SPACE COMPLEXITY

Measuring space

- The ***time complexity*** of an algorithm counts the *number* of steps (i.e. configurations) in the trace.
- The ***space complexity*** of an algorithm counts the *maximal size* of configurations in the trace.
- Space $O(f)$ (where $f : \mathbb{N} \rightarrow \mathbb{N}$) defined like for time.

Examples

- Symbolic multiplication is in quadratic space.
- **BOOL-SAT** is in linear space:

  Given a boolean expression E, the Turing acceptor

  lists its variables,

  cycles through all valuations for them,

  and accepts if some valuation satisfies E.
- So already linear space captures an NP-complete problem,

  which is probably not recognized in time $O(n^k)$ for any k.

Polynomial space

- **Space**(f) stands for the collection of languages recognized by a Turing machine in space $O(f)$.
- Polynomial Space = space $O(n^k)$ for some k:

  $\cup_k O(n^k)$.

- (PSpace) is the class of problems decidable by a Turing machine in space $O(n^k)$ for some k .

The Time Hierarchy Theorem (reminder)

- **Time Hierarchy Theorem.**

  If $t, T : \mathbb{N} \to \mathbb{N}$ *are "reasonable" functions,*

  and $t \cdot \log t = o(T)$*, then then there are problems decidable in* **Time**(T) *but not in* **Time**(t).

The Space Hierarchy Theorem

- ***Space Hierarchy Theorem.***

  If $f : \mathbb{N} \to \mathbb{N}$ is a “reasonable” function,

  then there are problems in **Space**$(f \log(f))$

  that are not in **Space**(f).

- Simpler than the Time Hierarchy Theorem

  because here the proof need not use asymptotic growth

  for extra resources to build a universal interpreter for the class.

- Example: **Space**$(n \log n) \supsetneq$ **Space**(n).

Relations between time and space complexity

- In time $f(n)$ we cannot use more than $f(n)$ symbols,

  so **Time**$(f) \subseteq$ **Space**(f).

- Such inclusion is less evident for non-deterministic time:

  NTime$(f) \subseteq$ **Space**(f) .

- Proof: For a given input w of length n

  we can scan in space $f(n)$ a computation tree of height $f(n)$.

- The simulating acceptor has

  one dedicated string of length $f(n)$

  for the address of the currently inspected cfg,

  and another containing the cfg itself.

- In particular, **NPTime** $\subseteq$ **PSpace**.

Exponential blowup from space to time

- With k states and a symbols there are $k \cdot a^n$ different cfgs of size $n+1$.
- So an acceptor M over alphabet of size a running in space $f(n)$ may go through $O(f(n))$ different cfgs.
- If a cfg repeats on the trace for input w then M would run indefinitely and will never accept w.

  So we may limit searches for an accepting cfg to traces with no repetitions.
- So $\text{Space}(f(n)) \subseteq \cup_a \text{Time}(a^{f(n)})$.
- Since $a^{f(n)} = 2^{\ell \cdot f(n)}$ for $\ell = \log a$,

  $\text{Space}(O(f(n))) \subseteq \text{Time}(2^{O(f(n))})$.
- This motivates attention to a larger time-complexity class:

  Acceptor M is in ***exponential time*** if

M terminates in fewer than a^{n^k} steps (some a, k).

- So we have **PSpace** $\subseteq$ **ExpTime**

A broad space/time hierarchy

- **PTime $\subseteq$ NPTime $\subseteq$ PSpace $\subseteq$ ExpTime**
- By the Time Hierarchy theorem **PTime $\subsetneq$ ExpTime**
- So at least one containment in the chain above is strict.
- Most people believe they all are strict, but so far this has not been proved for any of the three.

PSpace and non-determinism

- What about non-deterministic **PSpace** (notation: **NPSpace**)?
- In fact **NPSpace** $\subseteq$ **PSpace**.
- Our proof of **NTime**$(f) \subseteq$ **Space**(f) does not work here because a nondeterministic computation in space $f(n)$

  is a tree whose branches, i.e. it different computation-traces,

  might go through $2^{f(n)}$ different cfgs.

Savitch's Theorem

- **Theorem**: If $f(n) \geqslant n$ then NSpace(f(n)) $\subseteq$ Space$(f(n)^2)$.

- For given acceptor M ,

  consider the more general property $Lead(c, c', t)$:

  M *has a trace of length* $\leqslant t$ *from cfg* c *to* c' .

- An algorithm can recognize *Lead*(c, c', t) by searching for an intermediate cfg m

  such that $Lead(c, m, t/2)$ and $Lead(m, c', t/2)$.

- This yields a recursive algorithm where the recursive stack

  has depth $O(\log(a^{2^{f(n)}} = O(f(n))$.

- The size of each cfg is $O(f(n)$,

  so the total size of the stack is $O(f(n)^2)$.

- In particular, **NPSpace** = **PSpace**.

PSpace is about alternating roles

- The **Geography** Game.
- Take turns proposing city names, subject to:
  - No city repeating
  - Each name starts with the previous' last letter.

Bloomington $\Rightarrow$ *Nashville*
$\Rightarrow$ *Erie*
$\Rightarrow$ *El Paso*
$\Rightarrow$ *Omaha*
$\Rightarrow$ *Amherst*
$\Rightarrow$ *Trenton*
$\Rightarrow$ *New York*
$\Rightarrow$ *Knoxville*

The essence of a two-player game

- The computational core of a game is the switch between players.
- Generic players: *Alice* and *Bob*.

Alice wins if she has a first move so that

for every first move of Bob

Alice has a second move so that

for every second move of Bob

Alice has a third move so that

for every third move of Bob

Alice has a fourth move so that

for every fourth move of Bob

. . . Alice has a 17,019'th move that wins the game.

Quantified boolean expressions

- Such role-changes are captured by **quantified boolean expressions (QBE's).**
- These are generated from 0, 1, variables, negation, conjunction, disjunction and:
  - If E is a QBE, x a variable,
    then $\forall x\, E$ and $\exists x\, E$ are QBE's.
- $\forall x\, E[x]$ is true under valuation V
  if both $E[0]$ and $E[1]$ are true under V.
- A QBE is **prenex** if it is of the form $Q_1 x_1 \cdots ; Q_k x_k\, F$
  where each Q_i is $\exists$ or $\forall$ and
  E is a boolean expression (no quantifiers).
- Example: The prenex expression
  $\forall x \forall y \forall z\, ((x = y) \vee (x = z) \vee (y = z))$ is true,
  where $x = y)$ abbreviates $(x \vee \bar{y}) \wedge (\bar{x} \vee y)$.

- Every QBE can be converted into an equivalent ***prenex*** QBE of no larger size.

The problem QSAT

- A generalization of BOOL-SAT.
- **Q-SAT** : *Given a QBE, is it satisfiable*
- Equivalently:

  Given a QBE without unbounded variables, is it true

- This is an equivalent formulation:

  A QBE E with un-quantified variables $x_1 \quad \ldots x_k$

  is satisfiable iff the $\exists x_1 \cdots x_k \, E$ is satisfiable.

Q-SAT *is PSpace*

- Enough to deal with prenex QBE with no un-quantified variable.
- To evaluate $\forall x \exists y \forall z\, F[x, y, z]$ evaluate $\exists y \forall z\, F[0, y, z]$ and then $\exists y \forall z\, F[1, y, z]$, etc.
- This is implementable in linear space.

PSPACE COMPLETENESS

Reductions between PSpace problems

- Problem $\mathcal{Q}$ is ***PSpace-complete*** if
  - $\mathcal{Q}$ is in PSpace, and
  - $\mathcal{P} \leqslant_p \mathcal{Q}$ for every PSpace problem $\mathcal{P}$.
- Note that the reduction is in **PTime**.

A PSpace complete problem

- **LBA-ACCEPTANCE** :

  Given an LBA M and a string w

  does M accept w.

- **Theorem** **LBA-ACCEPTANCE** is PSpace complete.

- It is in linear space: the universal interpreter needs only the instructions of M and the string w.

- It is complete under PTime reductions:

  Suppose $\mathcal{P}$ is in space n^k,

  i.e. some acceptor M recognizes $\mathcal{P}$ in space $\leqslant n^k$ (some k),

  i.e. M accepts an instance w of $\mathcal{P}$ in $\leqslant |w|^k$ space.

- Let ρ map each instance w of $\mathcal{P}$

  to the instance M, w' of LBA-ACCEPT,

  where w' is w padded with $|w|^k$ blanks.

- This is a PTime mapping, and it is a reduction:

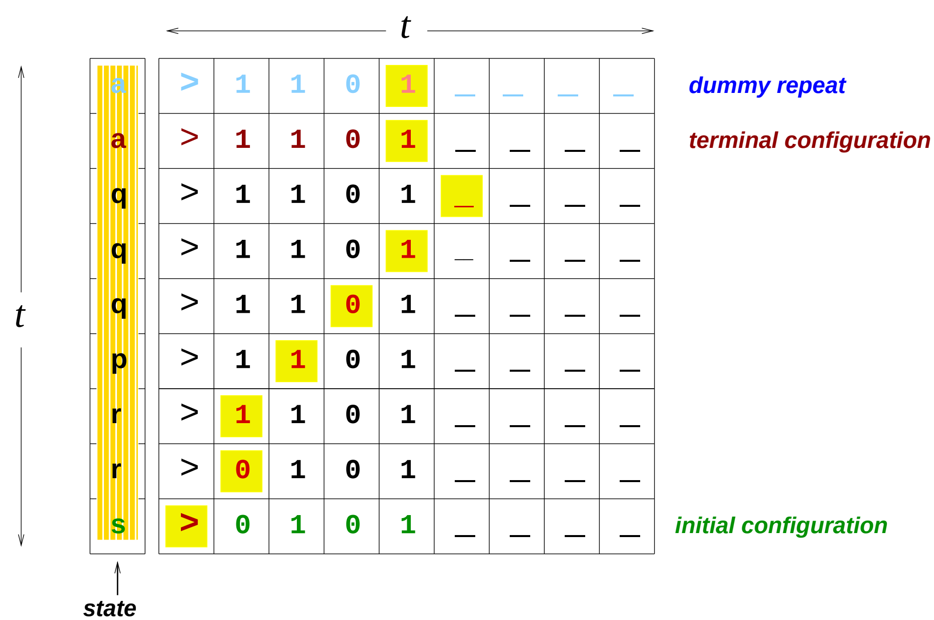
$w \in \mathcal{P}$ IFF $\boldsymbol{M}$ accepts $\boldsymbol{w}$

IFF $\boldsymbol{M}$ accepts $\boldsymbol{w'}$ on-site (since $\boldsymbol{M}$ is in space $\boldsymbol{n^k}$)

IFF $\rho(I) \in$ LBA-ACCEPT

QSAT *is PSpace hard*

- Reminder from the NP-hardness of **BOOL-SAT** :



- Variables:
  - $x_{i,q}$: *the state of the* i*'th cfg is* q.
  - $y_{i,j}$: *the cursor of the* i*'th cfg is at* j.
  - $z_{i,j,\sigma}$: *the* j*-th tape-symbol of the* i*'th cfg is* σ.

So a cfg is given by the boolean values of $\vec{x}, \vec{y}, \vec{z}$.

Abbreviate this as $\vec{X}$.

- One can now construct a boolean expression describing the grid.

  Its size is polynomial in the input-size

  because the height of the grid is linear

  because we looked at linear time (after padding).

The very tall grid

- But the number of cfg's in a trace for linear ***space***
  is ***exponential*** in input size!
  So the grid is a very tall rectangle.
- However, we can use a recursive program as we did for Savitch's Theorem.
- Write $J_t(\vec{X}, \vec{X}')$
  if cfg $\vec{X}$ leads to $\vec{X}'$ in 2^t steps.
- J_0 is D_M.
- Attempt for recurrence:

$$J_{t+1}[\vec{X}, \vec{X}'] \quad \equiv \quad \exists \vec{M} \;\; J_t[\vec{X}, \vec{M}] \wedge J_t[\vec{M}, \vec{X}']$$

- Problem: The size of J_{t+1} is $\geqslant$ twice the size of J_t.

The magic of boolean quantification

- We want J_t to appear only once in defining J_{t+1} .
- Take

$$
\begin{aligned}
J_{t+1}[\vec{X}, \vec{X}'] \;\equiv\; & \exists \vec{M} \;\; \forall \vec{U}, \vec{V} \\
& (\vec{U} = \vec{X} \wedge \vec{V} = \vec{M}) \;\; \vee \;\; (\vec{U} = \vec{M} \wedge \vec{V} = \vec{X}') \\
& \rightarrow \qquad J_t[U, V]
\end{aligned}
$$

- So the size of the QBE J_{n^k} is $O(n^k)$.

LSpace: The complexity of the internet

Work-space

- How much space does a DFA use for input of size 10^{25} ?

Work-space

- How much space does a DFA use for input of size 10^{25} ?
- The computation space of a machine is the space it owns,
  i.e. that it can write on.
- Example: the computation space of a phone is its local memory, not the entire internet.
- The computation space of a electronic camera is its hardware,
  not its field of vision.

Space complexity — redefined

- A variant of Turing machines:
  - The input is read-only
  - There is a read/write work-string ("work-tape").
  - Actions are triggered by the cursored symbols on the two strings.
  - An *action* is a cursor-move on one of the strings,
    or an overwrite of the work-cursor symbol.
- This is a trade-off between
  - Extra flexibility of second string and
  - less flexibility on the input string.
- Over-all computation power is the same:
  - A normal Turing machine can simulate input-string + work-string
    as input$work.

- A work-string machine can simulate a Turing machine by preprocessing: the input is copied into the work-string.

- The point is to represent use of space more realistically, in particular refer to sub-linear space complexity.

Example

- Recognize $\{x\,\mathtt{c}\,x \mid x \in \{\mathtt{a},\mathtt{b}\}^*\}$
- Single string algorithm
  - Go back and forth between the strings before and after $\mathtt{c}$, comparing one-symbol per cycle.
  - Use markers to identify the latest compared symbols.

  This algorithm is in quadratic time.
- Two-strings algorithm:
  - Copy input-string to work-string up to $\mathtt{c}$.
  - Compare the work-string with the remaining input after $\mathtt{c}$/

  This algorithm is in linear time.

Example 2: $\{a^n b^n c^n \mid n \geqslant 0\}$

- Previous example needs an exact match.

  What if we want to match only the counts?

- Example: Recognize $\{a^n b^n c^n \mid n \geqslant 0\}$.

- Single-string algorithm:

  - Scan the input n times

    to count corresponding `a`, `bc`.

- Time: quadratic.

  Space: linear.

- Work-string algorithm:
  - Use the work-string as a binary counter (or three in succession!)

    $\log n$ bits are needed.

    Incrementing a count takes $\leqslant \log n$ steps.

  1. Scan the input, and use the counters to count the number of `a`,

     then the number of `b` and of `c`'s, each counts in a separate counter.

  - Compare the three counts. Accept if they are equal.

- Time: $O(n \cdot \log\, n) + O(\log^2 n) = O(n \cdot \log\, n)$.

  Space: $O(\log n)$

Example 2: $\{a^n b^n c^n \mid n \geqslant 0\}$

- Previous example needs an exact match.

  What if we want to match counts?

- Example: recognize $\{a^n b^n c^n \mid n \geqslant 0\}$.

- Algorithm:

  - Count on the work-string the number of `a`'s, in binary; place a marker.
  - Count beyond the marker the number of `b`'s; place new marker.
  - Count bend marker the number of `c`'s.
  - Compare the three counts. Accept if they are equal.

- This algorithm runs in space logarithmic in the size of the input.

Logarithmic Space

- The problems decidable in logarithmic space:

  LogSpace or just **L**.

- We have $c^{d \cdot \log n} = 2^{d(\log c)(\log n)} = n^k$ ($k = d \log c$).

- So **LogSpace** $\subseteq$ **PTime**.

- Conclude:

  L $\subseteq$ **P** $\subseteq$ **NP** $\subseteq$ **PSpace** $\subseteq$ **ExpTime**

  We believe all are strict,

  but none of the containments above is known for fact to be strict.

⋆*Surfing the web*

- Gigantic read-only

  + modest local storage (addresses, auxiliary computing)

  is all around.

- So **LogSpace** is nowadays the most important complexity class.

- ***Multi-cursor two-way automata*** are a natural model

  of computing over large data.

- Example: with two cursors we can recognize $\{a^n b^n \mid n \geqslant 0\}$ (which is not regular).

- With three cursors we can recognize $\{ a^n b^n c^n \mid n \geqslant 0 \}$,

  which is not CF.

- A multi-cursor automaton can be simulated in log-space

  using addresses for the cursors.

- **Theorem (Hartmanis)** ($\sim$ 1960)

  ***A language is in* L**

  iff it is recognized by a multi-cursor two-way automaton.

- $\Longleftarrow$: We have seen that a multi-cursor automaton can be simulated in log-space.

- $\Longrightarrow$: Simulate a Turing acceptor M over $\{0,1\}$ operating in space $k \cdot \log n$ by a k-cursor automaton.

- Example: Input is $w = \texttt{00010110}$ (length 8).

  Work-string x is of size $O(\log n)$, say size 3.

  A dedicated cursor on w keeps track of the binary value coded by x.

  E.g., if $x = 110$, i.e. binary for 6, then the dedicated cursor would be at the sixth position of the input string:

  00010<u>**1**10</u>

- Another dedicated cursor keeps track of the

  position of the work cursor of M:

  If M's work-string is 1<u>1</u>0 (the cursor at second position),

  then ***this*** dedicated cursor would be at the second position

  of the input string: 0<u>0</u>010110.

- Simulating an overwrite by M on the work-string

  requires additional cursors to keep track of powers of two.