典型习题讲解

Question 3

Show that P is closed under union, concatenation, and complement.

给定两个（或一个）属于P的语言，我们需要构造一个新的确定性图灵机 (DTM)，它能在多项式时间内判定经过这些运算后产生的新语言。

SAT(布尔可满足性问题)

输入： 一个布尔逻辑公式。

输出： 回答“是”或“否”。回答“是”当且仅当存在一种方式，将公式中的变量赋值为 真 或 假，使得整个公式的最终计算结果为 真。

多项式时间归约

多项式时间归约的核心思想是

如果我们可以将问题A的任何实例，快速地（在多项式时间内）转化成问题B的一个实例，并且这个转换过程保证“A的答案是‘是’当且仅当B的答案是‘是’”，那么就认为 A 可以多项式时间归约到 B。

问题A（找最大值）： 给你一列数字，找出其中最大的一个。
问题B（排序）： 给你一列数字，将它们从小到大排序。

现在我们想解决问题A（找最大值），但我们只有一个能解决问题B（排序）的黑盒子。

归约过程如下：

转换： 你把“找最大值”的输入（那列数字）原封不动地作为“排序”问题的输入。这个转换是瞬间完成的（多项式时间）。

调用黑盒子： 你将这个输入送给排序黑盒子。

输出转换： 排序黑盒子会返回给你一个排序好的列表。你只需要取出这个已排序列表的最后一个元素（即最大的那个），它就是问题A的答案。

Ladner's Theorem（拉德纳定理）

如果 P ≠ NP，那么NP问题中是否只有“容易的”（P）和“最难的吗”（NPC）这两类?

Ladner's Theorem 认为 如果 P ≠ NP，那么存在一类既不在P中，也不是NP完全的问题。这类问题被称为“NP中间问题”。

如果 P ≠ NP，那么存在一个问题 L，使得

L ∈ NP （L是一个NP问题）

L ∉ P （L不是多项式时间可解的）

L 不是 NP完全的 （L不是NP中最难的问题）

Cook–Levin Theorem

Cook–Levin Theorem： 布尔可满足性问题（SAT）是 NP–complete的。

Cook–Levin Theorem 表明 1）SAT本身是一个NP问题 2）NP中的每一个问题L，都存在一个多项式时间归约。

Ladner's Theorem（拉德纳定理）

如果 P ≠ NP，那么NP问题中是否只有“容易的”（P）和“最难的吗”（NPC）这两类?

Ladner's Theorem 认为 如果 P ≠ NP，那么存在一类既不在P中，也不是NP完全的问题。这类问题被称为“NP中间问题”。

如果 P ≠ NP，那么存在一个问题 L，使得

L ∈ NP （L是一个NP问题）

L ∉ P （L不是多项式时间可解的）

L 不是 NP完全的 （L不是NP中最难的问题）

算法的时间与空间复杂度

算法（Algorithm）是指用来操作数据、解决程序问题的一组方法。

如何去衡量不同算法之间的优劣：时间和空间 两个维度。

时间维度：是指执行当前算法所消耗的时间。
空间维度：是指执行当前算法需要占用多少内存空间。

往往，空间维度和时间维度 不能都达到最小，这时候需要看实际要求决定，以哪一个指标为准。

算法的时间复杂度

时间复杂度使用大O 表示法，时间复杂度的公式是： T(n) = O(f(n))

其中f(n) 表示每行代码执行次数之和，而 O 表示正比例关系

常见的时间复杂度量级有：

- 常数阶O(1)
- 对数阶O(logN)
- 线性阶O(n)
- 线性对数阶O(nlogN)
- 平方阶O(n^2)
- 立方阶O(n^3)
- K次方阶O(n^k)
- 指数阶(2^n)

时间复杂度越大，执行效率越低

算法的时间复杂度

常数阶O(1)：无论代码执行了多少行，只要是没有循环等复杂结构，那这个代码的时间复杂度就都是O(1)。

线性阶O(n)： 在for 循环中，代码会执行n遍，因此它消耗的时间是随着n的变化而变化的，因此这类代码都可以用O(n)来表示它的时间复杂度。

对数阶O(logN)：
```
int i = 1;
while(i<n)
{
  i = i * 2;
        } :
```

前面这段代码中，在循环中，i 会 每次乘 2，会越来越接近 n,当循环 log2^n 次以后，代码结束，这个代码的时间复杂度为：O(logn)

算法的时间复杂度

线性对数阶O(nlogN)：将时间复杂度为O(logn)的代码循环N遍，时间复杂度就是 n * O(logN），就是O(nlogN)。

平方阶$O(n^2)$：把 O(n) 的代码再嵌套循环一遍，时间复杂度就是$O(n^2)$。

立方阶$O(n^3)$、K次方阶O(n^k)，同理，可以认为是将O(n) 进行多次嵌套。

算法的空间复杂度

空间复杂度是对一个算法在运行过程中临时占用存储空间大小的一个量度，并不是实际占用的空间，用 S(n) 来定义。

空间复杂度比较常用的有：O(1)、O(n)：

空间复杂度 O(1)：如果算法执行所需要的临时空间不随着某个变量n的大小而变化，即此算法空间复杂度为一个常量，可表示为 O(1)

空间复杂度 O(n)：

```
int[] m = new int[n]
for(i=1; i<=n; ++i)
{
j = i;
j++;
}
```

以上面这段代码为例，第一行生成的一个数组，这个数据占用的大小为n，这段代码里虽然有循环，但是没有新的分配空间，所以代码空间复杂度主要与新数组的大小有关为n.

典型习题讲解

Question 1

Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet {0,1}.

a. $\{w \mid w$ contains an equal number of 0s and 1s$\}$

b. $\{w \mid w$ contains twice as many 0s as 1s$\}$

c. $\{w \mid w$ does not contain twice as many 0s as 1s$\}$

典型习题讲解

Question 2

Answer each part TRUE or FALSE.

a. $2n = O(n)$.

b. $n^2 = O(n)$.

c. $n^2 = O(n \log^2 n)$.

d. $n \log n = O(n^2)$.

e. $3^n = 2^{O(n)}$.

f. $2^{2^n} = O(2^{2^n})$.

Answer each part TRUE or FALSE.

a. $n = o(2n)$.

b. $2n = o(n^2)$.

c. $2^n = o(3^n)$.

d. $1 = o(n)$.

e. $n = o(\log n)$.

f. $1 = o(1/n)$.

典型习题讲解

Question 4

A ***triangle*** in an undirected graph is a 3-clique. Show that $TRIANGLE \in \mathrm{P}$, where $TRIANGLE = \{\langle G \rangle \mid G \text{ contains a triangle}\}$.

构造多项式时间算法

- 三角形： 在无向图中，三个顶点两两之间都有边相连的子图。
- 对于输入 〈G〉，其中 G 是一个有 n 个顶点的图：
  - 从 n 个顶点中选 3 个的组合数: o(n3)

典型习题讲解

Question 5

Call graphs G and H ***isomorphic*** if the nodes of G may be reordered so that it is identical to H. Let $ISO = \{\langle G, H\rangle \mid G \text{ and } H \text{ are isomorphic graphs}\}$. Show that $ISO \in \text{NP}$.

典型习题讲解

Question 6

Show that if P = NP, you can factor integers in polynomial time.