Traversing Graphs

Stepan Kuznetsov

Computer Science Department, Higher School of Economics

Outline

Traversing Trees

Traversing Graphs: DFS and BFS

Traversing Graphs

- *Traversing* a graph means *visiting* its vertices in a specific order.

Traversing Graphs

- *Traversing* a graph means *visiting* its vertices in a specific order.
- Traversing is used as a subprogram in many graph-theoretic algorithms.

Traversing Graphs

- *Traversing* a graph means *visiting* its vertices in a specific order.
- Traversing is used as a subprogram in many graph-theoretic algorithms.
- We start with an easier case of traversing trees, starting from the root node.

Recursive Traversing of Trees

- Trees are usually traversed *recursively*.

Recursive Traversing of Trees

- Trees are usually traversed *recursively*.
- If we cut off the root, the tree splits into several subtrees, each with its own root.

Recursive Traversing of Trees

- Trees are usually traversed *recursively*.
- If we cut off the root, the tree splits into several subtrees, each with its own root.
- These new roots are children of the old one.

Recursive Traversing of Trees

- Trees are usually traversed *recursively*.
- If we cut off the root, the tree splits into several subtrees, each with its own root.
- These new roots are children of the old one.
- We run our traversing function recursively for each of these new trees.

Recursive Traversing of Trees

- Trees are usually traversed *recursively*.
- If we cut off the root, the tree splits into several subtrees, each with its own root.
- These new roots are children of the old one.
- We run our traversing function recursively for each of these new trees.
- This is the *depth-first* traversing algorithm for trees.

Recursive Traversing of Trees

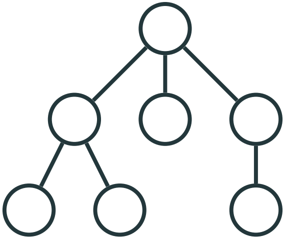
- Trees are usually traversed *recursively*.
- If we cut off the root, the tree splits into several subtrees, each with its own root.
- These new roots are children of the old one.
- We run our traversing function recursively for each of these new trees.
- This is the *depth-first* traversing algorithm for trees.
- The only question is when to visit the root.

Pre-order Traversing

In the *pre-order,* we visit the root **before** traversing the sub-trees.

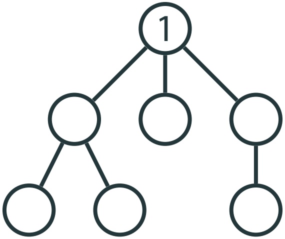
Pre-order Traversing

In the *pre-order,* we visit the root **before** traversing the sub-trees.



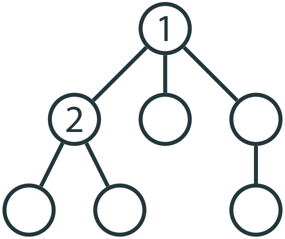
Pre-order Traversing

In the *pre-order,* we visit the root **before** traversing the sub-trees.



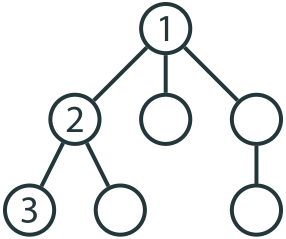
Pre-order Traversing

In the *pre-order,* we visit the root **before** traversing the sub-trees.



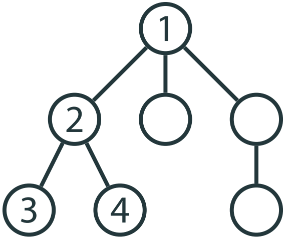
Pre-order Traversing

In the *pre-order*, we visit the root **before** traversing the sub-trees.



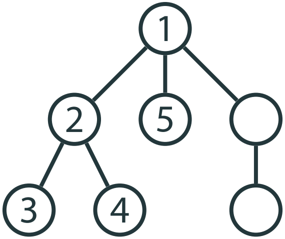
Pre-order Traversing

In the *pre-order*, we visit the root **before** traversing the sub-trees.



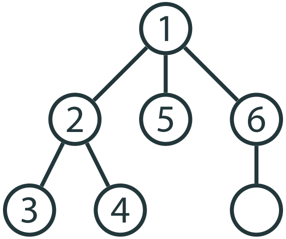
Pre-order Traversing

In the *pre-order*, we visit the root **before** traversing the sub-trees.



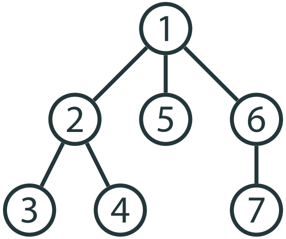
Pre-order Traversing

In the *pre-order*, we visit the root **before** traversing the sub-trees.



Pre-order Traversing

In the *pre-order,* we visit the root **before** traversing the sub-trees.

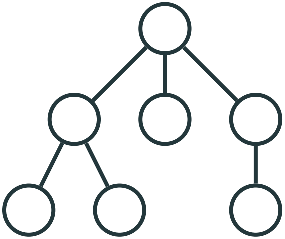


Post-order Traversing

In the *post-order,* we visit the root **after** traversing the sub-trees.

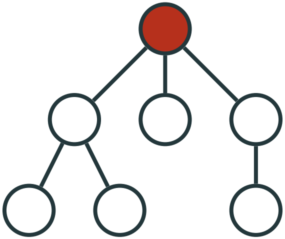
Post-order Traversing

In the *post-order,* we visit the root **after** traversing the sub-trees.



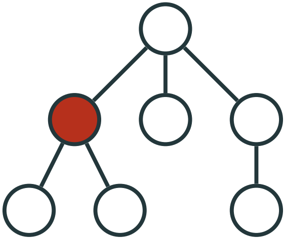
Post-order Traversing

In the *post-order,* we visit the root **after** traversing the sub-trees.



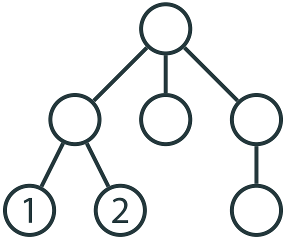
Post-order Traversing

In the *post-order,* we visit the root **after** traversing the sub-trees.



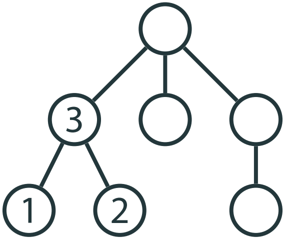
Post-order Traversing

In the *post-order,* we visit the root **after** traversing the sub-trees.



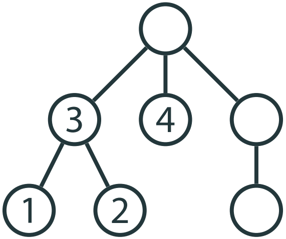
Post-order Traversing

In the *post-order,* we visit the root **after** traversing the sub-trees.



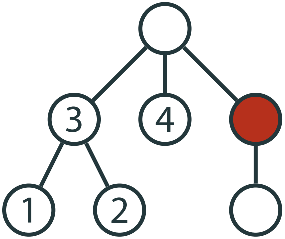
Post-order Traversing

In the *post-order,* we visit the root **after** traversing the sub-trees.



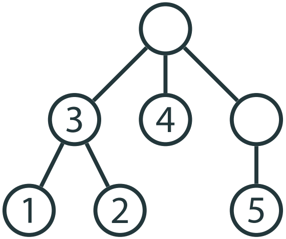
Post-order Traversing

In the *post-order,* we visit the root **after** traversing the sub-trees.



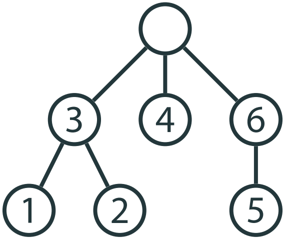
Post-order Traversing

In the *post-order,* we visit the root **after** traversing the sub-trees.



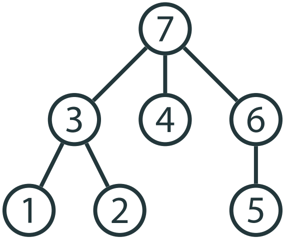
Post-order Traversing

In the *post-order,* we visit the root **after** traversing the sub-trees.



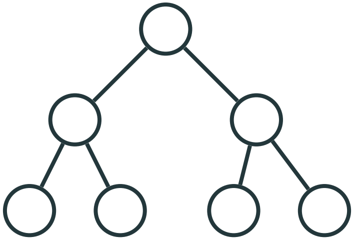
Post-order Traversing

In the *post-order,* we visit the root **after** traversing the sub-trees.



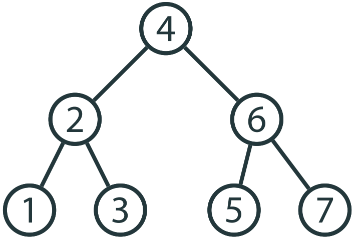
In-order Traversing

For binary trees, the third traversing order is available. In the *in-order*, we first traverse the left sub-tree, then visit the root, and then traverse the right sub-tree.

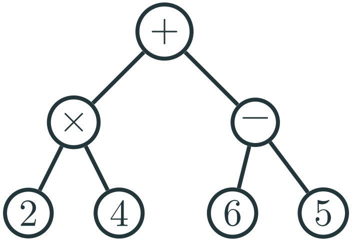


In-order Traversing

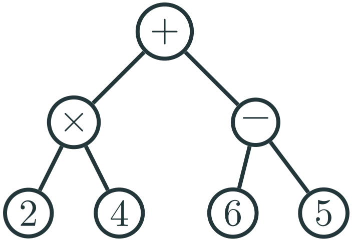
For binary trees, the third traversing order is available. In the *in-order*, we first traverse the left sub-tree, then visit the root, and then traverse the right sub-tree.



Example: Syntax Trees

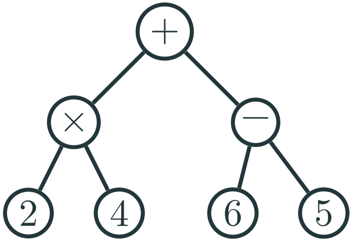


Example: Syntax Trees



Traversing orders correspond to different readings of the formula:

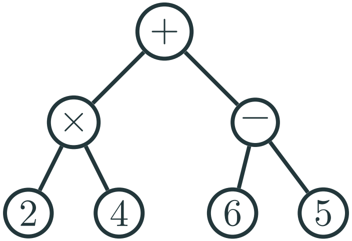
Example: Syntax Trees



Traversing orders correspond to different readings of the formula:

- Pre-order: Polish notation. $+ \times 2\,4 - 6\,5$

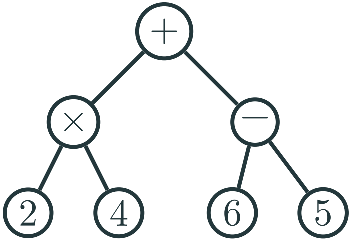
Example: Syntax Trees



Traversing orders correspond to different readings of the formula:

- Pre-order: Polish notation. $+ \times 2\,4 - 6\,5$
- Post-order: reverse Polish. $2\,4 \times 6\,5 - +$.

Example: Syntax Trees



Traversing orders correspond to different readings of the formula:

- Pre-order: Polish notation. $+ \times 2\,4 - 6\,5$
- Post-order: reverse Polish. $2\,4 \times 6\,5 - +$.
- In-order: infix notation. $(2 \times 4) + (6 - 5)$

Outline

Traversing Trees

Traversing Graphs: DFS and BFS

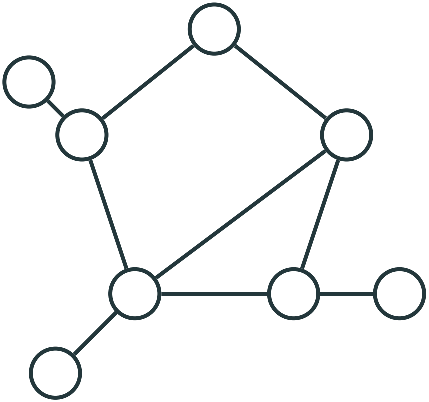
Depth-First Search

- Depth-first search (DFS) traversing, in the pre-order version, can be applied to arbitrary graphs.

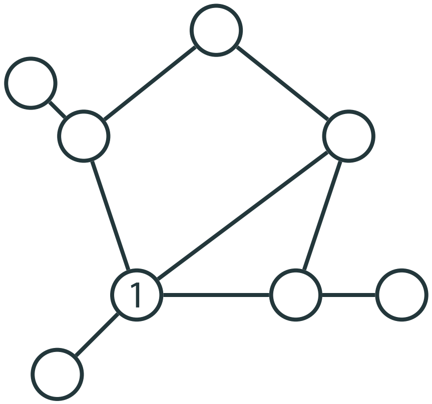
Depth-First Search

- Depth-first search (DFS) traversing, in the pre-order version, can be applied to arbitrary graphs.
- Difference from trees: now we have to remember which vertices were already visited (via another route).

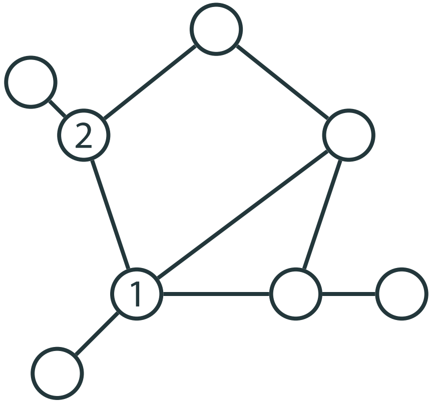
Depth-First Search



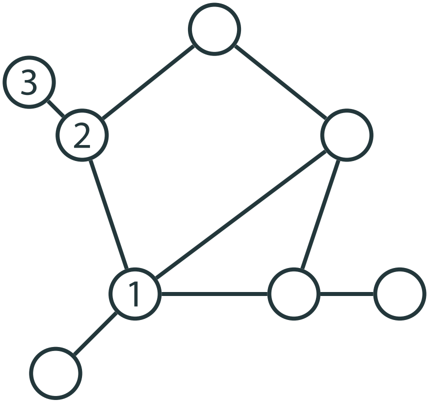
Depth-First Search



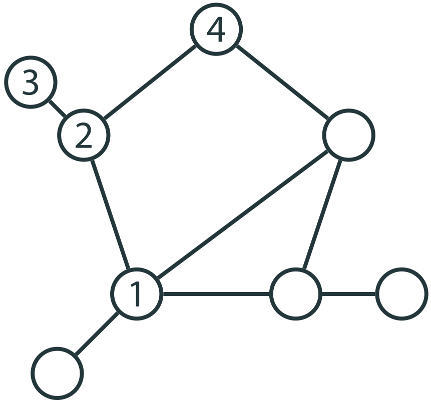
Depth-First Search



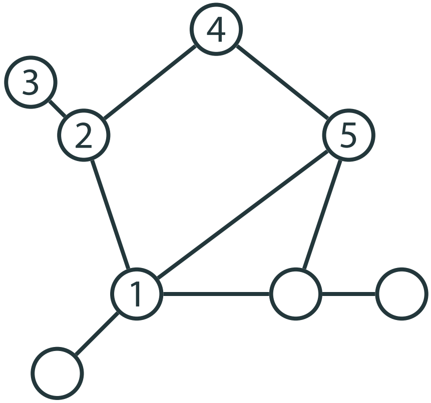
Depth-First Search



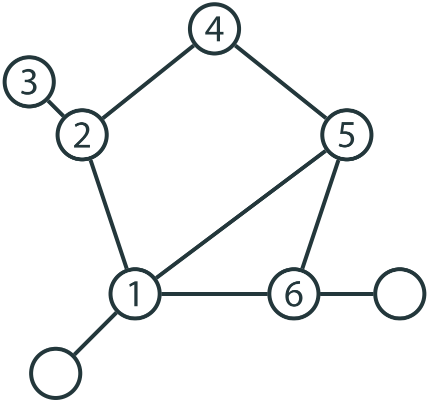
Depth-First Search



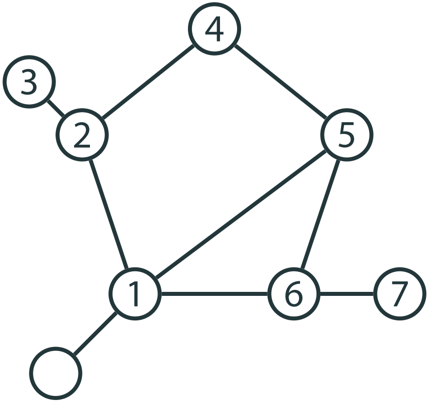
Depth-First Search



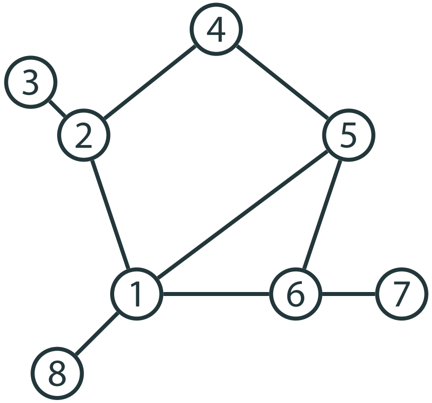
Depth-First Search



Depth-First Search



Depth-First Search



Breadth-First Search

- DFS does not always find the shortest way to a vertex.

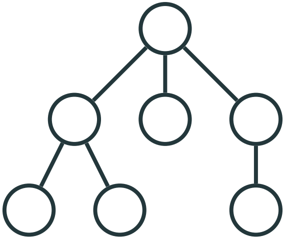
Breadth-First Search

- DFS does not always find the shortest way to a vertex.
- In breadth-first search (BFS), vertices are enumerated in a uniform manner.

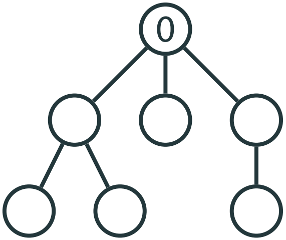
Breadth-First Search

- DFS does not always find the shortest way to a vertex.
- In breadth-first search (BFS), vertices are enumerated in a uniform manner.
- At each step, the BFS algorithm visits **all** new vertices which are adjacent to vertices visited at the previous step.

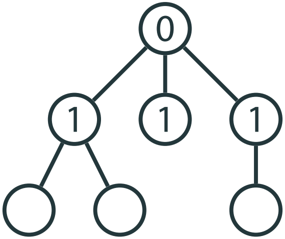
BFS on Trees



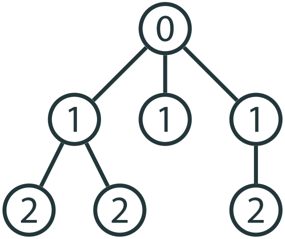
BFS on Trees



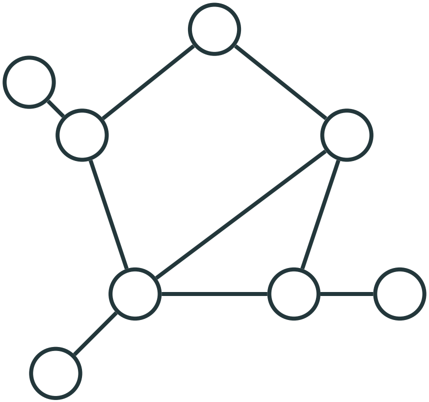
BFS on Trees



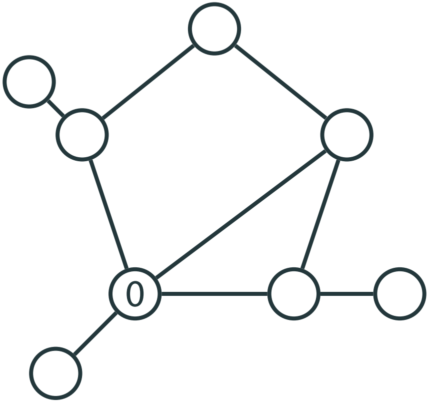
BFS on Trees



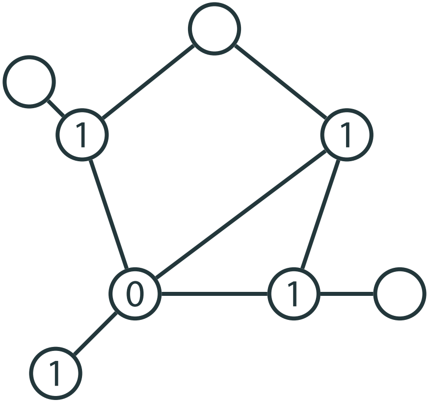
BFS on an Arbitrary Graph



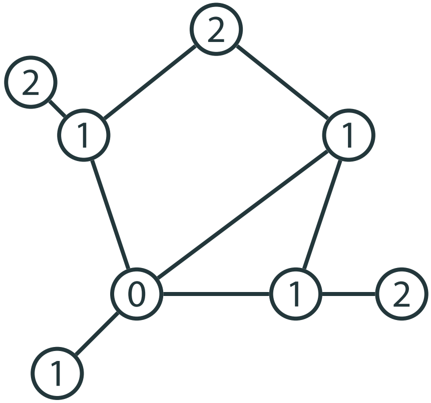
BFS on an Arbitrary Graph



BFS on an Arbitrary Graph



BFS on an Arbitrary Graph



Applications of BFS

- Computing *distances* between vertices (we'll see it later!).

Applications of BFS

- Computing *distances* between vertices (we'll see it later!).
- Testing whether a graph is bipartite (i.e., 2-coloring of a graph).

Applications of BFS

- Computing *distances* between vertices (we'll see it later!).
- Testing whether a graph is bipartite (i.e., 2-coloring of a graph).
- Finding paths in labyrinths.

Applications of BFS

- Computing *distances* between vertices (we'll see it later!).
- Testing whether a graph is bipartite (i.e., 2-coloring of a graph).
- Finding paths in labyrinths.
- Electric circuit routing.

Applications of BFS

- Computing *distances* between vertices (we'll see it later!).
- Testing whether a graph is bipartite (i.e., 2-coloring of a graph).
- Finding paths in labyrinths.
- Electric circuit routing.
- ...