算法设计与分析

贪心算法的应用（二）

1. 霍夫曼编码
2. 单源最短路径问题
   Dijkstra 算法
3. 最小生成树
   Kruskal 算法
   Prim 算法

大纲

1. 霍夫曼编码
2. 单源最短路径问题
   Dijkstra 算法
3. 最小生成树
   Kruskal 算法
   Prim 算法

编码的动机

设 Γ 为包含 n 个符号的字母表，符号 i 的频率为 f_i。显然，$\sum_{i=1}^{n} f_i = 1$。
在实践中，为了便于在数字世界中传输（提高效率和鲁棒性），我们使用编码将符号（文件中的字符）编码为码字，然后传输。

$$\begin{array}{ccccc} \text{编码} & \longrightarrow & \text{传输} & \longrightarrow & \text{解码} \\ \Gamma & \longrightarrow & C & \longrightarrow & C \longrightarrow \Gamma \end{array}$$

什么是好的编码方案?

- 效率：高效的编码/解码 & 低带宽
- 其他关于鲁棒性的良好性质，例如纠错

定长编码 vs. 变长编码

编码信源 (Γ, F) 的两种方法

- 定长编码：每个码字的长度是固定的数
- 变长编码：每个码字的长度可以不同

定长编码 vs. 变长编码

编码信源 (Γ, F) 的两种方法

- 定长编码：每个码字的长度是固定的数
- 变长编码：每个码字的长度可以不同

定长编码看起来很整洁。为什么还要引入变长编码?

定长编码 vs. 变长编码

编码信源 (Γ, F) 的两种方法

- 定长编码：每个码字的长度是固定的数
- 变长编码：每个码字的长度可以不同

定长编码看起来很整洁。为什么还要引入变长编码?

- 如果每个符号以相同的频率出现，那么定长编码是好的。
- 如果 (Γ, F) 是非均匀的，我们可以用更少的比特来表示更频繁的符号 $\Rightarrow$ 更经济的编码 $\Rightarrow$ 低带宽

平均码字长度

设 f_i 为第 i 个符号的频率，ℓ_i 为其码字长度。平均码长表示信源 (Γ, F) 的码字平均长度

$$L = \sum_{i=1}^{n} f_i \ell_i$$

问题：给定信源 (Γ, F)，找到其最优编码（最小化平均码字长度）。

平均码字长度

设 f_i 为第 i 个符号的频率，ℓ_i 为其码字长度。平均码长表示信源 (Γ, F) 的码字平均长度

$$L = \sum_{i=1}^{n} f_i \ell_i$$

问题：给定信源 (Γ, F)，找到其最优编码（最小化平均码字长度）。

等等... 这里还有一个微妙的问题。

无前缀编码

无前缀性质：没有码字可以是另一个码字的前缀。

前缀码（prefix code）$\neq$ 无前缀码（prefix-free code）

无前缀编码

无前缀性质：没有码字可以是另一个码字的前缀。

$$\text{前缀码（prefix code）} \neq \text{无前缀码（prefix-free code）}$$

所有定长编码自然满足无前缀性质。

- 无前缀性质仅对变长编码有意义。

无前缀编码

无前缀性质：没有码字可以是另一个码字的前缀。

$$\text{前缀码（prefix code）} \neq \text{无前缀码（prefix-free code）}$$

所有定长编码自然满足无前缀性质。

- 无前缀性质仅对变长编码有意义。

无前缀编码的应用（考虑二进制编码的情况）

- 非无前缀：编码可能不是唯一可解码的。需要带外信道传输分隔符。
- 无前缀：唯一可解码，解码不需要带外传输

非无前缀编码的歧义性

例：非无前缀编码

$$\langle a, 001\rangle,\ \langle b, 00\rangle,\ \langle c, 010\rangle,\ \langle d, 01\rangle$$

字符串 0100001 的解码是有歧义的

- 解码 1：$01 \mid 00 \mid 001 \Rightarrow (d, b, a)$
- 解码 2：$010 \mid 00 \mid 01 \Rightarrow (c, b, d)$

非无前缀编码的歧义性

例：非无前缀编码

$$\langle a, 001\rangle,\ \langle b, 00\rangle,\ \langle c, 010\rangle,\ \langle d, 01\rangle$$

字符串 0100001 的解码是有歧义的

- 解码 1：$01 \mid 00 \mid 001 \Rightarrow (d, b, a)$
- 解码 2：$010 \mid 00 \mid 01 \Rightarrow (c, b, d)$

问题的精化

如何找到最优的无前缀编码?

无前缀编码的树表示

任何无前缀编码都可以用二叉树表示。

- 所有符号都在叶子节点
  - 这个性质隐含了无前缀性，因为没有叶子节点可以是其他叶子节点的前缀
- 每个码字由从根到叶子的路径生成，将" 左" 解释为 0，" 右" 解释为 1
- 符号 i 的码字长度 $\ell_i \leftarrow$ 叶子节点 i 在树中的深度 d_i

无前缀编码的树表示

任何无前缀编码都可以用二叉树表示。

- 所有符号都在叶子节点
  - 这个性质隐含了无前缀性，因为没有叶子节点可以是其他叶子节点的前缀
- 每个码字由从根到叶子的路径生成，将"左"解释为 0，"右"解释为 1
- 符号 i 的码字长度 $\ell_i \leftarrow$ 叶子节点 i 在树中的深度 d_i

附加好处：解码是唯一的

- 从根开始解码比特串，从左到右读取字符串沿树向下移动。
- 每当到达叶子时，输出相应的符号并返回根。

简单事实

最优无前缀编码对应于一棵满二叉树

- 每个节点要么有零个子节点，要么有两个子节点
- 除叶子外，每个内部节点都有两个子节点

简单事实

最优无前缀编码对应于一棵满二叉树

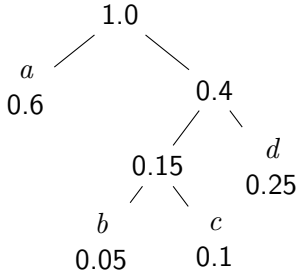
- 每个节点要么有零个子节点，要么有两个子节点
- 除叶子外，每个内部节点都有两个子节点

证明：如果不是，则必存在如右图所示的局部结构，我们可以移除只有一个子节点的节点。

[树结构变换示意图：移除单子节点]

树表示的例子 (1/2)

例：$\langle a, 0.6\rangle$, $\langle b, 0.05\rangle$, $\langle c, 0.1\rangle$, $\langle d, 0.25\rangle$



$$(0.05 + 0.1) \times 3 + 0.25 \times 2 + 0.6 \times 1 = 1.55$$

树的代价：$L(T)$，第 i 个符号在树中的深度：d_i

$$L(T) = \sum_{i}^{n} f_i \cdot d_i = \sum_{i}^{n} f_i \cdot \ell_i = L(\Gamma)$$

树表示的例子 (2/2)

$$(0.05 + 0.1) + 0.15 + 0.25 + 0.4 + 0.6 = 1.55$$

定义内部节点的频率为其两个子节点的和。树的代价是除根以外所有叶子和内部节点频率的和。

- 对于有 $n > 1$ 个叶子的满二叉树，有一个根节点，$n - 2$ 个内部节点

另一种写代价函数的方式：

$$L(T) = \sum_{i}^{2n-2} f_i$$

构建最优满二叉树的贪心算法

以贪心方式构建最优满二叉树：

1. 找到频率最小的两个符号，设为 1 和 2
2. 使它们成为新节点的子节点，新节点的频率为 $f_1 + f_2$
3. 从频率列表中移除 f_1 和 f_2，插入 $(f_1 + f_2)$，然后迭代重复上述步骤

[贪心构建树的示意图]

霍夫曼编码

[David A. Huffman, 1952] 《最小冗余码的构造方法》（A Method for the Construction of Minimum-Redundancy Codes）

- 霍夫曼码是一种特殊类型的最优前缀码，常用于无损数据压缩。

霍夫曼编码

[David A. Huffman, 1952] 《最小冗余码的构造方法》（A Method for the Construction of Minimum-Redundancy Codes）

- 霍夫曼码是一种特殊类型的最优前缀码，常用于无损数据压缩。

霍夫曼方法可以高效实现，如果权重已排序，则在与输入权重数量成线性的时间内找到编码。

霍夫曼编码

[David A. Huffman, 1952] 《最小冗余码的构造方法》（A Method for the Construction of Minimum-Redundancy Codes）

- 霍夫曼码是一种特殊类型的最优前缀码，常用于无损数据压缩。

霍夫曼方法可以高效实现，如果权重已排序，则在与输入权重数量成线性的时间内找到编码。

Shannon 的信源编码定理

熵是理论上可能的最小码字长度的度量

$$\tilde{H}(\Gamma) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i}$$

霍夫曼编码

[David A. Huffman, 1952] 《最小冗余码的构造方法》（A Method for the Construction of Minimum-Redundancy Codes）

- 霍夫曼码是一种特殊类型的最优前缀码，常用于无损数据压缩。

霍夫曼方法可以高效实现，如果权重已排序，则在与输入权重数量成线性的时间内找到编码。

Shannon 的信源编码定理

熵是理论上可能的最小码字长度的度量

$$\tilde{H}(\Gamma) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i}$$

霍夫曼编码非常接近 Shannon 建立的理论极限。

霍夫曼编码的历史

霍夫曼编码因其简单、高速和无专利覆盖而被广泛使用。

- 常被用作其他压缩方法的" 后端"。
- DEFLATE（PKZIP 的算法）和多媒体编码如 JPEG 和 MP3 都有一个前端模型和量化，然后使用前缀码

霍夫曼编码的历史

霍夫曼编码因其简单、高速和无专利覆盖而被广泛使用。

- 常被用作其他压缩方法的" 后端"。
- DEFLATE（PKZIP 的算法）和多媒体编码如 JPEG 和 MP3 都有一个前端模型和量化，然后使用前缀码

  1951 年，*David A. Huffman* 和他 *MIT* 信息论课程的同学们被给予选择：写学期论文或参加期末考试。教授 *Robert M. Fano* 指定了一篇关于寻找最有效二进制码问题的学期论文。*Huffman* 无法证明任何码是最有效的，正要放弃开始准备期末考试时，他想到了使用频率排序二叉树的想法，并很快证明了这种方法是最有效的。
  这样做，*Huffman* 超越了 *Fano*，后者曾与信息论发明者 *Claude Shannon* 合作开发了类似的编码。自底向上构建树保证了最优性，不像自顶向下的 *Shannon-Fano* 编码。

霍夫曼编码的伪代码

算法 1: HuffmanEncoding($S = \{x_i\}, 0 \le f(x_i) \le 1$)

输出： 具有 n 个叶子的编码树

令 Q 为整数（符号索引）的优先队列，按频率排序；
for $i = 1$ to n **do**
    insert(Q, i)；
end for
for $k = n + 1$ to $2n - 1$ **do**
    $i =$ deletemin(Q)，$j =$ deletemin(Q)；
    创建编号为 k 的节点，其子节点为 i, j；
    $f(k) \leftarrow f(i) + f(j)$ //i 是左子节点，j 是右子节点；
    记录三元组 (i, j, k)；
    insert(Q, k)；
end for

每次操作后，队列长度减少 1。当只剩一个元素时，霍夫曼树构建完成。

霍夫曼编码演示

输入：$\langle a, 0.45\rangle$, $\langle b, 0.13\rangle$, $\langle c, 0.12\rangle$, $\langle d, 0.16\rangle$, $\langle e, 0.09\rangle$, $\langle f, 0.05\rangle$

[霍夫曼树逐步构建过程]

初始：$f(5), e(9), c(12), b(13), d(16), a(45)$

↓ 合并 f 和 e 得到 14

↓ 合并 c 和 b 得到 25

↓ 合并 14 和 d 得到 30

↓ 合并 25 和 30 得到 55

↓ 合并 a 和 55 得到 100

编码：$\langle f, 0000\rangle$, $\langle e, 0001\rangle$, $\langle d, 001\rangle$, $\langle c, 010\rangle$, $\langle b, 011\rangle$, $\langle a, 1\rangle$

平均码长：$4 \times (0.05 + 0.09) + 3 \times (0.16 + 0.12 + 0.13) + 1 \times 0.45 = 2.24$

最优无前缀编码的性质：引理 1

引理 1

设 x 和 y 是 Γ 中频率最小的两个符号，则必存在一棵最优满二叉树，使得 x 和 y 是最深层的兄弟叶子节点。

最优无前缀编码的性质：引理 1

引理 1

设 x 和 y 是 Γ 中频率最小的两个符号，则必存在一棵最优满二叉树，使得 x 和 y 是最深层的兄弟叶子节点。

证明概要

- 根据 $|\Gamma|$ 分情况讨论。
- 由编码方案与树的对应关系，只需证明引理所述的树 T 比其他形式的树 T' 更优。

最优无前缀编码的性质：引理 1

引理 1

设 x 和 y 是 Γ 中频率最小的两个符号，则必存在一棵最优满二叉树，使得 x 和 y 是最深层的兄弟叶子节点。

证明概要

- 根据 $|\Gamma|$ 分情况讨论。
- 由编码方案与树的对应关系，只需证明引理所述的树 T 比其他形式的树 T' 更优。

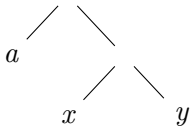
情况 $|\Gamma| = 1$：引理显然成立。只有一种可能：一个根节点。
情况 $|\Gamma| = 2$：引理显然成立。只有一种可能：具有两个叶子的一层满二叉树。

引理 1 的证明（续）

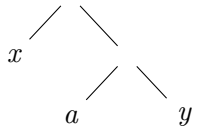
情况 $|\Gamma| = 3$：两种可能 T 和 T'

- T：x, y 都是第二层的兄弟叶子（由算法得到）
- T'：x 和 y 只有一个是第二层的叶子节点

T



T'



$$L(T') - L(T) = f_a \times 2 + f_x - (f_x \times 2 + f_a) = f_a - f_x \geq 0$$

$$\Rightarrow T \text{ 是最优的}$$

引理 1 的证明（续）

情况 $|\Gamma| \geq 4$：考虑以下可能的子情况：

- x 和 y 都不是最深层的兄弟叶子节点：用 (x, y) 替换最深层的兄弟叶子节点 (a, b)（这样的 (a, b) 必存在）得到 T

$$L(T') - L(T) = (d_a - d_x)(f_a - f_x) + (d_b - d_y)(f_b - f_y) \geq 0$$

- x 和 y 在最深层但不是兄弟节点：简单交换得到 T'：

$$L(T) = L(T')$$

- x 和 y 只有一个在最深层，不妨设 x 在最深层且其兄弟是 a，交换 y 和 a 得到 T。类似于 $|\Gamma| = 3$ 的情况，有：

$$L(T') - L(T) \geq 0$$

最优无前缀编码的性质：引理 2

引理 2

设 T 是 (Γ, F) 的无前缀编码二叉树，x 和 y 是两个兄弟叶子节点，z 是它们的父节点。设 T' 是从 T 导出的 $\Gamma' = (\Gamma - \{x, y\}) \cup \{z\}$ 的树，其中对所有 $c \in \Gamma - \{x, y\}$，$f'_c = f_c$，$f_z = f_x + f_y$。则：

$$T' \text{ 是满二叉树}$$
$$L(T) = L(T') + f_x + f_y$$

[从 T 到 T' 的变换：将叶子 x, y 收缩为节点 z]

引理 2 的证明

$\forall c \in \Gamma - \{x, y\}$，有 $d_c = d'_c \Rightarrow$

$$\begin{cases} f_c d_c = f_c d'_c \\ d_x = d_y = d'_z + 1 \\ \Gamma - \{x, y\} = \Gamma' - \{z\} \end{cases}$$

$$\begin{aligned} L(T) &= \sum_{c \in \Gamma} f_c d_c = \left(\sum_{c \in \Gamma - \{x,y\}} f_c d_c \right) + (f_x d_x + f_y d_y) \\ &= \left(\sum_{c \in \Gamma' - \{z\}} f_c d'_c \right) + f_z d'_z + (f_x + f_y) \\ &= L(T') + f_x + f_y \end{aligned}$$

霍夫曼编码的正确性证明

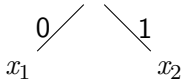
定理

对于所有 $|\Gamma| \geq 2$，霍夫曼算法产生最优无前缀编码二叉树。

证明：数学归纳法
归纳基础：对于 $|\Gamma| = 2$，霍夫曼算法产生最优无前缀编码。
归纳步骤：假设霍夫曼算法编码对规模 k 产生最优无前缀编码，则它对规模 $k+1$ 也产生最优无前缀编码。

归纳基础

$k = 2$，$\Gamma = \{x_1, x_2\}$
任何码字至少需要一个比特。霍夫曼算法产生码字 0 和 1，这是最优无前缀编码。



归纳步骤 (1/3)

假设霍夫曼算法对输入规模 k 产生最优无前缀编码。现在，考虑输入规模 $k+1$，即 $|\Gamma| = k+1$

$$\Gamma = \{x_1, x_2, \ldots, x_{k+1}\}$$

设 $\Gamma' = (\Gamma - \{x_1, x_2\}) \cup \{z\}$，$f_z = f_{x_1} + f_{x_2}$
归纳前提 $\Rightarrow$ 霍夫曼算法为 Γ' 生成最优无前缀编码树 T'，其中频率为 f_z 和 f_{x_i} $(i = 3, 4, \ldots, k+1)$。

归纳步骤 (2/3)

声明：将 (x_1, x_2) 作为 z 的子节点附加到 T'，得到 T（贪心算法的输出），T 是 $\Gamma = (\Gamma' - \{z\}) \cup \{x_1, x_2\}$ 的最优无前缀编码树。

$$[T' \text{ 对应 } \Gamma' \Rightarrow T \text{ 对应 } \Gamma]$$
$$[T^{*'} \text{ 对应 } \Gamma' \Leftarrow T^* \text{ 对应 } \Gamma]$$

归纳步骤 (3/3)

证明：若不是，则存在最优无前缀编码树 T^*，$L(T^*) < L(T)$。
引理 1 $\Rightarrow$ x_1 和 x_2 必须是最深层的兄弟叶子
思路：归约到 T' 对 Γ' 的最优性
从 T^* 中移除 x_1 和 x_2，得到 Γ' 的新编码树 $T^{*'}$。引理 2 $\Rightarrow$

$$\begin{aligned} L(T^{*'}) &= L(T^*) - (f_{x_1} + f_{x_2}) \\ &< L(T) - (f_{x_1} + f_{x_2}) \\ &= L(T') \end{aligned}$$

这与 T' 是 Γ' 的最优无前缀编码树的前提矛盾。

应用：文件归并

问题：给定文件集合 $\Gamma = \{1, \ldots, n\}$，每个文件中的项已排序，$f_i$ 表示文件 i 中的项数。现在，任务是使用二路归并排序将这些文件合并成一个项已排序的单一文件。
将归并过程表示为自底向上的二叉树。

- 叶子节点：标记为 $\{1, \ldots, n\}$ 的文件
- i 和 j 的父节点：值为 $f_i + f_j$（归并后的项数）

二路顺序归并演示

例：$\Gamma = \{21, 10, 32, 41, 18, 70\}$

[顺序归并树的构建过程]

最终树结构：根 192，左子树 104（左 31 右 73），右子树 88（左 18 右 70）

二路归并的复杂度 (1/2)

归并有序数组 $A[k]$ 和 $B[l]$ 为有序数组 $C[k+l]$ 的最坏情况复杂度

- 与归并排序中的归并操作相同
- $W(k,l) \leq k+l-1$

自底向上计算最坏归并复杂度：

$$(21+10-1)+(32+41-1)+(18+70-1)+(31+73-1)+(104+88-1)=483$$

二路归并的复杂度 (2/2)

从 n 个叶子节点计算

$$(21 + 10 + 32 + 41) \times 3 + (18 + 70) \times 2 - 5 = 483$$

最坏情况复杂度

$$W(n) = \left(\sum_{i=1}^{n} f_i d_i \right) - (n - 1)$$

二路归并的复杂度 (2/2)

从 n 个叶子节点计算

$$(21 + 10 + 32 + 41) \times 3 + (18 + 70) \times 2 - 5 = 483$$

最坏情况复杂度

$$W(n) = \left(\sum_{i=1}^{n} f_i d_i \right) - (n - 1)$$

如何证明公式的正确性?

二路归并的复杂度 (2/2)

从 n 个叶子节点计算

$$(21+10+32+41)\times 3+(18+70)\times 2-5=483$$

最坏情况复杂度

$$W(n)=\left(\sum_{i=1}^{n} f_i d_i\right)-(n-1)$$

如何证明公式的正确性?

- 树以自底向上方式生成，必须是满二叉树。每个非叶子节点对应一次归并操作，对 $W(n)$ 的贡献是 -1。
- 内部节点数 $=m$。除叶子节点外，所有内部节点：入度 $=1$，出度 $=2$。

$$\sum \text{出度}=2m,\quad \sum \text{入度}=n+(m-1)\Rightarrow m=n-1$$

最优文件归并

目标：找到一个序列使 $W(n)$ 最小化

- 这个问题本质上与霍夫曼编码相同（除了固定常数 $n-1$）。

解法：将项数作为频率，应用霍夫曼算法生成归并树。

特殊例子：$n = 2^k$ 个文件且每个文件有相同数量的项。在这种情况下，霍夫曼算法生成完美二叉树，这与迭代版本的二路归并排序相同。

- 这也证明了归并排序的最优性。

霍夫曼树归并演示

输入：$\Gamma = \{21, 10, 32, 41, 18, 70\}$

[霍夫曼树归并过程]

排序后：$10, 18, 21, 32, 41, 70$

$\rightarrow$ 合并 10 和 18 得 28 $\rightarrow$ 合并 21 和 28 得 49

$\rightarrow$ 合并 32 和 41 得 73 $\rightarrow$ 合并 49 和 70 得 119

$\rightarrow$ 合并 73 和 119 得 192

代价：$(10 + 18) \times 4 + 21 \times 3 + (70 + 41 + 32) \times 2 - 5 = 456 < 483$

霍夫曼编码回顾

将问题精化为最优无前缀编码
将编码方案建模为构建优化的满二叉树

- 优化函数：树的代价

证明贪心构造是最优的

- 引理 1：证明最优编码树必须满足某种局部结构
- 引理 2：通过对 $|\Gamma|$ 的归纳证明最优性
  - 对 k 最优 $\Rightarrow$ 对 $k+1$ 最优
  - 局部结构对于论证最优性很有用

大纲

1. 霍夫曼编码
2. 单源最短路径问题
   Dijkstra 算法
3. 最小生成树
   Kruskal 算法
   Prim 算法

单源最短路径（SSSP）问题

问题：给定带边权 $e(i,j) \geq 0$ 的有向图 $G = (V, E)$，找到从源节点 $s \in V$ 到 G 中所有节点的最短路径。

[带权图示例]

$$d(1,2) = 5 : \langle 1 \to 6 \to 2 \rangle$$
$$d(1,3) = 12 : \langle 1 \to 6 \to 2 \to 3 \rangle$$
$$d(1,4) = 9 : \langle 1 \to 6 \to 4 \rangle$$
$$d(1,5) = 4 : \langle 1 \to 6 \to 5 \rangle$$
$$d(1,6) = 3 : \langle 1 \to 6 \rangle$$

应用

在现实世界中的广泛应用

- 地图路由
- 接缝雕刻（Seam carving）
- 机器人导航
- 纹理映射
- LaTeX 中的排版
- 城市交通规划
- 电话营销员调度
- 电信消息路由
- 网络路由协议（OSPF、BGP、RIP）
- 通过给定交通拥堵模式的最优卡车路由

大纲

1. 霍夫曼编码
2. 单源最短路径问题
   Dijkstra 算法
3. 最小生成树
   Kruskal 算法
   Prim 算法

Dijkstra 简介

[Edsger Wybe Dijkstra：荷兰计算机科学家、程序员、软件工程师、科学随笔作家*]*

Dijkstra 简介

[Edsger Wybe Dijkstra：荷兰计算机科学家、程序员、软件工程师、科学随笔作家*]*

- 梦想成为联合国代表
- 高中时全 A 学生：被鼓励追求科学
- 大学主修理论物理：当时电子计算机出现，他的父亲送他去剑桥大学参加编程课程
- 投身编程：更多智力挑战，即使是单个比特错误也不允许的严苛事业

Dijkstra 的成就

计算科学最具影响力的人物之一，作为工程师和理论家塑造了这门新学科。

- 1972 年，他成为第一个获得图灵奖的非美国人、非英国人。

Dijkstra 的成就

计算科学最具影响力的人物之一，作为工程师和理论家塑造了这门新学科。

- 1972 年，他成为第一个获得图灵奖的非美国人、非英国人。

他的基础贡献涵盖计算机科学的多个领域：

- 算法、编译器、操作系统、分布式计算（PODC）、编程语言、编程范式和方法论、程序验证、图论...
- 开创了许多全新的研究领域（列举一些标准概念：互斥、死锁、信号量）

Dijkstra 的风格

Dijkstra 选择不引用参考文献的风格以保持他的自主性。
他的教学方法是非传统的

- 在讲课时是一个思维敏捷而深入的思考者
- 从不遵循教科书，可能的例外是他自己正在准备的书
- 布置具有挑战性的家庭作业，并会仔细研究学生的解答
- 口试形式进行期末考试，持续整整一周。每个学生在 Dijkstra 的办公室或家中接受考试，一次考试持续几个小时。

他发明了许多计算机技术但很少使用计算机。

关于 Dijkstra 算法的故事

Dijkstra 必须构建一个演示来推广 ARMAC。

- 在咖啡馆思考：找到荷兰两个城市之间的最短路径可能是个好问题。
- 在 20 分钟内设计了算法（也许是他职业生涯中最伟大的成就）
- 3 年后发表，如今被广泛使用，但在当时不被数学家承认。

据后来的采访，这个算法如此简单是因为咖啡馆里没有纸和笔：迫使他避免复杂的设计而追求简单。

Dijkstra 的大部分工作都是简单、高效和优雅的，源于他母亲的教导。资源的稀缺往往激发最好的创造力。

Dijkstra 算法

直觉：（广度优先搜索）一步一步探索未知世界，每一步的认知都是正确的。
贪心方法：维护一个已探索节点集合 S，算法已确定从 s^* 到这些节点的最短路径距离，以及未探索节点集合 U。

1. 初始化 $S = \emptyset$，$d(s^*, s^*) = 0$，$d(s^*, v) = \infty$；$U = V$
2. 重复选择未探索节点 $u \in U$，满足

$$\min_{u \in U} d(s^*, u)$$

   将 u 加入 S
   对于剩余节点 $v \in U$，如果右边更短则更新 $d(s^*, v) = d(s^*, u) + e(u, v)$，并设置 $\mathsf{prev}(v) = u$
3. 当 $S = V$ 时结束，即 $U = \emptyset$。

实现细节

数据结构：

- s^*：源点
- S：已探索的节点
- U：未探索的节点
- $d(s^*, v)$：从 s^* 到 v 的当前最短路径长度。如果 $v \in S$，则它是最短路径的最终长度。
- $\text{prev}(v)$：v 的前驱节点，用于追踪路径

关键数据结构：优先队列 $\Rightarrow$ 追踪使 $\min_{u \in U} d(s^*, u)$ 的 u

- 假设 u 被加入 S 且有一条边 (u, v) 从 u 到 $v \in U$。则只需更新：

$$d(s^*, v) = \min\{d(s^*, v), d(s^*, u) + e(u, v)\}$$

- 因此，对于每个 $v \notin S$，$d(s^*, v)$ 只能减少，因为 S 只增加并包含更多的 u。

优先队列

优先队列：一种抽象数据类型，其中每个元素都有一个" 优先级"。优先级最高（或最低）的元素总是首先被服务。堆是实现它们的最有效方式。它维护一组具有关联数值（表示优先级）的元素（即键），并支持以下操作。

- **Make-queue**：从给定元素及其给定值构建优先队列。
- **Insert**：向集合添加新的元素-值对。
- **Decrease-value**：适应特定元素值的减少。
- **Delete-min**：返回具有最小值的元素，并将其从集合中移除。

SSSP 的 Dijkstra 算法

算法 2: Dijkstra$(G = (V, E), s^*)$

1: $S = \emptyset$， $d(s^*, s^*) = 0$， $U = V$;
2: **for** $u \in U - \{s^*\}$ **do**
3: $\quad d(s^*, u) = \infty$， prev$(u) = \perp$;
4: **end for**
5: $Q \leftarrow$ makequeue(V, d)
6: **while** $S \neq V$ **do**
7: $\quad u =$ deletemin(Q)， $S = S \cup \{u\}$， $U = U - \{u\}$;
8: $\quad$ **for all** $v \in U$ **do**
9: $\quad\quad$ **if** $d(s^*, v) > d(s^*, u) + e(u, v)$ **then**
10: $\quad\quad\quad d(s^*, v) = d(s^*, u) + e(u, v)$;
11: $\quad\quad\quad$ prev$(v) = u$;
12: $\quad\quad\quad$ `decreasekey`(Q, v)
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: **end while**

Dijkstra 算法对有向图和无向图都正确工作，只要没有负边。

Dijkstra 算法：哪种优先队列?

优先队列操作的代价：$|V|$ 次插入，$|V|$ 次 delete-min，$O(|E|)$ 次 decrease-value（想想为什么）

性能：严重依赖优先队列实现

- 数组实现对稠密图最优
- 二叉堆对稀疏图快得多
- 4 叉堆在性能关键场合值得努力
- Fibonacci/Brodal 理论上最好，但不值得实现

实现	insert	delete-min	decrease-value	总代价
无序数组	$O(1)$	$O(\|V\|)$	$O(1)$	$O(\|V\|^2)$
二叉堆	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\|E\| \log \|V\|)$
Fibonacci 堆	$O(1)$	$O(\log \|V\|)^\dagger$	$O(1)^\dagger$	$O(\|E\| + \|V\| \log \|V\|)$

† 摊还

Dijkstra 算法演示

输入：$G = (V, E)$，$s^* = 1$，$V = \{1, 2, 3, 4, 5, 6\}$

[带权图示意图]

初始状态：$S = \emptyset$

$$
\begin{aligned}
&d(1,1) = 0, \mathsf{prev}(1) = \perp \\
&d(1,2) = \infty, \mathsf{prev}(2) = \perp \\
&d(1,3) = \infty, \mathsf{prev}(3) = \perp \\
&d(1,4) = \infty, \mathsf{prev}(4) = \perp \\
&d(1,5) = \infty, \mathsf{prev}(5) = \perp \\
&d(1,6) = \infty, \mathsf{prev}(6) = \perp
\end{aligned}
$$

Dijkstra 算法演示（续）

步骤	状态更新
$S = \{1\}$	$d(1,2) = 10,\ d(1,6) = 3$
$S = \{1,6\}$	$d(1,2) = 5,\ d(1,4) = 9,\ d(1,5) = 4$
$S = \{1,6,5\}$	无更新
$S = \{1,6,5,2\}$	$d(1,3) = 12$
$S = \{1,6,5,2,4\}$	无更新
$S = \{1,6,5,2,4,3\}$	完成

最终结果：

- $d(1,2) = 5 : \langle 1 \to 6 \to 2 \rangle$
- $d(1,3) = 12 : \langle 1 \to 6 \to 2 \to 3 \rangle$
- $d(1,4) = 9 : \langle 1 \to 6 \to 4 \rangle$
- $d(1,5) = 4 : \langle 1 \to 6 \to 5 \rangle$
- $d(1,6) = 3 : \langle 1 \to 6 \rangle$

正确性证明：Dijkstra 算法 (1/2)

命题

对于每个节点 $u \in S$，$d(s^*, u)$ 是最短 $s^* \rightsquigarrow u$ 路径的长度。即 Dijkstra 第 $|S|$ 步的结果已经正确（最终结果的一部分）。

证明：对 $|S|$ 进行归纳。
归纳基础：$|S| = 1$，$S = \{s^*\}$，$d(s^*, s^*) = 0$。显然成立。
归纳步骤：假设命题对 $|S| = k \geq 1$ 成立。证明命题对 $|S| = k + 1$ 也成立。

正确性证明：Dijkstra 算法 (2/2)

设 v 为下一个（第 $k+1$ 步）加入 S 的节点，u 为其前驱，有 $d(s^*, v) = d(s^*, u) + e(u, v)$。

正确性：考虑任意 $s^* \rightsquigarrow v$ 路径 P，证明 $\ell(P) \geq d(s^*, v)$。

设 (x, y) 是 P 中第一条离开 S 的边，P' 是到 x 的子路径，则 P 一到达 y 就已经太长了。（x 可能等于 u；y 可能等于 v）

$$
\begin{aligned}
\ell(P) &= \ell(P') + e(x, y) + \ell(y, v) && //P\text{的定义} \\
&\geq d(s^*, x) + e(x, y) + 0 && //\text{归纳假设} \\
&\geq d(s^*, y) && //d\text{的定义} \\
&\geq d(s^*, v) && //\text{因为 Dijkstra 选择了}v\text{而不是}y
\end{aligned}
$$

Dijkstra 算法的扩展

Dijkstra 算法及其证明可扩展到几个相关问题：

- 无向图中的最短路径
- 最大容量路径
- 最大可靠性路径

Dijkstra 算法的最优性

[此处为空白，可能用于课堂讨论]

大纲

1. 霍夫曼编码
2. 单源最短路径问题
   Dijkstra 算法
3. 最小生成树
   Kruskal 算法
   Prim 算法

MST 的动机

现实世界问题：你被要求通过连接它们来组网一组计算机。每条链接都有维护成本。

最便宜的可能网络是什么？

MST 的动机

现实世界问题：你被要求通过连接它们来组网一组计算机。每条链接都有维护成本。

最便宜的可能网络是什么？

这转化为一个图问题：

- 节点 – 计算机
- 无向边 – 潜在链接
- 边的权重 – 维护成本

优化目标：选择足够的边使所有节点连通且总权重最小。

基本分析

这些边有什么性质?

一个直接的观察：最优边集不能包含环，因为从环中移除一条边可以在不影响连通性的情况下降低成本。

事实 1：移除环边不能断开无向图的连通性。

基本分析

这些边有什么性质?

一个直接的观察：最优边集不能包含环，因为从环中移除一条边可以在不影响连通性的情况下降低成本。

事实 1：移除环边不能断开无向图的连通性。

解必须是连通且无环的

- 这种无向图称为树
- 我们称总权重最小的那棵为最小生成树

最小生成树

生成树：设 $G=(V,E)$ 为无向图，w_e 为边 e 的权重。

- 连通且无环的子图 $T=(V,E')$ 称为 G 的生成树。
- 树 T 的权重为 $\text{weight}(T)=\sum_{e\in E'} w_e$
- 最小生成树是使 $\text{weight}(T)$ 最小的树

最小生成树可能不唯一。

最小生成树的例子

[原图：6 个节点 A-F，带权边]
[MST：选中的边集，总权重最小]
你能发现另一棵吗?

树的性质

树的定义

连通且无环的无向图。

结构的简单性使树的概念极其有用

树的事实（节点与边的关系）

事实：具有 n 个节点的树有 $n-1$ 条边。
增加连通节点：从空图开始。初始时 n 个节点彼此断开。

- 第一条边：连接两个节点（剩余 $n-2$ 个节点）
- 然后：一条从现有连通子图出发的边加入，一个节点被连接

总边数：$1+(n-2)=n-1$
减少分离组件：将初始断开的 n 个节点视为 n 个独立组件。

- 当 $\langle u,v\rangle$ 出现时，它合并 u 和 v 之前所在的两个组件。
- 这一步将连通分量总数减少 1。
- 在这个增量过程中，连通分量数从 n 减少到 $1 \Rightarrow$ 必须添加了 $n-1$ 条边

树的性质 1

性质 1

任何连通的无向图 $G = (V, E)$，若 $|E| = |V| - 1$，则是树。

树的性质 1

性质 1

任何连通的无向图 $G = (V, E)$，若 $|E| = |V| - 1$，则是树。

证明思路：使用定义，只需证明 G 是无环的。假设它有环，我们可以运行以下迭代过程使其无环

1. 每次从环中移除一条边
2. 以某个图 $G' = (V, E')$，$E' \subseteq E$ 终止，它是无环的。

操作 $\Rightarrow$ G' 仍然连通 $\Rightarrow$ G' 是树（根据树的定义）
事实 $\Rightarrow |E'| = |V| - 1 \Rightarrow E' = E \Rightarrow G' = G$
没有边被移除，G 一开始就是无环的。

树的性质 1

性质 1

任何连通的无向图 $G = (V, E)$，若 $|E| = |V| - 1$，则是树。

证明思路：使用定义，只需证明 G 是无环的。假设它有环，我们可以运行以下迭代过程使其无环

1. 每次从环中移除一条边
2. 以某个图 $G' = (V, E')$，$E' \subseteq E$ 终止，它是无环的。

操作 $\Rightarrow$ G' 仍然连通 $\Rightarrow$ G' 是树（根据树的定义）
事实 $\Rightarrow |E'| = |V| - 1 \Rightarrow E' = E \Rightarrow G' = G$
没有边被移除，G 一开始就是无环的。

我们可以仅通过计算边数来判断连通图是否是树。

树的性质 2（另一个刻画）

性质 2

无向图 $G = (V, E)$ 是树当且仅当任意一对节点之间有唯一路径。

树的性质 2（另一个刻画）

性质 2

无向图 $G = (V, E)$ 是树当且仅当任意一对节点之间有唯一路径。

正向：

- 在树中，任意两个节点之间只能有一条路径；因为如果有两条路径，这些路径的并集将包含一个环。

树的性质 2（另一个刻画）

性质 2

无向图 $G = (V, E)$ 是树当且仅当任意一对节点之间有唯一路径。

正向：

- 在树中，任意两个节点之间只能有一条路径；因为如果有两条路径，这些路径的并集将包含一个环。

反向（定义：连通 + 无环 $\Rightarrow$ 树）：

- G 在任意两个节点之间有路径 $\Rightarrow$ G 是连通的
- 如果这些路径是唯一的，则 G 是无环的（因为环中任意一对节点之间至少有两条路径）。

简要总结

上述性质给出了判断无向图是否是树的三个标准

1. **定义**：连通且无环
2. **性质 1**：连通且 $|V| = |E| + 1$
3. **性质 2**：任意两个节点之间有唯一路径

生成树：命题 1

命题 1

如果 T 是 G 的生成树，$e \notin T$，则 $T \cup \{e\}$ 包含一个环 C。

[示意图：向生成树添加边 e 后形成环]

生成树：命题 2

命题 2

移除环 C 中的任意边（在命题 1 的语境下），得到 G 的另一棵生成树 T'。

[示意图：从环中移除边得到新生成树]

所有节点仍然连通 $+$ 性质 1 $\Rightarrow$ 命题 2

生成树：命题 2

命题 2

移除环 C 中的任意边（在命题 1 的语境下），得到 G 的另一棵生成树 T'。

[示意图：从环中移除边得到新生成树]

所有节点仍然连通 $+$ 性质 1 $\Rightarrow$ 命题 2

这个命题给出了从现有生成树创建新生成树的方法。

MST 的应用

MST 是具有多种应用的基本问题。

- 抖动处理（Dithering）
- 聚类分析
- 最大瓶颈路径
- 实时人脸验证
- 用于纠错的 LDPC 码
- 使用 Rényi 熵的图像配准
- 在卫星和航空图像中寻找道路网络
- 减少蛋白质氨基酸测序中的数据存储
- 湍流流体中粒子相互作用局部性建模
- 网络设计（通信、电气、计算机、道路）
- $\mathcal{NP}$-困难问题的近似算法（如 TSP、Steiner 树）

切割性质

假设在构建 MST 的过程中，我们已经选择了一些边并且目前在正确的轨道上。

下一步应该添加哪条边?

如果有正确的策略，那么我们可以迭代地解决 MST。以下引理给了我们很大的选择灵活性。

切割性质

假设在构建 MST 的过程中，我们已经选择了一些边并且目前在正确的轨道上。

下一步应该添加哪条边?

如果有正确的策略，那么我们可以迭代地解决 MST。以下引理给了我们很大的选择灵活性。

V 的**切割**是 V 的一个划分，比如 $(S, V-S)$。如果 X 中没有边跨越 S 和 $V-S$，则切割与边集 X **相容**。

切割性质

假设边集 X 是 $G=(V,E)$ 的某个 MST 的一部分。设 $(S, V-S)$ 是任何与 X 相容的切割，e 是跨越切割的最轻边。则 $X \cup \{e\}$ 是某个 MST 的一部分。

切割性质保证添加跨越任何切割的最轻边总是安全的，只要它与 X 相容。

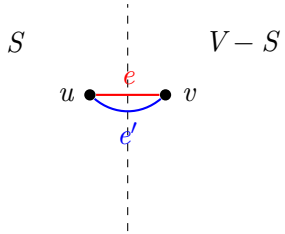
切割性质的证明 (1/2)

假设边集 X 是某个 MST T 的一部分（部分解，在正确轨道上）。

- 如果新边 e 恰好也是 T 的一部分，则无需证明。
- 如果 $e \notin T$，我们可以通过稍微修改 T 来构造一个包含 $X \cup \{e\}$ 的不同 MST T'，只改变它的一条边。
  1. 将 $e = (u, v)$ 加入 T。T 是连通的，它已经有一条 u 和 v 之间的路径，添加 e 创建一个环。
     - ⋆ 这个环必须还有另一条边 e' 跨越切割 $(S, V - S)$。
  2. 移除边 e'，我们剩下 $T' = T \cup \{e\} - \{e'\}$：

     命题 1 + 命题 2 $\Rightarrow$ T' 仍然是生成树

切割性质的证明 (2/2)



还需证明 T' 也是 MST。
证明思路：比较它的权重与 T 的权重

$$\text{weight}(T') = \text{weight}(T) + w(e) - w(e')$$

e 和 e' 都跨越 S 和 $V-S$，且 e 是这种类型的最轻边。

$$w(e) \leq w(e') \Rightarrow \text{weight}(T') \leq \text{weight}(T)$$

T 是 MST $\Rightarrow \text{weight}(T) = \text{weight}(T') \Rightarrow T'$ 也是 MST

切割性质的应用

[示例图：展示边集 X、切割、以及 MST T 和 T']

基于切割性质的通用 MST 算法

算法 3: GeneralMST($G = (V, E)$)：输出由 X 定义的 MST

1: $X = \emptyset$ //目前选择的边；
2: **while** $|X| < |V| - 1$ **do**
3: 选择一个集合 $S \subset V$，使得 X 没有边跨越 S 和 $V - S$ //找到与 X 相容的切割；
4: 令 $e \in E$ 为跨越 S 和 $V - S$ 的最小权重边；
5: $X \leftarrow X \cup \{e\}$；
6: **end while**

接下来，我们描述遵循此模板的两个著名 MST 算法。

大纲

1. 霍夫曼编码
2. 单源最短路径问题
   Dijkstra 算法
3. 最小生成树
   Kruskal 算法
   Prim 算法

Kruskal 算法（逐边）

Joseph Kruskal [美国数学学会，1956]

Kruskal 算法（逐边）

Joseph Kruskal [美国数学学会，1956]

粗略思路：从空图开始，然后根据以下规则从 E 中选择边

重复添加下一条不产生环的最轻边

Kruskal 算法（逐边）

Joseph Kruskal [美国数学学会，1956]
粗略思路：从空图开始，然后根据以下规则从 E 中选择边

重复添加下一条不产生环的最轻边

Kruskal 算法逐边构建树，除了注意避免环之外，只是在当前时刻选择最便宜的边。

- 这是一个贪心算法：它做出的每个决定都是具有最明显直接优势的决定。

X 初始为空，检查每个可能的切割 $(S, V - S)$。

Kruskal 算法演示

[Kruskal 算法演示：按权重从小到大添加边，避免形成环]

[初始图：6 个节点，带权边]

$\rightarrow$ 添加权重 1 的边（C-B）

$\rightarrow$ 添加权重 2 的边（C-D）

$\rightarrow$ 添加权重 3 的边（C-E）或权重 4 的边（B-D）...

$\rightarrow$ 继续直到有 $|V| - 1$ 条边

从切割角度解释 Kruskal 算法

一开始，将 $X = \emptyset$ 视为 n 个不相交的组件。
在任何给定时刻，它已选择的边集 X 对应于：

- 部分解 T'：连通分量的集合，每个分量都有树结构

下一条要添加的边 e 连接其中两个分量，比如 T_1 和 T_2

- 将 T_1 视为 $S \Rightarrow (T_1, V - T_1)$ 形成与 X 相容的切割
  - 在添加 e 之前，T_1 和 T_2 不连通，因此 X 中没有边跨越 T_1 和 T_2
- e 是不产生环的最轻边，它当然是 T_1 和 $V - T_1$ 之间的最轻边

Kruskal 算法隐式地在所有可能的相容切割中搜索最轻的跨越边

实现细节

选择并检查：在每个阶段，算法选择一条边添加到当前部分解中。

- 为此，它需要测试每条候选边 (u, v)，看 u 和 v 是否在不同的分量中 $\Rightarrow$ 否则该边产生环

这个测试隐式地确保 X 与任何形如 $(T_i, V - T_i)$ 的切割组合相容

合并：一旦选择了一条边，相应的分量需要合并。

什么样的数据结构支持这样的操作?

数据结构

将算法状态建模为不相交集合的集合，每个集合包含特定分量的节点。
初始时每个节点是一个独立的分量。

- makeset(x)：创建只包含 x 的单元素集合

重复测试节点对（候选边的端点）是否属于同一集合。

- find(x)：x 属于哪个集合?

每当我们添加一条边时，**合并**两个分量

- union(x, y)：合并包含 x 和 y 的集合

Kruskal 算法的伪代码

算法 4: Kruskal(G): 输出由 X 定义的 MST

按权重对边 E 排序，$X = \emptyset$
for 所有 $u \in V$ **do**
    makeset(u)
end for
while $|X| < |V| - 1$ **do**
    for 所有边 $(u, v) \in E$（按升序） **do**
        从 E 中移除 (u, v)
        if find(u) $\neq$ find(v) **then**
            $X \leftarrow X \cup \{(u, v)\}$
            union(u, v)
            break
        end if
    end for
end while

复杂度分析

- 排序 E 的代价：$O(|E| \log |E|)$；makeset(u) 的代价：$O(|V|)$
- find(x) 的总代价：$2|E| \cdot O(\log |V|)$（与 while 无关）

大纲

1. 霍夫曼编码
2. 单源最短路径问题
   Dijkstra 算法
3. 最小生成树
   Kruskal 算法
   Prim 算法

Prim 算法（逐节点）

首先由捷克数学家 Vojtěch Jarník 于 1930 年发现，后来由 Robert C. Prim 于 1957 年重新发现并发表，Edsger W. Dijkstra 于 1959 年也独立发现。因此被称为 DJP 算法。

粗略思路

1. 初始：$X = \emptyset$，$S = \{u_0\}$，u_0 可以是任意节点
2. 贪心选择：在每一步，选择连接 S 和 $V - S$ 的最轻边 $e_{u,v}$，其中 $u \in S$，$v \in V - S$。将 $e_{u,v}$ 加入 X，将 v 加入 S。
3. 继续此过程直到 $S = V$。

Prim 算法是 Kruskal 算法的流行替代方案，是通用切割算法的另一种实现

- X 初始为空，S 初始为任意节点
- X 总是形成子树，S 是第一步后 X 的顶点集。这种选择使 $(S, V - S)$ 自然构成相对于 X 的相容切割。

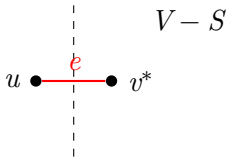
Prim 算法（逐节点）

我们可以等价地认为 S 增长以包含代价最小的 $v^* \notin S$。

$$\text{cost}(v^*) = \min_{u \in S} w(u, v^*)$$

[Prim 算法图示视图]

$T = (S, X)$ 形成一棵树，S 由其顶点组成。



Prim 算法演示

[带权图：6 个节点 A-F]

集合 S	A	B	C	D	E	F
$\{A\}$	–	5/A	6/A	4/A	$\infty/\perp$	$\infty/\perp$
$\{A, D\}$	–	2/D	2/D	–	$\infty/\perp$	4/D
$\{A, D, B\}$	–	–	1/B	–	$\infty/\perp$	4/D
$\{A, D, B, C\}$	–	–	–	–	5/C	3/C
$\{A, D, B, C, F\}$	–	–	–	–	4/F	–

选择的边：$(A, D), (D, B), (B, C), (C, F), (F, E)$

关键数据结构

在每一步，Prim 算法必须找到连接 S 和 $V - S$ 的最轻边。

如何实现这个操作？什么样的数据结构可以帮助?

我们使用优先队列。

Prim 算法的伪代码

算法 5: Prim(G): 输出由数组 prev 定义的 MST

```
1: for 所有 u ∈ V do
2:     cost(u) = ∞, prev(u) = ⊥;
3: end for
4: 选择任意初始节点 u_0: cost(u_0) = 0;
5: Q = makequeue(V);
6: while Q 非空 do
7:     v* = deletemin(Q) //当前 V − S 中离 S 最近的点;
8:     S ← S ∪ {v*};
9:     for 每个 v ∈ V − S do
10:        if w(v*, v) < cost(v) then
11:            cost(v) = w(v*, v), prev(v) = v* //将 v* 加入 S 后更新代价;
12:            decreasekey(Q, v);
13:        end if
14:    end for
15: end while
```

Prim 算法的正确性证明：数学归纳法

命题

对于任意 $k < n$，存在一个 MST 包含 Prim 算法在前 k 步选择的边。

Prim 算法每步选择一条边，共选择 $n-1$ 条边。因此，该命题证明了 Prim 算法的正确性。

证明概要：对步数进行数学归纳。

归纳基础：$k = 1$，存在一个 MST T 包含 $e_{u,i}$，其中 $e_{u,i}$ 是连接节点 u 的最小权重边。

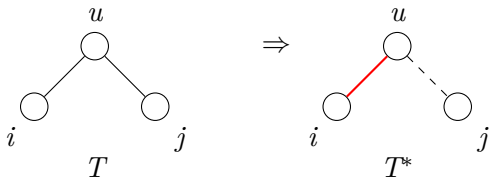
归纳步骤：假设前 k 步选择的边构成某个 MST 的子集，前 $k+1$ 步也是如此。

归纳基础

声明：存在一个 MST T 包含最小权重边 $e_{u,i}$。

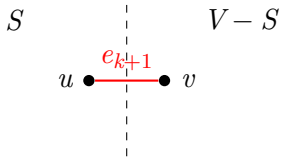
证明：设 T 为一个 MST。如果 T 不包含 $e_{u,i}$，则 $T \cup \{e_{u,i}\}$ 必包含一个环，且该环有另一条边 $e_{u,j}$ 连接节点 u。用 $e_{u,i}$ 替换 $e_{u,j}$ 得到 T^*，T^* 也是生成树。

- 如果 $e_{u,i} < e_{u,j}$，则 weight$(T^*) <$ weight(T)。这与 T 是 MST 的假设矛盾。
- 如果 $e_{u,i} = e_{u,j}$，则 weight$(T^*) =$ weight(T)。那么 T^* 是包含 $e_{u,i}$ 的 MST。



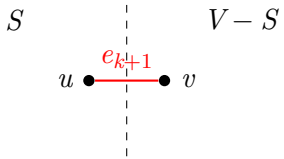
归纳步骤 (1/2)

经过 k 步后，Prim 算法选择了边 $e_1, e_2, \ldots, e_k$。这些边的节点形成节点集 S。
前提：$\exists$ MST $T = (V, E)$ 包含 $(e_1, \ldots, e_k)$。
设第 $k+1$ 步的选择是 $e_{k+1} = (u, v)$，$u \in S$，$v \in V - S$。



归纳步骤 (1/2)

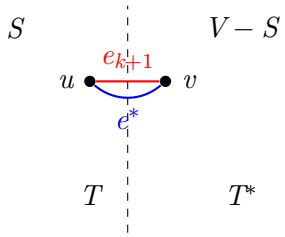
经过 k 步后，Prim 算法选择了边 $e_1, e_2, \ldots, e_k$。这些边的节点形成节点集 S。
前提：$\exists$ MST $T = (V, E)$ 包含 $(e_1, \ldots, e_k)$。
设第 $k+1$ 步的选择是 $e_{k+1} = (u, v)$，$u \in S$，$v \in V - S$。



情况 $e_{k+1} \in E$：Prim 算法在第 $k+1$ 步的归纳步骤显然正确。

归纳步骤 (2/2)

情况 $e_{k+1} \notin E$: 将 e_{k+1} 加入 E 会在 (u, v) 之间创建一个环。在这个环中，$\exists$ 另一条边 e^* 连接 S 和 $V - S$。



设 $T^* = (E - \{e^*\}) \cup \{e_{k+1}\}$，则 T^* 也是 G 的生成树，它由 $e_1, \ldots, e_k, e_{k+1}$ 组成。

- 如果 $e_{k+1} < e^*$，则 weight$(T^*) <$ weight(T)。这与 T 是 MST 的假设矛盾。
- 如果 $e_{k+1} = e^*$，则 weight$(T^*) =$ weight(T)。T^* 也是 MST，$k+1$ 步的输出仍是 T^* 的子集。

Kruskal 算法 vs. Prim 算法

Kruskal 算法

- 初始状态：$X = \emptyset$，$V(\text{MST}) = \emptyset$
- MST 的增长：X 总是形成最终 MST 的子图
- 切割性质：尝试所有与 X 相容的可能切割
- 数据结构：并查集

Prim 算法

- 初始状态：$X = \emptyset$，$V(\text{MST}) = \forall u$
- MST 的增长：X 总是形成最终 MST 的子树
- 切割性质：选择由 X 确定的特定切割（S 初始为任意顶点，然后是 X 的顶点集）
- 数据结构：优先队列

本讲总结

贪心算法：适用于组合优化问题：简单高效

- 逐步构建解
- 总是选择提供最明显和直接收益的下一部分（依赖启发式）

如何（反）证明贪心算法的正确性?（反例）

- 数学归纳法（对算法步数或输入规模）
- 交换论证（逐步将最优解转换为算法解而不影响最优性）

有时贪心算法只给出近似算法

一些经典贪心算法：霍夫曼编码、SSSP、MST