Recall that a directed graph is ***strongly connected*** if every two nodes are connected by a directed path in each direction. Let

$$STRONGLY\text{-}CONNECTED = \{\langle G \rangle \mid G \text{ is a strongly connected graph}\}.$$

Show that *STRONGLY-CONNECTED* is NL-complete.

证明NL-complete的一般思路：
1.本身是NL
2.证明PATH/2SAT问题可以规约到这个问题
PATH问题本身是有向图吗，无向和有向图之间的复杂度有区别吗

注意：path并不是指的直接连接，只是一个可达的路径

1的一般论述：非确定地选择一个xx，在这个分支下，想要证明这个xx符合题目定义，需要的存储空间是yy（一般是;

2的一般论述：
证明 PATH ≤ₗ STRONGLY-CONNECTED（对数空间归约）
需构造对数空间归约，将 PATH 的实例 <G, s, t> 转化为 STRONGLY-CONNECTED 的实例 <G'>，满足：
2.1 G 中 s 到 t 有路径 等价于 G' 是强连通图。
2.2 归约过程的空间开销是对数级的
归约的构造步骤[construction]
归约正确性验证说明[correctness]
归约的空间开销分析[log-space reduction]

Proof that STRONGLY-CONNECTED is NL-complete

To prove STRONGLY-CONNECTED is NL-complete, we must show:

1. STRONGLY-CONNECTED ∈ NL
2. All problems in NL reduce to STRONGLY-CONNECTED (shown via PATH ≤_L STRONGLY-CONNECTED)

STRONGLY-CONNECTED ∈ NL

A directed graph is strongly connected if every pair of nodes has directed paths in both directions. We design a nondeterministic log-space algorithm:

- Nondeterministically pick a starting node s (storing s uses O(log n) space)
- For every other node v:
  - Use the NL algorithm for PATH to verify there's a path from s to v

    - Use the same PATH algorithm to verify there's a path from v to s
- Accept if all verifications pass; otherwise reject

Space complexity: O(log n) for storing node indices and PATH computation rule.

PATH $\leq_L$ STRONGLY-CONNECTED

We reduce PATH (directed s-t connectivity) to STRONGLY-CONNECTED in log space.

Construction:
Given PATH instance ⟨G, s, t⟩, construct G' by:

- Keeping all original edges of G
- Adding edge t → s
- For every node v, adding edges v → s and s → v

Correctness:

- If PATH accepts: There's an s-t path in G. Combined with added edges, all nodes become mutually reachable via s, making G' strongly connected.
- If PATH rejects: No s-t path exists. The added edges don't create new s-t paths, so G' cannot be strongly connected.

Log-space reduction:
The reduction doesn't explicitly construct G'. Instead, when checking edges in G' (to decide whtether it is strongly-connected):

- Check if u → v exists in original G, OR
- Check if (u = t ∧ v = s), OR (v = s), OR (u = s)
  All checks use O(log n) space to store node indices.

Conclusion

Since STRONGLY-CONNECTED ∈ NL and PATH $\leq_L$ STRONGLY-CONNECTED (where PATH is NL-complete), STRONGLY-CONNECTED is NL-complete.

$$\text{Prove that } \mathrm{TIME}(2^n) = \mathrm{TIME}(2^{n+1}).$$

定义：TIME(2^n)也就是可以在这个时间内判定的语言的集合
即证明：这两个时间复杂度类包含的问题是一样的
常数因子在O符号中可以被忽略（constant factors are ignored in big-O notation）

Proof that TIME(2^n) = TIME(2^{n+1})

In computational complexity theory, the complexity class **TIME(t(n))** denotes the set of all decision problems that can be solved by a deterministic Turing machine within time **O(t(n))**.

We aim to prove that:

$$\mathrm{TIME}(2^n) = \mathrm{TIME}(2^{n+1})$$

Proof

Note that:

$$2^{n+1} = 2 \times 2^n$$

Therefore, for the time complexity:

$$O(2^{n+1}) = O(2 \times 2^n) = O(2^n)$$

since constant factors are ignored in big-O notation.

This implies that any problem solvable in $O(2^{n+1})$ time is also solvable in $O(2^n)$ time, and vice versa. Hence, the sets $\mathrm{TIME}(2^n)$ and $\mathrm{TIME}(2^{n+1})$ are identical.

Thus, we conclude:

$$\mathrm{TIME}(2^n) = \mathrm{TIME}(2^{n+1})$$