Let A be the language of properly nested parentheses. For example, (()) and (()(()))() are in A, but)(is not. Show that A is in L.

L = SPACE(logn),其中n是输入长度
L：确定性图灵机在对数空间（\(O(\log n)\)空间）内可解决的问题类。
题目要求证明：A可以一个O(logn)空间的确定性图灵机判定
对比课上知识：证明回文语言是L

Proof

To show that the language A (of properly nested parentheses) belongs to $\mathbf{L}$ (the class of languages decidable in deterministic logarithmic space), we'll construct a deterministic 2-tape Turing machine that operates within $O(\log n)$ working space.

Step 1: Understanding the Problem

The Language A

A consists of all strings over the alphabet $\{(,)\}$ that are properly nested. Formally, a string $w \in A$ if:

1. The number of '(' equals the number of ')'
2. In every prefix of w, the number of '(' is at least the number of ')'

About $\mathbf{L}$

A language is in $\mathbf{L}$ if there exists a deterministic Turing machine with:

- A read-only input tape
- A read-write work tape
- That uses only $O(\log n)$ space on the work tape
- And decides whether the input belongs to the language

Step 2: Building the Turing Machine

We design a 2-tape TM M that tracks the current nesting depth:

Tape Configuration:

- **Input Tape**: Read-only, contains the input string w of length n

- **Work Tape**: Read-write, stores the current depth counter d in binary

Algorithm:

1. Initialize depth $d = 0$ on work tape
2. For each character in input string:
   - If character is '(': increment d ($d \leftarrow d + 1$)
   - If character is ')':
     - If $d = 0$: REJECT (too many closing parentheses)
     - Else: decrement d ($d \leftarrow d - 1$)
3. After processing all characters:
   - If $d = 0$: ACCEPT
   - Else: REJECT

Step 3: Space Complexity Analysis

The work tape only stores the depth counter d:

- Maximum possible depth: n (if input is all '(')
- Binary representation of d requires: $\lceil \log_2(n+1) \rceil$ bits
- Therefore, space usage is $O(\log n)$

Step 4: Conclusion

Since our deterministic 2-tape Turing machine:

- Correctly decides language A
- Uses only $O(\log n)$ work space
- Follows the definition of $\mathbf{L}$

Thus, we prove that $A \in \mathbf{L}$.

An undirected graph is ***bipartite*** if its nodes may be divided into two sets so that all edges go from a node in one set to a node in the other set. Show that a graph is bipartite if and only if it doesn't contain a cycle that has an odd number of nodes. Let $BIPARTITE = \{\langle G \rangle | \ G \text{ is a bipartite graph}\}$. Show that $BIPARTITE \in \text{NL}$.

NL：非确定性图灵机在对数空间内可解决的问题类。
非确定对数空间问题可被限制在平方对数空间内的确定性图灵机解决。
是二分图 当且仅当 无环且有奇数个点
对比课上知识：证明PATH问题是NL

Proof of iif

To prove that *"an undirected graph is bipartite if and only if it contains no odd-length cycles"*, we split the proof into **necessity** (bipartite ⇒ no odd cycles) and **sufficiency** (no odd cycles ⇒ bipartite):

1. Necessity: If a graph is bipartite, it contains no odd-length cycles

Let G be a bipartite graph with node partition (A, B) (all edges connect nodes in A to nodes in B).

Take any cycle $C = v_1 \to v_2 \to \cdots \to v_k \to v_1$ (length k) in G:

- Since G is bipartite, the node set alternates between A and B: if $v_1 \in A$, then $v_2 \in B$, $v_3 \in A$, ..., $v_k \in B$ (to connect back to $v_1 \in A$).
- The length k of the cycle must be **even** (alternating k times to return to the starting set).

Thus, all cycles in a bipartite graph are even-length; **no odd-length cycles exist**.

2. Sufficiency: If a graph contains no odd-length cycles, it is bipartite

Assume G is a connected undirected graph (non-connected graphs can be handled by component) with no odd cycles. We construct a bipartition via **BFS-based 2-coloring**:

1. Pick a starting node s, assign $s \in A$.
2. Perform BFS on G:
   - If current node $u \in A$, assign all unvisited neighbors of u to B.
   - If current node $u \in B$, assign all unvisited neighbors of u to A.
3. Verify the partition is valid:
   Suppose there exists an edge between two nodes $u, v \in A$ (or $u, v \in B$):
   - By BFS coloring, the distances $d(u)$ (from s to u) and $d(v)$ (from s to v) have the **same parity** (both even or both odd).

- The edge (u, v) forms a cycle $s \to \cdots \to u \to v \to \cdots \to s$, with length $d(u) + d(v) + 1$.
- Since $d(u) + d(v)$ is even, the cycle length is $\text{even} + 1 = \text{odd}$—contradicting the "no odd cycles" premise.

Thus, the BFS partition (A, B) is a valid bipartition; G is bipartite.

Conclusion

A graph is bipartite **if and only if** it contains no odd-length cycles.

Proof of $\text{BIPARTITE} \in \mathbf{NL}$

To show that $\text{BIPARTITE} \in \mathbf{NL}$, we'll use the characterization that a graph is bipartite if and only if it contains **no odd-length cycles**, and construct a non-deterministic log-space Turing machine to decide BIPARTITE.

Step 1: Background

Bipartite Graphs

A graph is **bipartite** if and only if it contains **no odd-length cycles**. Equivalently, we can 2-color the vertices such that no two adjacent vertices share the same color.

Complexity Class $\mathbf{NL}$

$\mathbf{NL}$ consists of languages decidable by a **nondeterministic Turing machine** using $O(\log n)$ space on the work tape, where n is the input size.

Step 2: Constructing the Nondeterministic Turing Machine

Inspired by the 2-color problem, I design a nondeterministic TM M that checks for odd cycles using the 2-coloring approach.

Machine Components

- **Input Tape**: Read-only, stores the graph $G = (V, E)$
- **Work Tape**: Read & Write, stores:
  - Current node (binary representation, $O(\log |V|)$ space)
  - Current color (0 or 1, constant space)

- Step counter (binary representation, $O(\log |V|)$ space)

Algorithm

1. **Initialize**:
   - Nondeterministically choose starting node v_0
   - Set v_0's color to 0
   - Initialize step counter to 0
2. **Traverse and Color**:
   While step counter $< |V|$:
   - Read current node u and its color c from work tape
   - Nondeterministically choose adjacent node v
   - Check v's color:
     - If v is uncolored: assign color $1 - c$
     - If v is colored and color equals c: **REJECT** (odd cycle found)
     - If v is colored and color equals $1 - c$: continue
   - Update current node to v
   - Increment step counter
3. **Terminate**:
   - If no color conflicts found after $|V|$ steps: **ACCEPT**

Step 3: Space Complexity

The work tape stores:

- Current node: $O(\log |V|)$ bits
- Current color: $O(1)$ bits
- Step counter: $O(\log |V|)$ bits

Total space: $O(\log |V|) = O(\log n)$, satisfying $\mathbf{NL}$ requirements.

Step 4: Conclusion

Since we've constructed a NTM that:

- Correctly decides $\mathrm{BIPARTITE}$
- Uses only $O(\log n)$ work space
- Follows the definition of $\mathbf{NL}$

We conclude that $\mathrm{BIPARTITE} \in \mathbf{NL}$.