Euclidean Algo

GCD(x,y):
if(y == 0) return x
if(y == 1) return 1
return GCD(y, x mod y)

Potential Method (Leaky Bucket)

To prove some algorithm terminates, construct some potential function S_t:
→ $S_t \geq$ [some lower bound]
→ $S_{t+1} < S_t$ (integers)
Then, use proof by contradiction w/ potential function to prove that algorithm must terminate

Karatsuba Multiplication

- $x = a \cdot 10^{n/2} + b$
- $y = c \cdot 10^{n/2} + d$
- $xy = ac(10^n) + (ad+bc)10^{n/2} + bd$
  $xy = ac(10^n) + [(a+b)(cd) - ac - bd]\,10^{n/2} + bd$
- 3 mults instead of 4 (!!)

Dynamic Programming

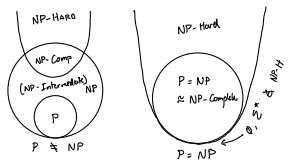
- optimal substructure
- overlapping subproblems
① Bottom-Up
  - no risk of overflowing stack
  - can optimize array-like structure
② Top-Down (memorization)
  - may save time (no unnecessary calls)
  - easier to logically implement

Recurrence Relations:

- Rod-Cut
given rod of length n, list P of prices where P[i] is price of length i, find max profit of cutting rod + selling pieces
$R(0) = 0$; $R(1) = P[1]$
$R(n) = \max_{1 \leq k \leq j} \{P(k) + R(j-k)\}$
- Edit-Distance:
→ Transform string A into B via substituting, deleting, inserting characters (cost s, d, i)
→ Find minimum-cost of transforming A to B
>> Let A = [1, ..., m] ; B = [1, ..., n]
>> If A[m] = B[n]:
return ED(A[1→m-1], B[1→n-1])
>> Else:
$\min\{c_i + ED(A[1\to m], B[1\to n-1]),$
$c_s + ED(A[1\to m-1], B[1\to n-1]),$
$c_d + ED(A[1\to m-1], B[1\to n])\}$

Complexity

- P: Class of languages with $O(n^k)$ decider
- NP: "Guess-and-check" languages — can verify efficiently, given some certificate
→ Formally: $\exists\, V(x,y)$ s.t. V is polynomial w.r.t x. If $x \notin L \Rightarrow \forall y,\ !V(x,y)$
If $x \notin L \Rightarrow \forall y,\ V(x,y)$
- $P \subseteq NP$, but $P = NP$??
- NP-Hard: $A \in$ NP-Hard iff $\forall L \in NP$, $L \leq_p A$
- NP-Complete: NP-Hard $\cap$ NP



- To prove $P \neq NP$, show $L \in NP$ s.t. no possible TM can decide L in poly-time
- To prove P=NP, show every language in NP has efficient decider (show for $L \in$ NP-Hard)

Cook-Levin Theorem

- SAT: Some boolean formula ϕ, is ϕ satisfiable?
- Prove $\forall L \in NP$, $L \leq_p SAT$
  - Given V_L, build efficient D_L using D_{SAT}
  - How? Construct $\phi_{V,x}$ that simulates verifier
    → inputs to $\phi_{V,x}$: simulate certificate y
    → if $\phi_{V,x}$ satisfiable → ∃ valid cert. y
  - Create a tableau encoding history of TM
    → each row contains tape + q_{TT} to indicate state/head
    → check ϕ_{start} (valid init. state), ϕ_{cell} (every cell contains 1 symbol), ϕ_{accept} (accept state somewhere), ϕ_{move} (2×3 windows valid)

Polytime Properties

- $L \in P$ iff $\bar{L} \in P$
- Let $coNP = \{\bar{L} \mid L \in NP\}$
→ If $NP \neq coNP$, then $P \neq NI$
  → $NP \neq coNP \rightarrow L \in NP,\ \bar{L} \notin NP$.
  → $\bar{L} \notin NP \rightarrow \bar{L} \notin P$
  → $\bar{L} \notin P \rightarrow L \notin P$ (above)
  → $L \notin P \wedge L \in NP \rightarrow P \neq NP$
- If P=NP, every non-trivial language is NP-Hard.
  → Let L be non-trivial; $\exists x \in L, \exists x' \notin L$
  → Show $\forall L' \in NP\ (L' \in P)$, $L' \leq_p L$
  → As $L' \in P$, $\exists f(w): L' \to L$ if $[D_L(w)]$ return x else return x'

Divide + Conquer

- Divide into smaller subproblems
- Combine answers in meaningful way
- Closest Pair in 2D, MergeSort
  ↓
  - Presort points by x and y coords (P_x, P_y)
  - Get mid-dist from first, second half of P_x
  - $\delta = \min(d_1, d_2)$
  - Filter P_y for points within δ of middle of P_x
  - For each point p in filter, check $\exists$ points below
  - Return closest pair found

Master Theorem

- $T(n) = kT(n/b) + O(n^d)$
  → k = # of recursive calls
  → n/b = size of input to each call
  → $O(n^d)$ = time needed to merge
  $k < b^d \rightarrow O(n^d)$
  $k = b^d \rightarrow O(n^d \log n)$
  $k > b^d \rightarrow O(n^{\log_b k})$
- $T(n) = kT(n/b) + O(n^d \log^w n)$
  $k < b^d \rightarrow O(n^d \log^w n)$
  $k = b^d \rightarrow O(n^d \log^{w+1} n)$
  $k > b^d \rightarrow O(n^{\log_b k})$

DP: 0-1 Knapsack

- Weights $W_1 \dots W_n$; Values $V_1 \dots V_n$
- Capacity C — Max value, $W \leq C$
- Memo [i][c]: Max value considering items 1 ... i, capacity $C' \leq C$
- If $C == 0 \ ||\ i == 0 \rightarrow 0$
- Else if $W_i > C \rightarrow K(i-1, C)$
- Else $\rightarrow \max\{K(i-1, C), K(i-1, C-W_i) + V[i]\}$

DP: Longest Common Subsequence

LCS(X, Y): LCS of strings X, Y
Memo[i, j]: stores LCS X[1 ... i], Y[1 ... j]
>> memo[0, j], memo[i, 0] := 0;
>> for memo[i][j]:
  if X[i] == Y[j]
    memo[i-1][j-1] + 1;
  else
    max{memo[i-1][j], memo[i][j-1]}

(dependent entries done when actual comp)

DP: Longest Increasing Subsequence

- LIS(X[1 ... n]): LIS of string X
- memo[i]: LIS that ENDS AT X[i]
- memo[i] = $1 + \max\{$memo[j] s.t. X[i] > X[j] AND $j < i\}$
- LIS(X): $\max\{$memo[i]$\}$
- Find subproblem to solve (KEY!!)

DP: Floyd-Warshall (All pairs shortest path)

- NO NEGATIVE CYCLES!!!
- Label vertices 1, 2, ..., |V|
- $d^k(i,j)$: Best path using intermediate vertices 1, 2, ..., k
- $d^0(i,j) = w(i,j)$ (doesn't use k)
- $d^k(i,j) = \min\{d^{k-1}(i,j),$
  $d^{k-1}(i,k) + d^{k-1}(k,j)\}$

Greedy:

- Make locally optimal choices
- Kruskal's (MST): Choose smallest edge unless it forms cycle. Repeat until |V-1| edges
- In general: Proving greedy = optimal can be done by transforming optimal sol to greedy sol

Polytime Reductions (Karp)

- To show $A \leq_p B$ (A is no harder than B):
>> Efficient D_A (on input x):
>> Construct f(x) s.t. $x \in A \Leftrightarrow f(x) \in B$
>> return $D_B(f(x))$
- So, just construct f(x): $X_A \rightarrow X_B$ to show $A \leq_p B$

NP-Complete Languages

ORs ↓ ANDs ↓
- 3-SAT: Boolean formula like: $(x \vee \neg y \vee z) \wedge (\dots) \wedge \dots$
  - EXACT 3-CNF formula: no repeated vars in a clause
- Vertex-Cover (G, k): G has vertex cover of size k
  - All edges of G are "covered" by vertices in C
  - VC ∈ NP because certificate = cover of size k
  - Prove: 3-SAT $\leq_p$ Vertex-Cover:
    f: $\phi \rightarrow \langle G_\phi, k\rangle$ s.t. $\phi \in$ 3-SAT $\Leftrightarrow (G,k) \in$ VC
    Don't need every graph to transform into valid ϕ; just need every ϕ to transform into valid (G, k)
- Set-Cover: Given set S containing subsets of U, does U have k-set-cover in S? $\langle U, S, k\rangle$
  - In NP: Give $S' \subseteq S$ s.t. $\bigcup S' = U$, $|S'| = k$
  - Show Vertex-Cover $\leq_p$ Set-Cover
    → U is set of edges
    → S contains 1 set per vertex w/ edges covered
    → Note: each elem appears at most 2 times in S; that's okay!
    → Doesn't have to apply to all (U, S, k) — just all graphs
- Hampath $\langle G, s, t\rangle$: G has path from vertex s to t such that every vertex visited at most once
- Hamcycle $\langle G\rangle$: G has some path that starts/ends at some vertex, and visits each vertex exactly once
  - NP: certificate is just the path
  - Hampath $\leq_p$ Hamcycle: Consider $\langle G, s, t\rangle$. Add a new vertex, u, connected to s and t. So, if G has Hampath from s and t, Hamcycle can take the hampath and go thru u. To show !Hampath → !Hamcycle, prove contrapositive
- Subset-Sum $\langle S, t\rangle$: Given set of integers S, is there some subset that sums to t?
- Independent-Set $\langle G, k\rangle$: Given graph, does there exist subset of vertices I s.t. no two vertices share edge?
  → Call VertexCover $(G, n-k)$
- Clique $\langle G, k\rangle$: G has clique of size $\geq k$ (clique: all vertices in clique share edge between them)
  → construct G' s.t. $\exists (u,v) \in G \Leftrightarrow \nexists (u,v) \in G'$
  → Independent-Set (G', k)
- TSP $\langle G, k\rangle$: Undirected, weighted G has tour (Hamcycle) with weight at most k
- 3-Colorable $\langle G\rangle$: All vertices in G can be colored such that no neighbors have the same color
- Max-Cut $\langle G, k\rangle$: Vertices of G can be partitioned into two sets s.t. k edges cross the cut

- $A \leq_p A$ for all languages A
- $A \leq_p B,\ B \leq_p C \rightarrow A \leq_p C$
- $A \leq_p B \rightarrow A \leq_T B$
- $\exists\, A \leq_T B$ and $A \not\leq_p B$
(use language known to not be in P, like Bounded-Halt, to show)

Search Problems

- NP-Hard search problem has efficient algorithm iff decision version does
- Maximization (Max Clique, Knapsack), Minimization (TSP), Exact (SAT)

Big-O Definitions + Proofs

$f(n) = O(g(n)) \rightarrow |f(n)| \leq M|g(n)|,\ \forall n > n_0$
$f(n) = \Omega(g(n)) \rightarrow |f(n)| \geq M|g(n)|,\ \forall n > n_0$
$f(n) = \Theta(g(n)) \rightarrow f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$
$f(n) = o(g(n)) \rightarrow |f(n)| < M|g(n)|,\ \forall n > n_0$
$f(n) = \omega(g(n)) \rightarrow |f(n)| > M|g(n)|,\ \forall n > n_0$

Proving $f(n) = O(g(n))$:
- Master Thm
- $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < c$ for $c \in \mathbb{R}$
(L'Hopital: $\lim \frac{f(n)}{g(n)} = \lim \frac{f'(n)}{g'(n)}$)
Note: $\frac{d}{dx} a^x \rightarrow \ln(a)\, a^x$
- Find witnesses M, n_0

Prove $f(n) \neq O(g(n))$:
- Use Master Thm to convert
- $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = \infty$ (diverging not enough)
- Show $g(x) = o(f(x)) \Leftrightarrow \lim_{n\to\infty} \left|\frac{g(n)}{f(n)}\right| = 0$
- Argue 2D3-style: $\forall M, n_0, \exists n > n_0$ s.t. $|f(n)| \geq M|g(n)|$ (Construct n value)

Proof: Kruskal finds MST
>> Suppose T' is MST. Show that T' can be transformed to T (Kruskal)
>> Let $e_1, e_2, \dots e_{n-1}$ be edges of T (sorted, in order of addition)
>> Proof by induction:
  >> Base Case: $T(e_1 \rightarrow e_n) = T'(e_1 \rightarrow e_k)$
  >> IS: Suppose $e_1 \rightarrow e_k$ OK. Consider e_{k+1} ($k+1^{th}$ in T)
  >> if $(e_{k+1} \in T')$, we are good.
  >> else: adding e_{k+1} to T' creates a cycle C. Then, $\exists f \in T'$ on cycle C
  Remove f and swap w/ e_{k+1}. This is optimal b/c as f & T,
  Weight(f) ≥ Weight(e_{k+1}).

Undecidable Languages

- $L_{DIAG} = \{P \mid \& \notin L(P)\}$ (rejects own source code)
  - Claim: $L(P^*) \neq L_{DIAG}$ for any program P^*
    → Suppose $L(P^*) = L_{DIAG}$.
    → If $P^* \in L(P^*)$, then by defn. $P^* \in L_{DIAG}$
    ⇒ P^* rejects own source code ⇒ $P^* \notin L(P^*)$
    → If $P^* \notin L(P^*)$, then $P^* \in L_{DIAG}$ ⇒
    P^* accepts own source code ⇒ $P^* \in L(P^*)$
- $L_{ACC} = \{\langle M, x\rangle : M$ accepts $x\}$
→ Assume: D_{ACC}
→ D_{DIAG} := return ! $D_{ACC}(\langle M, \langle M\rangle\rangle)$
→ So, contradiction! $L_{DIAG} \leq_T L_{ACC}$, L_{ACC} undecidable
- $L_{HALT} = \{\langle M, x\rangle : M$ halts on $x\}$
→ Assume: D_{HALT}
→ D_{ACC} := if $[D_{HALT}(M,x))$ return $M(x)$
else return false (reject)
→ $L_{ACC} \leq_T L_{HALT}$
- $L_{\varepsilon\text{-ACC}} = \{\langle M\rangle : M$ accepts $\varepsilon\}$
→ Assume $L_{\varepsilon\text{-ACC}}$ decidable ($\exists\, D_{\varepsilon\text{-ACC}}$)
→ D_{ACC} := Let M' = "on input w:
if w = ε, return M(x)
else accept"
return $D_{\varepsilon\text{-ACC}}(\langle M'\rangle)$ // if M' accepts, M acc x
- $L_{\varepsilon\text{-HALT}}(\langle M\rangle)$: M halts on ε
- $L_{EQ}(\langle M_1, M_2\rangle)$: $L(M_1) = L(M_2)$

Properties of Decidable Languages:

- If L is decidable, $\bar{L}$ is decidable
- If L_1, L_2 decidable, $L_1 \cap L_2$ decidable
- If L_1, L_2 decidable, $L_1 \cup L_2$ decidable
- If L decidable, $L \leq_T L'$ for all L'
- $\forall L,\ L \leq_T \bar{L}$

Properties of Undecidable Languages

- If L undecidable, $\bar{L}$ also undecidable
- For undecidable L_1, L_2, $(L_1 \cap L_2)$ not necessarily undecidable (consider: $L_1 \cap L_2 = \phi$)
- Similarly, $L_1 \cup L_2$ not necessarily undecidable (consider: $L_2 = \bar{L_1}$)

- If $A \leq_T B$, $B \leq_T C \rightarrow A \leq_T C$

Search-To-Decision Examples

- Vertex-Cover: First, find min VC through linear search (determine k^*)
Run decider on $G - \{v\}$; if v in some min VC, $G - \{v\}$ has VC of size k^*-1. Else, does not
If v in min-VC, add v to separate set; $G = G - v$
- SAT:
>> For each variable x_i:
  $\phi[x_i] = F$
  if $(!D(\phi))$ $\phi[x_i] = T$
  return ϕ
- Linear-Inequality (Given conditions $(x \vee x_2), (\dots)$ find minimum # of variables set to true to satisfy ALL
>> for all x_i
  $M' = M + \{x_i \vee x_i\}$
  if $(L-1(M', M'))$ → $sol[x_i] = True$, $M = M - x_i$

Approximation for Optimization

- Max: $\alpha \cdot OPT \leq$ soln $\leq OPT$ ($\alpha \leq 1$)
- Min: $OPT \leq$ soln $\leq \alpha \cdot OPT$ ($\alpha \geq 1$)
- Can restrict input to get efficient soln.
→ Knapsack: $W = O(n^k)$
→ Max-Cut: Planar Graphs

Big-O Definitions + Proofs

- Alphabet Σ: non-empty, finite set of symbols
- String: finite sequence of symbols from alphabet
- Σ^n: Set of all strings of length = n
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$
- Language $L \subseteq \Sigma^*$
- Grammar: Finite set of rules that defines a language

Chomsky's Hierarchy

- Regular (DFA)
- Context-Free (Push-down A)
- Context-Sensitive (Linear Bounded Automata - LBA)
- Recursively Enumerable (TMs)

Existence of Undecidable Language (DIAG)

- Show $|L_{Decidable}|$ countable
- Show $|L_{ALL}|$ uncountable
① L_D is countable b/c each C++ program corresponds to at most one language
C++ program = binary string
Order LEXICOGRAPHICALLY
② L_{ALL} is uncountable by diagonalization proof:

N	ε?	0	1	00	01	10	...
L_0	N	Y	N	N	Y	N	
L_1	N	N	Y	Y	Y	Y	
L_2	N	Y	N	N	Y	Y	
⋮							

Consider $\bar{L}$, diagonal with answers flipped. $\bar{L}$ not in table, which creates contradiction

Table of Languages

Name	Defn.	D	R	C-R
L_{DIAG}	$\langle M\rangle : \langle M\rangle \notin L(M)$	X	X	✓
L_{ACC}	$\langle M, x\rangle$: M accepts x	X	✓	X
L_{HALT}	$\langle M, x\rangle$: M halts on x	X	✓	X
L_ϕ	$\langle M\rangle : L(M) = \phi$	X	X	✓
L_{EQ}	$\langle M_1, M_2\rangle : L(M_1) = L(M_2)$	X	X	✓
$L_{\varepsilon\text{-HALT}}$	$\langle M\rangle$: M halts on ε	X	✓	X

Recognizability

- Language L recognizable iff $L = L(M)$
- If L_1, L_2 recognizable, so are $L_1 \cup L_2$; $L_1 \cap L_2$
→ Alternately simulate steps of M_1, M_2
- If L, $\bar{L}$ recognizable → L is decidable
→ Again, alternately simulate M_1, M_2 — use output corresponding to whichever one halts
- If L undecidable, then either L, $\bar{L}$ unrecognizable (or both)
- Co-recognizable: if $\bar{L}$ is recognizable
  - Countably many langs either recognizable or co-recognizable
  - If both, then decidable; L_{EQ}: Neither
- Prove $\bar{L}_\phi$ is recognizable:
  - $\bar{L}_\phi(\langle M\rangle)$: All TMs s.t. $L(M) \neq \phi$
  - On input $\langle M\rangle$: simulate M(x) by dovetailing $\forall x$
  - Dovetailing: Order inputs lexicographically $(x_0, x_1, \dots)$
    → Simulate 1 step M(x0)
    → Simulate 1 step M(x0), M(x1) ...
    → ∞ machines in parallel

Approximation Examples

- VC: ① Cover-and-Remove: Find vertex, delete it + edges
  - Worst-Case: $\alpha = |n-1| \cdot |M|$
  ② Greedy: Find vertex w/ largest # of edges
  - Still worst-case $\alpha = |n-1|$
  ③ Double-Cover: Find edge, add both u, v to VC
  - Worst Case: $\alpha = 2$
- Max-Cut (see NP-Comp list)
  - Local Search (Greedy):
  >> color all vertices red initially
  >> find vertex w/ more same-color neighbors and flip color until no such vertices exist
  >> return two subsets of vertices
  → Efficient b/c # of cutting edges increases w/ each flip
  → $\alpha = \frac{1}{2}$ (b/c every vertex has at least 50% diff. color neighbors, so at least half edge cross)
- Knapsack:
  ① Pick based on v/w ratio
  ② Take item w/ highest value that fits
  ③ Run both ①, ② and pick better soln.
  - $\alpha = 1/2$
  - Why? $V_{frac} \geq V_{0-1}$. $V_{frac} \leq V_0 + V_1$
- 3-CNF, Max-Cut can use randomized algo

Membership Problem

- Given $w \in \Sigma^*$, is $w \in L$?
- M decides L if accepts if $w \in L$ and rejects if $w \notin L$ (no looping)
- M recognizes L if accepts if $w \in L$, does not accept (reject OR loop) if $w \notin L$

- DFA (Deterministic Finite Automata)
  - can decide any language a human can without external memory
  - $\{Q, \Sigma, \delta, q_0, F\}$
  → Q: Set of states
  → Σ: Alphabet
  → $\delta: Q \times \Sigma \rightarrow Q$ (transition func)
  → $q_0 \in Q$: initial state
  → $F \subseteq Q$: Accept states
- Decidable by DFA ⇔ Regular Lang
- $L \in$ Regular → L', L^*, Lreverse ∈ REG
- Reg Ex = DFA (*: ≥0 ; +: ≥1)
- Read input ONCE, left-to-right

Turing Machines

- Inf. 1D tape of cells — initially have input followed by blanks ⊥ ∉ Σ
- Tape head, moves L or R
  - reads symbol under + can write
- State machine controlling tape head; special accept/reject states for termination
- $(Q, \Gamma, \Sigma, \delta, q_s, q_a, q_r)$
  → Q: set of states
  → Σ: input alphabet
  → T: tape alphabet, $\Sigma \cup \{\sqcup\} \cup ??$
  → δ: transition function
  → $q_s \in Q$: start; q_a: accept; q_r: reject
- Language of M ⇔ whatever M recognizes
- M decides L(M) if always halt
- C-T Thesis: All known computational models equivalent to TM
- Can simulate 2-tape TM w/ 1-tape
  → parity to determine tape
  → invent symbols to keep track of head
  → To change state: Find head 1, modify
  Find H2, modify
- C++ is Turing Complete

General Turing Reduction ($A \leq_T B$)

- Assume B decidable (for contradiction)
- Then, ∃ D_B (decider for B)
- Construct D_A "using" D_B
- Then, A decidable ⇒ contradiction!!

This doesn't have to be w/ undecidable A/B — can just show $A \leq_T B$ for any 2 languages

L_A is undecidable (all TMs s.t. $L(M) = A$)

- Assumption: A recognizable, $A \neq \phi$
- $L_{HALT} \leq_T L_A$:
  → $D_{HALT}(M, x)$:
  Let M' = "on input w:
  simulate M(x)
  return $R_A(w)$"
  return $D_A(M')$ — either A ($M(x)$ halts) bcs ϕ ($M(x)$ loops) false

Rice's Theorem

- For some set of languages $\mathbb{P}$, if ∃ recognizable $A \in \mathbb{P}$, $B \notin \mathbb{P}$, then $L_{\mathbb{P}}$ undecidable ($L_{\mathbb{P}}(\langle M\rangle) = L(M) \in \mathbb{P}$)
- Any non-trivial question about behavior of a program tends to be undecidable
  → M writes "1" to tape on input $\varepsilon \Rightarrow$ UNDEC.
  → M accepts odd # of strings

Prove Decidability w/ Rice's

① Find $\mathbb{P}$ s.t. $L = L_{\mathbb{P}}$
($\forall M \in L$, $L(M) \in \mathbb{P}$)
② Show $\mathbb{P}$ is trivial
③ Conclude L decidable.

Proving Undecidability using Rice's

① Find semantic property $\mathbb{P}$ s.t. $L = L_{\mathbb{P}}$
② Show $\mathbb{P}$ is non-trivial ($L_1 \in \mathbb{P}$, $L_2 \notin \mathbb{P}$)
③ Conclude L is undecidable

Ex: $L = \{\langle M\rangle$: M accepts all strings of length less than 376$\}$

Let S_{376} be set of all strings less than length 376. Then let $\mathbb{P}$ be the set of all languages L^* s.t. $S_{376} \in L^*$. Then, if $\langle M\rangle \in L$, M accepts all s s.t. $s \in S_{376}$. As $S_{376} \in L(M)$, $L(M) \in \mathbb{P}$ and thus $\langle M\rangle \in L_{\mathbb{P}}$.
Furthermore, $\Sigma^* \in \mathbb{P}$ and $\phi \notin \mathbb{P}$, so $\mathbb{P}$ is non-trivial (both recognizable).
So, by Rice's, L is undecidable.

Kolmogorov Complexity

- $KC(x)$: length of shortest program outputting x
- Complexity independent of prog. lang.
- Implication: ∃ some incompressible string for every length $n \geq 1$.

Randomized Algorithms

- Assume "good" source of randomness
- Can avoid worst-case through random behavior
- Metrics: Expected (average) runtime, runtime w/ high probability $(1-\delta)$, error probability (1-sided / 2 sided)

Max 3-SAT

- Given 3CNF formula (exact), maximize the number of satisfied clauses
- Claim: Every exact 3CNF formula can satisfy $7/8$ of the clauses
  - random assignment, on average, does this
- Let $X(S)$ = # of clauses satisfied by S, a random assignment to ϕ ($\frac{1}{2}T$, $\frac{1}{2}F$ for each)
  - $\rightarrow E[X] = E[X_1 + X_2 + \dots + X_n]$ (n clauses)
  - $= E[n(7/8)] \leftarrow Pr(X_i) = 7/8$
  - $= \frac{7}{8}n$
- Markov's Inequality: (RV X, non-negative)
  $Pr(X \geq a) \leq E[X]/a$
  - can use to bound probability of Randomized
  - $z' = n - z$; $Pr(z' \geq n/2) \leq (n/8)/(n/2) = 1/4$
  - Corollary: $Pr[z \geq a \cdot E[z]] \leq 1/a$
  - Reverse: $Pr[z > a] \geq \frac{E[z] - a}{B - a}$ if B is upper bound for vol. z

Union Bound: idea that union $\leq$ sum
$P(A \cup B \cup C \dots) \leq P(A) + P(B) + \dots$

Quicksort

- Randomly pick pivot, partition on pivot, and recursively sort each half
- Idea: $O(QS) = O(\#$ of comparisons$)$
  - 2 elements compared at most once (one is pivot)
  - X_{ij}: indicator R.V. if i, j compared
    - $\rightarrow$ at time of comparison, nothing between i,j pivot
    - $\rightarrow$ So, $E[X_{ij}] = \frac{2}{j-i+1}$
  - Total runtime becomes $\approx \ln(n)$

Randomized Primality Testing

- Theme: randomly test property ALWAYS satisfied by primes and rarely satisfied by composite #s
  - $\rightarrow$ False positives possible ("probably" prime)
- Extended Fermat's Little Theorem:
  - if n is prime, $a^{n-1} \equiv 1 \mod n$ for all $1 \leq a \leq n-1$
  - if n is composite, $a^{n-1} \not\equiv 1 \mod n$ for at least half the set of $2 \leq a \leq n-1$ except in the case of Carmichael numbers, which behave like primes
- Proof: Suppose $p = n$ is prime and consider $2 \leq a \leq p-1$.
  - Fact: Since $\gcd(a,p) = 1$, there exists a^{-1} such that $a \cdot a^{-1} \equiv 1 \pmod p$.
  - Claim: $\{a, 2a, 3a, \dots, (p-1)a\} \equiv \{1, \dots, p-1\} \pmod p$.
    - It's impossible for $ia \equiv 0 \pmod p$ for $1 \leq i \leq p-1$. (not possible as $i, a < p$; multiple exist)
    - It's impossible for $ia \equiv ja \pmod p$ for $1 \leq i < j \leq p-1$. (same $(i-j)a$)
  - So $a \cdot 2a \cdot 3a \cdot \dots \cdot (p-1)a \equiv 1 \cdot 2 \cdot \dots \cdot (p-1) \pmod p$
  - Therefore, $a^{p-1} \equiv 1 \pmod p$. (all of how)
- Fermat's Primality Test:
  - Pick witness $a \in \{2, \dots, n-1\}$ randomly
  - if $(a^{n-1} \equiv 1 \mod n) \rightarrow$ prime
  - otherwise, composite
  - (do multiple times to lower error)
  - compute a^{n-1} using FME
- Miller-Rabin Primality Test
  - $n = 2^s d + 1$ (n always odd)
  - Pick witness uniformly at random
  - "Prime" if $a^{n-1} \equiv 1 \mod n$ AND ($a^d \equiv 1 \mod n$ OR $a^{2^r d} \equiv -1 \mod n$ for $r < s$)
  - works against Carmichael Numbers

FLT (extended)

For prime p, $0 \leq a \leq p-1$
$a^{1 + k(p-1)} \equiv a \mod p$

Euler's Theorem

For any $a \in \mathbb{Z}_n$ coprime with n ($\gcd = 1$):
$a^{\phi(n)} = 1 \pmod n$
$\phi(n)$: # of elements in $\mathbb{Z}_n$ coprime w/ n
$\phi(p) = p - 1$ for prime p
$\phi(pq) = (p-1)(q-1)$ for primes p, q

Euler's For FME

Compute $34^{343434} \mod 77$.
- $\gcd(34, 77) = 1$; $77 = 7 \cdot 11$
- $\phi(77) = (7-1)(11-1) = 60$
- $34^{343434} \equiv 34^{(5723)(60) + 54} \mod 77$
  $\equiv (1)^{5723} 34^{54} \mod 77$

[Do normal FME from here on out]

- Show $L \in NP$ when proving $L \in$ NP-Complete

- Los Vegas Algos: Worst-case runtime maybe unbounded, but output always correct
  - $\rightarrow$ Randomized Quicksort ($n \log n$ vs. n^2)
  - $\rightarrow$ Using unfair coin to flip H/T w/ equal probability (flip in pairs until TH or HT; return first flip)
- Monte Carlo Algos: bounded runtime, but can sometimes produce wrong answer (one-sided vs. two-sided)
  - can run repeatedly to decrease error
- Atlantic City Algo: numeric val $\pm \varepsilon$; run multiple time + take median

Extended Euclidean Algo

$\gcd(259, 23) = \gcd(23, 259 - 11(23))$
$= \gcd(23, 6)$
$= \gcd(6, 23 - 3(6))$
$= \gcd(6, 5)$
$= \gcd(5, 6 - 1(5)) = \gcd(5, 1) = 1$

Find $23^{-1} \mod 259$:
$1 = 6 - 5 = 6 - (23 - 3(6))$
$1 = 4(6) - 23$
$1 = 4(259 - 11(23)) - 23$
$1 = 4(259) - 45(23)$
$1 \mod 259 = (-45)(23) \mod 259 = (214)(23) \mod 259$

Fast Modular Exponentiation

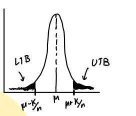
- Use binary representation to calc faster
- $7^{117} \mod 13 = 7^{(64+32+16+4+1)} \mod 13$

$7^1 \rightarrow 7$ | $7^8 \rightarrow 3$ | $7^{64} = -4$
$7^2 \rightarrow -3$ | $7^{16} \rightarrow -4$
$7^4 \rightarrow -4$ | $7^{32} \rightarrow 3$

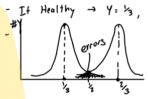
- $7^{117} \mod 13 \equiv (-4 \cdot 3 \cdot -4 \cdot -4 \cdot 7) \mod 13$
  $\equiv (16 \cdot 27 \cdot 7) \mod 13$
  $\equiv 8 \mod 13$

Chernoff Bounds

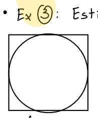
- If $X_1, X_2, \dots$ are i.i.d RVs w/ expectation μ, then $\frac{1}{n}\sum_{i=1}^{n} X_i$ converges to μ ($n \rightarrow \infty$)
- Lower Tail Bound:
  $Pr\left(\sum_{i=1}^{n} X_i \leq \mu n - k\right) \leq e^{-2k^2/n}$
- Upper Tail Bound:
  $Pr\left(\sum_{i=1}^{n} X_i \geq \mu n + k\right) \leq e^{-2k^2/n}$
- These hold for X_i w/ range $[0, 1]$, expected val μ, $k > 0$.



- Ex ①: Distinguishing Biased Coins:
  - $\rightarrow$ Given coin w/ $Pr(H) = \frac{1}{2} + \varepsilon$, $P(T) = \frac{1}{2} - \varepsilon$
  - ε can be positive / negative
  - $\rightarrow$ Test: Flip coin n-times; if at least $n/2$ heads, determine H-bias. Otherwise, determine T-bias.
  - $\rightarrow P(FN) = P(FP) = P(\text{error}) \leq e^{-2\varepsilon^2 n}$
  - $\rightarrow$ This is because $k = \varepsilon n$
- Ex ②: COVID test w/ 2-sided error
  - If COVID $\rightarrow Y = 2/3$, $N = 1/3$
  - If Healthy $\rightarrow Y = 1/3$, $N = 2/3$



  - Same as distinguishing biased coins with $\varepsilon = 1/6$
- Ex ③: Estimating π with darts



$\frac{\text{Area of Circle}}{\text{Area of Board}} = \frac{\pi}{4}$

- $Pr\left(\left|\sum X_i - \mu n\right| \geq k\right) \leq 2e^{-2k^2/n}$
  $Pr\left(\left|\frac{4}{n}\sum X_i - \pi\right| \geq \frac{4k}{n}\right) \leq 2e^{-2k^2/n}$
  Accurate within ε: $k = n\varepsilon/4$
  Correct with prob $= 1 - \delta$: $n \geq 8\ln(2/\delta)/\varepsilon^2$

Things to Remember

- Everything in NP is decidable
- f for polytime mappings is neither one-to-one nor onto
- $L \leq_p L_1$ if $L \in P$, L_1 nontrivial
- Don't Assume $P \neq NP$!! For T/F problems especially, don't assert NP-Hard Problems are undecidable in poly-time.
- Set of ALL languages uncountable
- Read Carefully ☺ Check work - did you answer everything Q was asking for??

Cryptography

- Alice + Bob communicating over insecure channel; adversary Eve seeing every message and trying to decode it
- Eve should not be able to decipher even if she knows protocol

One-Time Pads + Caesar Cipher

- C = Ciphertext; M = msg; k = key
- $c = m + k$
- Weaknesses: must agree on key prior; cannot reuse same key (info leakage)
  - $\rightarrow c = m + k$; $c' = m' + k$
  - $\rightarrow c - c' = m - m' \rightarrow$ valuable info

Diffie-Hellman

- Generators:
  - $\mathbb{Z}_p = \{0, 1, \dots, p-1\}$
  - g is generator of $\mathbb{Z}^*_p = \mathbb{Z}_p - \{0\}$ iff for all $x \in \mathbb{Z}^*_p$, $\exists i \in \mathbb{N}$ such that $g^i \equiv x \mod p$.
  - g's powers cycle thru all $p-1$ values possible for $g^? \mod p$
  - Fact: every prime has some generator
  - DLog Conjecture: Given $p, g, x \in \mathbb{Z}^p$, no efficient algo exists to find $i \in \mathbb{N}$ s.t. $g^i = x \pmod p$
- D-H Protocol:
  - Params: Big prime p, generator g (known)
  - Alice picks $a \in \mathbb{Z}_p$, Bob pick $b \in \mathbb{Z}_p$
  - Alice sends $x = g^a \mod p$ to Bob
  - Bob sends $y = g^b \mod p$ to Alice
  - Secret key $= k = g^{ab} \mod p = y^a = x^b$
  - Eve cannot figure out a, b in P-time
  - Alice, Bob both now know secret key
- Man-in-the-Middle Attack:
  - If messages modified, this can be messed with
  - Eve chooses $c \in \mathbb{Z}_p$, uses as fake key

RSA Protocol

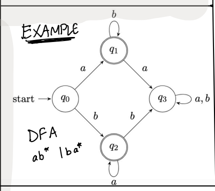
- Idea: Public Key (lock) that everyone has; Private key (key) that only you have; Anyone can send messages to you
- Everything is a number with some upper bound; we want to securely transmit $m \in \mathbb{Z}_n$ ($\lceil \log_2 n \rceil$ bits) (like file size limit in email)
- $\mathbb{Z}_n$: set of messages
  E: encoding func.; D: decoding func.
  $D \circ E = D(E(m))$ = identity function
  Find non-obvious identity functions
- RSA identity:
  $a^{1 + k\phi(n)} = a \mod (n)$
  $n = p \cdot q$ (2 primes)
- Protocol:
  Pick large, secret primes p, q. $n = pq$
  Generate public/private keys (e, d)
  $\rightarrow ed = 1 + k(p-1)(q-1)$
  - Alice sends Bob (n, e)
  - $c = m^e \mod n$ (Bob sends alice)
  - $m \mod n = c^d$ (Alice decodes)

RSA Example

- $n = pq = 3 \cdot 17 = 51$ (secret)
- $(e, d) = (3, 11)$ — $ed = 33 = 1 + (2)(16)$
- Alice sends $(n = 51, e = 3)$ publicly
- Bob encodes $m = 4$ as $4^3 \mod 51 = 13$
- Alice recieves 13, decodes as: $13^{11} \mod 51 = 4$

- D-H, RSA both rely on NP-Intermediate problems like integer factorization or Discrete Logarithm (DLog). These problems are "expected to be" NP-intermediate. If P = NP, they become P, which kinda kills modern security systems

Example



Example: Knapsack is NP-Complete

KNAPSACK(n, V, W, C, k)

① KNAPSACK $\in$ NP

Consider a verifier for KNAPSACK that takes in subset of items S. Then, Verifier can sum both weights and values of n items in $O(n)$ time. Comparisons of these sums to C, k respectively is in $O(1)$ time. Furthermore, solution is correct iff $\sum W_i \leq C$ and $\sum V_i \geq k$. So, this verifier is both efficient and correct; thus, KNAPSACK $\in$ NP.

② KNAPSACK $\in$ NP-Hard

We will show SUBSET-SUM $\leq_p$ KN.. by constructing $f: \langle B, t \rangle \rightarrow \langle n, V, W, C, k \rangle$ such that $\langle B, t \rangle \in$ SUBSET-SUM iff $f(\langle B, t \rangle) \in$ KNAPSACK.

$f(\langle B, t \rangle)$:
- $n = |B|$
- $V, W = B$
- $C, k = t$
- return $\langle n, V, W, C, k \rangle$

Complexity: This mapping can be computed in worst-case linear time for any $\langle B, t \rangle$ as values are simply copied over. Thus, f is efficient.

Correctness: Will prove both ways to show iff

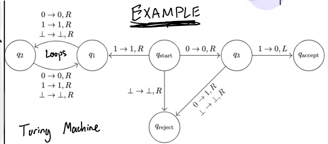
① Suppose $\langle B, t \rangle \in$ SUBSET-SUM. Then, $\exists B^*$ s.t. $\sum_{i=1} B^*_i = t$ by defn. Then, some subset of items $B^* \in$ KNAPSACK.
$\sum W[i] = \sum B^* = t = C$ (so $\sum W_i \leq C$)
$\sum V[i] = \sum B^* = t = k$ (so $\sum V_i \leq k$)
So, $\langle B, t \rangle \in SS \rightarrow f(\langle B, t \rangle) \in$ KNAPSACK

② Suppose $f(\langle B, t \rangle) \in$ KNAPSACK. Then, $\exists$ some subset of items S such that $\sum_S W \leq C \wedge \sum_S V \geq k$. As $W = V = B$ and $C = k = t$, we can substitute:
$\sum_S B \leq t \wedge \sum_S B \geq t$
So, $\sum_S B = t$; so, S forms the subset that sums to t, and $\langle B, t \rangle \in SS$ by definition.

As we have proved both "sides" of mapping, we can assert $SS \leq_p KP$. As SS is known to be NP-Hard, KNAPSACK is thus NP-Hard.

As KNAPSACK $\in$ NP $\wedge$ KNAPSACK $\in$ NP-Hard, KNAPSACK $\in$ NP-Complete.

Example



L = Anything starting w/ 01

Example: Turing Reduction

$L_2 = \{\langle M \rangle : M$ is a TM that loops on the input string "eecs376"$\}$.

Solution: The language L_2 is undecidable. We will prove this via Turing reduction with $L_{\text{HALT}} \leq_T L_2$. Assume that we have some decider D_2 for L_2. We will construct a Turing machine D_{HALT} that decides L_{HALT} using D_2 as follows:

```
D_HALT(⟨M⟩, x)
1: Construct a Turing machine M' as follows:
     M'(w)
     1: Run M(x)
     2: Accept
2: Run D_2(M')
3: if D_2 accepts M' then
4:     Reject
5: else
6:     Accept
```

Note that the Turing machine M' we construct have $L(M') = \Sigma^*$ when M halts on x and it will loop on all the strings if M loops on x.

$\forall(\langle M \rangle, x) \in L_{\text{HALT}} \implies M$ halts on $x \implies L(M') = \Sigma^* \implies M'$ accepts the string "eecs376" $\implies D_2$ rejects $M' \implies D_{HALT}$ accepts $(\langle M \rangle, x)$.
$\forall(\langle M \rangle, x) \notin L_{\text{HALT}} \implies M$ loops on $x \implies M'$ loops on all the strings, including the string "eecs376" $\implies D_2$ accepts $M' \implies D_{HALT}$ rejects $(\langle M \rangle, x)$.
Since D_{HALT} decides L_{HALT}, we have the Turing reduction $L_{\text{HALT}} \leq_T L_2$. Then we can know L_2 is undecidable as L_{HALT} is undecidable.

Prove that $L_\varnothing = \{\langle M \rangle : L(M) = \varnothing\}$ is NP-Hard by a reduction from 3SAT. Explain why $L_\varnothing$ is not NP-Complete.

Example: Polytime for undecidable lang

Solution: Recall that since 3SAT is NP-Complete, it has some exponential time decider D. Consider the following function $f : 3SAT \rightarrow L_\varnothing$:

$f(\phi)$:
- Construct the following program
- $M(w)$: return opposite of $D(\phi)$
- return $\langle M \rangle$

This function is efficient as we are constructing the program M, not running it. Further, this reduction is correct as

$\phi \in 3SAT \implies D(\phi)$ accepts $\implies M(w)$ rejects for all $w \implies \langle M \rangle \in L_\varnothing$
$\phi \notin 3SAT \implies D(\phi)$ rejects $\implies M(w)$ accepts for all $w \implies \langle M \rangle \notin L_\varnothing$

Therefore $L_\varnothing$ is NP-Hard. However, $L_\varnothing$ is not NP-Complete because $L_\varnothing$ is undecidable, whereas everything in NP is decidable (in at worst exponential time).

Solution:

Example: Polytime (full) includ NP-Proof

Field-Trip $= \{G = (V, E), C = (c_1, c_2, \dots, c_t) : \sum_j c_j \geq n$ and some assignment exists s.t for all vehicles, every pair of students in the vehicle knows each other$\}$.

Proof that Field-Trip $\in$ NP:

We define our verifier as follows. We expect a correct certificate to be a list of vehicles, where each vehicle specifies a set of the students it should contain. The vehicle at index i in this list should correspond to c_i in the list of vehicle carrying capacities.

V(G = (V, E), C, cert)

- Check that the certificate does not have more than the allowed number of vehicles and only uses students that exist and only uses them a single time. If this is not true, REJECT
- For each vehicle in cert, where its index is i, check that it has not exceeded its carrying capacity by checking $c_i \geq$ the number of students in the vehicle. Furthermore, check that each of the students knows each other by verifying there is an edge between them in the graph. If either of these statements is not true, REJECT
- Check that every student is assigned to a single vehicle by looping through the vehicles and checking that each student is within a vehicle. If this is not the case, REJECT
- ACCEPT

Correctness Analysis:

G, C, cert $\in$ Field-Trip $\implies$ there is some assignment of students to vehicles s.t. we will not exceed vehicle carrying capacity and every student in a vehicle knows each other $\implies \exists certs.t.V$ accepts

G, C, cert $\notin$ Field-Trip $\implies$ there is no assignment of students to vehicles s.t. we will not exceed vehicle carrying capacity and every student in a vehicle knows each other $\implies \forall certs.t.V$ rejects

Efficiency Analysis:

The first bullet point is a simple size check and a loop through the certificate.

For the second bullet point, checking the number of students in a vehicle can trivially be done efficiently. Checking that each student in a vehicle knows each other can be done in n^2 comparisons, where n is the number of students.

For the third bullet point, checking that every student is assigned to a vehicle can be done by simply looping through the assignments and making sure every student is present.

Since each line can be computed efficiently, the entire verifier is efficient.

Proof that Field-Trip is NP-hard by proving Clique $\leq_p$ Field-Trip:
$f(G = (V, E), k)$:

- Let $C = (k, 1, \dots, 1)$ be a list of vehicle capacities such that the first element is k and the rest $n - k$ elements are 1's.
- Return $(G = (V, E), C = (k, 1, \dots, 1))$

Analysis:

($\Rightarrow$) Suppose $(G = (V, E), k) \in$ Clique. There exists some set of students of size k such that every student in the set knows every other student in the set. On the vehicle of size k, we can allocate the above set of students, and every other vehicle is filled with a singleton set. Clearly, this assignment satisfies the second condition. Furthermore, $n = k + n - k$, ensures that we are able to transport all the students and $f(G = (V, E), k) \in$ Field-Trip.

($\Leftarrow$) Suppose $(G = (V, E), k) \notin$ Clique. Then there is no set of students of size k such that every student in the set knows every other student in the set. Regardless of how we fill the vehicles, if the vehicle of capacity k is filled to the maximum, there will be some pair of students in that vehicle who do not know each other. Because the capacity of all of the vehicles is exactly $n = k + n - k$, the vehicle of capacity k must be filled to the maximum, so $f(G = (V, E), k) \notin$ Field-Trip.

Clearly, $|C| \leq |V|$, so C can be constructed efficiently. Outputting the result is also efficient as each individual component to be output is efficiently sized, so efficiency holds for this mapping.

Choices C was selected in the previous question, so we will prove $L_{\text{NO-PUNCTUATION}}$ is undecidable using Rice's Theorem.

$L_{\text{NO-PUNCTUATION}} = \{\langle M \rangle :$ L(M) does not contain punctuation$\}$
$\mathbb{P} = \{L \subseteq \Sigma^* :$ L(M) does not contain punctuation$\}$
Then, by the following chain of equivalences, $L = L_{\mathbb{P}}$

$\langle M \rangle \in L_{\text{NO-PUNCTUATION}} \iff$ M does not accept any punctuation
$\iff$ L(M) does not contain punctuation
$\iff L(M) \in \mathbb{P}$
$\iff \langle M \rangle \in L_{\mathbb{P}}$

$\mathbb{P}$ is also nontrivial. $\emptyset \in \mathbb{P}$, because $\emptyset$ is empty, so it does not contain any punctuation. $\Sigma^* \notin \mathbb{P}$, because Σ^* contains all inputs. Both $\emptyset$ and Σ^* are recognizable languages. Thus, $L_{\text{NO-PUNCTUATION}}$ is undecidable by Rice's Theorem.