INF421, Lecture 6
Trees

Leo Liberti

LIX, École Polytechnique, France

Course

- **Objective**: to teach you some data structures and associated algorithms
- **Evaluation**: TP noté en salle info le 16 septembre, Contrôle à la fin. Note: $\max(CC, \frac{3}{4}CC + \frac{1}{4}TP)$
- **Organization**: fri 26/8, 2/9, 9/9, 16/9, 23/9, 30/9, 7/10, 14/10, 21/10, amphi 1030-12 (Arago), TD 1330-1530, 1545-1745 (SI31,32,33,34)
- **Books**:
  1. Ph. Baptiste & L. Maranget, *Programmation et Algorithmique*, Ecole Polytechnique (Polycopié), 2006
  2. G. Dowek, *Les principes des langages de programmation*, Editions de l'X, 2008
  3. D. Knuth, *The Art of Computer Programming*, Addison-Wesley, 1997
  4. K. Mehlhorn & P. Sanders, *Algorithms and Data Structures*, Springer, 2008
- **Website**: `www.enseignement.polytechnique.fr/informatique/INF421`
- **Contact**: `liberti@lix.polytechnique.fr` (e-mail subject: INF421)

Lecture summary

- Introduction and reminders
- Definitions and properties
- Listing chemical trees
- Trees in psychology and languages
- Depth-First Search (DFS)
- Spanning trees

The minimal knowledge

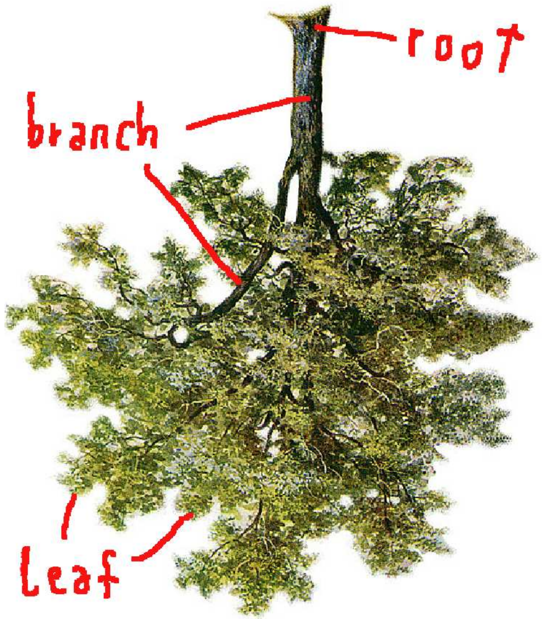
- A tree is a connected relation without cycles
- A tree on n nodes has $n-1$ branches
- There are n^{n-2} labelled trees
- The same molecular formula can correspond to different bond trees (isomers)
- The analysis of sentences yields grammatical trees
- The Graph Scanning algorithm, DFS and BFS
- The cheapest kind of distribution network is a spanning tree

Introduction and reminders

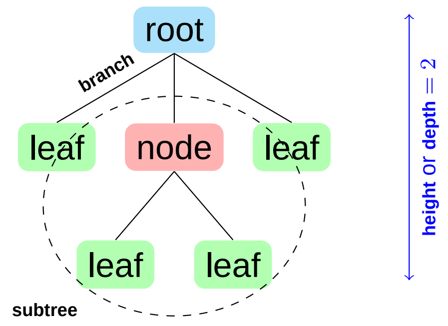
Trees

How we draw them

Nomenclature



Graphical representation



height/depth $=$ length (#branches) of longest walk [root $\rightarrow$ leaf]

Recall from INF311

- Binary trees
- Their implementations
- How to explore them in prefix, infix, postfix order
- How to store mathematical expressions in trees

Some applications of trees

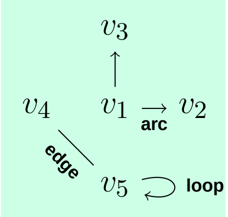
- Chemistry (molecular composition and structure)
- Psychology (natural language)
- Distribution networks of minimum cost
- Computer science
  - model for recursion (Lecture 3)
  - data structures for sorting and searching (Lecture 7)

Definitions and properties

Relations

- A **relation** A on a set V is a subset of $V \times V$

$V = \{v_1, \ldots, v_5\}$
$A = \{(v_1, v_3), (v_1, v_2), (v_4, v_5), (v_5, v_4), (v_5, v_5)\}$



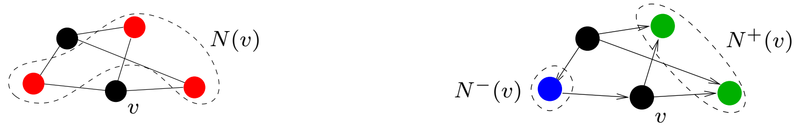
- **Arc**: an element of A; **loop**: a pair (v, v)
- **Edge**: $e = \{(u, v), (v, u)\}$ (denote by $e = \{u, v\}$)

  *(u, v are **incident to** e, and u, v are **adjacent**)*

- **Symmetric** relation: if $(u, v) \in A$, then $(v, u) \in A$
- **Reflexive** relation: $(v, v) \in A$ for all $v \in V$
- **Irreflexive** or **simple** relation: $(v, v) \notin A$ for all $v \in V$
- **Transitive** relation: if $(u, v), (v, w) \in A$ then $(u, w) \in A$

Graphs and digraphs

- A relation A on V is also called a **digraph** $G = (V, A)$
- A symmetric relation E on V is also called a **graph** $G = (V, E)$
- Digraphs have arcs (u, v), graphs have edges $\{u, v\}$
- A digraph/graph is **simple** if it has no loops
- In a graph context, nodes are also called **vertices**
- Notation: given $v \in V$,
  - *if E is symmetric* $N(v) = \{u \in V \mid \{u, v\} \in E\}$ is the **star** of v



  - *if A is not symmetric* $N^+(v) = \{u \in V \mid (v, u) \in A\}$ =**outgoing star**

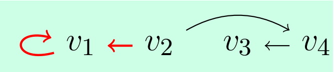
    and $N^-(v) = \{u \in V \mid (u, v) \in A\}$ =**incoming star** of v
  - Also $\delta(v) = \{\{u, v\} \mid u \in N(v)\}$, $\delta^+(v) = \{(v, u) \mid u \in N^+(v)\}$

    and $\delta^-(v) = \{(u, v) \mid u \in N^-(v)\}$ defined equivalently

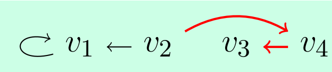
Walks and paths

- Let $i = (i_1, \ldots, i_k)$ with $k > 1$; $P = \{(v_{i_j}, v_{i_{j+1}}) \mid j < k\}$ is a **walk** $v_1 \to v_k$

$(i_1, i_2, i_3) = (2, 1, 1)$
$P = \{(v_2, v_1), (v_1, v_1)\}$



simple $(i_1, i_2, i_3) = (2, 4, 3)$
$P = \{(v_2, v_4), (v_4, v_3)\}$



- $G = (V, A)$ *a digraph, G^{-1} obtained by reversing all arcs in* A

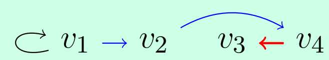
  Thm.

  If W is a walk in G, W^{-1} is a walk in G^{-1}
- A relation P is a **path** $u \to v$ if there is a walk $W \subseteq P$ from u to v such that $P = W \cup W^{-1}$

graphical representation of a path: • — • — •

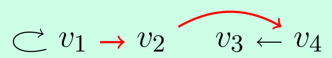
Properties of walks and paths

- Let W be a walk given by the node sequence $v_{i_1}, \ldots, v_{i_k}$
- Every contiguous subsequence of $v_{i_1}, \ldots, v_{i_k}$ is also a walk



v_4, v_3 subwalk of v_1, v_2, v_4, v_3

- If W_1 is a walk $u \to v$ and W_2 is a walk $v \to w$, then the sequence $W = W_1 \cup W_2$ is a walk $u \to w$

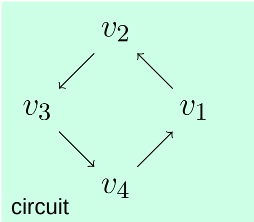


v_1, v_2 and v_2, v_4 walks $\Rightarrow v_1, v_2, v_4$ a walk

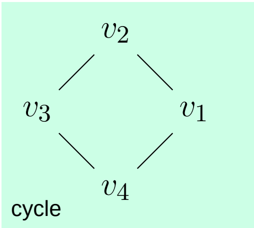
- The same holds for paths

Circuits and cycles

- If a walk has $i_1 = i_k$: **circuit**

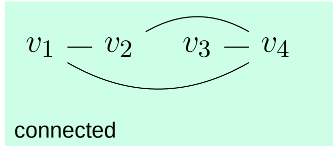


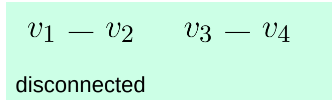
- If a path with at least 3 nodes has $i_1 = i_k$: **cycle**



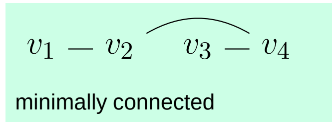
Connectedness

- Let A be a symmetric relation
- If for all $u, v \in V$ there is a path $u \to v$ in A, then A is **connected**, otherwise **disconnected**





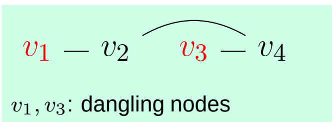
- *If A is not symmetric, equivalent notion is* **strong connectivity** *(replace "path" with "walk")*
- Let e be an edge in A, if $A \smallsetminus \{e\}$ is disconnected, A is **minimally connected**



Mathematical definition of a tree

Tree: a minimally connected relation T on a set V

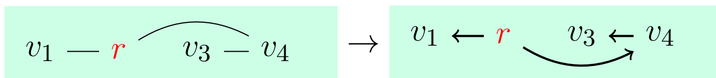
- If one node is specified as the **root**, then the tree is **rooted**
- Every node which only appears as part of a single edge is called a **dangling node**



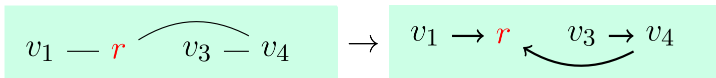
- A **dangling node** which is not the root is called a **leaf**
- Edges of a rooted tree are also called **branches**

Orientations

- The **outward orientation** of a tree T with root $r \in V$ is a relation U such that:
  - for every edge $\{(u, v), (v, u)\}$ of T, U contains only one of the arcs
  - for every leaf node ℓ of T, U has a path $r \to \ell$



- The **inward orientation** is such that for every leaf node ℓ of T, U has a path $\ell \to r$



A tree has no cycles

Lemma

A cycle is not minimally connected

Proof

Cycle: a path $C = W \cup W^{-1}$ where W is a walk $(v_{i_1}, \ldots, v_{i_k})$ with $i_1 = i_k$ and $k \geq 3$

Every contiguous subsequence of W is a (sub)walk of W

Consider any subwalk $W_1 = (v_{i_j}, \ldots, v_{i_h})$ of W with $j < h$

Both $(v_{i_1}, \ldots, v_{i_j})$ and $(v_{i_h}, \ldots, v_{i_k})$ are contiguous subseq. of W, hence walks in W

Their union $W_0 = (v_{i_h}, v_{i_{h+1}}, \ldots, v_{i_k} = v_{i_1}, \ldots, v_{i_j})$ is also a walk in W

Since $W^{-1} \subseteq C$, the walk $W_2 = W_0^{-1}$ is also in C

Since C is symmetric, the paths P_1, P_2 induced by W_1, W_2 are both in C

Notice P_1, P_2 are two paths $v_{i_j} \to v_{i_h}$ that have no common edges

Notice also that $P_1 \cup P_2 = C$

Taking away an edge from P_1 or P_2 does not disconnect C

C is not minimally connected

Thm.

A tree has no cycles

A tree has $|V| - 1$ edges

Thm.

A tree T on a set V has $|V| - 1$ edges

Proof

Let $m(T)$ be the number of edges in T

Show $m(T) = |V| - 1$ by induction on $|V|$

If $|V| = 2$, a minimally connected relation requires one edge

Induction hypothesis: Suppose $m(T) = |V| - 2$ for all trees T on $|V| - 1$ nodes

Let T be any tree on V

Any tree must have at least one leaf node ℓ (why?)

Because ℓ is a leaf, it is incident to only one edge e

Consider the tree $T' = T \smallsetminus \{e\}$ on $V' = V \smallsetminus \{\ell\}$

Because $|V'| = |V| - 1$, $m(T') = |V| - 2$ by the induction hypothesis

Thus, T has exactly $m(T) = m(T \cup \{e\}) = m(T) + 1 = |V| - 1$ edges

The converse

Thm.

If T is a symmetric relation on V with no cycles and $m(T) = |V| - 1$, then T is a tree

Proof

By induction on $|V|$, aim to show T is a tree

Recall: $\forall v \in V$, $\delta(v)$ is the set of edges incident to v

Since T has no cycles, there must be at least one node ℓ with $|\delta(\ell)| = 1$ (why?)

Let $V' = V \smallsetminus \{\ell\}$ and $T' = T \smallsetminus \{e\}$, where $\{e\} = \delta(\ell)$

Since T has no cycles, T' has no cycles either (why?)

Since $|T'| = |T| - 1$ and $|V'| = |V| - 1$, we have $|T'| = |V'| - 1$

By the induction hypothesis, $|T'|$ is a tree

Hence T is minimally connected

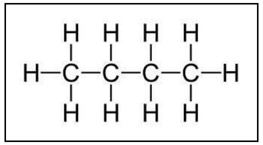
Since e is the only edge in T incident to ℓ, $T = T' \cup \{e\}$ is also minimally connected

Hence T is a tree

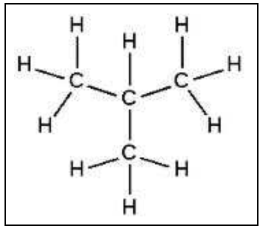
Chemical trees

Molecular descriptions

- Until the mid-XIX century, people thought molecules were completely defined by their atomic formula
- E.g. paraffins are $C_k H_{2k+2}$
- Then people started to notice that different bond relations gave rise to substances with different properties: **isomers**



butane



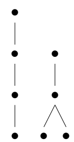
isobutane

Listing isomers

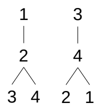
- Carbons have valence 4 (they can be incident to 4 edges)
- Hydrogens have valence 1 (they can be incident to 1 edge)
- Paraffins are known to have tree-like bond relations
- Finding paraffin isomers in the mid-XIX century:
  - list all trees on $n = 3k + 2$ nodes
  - remove those whose valences does not match the paraffin chemical formula
- How do we list all trees? How many are there?

Listing labelled trees

- Two possible interpretations
- These two are different *(unlabelled trees)*:



- These two are different *(labelled trees)*:



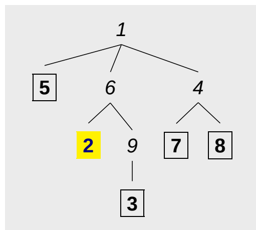
- Counting/listing labelled trees easier than unlabelled ones
- There are more labelled than unlabelled trees (why?)

Prüfer sequences

Mapping trees on V to sequences in $V^{|V|-2}$

- For a tree T let $L(T)$ be the set of leaf nodes of T

for $k \in \{1, \dots, |V| - 2\}$ **do**
  $v = \min L(T)$;
  let e be the only edge incident to v;
  let $t_k \neq v$ be the other node incident to e;
  $T \leftarrow T \smallsetminus \{v\}$;
end for
return $t = (t_1, \dots, t_{|V|-2})$

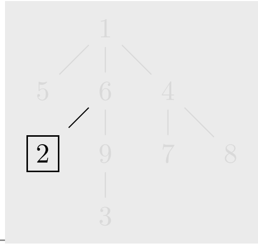


First iteration
$L(T) = \{5, 2, 3, 7, 8\}$, $v = 2$, $t = (6)$
Prüfer sequence of example: $(6, 9, 1, 4, 4, 1, 6)$

Back to the trees

Mapping $V^{|V|-2}$ to trees

1. Given a Prüfer sequence p on V, e.g. $(6,9,1,4,4,1,6)$
2. Find smallest index ℓ in $V \smallsetminus p$, e.g. 2
3. Add $\{\ell, t_1\}$ to T, e.g. $\{2,6\}$
4. Remove t_1 from t, e.g. $t = (9,1,4,4,1,6)$
5. Remove ℓ from V, e.g. $V \smallsetminus t = \{3,5,7,8\}$
6. Repeat from Step 2 until $t = \varnothing$
7. At this point $|V \smallsetminus t| = 2$ (it is an edge): add it



First iteration
$V \smallsetminus t = \{\boxed{2}, 3, 5, 7, 8\}$, $\ell = 2$,
$p = (6, 9, 1, 4, 4, 1, 6)$, edge $\{2, 6\}$

Bijection

Thm.

There is a bijection between trees on V and sequences in $V^{|V|-2}$

Proof

Essentially follows by two algorithms above

Left to prove: no cycles occur when constructing the tree from the sequence

Then result will follow by the "converse theorem" on slide 22 (why?)

Claim: no cycles, proceed by contradiction

Notice the mapping trees $\to$ sequences always deletes leaf nodes

By definition, a cycle must have ≥ 3 nodes, and none of these can be a leaf

So the resulting sequence has at most $|V| - 3$ nodes, contradiction (why?)

Thm.

[Cayley 1889] Let $|V| = n$. There are n^{n-2} labelled trees on V

Proof

By previous theorem, the number of labelled trees is the same as the number of sequences in $V^{|V|-2}$ (this proof is by Prüfer, 1918)

Psychology and natural language

A remark

- Most people find arrays, lists, maps, queues and stacks "easier" than trees
- **Thesis 1**: the graphical representation
  People are used to read sequence-like rather than tree-like text
- **Thesis 2**: iterative *vs.* recursive
  - *Sequences are models of* **iteration** *and trees models of* **recursion**
  - *Most people think iteratively rather than recursively (?)*
- **Thesis 3**: trees require decisions
  - *Every node has* ≤ 1 *next node in a sequence*
    tree nodes might have more than one subnodes
  - $\Rightarrow$ *Scanning a sequence: no decisions to take*
    $\Rightarrow$ *Exploring a tree: which subnode to process next?*

Languages and grammars

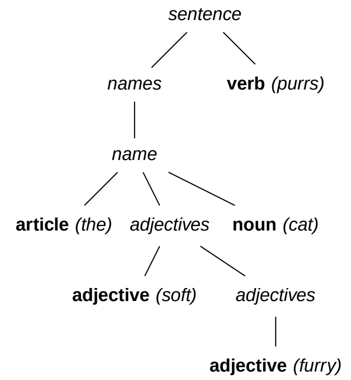
- Remember *nouns, adjectives, transitive verbs* from school?
- Analyzing sentences means to identify and name their grammatical components
- We can analyze such components recursively:

<u>sentence</u>	$\longrightarrow$	*names verb*
names	$\longrightarrow$	*name names*
name	$\longrightarrow$	*noun*
	$\|\|$	*article noun*
	$\|\|$	*adjectives noun*
	$\|\|$	*article adjectives noun*
adjectives	$\longrightarrow$	*adjective adjectives*
verb	$\longrightarrow$	...

Parse trees

The soft, furry cat purrs

<u>sentence</u>	$\longrightarrow$	*names verb*
names	$\longrightarrow$	*name names*
name	$\longrightarrow$	*noun*
	$\|\|$	*article noun*
	$\|\|$	*adjectives noun*
	$\|\|$	*article adjectives noun*
adjectives	$\longrightarrow$	*adjective adjectives*
verb	$\longrightarrow$	...



Formal and natural languages

- If there's more than one parse tree to a given sentence, the grammar is **ambiguous**
- If the different parse trees for a sentence lead to different meanings, the language itself is ambiguous
- Non-ambiguous languages are also called **formal**
  (e.g. formal logic, C/C++, Java,. . .)
- Ambiguous languages are also called **natural**
  (e.g. common mathematical language, English, French,. . .)
- Richard Montague (1930-1971) tried to supply grammar-like mechanisms that were able to disambiguate some subsets of English

Tree exploration

- Breadth-First Search (BFS — seen in Lecture 2)
  find the way out of a maze in the smallest number of steps
- Depth-First Search (DFS — seen in *polycopié* of INF311)
  find the way out of a maze
- DFS: recursive call to `dfs`(*node* v):

  1: optionally perform an action on v;
  2: **for** all subnodes u of v **do**
  3:   `dfs`(u);
  4: **end for**
  5: optionally perform an action on v;

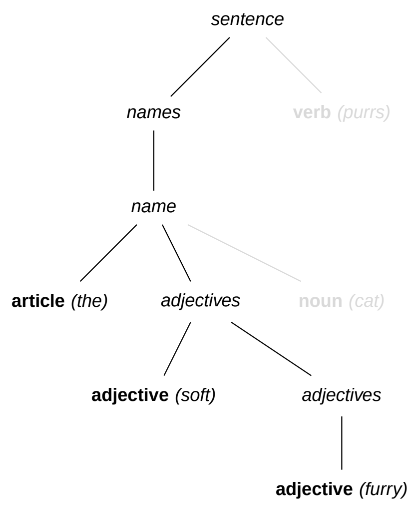
- DFS is `dfs`(*root*)

Thesis [XX century]: our brain treats sentences like mazes, and inherently uses DFS to find the way out (i.e., parse them)

How much memory?

- How much do we need to remember during DFS?
- Notice that the recursive code makes no explicit use of memory
- From Lecture 3, remember recursion is implemented using **stacks**
- What is the maximum size of the stack in exploring a tree by DFS?
- Let's see the DFS once again, and keep track of stack size

DFS on parse trees: memory



max=5 5

Memory and depth

Need as much memory as the tree depth

Recall: **depth** = longest path from root to a leaf

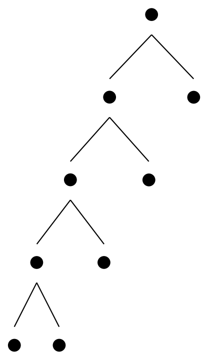
Miracles of the human mind

- However, consider this:

  We (humans) process input in a given order

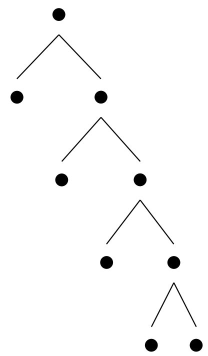
- *Reading*: left→right | right→left | top→bottom
- **Question**: *are there bottom→top languages?*
- **Western languages: left→right**
- ⇒ **DFS: no need to use stack at rightmost branch!**
- If we know we're on rightmost path and we process subnodes in left→right order, then rightmost=last
- No "climbing back up the tree" at rightmost path
- [Yngve, 1960]: *western language trees develop in depth on the right; depth on the left is limited to a constant*

Regressive and progressive trees



Regressive tree

In left→right node order, requires as much stack as the depth (4 in this case)



Progressive tree

In left→right node order, only requires a stack of constant size (1 in this case)

The "7" brain

- [Miller 1956] On average, the human memory can recall seven random words without effort
- ⇒ In western languages, it employs progressive trees with maximum "left depth" of 7
- This is why the "progressive sentence":

  l'élève retardataire n'apprend que la moitié des choses qu'on lui enseigne

  sounds much more natural than the "regressive" one:

  on enseigne des choses dont la moitié seulement est apprise par le retardataire élève

Brain and languages

- Anglosaxon languages are regressive on adjectives and appositions (often before the noun)
- Latin-derived languages decrease this tendency
- Classical latin is very difficult to understand: one has the impression that there is no fixed order!

  Inde toro pater Æneas sic orsus ab alto

  → *Thereafter seat father Eneas thus standing from a high*

  → *Thereafter father Eneas, thus standing from a high seat*
- Perhaps this is why classical latin is a dead language: it required too much "brain stack" to process sentences

Depth-First Search

(Di)Graph scanning

- DFS above explores nodes of a tree starting from the root, visit each (connected) node only once
- Generalization: scan the nodes of a digraph *(or the vertices of a graph)* starting from a node s

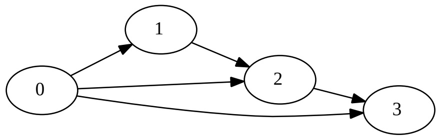
Require: $G = (V, A)$, $s \in V$, $R = \{s\}$, $Q = \{s\}$

1: **while** $Q \neq \varnothing$ **do**
2:   choose $v \in Q$ `//` v `is scanned`
3:   $Q \leftarrow Q \smallsetminus \{v\}$
4:   **for** $w \in N^+(v) \smallsetminus R$ **do**
5:     $R \leftarrow R \cup \{w\}$
6:     $Q \leftarrow Q \cup \{w\}$
7:   **end for**
8: **end while**

Storing a graph

- Seen in Lecture 1: use the *jagged array* representation (*also called* **adjacency list**)

$N^+(0) = (1, 2, 3)$
$N^+(1) = (2)$
$N^+(2) = (3)$



- Seen in Lecture 2: use the *list of arcs* representation

$$L = ((0, 1), (0, 2), (0, 3), (1, 2), (2, 3))$$

Different efficiency on different algorithms

The algorithm is correct

Thm.

If there is an oriented path P from s to $z \in V$, then DIGRAPH SCANNING scans z

Proof

- Suppose not, then $\exists (x, y) \in P$ with $x \in R$ and $y \notin R$ (for otherwise, by induction on the path length, $z \in R$ by Step 5 and hence in Q by Step 6)
- By Step 6 x was added to Q
- The algorithm does not stop before eliminating x from Q in Step 3 at some iteration
- This happens only if $\delta^+(x) \subseteq R$ by Steps 4-5
- Hence $y \notin \delta^+(x)$, which implies $(x, y) \notin P$, which yields a contradiction

The algorithm takes $O(n + m)$

Thm.

If the digraph is encoded as adjacency lists, DIGRAPH SCANNING takes CPU time proportional $O(n + m)$ in the worst case

Proof

- **Each node is considered only once**:
  - Whenever a node x is eliminated from Q, it was previously inserted by Step 6, which means that it was also added to R by Step 5
  - By Step 4, x is never re-added to Q
- **Each arc (x, y) is considered only once**:
  - When $x = v$ in Step 2 then $y \in \delta^+(x)$, so either $y = w$ in Step 4 or it must be verified that $y \in R$
  - In both cases, the relation (x, y) was considered once

The choice of $v \in Q$

- In Step 2, the choice of $v \in Q$ determines the order in which the nodes are scanned
- Can alter this using different data structures for implementing the set Q
- Two data structures are commonly used:
  1. Stacks

     DEPTH-FIRST SEARCH (DFS): this corresponds to the order being Last-In, First-Out (LIFO)
  2. Queues

     BREADTH-FIRST SEARCH: this corresponds to the order being First-In, First-Out (FIFO)

If you failed to understand BFS in Lecture 2, here's another chance!

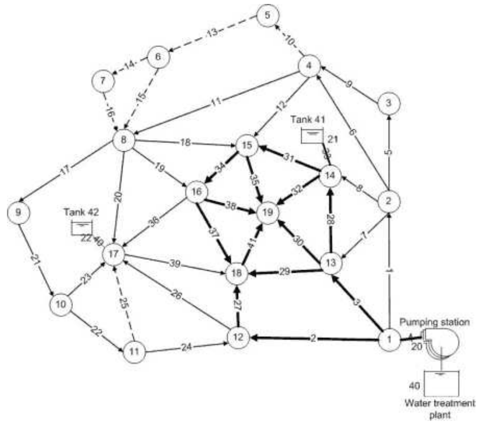
Spanning trees

Distribution networks

- A **network** is a connected relation on a set V of entities that models a distribution process
- E.g. V: production sites, customer sites
- Two sites are related if there is an exchange of material between them
- Two production sites are related if there is an exchange of raw material
- Other pairs of sites are related if there is an exchange of finished material
- Main cost of distribution: *transportation*
- How do you guarantee that each site has access to the material?

Electricity/water distribution

- Raw and finished material is the same
- Blurred distinction between production and customer sites
- Cable/duct reaches customer γ_1, it is then extended to customer γ_2 (γ_1 *is both production and customer*)
- The main cost is laying the cables/ducts



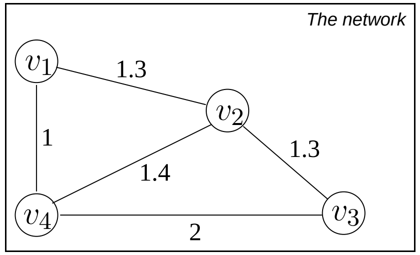
Spanning trees

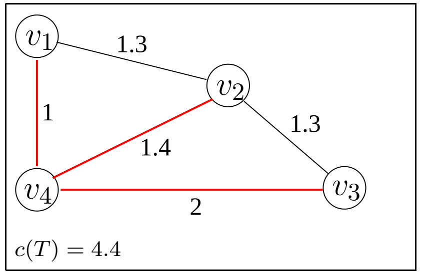
- Cost is optimized if material can be distributed to all sites using as few cables/duct as possible
- A tree on $U \subseteq V$ is **spanning** if $U = V$
- If each edge e in the network has cost c_e, the cost of T is

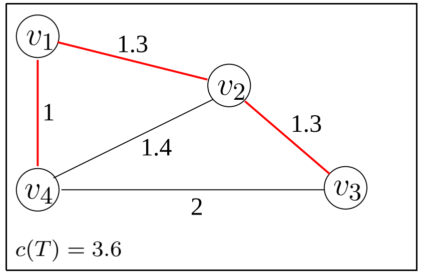
$$c(T) = \sum_{e \in T} c_e$$

Find a spanning tree of minimum cost

Example







Kruskal's algorithm: a sketch

- Two classical algorithms: Kruskal's and Prim's
- *Implementation in INF431: requires union-find data structure*
- Let E be the set of edges in the network

```
T = ∅
while |T| < |V| − 1 do
  find the edge e of minimum cost in the network E;
  if T ∪ {e} has no cycle then
    T ← T ∪ {e};
    E ← E ∖ {e};
  end if
end while
```

- At the end, T has $|V| - 1$ edges and has no cycle: it is a tree by the "converse theorem" (slide 22)

Try and prove that Kruskal's algorithm terminates

End of Lecture 6