Algorithms & Complexity

1. Basics

What is an algorithm?
What does "complexity" mean?
What is a datastructure?

- word "algorithm" has its root in Latinizing the name of Muhammad ibn Musa al-khwarizmi (~780-850)
  [Persian mathematician, astronomer, ...)
  → he wrote Thesis "On the Hindu-Arabic numeral system"
  → Translated into Latin: "Algoritmi on the number of Indians"
- first computer prog was written by Ada Lovelace (1843 for the Analytical Engine (by Charles Babbage) to compute Bernoulli numbers.

Algorithm (informal) every well-defined computable? procedure which:

- has as input (finite) set of values
- applies a sequence of operators on values
- has as output (finite) set of values.

[we will make this precise later (Turing machines)]

Datastructure = Object to store & organize data to get efficient access / modification / organisation of data

= speed in terms of runtime + "economical" use of resources.

1.1. What is an algorithm?

The definition of "algorithm" goes back to Alan Turing (1912-1954)
[english mathem. computer scient. logician, physiosophe ....]

Def 1.1 A calculation rule for solving a problem is called **algorithm**
$\Leftrightarrow$ $\exists$ Turing machine equivalent to this calculation rule which stops for every input that has a solution.

→ what is a Turing machine (TM) ?

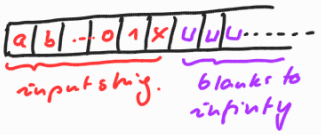
https://plato.stanford.edu/entries/turing-machine/

TM = theoretical machine that simulates the operating principles of a computer [CPU]

TM consist of:

- infinite tape divided into cells

- input string at initial state q_0
- blank symbol "␣"
- head that can read / write / move left or right by 1 position



[input string has no "␣"-symbol]

[rule: · head starts at left-most pos.
· if we are at left-most pos & command is "L", then don't move]

- control (set of rules $\hat{=}$ transition function δ)

+ two finite states: accept state
reject state

A computation on TM can either:

- HALT & accept
- HALT & reject
- FAIL TO HALT

Def. 1.2 (simplified version)

A TM is a 5-tupel $M = (Q, \Sigma, q_0, \delta, F)$

Q: finite nonempty set of states

Σ: finite nonempty set of symbols

$q_0 \in Q$: initial state

$F \subseteq Q$: set of final or accepting states

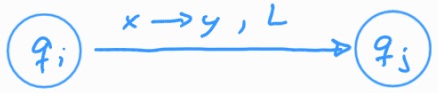
$\delta: Q \times \Sigma \longrightarrow Q \times \Sigma \times \{L, R\}$ transition function

[IF δ not defined for current state THEN HALT.]

▷ Exmpl: $(q_i, x) \longmapsto (q_j, y, L)$

" IF at start of state q_i the head reads symbol x THEN goto state q_j, write y (in pos. of x), & move head by 1 pos. to the left "

≠ FINITE STATE REPRESENTATION (visual)



"control"

[$x \to y$ also by $x|y$]

There is a special symbol $b \in \Sigma$: blank symbol [the only symbol that is allowed to appear infinitely often in each step of computation & not part of input-string]

Turing essentially showed that any computation that can be done by mechanical means can be formulated by some TM.

Def 1.3 A language $L \subseteq \Sigma^*$ (Σ^* - all words (strings) over alphaset Σ without blanks)

Language L is TM-recognizable, if $\exists$ TM that accepts it.

also called "recursively enumerable languages"

TM-Exmpl 1 $L \subseteq \Sigma^*$ with $\Sigma = \{0,1\}$

s.t. $L = \{00, 010, 0110, 01110, \dots\}$

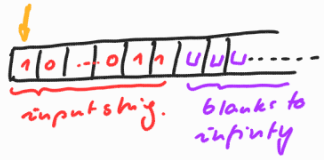
that is L consist of all string 01^*0 for any number ≥ 0 of 1's between 0 & 0

Q: is L TM-recognicable?

⇒ let's build a TM for L:

the input will be some string $\in \Sigma^*$ & we must design TM that decided on whether $s \in L$ or not.

1. get init state q_0 :



(q_0) ⇒ there are now 3 cases: Head reads one of the symbols $0, 1, \sqcup$ ("empty input")

⇒ $0 \rightsquigarrow$ continue, $1, \sqcup$ stop

this must be expressed as transition function:

$\delta: (q_0, 0) \longmapsto (q_1, 0, R)$

$(q_0, 1) \longmapsto (q_{reject} = \text{"HALT \& reject"}, 1, R)$

$(q_0, \sqcup) \longmapsto$

$\equiv q_1 / q_{reject}$.

(q_1) ⇒ there are 3 case HEAD reads $0, 1, \sqcup$

⇒ $0 \rightsquigarrow$ input string so-far 00 ....

⇒ $\delta: (q_1, 0) \longmapsto (q_2, 0, R)$ "check if next symbol is ␣"

$\sqcup \rightsquigarrow$ input string: 0␣ ..... "reject".

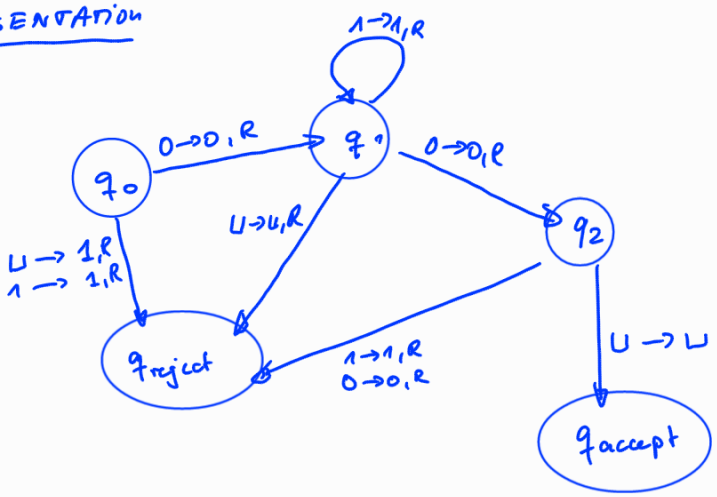
$\delta: (q_1, \sqcup) \longmapsto (q_{reject}, \sqcup, R)$

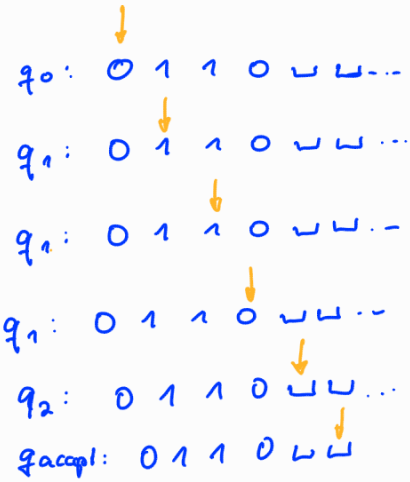
$1 \rightsquigarrow$ input string: 01 ... "check next symbol"

$\delta: (q_1, 1) \longmapsto (q_1, 1, R)$

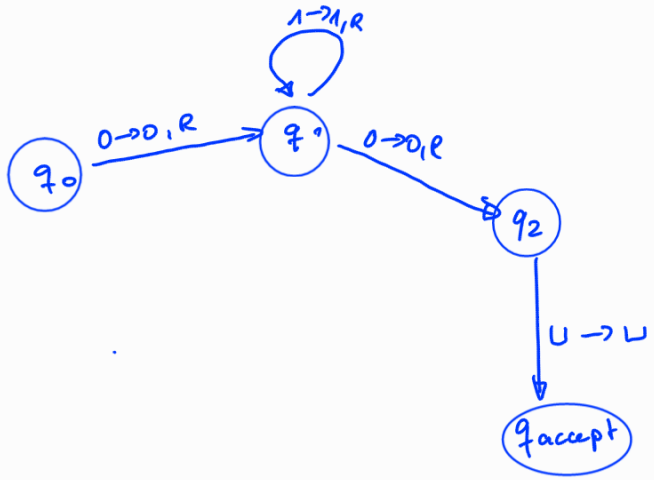
.... and so on....

Finite STATE REPRESENTATION
TM:



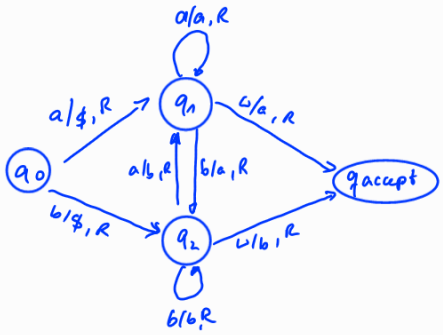


Symplified FINITE STATE REPR:
just draw those arcs that yield "q accept"
all missing arc mean by default: "reject".



TM - Exmpl 2

What is TM with these transition function doing? (input string $\in \{a,b\}^*$)



q0 ↓abaaU...
q1 $↓baaU
q2 $a↓aaU
q1 $ab↓aU
q1 $aba↓U
q3 $abaa↓U
q3 ⋮
↓$ → accept.

⇒ TM add $ at left-most pos & shift each symbol of input string by 1 pos to the right.
at Head is at the end of procedure again on left-most position

$\Sigma = \{a,b\}$

[TM can modify strings]

Variations of TM:

- one Tape vs many Tape
- infinite on both "ends"
- allow head to stay on same place
- allow "non-determinism" [later]

Q: Can this lead to more "powerful" TM?

A: No, all these variants can be shown to be equivalent in the sense that they can be formulated as TM that we just learned.

[NO PROOF]

In particular, TM are useful models of real computers:

- anything a computer can compute, can be computed by TM
- TM describe algorithms.

NOTE: Any computer stores input, programs, ... as a finite binary string & a problem can be expressed as a language over $\Sigma = \{0,1\}$ (or some other alphabet)

+

Exmpl: Is string of form $0^N 1^N$? $L = \{01, 0011, \dots\}$
(problem)

if string in L → accept
else reject.

Any instance of the problem is then any string over Σ

Exmpl: Is given integer n even, ie, $n = 2k$, $k \in \{0,1,\dots\}$

Input: n as binary string $b_1 \dots b_e$

$L = \{ b_1 \dots b_e \mid b_i \in \{0,1\} \ \& \ b_e = 0 \}$

$\Rightarrow$ $n = 5 \hat{=} 101$ odd $\Rightarrow 101 \notin L$

$n = 6 \hat{=} 110$ even $110 \in L$

Def: $f : \Sigma_1^* \to \Sigma_2^*$ computable, if $\exists$ TM st $\forall x \in \Sigma_1^*$ TM halts & has as output $f(x)$
[if f not def on x, then doesn't hold]

Decisionproblem (output: Yes/NO) $L \subseteq \Sigma^*$ is decidable, if $\exists$ TM that halt for every input $x \in \Sigma^*$ & $x \in L \Leftrightarrow$ TM accepts x

L contains all "yes instances".

Of course defining languages + TM can become quite difficult.

> "Algorithm $\Leftrightarrow$ $\exists$ equiv. TM"

Last part should give you a general idea on how algorithms are defined.

FOR US: A procedure is called algorithm if it satisfies

(1) procedure is described in a unique way by a finite Text

(2) Every step in procedure is executable

Sloppy: any compilable procedure, you can write in some programming-language.

(3) in every step only a finite amount of memory is used

(4) must terminate after a finite number of steps

Q:

- Can all problems be solved by algorithms?
- Which problems can be solved in polynomial time & which not?
- Does the structure of a problem "imply" ways to design efficient algorithms to solve it?
- What are "good" datastructures for certain problems?
- What if we can only approximate solutions & how can we prove that finding exact solutions is not "doable"?

A: This course!

FURTHER TM-Exmpls (EXERCISE)

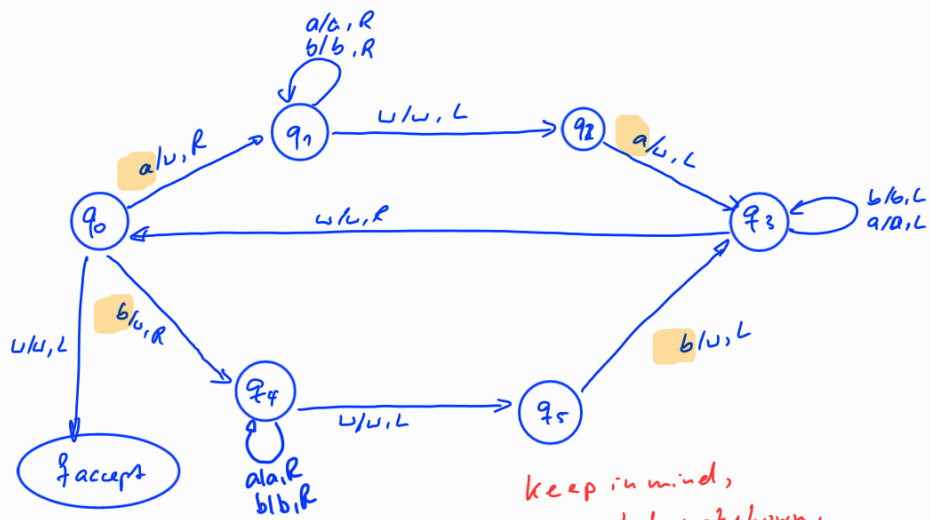
- TM even palindroms

$\Sigma = \{a, b\}$

string $s_1 \dots s_m$ is palindrom, if $s_1 s_2 \dots s_{m-1} s_m = s_m s_{m-1} \dots s_2 s_1$

if m even, palindrom is called even

Exmpl: a b a a b a



keep in mind, any state not shown, leads to REJECT!

- Find TM that recognizes $L \subseteq \Sigma^*$, $\Sigma = \{0,1\}$

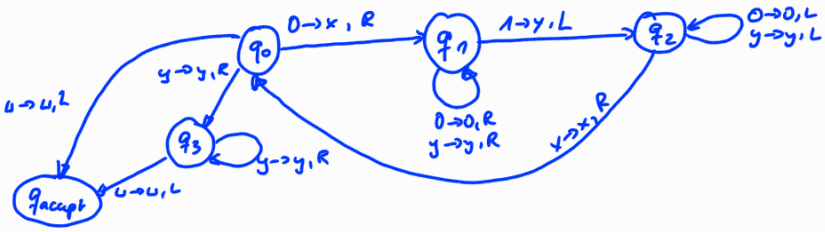
with $L = \{01, 0011, \dots 0^N 1^N, \dots\}$

↗ N 0's followed by N 1's

Pseudo code (accepting states only)

- change 0 → x
- move right to first 1 (if none exists → reject)
- change 1 → y
- move left to leftmost 0
- Repeat above steps until no more 0's remain & make sure that no 1's remain.

0 0 1 1 ␣ ␣
x 0 1 1 ␣ ␣
x 0 y 1 ␣ ␣
x x y 1 ␣ ␣
x x y y ␣ ␣
↳ accept.



- TM to recognize language $0^N 1^N 0^N$

→ have TM to recognize $0^N 1^N$ → repl. & get $x^N y^N$

Step 1: $0^N 1^N 0^N \xrightarrow{TM} x^N y^N 0^N$

Step 2: adj TM to recognize $y^N 0^N$

Step 3: Build final TM by combining these two smaller TM's

- TM to decide whether 2 strings are the same, ie.
  accepted are all words in $L = \{w \# w \mid w \in \Sigma^*, w \neq \#\}$
  (for simplicity $\Sigma = \{a, b\}$)

IDEA: · use new symbol, eg "x"
· replace each symbol by "x" after it has been examined.

Exmpl: a b b a a # a b b a a

↕

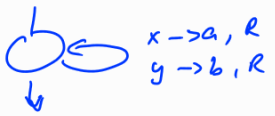
x b b a a # x a b b a a (reg oc. # & check if next symbol is a
go back to 'x' ...

↓

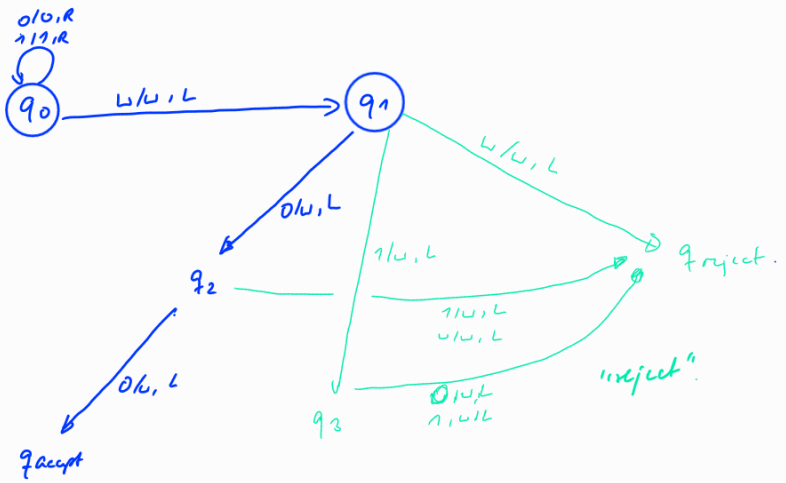
x x x x x # x x x x x

⇒ lost string: "solution" repl. a by x
" b by y

& restore:



- Q: what does this TM recognize?



1.2 Short intro into graph theory

A graph $G = (V, E)$ is a tuple where $V(G) := V$ is the vertex set of G & $E(G) := E$ is the edge set of G.

IF $E \subseteq V \times V$, G is calle directed

IF $E \subseteq \binom{V}{2}$, G is called undirected

$\binom{V}{2}$ ← 2-element subset of V

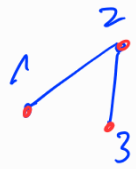
we usually consider directed graphs without loops, i.e. $(v,v) \notin E$.

Visualize

directed graph with

$V = \{1,2,3\}$

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



undirected graph with

$V = \{1,2,3\}$

$E = \{\{1,2\}, \{2,3\}\}$

IF context clear, we write (uv) instead of $\{u,v\}$

! IF not stated differently: "graph" means undirected graph & "di-graph" directed graph.

$u, v \in V$ adjacent $\Leftrightarrow (uv) \in E$

$u \in V$ incident to $e \in E \Leftrightarrow e = (uv)$ or $e = (vu)$ // notation: "$v \in e$"

Neighboorhood: digraph

$N^+(v) = \{u \mid u \in V, (vu) \in E\}$

$N^-(v) = \{u \mid u \in V, (uv) \in E\}$

$N(v) = N^+(v) \cup N^-(v)$

undirected graph:

$N(v) = \{u \mid u \in V, \{uv\} \in E\}$

$N^+(v) = \{2\}$

$N^-(v) = \{1,2\}$

$N(v) = \{1,2\}$

$\deg(v) = 2$

Degree

$\deg^+(v) = |N^+(v)|$

$\deg^-(v) = |N^-(v)|$

$\deg(v) = |N(v)|$

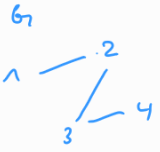
max degree of G: $\Delta(G) := \max_{v \in V} \{\deg(v)\}$

Exercise:

- $\sum_{v \in V} \deg(v) \leq 2|E|$ & equality holds if G is undirected.
- Every (undirected) graph has an even number of vertices of odd degree
- G graph, $|V(G)| \geq 2 \Rightarrow \exists\, u,v \in V$ with $\deg(u) = \deg(v)$.

A (di)graph H is **subgraph** of (di)graph G, in symbols "$H \subseteq G$" if $V(H) \subseteq V(G)$ & $E(H) \subseteq E(G)$

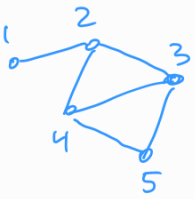
H: 

is subgraph of G: 

H': 

is not subgraph of G.

For $W \subseteq V$, the **induced** subgraph $G[W]$ has vertex set W & contains all edges $(u,v) \in E(G)$ with $u,v \in W$

G: 

$W = \{1,2,4,5\}$

$G[W]$: 

H not induced subgraph

H: 

A graph $G = (V,E)$ is a **complete** graph, denoted by $K_{|V|}$ if $\forall\, u,v \in V$ distinct: $(uv) \in E$.

K_1: • / K_2 / K_3  / K_4  ----

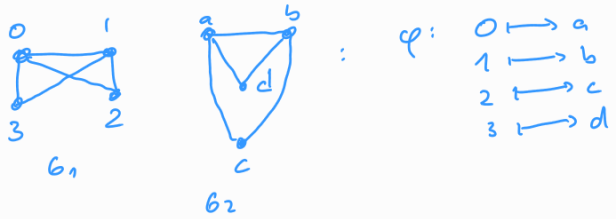
The **complement** $\overline{G} = (V, \overline{E})$ of a (di) graph $G = (V,E)$ has vertex set V & $\overline{E} := \{(uv) \mid u,v \in V \text{ distinct}, (uv) \notin E\}$.

G: 

$\overline{G}$: 

Two (di) graphs $G_1 = (V_1, E_2)$, $G_2 = (V_2, E_2)$ are **isomorphic**
if $\exists$ bijective map $\varphi : V_1 \to V_2$
s.t. $(u,v) \in E_1 \iff (\varphi(u), \varphi(v)) \in E_2$

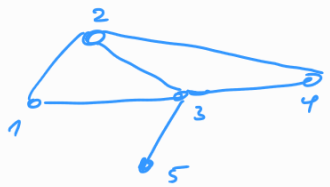
φ is call **isomorphism** & we write $G_1 \cong G_2$



A **path** in a (di) graph $G = (V, E)$ is sequence $\langle v_0, v_1 \dots, v_k \rangle$
of vertices in V s.t. $(v_i, v_{i+1}) \in E$, $i \in \{0, \dots, k-1\}$

A path is **simple** if $v_i \neq v_j$ $\forall\, i \neq j$

A path $P = \langle v_0 \dots v_k \rangle$ is also called **$v_0 - v_k$-path**.



$\langle 1\,3\,4\,2\,3\,5 \rangle$ 1-5-path not simple
$\langle 1\,2\,4\,3\,5 \rangle$ 1-5 path, simple

Subpath of path $\langle v_0 \dots v_k \rangle$ is contiguous subsequence
of its vertices $\langle v_i, v_{i+1} \dots v_{j-1}, v_j \rangle$
subpath of $\langle 1\,2\,4\,3\,5 \rangle$ is eg $\langle 2\,4\,3\,5 \rangle$

We will consider paths also as subgraphs, ie,
Path $P = \langle v_0, \dots v_k \rangle$ in G corresponds to subgraph H of G
with $V(H) = \{v_0, \dots v_k\}$
$E(H) = \{(v_i, v_{i+1}) \mid 0 \leq i \leq k-1\}$

path $\langle v_0 \dots v_k \rangle$ called **cycle**, if $v_0 = v_k$
cycle is **simple**, if $v_1 \dots v_k$ are distinct

A graph that does not contain cycles is **acyclic**

$C \subseteq V$ is <u>connected component</u> of G, if $G[C]$ connected & C is inclusion maximal.

→ 2 conn. comp. $\{1,2,3\}$ $\{x,y\}$.

$G[\{2,3\}]$ connect. subgraph, but no component!

A (di)graph $G = (V, E)$ is (strongly) connected if $\forall u, v \in V$ $\exists$ u-v path in G.

connected graph.

A di-graph is a <u>DAG</u> if it is acyclic // DAG = <u>D</u>irected <u>A</u>cyclic <u>G</u>raph

A graph is a <u>forest</u> if it is acyclic.

A graph is a <u>tree</u> if it is a connected forest

<u>Theorem 1.12:</u> Let $G = (V, E)$ be a graph
The following statements are equivalent:

(1) G is a tree

(2) $\forall u, v \in V$: $\exists!$ simple u-v-path in G

(3) G is connected, but $(V, E \setminus \{e\})$ is disconnected $\forall e \in E$

(4) G is acyclic, but $(V, E \cup \{e\})$ contains cycle $\forall e \notin E$

<u>proof:</u> (1) ⇒ (2). G connected ⇒ $\exists$ simple u-v-path $\forall u, v \in V$.

IF 2 simple paths:

⇒ exists cycle ↯

(2) ⇒ (3) $\forall u, v$: $\exists!$ u-v-path

⇒ $\forall e = (u, v)$ the path $\langle u, v \rangle$ is this <u>unique</u> path

⇒ after removal of e there is <u>no</u> u-v-path

⇒ $(V, E \setminus \{e\})$ disconnected $\forall e \in E$

(3) $\Rightarrow$ (4) Assume, for contradiction, that

G contains cycle (wlog simple)

$\Rightarrow$ removal of e results in connected graph $\lightning$

$\Rightarrow$ G acyclic.

Since G is conn., let $e \notin E$ ($e = xy$)

$\exists$ x ... y path

G + e cont. cycle ✓

(4) $\Rightarrow$ (1) Since G acyclic, it remains to show that G is connected (and thus, a tree)

Let $x, y \in V$

IF $(xy) \in E \Rightarrow \exists$ x-y-path

IF $(xy) \notin E \Rightarrow$ "$G + (xy)$" has cycle (wlog simple)

$\Rightarrow \exists$ x-y path in G

$\Rightarrow$ G connected.

□

Theorem 1.2 Let $G = (V, E)$ be a connected graph

G is a tree $\Leftrightarrow |E| = |V| - 1$

proof: $\Rightarrow$ "

// "

induction over $|V|$

Base case: $|V| = 1 \Rightarrow G \hat{=}$ • $\Rightarrow |E| = 0$ ✓

$|V| = 2 \Rightarrow G \hat{=}$ •—• $\Rightarrow |E| = 1$ ✓

Hyp: Statement true for all graphs with $< k$ vertices.

Let $G = (V, E)$ with k vertices.

Remove edge $e \in E \overset{\text{Thm 1.1}}{\Rightarrow} G' = (V, E \setminus \{e\})$ disconnected.

has 2 connected components C_1 & C_2,

where $G_1 = G[C_1]$, $G_2 = G[C_2]$ must be trees!

$$\Rightarrow |E| = |E(G_1)| + |E(G_2)| + 1$$

$$\overset{\text{ind. hyp.}}{=} |V(G_1)| - 1 + |V(G_2)| - 1 + 1$$

$$= \underbrace{|V(G_1)| + |V(G_2)|}_{= |V(G)|} - 1$$

$\Leftarrow$ 4

Assume $|E| = |V| - 1$, but connected G is not a tree

$\Rightarrow$ G is not acyclic $\Rightarrow$ it contains simple cycles.

Let C_1 be such a cycle & remove $e_1 \in E(C_1)$.

$\Rightarrow$ C_1 is "destroyed" in "$G - e_1$" & "$G - e_1$" remains connected.

repeat the latter, by looking at simple cycle C_2 in "$G_1 - e_1$" & remove $e_2 \in E(C_2)$ & so on

$\Rightarrow$ after k steps we are done

$G' = "G - \{e_1, e_2 \dots e_k\}"$ is a tree.

$\Rightarrow$ $|E(G')| = |V| - 1 = |E(G)|$

$\Rightarrow$ $k = 0$, that is no edges have been removed $\lightning$

$\Rightarrow$ G is acyclic.

$\square$

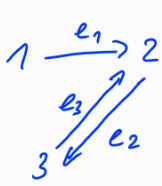
Corollary 1.3 Every connected graph $G = (V, E)$ has tree $T = (V, F)$ as subgraph ($=$ spanning tree) & $|E| \geq |V| - 1$.

Graphs can be represented in several ways:

1) "visual"

2) Adjacency-matrix $A(G)$

is $|V| \times |V|$ matrix s.t. $A(G)_{ij} = \begin{cases} 1, & (i,j) \in E \\ 0, & \text{else} \end{cases}$



$A(G)$:

	1	2	3
1	0	1	0
2	0	0	1
3	0	1	0

3) Incidence-matrix $I(G)$

is $|E| \times |V|$ matrix s.t. $I(G)_{e,i} = \begin{cases} 1, & e=(i,x) \\ -1, & e=(x,i) \\ 0, & \text{else} \end{cases}$

	1	2	3
e_1	1	-1	0
e_2	0	-1	1
e_3	0	1	-1

and many more ...