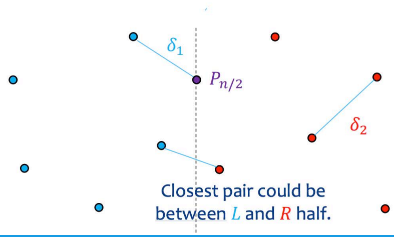
Divide and Conquer?

ClosestPair$(P_1, \ldots, P_n)$: // $n \geq 2$ pts in the plane, x-sorted asc.
if $n = 2$ then return dist(P_1, P_2) // base case
$(L, R) \leftarrow$ partition points by $P_{n/2}$ // split by median
$\delta_1 \leftarrow$ **ClosestPair**(L) // min dist on left
$\delta_2 \leftarrow$ **ClosestPair**(R) // min dist on right
need to know min dist between L and R // … how?



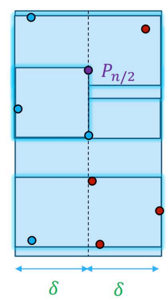
Q: How many blue/red pairs are there?

Q: Do we need to check all of them?

Properties of the δ-strip

ClosestPair$(P_1, \ldots, P_n)$: // $n \geq 2$ pts in the plane, x-sorted asc.
if $n = 2$ then return dist(P_1, P_2) // base case
$(L, R) \leftarrow$ partition points by $P_{n/2}$ // split by median
$\delta_1 \leftarrow$ **ClosestPair**(L) // min dist on left
$\delta_2 \leftarrow$ **ClosestPair**(R) // min dist on right
need to know min dist between L and R // … look at δ-strip

- Let $\delta = \min\{\delta_1, \delta_2\}$.
- **Q:** How many pts can there be in the δ-strip?
- **Q:** How many blue pts can there be in a $\delta \times \delta$ square?
- **Q:** How many pts can there be in a $\delta \times 2\delta$ rectangle?



How to find a close red/blue pair:
Slide a $\delta \times 2\delta$ rectangle down!

$$T(n) = kT(n/b) + O(n^d)$$
$$= \begin{cases} O(n^d) & \text{if } k < b^d \\ O(n^d \log n) & \text{if } k = b^d \\ O(n^{\log_b k}) & \text{if } k > b^d \end{cases}$$

Karatsuba's Algorithm

Karatsuba(x, y): // x, y are n-digit positive integers
if $n = 1$ then return $x \cdot y$ // base case; hard-code
$(a, b) \leftarrow$ split digits of x into halves // $x = a \cdot 10^{n/2} + b$
$(c, d) \leftarrow$ split digits of y into halves // $y = c \cdot 10^{n/2} + d$
$t_1 \leftarrow$ **Karatsuba**(a, c) // $= ac$
$t_4 \leftarrow$ **Karatsuba**$(a + b, c + d)$ // $= (a + b)(c + d)$
$t_3 \leftarrow$ **Karatsuba**(b, d) // $= bd$
$t_2 \leftarrow t_4 - t_1 - t_3$ // $= ad + bc$
return $(t_1 \ll n) + (t_2 \ll n/2) + t_3$

Next: The runtime of **Karatsuba** is $O(n^{1.585})$.

- An ***alphabet*** is a finite set of characters, usually denoted Σ
  - Typically implicit, e.g., ASCII characters or binary $\{0,1\}$
- A (Σ-)***string*** is a finite sequence of characters from Σ
  - The ***length*** of a string x (# chars) is denoted $|x|$
  - The ***empty string*** is denoted ε; it has length 0
- A (Σ-)***language*** is (possibly infinite) set of (Σ-)strings: $L \subseteq \Sigma^*$
  - The language of all strings is denoted Σ^*
- **Example:** $\Sigma = \{0,1\}, \Sigma^* = \{\varepsilon, 0, 1, 00, \ldots\}, |010| = 3,\ 0^3 1^2 = 00011$

Recurrence for LCS

$$LCS(i,j) = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ 1 + LCS(i-1, j-1) & X[i] = Y[j] \\ ? & X[i] \neq Y[j] \end{cases}$$

- **Case 2:** $X[i] \neq Y[j]$ (end with different characters)
  - **Example:** $X[1..i] =$ "GTCA" and $Y[1..j] =$ "GTC"
  - At least one of the letters is not part of LCS
  - **Q:** How do we know which one?
    - Try both! $LCS(i,j) = \max\{LCS(i-1, j), LCS(i, j-1)\}$

Recurrence for LIS_{at}

$$LIS_{at}(i) = \begin{cases} 0 & i = 0 \\ 1 + \max\left\{ LIS_{at}(j) \;\middle|\; \begin{matrix} (A[j] < A[i] \text{ and } j < i) \\ \text{or } j = 0 \end{matrix} \right\} & i \neq 0 \end{cases}$$

LIS$(A[1..n])$: // table implementation of LCS
allocate $L[0..n]$
$L[0] \leftarrow 0$
for $i = 1..n$: // fill table
  $l \leftarrow 0$
  for $j = 1..i-1$:
    if $A[j] < A[i]$: $l \leftarrow \max\{l, L[j]\}$
  $L[i] \leftarrow l + 1$
return ?

- The conversion from recurrence to table is *mechanical*
- **Q:** Given this recurrence, how do we determine the length of a LIS?

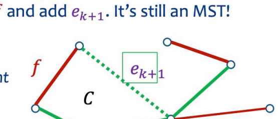
Kruskal(G): // G is weighted, undirected graph
$T \leftarrow \emptyset$ // invariant: T is a spanning *forest* (set of trees) of G
for each edge e **in** *increasing order of weight*:
  if $T + e$ is acyclic: $T \leftarrow T + e$
return T

Correctness

Kruskal(G): // G is weighted, undirected graph
$T \leftarrow \emptyset$ // invariant: T is a spanning *forest* (set of trees) of G
for each edge e in *increasing order of weight*:
  if $T + e$ is acyclic: $T \leftarrow T + e$
return T

- Let $e_1, e_2, \ldots$ be the edges of T, *in order of addition to* T
- **Idea:** Show by induction, every time we "swap" an edge, still have an MST
- **Base case:** $k = 0$ swaps; still have T', an MST
- **Ind. step:** Suppose we've swapped in first k edges and it's still an MST
  - Consider the next edge e_{k+1} added to T.
  - If $e_{k+1} \in T'$, MST doesn't change
  - If $e_{k+1} \notin T'$, then adding it creates a cycle C (adding any edge to MST makes a cycle)
  - Since T is acyclic, there is an edge $f \in T'$ on the cycle C.
  - "Swap in e_{k+1}": Remove f and add e_{k+1}. It's still an MST!

Claim: e_{k+1}'s weight $\leq f$'s weight.
- edges added *in increasing order of weight*
- f + first k edges do not form a cycle
  - Kruskal would have considered adding f, but added e_{k+1} instead

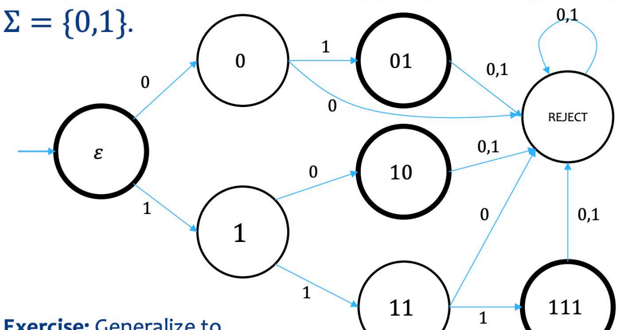


- Formally, a DFA M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:
  - Q is the finite set of ***states***
  - Σ is the ***input alphabet***
  - $\delta : Q \times \Sigma \rightarrow Q$ is the ***transition function***
  - $q_0 \in Q$ is the ***initial*** state
  - $F \subseteq Q$ is the subset of ***accepting*** states
- **Takeaway:** DFAs are a simple & weak, but well defined, kind of "computer."

Regular Expression Exercises

* All strings over $\{a, b\}$ with an ***even*** number of as.
  * $b^*(b^*ab^*ab^*)^*$
* All strings over $\{a, b\}$ without 2 consecutive as.
  * $(b^*ab)^*(b^*(a|\epsilon))$
* All strings over $\{0,1\}$ that begin and end with the same symbol.
  * $(0(0|1)^*0)|(1(0|1)^*1)$
* $N = (0|1|2|\cdots|9)$ $\quad$ $L = (A|B|\cdots|Z)$
  * Dates: $NN - LLL - NN(NN|\epsilon)$ (E.g., 16-Feb-2023 or 16-Feb-23)
  * Michigan License Plates: $LLL\ NNNN$

Example: DFA that decides $L = \{\varepsilon, 01, 10, 111\}$, for alphabet $\Sigma = \{0,1\}$.



A
inesc
"rejec
is ofte
use

Exercise: Generalize to any *finite* language.

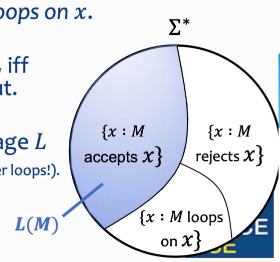
* A ***Turing Machine*** is a 7-tuple $(Q, \Gamma, \Sigma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$:
  * Q is a finite set of ***states***
  * $q_0 \in Q$ is the ***initial state***
  * $F = \{q_{\text{accept}}, q_{\text{reject}}\} \subseteq Q$ are the ***final*** (***accept/reject***) states
  * Σ is the ***input alphabet***
  * $\Gamma \supseteq \Sigma \cup \{\perp\}$ is the ***tape alphabet*** ($\perp \notin \Sigma$ is the ***blank symbol***)
  * $\delta : (Q \setminus F)\times\Gamma \to Q\times\Gamma\times\{L, R\}$ is the ***transition function***
* **Takeaway:** TMs are a well-defined type of "computer".

* **Definition:** A TM ***decides*** language if :
  1. *accepts* every string (" accepts"), and
  2. *rejects* every string (" rejects").

We say that is a ***decider*** (for), and is ***decidable***.

* **Note:** By definition, *does not loop* on any input!

* **Definition:** The language of a TM M is $L(M) = \{x \mid M \text{ accepts } x\}$.
* **Question:** What if $x \notin L(M)$? (does not accept.)
* **Answer:** Then M either *rejects* x, or *loops* on x.
* **Conclusion:** TM M decides language L iff $L = L(M)$ and M halts on *every* input.
* Definition: TM M recognizes language L if $L = L(M)$ (regardless of whether ever loops!).
* More on this later…



$$L^*[j] = 1 - T[j, j]$$

	s_1	s_2	s_3	s_4	s_5	s_6	…
$L(M_1)$	1	0	0	1	1	0	…
$L(M_2)$	0	1	1	0	0	0	…
$L(M_3)$	1	1	1	1	1	1	…
$L(M_4)$	0	0	0	0	0	0	…
$L(M_5)$	1	0	1	0	0	0	…
…	…	…	…	…	…	…	…

* **Proof.** If L … for some i … $L^* \neq L(M_i$

L^*	0	0	

$$L_{\text{BARBER}} = \{\langle M\rangle : M \text{ does not accept } \langle M\rangle\}$$

* **Result:** "*Program B accepts the source code of those, and only those, who do not accept their own source code.*"

* **Question:** Does B accept its own source code?
* **Answer:** Suppose P is a program.
  1. P accepts its own code $\Longrightarrow B$ does not accept P's code.
  2. P does not accept its own code $\Longrightarrow B$ accepts P's code.

* **Question:** What if $P = B$?
  1. B accepts its own code $\Longrightarrow B$ does not accept B's code.
  2. B does not accept its own code $\Longrightarrow B$ accepts B's code.

Paradox! (Program B cannot exist)

L_{HALT} is Undecidable

We need to implement:
C is given two input: $\langle M\rangle$ and x
M accepts $x \Longrightarrow C$ accepts$(\langle M\rangle, x)$
M does not accept $\langle M\rangle \Longrightarrow C$ rejects$(\langle M\rangle, x)$

We have:
H is given two inputs: $\langle M\rangle$ and x
M accepts or rejects $x \Longrightarrow H$ accepts $(\langle M\rangle, x)$
M loops on $x \Longrightarrow H$ rejects $(\langle M\rangle, x)$

* **In code:**
* C on input$(\langle M\rangle, x)$:
  * Run H on $(\langle M\rangle, x)$ (*Ask H: "does M halt on x"?*)
  * If H rejects, *reject* (*M loops on x, so it can't accept it*)
  * If H accepts, run M on x (*M halts on x, so this is safe to do*)
    * If M accepts, *accept*; If M rejects, *reject* (*answer like M*)

* **Analysis:** C always halts (why?). Moreover:
  * M accepts $x \Longrightarrow H$ accepts $(\langle M\rangle, x) \Longrightarrow C$ accepts $(\langle M\rangle, x)$
  * M rejects $x \Longrightarrow H$ accepts $(\langle M\rangle, x) \Longrightarrow C$ rejects $(\langle M\rangle, x)$
  * M loops on $x \Longrightarrow H$ rejects $(\langle M\rangle, x) \Longrightarrow C$ rejects $(\langle M\rangle, x)$

* **Conclusion:** L_{HALT} decidable $\Longrightarrow L_{\text{ACC}}$ decidable
* **Contrapositive:** L_{ACC} undecidable $\Longrightarrow L_{\text{HALT}}$ undecidable

* **Definition:** Language A is ***Turing reducible*** to language B, written $A \leq_T B$, if there exists a program M that decides A using a "*black box*" that decides B.

* **Previous results:** $L_{BARBER} \leq_T L_{\text{ACC}}$ and $L_{\text{ACC}} \leq_T L_{\text{HALT}}$
* **Intuition:** B is "*no easier*" than A to decide.
* **Theorem:** Suppose $A \leq_T B$. Then B is decidable $\Longrightarrow A$ is decidable.
* **Definition:** A language A is ***recognizable*** if there exists a program M (a "recognizer") that recognizes it: L(M)=A.
* **Theorem:** If a language A and its complement $\overline{A}$ are both *recognizable*, then A is *decidable*.

$L_{\emptyset}$ is Unrecognizable

* **Claim:** $L_{\emptyset} = \{\langle M\rangle : L(M) = \emptyset\}$ is unrecognizable.
* **Proof:** We show that $L_{\emptyset}$ is undecidable ($L_{\text{ACC}} \leq_T L_{\emptyset}$) and $\overline{L_{\emptyset}}$ is recognizable.
* **Step 1:** Let N be a decider for $L_{\emptyset}$. Construct a decider C for L_{ACC}:
* $C(\langle M, x\rangle)$:
  1. Construct a program "$M'(w)$: run M on x and answer as M does"
  2. Call N on $\langle M'\rangle$ and return the *opposite* output
* **Analysis:** C halts since N does. Moreover:
  * M accepts $x \Leftrightarrow L(M') \neq \emptyset \Leftrightarrow N$ rejects $\langle M'\rangle \Leftrightarrow C$ accepts $(\langle M\rangle, x)$

"Dovetailing"

* **Claim:** $L_{\emptyset} = \{\langle M\rangle : L(M) = \emptyset\}$ is unrecognizable.
* **Proof:** We show that $L_{\emptyset}$ is undecidable ($L_{\text{ACC}} \leq_T L_{\emptyset}$) and $\overline{L_{\emptyset}}$ is recognizable.
* **Step 2:** We need to construct a recognizer for $\overline{L_{\emptyset}} = \{\langle M\rangle : L(M) \neq \emptyset\}$.
* **Idea:** Do step i of $M(s_j)$ in "block" $i + j$ (like in proof that $\mathbb{Q}^+$ is countable).
* $R(\langle M\rangle)$:
  1. For $t = 1, 2, 3, \ldots$:
     2. For $j = 1, 2, \ldots, t$:
        3. Run one (additional) step of $M(s_j)$
        4. If $M(s_j)$ accepts, *accept*.
* **Analysis:**
  * $\langle M\rangle \in \overline{L_{\emptyset}} \Rightarrow \exists j, k.\ M$ accepts s_j in k steps $\Longrightarrow R$ accepts $\langle M\rangle$.
  * $\langle M\rangle \notin \overline{L_{\emptyset}} \Rightarrow L(M) = \emptyset \Longrightarrow R$ loops on $\langle M\rangle$.

Berry's Paradox

* **Aside (Berry's Paradox):** "The first positive integer that cannot be defined in <70 characters."
* Let $S \subset \mathbb{Z}^+$ be the set of positive integers that cannot be defined in <70 characters. Let $x = \min(S)$. Then:
  * x cannot be defined in <70 characters, since $x \in S$
  * x can be defined by the sentence "The first positive integer that cannot be defined in <70 characters.", which has 68 characters

Paradox!

$K_U(w)$ is Uncomputable

* **Proof:** Pick the language U. Suppose M is a program that computes $K_U(w)$.
* **Definition:** For every n define new program Q_n:
  * const int LENGTH = n; (*n is "hardcoded"*)
  * Iterate over all $x \in \{0,1\}^{\text{LENGTH}}$:
    * Compute $K_U(x)$ (*using M as a black-box*)
    * Output the first x such that $K_U(x) \geq$ LENGTH

n encoded in binary

* **Observation:** For every n the size of the program Q_n is $O(\log n)$.
* **Analysis:** Let w_n be the output of Q_n. What is $K_U(w_n)$?
* **By definition:** $K_U(w_n) \geq n$, since Q_n outputs an x such that $K_U(x) \geq n$.
  On the other hand, Q_n outputs w_n, so Q_n is a program that outputs w_n; by definition of Kolmogorov complexity, $K_U(w_n) \leq |Q_n|$.
* **Therefore:** $K_U(w_n) \geq n$ and $K_U(w_n) \leq O(\log n)$
* **Contradiction:** Conditions cannot be fulfilled simultaneously!
* **Conclusion:** No such M exists.

Note: this could hide a large constant depending on the size of the program M

* **Definition:** The ***Kolmogorov Complexity*** of w:

$$K_U(w) = \min\{|\langle M\rangle| : M \text{ outputs } w, \text{on empty input } \varepsilon\}.$$

That is, the size of the shortest U-program that outputs w.