra's algorithm

$l(v)$ – distances (the weight of a path from s to v)

$p(v)$ – a neighbor of v on the path

1. Let $l(s) = 0$, $l(v) = \infty$ for $v \neq s$ and $p(v) = 0$ for $v \in V$
2. Let $S = \emptyset$.

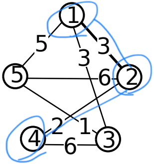
   Unless $S \neq V$ do

   1. pick $u \in V \setminus S$ with minimal $l(u)$ and add it to S
   2. $\forall v \in V \setminus S : uv \in E(G)$ and $l(v) > l(u) + w(vu)$ set $l(v) = l(u) + w(vu)$ and $p(v) = u$ if it is smallest than the previous one, then we put it in the table

The shortest path from s to v is $v, p(v), p(p(v)), \dots, s$.

Dijkstra's algorithm

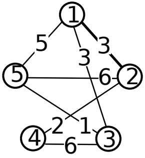
Dijkstra's algorithm



	1	2	3	4	5	S
1	0	∞	∞	∞	∞	1
2	x	3	3	∞	5	2
3	x	x	3	5	5	3
4	x	x	x	5	4	5
5	x	x	x	5	x	4

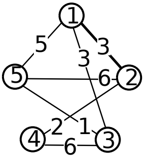
	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1
4	x	x	x	2	3
5	✓	✓	X	2	X

Dijkstra's algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	

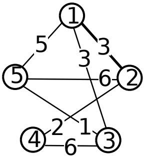
	1	2	3	4	5
1	0	0	0	0	0



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1

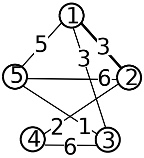
	1	2	3	4	5
1	0	0	0	0	0

Dijkstra's algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	

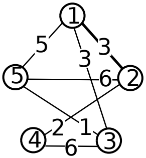
	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1

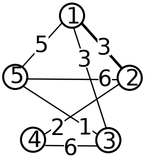
Dijkstra's algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	5	5	

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1

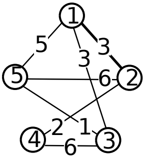
Dijkstra's algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	5	5	3

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1

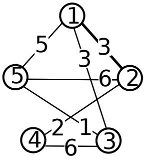
Dijkstra’s algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	5	5	3
4	x	x	x	5	4	

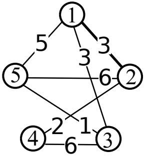
	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1
4	x	x	x	2	3

Dijkstra's algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	5	5	3
4	x	x	x	5	4	5

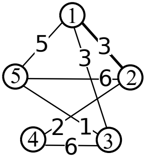
	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1
4	x	x	x	2	3



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	5	5	3
4	x	x	x	5	4	5
5	x	x	x	5	x	

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1
4	x	x	x	2	3
5	x	x	x	2	x

Dijkstra’s algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	5	5	3
4	x	x	x	5	4	5
5	x	x	x	5	x	4

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1
4	x	x	x	2	3
5	x	x	x	2	x

Definition:

A minimum spanning tree in an undirected connected weighted graph is a spanning tree of minimum weight (among all spanning trees).

Definition:

A minimum spanning tree in an undirected connected weighted graph is a spanning tree of minimum weight (among all spanning trees).

Uwaga: Remark: The minimum spanning tree may not be unique.

Weighted graphs

Definition:

A minimum spanning tree in an undirected connected weighted graph is a spanning tree of minimum weight (among all spanning trees).

Uwaga: Remark: The minimum spanning tree may not be unique. However, if the weights of all the edges are pairwise distinct, it is indeed unique.

Kruscal algorithm

Kruscal algorithm

Let $T = (V, E)$ be a tree of size $|E| = m$

Kruscal algorithm

Let $T = (V, E)$ be a tree of size $|E| = m$

1. $T = (V, E_1)$, $E_1 = \varnothing$, (T – forest of isolated vertices)

Kruscal algorithm

Let $T = (V, E)$ be a tree of size $|E| = m$

1. $T = (V, E_1)$, $E_1 = \emptyset$, (T – forest of isolated vertices)
2. Let $e_1, \ldots, e_m$: $w(e_1) \leq w(e_2) \leq \ldots w(e_m)$

Kruscal algorithm

Let $T = (V, E)$ be a tree of size $|E| = m$

1. $T = (V, E_1)$, $E_1 = \emptyset$, (T – forest of isolated vertices)
2. Let $e_1, \ldots, e_m$: $w(e_1) \leq w(e_2) \leq \ldots w(e_m)$
3. for $i = 1, \ldots, m$ if $e_i = uv$ for u and v in different componets of T, then $e_i \in E_1$

Kruscal algorithm

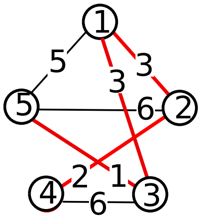
Let $T = (V, E)$ be a tree of size $|E| = m$

1. $T = (V, E_1)$, $E_1 = \varnothing$, (T – forest of isolated vertices)
2. Let $e_1, \ldots, e_m$: $w(e_1) \le w(e_2) \le \ldots w(e_m)$
3. for $i = 1, \ldots, m$ if $e_i = uv$ for u and v in different componets of T, then $e_i \in E_1$

Kruscal algorithm

Let $T = (V, E)$ be a tree of size $|E| = m$

1. $T = (V, E_1)$, $E_1 = \emptyset$, (T – forest of isolated vertices)
2. Let $e_1, \ldots, e_m$: $w(e_1) \leq w(e_2) \leq \ldots w(e_m)$
3. for $i = 1, \ldots, m$ if $e_i = uv$ for u and v in different componets of T, then $e_i \in E_1$



$v_3v_5, v_2v_4, v_1v_3, v_1v_2, v_1v_5, v_3v_4, v_3v_6$

$E_1 = (v_3v_5, v_2v_4, v_1v_3, v_1v_2)$

Prim algorithm

Prim algorithm similar to Dijkstra (we count distances to the tree not a vertex).

Prim algorithm similar to Dijkstra (we count distances to the tree not a vertex).
$k(v)$ – a distance form a tree

Prim algorithm

Prim algorithm similar to Dijkstra (we count distances to the tree not a vertex).
$k(v)$ – a distance form a tree
$p(v)$ – a neighbor of v in the tree

Prim algorithm

Prim algorithm similar to Dijkstra (we count distances to the tree not a vertex).

$k(v)$ – a distance form a tree

$p(v)$ – a neighbor of v in the tree

1. Take any $s \in V$ and let $l(s) = 0$, $k(v) = \infty$ for $v \neq s$ and $p(v) = 0$ for $v \in V(S)$

Prim algorithm similar to Dijkstra (we count distances to the tree not a vertex).

$k(v)$ – a distance form a tree

$p(v)$ – a neighbor of v in the tree

1. Take any $s \in V$ and let $l(s) = 0$, $k(v) = \infty$ for $v \neq s$ and $p(v) = 0$ for $v \in V(S)$
2. Let $S = \emptyset$.
   Unless $S \neq V$ do

Prim algorithm similar to Dijkstra (we count distances to the tree not a vertex).
$k(v)$ – a distance form a tree
$p(v)$ – a neighbor of v in the tree

1. Take any $s \in V$ and let $l(s) = 0$, $k(v) = \infty$ for $v \neq s$ and $p(v) = 0$ for $v \in V(S)$
2. Let $S = \emptyset$.
   Unless $S \neq V$ do
   1. pick $u \in V \setminus S$ with minimal $k(u)$ and add it to S

Prim algorithm

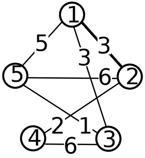
Prim algorithm similar to Dijkstra (we count distances to the tree not a vertex).

$k(v)$ – a distance form a tree

$p(v)$ – a neighbor of v in the tree

1. Take any $s \in V$ and let $l(s) = 0$, $k(v) = \infty$ for $v \neq s$ and $p(v) = 0$ for $v \in V(S)$
2. Let $S = \emptyset$.
   Unless $S \neq V$ do
   1. pick $u \in V \setminus S$ with minimal $k(u)$ and add it to S
   2. $\forall v \in V \setminus S$: $uv \in E(G)$ and $k(v) > (vu)$ set $k(v) = w(vu)$ and $p(v) = u$ moreover $E_1 = \{vp(v) : v \in S\}$

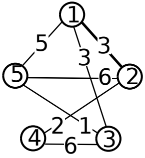
Prim algorithm



	1	2	3	4	5	S

	1	2	3	4	5

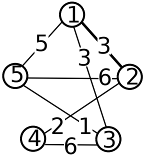
Prim algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	

	1	2	3	4	5
1	0	0	0	0	0

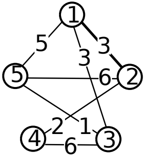
Prim algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1

	1	2	3	4	5
1	0	0	0	0	0

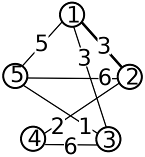
Prim algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1

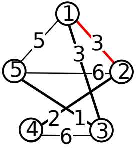
Prim algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1

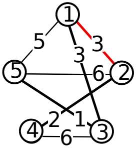
Prim algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	2	5	

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1

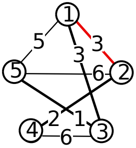
Prim algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	2	5	4

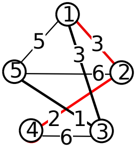
	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1

Prim algorithm



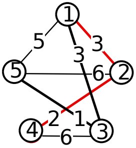
	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	2	5	4

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	2	5	4

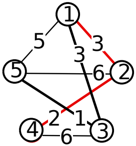
	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	2	5	4
4	x	x	3	x	5	

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1
4	x	x	1	x	1

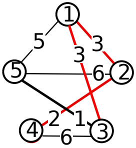
Prim algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	2	5	4
4	x	x	3	x	5	3

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1
4	x	x	1	x	1

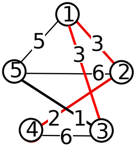
Prim algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	2	5	4
4	x	x	3	x	5	3

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1
4	x	x	1	x	1

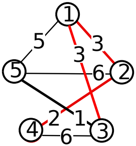
Prim algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	2	5	4
4	x	x	3	x	5	3
5	x	x	x	x	1	

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1
4	x	x	1	x	1
5	x	x	x	x	3

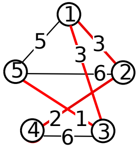
Prim algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	2	5	4
4	x	x	3	x	5	3
5	x	x	x	x	1	5

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1
4	x	x	1	x	1
5	x	x	x	x	3

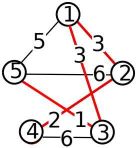
Prim algorithm



	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	2	5	4
4	x	x	3	x	5	3
5	x	x	x	x	1	5

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1
4	x	x	1	x	1
5	x	x	x	x	3

Prim algorithm



$E_1=(v_3v_5,v_2v_4,v_1v_3,v_1v_2)$

	1	2	3	4	5	S
1	0	oo	oo	oo	oo	1
2	x	3	3	oo	5	2
3	x	x	3	2	5	4
4	x	x	3	x	5	3
5	x	x	x	x	1	5

	1	2	3	4	5
1	0	0	0	0	0
2	x	1	1	0	1
3	x	x	1	2	1
4	x	x	1	x	1
5	x	x	x	x	3