CS244: Theory of Computation

Fu Song
ShanghaiTech University

Fall 2020

Outline

Space Complexity

- Space complexity is another important consideration when we seek practical solutions to many computational problems.
- Space complexity shares many of the features of time complexity and serves as a further way of classifying problems according to their computational difficulty.

Definition

Let M be a DTM that halts on all inputs. The space complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of tape cells that M scans on any input of length n. If the space complexity of M is $f(n)$, we also say that M runs in space $f(n)$.

If M is an NTM wherein all branches halt on all inputs, we define its space complexity $f(n)$ to be the maximum number of tape cells that M scans on any branch of its computation for any input of length n.

Definition

Let $f : \mathbb{N} \to \mathbb{R}^+$ be a function. The space complexity classes,

$$\begin{aligned}\mathrm{SPACE}(f(n)) = \big\{L \;\big|\; & L \text{ is a language decided} \\ & \text{by an } O(f(n)) \text{ space DTM}\big\},\\ \mathrm{NSPACE}(f(n)) = \big\{L \;\big|\; & L \text{ is a language decided} \\ & \text{by an } O(f(n)) \text{ space NTM}\big\}.\end{aligned}$$

SAT $\in$ NSPACE(n)

M_1 on $\langle \varphi \rangle$, where φ is a Boolean formula:

```
1. For each truth assignment to the variables $x_1, \ldots, x_m$ of $\varphi$:
2.         Evaluate $\varphi$ on that truth assignment.
3. If $\varphi$ ever evaluated to 1, then accept; if not, then reject.
```

M_1 runs in linear space.

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

$$ALL_{\text{NFA}} = \{\langle A \rangle \mid A \text{ is an NFA and } L(A) = \Sigma^*\}.$$

N on $\langle M \rangle$, where M is an NFA:

1. Place a marker on the start state of the NFA.
2. Repeat 2^q times, where q is the number of states of M:
3. Nondeterministically select an input symbol and change the positions of the markers on M's states to simulate reading the symbol.
4. Accept if stages 2 and 3 reveal some string that M rejects, that is if at some point none of the markers lie on accept states of M. Otherwise, reject.

Why 2^q times? How many space is needed?

Outline

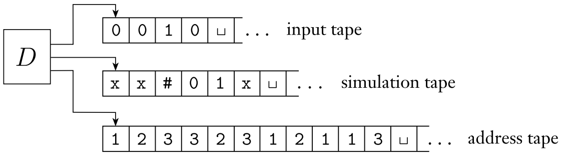
Savitch's Theorem

Theorem (Savitch, 1969)

For any function $f : \mathbb{N} \to \mathbb{R}^+$*, where* $f(n) \geq n$*,*

$$\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n)).$$

Recall NTM to DTM



D
0 0 1 0 ␣ . . . input tape
x x # 0 1 x ␣ . . . simulation tape
1 2 3 3 2 3 1 2 1 1 3 ␣ . . . address tape

1. Initially, tape 1 contains the input w, and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2.
3. Use tape 2 to simulate N with input w on one branch of its nondeterministic computation. Before each step of N, consult the next symbol on tape 3 to determine which choice to make among those allowed by N's transition function. If no more symbols remain on tape 3 or if this nondeterministic choice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, accept the input.
4. Replace the string on tape 3 with the next string in the string ordering. Simulate the next branch of Ns computation by going to stage 2. $2^{f(n)}$ space!

Proof (1)

CANYIELD on input c_1, c_2, and t:

```
1. If $t = 1$, then test whether $c_1 = c_2$ or whether $c_1$ yields $c_2$ in one
   step according to the rules of $N$. Accept if either test succeeds;
   reject if both fail.
2. If $t > 1$, then for each configuration $c_m$ of $N$ using space $f(n)$:
3.        Run CANYIELD$(c_1, c_m, t/2)$.
4.        Run CANYIELD$(c_m, c_2, t/2)$.
5.        If steps 3 and 4 both accept, then accept.
6. If haven't yet accepted, then reject.
```

Proof (2)

- Modify N so that when it accepts, it clears its tape and moves the head to the leftmost cell – thereby entering a configuration c_{accept}.
- Let c_{start} be the start configuration of N on w.
- We select a constant d so that N has no more than $2^{d \cdot f(n)}$ configurations using $f(n)$ tape, where $n = |w|$.

Proof (3)

CANYIELD on input c_1, c_2, and t:

```
1. If $t = 1$, then test whether $c_1 = c_2$ or whether $c_1$ yields $c_2$ in one
   step according to the rules of $N$. Accept if either test succeeds;
   reject if both fail.
2. If $t > 1$, then for each configuration $c_m$ of $N$ using space $f(n)$:
3.        Run CANYIELD$(c_1, c_m, t/2)$.
4.        Run CANYIELD$(c_m, c_2, t/2)$.
5.        If steps 3 and 4 both accept, then accept.
6. If haven't yet accepted, then reject.
```

M on input w:

```
1. Output the result of CANYIELD $\left(c_{\text{start}}, c_{\text{accept}}, 2^{d \cdot f(n)}\right)$.
```

M uses space

$$O\left(\log 2^{d \cdot f(n)} \cdot f(n)\right) = O(f^2(n)).$$

We do not know d. Where do we get $f(n)$?

$$M \text{ tries } f(n) = 1, 2, 3, \ldots.$$

Outline

The Class PSPACE

Definition

PSPACE is the class of languages that are decidable in polynomial space on a deterministic Turing machine. In other words,

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}\left(n^k\right).$$

We could also define

$$\mathrm{NPSPACE} = \bigcup_k \mathrm{NSPACE}\left(n^k\right).$$

Then by Savitch's Theorem, $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$:

$$\mathrm{NSPACE} = \mathrm{PSPACE}.$$

The relationship of PSPACE with **P** and **NP**

A machine which runs in time t can use at most space t.

Hence

$$\mathbf{P} \subseteq \mathrm{PSPACE} \quad \text{and} \quad \mathbf{NP} \subseteq \mathrm{NPSPACE} = \mathrm{PSPACE}.$$

PSPACE and EXPTIME

Let $f : \mathbb{N} \to \mathbb{R}^+$ satisfy $f(n) \geq n$. Then a TM uses $f(n)$ space can have at most

$$f(n) \cdot 2^{O(f(n))}$$

configurations. Therefore it must run in time $f(n) \cdot 2^{O(f(n))}$.

Hence

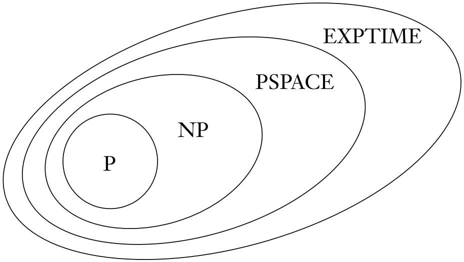
$$\mathrm{PSPACE} \subseteq \mathrm{EXPTIME} = \bigcup_k \mathrm{TIME}\left(2^{n^k}\right).$$

We know

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME},$$

and it is easy to show $\mathbf{P} \neq \mathrm{EXPTIME}$.

The general consensus is



$\mathbf{P} \subsetneq \mathbf{NP}$? $\mathbf{NP} \subsetneq \mathrm{PSPACE}$? $\mathrm{PSPACE} \subsetneq \mathrm{EXPTIME}$? YES

Outline

PSPACE Completeness

Definition

A language B is PSPACE-complete if

1. B is in PSPACE, and
2. every $A \in$ PSPACE is polynomial time reducible (not polynomial space reducible) to B.

If B merely satisfies condition 2, then it is PSPACE-hard.

Why not polynomial space reducibility?

Let $A, B \subseteq \Sigma^*$. Then A is polynomial space reducible to B, if a polynomial space computable function $f : \Sigma^* \to \Sigma^*$ exists, where for every w

$$w \in A \iff f(w) \in B.$$

Remark

1. *Let B be a language with $\emptyset \neq B \neq \Sigma^*$. Then every $A \in$ PSPACE is polynomial space reducible to B.*
2. *Let $B \in \mathbf{P}$ and A be polynomial space reducible to B. It is not known that $A \in \mathbf{P}$.*
3. *Let $B \in \mathbf{P}$ and A be polynomial time reducible to B. It is known that $A \in \mathbf{P}$.*

The TQBF problem

- A Boolean formula contains Boolean variables, the constants 0 and 1, and the Boolean operations $\wedge$, $\vee$, and $\neg$.
- The universal quantifiers $\forall$ in $\forall x \varphi$ means that φ is true for every value of x in the universe.
- The existential quantifiers $\exists$ in $\exists x \varphi$ means that φ is true for some value of x in the universe.
- Boolean formulas with quantifiers are quantified Boolean formulas. For such formulas, the universe is $\{0, 1\}$. E.g.

$$\forall x \exists y \big[(x \vee y) \wedge (\bar{x} \vee \bar{y})\big].$$

- When each variable of a formula appears within the scope of some quantifier, the formula is fully quantified. A fully quantified Boolean formula is always either true or false.

The TQBF problem

The *TQBF* problem is to determine whether a fully quantified Boolean formula is true or false, i.e.,

$$\mathit{TQBF} = \{\langle\varphi\rangle \mid \varphi \text{ is a true fully quantifier Boolean formula}\}.$$

Theorem

TQBF is PSPACE*-complete.*

Recalling that SAT is also in PSPACE, but **NP**-complete.

1. To show that *TQBF* is in PSPACE, we give a straightforward algorithm that assigns values to the variables and recursively evaluates the truth of the formula for those values.

2. To show that *TQBF* is PSPACE-hard, i.e., every language A in PSPACE reduces to *TQBF* in polynomial time, we give a polynomial time reduction that reduce a polynomial space-bounded Turing machine of A to a quantified Boolean formula that encodes a simulation of the machine on that input. The formula is true iff the machine accepts.

Proof (1)

TQBF is in PSPACE

T on $\langle\varphi\rangle$, a fully quantifier Boolean formula:

1. If φ contains no quantifiers, then it contains only constant, so evaluate φ and accept if it is true; otherwise reject.
2. If $\varphi = \exists x \psi$, recursively call T on ψ first with $x := 0$ and second with $x := 1$. If either result is accept, then accept; otherwise reject.
3. If $\varphi = \forall x \psi$, recursively call T on ψ first with $x := 0$ and second with $x := 1$. If both results are accept, then accept; otherwise reject.

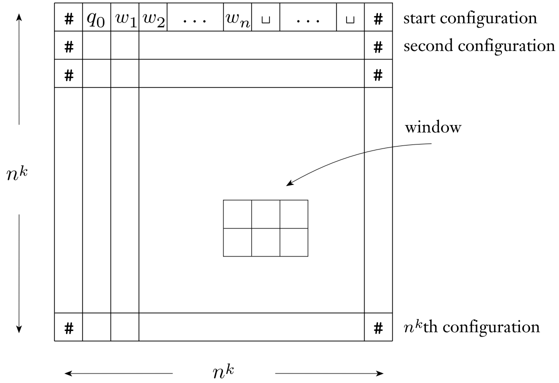
T runs in polynomial space for storing each assignment of variables.

How much time?

Recall Cook-Levin Theorem: $A \leq_P$ SAT

Let N be an NTM that decides a language A in time n^k for some $k \in \mathbb{N}$. We show $A \leq_P$ SAT.

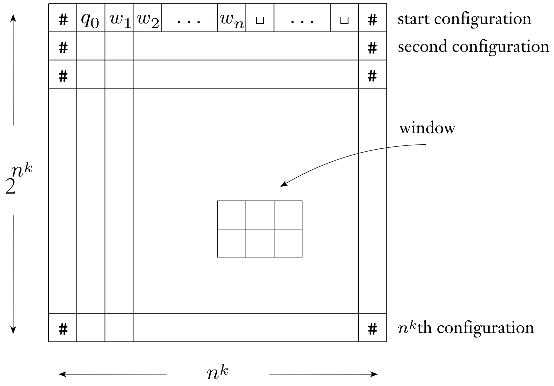
A tableau for N on w is an $n^k \times n^k$ table whose rows are the configurations of the branch of the computation of N on input w.



$A \leq_P TQBF$

Let N be an TM that decides a language A in space n^k for some $k \in \mathbb{N}$. We show $A \leq_P TQBF$.

A tableau for N on w is an $2^{n^k} \times n^k$ table whose rows are the configurations of the branch of the computation of N on input w.



- A polynomial time reduction cannot produce an exponential-size result, so this attempt fails to show that $A \leq_P TQBF$
- Savitch's theorem: divides the tableau into halves and employs the universal quantifier to represent each half with the same part of the formula.

Proof (2)

Let A be a language decided by a TM M in space n^k. We need to show $A \leq_{\mathrm{P}} TQBF$.

For an input w, maps w into a $TQBF$ formula ϕ such that M accepts w iff ϕ is true.

Using two collections of variables c_1, c_2 and $t > 0$, we define a formula $\varphi_{c_1,c_2,t}$. If we assign c_1 and c_2 to actual configurations, the formula is true if and only if M can go from c_1 to c_2 in at most t steps.

Then we can let φ be the formula

$$\varphi_{c_{\mathrm{start}},c_{\mathrm{accept}},h},$$

where $h = 2^{d \cdot n^k}$, where d is chosen so that M has no more than $2^{d \cdot n^k}$ configurations on an input of length n.

Proof (3)

The formula encodes the contents of configuration cells as in the proof of the Cook-Levin theorem.

- Each cell has several variables associated with it, one for each tape symbol and state, corresponding to the possible settings of that cell.
- Each configuration has n^k cells and so is encoded by $O(n^k)$ variables.

Proof (4)

How to construct $\varphi_{c_1,c_2,t}$?

Let $t = 1$. We define $\varphi_{c_1,c_2,1}$ to say that either $c_1 = c_2$, or c_2 follows from c_1 in a single step of M.

- We express $c_1 = c_2$ by saying that each of the variables representing c_1 contains the same Boolean value as the corresponding variables representing c_2.
- We express the second possibility (c_1 yields c_2) by using the technique presented in the proof of the Cook-Levin theorem, window-by-window

Proof (5)

How to construct $\varphi_{c_1,c_2,t}$?

Let $t > 1$. Our first try is to define

$$\varphi_{c_1,c_2,t} = \exists m_1 \left[\varphi_{c_1,m_1,t/2} \wedge \varphi_{m_1,c_2,t/2} \right],$$

where m_1 represents a configuration of M.

- $\varphi_{c_1,c_2,t}$ is true iff M can go from c_1 to c_2 within t steps iff exists a configuration m_1 (using $O(n^k)$ variables) such that M go from c_1 to m_1 within $\frac{t}{2}$ steps and M go from m_1 to m_2 within $\frac{t}{2}$ steps.
- $\varphi_{c_1,m_1,t/2}$ and $\varphi_{m_1,c_2,t/2}$ can be constructed recursively, each recursion t cuts half.
- However, each recursion doubles the size of $\varphi_{c_1,c_2,t}$ and t could be $2^{d \cdot n^k}$, exponential in n, hence the resulting formula ϕ is still exponential in n

Proof (6)

Instead we define

$$\varphi_{c_1,c_2,t} = \exists m_1 \forall (c_3, c_4) \in \big\{(c_1, m_1), (m_1, c_2)\big\} \big[\varphi_{c_3,c_4,t/2}\big].$$

where $\forall (c_3, c_4) \in \big\{(c_1, m_1), (m_1, c_2)\big\}$ means that either $c_3 = c_1 \wedge c_4 = m_1$ or $c_3 = m_1 \wedge c_4 = c_2$.

$$\begin{aligned} \varphi_{c_1,c_2,t} &\equiv \exists m_1 \forall (c_3, c_4) \in \big\{(c_1, m_1), (m_1, c_2)\big\} \big[\varphi_{c_3,c_4,t/2}\big] \\ &\equiv \exists m_1 \big[\varphi_{c_1,m_1,t/2} \wedge \varphi_{m_1,c_2,t/2}\big] \end{aligned}$$

The size of $\varphi_{c_1,c_2,t}$ is $\log t$ and $t = 2^{d \cdot n^k}$.
Therefore, the size of $\varphi_{c_{\text{start}},c_{\text{accept}},2^{d\cdot n^k}}$ is bounded by

$$\log 2^{d \cdot n^k} = n^{O(k)}.$$

Winning strategies for games

- A game is loosely defined to be a competition in which opposing parties attempt to achieve some goal according to pre-specified rules
- Games appear in many forms, from board games such as chess to economic and war games that model corporate or societal conflict
- Games are closely related to quantifiers. A quantified statement has a corresponding game; conversely, a game often has a corresponding quantified statement.
- These correspondences are helpful in several ways
  - expressing a mathematical statement that uses many quantifiers in terms of the corresponding game may give insight into the statement's meaning
  - expressing a game in terms of a quantified statement aids in understanding the complexity of the game
- To illustrate the correspondence between games and quantifiers, we turn to an artificial game called the formula game

The formula game

Let

$$\varphi = \exists x_1 \forall x_2 \exists x_3 \cdots Q x_k \left[\psi \right].$$

We associate a game with φ.

1. Two players, Player A and Player E, take turns selecting the values of the variables $x_1, \ldots, x_k$.
2. Player A selects values for the variables bound to $\forall$.
3. Player E selects values for the variables bounded to $\exists$.
4. At the end, if ψ is true, then Player E wins; otherwise Player A wins.

Definition

Given a game ϕ, Player E has a winning strategy for the game ϕ if Player E has a strategy for the game such that whatever Player A chooses, Player E always wins.
Otherwise, we say Player A has a winning strategy.

Example

Player E has a winning strategy for

$$\exists x_1 \forall x_2 \exists x_3 \left[(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})\right].$$

Example

Player A has a winning strategy for

$$\exists x_1 \forall x_2 \exists x_3 \left[(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3})\right].$$

Let

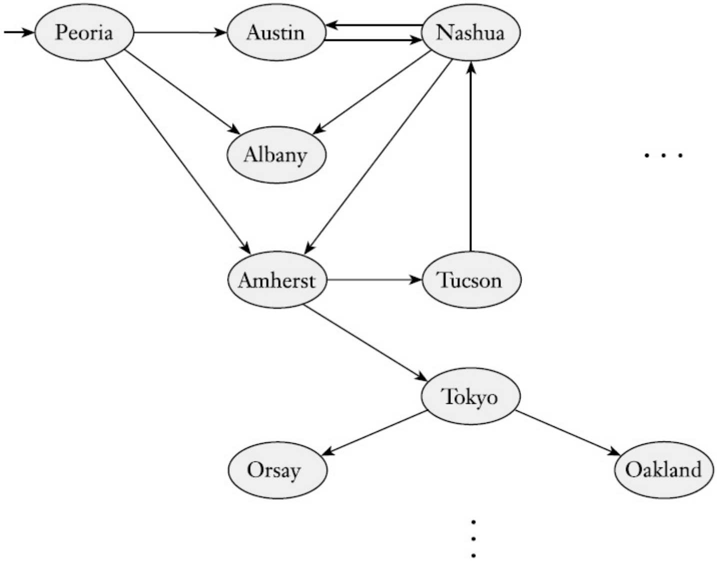
$$\textit{FORMULA-GAME} := \{\langle\varphi\rangle \mid \text{Player E has a winning strategy the formula game associated with } \varphi\}.$$

Theorem

FORMULA-GAME is PSPACE*-complete.*

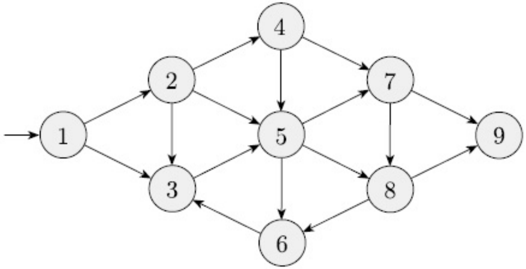
Generalized geography

1. Two players take turns to name cities from anywhere in the world.
2. Each city chosen must begin with the same letter that ended the previous city's name.
3. Repetition isn't permitted.
4. The game starts with some designated starting city and ends when some player can't continue and thus loses the game.


Peoria
Austin
Nashua
Albany
Amherst
Tucson
Tokyo
Orsay
Oakland

Generalized Geography

1. Take an arbitrary directed graph with a designated start node.



2. Player I starts by selecting the designated start node.
3. Then the players take turns picking nodes that form a simple path in the graph.
4. The first player unable to extend the path loses the game.

Let

$$GG := \Big\{\langle G, b\rangle \;\Big|\; \text{Player I has a winning strategy for the geography game played on graph } G \text{ starting at } b\Big\}.$$

Theorem

GG is PSPACE*-complete.*

Proof (1)

GG is in PSPACE

M on $\langle G, b \rangle$:

1. If b has outdegree 0, then reject and Player I loses the game.
2. Remove node b and all connected arrows to get a new graph G'.
3. For each of the nodes $b_1, b_2, \ldots, b_k$ that b originally pointed at, recursively call M on $\langle G', b_i \rangle$.
4. If all of these accept, then Player II has a winning strategy in the original game, so reject and Player I loses the game. Otherwise, accept and Player I wins the game.

M runs in linear space for storing nodes in the recursive stack.

Proof (2)

To show the hardness, we give a reduction from *FORMULA-GAME*.

Let

$$\varphi = \exists x_1 \forall x_2 \exists x_3 \cdots Q x_k \left[\psi\right],$$

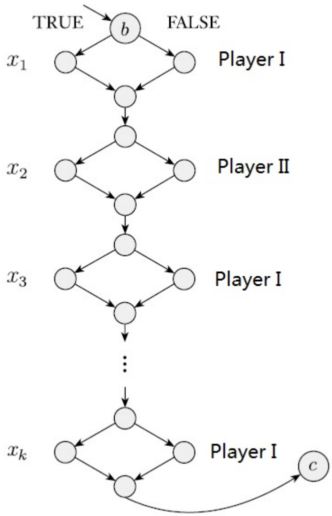
where

- the quantifiers begin and end with $\exists$, and alternate between $\exists$ and $\forall$,
- ψ is in conjunctive normal form
- Any *FORMULA-GAME* formula that does not follow this assumption can be modified

We constructs a geography game on a graph G where optimal play mimics optimal play of the formula game on φ, in which

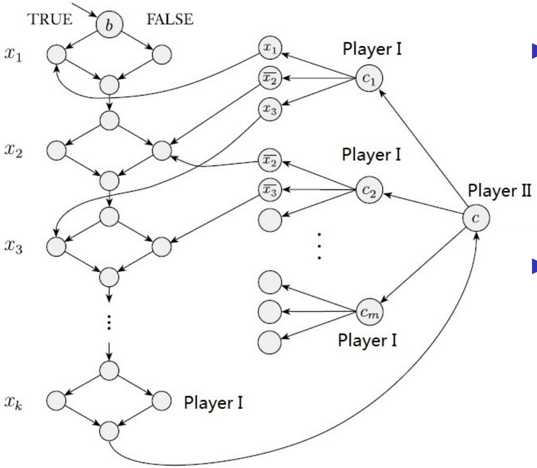
- Player I in the geography game takes the role of Player E in the formula game, and
- Player II takes the role of Player A.

Proof (3)



- For each variable x_i, create a diamond structural subgraph
- Left child denotes true
- Right child denotes false
- *GG* starts with b controlled by Player I
- *GG* ends with c controlled by Player II

Proof (4)



- From c, Player II can choose a clause c_i which asks Player I to show that the clause c_i is true
- From c_i, Player I has to choose one literal ℓ of c_i to show that c_i is true by showing ℓ is true

$$\exists x_1 \forall x_2 \cdots \exists x_k \left[(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee \cdots) \wedge \cdots \right].$$

Outline

$$\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}\left(n^k\right) \text{ and } \text{NPSPACE} = \bigcup_{k \in \mathbb{N}} \text{NSPACE}\left(n^k\right)$$

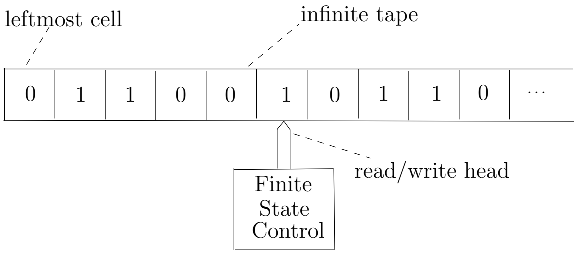
$$\mathbf{P} = \bigcup_{k \in \mathbb{N}} \text{TIME}\left(n^k\right) \text{ and } \mathbf{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}\left(n^k\right).$$

Can we define logarithmic space or time complexity?

$$\text{LSPACE} = \text{SPACE}(\log n) \text{ and } \text{NLTIME} = \text{NSPACE}(\log k).$$

or

$$\text{LTIME} = \text{TIME}(\log n) \text{ and } \text{NLTIME} = \text{NTIME}(\log k).$$



It does not make any sense to discuss time or space complexity that is less than linear on standard single tape TM

- The machine has to read entire input
- The input has to be stored for reading
- We consider logarithmic space complexity on two tapes TM model:
  - one read-only tape and
  - one work tape (read and write) on which the space complexity is measured

L and **NL**

Definition

L is the class of languages that are decidable in logarithmic space on a deterministic (two-tape) Turing machine. In other words,

$$\mathbf{L} = \mathrm{SPACE}(\log n).$$

Definition

NL is the class of languages that are decidable in logarithmic space on a nondeterministic (two-tape) Turing machine. In other words,

$$\mathbf{NL} = \mathrm{NSPACE}(\log n).$$

Recalling $A = \{0^k1^k \mid k \geq 0\}$

Theorem

A is in **P**.

M_1 on w:

1. Scan across tape 1 and reject if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape.
3. Scan across tape 1, crossing off a single 0 and a single 1 but in tape 2.
4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, reject.

   Otherwise if neither 0s nor 1s remain on the tape 1, accept.

How many space used? $O(n)$

Theorem

A is in **L**.

```
1. Tape 1 store the input and initial tape 2 with 0
2. Scan across tape 1 and reject if a 0 is found to the right of a 1
3. Repeat scan the tape 1 from left to right until the last 0
4.     increase the value on tape 2 by 1 using binary encoding
5. Repeat scan the tape 1 from the first 1 until the last 1
6.     decrease the value on tape 2 by 1
7. Test the value in the tape 2. If it is 0, then accept, otherwise reject.
```

For the input with length $2n$, $O(\log n)$ space is used!

Recalling the path problem

$$PATH = \Big\{ \langle G, s, t \rangle \;\Big|\; G \text{ is a directed graph that has a directed path from } s \text{ and } t \Big\}.$$

Theorem
$PATH \in \mathbf{P}$.

M on input $\langle G, s, t \rangle$: (breadth-first search)

```
1. Place a mark on node $s$.
2. Repeat the following until no additional nodes are marked:
3.    Scan all the edges of $G$. If an edge $(a, b)$ is found going from a
      marked node a to an unmarked node $b$, mark node $b$.
4. If $t$ is marked, accept. Otherwise, reject
```

For the input with length n, $O(n)$ space is used!

$PATH = \{\langle G, s, t\rangle \mid G$ is a directed graph that has a directed path from s and $t\}$.

Theorem

$PATH \in \mathbf{NL}$.

M on input $\langle G, s, t\rangle$: (breadth-first search)

```
1. Place the start node $s$ on the work tape (i.e., tape 2).
2. Repeat the following at most $m$ steps until $t$ is reached, $m$ is the
   number of nodes:
3.     Nondeterministically chooses a next node $n$ of the current node $c$
4.     Replace the current node $c$ on tape 2 by the node $n$
5. If $t$ is reached, accept. Otherwise, reject
```

Only use $O(\log m)$ (binary encoding) space for storing the number of steps and the current node in tape 2

Time bounds from Space complexity

- For $f(n)$ space single tape TM,
  - there are at most $2^{O(f(n))}$ configurations,
  - then the time complexity is bounded by $2^{O(f(n))}$
- For $f(n)$ space two tapes TM (one read-only tape for input and one work tape):
  - there are at most $n2^{O(f(n))}$ configurations, as the input tape does not change, we only need to record the heap position
  - then the time complexity is bounded by $n2^{O(f(n))}$

Relations between Classes **L**, **NL** and **P**

Theorem

$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P}$

1. It is easy to follow the relation $\mathbf{L} \subseteq \mathbf{NL}$
2. For $\mathbf{NL} \subseteq \mathbf{P}$: $O(\log n)$ space implies that there are at most $n2^{O(\log n)}$ configurations, then in **P** time

It is open whether $\mathbf{L} \subsetneq \mathbf{NL}$ or $\mathbf{L} = \mathbf{NL}$, and $\mathbf{NL} \subsetneq \mathbf{P}$ or $\mathbf{NL} = \mathbf{P}$.

Quiz

EVEN $= \{x \mid x$ has even number of 1's$\}$

Is EVEN in **L**?

Outline

NL-Complete

Definition

A log space transducer is a Turing machine with

- a read-only input tape, a write-only output tape, and a read/write work tape
- head on the output tape cannot move leftward, so it cannot read what it has written
- work tape may contain $O(\log n)$ symbols

A log space transducer M computes a function $f : \Sigma^* \to \Sigma^*$, where $f(w)$ is the string remaining on the output tape after M halts when it is started with w on its input tape. We call f a log space computable function.

Language A is log space reducible to language B, written $A \leq_{\mathsf{L}} B$, if A is mapping reducible to B by means of a log space computable function f.

Definition

A language B is **NL**-complete if

1. B is in **NL**, and
2. every $A \in$ **NL** is log space reducible to B.

NL-Complete

Definition
A language B is **NL**-complete if

1. B is in **NL**, and
2. every $A \in$ **NL** is log space reducible to B.

Definition
A language B is PSPACE-complete if

1. B is in PSPACE, and
2. every $A \in$ PSPACE is polynomial time reducible to B.

- **NL** $\subseteq$ **P**, all problems in **NL** are solvable in polynomial time
- every two problems in **NL**(except $\emptyset$ and Σ^*) are polynomial time reducible to one another
- hence, polynomial time reducibility is too strong to differentiate problems in **NL** from one another

Theorem

If $A \leq_{\mathrm{L}} B$ and $B \in \mathbf{L}$, then $A \in \mathbf{L}$.

- A log space reduction f for A first maps its input w to $f(w)$
- then applies the log space algorithm for B
- However, the storage required for $f(w)$ may be too large to fit within the log space bound

Instead

1. A's machine M_A computes individual symbols of $f(w)$ as requested by B's machine M_B
2. In the simulation, M_A keeps track of where M_B's input head would be on $f(w)$
3. Every time M_B moves, M_A restarts the computation of f on w from the beginning and ignores all the output except for the desired location of $f(w)$.

Corollary

*If any **NL**-complete language is in **L**, then $\mathbf{L} = \mathbf{NL}$.*

Recalling the path problem

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ and } t\}.$$

Theorem

PATH is **NL***-complete.*

Idea:

- We have already shown that *PATH* is in **NL**.
- For **NL**-hardness, we present a log space reduction from any language $A \in \mathbf{NL}$ to *PATH*
  - Assume M_A has a unique accept state q_{accept} and empties the work tape before entering the accepting state
  - Each configuration is denoted by a node
  - Configuration c yields c' if there is an edge between c and c'
- M_A accepts w iff there is a path from $s =$start configuration to $t =$accepting configuration

$A \in \mathbf{NL}$ to *PATH*

Transducer TM M_f on input w:

1. M_f lists all the legal configurations that are nodes in output tape (at most $2^{O(\log n)}$, as $A \in \mathbf{NL}$)
2. M_f lists all the edges in output tape by testing pairs of configurations
3. M_f outputs the start and accepting configurations

Outline

NL=co**NL**

Definition

A language A is in co**NL** if the complement $\bar{A}$ of A is in **L**.

Theorem

NL $=$ *co***NL**.

Note: It is known that **P** $=$ *co***P**, **L** $=$ *co***L** and PSPACE $=$ NPSPACE $=$ *co*PSPACE $=$ *co*NPSPACE.

It is unknown that **NP** $=$ *co***NP**?

NL=coNL

1. *PATH* is **NL**-complete, then $\overline{PATH}$ is *co***NL**-complete
2. We show that $\overline{PATH}$ is in **NL**. Suppose m denotes the number of nodes in G.
   - 2.1 First, compute the number c of reachable nodes from s
   - 2.2 Then, nondeterministically guess and check each node (except t) whether it is reachable from s in at most m steps, if it is reachable, increase counter d;
   - 2.3 Finally check whether $d = c$ or not. If $d \neq c$, then t is not reachable (i.e., accept), otherwise t is reachable (i.e., reject).

How to compute the number c of reachable nodes from s in log space?

How to compute the number c of reachable nodes from s in log space?

1. Compute sequence of sets of nodes $A_0, A_1, \cdots, A_m$
2. A_i is the set of nodes that are reachable from s using at most i steps
3. $A_0 = \{s\}$
4. A_{i+1} is computed based on A_i
5. Finally $c = |A_m|$

However we cannot explicitly store the sets A_i's,
which may require m space!

Proof (1)

M_1 on input $\langle G, s, t\rangle$: (m denotes the number of nodes in G)

```
1.  $c_0 = 1$                                                        [$c_0 = |A_0|$]
2.  For each $i = 0$ to $m - 1$
3.      $c_{i+1} = 1$                                                [$c_i = |A_i|$]
4.      For each node $v \neq s$ in $G$:
5.          $d = 0$                                        [Count is a log space]
6.          For each node $u \in G$, nondeterministically either perform
7.          or skip these steps:
8.              Nondeterministically follow a path of length at most $i$
9.              from $s$ and reject if it doesn't end at $u$.
10.             ++d                                                  [$u \in A_i$]
11.             If $(u, v)$ is an edge of $G$                        [$v \in A_{i+1}$]
12.                 $++c_{i+1}$ and go to stage 5 with the next $v$
13.     If $d \neq c_i$, then reject        [Check found whether all of $A_{i+1}$]
14. Output $c_m$
```

Proof (2)

M_1 on input $\langle G, s, t\rangle$: (m denotes the number of nodes in G)

```
1. $c_m$ as computed above and $d = 0$
2. For each node $u$ in $G$ nondeterministically either perform
3. or skip these steps:
4.         Nondeterministically follow a path of length at most $m$
5.         from $s$ and reject if it doesn't end at $u$.
6.         If $t = u$, then reject                                  $[t \in A_m]$
7.         ++d                                                      $[u \in A_m]$
8. If $d \neq c_m$, then reject,                                    $[t \in A_m]$
9. Otherwise accept
```

Theorem

$\mathbf{L} \subseteq \mathbf{NL} = co\mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathrm{PSPACE}$ *and* $\mathbf{NL} \subsetneq \mathrm{PSPACE}$*.*

- We will prove that $\mathbf{NL} \subsetneq \mathrm{PSPACE}$ later
- Either $\mathbf{NL} \subset \mathbf{P}$ or $\mathbf{P} \subset \mathrm{PSPACE}$, but we do not know which one does or both hold