BIOINFORMATICS AND STATISTICAL GENETICS

GABRIEL VALIENTE

ALGORITHMS, BIOINFORMATICS, COMPLEXITY AND FORMAL METHODS RESEARCH GROUP,
TECHNICAL UNIVERSITY OF CATALONIA

2023–2024

- K. Sayood and H. H. Otu. *Bioinformatics: A One Semester Course*. Springer, Cham, Switzerland, 2022
- T. Dandekar and M. Kunz. *Bioinformatics: An Introductory Textbook*. Springer, Berlin, Heidelberg, 2023
- J. Ramsden. *Bioinformatics: An Introduction*. Computational Biology. Springer, Cham, Switzerland, 4nd edition, 2023

- D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, England, 1997
- N. C. Jones and P. A. Pevzner. *An Introduction to Bioinformatics Algorithms*. The MIT Press, Cambridge, MA, 2004
- P. Compeau and P. A. Pevzner. *Bioinformatics Algorithms: An Active Learning Approach*. Active Learning Publishers, La Jolla, CA, 3rd edition, 2018

- D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, England, 1997
- G. Valiente. *Combinatorial Pattern Matching Algorithms in Computational Biology Using Perl and R*. Chapman & Hall/CRC, Boca Raton, FL, 2009

- D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, England, 1997
- G. Valiente. *Combinatorial Pattern Matching Algorithms in Computational Biology Using Perl and R*. Chapman & Hall/CRC, Boca Raton, FL, 2009

- Array of parents
- First-child, next-sibling
- Graph-based representation

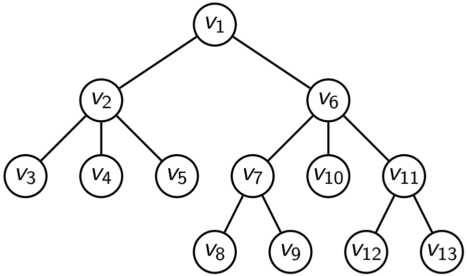
- Adjacency matrix
- Adjacency list
- Extended adjacency list
- Adjacency map

- Newick string

- Extended Newick string

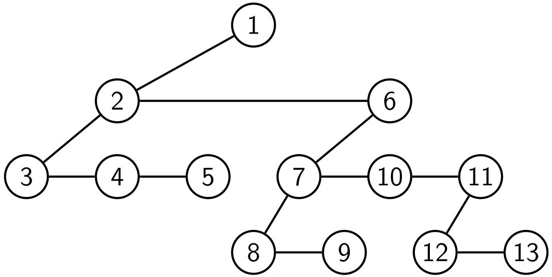
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 4th edition, 2022
- G. Valiente. *Algorithms on Trees and Graphs*. Texts in Computer Science. Springer, Cham, Switzerland, 2nd edition, 2021

- Array of parents



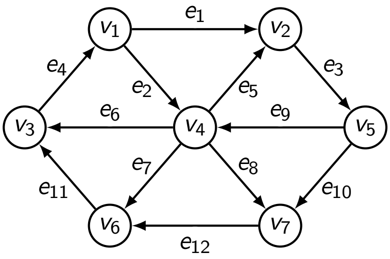
v	1	2	3	4	5	6	7	8	9	10	11	12	13
$P[v]$	*nil*	1	2	2	2	1	6	7	7	6	6	11	11

- First-child, next-sibling



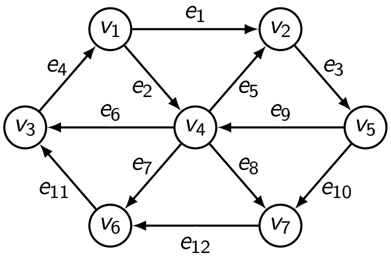
v	1	2	3	4	5	6	7	8	9	10	11	12	13
$F[v]$	2	3	*nil*	*nil*	*nil*	7	8	*nil*	*nil*	*nil*	12	*nil*	*nil*
$N[v]$	*nil*	6	4	5	*nil*	*nil*	10	9	*nil*	11	*nil*	13	*nil*

- Adjacency matrix



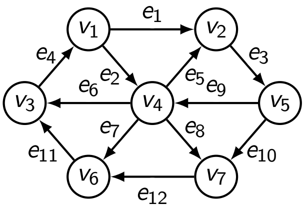
$$\begin{array}{c|ccccccc|}
 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\
\hline
v_1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
v_2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
v_3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
v_4 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
v_5 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
v_6 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
v_7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\hline
\end{array}$$

- Adjacency list



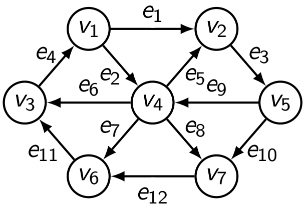
vertex	adjacent vertices
v_1	$[v_2, v_4]$
v_2	$[v_5]$
v_3	$[v_1]$
v_4	$[v_2, v_3, v_6, v_7]$
v_5	$[v_4, v_7]$
v_6	$[v_3]$
v_7	$[v_6]$

- Extended adjacency list



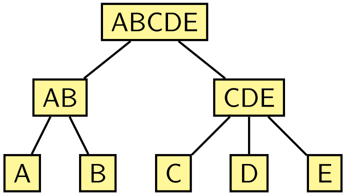
vertex	incoming edges	outgoing edges	edge	source	target
v_1	$[e_4]$	$[e_1, e_2]$	e_1	v_1	v_2
v_2	$[e_1, e_5]$	$[e_3]$	e_2	v_1	v_4
v_3	$[e_6, e_{11}]$	$[e_4]$	e_3	v_2	v_5
v_4	$[e_2, e_9]$	$[e_5, e_6, e_7, e_8]$	e_4	v_3	v_1
v_5	$[e_3]$	$[e_9, e_{10}]$	e_5	v_4	v_2
v_6	$[e_7, e_{12}]$	$[e_{11}]$	e_6	v_4	v_3
v_7	$[e_8, e_{10}]$	$[e_{12}]$	e_7	v_4	v_6
			e_8	v_4	v_7
			e_9	v_5	v_4
			e_{10}	v_5	v_7
			e_{11}	v_6	v_3
			e_{12}	v_7	v_6

- Adjacency map



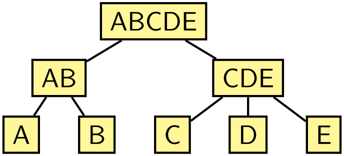
vertex	incoming edges	outgoing edges
v_1	$[v_3 \rightarrow e_4]$	$[v_2 \rightarrow e_1, v_4 \rightarrow e_2]$
v_2	$[v_1 \rightarrow e_1, v_4 \rightarrow e_5]$	$[v_5 \rightarrow e_3]$
v_3	$[v_4 \rightarrow e_6, v_6 \rightarrow e_{11}]$	$[v_1 \rightarrow e_4]$
v_4	$[v_1 \rightarrow e_2, v_5 \rightarrow e_9]$	$[v_2 \rightarrow e_5, v_3 \rightarrow e_6, v_6 \rightarrow e_7, v_7 \rightarrow e_8]$
v_5	$[v_2 \rightarrow e_3]$	$[v_4 \rightarrow e_9, v_7 \rightarrow e_{10}]$
v_6	$[v_4 \rightarrow e_7, v_7 \rightarrow e_{12}]$	$[v_3 \rightarrow e_{11}]$
v_7	$[v_4 \rightarrow e_8, v_5 \rightarrow e_{10}]$	$[v_6 \rightarrow e_{12}]$

- The Newick format is the de facto standard for representing phylogenetic trees, and it is quite convenient since it makes it possible to describe a whole tree in linear form in a unique way once the tree is drawn or the ordering among children nodes is fixed.
- The Newick description of a tree is a string of nested parentheses annotated with taxa names and possibly also with branch lengths or bootstrap values (which measure how consistently the phylogenetic tree topology is supported by the underlying data).



```
((A,B)AB,(C,D,E)CDE)ABCDE;
```

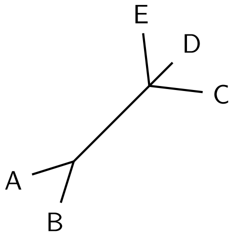
- The Newick description of a given tree can be obtained by traversing the tree in postorder and writing down the name or label of the node when visiting a terminal (taxon) node, a left parenthesis (preceded by a comma unless the node is the first child of its parent) when visiting a non-terminal node for the first time, and a right parenthesis followed by the name or label of the node (if any) when visiting a non-terminal node for the second time, that is, after having visited all its descendants.
- The name of a node is preceded by a comma unless it is the first child of its parent, and it is followed by a colon and the length (if any) of the branch from its parent.
- The description of the tree is terminated with a semicolon.



```
((A,B)AB,(C,D,E)CDE)ABCDE;
```

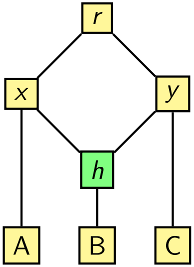
first visit non-terminal node ABCDE	`(`
first visit non-terminal node AB	`((`
visit terminal node A	`((A`
visit terminal node B	`((A,B`
second visit non-terminal node AB	`((A,B)AB`
first visit non-terminal node CDE	`((A,B)AB,(`
visit terminal node C	`((A,B)AB,(C`
visit terminal node D	`((A,B)AB,(C,D`
visit terminal node E	`((A,B)AB,(C,D,E`
second visit non-terminal node CDE	`((A,B)AB,(C,D,E)CDE`
second visit non-terminal node ABCDE	`((A,B)AB,(C,D,E)CDE)ABCDE;`

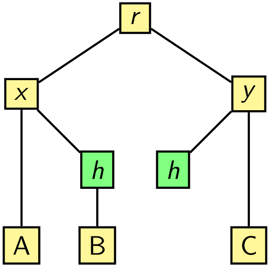
- In the Newick representation of an unrooted phylogenetic tree, there are at least three siblings connected to some internal node.



```
((A,B),C,D,E);
(A,B,(C,D,E));
```

- The eNewick format is an extension of the Newick format for representing phylogenetic trees, and it is quite convenient for representing phylogenetic networks since it makes it possible to describe a whole network in linear form in a unique way once the network is drawn or the ordering among parents and children nodes is fixed.
- The eNewick description of a network is a string of nested parentheses annotated with taxa names and possibly also with branch lengths or bootstrap values, with hybrid nodes appropriately tagged.

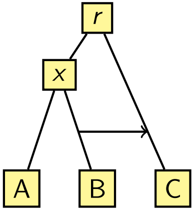


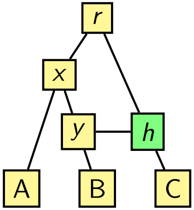


```
((A,(B)h#H1)x,(h#H1,C)y)r;
```

- The eNewick description of a given network can be obtained by first splitting each hybrid node into as many copies as parents has the node, where the first such copy carries the children and the other copies have no children, and then obtaining the Newick description of the resulting tree.
- In this way, the leftmost occurrence of each hybrid node in an eNewick string corresponds to the full description of the network rooted at that node, and all labeled occurrences of a hybrid node in an eNewick string carry the same label.
- A phylogenetic network can be recovered from an eNewick string by first recovering the tree and then identifying all copies of the same hybrid node, that is, identifying those nodes that are labeled as hybrid nodes and are tagged with the same identifier.

- The reticulate evolutionary event represented by a hybrid node in a phylogenetic network can be a recombination between genes, a hybridization between lineages, or a lateral gene transfer.
- The unique representation of the latter as hybrid nodes requires encoding each gene transfer event as a hybrid edge.
- Representation of a lateral gene transfer event as a hybrid edge in a phylogenetic network





- The eNewick string

  `((A,(B,(C)h#LGT1)y)x,h#LGT1)r;`

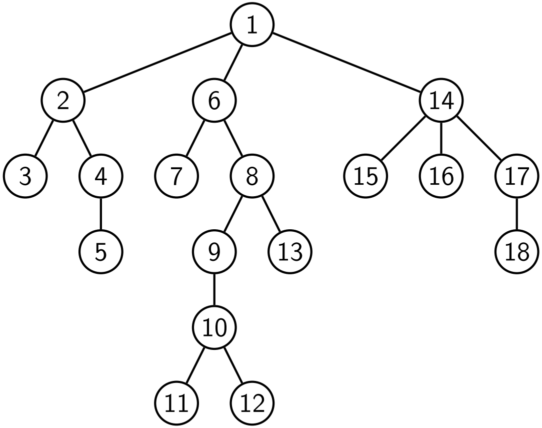
  describes such a phylogenetic network in a unique way.

- Tree traversal
  - Preorder
  - Postorder

- Graph traversal
  - Depth-first
  - Breadth-first

- T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 4th edition, 2022
- G. Valiente. *Algorithms on Trees and Graphs*. Texts in Computer Science. Springer, Cham, Switzerland, 2nd edition, 2021

- Preorder traversal of a tree



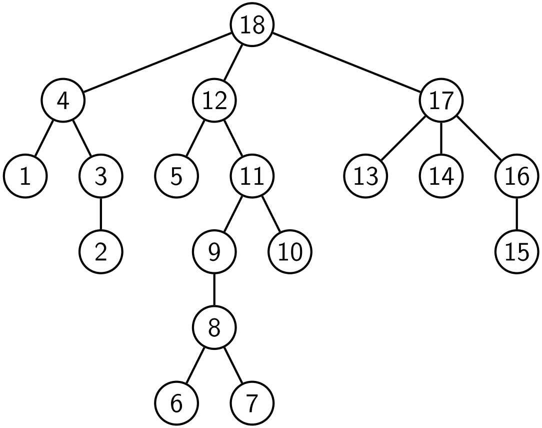
- Nodes are numbered according to the order in which they are visited during the traversal

```
**procedure** preorder_tree_traversal($T$)
    *preorder_tree_traversal*($T$, *root*[$T$])

**procedure** preorder_tree_traversal($T$, $v$)
    visit $v$
    **for all** children $w$ of node $v$ in $T$ **do**
        *preorder_tree_traversal*($T$, $w$)
```

```
procedure preorder_tree_traversal($T$)
    let $S$ be an empty stack (of nodes)
    push $root[T]$ onto $S$
    while $S$ is not empty do
        pop from $S$ the top node $v$
        visit $v$
        for all children $w$ of node $v$ in $T$ in reverse order do
            push $w$ onto $S$
```

- Postorder traversal of a tree



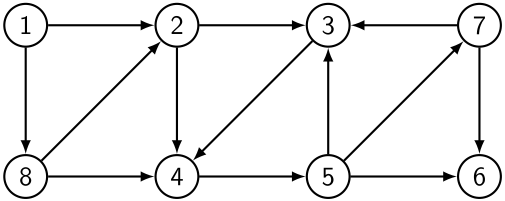
- Nodes are numbered according to the order in which they are visited during the traversal

```
**procedure** postorder_tree_traversal($T$)
    *postorder_tree_traversal*($T$, *root*[$T$])

**procedure** postorder_tree_traversal($T$, $v$)
    **for all** children $w$ of node $v$ in $T$ **do**
        *postorder_tree_traversal*($T$, $w$)
    visit $v$
```

```
**procedure** postorder_tree_traversal($T$)
    let $S$ be an empty stack (of nodes)
    push $root[T]$ onto $S$
    **while** $S$ is not empty **do**
        pop from $S$ the top node $v$
        reverse visit $v$
        **for all** children $w$ of node $v$ in $T$ **do**
            push $w$ onto $S$
```

- Depth-first traversal of a graph



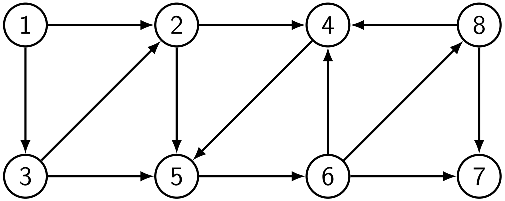
- Vertices are numbered according to the order in which they are first visited during the traversal
- The relative order of the vertices adjacent with a given vertex corresponds to the counter-clockwise ordering of the outgoing edges of the vertex in the drawing of the graph

```
**procedure** depth_first_traversal($G$)
    **for all** vertices $v$ of $G$ **do**
        $v.visited \leftarrow$ false
    **for all** vertices $v$ of $G$ **do**
        **if not** $v.visited$ **then**
            *depth_first_traversal*($G, v$)

**procedure** depth_first_traversal($G, v$)
    $v.visited \leftarrow$ true
    visit $v$
    **for all** vertices $w$ adjacent with vertex $v$ in $G$ **do**
        **if not** $w.visited$ **then**
            *depth_first_traversal*($G, w$)
```

```
procedure depth_first_traversal($G$)
    for all vertices $u$ of $G$ do
        $u.visited \leftarrow$ false
    let $S$ be an empty stack (of vertices)
    for all vertices $u$ of $G$ do
        if not $u.visited$ then
            push $u$ onto $S$
            while $S$ is not empty do
                pop from $S$ the top vertex $v$
                if not $v.visited$ then
                    $v.visited \leftarrow$ true
                    visit $v$
                    for all vertices $w$ adjacent with $v$ in $G$ in reverse order do
                        if not $w.visited$ then
                            push $w$ onto $S$
```

- Breadth-first traversal of a graph



- Vertices are numbered according to the order in which they are first visited during the traversal
- The relative order of the vertices adjacent with a given vertex corresponds to the counter-clockwise ordering of the outgoing edges of the vertex in the drawing of the graph

```
**procedure** breadth_first_traversal($G$)
    **for all** vertices $u$ of $G$ **do**
        $u.visited \leftarrow$ false
    let $Q$ be an empty queue (of vertices)
    **for all** vertices $u$ of $G$ **do**
        **if not** $u.visited$ **then**
            enqueue $u$ into $Q$
            **while** $Q$ is not empty **do**
                dequeue from $Q$ the front vertex $v$
                **if not** $v.visited$ **then**
                    $v.visited \leftarrow$ true
                    visit $v$
                    **for all** vertices $w$ adjacent with vertex $v$ in $G$ **do**
                        **if not** $w.visited$ **then**
                            enqueue $w$ into $Q$
```