

Discrete Structures Support Lecture

Channa Dias Perera Aron Hardeman

(slides by Evi Xhelo, Nico Stegeman, C.D.P., Eertze van de Riet and A.H.)



StudCee

26-01-2024

Cover

Table of Contents

- 1 Foundations
- 2 Counting
- 3 Proof Techniques
 - Induction
- 4 Relations
- 5 Functions
- 6 Order Relations
 - Posets and lattices
 - Boolean Algebra
- 7 Tree
 - Search
- 8 Graph
 - Prim's Algorithm
 - Kruskal's Algorithm
 - Euler Paths / Circuits

Foundations

- Sets, sequences, integer properties, matrices
- Languages and Regular Expressions
 - ϵ (or Λ) corresponds to an empty set
 - x corresponds to $\{x\}$
 - $\alpha\beta$ corresponds to $\{s \cdot t \mid s \in A \wedge t \in B\}$
 - $\alpha \vee \beta$ corresponds to $A \cup B$
 - α^* corresponds to A^*
 - $0^*(1 \vee 2)^*, \Sigma = \{0, 1, 2\}$

Counting

Sometimes you want to find out how many ways there are to pick something from a given set of objects. There are fundamentally two ways of doing so, with

Counting

Sometimes you want to find out how many ways there are to pick something from a given set of objects. There are fundamentally two ways of doing so, with

- ① **Permutations**, where we care about the *order* in which items are picked

Counting

Sometimes you want to find out how many ways there are to pick something from a given set of objects. There are fundamentally two ways of doing so, with

- ① **Permutations**, where we care about the *order* in which items are picked
- ② **Combinations**, where the order is not important



Counting

Say you have a set of n elements, and you want to find the number of **ordered** sequences of r elements which are chosen from this set. How many ways are there to do so?

Counting

Say you have a set of n elements, and you want to find the number of **ordered** sequences of r elements which are chosen from this set. How many ways are there to do so?

- Repetition is allowed (multiplication principle):

$$n^r$$



Counting

Say you have a set of n elements, and you want to find the number of **ordered** sequences of r elements which are chosen from this set. How many ways are there to do so?

- Repetition is allowed (multiplication principle):

$$n^r$$

- Repetition not allowed (permutation):

$${}_nP_r = \frac{n!}{(n-r)!} = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot (n-r+1)$$

Counting (special permutation)

One special case of permutations is when you pick all items, then:

$${}_nP_n = \frac{n!}{(n-n)!} = \frac{n!}{1} = n!$$

Counting (special permutation)

One special case of permutations is when you pick all items, then:

$${}_n P_n = \frac{n!}{(n-n)!} = \frac{n!}{1} = n!$$

- . But what if among those n items you have items that are *indistinguishable*?

Then you cannot tell (some of) the $n!$ orderings apart!

Counting (special permutation)

One special case of permutations is when you pick all items, then:

$${}_n P_n = \frac{n!}{(n-n)!} = \frac{n!}{1} = n!$$

. But what if among those n items you have items that are *indistinguishable*?

Then you cannot tell (some of) the $n!$ orderings apart!

- Distinguishable Permutations: distinguishable variant of ${}_n P_n$, where k_i is the number of duplicates of each object i

$$\frac{n!}{k_1! k_2! \cdots k_i!}$$

Counting (special permutation)

One special case of permutations is when you pick all items, then:

$${}_n P_n = \frac{n!}{(n-n)!} = \frac{n!}{1} = n!$$

. But what if among those n items you have items that are *indistinguishable*?

Then you cannot tell (some of) the $n!$ orderings apart!

- Distinguishable Permutations: distinguishable variant of ${}_n P_n$, where k_i is the number of duplicates of each object i

$$\frac{n!}{k_1! k_2! \cdots k_i!}$$

- Example: you can rearrange the letters of the word ALBUQUERQUE in $\frac{11!}{1!1!1!3!2!2!1!}$ ways.

Counting (combinations)

Say you have a set of n objects you want to choose r objects from,
and the **order does not matter**.

How many ways are there to do so?

Counting (combinations)

Say you have a set of n objects you want to choose r objects from, and the **order does not matter**.

How many ways are there to do so?

- Repetition not allowed (combinations):

$${}_n C_r = \binom{n}{r} = \frac{n!}{(n-r)!r!} = \frac{{}_nP_r}{r!}$$

Example: number of ways you can draw 5 cards out of a full set of 52 cards is $\binom{52}{5} = \frac{52!}{(52-5)!5!} = \dots$

Counting (combinations)

Say you have a set of n objects you want to choose r objects from, and the **order does not matter**.

How many ways are there to do so?

- **Repetition not allowed (combinations):**

$${}_n C_r = \binom{n}{r} = \frac{n!}{(n-r)!r!} = \frac{{}_nP_r}{r!}$$

Example: number of ways you can draw 5 cards out of a full set of 52 cards is $\binom{52}{5} = \frac{52!}{(52-5)!5!} = \dots$

- **Repetition allowed:**

$${}_{r+(n-1)} C_r = \binom{r+n-1}{r}$$

Example: next slide

Question: how many numbers $0 \leq x < 10^8$ have their digits appear in non-decreasing order (e.g. 1334677)?

Question: how many numbers $0 \leq x < 10^8$ have their digits appear in non-decreasing order (e.g. 1334677)?

"Rephrase": in how many ways can we choose 8 digits (from 0, 1, . . . , 9), where repetition is allowed and order does not matter?
(Why does this work? You can always sort the numbers in your selection, so each selection of 8 digits corresponds to exactly one number with non-decreasing digits. Also think about what it means if our selection includes zeroes.)

Question: how many numbers $0 \leq x < 10^8$ have their digits appear in non-decreasing order (e.g. 1334677)?

"Rephrase": in how many ways can we choose 8 digits (from $0, 1, \dots, 9$), where repetition is allowed and order does not matter?
(Why does this work? You can always sort the numbers in your selection, so each selection of 8 digits corresponds to exactly one number with non-decreasing digits. Also think about what it means if our selection includes zeroes.)

Solution: we want to choose $r = 8$ elements from a set of $n = 10$ elements (namely the digits $0, 1, \dots, 9$). Order does not matter and repetition is allowed. Hence, the number of ways to choose this is

Question: how many numbers $0 \leq x < 10^8$ have their digits appear in non-decreasing order (e.g. 1334677)?

"Rephrase": in how many ways can we choose 8 digits (from $0, 1, \dots, 9$), where repetition is allowed and order does not matter?
(Why does this work? You can always sort the numbers in your selection, so each selection of 8 digits corresponds to exactly one number with non-decreasing digits. Also think about what it means if our selection includes zeroes.)

Solution: we want to choose $r = 8$ elements from a set of $n = 10$ elements (namely the digits $0, 1, \dots, 9$). Order does not matter and repetition is allowed. Hence, the number of ways to choose this is

$$\begin{aligned} \binom{r+n-1}{r} &= \binom{8+10-1}{8} = \binom{17}{8} = \frac{17!}{(17-8)!8!} \\ &= \frac{17!}{9!8!} = \frac{17 \cdot 16 \cdot 15 \cdot 14 \cdot 13 \cdot 12 \cdot 11 \cdot 10}{8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = \boxed{24310} \end{aligned}$$

Pigeonhole Principle

Another concept to do with counting is the pigeonhole principle, which talks about how to divide objects among sets. It is a formal way to show the minimum size each set must have:

Pigeonhole Principle

Another concept to do with counting is the pigeonhole principle, which talks about how to divide objects among sets. It is a formal way to show the minimum size each set must have:

- If there are n pigeons assigned to m pigeonholes, one of the pigeonholes must contain $\lfloor \frac{n-1}{m} \rfloor + 1$ pigeons.

The challenge is to determine what is a pigeon and what is a pigeonhole.

Pigeonhole Principle

Another concept to do with counting is the pigeonhole principle, which talks about how to divide objects among sets. It is a formal way to show the minimum size each set must have:

- If there are n pigeons assigned to m pigeonholes, one of the pigeonholes must contain $\lfloor \frac{n-1}{m} \rfloor + 1$ pigeons.

The challenge is to determine what is a pigeon and what is a pigeonhole. Tip: Construct pigeonholes based on some condition.

Pigeonhole Principle - Example Exercise

- Show that if seven integers from 1 to 12 are chosen, then two of them will add up to 13.

Pigeonhole Principle - Example Exercise

- Show that if seven integers from 1 to 12 are chosen, then two of them will add up to 13.
- Choose $n = 7$ “pigeons” as the seven integers that were chosen

Pigeonhole Principle - Example Exercise

- Show that if seven integers from 1 to 12 are chosen, then two of them will add up to 13.
- Choose $n = 7$ “pigeons” as the seven integers that were chosen
- Note that all sets of two elements that sum up to 13 are given by

$$\{1, 12\}, \{2, 11\}, \{3, 10\}, \{4, 9\}, \{5, 8\}, \{6, 7\}$$

So there are $m = 6$ “pigeon holes”, i.e. pairs summing to 13.

Pigeonhole Principle - Example Exercise

- Show that if seven integers from 1 to 12 are chosen, then two of them will add up to 13.
- Choose $n = 7$ “pigeons” as the seven integers that were chosen
- Note that all sets of two elements that sum up to 13 are given by

$$\{1, 12\}, \{2, 11\}, \{3, 10\}, \{4, 9\}, \{5, 8\}, \{6, 7\}$$

So there are $m = 6$ “pigeon holes”, i.e. pairs summing to 13.

- By the Pidgeonhole Principle, one of the pairs summing to 13 contains $\lfloor \frac{n-1}{m} \rfloor + 1 = \lfloor \frac{7-1}{6} \rfloor + 1 = 2$ of the chosen numbers.

Recurrence Relations (definition)

Recurrence relations are functions that are defined in terms of themselves.

Examples are

Recurrence Relations (definition)

Recurrence relations are functions that are defined in terms of themselves.

Examples are

$$s_n = s_{n-1} + 4$$

$$s_1 = 10$$

$$a_n = n \cdot a_{n-1}$$

$$a_1 = 2$$

$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = 1, f(2) = 1$$

Recurrence Relations (definition)

Recurrence relations are functions that are defined in terms of themselves.

Examples are

$$s_n = s_{n-1} + 4$$

$$s_1 = 10$$

$$a_n = n \cdot a_{n-1}$$

$$a_1 = 2$$

$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = 1, f(2) = 1$$

Finding the value of e.g. s_n is tricky, because we first need to know the value of s_{n-1} , s_{n-2} and so on.

That's why it is nice to find the explicit formula of the recurrence relations:

Recurrence Relations (definition)

Recurrence relations are functions that are defined in terms of themselves.

Examples are

$$s_n = s_{n-1} + 4$$

$$s_1 = 10$$

$$a_n = n \cdot a_{n-1}$$

$$a_1 = 2$$

$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = 1, f(2) = 1$$

Finding the value of e.g. s_n is tricky, because we first need to know the value of s_{n-1} , s_{n-2} and so on.

That's why it is nice to find the explicit formula of the recurrence relations:

$$s_n = 4 \cdot (n - 1) + 10$$

which defines s_n only in terms of n and not with the previous terms

s_{n-1}, s_{n-2}, \dots

Recurrence Relations (solving)

Deriving an explicit formula for s_n involves either:

- ① Backtracking (**B**)
- ② Characteristic equation (**C**)

Which method you use depends on the format of s_n :

Recurrence Relations (solving)

Deriving an explicit formula for s_n involves either:

- ① Backtracking (**B**)
- ② Characteristic equation (**C**)

Which method you use depends on the format of s_n :

- s_n is a simple sum of past terms s_{n-1}, s_{n-2}, \dots
(linear homogeneous)? Use **C**

Recurrence Relations (solving)

Deriving an explicit formula for s_n involves either:

- ① Backtracking (**B**)
- ② Characteristic equation (**C**)

Which method you use depends on the format of s_n :

- s_n is a simple sum of past terms s_{n-1}, s_{n-2}, \dots
(linear homogeneous)? Use **C**
- s_n contains (a function of) n directly in its definition? Use **B**

Recurrence Relations (solving)

Deriving an explicit formula for s_n involves either:

- ① Backtracking (**B**)
- ② Characteristic equation (**C**)

Which method you use depends on the format of s_n :

- s_n is a simple sum of past terms s_{n-1}, s_{n-2}, \dots (linear homogeneous)? Use **C**
- s_n contains (a function of) n directly in its definition? Use **B**
- s_n contains past terms s_{n-1}, \dots as a product, square, square root etc.? Use **B**

Recurrence Relations (solving)

Deriving an explicit formula for s_n involves either:

- ① Backtracking (**B**)
- ② Characteristic equation (**C**)

Which method you use depends on the format of s_n :

- s_n is a simple sum of past terms s_{n-1}, s_{n-2}, \dots (linear homogeneous)? Use **C**
- s_n contains (a function of) n directly in its definition? Use **B**
- s_n contains past terms s_{n-1}, \dots as a product, square, square root etc.? Use **B**

Recurrence Relations (solving)

Deriving an explicit formula for s_n involves either:

- ① Backtracking (**B**)
- ② Characteristic equation (**C**)

Which method you use depends on the format of s_n :

- s_n is a simple sum of past terms s_{n-1}, s_{n-2}, \dots (linear homogeneous)? Use **C**
- s_n contains (a function of) n directly in its definition? Use **B**
- s_n contains past terms s_{n-1}, \dots as a product, square, square root etc.? Use **B**

In other words, unless s_n looks like

$$s_n = c_1 \cdot s_{n-1} + c_2 \cdot s_{n-2} + \dots + c_k \cdot s_{n-k}$$

use **B**

Backtracking

For this technique, you work your way down from s_n all the way to the base case (usually s_1):

Backtracking

For this technique, you work your way down from s_n all the way to the base case (usually s_1):

- ① Given s_n , substitute s_{n-1} with s_{n-2} using the definition of s_n (now replacing n with $n - 1$)

Backtracking

For this technique, you work your way down from s_n all the way to the base case (usually s_1):

- ① Given s_n , substitute s_{n-1} with s_{n-2} using the definition of s_n (now replacing n with $n - 1$)
- ② Repeat to find s_{n-3}, s_{n-4}, \dots and collect the other terms in a pattern

Backtracking

For this technique, you work your way down from s_n all the way to the base case (usually s_1):

- ① Given s_n , substitute s_{n-1} with s_{n-2} using the definition of s_n (now replacing n with $n - 1$)
- ② Repeat to find s_{n-3}, s_{n-4}, \dots and collect the other terms in a pattern
- ③ Once you see the pattern, go directly to the base case

Backtracking

For this technique, you work your way down from s_n all the way to the base case (usually s_1):

- ① Given s_n , substitute s_{n-1} with s_{n-2} using the definition of s_n (now replacing n with $n - 1$)
- ② Repeat to find s_{n-3}, s_{n-4}, \dots and collect the other terms in a pattern
- ③ Once you see the pattern, go directly to the base case
- ④ Replace the base case with its value and you have the explicit formula!

Backtracking - Resit 18 Ex. 3

Consider the recurrence relation:

$$x_n = n + x_{n-1}$$

Find an explicit formula for x_n , with $x_1 = 4$

The explicit formula of x_n is given by

$$x_n = n + x_{n-1}$$

Backtracking - Resit 18 Ex. 3

Consider the recurrence relation:

$$x_n = n + x_{n-1}$$

Find an explicit formula for x_n , with $x_1 = 4$

The explicit formula of x_n is given by

$$x_n = n + x_{n-1}$$

$$\stackrel{(1)}{=} n + ((n-1) + x_{n-2})$$

Backtracking - Resit 18 Ex. 3

Consider the recurrence relation:

$$x_n = n + x_{n-1}$$

Find an explicit formula for x_n , with $x_1 = 4$

The explicit formula of x_n is given by

$$x_n = n + x_{n-1}$$

$$\stackrel{(1)}{=} n + ((n-1) + x_{n-2})$$

$$\stackrel{(2)}{=} n + (n-1) + ((n-2) + x_{n-3}) = n + (n-1) + (n-2) + x_{n-3}$$

Backtracking - Resit 18 Ex. 3

Consider the recurrence relation:

$$x_n = n + x_{n-1}$$

Find an explicit formula for x_n , with $x_1 = 4$

The explicit formula of x_n is given by

$$x_n = n + x_{n-1}$$

$$\stackrel{(1)}{=} n + ((n-1) + x_{n-2})$$

$$\stackrel{(2)}{=} n + (n-1) + ((n-2) + x_{n-3}) = n + (n-1) + (n-2) + x_{n-3}$$

$$= n + (n-1) + (n-2) + ((n-3) + x_{n-4}) = \sum_{i=n-3}^n i + x_{n-4}$$

Backtracking - Resit 18 Ex. 3

Find an explicit formula for x_n , with $x_1 = 4$

The explicit formula of x_n is given by

$$x_n = n + x_{n-1}$$

$$= \sum_{i=n-3}^n i + x_{n-4}$$

Backtracking - Resit 18 Ex. 3

Find an explicit formula for x_n , with $x_1 = 4$

The explicit formula of x_n is given by

$$x_n = n + x_{n-1}$$

$$= \sum_{i=n-3}^n i + x_{n-4}$$

(3)

⋮

Backtracking - Resit 18 Ex. 3

Find an explicit formula for x_n , with $x_1 = 4$

The explicit formula of x_n is given by

$$x_n = n + x_{n-1}$$

$$= \sum_{i=n-3}^n i + x_{n-4}$$

(3)

⋮

$$= \sum_{i=n-(n-2)}^n i + x_{n-(n-1)} = \sum_{i=2}^n i + x_1$$

Backtracking - Resit 18 Ex. 3

Find an explicit formula for x_n , with $x_1 = 4$

The explicit formula of x_n is given by

$$x_n = n + x_{n-1}$$

$$= \sum_{i=n-3}^n i + x_{n-4}$$

(3)

⋮

$$= \sum_{i=n-(n-2)}^n i + x_{n-(n-1)} = \sum_{i=2}^n i + x_1$$

$$\stackrel{(4)}{=} 4 + \left(\sum_{i=1}^n i - 1 \right) = 3 + \frac{n(n+1)}{2}$$

Characteristic Equation (general solution)

- For this technique, you need s_n to be of form

$$s_n = a_1 \cdot s_{n-1} + a_2 \cdot s_{n-2} + \cdots + a_k \cdot s_{n-k}$$

i.e. linear homogeneous

Characteristic Equation (general solution)

- For this technique, you need s_n to be of form

$$s_n = a_1 \cdot s_{n-1} + a_2 \cdot s_{n-2} + \cdots + a_k \cdot s_{n-k}$$

i.e. linear homogeneous

- Then the **characteristic equation** of s_n is given by

$$r^k - a_1 \cdot r^{k-1} - a_2 \cdot r^{k-2} - \cdots - a_k = 0 \quad (\text{general})$$

$$r^2 - a_1 \cdot r - a_2 = 0 \quad (\text{degree 2})$$

which has k solutions (*roots*)

Characteristic Equation (general solution)

- For this technique, you need s_n to be of form

$$s_n = a_1 \cdot s_{n-1} + a_2 \cdot s_{n-2} + \cdots + a_k \cdot s_{n-k}$$

i.e. linear homogeneous

- Then the **characteristic equation** of s_n is given by

$$r^k - a_1 \cdot r^{k-1} - a_2 \cdot r^{k-2} - \cdots - a_k = 0 \quad (\text{general})$$

$$r^2 - a_1 \cdot r - a_2 = 0 \quad (\text{degree 2})$$

which has k solutions (*roots*)

- For characteristic equations of degree 2 with roots r_1, r_2 , we can easily find the general formula:

- If $r_1 \neq r_2$ and $r_1, r_2 \in \mathbb{R}$, then $s_n = c_1 r_1^n + c_2 r_2^n$

Characteristic Equation (general solution)

- For this technique, you need s_n to be of form

$$s_n = a_1 \cdot s_{n-1} + a_2 \cdot s_{n-2} + \cdots + a_k \cdot s_{n-k}$$

i.e. linear homogeneous

- Then the **characteristic equation** of s_n is given by

$$r^k - a_1 \cdot r^{k-1} - a_2 \cdot r^{k-2} - \cdots - a_k = 0 \quad (\text{general})$$

$$r^2 - a_1 \cdot r - a_2 = 0 \quad (\text{degree 2})$$

which has k solutions (*roots*)

- For characteristic equations of degree 2 with roots r_1, r_2 , we can easily find the general formula:

- If $r_1 \neq r_2$ and $r_1, r_2 \in \mathbb{R}$, then $s_n = c_1 r_1^n + c_2 r_2^n$

Characteristic Equation (general solution)

- For this technique, you need s_n to be of form

$$s_n = a_1 \cdot s_{n-1} + a_2 \cdot s_{n-2} + \cdots + a_k \cdot s_{n-k}$$

i.e. linear homogeneous

- Then the **characteristic equation** of s_n is given by

$$r^k - a_1 \cdot r^{k-1} - a_2 \cdot r^{k-2} - \cdots - a_k = 0 \quad (\text{general})$$

$$r^2 - a_1 \cdot r - a_2 = 0 \quad (\text{degree 2})$$

which has k solutions (*roots*)

- For characteristic equations of degree 2 with roots r_1, r_2 , we can easily find the general formula:

- If $r_1 \neq r_2$ and $r_1, r_2 \in \mathbb{R}$, then $s_n = c_1 r_1^n + c_2 r_2^n$
- If $r_1 = r_2 = r$ and $r_1, r_2 \in \mathbb{R}$, then $s_n = r^n(c_1 + c_2 n)$
- If $r_1, r_2 \notin \mathbb{R}$, then **oops** (this is not covered)

Characteristic Equation (explicit formula)

- We almost have the explicit formula for s_n , now it's just a matter of solving for c_1 , c_2 :

$$s_n = c_1 r_1^n + c_2 r_2^n \quad \text{or}$$

$$s_n = r^n(c_1 + c_2 n)$$

Characteristic Equation (explicit formula)

- We almost have the explicit formula for s_n , now it's just a matter of solving for c_1 , c_2 :

$$\begin{aligned}s_n &= c_1 r_1^n + c_2 r_2^n \quad \text{or} \\ s_n &= r^n(c_1 + c_2 n)\end{aligned}$$

- To solve for c_1, c_2 , two base cases of s should be given in the exercise.

Usually these are s_1 , s_2 and for this example we'll choose $s_1 = 4$, $s_2 = 10$

Characteristic Equation (explicit formula)

- We almost have the explicit formula for s_n , now it's just a matter of solving for c_1, c_2 :

$$\begin{aligned}s_n &= c_1 r_1^n + c_2 r_2^n \quad \text{or} \\ s_n &= r^n(c_1 + c_2 n)\end{aligned}$$

- To solve for c_1, c_2 , two base cases of s should be given in the exercise.

Usually these are s_1, s_2 and for this example we'll choose $s_1 = 4, s_2 = 10$

- Then we plug in $n = 1$ and $n = 2$ and solve the following system:

$$\begin{cases} s_1 &= c_1 r_1^1 + c_2 r_2^1 = c_2 r_1 + c_2 r_2 = 4 \\ s_2 &= r^2(c_1 + c_2 \cdot 2) = c_1 r^2 + 2c_2 r^2 = 10 \end{cases}$$

Characteristic Equation - Exam 18 Ex. 2

Consider the recurrence relation:

$$a_n = -8a_{n-1} - 16a_{n-2}$$

Find an explicit formula for a_n , with $a_1 = \alpha$ and $a_2 = \beta$

Characteristic Equation - Exam 18 Ex. 2

Consider the recurrence relation:

$$a_n = -8a_{n-1} - 16a_{n-2}$$

Find an explicit formula for a_n , with $a_1 = \alpha$ and $a_2 = \beta$

- The characteristic equation of a_n is given by

$$r^2 + 8r + 16 = 0$$

$$(r + 4)(r + 4) = 0$$

Characteristic Equation - Exam 18 Ex. 2

Consider the recurrence relation:

$$a_n = -8a_{n-1} - 16a_{n-2}$$

Find an explicit formula for a_n , with $a_1 = \alpha$ and $a_2 = \beta$

- The characteristic equation of a_n is given by

$$r^2 + 8r + 16 = 0$$

$$(r + 4)(r + 4) = 0$$

- The roots of the char. eq. are given by

$$r_1 = r_2 = r = -4$$

Characteristic Equation - Exam 18 Ex. 2

Consider the recurrence relation:

$$a_n = -8a_{n-1} - 16a_{n-2}$$

Find an explicit formula for a_n , with $a_1 = \alpha$ and $a_2 = \beta$

- The characteristic equation of a_n is given by

$$r^2 + 8r + 16 = 0$$

$$(r + 4)(r + 4) = 0$$

- The roots of the char. eq. are given by

$$r_1 = r_2 = r = -4$$

- The general formula of a_n is given by

$$a_n = (-4)^n \cdot (c_1 + c_2 n)$$

for $c_1, c_2 \in \mathbb{C}$

Characteristic Equation - Exam 18 Ex. 2

Find an explicit formula for a_n , with $a_1 = \alpha$ and $a_2 = \beta$

- The general formula of a_n is given by

$$a_n = (-4)^n \cdot (c_1 + c_2 n)$$

for $c_1, c_2 \in \mathbb{C}$

Characteristic Equation - Exam 18 Ex. 2

Find an explicit formula for a_n , with $a_1 = \alpha$ and $a_2 = \beta$

- The general formula of a_n is given by

$$a_n = (-4)^n \cdot (c_1 + c_2 n)$$

for $c_1, c_2 \in \mathbb{C}$

- Using $a_1 = \alpha$ and $a_2 = \beta$, we need to solve the following system:

$$\begin{cases} a_1 &= (-4)^1 \cdot (c_1 + c_2 \cdot 1) = -4c_1 - 4c_2 = \alpha \\ a_2 &= (-4)^2 \cdot (c_1 + c_2 \cdot 2) = 16c_1 + 32c_2 = \beta \end{cases}$$

which after some arithmetic should give

$$c_1 = -\frac{\beta+4\alpha}{16} \text{ and } c_2 = \frac{\beta+4\alpha}{16}$$

Characteristic Equation - Exam 18 Ex. 2

Find an explicit formula for a_n , with $a_1 = \alpha$ and $a_2 = \beta$

- The general formula of a_n is given by

$$a_n = (-4)^n \cdot (c_1 + c_2 n)$$

for $c_1, c_2 \in \mathbb{C}$

- Using $a_1 = \alpha$ and $a_2 = \beta$, we need to solve the following system:

$$\begin{cases} a_1 &= (-4)^1 \cdot (c_1 + c_2 \cdot 1) = -4c_1 - 4c_2 = \alpha \\ a_2 &= (-4)^2 \cdot (c_1 + c_2 \cdot 2) = 16c_1 + 32c_2 = \beta \end{cases}$$

which after some arithmetic should give

$$c_1 = -\frac{\beta+4\alpha}{16} \text{ and } c_2 = \frac{\beta+4\alpha}{16}$$

- We conclude the explicit formula of a_n is given by

$$a_n = (-4)^n \cdot \frac{\beta + 4\alpha}{16} (n - 1)$$

Tips

- Use all definitions
- Take care when using \iff and \implies
- Verbalize your argument
- Use a specific example
- Structure your proofs neatly! Make a claim, followed by the proof, followed by the conclusion.

Induction

- **Claim:** Define the claim as $P(k)$. Show that $P(n)$ holds for $n \geq n_0$.
- **Base case:** Show $P(n_0)$
- **Induction Hypothesis:** Assume $P(k)$, where $k \geq n_0$
- **Inductive Step:** Show that assuming **IH** $P(k + 1)$ is true.
- **Conclusion:** Since $P(k) \implies P(k + 1)$ and that $P(n_0)$ is true, we conclude that $P(n)$ is true for $n \geq n_0$, by mathematical induction.

Strong Induction

- **Claim:** Define the claim as $P(k)$. Show that $P(n)$ holds for $n \geq n_0$.
- **Base case:** Show $P(n_0), \dots, P(n_1)$ is true. Choose n_1 depending on how many base cases are needed for the inductive step.
- **Induction Hypothesis:** Assume $P(n_0), P(n_0 + 1), \dots, P(k)$, where $k \geq n_0$
- **Inductive Step:** Show that assuming **IH** $P(k + 1)$ is true.
- **Conclusion:** Since $(P(n_0) \wedge \dots \wedge P(k)) \implies P(k + 1)$ and that $P(n_0)$ is true, we conclude that $P(n)$ is true for $n \geq n_0$, by mathematical induction.

Ex. 1 - Exam 2018

- **Question:** Let A_1, \dots, A_n and B_1, \dots, B_n be sets such that $A_i \subseteq B_i$ for every $i \in \{1, \dots, n\}$. Prove that:

$$\bigcup_{i=1}^n A_i \subseteq \bigcup_{i=1}^n B_i$$

Ex. 1 - Exam 2018

- **Question:** Let A_1, \dots, A_n and B_1, \dots, B_n be sets such that $A_i \subseteq B_i$ for every $i \in \{1, \dots, n\}$. Prove that:

$$\bigcup_{i=1}^n A_i \subseteq \bigcup_{i=1}^n B_i$$

- Base case: $n = 1$

$$\bigcup_{i=1}^1 A_i \subseteq \bigcup_{i=1}^1 B_i \iff A_1 \subseteq B_1$$

True, since $A_i \subseteq B_i, \forall i \in \{1, \dots, n\}$

Ex. 1 - Exam 2018

- **Question:** Let A_1, \dots, A_n and B_1, \dots, B_n be sets such that $A_i \subseteq B_i$ for every $i \in \{1, \dots, n\}$. Prove that: $\bigcup_{i=1}^n A_i \subseteq \bigcup_{i=1}^n B_i$
- Induction step: Assuming

$$\bigcup_{i=1}^n A_i \subseteq \bigcup_{i=1}^n B_i \text{ (IH)}$$

We wish to prove:

$$\bigcup_{i=1}^{n+1} A_i \subseteq \bigcup_{i=1}^{n+1} B_i$$

Ex. 1 - Exam 2018

- **Question:** Let A_1, \dots, A_n and B_1, \dots, B_n be sets such that $A_i \subseteq B_i$ for every $i \in \{1, \dots, n\}$. Prove that: $\bigcup_{i=1}^n A_i \subseteq \bigcup_{i=1}^n B_i$
- Observe:

$$\bigcup_{i=1}^{n+1} A_i \subseteq \bigcup_{i=1}^{n+1} B_i$$

$$\iff \bigcup_{i=1}^n A_i \cup A_{n+1} \subseteq \bigcup_{i=1}^n B_i \cup B_{n+1} \text{ (definition of } \bigcup)$$

Ex. 1 - Exam 2018

- **Question:** Let A_1, \dots, A_n and B_1, \dots, B_n be sets such that $A_i \subseteq B_i$ for every $i \in \{1, \dots, n\}$. Prove that: $\bigcup_{i=1}^n A_i \subseteq \bigcup_{i=1}^n B_i$
- Goal:

$$\bigcup_{i=1}^n A_i \cup A_{n+1} \subseteq \bigcup_{i=1}^n B_i \cup B_{n+1}$$

We know:

$$\bigcup_{i=1}^n A_i \subseteq \bigcup_{i=1}^n B_i \wedge A_{n+1} \subseteq B_{n+1}$$

From inductive hypothesis + question assumptions

Ex. 1 - Exam 2018

We have

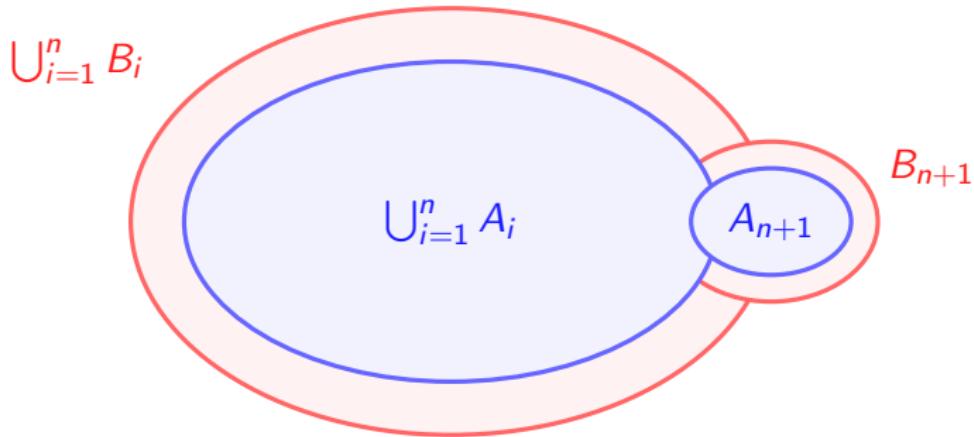
$$\begin{aligned} \bigcup_{i=1}^n A_i &\subseteq \bigcup_{i=1}^n B_i \wedge A_{n+1} \subseteq B_{n+1} \\ \implies \bigcup_{i=1}^n A_i \cup A_{n+1} &\subseteq \bigcup_{i=1}^n B_i \cup B_{n+1} \end{aligned}$$

And so we have our claim! Thus, if A_1, \dots, A_n and B_1, \dots, B_n are sets such that $A_i \subseteq B_i$ for every $i \in \{1, \dots, n\}$, $\bigcup_{i=1}^n A_i \subseteq \bigcup_{i=1}^n B_i$ by induction.

Why? - Mathematically

- Let $x \in \bigcup_{i=1}^n A_i \cup A_{n+1}$.
- Then $x \in \bigcup_{i=1}^n A_i$ or $x \in A_{n+1}$.
 - In the former, $x \in \bigcup_{i=1}^n B_i$, by IH.
 - In the latter, $x \in B_{n+1}$, by assumption of question.

Why? - Visually



Relations

A (binary) relation R from S to T is a subset $R \subseteq (S \times T)$. It “relates” elements in S to elements in T ; that is, $(a, b) \in R$ means that a is R -related to b .

There are 3 ways to represent a relation R .

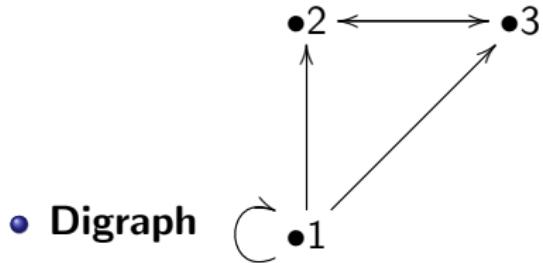
Three ways to express a relation on the set $\{1, 2, 3\}$:

Relations

A (binary) relation R from S to T is a subset $R \subseteq (S \times T)$. It “relates” elements in S to elements in T ; that is, $(a, b) \in R$ means that a is R -related to b .

There are 3 ways to represent a relation R .

Three ways to express a relation on the set $\{1, 2, 3\}$:

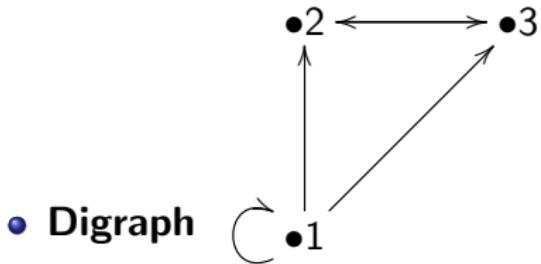


Relations

A (binary) relation R from S to T is a subset $R \subseteq (S \times T)$. It “relates” elements in S to elements in T ; that is, $(a, b) \in R$ means that a is R -related to b .

There are 3 ways to represent a relation R .

Three ways to express a relation on the set $\{1, 2, 3\}$:



- **Set**

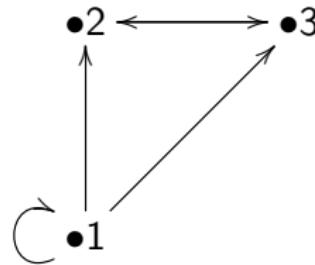
$$R = \{(1, 1), (1, 2), (1, 3), (2, 3), (3, 2)\}$$

Relations

A (binary) relation R from S to T is a subset $R \subseteq (S \times T)$. It “relates” elements in S to elements in T ; that is, $(a, b) \in R$ means that a is R -related to b .

There are 3 ways to represent a relation R .

Three ways to express a relation on the set $\{1, 2, 3\}$:

- **Digraph** 
- **Set** $R = \{(1, 1), (1, 2), (1, 3), (2, 3), (3, 2)\}$
- **Matrix**

$$M_R = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \\ 2 & 0 & 0 & 1 \\ 3 & 0 & 1 & 0 \end{pmatrix}$$

$$R = \{(1, 1), (1, 2), (1, 3), (2, 3), (3, 2)\}$$

Relation paths

If we have $R : S \rightarrow S$, an interesting question is whether any two elements $a, b \in S$ are related through R .

Relation paths

If we have $R : S \rightarrow S$, an interesting question is whether any two elements $a, b \in S$ are related through R . To find the answer, we can try to construct a **path** π between a, b . This is possible with digraphs, but for multiple computations using matrix M_R is easiest.

Relation paths

If we have $R : S \rightarrow S$, an interesting question is whether any two elements $a, b \in S$ are related through R . To find the answer, we can try to construct a **path** π between a, b . This is possible with digraphs, but for multiple computations using matrix M_R is easiest.

Notation:

- $aR^n b$ means that there is a path of length n from a to b .
If so, there is a 1 in the (a, b) cell of M_{R^n} where M_{R^n} is computed as

$$M_{R^n} = M_R \odot M_R \odot \cdots \odot M_R$$

Relation paths

If we have $R : S \rightarrow S$, an interesting question is whether any two elements $a, b \in S$ are related through R . To find the answer, we can try to construct a **path** π between a, b . This is possible with digraphs, but for multiple computations using matrix M_R is easiest.

Notation:

- $aR^n b$ means that there is a path of length n from a to b .
If so, there is a 1 in the (a, b) cell of M_{R^n} where M_{R^n} is computed as

$$M_{R^n} = M_R \odot M_R \odot \cdots \odot M_R$$

- $aR^\infty b$ means that there's a path of length 1 or more from a to b .

$$M_{R^\infty} = M_R \vee M_{R^2} \vee \cdots \vee M_{R^n}$$

If there is a 1 in the (a, b) cell of M_{R^∞} , we say b is *reachable* from a , or $aR^\infty b$.

Properties of Relations

The following properties are of interest when talking about relation R on set S :

Properties of Relations

The following properties are of interest when talking about relation R on set S :

Reflexive $\forall x \in S : (x, x) \in R$

Properties of Relations

The following properties are of interest when talking about relation R on set S :

Reflexive $\forall x \in S : (x, x) \in R$

Symmetry $\forall x, y \in S : (x, y) \in R \Rightarrow (y, x) \in R$

Properties of Relations

The following properties are of interest when talking about relation R on set S :

Reflexive $\forall x \in S : (x, x) \in R$

Symmetry $\forall x, y \in S : (x, y) \in R \Rightarrow (y, x) \in R$

Transitivity $\forall x, y, z \in S : (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$

Properties of Relations

The following properties are of interest when talking about relation R on set S :

Reflexive $\forall x \in S : (x, x) \in R$

Symmetry $\forall x, y \in S : (x, y) \in R \Rightarrow (y, x) \in R$

Transitivity $\forall x, y, z \in S : (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$

Irreflexive $\forall x \in S : (x, x) \notin R$

Properties of Relations

The following properties are of interest when talking about relation R on set S :

Reflexive $\forall x \in S : (x, x) \in R$

Symmetry $\forall x, y \in S : (x, y) \in R \Rightarrow (y, x) \in R$

Transitivity $\forall x, y, z \in S : (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$

Irreflexive $\forall x \in S : (x, x) \notin R$

Asymmetric $\forall x, y \in S : (x, y) \in R \Rightarrow (y, x) \notin R$

Properties of Relations

The following properties are of interest when talking about relation R on set S :

Reflexive $\forall x \in S : (x, x) \in R$

Symmetry $\forall x, y \in S : (x, y) \in R \Rightarrow (y, x) \in R$

Transitivity $\forall x, y, z \in S : (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$

Irreflexive $\forall x \in S : (x, x) \notin R$

Asymmetric $\forall x, y \in S : (x, y) \in R \Rightarrow (y, x) \notin R$

Anti-symmetric $\forall x, y \in S : (x, y) \in R \wedge (y, x) \in R \Rightarrow x = y$

Properties of Relations

The following properties are of interest when talking about relation R on set S :

Reflexive $\forall x \in S : (x, x) \in R$

Symmetry $\forall x, y \in S : (x, y) \in R \Rightarrow (y, x) \in R$

Transitivity $\forall x, y, z \in S : (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$

Irreflexive $\forall x \in S : (x, x) \notin R$

Asymmetric $\forall x, y \in S : (x, y) \in R \Rightarrow (y, x) \notin R$

Anti-symmetric $\forall x, y \in S : (x, y) \in R \wedge (y, x) \in R \Rightarrow x = y$

Properties of Relations

The following properties are of interest when talking about relation R on set S :

Reflexive $\forall x \in S : (x, x) \in R$

Symmetry $\forall x, y \in S : (x, y) \in R \Rightarrow (y, x) \in R$

Transitivity $\forall x, y, z \in S : (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$

Irreflexive $\forall x \in S : (x, x) \notin R$

Asymmetric $\forall x, y \in S : (x, y) \in R \Rightarrow (y, x) \notin R$

Anti-symmetric $\forall x, y \in S : (x, y) \in R \wedge (y, x) \in R \Rightarrow x = y$

Don't think that failing one property implies its "opposite"!

Equivalence relation and partitions

- R is an **equivalence relation** on A if it is

Equivalence relation and partitions

- R is an **equivalence relation** on A if it is

Equivalence relation and partitions

- R is an **equivalence relation** on A if it is symmetric, reflexive and transitive.
- Intuitively, this means we have isolated "nests" of connected elements.

Equivalence relation and partitions

- R is an **equivalence relation** on A if it is symmetric, reflexive and transitive.
- Intuitively, this means we have isolated "nests" of connected elements.
- The elements of these nests can be collected into sets, forming **partitions** of S . These partitions are called equivalence classes and are written as $[a] = R(a)$.

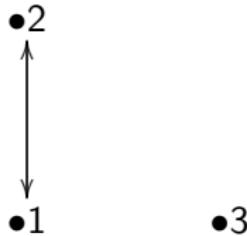
Equivalence relation and partitions

- R is an **equivalence relation** on A if it is symmetric, reflexive and transitive.
- Intuitively, this means we have isolated "nests" of connected elements.
- The elements of these nests can be collected into sets, forming **partitions** of S . These partitions are called equivalence classes and are written as $[a] = R(a)$.

Equivalence relation and partitions

- R is an **equivalence relation** on A if it is symmetric, reflexive and transitive.
- Intuitively, this means we have isolated "nests" of connected elements.
- The elements of these nests can be collected into sets, forming **partitions** of S . These partitions are called equivalence classes and are written as $[a] = R(a)$.

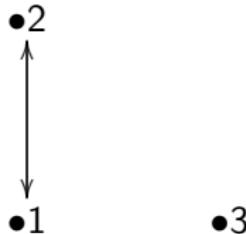
Take $S = \{1, 2, 3\}$ and a relation R on S given by the figure below



Equivalence relation and partitions

- R is an **equivalence relation** on A if it is symmetric, reflexive and transitive.
- Intuitively, this means we have isolated "nests" of connected elements.
- The elements of these nests can be collected into sets, forming **partitions** of S . These partitions are called equivalence classes and are written as $[a] = R(a)$.

Take $S = \{1, 2, 3\}$ and a relation R on S given by the figure below

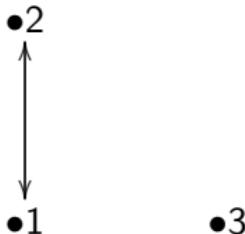


- ① You can see from R that S is partitioned into two subsets: $\{1, 2\}$ and $\{3\}$
- ② This implies we have equivalence classes $[1] = [2]$ and $[3]$.

Equivalence relation and partitions

- R is an **equivalence relation** on A if it is symmetric, reflexive and transitive.
- Intuitively, this means we have isolated "nests" of connected elements.
- The elements of these nests can be collected into sets, forming **partitions** of S . These partitions are called equivalence classes and are written as $[a] = R(a)$.

Take $S = \{1, 2, 3\}$ and a relation R on S given by the figure below



- ① You can see from R that S is partitioned into two subsets: $\{1, 2\}$ and $\{3\}$
- ② This implies we have equivalence classes $[1] = [2]$ and $[3]$.

Note that the self-loops in the figure are missing! This is a mistake; they are meant to be there.

Closures

The **closure** C of a relation R with respect to some property P is

Closures

The **closure** C of a relation R with respect to some property P is

- ① An extension of R , i.e. $R \subseteq C$

Closures

The **closure** C of a relation R with respect to some property P is

- ① An extension of R , i.e. $R \subseteq C$
- ② A relation that satisfies P , i.e. $P(C)$ is true

Closures

The **closure** C of a relation R with respect to some property P is

- ① An extension of R , i.e. $R \subseteq C$
- ② A relation that satisfies P , i.e. $P(C)$ is true
- ③ The smallest relation to satisfy P , i.e. $R' \subseteq C$ for all R' that satisfy P

Closures

The **closure** C of a relation R with respect to some property P is

- ① An extension of R , i.e. $R \subseteq C$
- ② A relation that satisfies P , i.e. $P(C)$ is true
- ③ The smallest relation to satisfy P , i.e. $R' \subseteq C$ for all R' that satisfy P

Some common closures:

Closures

The **closure** C of a relation R with respect to some property P is

- ① An extension of R , i.e. $R \subseteq C$
- ② A relation that satisfies P , i.e. $P(C)$ is true
- ③ The smallest relation to satisfy P , i.e. $R' \subseteq C$ for all R' that satisfy P

Some common closures:

Reflexive closure $r(R) = R \cup I_A$

Closures

The **closure** C of a relation R with respect to some property P is

- ① An extension of R , i.e. $R \subseteq C$
- ② A relation that satisfies P , i.e. $P(C)$ is true
- ③ The smallest relation to satisfy P , i.e. $R' \subseteq C$ for all R' that satisfy P

Some common closures:

Reflexive closure $r(R) = R \cup I_A$

Symmetric closure $s(R) = R \cup R^{-1}$

Closures

The **closure** C of a relation R with respect to some property P is

- ① An extension of R , i.e. $R \subseteq C$
- ② A relation that satisfies P , i.e. $P(C)$ is true
- ③ The smallest relation to satisfy P , i.e. $R' \subseteq C$ for all R' that satisfy P

Some common closures:

Reflexive closure $r(R) = R \cup I_A$

Symmetric closure $s(R) = R \cup R^{-1}$

Transitive closure $t(R) = R^\infty = R \cup R^2 \cup R^3 \cup \dots$

Warshall's Algorithm

This is an algorithm to find $t(R)$, i.e. the transitive closure for some relation R . Usually this is a very tedious job, but using M_R it will only take a couple of steps!

Warshall's Algorithm

This is an algorithm to find $t(R)$, i.e. the transitive closure for some relation R . Usually this is a very tedious job, but using M_R it will only take a couple of steps!

Procedure

- ① Let $W_0 = M_R$. Let $k = 0$.

Warshall's Algorithm

This is an algorithm to find $t(R)$, i.e. the transitive closure for some relation R . Usually this is a very tedious job, but using M_R it will only take a couple of steps!

Procedure

- ① Let $W_0 = M_R$. Let $k = 0$.
- ② Transfer all **1s** in W_k to W_{k+1} .

Warshall's Algorithm

This is an algorithm to find $t(R)$, i.e. the transitive closure for some relation R . Usually this is a very tedious job, but using M_R it will only take a couple of steps!

Procedure

- ① Let $W_0 = M_R$. Let $k = 0$.
- ② Transfer all **1s** in W_k to W_{k+1} .
- ③ Let **I** be the set of all rows in column k with a 1.

Warshall's Algorithm

This is an algorithm to find $t(R)$, i.e. the transitive closure for some relation R . Usually this is a very tedious job, but using M_R it will only take a couple of steps!

Procedure

- ① Let $W_0 = M_R$. Let $k = 0$.
- ② Transfer all **1s** in W_k to W_{k+1} .
- ③ Let **I** be the set of all rows in column k with a 1.
- ④ Let **J** be the set of all columns in row k with a 1.

Warshall's Algorithm

This is an algorithm to find $t(R)$, i.e. the transitive closure for some relation R . Usually this is a very tedious job, but using M_R it will only take a couple of steps!

Procedure

- ① Let $W_0 = M_R$. Let $k = 0$.
- ② Transfer all **1s** in W_k to W_{k+1} .
- ③ Let **I** be the set of all rows in column k with a 1.
- ④ Let **J** be the set of all columns in row k with a 1.
- ⑤ Put **1** in $W_{k+1}(i,j)$ if $(i,j) \in I \times J$.

Warshall's Algorithm

This is an algorithm to find $t(R)$, i.e. the transitive closure for some relation R . Usually this is a very tedious job, but using M_R it will only take a couple of steps!

Procedure

- ① Let $W_0 = M_R$. Let $k = 0$.
- ② Transfer all **1s** in W_k to W_{k+1} .
- ③ Let **I** be the set of all rows in column k with a 1.
- ④ Let **J** be the set of all columns in row k with a 1.
- ⑤ Put **1** in $W_{k+1}(i,j)$ if $(i,j) \in I \times J$.
- ⑥ Repeat steps 2 - 5 for $1 \leq k \leq |S|$.

Warshall - Example

- $W_0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \vdots$
•
•

Warshall - Example

$$\bullet \quad W_0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \dots$$
$$\bullet \quad W_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \dots$$
$$\dots$$

Warshall - Example

$$\bullet W_0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

So (2, 2) is added to W_1

$$\bullet W_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

•

Warshall - Example

$$\bullet \mathbf{W}_0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet \mathbf{W}_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \vdots$$

$$\bullet \mathbf{W}_2 = \begin{pmatrix} 0 & \textcolor{teal}{1} & 0 & 0 \\ \textcolor{teal}{1} & \textcolor{teal}{1} & \textcolor{teal}{1} & 0 \\ 0 & 0 & 0 & \textcolor{teal}{1} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Warshall - Example

$$\bullet W_0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet W_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \dots \quad \dots$$

So (1, 1), (1, 2), (1, 3),
(2, 1), (2, 2), (2, 3) are
added to W_2

$$\bullet W_2 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Warshall - Example

$$\bullet \mathbf{W}_0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet \mathbf{W}_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet \mathbf{W}_2 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet \mathbf{W}_3 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

•

•

Warshall - Example

$$\bullet W_0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet W_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet W_2 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet W_3 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

•
•

So (1, 4), (2, 4) are added to W_3

Warshall - Example

$$\bullet \mathbf{W}_0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet \mathbf{W}_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet \mathbf{W}_2 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet \mathbf{W}_3 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet \mathbf{W}_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

•

Warshall - Example

$$\bullet W_0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet W_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet W_2 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet W_3 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

So no pairs are added to
 W_4

$$\bullet W_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

•

Warshall - Example

$$\bullet \mathbf{W}_0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet \mathbf{W}_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet \mathbf{W}_2 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet \mathbf{W}_3 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet \mathbf{W}_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bullet t(R) = M_{R^\infty} = \mathbf{W}_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Properties

- Everywhere defined: $f : A \rightarrow B$ is everywhere defined if $\text{Dom}(f) = A$.
- Injective: Maps elements of the domain uniquely.
- Surjective (onto) : The range of f is its codomain.
- Bijective: Total, injective, surjective. $f : A \rightarrow B$ all elements of A to all elements of B , uniquely. This means that $|A| = |B|$.

Growth of Functions

- Let f and g be functions whose domains are subsets of \mathbb{Z}^+ .

Growth of Functions

- Let f and g be functions whose domains are subsets of \mathbb{Z}^+ .
- f is big-Oh of g ($f = O(g)$ or $f \in O(g)$) if there exist constants c and k such $|f(n)| \leq c \times |g(n)|$ for all $n \geq k$.

Growth of Functions - Example Exercise

- Let $f(n) = n^5$ and $g(n) = 5^n$. Prove that f is $\mathcal{O}(g)$ and that g is not $\mathcal{O}(f)$.
- f is $\mathcal{O}(g)$ if we can find constants c and k such that $|f(n)| \leq c * |g(n)|$ for all $n \geq k$.

Growth of Functions - Example Exercise

- Let $f(n) = n^5$ and $g(n) = 5^n$. Prove that f is $\mathcal{O}(g)$ and that g is not $\mathcal{O}(f)$.
- f is $\mathcal{O}(g)$ if we can find constants c and k such that

$$|f(n)| \leq c * |g(n)| \text{ for all } n \geq k$$

We observe that:

$$\forall n > 20 : |f(n)| = |n^5| = n^5 \leq 1 \cdot |5^n|$$

Hence f is $\mathcal{O}(g)$.

Growth of Functions - Example Exercise

- Let $f(n) = n^5$ and $g(n) = 5^n$. Prove that f is $\mathcal{O}(g)$ and that g is not $\mathcal{O}(f)$.
- Suppose that g is $\mathcal{O}(f)$. Then there must be constants c, k such that:

$$|g(n)| \leq c * |f(n)| \text{ for all } n \geq k$$

.

Growth of Functions - Example Exercise

- Let $f(n) = n^5$ and $g(n) = 5^n$. Prove that f is $\mathcal{O}(g)$ and that g is not $\mathcal{O}(f)$.
- Suppose that g is $\mathcal{O}(f)$. Then there must be constants c, k such that:

$$|g(n)| \leq c * |f(n)| \text{ for all } n \geq k$$

- Take $m \geq \max(20, k)$ and $d \geq \max(20, c)$. Then $md \geq k$, but we also have:

$$|g(md)| = 5^{md} = 5^m \cdot 5^d \geq m^5 d^6 > cm^5 \cdot d^5 \geq c(md)^5 = c|f(md)|$$

This is a contradiction with the initial assumption. Hence g is not $\mathcal{O}(f)$.



Important Justifications

- We (somewhat implicitly) utilize $\forall m \geq 20 : 5^m \geq m^5$ which we proved earlier.
- We implicitly utilized $\forall m \geq 400 : 5^d \geq d^6$. In the exam, you can say you can prove this similarly to $n^5 = \mathcal{O}(5^n)$ proof.
- Note that $d^6 \geq d^5$ as $d \geq \max(20, c)$.

Posets

Often we want to compare elements in a set to each other through some kind of ordering. The next couple of slides will talk about different levels of ordering

Posets

Often we want to compare elements in a set to each other through some kind of ordering. The next couple of slides will talk about different levels of ordering

Poset

We call (S, R) a **poset** when S is a set and $R : S \rightarrow S$ is a relation on S that is

- ① Reflexive
- ② Transitive
- ③ Anti-symmetric

Some examples:

Posets

Often we want to compare elements in a set to each other through some kind of ordering. The next couple of slides will talk about different levels of ordering

Poset

We call (S, R) a **poset** when S is a set and $R : S \rightarrow S$ is a relation on S that is

- ① Reflexive
- ② Transitive
- ③ Anti-symmetric

Some examples:

- (\mathbb{Z}, \leq)
- $(\mathcal{P}(S), \subseteq)$
- $(D_{20}, |)$

Posets (definitions)

Some definitions related to posets (S, R) :

Posets (definitions)

Some definitions related to posets (S, R) :

Weak vs Strict Whether R is reflexive (weak) or irreflexive (strict)

Example: \subseteq / \leq are weak, but $\subset / <$ are strict

Posets (definitions)

Some definitions related to posets (S, R) :

Weak vs Strict Whether R is reflexive (weak) or irreflexive (strict)

Example: \subseteq / \leq are weak, but $\subset / <$ are strict

Dual The *dual* of (S, R) is (S, R^{-1}) , i.e. taking the inverse relation

Example: the dual of (S, \leq) is (S, \geq)

Posets (definitions)

Some definitions related to posets (S, R) :

Weak vs Strict Whether R is reflexive (weak) or irreflexive (strict)

Example: \subseteq / \leq are weak, but $\subset / <$ are strict

Dual The *dual* of (S, R) is (S, R^{-1}) , i.e. taking the inverse relation

Example: the dual of (S, \leq) is (S, \geq)

Connected All elements of S are comparable

Formally: $\forall x, y \in S : x \neq y \Rightarrow xRy \vee yRx$

Posets (definitions)

Some definitions related to posets (S, R) :

Weak vs Strict Whether R is reflexive (weak) or irreflexive (strict)

Example: \subseteq / \leq are weak, but $\subset / <$ are strict

Dual The *dual* of (S, R) is (S, R^{-1}) , i.e. taking the inverse relation

Example: the dual of (S, \leq) is (S, \geq)

Connected All elements of S are comparable

Formally: $\forall x, y \in S : x \neq y \Rightarrow xRy \vee yRx$

(Strict) Total Order Poset which is connected. If it is a strict total order, then it is irreflexive instead of reflexive.

Ex. 4 - Exam 2018

- Let $S = \{a, b, c\}$. Prove or disprove that $R = (\mathcal{P}(S), \subseteq)$ is a poset.

Ex. 4 - Exam 2018

- Let $S = \{a, b, c\}$. Prove or disprove that $R = (\mathcal{P}(S), \subseteq)$ is a poset.
- Reflexive: $\forall x \in \mathcal{P}(S), x \subseteq x$
True, since each set contains itself.

Ex. 4 - Exam 2018

- Let $S = \{a, b, c\}$. Prove or disprove that $R = (\mathcal{P}(S), \subseteq)$ is a poset.
- Reflexive: $\forall x \in \mathcal{P}(S), x \subseteq x$
True, since each set contains itself.
- Anti-symmetric: $\forall (x, y) \in R \wedge (y, x) \notin R \implies x = y$
True, since if $x \subseteq y$ and $y \subseteq x$, then it must be that $x = y$.

Ex. 4 - Exam 2018

- Let $S = \{a, b, c\}$. Prove or disprove that $R = (\mathcal{P}(S), \subseteq)$ is a poset.
- Reflexive: $\forall x \in \mathcal{P}(S), x \subseteq x$
True, since each set contains itself.
- Anti-symmetric: $\forall (x, y) \in R \wedge (y, x) \notin R \implies x = y$
True, since if $x \subseteq y$ and $y \subseteq x$, then it must be that $x = y$.
- Transitive: $(x, y) \in R \wedge (y, z) \in R \implies (x, z) \in R$
True, since if $x \subseteq y$ and $y \subseteq z$, then $x \subseteq z$.

Ex. 4 - Exam 2018

- Let $S = \{a, b, c\}$. Prove or disprove that $R = (\mathcal{P}(S), \subseteq)$ is a poset.
- Reflexive: $\forall x \in \mathcal{P}(S), x \subseteq x$
True, since each set contains itself.
- Anti-symmetric: $\forall (x, y) \in R \wedge (y, x) \notin R \implies x = y$
True, since if $x \subseteq y$ and $y \subseteq x$, then it must be that $x = y$.
- Transitive: $(x, y) \in R \wedge (y, z) \in R \implies (x, z) \in R$
True, since if $x \subseteq y$ and $y \subseteq z$, then $x \subseteq z$.
- Since R is reflexive, antisymmetric and transitive, R must be a poset.

Hasse Diagrams

Drawing the relation of a poset is tedious, because since it is reflexive/transitive, there's a lot of lines to draw.

Hasse Diagrams

Drawing the relation of a poset is tedious, because since it is reflexive/transitive, there's a lot of lines to draw.

That's why it's nice to instead represent the relation with a **Hasse diagram**

Hasse diagram

A simplified graph of a relation which removes self-loops or transitive arcs from a digraph

Hasse Diagrams

Drawing the relation of a poset is tedious, because since it is reflexive/transitive, there's a lot of lines to draw.

That's why it's nice to instead represent the relation with a **Hasse diagram**

Hasse diagram

A simplified graph of a relation which removes self-loops or transitive arcs from a digraph

How to draw a Hasse Diagram:

- ① Draw (S, R) as a digraph with the nodes the elements of S and the edges the pairs defined in R

Hasse Diagrams

Drawing the relation of a poset is tedious, because since it is reflexive/transitive, there's a lot of lines to draw.

That's why it's nice to instead represent the relation with a **Hasse diagram**

Hasse diagram

A simplified graph of a relation which removes self-loops or transitive arcs from a digraph

How to draw a Hasse Diagram:

- ① Draw (S, R) as a digraph with the nodes the elements of S and the edges the pairs defined in R
- ② Remove self-loops

Hasse Diagrams

Drawing the relation of a poset is tedious, because since it is reflexive/transitive, there's a lot of lines to draw.

That's why it's nice to instead represent the relation with a **Hasse diagram**

Hasse diagram

A simplified graph of a relation which removes self-loops or transitive arcs from a digraph

How to draw a Hasse Diagram:

- ① Draw (S, R) as a digraph with the nodes the elements of S and the edges the pairs defined in R
- ② Remove self-loops
- ③ Remove implied transitive pairs

Hasse Diagrams

Drawing the relation of a poset is tedious, because since it is reflexive/transitive, there's a lot of lines to draw.

That's why it's nice to instead represent the relation with a **Hasse diagram**

Hasse diagram

A simplified graph of a relation which removes self-loops or transitive arcs from a digraph

How to draw a Hasse Diagram:

- ① Draw (S, R) as a digraph with the nodes the elements of S and the edges the pairs defined in R
- ② Remove self-loops
- ③ Remove implied transitive pairs
- ④ Arrange nodes such that they are ordered in height, e.g. $a \leq b$ implies a is below b in the Hasse diagram

Hasse Diagrams

Drawing the relation of a poset is tedious, because since it is reflexive/transitive, there's a lot of lines to draw.

That's why it's nice to instead represent the relation with a **Hasse diagram**

Hasse diagram

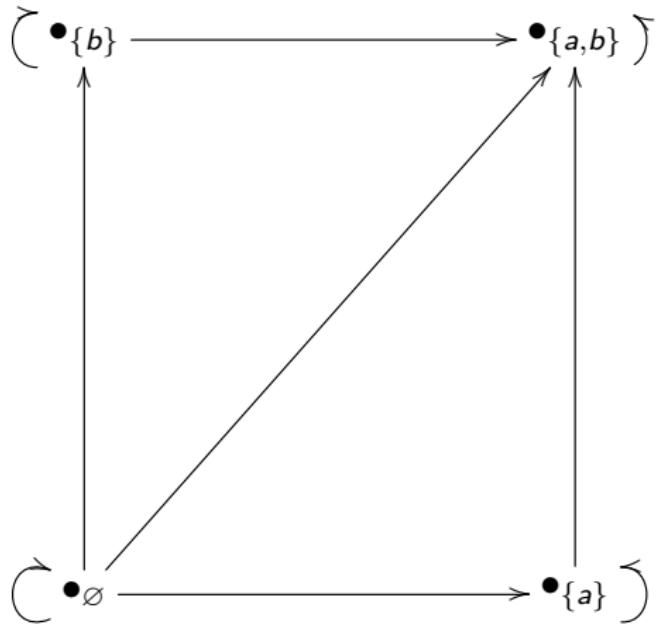
A simplified graph of a relation which removes self-loops or transitive arcs from a digraph

How to draw a Hasse Diagram:

- ① Draw (S, R) as a digraph with the nodes the elements of S and the edges the pairs defined in R
- ② Remove self-loops
- ③ Remove implied transitive pairs
- ④ Arrange nodes such that they are ordered in height, e.g. $a \leq b$ implies a is below b in the Hasse diagram
- ⑤ Remove the arrow heads

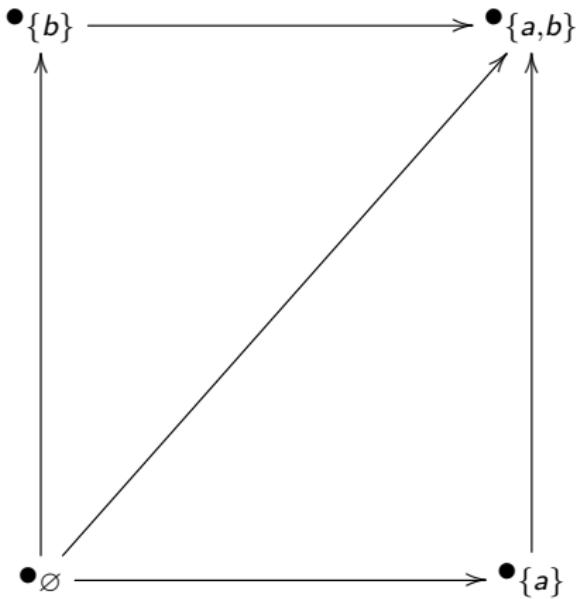
Hasse Diagrams (example)

Draw (S, R) as a digraph with the nodes the elements of S and the edges the pairs defined in R



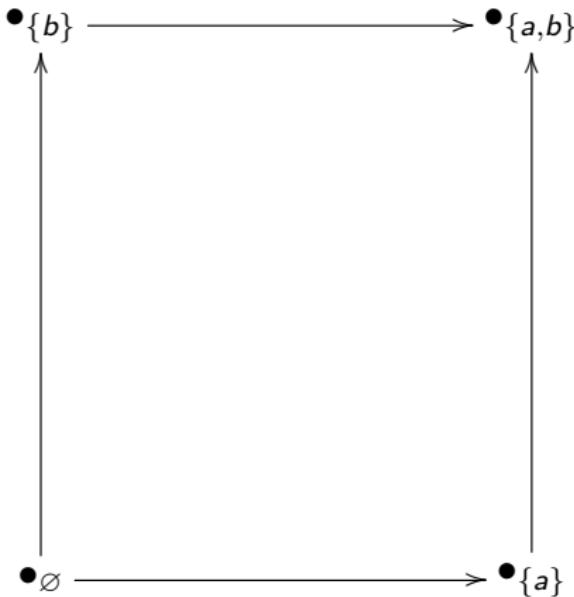
Hasse Diagrams (example)

Removed self-loops



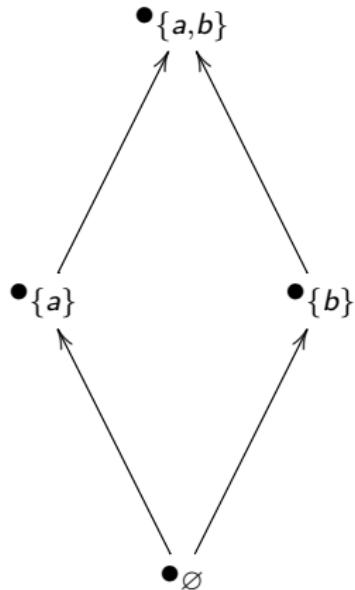
Hasse Diagrams (example)

Removed implied transitive pairs



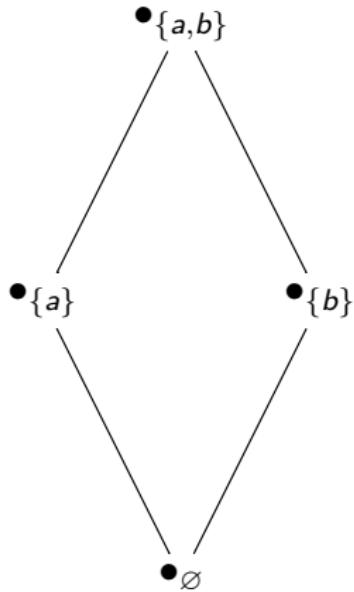
Hasse Diagrams (example)

Arranged nodes such that they are ordered in height, e.g. $a \leq b$ implies a is below b in the Hasse diagram



Hasse Diagrams (example)

Removed arrow heads



Extremal elements of posets (part I)

Now that we know about Hasse diagrams, it's time to talk about **extremal elements** of posets and how to find them using the Hasse diagrams.

Suppose again we have a poset (S, \leq)

Extremal elements of posets (part I)

Now that we know about Hasse diagrams, it's time to talk about **extremal elements** of posets and how to find them using the Hasse diagrams.

Suppose again we have a poset (S, \leq)

Minimal element(s) of S

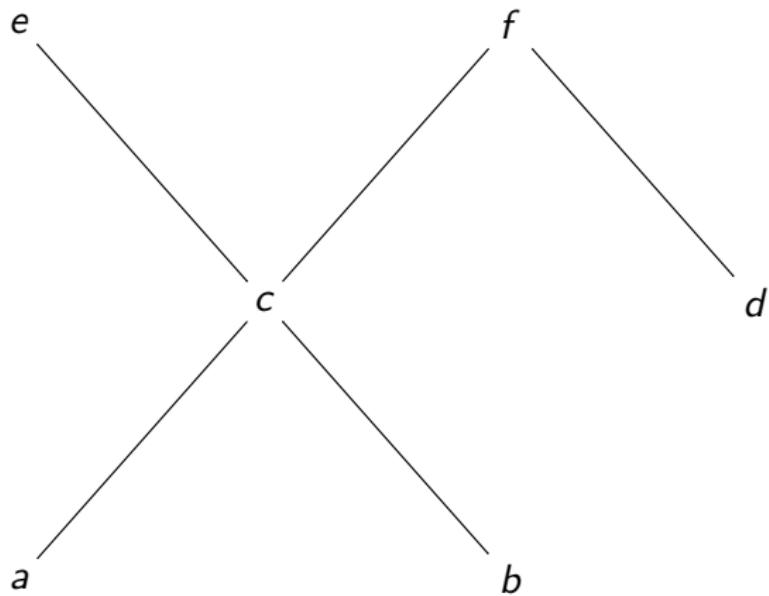
Element $z \in S$ is **minimal** if there is no $s \in S$ such that $s < z$

Maximal element(s) of S

Element $z \in S$ is **maximal** if there is no $s \in S$ such that $s > z$

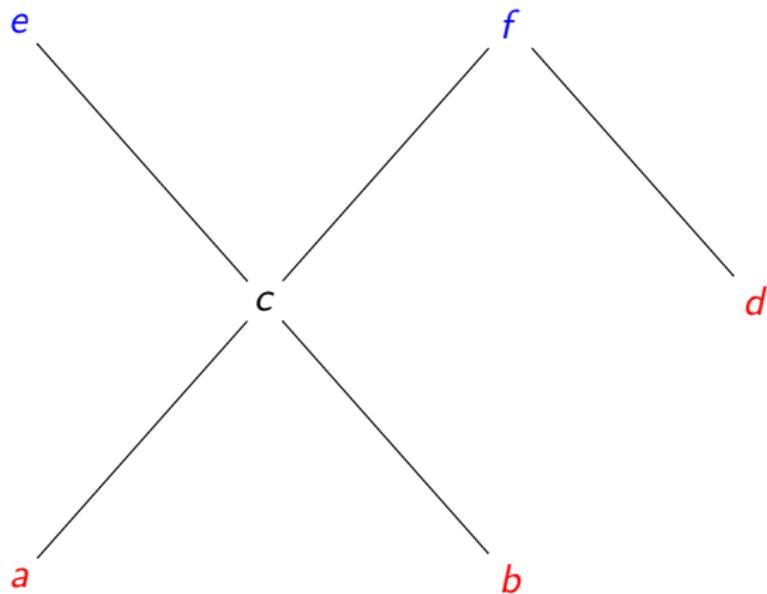
Extremal elements of posets (Example)

What are the **min** and **max** elements of the poset?



Extremal elements of posets (Example)

Extremal elements of posets (Example)



Extremal elements of posets (part II)

Note that minimal/maximal elements do not need to be comparable to all elements in S .

Extremal elements of posets (part II)

Note that minimal/maximal elements do not need to be comparable to all elements in S . For the following definitions, we *do* need to be able to compare against all elements:

Least element of S

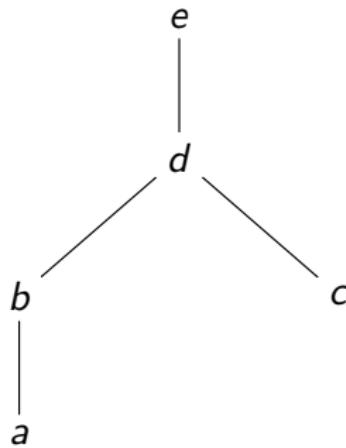
Element $l \in S$ is the **least** element of S if $l \leq s$ for all $s \in S$

Greatest element of S

Element $g \in S$ is the **greatest** element of S if $g \geq s$ for all $s \in S$

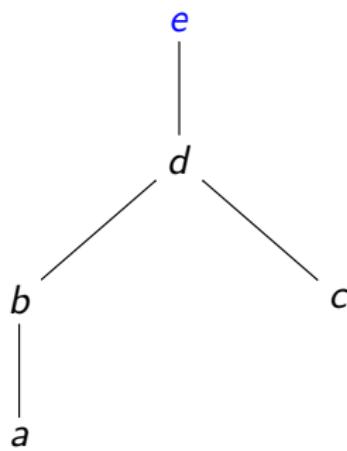
Extremal elements of posets - Greatest/least example

What are the **least** and **greatest** element of the poset?



Extremal elements of posets - Greatest/least example

Extremal elements of posets - Greatest/least example



Note there is no least element!

This is because the only candidates are the minimal elements a and c . Since they are not comparable, i.e. $a \not\leq c$ and $c \not\leq a$, there is no least.

The greatest element does exist and is e .

Extremal elements of posets (part III)

Let's take our poset (S, \leq) again.

Sometimes we are only interested in a subset $B \subseteq S$ to find the extremals for.

Extremal elements of posets (part III)

Let's take our poset (S, \leq) again.

Sometimes we are only interested in a subset $B \subseteq S$ to find the extremals for.

Lower bounds of B

Element $l \in S$ is a **lower bound** of $B \subseteq S$ if $l \leq b$ for all $b \in B$

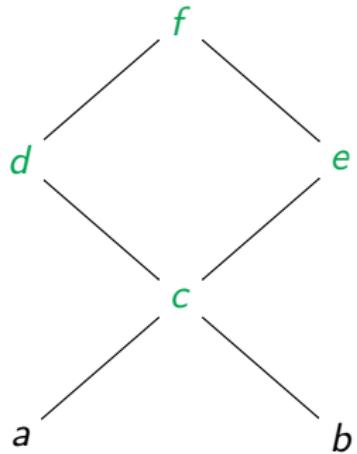
Greatest element of B

Element $u \in S$ is a **upper bound** of $B \subseteq S$ if $u \geq b$ for all $b \in B$

Note the definition are very similar to that of the least/greatest element! In fact if $B = S$, finding a lower (upper) bound is the same as finding the least (greatest) element.

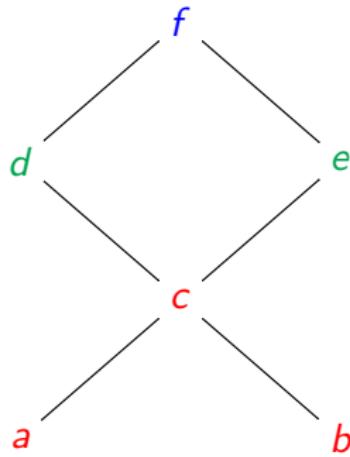
Extremal elements of posets - Bounds example

What are the **lower** and **upper** bounds of B?



Extremal elements of posets - Bounds example

Extremal elements of posets - Bounds example



Note the lower/upper bounds can be part of B !

Extremal elements of posets (part IV)

Let's take our poset (S, \leq) and $B \subseteq S$ again.

Finally, we can wonder about the greatest/least elements among the lower/upper bounds we have found.

Extremal elements of posets (part IV)

Let's take our poset (S, \leq) and $B \subseteq S$ again.

Finally, we can wonder about the greatest/least elements among the lower/upper bounds we have found.

Greatest lower bound of B

Element $x = \text{glb}(B) \in LB$ is the **greatest lower bound** of $B \subseteq S$ if $x \geq l$ for all $l \in LB$.

Here LB is the set of all lower bounds of B .

Least upper bound of B

Element $x = \text{lub}(B) \in UB$ is the **least upper bound** of $B \subseteq S$ if $x \leq u$ for all $u \in UB$

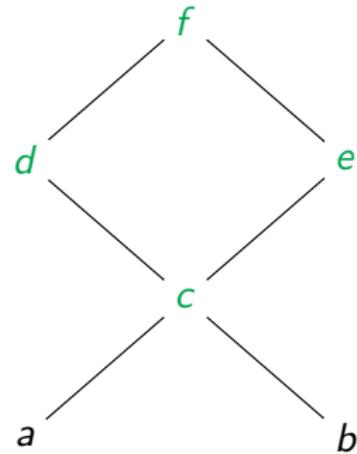
Here UB is the set of all upper bounds of B .

Extremal elements of posets - GLB/LUB example

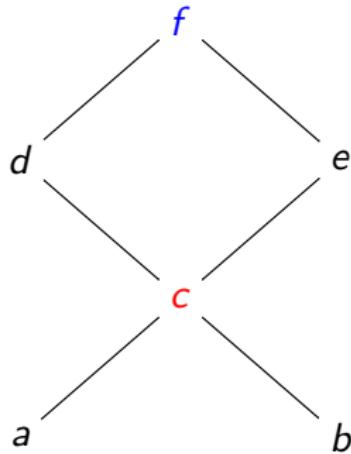
What are the **GLB** and **LUB** bounds of **B**?

Extremal elements of posets - GLB/LUB example

What are the **GLB** and **LUB** bounds of B?



Extremal elements of posets - GLB/LUB example



Topological Sorting

Sometimes it is convenient to transform partial order \leq into a total order \prec , i.e. to be able to compare all elements in the set S . This is called a **topological sorting**, which should preserve the ordering of \leq .

Topological Sorting

Sometimes it is convenient to transform partial order \leq into a total order \prec , i.e. to be able to compare all elements in the set S .

This is called a **topological sorting**, which should preserve the ordering of \leq . To create such an ordering, you need to

- ① Create the Hasse diagram of \leq (or otherwise make clear for yourself how \leq orders the elements of S)

Topological Sorting

Sometimes it is convenient to transform partial order \leq into a total order \prec , i.e. to be able to compare all elements in the set S .

This is called a **topological sorting**, which should preserve the ordering of \leq . To create such an ordering, you need to

- ① Create the Hasse diagram of \leq (or otherwise make clear for yourself how \leq orders the elements of S)
- ② Take (one of) the minimal element(s) of the diagram, say x . This element will be ordered below the remaining elements in the Hasse diagram, i.e.

$$x \prec (\text{remaining})$$

Now remove x and its connections from the diagram

Topological Sorting

Sometimes it is convenient to transform partial order \leq into a total order \prec , i.e. to be able to compare all elements in the set S .

This is called a **topological sorting**, which should preserve the ordering of \leq . To create such an ordering, you need to

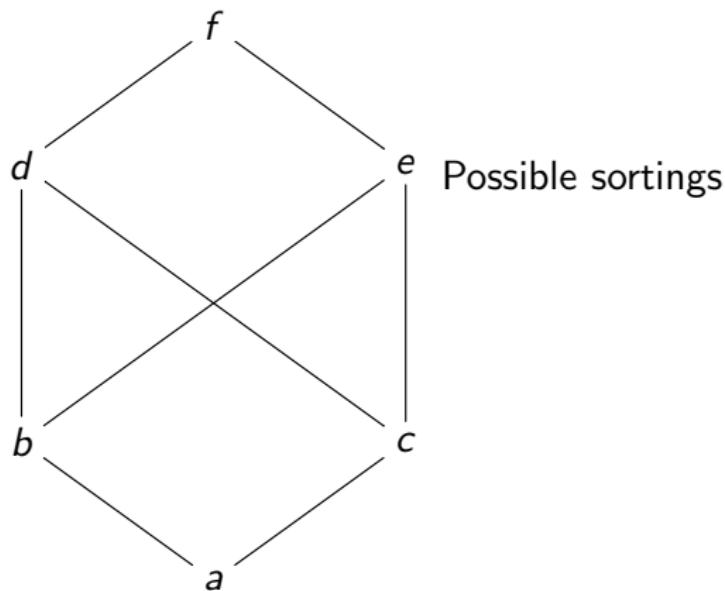
- ① Create the Hasse diagram of \leq (or otherwise make clear for yourself how \leq orders the elements of S)
- ② Take (one of) the minimal element(s) of the diagram, say x . This element will be ordered below the remaining elements in the Hasse diagram, i.e.

$$x \prec (\text{remaining})$$

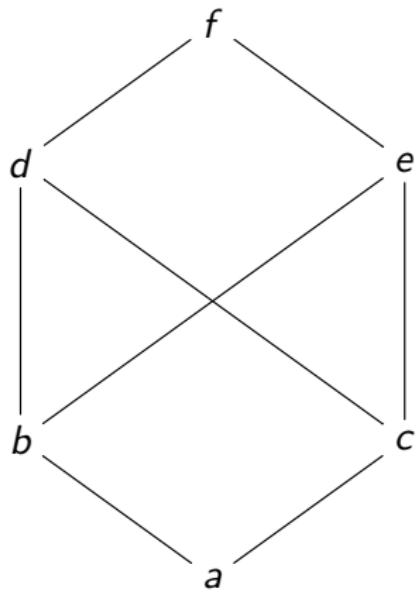
Now remove x and its connections from the diagram

- ③ Repeat step 2 until the diagram is empty

Topological Sorting (example)



Topological Sorting (example)



Possible sortings

- ① $a \prec b \prec c \prec d \prec e \prec f$
- ② $a \prec b \prec c \prec e \prec d \prec f$
- ③ $a \prec c \prec b \prec d \prec e \prec f$
- ④ $a \prec c \prec b \prec e \prec d \prec f$

Product / Lexographical Partial Orders

What if you want to combine multiple partial orders into a single one?

Product / Lexographical Partial Orders

What if you want to combine multiple partial orders into a single one?

- Given two partial orders \leq_1, \leq_2 , we define the **product partial order** \preceq such that

$$(a, b) \preceq (a', b') \iff (a \leq_1 a') \wedge (b \leq_2 b')$$

Product / Lexographical Partial Orders

What if you want to combine multiple partial orders into a single one?

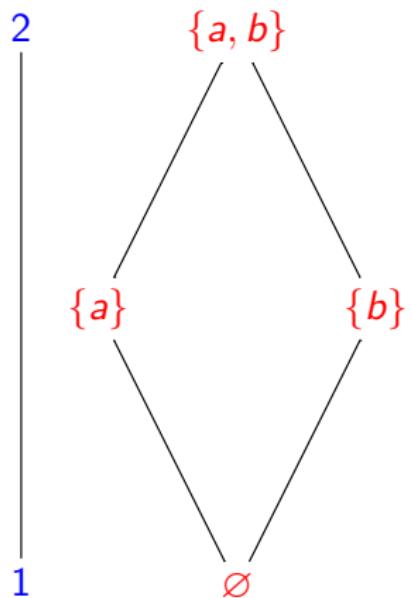
- Given two partial orders \leq_1, \leq_2 , we define the **product partial order** \preceq such that

$$(a, b) \preceq (a', b') \iff (a \leq_1 a') \wedge (b \leq_2 b')$$

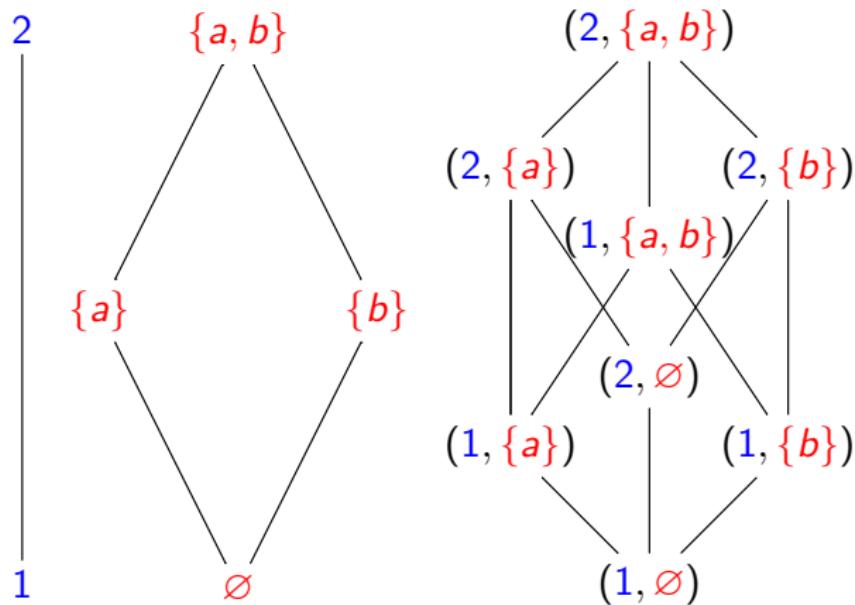
- A **lexicographic order** \prec is a type of product partial order, except that its first component dominates. For example

$$(a, b) \prec (a', b') \iff (a \leq_A a') \vee (a = a' \wedge b \leq_B b')$$

Product Partial Orders (example)



Product Partial Orders (example)



Lattices

- A lattice is a poset S where for every two elements $a, b \in S$ there is a Greatest Lower Bound and a Least Upper Bound in S .
- Isomorphic posets must either both be lattices or neither be lattices.

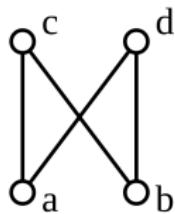
Lattices

- A lattice is a poset S where for every two elements $a, b \in S$ there is a Greatest Lower Bound and a Least Upper Bound in S .
- A sublattice of a lattice is a subset of the lattice such that all meets and joins of the subset lie within it.
- Isomorphic posets must either both be lattices or neither be lattices.

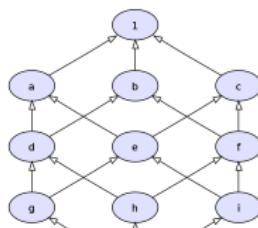
Lattices

- A bounded lattice is a lattice that has a greatest and a least element. Every finite lattice is bounded.
- A complete lattice is a lattice (L, \leq) if for all (possibly infinite) $A \subseteq L$ the join $\bigvee A$ and meet $\bigwedge A$ exist in L . Every finite lattice is complete.
- A complemented lattice is a lattice such that for any element $a \in L$, there exists at least one *complement* $a' \in L$ such that $a \wedge a' = 0$ and $a \vee a' = I$.

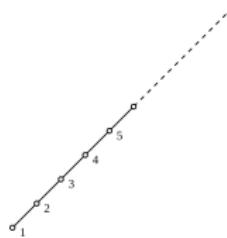
Lattices: Is this a lattice?



A

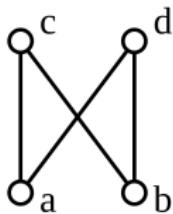


B

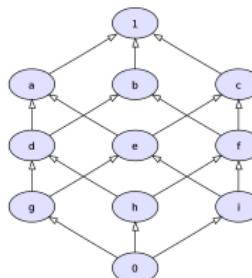


C

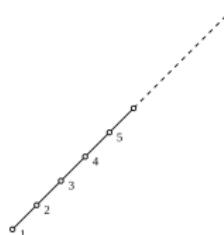
Lattices: Is this a lattice?



A



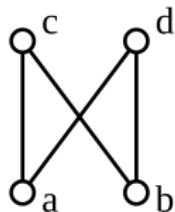
B



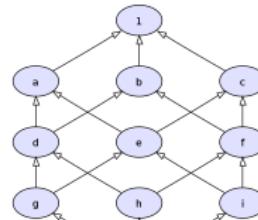
C

- A is clearly not a lattice as c and d have no common upper bound.

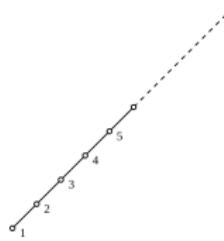
Lattices: Is this a lattice?



A



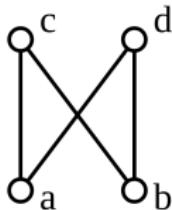
B



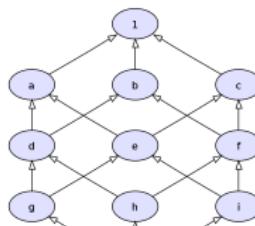
C

- A is clearly not a lattice as c and d have no common upper bound.
- B is surprisingly not a lattice: a and b have multiple common lower bounds: $d, g, h, i, 0$. However none of them is the greatest lower bound.

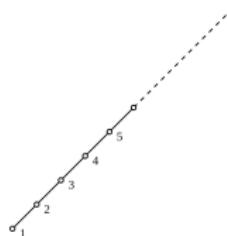
Lattices: Is this a lattice?



A



B



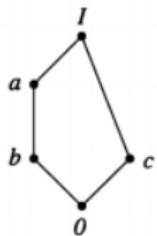
C

- A is clearly not a lattice as c and d have no common upper bound.
- B is surprisingly not a lattice: a and b have multiple common lower bounds: $d, g, h, i, 0$. However none of them is the greatest lower bound.
- C is a lattice: the least upper bound of a, b is $\max(a, b)$ and the greatest lower bound is $\min(a, b)$.

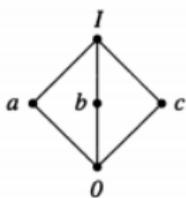
Distributive Lattices

- A lattice is distributive if \vee is distributive over \wedge and \wedge is distributive over \vee .
- A lattice is non-distributive iff it contains a sublattice isomorphic to the lattices below.

N_5 (pentagon)



M_3 (diamond)

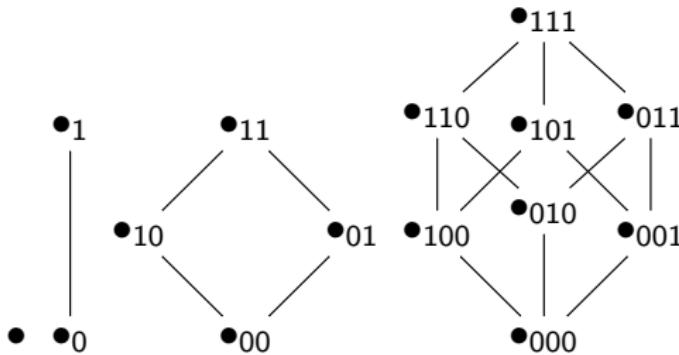


Proving L is a lattice

- Prove that L is a poset
- Choose a candidate binary function for the join and meet.
Usually, your lattice is defined as a partial order on a familiar function, and there should be another familiar function that would apply.
- Prove that those functions are a LUB / GLB.
 - LUB: Show that the function gives a lower bound. Show that it is the greatest lower bound.
 - GLB: Show that the function gives a upper bound. Show that it is the least of any upper bound.

Boolean Algebras: (B_n, \leq)

The Partial Order is a set of sequences of zeroes and ones, of length n . The order of these sequences is isomorphic to the power-set of a set of length n .



An easy way to remember how to draw these is that each level has more "1"s and each vertex in B_n has degree n .

Boolean Polynomials

- Boolean polynomials are also called Boolean expressions.
- A Boolean polynomial $p(x_1, x_2, \dots, x_n)$ is defined recursively.
- Boolean polynomials can be rewritten by applying the properties of Boolean algebras (which should look eerily similar to your traditional properties on logical operators).

Boolean Polynomials

- Boolean polynomials are also called Boolean expressions.
- A Boolean polynomial $p(x_1, x_2, \dots, x_n)$ is defined recursively.
- Boolean polynomials can be rewritten by applying the properties of Boolean algebras (which should look eerily similar to your traditional properties on logical operators).

An example of Boolean polynomial equivalence:

To prove: $(x \vee y) \wedge (x' \vee y) = y$

Boolean Polynomials

- Boolean polynomials are also called Boolean expressions.
- A Boolean polynomial $p(x_1, x_2, \dots, x_n)$ is defined recursively.
- Boolean polynomials can be rewritten by applying the properties of Boolean algebras (which should look eerily similar to your traditional properties on logical operators).

An example of Boolean polynomial equivalence:

To prove: $(x \vee y) \wedge (x' \vee y) = y$

$$\begin{aligned} & (x \vee y) \wedge (x' \vee y) \\ &= (x \wedge (x' \vee y)) \vee (y \wedge (x' \vee y)) \quad \{\text{distribution}\} \end{aligned}$$

Boolean Polynomials

- Boolean polynomials are also called Boolean expressions.
- A Boolean polynomial $p(x_1, x_2, \dots, x_n)$ is defined recursively.
- Boolean polynomials can be rewritten by applying the properties of Boolean algebras (which should look eerily similar to your traditional properties on logical operators).

An example of Boolean polynomial equivalence:

To prove: $(x \vee y) \wedge (x' \vee y) = y$

$$\begin{aligned} & (x \vee y) \wedge (x' \vee y) \\ &= (x \wedge (x' \vee y)) \vee (y \wedge (x' \vee y)) \quad \{\text{distribution}\} \\ &= (x \wedge (x' \vee y)) \vee y \quad \{\text{absorption}\} \end{aligned}$$

Boolean Polynomials

- Boolean polynomials are also called Boolean expressions.
- A Boolean polynomial $p(x_1, x_2, \dots, x_n)$ is defined recursively.
- Boolean polynomials can be rewritten by applying the properties of Boolean algebras (which should look eerily similar to your traditional properties on logical operators).

An example of Boolean polynomial equivalence:

To prove: $(x \vee y) \wedge (x' \vee y) = y$

$$\begin{aligned} & (x \vee y) \wedge (x' \vee y) \\ &= (x \wedge (x' \vee y)) \vee (y \wedge (x' \vee y)) \quad \{\text{distribution}\} \\ &= (x \wedge (x' \vee y)) \vee y \quad \{\text{absorption}\} \\ &= ((x \wedge x') \vee (x \wedge y)) \vee y \quad \{\text{distribution}\} \end{aligned}$$

Boolean Polynomials

- Boolean polynomials are also called Boolean expressions.
- A Boolean polynomial $p(x_1, x_2, \dots, x_n)$ is defined recursively.
- Boolean polynomials can be rewritten by applying the properties of Boolean algebras (which should look eerily similar to your traditional properties on logical operators).

An example of Boolean polynomial equivalence:

To prove: $(x \vee y) \wedge (x' \vee y) = y$

$$\begin{aligned} & (x \vee y) \wedge (x' \vee y) \\ &= (x \wedge (x' \vee y)) \vee (y \wedge (x' \vee y)) \quad \{\text{distribution}\} \\ &= (x \wedge (x' \vee y)) \vee y \quad \{\text{absorption}\} \\ &= ((x \wedge x') \vee (x \wedge y)) \vee y \quad \{\text{distribution}\} \\ &= (0 \vee (x \wedge y)) \vee y \quad \{\text{complement}\} \end{aligned}$$

Boolean Polynomials

- Boolean polynomials are also called Boolean expressions.
- A Boolean polynomial $p(x_1, x_2, \dots, x_n)$ is defined recursively.
- Boolean polynomials can be rewritten by applying the properties of Boolean algebras (which should look eerily similar to your traditional properties on logical operators).

An example of Boolean polynomial equivalence:

To prove: $(x \vee y) \wedge (x' \vee y) = y$

$$\begin{aligned} & (x \vee y) \wedge (x' \vee y) \\ &= (x \wedge (x' \vee y)) \vee (y \wedge (x' \vee y)) \quad \{\text{distribution}\} \\ &= (x \wedge (x' \vee y)) \vee y \quad \{\text{absorption}\} \\ &= ((x \wedge x') \vee (x \wedge y)) \vee y \quad \{\text{distribution}\} \\ &= (0 \vee (x \wedge y)) \vee y \quad \{\text{complement}\} \\ &= (x \wedge y) \vee y \quad \{\text{identity}\} \end{aligned}$$

Boolean Polynomials

- Boolean polynomials are also called Boolean expressions.
- A Boolean polynomial $p(x_1, x_2, \dots, x_n)$ is defined recursively.
- Boolean polynomials can be rewritten by applying the properties of Boolean algebras (which should look eerily similar to your traditional properties on logical operators).

An example of Boolean polynomial equivalence:

To prove: $(x \vee y) \wedge (x' \vee y) = y$

$$\begin{aligned} & (x \vee y) \wedge (x' \vee y) \\ &= (x \wedge (x' \vee y)) \vee (y \wedge (x' \vee y)) \quad \{\text{distribution}\} \\ &= (x \wedge (x' \vee y)) \vee y \quad \{\text{absorption}\} \\ &= ((x \wedge x') \vee (x \wedge y)) \vee y \quad \{\text{distribution}\} \\ &= (0 \vee (x \wedge y)) \vee y \quad \{\text{complement}\} \\ &= (x \wedge y) \vee y \quad \{\text{identity}\} \\ &= y \quad \{\text{absorption}\} \end{aligned}$$



Trees

Trees, just like partial orders, are a special kind of *relation*.
They have the following properties:

Trees

Trees, just like partial orders, are a special kind of *relation*. They have the following properties:

- There are no cycles in a tree T .

Trees

Trees, just like partial orders, are a special kind of *relation*. They have the following properties:

- There are no cycles in a tree T .
- The root has an in-degree of 0 and all other vertices have an in-degree of 1.

Trees

Trees, just like partial orders, are a special kind of *relation*. They have the following properties:

- There are no cycles in a tree T .
- The root has an in-degree of 0 and all other vertices have an in-degree of 1.
- A tree is irreflexive.

Trees

Trees, just like partial orders, are a special kind of *relation*. They have the following properties:

- There are no cycles in a tree T .
- The root has an in-degree of 0 and all other vertices have an in-degree of 1.
- A tree is irreflexive.
- A tree is asymmetric.

Trees

Trees, just like partial orders, are a special kind of *relation*. They have the following properties:

- There are no cycles in a tree T .
- The root has an in-degree of 0 and all other vertices have an in-degree of 1.
- A tree is irreflexive.
- A tree is asymmetric.
- For all $a, b, c \in S$, if $(a, b) \in T$ and $(b, c) \in T$ then $(a, c) \notin T$ (non-transitivity)

n-ary Trees

- All vertices of the tree have at most n children.

n-ary Trees

- All vertices of the tree have at most n children.
- If all vertices (except the leaves) have exactly n children then the tree is a **complete n -ary tree**.

n-ary Trees

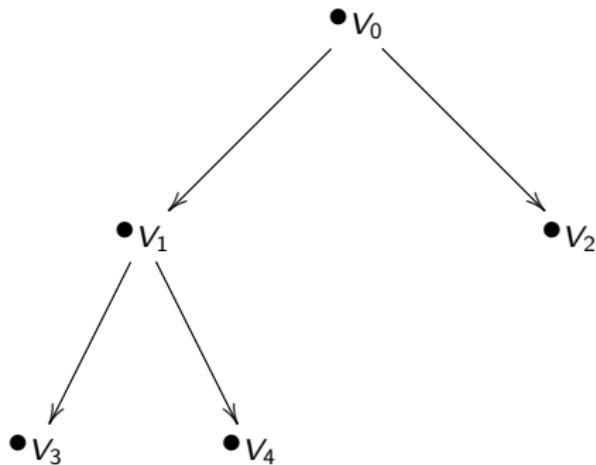
- All vertices of the tree have at most n children.
- If all vertices (except the leaves) have exactly n children then the tree is a **complete n -ary tree**.
- An example of the n -ary tree is the binary tree:

n-ary Trees

- All vertices of the tree have at most n children.
- If all vertices (except the leaves) have exactly n children then the tree is a **complete n -ary tree**.
- An example of the n -ary tree is the binary tree:

n-ary Trees

- All vertices of the tree have at most n children.
- If all vertices (except the leaves) have exactly n children then the tree is a **complete n -ary tree**.
- An example of the n -ary tree is the binary tree:



Search

To traverse trees, there are 3 common method that "search" for the nodes in a given order:

Search

To traverse trees, there are 3 common method that "search" for the nodes in a given order:

Preorder Search

- ① Visit the **root**
- ② Search its **left** subtree
- ③ Search its **right** subtree

Search

To traverse trees, there are 3 common method that "search" for the nodes in a given order:

Preorder Search

- ① Visit the **root**
- ② Search its **left** subtree
- ③ Search its **right** subtree

Inorder Search

- ① Search the **left** subtree
- ② Visit the **root**
- ③ Search the **right** subtree

Search

To traverse trees, there are 3 common method that "search" for the nodes in a given order:

Preorder Search

- ① Visit the **root**
- ② Search its **left** subtree
- ③ Search its **right** subtree

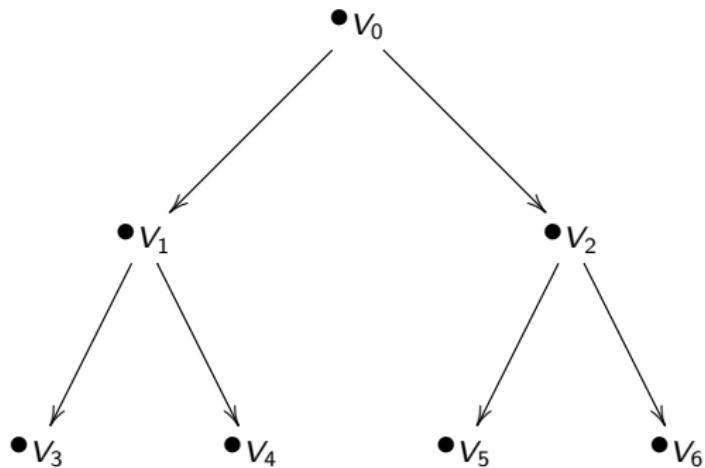
Inorder Search

- ① Search the **left** subtree
- ② Visit the **root**
- ③ Search the **right** subtree

Postorder Search

- ① Search the **left** subtree
- ② Search the **right** subtree
- ③ Visit the **root**

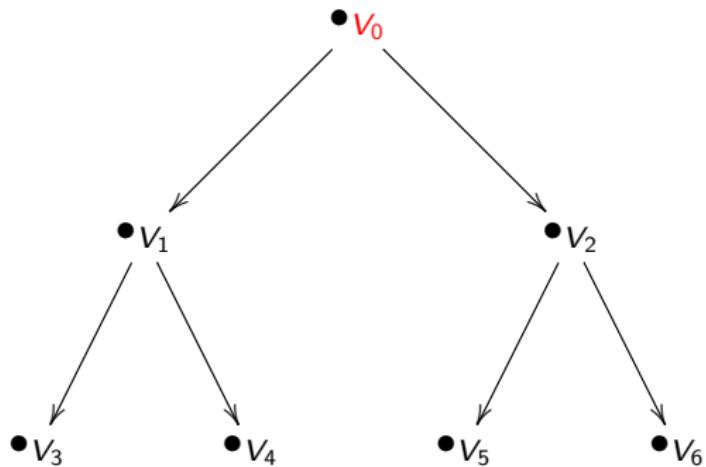
Preorder search example



Visited: \emptyset

Preorder search Example

Preorder search Example

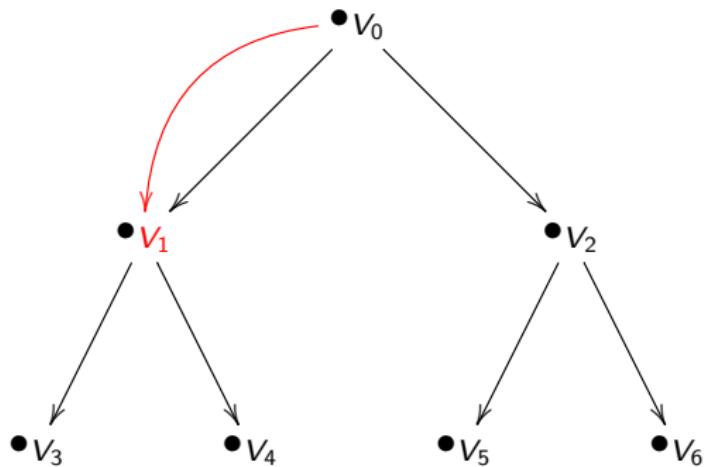


Visited: $\{V_0\}$

Preorder search Example

Preorder search Example

Preorder search Example



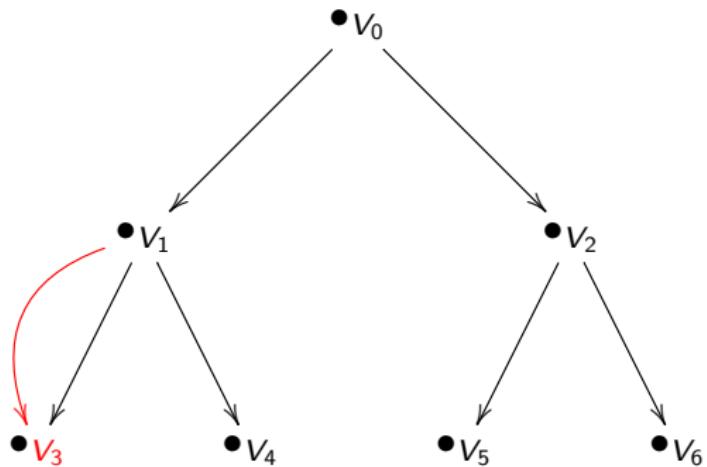
Visited: $\{V_0, V_1\}$

Preorder search Example

Preorder search Example

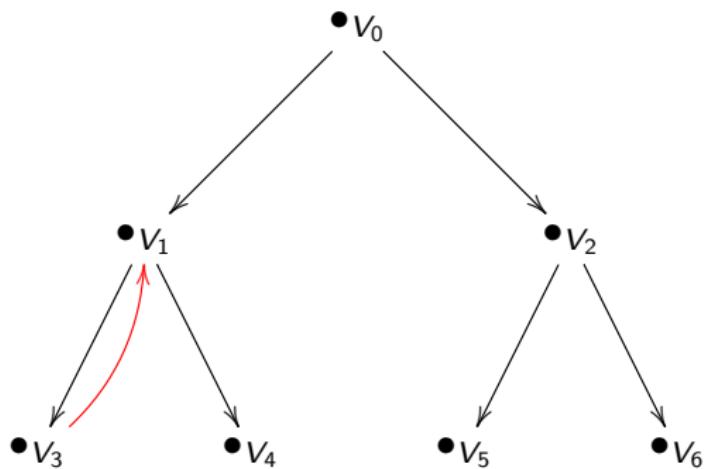
Preorder search Example

Preorder search Example



Visited: $\{V_0, V_1, V_3\}$

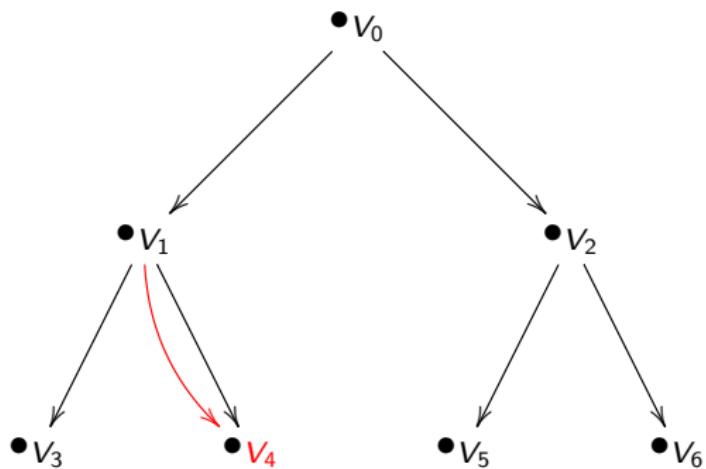
Preorder search Example



Visited: $\{V_0, V_1, V_3\}$

Preorder search Example

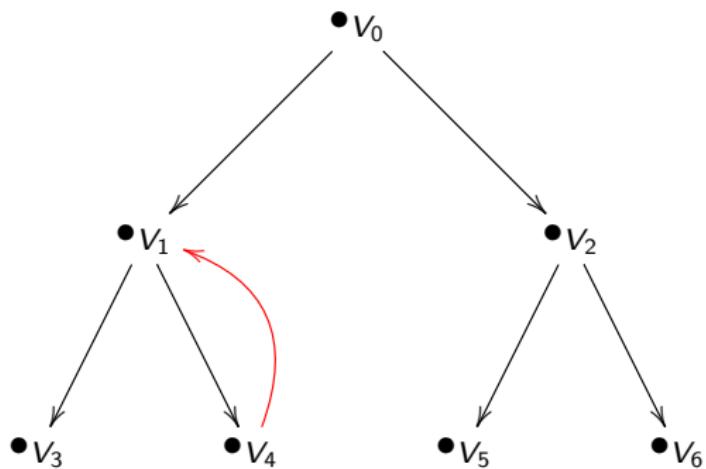
Preorder search Example



Visited: $\{V_0, V_1, V_3, V_4\}$

Preorder search Example

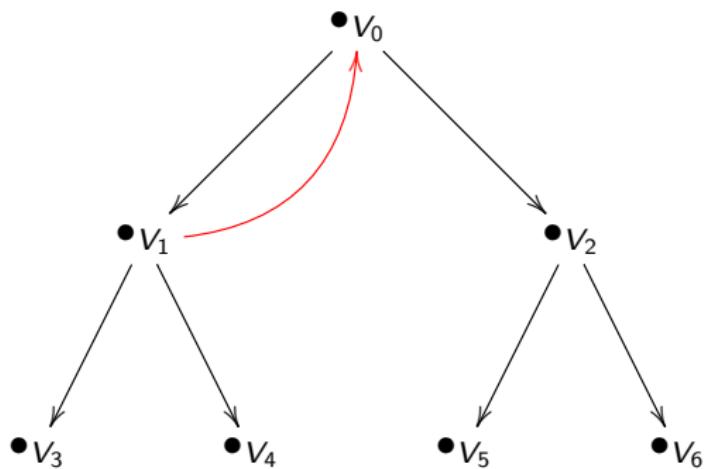
Preorder search Example



Visited: $\{V_0, V_1, V_3, V_4\}$

Preorder search Example

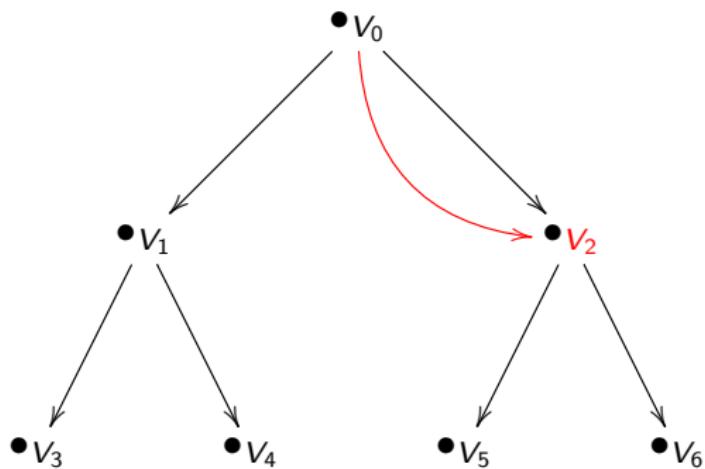
Preorder search Example



Visited: $\{V_0, V_1, V_3, V_4\}$

Preorder search Example

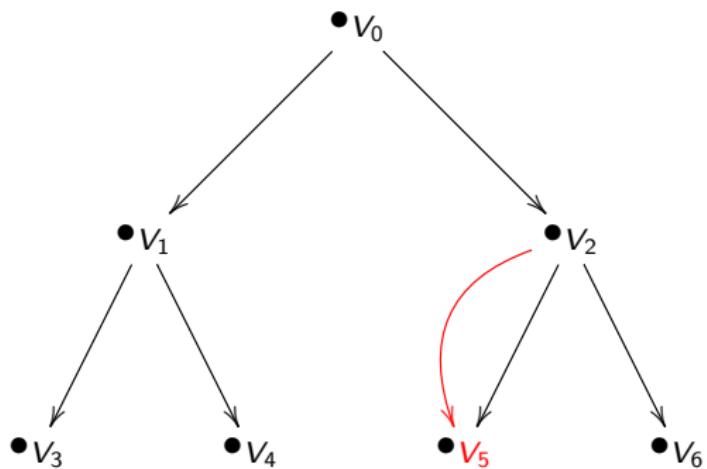
Preorder search Example



Visited: $\{V_0, V_1, V_3, V_4, V_2\}$

Preorder search Example

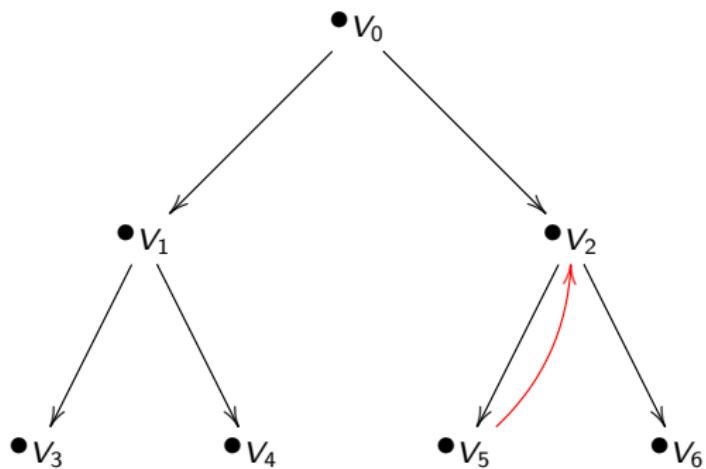
Preorder search Example



Visited: $\{V_0, V_1, V_3, V_4, V_2, V_5\}$

Preorder search Example

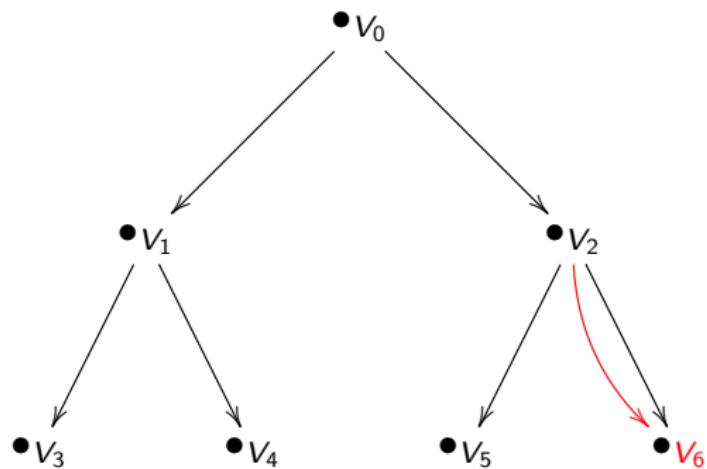
Preorder search Example



Visited: $\{V_0, V_1, V_3, V_4, V_2, V_5\}$

Preorder search Example

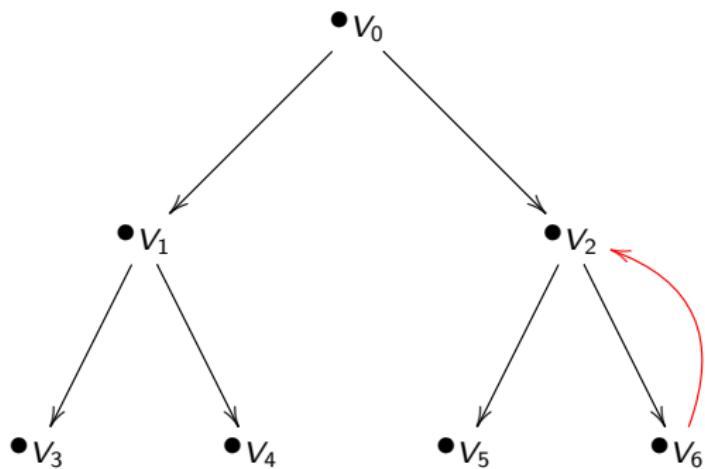
Preorder search Example



Visited: $\{V_0, V_1, V_3, V_4, V_2, V_5, V_6\}$

Preorder search Example

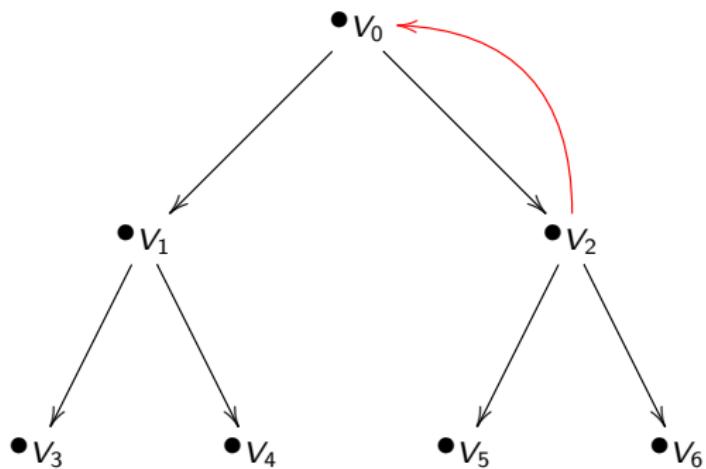
Preorder search Example



Visited: $\{V_0, V_1, V_3, V_4, V_2, V_5, V_6\}$

Preorder search Example

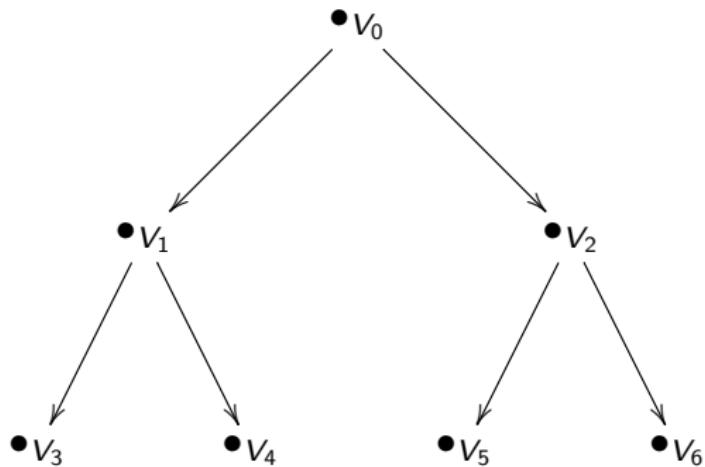
Preorder search Example



Visited: $\{V_0, V_1, V_3, V_4, V_2, V_5, V_6\}$

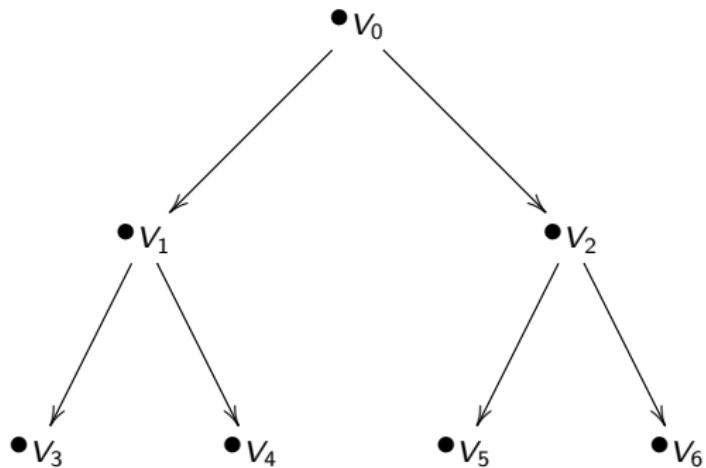
Preorder search Example

Preorder search Example



Result: $\{V_0, V_1, V_3, V_4, V_2, V_5, V_6\}$

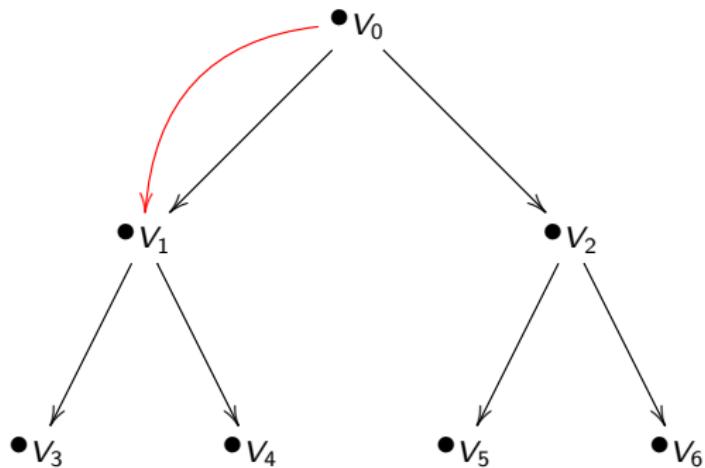
Inorder search example



Visited: \emptyset

Inorder search Example

Inorder search Example

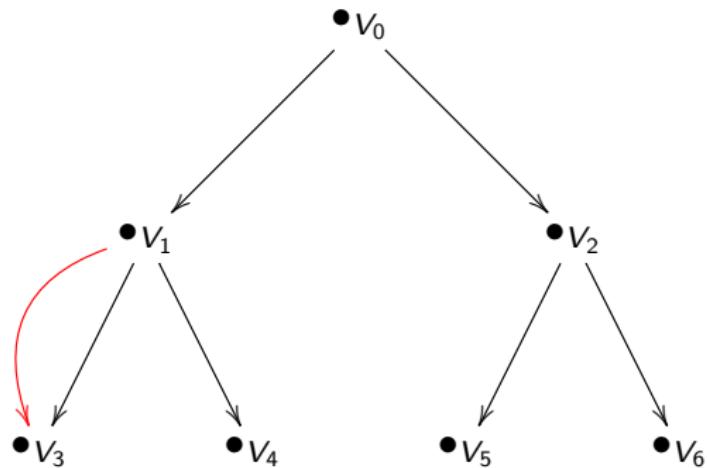


Visited: \emptyset

Inorder search Example

Inorder search Example

Inorder search Example



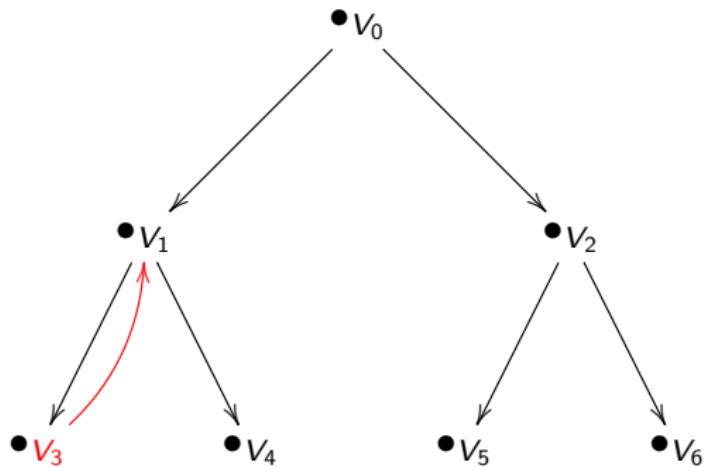
Visited: \emptyset

Inorder search Example

Inorder search Example

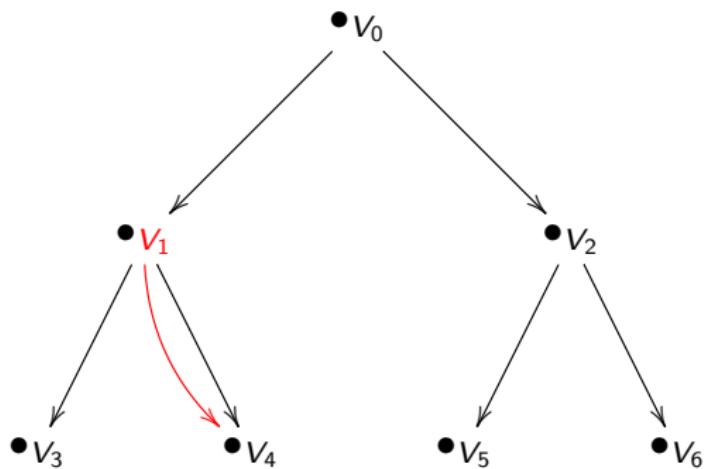
Inorder search Example

Inorder search Example



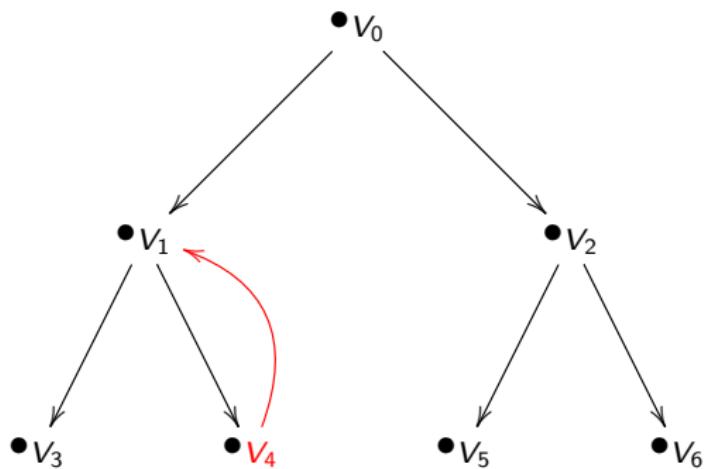
Visited: $\{V_3\}$

Inorder search Example



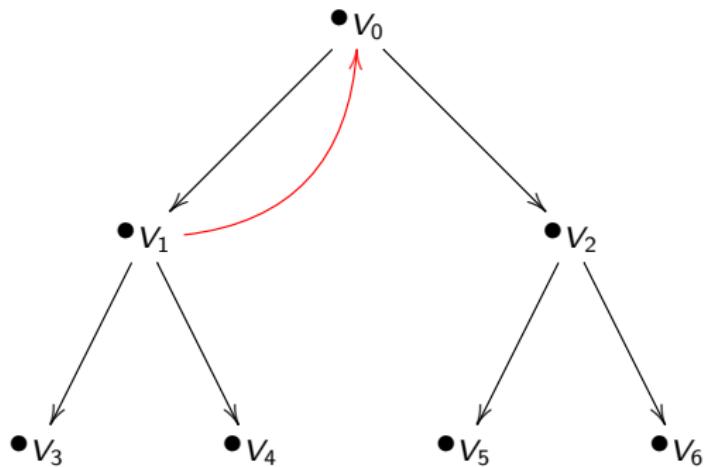
Visited: $\{V_3, V_1\}$

Inorder search Example



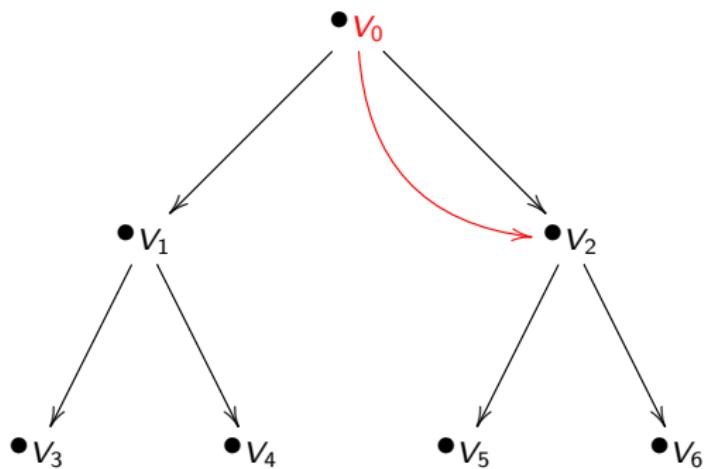
Visited: $\{V_3, V_1, V_4\}$

Inorder search Example



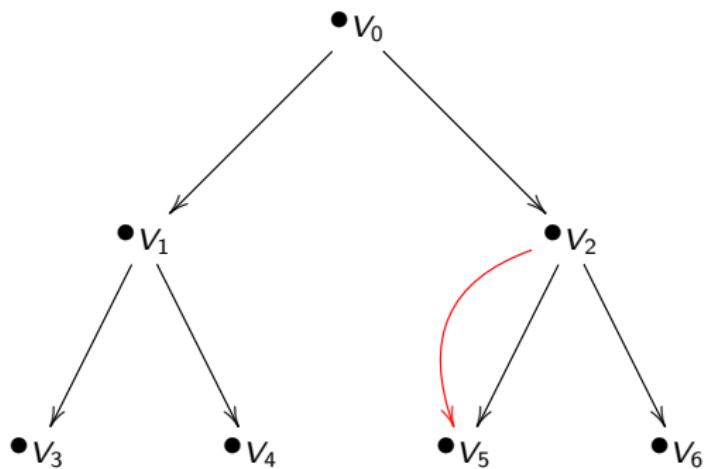
Visited: $\{V_3, V_1, V_4\}$

Inorder search Example



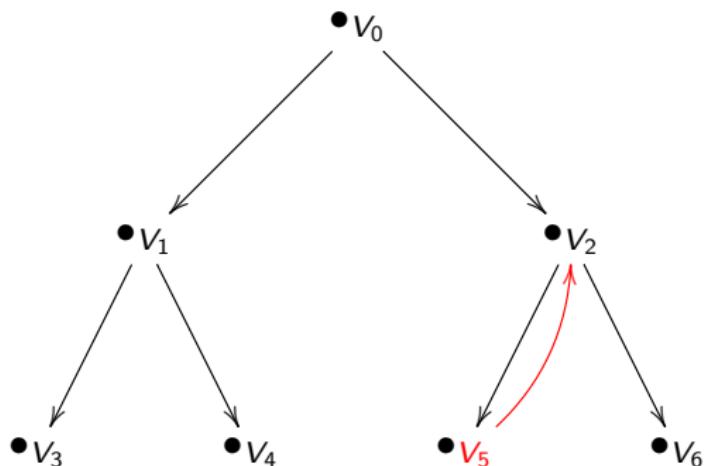
Visited: $\{V_3, V_1, V_4, V_0\}$

Inorder search Example



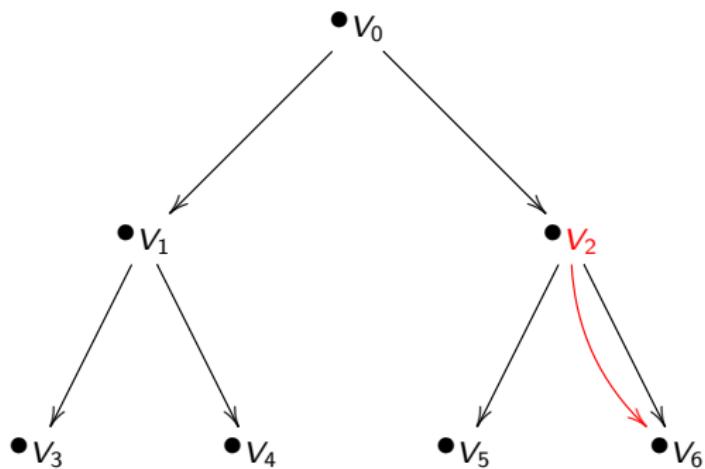
Visited: $\{V_3, V_1, V_4, V_0\}$

Inorder search Example



Visited: $\{V_3, V_1, V_4, V_0, V_5\}$

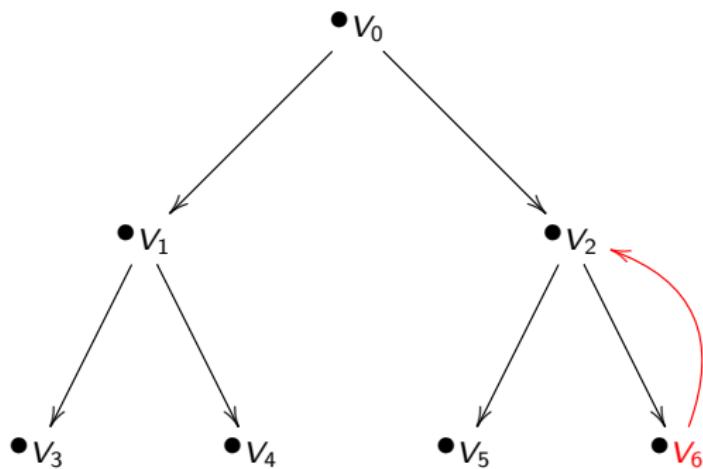
Inorder search Example



Visited: $\{V_3, V_1, V_4, V_0, V_5, V_2\}$

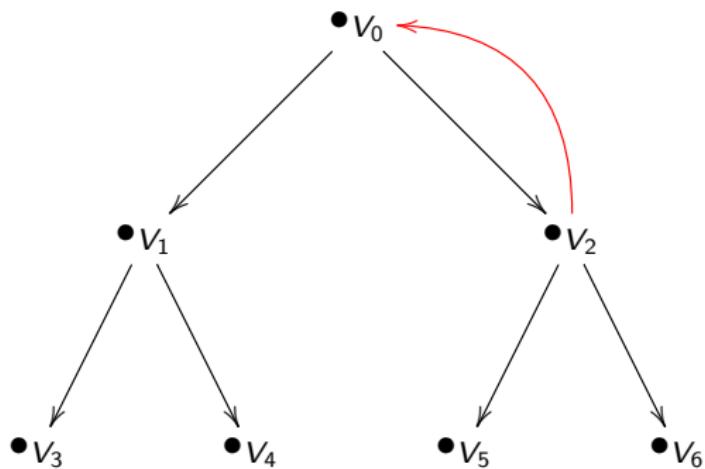
Inorder search Example

Inorder search Example



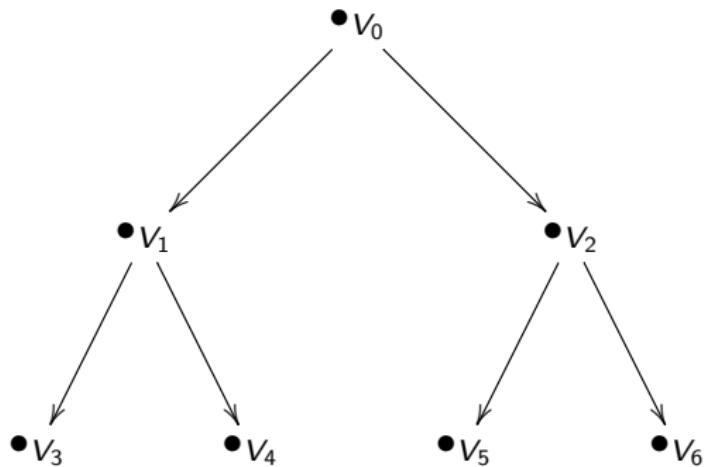
Visited: $\{V_3, V_1, V_4, V_0, V_5, V_2, V_6\}$

Inorder search Example



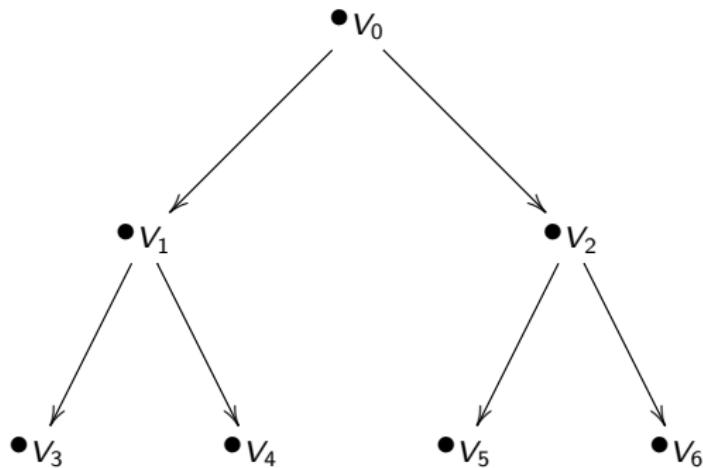
Visited: $\{V_3, V_1, V_4, V_0, V_5, V_2, V_6\}$

Inorder search Example



Result: $\{V_3, V_1, V_4, V_0, V_5, V_2, V_6\}$

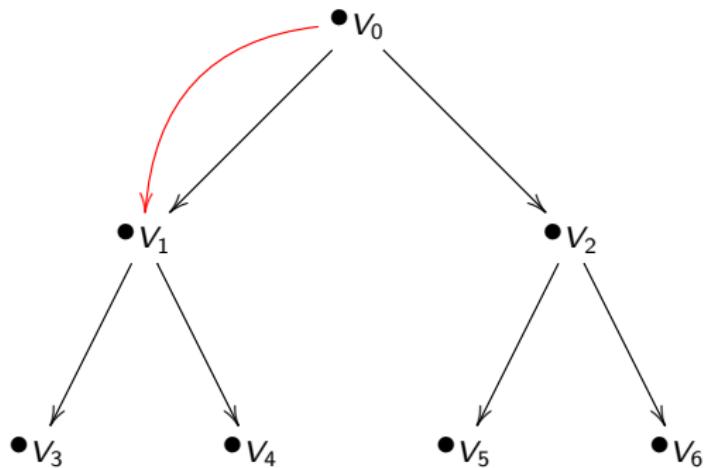
Postorder search example



Visited: \emptyset

Postorder search Example

Postorder search Example

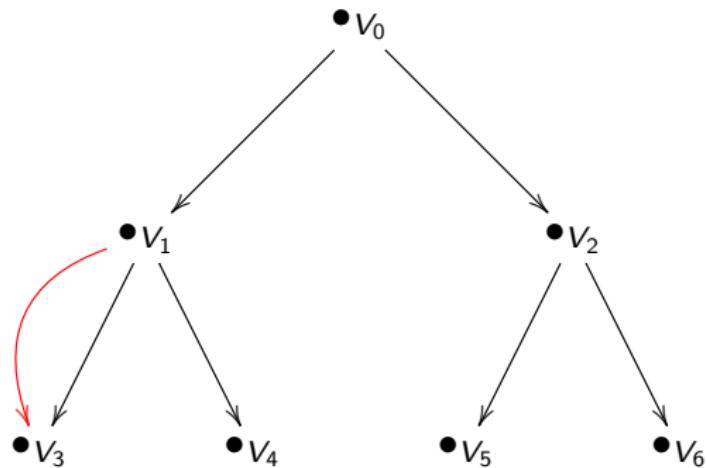


Visited: \emptyset

Postorder search Example

Postorder search Example

Postorder search Example



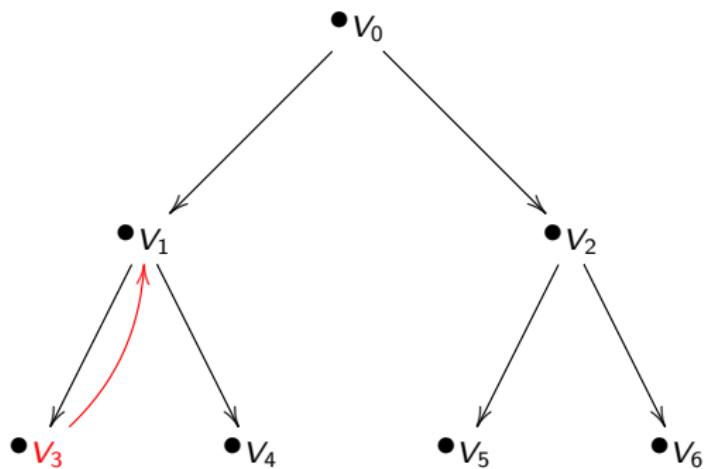
Visited: \emptyset

Postorder search Example

Postorder search Example

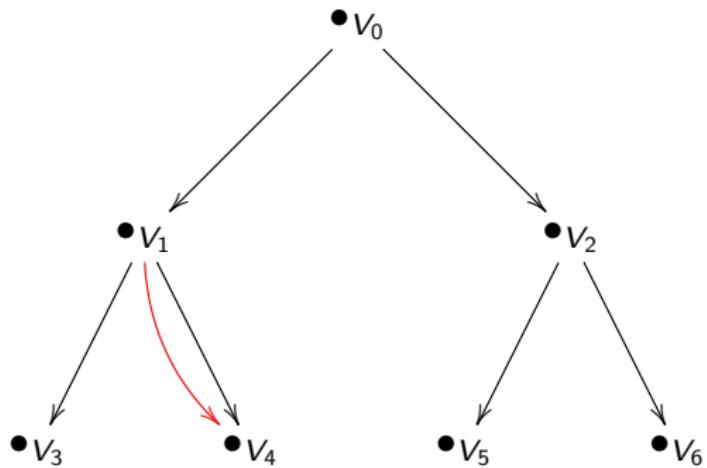
Postorder search Example

Postorder search Example



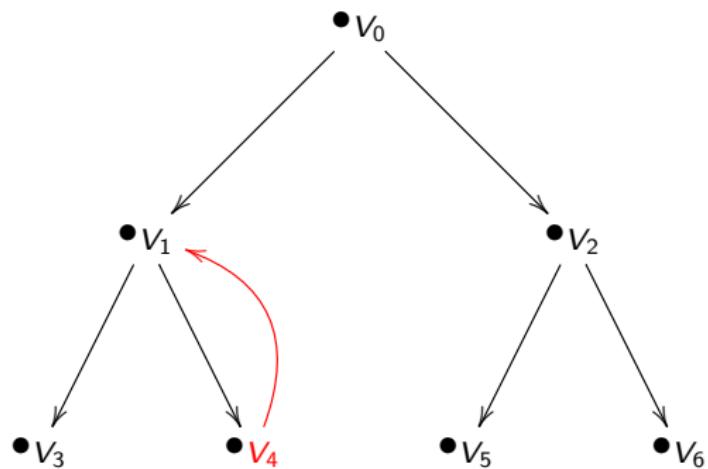
Visited: $\{V_3\}$

Postorder search Example



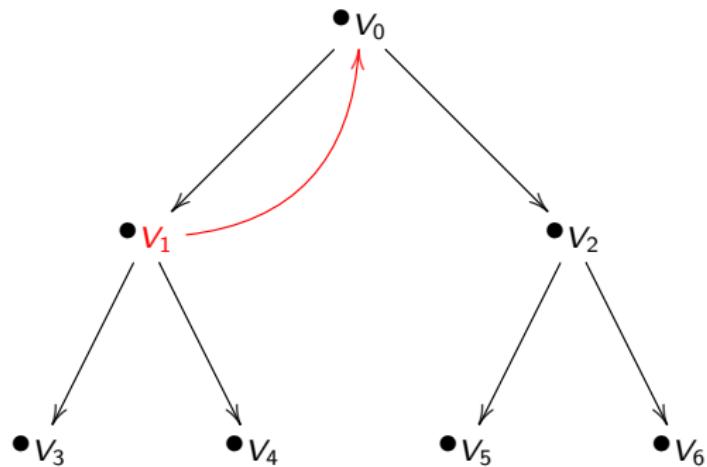
Visited: $\{V_3\}$

Postorder search Example



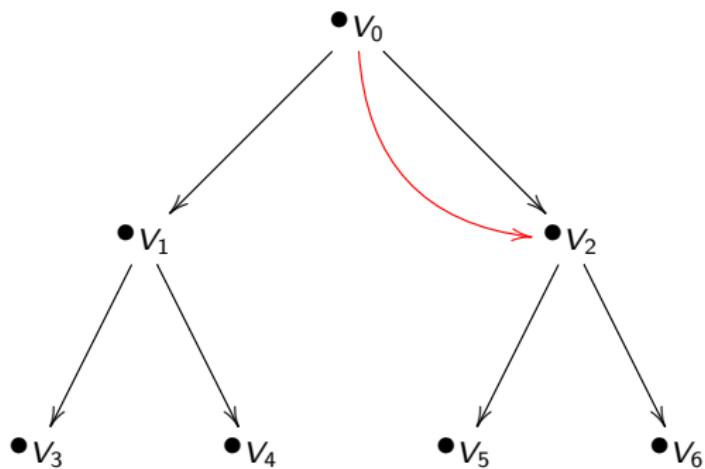
Visited: $\{V_3, V_4\}$

Postorder search Example



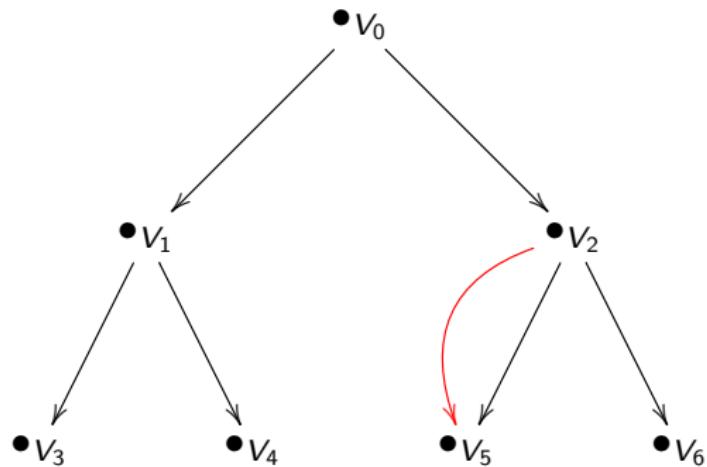
Visited: $\{V_3, V_4, V_1\}$

Postorder search Example



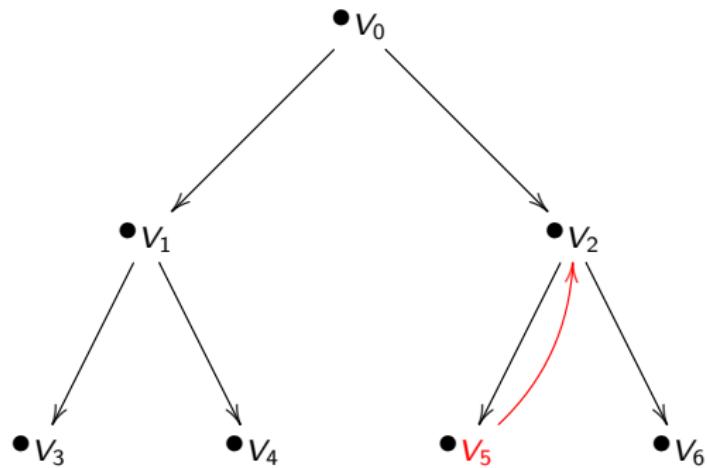
Visited: $\{V_3, V_4, V_1\}$

Postorder search Example



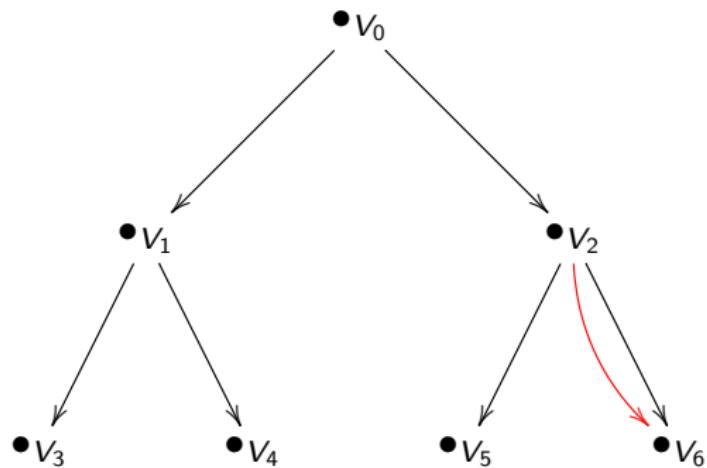
Visited: $\{V_3, V_4, V_1\}$

Postorder search Example



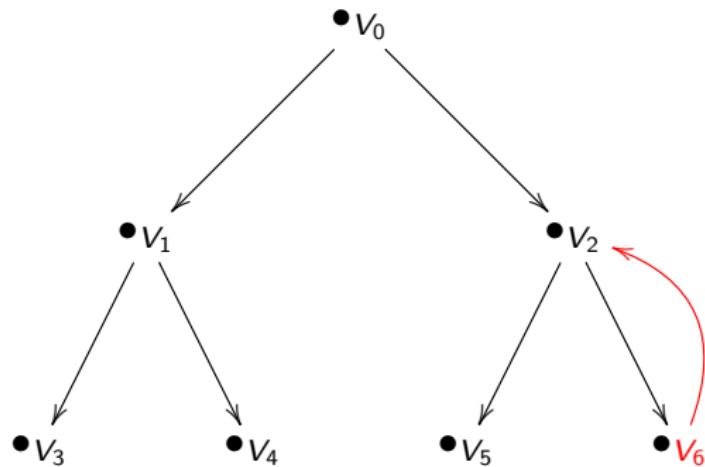
Visited: $\{V_3, V_4, V_1, V_5\}$

Postorder search Example



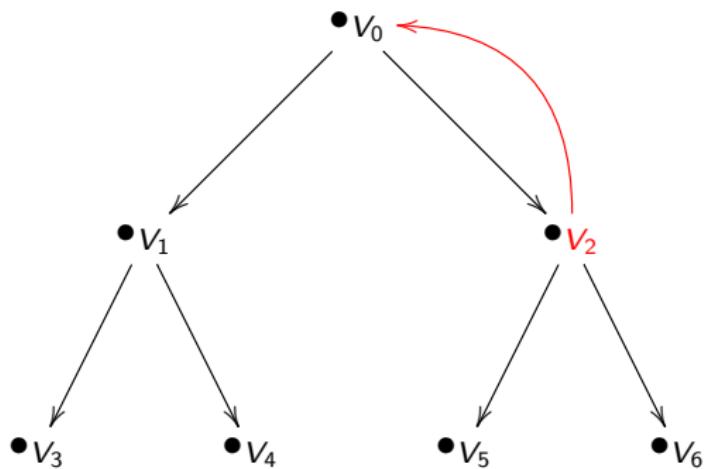
Visited: $\{V_3, V_4, V_1, V_5\}$

Postorder search Example



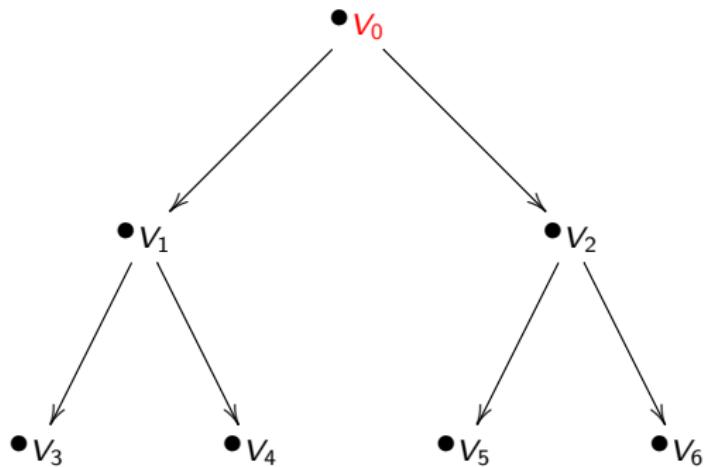
Visited: $\{V_3, V_4, V_1, V_5, V_6\}$

Postorder search Example



Visited: $\{V_3, V_4, V_1, V_5, V_6, V_2\}$

Postorder search Example



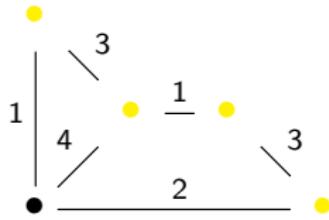
Result: $\{V_3, V_4, V_1, V_5, V_6, V_2, V_0\}$

Spanning Tree

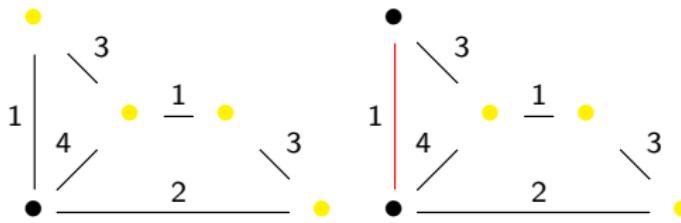
Tree T is a spanning tree on relation R if:

- T has exactly the same vertices as R .
- R can be transformed into T by deleting some edges of R .

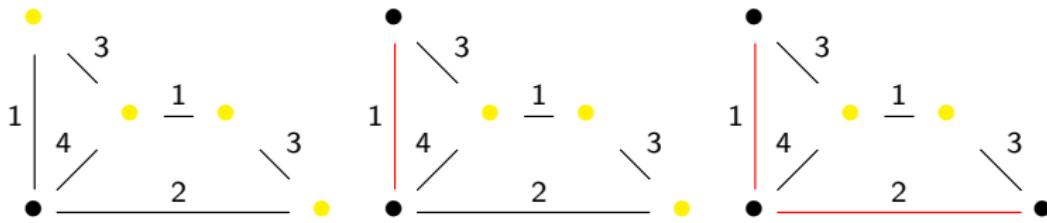
Prim's Algorithm



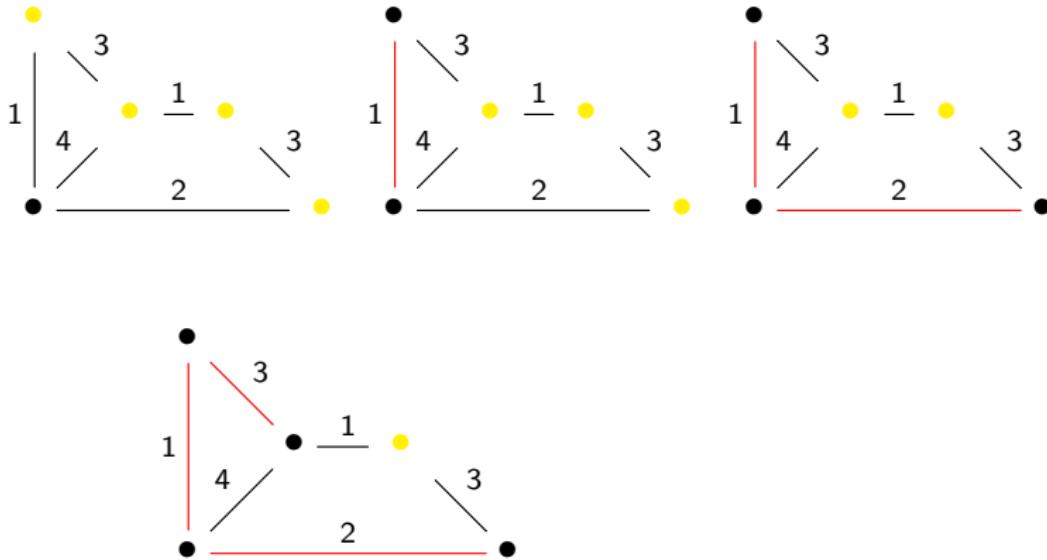
Prim's Algorithm



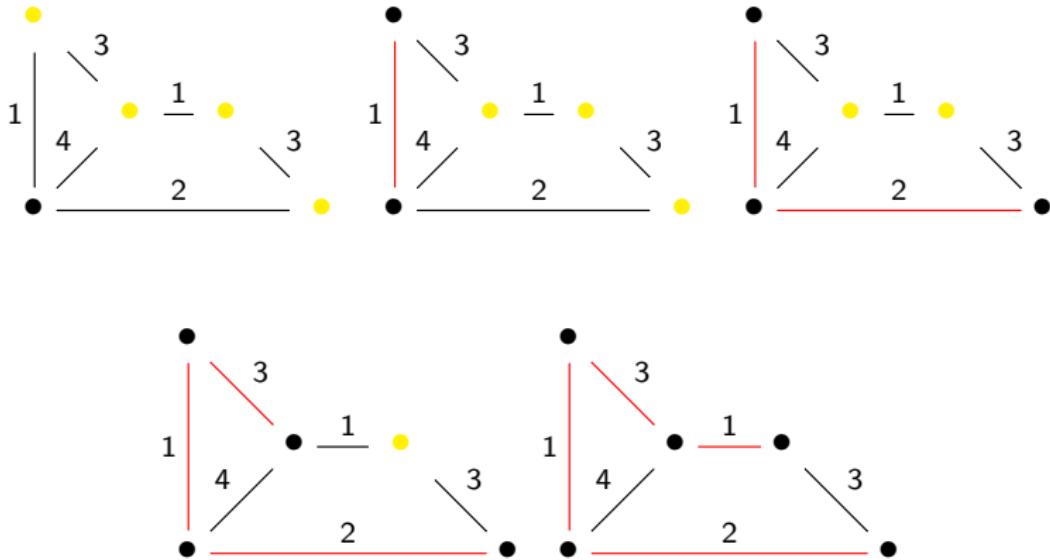
Prim's Algorithm



Prim's Algorithm



Prim's Algorithm



Kruskal's Algorithm

- Here, we add edges instead of worrying about the nearest neighbour. We simply add the edge with the smallest weight, as long as it does not create a cycle.

Paths

- A path (p) in a graph is a pair of sequences of vertices and edges (V_p, E_p) .

Paths

- A path (p) in a graph is a pair of sequences of vertices and edges (V_p, E_p) .
- In a path, each vertex v_i in the sequence is adjacent to the next vertex v_{i+1} in the sequence, and they are both endpoints for the edge at the same location e_i .

Paths

- A path (p) in a graph is a pair of sequences of vertices and edges (V_p, E_p) .
- In a path, each vertex v_i in the sequence is adjacent to the next vertex v_{i+1} in the sequence, and they are both endpoints for the edge at the same location e_i .
- No edge occurs more than once in E_p .

Paths

- A path (p) in a graph is a pair of sequences of vertices and edges (V_p, E_p) .
- In a path, each vertex v_i in the sequence is adjacent to the next vertex v_{i+1} in the sequence, and they are both endpoints for the edge at the same location e_i .
- No edge occurs more than once in E_p .
- A circuit is a path that starts and ends on the same vertex.

Paths

- A path (p) in a graph is a pair of sequences of vertices and edges (V_p, E_p) .
- In a path, each vertex v_i in the sequence is adjacent to the next vertex v_{i+1} in the sequence, and they are both endpoints for the edge at the same location e_i .
- No edge occurs more than once in E_p .
- A circuit is a path that starts and ends on the same vertex.
- If no vertex in a path occurs more than once (except, possibly, for the first and the last), then the path is simple.

Euler

- If a path uses every edge exactly once then it is an Euler path.

Euler

- If a path uses every edge exactly once then it is an Euler path.
- If an Euler path starts and ends on the same vertex it is an Euler circuit.

Euler

- If a path uses every edge exactly once then it is an Euler path.
- If an Euler path starts and ends on the same vertex it is an Euler circuit.
- Fleury's Algorithm can be used to find Euler paths and circuits.

Euler

Euler Circuits

Graph G does not have a Euler circuit if there is a vertex of odd degree. If G is connected and every vertex has even degree, then there is an Euler circuit in G .

Euler Paths

If there are exactly two vertices with an odd degree in graph G then there is an Euler path. That path starts at a vertex with an odd degree and ends at the other vertex with an odd degree. If there are more than 2 vertices with odd degrees, then there is no Euler path.

Fleury's Algorithm

TODO: This description of the algorithm is unclear. See the lecture slides for a better description.

- ① Start at a vertex with an odd degree or any vertex if no vertex with odd degrees exists.
- ② Go to any adjacent vertex, as long as the edge does not cause the graph to become disconnected.
- ③ Modify G such that the edge most recently travelled is removed.
- ④ Repeat steps 2 and 3 till a path/circuit is created.

Hamiltonian Paths

- A Hamiltonian path visits each vertex in graph G exactly once.
- A Hamiltonian circuit is a circuit that visits each vertex in graph G exactly once, except the begin/end node, which is visited twice.
- A Hamiltonian graph is graph with a Hamiltonian circuit.
- As long as there are enough "edges" to travel, it is a Hamiltonian graph.

Sufficient Conditions for Hamiltonian Graphs

- The degree of each vertex is greater than half the number of vertices
- The number of edges is greater than $\frac{n^2 - 3n + 6}{2}$.
- Hamiltonian graphs can exist that do not satisfy the above conditions!

Cover Events

Owl design competition

Sunday 11th of February

Google Forms

Hackathon with Belsimpel

Saturday 17th of February

TBA

IoT workshop at CGI

Thursday 22nd of February

Eemsgolaan 1, 9727 DW Groningen

See svcover.nl for more events!

Fin.

These slides can be found on our website, **studcee.svcover.nl**
Please don't forget to fill in an evaluation form!



Figure: QR code

Acknowledgements

Based on the slides by Evi Xhelo and Nico Stegeman