Week 6 - Lectures 11, 12

1) Trees
   ↳ algorithms for spanning trees

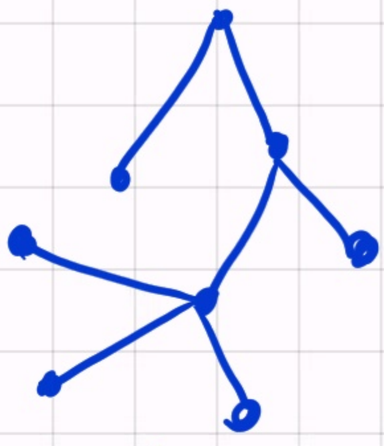
2) Combinatorial Optimization I

- Weighted graphs & minimal spanning tree

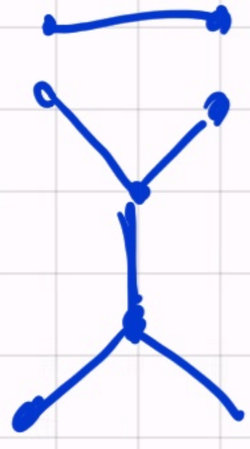
Trees

Def: A tree is a connected (loop free simple) graph with no cycles

A Forest is a (loop-free, simple) graph with no cycles



✓



X (not connected

Forest

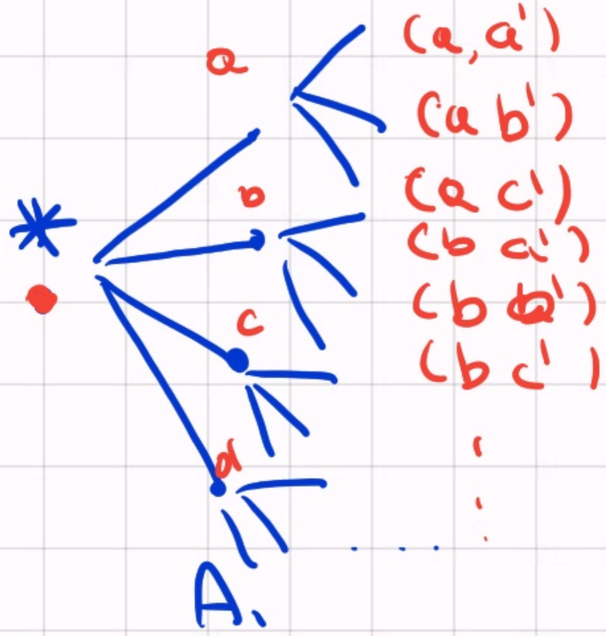
X there is a cycle

Example $A_1 \dots A_n$ sets $\left(\bigcup_{i=0}^{n} A_1 \times \dots \times A_i\right) = V(G)$

(Convention the empty product is a $\{*\}$)

$(a_1 \dots a_i)$ is adjacent to $(a_1 \dots a_i\ a_{i+1})$

$(a_1 \dots a_{i-1})$



Def A **spanning tree** (forest) for a graph G is a tree (forest) T, with $V(G) = V(T)$

Proposition A graph has a spanning tree $\Leftrightarrow$ It is connected.

Proof (G finite)

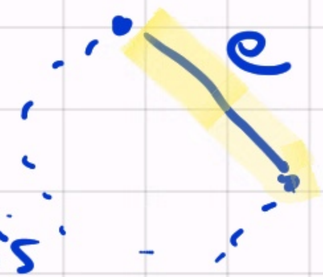
$\Rightarrow$) We assume that G has a spanning tree T we take $a, b \in V(G) = V(T)$
T is connected $\Rightarrow \exists$ a path in T joining a and b $\Rightarrow \exists$ a path in G joining a and b $\Rightarrow$ G is connected.

$\Leftarrow$ We assume that G is connected

If it is a tree there is nothing to prove. So we can assume that G has a cycle. Let e an edge of this cycle

$G - e \subseteq G$ $\quad V(G) = V(G-e)$

$V(G-e)$ is connected.

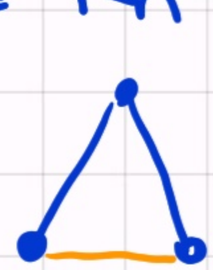


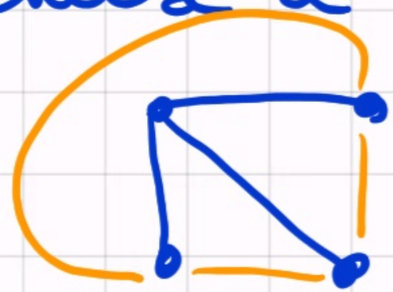
If it is a tree we stop otherwise

we reiterate (G finite) this process end.

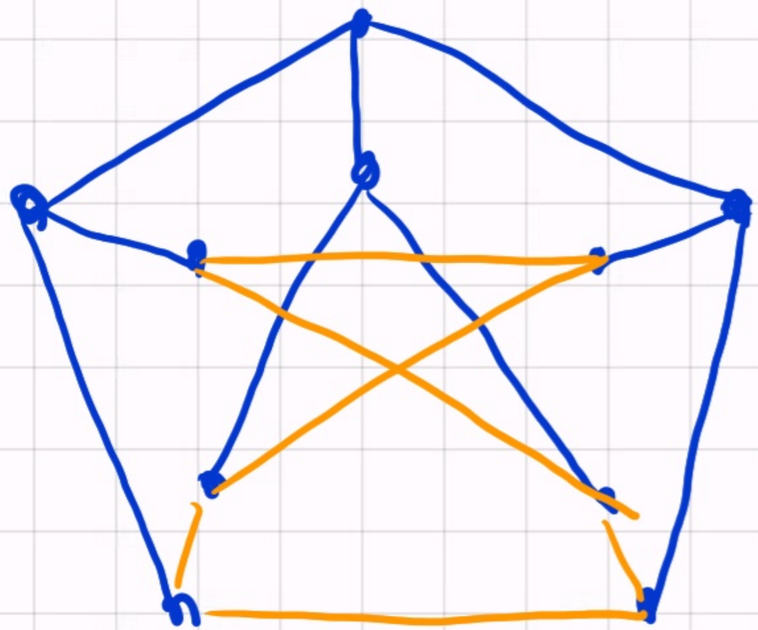
QED

Example $G = K_n$: choose a vertex & connect the other.





(blue are spanning tree)



(? there might be many spanning trees)

Theorem The following are equivalent for a loop free Graph G:

1) G is a tree

2) $\forall x, y \in V(G)$, $x \neq y$ $\exists!$ path connecting x and y

If furthermore G is finite we have that 1) & 2) are equivalent to

3) G is connected and $|V(G)| = |E(G)| + 1$

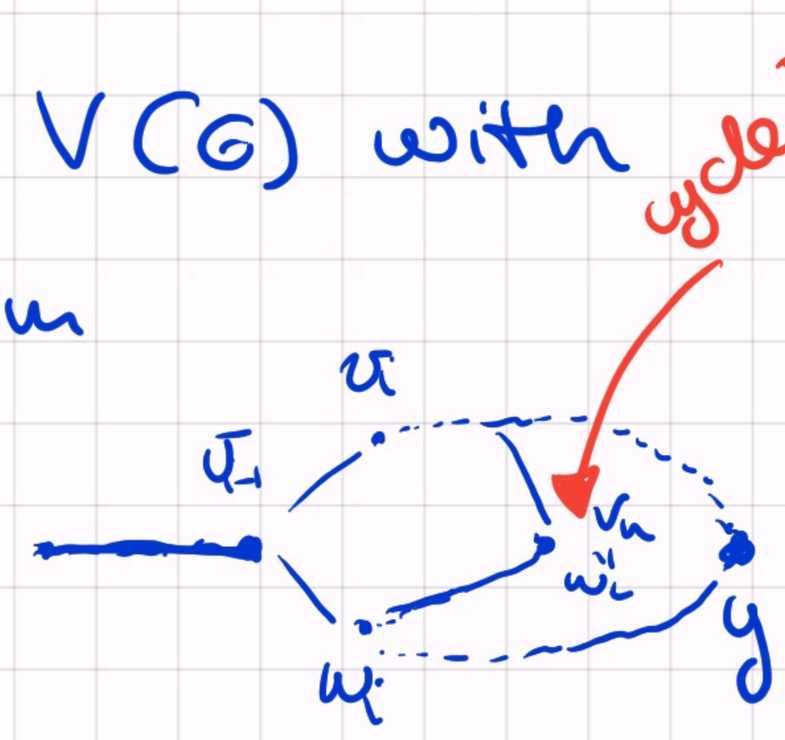
Proof

1) ⇒ 2) | (not ↙ ¬2) ⇒ ¬1))

We assume that ∃ x, y ∈ V(G) with two paths connecting them

$(x = v_1, v_2, \dots, v_n = y)$

$(x = w_1, w_2 \dots\dots w_k = y)$



$i = \min \{ h \mid v_h \neq w_h \}$

$j = \min \{ h \geq i \mid v_h$ is a vertices in the second walk $\}$

$v_h = w_L$ for some $L \geq i$

$(v_i = w_i, v_{i+1} \dots v_n = w_l, w_{l-1} \dots w_i)$

this is a cycle $\Rightarrow$ G is not a tree.

2) $\Rightarrow$ 1) G is connected. Suppose that there is a cycle in g $(v_1 v_2 \dots v_{n-1}, v_1)$

$(v_1 v_2)$ are two different path connecting v_1 and v_2

$(v_1 v_{n-1}, v_{n-2} \dots v_2)$

$\Rightarrow$ -12)

G is finite

1) => 3) by induction on $|E|$

$|E| = 0$ $\quad G = \bullet$ $\quad |V| = 1 = 0 + 1$

$= |E| + 1$

Assume that this is true if $|E| \leq k$

Let G with $|E(G)| = k+1$. We remove an edg from G $\Rightarrow$ split G in two trees.

$|E(G)| = k+1$

$|V(G)| = |V_1 \cup V_2| = |V_1| + |V_2|$

induction $\rightarrow$ $= |E_1| + 1 + |E_2| + 1 = (|E_1| + |E_2| + 1) + 1$

$|E_1 \cup E_2| + |\{e\}|$

3) => 1) We find a spanning tree T

$$|E(T)| = |V(T)| - 1 = |V(G)| - 1 = |E(G)|$$

$$E(T) \subseteq E(G)$$

$$|E(G)| < +\infty$$

QED

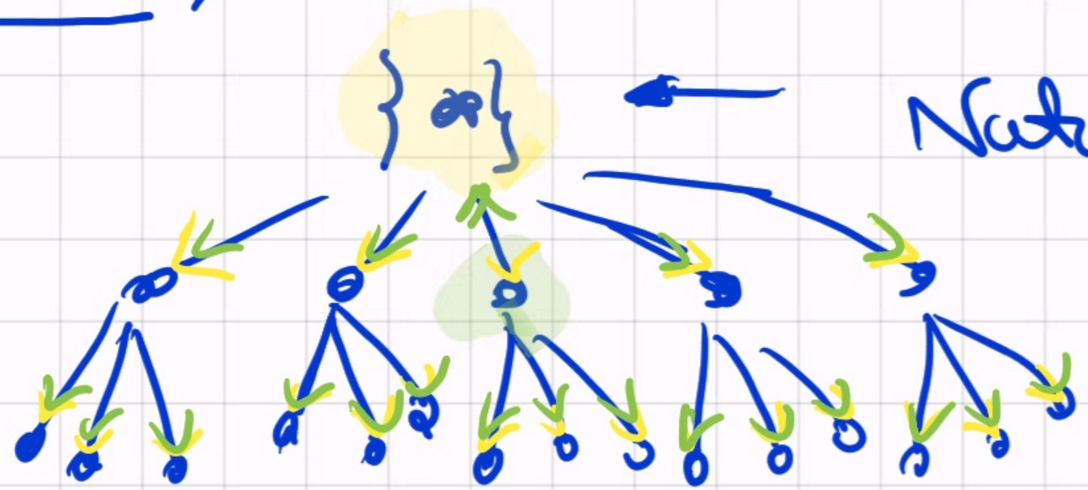
Rooted trees :

A rooted tree is a tree T with a distinguished vertex $v \in V(T)$ (the root)

Example

A_1

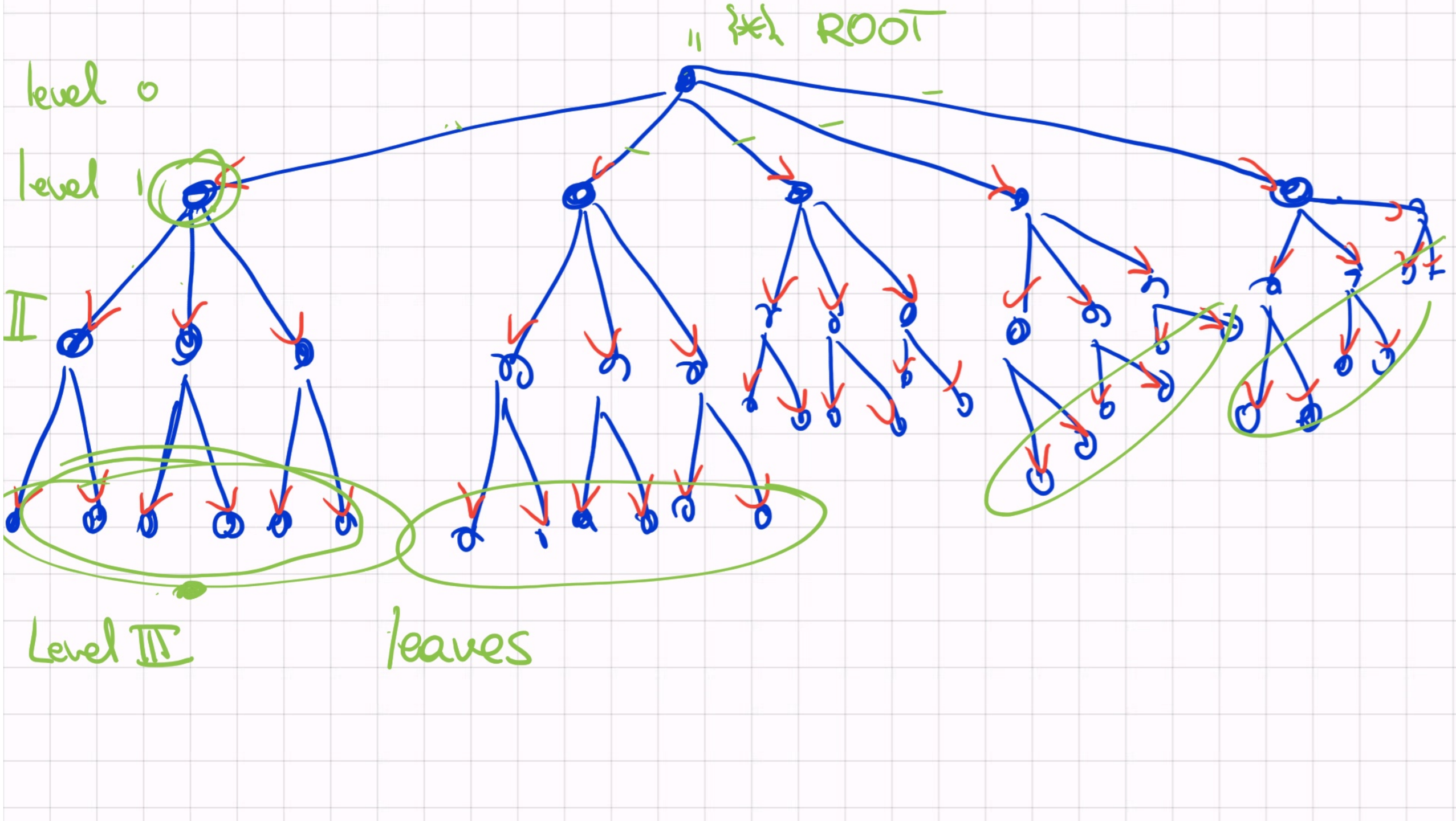
$A_2 \times A_2$



Natural root

Def A directed tree is a direct graph such that the underlying graph is a tree

Prop: Let G a rooted tree. There is just one orientation (choice of direction for edges) such that 1) The incoming degree of the root v is 0
2) The incoming degree of any other vertex is 1


ROOT
level 0
level 1
II
Level III
leaves

Def In a tree a terminal vertex is called leaf.

The non terminal vertices are called internal.

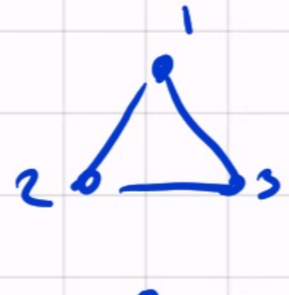
If (T, r) is a rooted tree and $v \in V(G)$ the level of v is the length of the unique path from r to v

Given w adjacent to v we say that w is a

1) child if level $w >$ level v (descendant)
2) parent if level $w <$ level v (ancestors)

Algorithm for spanning tree

$G = \{\{v_1 \dots v_n\}, E\}$



Step 1 $T := (\{v_1\}, \phi)$, $v = v_1$

$T = (\{v_1 \dots v_i\}, E(T))$ $v_i = v$

Step 2 if $\exists\, v_k \notin V(T)$ with $\{v, v_k\} \in E$

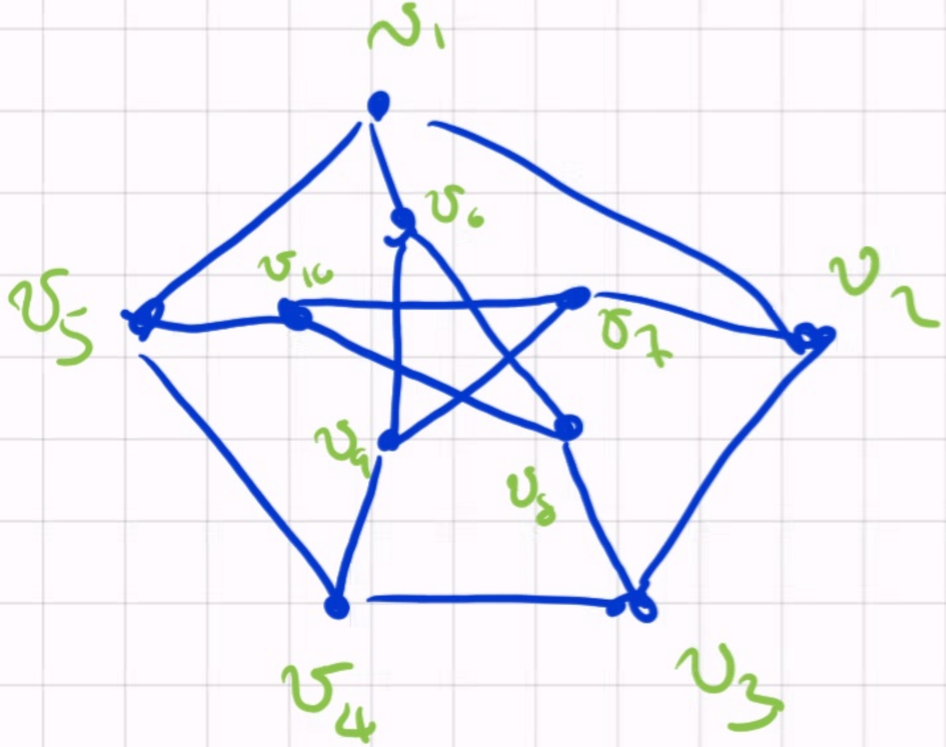
$i = \min\{k\}$

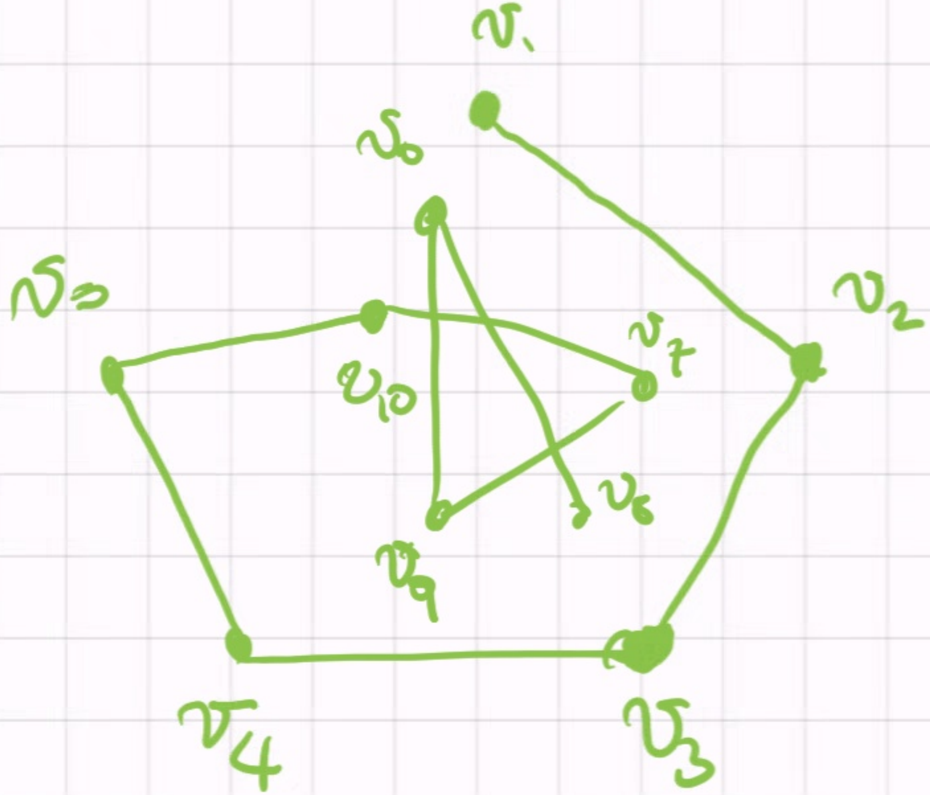
$T = T + (v, v_{i+1})$ $\boxed{v = v_{i+1}}$ Return to 2

Else go to step 3

Step 3 if $v = v_1$ then T is the spanning tree

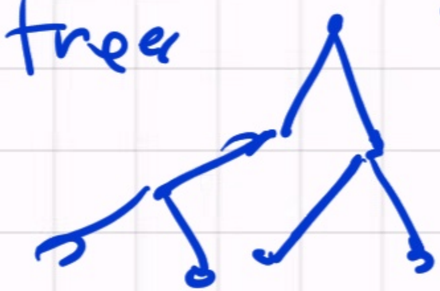
otherwise set u = parent v . $v = u$.





(next slide)

In the book you have another algorithm that creates more "balanced" tree



Q = Queue (First in First out)

Step 1 $T = (\{v_1\}\ \emptyset)$

$Q = [v_1]$

Step 2 If $Q \neq \emptyset$

$v = \text{front } Q$

$Q = Q \setminus v$

for $i = 2 \ldots n$

if v_i is adjacent to v & not visited

$Q = [Q, v_i]$ $T = T + \{v\ v_i\}$

$i = i + 1$

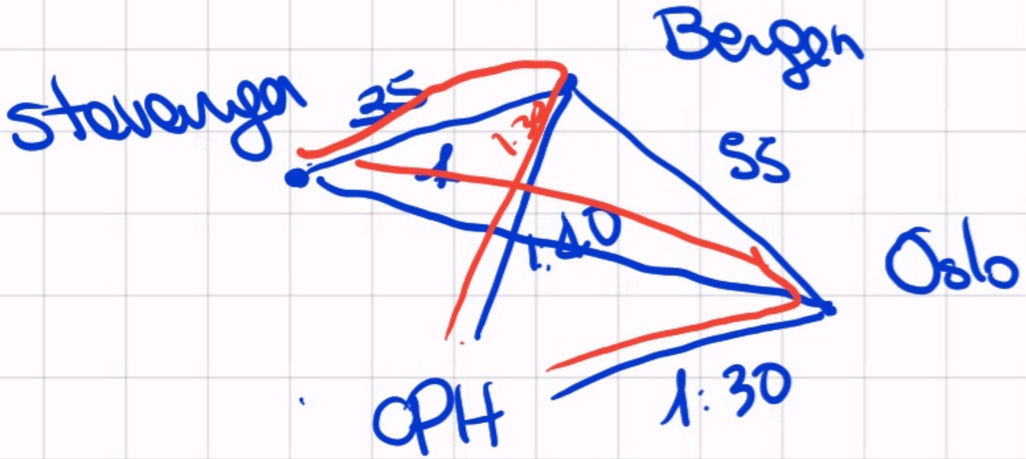
Combinatorial Optimization I

Weighted Graphs

A weighted Graph is a pair (G, ω) with G a graph and $\omega: E(G) \longrightarrow \mathbb{R}^+$

Examples

- Model flight time between different cities
- Transportation costs on different root



If $\Gamma: (v_1 \; .. \; v_n)$ is a path in a weighted graph the length of Γ is

$$L(\Gamma) = \sum_{i=1}^{n-1} w\left(\{v_i \; v_{i+1}\}\right)$$

Example Flight from CPH to Stavanger

Via Oslo $L(\Gamma) = 3\text{h } 70 \text{ minute}$

Via Bergen $L(\Gamma) = 2\text{h } 06 \text{ minutes}$

The pseudo distance associated to ω

$$d : V(G) \times V(G) \longrightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$$

$$d(v, w) = \operatorname{Inf}\{ l(\Gamma) \mid \Gamma \text{ is a path from } v \text{ to } w \}$$

G finite $=$ $\min\{ l(\Gamma) \mid \Gamma \text{ is a path} \dots \}$

Properties

$$d(v\,v) = 0$$

$$d(v\,w) = d(w\,v)$$

$$d(v, w) + d(w, u) \geq d(v, u)$$

Prof (Exercise)

Convention: any weigth function $w : E \longrightarrow \mathbb{R}_{\geq 0}$ can be extended to a function

$$\overline{w} : V(G) \times V(G) \longrightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$$

$$\overline{w}(v, u) = \begin{cases} w\{u,v\} & \text{if } \{u,v\} \in E(G) \\ \infty & \text{otherwise} \end{cases}$$

Dijkstra's shortest path algorithm

Goal (G, w) weighted graph $v_0 \in V(G)$

Find the shortest path from v_0 to every vertex of G

Notation $S \subseteq V(G)$

$\overline{S} := S \cup \{ v \in V(G) \mid \exists\, u \in S \text{ adjacent to } v \}$

$n = |V(G)|$

Dijkstra's Shortest-Path Algorithm

Step 1: Set the counter $i = 0$ and $S_0 = \{v_0\}$. Label v_0 with $(0, -)$ and each $v \neq v_0$ with $(\infty, -)$.

If $n = 1$, then $V = \{v_0\}$ and the problem is solved.
If $n > 1$, continue to step (2).

Step 2: For each $v \in S_i$ replace, when possible, the label on v by the new label $(L(v), y)$ where

$$L(v) = \min_{u \in S_i}\{L(v), L(u) + \text{wt}(u, v)\},$$

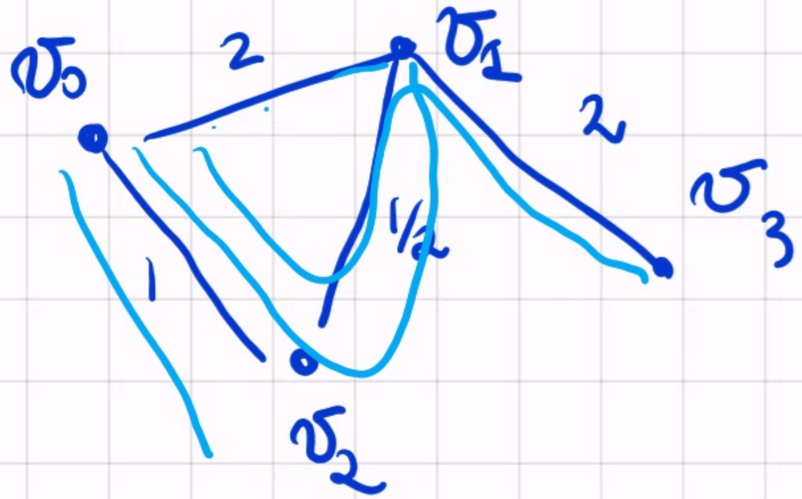
and y is a vertex in S_i that produces the minimum $L(v)$. [When a replacement does take place, it is due to the fact that we can go from v_0 to v and travel a shorter distance by going along a path that includes the edge (y, v).]

Step 3: If every vertex in $\overline{S}_i$ (for some $0 \leq i \leq n - 2$) has the label $(\infty, -)$, then the labeled graph contains the information we are seeking.

If not, then there is at least one vertex $v \in \overline{S}_i$ that is not labeled by $(\infty, -)$, and we perform the following tasks:

1) Select a vertex v_{i+1} where $L(v_{i+1})$ is a minimum (for all such v). There may be more than one such vertex, in which case we are free to choose among the possible candidates. The vertex v_{i+1} is an element of $\overline{S}_i$ that is closest to v_0.
2) Assign $S_i \cup \{v_{i+1}\}$ to S_{i+1}.
3) Increase the counter i by 1.

If $i = n - 1$, the labeled graph contains the information we want.
If $i < n - 1$, return to step (2).



$f(v_i)$ = label $(L(v), v)$

$S = \{v_0, v_1, v_2\}$

Initialization

$S_0 = \{v_0\}$

$P(v_0) = (0, -)$

$f(v_i) = (\infty, -)$

First iteration.

set $f(v_2) = (1\ v_0)$

$f(v_1) = (2, v_0)$

$S = \{v_0, v_2\}$

second iteration

$$L(v_1) = \min\left\{2 , \underbrace{1}_{L(v_2)} + \frac{1}{2}\right\} = \frac{3}{2}$$

$$f(v_1) = \left(\frac{3}{2}, v_2\right)$$

$$f(v_3) = \min\{+\infty , +\infty\} = (+\infty, -)$$

$$S = \{v_0\ v_1\ v_2\}$$

$f(v_0) = (0, -)$ $\qquad$ $f(v_2) = (1, v_0)$

$f(v_1) = (3/2, v_2)$ $\qquad$ $f(v_3) = (4, v_1)$

$\vdots$

Step 2

$\overline{S_0} =$

Minimal spanning tree

(G, w) weighted graph, a minimal spanning tree is a Spanning tree T for G such thhe

$$w(T) := \sum_{e \in E(T)} w(e) \leq \sum_{e \in E(T')} w(e) \quad \forall \text{ the}$$

Spanning tree.

$n = |V(G)|$

$S = \phi$

Kruskal's Algorithm

Step 1: Set the counter $i = 1$ and select an edge e_1 in G, where $\text{wt}(e_1)$ is as small as possible.

$S = \{e_1\}$

Step 2: For $1 \leq i \leq n-2$, if edges $e_1, e_2, \ldots, e_i$ have been selected, then select edge e_{i+1} from the remaining edges in G so that (a) $\text{wt}(e_{i+1})$ is as small as possible and (b) the subgraph of G determined by the edges $e_1, e_2, \ldots, e_i, e_{i+1}$ (and the vertices they are incident with) contains no cycles.

Step 3: Replace i by $i + 1$.

If $i = n - 1$, the subgraph of G determined by edges $e_1, e_2, \ldots, e_{n-1}$ is connected with n vertices and $n - 1$ edges, and is an optimal spanning tree for G.

If $i < n - 1$, return to step (2).

Theorem Any spanning tree obtained by Kruskal Algorithm is optimal.

Proof $|V(G)| = n$ T spanning tree obtaine by the algoritm. $E(T) = \{ e_1 \dots e_{n-1} \}$

T' optimal spanning tree

$$d(T') = \min_k \{ E(T') \ni e_1 \dots e_{k-1}, e_k \notin E(T') \}$$

Let T_1 optimal & such that $d(T_1)$ is maximal

If $d(T_1) = n$ ✓ $T = T_1$ so is optimal

$\{a_1 \cdots a_{n-1}\}$

Suppose that $d(T_1) \leq r < n$ $e_r \notin E(T_1)$

$T_1 + e_r$ has a cycle. Let e'_r to be another edge of the cycle.

$T_2 = T_1 + e_r - e'_r$

$w(T_2) = w(T_1) + \overbrace{w(e_r) - w(e'_r)}^{\leq 0}$ algorithm.

$\leq w(T_1)$

T_1 optimal $\Rightarrow$ $w(T_2) = w(T_1)$

T_2 is optimal.

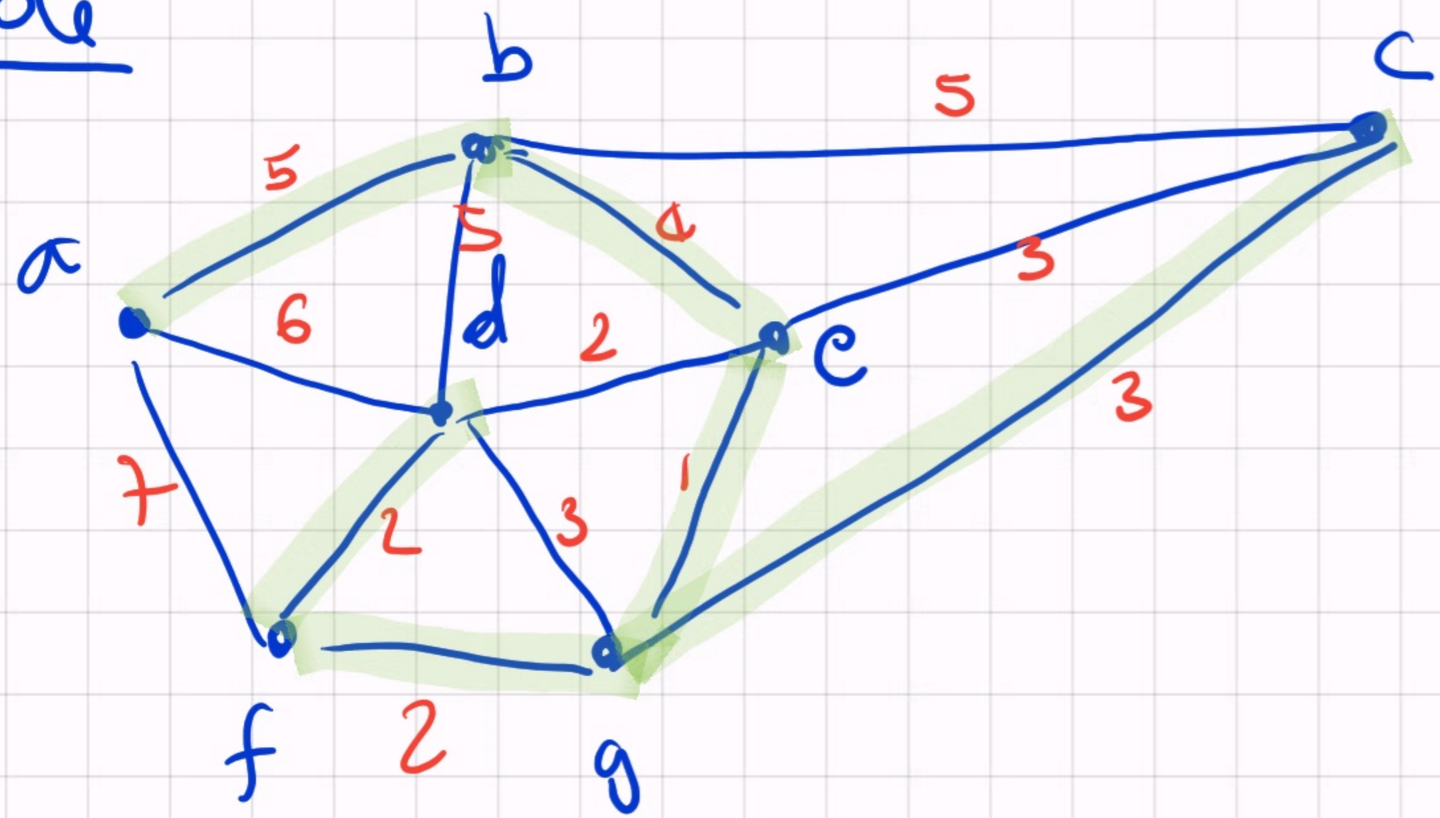
$d(T_2) = r+1$ $\Rightarrow T_2 = T$

So T is optimal

:)

QED

Example



$S = \{\{e, g\}\}$ 1st iteration.

S = {{eg}, {fd}} 2nd iteration

S = {{eg} {f,d}} {fg}} 3rd iteration

{gc}} 4th

{bc}

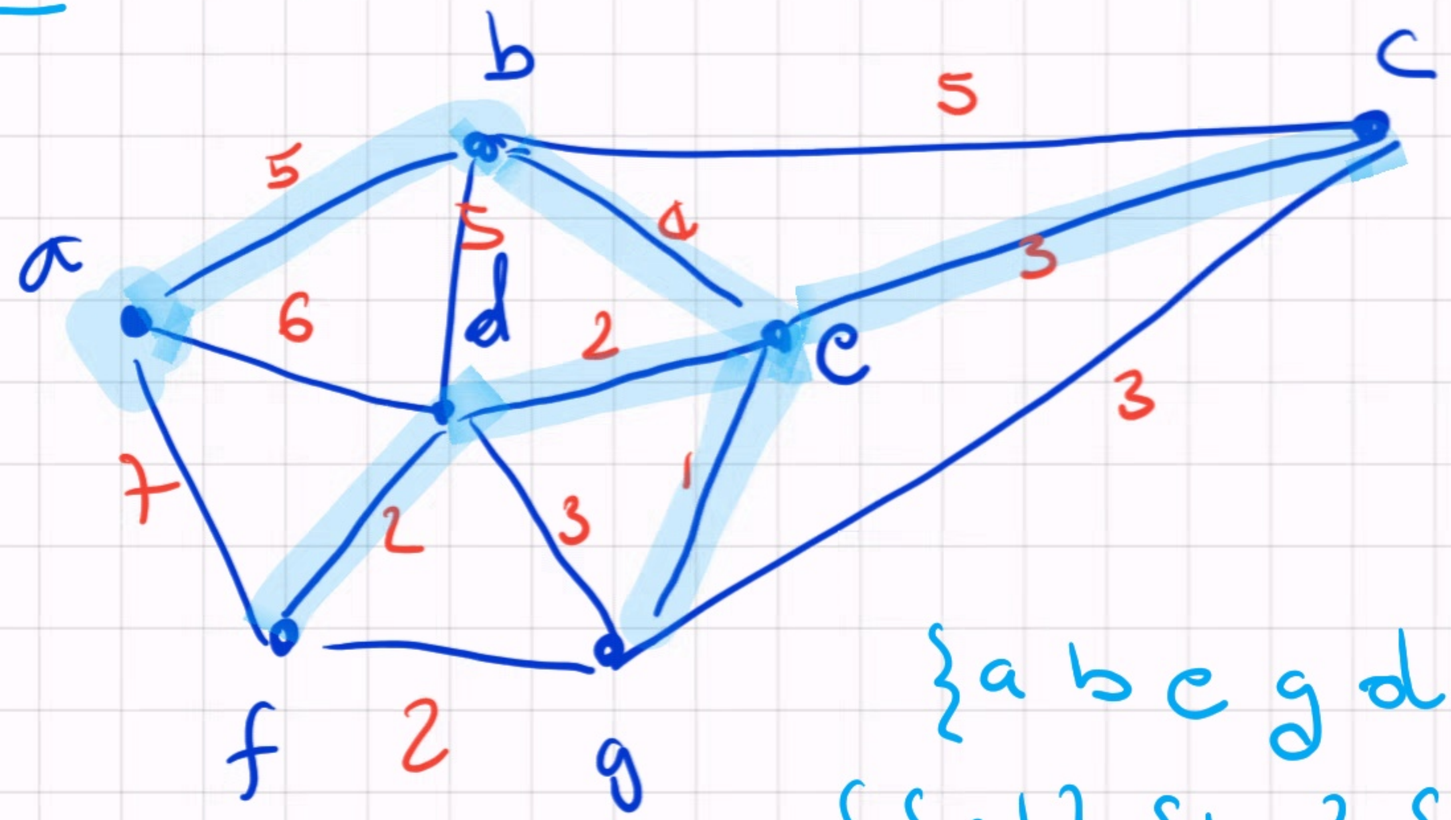
{ab} I have a spanning tree

Prim's Algorithm

Step 1: Set the counter $i = 1$ and place an arbitrary vertex $v_1 \in V$ into set P. Define $N = V - \{v_1\}$ and $T = \emptyset$.

Step 2: For $1 \leq i \leq n - 1$, where $|V| = n$, let $P = \{v_1, v_2, \ldots, v_i\}$, $T = \{e_1, e_2, \ldots, e_{i-1}\}$, and $N = V - P$. Add to T a shortest edge (an edge of minimal weight) in G that connects a vertex x in P with a vertex y ($= v_{i+1}$) in N. Place y in P and delete it from N.

Example



{a b e g d f c}

{{ab} {be} {eg} {ed}

{df} {ec}}.

Set up $P = \{a\}$ $N = V(G) - \{a\} = \{b\ c\ d \dots g\}$

$E(T) = \phi$

First iteration $P = \{a, b\}$ $N = \{c \dots g\}$

$E(T) = \{\{a, b\}\}$

Second iteration $P = \{a, b, e\}$ $N = \{c\ d\ f\ g\}$

$E(T) = \{\{ab\}\ \{be\}\}$

{a, b, e, g}

{ {ab} {be} {eg} }

{ a b c e g d }

{ . - - - . {ed} }

{ . - - - . f }

{df} }