理论计算机科学导引

TCS - 毛宇尘老师班

shrike505

目录

> Reference: introtcs.org

问题与编码

> 一言以蔽之，它（TCS）研究的是问题的上界与下界。

需要界定计算所需要解决的问题，以及计算所需要的设备（模型）。

这一节先规定前者。回顾一些经典的算法或数学上的问题：给定带权重的图 G，求其中的最短路/其的最小生成树；提供矩阵 A, B，求其乘积 AB。这些问题都可以看作一个函数：给定输入，求输出。

与程序设计中的函数强调 implementation（即 How to compute the answer）相比，这里的函数更多具有数学意义，强调 specification（即 What should the answer be）。

接下来聚焦这些函数的输入，计算机无法理解图、矩阵这些概念，只能理解二进制串（binary string），也就是一串又一串的 0 和 1——于是要通过某些编码方式将这些元素编码为 01 串。先定义一个字符表（Alphabet）：$\Sigma = \{0, 1\}$，于是长度为 n 的二进制串的集合可表示为 $\Sigma^n = \Sigma \times \Sigma \times \ldots \times \Sigma = \{(a_1, a_2, \ldots, a_n) \mid a_i \in \Sigma\}$

特别规定 Σ^0 是长度为 0 的串的集合，这个串用 e 表示，即 $\Sigma^0 = \{e\}$

$\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$ 即为所有长度的二进制串集合。

前缀（Prefix）

$x = a_1 a_2 \ldots a_n, y = b_1 b_2 \ldots b_n$ 的拼接（Concatenation）为 $xy = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_n$
x 是 y 的一个前缀（Prefix），当对于某些 $z \in \Sigma^*, y = xz$
类似的可以定义后缀（Suffix），不再赘述

可以将 Σ 中的 0 和 1 换成任意字符，例如 26 字母，方框三角圆，以此组建你自己的 Alphabet!

编码（Encoding）

有了最基础的元素（字符），将图、矩阵、等等等等计算函数的输入转化为字符串的过程，称为编码，即一个映射 $E : A \to \{0, 1\}^*$。

编码性质

显然，这个映射需要是单射（injective，在下文中会频繁表示为 one-to-one），即不同元素的映射结果（得到的字符串）必须是不同的。

例子

- 自然数 $n \in N$（parity(n) 是 n 对 2 取余的结果）：$NtS(n) = \begin{cases} 0 & \text{if } n{=}0 \\ 1 & \text{if } n{=}1 \\ NtS(\lfloor \frac{n}{2} \rfloor)\ \text{parity(n)} & \text{if } n{>}1 \end{cases}$
  - 亦即 n 的二进制表示
- 自然数对 $(a, b) \in N \times N$，自然的想法是 a 的编码拼接 b 的编码，但是会出现编码重复，并不是单射
  - 对于 1110，可以解释为 $(1, 6)$ 和 $(3, 2)$，这实质上是因为在计算机读取完前两个 1 时，并不知道它代表 3 还是一个其他数的前缀

prefix-free 编码

在第二个例子的教训下，我们需要找到的编码映射是 prefix-free 的，即对于任何的 $x \neq x'$，$E(x)$ 都不是 $E(x')$ 的前缀。

接下来 myc 老师突然就这个 prefix-free 证明了两个*寻找另一种编码*的引理，感觉很突兀。

引理 1

假设已经存在一个 prefix-free 的编码 $E : A \to \{0,1\}^*$，那么对于编码 $\bar{E} : A^* \to \{0,1\}^*$（$A^* = \bigcup_{n \geq 0} A^n$，我理解为一个由*任意长待映射元素*（$a_i \in A$）*序列组成的集合*，接下来要找到对这些元素序列的编码），命 $\bar{E}(a_1 a_2 ... a_n) = \begin{cases} E(a_1)E(a_2)...E(a_n) & \text{if } n{\geq}1 \\ e & \text{if } n{=}0 \end{cases}$，那么 $\bar{E}$ 是 one-to-one 的。

证明. 假设存在 $(a_1, a_2, ..., a_n) \neq (b_1, b_2, ..., b_m)$，使得 $\bar{E}(a_1 a_2 ... a_n) = \bar{E}(b_1 b_2 ... b_m)$，那么 $E(a_1)E(a_2)...E(a_n) = E(b_1)E(b_2)...E(b_m)$，且 $\exists i, s.t. \forall j < i, a_j = b_j$，且 $a_i \neq b_i$（即在第 i 个字符前两个元素序列的每个元素都相同）

那么 $E(a_1)E(a_2)...E(a_{i-1}) = E(b_1)E(b_2)...E(b_{i-1})$，则 $E(a_i)...E(a_n) = E(b_i)...E(b_m)$，那么对于 $E(a_i)$ 和 $E(b_i)$，要么前者是后者的前缀，要么后者是前者的前缀，又考虑到 $a_i \neq b_i$，则 E 不是 prefix-free 的，这与题设冲突。 □

引理 2

如果存在 one-to-one 的 $E : A \to \{0,1\}^*$，那么存在 prefix-free 的 $E' : A \to \{0,1\}^*$，且使得 $|E'(a)| \leq 2|E(a)| + 2, \forall a \in A$

证明. 将原编码中的 0 映射为 00，1 映射为 11，该元素再次编码结束后再添加一个 01，例如对于 $E(a) = 010$，$E'(a) = 00110001$

这种编码显然有性质 0：01 不会出现在任何编码的奇数-偶数位置，即 $\forall k \in N, E'(a_{2k+1})E'(a_{2k+2}) \neq 01$

试证 prefix-free 性：假设 $E'(a)$ 是 $E'(b)$ 的前缀，由于 01 标识了 $E'(a)$ 和 $E'(b)$ 的结束，且由于 性质 0，很明显有 $E'(a) = E'(b)$，那么 $E(a) = E(b)$，且 E 是单射，则 $a = b$，于是 prefix-free 得证。 □

结合两个引理可以得到一个结论：

定理

如果存在 one-to-one 的 $E : A \to \{0,1\}^*$，那么便存在 one-to-one 的 $E' : A^* \to \{0,1\}^*$

这是很重要的，对于数学元素，如果我们可以给数字做编码，那么就可以编码向量，再运用一次定理就能编码矩阵，然后是更高维的张量。

编码与可数的关系

下面四条等价

1. A 是可数的
2. A 是有限的，要么存在一个双射 $f : A \to N$
3. 存在单射 $g : A \to N$
4. 存在满射 $h : N \to A$

引理 3

$\{0,1\}^*$ 是可数的

证明. 对 $\{0,1\}^*$ 的元素进行这样的排序：e,0,1,00,01,10,11⋯

即先按长度排序，内部再按二进制数大小排序。

那么定义从字符串映射到排序后序号的函数 $f : \{0,1\}^* \to N$： $\forall x \in \{0,1\}^*, f(x) = \begin{cases} 0 & \text{if } x{=}e \\ 2^{|x|}+(x\ \text{对应的二进制数大小}) & \text{if } |x| \geq 1 \end{cases}$

可理解为组的序号+组内的序号，这是一个单射，于是可数。 □

这个引理也引导出一个定理，即可数性和单射编码的关系：

定理

A 是可数的当且仅当存在单射 $E : A \to \{0,1\}^*$

编码的设定完备后，我们面临的问题（Problem）就抽象为了一个从二进制串到二进制串的函数了（即对输入和输出都做编码）

计算模型

布尔电路（Boolean Circuit）

可以可以，问题（Problem）的输入输出已经被我们编码，可以投诸于计算了——那么，计算的具体步骤，或者说方法，是什么呢?

> 这一节里探讨的一类问题/函数统称为有限函数（Finite Function），其输入输出均为**固定**长度的一个二进制串，即 $f : \{0,1\}^n \to \{0,1\}^m$

一个很经典的有限函数计算模型即布尔电路（Boolean Circuit，我去，计算机系统 1），显然对于与门（AND）和或门（OR）而言，$n = 2, m = 1$，非门的 n 为 1，别的门电路可类比；另一个例子是 MAJ 函数，即对于 $n = 3$ 的输入的每一位，如果 1 占多数，则输出 1，否则输出 0

一个『电路』还是太具体了，不利于更本质的计算理解——将一个布尔电路 C 抽象为一个有向无环图 G，它包含如下节点（Nodes）：

- n input nodes：记为 $X[0], X[1], ..., X[n-1]$，均没有入度且出度至少为 1
- s gates：即逻辑门，根据种类有不同的度
  - 一个电路 C 的 size 定义为 $|C| = s$
- m output nodes：记为 $Y[0], Y[1], ..., Y[1]$

舒服了，可以利用门电路模拟从输入到输出的计算过程了：对于输入长为 n 的 $x \in \{0,1\}^n$，令其第 i 位即为 input nodes 中的 $X[i-1]$，输出答案 $y \in \{0,1\}^m$ 同理。

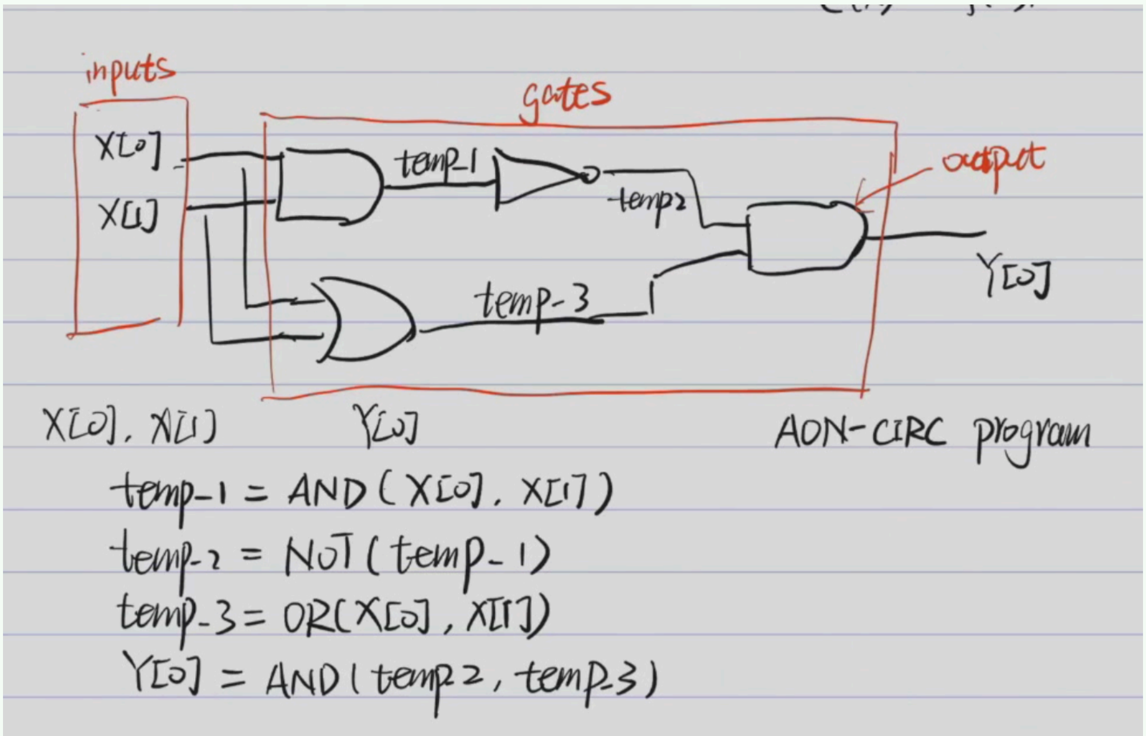
定义 1

记 $C(X) = (Y[0], Y[1], ..., Y[m-1])$，如果 $\forall x \in \{0,1\}^n, C(x) = f(x)$, 则称电路 C 计算了函数 f.

♣

现在还亟待一种对计算过程的书面化描述，即，这个电路的每一步，信号（或者说数据）流过每一个门时得到了什么中间结果？于是 ~~Anonjac~~ ~~Anon-Circ~~ AON-Circ（AND-OR-NOT Circuit）Program 登场了。

定义 2 (AND-OR-NOT Circuit Program)



对于图中的电路，将其每个中间逻辑门的输出保存为一个 temp 变量，便得到了下方的多行（Lines），这就是 AON-Circ Program

不难发现每一行对应一个逻辑门的计算过程，于是 Program 的行数与其对应电路中逻辑门的个数相同。此时 Program 的行数即为电路的 size

AON-C Program 计算了某个函数的定义与上方电路 C 计算函数类似，不再赘述。♣

实际上程序和电路这种对应关系就是等价的。

定理 3

一个函数可被一个有 s 个逻辑门的布尔电路计算，当且仅当它可以被 s 行的 AON-C Program 计算。♡

NAND 电路（NAND Circuit）

接下来介绍一种门：NAND，即 NOT(AND)，易知 AON 三者都可以只用 NAND 实现，于是可以搭建一个仅由 NAND 门构成的电路。

NAND Circuit	$\Leftrightarrow$	AON Circuit
s gates	$\rightarrow$	$\leq 2s$ gates, for NAND decomposes to NOT(AND)
$\leq 3s$ gates,for OR(a,b) $\equiv$ NAND(NAND(a,a),NAND(b,b))	$\leftarrow$	s gates

相类似的，有 NAND-Circ Program，其每一行都形如 `foo = NAND(bar,blah)`。

定理 4

Boolean Circuit, AON-Circ Program, NAND-Circ Program,NAND Circuit 四者的转化只需要 s 的常数倍（即 $\Theta(s)$）

定理 5

$\forall n, m, f : \{0,1\}^n \to \{0,1\}^m$，都存在一个布尔电路计算这个函数，其含有 $O(m \cdot n \cdot 2^n)$ 个逻辑门。

证明. 对于输出的某一位 $Y[j]$，其计算情况可枚举如下表：

$X[0]$	$\cdots$	X[n-1]	Y[j]
0	$\cdots$	0	some value
0	$\cdots$	1	some value
$\cdots$	$\cdots$	$\cdots$	$\cdots$
1	$\cdots$	1	some value

写出 $Y[j]$ 的具体计算式, 用析取范式表示: $Y[j] = (\ldots \wedge \ldots \wedge \ldots) \vee (\ldots \wedge \ldots \wedge \ldots) \vee \ldots \vee (\ldots \wedge \ldots \wedge \ldots)$，其中共有 2^n 个括号，对应上表中的 2^n 行，每一行通过合取计算出一种情况下的 $Y[j]$，再析取得到 $Y[j]$ 的具体表达；每个括号中共有 n 项，对应 $X[0]$ 到 $X[n-1]$ 本身或取反再进行合取，于是得到 $n \cdot 2^n$

而 Y 的长度为 m，于是得到 $O(m \cdot n \cdot 2^n)$ □

满足上述定理，即可以计算任意函数的电路，称为**通用**（universal）的；要判断一个函数集合（化成的电路）是否是通用的，只需要判断其是否能计算 NAND。

计算规模

可以可以，已经可以用电路/程序计算某个函数/问题了，那么对于某个函数，我们需要多少个门的电路, 多少行的程序, 这是可以估量的吗? 由上面的定理似乎已经有了上界: $O(m \cdot n \cdot 2^n)$ (myc 剧透：其实可以把数量打到 $O(\frac{m \cdot 2^n}{n})$)。我们来从 NAND 电路入手。

ADD Function

试设计电路程序，计算 $\mathrm{ADD}:\{0,1\}^{2n}\to\{0,1\}^{n+1}$，$\mathrm{ADD}(x_0,\ldots,x_{2n-1})=x_0\ldots x_{n-1}+x_n\ldots x_{2n-1}$

解.

```
def ADD(X[0], ..., X[2n-1]):
  Result = [0] * (n+1)
  Carry = [0] * (n+1)
  for i in range(n):
    Result[i] = XOR(Carry[i], XOR(X[i], X[i+n]))
    Carry[i+1] = MAJ(Carry[i], X[i], X[i+n])
  Result[n] = Carry[n]
  return Result
```

XOR 和 MAJ 函数只需要常数行的 NAND-Circ Program 实现，经过 n 次循环，于是该加法函数的规模（行数）即为 $O(n)$

这是很好的，计算规模与输入串的长度成正比。

MUL Function

乘法基于加法，摆了。

规模可以不断优化：$O(n^2)\to O\left(n^{\log_2 3}\right)\to$ even better.

LOOKUP Function

试设计电路程序，计算 $\mathrm{LOOKUP}:\{0,1\}^{2^k+k}\to\{0,1\}$，具体而言，输入分为两段：$2^k$ 位作为表，k 位视作一个最大可表达 2^k-1 的二进制数 i，输出为表中的第 i 位。

解. 利用归纳思想，$k=1$ 时：

```
def LOOKUP_1(X[0],X[1],i[0]):
 if i[0] == 0:
  Y[0] = X[0]
 else:
  Y[0] = X[1]
```

对于 k 时的情形

```
def LOOKUP_k(X[0],X[1],...,X[2^k-1],i[0],...,i[k-1]):
 if i[0] == 0: # 根据剩下的 i 查 X 的后半段
  Y[0] = LOOKUP_(k-1)(X[2^(k-1)],...,X[2^k-1],i[1],...,i[k-1])
 else:
  Y[0] = LOOKUP_(k-1)(X[0],...,X[2^(k-1)-1],i[1],...,i[k-1])
```

于是有规模 $\begin{cases}L(k)=C+2L(k-1)\\L(1)=O(1)\end{cases}$，解得 $O(2^k)$