BU CS 332 – Theory of Computation

Lecture 22:

- NP-Completeness Example
- Space Complexity
- Savitch's Theorem

Reading:

Sipser Ch 8.1-8.2

HW 9 due on Monday 4/27

Final: 48-hour take-home

due 5/7 (?)

Mark Bun

April 22, 2020

NP-completeness

Definition: A language B is NP-complete if

1) $B \in \mathrm{NP}$, and
2) Every language $A \in \mathrm{NP}$ is poly-time reducible to B, i.e., $A \leq_{\mathrm{p}} B$ ("B is NP-hard")

Theorem: If $C \in \mathrm{NP}$ and $B \leq_{\mathrm{p}} C$ for some NP-complete language B, then C is also NP-complete

e.g. SAT, 3-SAT

$3SAT$ (3-CNF Satisfiability)

Definition(s):

- A literal either a variable of its negation x_5 , $\overline{x_7}$
- A clause is a disjunction (OR) of literals Ex. $x_5 \vee \overline{x_7} \vee x_2$
- A 3-CNF is a conjunction (AND) of clauses where each clause contains exactly 3 literals

Ex. $C_1 \wedge C_2 \wedge \ldots \wedge C_m =$

$$(x_5 \vee \overline{x_7} \vee x_2) \wedge (\overline{x_3} \vee x_4 \vee x_1) \wedge \cdots \wedge (x_1 \vee x_1 \vee x_1)$$

$3SAT = \{\langle \varphi \rangle | \varphi \text{ is a satisfiable } 3 - \text{CNF}\}$

Cook-Levin Theorem: $3SAT$ is NP-complete

$3SAT \le_p A_{TM}$ (A_{TM} is NP-hard)

Idea: Hardcode input formula φ into a TM M

s.t., say, M accepts ϵ iff φ is satisfiable

M = "On input w:

1) Ignore w
2) Try φ on all possible assignments and check if any accept"

Computing $\langle M, \epsilon \rangle$ from $\langle \varphi \rangle$ can be done in poly-time

---

$A_{TM} \notin NP$ (so A_{TM} <u>not</u> NP-complete)

$A \le_p B$ and $B \in NP \Rightarrow A \in NP$

Some general reduction strategies

- Reduction by simple equivalence

  Ex. $INDEPENDENT - SET \leq_{\mathrm{p}} VERTEX - COVER$
  and $VERTEX - COVER \leq_{\mathrm{p}} INDEPENDENT - SET$

G has an ind. set of size $\geq k$ $\Leftrightarrow$ G has a vertex cover of size $n-k$
[complement of an ind. set is a vertex cover]

- Reduction from special case to general case

  Ex. $VERTEX - COVER \leq_{\mathrm{p}} SET - COVER$

- Gadget reductions

  Ex. $3SAT \leq_{\mathrm{p}} INDEPENDENT - SET$

Independent Set

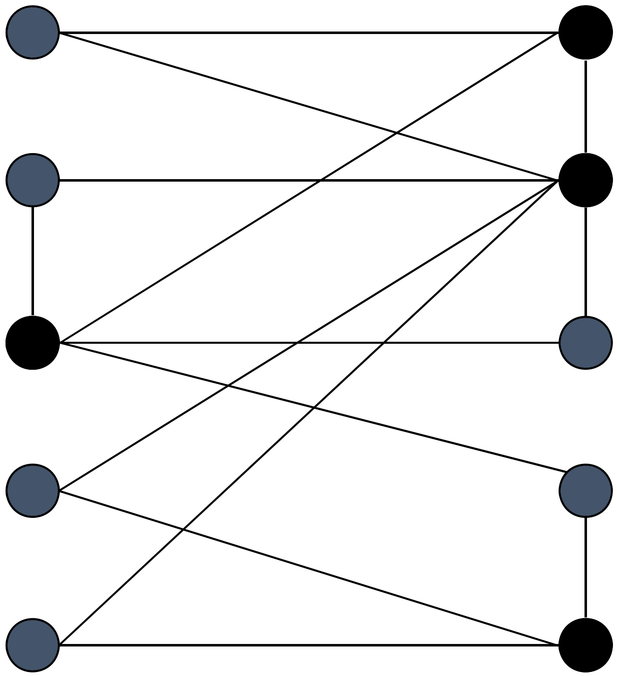
An **independent set** in an undirected graph G is a set of vertices that includes at most one endpoint of every edge.

$$INDEPENDENT-SET$$
$$= \{\langle G, k\rangle | G \text{ is an undirected graph containing an independent set with} \geq k \text{ vertices}\}$$

- Is there an independent set of size ≥ 6?
  - Yes.

independent set

- Is there an independent set of size ≥ 7?
  - No.



Independent Set is NP-complete

1) $INDEPENDENT-SET \in \mathrm{NP}$ [certificate = candidate ind. set of size k]
2) Reduce $3SAT \leq_{\mathrm{p}} INDEPENDENT-SET$

convert φ into a pair $\langle G,k \rangle$ s.t. φ satisfiable $\Longleftrightarrow$ G has an ind. set of size k

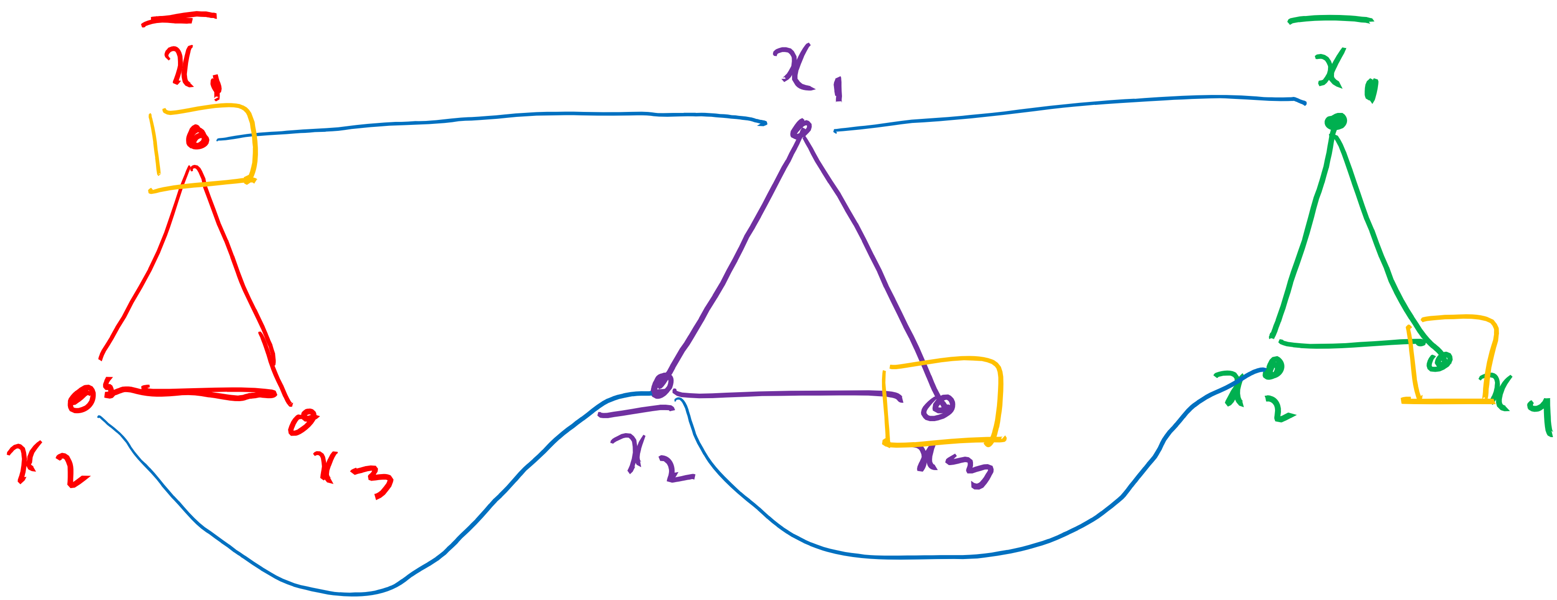
Proof. "On input $\langle \varphi \rangle$, where φ is a 3CNF formula,

1. Construct graph G from φ
   - G contains 3 vertices for each clause, one for each literal.
   - Connect 3 literals in a clause in a triangle.
   - Connect literal to each of its negations.
2. Output $\langle G, k \rangle$, where k is the number of clauses in φ."

Example of the reduction

$$\varphi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

$n = 3$
$= \#$ clauses



$x_1 = 0$
$x_2 =$ either 0 or 1
$x_3 = 1$
$x_4 = 1$

$\leftarrow$ this is a satisfying assignment to φ

Proof of correctness for reduction

$O(\log l)$ bits/clause $\Rightarrow$ total input length $= O(k \cdot \log l)$

l = # distinct literals

$\underbrace{(x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2 \vee \bar{x}_4 \vee x_5) \wedge \ldots}_{k}$

Let k = # clauses and l = # literals in φ

Claim: φ is satisfiable iff G has an ind. set of size k ← correctness

$\Longrightarrow$ Given a satisfying assignment, select one literal from each triangle. This is an ind. set of size k

$l \leq 3k$

$\Longleftarrow$ Let S be an ind. set of size k

- S must contain exactly one vertex in each triangle
- Set these literals to true, and set all other variables in an arbitrary way
- Truth assignment is consistent and all clauses satisfied

Runtime: $O(k + l^2)$ which is polynomial in input size

$\leq O(k^2)$

Space Complexity

Complexity measures we've studied so far

- Deterministic time TIME
- Nondeterministic time NTIME
- Classes P, NP

Resources: time, nondeterminism

E.g. $NTIME(f(n)) \subseteq TIME(2^{O(f(n))})$

Many other resources of interest:

Space (memory), randomness, parallel runtime / #processors, quantum entanglement, interaction, communication, …

Space analysis

Space complexity of a TM (algorithm) = maximum number of tape cells it uses on a worst-case input

Formally: Let $f : \mathbb{N} \to \mathbb{N}$. A TM M runs in space $f(n)$ if on every input $w \in \Sigma^{n}$, M halts on w using at most $f(n)$ cells

For nondeterministic machines: Let $f : \mathbb{N} \to \mathbb{N}$. An NTM N runs in space $f(n)$ if on every input $w \in \Sigma^{n}$, N halts on w using at most $f(n)$ cells on every computational branch

Space complexity classes

Let $f : \mathbb{N} \rightarrow \mathbb{N}$

A language $A \in \mathrm{SPACE}(f(n))$ if there exists a basic single-tape (deterministic) TM M that

1) Decides A, and
2) Runs in space $O(f(n))$

A language $A \in \mathrm{NSPACE}(f(n))$ if there exists a single-tape nondeterministic TM N that

1) Decides A, and
2) Runs in space $O(f(n))$

Example: Space complexity of SAT

Theorem: $SAT \in \mathrm{SPACE}(n)$

Proof: The following deterministic TM decides SAT using linear space

On input $\langle \varphi \rangle$ where φ is a Boolean formula:

1. For each truth assignment to the variables $x_1, \ldots, x_m$ of φ:
2. Evaluate φ on $x_1, \ldots, x_m$
3. If any evaluation $= 1$, accept. Else, reject.

Reuse the same space for evaluation

Example: NFA analysis

Theorem: Let $ALL_{NFA} = \{A \mid A \text{ is an NFA with } L(A) = \Sigma^*\}$

Then $\overline{ALL_{NFA}} \in \text{NSPACE}(n)$.

Proof: The following NTM decides $\overline{ALL_{NFA}}$ in linear space

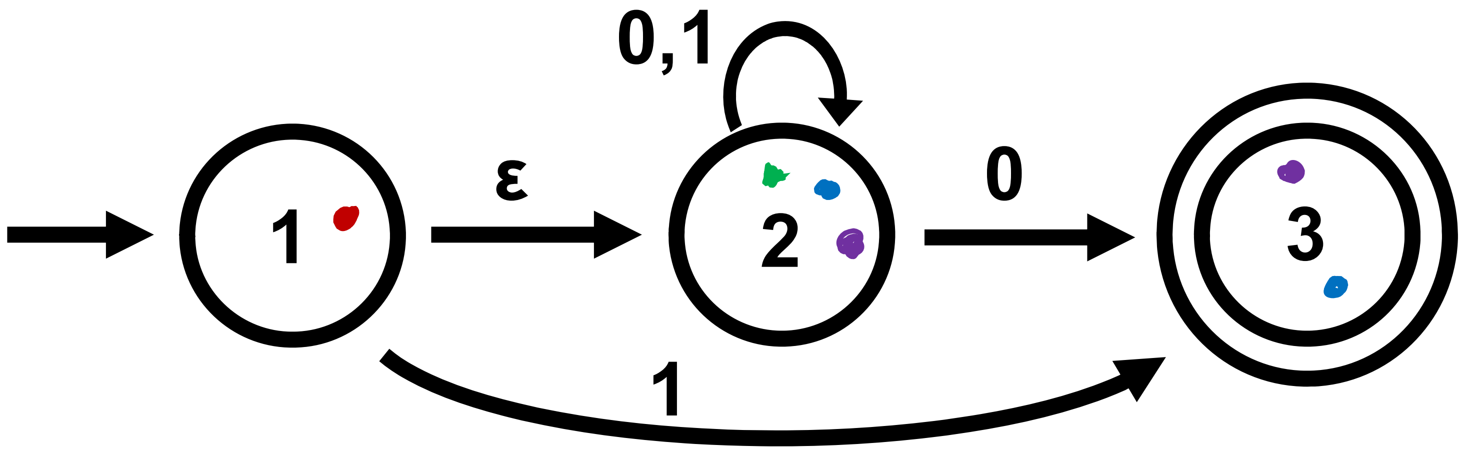
But exponential time!

On input $\langle A \rangle$ where A is an NTM:

NFA reject some string $\Leftrightarrow$ rejects some string of length $\leq 2^q$

1. Place a marker on the start state of A.
2. Repeat 2^q times where q is the # of states of A:
3. Nondeterministically select $a \in \Sigma$.
4. Adjust the markers to simulate all ways for A to read a.
5. Accept if at any point *none* of the markers are on an accept state. Else, reject.

Example



Step 0: Mark 1

Step 1: Simulate on 1   Current string: 1

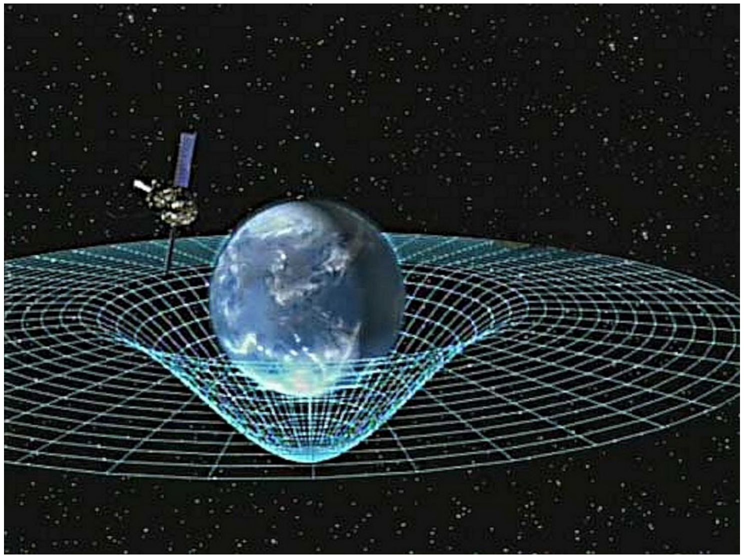
Step 2: Simulate on 0   Current: 10

Step 3: Simulate on 1   Current: 101

Conclude: $101 \notin L(A)$

$\Rightarrow A \in \overline{ALL_{NFA}}$

Space vs. Time



Space vs. Time

$$TIME(f(n)) \subseteq NTIME(f(n)) \subseteq SPACE(f(n))$$

How about the opposite direction? Can low-space algorithms be simulated by low-time algorithms?

Reminder: Configurations

A configuration is a string uqv where $q \in Q$ and $u, v \in \Gamma^*$

- Tape contents = uv (followed by blanks ⊔)
- Current state = q
- Tape head on first symbol of v

Example: $101q_5 0111$

Start configuration: $q_0 w$

Accepting configuration: $q = q_{\text{accept}}$

Rejecting configuration: $q = q_{\text{reject}}$

Reminder: Configurations

Consider a TM with

- k states
- tape alphabet $\{0, 1\}$
- space $f(n)$

$$\overbrace{0\ 1\ 0\ 1\ \boxed{q_7}\ 0\ 1\ 0}^{f(n)}$$

k choices

How many configurations are possible when this TM is run on an input $w \in \{0,1\}^n$?

- $2^{O(f(n))}$ possible strings representing tape content $\{3^{f(n)}\}$
- k states
- $f(n)$ locations for head

total $k \cdot f(n) \cdot 3^{f(n)}$

Observation: If a TM enters the same configuration twice when run on input w, it loops forever

Corollary: A TM running in space $f(n)$ also runs in time $2^{O(f(n))}$

$\mathrm{SPACE}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$

Savitch's Theorem

Savitch's Theorem: Deterministic vs. Nondeterministic Space

Theorem: Let f be a function with $f(n) \geq n$. Then $NSPACE(f(n)) \subseteq SPACE\left((f(n))^2\right)$.

$$PSPACE = \bigcup_{k=0}^{\infty} SPACE(n^k)$$

$$NPSPACE = \bigcup_{k=0}^{\infty} NSPACE(n^k)$$

$PSPACE = NPSPACE$

Corollary: $\overline{ALL_{NFA}} \subseteq PSPACE$