Space Complexity Measure

- A language L on an alphabet Σ is *log-space* (*resp.*, *polynomial-space*) decidable, if there is a Turing machine $\mathcal{M}$ that accepts L and a polynomial $p(n)$ such that for every $x \in L$, $\mathcal{M}$ scans at most $p(|x|)$ (*resp.*, $O(\log_2(|x|))$ tape squares on its tape.
- Replacing the deterministic Turing machine in the above definition with a non-deterministic one, we have *non-deterministic log-space* and *non-deterministic polynomial-space*.
- The classes of all *log-space*, *non-deterministic log-space*, *polynomial-space* and *non-deterministic polynomial-space* acceptable languages are denoted as **L**, **NL**, **PSPACE** and **NPSPACE**.
- A language L on an alphabet Σ is *log-square-space* decidable, if there is a Turing machine $\mathcal{M}$ that accepts L such that for every $x \in L$, $\mathcal{M}$ scans at most $O(\log_2^2(|x|))$ tape squares on its tape.
- The class of all *log-square-space* acceptable languages is denoted as $\mathbf{L}^2$.

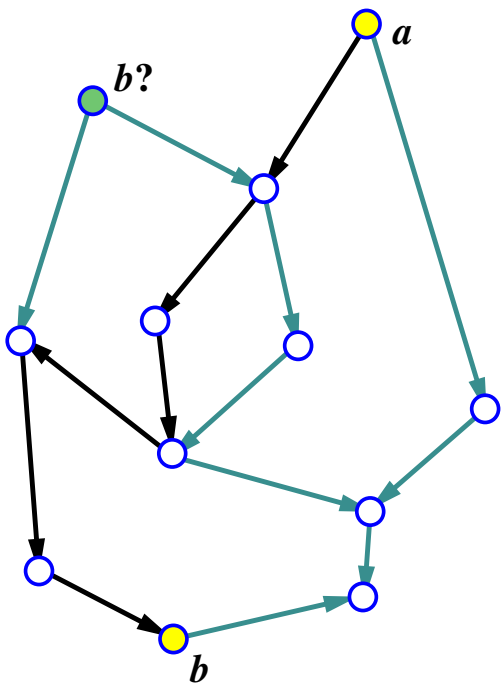
- Languages in $\mathbf{L}$, $\mathbf{L}^2$, $\mathbf{NL}$, $\mathbf{PSPACE}$ and $\mathbf{NPSPACE}$ may take an exponential number of steps, although they need at most polynomial number of space. They are more complex than languages in $\mathbf{P}$ and $\mathbf{NP}$.
- A language L on an alphabet Σ is *exponential time* and *non-deterministic exponential-time* decidable, if there is a deterministic and non-deterministic Turing machine $\mathcal{M}$ that accepts L and a polynomial $p(n)$ such that for every $x \in L$, the number of steps in an accepting computation by $\mathcal{M}$ with input is $\leq 2^{p(|x|)}$.
- The classes of all *exponential time* and *non-deterministic exponential time* acceptable languages are denoted as $\mathbf{EXPTIME}$ and $\mathbf{NEXPTIME}$, respectively.
- The following facts are obvious:
  1. $\mathbf{L} \subseteq \mathbf{NL}$
  2. $\mathbf{L} \subseteq \mathbf{L}^2 \subseteq \mathbf{PSPACE} \subseteq \mathbf{NPSPACE}$
  3. $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{NEXPTIME}$
- We want to prove:

  $$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{NEXPTIME}$$

  and $\mathbf{PSPACE} = \mathbf{NPSPACE}$.

The Reachability Problem

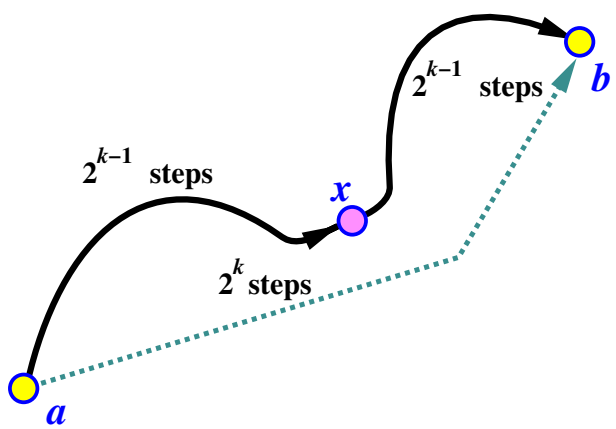
- **Given a directed graph $G = (V, E)$ and two vertices a and b, is there a directed path from a to b?**



- With the *Depth-First-Search* (DFS), we can perform this test in $O(|V| + |E|)$. However, the space requirement is $O(|V|)$.
- We need an algorithm that uses space in a more efficient way.

Savitch's Theorem

- **The Reachability Problem can be solved in** $O(\log_2^2 |V|)$ **space** on a Turing machine.
- We shall use a divide-and-conquer technique. Define predicate $\mathsf{Reach}(a, b, 2^k)$ to be "**there is a directed path from** a **to** b **with a length of** $\leq 2^k$".
- If $\mathsf{Reach}(a, b, 2^k)$ is TRUE, then there exists a vertex x on this path such that $\mathsf{Reach}(a, x, 2^{k-1})$ and $\mathsf{Reach}(x, b, 2^{k-1})$ both hold. More precisely, x is the mid-point!

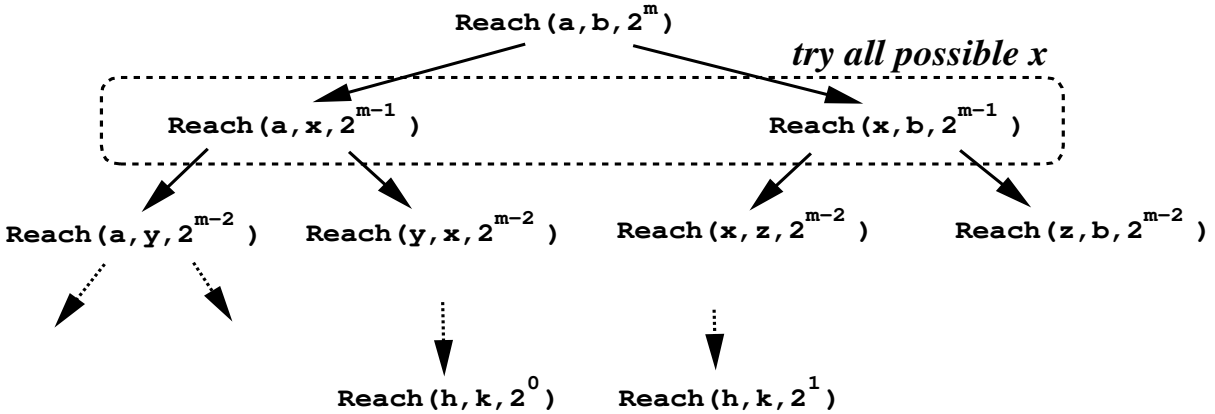


- In this way, a recursive procedure is possible.

- Since there are $|V|$ vertices in the graph, the longest possible path has a length of $|V|$. Since a path is always divided in the middle (hopefully), this procedure is called with $\mathsf{Reach}(a, b, 2^{\lceil \log_2 |V| \rceil})$, where a and b are the origin and destination vertices.

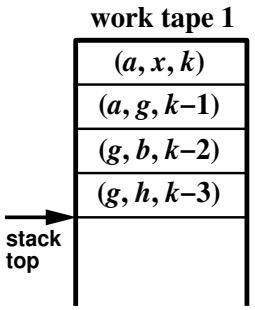
```
BOOLEAN FUNCTION  Reach(Vertex a, Vertex b, INTEGER m)
BEGIN
   IF (m = 1) THEN
      IF ((a = b) OR (a ⟶ b is an edge)) THEN
         RETURN TRUE
      ELSE
         RETURN FALSE
   ELSE
      BEGIN
         FOR each vertex x DO
            BEGIN
               IF (Reach(a,x,m/2) AND Reach(x,b,m/2)) THEN
                  RETURN TRUE
            END
         RETURN FALSE
      END
END FUNCTION  Reach
```

- Thus, searching for a solution is equivalent to performing a tree traversal. The depth of this recursion, or tree traversal, is at most $\lceil \log_2(|V|) \rceil$



- How do we realize it on a Turing machine?

  1. The **input** tape contains the graph and vertices a and b. The first **work** tape initially contains $(a, b, |V|)$ and is treated as a stack.

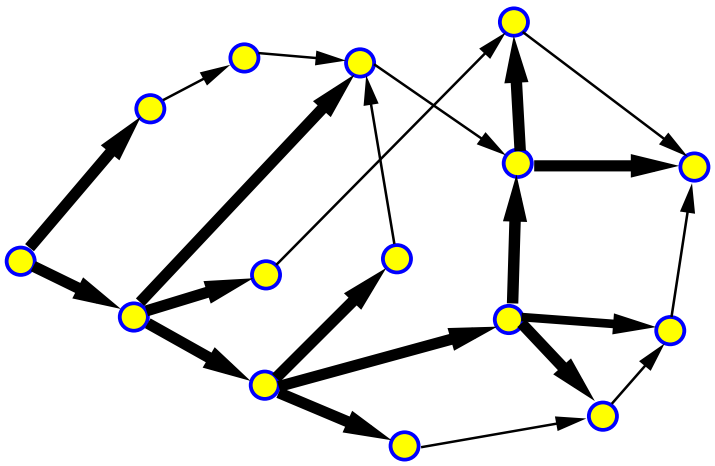


  2. What we need to do is to remove the recursion using a stack (*i.e.*, the first **work** tape).
  3. The second **work** tape is used for enumerating x and determining if we should go left or right.
  4. Hence, at any moment, the first **work** tape contains no more than $\lceil \log_2 |V| \rceil$ records.

- Each vertex is represented by an integer, and each integer uses $\log_2 |V|$ bits (*i.e.*, tape squares). Since there are no more than $\lceil \log_2 |V| \rceil$ records on the first **work** tape, it contains $(\lceil \log_2 |V| \rceil)^2$ "used" tape cells.
- The second tape only contains $O(\log_2 |V|)$ cells for storing x and other local variables.
- Thus, the total tape cells scanned is $O(\log_2^2 |V|)$.

Configuration and Configuration Graph

- **The configuration of a k-tape Turing machine** is a $(2k+1)$-tuple $(p, h_1, h_2, \ldots, h_k, x_1, x_2, \ldots, x_k)$, where p is a state, h_i is the tape head position of tape i, and x_i is the string on tape i, including all the non-blank symbols and the symbol under the tape head.
- Given a non-deterministic Turing machine $\mathcal{M}$ and its input string w, the associated **configuration graph** is a graph whose vertices are configurations of $\mathcal{M}$. There is an edge from configuration i to configuration j, if for some choice of input vector c, $\mathcal{M}$ can move from i to j.



- In the above, thick and thin arrows are non-deterministic and deterministic moves, respectively.

- Suppose non-deterministic Turing machine $\mathcal{M}$ has Q states, 1 input tape, 1 output tape, s work tapes, and c symbols. Suppose further that $\mathcal{M}$ scans no more than $p(n)$ tape cells, where n is the length of an input string and $p(n)$ is a **function** in n. We want to count the number of configurations.

- A configuration consists of three components:

  1. **Content on each tape:**
     Since each tape is scanned at most $p(n)$ positions, there are $s+2$ tapes, and each tape cell can have one of the c symbols, these tapes have at most $c^{(s+2)p(n)}$ different combinations.

  2. **Tape head positions:**
     Since the head of the input tape can assume one of the n positions, the total number of all possible tape head positions is

$$\underbrace{n}_{\text{input}} \times \underbrace{p(n)^{s+1}}_{\text{output \& work}}$$

  3. **Number of states:**
     There are Q states.

- Therefore, the total number of *all possible* configurations is

$$Q \times \left[c^{(s+2)p(n)}\right] \times \left[np(n)^{s+1}\right]$$

- We can find some $k > 1$ such that

$$Q \times \left[c^{(s+2)p(n)}\right] \times \left[np(n)^{s+1}\right] \leq k^{p(n)+\log_2(n)}$$

- The reason we have $\log_2(n)$ in the above is because in some applications $p(n)$ may be less than a polynomial (*i.e.*, $p(n) = \log_2(n)$ or $p(n) = \log_2^2(n)$).
- If $p(n)$ is a polynomial, since $\log_2 n \leq p(n)$, the bound becomes $k^{p(n)}$.
- Thus, the number of configurations for accepting an input string is bounded above by $k^{p(n)+\log_2(n)}$.

PSPACE = NPSPACE

- This is a direct consequence of Savitch's Theorem.
- Since **PSPACE** $\subseteq$ **NPSPACE** is obvious, we need to prove the converse **NPSPACE** $\subseteq$ **PSPACE**.
- We shall prove that a non-deterministic Turing machine, with space bound $O(p(|x|))$, can be simulated on a deterministic Turing machine with space bound $O(p^2(|x|))$, where x is an input string.
- Suppose L is a language in **NPSPACE**. The space complexity of L is bounded by a polynomial $p(n)$, where n is the length of an input string $x \in L$.
- From this non-deterministic Turing machine and its input, we can construct a configuration graph. The number of vertices of this configuration graph is bounded by $k^{p(n)}$. We do the following:
  1. Put the non-deterministic Turing machine description and the input string x on the input tape.
  2. The configurations are stored on the work tape. The adjacency relation of configurations can be deduced from the original Turing machine.

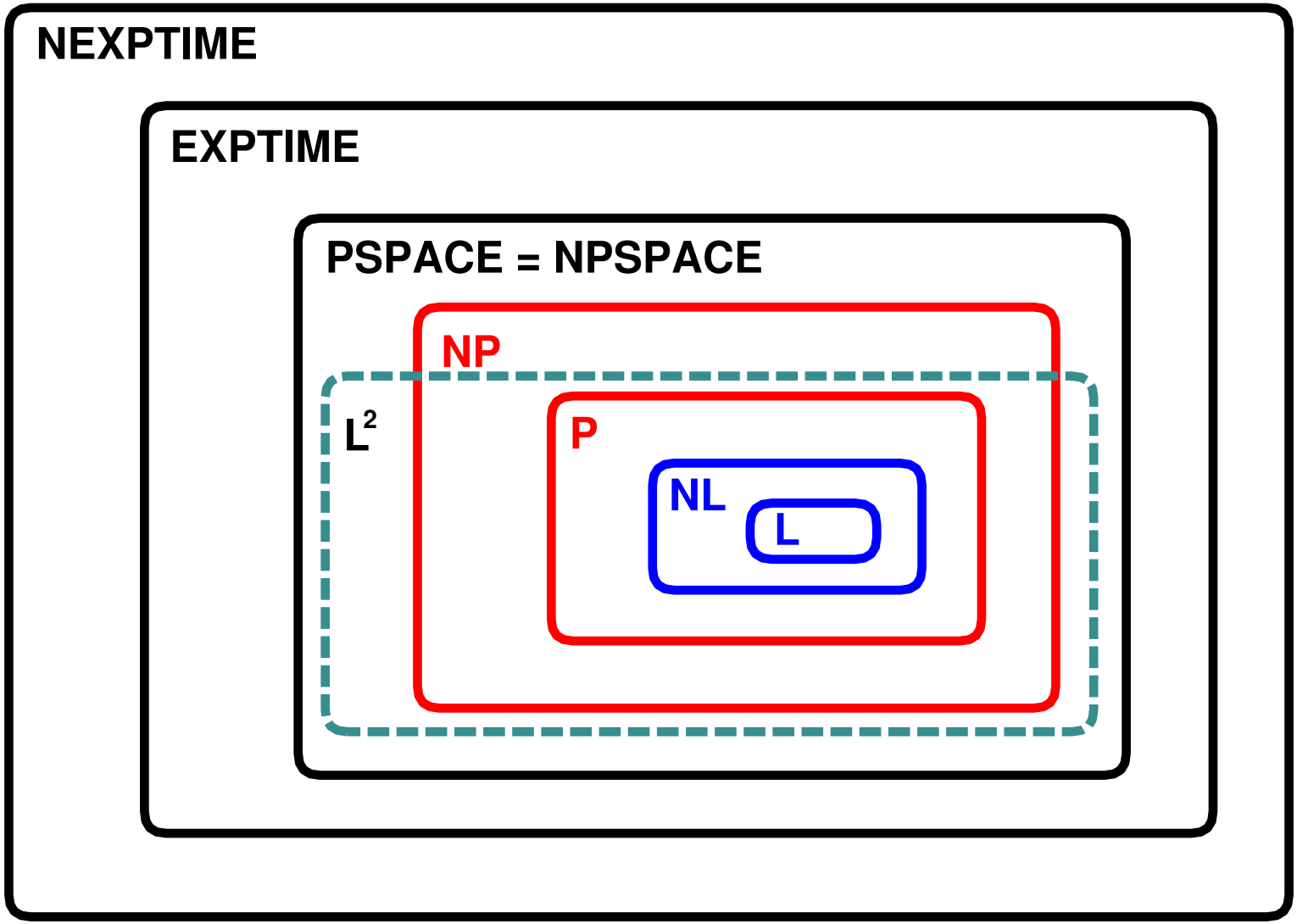
3. Since the number of configurations is $O(k^{p(n)})$, each configuration can be stored on the work tape using $O(\log_2(k^{p(n)})) = O(p(n))$ tape squares.

- Determining if string x is acceptable is equivalent to find a path from the initial configuration to a configuration (*i.e.*, "reachability").
- Because there are at most $k^{p(n)}$ vertices in the configuration graph, by Savitch's Theorem, the reachability problem can be solved with space bound $O(\log_2^2(k^{p(n)})) = O(p^2(n))$. Hence, $L \in \mathbf{PSPACE}$.
- **Therefore, non-deterministic Turing machines do not introduce extra power to space bound problems.**
- This result also implies $\mathbf{NL} \subseteq \mathbf{L}^2$, since a non-deterministic machine that accepts $L \in \mathbf{NL}$ using space $O(\log_2 |x|)$ can be simulated with a deterministic Turing machine using space $O(\log_2^2 |x|)$.

Relation between Non-deterministic Space and Deterministic Time

- **A non-deterministic Turing machine with space bound $O(p(n))$ can be simulated with a deterministic Turing machine with time bound $O(k^{\log_2 n + p(n)})$ for some $k > 1$.**
- This is a variation of Savitch's Theorem.
- We can construct a configuration graph from the given non-deterministic machine and an input string. The configuration graph has at most $O(k^{\log_2 n + p(n)})$ vertices, where n is the length of the input string.
- The depth-first search algorithm can report if there is a directed path from the initial to final configuration.
- The time bound of DFS is $O(|V| + |E|)$. Since $|E| \leq |V|^2$, the time bound of DFS is $O(|V|^2)$.
- The "reachability" has a time bound of $O(((k^{\log_2 n + p(n)})^2) = O(k^{2(\log_2 n + p(n))})$.
- **Consequences: NL $\subseteq$ P** and **NPSPACE $\subseteq$ EXPTIME**.

Relation between Non-deterministic Time and Deterministic Space

- **A non-deterministic Turing machine with time bound $O(p(n))$ can be simulated with a deterministic Turing machine with space bound $O(p(n))$.**
- Check your Theory book and you should find such a simulation. This simulation uses three tapes: an input tape that is never modified, a work tape that is used to simulate the non-deterministic Turing machine, and an **enumerating** tape that contains choice sequences used by the deterministic machine to decide which move to make when simulating the non-deterministic one. All tapes will only be used $O(p(n))$ squares.
- **Consequences: NP $\subseteq$ PSPACE**.

This is what we have so far