CPSC-406: Algorithm Analysis, Spring 25

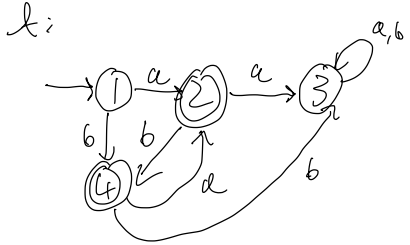
02/18/25

DFA $A = (Q, \Sigma, \delta, q_0, F)$

Accepted lang., experiences from lab:

- brute force
- look at q_0 and F, and analyze
  ⇝ invariants
- implementation

⇝ Product automata:
merge or combine DFAs



Q: What are the langs?

- $L(A)$ = ? alternating words, nonempty: no two consecutive same letters
- $L(B)$ = ? every odd pos. has a

$L(A) = \{a, b, ab, ba, aba, bab, \ldots, ababa, \ldots\}$

$L(B) = \{a, aba, ababa, aaa, \ldots\}$

$\rightsquigarrow$ have elts. in common, i.e. $L(A) \cap L(B) \neq \emptyset$ intersection

$L(A) \subseteq L(B)$? No, e.g. ab

$L(B) \subseteq L(A)$? No, e.g. aaa

Basic set theory:

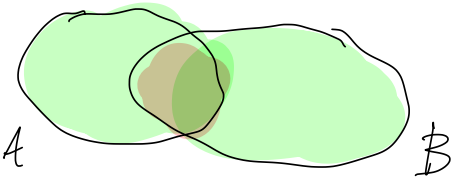
$A \subseteq B$

A subset of B if: $x \in A \Rightarrow x \in B$

Operations:

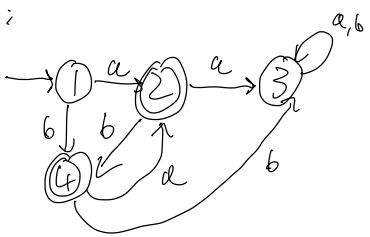
$A \cap B := \{x \mid x \in A \text{ and } x \in B\}$

$A \cup B := \{x \mid x \in A \text{ or } x \in B\}$

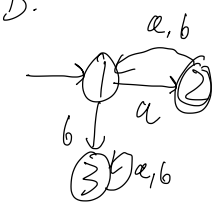


What do we take from this? Can compare languages. But might want to operate on them.

A:



B:



"alt. words"
"L(A)

"a at odd"
"L(B)

$L(A) \cap L(B) = \{$start at a, odd length, alternating$\}$

Q:
How to construct DFA C such that $L(C) = L(A) \cap L(B)$?

A: 1) come up with C based on description.
2) product automaton

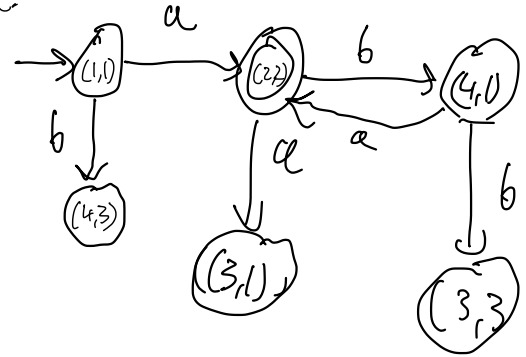
In general:

Given $A^{(1)} = (Q^{(1)}, \Sigma, \delta^{(1)}, q_0^{(1)}, F^{(1)})$,
$A^{(2)} = (Q^{(2)}, \Sigma, \delta^{(2)}, q_0^{(2)}, F^{(2)})$

$\}$

def. $A := (Q, \Sigma, \delta, q_0, F)$

A:

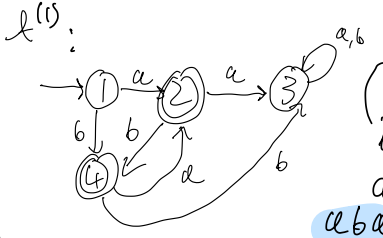


$Q := Q^{(1)} \times Q^{(2)}$

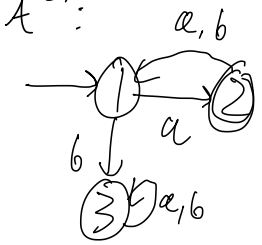
$\delta((q_1, q_2), a) := (\delta^{(1)}(q_1, a), \delta^{(2)}(q_2, a))$

02/20/25 # Operations on automata

Q: Construct A s.t. $L(A) = L(A^{(1)}) \cap L(A^{(2)})$?



$L(A_1)$: non-empty alternating aba, bab



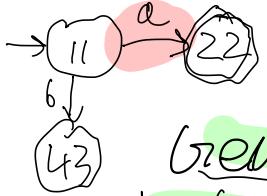
$L(A_2)$: non-empty, odd length, a at every odd index aba, aaa

$\delta((1,1),a) = (2,2)$
$= (\delta^{(1)}(1,a), \delta^{(2)}(1,a))$

Activity: Do this construction for $A^{(1)}$ and $A^{(2)}$.

Idea: label states in A as (p,q), $p \in Q^{(1)}$, $q \in Q^{(2)}$

Start: $pq := (p,q)$



$A^{(1)} = (Q^{(1)}, \Sigma, \delta^{(1)}, q_0^{(1)}, F^{(1)})$

$A^{(2)} = (Q^{(2)}, \Sigma, \delta^{(2)}, q_0^{(2)}, F^{(2)})$

General construction:

$A := (Q, \Sigma, \delta, q_0, F)$

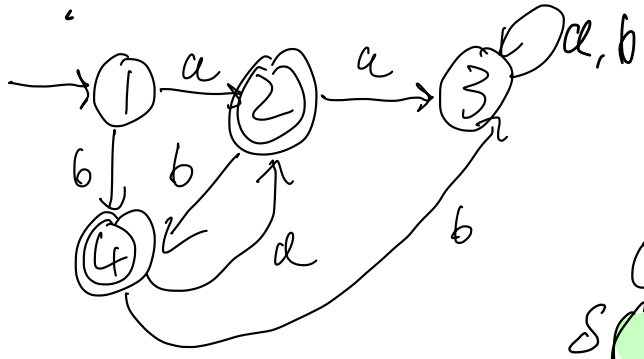
$Q := Q^{(1)} \times Q^{(2)} := \{(p,q) \mid p \in Q^{(1)}, q \in Q^{(2)}\}$,

$\delta((q^{(1)}, q^{(2)}), a) := (\delta^{(1)}(q^{(1)}, a), \delta^{(2)}(q^{(2)}, a))$

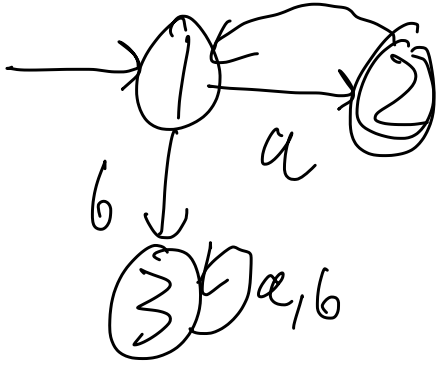
$q_0 := (q_0^{(1)}, q_0^{(2)})$

$F := F^{(1)} \times F^{(2)}$

$A^{(1)}$:



$A^{(2)}$:



General construction:

$A := (Q, \Sigma, \delta, q_0, F)$

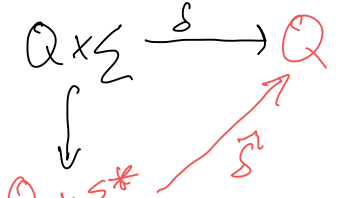
$Q := Q^{(1)} \times Q^{(2)} := \{(p,q) \mid p \in Q^{(1)}\ q \in Q^{(2)}\}$

$\delta((q^{(1)}, q^{(2)}), a) := (\delta^{(1)}(q^{(1)}, a), \delta^{(2)}(q^{(2)}, a))$

$q_0 := (q_0^{(1)}, q_0^{(2)})$

$F := F^{(1)} \times F^{(2)}$

02/25 Recall transition function:



$\delta(q,a)=q'$

$\hat{\delta}(q,w)=q'$

- Σ: letters $\Sigma=\{a,b\}$
- Σ^*: words $\Sigma^*=\{\varepsilon, a, b, aa, ba, \dots\}$

This description can be used for
- proofs, or
- definitions

for words $w\in\Sigma^*$

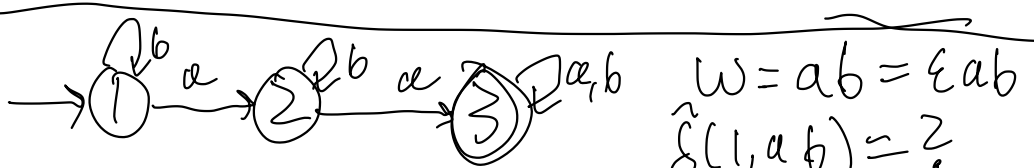
→ Define extended transition function

Inductive description of words:

(1) Base: $w=\varepsilon$ (empty word) $|w|=|\varepsilon|=0$

(2) Induction step: $w=xa$, for some $x\in\Sigma^*$, $a\in\Sigma$ $|w|=|x|+|1|$

$$\hat{\delta}(q,w) := \begin{cases} q & \text{if } w=\varepsilon \\ \delta(\hat{\delta}(q,x),a) & \text{if } w=xa \end{cases}$$

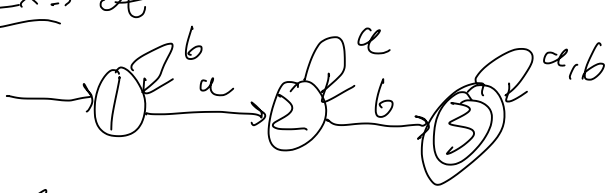


$w=ab=\varepsilon ab$

$\hat{\delta}(1,ab)=?$

$$\hat{\delta}(q,w) := \begin{cases} q & \text{if } w = \varepsilon \\ \delta(\hat{\delta}(q,x),a) & \text{if } w = xa \end{cases}$$

Ex.: $\mathcal{A}$



Σ δ Δ δ

Compute $\hat{\delta}(1, baba)$:

$\hat{\delta}(1,\varepsilon) = 1$

$L(\mathcal{A}) = \{ w \in \{a,b\}^* \mid \hat{\delta}(1,w) = 3 \}$

$\hat{\delta}(1,b) = \hat{\delta}(1,\varepsilon b) = \delta(\hat{\delta}(1,\varepsilon), b) = \delta(1,b) = 1$

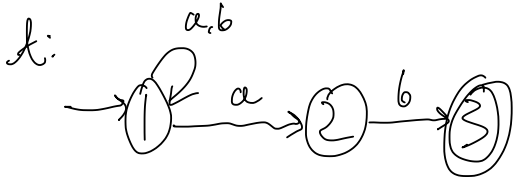
$\hat{\delta}(1,ba) = \delta(\hat{\delta}(1,b), a) = \delta(1,a) = 2$

$\hat{\delta}(1,bab) = \delta(\hat{\delta}(1,ba), b) = \delta(2,b) = 3$

$\hat{\delta}(1,baba) = \delta(\hat{\delta}(1,bab), a) = \delta(3,a) = 3$

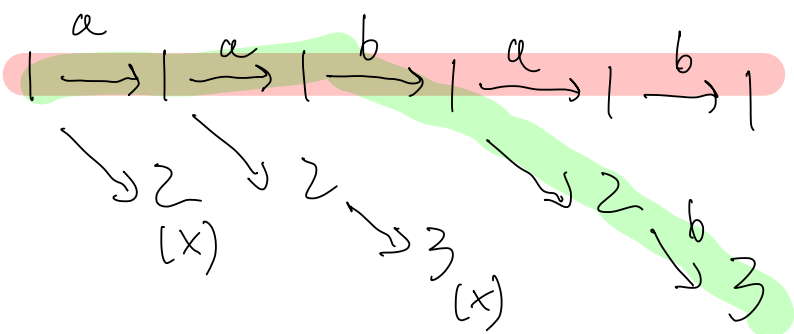
In general: $L(\mathcal{A}) = \{ w \in \Sigma^* \mid \hat{\delta}(q_0,w) \in F \}$

Non-deterministic finite automata (NFAs)



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

$w = aabab$



(x) = stuck

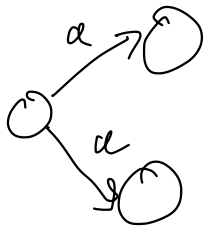
Two paths to process the same word

- red: ends in rejecting state
- green: ends in accepting state

NFA accepts a word if there is some processing path ending in an accepting state

Diff. in transitions:

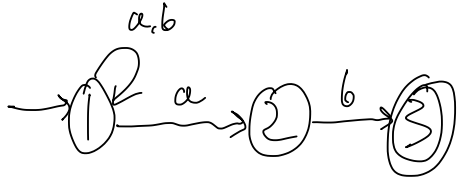
could have

① 

non-uniqueness ("guessing")

② 

Partiality ("undef. inputs")

A: 

$\delta(1,a) = \{1,2\}$

$\delta(2,a) = \emptyset$

Formal def. of DFA:

$D = (Q, \Sigma, \delta: Q \times \Sigma \to Q, q_0 \in Q, F \subseteq Q)$

Formal def. of NFA:

$N = (Q, \Sigma, \delta: Q \times \Sigma \to \mathcal{P}(Q), q_0, F)$

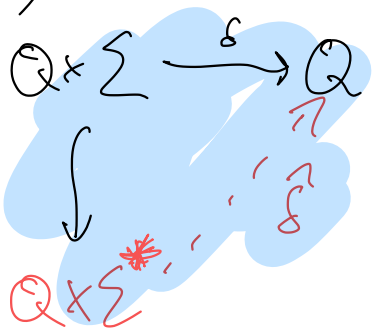
$\mathcal{P}(Q) := \{S \mid S \subseteq Q\}$

(Power set of Q)

4/27

Recap:

1) extended transition fn.:

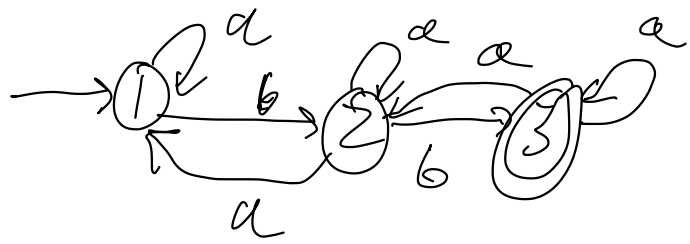


"looks ahead 1 step"

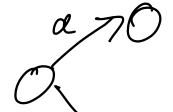
"looks ahead $\ell = |w|$ many steps"

$\hat{\delta}(q, a) = \delta(q, a)$

2) NFAs: $(\Sigma = \{a, b\})$



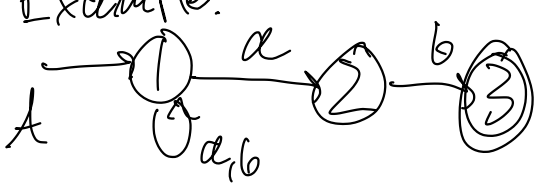
° Diff's to DFA:

-  ambiguity / guessing
-  undefined

DFA: $D = (Q_D, \Sigma, Q \times \Sigma \xrightarrow{\delta_D} Q, (q_0)_D, F_D)$

NFA: $N = (Q_N, \Sigma, Q \times \Sigma \xrightarrow{\delta_N} \mathcal{P}(Q), (q_0)_N, F_N)$

Example:



$A = (Q, \Sigma, \delta, q_0, F)$

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

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

$q_0 = 1$

$F = \{3\}$

$\delta(1, a) = \{1, 2\}$

$\delta(2, b) = \{3\}$

$w = aabab$

$\delta(1, aabab) = \{1, 3\}$

two paths:

$1 \xrightarrow{a} 1 \xrightarrow{a} 1 \xrightarrow{b} 1 \xrightarrow{a} 1 \xrightarrow{b} 1$ ✓ reject

$1 \xrightarrow{b} 1 \xrightarrow{a} 2 \xrightarrow{b} 3$ ✓ accepts

↓

so A accepts!

For an NFA,

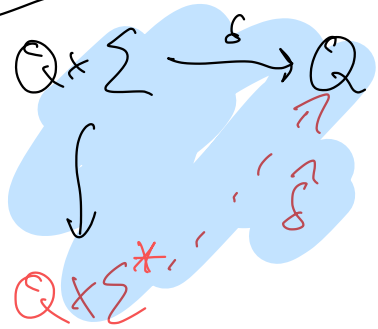
$L(A)$ should be all words that are accepted by at least one path starting at q_0.

Extended transition function:

Want: $\hat{\delta}(q,w)$ for NFAs

Induction on $w \in \Sigma^*$:

DFAs:

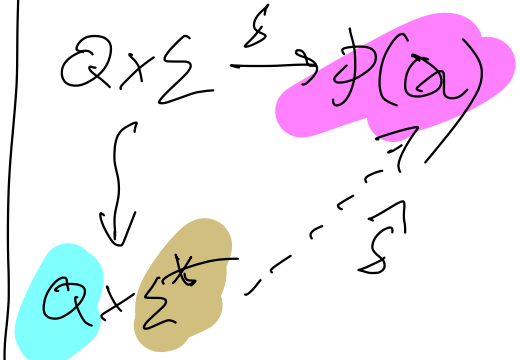


(1) $\hat{\delta}(q,\varepsilon) := q$

(2) $\hat{\delta}(q,xa) := \delta(\hat{\delta}(x,a), a)$

↑ can use $\hat{\delta}(q,x)$ is def.

NFAs:



(1) $w = \varepsilon$:

$\hat{\delta}(q,w) := \hat{\delta}(q,\varepsilon) = \{q\}$

Induction = "Lego principle"

(2) $w = xa$, $x \in \Sigma^*$, $a \in \Sigma$:

$\hat{\delta}(q,w) := \hat{\delta}(q,xa) := \bigcup_{i=1}^{k} \delta(q_i, a)$

assume defined: $\hat{\delta}(q,x) = \{q_1, \ldots, q_k\}$ ← set of states

Induction step

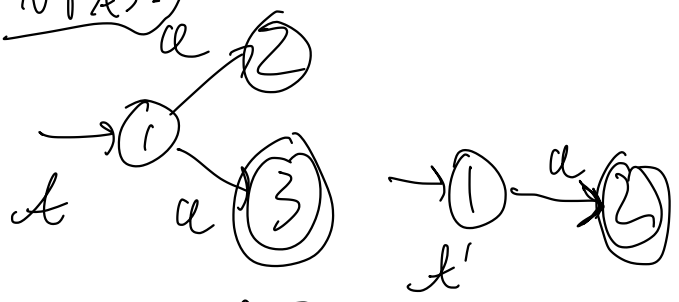
Language of NFA:

$$L(A) = \{ w \in \Sigma^* \mid \hat{\delta}(w, q_0) \text{ contains at least one accepting state} \}$$

$$= \{ w \in \Sigma^* \mid \hat{\delta}(w, q_0) \cap F \neq \emptyset \}$$

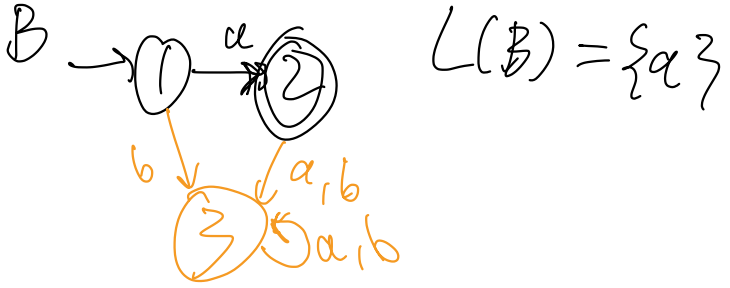
all states that are reachable (from q_0) reading w

NFAs:



$L(A) = \{a\} = L(A')$

DFA (2):



$L(B) = \{a\}$

{p,q,v}

From NFA to DFA (determinization):

{p,q} {p,r} {q,r}

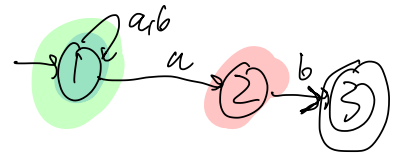
Goal: NFA A ⇝ DFA A_D s.t. $L(A) = L(A_D)$

{p} {q} {r} ∅

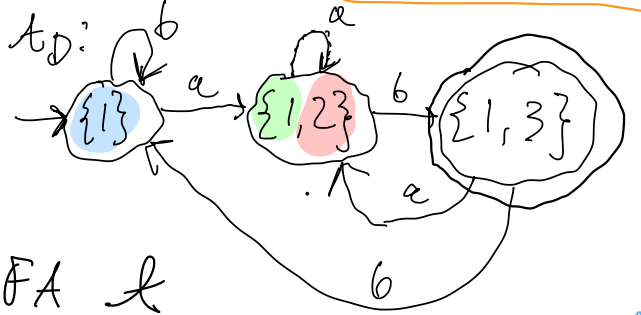
How? Power set construction!

Idea: capture all reachable states

Example: A:



A_D:



General construction:

NFA A

DFA A_D: $|\mathcal{P}(Q)| = 2^{|Q|}$

- $Q_D := \mathcal{P}(Q)$
- $q_{D_0} = \{q_0\}$
- $\delta_D(S \subseteq Q, a) = \bigcup_{q \in S} \delta(q, a)$
- $F_D := \{S \subseteq Q \mid S \cap F \neq \emptyset\}$

a state S in A_D is final if it contains at least one final state in F

$[S = \{q_1, \dots, q_k\}]$

03/04 Regular expressions

$\Sigma = \{0, 1\}$

$R = (0+1)^* 01 (0+1)^*$

"strings with 01 somewhere"

apps: search, lexical analysis, parsing, error det., ...

Each regular expression (RE) represents a language

$L(R) = \{01, 001, 0011, 0, 01, \ldots\}$

Goal:

RE $\leftrightarrow$ regular languages $\leftrightarrow$ DFAs / NFAs

Operations on languages:

① union $\cup$: $L=\{01,10\}$, $M=\{\varepsilon,0,1\}$

$L\cup M=\{01,10,\varepsilon,0,1\}$

② concatenation $\cdot$: $L=\{01,10\}$, $M=\{\varepsilon,0,1\}$

$L\cdot M:=\{01\varepsilon, 010, 011, 10\varepsilon, 100, 101\}$

$=\{01, 10, \ldots\}$

cf. Σ^*

③ (Kleene) closure/star $*$: $L:=\{0,1\}$

$L^*=\{\varepsilon,0,1,00,11,000,111,01,10,\ldots\}=\bigcup_{i=0}^{\infty} L^i$

$L^0:=\{\varepsilon\}$, $L^{i+1}:=L^i\cdot L$ [cool, ...]

Regular expressions:

Induction:

Base: (1) $\varepsilon, \emptyset$ reg.

$L(\varepsilon)=\{\varepsilon\}$, $L(\emptyset)=\emptyset$

(2) $a\in\Sigma \Rightarrow a$ reg

[In the book/lit often **a** (boldface)

$\underline{a}$

'a' (quote)

(3) $L, k, \ldots$ var. (L lang.) $\Rightarrow L, k, \ldots$ reg.

Regular expressions:

Induction:

Base: (1) $\varepsilon, \emptyset$ reg.

$L(\varepsilon) = \{\varepsilon\}$, $L(\emptyset) = \emptyset$

(2) $a \in \Sigma \Rightarrow a$ reg

「In the book/lit often **a** (boldface)

$\underline{a}$

'a' (quote)」

(3) $L, K, \ldots$ var. (L lang.) $\Rightarrow L, K, \ldots$ reg.

Step:

(1) E, F reg. $\Rightarrow E+F$ reg.

0, 1 reg. $\Rightarrow 0+1$ reg.

$L(E+F) := L(E) \cup L(F)$

(2) E, F reg. $\Rightarrow EF$ reg.

0, 1 reg. $\Rightarrow 01$ reg.

$L(EF) := L(E)L(F)$

(3) E reg. $\Rightarrow E^*$ reg.

$L(E^*) := (L(E))^*$

(4) E reg $\Rightarrow (E)$ reg.

$L((E)) := L(E)$

Order of Precedence:

1.) * analogous to
2.) · arithmetic
3.) +

Ex: $R = 01^* + 1 = ((0(1^*)) + 1)$

Want: DFA $\xrightarrow{①}$ RE, $\xleftarrow{②}$

① DFA 2 RE:

In: DFA A

Out: RE R s.t. $L(R) = L(A)$

Kleene's algorithm:

$Q = \{1, 2, \dots, n\}$

$R_{ij}^{(k)}$: Paths i to j with no intermediate $> k$

$L_{ij}^{(k)} := L(R_{ij}^{(k)})$

$i \to q_1 \to q_2 \to \dots \to q_r \to j$ ($q_1, \dots, q_r \le k$)

Def. $R_{ij}^{(k)}$'s by ind.:

1) Base $k=0$:

$$L_{i,j}^{(0)} := \begin{cases} \{a \in \Sigma \mid \delta(i,a) = j\} & \text{if } i \neq j \\ \{\varepsilon\} \cup \{a \in \Sigma \mid \delta(i,a) = j\} & \text{if } i = j \end{cases}$$

$i \neq j$:

a) 

$R_{ij}^{(0)} = \emptyset$

b) 

$R_{i,j}^{(0)} = a$

c) 

$R_{i,j}^{(0)} = a_1 + a_2 + \dots + a_m$

$i = j$: Same as $i \neq j$, but add ε

2) Induction step:

$(k-1) \mapsto k$: $R_{ij}^{(k-1)}$ def.

$$R_{ij}^{(k)} := R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}$$

$$R := \sum_{q \in F} R_{q_0,q}^{(n)} = R_{q_0,q_1}^{(n)} + R_{q_0,q_2}^{(n)} + \dots + R_{q_0,q_\ell}^{(n)}$$

$F = \{q_1, \dots, q_\ell\}$

$$L(R) = \bigcup_{q \in F} L_{q_0,q}^{(n)}$$

Example:

A:

$k=0$: $1^*0(0+1)^* = R$

$R_{11}^{(0)} : \varepsilon + 1$

$R_{12}^{(0)} : 0 \quad (L(z) = \{\varepsilon\})$

$R_{21}^{(0)} : \phi \quad (L(\phi) = \phi)$

$R_{22}^{(0)} : \varepsilon + 0 + 1$

$R_{ij}^{(k)}$: $\leq k$

$$R_{ij}^{(k)} := R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}$$

$k=1$:

$$R_{11}^{(1)} = R_{11}^{(0)} + R_{11}^{(0)} \left(R_{11}^{(0)}\right)^* R_{11}^{(0)} = (\varepsilon+1) + (\varepsilon+1)(\varepsilon+1)^*(\varepsilon+1)$$
$$= \varepsilon + 1^* = 1^*$$

$$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)} \left(R_{11}^{(0)}\right)^* R_{12}^{(0)} = \ldots = 1^*0$$

$$R_{21}^{(1)} = R_{21}^{(0)} + R_{21}^{(0)} \left(R_{11}^{(0)}\right)^* R_{11}^{(0)} = \phi + \phi(\varepsilon+1)^*(\varepsilon+1)$$
$$= \phi$$

$$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)} \left(R_{11}^{(0)}\right)^* R_{12}^{(0)}$$
$$= (\varepsilon+0+1) + \phi(\varepsilon+1)^*0 = \varepsilon + 0 + 1$$

$k=2$: only need

$$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} \left(R_{22}^{(1)}\right)^* R_{22}^{(1)}$$
$$= 1^*0 + (\varepsilon+1)(\varepsilon+0+1)^*(\varepsilon+0+1) = R$$

Minimization:

Given DFA, want DFA for the same language, but as few states as possible.

Strategy:

merge certain states

Equivalence of states:

- A DFA
- $p, q \in Q$ states

Def. Call p and q equivalent iff: for all $w \in \Sigma^*$ either $\hat{\delta}(p, w)$, $\hat{\delta}(q, w)$ both accepting or both rejecting.

Algorithm idea:

1) group together all equiv. states
2) make automaton of equiv. class

Notation:

$p \sim_A q$ or $p \sim q$

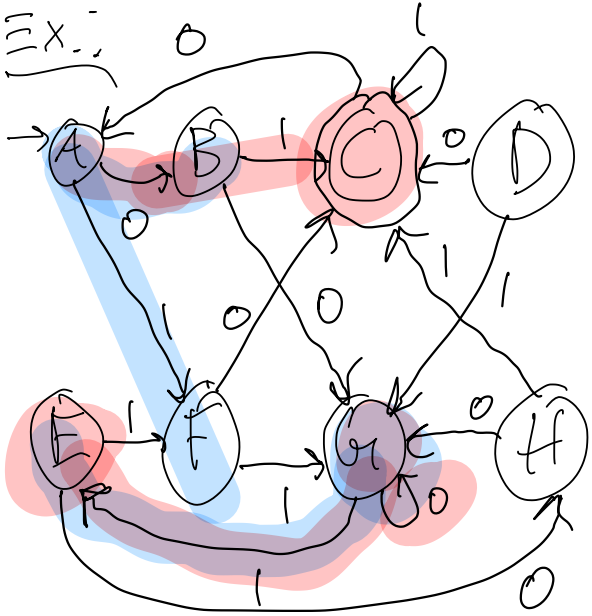
connected to equivalence relations and equivalence classes

$p \sim q$ iff $\forall w \in \Sigma^*: \hat{\delta}(p,w), \hat{\delta}(q,w)$ both acc. or both rej.

Walk: pairs $p \not\sim q$ (not equiv.)

- For $w = \varepsilon$, p and q are only equiv. iff either both acc./rej.
  - ↳ C is not equiv. to anything else

Ex.:



$|Q| = 8$

Is $A \sim G$?

(1) $w = \varepsilon$: ✓

(2) $w = 0$: ✓

$w = 1$: ✓

(3) $w = 01$: ✗ $A \not\sim G$

$\hat{\delta}(A, 01) = C$ accepting

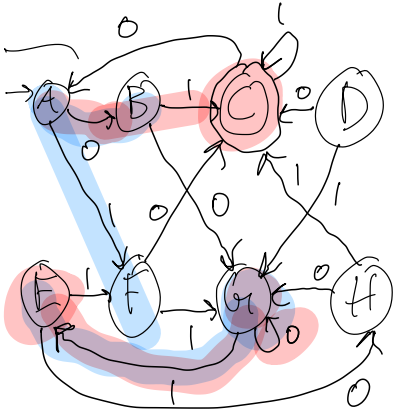
$\hat{\delta}(G, 01) = E$ rejecting

Equivalence of words is an eq.-relation:

(1) reflexivity: $p \sim p$

(2) symmetry: $p \sim q \Rightarrow q \sim p$

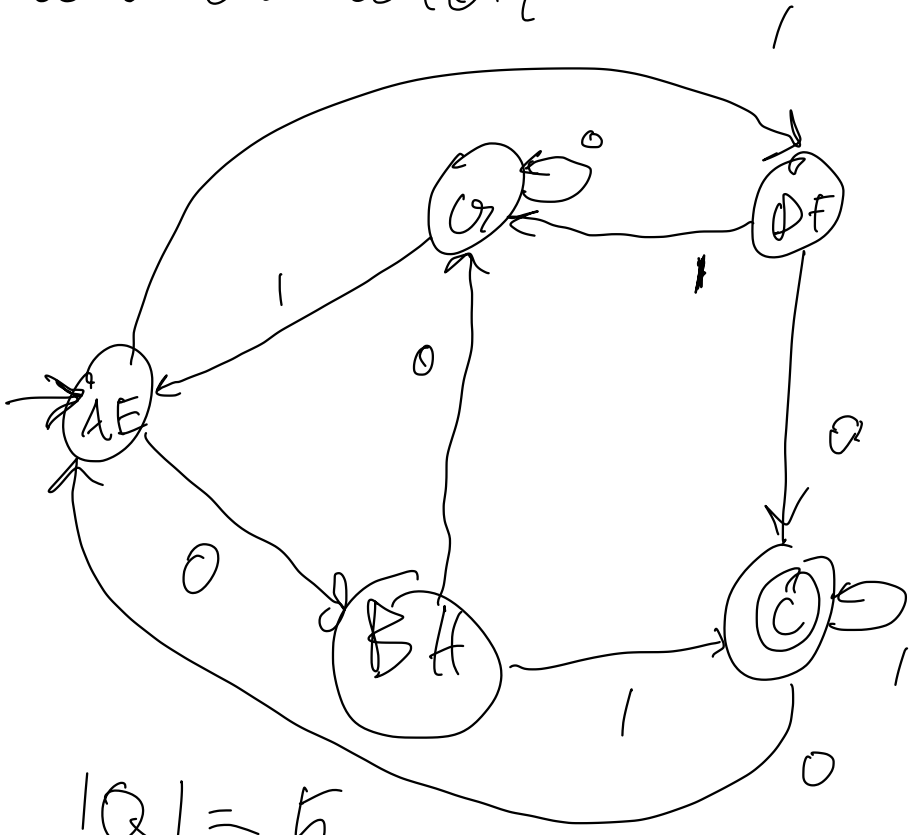
(3) transitivity: $p \sim q, q \sim r \Rightarrow p \sim r$



1) Table filling:

X: non-equiv.

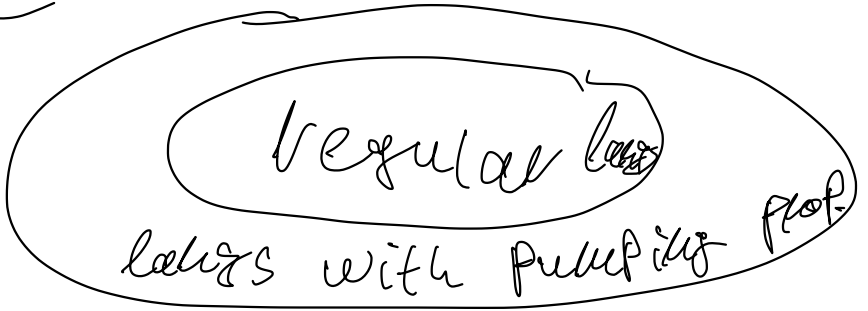
- Create as new automaton



$|Q| = 5$

03/18:) The pumping lemma

- Necessary criterion for regular langs



- Can't use this to prove that L is reg.,
but used to disprove regular:
L reg. $\Rightarrow$ L sat. pumping
L is not reg. $\Leftarrow$ L doesn't sat. pumping

Theorem. Pumping Lemma

Let L be regular.

Then: there exists some $n = n(L)$ (a pumping no. of L)

such that for every word $w \in L$ with $|w| \geq n$, $w = xyz$ such that:

1.) $y \neq \varepsilon$

2.) $|xy| \leq n$ ($\leq n$)

3.) For all $k \geq 0$, the word $w' := xy^k z$ is also in L.

Idea: If L is reg. $\Rightarrow$ recog. by DFA A $\Rightarrow$ for any word $w \in L(A)$ we get a longer word $w' \in L(A)$ by "pumping up w"

Pumping Property:

$w = xyz$ with:

1.) $y \neq \varepsilon$

2.) $|xy| \leq n$

3.) For all $k \geq 0$, the word $w' := xy^k z$ is also in L.

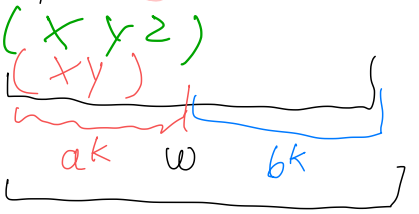
Ex.: WTS $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

Proof, using PL: Assume, L regular. Then, ex. a pumping number k such that all words $w \in L$ with $|w| \geq n$ can be "pumped" in that way ($w = xyz$ s.t. (1)-(3))

Set $w := a^k b^k$, $|w| = 2k > k$.

PL $\Rightarrow$ exist x, y, z such that $w = xyz$, and (1)-(3).

By 2) $|xy| \leq k$:



- xy only consists of a
- By (1): xy contains at least one a.

Pumping: By (3), for any $m \geq 0$, we have $xy^m z \in L$.

Take $m := 2$. Then:

$w' = xy^2 z \in L$. But:

$|w|_a = k = |w|_b = |w'|_b < |w'|_a$

03/20 **Thm (Pumping Lemma):**

For all reg. langs L there exists an $n \geq 1$ such that: for all $w \in L$ such that $|w| \geq n$ there exist x, y, z s.t.: $w = xyz$ and:

(1) $y \neq \varepsilon$ ($\Leftrightarrow |y| \geq 1$)

(2) $|xy| \leq n$

(3) for all $k \geq 0$ also $w' := xy^k z \in L$.

$\forall$: "for all"

$\exists$: "there exists"

$\exists\forall\exists$

Quantifiers:

- logic ($\wedge, \vee, \rightarrow, \leftrightarrow, \exists, \forall$)
- game theory

Proof. Have: L regular $\Rightarrow$ ex. DFA A s.t. $L(A) = L$.
Let $|Q| = n$. Consider: $w = a_1 a_2 \cdots a_m \in L$, where $m \geq n$.

Want: f.a. long enough w, want:

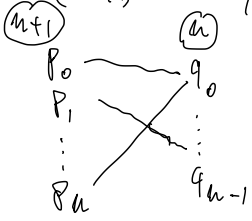
$w = xyz$

$w' = xy^*z \in L$

Def. p_i's accord. to w: For $0 \leq i \leq n$, def.

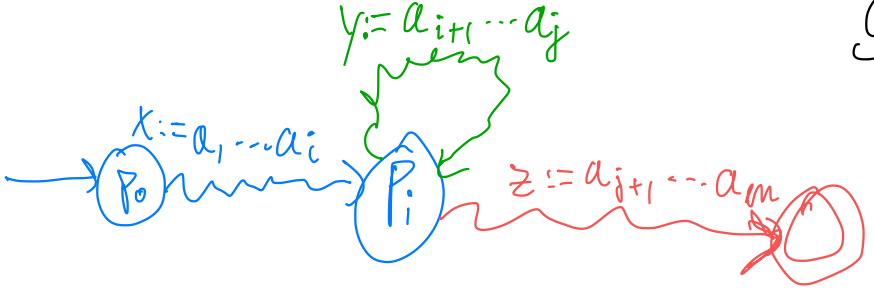
$$p_i := \begin{cases} q_0 & \text{if } i = 0 \\ \hat{\delta}(q_0, a_1 a_2 \cdots a_i) & \text{if } i \geq 1 \end{cases}$$

($n+1$)-many states, but only n states in A (Pidgeonhole principle)



$\Rightarrow$ ex. $0 \leq x < y \leq n$ s.t. $p_i = p_j$

So break apart the path for w as follows:



Check: (1) $y \neq \varepsilon$: $j \neq i$ ✓

(2) $|xy| \leq n$: $|xy| = j \leq n$ ✓

(3) $w' := xy^*z \subseteq L$: By constr. ✓

Application: $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

How to show? Via P.L.: Assume L were regular.

Then would ex. $n \geq 1$ Pumping no.

Take $w := a^n b^n$, $|w| = 2n > n$.

P.L.: $\Rightarrow$ There ex. x, y, z s.t.: $w = xyz$, $y \neq \varepsilon$, $|xy| \leq n$, and f.a. $k \geq 0$: $xy^k z =: w' \in L$.

- $|xy| \leq n \Rightarrow xy$ only cont's a's
- $y \neq \varepsilon \rightsquigarrow xy$ cont's at least one a

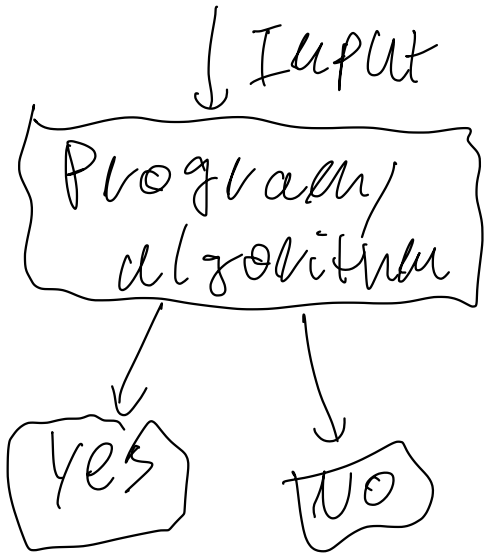
$\Rightarrow k := 2 \rightsquigarrow w' := xy^2 z \in L$, but w' cont's "too many a's now". Contradiction!

- This is an example of proof by contradiction (P.-L., undecidability...)

(Un)decidability

04/08

Decision problem:



Examples:

1) Does str. S contain a substring T?
2) Is an integer n prime?
3) Is a real (rat.) no. x rational?
4) Is a point x reachable from a point y (map)?
5) Does a Prog. P halt on an input w?

→ Translate as acceptance Problems for a TM: $w \in L(M)$?

Def. A lang. L **recursively enumerable** ($r.e.$) or **semi-decidable** if ex. a TM M with $L(M)=L$. ("can say yes")

A lang. L is **decidable** or **recursive** if ex. TM M s.t.: $L=L(M)$, and

(1) $w\in L \Rightarrow M$ accepts ($\Rightarrow M$ holds)

(2) $w\notin L \Rightarrow M$ holds (non-accepting)

("can say yes or no")

Halting problem:

$L_H := \{(M,w) \mid M \text{ TM (code)}, w\in\{0,1\}^*, M \text{ accepts } w\}$

- enough to reduce to binary alphabet
- want to encode M through binary code

$\{M \text{ TM}\} \underset{\text{decode}}{\overset{\text{code}}{\rightleftarrows}} \{0,1\}^*$

If $w\in\{0,1\}^*$ doesn't encode a TM, then decode(w) should be TM w./o. transitions

→ So can view every $\{0,1\}^*$-string as a TM.

→ In turn, we can enumerate all words in $\{0,1\}^*$ by $\mathbb{N}$:

ε	1
0	2
1	3
00	4
01	5
10	.
11	.
000	.
001	
...	

1-1 function

→ $\{0,1\}^* \xrightarrow{\text{num}} \mathbb{N}$

$\leftarrow$

TM $M \xmapsto{\text{code}} w_M \in \{0,1\}^* \xmapsto{\text{num}} \text{num}(w_M) = i \in \mathbb{N}$

So: every TM M has an index $i \in \mathbb{N}$, so it is the i-th Turing machine.

Encode TM_S:

- $Q = \{q_1, \dots, q_k\}$ $k \in \mathbb{N}$
  - q_1 start
  - q_2 accept
- $X_1, \dots, X_s$
  - X_1: 0
  - X_2: 1
  - X_3: B (or ␣)
- $L \rightsquigarrow D_1$, $R \rightsquigarrow D_2$

TODO: ① $\delta(q_i, X_j) = (q_k, X_\ell, D_m) \rightsquigarrow 0^i 1 0^j 1 0^k 1 0^\ell 1 0^m$ for every trans.

② $M: C_1, \dots, C_n \rightsquigarrow C_1 11 C_2 11 C_3 11 \dots 11 C_n$

Summary: Every TM has (at least) one binary code (non-uniquely, but that's ok) $\rightsquigarrow$ Enumeration of all TMs $(M_i)_{i \in \mathbb{N}}$!

Ex.: $M = (\{q_1, q_2, q_3\}, \{0,1\}, \{0,1,B\}, \delta, q_1, B, \{q_2\})$

$\delta(q_1, 1) = (q_3, 0, R) \rightsquigarrow 0^1 1 0^2 1 0^3 1 0^1 1 0^2 = 0100100010100$

From strings, can reconstruct (given the enum.).

Obs.: Every TM has an index in $\mathbb{N}$

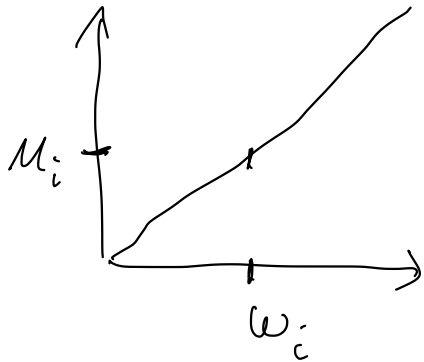
$\Rightarrow$ every possible program has such an index

$\Rightarrow$ ex. no more programs than nat. numbers.

Application: Show existence of undecidable problems!

① Diagonalization Lang.

$$L_D := \{ w_i \in \{0,1\}^* \mid w_i \notin L(M_i) \}$$



Claim: L_D is not r.e.

Proof. Idea: By contradiction. Assume: ex. M s.t. $L(M) = L_D$.

$\Rightarrow M$ has index $M = M_i$.

Q: Is $w_i \in L_D$?

- $w_i \in L_D \Rightarrow M_i$ accept $w_i \Rightarrow w_i \notin L(M_i)$. ↯
- $w_i \notin L_D \Rightarrow M_i$ doesn't accept $w_i \Rightarrow w_i \in L(M_i)$. ↯

$M_i = M$ can't exist!

∎

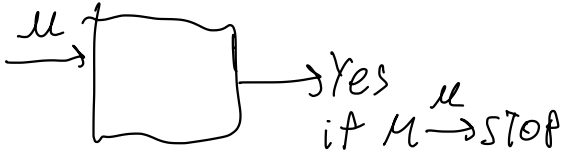
Next time:

- Halting problem
- Practice problems

Halting Problem

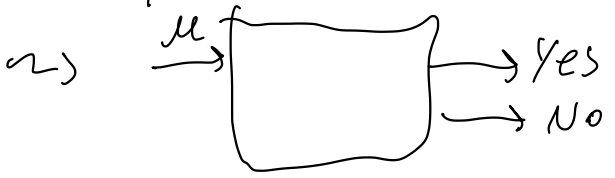
$L := H := \{M \mid M \text{ halts on } M\} =: H$

N.e.:

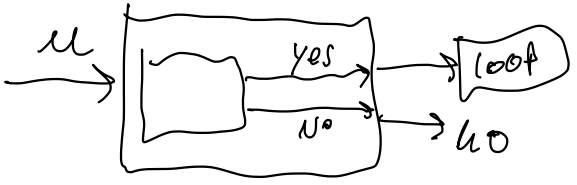


not dec.: ass. M dec. L_1.

M_1:



$\{ M_1'$:



M_1' halts on M_1'

$\Rightarrow M_1' \xrightarrow{M_1'}$ No

$\Leftarrow M_1 \xrightarrow{M_1'}$ No

$\Rightarrow M_1'$ does not hold on M_1'.

Last time: (un)decidability

cf. Lieb's Paradox, barber

- Can we solve certain problems, like

E.g. halting problem (undecidable)

- Problem: 

- (un)decideble?

↪ encode as: $L \subseteq \{0,1\}^*$

Is $w \in L$ or $w \notin L$?

code(M) $\in \{0,1\}^*$ ...

$H := \{M \text{ DTM} \mid M \text{ halts on itself}\}$

1) Can ex. DTM M_H s.t. $L(M_H) = H$ ($\Leftrightarrow (w \in L \Leftrightarrow M_H : w \mapsto \text{YES})$) Yes!
(rec. enumerable)
2) Is H decidable, meaning: $w \notin L \Rightarrow M_H : w \mapsto \text{NO}$? NO!

Rice's Thm: all nontrivial prop's of TMs are undecidable

Complexity theory

04/15

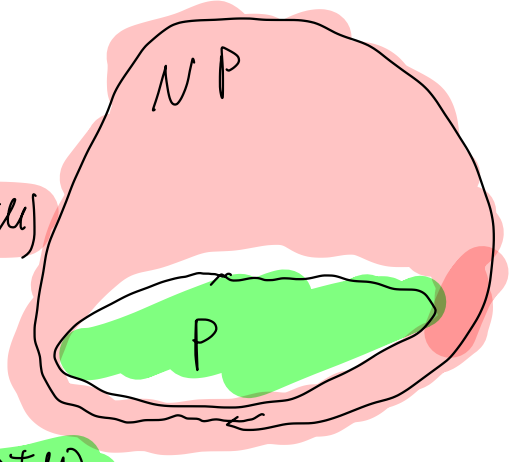
In terms of computational power:

$NTMs \cong DTMs$

P vs NP

NP: nondet. polynomial time (via NTM)

P: Polynomial time (via DTM)



$$p(n) = \sum_{i=1} a_i n^i$$

$n^3 + 2n$ $\quad n^2$

$n^2 + 1$ $\quad n^{1000} + 2n^2 - n + 100$

P ⊂ NP

But P = NP?

Class P:

solvable in poly. time by a det. TM

"Quick to solve"

- sorting
- finding primes
- hashing
- Rubik's cube
- multiplication

Class NP:

solvable in poly. time by a nondet. TM

"Quick to check"

- traveling salesman TSP (Vehicle) routing)
- Sudoku
- protein folding
- public-key cryptography

Time complexity:

A TM M has time complexity $T = T(n)$, if for input w, $|w| = n$, M stops after at most $T(n)$ steps.

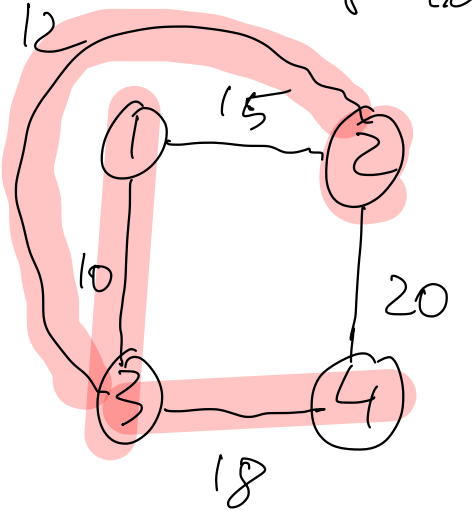
$\rightsquigarrow$ Landau notation / Big O

$T(n) = n^3 + 2n + 1 \in O(n^3)$

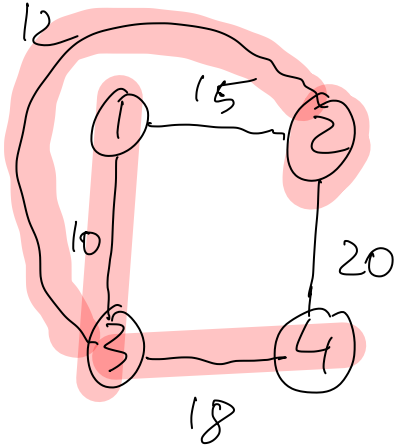
Poly. time P:

Kruskal's algo.

Prob: In a weighted graph, find a min.-wted spanning tree (no cycles!)



MWST



1. (1,3)
2. (2,3)
3. discard (1,2)
4. (3,4)
5. marked 3 = 4 − 1 edges → (✓)

Kruskal:

① maintain conn. comp. for each node
- Initially pts are isolated

② Pick lowest edge not considered yet

If conn. diff. comp.: mark & merge
Else: disregard

③ repeat until either all edges seen, or # marked edges = # vertices − 1

Runtime for kruskal =

Implement on a computer:

$G = (V, E)$ $e = |E|$ $m = |V|$

- find lowest edge: $O(e)$
- check components: $O(m)$

} per each edge

$\rightsquigarrow O(e(e+m))$

improve to $O(m + e\log(e))$

Implement on a DTM (multitape): translate

- Only Y/n answer (if Y/n hard, then computing it is hard)
- Overhead on input, but won't change complexity

Multitape TM:

Input: wt-ed graph as $w \in \{0,1\}^*$

Summary:
- one round: $O(n)$
- at most $e \leq n$ rounds

$\rightsquigarrow O(n)$

$T(n) = O(n^2)$

(1) one tape: store nodes + comm. comp. $O(n)$

(2) one tape: cache current min. edge (if not mkd)
$\rightsquigarrow$ saving of previously unmarked steps on input tape (sep. track): $O(n)$

(3) if edge selected: place nodes on another tape, search for components $O(n)$

(4) another tape: to cache two nodes, that potentially get merged (scan & update: $O(n)$)

Thm. If mTM takes k steps, then ex. STM taking k^2 steps.

For Kruskal:

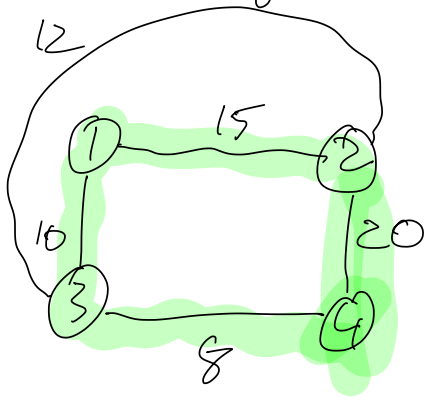
$T_{mult}(n) = n^2$

$T_{single}(n) = n^4$

↳ still polynomial

An example for NP:

Traveling salesman



Find: Hamiltonian cycle, min.
(visit each vertex ex. once)

In gen. $O(2^n)$ cycles!

With an NTM: guess cycles, permute $O(n^2)$, compute total wts $O(n^2)$ ⇝ $T(n) = O(n^4)$

Recall: $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$

$f \in O(g)$ if and only if there are $M_f, N_f > 0$ s.t.

$f(n) \leq M_f \cdot g(n)$, for all $n > N_f$.

Ex. 2: (1) Reflexivity:

$f \in O(f)$ (here $g := f$)

Want: M, N s.t.

$f(n) \leq M \cdot f(n)$

for all $n > N$.

$M := 1 \rightsquigarrow f(n) \leq f(n)$, $N := 1$ ✓

(2) $O(cf) = O(f)$ for all $c > 0$.

In particular, for a polynomial

$P(n) = a_k x^k + a_{k-1} x^{k-1} + \dots + a_1 x + a_0$

$O(P(n)) = O(x^k)$

Have to show:

(1) $O(f) \subseteq O(cf)$

(2) $O(cf) \subseteq O(f)$

...

Have to show:

(1) $O(f) \subseteq O(cf)$

(2) $O(cf) \subseteq O(f)$

for any $c > 0$.

(1):

Let's assume

$g \in O(f)$

$\Rightarrow$ ex. M, N s.t.

$g(n) \leq M f(n)$, f.a. $n \geq N$.

Want: $M', N' > 0$ s.t.

$g(n) \leq M' \cdot c \cdot f(n)$, f.a. $n \geq N'$.

We know:

$g(n) \leq M f(n)$ (for large n)

$g(n) \leq M' \cdot c \cdot f(n)$

- $c = 1$: Take $M' := M$, $N' := N$.
- $c > 1$: $g(n) \leq M f(n) = M \cdot 1 \cdot f(n)$

  $< M \cdot c \cdot f(n)$,

  $M' := M$, $N' := N$.
- $c < 1$: $g(n) \leq M f(n) \leq M' c f(n)$

  want: $g(n) \leq M' c f(n)$

  $(< M' f(n))$

Sufficient that:

$M \leq M' c$, $M' \geq \lceil \frac{M}{c} \rceil$

(for n large enough)

(3) $f, g, h : \mathbb{N} \to \mathbb{R}_{>0}$:

$O(f) \subseteq O(g) \Rightarrow O(f+h) \subseteq O(g+h)$

$f_0 \in$ ⇝ show: $f_0 \in$

If "$f \leq g$", then also "$f+h \leq g+h$".

Example:

$O(n) \subseteq O(n^2)$, $h(n) := \log(n)$

$\Rightarrow O(n+\log(n)) \subseteq O(n^2+\log(n))$

Proof: Take $f_0 \in O(f+h)$

$\Rightarrow$ ex. M, N s.t. f.a. $n > N$:

$f_0(n) \leq M\,[f(n)+h(n)]$,

$= M f(n) + M h(n)$

We know that: $f \in O(f)$

$\Rightarrow f \in O(g)$

$\Rightarrow f(n) \leq M' g(n)$ f. a. $M', N' > 0$ and all $n > N'$!

$f_0(n) \leq M f(n) + M h(n)$

$\leq M' g(n) + M h(n)$ ← $M'' := \max(M, M')$

$\leq M'' g(n) + M'' h(n)$

for all $n > N'' := \max(N, N')$.

Another important prop.:

(4) $f(n) \leq g(n)$ for "large" n

(i.e. ex. N s.t. $f(n) \leq g(n)$ for all $n > N$),

then $O(f) \subseteq O(g)$

↳ $O(\log(n)) \subseteq O(n)$

$O(n) \subseteq O(n^2)$

↝ cf. Ex. 4

Abuse of notation sometimes:

$f(n) = n^3 + 2n^2 + O(\log(n))$

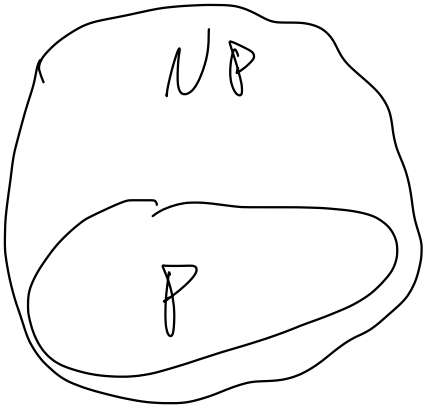
$h(n)$

$O(f)$ is a set of functions, e.g.:

$O(2^n) = \{c, n, 3n+2, \log(n), n^2, n^3, 5000\log(n+1), 2^n, 10 \cdot 2^n, \dots\}$

04/24 CONSTRAINT SATISFACTION PROBLEM (CSPs)

Recall:



$$P = \{ L \mid \text{ex. } M \text{ DTM with } L(M)=L \text{ and } T_M(n) = p(n), \text{ f.s. polynomial } p \}$$

$$NP = \{ L \mid \text{ex. } M \text{ NTM with } L(M)=L \text{ and } T_M(n) = p(n), \text{ f.s. polynomial } p \}$$

In practice:
Often instead of an exact algo. in NP, rather use approx. alg. that is in P.

P feasible \ in practice

NP \ P infeasible

Q: How to show that some L is in NPC?

$\rightsquigarrow$ polynomial-time reduction

Idea: If $L_1 \notin P$, and can reduce L_2 to L_1, then $L_2 \notin P$.

Reduction should be in $O(n^k)$, f.s. k, and input length n.

Notation: $L_1 \leq_p L_2$

Examples:

TSP, SAT ~

NP-complete Problems:

(1) $L \in NP$

(2) f.a. $L' \in NP$ we have $L' \leq_p L$.

Thm. If P_1 NP-complete, $P_2 \in NP$, and $P_1 \leq_p P_2$, then: P_2 is NP-complete.

Proof. WTS: All $L \in NP$ poly-red. to P_1.

- Have: (1) $L \leq_p P_1$, say in $O(P(n))$

$w \in L \rightsquigarrow x = w'$, $|x| \leq P(n)$
$|w| = n$

(2) $P_1 \leq_p P_2$, say in $O(q(n))$

$\rightsquigarrow w \in L \overset{O(P(n))}{\rightsquigarrow} x \in P_1 \overset{O(q(n))}{\rightsquigarrow} y \in P_2$
$n = |w|$
$y \in L_2$ in $O(q(P(n)))$

For example:

$P(n) = n^2 + 2n$

$q(n) = n^3 + n^2 + 5$

$q(P(n)) = (n^2+2n)^3 + (n^2+2n)^2 + 5$

$= n^6 + \alpha n^5 + \beta n^4 + \ldots \in O(n^6)$

$L \leq_p P_1 \leq_p P_2$ also Poly.

in time $O(P(n) + q(P(n)))$

poly poly poly

NB: If we could show that some NP-complete problem is in P, then P=NP. (!!)

The satisfiability problem SAT:

Q: Given a boolean expression, is it satisfiable?

Truth value assignment $\rightarrow$ $T: BExp \rightarrow \{0,1\}$

A boolean expr. are built:

(1) Variables: $x, y, z, \ldots$ can be 0 or 1

(2) binary OPs: OR $\vee$ (1) and AND $\wedge$ (2)

(3) unary OP: NEGATION $\neg$ (3)

(4) parentheses: ( )

- $x \wedge \neg(y \vee z)$

$T(x) = 1 \quad T(z) = 0$

$T(y) = 0$ Sat! (4)

- $x \wedge (\neg x \vee y) \wedge \neg y$

$T(x) = 1$

$T(y) = 0$ not Sat! (5)

Thm (Cook-Levin): SAT is NP-complete.

Proof. Have to show:

(1) $SAT \in NP$

(2) $L \in NP \Rightarrow L \leq_p SAT$

Ad (1): multitape NTM to guess assignments and eval.

$O(n^2)$ on mNTM

$[\rightsquigarrow O(n^4)$ on SNTM$]$ ✓

Ad (2):

Hard! B/c: universal reduction.

Two statements:

- $L \leq SAT$ (reduction)
- red. is polynomial.

Let $L \in NP \Rightarrow$ ex. M sNTM s.t. M takes $O(P(n))$ steps ($n = |w|$ input size), and $L(M) = L$.

May assume:
- M never writes a B
- M never moves left of start

$\rightsquigarrow$ If $|w|=n$, M acc. w, then exists a run:

(1) have α_0 initial ID w/ input w

(2) $\alpha_0 \vdash \alpha_1 \vdash \cdots \vdash \alpha_k$, $k \le p(n)$

(3) α_k accepting

(4) each α_i has n nonblanks (unless for edge case), and moves left to right

Strategy:

(a) $\alpha_i = X_{i0} X_{i1} \cdots X_{i\,p(n)}$

one symb. a state, others tape

(b) create boolean var's Y_{ijA} for "$X_{ij} = A$"

($i, j \in \{0, \ldots, p(n)\}$, A tape sym. or a state)

$\alpha_i = x_{i0} x_{i1} \cdots x_{iP(n)}$

y_{isA}

(c) Express that α_i accepts w, through a boolean expr:
(sat. only if $\leq P(n)$ moves)

(i) "start right"

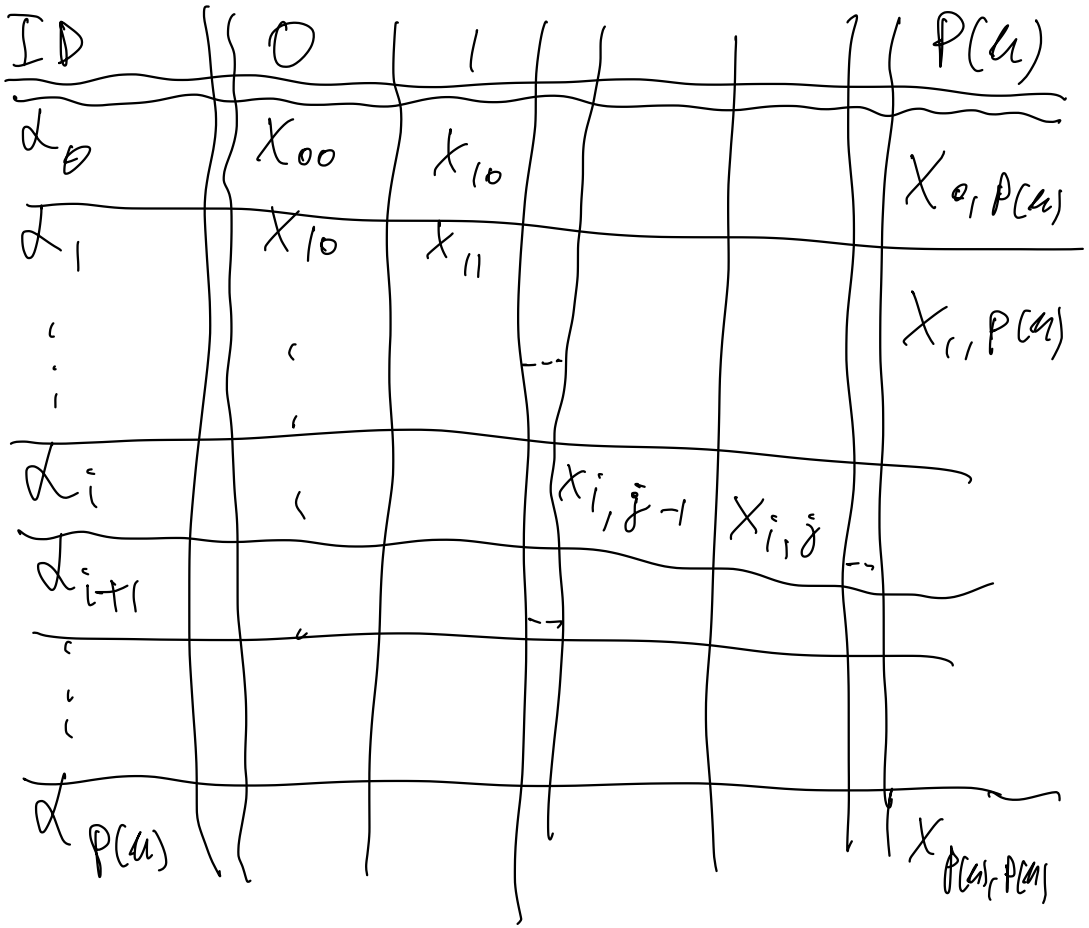
(ii) "next move is on the right"

(iii) "finishes right"

Simplification:

- all ID's have len. $P(n)+1$
- all ID's have exactly $P(n)+1$ moves
(allow empty trails. $\alpha + \alpha$)

$(P(M)+1)^2$ – array:



Want to model Valid Moves!

Want:

$(M,W) \mapsto E_{M,W} = M_1 S_1 M_1 F$

- W: unique symbol
- S: Stuck right
- M: move right
- F: finish right

- Uniqueness u:

$$u := \bigwedge_{i,j,A \neq B} \neg(y_{ijA} \wedge y_{ijB})$$

$A \neq B$, i,j ↖ big conjunction

↝ length $O((p(n))^2)$

$E_{M,w} := u \wedge s \wedge v \wedge F$

↝ For $w = a_1 a_2 \dots a_n$:

$$s := y_{00q_0} \wedge y_{01a_1} \wedge y_{02a_2} \wedge \dots \wedge y_{0na_n}$$

$$\wedge y_{0,n+1,B} \wedge y_{0,n+2,B} \wedge \dots \wedge y_{0,p(n),B}$$

- start right S:

x_{00} $x_{01} \dots x_{0n}$ $x_{0,n+1} \dots x_{0,p(n)}$

q_0 w $(|w| = n)$ $BB \dots B$

Part III: Graph theory

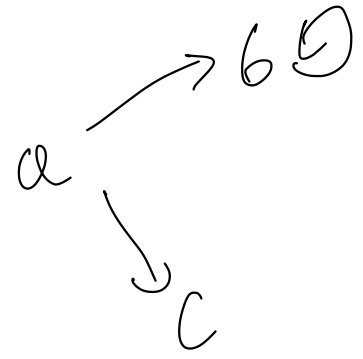
A graph is given as:

$G = (V, E)$, where:

- V: set of vertices
- $E \subseteq V \times V$: set of edges

- no constraints in terms of connectivity
- no labels
- no multi-edges (no ≥ 2 edges from a to b, for any $a, b \in V$)

G:



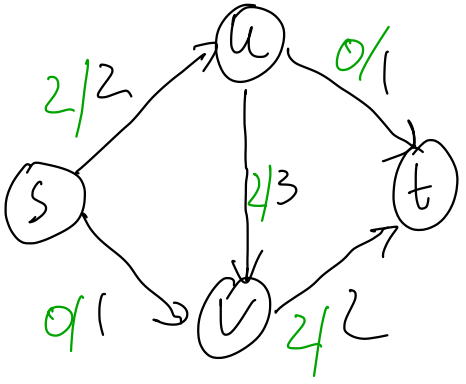
$V = \{a, b, c\}$

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

$(a,b), (b,a)$

Flow networks: graphs with capacities and a "flow"

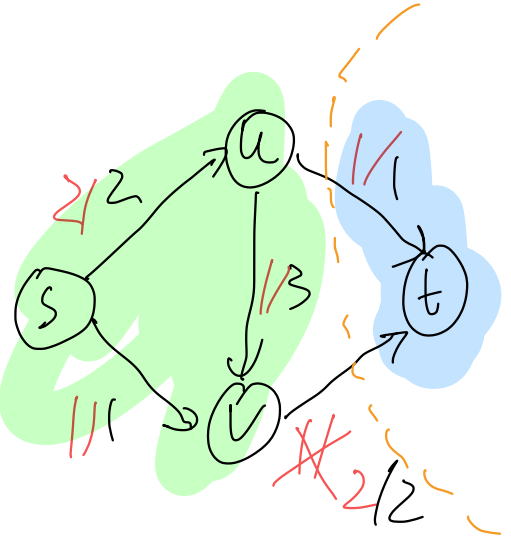
Example:



flow f

$|f| = 2 = 2+0-0$

= #sum of flows out of v − #sum of flows into v ✓



new flow f'

Flow value

$|f'| = 2+1-0 = 3$

$> |f|$

→ actually max!

Objective:

- an algorithm to find a maximal flow ← Ford-Fulkerson alg.
- a way to relate it to a "dual concept": minimal cuts ↔ maximal flows

max-flow min-cut thm.

→ optimization, dual problems

Def. Flow network:

graph $G_N = (V, E, s, t, c)$, where:

- $s \in V$ source
- $t \in V$ target, $s \neq t$

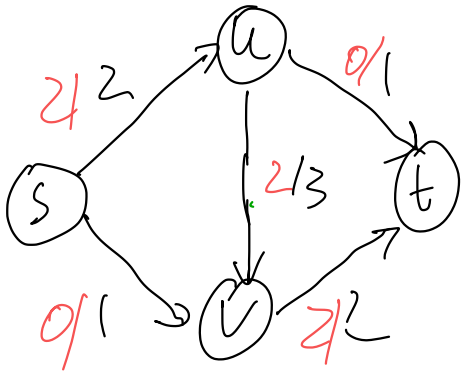
such that for all $v, u \in V$

- $(v,v) \notin E$ (no edges)
- $(u,v) \in E \Rightarrow (v,u) \notin E$ (not symmetric)

and capacities $c : V \times V \to \mathbb{N}_{\geq 0}$, i.e.

- $c(u,v) \geq 0$ (such that $c(u,v) := 0$ if $(u,v) \notin E$)

Ex. flow network:



$|f| = 2 + 0 - 0 = 2$

$\rightsquigarrow$ def. a flow, (not maximal)

Def. Flow:

A flow f in a flow netw. G cons. of values $f(u,v) \geq 0$, f.a. $(u,v) \in V \times V$ such that:

① capacity constraint:

$$f(u,v) \leq c(u,v), \text{ f.a. } (u,v) \in E$$

② conservation: f.a. $v \in V$ with $v \neq s$:

$$\sum_{u \in V} f(u,v) = \sum_{w \in V} f(v,w)$$

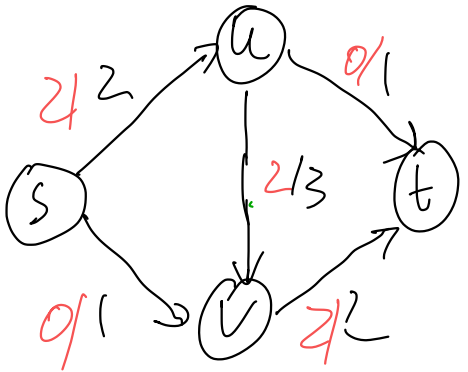
Def. Flow value:

$$|f| := \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)$$

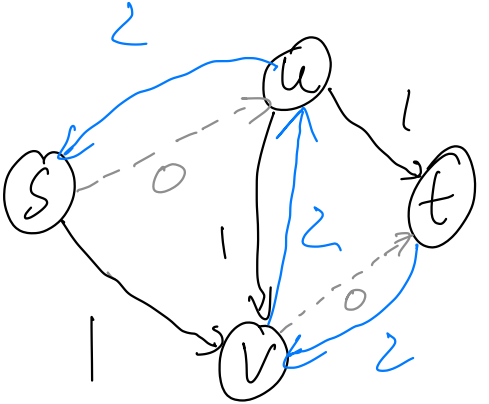
How to find max. flows?

Ford-Fulkerson, idea:

G with flow f:



Residual graph G_f:



Residual cap.
= cap. − flow

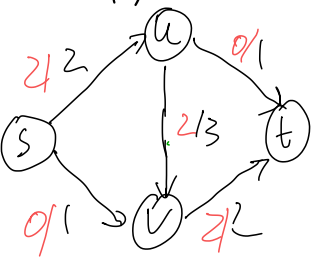
- add backwards edges for the flow $f > 0$
- add forward edges for residual capacity, if > 0

F-F: (1) From (G, f), create G_f.

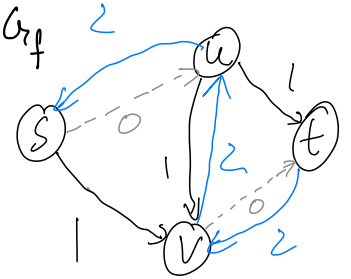
(2) If G_f has an s-t path, augment by this path → update $f \rightsquigarrow f'$

(G, f')

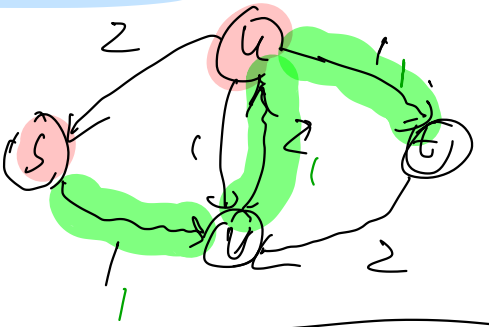
① (G, f)



$\rightsquigarrow$ G_f:

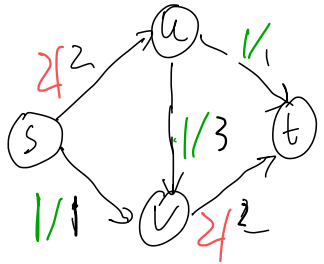


$=$



Find s-t path

② (G, f')



$\rightsquigarrow$ $G_{f'}$:

No s-t path!

Max. flow found!