CPSC-406: Algorithm Analysis, Spring 25

02/18/25

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

- Q: states
- Σ: alphabet
- δ: trans.
- q_0: starting state
- F: final states

- recognize languages:
- modeling systems: parking meter, shopping bank sys.
- flexible formalism
- application: ways of interaction, accepted words certify that behavior is acc. to specification, exhibit certain invariants (cf. rewriting theory)

Feedback from lab:

- tables → find pattern, compare languages
- DFAs ⇄ languages

"→": effective dec word problem

"←": seems difficult; might not be unique, how difficult to control

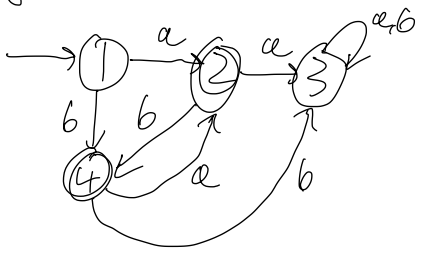
So far:

- worked by hand
- now: start on algo's for DFAs

Comparing DFAs with each other, in a systematic way

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

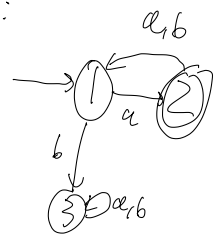
A:



vs.

B:

abc
abaa
aaa



Q: What are $L(A)$ and $L(B)$?

- $L(A)$: non-empty alternating words (i.e., no two consecutive letters are the same)
- $L(B)$: words with odd no. of a's

rec:
- start, end w./ a

Char.: a at odd indices, of odd length

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

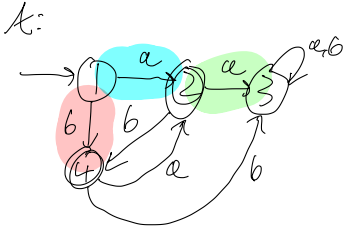
$L(B) = \{a, aaa, abc, aaaaa, abcabc, \ldots\}$

not subsets of each other

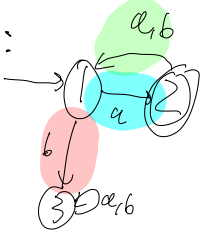
have elts. in common

Q: Can construct an automaton that recognizes exactly the words that $L(A)$ and $L(B)$ have in common, i.e., the intersection $L(A) \cap L(B)$?

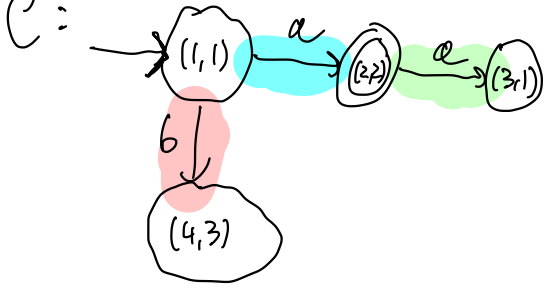
A:



B:



C:



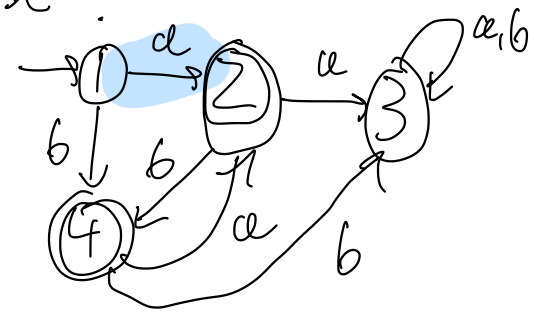
Operations for automata

02/20/25

Q: How to get $L(A^{(1)}) \cap L(A^{(2)})$?

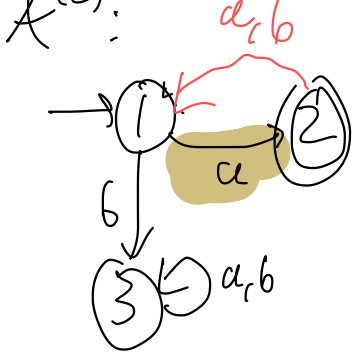
A: Product aut.

$A^{(1)}$:



$L(A_1)$: non-empty, alternating, i.e., no two same consec ltrs

$A^{(2)}$:



$L(A_2)$: odd len, every odd pos. is a

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

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

$(p,q) =: pq$

Generally: ...

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

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

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

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

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

02/25 Recall transition function

$Q \times \Sigma \xrightarrow{\delta} Q$

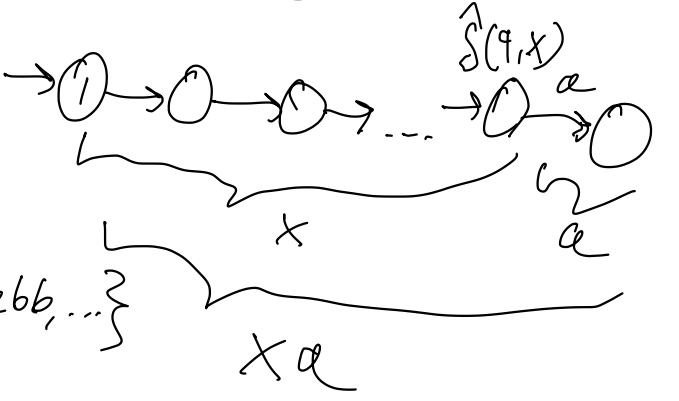
$\downarrow$

$Q \times \Sigma^* \xrightarrow{\hat{\delta}} Q$

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

w word $w \in \Sigma^*$

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



Recall inductive def.: $\Sigma = \{a, b\}$

1) base: $w = \varepsilon$ (empty)

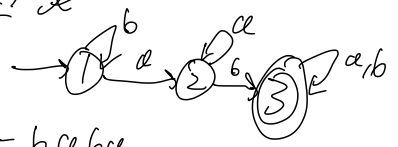
$\Sigma^* = \{\varepsilon, a, b, ab, ba, aa, bb, \ldots\}$

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

$|w| = |x| + |a| = |x| + 1$

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

Ex.: $\mathcal{A}$



$w := baba$

$F = \{3\}$

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

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

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

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

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

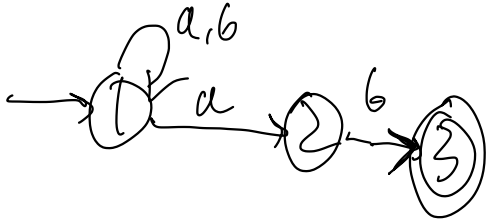
$\hat{\delta}(1,bab) = \delta(\hat{\delta}(1,ba), b) = \delta(2,b) = 3$

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

($baba = x\,a$)

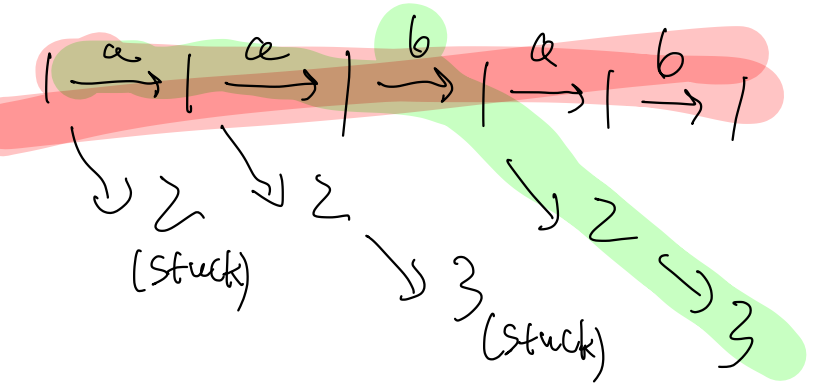
$$L(\mathcal{A}) = \{ w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F \}$$

Non-deterministic finite automata

A: 

NFA

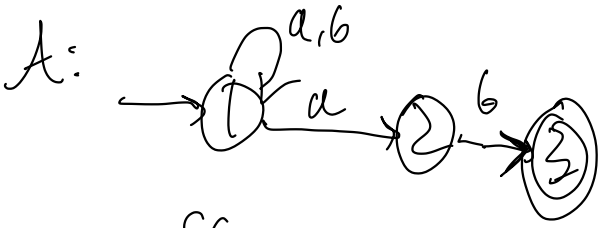
w = aabab



- two paths processing w
  - one rejects
  - one accepting

In an NFA, a word gets accepted if there is (at least) one accepting path.

Formal def's:



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

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

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

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

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

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

$\delta(3,a) = \emptyset = \delta(3,b)$

Power set $\mathcal{P}(Q)$:

alternatively: $\delta \subseteq Q \times \Sigma \times Q$

$\delta = \{(1,a,1), (1,a,2), (1,b,1), (2,b,3)\}$

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

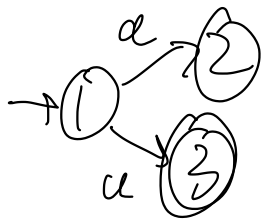
$Q = \{1,2,3\}$, $|Q| = 3 = n$

$|\mathcal{P}(Q)| = 8 = 2^3 = 2^n$

$\mathcal{P}(Q) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

$$|\mathcal{P}(X)| = 2^{|X|}$$

How to describe $L(A)$ for NFA?



should accept a!

want to extend

$$Q \times \Sigma \xrightarrow{\delta} \mathcal{P}(Q)$$

$$\downarrow \qquad \nearrow \hat{\delta}$$

$$Q \times \Sigma^*$$

Def. $\hat{\delta}(q, w)$ via induction:

(1) base $w = \varepsilon$:

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

(2) step $w = xa$:

- assume $\hat{\delta}(q, x)$ is already def., say $\hat{\delta}(q, x) = \{p_1, \dots, p_k\} \subseteq Q$
- $$\hat{\delta}(q, xa) = \bigcup_{i=1}^{k} \delta(p_i, a) = \delta(p_1, a) \cup \delta(p_2, a) \cup \dots \cup \delta(p_k, a)$$

2/27 Recap:

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

1) Extended transition function:

$\delta : Q \times \Sigma \to Q$ "look 1 letter ahead"

What is next step?

$\hat{\delta} : Q \times \Sigma^* \to Q$

Reading w, where do I end up?

Recall inductive def:

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

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

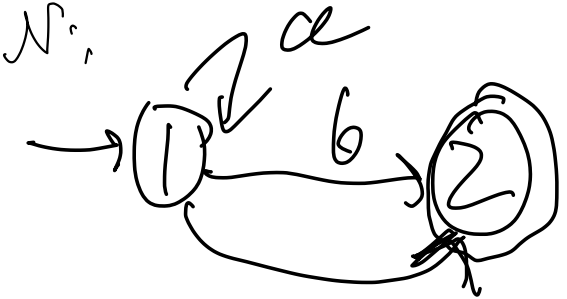
$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$

assume $\hat{\delta}(q, x)$

2) NFAs:

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

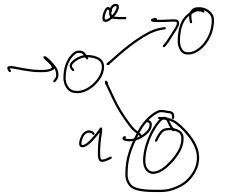
N:



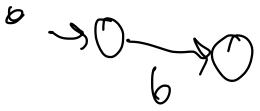
a

Diff's to DFA:

- "ambiguous transitions" ("guessing")



- "partiality"



DFA: $D = (Q_D, \Sigma, \delta_D : Q_D \times \Sigma \to Q_D, (q_D)_0, F_D)$

NFA:

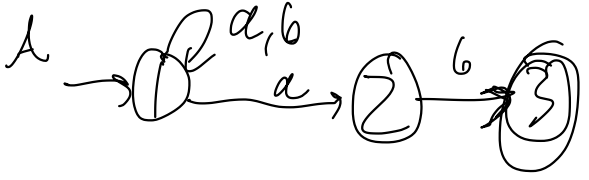
$N = (Q_N, \Sigma, \delta_N : Q_N \times \Sigma \to \mathcal{P}(Q_N), (q_N)_0, F_N)$

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

$\delta_N(1, b) = \{2\}$

$\delta_N(2, a) = \emptyset = \delta_N(2, b)$

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

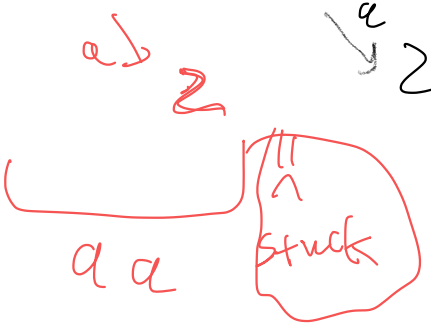


In NFA, if $w \in \Sigma^*$ gets accepted by at least one path, $w \in L(A)$ (started from q_0).

$w = aababb$

$1 \xrightarrow{a} 1 \xrightarrow{a} 1 \xrightarrow{b} 1 \xrightarrow{a} 1 \xrightarrow{b} 1$ rejecting

$1 \xrightarrow{a} 2 \xrightarrow{b} 3$ accepting



Want: $\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$

$\hat{\delta}(1, aababb) = \{1, 3\}$

DFA:

$Q \times \Sigma \xrightarrow{\delta} Q$

$\downarrow$

$Q \times \Sigma^*$ ⋯ $\hat{\delta}$

Def. via induction on $w \in \Sigma^*$:

(1) base: $w = \varepsilon$

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

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

have: $\hat{\delta}(q, x)$

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

NFA:

$Q \times \Sigma \xrightarrow{\delta} \mathcal{P}(Q)$ ← Power set of Q

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

$\downarrow$

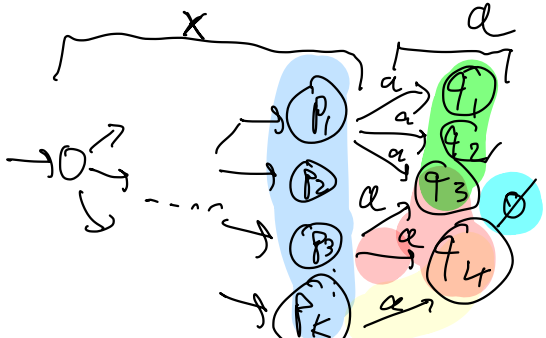
$Q \times \Sigma^*$ ⋯ $\hat{\delta}$

Def. via induction on $w \in \Sigma^*$.

(1) base $w = \varepsilon$

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

(2) step: $x \mapsto w = xa$



Generally:

$$\hat{\delta}(q_1, xa) = \bigcup_{i=1}^{k} \delta(p_i, a)$$

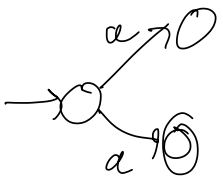
write:

$$\hat{\delta}(q_1, x) = \{p_1, \ldots, p_k\}$$

$\hat{\delta}(q_1, xa) = \delta(p_1, a) \cup \delta(p_2, a) \cup \delta(p_3, a) \cup \ldots \cup \delta(p_k, a)$

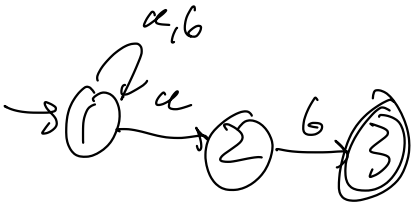
$= \{q_1, q_2, q_3\} \cup \emptyset \cup \{q_3, q_4\} \cup \ldots \cup \{q_4\}$

From NFA to DFA: Determinization



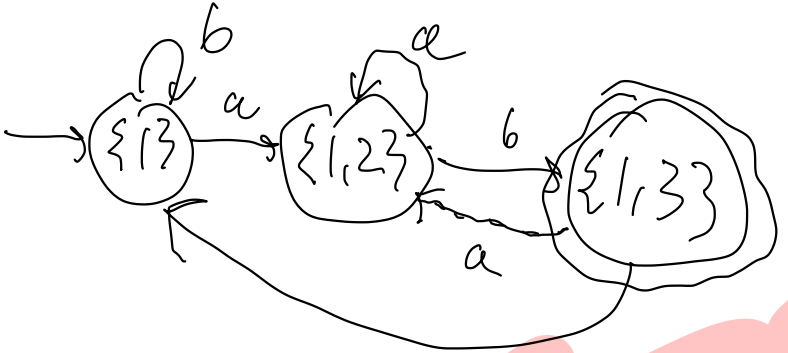
Goal: NFA $\mathcal{A}$ $\rightsquigarrow$ DFA $\mathcal{A}_D$ such that $L(\mathcal{A}) = L(\mathcal{A}_D)$.

Example: $\mathcal{A}$



Idea: Capture all reachable states per step.

$\mathcal{A}_D$:



DFA $\mathcal{A}_D$:

- $Q_D := \mathcal{P}(Q)$
- $\delta_D(S \subseteq Q, a) := \bigcup_{p \in S} \delta(p, a)$
- $F_D := \{S \subseteq Q \mid S \cap F \neq \emptyset\}$
- $(q_D)_0 := \{q_0\}$

03/04 Regular expressions:

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

$\hookrightarrow L(R) = \{$all words cont. $01\}$

$= \{001, 0100, 01110, 01, \ldots\}$

Each regular expression (RE) represents a language.

Applications:

search,
lexical analysis,
parsing,
error detection,
...
File management/
text process
in terminal
à la unix
(find, grep, ls, awk,
sed, ...)

Operations on languages:

(1) Union $\cup$: $L=\{01,10\}$, $M=\{\varepsilon,0,1\}$

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

(2) Concatenation: $L=\{01,10\}$, $M=\{\varepsilon,0,1\}$

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

$=\{01,010,011,10\varepsilon,100,101\}$

(3) (Kleene) closure *: $L:=\{0,1\}$ (compare with Σ^*)

$L^*=\{\varepsilon,0,1,00,01,10,11,000,\dots\}$

$=\bigcup_{i=0}^{\infty} L^i = L^0\cup L^1\cup L^2\cup\dots$,

$L^0:=\{\varepsilon\}$, $L^1:=L$, $L^{i+1}:=L^i\cdot L=\underbrace{L\dots L}_{(i+1)\text{-times}}$

Regular expressions:

Inductive definition:

1) Base cases:

(a) $\varepsilon,\emptyset$ reg.

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

(b) $a\in\Sigma \Rightarrow a$ reg.

Token: $\mathbf{a}$ (boldface)

$\underline{a}$

$'a'$

Regular expressions:

Inductive definition:

1) Base cases:

(a) $\varepsilon, \emptyset$ reg.

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

(b) $a \in \Sigma \Rightarrow a$ reg.

(c) L var $\Rightarrow$ L reg

(L reg. language)

(2) Induction step:

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

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

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

(b) E, F reg $\Rightarrow EF$ reg

0, 1 reg $\Rightarrow 01$ reg

$L(EF) := L(E) \cdot L(F) = \{uv \mid u \in L(E), v \in L(F)\}$

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

$L(E^*) := (L(E))^* = \{\varepsilon\} \cup L(E) \cup (L(E))^2 \cup \ldots$

(d) E reg $\Rightarrow (E)$ reg

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

Order of op's:

3.) "addition"

2.) "multiplic."

1.) "Power"

Order of operations:

1) *
2) ·
3) +

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

Goal:

reg. exp.

$\Updownarrow$

reg. languages

$\Updownarrow$

DFAs/NFAs

First: DFA $\rightarrow$ RE

DFA 2 RE:

In: DFA A

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

Kleene's Thm.

Idea: $Q = \{1, 2, \ldots, n\}$

Build auxiliary REs $0 \le k \le n$

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

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

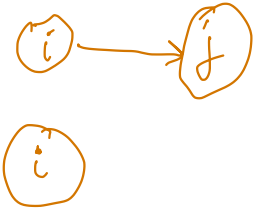
$i \to q_1 \to q_2 \to \ldots \to q_m \to j$

$\le k$

construct REs inductively.

Base $k=0$:

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



If $i \neq j$: a) 

$R_{ij}^0 := \emptyset$

b) 

$R_{ij}^0 := a$

c) 

$R_{ij}^0 := a_1 + a_2 + \ldots + a_l$

If $i = j$: Same a) - c) Plus ε

Ind.-step.: $(k-1) \mapsto k$

Assume: all $R_{ij}^{(k-1)}$ are def.

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

$$R := \sum_{q \in F} R_{q_0 q}^{(n)}$$

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

Read off:

Base $k=0$:

$R_{11}^{(0)}$	$\varepsilon + 1$
$R_{12}^{(0)}$	0
$R_{21}^{(0)}$	$\emptyset$
$R_{22}^{(0)}$	$\varepsilon + 0 + 1$

Step $k=1$

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

$$= (\varepsilon + 1) + (\varepsilon + 1)(\varepsilon + 1)^* (\varepsilon + 1)$$

$$= \ldots = 1^*$$

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

$$= 0 + (\varepsilon + 1)(\varepsilon + 1)^* 0$$

$$= 0 + (\varepsilon + 1)^* 0 = 0 + 1^* 0 = 1^* 0$$

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

$$= (\varepsilon + 0 + 1) + \emptyset (\varepsilon + 1) 0$$

$$= \varepsilon + 0 + 1$$

Ex.: t:

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

$$L(1^*) = L(1^*) \setminus \{\varepsilon\} = \{\varepsilon, 1, 11, 111, \ldots\}$$

$$L(1^*) = \{\varepsilon, 1, 11, 111, \ldots\}$$

$$L(\emptyset) = \emptyset \quad \text{vs.} \quad L(\varepsilon) = \{\varepsilon\}$$

$$L(\emptyset q) = \emptyset$$

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

$$= \emptyset + \emptyset (\varepsilon + 1)^* (\varepsilon + 1) = \emptyset$$

Step $k=2$:

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

$$= 1^* 0 + 1^* 0 (\varepsilon + 0 + 1)^* (\varepsilon + 0 + 1)$$

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

MINIMIZATION, 03/11/25

Intuition: "merge" states if they are equivalent.

Equivalence of states

Let A be a DFA.

p and q equivalent if:

for all $w \in \Sigma^*$ either $\hat{\delta}(p,w)$, $\hat{\delta}(q,w)$ both accept. or both rejecting.

Notation: $p \sim_A q$

$p \sim q$

$\sim_A$ defines an equivalence relation on the states:

(1) reflexivity: $p \sim p$

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

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

Idea Alg.:

- in: DFA A
- out: DFA A_{min} such that: $L(A) = L(A_{min})$ and $|Q_{min}|$ is minimal

1) group together equivalent states

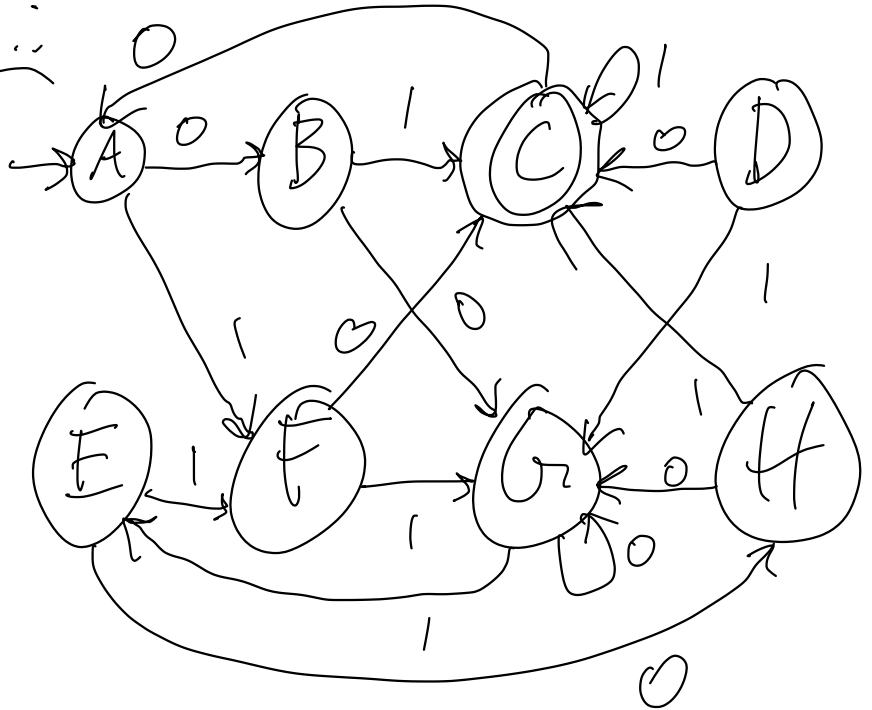
2) construct A_{min} from 1)

Recall: $p \sim q$ iff f.a. $w \in \Sigma^*$: $\hat{\delta}(q,w), \hat{\delta}(p,w)$ both acc/rej.

$\hat{\delta}(p,\varepsilon) = p \Rightarrow$ If $p \not\sim q$, then either both acc. or both rej.

$\Rightarrow C$ not equiv. to anything else

Ex.:



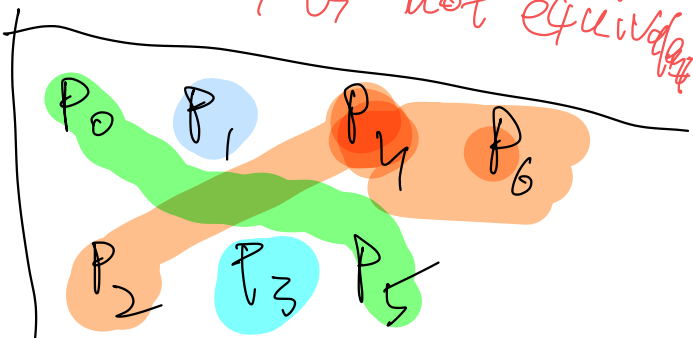
$A \not\sim G$? 1) $w = \varepsilon$: ✓

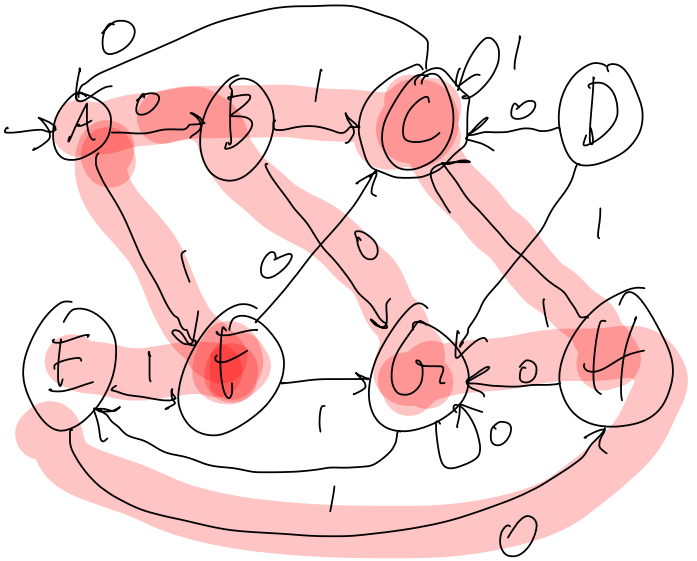
2) $w = 0$: ✓, $w = 1$: ✓

3) $w = 01$: ✗

$A \xrightarrow{0} B \xrightarrow{1} C$ $\quad G \xrightarrow{0} G \xrightarrow{1} E$

$\rightsquigarrow A \not\sim G$ not equiv

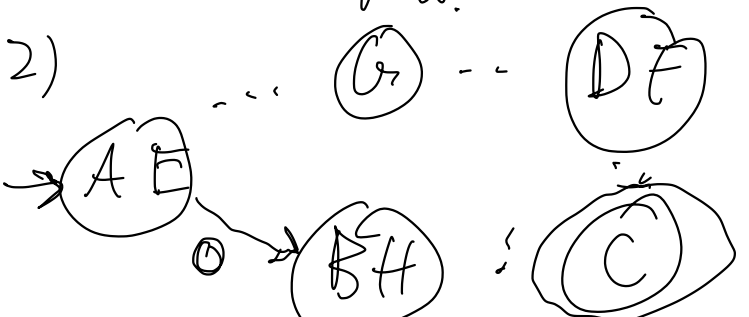




1) Table filling:

X: not equiv.

2)

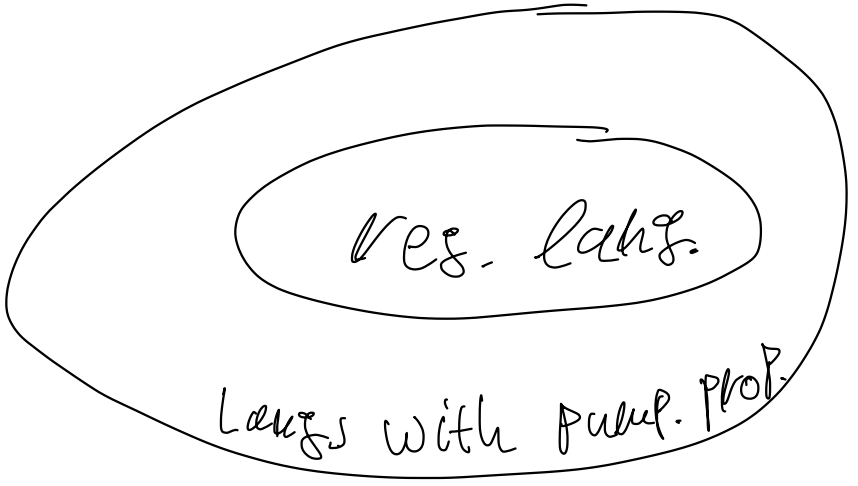


Amin.

5L8

03/18: # The pumping lemma

Pumping property: If $w \in L$, then can "pump it up" to a longer word w', $|w'| > |w|$, $w' \in L$.



L reg. $\Rightarrow$ L P.P.

L not reg. $\Leftarrow$ L not P.P.

$A \Rightarrow B$

NOT $A \Leftarrow$ NOT B

Thm. (Pumping lemma)

Let L be a regular language.
Then there exists a constant $n = n(L)$
(a pumping number of L) such that
for each $w \in L$ such that $|w| \geq n$
there exists a decomposition $w = xyz$ s.t.:

1.) $y \neq \varepsilon$

2.) $|xy| \leq n$

3.) For any $k \geq 0$ the string $w' := xy^k z$ is in L.

Pumping: Ex. n, s.t. f.a. w with $|w| \geq n$, $w = xyz$ s.t.:

1.) $y \neq \varepsilon$

2.) $|xy| \leq n$

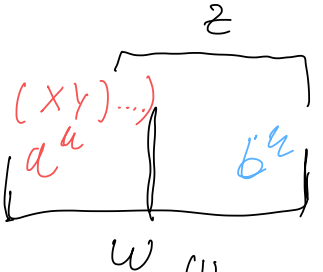
3.) For any $k \geq 0$ the string $w' := xy^k z$ is in L.

← use this to show that pumping gets us out of L!

Ex.: WTS: $L = \{a^n b^n \mid n \geq 1\}$ not regular.

Proof (by P.L.): If L were regular, by PL, ex. $n \geq 1$ s.t. f.a. w ... (see above).

Need: def. $w := a^n b^n$ $(|w| = 2n \geq n)$



By PL, $w = a^n b^n = xyz$, with:

2.) $\Rightarrow$ xy only contains a's.

1.) $\Rightarrow$ xy contains at least one a.

3.) Can pump: Try $k := 2$ (!!) $w' := xy^2z = xyyz \in L$

$|w|_a = |w|_b = n$ (!!)

$|w'|_b = |w|_b$ $\quad |w'|_a > |w|_a = |w|_b = |w'|_b$ $\quad \lightning$ Contradiction: $w' \notin L$ (!)

03/20] Pumping lemma: disprove that a lang. is regular, by proof via contradiction.

In a nutshell: L reg. $\Rightarrow$ L sat. pumping prop.

L not reg. $\Leftarrow$ L not sat. p.p.

Precise thm.: For all reg. langs L there exists $n \geq 1$ such that for all words $w \in L$ s.t. $|w| \geq n$ 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 $k \geq 0$: $w' := xy^k z$ is also in L.

Quantifiers

$\forall \exists \forall \exists$

$\forall$: "for all"

$\exists$: "there exists"

$\rightsquigarrow$ formalization, game theory

Proof. Have: L reg., n p.no.

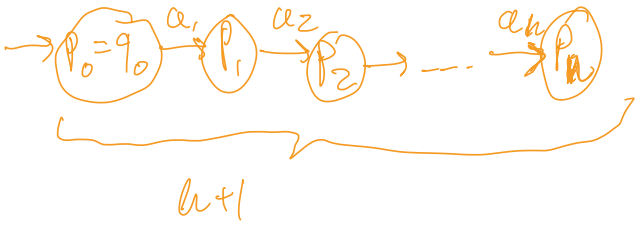
Want: f.a. w, $|w| \geq n$ a decomp. $w = xyz$ such that ...

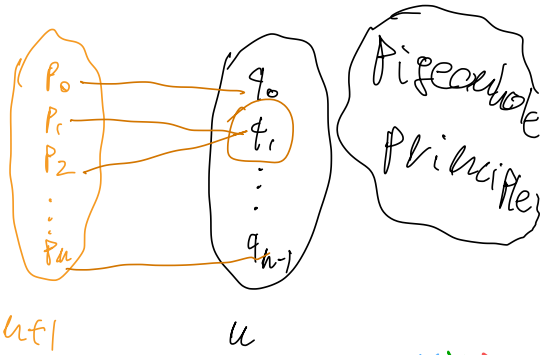
1) L reg. $\Rightarrow$ ex. DFA s.t. $L(A) = L$. Assume A has n states.

2) Let $w = a_1 a_2 \cdots a_m \in L$, such that $m \geq n$.

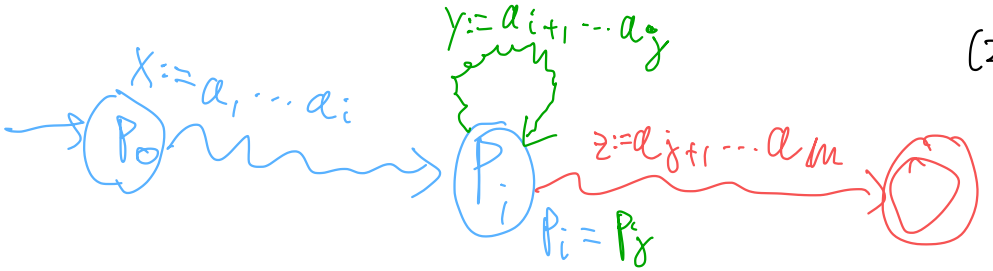
$a_1 \cdots a_n$

Def. p_i's corresp. to $a_1 \cdots a_i$





$\Rightarrow$ ex. $0 \leq i < j \leq n$ s.t. $p_i = p_j$



$w := xyz$:

(1) $y \neq \varepsilon$: b/c $i \neq j$ ✓

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

(3) $\forall k \geq 0$: $xy^k z \in L$

by construction ✓

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

Proof by contradiction: assume L reg.

PL $\Rightarrow$ ex. n pumping no.

Set $w := a^n b^n$, $|w| = 2n > n$, $w \in L$.

PL $\Rightarrow$ ex. x, y, z: $w = xyz$ s.t.:

(1) $y \neq \varepsilon$ → xy can't contain b's

(2) $|xy| \leq n$ → $xy \neq \varepsilon$, hence contains at least one a (and only a's).

(3) $\forall k \geq 0: xy^k z \in L$

$k := 2$, $w' = xy^2z$

$|w'|_a > |w|_a$

$\neq$ $\quad$ $=$

$|w'|_b = |w|_b$

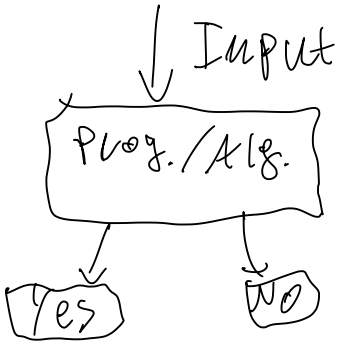
↝ contradiction

$\Rightarrow L$ not regular.

(un)decidability | 04/08

Last week: TMs as a universal model of computation ⇝ every possible algo/prog. can be encoded as a TM.

Decision Problems:



Example:

1.) Does a string S cont. substring T?
2.) Is an integer n prime?
3.) For a real no. x, is $x \neq 0$?
4.) Is a pt. Y reachable from a pt. X?
5.) Does a prog. P halt on an input w?

In logic/ mathi: $\varphi \vee \neg\varphi$ ("φ OR (NOT φ)")

$(x=0) \vee (x \neq 0)$ always true (tautologies).

In computation/CS: not "effectively/computationally" true that $\varphi \vee \neg\varphi$ holds

(e.g.: $(x=0) \vee (x \neq 0)$)

Is $x=y$ over $\mathbb{R}$?

Translate/encode every decision problem as acceptance by a TM: $w \in TM$?

Def.: A language L is recursively enumerable (r.e.) or semi-decidable if there is a TM M with

$$L(M) = L.$$

- L is decidable or recursive if it's r.e., and:
  (1) $w \in L \Rightarrow M$ accepts w ($\Rightarrow M$ halts)
  (2) $w \notin L \Rightarrow M$ halts on w, non-acceptingly

One famous problem: Halting problem

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

H.P.: $(M, w) \in L_H$?

Theorem: undecidable prob.!

Codes for TMs?

- Goal: construct binary codes for TMs (over bin. alph.)

$\rightsquigarrow$ $\mathbb{N}$-enumeration of TMs $(M_i)_{i \in \mathbb{N}}$

1) Enumerate $\{0,1\}$-strings:

$\{0,1\}^* \longrightarrow \mathbb{N}$ (1-1)

0	ε_0
1	0_1 1_2
2	00_3 01_4 10_5 11
3	000 001 ...

2) Enumerate TMs:

$Q = \{q_1, q_2, \dots, q_r\}$

- q_1: start
- q_2: accept

$X_1, \dots, X_s$

- $X_1 = 0$ $X_2 = 1$ $X_3 = B$

- $D_1 = L$
- $D_2 = R$

① $\delta(q_i, X_j) := (q_k, X_\ell, D_m)$ transitions

ξ

(i, j, k, ℓ, m)

$0^i 1 0^j 1 0^k 1 0^\ell 1 0^m$

② M: $C_1 C_2 \cdots C_n$

ξ

$M\, C_1 11 C_2 11 C_3 11 \cdots 11 C_n$

$\Rightarrow$ code $w \in \{0,1\}^*$ for each M $\Rightarrow$ every TM has idx.

$M = M_i$, for some index $i \in \mathbb{N}$.

For now: Have some encoding of TMs:

TM $M \rightsquigarrow$ idx $M = M_i$

Get a TM $\leftsquigarrow$ code $w \in \{0,1\}^*$

corresp. to code

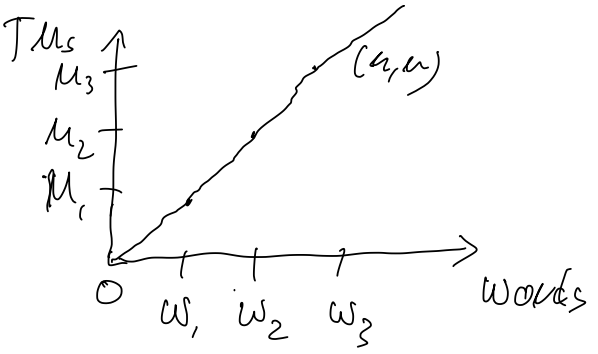
(if encoding invalid, ret. TM w./o. encodings)

Next: Apply to undecidability:

$\hookrightarrow$ Diagonalization!

Diagonal Lang.:

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



Diagonalization

Claim: L_D is not r.e.

Proof: Idea: Pf. by contradiction, assume there did exist TM M s.t. $L_D = L(M)$.

$\Rightarrow$ look at index $M = M_i$, $i \in \mathbb{N}$.

Q: Is $w_i \in M = M_i$?

(1) $w_i \in L_D \Rightarrow M_i$ accepts $w_i \Rightarrow w_i \notin L_D$ ↯

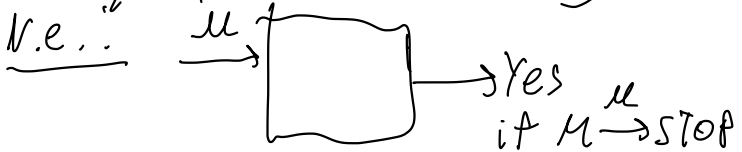
(2) $w_i \notin L_D \Rightarrow M_i$ doesn't acc $w_i \Rightarrow w_i \in L_D$ ↯

Contradi $\Rightarrow$ such an M can't exist! ▨

Halting Problem:

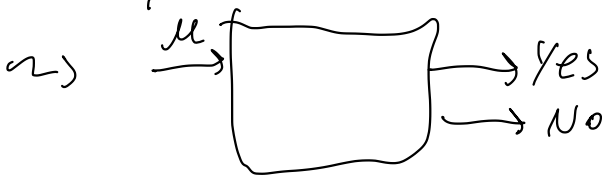
$L := H := \{ M \mid M \text{ halts on } M \} =: H$

i.e.:

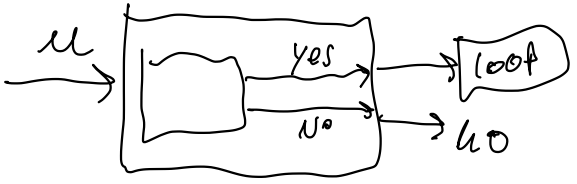


not dec.: ass. M dec. L_1.

M_1:



$\Rightarrow M_1'$:



M_1' halts on M_1'

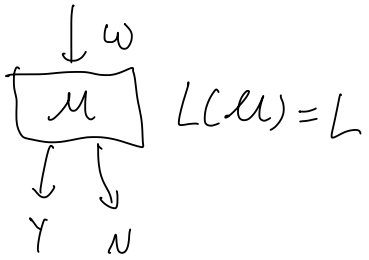
$\Rightarrow M_1' \xrightarrow{M_1'}$ No

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

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

04/15 Last time: (un)decidability

- When is a problem decidable?



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

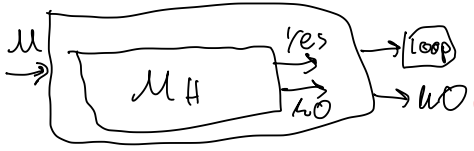
⇝ showed that:

(1) ex. M_H s.t. $L(M_H) = H$.

Idea: on M, M_H simulates M as input for M

⇝ H is recursively enumerable.

M'_H:



(2) Is H decidable?

PARADOX: self-referentiality

Cf. Liar's Paradox, barber's paradox

Complexity theory

Before: can we solve (decide) a prob.?

How fast can we solve a prob.?

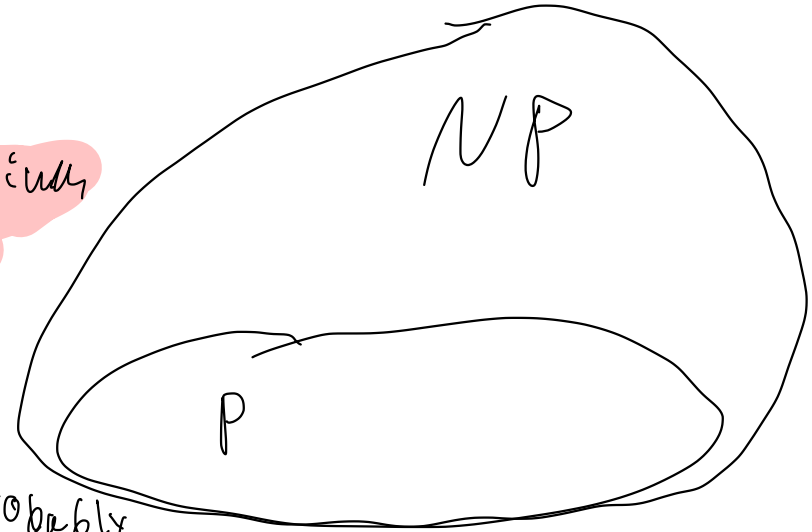
Recall: DTMs and NTMs are equally powerful

$P \subseteq NP$

$P \stackrel{?}{=} NP$

P vs. NP

- Millennium prob!
- Common belief: most probably false (but no proof!)



nondet. poly time:
Solvable by an NTM in poly. time

Polynomial time:
Solvable by a DTM in polyn. time

Time complexity / runtime:

A TM M has time complexity $T = T(n)$ (n: length of input) if for every input w, $|w| = n$, the TM M stops.

Notation: $f(n) \in O(g(n))$ iff "f grows at most as fast as g" (as $n \to \infty$)

(Big O / Landau notation)

Examples:

$T_1(n) = 2n + 100000 \in O(n)$ linear

$T_2(n) = 3n^2 + 3n + 1000 \in O(n^2)$ quadratic

$T_3(n) = 5n^{42} + 2n^{20} + n^{10} + 10^6$ polynomial

$T_4(n) = 2^n + n^3 + 3n^2 \in O(2^n)$ exponential

$T_5(n) = (3n)! + n^2$ factorial

$f \in O(g)$

$f(n) = O(g(n))$

$f \leq O(g(n))$

$f(n) \in O(g(n))$

Class P: solvable by a DTM M s.t. $T_M(n) = P(n)$, for some polynomial $P(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_2 n^2 + a_1 n + a_0 = \sum_{i=0}^{k} a_i n^i$

Ex.:

$P_0(n) = 100n^5 + 2n^2 + 7000$

$P_1(n) = n^3 + n^2$

$P_2(n) = n$

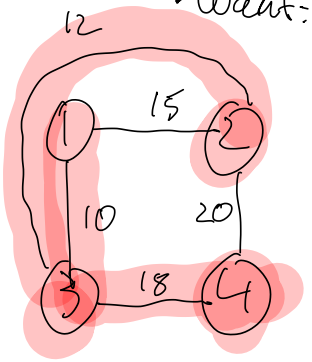
$P_3(n) = 40$

Class P: "Quick to solve!"	Class NP: "Quick to check!"
• Sorting • Rubik's cube • Multiplication	• Traveling salesman (TSP) • Cryptography (Public key) ↝ factorizability of integers • Sudoku • Protein folding

An ex. for P:

Kruskal's algorithm

Problem: • Have: wt-ed graph
• Want: Min. Weighted Spanning tree (MWST)



Q: How long does it take?

1. (1,3) mark
2. (2,3) mark
3. discard (1,2)
4. (3,4) mark
5. stop: marked 3 = 4-1 edges

Alg (Kruskal's alg.):

(1) look at all conn. components:
- for each, preserve conn. com.
- initially: all vertices are isolated, i.e., their own conn. comp.

(2) Consider a lowest-wted (unmkd)
If conn. comp. are diff:
- mark edge
- merge comp.
If conn. same comp.:
- disregard

(3) repeat, until: all edges seen, or # mked edges = # vert. - 1

Time complexity:

On computer:

In: $G_7 = (V, E)$, $|V| = n$, $|E| = e$

For each comm. comp. $O(e)$
- have to pick lowest-wt. edge $O(e)$
- find comm. comp. by lookup on vertices $O(n)$ (maybe merge: $O(n)$)

↳ $T(n) = O(e(e+n))$

On DTM:

Claiming: on a multi-tape TM, can do it in $O(n^2)$ steps ($O(n^4)$ for a single-tp. TM).

Encoding?

- If Y/n question is hard, then finding answer is (at least) as hard.
- Input overhead won't change complexity.

Implementation for mTM:

- several tapes for:
  - store nodes & comp.
  - cache current min.-edge
  - store unmarked edges
  - cache nodes i, j for Pot.-merge
- search for comb. comp's

- all of these take at most $O(n)$ steps
- at most $O(n)$ iterations.

$\rightsquigarrow T_{\text{multi}}(n) = O(n^2)$ steps

$T_{\text{sing}}(n) = O(n^4)$

04/22: Recall:

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

functions modeling runtime

$f \in O(g)$ if and only if:

There are $M, N > 0$ s.t. $f(n) \leq M \cdot g(n)$, for all $n > N$. (*)

"f is at most g, up to:
- a constant
- an initial number of steps"

The notation $O(g)$ denotes a set of functions (satisfying (*))

For example:

$O(n^3) = \{c, n, 2n+1, n^2, 2n^2+1000, n^3+2n^2+n, 1000n^3, \ldots\}$

Ex. 2: (1) reflexivity:

For all f: $f \in \mathcal{O}(f)$.

Proof: $g := f$

Want: $M, N > 0$ s.t.

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

f.a. $n > N$.

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

$N := 1$.

$f \in \mathcal{O}(g)$ if and only if:

There are $M, N > 0$ s.t.

$f(n) \leq M \cdot g(n)$, for all $n > N$.

Ex. 2.2:

f.a. $c > 0$:

$\mathcal{O}(cf) = \mathcal{O}(f)$

Recall:

$(A = B) \Leftrightarrow (A \subseteq B \;\&\; B \subseteq A)$

Ex.:

$\mathcal{O}(cf) \subseteq \mathcal{O}(f)$:

Want: If $g \in \mathcal{O}(cf)$, then $g \in \mathcal{O}(f)$.

If $g \in \mathcal{O}(cf) \Rightarrow$ ex. M, N: $g(n) \leq M \cdot cf(n)$,

f.a. $n > N$.

Want: M', N': $g(n) \leq M' f(n)$,

f.a. $n > N'$.

$M' := Mc$, $N' := N$.

!! ✓

$g \in O(f) \Rightarrow g \in O(cf)$:

$g \in O(f) \Rightarrow$ ex. $M, N > 0$ s.t.

$g(n) \le M \cdot f(n)$.

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

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

f.a. $N' \ge n$.

- $c = 1$: want $g(n) \le M' f(n)$

  Take $M' := M$, $N' := N$

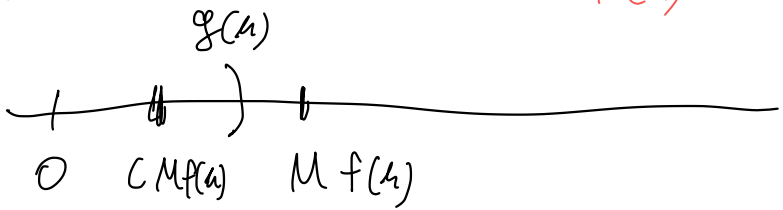
- $c > 1$: Want $g(n) \le M' c f(n)$

  Have: $g(n) \le M \cdot f(n)$

  $(c \ge 1)$

  $\le \underbrace{M \cdot c}_{=: M'} \cdot f(n)$, $N' := N$ ✓

- $c < 1$: Want $g(n) \le M' c f(n)$



Need: M' s.t. $g(n) \le M' c f(n)$

$g(n) \le M f(n) \le M' c f(n)$ ✓

$\rightsquigarrow M \le M' c \Leftarrow M' \ge \frac{M}{c}$, $M' := \lceil M/c \rceil$ (ceiling), $N' := N$

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

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

Prove:

$$\mathcal{O}(f) \subseteq \mathcal{O}(g) \Rightarrow \mathcal{O}(f+h) \subseteq \mathcal{O}(g+h)$$

$f_0 \in \mathcal{O}(f) \rightsquigarrow f_0 \in \mathcal{O}(g)$

Example:

(1) $\mathcal{O}(n) \subseteq \mathcal{O}(n^2)$, $h := \log(n)$

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

(2) $h := n^2$

$\rightsquigarrow \mathcal{O}(n+n^2) = \mathcal{O}(n^2) \subseteq \mathcal{O}(2n^2) = \mathcal{O}(n^2)$

In general:

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

Proof: $f_0 \in \mathcal{O}(f+h)$

$\Rightarrow$ ex. M, N: $f_0(n) \leq M(f(n)+h(n))$

$= Mf(n) + Mh(n)$

Wahl: M', N': $f_0(n) \leq M' g(n) + M' h(n)$

$f \in \mathcal{O}(f) \Rightarrow f \in \mathcal{O}(g)$

$\rightsquigarrow f(n) \leq M' g(n)$ f.s. M', N'

$\rightsquigarrow f_0(n) \leq Mf(n) + Mh(n)$

$\leq$

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

$\Rightarrow$ ex. M, N: $f_0(n) \leq M(f(n)+h(n))$

$= Mf(n) + Mh(n)$

Wahl: M', N': $f_0(n) \leq M'g(n) + M'h(n)$

$f \in O(f) \Rightarrow f \in O(g)$

$\rightsquigarrow f(n) \leq M' g(n)$ f.s. M', N'

$\rightsquigarrow f_0(n) \leq Mf(n) + Mh(n)$

$\leq M'Mg(n) + Mh(n)$

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

$n > N''$, with M'', N''

bis enough. ▨

04/24 CONSTRAINT-SATISFACTION PROBLEMS (CSP)

circuit design, databases, ...

Recall:

$P \subseteq NP$

$NP \setminus P = ??$

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

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

Heuristically/empirically:

P : tractable | NP \ P : intractable ← tuned approx. alg.

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

↝ poly-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. ($O(a_k n^k + a_{k-1} n^{k-1} + \dots + a_0)$)

Notation: $L_1 \leq_p L_2$

NP-complete problems:

(1) $L \in NP$

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

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_2.

Have: (i) $L \leq_p P_1$, say in $O(p(n))$ ($|w| = n$. Input length)

$w \in L$, $|w| = n$ $\rightsquigarrow$ $w' = x \in P_1$, $|x| \leq p(n)$.

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

ξ

$w \in L$ $\rightsquigarrow$ $x \in L_1$ $\rightsquigarrow$ $y \in L_2$

$O(p(n))$ $O(q(p(n)))$

$|x| \leq p(n)$ $|y| \leq q(n)$

$n = p(n)$

ξ

$|y| \leq q(p(n))$

Total runtime:

$O(p(n) + q(p(n)))$

poly poly

poly

The satisfiability problem SAT:

Q: Is a given boolean expr. satisfiable?

A boolean expr. is gen. by:

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

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

(iii) unary op: NEGATION $\neg$ (3)

(iv) parentheses (,)

$\neg 0 = 1$, $\neg 1 = 0$

Truth value assignments:
$T: \text{BoolExpr} \to \{0,1\}$

Ex.:

$\varphi := x \wedge \neg(y \vee z)$

$T(x) = 1$
$T(y) = 0$
$T(z) = 0$
$\Big\} \; T(\varphi) = 1 \wedge \neg 0 = 1 \wedge 1 = 1$ Sat! (11)

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

$T(x) = 1$
$T(y) = 0$

↯ not Sat.

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

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 (i) multi-tape NTM guess t.v. assignments and eval.

$\rightsquigarrow O(n^2)$ time

[$\rightsquigarrow O(n^4)$ on sNTM] $\rightsquigarrow SAT \in NP$

Ad (2): univ. reduction:

- f.a. $L \in NP$: $L \leq SAT$
- red. can be made polyn.

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

May assume: M never write blank, never go left of q_0.

Analyze M:

If $|w| = n$, and M acc. w, then:

(i) α_0 initial ID w./ input w

(ii) $\alpha_0 \vdash \alpha_1 \vdash \dots \vdash \alpha_k$, $k \leq P(n)$

(iii) α_k accepting

(iv) each α_i doesn't contain blanks (except edge cases), and goes left to right

Strategy:

(a) $\alpha_i := X_{i0} X_{i1} \cdots X_{ip(w)}$,
w/ one symb.,
others are tape

(b) Create boolean var's:
Y_{ijA} for "$X_{ij} = A$"
for A a state or
tape symbol
$(i, j \in \{0, \ldots, p(|w|)\})$

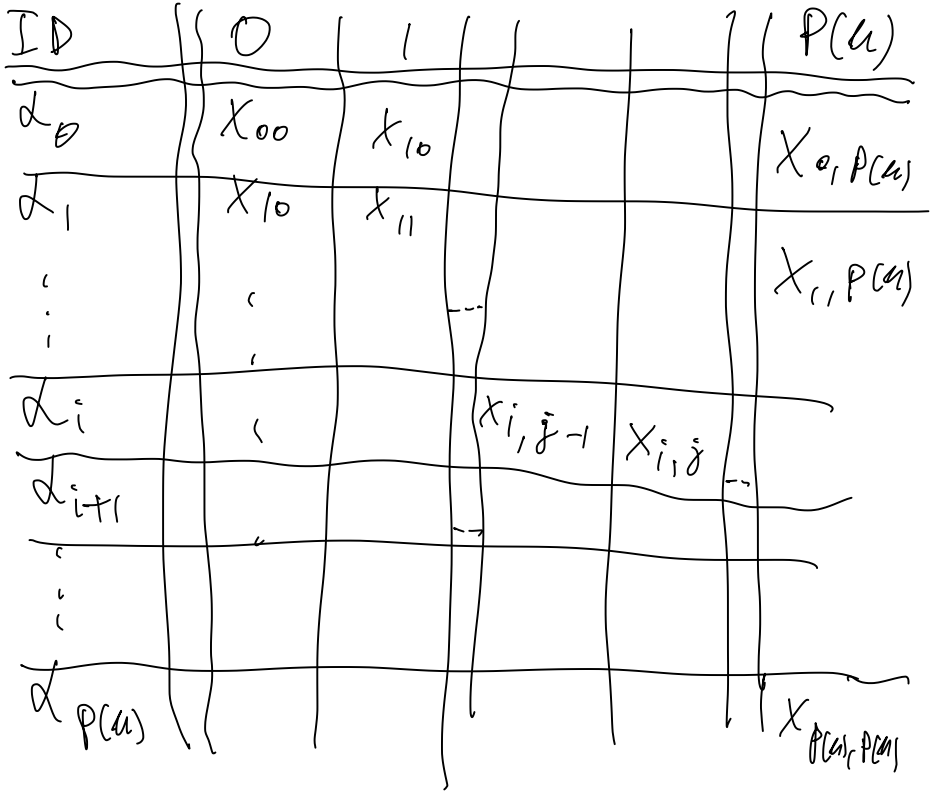
(c) Express that α_i accepts w, sat. only in P-time, in at most $p(|w|)$ moves.

⇝ Express validity of computations

$(M, w) \longmapsto E_{M,w} := \mathrm{US} \wedge \mathrm{N} \wedge \mathrm{F}$

- U: "unique symb. in each cell"
- S: "starting right"
- N: "move right"
- F: "finishing right"

$(P(u)+1)^2$ – array:



Want to model valid moves!

Want:

$(u,w) \mapsto E_{u,w} = u\,\mathrm{S}\,\mathrm{S}\,\mathrm{M}\,\mathrm{M}\,\mathrm{F}$

- u: unique symbol
- S: stuff right
- M: move right
- F: finish right

Uniqueness u:

Y_{ijA}: "$X_{ij} = A$"

$$\bigwedge_{i,j,\, A \neq B} \neg(Y_{ijA} \wedge Y_{ijB})$$

big conj. !!

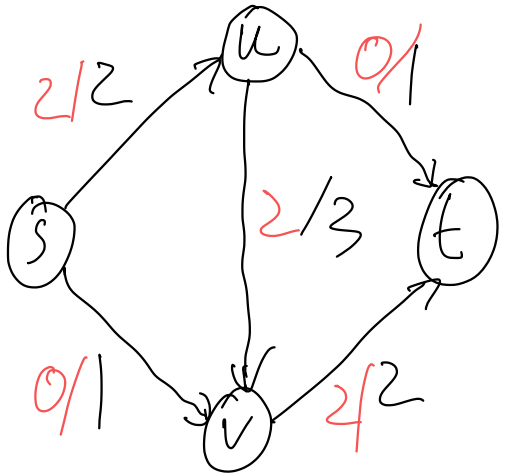
$$u := \bigwedge_{i,j,\, A \neq B} \neg(Y_{ijA} \wedge Y_{ijB})$$

$\rightsquigarrow$ length $\mathcal{O}(P(u)^2)$

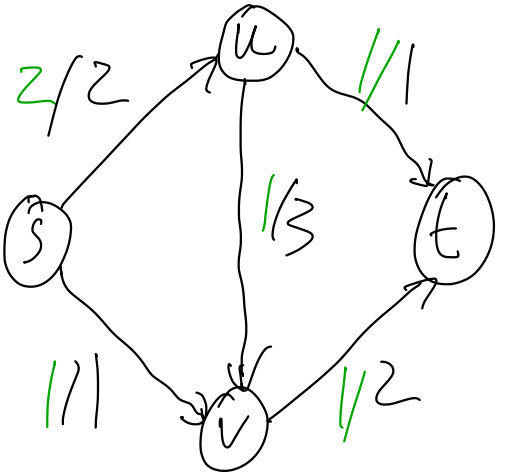
TODO: def.
rest of
encoding

⇝ application.

Part III: Graph theory — Flow networks



#packages = 2

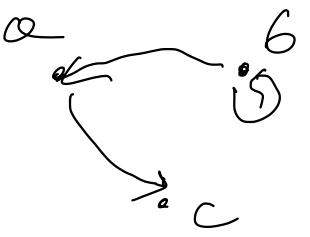


#packages = 3

Problem of finding a maximal flow

Def. Graph: $G = (V, E)$, where:

- Vertices $u, v, w, \ldots \in V$
- edges $(u,v), (v,v), (w,u), \ldots \in E \subseteq V \times V$



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

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

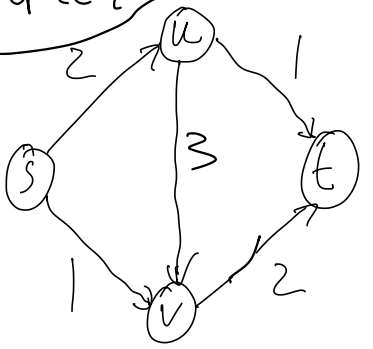
NB:
- no distinguished states (no start, no end, ...)
- no labels
- no connectivity requirements
- no multi-edges (so no $\bullet \rightrightarrows \bullet$)
- do have direction

Def Flow networks:

$G = (V, E, s, t, c)$:

- $s, t \in V$ source/target vertices ($s \neq t$)
- Such that $(v, v) \notin E$ (no loops) and:
  $(v, w) \in E \Rightarrow (w, v) \notin E$ (not symmetric)
- c capacity function: $c(u, v) \geq 0$,
  f.a. $(u, v) \in V \times V$ (with $c(u, v) := 0$ if $(u, v) \notin E$)

Example:



Def. Flow: A *flow* in a flow netw. G consists of data $f(u,v) \geq 0$ f.a. $(u,v) \in V \times V$ s.t.

① **Capacity constraint:**

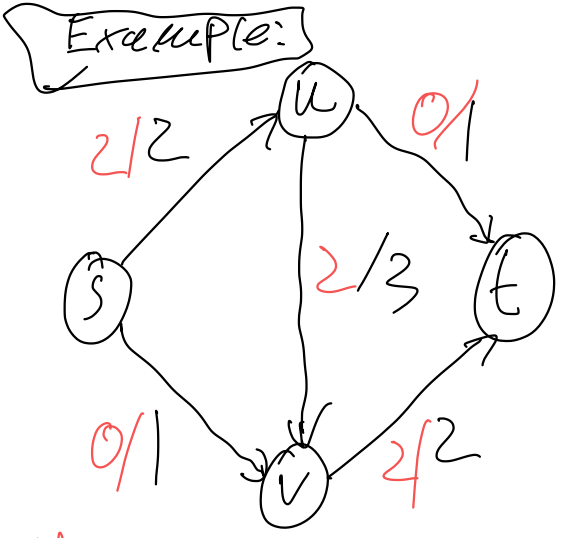
f.a. $(u,v) \in E$: $f(u,v) \leq c(u,v)$

② **Flow conservativity:**

f.a. $v \in V$ s.t. $v \neq s,t$:

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

Example:



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

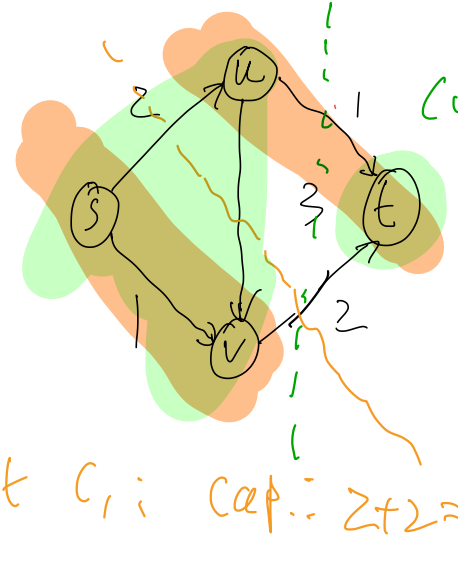
Def. Flow value:

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

$\rightsquigarrow$ find max. flow!

Will see:

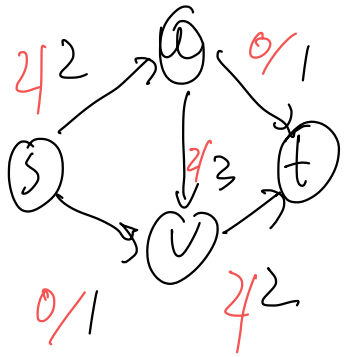
- algorithm to find a max. flow (Ford-Fulkerson alg.)
- <u>duality principle:</u> max. flows ⟷ min. cuts (dual optimization problems)



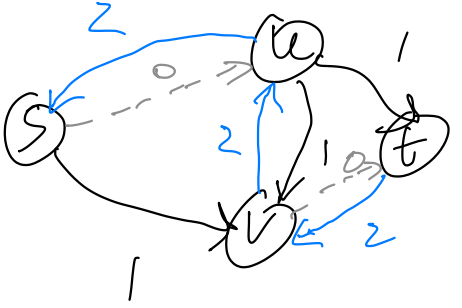
Applications:
profitable proj. selection
matrix rounding
airline scheduling
image segm.
baseball elimination

Ford-Fulkerson to find a max. flow:

(G, f):



Residual graph:



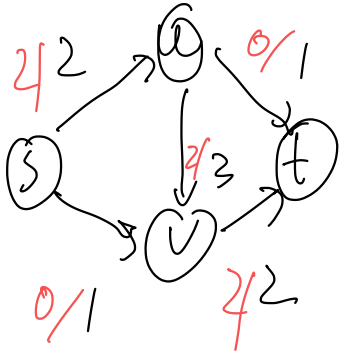
1) add backward edges for flows > 0

2) add forward edges for residual capacities > 0: $c_{res}(u,v) = c(u,v) - f(u,v)$

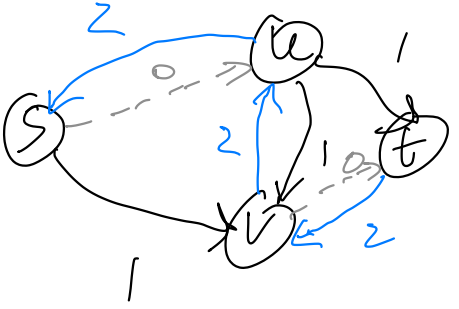
① start from (G, f), create residual graph G_f

② In G_f, choose a path $s \rightsquigarrow t$, use this to augment the flow $f \rightsquigarrow f'$, apply ① to (G, f').

FF: step 1:

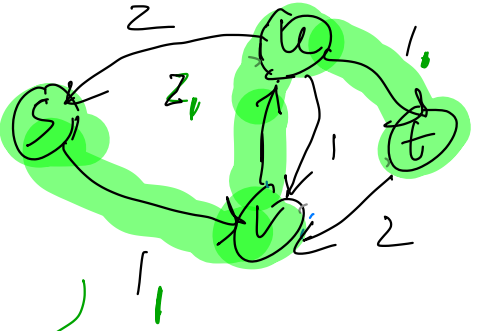


(G, f)

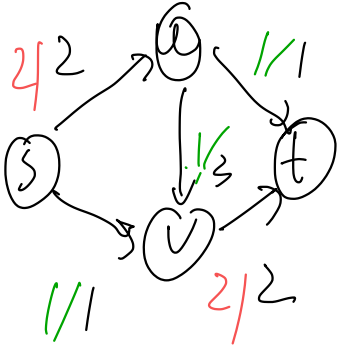


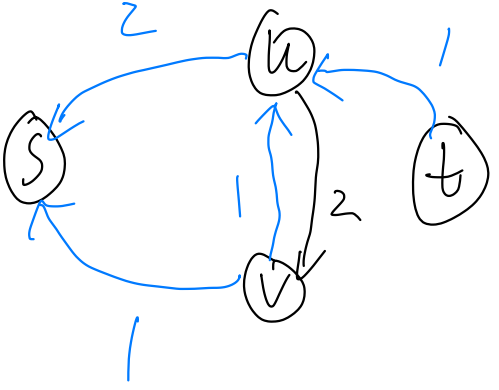
G_f

Find s-t paths:



augmentation

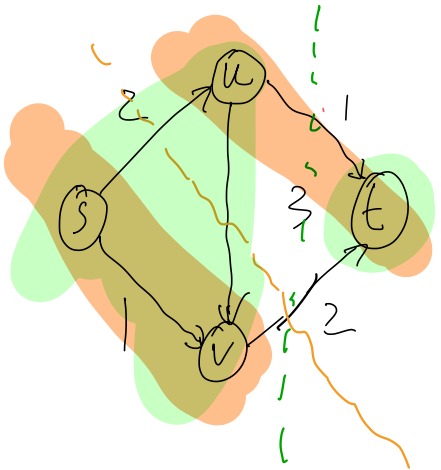




No more s-t paths

⇝ DONE!

Cuts: In flow network is a partition $C = (S, T)$ of $V = S \uplus T$, where $s \in S$, $t \in T$.



Capacity of C:

$$c(S,T) := \sum_{u \in S} \sum_{v \in T} c(u,v)$$

What is the connection?

↝ net flow

Thm. Let G be a flow netw., f be a flow, and $C = (S,T)$ any cut.

Then the flow value $|f|$ is equal to the net flow of f through (S,T):

$$|f| = \sum_{(u,v) \in S \times T} f(u,v) - \sum_{(u,v) \in S \times T} f(v,u)$$

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

Thm. $|f| \leq c(S,T)$

Proof. Follows from net flow thm.

Proof.

$$|f| \overset{\text{def}}{=} \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) + \sum_{u \in S \setminus \{s\}} \underbrace{\sum_{v \in V} f(u,v) - f(v,u)}_{=0 \text{ by conservation}}$$

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

$$= \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u). \quad \blacksquare$$

$V = S \dot\cup T$

Main Thm: The following are equivalent:

(1) exists a cut (S,T) such that $|f| = c(S,T)$

(2) $|f|$ max.

(3) G_f has no augm. path.

Idea: construct the min. cut ($\sim$ max. flow) by taking $(S, V \setminus S)$, where

$S = \{$ all nodes of G_f that s can reach $\}$

$\rightsquigarrow |f| = c(S, V \setminus S)$. (net flow formula)