# Algorithms and Computation

# (grad course)

## Lecture 9: Space Complexity, Time and Space Hierarchy

Instructor: Hossein Jowhari

jowhari@kntu.ac.ir

Department of Computer Science and Statistics
Faculty of Mathematics
K. N. Toosi University of Technology

Fall 2021

# Time Complexity classes

$\mathbf{P} = \mathrm{DTIME}(\mathrm{poly}(n))$ $\qquad$ $\mathbf{NP} = \mathrm{NTIME}(\mathrm{poly}(n))$

$\mathbf{EXP} = \mathrm{DTIME}(2^{\mathrm{poly}(n)})$ $\qquad$ $\mathbf{NEXP} = \mathrm{NTIME}(2^{\mathrm{poly}(n)})$

$\mathbf{EXP}$ is the class of languages that are decidable by Turning machines with time complexity bounded by $2^{\mathrm{poly}(n)}$.

$\mathbf{NEXP}$ is the class of languages that are decidable by non-deterministic Turning machines with time complexity bounded by $2^{\mathrm{poly}(n)}$.

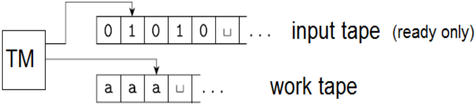
$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP} \subseteq \mathbf{NEXP}$$

**Open Questions**: $\mathbf{NP} \neq \mathbf{P}?$, $\quad \mathbf{NP} \neq \mathbf{EXP}?$

**Known**: $\mathbf{P} \neq \mathbf{EXP}$ and $\mathbf{NP} \neq \mathbf{NEXP}$
(Time Hierarchy Theorem) Next Lecture!

# Space Complexity

The space complexity of a Turing machine $M$ is the function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of work tape cells that $M$ scans on any input of length $n$.



# Space Complexity Classes

$\mathbf{PSPACE} = \mathrm{SPACE}(\mathrm{poly}(n))$

$\mathbf{NPSPACE} = \mathrm{NSPACE}(\mathrm{poly}(n))$

$\mathbf{L} = \mathrm{SPACE}(O(\log n))$ $\qquad$ $\mathbf{NL} = \mathrm{NSPACE}(O(\log n))$

$\mathbf{PSPACE}$ is the class of languages that are decidable by Turning machines with space complexity bounded by $\mathrm{poly}(n)$.

$\mathbf{NPSPACE}$ is the class of languages that are decidable by non-deterministic Turning machines with space complexity bounded by $\mathrm{poly}(n)$.

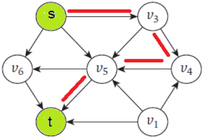
- $A = \{0^n 1^n \mid n \geq 0\}$

  $A \in \mathbf{L}$. (Why?)


- PATH $= \{< G, s, t > \mid G$ is a directed graph with a path from $s$ to $t\}$

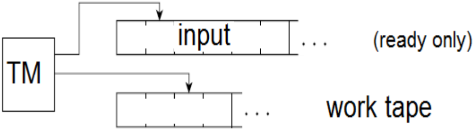
  PATH $\in \mathbf{NL}$.

  Starting from the vertex $s$, the algorithm non-deterministically guesses the next vertex of the path. It accepts the input if it reaches $t$ after at most $n-1$ correct guesses otherwise it rejects.



Is $\mathrm{Distinct}(A)$ in in $L$?

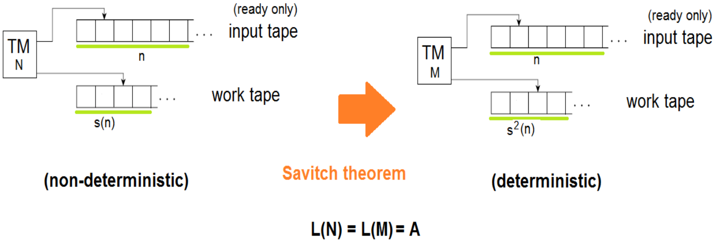
The function $\mathrm{Distinct}(A)$ returns the number of distinct elements in the list $A$.



# Savitch's Theorem

**Theorem**: For any nice space bound $s(n) \geq \log(n)$ we have $\mathrm{NSPACE}(s(n)) = \mathrm{SPACE}(s^2(n))$.

In other words, let $N$ be a non-deterministic Turing machine with space complexity $s(n)$, where $s(n) \geq \log(n)$, that decides the language $A$. There is a deterministic Turing machine $M$ with space complexity $O(s^2(n))$ that decides $A$.



# Proof of Savitch's Theorem

We begin with the concept of the configuration of a Turing machine.

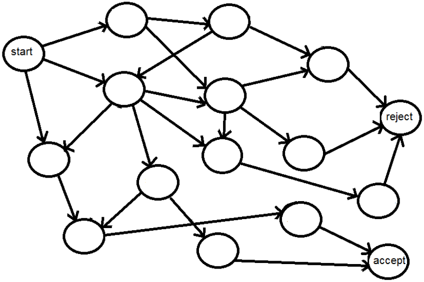
**Configuration of a Turing machine**: Snapshot of the machine at some point in time when processing an input string. It consists of the state of the machine, positions of the (read/write) heads and the content of the work tape.

Configuration of the machine $N =$

$$\underbrace{(\underbrace{\text{state}}_{O(1)\text{ bits}},\ \underbrace{\text{position of the heads}}_{O(\log n)+O(\log s(n))\text{ bits}},\ \underbrace{\text{content of the work tape}}_{s(n)\text{ bits}})}_{O(s(n))\text{bits}}$$

# Configuration graph

- Given a machine $N$ and input $x$, we can imagine a configuration graph $G_x$.




- Every node in $G_x$ is a configuration of the machine. $G_x$ includes all possible configurations.
- There is an edge from the configuration $C$ to $C'$ if the machine can go from $C$ to $C'$ in one step.

- The non-deterministic machine accepts the input $x$ iff there is a path from the node $start$ (start configuration) to the node $accept$ (accept configuration).

- **Question**: Assuming $N$ works in $s(n)$ space and $|x| = n$, how many nodes does the graph $G_x = (V, E)$ have?

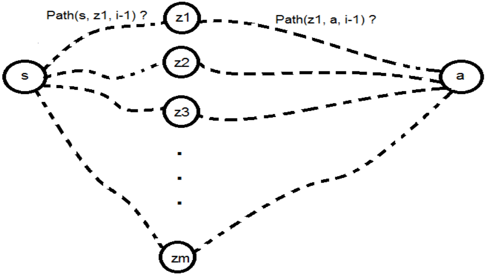
  $|V| = 2^{O(s(n))}$ nodes.

- Given the machine $N$ and input $x$, Savitch theorem decides the existence of a $start - accept$ path in $G_x$ using $O(s^2(n))$ space via a deterministic strategy.

- Savitch uses a subroutine $Path(x, y, i)$ which outputs accepts iff there is path from $x$ to $y$ in $G_x$ with length at most $2^i$.

```
Algo Path(x, y, i)
    if i = 0 then
        Accept iff x = y OR (x, y) ∈ E
    end if
    for every z ∈ V
        if Path(x, z, i − 1) accepts AND
          Path(z, y, i − 1) accepts % we re-use space here!
        then Accept
        end if
    end for
    Reject % if no “middle” point z was found, we reject
end Algo
```

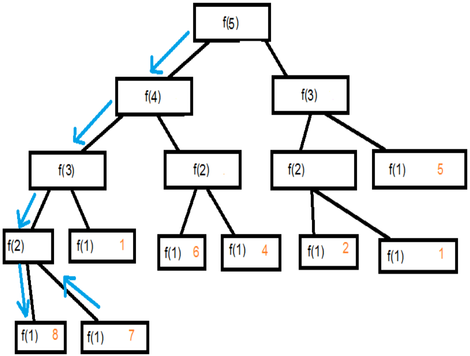
$P(x, y, i) \equiv$
is there a path from $x$ to $y$ with length at most $2^i$?

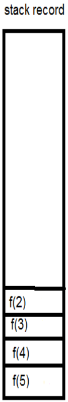


# Space usage of a recursive function

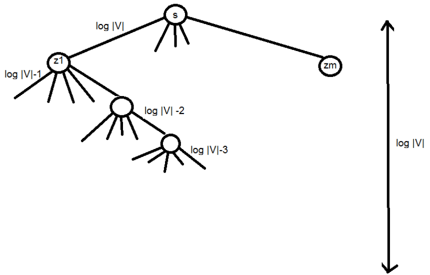
$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = random(1, 10)$$



- $N$ accepts $x$ iff $Path(start, accept, \log|V|)$ accepts.




- The depth of the recursion is $\log|V|$
- The size of each configuration (stack record) is $O(s(n)) = O(\log|V|)$ bits.
- Therefore the space complexity of algorithm $Path(start, accept, \log|V|)$ is $O(\log^2|V|)$ which is $O(s^2(n))$

# Implications of the Savitch Theorem

- $\mathbf{NPSPACE} = \mathbf{PSPACE}$
- $\mathbf{NL} \subseteq \mathbf{L}^2 = \mathrm{SPACE}(\log^2(n))$
- Given a directed graph $G$ on $n$ nodes, we can check $s-t$ connectivity (is there a path from $s$ to $t$) using $O(\log^2 n)$ space.

  What is the running time of the Savitch algorithm?

# Running time of the Savitch algorithm

- The running time of the Savitch algorithm is the size of the recursion tree.
- Each node has $2n$ children. Depth of the tree is $\log n$.
- Size of the tree =

$$(2n) + (2n)^2 + (\ldots) + (2n)^{\log n} = O(n^{\log n})$$

- Note that $n^{\log n}$ is not polynomial time.
- **Question:** $\mathbf{NL} \subseteq \mathbf{P}$ **?**

  (Yes)

# NL and P

- Note that we can deterministically check $s-t$ connectivity of an $n$-node graph using $O(n)$ space and in $O(n^2)$ time. (BFS algorithm).
- Given a non-deterministic machine $N$ that uses $O(\log n)$ space, we can write down all the different $2^{O(\log n)} = O(n)$ configurations on the work tape and solve the associated reachability problem for an input of size $n$ in $O(n^2)$ time. Therefore

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

- Note that it is an open question whether we can solve the reachability problem in polynomial time using $\mathrm{poly}(\log n)$ space or not.

Also we have

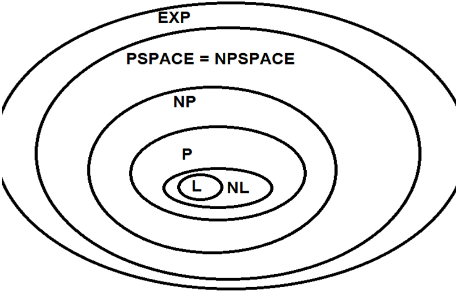
$$\mathbf{NP} \subseteq \mathbf{PSPACE}$$

Because

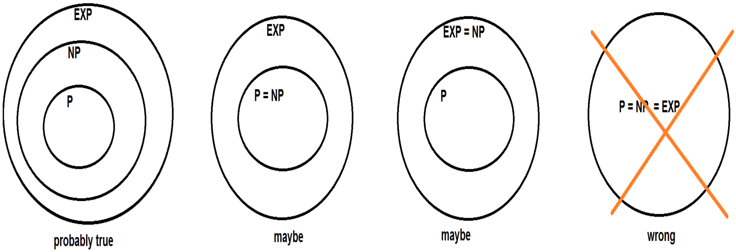
$$\mathrm{SAT} \in \mathbf{PSPACE}$$

we can try all the different assignments of the variables of a SAT formula of size $n$ using $\mathrm{poly}(n)$ space.

If $\mathbf{P} \neq \mathbf{NP} \neq \mathbf{EXP}$ then we have the following situation



# P and NP and EXP




It seems almost obvious that $\mathbf{P} \neq \mathbf{EXP}$ but how can we prove it?

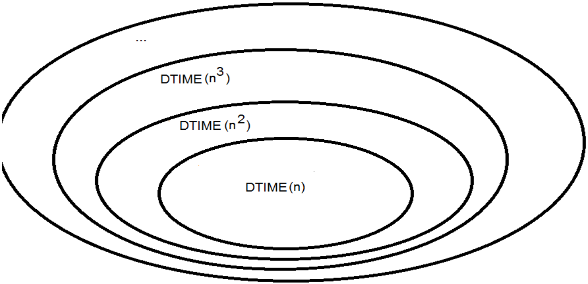
We need to show that there is a problem $A \in \mathbf{EXP}$ that is not in $\mathbf{P}$.

Common sense suggests that giving a Turing machine more time or more space should increase the class of problems that it can solve. For example, Turing machines should be able to decide more languages in time $n^3$ than they can in time $n^2$. The ***hierarchy theorems*** prove that this intuition is correct, subject to certain conditions described below. We use the term *hierarchy theorem* because these theorems prove that the time and space complexity classes aren't all the same—they form a hierarchy whereby the classes with larger bounds contain more languages than do the classes with smaller bounds.

Introduction to the theory of computation, Michael Sipser

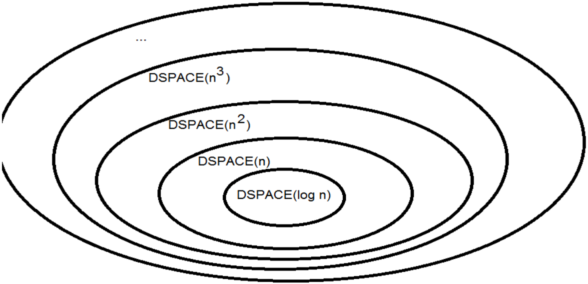
**Time Hierarchy Theorem**: Roughly speaking time hierarchy theorem says that there are problems that are decidable by Turing machines with time complexity $f(n)$ while they are not decidable by Turing machines with time complexity $f(n)/\log f(n)$. For example there is a problem in $\mathrm{DTIME}(n^3)$ but not in $\mathrm{DTIME}(n^2)$.

# Time Hierarchy



# Space Hierarchy

It is shown there are languages in $\mathrm{DSPACE}(f(n))$ that are not in $\mathrm{DSPACE}(o(f(n)))$. For example there are languages in $\mathrm{DSPACE}(n^2)$ that are not in $\mathrm{DSPACE}(n)$.




Time and Space hierarchy theorems are proven via a technique called **Diagonalization.**