Acyclic graphs, trees and spanning trees

Stanislav Palúch

Fakulta riadenia a informatiky, Žilinská univerzita

20. apríla 2016

Opakovanie – Cyklus

Definition

Cycle (directed cycle, quasi-cycle) *is non trivial closed walk (directed walk, quasi-walk) in which every vertex but the first and the last appears at most once.*

Acyklic graph and tree

Definition

An acyclic graph *is a graph that has no cycles.*

Definition

A tree *is a connected acyclic graph.*

Remark

Trivial graph is a tree.

Remark

Every component of an acyclic graph is a tree (is connected and dous not contain a cycle). Hence an acyclic graph can be considered as several trees. That is why the term **forest** *is often used as a synonym for „acyclic graph".*

Acyklic graph and tree

Definition

An acyclic graph *is a graph that has no cycles.*

Definition

A tree *is a connected acyclic graph.*

Remark

Trivial graph is a tree.

Remark

Every component of an acyclic graph is a tree (is connected and dous not contain a cycle). Hence an acyclic graph can be considered as several trees. That is why the term **forest** *is often used as a synonym for „acyclic graph".*

Acyklic graph and tree

Definition

An acyclic graph *is a graph that has no cycles.*

Definition

A tree *is a connected acyclic graph.*

Remark

Trivial graph is a tree.

Remark

Every component of an acyclic graph is a tree (is connected and dous not contain a cycle). Hence an acyclic graph can be considered as several trees. That is why the term **forest** *is often used as a synonym for „acyclic graph".*

Acyklic graph and tree

Definition

An acyclic graph *is a graph that has no cycles.*

Definition

A tree *is a connected acyclic graph.*

Remark

Trivial graph is a tree.

Remark

Every component of an acyclic graph is a tree (is connected and dous not contain a cycle). Hence an acyclic graph can be considered as several trees. That is why the term **forest** *is often used as a synonym for „acyclic graph“.*

A nontrivial tree contains at least 2 vertices with degree 1

Theorem

Let $G = (V, H)$ be a tree with at least two vertices.
Then the set V contains at least two vertices with degree 1.

PROOF.
Let

$$(v_1, \{v_1, v_2\}, v_2, \dots, \{v_{k-1}, v_k\}, v_k) \tag{1}$$

be a path in treee G with largest number of edges. We show that $\deg(v_k) = 1$.

Obr.: If $\deg(v_k) > 1$,

then there exists at least one edge (dashed) incident with v_k, creating one of situations a) or b).

A nontrivial tree contains at least 2 vertices with degree 1

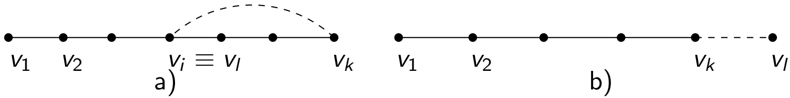
Theorem

Let $G = (V, H)$ be a tree with at least two vertices.
Then the set V contains at least two vertices with degree 1.

PROOF.
Let

$$(v_1, \{v_1, v_2\}, v_2, \ldots, \{v_{k-1}, v_k\}, v_k) \tag{1}$$

be a path in treee G with largest number of edges. We show that $\deg(v_k) = 1$.



Obr.: If $\deg(v_k) > 1$,

then there exists at least one edge (dashed) incident with v_k, creating one of situations a) or b).

Theorem

Following assertions are equivalent:

- *a)* $G = (V, H)$ *is a tree.*
- *b) There exists exactly one u–v path in graph* $G = (V, H)$ *for every* $u, v \in V$.
- *c) Graph* $G = (V, H)$ *is connected and every edge* $h \in H$ *is a bridge in* G.
- *d) Graph* $G = (V, H)$ *is connected and* $|H| = |V| - 1$.
- *e) Graph* $G = (V, H)$ *is acyclic and* $|H| = |V| - 1$.

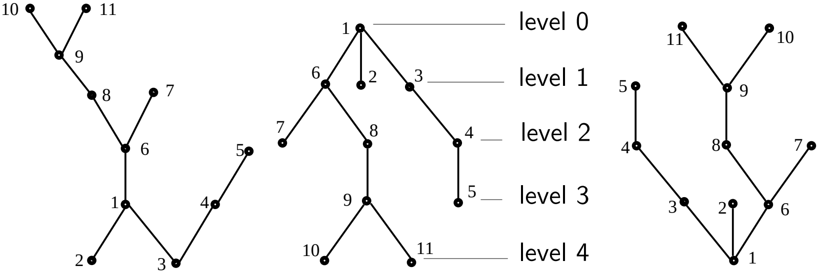
Koreňový tree

Definition

A rooted tree *is a tree* $G = (V, H)$ *having a distinguished vertex* $k \in V$, *called* **the root**.

The level of vertex u *or* **the depth of a vertex** u *in rooted tree* $G = (V, H)$ *with root* k *is the length (number of edges) of (unique)* k–u *paht.*

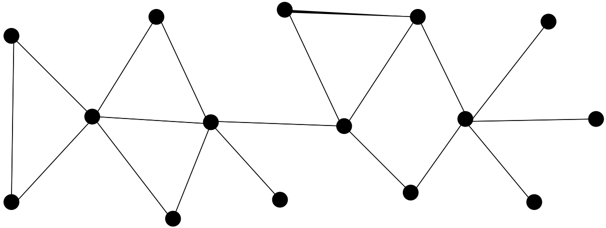
The height of the rooted tree $G = (V, H)$ *is the maximum of levels of all vertices of the rooted tree* G.



Obr.: Several ways how to draw a diagram of a rooted tree with root 1.

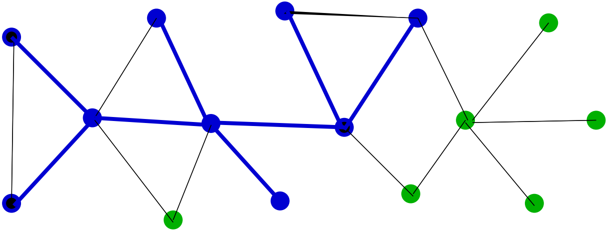
Definition

Let the tree $T = (V_T, H_T)$ is a subgraph of graph $G = (V, H)$. We will say that the edge $h = \{u, v\} \in H$ is **the border edge**, *if $u \in V_T$ and $v \notin V_T$.*
Let $h = \{u, v\}$ be a border edge, $u \in V_T$, $v \notin V_T$. We will say that u je **the included vertex**, *v is* **the free vertex** *of border edge h.*



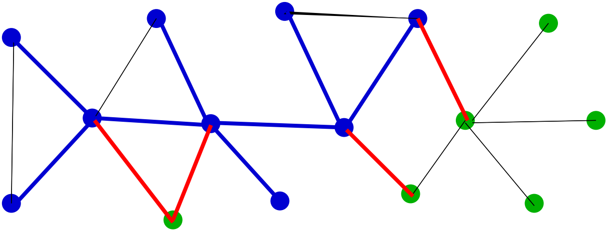
Definition

Let the tree $T = (V_T, H_T)$ is a subgraph of graph $G = (V, H)$. We will say that the edge $h = \{u, v\} \in H$ is **the border edge**, *if $u \in V_T$ and $v \notin V_T$.*
Let $h = \{u, v\}$ be a border edge, $u \in V_T$, $v \notin V_T$. We will say that u je **the included vertex**, *v is* **the free vertex** *of border edge h.*



Definition

Let the tree $T = (V_T, H_T)$ is a subgraph of graph $G = (V, H)$. We will say that the edge $h = \{u, v\} \in H$ is **the border edge**, *if $u \in V_T$ and $v \notin V_T$.*
Let $h = \{u, v\}$ be a border edge, $u \in V_T$, $v \notin V_T$. We will say that u je **the included vertex**, *v is* **the free vertex** *of border edge h.*



Algorithm

Depth-First Search.

- **Step 1. Initialization.**
  Let the tree T be a trivial tree containing single vertex $v \in V$.
  Set $p(v) := 1$, $k := 1$.
- **Step 2.** *If T does not contain all vertices of graph GOTO Step 3. otherwise STOP.*
- **Step 3.** *Find a border line $h = \{u, v\}$ in graph G with tree T* **with maximal label** *$p(u)$ of included vertex u.*
  If such an edge does not exist STOP.
  Otherwise continue in Step 4.
- **Step 4.** *Set $T := T \cup \{h\} \cup \{v\}$, $k := k + 1$, $p(v) := k$.*
  GOTO Step 2.

Algorithm

Depth-First Search.

- **Step 1. Initialization.**
  Let the tree T be a trivial tree containing single vertex $v \in V$.
  Set $p(v) := 1$, $k := 1$.
- **Step 2.** *If T does not contain all vertices of graph GOTO Step 3. otherwise STOP.*
- **Step 3.** *Find a border line $h = \{u, v\}$ in graph G with tree T* **with maximal label** *$p(u)$ of included vertex u.*
  If such an edge does not exist STOP.
  Otherwise continue in Step 4.
- **Step 4.** *Set $T := T \cup \{h\} \cup \{v\}$, $\quad k := k + 1$, $\quad p(v) := k$.*
  GOTO Step 2.

Algorithm

Depth-First Search.

- **Step 1. Initialization.**
  Let the tree T *be a trivial tree containing single vertex* $v \in V$.
  Set $p(v) := 1$, $k := 1$.
- **Step 2.** *If* T *does not contain all vertices of graph GOTO Step 3. otherwise STOP.*
- **Step 3.** *Find a border line* $h = \{u, v\}$ *in graph* G *with tree* T **with maximal label** $p(u)$ *of included vertex* u.
  If such an edge does not exist STOP.
  Otherwise continue in Step 4.
- **Step 4.** *Set* $T := T \cup \{h\} \cup \{v\}$, $k := k + 1$, $p(v) := k$.
  GOTO Step 2.

Algorithm

Depth-First Search.

- **Step 1. Initialization.**
  Let the tree T be a trivial tree containing single vertex $v \in V$.
  Set $p(v) := 1$, $k := 1$.
- **Step 2.** *If T does not contain all vertices of graph GOTO Step 3.*
  otherwise STOP.
- **Step 3.** *Find a border line $h = \{u, v\}$ in graph G with tree T* **with maximal label** *$p(u)$ of included vertex u.*
  If such an edge does not exist STOP.
  Otherwise continue in Step 4.
- **Step 4.** *Set $T := T \cup \{h\} \cup \{v\}$, $\quad k := k + 1$, $\quad p(v) := k$.*
  GOTO Step 2.

♣

Algorithm

Breadth-First Search.

- **Step 1.** *Initialization. Let T be a trivial tree containing single vertex $v \in V$. Set $p(v) := 1$, $k := 1$.*
- **Step 2.** *If T does not contain all vertices of graph GOTO Step 3. otherwise STOP.*
- **Step 3.** *Find a border line $h = \{u, v\}$ in graph G with tree T* **with minimal** *$p(u)$ of included vertex u.*
  If such an edge does not exist STOP.
  Otherwise continue in Step 4.
- **Step 4.** *Set $T := T \cup \{h\} \cup \{v\}$, $\quad k := k + 1$, $\quad p(v) := k$. GOTO Step 2.*

♣

Breadth-First Search

Algorithm

Breadth-First Search.

- **Step 1.** *Initialization. Let T be a trivial tree containing single vertex $v \in V$. Set $p(v) := 1$, $k := 1$.*
- **Step 2.** *If T does not contain all vertices of graph GOTO Step 3. otherwise STOP.*
- **Step 3.** *Find a border line $h = \{u, v\}$ in graph G with tree T* **with minimal** *$p(u)$ of included vertex u.*
  If such an edge does not exist STOP.
  Otherwise continue in Step 4.
- **Step 4.** *Set $T := T \cup \{h\} \cup \{v\}$, $k := k + 1$, $p(v) := k$. GOTO Step 2.*

♣

Algorithm

Breadth-First Search.

- **Step 1.** *Initialization. Let T be a trivial tree containing single vertex $v \in V$. Set $p(v) := 1$, $k := 1$.*
- **Step 2.** *If T does not contain all vertices of graph GOTO Step 3. otherwise STOP.*
- **Step 3.** *Find a border line $h = \{u, v\}$ in graph G with tree T* **with minimal** *$p(u)$ of included vertex u.*
  If such an edge does not exist STOP.
  Otherwise continue in Step 4.
- **Step 4.** *Set $T := T \cup \{h\} \cup \{v\}$, $k := k + 1$, $p(v) := k$. GOTO Step 2.*

♣

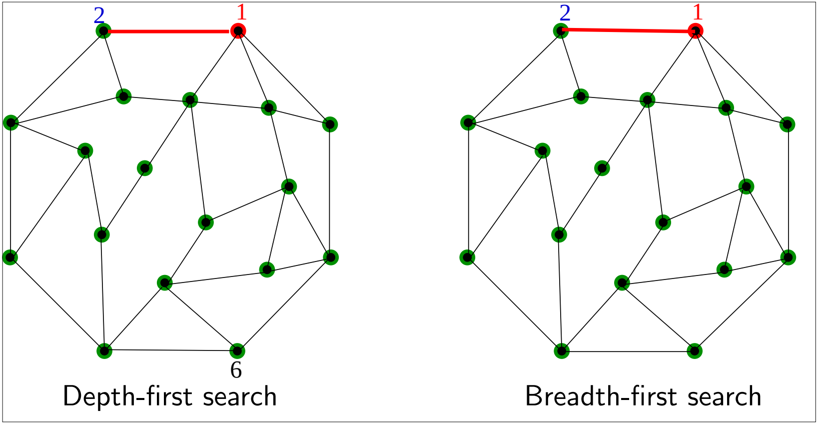
Algorithm

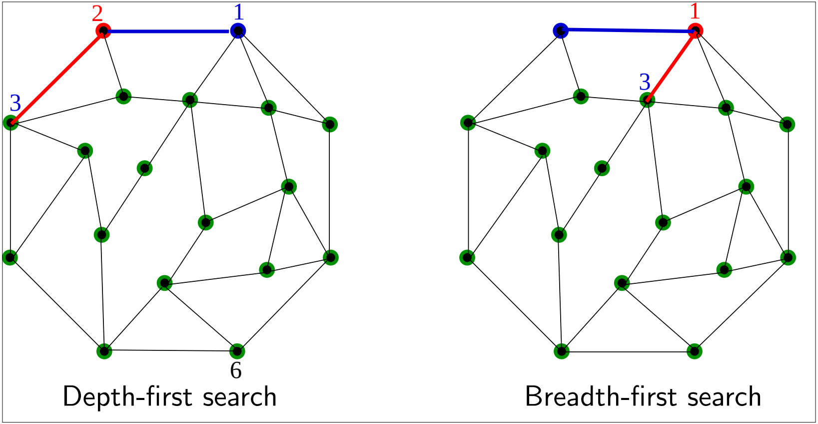
Breadth-First Search.

- **Step 1.** *Initialization. Let T be a trivial tree containing single vertex $v \in V$. Set $p(v) := 1$, $k := 1$.*
- **Step 2.** *If T does not contain all vertices of graph GOTO Step 3. otherwise STOP.*
- **Step 3.** *Find a border line $h = \{u, v\}$ in graph G with tree T* **with minimal** *$p(u)$ of included vertex u.*
  If such an edge does not exist STOP.
  Otherwise continue in Step 4.
- **Step 4.** *Set $T := T \cup \{h\} \cup \{v\}$, $\quad k := k + 1$, $\quad p(v) := k$.*
  GOTO Step 2.

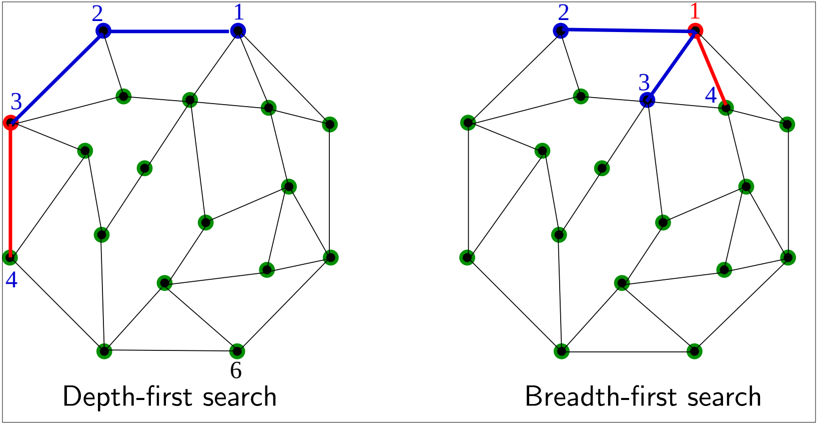
♣

Depth-first search and breadth-first search

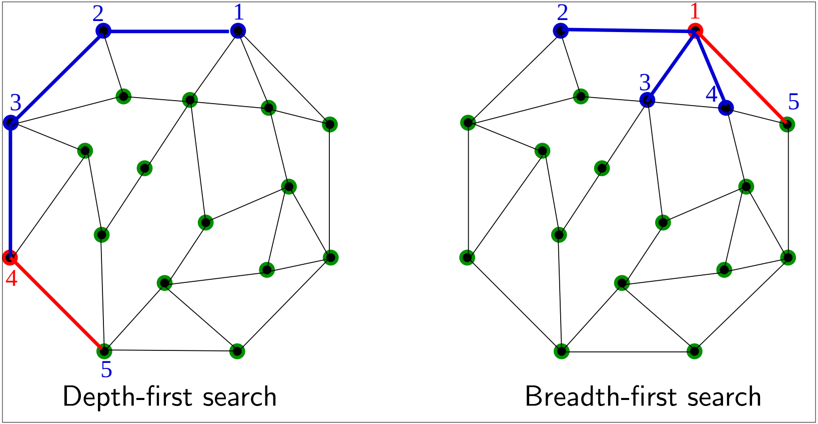



2
1
3
6
Depth-first search
1
3
Breadth-first search

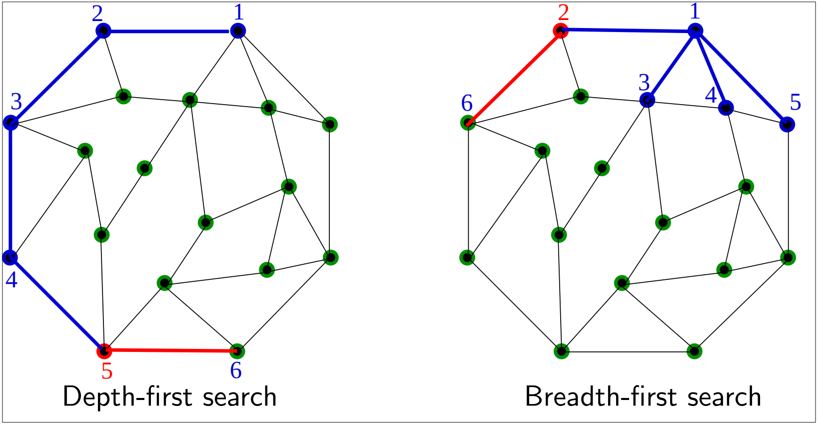
Depth-first search and breadth-first search

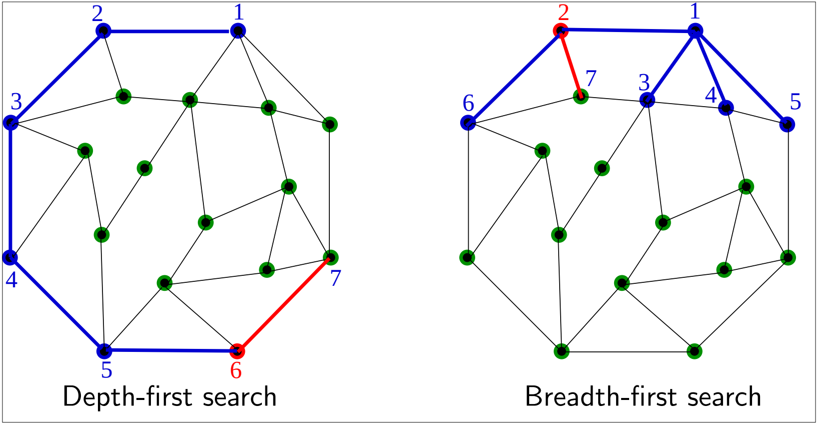


Depth-first search and breadth-first search

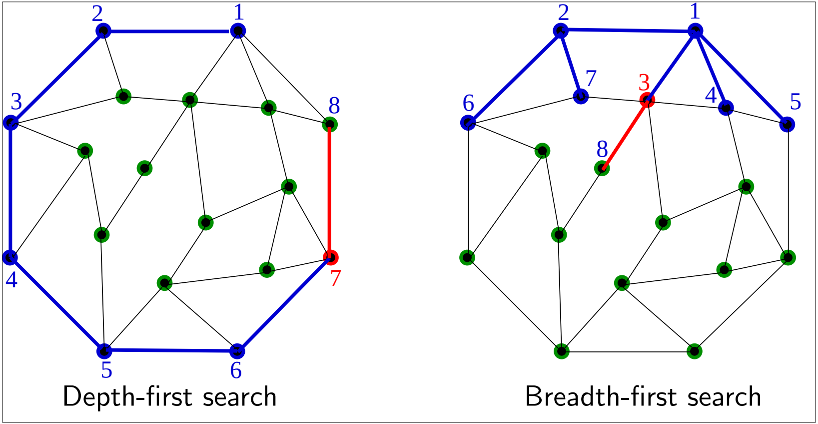


Depth-first search and breadth-first search

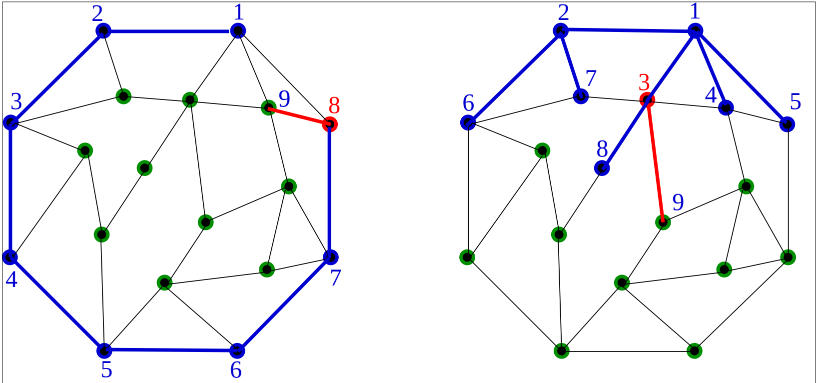



2
1
3
4
5
6
7
Depth-first search
2
1
7
3
4
5
6
Breadth-first search

Depth-first search and breadth-first search

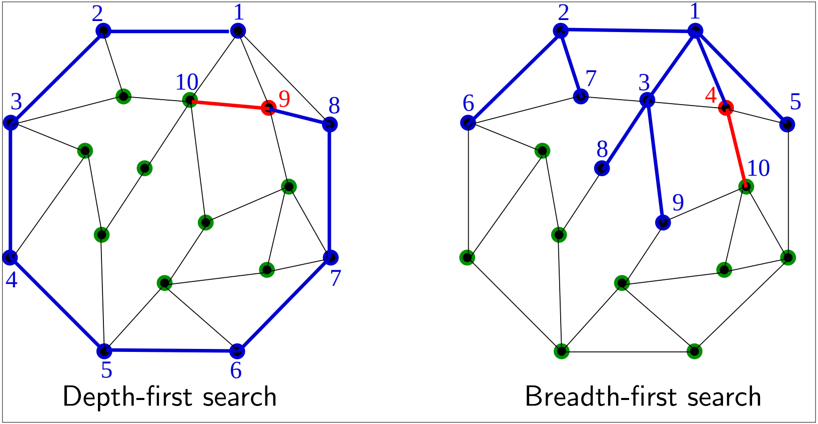


Depth-first search and breadth-first search

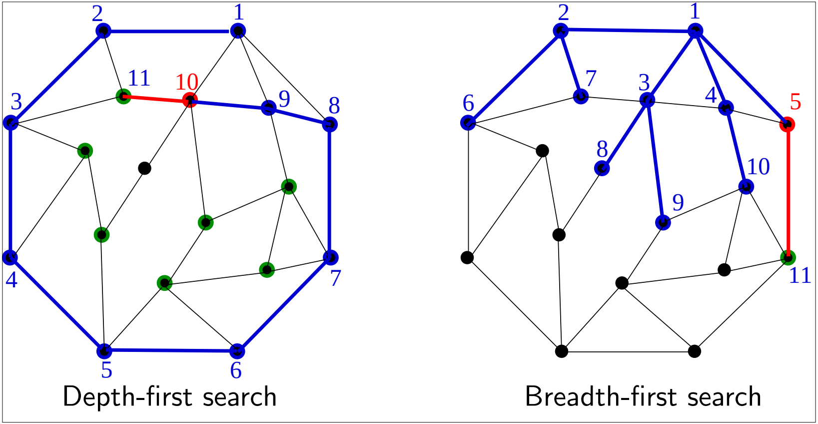


Depth-first search Breadth-first search

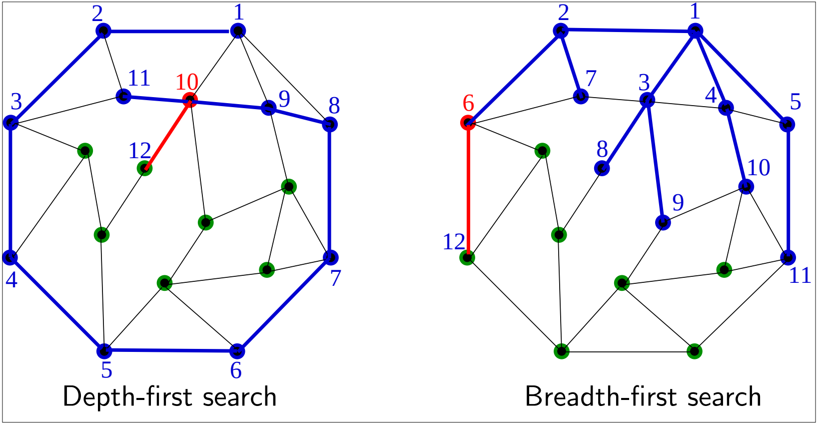
Depth-first search and breadth-first search

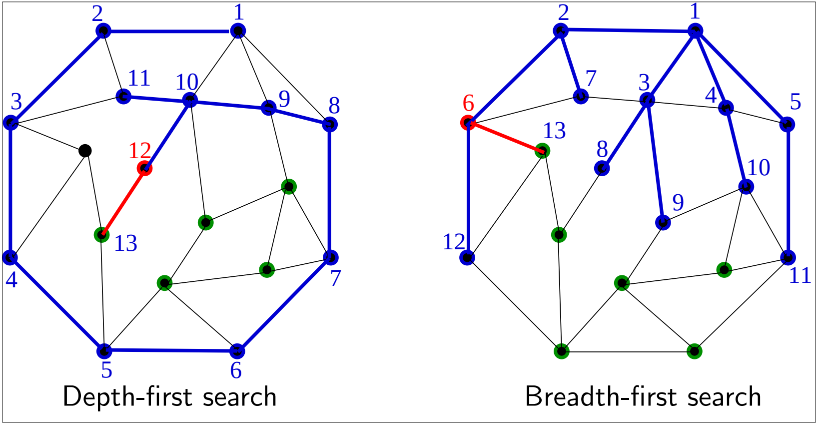


Depth-first search and breadth-first search



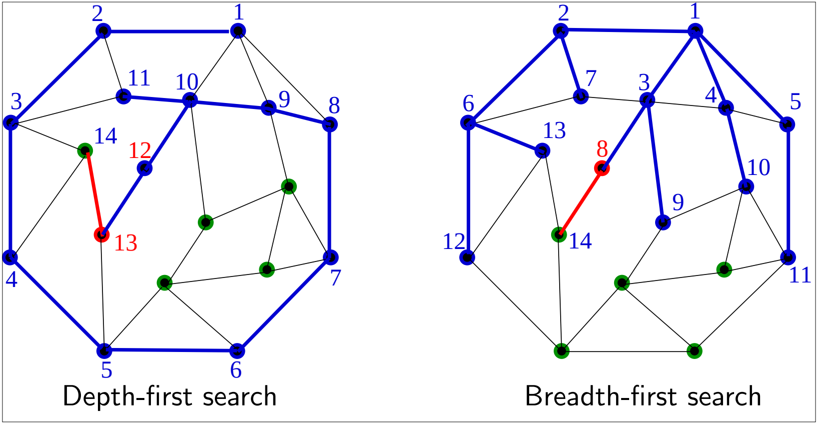
Depth-first search and breadth-first search

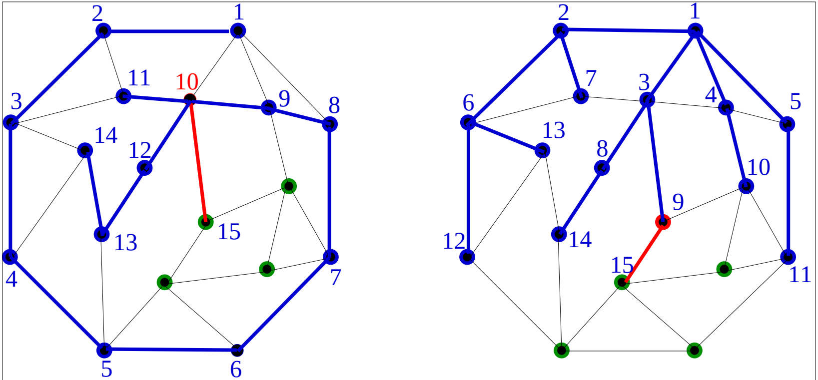




2
1
11
10
9
8
3
12
13
4
7
5
6
Depth-first search
2
1
7
3
4
5
6
13
8
10
9
12
11
Breadth-first search

Depth-first search and breadth-first search

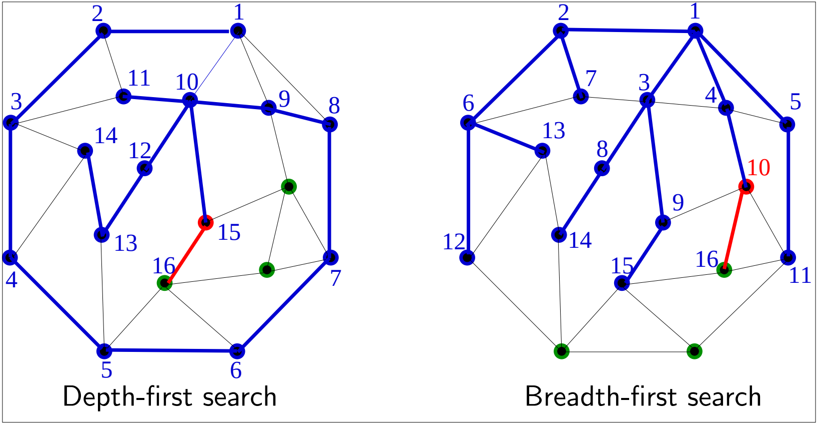




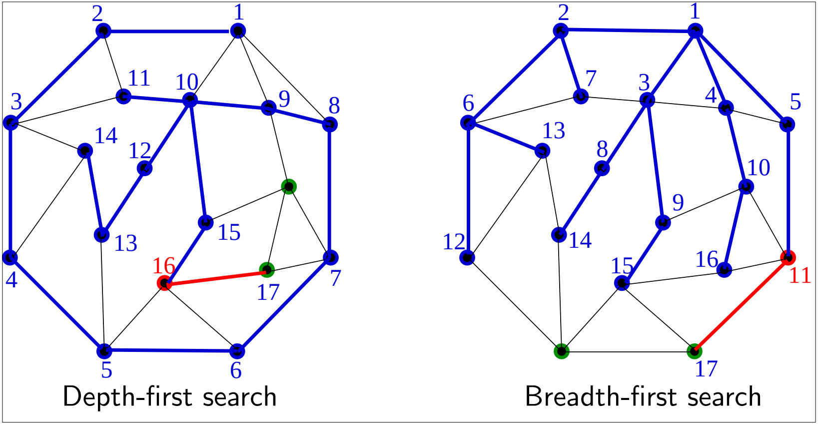
Depth-first search

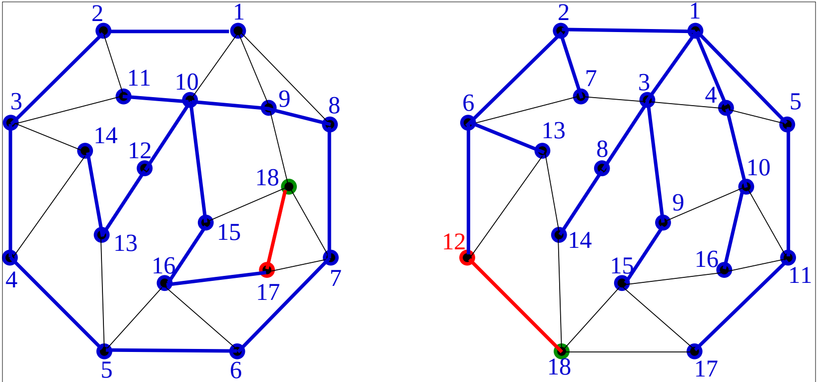
Breadth-first search

Depth-first search and breadth-first search



Depth-first search and breadth-first search





Depth-first search Breadth-first search

Minimum cost and Maximum cos spanning tree

Definition

A spanning tree *of a connected graph* $G = (V, H)$ *is such spanning subgraph of* G *that is a tree.*

Let $G = (V, H, c)$ *be an edge weighted graph,* K *a spanning tree of* G*.*
The cost $c(K)$ **of spanning tree** K *is the sum of edge weights of all edges of* K*.*

The minimum cost spanning tree *of graph* G *is the spanning tree of* G *having the minimum cost.*

The maximum cost spanning tree *of graph* G *is the spanning tree of* G *having the maximum cost.*

Minimum cost and Maximum cos spanning tree

Definition

A spanning tree *of a connected graph* $G = (V, H)$ *is such spanning subgraph of* G *that is a tree.*

Let $G = (V, H, c)$ *be an edge weighted graph,* K *a spanning tree of* G*.* **The cost** $c(K)$ **of spanning tree** K *is the sum of edge weights of all edges of* K*.*

The minimum cost spanning tree *of graph* G *is the spanning tree of* G *having the minimum cost.*

The maximum cost spanning tree *of graph* G *is the spanning tree of* G *having the maximum cost.*

Algorithm

Kruskal's algorithm I. *to find minimum (maximum) cost spanning tree of an edge weighted graph $G = (V, H, c)$.*

- **Step 1.** *Let $K = (V, E)$ be a spanning subgraph of G with $E = \emptyset$. Arrange all edges of H in their increasing (decreasing) order of weight into sequence $\mathcal{P}$.*
- **Step 2.** *Let $\{u, v\}$ be the first edge in sequence $\mathcal{P}$. Exclude the edge $\{u, v\}$ from the sequence $\mathcal{P}$. If the edge $\{u, v\}$ does not create a cycle with till now chosen edges of the set E then insert the edge $\{u, v\}$ into E, i.e. set $E = E \cup \{\{u, v\}\}$.*
- **Step 3.** *If the number of chosen edges is equal to $|V| - 1$ or if the sequence $\mathcal{P}$ is empty, then STOP. Otherwise GOTO Step 2.* ♣

Algorithm

Kruskal's algorithm I. *to find minimum (maximum) cost spanning tree of an edge weighted graph* $G = (V, H, c)$.

- **Step 1.** *Let* $K = (V, E)$ *be a spanning subgraph of* G *with* $E = \emptyset$. *Arrange all edges of* H *in their increasing (decreasing) order of weight into sequence* $\mathcal{P}$.
- **Step 2.** *Let* $\{u, v\}$ *be the first edge in sequence* $\mathcal{P}$. *Exclude the edge* $\{u, v\}$ *from the sequence* $\mathcal{P}$. *If the edge* $\{u, v\}$ *does not create a cycle with till now chosen edges of the set* E *then insert the edge* $\{u, v\}$ *into* E, *i.e. set* $E = E \cup \{\{u, v\}\}$.
- **Step 3.** *If the number of chosen edges is equal to* $|V| - 1$ *or if the sequence* $\mathcal{P}$ *is empty, then STOP. Otherwise GOTO Step 2.*

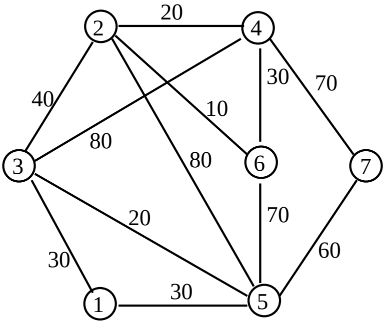
♣

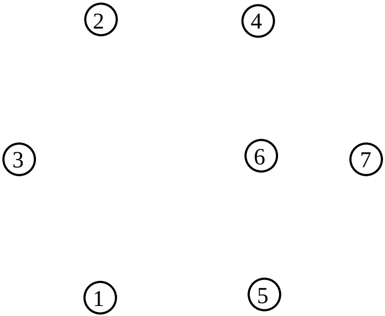
Algorithm

Kruskal's algorithm I. *to find minimum (maximum) cost spanning tree of an edge weighted graph* $G = (V, H, c)$.

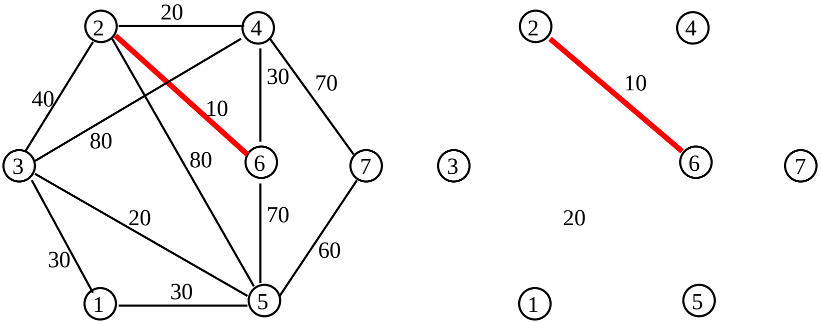
- **Step 1.** *Let* $K = (V, E)$ *be a spanning subgraph of* G *with* $E = \emptyset$. *Arrange all edges of* H *in their increasing (decreasing) order of weight into sequence* $\mathcal{P}$.
- **Step 2.** *Let* $\{u, v\}$ *be the first edge in sequence* $\mathcal{P}$. *Exclude the edge* $\{u, v\}$ *from the sequence* $\mathcal{P}$. *If the edge* $\{u, v\}$ *does not create a cycle with till now chosen edges of the set* E *then insert the edge* $\{u, v\}$ *into* E, *i.e. set* $E = E \cup \{\{u, v\}\}$.
- **Step 3.** *If the number of chosen edges is equal to* $|V| - 1$ *or if the sequence* $\mathcal{P}$ *is empty, then STOP. Otherwise GOTO Step 2.* ♣

Example

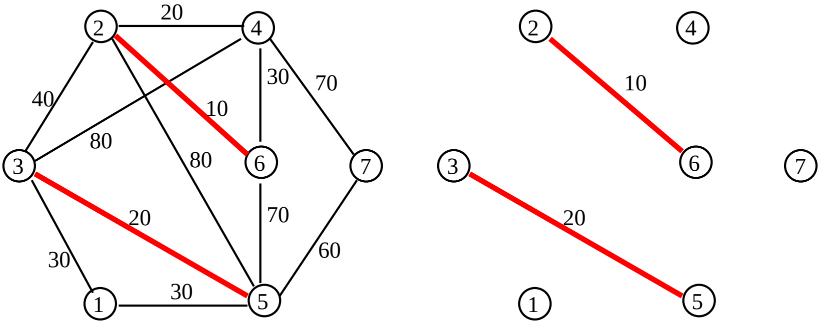




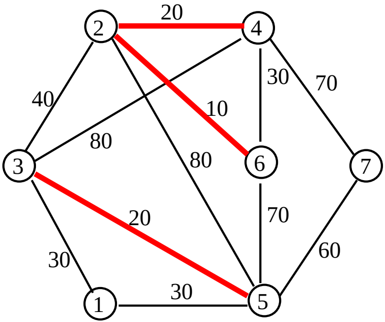
Example

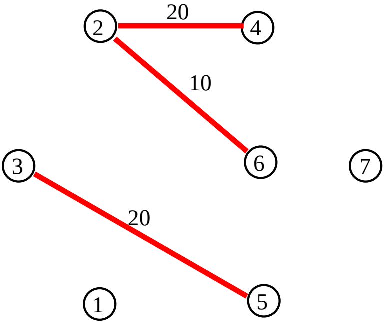


Example

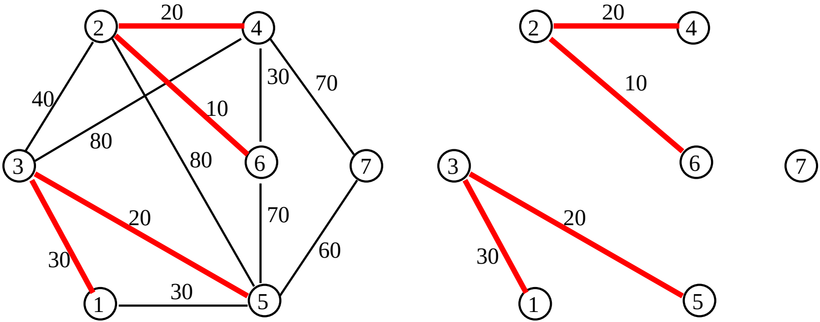


Example

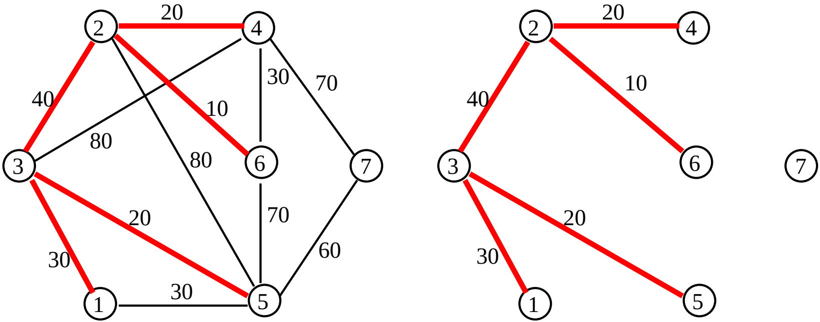




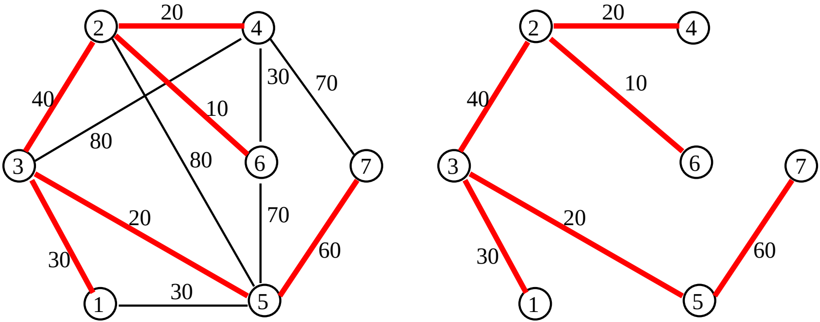
Example



Example



Example


2
4
20
30
70
40
10
80
80
3
6
7
20
70
30
60
1
30
5
2
20
4
40
10
3
6
7
20
30
60
1
5

Kruskal's spanning-tree algorithm II.

Algorithm

Kruskal's algorithm II. *to find minimum (maximum) cost spanning tree of an edge weighted graph $G = (V, H, c)$.*

- **Step 1.** *Let $K = (V, E)$ be a spanning subgraph of G with $E = \emptyset$. Arrange all edges of H in their increasing (decreasing) order of weight into sequence $\mathcal{P}$.*
- **Step 2.** *Set label $k(i) = i$ for every vertex $i \in V$.*
- **Step 3.** *Let $\{u, v\}$ be the first edge in sequence $\mathcal{P}$. Exclude the edge $\{u, v\}$ from the sequence $\mathcal{P}$. If $k(u) \neq k(v)$ then insert the edge $\{u, v\}$ into E, i.e. set $E = E \cup \{\{u, v\}\}$. and $\forall i \in V$ such that $k(i) = k(v)$ set $k(i) := k(u)$*
- **Step 4.** *If the number of chosen edges is equal to $|V| - 1$ or if the sequence $\mathcal{P}$ is empty, then STOP. Otherwise GOTO Step 3.*

♣

Algorithm

Kruskal's algorithm II. *to find minimum (maximum) cost spanning tree of an edge weighted graph* $G = (V, H, c)$.

- **Step 1.** *Let* $K = (V, E)$ *be a spanning subgraph of* G *with* $E = \emptyset$. *Arrange all edges of* H *in their increasing (decreasing) order of weight into sequence* $\mathcal{P}$.
- **Step 2.** *Set label* $k(i) = i$ *for every vertex* $i \in V$.
- **Step 3.** *Let* $\{u, v\}$ *be the first edge in sequence* $\mathcal{P}$. *Exclude the edge* $\{u, v\}$ *from the sequence* $\mathcal{P}$. *If* $k(u) \neq k(v)$ *then insert the edge* $\{u, v\}$ *into* E, *i.e. set* $E = E \cup \{\{u, v\}\}$. *and* $\forall i \in V$ *such that* $k(i) = k(v)$ *set* $k(i) := k(u)$
- **Step 4.** *If the number of chosen edges is equal to* $|V| - 1$ *or if the sequence* $\mathcal{P}$ *is empty, then STOP. Otherwise GOTO Step 3.*

♣

Algorithm

Kruskal's algorithm II. *to find minimum (maximum) cost spanning tree of an edge weighted graph $G = (V, H, c)$.*

- **Step 1.** *Let $K = (V, E)$ be a spanning subgraph of G with $E = \emptyset$. Arrange all edges of H in their increasing (decreasing) order of weight into sequence $\mathcal{P}$.*
- **Step 2.** *Set label $k(i) = i$ for every vertex $i \in V$.*
- **Step 3.** *Let $\{u, v\}$ be the first edge in sequence $\mathcal{P}$. Exclude the edge $\{u, v\}$ from the sequence $\mathcal{P}$. If $k(u) \neq k(v)$ then insert the edge $\{u, v\}$ into E, i.e. set $E = E \cup \{\{u, v\}\}$. and $\forall i \in V$ such that $k(i) = k(v)$ set $k(i) := k(u)$*
- **Step 4.** *If the number of chosen edges is equal to $|V| - 1$ or if the sequence $\mathcal{P}$ is empty, then STOP. Otherwise GOTO Step 3.*

Algorithm

Kruskal's algorithm II. *to find minimum (maximum) cost spanning tree of an edge weighted graph $G = (V, H, c)$.*

- **Step 1.** *Let $K = (V, E)$ be a spanning subgraph of G with $E = \emptyset$. Arrange all edges of H in their increasing (decreasing) order of weight into sequence $\mathcal{P}$.*
- **Step 2.** *Set label $k(i) = i$ for every vertex $i \in V$.*
- **Step 3.** *Let $\{u, v\}$ be the first edge in sequence $\mathcal{P}$. Exclude the edge $\{u, v\}$ from the sequence $\mathcal{P}$. If $k(u) \neq k(v)$ then insert the edge $\{u, v\}$ into E, i.e. set $E = E \cup \{\{u, v\}\}$. and $\forall i \in V$ such that $k(i) = k(v)$ set $k(i) := k(u)$*
- **Step 4.** *If the number of chosen edges is equal to $|V| - 1$ or if the sequence $\mathcal{P}$ is empty, then STOP. Otherwise GOTO Step 3.*

♣

Kruskal's spanning-tree algorithm II. Example

Sequence $\mathcal{P}$ containing all edges of H in their increasing order of weight:

{2,6}	{2,4}	{3,5}	{1,3}	{1,5}	{4,6}	{2,3}	{5,7}	{4,7}	{5,6}	{2,5}	{3,4}
10	20	20	30	30	30	40	60	70	70	80	80

Edge into spanning tree	1	2	3	4	5	6	7
				$k(v)$			
-	1	2	3	4	5	6	7
{2,6}	1	2	3	4	5	2	7
{2,4}	1	2	3	2	5	2	7
{3,5}	1	2	3	2	3	2	7
{1,3}	1	2	1	2	1	2	7
{2,3}	1	1	1	1	1	1	7
{5,7}	1	1	1	1	1	1	1

Kruskal's spanning-tree algorithm II. Example

Sequence $\mathcal{P}$ containing all edges of H in their increasing order of weight:

{2,6}	{2,4}	{3,5}	{1,3}	{1,5}	{4,6}	{2,3}	{5,7}	{4,7}	{5,6}	{2,5}	{3,4}
10	20	20	30	30	30	40	60	70	70	80	80

Edge $\{u, v\} = \{2, 6\}$
$k(2) = 2, \; k(6) = 6$

$k(2) \neq k(6) \Rightarrow$
insert $\{2, 6\}$ into spanning tree

Edge into spanning tree	1	2	3	4	5	6	7
				$k(v)$			
-	1	2	3	4	5	6	7
{2,6}	1	2	3	4	5	2	7
{2,4}	1	2	3	2	5	2	7
{3,5}	1	2	3	2	3	2	7
{1,3}	1	2	1	2	1	2	7
{2,3}	1	1	1	1	1	1	7
{5,7}	1	1	1	1	1	1	1

Kruskal's spanning-tree algorithm II. Example

Sequence $\mathcal{P}$ containing all edges of H in their increasing order of weight:

{2,6}	{2,4}	{3,5}	{1,3}	{1,5}	{4,6}	{2,3}	{5,7}	{4,7}	{5,6}	{2,5}	{3,4}
10	20	20	30	30	30	40	60	70	70	80	80

Edge $\{u, v\} = \{2, 4\}$
$k(2) = 2,\ k(4) = 4$

$k(2) \neq k(4) \Rightarrow$
insert $\{2, 4\}$ into
spanning tree

Edge into spanning tree	1	2	3	4	5	6	7
				$k(v)$			
-	1	2	3	4	5	6	7
{2,6}	1	2	3	4	5	2	7
{2,4}	1	2	3	2	5	2	7
{3,5}	1	2	3	2	3	2	7
{1,3}	1	2	1	2	1	2	7
{2,3}	1	1	1	1	1	1	7
{5,7}	1	1	1	1	1	1	1

Kruskal's spanning-tree algorithm II. Example

Sequence $\mathcal{P}$ containing all edges of H in their increasing order of weight:

{2,6}	{2,4}	{3,5}	{1,3}	{1,5}	{4,6}	{2,3}	{5,7}	{4,7}	{5,6}	{2,5}	{3,4}
10	20	20	30	30	30	40	60	70	70	80	80

Edge $\{u, v\} = \{3, 5\}$
$k(3) = 3,\ k(5) = 5$

$k(3) \neq k(5) \Rightarrow$
insert $\{3, 5\}$ into spanning tree

Edge into spanning tree	1	2	3	4	5	6	7
				$k(v)$			
-	1	2	3	4	5	6	7
{2,6}	1	2	3	4	5	2	7
{2,4}	1	2	3	2	5	2	7
{3,5}	1	2	3	2	3	2	7
{1,3}	1	2	1	2	1	2	7
{2,3}	1	1	1	1	1	1	7
{5,7}	1	1	1	1	1	1	1

Kruskal's spanning-tree algorithm II. Example

Sequence $\mathcal{P}$ containing all edges of H in their increasing order of weight:

{2,6}	{2,4}	{3,5}	{1,3}	{1,5}	{4,6}	{2,3}	{5,7}	{4,7}	{5,6}	{2,5}	{3,4}
10	20	20	30	30	30	40	60	70	70	80	80

Edge $\{u, v\} = \{1, 3\}$
$k(1) = 1,\ k(3) = 3$

$k(1) \neq k(3) \Rightarrow$
insert $\{1, 3\}$ into
spanning tree

Edge into spanning tree	1	2	3	4	5	6	7
				$k(v)$			
-	1	2	3	4	5	6	7
{2,6}	1	2	3	4	5	2	7
{2,4}	1	2	3	2	5	2	7
{3,5}	1	2	3	2	3	2	7
{1,3}	1	2	1	2	1	2	7
{2,3}	1	1	1	1	1	1	7
{5,7}	1	1	1	1	1	1	1

Kruskal's spanning-tree algorithm II. Example

Sequence $\mathcal{P}$ containing all edges of H in their increasing order of weight:

{2,6}	{2,4}	{3,5}	{1,3}	{1,5}	{4,6}	{2,3}	{5,7}	{4,7}	{5,6}	{2,5}	{3,4}
10	20	20	30	30	30	40	60	70	70	80	80

Edge $\{u, v\} = \{1, 5\}$
$k(1) = 1,\ k(5) = 1$

$k(1) = k(5) \Rightarrow$
throw away $\{1, 5\}$

Edge into spanning tree	1	2	3	4	5	6	7
				$k(v)$			
-	1	2	3	4	5	6	7
{2,6}	1	2	3	4	5	2	7
{2,4}	1	2	3	2	5	2	7
{3,5}	1	2	3	2	3	2	7
{1,3}	1	2	1	2	1	2	7
{2,3}	1	1	1	1	1	1	7
{5,7}	1	1	1	1	1	1	1

Kruskal's spanning-tree algorithm II. Example

Sequence $\mathcal{P}$ containing all edges of H in their increasing order of weight:

{2,6}	{2,4}	{3,5}	{1,3}	{1,5}	{4,6}	{2,3}	{5,7}	{4,7}	{5,6}	{2,5}	{3,4}
10	20	20	30	30	30	40	60	70	70	80	80

Edge $\{u, v\} = \{4, 6\}$
$k(4) = 2,\ k(6) = 2$

$k(4) = k(6) \Rightarrow$
throw away $\{4, 6\}$

Edge into spanning tree	1	2	3	4	5	6	7
				$k(v)$			
-	1	2	3	4	5	6	7
{2,6}	1	2	3	4	5	2	7
{2,4}	1	2	3	2	5	2	7
{3,5}	1	2	3	2	3	2	7
{1,3}	1	2	1	2	1	2	7
{2,3}	1	1	1	1	1	1	7
{5,7}	1	1	1	1	1	1	1

Kruskal's spanning-tree algorithm II. Example

Sequence $\mathcal{P}$ containing all edges of H in their increasing order of weight:

{2,6}	{2,4}	{3,5}	{1,3}	{1,5}	{4,6}	{2,3}	{5,7}	{4,7}	{5,6}	{2,5}	{3,4}
10	20	20	30	30	30	40	60	70	70	80	80

Edge $\{u, v\} = \{2, 3\}$
$k(2) = 2,\ k(3) = 1$

$k(2) \neq k(3) \Rightarrow$
insert $\{2, 3\}$ into spanning tree

Edge into spanning tree	1	2	3	4	5	6	7
				$k(v)$			
-	1	2	3	4	5	6	7
{2,6}	1	2	3	4	5	2	7
{2,4}	1	2	3	2	5	2	7
{3,5}	1	2	3	2	3	2	7
{1,3}	1	2	1	2	1	2	7
{2,3}	1	1	1	1	1	1	7
{5,7}	1	1	1	1	1	1	1

Kruskal's spanning-tree algorithm II. Example

Sequence $\mathcal{P}$ containing all edges of H in their increasing order of weight:

{2,6}	{2,4}	{3,5}	{1,3}	{1,5}	{4,6}	{2,3}	{5,7}	{4,7}	{5,6}	{2,5}	{3,4}
10	20	20	30	30	30	40	60	70	70	80	80

Edge $\{u, v\} = \{5, 7\}$
$k(5) = 1,\ k(7) = 7$

$k(5) \neq k(7) \Rightarrow$
insert $\{5, 7\}$ into spanning tree

Edge into spanning tree	1	2	3	4	5	6	7
				$k(v)$			
-	1	2	3	4	5	6	7
{2,6}	1	2	3	4	5	2	7
{2,4}	1	2	3	2	5	2	7
{3,5}	1	2	3	2	3	2	7
{1,3}	1	2	1	2	1	2	7
{2,3}	1	1	1	1	1	1	7
{5,7}	1	1	1	1	1	1	1

Maximum capacity path problem

Definition

Let $G = (V, H, c)$ be a ege weighted graph where edge cost $c(h) > 0$ of an edge $h \in H$ means the capacity of the edge h.

Capacity $c(\mu(u, v))$ **of** u–v **path (walk, trail, etc.)** $\mu(u, v)$ *is defined as*

$$c(\mu(u, v)) = \min\{c(h) \mid h \in \mu(u, v)\}.$$

Definition

We will say that u–v *path* $\mu(u, v)$ *in graph* $G = (V, H, c)$ *is* **maximum capacity** u–v **path** *if the path* $\mu(u, v)$ *has largest capacitu of all* u–v *paths in* G.

Remark

The maximum capacity path problem is also known as the bottleneck shortest path problem or the widest path problem.

Maximum capacity path problem

Definition

Let $G = (V, H, c)$ *be a ege weighted graph where edge cost* $c(h) > 0$ *of an edge* $h \in H$ *means the capacity of the edge* h*.*

Capacity $c(\boldsymbol{\mu}(u,v))$ **of** u–v **path (walk, trail, etc.)** $\boldsymbol{\mu}(u,v)$ *is defined as*

$$c(\boldsymbol{\mu}(u,v)) = \min\{c(h) \mid h \in \boldsymbol{\mu}(u,v)\}.$$

Definition

We will say that u–v *path* $\boldsymbol{\mu}(u,v)$ *in graph* $G = (V, H, c)$ *is* **maximum capacity** u–v **path** *if the path* $\boldsymbol{\mu}(u,v)$ *has largest capacitu of all* u–v *paths in* G*.*

Remark

The maximum capacity path problem is also known as the bottleneck shortest path problem or the widest path problem.

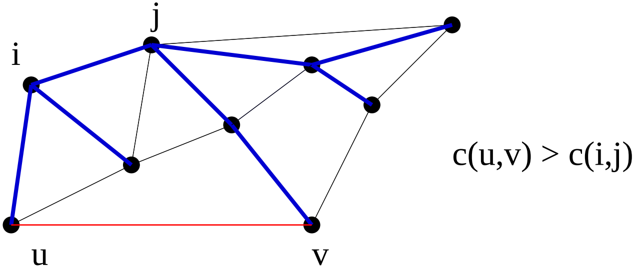
Maximum capacity path problem

Theorem

Let K be a maximum capacity spanning tree in a connected edge weighted graph $G = (V, H, c)$, let $\{u, v\} \in H$ be such an edge of graph G which is not an element of edge set of K.
Let $\boldsymbol{\mu}(u, v)$ be a (unique) u-v path in spanning tree K.
Then the capacity of the path $\boldsymbol{\mu}(u, v)$ is greater or equal to the capacity of edge $\{u, v\}$, i. e.

$$c(\boldsymbol{\mu}(u, v)) \geq c(u, v).$$

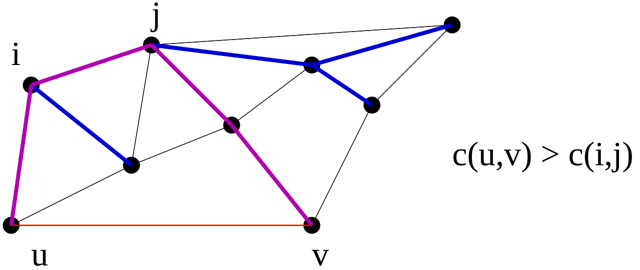
PROOF.
Let us have a maximum cost spanning tree $\mathcal{K}$ and let there exists an edge $\{u, v\}$ such that capacity of u-v path along edges of spanning tree is less than $c(u, v)$).



Spanning tree $\mathcal{K}$ blue, edge $h = \{u, v\}$ (red)
u-v path along edges of spanning tree (violet) with less capacity than $c(u, v)$
Then there exists and edge $\{i, j\}$ of this path such that $c(u, v) > c(i, j)$
By replacing of edge $\{i, j\}$ by edge $\{u, v\}$ we get a spanning tree with greater cost – contradiction with assupmtion that $\mathcal{K}$ was a maximum cost spanning tree.

PROOF.

Let us have a maximum cost spanning tree $\mathcal{K}$ and let there exists an edge $\{u, v\}$ such that capacity of u-v path along edges of spanning tree is less than $c(u, v)$).



Spanning tree $\mathcal{K}$ blue, edge $h = \{u, v\}$ (red)

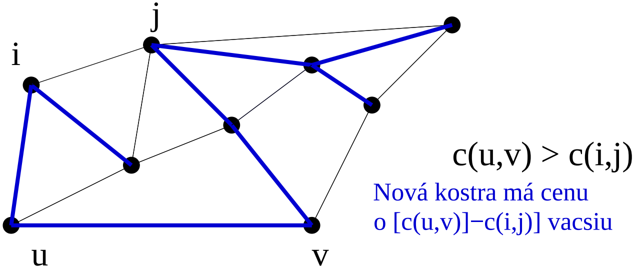
u-v path along edges of spanning tree (violet) with less capacity than $c(u, v)$

Then there exists and edge $\{i, j\}$ of this path such that $c(u, v) > c(i, j)$

By replacing of edge $\{i, j\}$ by edge $\{u, v\}$ we get a spanning tree with greater cost – contradiction with assupmtion that $\mathcal{K}$ was a maximum cost spanning tree.

PROOF.

Let us have a maximum cost spanning tree $\mathcal{K}$ and let there exists an edge $\{u, v\}$ such that capacity of u-v path along edges of spanning tree is less than $c(u, v)$).



Spanning tree $\mathcal{K}$ blue, edge $h = \{u, v\}$ (red)

u-v path along edges of spanning tree (violet) with less capacity than $c(u, v)$

Then there exists and edge $\{i, j\}$ of this path such that $c(u, v) > c(i, j)$

By replacing of edge $\{i, j\}$ by edge $\{u, v\}$ we get a spanning tree with greater cost – contradiction with assupmtion that $\mathcal{K}$ was a maximum cost spanning tree.

Maximum capacity path problem

Theorem

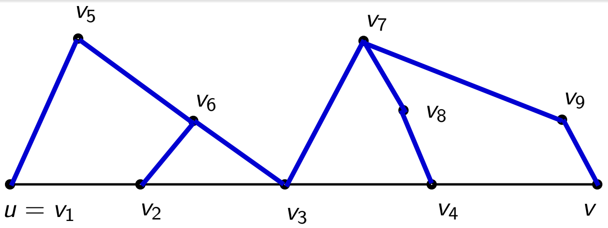
Let K be a maximum cost spanning tree in a connected edge weighted graph $G = (V, H, c)$. Then for all $u, v \in V$ the (unique) u–v path in K is a maximum capacity u–v path in G.

PROOF.

Theorem

Let K be a maximum cost spanning tree in a connected edge weighted graph $G = (V, H, c)$. Then for all $u, v \in V$ the (unique) u–v path in K is a maximum capacity u–v path in G.

PROOF.



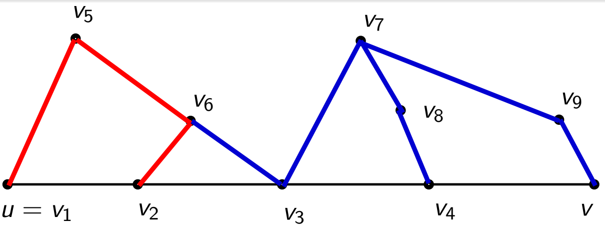
Maximum capacity path:

$$\boldsymbol{\mu}(u, v) = (u, \{u \equiv v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_4\}, v_4, \{v_4, v\}, v),$$

Theorem

Let K be a maximum cost spanning tree in a connected edge weighted graph $G = (V, H, c)$. Then for all $u, v \in V$ the (unique) u–v path in K is a maximum capacity u–v path in G.

PROOF.

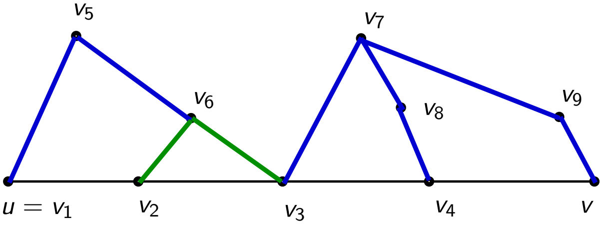


$$\mu(u, v_2) = (u, \{u, v_5\}, v_5, \{v_5, v_6\}, v_6, \{v_6, v_2\}, v_2),$$

Theorem

Let K be a maximum cost spanning tree in a connected edge weighted graph $G = (V, H, c)$. Then for all $u, v \in V$ the (unique) u–v path in K is a maximum capacity u–v path in G.

PROOF.



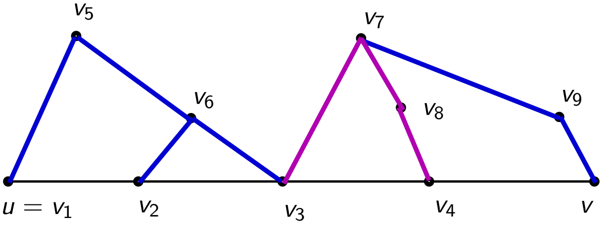
$$\mu(v_2, v_3) = (v_2, \{v_2, v_6\}, v_6, \{v_6, v_3\}, v_3),$$

Maximum capacity path problem

Theorem

Let K be a maximum cost spanning tree in a connected edge weighted graph $G = (V, H, c)$. Then for all $u, v \in V$ the (unique) u–v path in K is a maximum capacity u–v path in G.

PROOF.

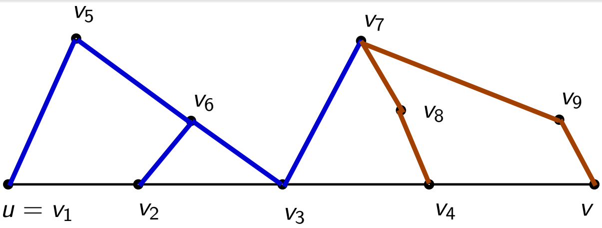


$\mu(v_3, v_4) = (v_3, \{v_3, v_7\}, v_7, \{v_7, v_8\}, v_8, \{v_8, v_4\}, v_4),$

Theorem

Let K be a maximum cost spanning tree in a connected edge weighted graph $G = (V, H, c)$. Then for all $u, v \in V$ the (unique) u–v path in K is a maximum capacity u–v path in G.

PROOF.

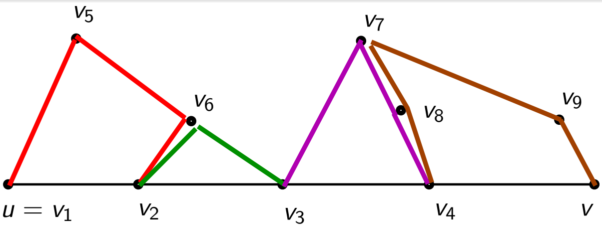


$$\mu(v_4, v) = (v_4, \{v_4, v_8\}, v_8, \{v_8, v_7\}, v_7, \{v_7, v_9\}, v_9, \{v_9, v\}, v).$$

Theorem

Let K be a maximum cost spanning tree in a connected edge weighted graph $G = (V, H, c)$. Then for all $u, v \in V$ the (unique) u–v path in K is a maximum capacity u–v path in G.

PROOF.



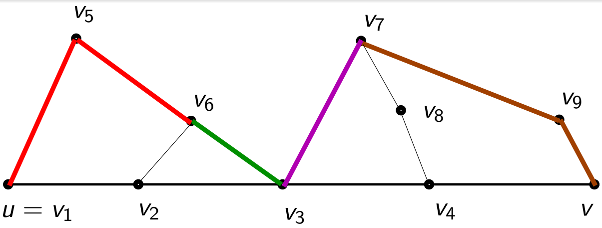
u-v sled po hranách kostry s priepustnosťou $\geq$ než priepustnnosť cesty $\mu(u, v)$

Maximum capacity path problem

Theorem

Let K be a maximum cost spanning tree in a connected edge weighted graph $G = (V, H, c)$. Then for all $u, v \in V$ the (unique) u–v path in K is a maximum capacity u–v path in G.

PROOF.



Maximum capacity path:

$$\mu(u, v) = (u, \{u \equiv v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_4\}, v_4, \{v_4, v\}, v),$$

Maximum capacity path along edges of spanning tree

$u, \{u, v_5\}, v_5, \{v_5, v_6\}, v_6, \{v_6, v_3\}, v_3, \{v_3, v_7\}, v_7, \{v_7, v_9\}, v_9, \{v_9, v\}, v.$

Algorithm

Maximum capacity u–v path algorithm in a connected edge weighted graph $G = (V, H, c)$.

- **Step 1.** *Create a maximum cost spanning tree K in graph G.*
- **Step 2.** *Find unique u–v path in spanning tree K.*
  This u–v path along edges of K is a maximum capacity u–v path in graph G.

♣

Remark

Last algorithm wil find a maximum capacity u–v path, but this path is not in many cases optimal from the point of view of traveled distance. In the case that we are looking for maximum capacity shortest u–v path we need be given in corresponding graph (together with capacity) additional edge cost representing the length of edges.

Maximum capacity u–v path algorithm

Algorithm

Maximum capacity u–v path algorithm in a connected edge weighted graph $G = (V, H, c)$.

- **Step 1.** *Create a maximum cost spanning tree K in graph G.*
- **Step 2.** *Find unique u–v path in spanning tree K.*

  This u–v path along edges of K is a maximum capacity u–v path in graph G.

♣

Remark

Last algorithm wil find a maximum capacity u–v path, but this path is not in many cases optimal from the point of view of traveled distance. In the case that we are looking for maximum capacity shortest u–v path we need be given in corresponding graph (together with capacity) additional edge cost representing the length of edges.

Maximum capacity u–v path algorithm

Algorithm

Maximum capacity u–v path algorithm in a connected edge weighted graph $G = (V, H, c)$.

- **Step 1.** *Create a maximum cost spanning tree K in graph G.*
- **Step 2.** *Find unique u–v path in spanning tree K.*

  This u–v path along edges of K is a maximum capacity u–v path in graph G.

♣

Remark

Last algorithm wil find a maximum capacity u–v path, but this path is not in many cases optimal from the point of view of traveled distance. In the case that we are looking for maximum capacity shortest u–v path we need be given in corresponding graph (together with capacity) additional edge cost representing the length of edges.

Algorithm

Maximum capacity u–v shortest path algorithm in a connected edge weighted graph $G = (V, H, c, d)$, where $c(h)$ je the capacity and $d(h)$ is the length of edge $h \in H$.

- **Step 1.** *Create a maximum cost spanning tree K in graph G subject to edge cost $c(\ )$.*

  Find unique u–v path in spanning tree K.

  Let C be the capacity of $\boldsymbol{\mu}(u, v)$.

- **Step 2.** *Create a new graph $G' = (V, H', d)$, where $H' = \{h | h \in H,\ c(h) \geq C\}$.*
  $\{$edge set H' contains only those edges of original graph with capacity greater or equal to C.$\}$

- **Step 3.** *Find the shortest u–v path in G' with respect to edge cost d.*

Algorithm

Maximum capacity u–v shortest path algorithm in a connected edge weighted graph $G = (V, H, c, d)$, where $c(h)$ je the capacity and $d(h)$ is the length of edge $h \in H$.

- **Step 1.** *Create a maximum cost spanning tree K in graph G subject to edge cost $c(\ )$.*

  Find unique u–v path in spanning tree K.

  Let C be the capacity of $\boldsymbol{\mu}(u, v)$.

- **Step 2.** *Create a new graph $G' = (V, H', d)$, where $H' = \{h | h \in H,\ c(h) \geq C\}$.*
  $\{$edge set H' contains only those edges of original graph with capacity greater or equal to C.$\}$

- **Step 3.** *Find the shortest u–v path in G' with respect to edge cost d.*

♣

Algorithm

Maximum capacity u–v shortest path algorithm in a connected edge weighted graph $G = (V, H, c, d)$, where $c(h)$ je the capacity and $d(h)$ is the length of edge $h \in H$.

- **Step 1.** *Create a maximum cost spanning tree K in graph G subject to edge cost $c(\ )$.*

  Find unique u–v path in spanning tree K.

  Let C be the capacity of $\boldsymbol{\mu}(u, v)$.

- **Step 2.** *Create a new graph $G' = (V, H', d)$, where $H' = \{h | h \in H,\ c(h) \geq C\}$.*
  {edge set H' contains only those edges of original graph with capacity greater or equal to C.}

- **Step 3.** *Find the shortest u–v path in G' with respect to edge cost d.*

♣