arXiv:2601.13822v1 [cs.DS] 20 Jan 2026

高效并行 $(\Delta+1)$-边着色

Michael Elkin以色列内盖夫本-古里安大学计算机科学系*，贝尔谢巴，以色列。elkinm@cs.bgu.ac.il

Ariel Khuzman内盖夫本-古里安大学计算机科学系*，贝尔谢巴，以色列。huzmana@post.bgu.ac.il

摘要

我们研究并行（PRAM）计算模型中的 $(\Delta+1)$-边着色问题。著名的维津定理 [Viz64] 指出，每个简单图 $G=(V,E)$ 都可以进行正确的 $(\Delta+1)$-边着色。在一篇开创性论文中，Karloff和Shmoys [KS87] 设计了一种并行算法，其运行时间为 $O\left(\Delta^5\cdot\log n\cdot\left(\log^3 n+\Delta^2\right)\right)$，使用 $O(m\cdot\Delta)$ 个处理器。Liang 等人 [LSH96] 将这一结果改进为运行时间 $O\left(\Delta^{4.5}\cdot\log^3\Delta\cdot\log n+\Delta^4\cdot\log^4 n\right)$，使用 $O\left(n\cdot\Delta^3+n^2\right)$ 个处理器。 [LSH96] 声称运行时间为 $O\left(\Delta^{3.5}\cdot\log^3\Delta\cdot\log n+\Delta^3\cdot\log^4 n\right)$，但我们指出了其分析中的一个缺陷；纠正后即可得到上述界限。我们为这一基础问题设计了一种更快的并行算法。具体而言，我们的算法运行时间为 $O\left(\Delta^4\cdot\log^4 n\right)$，使用 $O(m\cdot\Delta)$ 个处理器。我们算法的另一种变体在任意小的 $\delta>0$ 下，运行时间为 $O\left(\Delta^{4+o(1)}\cdot\log^2 n\right)$，使用 $O\left(m\cdot\Delta\cdot\log n\cdot\log^\delta\Delta\right)$ 个处理器。我们还设计了时间与处理器数量之间的其他几种权衡方案，并为森林度较小的图设计了改进算法。在取得这些结果的过程中，我们还为更新 $(\Delta+1)$-边着色提供了一种速度极快的并行算法。对于这一问题，我们的算法远快于且简单于此前最先进的算法（由 [LSH96] 提出）。

*本研究由ISF资助34 13/25。

1 引言

1.1 一般图

给定一个含有 n 个顶点和 m 条边的无向简单图 $G=(V,E)$，若边着色 $\varphi: E \to \mathbb{N}$ 满足对任意一对共享端点的不同边 $e \neq e' \in E$，均有 $\varphi(e) \neq \varphi(e')$，则称其为正确着色。若一个（正确）着色 φ 只使用颜色 $\{1,2,\ldots,t\}$，其中 t 为正整数，则称其为图 G 的（正确）t-边着色。著名的维津定理 [Viz64]指出，任意（简单）图 G都存在 $(\Delta+1)$-边着色，其中 Δ 是图 G 的最大度。在各种计算模型中高效地计算 $(\Delta+1)$-边着色 [MG92, Ber22, BD23, GNK+85, BCSZ25, Ass24, KS87, LSH96, FR96, LHS97, BE11b, BEM17, GHK+20, JK25, CY89, CN90, Kow24] ，是一个极其重要的算法问题。

本文研究并行（PRAM）计算模型中的边着色问题。在一篇经典论文中，Karloff 和 Shmoys [KS87] 设计了一种 $(\Delta+1)$-边着色算法，其运行时间为 $O\left(\Delta^5 \cdot \log n \cdot \left(\log^3 n + \Delta^2\right)\right)$，使用 $O(m \cdot \Delta)$ 个处理器；该算法以目前已知最快的极大独立集（MIS）计算算法[1] [GS89] 作为子程序。Liang 等人 [LSH96]取得了进一步进展。他们的算法需要$O\left(\Delta^{4.5} \cdot \log^3 \Delta \cdot \log n + \Delta^4 \cdot \log^4 n\right)$ 的时间和 $O\left(n \cdot \Delta^3 + n^2\right)$ 个处理器。（事实上，他们声称运行时间为 $O\left(\Delta^{3.5} \cdot \log^3 \Delta \cdot \log n + \Delta^3 \log^4 n\right)$，但我们认为他们的论证存在缺陷（见附录 A）。修正后，其运行时间即为上文所述。） [LHS97]则给出了一种不同的权衡：其算法的运行时间为 $O\left(\Delta^9 \cdot \log^2 n\right)$，但需要使用 $O(m \cdot \Delta)$ 个处理器。

我们显著改进了此前的界，并设计了一系列（$\Delta+1$）边着色算法，其各自的运行时间和处理器数量总结于表 1。具体而言，我们算法的一个变体的运行时间为 $O\left(\Delta^4 \cdot \log^4 n\right)$，使用 $O(m \cdot \Delta)$ 个处理器。对于 $\Delta = \omega\left(\log^6 n\right)$，我们有 $\Delta^4 \cdot \log^4 n \ll \Delta^{4.5} \cdot \log^3 \Delta \cdot \log n$，也就是说，在该范围内，我们的运行时间优于 [LSH96] 。该算法使用的处理器数量（$O(m \cdot \Delta)$）也少于 [LSH96] 所用的 $O\left(n \cdot \Delta^3 + n^2 \cdot \log \Delta\right)$ 个处理器。（事实上，$n \cdot \Delta^3 \geq m \cdot \Delta^2 \gg m \cdot \Delta$。）对于多对数级的 Δ，即 $\Delta = \log^{O(1)} n$，可以使用我们算法的另一种变体，其运行时间为 $\Delta^{4+o(1)} \cdot \log^2 n$，并使用 $O\left(m \cdot \Delta \cdot \log^{\delta} \Delta \cdot \log n \log(\Delta \cdot \log n)\right)$ 个处理器。在此范围内，我们的运行时间为 $\Delta^4 \cdot \log^{2+o(1)} n$，处理器数量为 $\tilde{O}(n)$；而 [LSH96] 的算法运行时间为 $O\left(\Delta^4 \cdot \log^4 n\right)$，使用 $O\left(n^2\right)$ 个处理器。我们算法的另一个变体的运行时间为 $O\left(\Delta^6 \cdot \log^2 \Delta \cdot \log n\right)$，并使用 $O(m \cdot \Delta)$ 个处理器。该运行时间严格优于 $O\left(\Delta^9 \cdot \log^2 n\right)$，后者是 [LHS97] 的运行时间。（后者与前面提到的 [LSH96] 的结果不可比较。） [LHS97] 算法的处理器数量也是 $O(m \cdot \Delta)$，与我们算法的这一变体相同。

[1] 请参阅定义12 以了解独立集的定义。

算法	运行时间	处理器数量
[KS87]	$O\left(\Delta^5 \cdot \text{对数}\, n \cdot \left(\text{对数}^3 n + \Delta^2\right)\right)$	$O(m \cdot \Delta)$
[LSH96]	$O\left(\Delta^{4.5} \cdot \text{对数}^3 \Delta \cdot \text{对数}\, n + \Delta^4 \cdot \text{对数}^4 n\right)$	$O\left(n \cdot \Delta^3 + n^2\right)$
[LHS97]	$O\left(\Delta^9 \cdot \text{对数}^2 n\right)$	$O(m \cdot \Delta)$ (*)
Ours	$O\left(\Delta^4 \cdot \log^4 n\right)$	$O(m \cdot \Delta)$
Ours	$O\left(\Delta^4 \cdot \text{对数}^2 n + \Delta^6 \cdot \text{对数}^2 \Delta \cdot \text{对数}\, n\right)$	$O(m \cdot \Delta)$
Ours	$O\left(a^2 \cdot \Delta^4 \cdot \text{对数}\, \Delta \cdot \text{对数}^2 n\right)$	$O(m \cdot \Delta)$
Ours	$O\left(\Delta^{3+o(1)} \cdot a^{1+o(1)} \cdot \text{对数}^2 n\right) = O\left(\Delta^{4+o(1)} \cdot \text{对数}^2 n\right)$	$O\left(m \cdot \Delta \cdot \text{对数}^{\delta} \Delta \cdot \text{对数}\, n \, \text{对数}(\Delta \cdot \text{对数}\, n)\right)$ (**)
Ours	$O\left(\Delta^5 \cdot \text{对数}^2 n\right)$	$O\left(m \cdot \left(\Delta \cdot \text{对数}\, \Delta + \sqrt{\Delta} \cdot \text{对数}\, n \, \text{对数}(\Delta \cdot \text{对数}\, n)\right)\right)$

表 1：适用于具有 n 个顶点、m 条边，最大度为 Δ、森林度为 a 的图的新型和现有并行 $(\Delta+1)$ 边着色算法汇总。(*) 在 [LHS97]，中，作者声称处理器数量为 $O(n+m)$。(**) 该结果适用于任意常数 $\delta > 0$。

1.2 有界森林度图

森林度 $a = a(G)$ 图 $G = (V, E)$ 的定义为

$$a(G) = \max_{\substack{U \subseteq V \\ |U| \geq 2}} \frac{|E(U)|}{|U| - 1}.$$

根据 Nash-Williams 定理，它等于覆盖 G 的边集 E 所需的边不相交森林的最少数量。有界森林度图是一个重要的图类，包含平面图、有界亏格图、排除任意固定小图的图、具有次线性递归分隔集的图，以及许多其他稀疏图类。Zhou 等人 [ZN94] 设计了一种并行算法，可在 Δ 时间内以 $\tilde{O}(n)$ 的工作量为满足 $\Delta \geq \left\lceil (a+2)_2^{\,2} \right\rceil - 1$ 的图进行 $O\left(\log^3 n\right)$-时间 Δ-边着色，只要 $\Delta = O(1)$ 成立。我们为满足 $a = o(\Delta)$的图提供了改进的并行 $(\Delta+1)$-边着色算法。具体而言，我们算法的一个变体所需时间为 $O\left(a^2 \cdot \Delta^4 \cdot \log \Delta \cdot \log^2 n\right)$，并使用 $O(m \cdot \Delta)$ 个处理器；另一个变体所需时间为 $O\left(\Delta^{3+o(1)} \cdot a^{1+o(1)} \cdot \log^2 n\right)$，并使用 $O\left(m \cdot \Delta \cdot \log^{\delta} \Delta \cdot \log n \log(\Delta \cdot \log n)\right)$ 个处理器。表 1 简要总结了这些算法。

1.3 $(1+\varepsilon)\Delta$-边着色

基于 [LSH96、LHS97], Liang [Lia95] 以及 Furer 和 Raghavachari [FR96] 的 $(\Delta+1)$-边着色算法，针对参数 $\varepsilon > 0$，我们设计了两种 $(1+\varepsilon)\Delta$-边着色算法。[Lia95] 的算法运行时间为 $O\left(\varepsilon^{-4.5} \cdot \log^3 \varepsilon^{-1} \cdot \log n + \varepsilon^{-4} \cdot \log^4 n\right)$，使用 $O\left(n \cdot \varepsilon^{-3} + n^2\right)$个处理器；而 [FR96] 的算法运行时间为 $O\left(\varepsilon^{-9} \cdot \log^2 n\right)$，使用 $O\left(m \cdot \varepsilon^{-1}\right)$ 个处理器。我们还采用改进后的 $(\Delta+1)$-边着色算法，推导出若干种 $(1+\varepsilon)\Delta$-边着色算法。具体而言，我们给出了时间为 $O\left(\varepsilon^{-4} \cdot \log^4 n\right)$、使用 $O\left(m \cdot \varepsilon^{-1}\right)$ 个处理器的算法，或者时间

$O\left(\varepsilon^{-4-o(1)} \cdot \log^2 n\right)$ 时间和 $O\left(m \cdot \varepsilon^{-1} \cdot \log^{\delta}\left(\varepsilon^{-1}\right) \cdot \log n \log\left(\varepsilon^{-1} \cdot \log n\right)\right)$ 个处理器。有关我们实现的其他权衡，请参见表2，其中还简要总结了关于 $(1+\varepsilon)\Delta$-边着色问题的已有界和新界。

算法	运行时间	处理器数量
[Lia95]	$O\left(\varepsilon^{-4.5} \cdot \text{对数}^3 \varepsilon^{-1} \cdot \text{对数} n + \varepsilon^{-4} \cdot \text{对数}^4 n\right)$	$O\left(n \cdot \varepsilon^{-3} + n^2\right)$
[右侧96]+[左侧97] (*)	$O\left(\varepsilon^{-9} \cdot \text{对数}^2 n\right)$	$O\left(m \cdot \varepsilon^{-1}\right)$
Ours	$O\left(\varepsilon^{-4} \cdot \text{对数}^4 n\right)$	$O\left(m \cdot \varepsilon^{-1}\right)$
Ours	$O\left(\varepsilon^{-4} \cdot \text{对数}^2 n + \varepsilon^{-6} \cdot \text{对数}^2 \varepsilon^{-1} \cdot \text{对数} n\right)$	$O\left(m \cdot \varepsilon^{-1}\right)$
Ours	$O\left(\varepsilon^{-4-o(1)} \cdot \text{对数}^2 n\right)$	$O\left(m \cdot \varepsilon^{-1} \cdot \text{对数}^{\delta}\left(\varepsilon^{-1}\right) \cdot \log n \log\left(\varepsilon^{-1} \cdot \text{对数} n\right)\right)$ (**)
Ours	$O\left(\varepsilon^{-5} \cdot \text{对数}^2 n\right)$	$O\left(m \cdot \left(\varepsilon^{-1} \cdot \text{对数} \varepsilon^{-1} + \varepsilon^{-1/2} \cdot \text{对数} n \cdot \text{对数}\left(\varepsilon^{-1} \cdot \text{对数} n\right)\right)\right)$

表2：新算法和现有 $(1+\varepsilon)\Delta$-边着色算法的总结。(*) 对 ε 的依赖在 [FR96] 中是隐含的。他们表明，当 ε 为常数时，运行时间为 $O\left(\log^2 n\right)$，处理器数量为 $O(m+n)$。(**) 该结果对任意常数 $\delta > 0$ 均成立。

1.4 边着色更新问题

LSH 算法中的一个核心组成部分本身也颇具研究价值，即以下动态版本的 $(\Delta+1)$-边着色问题：给定一个带有适当 $(\Delta+1)$-边着色的图 $G=(V,E)$，假设向图中添加一个新顶点 $v \notin V$，并添加至多 Δ 条将其与 G 的现有顶点相连的边。LSH 的动态更新算法可在 $O\left(\Delta^{\frac{3}{2}} \cdot \log^3 \Delta + \Delta \cdot \log n\right)$ 时间内解决此问题，使用 $O\left(n \cdot \Delta + \Delta^3\right)$ 个处理器。我们为这个问题设计了一种截然不同且高效得多的解法。我们的算法可在 $O(\Delta \cdot \log n)$ 时间内解决该问题，使用 $O(n)$ 个处理器。事实上，我们还能做到更进一步：在 $O(\log n)$ 并行时间内处理每一次边更新，使用 $O(n)$ 个处理器。

1.5 技术概述

Karloff-Shmoys算法 Karloff和Shmoys的开创性并行 $(\Delta+1)$-边着色算法 [KS87] 由 $O\left(\Delta^5 \cdot \log n\right)$ 个阶段组成，每个阶段都会为剩余未着色边中的很大一部分（$\Omega\left(\frac{1}{\Delta^5}\right)$）着色。该算法维护一个活跃顶点集合 $A \subseteq V$，即至少关联一条未着色边的顶点。它在这些顶点上构造一个图 G_A：如果两个顶点 u 和 v 在原图 G中的距离至多为 2，则称它们相邻。然后，该算法在 G_A中计算一个极大独立集 I，其基数至少为 $\frac{|A|}{\Delta^2}$。这些顶点并行计算扇（有关扇的定义，请参见第 3.1 节；示意图请参见图1）。扇是 Vizing 的 [Viz64] 算法以及其后所有高效实现中的基本结构（例如参见 [MG92]）。扇由一对可容许颜色刻画，而由同一对颜色刻画的扇具有一项有用性质：它们可以并行处理。（处理一个扇会为至少一条未着色边着色，同时可能会对一些已着色边重新着色。） [KS87] 算法选择能够刻画数量最多的

已构造扇。由于存在 $O(\Delta^2)$ 个可能的可容许颜色对，所选扇的集合大小为 $\Omega\left(\left\lfloor \mathcal{A}_{\Delta^4} \right\rfloor\right)$。随后，这些扇被并行处理。因此，有 $\Omega\left(\left\lfloor \mathcal{A}_{\Delta^4} \right\rfloor\right)$ 条未着色边变为已着色边。由于最初未着色边的数量为 $O(|\mathcal{A}|\cdot\Delta)$，因此可知，在每个阶段中，至少有 $\Omega\left(\frac{1}{\Delta^5}\right)$ 比例的未着色边被着色。每个阶段的 [KS87] 需要 $O\left(\log^3 n+\Delta^2\right)$ 时间，因此总时间为 $O\left(\Delta^5\cdot\log n\cdot\left(\log^3 n+\Delta^2\right)\right)$。

Liang-Shen-Hu 算法 Liang 等人的算法 [LSH96]首先将图拆分为 $\approx\ \Delta$ 个最大度为常数的边不相交子图。每个子图G_1、G_2、.....、G_Δ 都使用 Vizing 着色进行着色（即使用 $\Delta'+1$ 种颜色的着色，其中Δ' 是相应子图的最大度），随后将这些子图两两配对，组成（G_1，G_2）、（G_3，G_4）、.....。每一对子图得到的边着色现在最多有两种冗余颜色，而 [LSH96]算法会从每个这样的边着色中消除一种颜色。接下来，我们得到度更大的$\approx\frac{\Delta}{2}$ 个子图G_1、2、G_3、4、.....，每个子图都使用 Vizing 着色进行着色。随后再次将它们配对、消除一种颜色，依此类推。该算法中最耗时的步骤是颜色消除。考虑一个图 $G=(V,\ E)$，其边采用（$\Delta+2$）-边着色。颜色为 $\Delta+2$ 的边未着色。将这组边（它们构成一个匹配）记为 F。现在，Liang 等人的算法 [LSH96] 在 F上构造一个图，与 [KS87]算法在活跃顶点上构造的图类似，并在该图中计算一个独立集 I_F。我们认为，[LSH96] 构造和分析该图时存在错误（见附录 A），因此他们给出的 $|I_F|$ 下界为 $\Omega\left(\frac{|F|}{\Delta}\right)$；而修正构造后，下界应为 $\Omega\left(\frac{|F|}{\Delta^2}\right)$。随后，[LSH96] 算法围绕 I_F中的边构造扇，并找出其中具有相同颜色对的最大子集。该过程与 [KS87], 算法类似，对这些扇的处理方式也与 [KS87]算法类似。该集合的基数因此为 $\Omega\left(\frac{|I_F|/\Delta^2}{\Delta^2}\right)=\Omega\left(\frac{|I_F|}{\Delta^4}\right)$。（他们错误地声称该基数为 $\Omega\left(\frac{|F|}{\Delta^3}\right)$。）因此，[LSH96] 算法需要 $O(\Delta^4\cdot\log n)$ 个阶段。每个阶段中，他们通过一个复杂的流程构造扇，该流程将问题归约为边着色更新问题（见第 1.4 节），再将后者归约为计算极大节点不相交路径集合的问题，并调用 [GPV93]算法求解后者。这使该步骤的运行时间达到 $\tilde{O}\left(\sqrt{\Delta}\right)$，总复杂度为 $O\left(\Delta^{4.5}\cdot\log^3\Delta\cdot\log n+\Delta^4\cdot\log^4 n\right)$（由于上述错误，他们声称运行时间为 $O\left(\Delta^{3.5}\cdot\log^3\Delta\cdot\log n+\Delta^4\cdot\log^4 n\right)$）。

—

我们的算法 我们的算法采用 [LSH96], 的通用框架：我们同样将图 G 拆分为子图 G_1、.....、G_Δ，将它们两两配对，为每个合并子图 G_1、2、G_3、4、.....计算多出两种颜色的着色，从这些子图的着色中各减少一种颜色，然后将所得子图两两配对，进入下一次迭代，依此类推。我们与 [LSH96] 的第一个不同之处在于定义未着色边集 F 上的图 $G^{(F)}$ 的方式：若两条边 e、$e'\in F$ 之间的距离至多为 2，则它们相连（见方程（1））。该定义保证独立集 $I_F\subseteq F$ 具有这样的性质：对应于 I_F 中两条不同边的两个扇可以并行处理。第二个不同之处在于构造扇的方式。我们设计了一种直接且高效得多的扇构造过程（其解决边更新问题的速度也远快于 [LSH96]）。该过程首先构造图 $G^{(v,u)}$ fan，其中 (v,u) 是一条与未来扇的中心顶点 v 关联的未着色边。该图的顶点集为 $\{u\}\cup\{1,2,\ldots,\Delta+1\}$。顶点 u 通过一条弧 u,β_u 与 u 的任意一个缺失颜色 β_u 相连（即，没有任何与其关联的边使用的颜色）。

它）通过弧 u, β_u 相连。此外，对于任何颜色 α，若一条与 v 关联的边 (v, w) 被着色为 α，我们就将 α （在 $G^{(v,u)}$ fan 中）通过弧 α, β_w 与 w 的一个缺失颜色 β_w 相连。不难看出，该图中的一条极大路径可直接转换为以 u 为中心的极大扇。这样的路径也可以通过并行随机存取机高效计算。采用这种高效的扇构造过程后，我们算法每个阶段运行时间的主导项就是在图 $G^{(F)}$ 中计算极大独立集所需的时间。我们注意到，无须计算极大独立集，只需构造一个大型独立集；为此，我们设计了多种在 $G^{(F)}$ 中构造大型独立集的高效过程。为此，我们将多种分布式顶点着色算法 [BE08、BE11a、BEK14、Bar16、BEG18] 改造为适用于并行环境的算法。这些不同的过程带来了多种权衡，可用于 $(\Delta+1)$-边着色问题。具体而言，如果使用由 [GS89]，提出的最快已知并行确定性极大独立集算法，该算法需要 $O\left(\log^3 n\right)$ 时间和 $O\left(^{n+m}\log\ _n\right)$ 个处理器，那么我们可获得 $O\left(\Delta^4 \cdot \log^4 n\right)$ 时间以及 $O(m \cdot \Delta)$ 个处理器。这已经在广泛的参数范围内改进了此前的界。（ [LSH96] 的算法也使用了 [GS89] 的 MIS 算法。我们对上述 [KS87] 算法的分析也假设将 [GS89] 算法用作子程序。）不过，我们还可以在时间 $O\left(\log^{2+\delta} a \cdot \log\ n\right)$ 内，使用 $O\left(m \cdot \log^{\delta} a \cdot \log\ _n \log(a \cdot \log\ n)\right)$ 个处理器，计算一个大小为 $_{a^{1+o(1)}}^{n}$ 的独立集，其中 a 为森林度。由此，对于任意小的 $\delta > 0$，我们能够使用 $O\left(m \cdot \Delta \cdot \log^{\delta} \Delta \cdot \log\ _n \log(\Delta \cdot \log\ n)\right)$ 个处理器，在时间 $\Delta^{3+o(1)} \cdot a^{1+o(1)} \cdot \log^2 n$ 内完成 $(\Delta+1)$-边着色。我们还给出了其他几种权衡，见表 1。

一般而言，我们的算法的时间和工作复杂度取决于其所采用的用于计算大型独立集的子程序的相应复杂度。对于一个具有 n 个顶点、 m 条边的图 G ，其最大度为 Δ 、森林度为 a，以及一个关于 Δ 和 / 或 a 为多项式的参数 λ，我们将计算一个 λ- 大型独立集 G （即大小为 $\Omega\left(\binom{n}{\lambda}\right)$ 的独立集）所需的时间记为 $IST_\lambda(n, \Delta, a)$，并将其使用的处理器数量记为$m \cdot ISP_\lambda(n, \Delta, \alpha)$。用这些表达式表示，我们的 $(\Delta+1)$- 边着色算法使用 $O\left(m \cdot \Delta \cdot ISP_\lambda\left(n, \Delta^2, a \cdot \Delta\right)\right)$ 个处理器，需要 $O\left(\lambda\left(\Delta^2, a \cdot \Delta\right) \cdot \Delta^2 \cdot \log\ n \cdot IST_\lambda\left(n, \Delta^2, a \cdot \Delta\right)\right)$ 的时间。在这些表达式下， [KS87] 的算法需要 $O\left(\lambda\left(\Delta^2, \Delta^2\right) \cdot \Delta^3 \cdot \log\ n \cdot \left(IST_\lambda\left(n, \Delta^2, \Delta^2\right) + \Delta^2\right)\right)$ 的时间，并使用 $O\left(m \cdot \Delta \cdot ISP_\lambda\left(n, \Delta^2, \Delta^2\right)\right)$ 个处理器； [LSH96] 的算法需要$O\left(\lambda\left(\Delta^2, a \cdot \Delta\right) \cdot \Delta^2 \cdot \log\ n \cdot \left(IST_\lambda\left(n, \Delta^2, a \cdot \Delta\right) \pm \sqrt{\Delta \cdot \log^3 \Delta}\right)\right)$ 的时间，并使用$O\left(n^2 + n \cdot \Delta^3 + m \cdot \Delta \cdot ISP_\lambda\left(n, \Delta^2, a \cdot \Delta\right)\right) = O\left(n^2 + n \cdot \Delta^3\right)$ 个处理器[2]。参见表 3。

算法	运行时间	处理器数量
[KS87]	$O\left(\lambda\left(\Delta^2, \Delta^2\right) \cdot \Delta^3 \cdot \log\ n \cdot \left(IST_\lambda\left(n, \Delta^2, \Delta^2\right) + \Delta^2\right)\right)$	$O\left(m \cdot \Delta \cdot ISP_\lambda\left(n, \Delta^2, \Delta^2\right)\right)$
[LSH96]	$O\left(\lambda\left(\Delta^2, a \cdot \Delta\right) \cdot \Delta^2 \cdot$ 对数 $n \cdot \left(IST_\lambda\left(n, \Delta^2, a \cdot \Delta\right) + \sqrt{\Delta} \cdot$对数$^3 \Delta\right)\right)$	$O\left(n^2 + n \cdot \Delta^3\right)$
Ours	$O\left(\lambda\left(\Delta^2, a \cdot \Delta\right) \cdot \Delta^2 \cdot$ <style 标识符="5">对数 $n \cdot IST_\lambda\left(n, \Delta^2, a \cdot \Delta\right)\right)$	$O\left(m \cdot \Delta \cdot ISP_\lambda\left(n, \Delta^2, a \cdot \Delta\right)\right)$

表 3：新旧并行（$\Delta+1$）-边着色算法的总结，以表达式 IST 和 ISP为依据。

[2]只要 $\Delta \geq ISP_\lambda\left(n,\ \Delta^2,\ a \cdot \Delta\right)$，第二项就支配第三项。在所有已知算法中（见定理 2）， $ISP_\lambda\left(n,\ \Delta^2,\ a \cdot \Delta\right) \leq \max\{\Delta,\ \text{poly}(\log n)\}$。若 $\Delta \leq \text{poly}(\log n)$，则第一项支配第二项和第三项。

1.6 相关工作

边着色问题是分布式计算领域中受到极其广泛研究的主题。例如，参见 [BE11b、BEM17、Ber22、BBKO22、BD23、SV19、Chr23]，及其中的参考文献。另请参见 [BE13a]，其中综述了这一迷人主题较早期的研究工作。然而，直到最近，人们才提出首个分布式 $(\Delta+1)$-边着色算法 [Ber22、Chr23、BD23]。据我们所知，这些算法无法转化为针对这一基础问题的高效并行算法。以 [BD23] 的最先进的 $(\Delta+1)$-边着色算法为例。该算法依赖一个辅助随机化例程（定理 8.1, [BD23]）：经过 $O\left(\Delta^{16}\cdot\log\ n\right)$ 的分布式时间后，它输出一个期望大小为 $|W|=\Omega\left(\frac{|U|}{\Delta^{20}}\right)$ 的子集 W，其中 U 是仍需着色的边集；同时，对于每个 $e\in W$，它还输出连通且两两不相交的 e-增广子图 H_e。（这些子图是多步 Vizing 链——为便于讨论，可以将其视为经典维辛链的扩展。详情参见 [BD23]。）利用这些子图，可以扩充当前的边着色，从而也为 W 中的边着色。结果是，经过总计 $O\left(\Delta^{20}\cdot\log\ n\right)$ 次迭代（每次至少需要 $O\left(\Delta^{16}\cdot\log n\right)$ 的时间），便可获得一个 $(\Delta+1)$-边着色。然而，要计算这一边集 W，需要充分利用分布式 LOCAL 模型。对于每条边 $e\in E$，都要收集其 $O\left(\Delta^{16}\cdot\log n\right)$ 邻域，并利用这些信息在本地计算一条多步 Vizing 链，即一个 e-增广子图。即使这一过程能够高效实现（例如在 $O\left(\Delta^{16}\cdot\log n\right)$ 时间内使用$O(m)$ 个处理器），也需要并行地对所有边 $e\in E$ 执行该过程，从而使处理器数量至少增加到 $\Omega\left(|E|^2\right)$。此外，在计算出这些 e-增广子图 $\{H_e\mid e\in E\}$ 后，[BD23] 的算法会构造一个图：当且仅当两个 H_e 相交时，它们才在该图中相连。由于该图包含 $|E|$ 个顶点，使用最先进的并行极大独立集算法 [GS89] 在其上计算一个极大独立集，将需要 $O\left(\log^3 n\right)$ 的时间和 $O\left(|E|^2\right)$ 个处理器。因此，合理推测可以获得一种运行时间为$O\left(\Delta^{36}\cdot\log^5 n\right)$、使用 $O\left(|E|^2\right)$ 个处理器的随机化并行算法；而我们提出的确定性并行算法具有小得多的运行时间和处理器数量。

1.7 论文结构

在第 3 节中，我们关注 $(\Delta+1)$-边着色问题。在第 4 节中，我们使用新的并行 $(\Delta+1)$-边着色算法，构建更高效的 $(1+\varepsilon)\Delta$-边着色算法。第 5节专门介绍我们的边着色更新算法。在附录 A 中，我们描述了 [LSH96] 算法中的缺陷。第 3 节中的一些证明推迟到附录 B。附录C包含路径和环边着色的标准例程，而附录D 专门介绍极大路径的并行计算。我们将分布式顶点着色算法改造为并行环境下算法的相关内容见附录E。

2 预备知识

除非另有说明，本文中的所有图都是无向图。

令 $G=(V,E)$ 为一个无向图。对于顶点 $v\in V$，记 v 的邻居集合为 $N(v)$，其在 G 中的度为 $\deg_G(v)=|N(v)|$。我们用 $|V|=n$、$|E|=m$ 和 G 的最大度表示为 $\Delta(G)=\max_{v\in V}\deg(v)$（或者，若图 G 的含义在上下文中明确，则记为 Δ）。

Let $v\in V$和 $e\in E$。将图 $G'=(V,\ E')$ 记为 $G\setminus e$，其中 $E'=E\setminus\{e\}$。

将图 $G' = (V', E')$ 记为 $G \setminus v$，其中 $V' = V \setminus \{v\}$，且 $E' = \{e \in E \mid v \notin e\}$。若 $v \in V$，我们使用记号 $v \in G$；若 $e \in E$，则使用记号 $e \in G$。

对于有向图 $G = (V, E)$，若$\langle v, u\rangle \in E$，则称顶点 u 是 v 的出邻居。对于 $v \in V$，我们用 $\deg_{\text{out}}(v)$ 表示 v 在 G 中的出度，即 v 在 G 中的出邻居数量。

定义 1 (欧拉图)。 当且仅当图 $G = (V, E)$ 的所有顶点的度均为偶数时，称该图为欧拉图。

定义 2 (邻接边)。 给定图 $G = (V, E)$，若 $e \neq e'$ 且两条边 $e, e' \in E$ 共享一个端点，则称它们相邻，或互为邻居。

定义 3 (正常边着色)。 图 $G = (V, E)$ 的正常 k-边着色是一个映射φ: $E \to \{1,2, ..., k\}$，满足对于任意一对邻接边 e, e'，$\varphi(e) \neq \varphi(e')$。图 (V, E) 的正常部分 k-边着色，是图 $G' = (V, F)$ 的正常边着色，其中某个$F \subseteq E$。

定义 4 （森林度）。 给定一个图 $G =$ （V， E），其森林度 a（G）是将图 G分解成边不相交森林所需的最小数量。等价地， a（G） =最大值$U \subseteq V$，$|U| \geq 2$ $\left\{ \frac{|E(U)|}{|U|-1} \right\}$ [NW64]。

定义 5 （定向）。 令 $G =$ （V, E）为一个无向图。图 G 的定向 μ 是为图中的每条边 （u, v）指定一个方向，即 $\langle u, v\rangle$ 或 $\langle v, u\rangle$。对于顶点 $v \in V$，定向中 v 的出度 是与 v 关联的出边数量。定向 μ 的出度是图中所有顶点的最大出度。

断言 1。 令 $G =$ （V, E）为一个赋有定向且出度至多为 k 的图。 G 的森林度至多为 $2k$。

证明。 设 $G = (V, E)$为一个带有定向的图，其出度至多为 k。考虑 G的一个导出子图 $G[S] = (S, F)$，其中 $S \subseteq V$, $|S| \geq 2$。由于 S中的每个顶点至多有 k条出边，因此有 $|F| \leq k \cdot |S|$。由此可得

$$a(G) = \max_{U \subseteq V, |U| \geq 2} \left\{ \frac{|E(U)|}{|U| - 1} \right\} \leq 2k.$$

□

定义 6 (退化度)。 设 $G = (V, E)$ 为一个图。 G 的退化度 是满足以下条件的最小整数 d： 其顶点存在一个排序$(v_1, v_2, \ldots, v_n)$，使得每个顶点 v_i在$\{v_{i+1}, v_{i+2}, \ldots, v_n\}$中至多有 d 个邻居。

众所周知（例如参见 [BE13b], 第1章），图的退化度至多是其森林度的两倍。

命题 2 (退化度的界)。 对于森林度为 a 、退化度为d的图 $G = (V, E)$，有 $d \leq 2a - 1$。

不失一般性，我们假设图中没有孤立顶点，即$m \geq \frac{n}{2}$。否则，我们可以在每个递归层级中从图中移除孤立顶点。

我们的所有结果均针对任意并发读并发写 PRAM 模型给出。

我们用邻接表表示输入图，并为每个顶点和每条边分别指定一个处理器。为了存储边着色 φ，对于每条边 e，负责e 的处理器都会存储 $\varphi(e)$。对于每个顶点 v，指定给 v 的处理器 p_v 会存储 v 处缺失的一种颜色（如果没有这样的颜色，则存储$\perp$）。对于每个顶点 v，我们还会存储两个哈希表 Color2Edge(v) 和 Edge2Color(v)，二者的长度均为 deg(v)。给定一种颜色 c，表 Color2Edge(v) 会在 $O(1)$ 时间内返回是否存在一条 v 关联的边 e，且该边已由 c着色为 φ。给定一条与 v关联的边 e，表 Edge2Color(v) 会在 $O(1)$ 时间内返回该边的颜色 $\varphi(e)$。

设 $G = (V, E)$ 是一个无向图，并设 φ 是 G 的某个固定的部分正常边着色，它至少使用 $\Delta + 1$ 种颜色，后文将会用到。对于顶点 v 和颜色 $\alpha \in \{1, 2, \ldots\ldots, \Delta + 1\}$，如果没有关联于 v 的边在 φ 下被着色为 α，我们就称颜色 α 在 v 处可用。记 $M(v) = \{\alpha \mid$ 颜色 α 在 v 处可用$\}$。注意，对于每个 $v \in V$，在 v处始终至少有一种可用颜色。

3 $(\Delta + 1)$-边着色

在本节 中，我们描述用于 $(\Delta + 1)$-边着色问题的算法，并分析其 t.

在第 3.1 节中，我们介绍以构造性方式证明 Vizing 定理所需的基本概念。最重要的是，我们定义了扇，并介绍一种并行构建单个扇的例程。接下来，在第 3.2 节中，我们展示如何并行构建多个扇。随后（第 3.3 节）介绍我们的并行（$\Delta + 1$）-边着色算法。

3.1 扇的操作

Vizing 定理的构造性证明 [Viz64] 会迭代地为图的边着色，并使用一种称为 扇 的结构，定义如下。扇的示例见图 1 。

定义 7 （扇）。设 $G =$ （V, E）是一个无向图，且 φ 是 G的某个固定的部分正常边着色。设 $v \in V$是一个称为扇中心的顶点。 v的一个扇 $\langle u_1, \ldots, u_k\rangle$，其（指定的）缺失颜色为 m（v）$= m_\varphi$（v）以及 $\langle m$（u_1）$\ldots, m$（u_k）$\rangle = \langle m_\varphi$（$u_1$）$\ldots, m_\varphi$（$u_k$）$\rangle$，是一个满足以下所有条件的有序顶点序列：

(i) $\langle u_1, \ldots, u_k\rangle$ 是 v 的一个非空且由不同相邻顶点构成的序列。

(ii) 对于每个 $w \in \{v, u_1, \ldots, u_k\}$，颜色 m（w）在 w 处是空闲的。

(iii) 边 （v, u $_1$）未着色；并且对于任意 $i \in \{2, 3, \ldots, k\}$，边 （$v, u_i$）着有 m（u_{i-1} ）.

- 边 $\{(v, u_i) \mid i \in \{1, 2, \ldots, k\}\}$ 称为扇的边。
- 若一个由 v 组成、缺失颜色为 $m(v)$ 的扇 $\langle u_1, \ldots, u_k\rangle$，以及 $\langle m(u_1) \ldots, m(u_k)\rangle$，无法再扩展，则称其为极大扇；也就是说，$m(u_k)$ 要么在 v 处是自由的，要么 v 的关联边已着 $m(u_k)$ 颜色且已在扇中。
- 我们称，v 的一个扇 $\langle u_1, \ldots, u_k\rangle$，其缺失颜色为 $m(v)$ 和 $\langle m(u_1) \ldots, m(u_k)\rangle$，由颜色对 $(\alpha(v)\ \beta(v))$ 表征，其中 $\alpha(v) = m(v)$ 且 $\beta(v) = m(u_k)$。

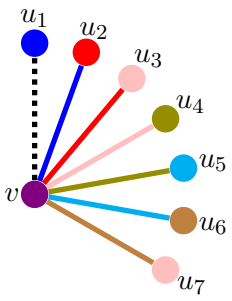

图1：在所有图中，顶点 $u_i \in u_1, u_2, ..., u_7$ 的颜色表示 u_i 处的一个自由颜色（即缺失颜色）。未着色边用虚线表示。该图描绘了一个以 v 为中心的极大扇 $u_1, u_2, ..., u_7$，其缺失颜色为 ● 和 ●, ●, ●, ●, ●, ●, ●，该扇由 $(\alpha, \beta) = (●, ●)$ 刻画。

为了计算以 $v \in V$ 为中心的极大扇，我们定义一个辅助有向图$G_{fan}^{(v,u)}$，其中 u 是 v 的邻居，且边 (v, u) 未着色。该图具有如下性质：从顶点 $u \in G_{fan}^{(v,u)}$ 出发的极大路径（即无法延伸且不经过已在路径上的顶点的简单路径）对应于以 v 为中心、包含未着色边 (v, u) 的极大扇。

定义 8. $\left(\text{辅助图 } G_{fan}^{(v,u)}\right)$。设 $G = (V, E)$ 为一个无向图，$v \in V$ 为一个顶点。设 φ 是 G 的一个固定部分正常边着色，并且 $u \in V$ 是 v 的一个邻居，使得边 (v, u) 未着色。我们将定义一个辅助有向图 $G_{fan}^{(v,u)} = \left(V_{(v,u)\,fan}, E_{fan}^{(v,u)}\right)$，它是定义在顶点集 $V_{(v,u)\,fan} = \{u, 1, 2, ..., \Delta + 1\}$ 上的 v。该图的边定义如下：

(1) 对于每个 $\alpha \in \{1、2、...、\Delta + 1\}$，如果存在 v 的一个邻居 $w \in V$，使得边 (v, w)（在 G 中）被 φ-着色为 α，我们任选一个颜色 $\beta \in M(w)$，并定义一条有向边 $\langle \alpha, \beta \rangle$。

(2) 对于任意颜色 $\beta_u \in M(u)$，存在一条有向边 $\langle u, \beta_u \rangle$。

注意，$G_{fan}^{(v,u)}$ 的最大出度为 *1*。

在分析 $G_{fan}^{(v,u)}$ 与一个具有未着色边（v，u）的极大扇之间的关系之前，我们设计了一种高效算法：给定一个最大出度至多为 1 的有向图 $G = (V, E)$，以及一个顶点 $r \in V$，该算法能够在 G 中找到一条以 r为起点的极大路径。我们将在扇的构造中使用该算法。该算法的描述及其分析见附录 D。

引理 1 (最大路径算法)。设 $G = (V, E)$ 是一个具有 n 个顶点、最大出度至多为 *1* 的有向图，$r \in V$ 是一个顶点。*Maximal-Path*过程使用 $O(n)$ 个处理器，在 $O(\log n)$ 时间内计算出从 r 开始的 G 中的一条极大路径。

在下一个引理中，我们将介绍并分析一种高效算法，该算法使用图 $G_{fan}^{(v,u)}$ 构造以 v 为中心、带有一条未着色边 (v, u) 的极大扇。

引理 2（极大扇的构造）。设 $G=(V, E)$ 是一个最大度为 Δ 的图，且 $v \in V$ 是一个顶点。设 φ 是 G 的部分正常边着色，$u_1 \in V$ 是 v 的一个邻居，且边 (v, u_1) 未着色。以 v 为扇中心、带有一条未着色边 (v, u_1) 的极大扇，可使用 O（deg(v) 个处理器，在 $O(\log \Delta)$ 时间内计算得到。

证明。设 $G^{(v,u_1)}fan$ 为定义 8 中定义的图，令 $P=\langle u_1, \alpha_1, \alpha_2, ..., \alpha_k\rangle$ 为图 $G^{(v,u_1)}fan$ 中从 u_1 出发的一条极大路径，并令 $\alpha \in \{1,2, ..., \Delta+1\}$ 为 v 处的一种可用颜色。根据 $G^{(v,u_1)}$ fan 的构造，我们知道：

- α_1 在 u_1 处是可用的。
- 对于每个 $i \in \{2, 3, ..., k\}$，存在一个顶点 $u_i \in V$，使得边 (v, u_i) 被染成 α_{i-1}，且 α_i 在 u_i 处是可用的。

因此，$f=\langle u_1, u_2, ..., u_k\rangle$ 是一个以 v 为起点、缺失颜色为 α 的扇，且 $\langle \alpha_1, \alpha_2, ..., \alpha_k\rangle$。不过，这个扇不一定是极大的。
由于 α_k 是极大路径 P 中的最后一个顶点，且该路径位于 $G^{(v,u_1)}$ fan 中，因此以下条件之一成立：要么不存在这样的 $\alpha_{k+1} \in \{1, 2,..., \Delta+1\}$，使得 $\langle \alpha_k, \alpha_{k+1}\rangle \in E^{(v,u_1)}fan$；要么存在$\langle \alpha_k, \alpha_{k+1}\rangle \in E^{(v,u_1)}fan$，使得 $\alpha_{k+1} \in \{\alpha_1, \alpha_2, ..., \alpha_k\}$。接下来我们分析这两种情况。

(1) 在第一种情况下，α_k 在 v 处是自由的。（否则，存在一个顶点 u_{k+1}，使得边 (v, u_{k+1}) 被赋予颜色 α_k。但这样一来，就存在一个缺失颜色 $\alpha_{k+1} \in M(u_{k+1})$，使得$\langle \alpha_k, \alpha_{k+1}\rangle \in E^{(v,u_1)}$ fan。）因此，$\langle u_1, u_2, ..., u_k\rangle$ 是 v 的一个极大扇，其缺失颜色为 α，并且$\langle \alpha_1, \alpha_2, ..., \alpha_k\rangle$。

(2) 在第二种情况下，有一条边 (v, u_{k+1})，其颜色为 α_k，而颜色 α_{k+1} 在u_{k+1}处是空闲的。此外，由于 α_{k+1} 此前已出现在路径中，因此与 v 关联且颜色为 α_{k+1} 的边已在扇中。注意，由于 $\varphi(v,$ $u_{k+1}) = \alpha_k$，且 $\alpha_1, \alpha_2, ..., \alpha_k$都是不同的颜色 $\Big($，又因为 P是 $G^{(v,u_1}$中的一条简单路径) $_{fan}$ $\Big)$，因此可得 $u_{k+1} \notin \{u_1, u_2, ..., u_k\}$。所以 $f'=f \circ (u_{k+1}) = u_1, u_2, ..., u_k, u_{k+1}$ 是 v 的一个极大扇，其缺失颜色为 α 和 $\alpha_1, \alpha_2, ..., \alpha_k, \alpha_{k+1}$（见图 2）。

现在我们分析这一构造的复杂度。

- 对于 $G(v,u_1)$ fan 的构造，我们为 v的每条关联边 (v, w) 指定一个处理器。如果该边的颜色为 α，处理器就任意选择一种颜色 $\beta \in M(w)$，并将有向边 α, β 添加到 $G(v,u_1)$ fan 中。此外，我们指定一个处理器，由它任意选择一种颜色 $\beta \in M(u_1)$，并将有向边 u_1, β 添加到 $G(v,u_1)$ fan 中。此过程需要 $O(1)$ 时间和 $O(\deg(v))$ 个处理器。

- 注意，$G_{(v,u_1)}$ fan 中的顶点数为 $\deg(v) + 2 \leq \Delta + 2$。因此，根据引理 1，在 $G_{(v,u_1)}$ fan 中计算一条从 u_1 出发的极大路径，需要 $O(\log \Delta)$ 时间，并使用 $O(\deg(v))$ 个处理器。

•我们为计算得到的极大路径 P =(u_1，α_1，α_2，..... α_k) 中的每种颜色 α_i 指定一个处理器 $p(\alpha_i)$。处理器 $p(\alpha_1)$ 将边 (v，u_1) 加入扇中。此外，对于 $i \in \{1, 2, ..., k-1\}$，处理器 $p(\alpha_i)$ 从数据结构 Color2Edge(α_i) 中获取顶点 u_{i+1}，使得 $\varphi(v, u_{i+1}) = \alpha_i$(并且

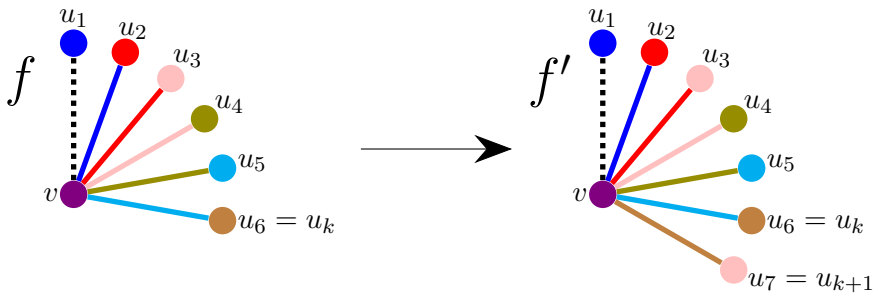

图 2：极大扇构造过程的最后一步。在此例中，$\alpha_k =$ ●，$\alpha_{k+1} = \alpha_3 =$ ● 和 $(v,\ u_4)$ 已着色为 ●，且已经在扇中。此外，$(v,\ u_{k+1})$ 已着色为 ●。这种颜色未出现在扇 f中。

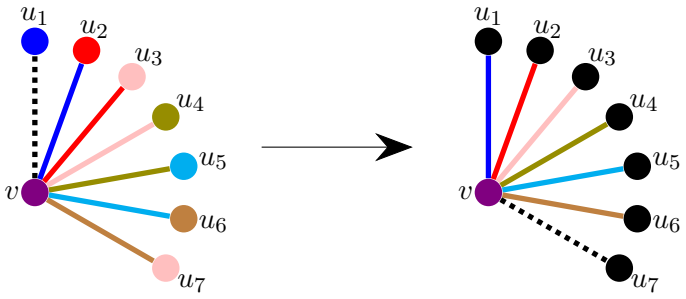

图 3：以 v 为中心的扇 $\langle u_1,\ u_2,\ \ldots\ldots,\ u_7\rangle$ 的旋转。在所有图中，黑色表示“通配符”，即未确定的缺失颜色。

> $\alpha_{i+1} \in M(u_{i+1})$)，并将边 $(v,\ u_{i+1})$ 加入扇的第 $((i+1)$ 个位置。如果调用了构造的第 2 步，则将在该步骤中计算出的顶点 u_{k+1} 追加到扇序列末尾，额外耗时 $O(1)$。使用 $O(\deg(v))$ 个处理器，此过程耗时 $O(1)$。

我们得出结论：对于以 v 为中心且包含未着色边 $(v,\ u_1)$ 的极大扇，其构造需要 $O(\log \deg(v)) = O(\log\ \Delta)$ 时间，并使用 $O(\deg(v))$ 个处理器。 □

接下来，我们将介绍扇的用途。回顾一下，扇恰好包含一条未着色边。我们将使用极大扇为其中的未着色边着色。在介绍这一着色过程之前，我们先给出后文将用到的一些定义和记号。

对于 $v \in V$ 的扇 $f = u_1, u_2, ..., u_k$，我们将该扇的一次旋转定义为：对 $i \in \{1, 2, ..., k-1\}$ 中的每个索引，使用边 (v, u_{i+1}) 的颜色重新着色边 (v, u_i)，并将边 (v, u_k) 取消着色。旋转示意见图 3。注意，根据扇的定义（边 (v, u_{i+1}) 的颜色必须在 u_i 处缺失），这一过程定义了 G 的一个新的正常部分边着色，其中边 (v, u_k) 变为未着色，边 (v, u_1) 变为已着色（而 f 的所有其他已着色边都将重新着色）。

令 $\alpha, \beta \in \{1, 2, ..., \Delta + 1\}$ 为两种颜色。我们定义图 $G_{\alpha,\beta} = (V, E_{\alpha,\beta})$，其中 $E_{\alpha,\beta} = \{e \in E \mid \varphi(e) = \alpha$ 或 $\varphi(e) = \beta\}$。注意，由于 φ 是正常边着色，因此 $\Delta(G_{\alpha,\beta}) \leq 2$

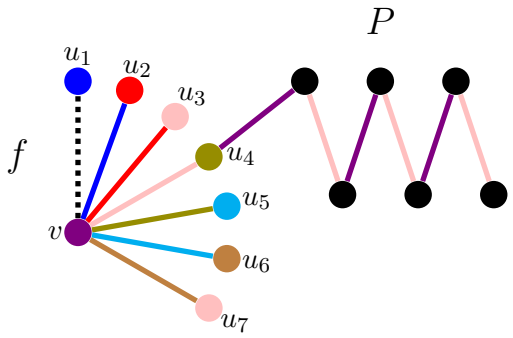

图4： P 是极大扇 $\alpha\beta$的 - 路径 f。该扇由 $(\alpha, \beta) = (\bullet, \bullet)$刻画。

（每个顶点至多有一条与其关联的、颜色为每种颜色的边）。因此，$G_{\alpha,\beta}$ 由若干简单路径和偶数长度环组成。我们将 $G_{\alpha,\beta}$ 中的路径连通分支称为一条 $\alpha\beta$-路径（$\alpha\beta$-路径可能是一个单独的顶点，即 α 和 β 均缺失的顶点）。对于 $G_{\alpha,\beta}$中的一条 $\alpha\beta$-路径$P = (w_1, w_2, ..., w_l)$，我们将该路径上的交换定义为：将P中所有 α-颜色的边重新着色为 β，反之亦然。对于颜色 $\gamma \in \{\alpha, \beta\}$，令 γ 表示满足 $\{\gamma\} = \{\alpha, \beta\} \setminus \{\gamma\}$ 的颜色。一条 $\alpha\beta$-路径 P必定是极大的，即 $\varphi(w_1, w_2) \in M(w_1)$ 且 $\varphi(w_{l-1}, w_l) \in M(w_l)$。交换 P 会产生 G的一个新的部分正常边着色，其中 $\varphi(w_1, w_2)$现在在w_1处空闲，$\varphi(w_{l-1}, w_l)$在 w_l处空闲，并且对于每个 $i \in \{2, 3, ..., l-1\}$，$M(w_i)$保持不变。请注意，交换后，边着色仍然是正常的。

观察1。对于满足 $|\{\alpha, \beta\} \cap M(v)| \geq 1$ 的顶点 $v \in V$，v 是一条 $\alpha\beta$-路径$P_{\alpha,\beta}$ 中的端点。*(如果 α 和 β 均属于 $M(v)$，那么 $P_{\alpha,\beta} = \{v\}$。)*

对于一条 w 的 $\alpha\beta$-路径 P的端点，我们可以将 P 称为 $\alpha\beta$-路径（属于 w）。对于一个扇$f = \langle u_1, u_2, ..., u_k \rangle$，其中心为某个 $v \in V$，且由 (α, β) 刻画，我们也将 $\alpha\beta$- 路径（属于 v ）称为 $\alpha\beta$-路径（属于 f ）（见图4）。

定义 9. 对于一个极大扇 $\langle u_1, u_2, ..., u_k \rangle$，其中心为 $v \in V$，并由 (α, β) 刻画，若存在索引 $i \in \{2, 3, ..., k-1\}$，使得 (v, u_i) 被染成 β，则定义 $x(v) = u_i$，$y(v) = u_k$，并且 $z(v) = u_{i-1}$。否则，没有染成 β 的边与 v 关联。（回想一下，$m(u_k) = \beta$，因此 $\varphi(v, u_k) \neq \beta$。此外，若存在一条 (v, w) 被染成 β 的边，就可以将其作为 $w = u_{k+1}$ 添加到扇 f 中，这与 f 的极大性矛盾。）在这种情况下，我们定义 $x(v) = \emptyset$，$y(v) = u_k$，并且 $z(v) = \emptyset$。

注意，如果 $x(v)$ $z(v) \neq \emptyset$，那么 $x(v)$ 和 $z(v)$ 是扇中的两个相邻顶点（$z(v) = u_{i-1}$ 以及 $x(v) = u_i$，其中 $i \in \{2, 3, ..., k-1\}$），而 $x(v)$ $y(v)$ 和 $z(v)$ 是扇中的三个不同顶点。图5给出了示意图。

为了计算 $\alpha\beta$-路径，我们使用 Shiloach 和 Vishkin 的一个结果 [SV82]，该结果可计算图的连通分量。

引理 3 (连通分量 [SV82])。设 $G = (V, E)$ 是一个 n-顶点 m-边图。存在一种确定性 CRCW PRAM 算法，可在 G 上用 $O(\log n)$时间计算连通分量，使用 $O(n + m)$ 个处理器。

现在，我们可以介绍如何利用极大扇和 $\alpha\beta$-路径为一条未着色边着色。设 $\langle u_1, ..., u_k \rangle$ 是 $v \in V$的一个极大扇，其特征由 (α, β) 给出。我们

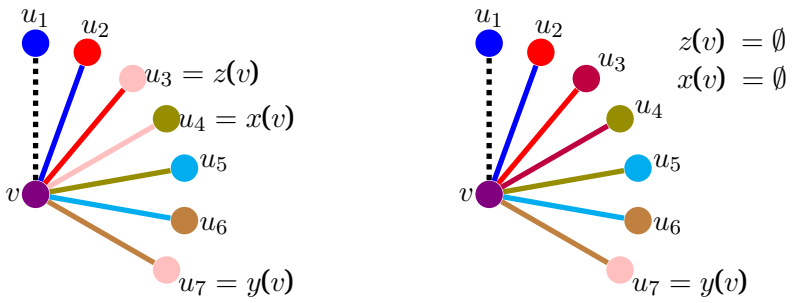

图5：扇的两个示例 $\langle u_1, u_2, \ldots\ldots, u_7\rangle$ 以 v 为中心，并由（α，β）=（●，●）表征，其特殊顶点为 x（v）、y（v）和 z（v），对应两种可能的情况。

给出 Misra 和 Gries 提出的算法 [MG92] (基于 Vizing 的构造性证明 [Viz64])，该算法定义了图 φ' 的一种新的部分正常边着色 G。该着色的支撑集（即已着色边的集合）φ' 包含原着色的支撑集 φ，并且还多出一条边（v, u_1）。算法将在过程RECOLOR-FAN中描述。另请参见图 6 以了解该算法的示意图。

Recolor-Fan 过程 ($G = (V, E)$ φ，(v，$\langle u_1, \ldots\ldots, u_k\rangle$，($\alpha$，$\beta$)))

步骤 1。交换 $\alpha\beta$-路径 P_v 中 v 的颜色，使得 β 在 v处变为自由。（根据观察 1，这样的路径是明确定义的。）

步骤 2。令 $i \in \{1, 2, \ldots\ldots, k\}$ 为一个索引，使得在原着色 φ 和交换 P_v 后得到的着色 φ' 中，β 均为 u_i的缺失颜色。（如果$z(v) \neq \emptyset$，并且步骤 1 后 β 在 $z(v)$处仍然自由，那么这个顶点就是 $z(v)$。否则，它就是 $y(v)$。）于是，$\langle u_1, \ldots\ldots, u_i\rangle$ 是 v的一个扇。旋转扇$\langle u_1, \ldots\ldots, u_i\rangle$，并用颜色 β为边(v，u_i)着色。

现在我们证明 Recolor-Fan 过程定义良好（即正确地为扇的边和 $\alpha\beta$-路径着色），并分析其复杂度。

定理1 (RECOLOR-FAN过程)。设 G 是一个 n 顶点、m 边的图，φ 是 G 的部分正常边着色，且 $\langle u_1, \ldots, u_k\rangle$ 是由颜色对（α、β）确定的 $v \in V$ 的极大扇。那么，*Recolor-Fan* 过程使用调色板 {1、2、...、$\Delta + 1$} 中的颜色为扇的边及其 $\alpha\beta$-路径着色。所得的部分着色 φ' 也满足正常边着色。此外，*Recolor-Fan* 过程的步骤 *1* 可并行地应用于多个 $\alpha\beta$-路径，并且使用 $O(n)$ 个处理器时需要 $O(\log n)$ 时间；*Recolor-Fan* 过程的步骤 *2* 使用 $O(\deg(v))$ 个处理器时需要 $O(1)$ 时间。

证明。首先注意，在交换 v 的 $\alpha\beta$-路径后，颜色 β 在 v 处可用。回想一下，交换一条路径只会影响其端点处的可用颜色。根据扇的定义，对于每个 $u_j \in \{u_1, \ldots, u_{k-1}\}\backslash\{z(v)\}$，我们有 $m(u_j) \neq \beta$。（因为只有 $(v, x(v))$ 才可能被 φ 着色 β，且对于 $j \in \{1, 2, \ldots, k-1\}$，$m(u_j) = \beta$ 意味着 $\varphi(v, u_{j+1}) = \beta$。）此外，对于每个 $u_j \in \{u_1, \ldots, u_{k-1}\}$，我们有 $m(u_j) \neq \alpha$。（由于 α 在 v 处可用，且对于 $j \in \{1, 2, \ldots, k-1\}$，$m(u_j) = \alpha$ 意味着 $\varphi(v, u_{j+1}) = \alpha$，矛盾。）因此，交换 $\alpha\beta$ 的 v 的 $\alpha\beta$-路径不会影响每个 $u_j \in \{u_1, \ldots, u_{k-1}\} \setminus \{z(v)\}$ 的缺失颜色 $m(u_j)$。现在我们考虑三种情况：

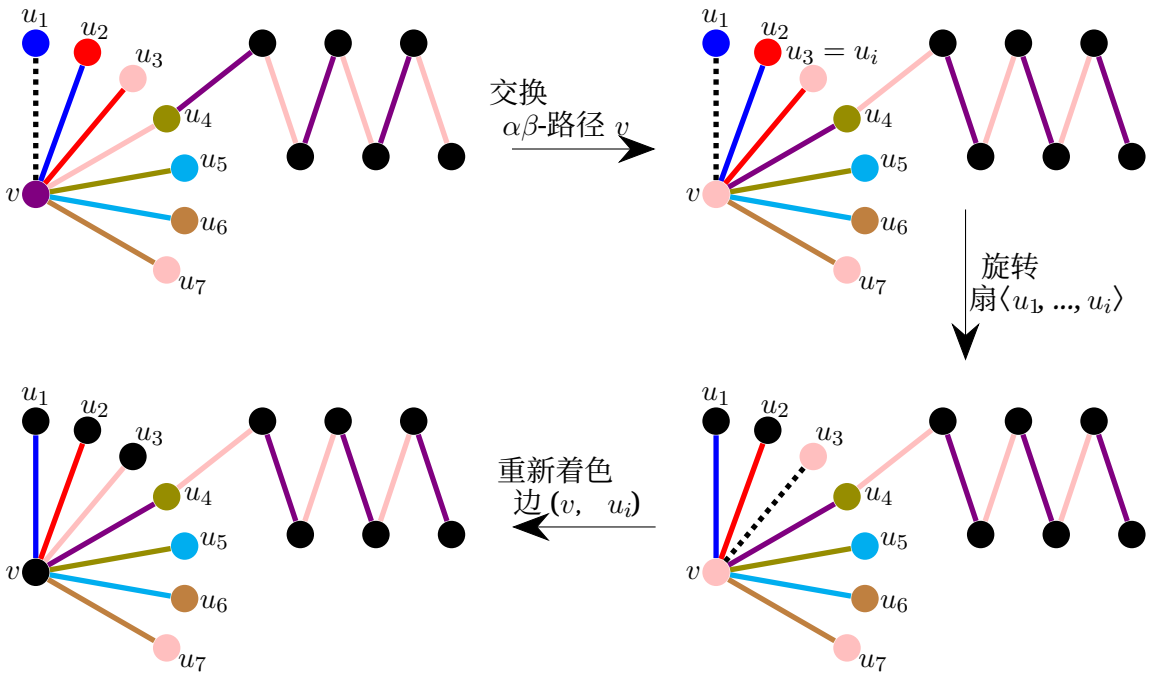

图6：过程 RECOLOR-FAN 应用于扇 $\langle u_1,\ u_2,\ .....,\ u_7\rangle$ 以 v 为中心，特征为（$\alpha,\ \beta$）=（●, ●）。

(1) 如果 $z(v)$ 和 $x(v)$ 不存在（即 $x(v) = z(v) = \emptyset$），那么与 v 关联的边中没有任何一条被 φ 着色为 β。因此， v 的 $\alpha\beta$-路径为空，交换操作不会改变任何内容。于是 $\langle u_1, ..., u_k\rangle$ 仍是 v 的一个扇，且 β 在 u_k 处仍然可用。因此，Recolor-Fan 过程的第 2 步定义良好，可以执行。

(2) 考虑 $z(v)$ 是 $\alpha\beta$-路径 $P_{\alpha,\beta}$ of v 的一个端点的情况。（回想一下， $m(z(v)) = \beta$ 和 $\varphi(v, x(v)) = \beta$。因此，如果 $z(v)$ 是 $\alpha\beta$-路径 of v 的一个端点，这意味着它有一条颜色为 α 的边，该边属于与它关联的 $\alpha\beta$-路径。）交换 v 的 $\alpha\beta$-路径后， α 将在 $z(v)$ 处变为缺失，而边 $(v, x(v))$ 将被着为 α。因此，$u_1, ..., u_k$ 仍是 v 的一个扇。（事实上， $z(v)$ 和 $x(v)$ 是扇中相邻的顶点，$\alpha = m_{\varphi'}(z(v))$ 是边 $(v, x(v))$ 的 φ'-颜色，而边 (v, u_j) 的颜色以及扇中其他顶点 u_j 的缺失颜色$m(u_j)$ 均保持不变。注意， β 在 $y(v)$ 处缺失，因此 $y(v)$ 不可能出现在 $P_{\alpha,\beta}$ 上。由于 v 的 $\alpha\beta$-路径终止于 $z(v)$，颜色 β 在 $u_k \neq z(v)$ 处仍然缺失（示意见图7）。此外， β 现在在 v 处缺失。因此，Recolor-Fan 过程的步骤2定义良好且可以执行。

(3) 否则，在交换 v 的 $\alpha\beta$-路径后，颜色 β 在 $z(v)$ 处仍然空闲（并且边 $(v,\ u_j)$ 的颜色以及扇起始处顶点 u_j $\langle u_1,\ .....,\ u_i = z(v)\rangle$、 $1 \leq j \leq i$ 的缺失颜色均未改变）。因此，$\langle u_1,\ .....,\ u_i = z(v)\rangle$ 是 v 的一个扇，并且 Recolor-Fan 过程的第 2 步定义良好且可以执行。

我们得出结论，在 Recolor-Fan 过程的步骤 1 之后，颜色 β 在 v 和 u_i处均可用（见步骤 2）。因此，旋转扇$\langle u_1, ..., u_i\rangle$后，我们可以用颜色 β为边 (v, u_i) 正确着色。注意，扇中各边的颜色以及颜色 α 和 β 均来自颜色集$\{1, 2, ..., \Delta + 1\}$。因此，Recolor-Fan 过程能对扇中的边正确重新着色

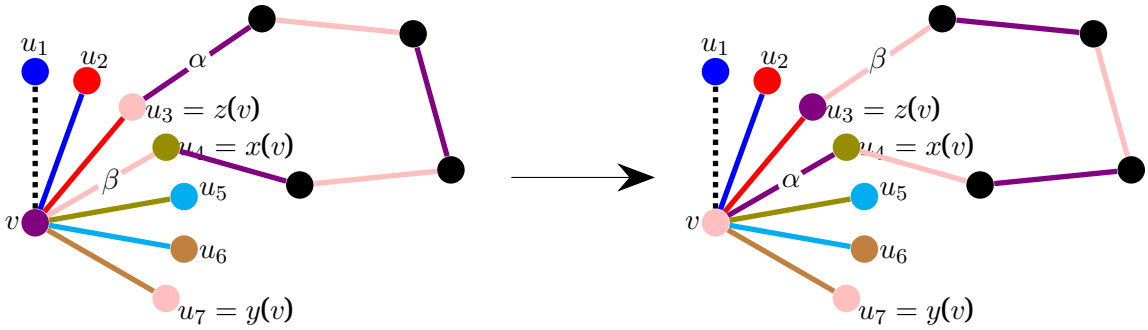

图7：极大扇的 $\alpha\beta$-路径终止于 $z(v)$。右侧展示了交换该路径后得到的着色。

以及其 $\alpha\beta$-路径，使用颜色集$\{1, 2, ..., \Delta + 1\}$中的颜色。由此得到的部分边染色是正确的，且已着色边的数量增加了 1。

现在我们分析 Recolor-Fan 过程第 1 步的复杂度。为此，我们首先使用引理 3，在 $O(\log n)$ 时间内，利用 $O(n)$ 个处理器计算图 $G_{\alpha,\beta}$ 的连通分量；对于 v 的 $\alpha\beta$-路径中的每条边 e，分配一个处理器：若 e 着色为 α，则该处理器将 e 重新着色为 β，反之亦然。该过程需要 $O(1)$ 时间，使用 $O(n)$ 个处理器。注意，该过程可以并行应用于多条不同的 $\alpha\beta$-路径（这些路径由相同的颜色对 (α, β) 或 (β, α) 确定），且复杂度不变。（注意，这些路径必然顶点不相交。）

为了计算顶点 u_i，我们为 v 的每条关联边分配一个处理器。如果不存在边 (v,u_j)，使其在交换之前的颜色为 φ（即 $u_j = x(v)$），而交换后为 φ' α，则令 $u_i = u_k = y(v)$。否则，若存在这样的索引 j，且 $u_{j-1} = z(v)$ 是 v 的 $\alpha\beta$-路径上除 v 以外的另一个端点，则令 $u_i = u_k = y(v)$；否则令 $u_i = u_{j-1} = z(v)$。无论哪种情况，此时都可在 $O(1)$ 时间内确定顶点 $z(v)$。接下来，为了旋转扇，我们为每个 $l \in \{1, 2, ..., i-1\}$ 分配一个处理器，将 (v, u_l) 重新着色为 (v, u_{l+1}) 的颜色。最后，我们将 (v, u_i) 着色为颜色 β。因此，Recolor-Fan 过程的第 2 步需要 $O(1)$ 时间，使用 $O(\deg(v))$ 个处理器。

□

3.2 并行扇形重染

正如我们在上一节所见，在 Vizing 定理的构造性证明中，可以通过反复执行 Recolor-Fan 过程，逐条用 $\Delta + 1$ 种颜色对边进行正确着色。在本节中，我们将对一大批扇并行执行 Recolor-Fan 过程；也就是说，我们希望找到一个较大的扇集合，使 Recolor-Fan 过程能够并行地对其中的扇重新着色，且不同的重新着色不会相互干扰。

我们称以 $v \in V$ 为中心的扇$\langle u_1, ..., u_k\rangle$ 和以 $v^* \in V$ 为中心的扇$\langle u_1^*, ..., u_l^*\rangle$ 为两两不相交的扇，如果$\{v, u_1, ..., u_k\} \cap \{v^*, u_1^*, ..., u_l^*\} = \emptyset$。否则，我们称这些扇为相交。作为寻找一个可并行应用 Recolor-Fan 过程的大型扇集合的第一步，我们将寻找一个两两不相交扇的大型集合。为此，令 F 为输入图 G 中未着色边集合的一个固定子集。我们定义一个辅助图

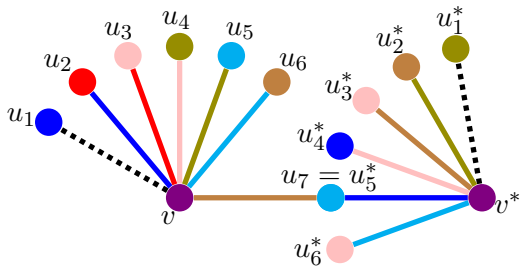

图 8：两个相交扇 $\langle u_1,\ u_2,\ \ldots\ldots,\ u_7\rangle$ 的中心为 v，以及 $\langle u_1^*,\ u_2^*,\ \ldots\ldots,\ u_6^*\rangle$ 的中心为 v^*，其中这两个扇的未着色边之间的距离为 2。

（我们称之为扇图） $G^{(F)}$，其顶点集为 F。该图具有如下性质：当 $G^{(F)}$ 中的两个边(顶点可能分别属于两个相交扇时，它们之间就有边相连。首先，我们定义 G 中一对边 $e_1=(v_1,u_1)\in E$ 和 $e_2=(v_2,u_2)\in E$ 之间的距离为

$$\mathrm{dist}_G(e_1,e_2)=\min\left\{\mathrm{dist}_G(u_1,u_2),\mathrm{dist}_G(u_1,v_2),\mathrm{dist}_G(v_1,u_2),\mathrm{dist}_G(v_1,v_2)\right\}. \tag{1}$$

定义 10。 (扇-图 $G^{(F)}$)。设 $G=(V,E)$ 为一个图，φ 为 G 的部分正常边着色，并设 F 为 G 中未着色边集合的一个固定子集。我们定义一个辅助图 $G^{(F)}=\left(F,E^{(F)}\right)$，其中对于 $e_1\neq e_2\in F$，如果 (e_1,e_2) 属于 $E^{(F)}$，则有 $\mathrm{dist}_G(e_1,e_2)\leq 2$。

正如下面将要论证的，如果两条未着色边之间的距离大于 2，那么它们不可能属于相交扇。有关两个相交扇的示例，请参见图 8，其中这两个扇的未着色边之间的距离为 2。

引理 4. (关于 $G^{(F)}$) 的性质。设 $G=(V,E)$ 是一个图，且 φ 是 G的一个部分合法边着色。设 F是 G中未着色边集合的一个固定子集，且 $G^{(F)}=\left(F,E^{(F)}\right)$是上文定义的扇图。对于 G中任意一对不同的顶点（边） e、$e^*\in F$，只要$(e,e^*)\notin E^{(F)}$，任何含有未着色边 e 的扇与任何含有未着色边 e^* 的扇都不相交。

证明很简单。为完整起见，我们在附录 B 中给出证明。

在我们的算法中，我们将重点关注一种特殊情况，即未着色边的集合 F 是 G中的一个匹配 。

定义 11 （匹配）。图 $G=(V,E)$ 中的一个匹配 M 是一个边集合，其中图 G 中的每个顶点至多与 M中的一条边关联。

从现在起，我们假设未着色边的固定子集 F 是 G 中的一个匹配。现在我们来界定 $G^{(F)}$ 的边数、最大度和森林度。

断言 3. (边数、最大度和森林度 $G^{(F)}$)。设 $G=(V,E)$ 是一个有 m 条边的图，最大度为 Δ，森林度为 a；设 φ 是 G 的部分正常边着色。令 F 为 G 中未着色边集合的一个固定子集，并假设 F 是 G 中的一个匹配；令 $G^{(F)}=\left(F,E^{(F)}\right)$ 为相对于 F 的 G 的扇图。$G^{(F)}$ 中的边数至多为 $4m\Delta$，其最大度不超过 $2(\Delta-1)^2$，且其森林度至多为 $16\Delta a$。

证明。对于顶点 z，记 $h(z)$ 为 F 中唯一一条与 z 关联的边（如果存在）。注意，对于每条边 $e=(u,v)$ 以及 u 或 v 的一个邻居 w，F 中至多有两对边 (e_1,e_2)、$(e_1,e_2)\in E^{(F)}$，满足 $e\cap e_1\neq\emptyset$ 和 $w\in e_2$。事实上，这样的边必定是 $(h(u)\ h(w))$ 或 $(h(v)\ h(w))$。另一方面，$G^{(F)}$ 中的每条边都可以用这种方式表示。因此，$G^{(F)}$ 中的边数至多为 $4m\Delta$。

现在我们对 $G^{(F)}$ 的最大度进行界定。令 $e\in F$。由于 F 是一个匹配，因此，对于每个与 e 的某个端点距离为 2 的顶点 $v\in V$，F 中至多有一条边包含该顶点。此外，每条满足 $dist_G(e,\ e')\leq 2$ 的边 $e'\in F$ 都必须包含这样一个顶点。这样的边 e' 至多有 $2(\Delta-1)^2$ 条，因此我们得到 $\deg_{G^{(F)}}(e)\leq 2(\Delta-1)^2$.[3]

接下来，我们对 $G^{(F)}$ 的森林度进行界定。设 $(v_1,v_2,...,v_n)$ 是 G 的一个退化序；也就是说，根据断言 2，对于每个 $i\in\{1,2,...,n\}$，顶点 v_i 至多有 $2a$ 个右邻点，即满足 $j>i$ 的邻居 v_j。我们将证明，该排序诱导出 $G^{(F)}$ 的一种定向，使得每个顶点的出度至多为 $8\Delta a$。由断言 1 可知，$G^{(F)}$ 的森林度至多为 $16\Delta a$。考虑一条边 $(e,e')\in E^{(F)}$。如果存在 e 的端点 v_i 和 e' 的端点 v_j，使得 $\text{dist}(v_i,v_j)=2$ 且 $i<j$，我们就将这条边定向至 e'。注意，对于每条 $(e,e')\in E^{(F)}$，都至少存在一对距离为二的此类端点。此外，还要注意，一条边可能会被定向为两个方向。（在这种情况下，可以任意选择其定向。）现在我们对该定向的出度进行界定。首先注意，对于任意顶点 v_i，满足 $j>i$ 且 $\text{dist}(v_i,v_j)=2$ 的顶点 v_j 的数量至多为 $4\Delta a$。具体而言，对于每个这样的 v_j，都存在一个顶点 v_k，使得以下情况之一成立：

- v_k 是 v_i 的右邻点，且v_j 是 v_k的邻居，或者
- v_k 是 v_i 的邻居，且v_j 是 v_k的右邻点。

在每种情况下，这样的顶点v_j至多有 $2\Delta a$ 个，因此这类顶点总数至多为 $4\Delta a$ 。现在考虑边 $e=(v_i,\ v_j)\in F$。如前所述，它的每个端点至多有$4\Delta a$ 个顶点 w 位于其右侧且与其距离为二。此外，e 的每条出边都定向指向一条形如 $h(w)$的边，其中 w满足上述条件。因此，此定向中每个顶点的出度至多为 $8\Delta a$。 □

接下来，我们分析构造 $G^{(F)}$的复杂度。

引理 5. $\left(\text{构造 }G^{(F)}\right)$。**</style>设**$(V,E)$ 是一个有 n个顶点、m条边且最大度为 Δ 的图，并令 φ 为 G的部分正常边着色。设 F 是 G中未着色边集合的一个固定子集，并假设 F 是 G中的一个匹配。构造图$G^{(F)}$（其中某些边可能重复出现）需要 $O(1)$ 时间，并使用 $O(m\cdot\Delta)$个处理器。删除可能重复的边还需要额外的 $O(\log n)$ 时间和$O(m\cdot\Delta)$ 个处理器。

证明。由于 F 是一个匹配，每个顶点 v 至多存储一条边 $h(v)\in F$，使得 v 是其一个端点（如果存在）。我们为每条边 $e\in E$ 分配一个处理器 p_e，并为每个边-顶点对 (e,w) 分配一个处理器$p_{e,\ w}$；其中该边为 $e=(v,u)$，该邻居为端点 e的一个邻居，即 v或 u的邻居。若 w是 v 的邻居，且 $h(u)$ 和 $h(w)$ 均有定义，则后一个处理器会将边 $(h(u)\ h(w))$ 添加到 $G^{(F)}$；对称地，该处理器会添加

[3]一条边 $e'=(v,\ u)\in F$如果关联于 v 某个端点的邻居，那么它也关联于一个顶点 u，该顶点与 e 的距离为 2。

边 $(h(v)\ h(w))$ 到 $G^{(F)}$，若 w 是 u 的邻居，且 $h(v)$ 和 $h(w)$ 均有定义。因此， $G^{(F)}$ 的构造可使用 $O(m\cdot\Delta)$ 个处理器在 $O(1)$ 时间内实现。

请注意，在上述构造中，可能会有多个处理器并行地将同一条边（e， e'）添加到 $G^{(F)}$中。然而，根据断言 3，所构造的多重图 $G^{(F)}$ 的总规模为 $O(m\cdot\Delta)$。现在可以通过排序去除冗余，这需要额外的 $O(\log n)$ 时间，并使用 $O(m\cdot\Delta)$ 个处理器。

□

根据引理 4，为了找到一个由两两不相交的扇组成的大集合，只需找到一个“大的” 独立集，其属于 $G^{(F)}$。

定义 12 (独立集)。 对于一个最大度为 Δ、森林度为 a 的图 $G=(V,E)$，独立集 $I\subseteq V$ 是 G 中的一个顶点集合，使得对于每个 v、$u\in I$，$(v,u)\notin E$。令$\lambda(\cdot,\cdot)$ 为一个二元多项式函数，即它关于第一个变量和第二个变量均为多项式。若一个独立集的大小为 $\Omega\left(_{\lambda}{}^{n}{}_{\Delta}(,a)\right)$，则称其为 $\lambda(\Delta,a)$-大 （若上下文已明确 Δ 和 a，则称为 λ-大）。

对于我们的边着色算法，我们考虑了多种计算独立集的算法。这些算法基于不同的顶点着色算法，这些算法呈现在附录E中。下一定理总结了这些结果。

定理 2 （计算独立集）。 设 $G=(V,E)$ 是一个具有 n 个顶点、 m 条边，最大度为 Δ且森林度为 a 的图。

(1)（[GS89]）一个 $(\Delta+1)$-大型独立集可以在 $O\left(\log^3 n\right)$ 时间内，使用 $O\left(^{m}\log\ n\right)$ 个处理器计算得到。(2)（使用定理 21 中的顶点着色算法）一个 $(\Delta+1)$-大型独立集可以在 $O\left(\log n+\Delta\cdot\log^2\Delta\right)$ 时间内，使用 $O(m)$ 个处理器计算得到。(3)（使用定理 23 中的顶点着色算法）一个 $O(a)$-大型独立集可以在 $O\left((a\cdot\log a+\log\Delta)\cdot\log\ n\right)$ 时间内，使用 $O(m)$ 个处理器计算得到。(4)（使用定理 24 中的顶点着色算法）对于任意常数 $\delta>0$，一个 $\left(a^{1+o(1)}\right)$-大型独立集可以在 $O\left(\log^{2+\delta}a\cdot\log\ n+\log\Delta\cdot\log n\right)$ 时间内，使用 $O\left(m\cdot\log^{\delta}\right.$

$\left.\frac{a\cdot\log\ n}{\log(a\cdot\log\ n)}\right)$

处理器。*(5)*（使用定理 *26* 中的顶点着色算法）一个 $O(\Delta)$-大型独立集，可在 $O\left(\sqrt{\Delta}\cdot\left(\log\Delta+\log\ n\log\ \Delta\cdot\log(\Delta\cdot\log\ n)\right)+\log\ n\right)$ 时间内使用 $O\left(m\cdot\left(\sqrt{\Delta}\cdot\log\ \Delta+\ ^{\Delta^{1/4}}\cdot\log\ \frac{n}{\log(\Delta\cdot\log\ n)}\right)\right)$ 处理器。

请注意，上述所有结果的时间复杂度都是 n、 Δ 和 a 的函数。此外，这些算法所需的处理器数量是 n 和 Δ 的函数，再乘以 m。因此，为简化边着色算法的分析，我们对这些项引入一种记法。对于一个有 n 个顶点、 m 条边的图 $G=(V,\ E)$，其最大度为 Δ、森林度为 a，且参数 λ 是关于 Δ 和 a 的多项式，我们将计算 G 的一个 λ-大型独立集所需的时间记为 $IST_{\lambda}(n,\ \Delta,\ a)$，并将此过程所需的处理器数量记为 $m\cdot$

$ISP_\lambda(n,\Delta,a)$。请注意，对于定理 2 中的算法，有 $IST_\lambda(O(n)\ O(\Delta)\ O(a)) = O(IST_\lambda(n,\Delta,a))$，且 $ISP_\lambda(O(n)\ O(\Delta)\ O(a)) = O(ISP_\lambda(n,\Delta,a))$；并且 IST_λ 和 ISP_λ 单调递增。此外，请注意，$IST_\lambda(n,\Delta,a) = \Omega(\log\ n)$。

假设我们已经构造出扇图 $G^{(F}$ 的一个 λ-大型独立集 $I = I^{(F)}$，其中 λ 是某个多项式函数。现在说明如何将这个不相交扇集合限制为一个可并行重新着色的扇集合。（并行重新着色两个不相交的扇仍可能相互干扰。）对于每条边 $e \in I$，我们任意选择一个端点 v_e，并定义 I^* 为所选端点的集合。接下来，对于每条边 $e \in I$ 及其端点 $v_e \in I^*$，我们构造一个以 v_e 为中心、含有未着色边 e 的扇 $f(v_e) = f_e$。根据引理 4，我们得到一个两两不相交的扇集合 $C_F = \{f_e \mid e \in I\}$。如上所述，我们希望找到一个扇集合，使得 Recolor-Fan 过程能够并行应用于其中所有扇，且不同的调用互不干扰。在 Recolor-Fan 过程的第一步中，给定一个由 (α,β) 表征的 $v = v_e$ 扇 $u_1, u_2, ..., u_k$，我们交换 v 的 $\alpha\beta$-路径。为了避免不同顶点在 I^* 中的这类 $\alpha\beta$-路径彼此相交，我们关注 I^* 中那些扇由固定颜色对 (α,β) 或 (β,α) 表征的顶点，其中 $\alpha,\beta \in \{1, 2, ..., \Delta+1\}$。对于颜色对 γ，$\delta \in \{1, 2, ..., \Delta+1\}$，记

$$I_{\gamma,\delta} = \{v \mid v \in I^*, \text{and the maximal fan } f(v) \text{ of } v \text{ is characterized by } (\gamma,\delta) \text{ or } (\delta,\gamma)\} \qquad (2)$$

令 α，$\beta \in \{1、2、...、\Delta+1\}$，使得 $|I_{\alpha,\ \beta}| = \max_{\delta,\ \gamma\in\{1,2,...,\Delta+1\}} |I_{\delta,\ \gamma}|$。用 $f(I_{\alpha,\ \beta})$ 表示来自 $I_{\alpha,\ \beta}$ 中顶点的扇集合。我们在下一个引理中总结这个扇集合的性质：

引理 6 (不相交扇与路径)。 $f(I_{\alpha,\ \beta})$ 中的扇两两不相交。此外，考虑分别属于两个不同顶点 $v \in I_{\alpha,\ \beta}$ 和 $v' \in I_{\alpha,\ \beta}$ 的两条 $\alpha\beta$-路径 P 和 P'。这两条路径要么顶点不相交，要么 $P = P'$。

当并行地对 $f(I_{\alpha,\beta})$ 中的扇重新着色时，仍可能出现的唯一干扰，是某个顶点 $I_{\alpha,\beta}$ 的一条 $\alpha\beta$-路径与另一个顶点 $I_{\alpha,\beta}$ 的扇相交。为避免这种情况，我们定义另一个辅助图 $G_{\text{conflict}} = (I_{\alpha,\beta}, E_{\text{conflict}})$；若两个不同顶点 $v, v^* \in I_{\alpha,\beta}$ 的扇和 $\alpha\beta$-路径可能彼此干扰，则将它们连边。具体而言，若 v 的 $\alpha\beta$-路径的一个端点是顶点 u_i^*，且该顶点位于 v^* 的扇 $u_1^*, ..., u_k^*$ 中，并且 $\beta(v^*)$ [4] 是 u_i^* 在此扇中的（指定）缺失颜色，则 (v, v^*) 属于 E_{conflict}。稍后我们将看到，只有在这种情况下，对 v 和 v^* 的扇重新着色才可能彼此干扰。图 9 给出了这样两个扇的示例。

现在我们正式定义图 G_{conflict}。

定义 13（辅助图 G_{conflict}）。设 $G = (V, E)$ 是一个图，且 $\alpha, \beta \in \{1, 2, ..., \Delta+1\}$ 是一对颜色。设 $I_{\alpha,\beta}$ 如上文所定义（见公式（2）），并设 $G_{\alpha,\beta}$ 如上文所定义 ($G_{\alpha,\beta} = (V, E_{\alpha,\beta})$，其中 $E_{\alpha,\beta} = \{e \in E \mid \varphi(e) = \alpha$ 或 $\varphi(e) = \beta\}$）。我们定义图 $G_{\text{conflict}} = (I_{\alpha,\beta}, E_{\text{conflict}})$，其顶点集为 $I_{\alpha,\beta}$。对于互不相同的顶点 $v, u \in I_{\alpha,\beta}$，边 (u, v) 属于 E_{conflict}，当且仅当 $G_{\alpha,\beta}$ 中存在一个路径连通分支，其端点为 v 和 $y(u)$、或 v 和 $z(u)$，或者对称地为 u 和 $y(v)$、或 u 和 $z(v)$。（回忆一下，特殊顶点 $x(v)$、$y(v)$ 和 $z(v)$ 在定义 9 中定义。）

我们首先证明，对于在 G_{conflict} 中不连通的两个顶点 v、$u \in I_{\alpha、\ \beta}$，可以通过 Recolor-Fan 过程并行重新着色 v 和 u 的扇。

[4] $\beta(v^*)$ 是 u_k^* 的缺失颜色。请注意，此扇可能由 $(\beta,\ \alpha)$ 表征，此时 $\beta(v^*) = \alpha$。否则，$\beta(v^*) = \beta$。

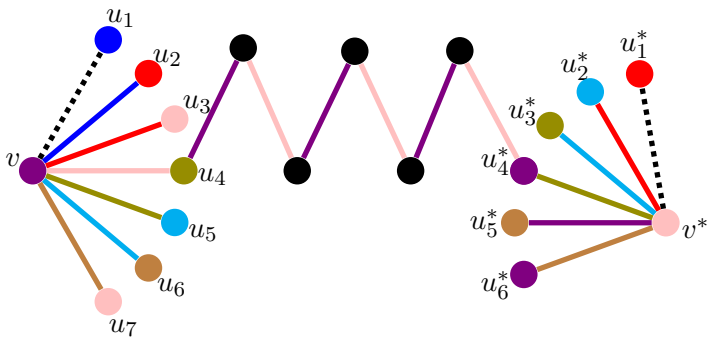

图 9：两个相互干扰的扇——以 v 为中心、由 (α,β) =(•, •) 表征的 $f(v) = u_1, u_2, ..., u_7$，以及以 v^* 为中心、由 (β,α) = (•, •) 表征的 $f(v^*) = u_1^*, u_2^*, ..., u_6^*$—— v 的 (••)-路径的一个端点，是 v^* 的扇中的一个顶点 $u_i^* = u_4^*$，且其缺失颜色为 β = •。

引理 7（关于 G_{conflict} 的性质）。设 $G = (V, E)$ 是一个图，且令 φ 为 G的部分正常边着色。设 $I_{\alpha,\beta}$ 和 G_{conflict} 分别如上文方程（*2*）和定义 *13* 中所定义。对于每一对顶点 v、$v^* \in I_{\alpha,\beta}$，只要满足 $(v, v^*) \notin E_{\text{conflict}}$，就可以并行重新着色 v 和 v^* 的扇。

证明。设 $u_1, ..., u_k$ 是 v 的扇，$u_1^*, ..., u_l^*$ 是 v^* 的扇。由于 v 和 v^* 的 $\alpha\beta$-路径要么顶点不相交，要么是同一条路径，因此可以并行进行 v 和 v^* 的 $\alpha\beta$-路径交换。现在我们说明，在这些交换之后，可以正确地旋转 v（以及 v^*）的扇（对于 v^*，证明对称）。不失一般性，假设 v 的扇由 (α,β) 表征（而非由 (β,α) 表征）。首先注意，在交换 v 的 $\alpha\beta$-路径之后，颜色 β 在 v 处可用。回顾一下，路径交换只会影响其端点处的可用颜色。根据扇的定义，注意到对于每个 $u_j \in \{u_1, ..., u_{k-1}\} \setminus \{z(v)\}$，有 $m(u_j) \neq \beta$（因为只有 $(v, x(v))$ 被 φ 颜色着色 β）；而对于每个$u_j \in \{u_1, ..., u_{k-1}\}$，有 $m(u_j) \neq \alpha$。（注意，α 在 v 处可用。另一方面，$m(u_j) = \alpha$ 蕴含 $\varphi(v,u_{j+1}) = \alpha$，矛盾。）因此，交换 v 和 v^* 的 $\alpha\beta$-路径不会影响 $\{u_1, ..., u_{k-1}\} \setminus \{z(v)\}$ 中顶点指定的缺失颜色。现在我们分三种情况讨论：

(1) 如果 $z(v)$ 和 $x(v)$ 不存在（见图5），那么 v 上没有关联边被 φ 着色为 β。因此，根据扇的定义，颜色 β 不是 $\{u_1, ..., u_{k-1}\}$ 中顶点的缺失颜色，并且 v 的 $\alpha\beta$-路径为空。因此，交换 v 的 $\alpha\beta$- 路径不会影响 $\{u_1, ..., u_{k-1}\}$ 中顶点的缺失颜色。此外，v^* 的 $\alpha\beta$- 路径不可能终止于 $u_k = y(v)$，因为 $(v, v^*) \notin E_{\text{conflict}}$。因此，交换该路径后，$\beta$ 在 u_k 处仍然缺失。由此可知，$u_1, ..., u_k$ 仍是 v 的一个扇，并且 β 在 $u_k = y(v)$ 处仍然空闲。因此，Recolor-Fan 过程的第 2 步（旋转）定义良好且可以执行。

(2) 考虑 $z(v)$ 是 v 的 $\alpha\beta$-路径的一个端点的情形。（另一个端点就是 v 本身。见图7。）那么交换 v 的 $\alpha\beta$-路径后，颜色 α 在 $z(v)$ 处空闲，而边 $(v, x(v))$ 被染成 α（扇中其余部分的颜色和缺失颜色保持不变）。因此，$u_1, ..., u_k$ 仍然是 v 的一个扇（$z(v)$ 和 $x(v)$ 是扇中的连续顶点）。由于 $(v, v^*) \notin E_{\text{conflict}}$，$v^*$ 的 $\alpha\beta$-路径不可能以 $u_k = y(v)$ 或 $z(v)$ 为端点。因此，交换 v^* 的 $\alpha\beta$- 路径不会影响 u_k 和 $z(v)$ 的缺失颜色。还要注意

如果 v 和 v^* 的 $\alpha\beta$-路径相同，那么 $v^* = z(v)$。另一方面， v 和 v^* 是 $I_{\alpha,\beta} \subseteq I^*$ 中两个不同的顶点，即它们分别是两条边 e 和 e^* 的端点， $e, e^* \in I^{(F)}$。由于 $G^{(F)}$ 中的 $I^{(F)}$ 是一个独立集，这意味着 $dist_G(e, e^*) > 2$，但 v 和 $v^* = z(v)$ 是 G 中的邻居，矛盾。因此， v^* 的 $\alpha\beta$-路径与 v 的 $\alpha\beta$-路径不相交。于是，交换 v^* 的 $\alpha\beta$-路径时， v 的扇中各边的颜色也保持不变（ v^* 的 $\alpha\beta$-路径不可能包含边 $(v, x(v))$ 或顶点 $z(v)$）。因此， $u_1, ..., u_k$ 仍然是 v 的一个扇，Recolor-Fan 过程的步骤 2 定义良好，可以执行。

(3) 否则，由于 $(v, v^*) \notin E_{\text{conflict}}$，因此 $z(v)$ 不是 v^* 的 $\alpha\beta$-路径的端点。（此外，它也不是 v 的 $\alpha\beta$- 路径的端点。）交换 v 和 v^* 的 $\alpha\beta$- 路径后，颜色 β 仍然是 $z(v)$ 的缺失颜色，并且子扇 $\langle u_1, ..., z(v)\rangle$的颜色和缺失颜色保持不变。因此， $\langle u_1, ..., z(v)\rangle$ 也是 v 的一个扇，且 Recolor-Fan 过程的第 2 步定义良好，可以执行。

□

现在我们来分析 G_{conflict}的最大度。

断言 4 (G_{conflict}的最大度）。设 $G = (V, E)$ 是一个图，且 φ 是 G的部分正常边着色。设 G_{conflict} 是定义 *13* 中定义的图。的最大度至多为 *3*。

证明。 令 $v \in I_{\alpha,\beta}$ 为 G_{conflict} 中的一个顶点。根据 G_{conflict} 的定义， v 只能与 $y(v)$ 的 $\alpha\beta$-路径的另一个端点、 $z(v)$ 的 $\alpha\beta$-路径的另一个端点，以及包含 v 的 $\alpha\beta$-路径的另一个端点的扇的中心连通。因此 $\Delta(G_{\text{conflict}}) \leq 3$。 □

接下来，我们分析 G_{conflict} 的构造复杂度。

引理 *8*（ G_{conflict} 的构造）。设 $G = (V, E)$ 是一个最大度为 Δ 的 n-顶点图，且 φ 是 G 的部分正常边着色。令 I_{α}、$_{\beta}$ 和 G_{conflict} 如上文所定义（分别参见公式（*2*）和定义 *13*）。使用 $O(n)$ 个处理器，构造图 G_{conflict} 需要 $O(\log\ n)$ 时间。

证明。 G_{conflict} 的构造可以分三步完成：

- 首先，我们使用引理 3 [SV82] 中的算法，在 $O(n)$ 个处理器上用 $O(\log n)$ 时间计算上文定义的图 $G_{\alpha,\beta} = (V, E_{\alpha,\beta})$ 的连通分量 ($G_{\alpha,\beta} = (V, E_{\alpha,\beta})$，其中 $E_{\alpha,\beta} = \{e \in E \mid \varphi(e) = \alpha$ 或 $\varphi(e) = \beta\}$)。注意，$G_{\alpha,\beta}$ 中至多有 n 条边。令 $l \leq n$ 为所得的连通分量数。

- 接下来，我们定义一个大小为 l 的辅助数组 L，其条目为数对。然后，我们为每个顶点 $v \in V$ 分配一个处理器。回顾一下，$G_{\alpha,\beta}$ 的连通分量要么是简单环，要么是简单路径。对于每个 $v \in V$，如果 v 是由 i 索引的路径连通分支的端点，则该处理器将 v 加到 $L[i]$ 中。这一步需要 $O(1)$ 时间和 $O(n)$ 个处理器。

- 最后，我们为 $G_{\alpha,\beta}$ 的每个连通分量 $i \in \{1, 2, ..., l\}$ 分配一个处理器。注意，对于 $G_{\alpha,\beta}$ 的一个非空路径连通分支 i，条目 $L[i]$ 将包含这条路径的两个

端点 v_i 和 u_i。如果 $v_i \in I_{\alpha,\beta}$，且对于某个 $u_i^* \in I_{\alpha,\beta}$，有 $u_i = y(u_i^*)$ 或 $u_i = z(u_i^*)$，那么该处理器会在 G_{conflict} 中定义边 (v_i, u_i^*)（每个形如 $x(v)$、$y(v)$ 或 $z(v)$ 的顶点都会存储其扇中心 v）。此步骤需要 $O(1)$ 时间和 $O(n)$ 个处理器。

因此，构造 G_{conflict} 需要 $O(\log n)$ 时间，并使用 $O(n)$ 个处理器。 □

根据引理 7，要找到一组可以并行重新着色的扇，只需找到一个大小为 G_{conflict} 的独立集。接下来，我们介绍PARALLEL-FANS过程，该过程计算一组可以并行重新着色的扇。

Parallel-Fans 过程（$G=(V,\ E)$ φ）

步骤 1. 令 F 为 G 中未着色边集合的一个固定子集。使用引理 $G^{(F)}$（参见定义 10）中的算法构造图 5。

步骤 2. 使用定理 λ 中的某个算法，计算图 $I^{(F)}$ 的一个 $G^{(F)}$ -大型独立集2。

步骤 3. 对每个 $e \in I^{(F)}$，并行地任意选择 v 的一个端点 e，并使用引理 v，计算一个包含未着色边 e 的极大扇 2。将所选端点的集合记为 I^*。

步骤4。 令 $\alpha, \beta \in \{1, 2, \ldots, \Delta+1\}$ 满足 $|I_{\alpha,\beta}| = \text{最大值}_{\delta,\gamma \in \{1,2,\ldots,\Delta+1\}} |I_{\delta,\gamma}|$。使用引理 $G_{\text{conflict}} = (I_{\alpha,\beta}, E_{\text{conflict}})$ 中的算法构造图 8。
步骤5。 使用定理 $\left(\Delta\left(G_{(\text{conflict})}\right)+1\right)$-大型独立集 I_{fan} 的算法，计算 G_{conflict} 的 2(2)。

返回 I_{fan} 中顶点的扇。

接下来，我们分析 Parallel-Fans 过程的复杂度。该过程由参数 λ 参数化，该参数决定了该过程在步骤2中计算的独立集的大小。我们将参数为 λ 的 Parallel-Fans 过程记作 Parallel-Fans$_\lambda$。若当前讨论的陈述与参数无关，则省略下标 λ。

我们首先分析 $I_{\alpha,\beta}$ 的计算（该过程的第 4 步）。$I_{\alpha,\beta}$ 的定义见方程（2）。

断言 5 （的计算 $_{\beta,}$ ）。设 $G=$ (V, E) 是一个具有 n 个顶点且最大度为 Δ 的图，并设 φ 是 G 的部分正常边着色。设 F 是 G 中未着色边集合的一个固定子集。设 I^* 是在 *Parallel-Fans* 过程中定义的扇中心集合。$I_{\alpha,\beta}$ 的计算需要 O（$\log n$） 时间，并使用 O（n） 个处理器。

证明。首先，我们描述一个需要 $O(n+\Delta^2)$ 个处理器的简单例程。我们为每个顶点 $v \in I^*$ 分配一个处理器，并为每个颜色对 α，$\beta \in$ {1、2、.....、$\Delta+1$} 分配一个处理器。利用 I^* 中各顶点的处理器，我们按照颜色对相应的扇进行排序。将所得的排序数组记为 $\mathcal{A}$。然后，对于每个颜色对 α，$\beta \in$ {1、2、.....、$\Delta+1$}，其处理器通过计算该颜色对在数组$\mathcal{A}$中的范围来计算 $I_{\alpha,\ \beta}$ 的大小。最后，我们返回其中最大的集合。总的来说，我们使用了 $O(n+\Delta^2)$ 个处理器，并在 O（$\log n$）时间内完成。

事实上，这项任务也可以使用 $O(n)$ 个处理器在相同时间内完成。对于每个颜色对 α，β，令 $v_{\alpha,\beta} \in I^*$ 为其扇 $f = f(v_{\alpha,\beta})$ 由 α，β 刻画的顶点，并且满足 f 是数组 $\mathcal{A}$ 中所有由 α，β 刻画的扇中最先出现的扇。现在，对于所有颜色对 α，β，若 $\mathcal{A}$ 中存在由该颜色对刻画的扇，则处理器 $v_{\alpha,\beta}$ 找出相应颜色对在$\mathcal{A}$中的范围，并找出最大范围。因此，这一计算所需的处理器总数为 $O(n)$。 □

现在我们来分析 Parallel-Fans 过程的复杂度。

引理9（Parallel-Fans过程的复杂度）。设 $G = (V, E)$ 是一个具有 n 个顶点和 m 条边的图，其最大度为 Δ，森林度为 a；设 φ 是 G 的部分正常边着色。令 F 为 G 中未着色边集合的一个固定子集，并假设 F 是 G 中的一个匹配。*Parallel-Fans* 过程$_\lambda$ 需要 $O\left(IST_\lambda\left(n, \Delta^2, a\cdot\Delta\right)\right)$ 的时间，并使用 $O\left(m\cdot\Delta\cdot ISP_\lambda\left(n, \Delta^2, a\cdot\Delta\right) + m\cdot\Delta\right)$ 个处理器。

证明。• 根据引理5，第 1 步中$G^{(F)}$ 的构造需要 $O(\log n)$ 的时间，并使用 $O(m\cdot\Delta)$ 个处理器。

- 注意到，由于 F 是一个匹配，$G(F)$ 中至多有 n 个顶点。根据断言 3，$G(F)$ 中有 $m(F) = O(m\cdot\Delta)$ 条边，其最大度为 $\Delta(F) = O(\Delta^2)$，森林度为 $a(F) = O(a\cdot\Delta)$。因此，在步骤 2 中计算一个 λ-大型独立集 $I(F)$ of$G(F)$ 需要 $O\left(IST_\lambda\left(n, \Delta^2, a\cdot\Delta\right)\right)$ 时间，使用 $O\left(m(F)\cdot ISP_\lambda\left(n, \Delta^2, a\cdot\Delta\right)\right) = O\left(m\cdot\Delta\cdot ISP_\lambda\left(n, \Delta^2, a\cdot\Delta\right)\right)$ 个处理器。

- 根据引理 2，对于所有 $(v, u) \in I(F)$ 以及 $v \in I^*$ 并行地，为 v 利用未着色边 (v, u) 构造一个极大扇，总共需要 $O(\log\Delta)$ 时间，并使用 $\sum_{v\in I^*} O(\deg(v)) = O(m)$ 个处理器。

- 根据断言 5，计算 $I_{\alpha,\beta}$ 需要 $O(\log n)$ 时间，并使用 $O(n)$ 个处理器。
- 根据引理 8，构造 $G_{冲突}$ 需要 $O(\log n)$ 时间，并使用 $O(n)$ 个处理器。
- 回顾一下，根据断言 4，$\Delta(G_{冲突}) \le 3$。因此，根据定理2(2)，在步骤5中计算 $G_{冲突}$ 的一个 $(\Delta(G_{冲突})+1)$-大型独立集需要 $O(\log n)$ 时间，并使用 $O(n)$ 个处理器。

总而言之，Parallel-Fans 过程需要 $O\left(IST_\lambda\left(n, \Delta^2, a\cdot\Delta\right) + \log n\right) = O\left(IST_\lambda\left(n, \Delta^2, a\cdot\Delta\right)\right)$时间，并使用 $O\left(m\cdot\Delta\cdot ISP_\lambda\left(n, \Delta^2, a\cdot\Delta\right) + m\cdot\Delta\right)$ 个处理器。（注意，$ISP_\lambda\left(n, \Delta^2, a\cdot\Delta\right)$ 可能小于 1。例如，参见定理 2(1)） □

最后，我们分析过程 Parallel-Fans 返回的扇集合大小。

引理 10 (Parallel-Fans 过程的性质Parallel-Fans)。设 $G = (V, E)$ 是一个最大度为 Δ、森林度为 a的 n 顶点图，且 φ是 G的部分正常边着色。令 F 为 G 中未着色边集合的一个固定子集，并假设 F 是 G中的一个匹配。*Parallel-Fans* 过程$_\lambda$ 输出一组 $\Omega\left(\frac{|F|}{\lambda\,\Delta^2 a(\cdot\Delta)\cdot\Delta^2}\right)$ 个可以并行重新着色的扇。

证明。令 $\Delta^{(F)} = \Delta\left(G^{(F)}\right)$。算法第 2 步中计算得到的 $G^{(F)}$ 的 λ-大型独立集 $I^{(F)}$ 的大小至少为 $\Omega\left(\frac{|F|}{\lambda\,\Delta^{(F)}\left(,\ a^{(F)}\right)}\right)$，即

$$|I^*| = \left|I^{(F)}\right| = \Omega\left(\frac{|F|}{\lambda\left(\Delta^{(F)}, a^{(F)}\right)}\right).$$

（回想一下，I^* 包含每条边 $e \in I^{(F)}$的一个端点。）注意，缺失颜色 γ、$\delta \in \{1, 2, ..., \Delta+1\}$共有 $\binom{\Delta+1}{2} + (\Delta+1) = \frac{\Delta^2+3\Delta+2}{2}$ 对（不一定互不相同）。还要注意，$I^* = \cup_{\alpha、\ \beta\in\{1,2,...,\Delta\}} I_{\alpha、\ \beta}$，并且对于不同的颜色对（$\alpha$，$\beta$）$\neq$（$\alpha'$，$\beta'$），相应的子集$I_{\alpha、\ \beta}$与$I_{\alpha'、\ \beta'}$互不相交。（见公式（2）。）由于$I_{\alpha、\ \beta}$是这些集合中最大的，因此我们有

$$|I_{\alpha,\beta}| \geq \frac{|I^*|}{\frac{\Delta^2+3\Delta+2}{2}} \geq \frac{|F|}{c\cdot\lambda\left(\Delta^{(F)}, a^{(F)}\right)\cdot\Delta^2},$$

其中 $c > 0$为常数。由于 G_{conflict} 的最大度至多为 3，根据定理 2(2)，算法在步骤5中计算得到的 G_{conflict} 的独立集 I_{fan} 的大小至少为$|I_{\text{fan}}| \geq \frac{|I_{\alpha,\beta}|}{4} \geq \frac{|F|}{4\,c\,\lambda\ \Delta^{(F)}\cdot\cdot\left(,a^{(F)}\right)\cdot\Delta^2}$。根据断言 3，我们有 $\Delta^{(F)} \leq 2\Delta^2 - 4\Delta + 2$，且 $a^{(F)} = O(a\cdot\ \Delta)$。因此，Parallel-Fans 过程输出一个包含至少 $\Omega\left(\frac{|F|}{\lambda\ \Delta^2\ a\left(,\cdot\Delta\right)\cdot\Delta^2}\right)$ 个扇的集合，这些扇可以并行重新着色。（由于 λ 是多项式函数，我们有 $\lambda(O(\Delta^2)\ O(a\cdot\ \Delta)) = O\left(\lambda\left(\Delta^2, a\cdot\ \Delta\right)\right)$。）

□

我们在以下定理中总结 Parallel-Fans 过程的性质。

定理 *3*（*Parallel-Fans* 过程）。设 $G = (V, E)$ 是一个具有 n 个顶点、m 条边的 n-顶点 m-边图，其最大度为 Δ，森林度为 a；设 φ 是 G的部分正常边着色。令 F 为 G 中未着色边集合的一个固定子集，并假设 F 是 G的一个匹配。*Parallel-Fans* 过程$_\lambda$输出一个集合，其中包含至少 $\Omega\left(\frac{|F|}{\lambda\ \Delta^2\ a\left(,\cdot\Delta\right)\cdot\Delta^2}\right)$ 个可并行重新着色的扇。该过程使用 $O\left(m\cdot\ \Delta\cdot\ ISP_\lambda\left(n, \Delta^2, a\cdot\ \Delta\right) + m\cdot\ \Delta\right)$ 个处理器，在 $O\left(IST_\lambda\left(n, \Delta^2, a\cdot\ \Delta\right)\right)$ 时间内完成。

3.3 边着色算法

本节介绍我们的并行边着色算法。首先介绍 Reduce-Color 过程：给定最大度为 Δ 的图 $G = (V, E)$，以及使用 $k > \Delta+1$ 种颜色对 G 进行的正常边着色 φ，该过程返回一种使用 $k-1$ 种颜色对 G 进行的正常边着色。该算法先将图中的一个颜色类取消着色，然后迭代地并行重新着色一大部分未着色边，并使用 Parallel-Fans 和 Recolor-Fan 过程。

Reduce-Color 过程（$G = (V,\ E)\ \varphi$） ▷ $|\varphi| = k$

步骤 1. 将 G 中着色为 k 的边取消着色。将未着色边的集合记为 F，并令 φ 为由此得到的部分正常边着色。

重复

步骤 2。使用输入参数（G，φ）的 Parallel-Fans 过程，计算一个可以并行重新着色的扇集合，并对其中的每个扇并行应用 Recolor-Fan 过程

以重新着色该扇的未着色边。令 H 表示在此步骤中着色的 F 的边集合。

步骤 3。 $F = F \setminus H$

直到 $F = \emptyset$

我们分析 Reduce-Color过程，然后继续介绍主要的边着色算法。

Reduce-Color过程的复杂度取决于参数 λ，该参数用于在前一个过程的步骤 2 中调用的 Parallel- Fans过程。我们将使用参数 λ 调用的 Reduce-Color过程记为 Reduce-Color$_\lambda$。只要当前陈述不依赖于参数 λ，我们就可以省略该参数。

定理 4（过程 REDUCE-COLOR的性质）。 设 $G = (V, E)$ 是一个具有 n 个顶点和 m 条边、最大度为 Δ、森林度为 a 的图。设 $k > \Delta + 1$ 为一个整数，并设 φ 是 G 的一个适当的k-边着色。*Reduce-Color*过程$_\lambda$ 在$O\left(\lambda\left(\Delta^2\right.\right.$中的 G 上，使用 $O\left(m \cdot \Delta \cdot ISP_\lambda\left(n, \Delta^2, a \cdot \Delta\right) + m \cdot \Delta\right)$ 个处理器，在 $a \cdot \Delta) \cdot \Delta^2 \cdot \log n \cdot IST_\lambda\left(n, \Delta^2, a \cdot \Delta\right)\right)$ 时间内计算出一个适当的（$k-1$）-边着色。

证明。令 F 为步骤 1 中所述的未着色边集合。注意， F 是 G 中的一个匹配，其大小至多为 $\frac{n}{2}$。根据定理 3，在步骤 2 的每次迭代中，我们计算一组 Ω $\left(\lambda\ \Delta^2 \frac{|F|}{a}(\ ,\cdot\Delta)\cdot\Delta^2\right)$个扇，这些扇可以并行重新着色。该计算使用 $O\left(m \cdot \Delta \cdot ISP_\lambda\left(n, \Delta^2, a \cdot \Delta\right) + m \cdot \Delta\right)$ 个处理器，需要 $O\left(IST_\lambda\left(n, \Delta^2, a \cdot \Delta\right)\right)$ 时间。因此，根据定理1，在每次迭代中，我们用来自 {1, 2, ..., $\Delta + 1$} $\subseteq$ {1, 2, ..., $k-1$} 的颜色，在 $O(\log\ n)$ 时间内重新着色 Ω $\left(\frac{}{\lambda\ \Delta^2 \frac{|F|}{a}(\ ,\cdot\Delta)\cdot\Delta^2}\right)$ 条边，并使用 $O(m)$个处理器。令 $c > 0$ 为一个充分小的常数，使得在 Reduce-Color过程的主循环每次迭代中，我们至少为 $c \cdot\ \lambda\ \Delta^2 \frac{|F|}{a}\Delta(\ ,\ )\cdot\Delta^2$ 条边着色。

对于每个整数 i，经过 Reduce-Color过程主循环的 i 次迭代后，剩下至多 $|F| \cdot \left(1 - \lambda\ \Delta^2 \frac{c}{a}(\ ,\cdot\Delta)\cdot \Delta^2\right)^i$ 条未着色边。因此，经过$\lambda(\Delta^2,a\cdot\Delta)\cdot\Delta^2\cdot\ln_c\ ^n$ 次迭代后，剩下至多 $|F| \cdot \left(1 - \lambda\ \Delta^2 \frac{c}{a}(\ ,\cdot\Delta)\cdot\Delta^2\right)^\lambda$ $(\Delta^2,a\cdot\Delta)\cdot\Delta^2\cdot\ln^n \leq |F| \cdot \frac{1}{n} < 1$ 条未着色边。因此，经过主循环的 $O\left(\lambda\left(\Delta^2, a \cdot \Delta\right) \cdot \Delta^2 \cdot \log\ n\right)$ 次迭代后，算法终止。我们得出结论：Reduce-Color过程在 $O(\lambda\left(\Delta^2, a \cdot \Delta\right) \cdot \Delta^2 \cdot \log\ n \cdot\ IST_\lambda\left(n, \Delta^2, a \cdot \Delta\right))$ 时间内，使用 $O\left(m \cdot \Delta \cdot ISP_\lambda\left(n, \Delta^2, a \cdot \Delta\right) + m \cdot \Delta\right)$ 个处理器，为 G 计算出一个正确的 $(k-1)$-边着色。 □

现在，我们可以介绍主要的（$\Delta + 1$）边着色算法了。我们先说明输入图 $G = (V, E)$ 是欧拉图且最大度为 $\Delta = 2^h$（其中 h 是正整数）这一简单情形下的算法思路。该算法采用分治方法。我们将图 G 划分为 $\frac{\Delta}{2}$ 个边不相交子图 $G_1^{(0)}, G_2^{(0)}, \ldots\ldots, G_\Delta^{(0)}$ 每个子图的^{2}最大度均为 2。我们快速求出每个子图的解，再将这些解合并为输入图 $\overline{G}$的一个解。图 10 展示了一个示例。下面我们将逐一介绍这些部分：

划分：我们在 G 中计算一个有向欧拉回路，并构造二分图 $G_B = \left(V(in) \cup V(out) E_B\right)$，其中 $V(in) = \left\{v^{(in)} | v \in V\right\}$、 $V(out) = \left\{v^{(out)} | v \in V\right\}$；若有向边 v, u 属于该欧拉回路，则存在边 $\left(v^{(out)}, u^{(in)}\right) \in E_B$。注意， G_B 的最大度为 $\frac{\Delta}{2}$，我们可以使用 [LPV81] 的算法，采用 $\frac{\Delta}{2}$ 种颜色对 G_B 进行边着色（见引理

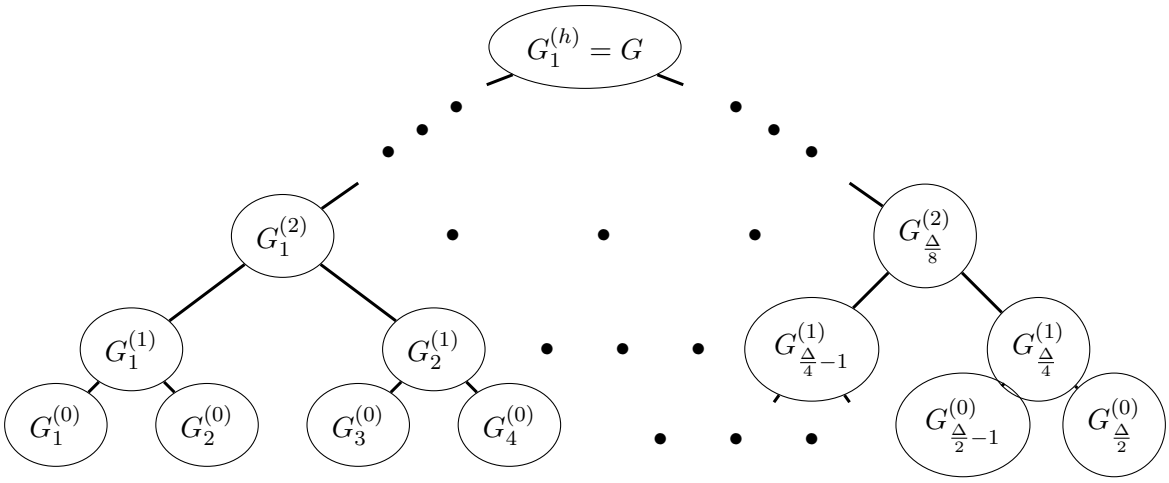

图 10：边着色过程的合并过程。

12 下文）。根据边着色的颜色类[2]，定义边不相交子图 $G_1^{(0)}$，$G_2^{(0)}$，……，$G_{\Delta}^{(0)}$。具体而言，对于每个 $i \in \left\{1, 2, \ldots\ldots, \frac{\Delta}{2}\right\}$，我们定义 $G_i^{(0)} = \left(V, E_i^{(0)}\right)$，其中$E_i^{(0)} = \left\{(v, u) \in E \mid \left(v^{(out)}, u^{(in)}\right) \text{ 在 } G_B\right\}$ 中被着色为 i。

颜色：注意，每个这样的子图的最大度为 2。因此，我们可以使用简单的边着色算法对这些子图分别进行三边着色。（参见定理 5中的算法。）

合并：这一部分由 $h = \log\left(\frac{\Delta}{2}\right)$ 次迭代组成。在每次迭代 $i \geq 1$ 中，给定子图 $G_1^{(i-1)}$、$G_2^{(i-1)}$、……、$G_{\frac{\Delta}{2^i}}^{(i-1)}$，它们分别使用 $\Delta\left(G_j^{(i-1)}\right) + 1$ 种颜色 $j \in \left\{1、2、\ldots\ldots、\frac{\Delta}{2^i}\right\}$ 着色。我们使用不相交的颜色集合并子图对 $G_j^{(i)} = G_{2j-1}^{(i-1)} \cup G_{2j}^{(i-1)}$；正如我们将看到的，这将得到 $G_j^{(i)}$ 的 $\left(\Delta\left(G_j^{(i)}\right) + 2\right)$-边着色。接下来，我们使用边着色过程 Reduce-Color 并行地从这些着色中各减少一种颜色，从而为每个 $G_j^{(i)}$ 得到 $\left(\Delta\left(G_j^{(i)}\right) + 1\right)$-边着色。经过这 h 次迭代后，我们得到 $G_1^{(h)} = G$ 的 a$\left(\Delta\left(G_1^{(h)}\right) + 1\right) = (\Delta(G) + 1)$-边着色。

在介绍过程 边着色之前，我们先回顾 Atallah 和 Vishkin 提出的一个结果 [AV84]，该结果用于在欧拉图中计算欧拉回路；再陈述 Lev、Pippenger 和 Valiant 关于二分图 [LPV81] 的 Δ-边着色结果，并介绍一种对最大度至多为 2 的图进行 (Δ+1)-边着色的算法（该算法的完整描述见附录 C）。

引理 11 (计算欧拉回路 [AV84])。设 $G = (V, E)$ 是一个具有 n 个顶点、m 条边的欧拉图。存在一种确定性 CRCWPRAM 算法，可在 G 内以 $O(\log n)$ 时间找到一条欧拉回路，使用 $O(n + m)$ 个处理器。

引理 *12*（用于二分图 [*LPV*81] 的 Δ-边着色算法）。设 $G = (V, E)$ 是一个有 n 个顶点、m 条边且最大度为 Δ 的二分图。存在一种确定性 *EREW PRAM* 算法，能够在 $O\left(\log^2 n \cdot \log \Delta\right)$ 时间内，使用 $O(n + m)$ 个处理器对 G 进行 Δ-边着色。

定理5 ((Δ + 1)-最大度至多为 2 的图的边着色)。 设 $G = (V, E)$ 是一个有 n 个顶点、 m 条边且最大度为 $\Delta \leq 2$ 的图。存在一种算法，能够在 $O(\log n)$ 时间内，使用 $O(m)$ 个处理器对 G 进行$(\Delta + 1)$-边着色。

现在我们可以介绍并行 (Δ+1)-边着色算法了。该算法的详细说明见边着色过程。

边着色过程 ($G = (V,\ E)$)

步骤 1. 如果图 G 不是欧拉图，我们就在 x 中添加一个虚拟节点，并将其连接到 G中的所有奇度顶点。将此图记为 $G^* = (V^*, E^*)$，注意到 G^* 是欧拉图。接下来，我们按如下方式为 $V(in) = \left\{v^{(in)} \mid v \in V\right\}$ 和 $V(out) = \left\{v^{(out)} \mid v \in V\right\}$ 构造二分辅助图 $G_B = \left(V(in) \cup V(out)\ E_B\right)$: 我们利用引理 11（归功于 [AV84]）计算 G^* 中的一条有向欧拉回路，并对该回路中的每条有向边 v, u（对于 $u, v \neq x$），在 G_B中定义一条边 $\left(v^{(out)}, u^{(in)}\right)$。注意， G_B 的最大度为 $\left\lceil \frac{\Delta}{2} \right\rceil$。

步骤 2。 对边进行着色 G_B，使用 $\left\lceil \frac{\Delta}{2} \right\rceil$ 颜色，采用引理 12中的算法，该算法归因于 [LPV81]。

步骤 3。 令 $h = \left\lceil \log \left\lceil \frac{\Delta}{2} \right\rceil \right\rceil = \left\lceil \log \frac{\Delta}{2} \right\rceil$，并令 $p = 2^h$。（等式 $\left\lceil \log \left\lceil \frac{\Delta}{2} \right\rceil \right\rceil = \left\lceil \log \frac{\Delta}{2} \right\rceil$ 对任意整数 $\Delta \geq 2$均成立。）

步骤 4。 对于 $i \in \left\{1、\ 2、\ \ldots\ldots、\ \left\lceil \frac{\Delta}{2} \right\rceil \right\}$，令 M_i 为被着色为 i 的边的集合。构造一个子图 $G_i^{(0)} = \left(V, E_i^{(0)}\right)$，其中对于每条边 $\left(u^{(out)}, v^{(in)}\right) \in M_i$，$(u,\ v) \in E_i^{(0)}$。注意，由于$M_i$ 是 G_B 中的一个匹配，因此 $\Delta\left(G_i^{(0)}\right) \leq 2$ （对于每个顶点 v，至多有一条边与 $v^{(in)}$ 关联，且至多有一条边与 $v^{(out)}$ 关联）。

对于 $i \in \left\{\left\lceil \frac{\Delta}{2} \right\rceil + 1,\ \ldots,\ p\right\}$，令 $G^{(0)}$
i 为一个虚拟空图。（我们定义这些图以简化记法。）

步骤5。**对于**每个 $1 \leq i \leq p$ 并行**执行**计算一个 $\left(\Delta\left(G_i^{(0)}\right) + 1\right)$-边着色 $\varphi_i^{(0)}$ 的 $G_i^{(0)}$（见定理5）。**结束循环**

步骤6。**对于** 每个 $0 \leq k \leq h - 1$，**对于** 每个 $1 \leq i \leq \frac{p}{2^{k+1}}$ 并行执行 ▷ p 在步骤 3 中定义。(1) 定义 $G_i^{(k+1)} = G_{2i-1}^{(k)} \cup G_{2i}^{(k)}$，并对每个 $e \in G_i^{(k+1)}$，通过 $\psi_i^{(k+1)}(e) = \begin{cases} \varphi_{2i-1}^{(k)}(\\ \end{cases}$

$e \qquad e \in G_{2i-1}^{(k)}$) ( ) 定义 $G_i^{(k+1)}$ 的边着色 ψ，▷ $\left|\varphi_{2i-1}^{(k)}\right|$ 是着色 $\varphi_{2i-1}^{(k)}$的颜色集大
$\varphi_{2i}^{(k)}\ e\ + \left|\varphi_{2i-1}^{(k)}\right| \quad e \in G_{2i}^{(k)}$
小。(2) 使用 Reduce-Color过程对 $G_i^{(k+1)}$ 的边 $E_i^{(k+1)}$ 重新着色。将得到的边着色记为 $\varphi_i^{(k+1)}$。**结束循环 结束循环**

步骤 7. 如果 $\left|\varphi_1^{(h)}\right| = \Delta + 2$ **则** ▷ 一种需要再减少一种颜色的特殊情况

$$\varphi = \textsc{Reduce-Color}\left(G, \varphi_1^{(h)}\right)$$

else $\varphi = \varphi_1^{(h)}$

返回 φ

在接下 一个引理中，我们分析递归过程中图 $G_i^{(k)}$ 的结构 .

引理 13 (递归分解)。 设 $G = (V, E)$ 是过程 *Edge*-着色 的一个最大度为 Δ 的输入图。那么对于每个 $k \in \{0, 1, ..., h\}$ 和 $i \in \left\{1, 2, ..., \frac{p}{2^k}\right\}$ $(h = \left\lceil \log \frac{\Delta}{2} \right\rceil$ 和 $p = 2^h)$，我们有 —

—

$$G_i^{(k)} = \bigcup_{j=1}^{2^k} G^{(0)}_{(i-1)\cdot 2^k + j}.$$

$\left(G_i^{(k)}$ 等于步骤 *4* 中定义的 2^k 子图的并集。$\right)$

证明。 我们通过对 k 进行归纳来证明该引理。
归纳基础为 $k = 0$。对于每个 $i \in \{1, 2, ..., p\}$，确实有

$$\bigcup_{j=1}^{1} G^{(0)}_{(i-1)+j} = G_i^{(0)}.$$

对于归纳步骤，考虑某个索引 $k > 0$。根据 $G_i^{(k)}$ 的定义以及归纳假设，

$$G_i^{(k)} = G_{2i-1}^{(k-1)} \cup G_{2i}^{(k-1)} = \left(\bigcup_{j=1}^{2^{k-1}} G^{(0)}_{(2i-2)\cdot 2^{k-1}+j}\right) \cup \left(\bigcup_{j=1}^{2^{k-1}} G^{(0)}_{(2i-1)\cdot 2^{k-1}+j}\right) = \bigcup_{j=1}^{2^k} G^{(0)}_{(i-1)\cdot 2^k+j}.$$

□

在下一个引理中，我们分析递归过程中边着色 $\varphi_i^{(k)}$ 以及子图 $G_i^{(k)}$ 的颜色集大小。回顾 $h = \left\lceil \log \frac{\Delta}{2} \right\rceil$ 和 $p = 2^h$. —

引理 14 （递归子图中的颜色集大小）。 令 $G = (V, E)$ 为边着色过程的输入图，并令 $k \in \{0, 1, ..., h\}$ 和 $i \in \left\{1, 2, ..., \frac{p}{2^k}\right\}$ 为一对索引。则 $\varphi_i^{(k)}$ 是 $G_i^{(k)}$ 的一个正确的 $\left(\left(\sum_{j=1}^{2^k} \Delta\left(G^{(0)}_{(i-1)\cdot 2^k+j}\right)\right) + 1\right)$-边着色。

证明。 首先，回顾引理 13, $G_i^{(k)} = \bigcup_{j=1}^{2^k} G^{(0)}_{i-1()\cdot 2^k+j}$。我们通过对 k 进行归纳法来证明该引理。

归纳基础为 $k = 0$。在步骤5中，对于每个 $i \in \{1,2, ..., p\}$，我们确实将 $\varphi_i^{(0)}$ 定义为 $G_i^{(0)}$ 的一个适当的 $\left(\Delta\left(G_i^{(0)}\right) + 1\right)$-边着色。
对于归纳步骤，考虑 $k > 0$。我们定义

$$\psi_i^{(k)}(e) = \begin{cases} \varphi_{2i-1}^{(k-1)}(e), & e \in G_{2i-1}^{(k-1)} \\ \varphi_{2i}^{(k-1)}(e) + \left|\varphi_{2i-1}^{(k-1)}\right|, & e \in G_{2i}^{(k-1)} \end{cases}.$$

根据归纳假设,

- $\varphi_{2i-1}^{(k-1)}$ is a proper $\left(\left(\sum_{j=1}^{2^{k-1}} \Delta\left(G_{(2i-2)\cdot 2^{k-1}+j}^{(0)}\right)\right)+1\right)$-edge-coloring of $G_{2i-1}^{(k-1)}$.
- $\varphi_{2i}^{(k-1)}$ is a proper $\left(\left(\sum_{j=1}^{2^{k-1}} \Delta\left(G_{(2i-1)\cdot 2^{k-1}+j}^{(0)}\right)\right)+1\right)$-edge-coloring of $G_{2i}^{(k-1)}$.

因此， $\psi_i^{(k)}$ 是 $\left(\sum_{j=1}^{2^k} \Delta\left(G^{(0)}{}_{(i-1)\cdot 2^k+j}\right)+2\right)$边着色的正常边着色。由于 $G_i^{(k)}=\bigcup_{j=1}^{2^k} G^{(0)}{}_{(i-1)\cdot 2^k+j}$, 我们有

$$\Delta\left(G_i^{(k)}\right) \leq \sum_{j=1}^{2^k} \Delta\left(G_{(i-1)\cdot 2^k+j}^{(0)}\right).$$

因此，根据定理 4， $\varphi_i^{(k)}$ （在边着色过程的步骤6中计算得到）是$\left(\sum_{j=1}^{2^k} \Delta\left(G^{(0)}{}_{(i-1)\cdot 2^k+j}\right)+1\right)$边着色的正常边着色，图为 $G_i^{(k)}$。 □

现在我们可以证明，最终边着色过程只使用 $\Delta+1$种颜色。

引理 15 (正常边着色)。 设 $G=(V,E)$ 是一个最大度为 Δ的图。边着色过程会为 G计算出一个正常的($\Delta+1$)边着色。

证明。 根据引理 14， $\varphi_1^{(h)}$ 是 $\left(\left(\sum_{j=1}^{2^h} \Delta\left(G_j^{(0)}\right)\right)+1\right)$-边着色 G的正常边着色。

接下来我们证明

$$\Delta \leq \sum_{j=1}^{2^h} \Delta\left(G_j^{(0)}\right) \leq \Delta+1.$$

注意，对于每个 $i \in \left\{1,\ 2,\ \ldots\ldots,\ \left\lceil \frac{\Delta}{2} \right\rceil\right\}$，由于 $G_i^{(0)}$ 中的边是由 G_B的正常边着色中的一个颜色类定义的，因此每个 $v \in V$ 在 $G_i^{(0)}$ 中至多有 2 条关联边（至多一条来自 $v^{(in)}$ 的关联边，以及至多一条来自 $v^{(out)}$的关联边）。因此 $\Delta\left(G_i^{(0)}\right) \leq 2$。对于每个$i \in \left\{\left\lceil \frac{\Delta}{2} \right\rceil+1,\ \ldots\ldots,\ p\right\}$，图 $G_i^{(0)}$ 为空图。因此 $\Delta\left(G_i^{(0)}\right)=0$。令 $v \in V$为一个满足 $\deg_G(v)=\Delta$的顶点。于是我们得到,

$$\Delta=\sum_{j=1}^{2^h} \deg_{G_j^{(0)}}(v) \leq \sum_{j=1}^{2^h} \Delta\left(G_j^{(0)}\right)=\sum_{j=1}^{\left\lceil \frac{\Delta}{2} \right\rceil} \Delta\left(G_j^{(0)}\right) \leq \sum_{j=1}^{\left\lceil \frac{\Delta}{2} \right\rceil} 2 \leq \Delta+1.$$

因此， $\varphi_1^{(h)}$ 要么是 $\Delta+1$ 的一个恰当（ G）-边着色，要么是 $\Delta+2$ 的一个恰当（ G）-边着色。因此，根据定理 4，在算法第 7 步中定义的着色 φ是 $\Delta+1$ 的一个恰当（ G）-边着色。

□

接下来，我们分析该算法的时间和工作复杂度。

边着色过程会调用 Reduce-Color过程（在步骤 6(2) 和 7 中）。后者过程取决于参数 λ。我们将以参数 λ 调用的边着色过程记为 Edge-Coloring$_\lambda$。当索引不影响当前所述内容时，将省略该索引。

引理 16（边着色过程的复杂度边着色）。设 $G = (V, E)$ 是一个最大度为 Δ 、森林度为 a 的图。边着色过程$_\lambda$ 需要

$$O\left(\lambda\left(\Delta^2, a\cdot\Delta\right)\cdot\Delta^2\cdot\log n\cdot IST_\lambda\left(n, \Delta^2, a\cdot\Delta\right)\right)$$

时间，使用 $O\left(m\cdot\Delta\cdot ISP_\lambda\left(n, \Delta^2, a\cdot\Delta\right)+m\cdot\Delta\right)$ 个处理器。

证明。我们首先分析步骤 1-5 的复杂度：

- 图 G^* 可以通过为每个顶点 $v \in V$分配一个处理器 p_v 来构造。如果 v 在 O中的度为奇数，处理器 p_v 会在 (1) 时间内将边 (v, x) 添加到图中。此步骤使用 $O(n)$ 个处理器。

- 根据引理 11，计算欧拉回路 C 在 G^* 中需要 $O(\log n)$ 时间，使用$O(n+m)$ 个处理器。

- 为了构造 G_B，我们为欧拉回路 C 中的每条有向边 $\langle u, v\rangle$ 分配一个处理器，该回路定义了 G_B中的边 $\left(u_{(out)}v_{(in)}\right)$ 。此过程需要 $O(1)$ 时间，使用$O(m)$ 个处理器。

- 根据引理12，边着色的计算 φ_B的 G_B需要 $O(\log^2 n\cdot\log\Delta)$ 时间，使用 $O(n+m)$ 个处理器。

- 对于每个 $i \in \{1, 2, ..., p\}$，子图 $G_{(0)\,i}$ 的构造需要 $O(1)$ 时间，使用$O(m)$ 个处理器。具体做法是为每条边 $\left(u_{(out)}v_{(in)}\right)$of G_B 分配一个处理器 $p_{(u_{(out)}v_{(in)})}$。该处理器定义边 $(u, v) \in E_{(0)\,i}$，其中 i 是边$\left(u_{(out)}v_{(in)}\right)$的 φ_B-颜色。

- 由于每个子图 G_i 都满足 $\Delta\left(G_{(0}{}_{i}\right) \le 2$), 根据定理 5，并行计算所有 $O(\Delta)$ 个图 $G_{(0)\,i}$ 的 $(\Delta+1)$-边着色需要 $O(\log n)$ 时间，使用 $O(n\cdot\Delta)$ 个处理器。（经过更仔细的分析，这一步可以使用 $O(n+m)$ 个处理器。不过，对于我们在此的目的，界 $O(n\cdot\Delta)$ 已经足够。）

我们现在分析步骤6的计算复杂度：

- 对于每个 $k \in \{0, 1, ..., h-1\}$ 和 $i \in \left\{1, 2, ..., {}^{p}_{2^{k+1}}\right\}$，构造所有图 $G_{(k+1)\,i}$及其着色 $\psi_{(k+1)\,i}$ 总共需要 $O(1)$ 时间和 $O(m)$ 个处理器。具体做法是为每条边 $e \in E$ 分配一个处理器 p_e 。当 $e \in G_{(k)\,2i-1}$ 或 $e \in G_{(k)\,2i}$ 时，该处理器将边 e 添加到 $G_{(k+1)\,i}$，并据此计算其新的颜色。

- 令 $k \in \{0, 1, ..., h-1\}$。注意，对于每个 $i \in \left\{1, 2, ..., {}^{p}_{2^{k+1}}\right\}$，根据引理 13，我们有$\Delta\left(G_{(}{}^{k+1}_{i}\right) \le \sum_{j=1}^{2^{k+1}}\Delta\left(G_{(0)\,(i-1)\cdot 2^{k+1}+j}\right) \le 2^{k+1}\cdot 2 = 2^{k+2}$。记 $m_{(k+1)\,i}$为图 $G_{(k+1)\,i}$ 中的边数。注意，子图 $G_{(k+1)\,i}$ 两两边不相交，即$\sum_{i=1}^{2^{k+1}}{}^{p} m_{(k+1)\,i} = m$。因此，对于所有 $i \in \left\{1, 2, ..., {}^{p}_{2^{k+1}}\right\}$，根据定理 4，并行执行

在所有图 $G_i^{(k+1)}$ 上使用着色 $\psi_i^{(k+1)}$ 执行 Reduce-Color过程$_\lambda$ 需要

$$O\left(\lambda\left(\left(2^{k+2}\right)^2, a\cdot 2^{k+2}\right)\cdot\left(2^{k+2}\right)^2\cdot\log n\cdot IST_\lambda\left(n,\left(2^{k+2}\right)^2, a\cdot 2^{k+2}\right)\right)$$

时间，使用

$$\sum_{i=1}^{\frac{p}{2^{k+1}}} O\left(m_i^{(k+1)}\cdot 2^{k+2}\cdot ISP_\lambda\left(n,\left(2^{k+2}\right)^2, a\cdot 2^{k+2}\right)+m_i^{(k+1)}\cdot 2^{k+2}\right)=$$
$$O\left(m\cdot 2^k\cdot ISP_\lambda\left(n,2^{2k}, a\cdot 2^{k+2}\right)+m\cdot 2^k\right)$$

个处理器。由此，步骤6需要

$$\begin{aligned}&\sum_{k=0}^{h-1} O\left(\lambda\left(\left(2^{k+2}\right)^2, a\cdot 2^{k+2}\right)\cdot\left(2^{k+2}\right)^2\cdot\log n\cdot IST_\lambda\left(n,\left(2^{k+2}\right)^2, a\cdot 2^{k+2}\right)\right)=\\ &=O\left(\lambda\left(\Delta^2, a\cdot\Delta\right)\cdot\log n\cdot IST_\lambda\left(n,\Delta^2, a\cdot\Delta\right)\right)\cdot\sum_{k=0}^{h-1}\left(2^{k+2}\right)^2=\\ &=O\left(\lambda\left(\Delta^2, a\cdot\Delta\right)\cdot\Delta^2\cdot\log n\cdot IST_\lambda\left(n,\Delta^2, a\cdot\Delta\right)\right)\end{aligned}\tag{3}$$

时间，使用

$$\max_{0\le k\le h-1}\left\{O\left(m\cdot 2^k\cdot ISP_\lambda\left(n,2^{2k}, a\cdot 2^k\right)+m\cdot 2^k\right)\right\}=$$
$$O\left(m\cdot\Delta\cdot ISP_\lambda\left(n,\Delta^2, a\cdot\Delta\right)+m\cdot\Delta\right)$$

个处理器。（方程（3）对任意多项式函数 $\lambda(\cdot)$）成立。

- 根据定理 4，步骤7也需要 $O\left(\lambda\left(\Delta^2, a\cdot\Delta\right)\cdot\Delta^2\cdot\text{对数}\ n\cdot IST_\lambda\left(n,\Delta^2, a\cdot\Delta\right)\right)$ 时间，使用 $O\left(m\cdot\Delta\cdot ISP_\lambda\left(n,\Delta^2, a\cdot\Delta\right)+m\cdot\Delta\right)$ 个处理器。

总而言之，边着色过程需要

$$O\left(\lambda\left(\Delta^2, a\cdot\Delta\right)\cdot\Delta^2\cdot\log n\cdot IST_\lambda\left(n,\Delta^2, a\cdot\Delta\right)\right)$$

使用 $O\left(m\cdot\Delta\cdot ISP_\lambda\left(n,\ \Delta^2,\ a\cdot\Delta\right)+m\cdot\Delta\right)$ 个处理器所需的时间。

□

我们的分析得出以下推论：

推论 1（边着色过程的 性质）。 设 $G=$（V，E）是一个最大度为 Δ 且森林度为 a的图。边着色过程在 $O\left(\lambda\left(\Delta^2,\ a\cdot\Delta\right)\cdot\Delta^2\cdot\log n\cdot IST_\lambda\left(n,\ \Delta^2,\ a\cdot\Delta\right)\right)$ 时间内，使用 $O\left(m\cdot\Delta\cdot ISP_\lambda\left(n,\ \Delta^2,\ a\cdot\Delta\right)+m\cdot\Delta\right)$个处理器，计算出该图的一个正确的（$\Delta$+1）边着色。

在下一个定理中，我们总结使用定理2中用于计算大型独立集的各种算法时，我们的边着色算法的复杂度。

定理 6 (边着色过程的性质 边着色)。 设 $G=(V,E)$ 是一个具有 n 个顶点、 m 条边的图，其最大度为 Δ ，森林度为 a。边着色过程可计算出图 G 的一个适当的$(\Delta+1)$边着色

(1)（使用定理 2*(1)*）在 $O\left(\Delta^4 \cdot \text{对数}^4 n\right)$ 时间内，使用 $O(m \cdot \Delta)$ 个处理器。

(2)（使用定理2*(2)*）在 $O\left(\Delta^4 \cdot \log^2 n + \Delta^6 \cdot \log^2 \Delta \cdot \log n\right)$ 时间，使用 $O(m \cdot \Delta)$ 个处理器。

(3)（使用定理2*(3)*）在 $O\left(a^2 \cdot \Delta^4 \cdot \log \Delta \cdot \log^2 n\right)$ 时间，使用 $O(m \cdot \Delta)$ 个处理器。

(4)（使用定理 2*(4)*）在 $O\left(\Delta^{3+o(1)} \cdot a^{1+o(1)} \cdot \log^2 n\right)$ 时间，使用 $O\left(m \cdot \Delta \cdot \log^{\delta} \frac{\Delta \cdot \log n \log(\Delta \cdot \log n)}{}\right)$ 个处理器，其中任意常数为 $\delta > 0$。

(5)（使用定理 2*(5)*）在 $O\left(\Delta^5 \cdot \log n \cdot \left(\log \Delta + \log n \log \Delta \cdot \log(\Delta \cdot \log n)\right) + \Delta^4 \cdot \log^2 n\right) = O\left(\Delta^5 \cdot \log^2 n\right)$ 时间，使用$O\left(m \cdot \left(\Delta \cdot \log \Delta + \frac{\sqrt{\Delta}}{\cdot \log n \log(\Delta \cdot \log n)}\right)\right)$ 个处理器。

证明。 (1) 在定理 2(1) 的算法中，我们有

$$\begin{aligned} \lambda\left(\Delta^2, a \cdot \Delta\right) &= O\left(\Delta^2\right) \\ IST_\lambda\left(n, \Delta^2, a \cdot \Delta\right) &= O\left(\log^3 n\right) \\ ISP_\lambda\left(n, \Delta^2, a \cdot \Delta\right) &= O\left(\frac{1}{\log n}\right) \end{aligned}$$

因此，根据推论 1，过程边着色 需要

$$O\left(\Delta^2 \cdot \Delta^2 \cdot \log n \cdot \log^3 n\right) = O\left(\Delta^4 \cdot \log^4 n\right)$$

时间，使用 $O(m \cdot \Delta)$ 个处理器。

(2) 在定理 2(2) 的算法中，我们有

$$\begin{aligned} \lambda\left(\Delta^2, a \cdot \Delta\right) &= O\left(\Delta^2\right) \\ IST_\lambda\left(n, \Delta^2, a \cdot \Delta\right) &= O\left(\log n + \Delta^2 \cdot \log^2 \Delta\right) \\ ISP_\lambda\left(n, \Delta^2, a \cdot \Delta\right) &= O(1) \end{aligned}$$

因此，根据推论 1，过程 边着色需要

$$O\left(\Delta^2 \cdot \Delta^2 \cdot \log n \cdot \left(\log n + \Delta^2 \cdot \log^2 \Delta\right)\right) = O\left(\Delta^4 \cdot \log^2 n + \Delta^6 \cdot \log^2 \Delta \cdot \log n\right)$$

时间，使用 $O(m \cdot \Delta)$ 个处理器。

(3) 在定理2(3) 的算法中，我们有

$$\begin{aligned} \lambda\left(\Delta^2, a \cdot \Delta\right) &= O(a \cdot \Delta) \\ IST_\lambda\left(n, \Delta^2, a \cdot \Delta\right) &= O(a \cdot \Delta \cdot \log \Delta \cdot \log n) \\ ISP_\lambda\left(n, \Delta^2, a \cdot \Delta\right) &= O(1) \end{aligned}$$

因此，根据推论1，过程边着色需要

$$O\left(a\cdot\Delta\cdot\Delta^2\cdot\log n\cdot(a\cdot\Delta\cdot\log\Delta\cdot\log n)\right)=O\left(a^2\cdot\Delta^4\cdot\log\Delta\cdot\log^2 n\right)$$

使用 $O(m\cdot\Delta)$ 个处理器的时间。

在定理 2(4) 的算法中，我们有

$$\begin{aligned}\lambda\left(\Delta^2,a\cdot\Delta\right)&=O\left((a\cdot\Delta)^{1+o(1)}\right)\\ IST_\lambda\left(n,\Delta^2,a\cdot\Delta\right)&=O\left(\log^{1+\delta}\Delta\cdot\log n\right)\\ ISP_\lambda\left(n,\Delta^2,a\cdot\Delta\right)&=O\left(\frac{\log^{\delta}\Delta\cdot\log n}{\log(\Delta\cdot\log n)}\right)\end{aligned}$$

因此，根据推论 1，过程边着色 需要

$$O\left((a\cdot\Delta)^{1+o(1)}\cdot\Delta^2\cdot\log n\cdot\left(\log^{1+\delta}\Delta\cdot\log n\right)\right)=O\left(\Delta^{3+o(1)}\cdot a^{1+o(1)}\cdot\log^2 n\right)$$

使用 $O\left(m\cdot\Delta\cdot\left(\frac{\log^{\delta}\Delta\cdot\log n}{\log(\Delta\cdot\log n)}\right)\right)$ 个处理器的时间。

在定理 2(5) 的算法中，我们有

$$\begin{aligned}\lambda\left(\Delta^2,a\cdot\Delta\right)&=O\left(\Delta^2\right)\\ IST_\lambda\left(n,\Delta^2,a\cdot\Delta\right)&=O\left(\Delta\cdot\left(\log\Delta+\frac{\log n}{\log\Delta\cdot\log(\Delta\cdot\log n)}\right)+\log n\right)\\ ISP_\lambda\left(n,\Delta^2,a\cdot\Delta\right)&=O\left(\Delta\cdot\log\Delta+\frac{\sqrt{\Delta}\cdot\log n}{\log(\Delta\cdot\log n)}\right)\end{aligned}$$

因此，根据推论 1，过程 边着色需要

$$\begin{aligned}&O\left(\Delta^2\cdot\Delta^2\cdot\log n\cdot\left(\Delta\cdot\left(\log\Delta+\frac{\log n}{\log\Delta\cdot\log(\Delta\cdot\log n)}\right)+\log n\right)\right)=\\ &O\left(\Delta^5\cdot\log n\cdot\left(\log\Delta+\frac{\log n}{\log\Delta\cdot\log(\Delta\cdot\log n)}\right)+\Delta^4\cdot\log^2 n\right)=O\left(\Delta^5\cdot\log^2 n\right)\end{aligned}$$ [5]

时间为 $O\left(m\cdot\left(\Delta\cdot\log\Delta+\frac{\sqrt{\Delta}\cdot\log n\log(\Delta\cdot\log n)}{\log(\Delta\cdot\log n)}\right)\right)$ 个处理器。

□

[5]左侧表达式中的第一项通常占主导地位，只有在一个很窄的范围内例外 $\omega(1)=\Delta=o(\log\log n)$。

4 $(1+\varepsilon)\Delta$-边着色

本节将介绍并分析我们的 $(1+\varepsilon)\Delta$-边着色算法。该算法使用度拆分方法。我们先介绍度分裂 问题。

度拆分问题。无向度拆分问题旨在将图的边 E 划分为两部分，使每个顶点周围的划分尽可能平衡。具体来说，我们需要将每条边染成红色或蓝色，使得对于每个节点，与其关联的红色边数和蓝色边数之差至多为某个较小的差异阈值κ。换句话说，我们希望找到一个赋值 $q\colon E \to \{1,-1\}$，使得对于每个节点 $v \in V$，都有

$$\left|\sum_{e\in E_v} q(e)\right| \leq \kappa,$$

其中 E_v 表示与 v关联的边的集合。
对于每个 $v \in V$，我们将总和 $\sum_{e\in E_v} q(e)$ 称为 v的平衡。

接下来，我们介绍一种算法，它可以计算出差异至多为 2 的度分裂。该算法需要 $O(\log n)$ 时间，并使用 $O(m)$ 个处理器（另请参见 [IS86]，其中介绍了一种相关算法）。注意，在第 3 节中，我们也使用了一种度分裂程序，将图划分为大约 $\Delta/2$个子图，每个子图的最大度至多为 2。相比之下，这里我们将图仅划分为两个子图，同时确保每个顶点在两个子图中的度大约减半。

接下来，我们陈述两个将在算法中使用的结果。第一个结果是 Atallah 和 Vishkin 提出的算法，用于在 $O(\log n)$ 时间内，使用 $O(m)$ 个处理器在欧拉图中计算欧拉回路 [AV84]（见引理11）。第二个结果是一个算法，用于对长度为 n 的路径进行 2-边着色。该算法使用 $O(\log n)$ 时间和 $O(n)$ 个处理器。后一个算法的完整描述（称为过程 ALTERNATING-COLORING）见附录 C。

引理 17（交替路径着色）。设 $P = (V, E) = (v_0, e_1, v_1, e_2, v_2, \ldots, v_{m-1}, e_m, v_m)$ 是一个具有 n 个顶点且含有 m 条边的路径或偶数长度环。交替着色过程使用两种交替颜色，在 $O(\log n)$ 时间内对 P 进行边着色，并使用 $O(m)$ 个处理器。

对于奇长度环，基本上可以使用相同的过程计算一种 2-边着色，使得所有边的颜色均交替出现，但其中有两条连续的边被着成相同的颜色。

现在我们介绍度拆分算法。该算法的思路源自 Beck 和 Fiala 的一项结果的特殊情形 [BF81]，，他们证明，任何秩为 t （每条超边至多包含 t 个顶点）的超图都存在一种 2-边着色，其差异至多为 $2t-2$。此前的研究 [IS86、GHK+20] 也在并行随机存取机和分布式环境中使用了度拆分算法。对于简单图（$t=2$），该算法的思路如下：

对于一个边数为偶数的连通输入图 $G = (V, E)$，我们向 G 添加一个虚拟节点，并将其连接到图中所有奇度顶点，从而得到图 G'。这样，G' 的所有顶点的度都是偶数，因此可以高效地计算该图的一条欧拉回路。注意，这条回路的长度为偶数。因此，我们可以用两种交替颜色为该回路的边着色，然后删除虚拟节点及其关联边，从而得到 G 的所需边着色，其差异至多为 1。对于边数为奇数的输入图

我们将更仔细地分析该算法，并为 G 的边着色，使差异至多为2。

该算法的描述见定理7。度拆分过程的示意图另见图 11 。

定理 7 （过程 度拆分）。 设 $G = (V, E)$ 是一个有 n 个顶点和 m 条边的图。可在 $O(\log n)$ 时间内，使用 $O(m)$ 个处理器计算出差异为 $\kappa = 2$ 的 G 的度拆分。

证明。 不失一般性，我们假设图 G 是连通的。否则，我们将在每个连通分量上分别应用该算法。

定义图 $G' = (V'E')$，其中，对于虚拟节点 $x \notin V$， $V' = V \cup \{x\}$，且 $E' = E \cup \{(x, v) \mid v \in V$，并且 $\deg_G(v)$ 为奇数$\}$。注意，由于我们只将奇度顶点的度增加了 1，因此对于任意 $v \in V$，它在 G' 中的度 $\deg_{G'}(v)$ 都是偶数。此外，由于$\sum_{v \in V} \deg_G(v) = 2 \cdot |E|$， G 中奇度顶点的数量必为偶数。因此，x （在 G' 中）的度也为偶数。由此， G' 中所有顶点的度都是偶数，我们可以使用引理 11 [AV84] 中的算法，在 $O(\log n)$ 时间内并使用 $O(m)$ 个处理器，在 G' 中计算欧拉回路。接下来，从虚拟节点 x 开始（如果 x 在 G' 中是孤立的，我们就从任意其他顶点开始），我们使用推论 4（见附录 C），在 $O(\log n)$ 时间内并使用 $O(m)$ 个处理器，将环的边染成两种交替颜色——红色和蓝色；但第一条和最后一条边可能染成相同的颜色（如果环的长度为奇数）。这样， G' 中每个顶点的平衡要么为 2（如果从该顶点开始为环着色，且环的长度为奇数），要么为 0（其他情况）。从G'中移除 x （以及与其关联的边）后，其邻居的差异将变为 1，而其他顶点的差异不变。因此，我们得到 G 的一个度拆分，其差异至多为 2；该过程在 $O(\log n)$ 时间内并使用$O(m)$ 个处理器。

□

对于 $i \in \{-1,1\}$，用 $\deg_i(v)$ 表示度拆分中与 v 关联且被着成 i 的边数。由于对于任意 $v \in V$，$|\deg_{-1}(v) - \deg_1(v)| \leq 2$，因此可得

$$\deg_i(v) \leq \frac{\deg(v)}{2} + 1. \tag{4}$$

对于每个 $i \in \{-1,1\}$ 均成立。

利用这个度拆分算法，我们现在描述我们的 $(1+\varepsilon)\Delta$-边着色算法。该算法接收一个图 $G = (V, E)$ 和一个非负整数参数 h 作为输入。首先，它计算输入图 $G = (V, E)$ 的度拆分，然后定义它的两个子图 G_1和 G_2 ，它们的顶点集均为 V，且每个子图的边集由第一步度拆分算法中被着成相同颜色的边构成。接着，我们并行递归地计算 G 的每个子图的着色，并使用不相交的颜色集合并这些着色。参数 h 决定递归深度。在算法的基本情况（$h = 0$）下，它使用定理 6 中的算法计算一个正确的 $(\Delta + 1)$-边着色。

过程 Approx-Edge-Coloring的伪代码如算法 1所示。另请参见图 12 ，其中展示了 Approx-Edge-Coloring过程。

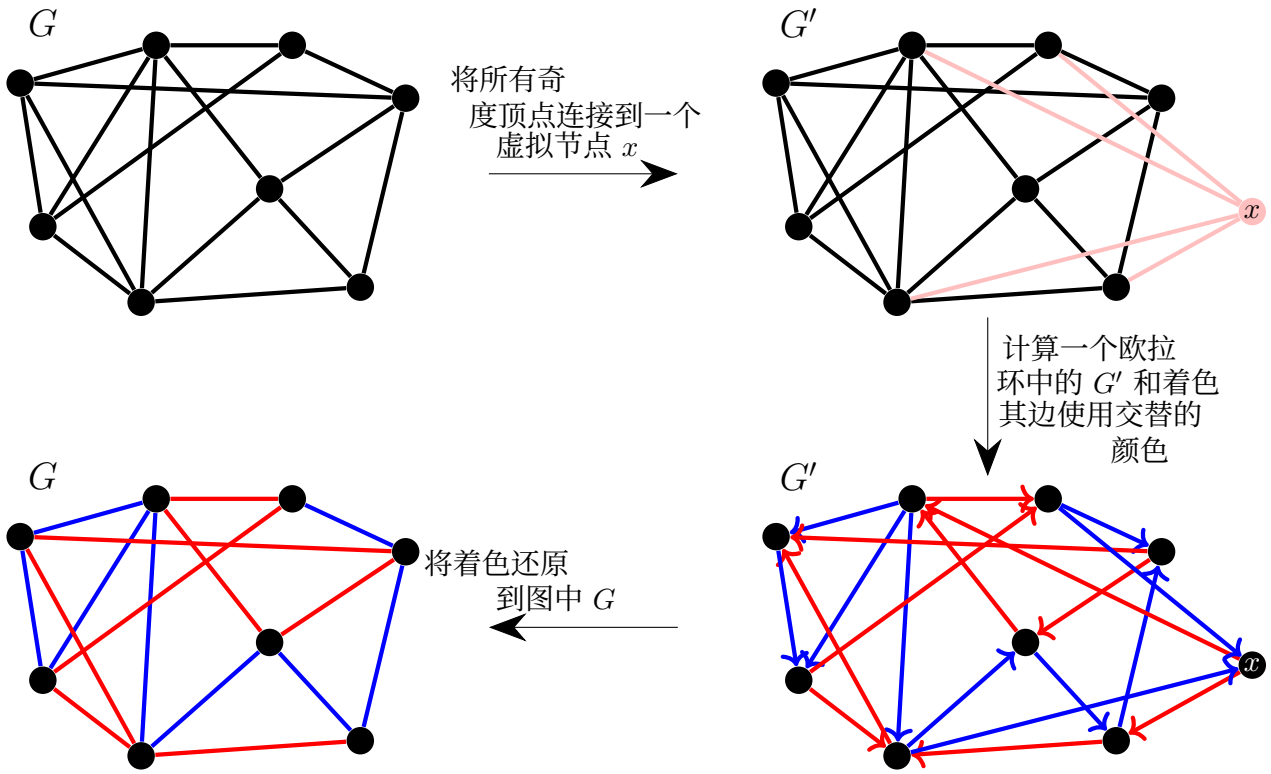

图 11：度拆分过程

算法 1 Aprrox-Edge-Coloring 过程 ($G = (V, E)$ h) 1：如果 $h = 0$，**则 2：使用定理 6 中的算法计算** G **的一个 (**$\Delta + 1$**)-边着色** φ **3：否则 4：** $G_1 = (V, E_1)$ $G_2 = (V, E_2)$ $\leftarrow$ **度分裂(**G**)** $\triangleright$ **参见定理 7 5：对每个** $i \in$ **{1, 2} 并行执行 6：使用 Aprrox-Edge-Coloring 过程(**$G_i, h - 1$**)计算** G_i **的一个着色** φ_i **7：结束循环 8：将** G **的着色定义为** φ，**其中** $\varphi(e) = \begin{cases} \varphi_1(\end{cases}$

e $e \in E_1$ **) ( )**$\triangleright$ $|\varphi_1|$ **是该着色颜色集的大小** φ_1 **9：结束如果 10：返回** φ

$|\varphi_1| + \varphi_2$ e $e \in E_2$

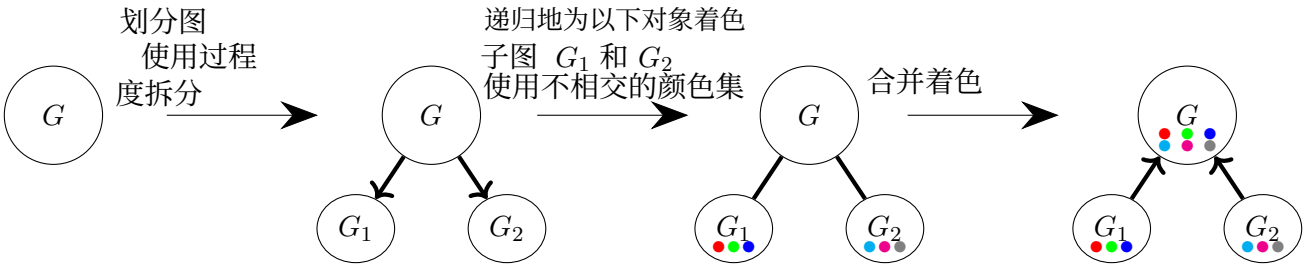

图 12：Aprrox-Edge-Coloring 过程

4.1 算法分析

本节分析 Aprrox-Edge-Coloring 过程。首先，我们证明该过程计算出的着色 φ 是正常边着色。然后，我们给出其使用的颜色数 φ 的上界。此外，我们还将给出该算法所用的时间和处理器数量的上界，并展示颜色数与算法的时间和工作复杂度之间的简单权衡。

在 [Lia95], Liang 提出了一种类似的算法，用于 $(1+\varepsilon)\Delta$-边着色，其运行时间为 $O(\varepsilon^{-4.5}\cdot\log^3(\varepsilon^{-1})\cdot\log n+\varepsilon^{-4}\cdot\log^4 n)$ 时间，且使用 $O\left(n^2+n\cdot\varepsilon^{-3}\right)$ 个处理器。我们对该算法使用的颜色数给出了更简单的分析，并对运行时间和处理器数量推导出了更好的上界，这得益于我们改进的 $(\Delta+1)$-边着色算法（见第 3 节）。

我们先介绍分析中将用到的一些记号。设 $G=(V,\ E)$ 是 Aprrox-Edge-Coloring 过程的一个输入。我们用记号 $G^{(i)}$ 表示经过 i 层递归后计算得到的图。我们用 $m^{(i)}$ 表示其边数，用 $\Delta^{(i)}$ 表示其最大度（例如 $G^{(0)}=G$、 $m^{(0)}=m$和 $\Delta^{(0)}=\Delta$）。

在我们的分析中，影响性能的主要参数之一是输入图的最大度。因此，我们从界定递归每一层输入图的最大度开始分析。

命题 6（关于 $G^{(i)}$ 的最大度界）。设 $G=(V、\ E)$ 和 $h\le\log\Delta$ 是 *Aprrox-Edge-Coloring* 过程的输入，并设 $0\le i\le h$。经过 i 层递归后计算得到的图 $G^{(i)}$ 的最大度 $\Delta^{(i)}$ 满足 $\Delta^{(i)}\le\frac{\Delta}{2^i}+2$。

证明。 我们对 i进行归纳法以证明该结论：
对于 $i=0$，我们有 $G^{(0)}=G$，并且 G 的最大度确实满足 $\Delta\le\Delta+2$。
对于 $i>0$，令 $\Delta^{(i)}$ 为图 $G^{(i)}$ 在递归的第 i层调用该过程时的最大度，并令 $\Delta^{(i-1)}$ 为图 $G^{(i-1)}$ 的最大度。我们在前一递归层将该图拆分，以创建图 $G^{(i)}$。根据归纳假设和方程（4），我们有

$$\Delta^{(i)}\le\frac{\Delta^{(i-1)}}{2}+1\le\frac{\frac{\Delta}{2^{i-1}}+2}{2}+1=\frac{\Delta}{2^i}+2.$$

□

现在我们证明，由 ProcedureAprrox-Edge-Coloring 生成的着色 φ 是正常边着色，并分析其使用的颜色数。

引理 18 (着色的分析 φ)。 设 $G = (V, E)$ 是一个最大度为 Δ 的图。在 G 上调用 *Aprrox-Edge-Coloring* 过程，参数为 $h \leq \log \Delta$ ，可计算出 G 的正常$(\Delta + 3 \cdot 2^h)$边着色 φ。

证明。 我们首先通过对 h 作归纳，证明着色 φ 是正常的:
对于 $h = 0$，根据定理 6 ，着色 φ 是一个正常的 $(\Delta + 1)$-边着色。
对于 $h > 0$，设 $e_1 \neq e_2 \in E$。根据归纳假设，在算法 φ_1 和 φ_2 中第 6 行定义的着色是正常边着色。
1areproperedge-colorings.
如果 e_1， $e_2 \in E_1$，那么 $\varphi(e_1) = \varphi_1(e_1) \neq \varphi_1(e_2) = \varphi(e_2)$。
如果 e_1， $e_2 \in E_2$，那么 $\varphi(e_1) = |\varphi_1| + \varphi_2(e_1) \neq |\varphi_1| + \varphi_2(e_2) = \varphi(e_2)$。
否则，不失一般性， $e_1 \in E_1$、 $e_2 \in E_2$和

$$\varphi(e_1) = \varphi_1(e_1) \leq |\varphi_1| < |\varphi_1| + \varphi_2(e_2) = \varphi(e_2).$$

因此， φ 是正常边着色。
接下来, 我们分析 Aprrox-Edge-Coloring 过程所使用的颜色数 .
回顾一下， $G^{(i)}$ 是经过 i个递归层级后计算得到的图，其最大度至多为$\Delta^{(i)}$。用 $f(i)$表示算法在图 $G^{(i)}$上使用的颜色数。我们通过对 i进行归纳来证明

$$f(i) \leq 2^{h-i} \cdot \left(\frac{\Delta}{2^h} + 3 \right).$$

对于 $i = h$，根据假设，算法使用 $f\ (h)\ = \Delta^{(h)} + 1$ 种颜色，并且根据命题6,

$$f(h) = \Delta^{(h)} + 1 \leq \left(\frac{\Delta}{2^h} + 2 \right) + 1 = \frac{\Delta}{2^h} + 3.$$

对于 $i < h$，我们合并两个大小均为 $f(i+1)$ 的不相交调色板。因此，根据归纳假设，我们有

$$f(i) = 2 \cdot f(i+1) \leq 2 \cdot 2^{h-i-1} \cdot \left(\frac{\Delta}{2^h} + 3 \right) = 2^{h-i} \cdot \left(\frac{\Delta}{2^h} + 3 \right).$$

因此，算法在图 $G = G^{(0)}$ 上使用的颜色数至多为$f(0) \leq 2^h \cdot \left(\frac{\Delta}{2^h} + 3 \right) = \Delta + 3 \cdot 2^h$。

—

□

接下来，我们分析该算法的复杂度。注意，定理 6 中所有结果的时间复杂度都是 n 和 Δ （以及 a）的函数，而 a 可以由 Δ 限定。此外，这些算法所需的处理器数量是 n 和 Δ （以及 a）的函数，其中 a 可以由 Δ 限定，再乘以 m。因此，为简化对我们 $(1+\varepsilon)\Delta$-边着色算法的分析，我们对这些项引入一种记号。对于一个有 n 个顶点、 m 条边的图 $G = (V, E)$，其最大度为 Δ，我们用 $ECT(n, \Delta)$ 表示为 G 计算 $(\Delta+1)$-边着色所需的时间，并用 $m \cdot ECP(n, \Delta)$ 表示这一过程所需的处理器数量。注意，对于定理 6 中的算法，我们有 $ECT(O(n)\ O(\Delta)) = O(ECT(n, \Delta))$，以及 $ECP(O(n)\ O(\Delta)) = O(ECP(n, \Delta))$，并且 ECT 和 ECP 单调递增。此外，注意 $ECT(n, \Delta) = \Omega(\log \Delta \cdot \log n)$，且 $ECP(n, \Delta) = \Omega(1)$。

引理 19 (时间和处理器数量的界). 设 $G=(V,E)$ 是一个有 n 个顶点、m 条边的图，最大度为 Δ。对 G 调用 *Aprrox-Edge-Coloring* 过程，参数为 $h \leq \log \Delta$，则该过程需要 $O\left(ECT\left(n, \frac{\Delta}{2^h}\right)\right)$ 时间，并使用 $O\left(m \cdot ECP\left(n, \frac{\Delta}{2^h}\right)\right)$ 个处理器。

证明。首先注意，在递归的每一层 $0 \leq i < h$，我们都会并行地对该层中的所有 $t_i \leq 2^i$ 子图执行 Procedure Degree- Splitting。设这些子图中的边数为 $m_1^{(i)}, m_2^{(i)}, ..., m_{t_i}^{(i)}$。由于这些子图的边彼此不相交，因此有$\sum_{j=1}^{t_i} m_j^{(i)} = m$。于是，根据定理 7，除基本情况外，递归的每一层都需要$O(\log n)$ 时间，并使用 $\sum_{j=1}^{t_i} O\left(m_j^{(i)}\right) = O(m)$ 个处理器。根据定理 6 和命题 6，基本情况需要 $O\left(ECT\left(n, \Delta^{(h)}\right)\right) = O\left(ECT\left(n, \frac{\Delta}{2^h}\right)\right)$时间，并使用 $\sum_{i=1}^{t_h} O\left(m_i^{(h)} \cdot ECP\left(n, \Delta^{(h)}\right)\right) = O\left(m \cdot ECP\left(n, \frac{\Delta}{2^h}\right)\right)$ 个处理器。由于共有 h 个递归层级（不包括基本情况），且$h \leq \log \Delta$，我们得出该算法的总时间为 $O\left(ECT\left(n, \frac{\Delta}{2^h}\right) + h \cdot \log n\right) = O\left(ECT\left(n, \frac{\Delta}{2^h}\right)\right)$，并使用 $O\left(m \cdot ECP\left(n, \frac{\Delta}{2^h}\right)\right)$ 个处理器。 □

现在我们可以总结 APRROX-EDGE-COLORING 过程的主要性质了。引理 18 和引理19 推出以下推论：

推论 2 （APPROX-EDGE-COLORING 过程的性质）。设 $G=(V,E)$ 是一个最大度为 Δ 的图。在 G 上调用 *Aprrox-Edge-Coloring* 过程，参数为 $h \leq \log \Delta$，可在 $O\left(ECT\left(n, \frac{\Delta}{2^h}\right)\right)$ 时间内使用 $O\left(m \cdot ECP\left(n, \frac{\Delta}{2^h}\right)\right)$ 个处理器计算出 G 的一个合法 $(\Delta + 3 \cdot 2^h)$ 边着色。

为了获得更简单且更实用的权衡，我们令 $h \approx \log(\varepsilon \cdot \Delta)$。

定理 8 （一般性的权衡）。设 $G=(V,E)$ 是一个有 n 个顶点、m 条边且最大度为 Δ 的图，并设 $\frac{1}{\Delta} \leq \varepsilon < 1$。在 G 上调用 *Aprrox-Edge-Coloring* 过程，参数为 $h = \max\left\{\left\lfloor \log\left(\frac{\varepsilon \cdot \Delta}{3}\right)\right\rfloor, 0\right\}$，可在 $O\left(ECT\left(n, \varepsilon^{-1}\right)\right)$ 时间内使用 $O\left(m \cdot ECP\left(n, \varepsilon^{-1}\right)\right)$ 个处理器计算出 G 的一个合法 $(1+\varepsilon)\Delta$ 边着色。

在下一个定理中，我们总结了通过定理 $\Delta + 1$ 中的各种（6）边着色算法可以实现的、我们边着色算法的可能权衡。

定理 9 （具体权衡）。设 $G=(V、E)$ 是一个具有 n 个顶点和 m 条边的图，最大度为 Δ，并且设 $\frac{1}{\Delta} \leq \varepsilon < 1$。在所有权衡中，令 $h = 最大\left\{\left\lfloor 对数\left(\frac{\varepsilon \cdot \Delta}{3}\right)\right\rfloor、0\right\}$，因此，该算法生成 G 的一个正确的 $(1+\varepsilon)\Delta$边着色。我们的算法需要：

(1) （使用定理 *6* （*1*）） $O\left(\varepsilon^{-4} \cdot 对数^4 n\right)$ 时间，使用 $O\left(m \cdot \varepsilon^{-1}\right)$ 个处理器。

(2) （使用定理 *6* （*2*）） $O\left(\varepsilon^{-4} \cdot 对数^2 n + \varepsilon^{-6} \cdot 对数^2 \varepsilon^{-1} \cdot 对数 n\right)$ 时间，使用 $O\left(m \cdot \varepsilon^{-1}\right)$ 个处理器。

(3)（利用定理 *6(4)*，当将森林度 a 界定为 Δ时） $O\left(\varepsilon^{-4-o(1)} \cdot \log^2 n\right)$ 时间，使用$O\left(m \cdot \varepsilon^{-1} \cdot \log^{\delta}\left(\varepsilon^{-1}\right) \cdot \log n \log\left(\varepsilon^{-1} \cdot \log n\right)\right)$ 个处理器，其中任意常数 $\delta > 0$。

(4)（利用定理 *6(5)*） $O\left(\varepsilon^{-5}\cdot \log n\cdot \left(\log \varepsilon^{-1}+\log n \log \varepsilon^{-1}\cdot\log(\varepsilon^{-1}\cdot\log n)\right)+\varepsilon^{-4}\cdot \log^2 n\right)=O(\varepsilon^{-5}\cdot \log^2 n)$ 时间，使用 $O\left(m\cdot \left(\varepsilon^{-1}\cdot \log \varepsilon^{-1}+\varepsilon^{-1/2}\cdot\overline{\log n \log(\varepsilon^{-1}\cdot\log n)}\right)\right)$ 个处理器。

对于任意小的常数 $\varepsilon > 0$，定理 9(2) 的结果可在 $O(\log^2 n)$ 时间内，使用 $O(m)$ 个处理器，提供一个 $(1+\varepsilon)\Delta$-边着色。当 ε 是 n 的缓慢递减函数时，即 $\varepsilon = \frac{1}{\text{poly}(\log\log n)}$，这也是我们能达到的最佳权衡。对于 $\varepsilon \leq \frac{1}{\log^c n}$，当常数 $c > 0$ 满足相应条件时，还存在其他权衡方案。

5 边着色更新算法

本节给出一个用于边着色更新问题的算法。在边着色更新问题中，给定一个图 $G = (V, E)$，其最大度的上界为 $\Delta_{\max}$，以及图 G 的一个合法 $(\Delta_{\max}+1)$-边着色。算法运行期间，图会经历一系列边插入和删除，同时保持其最大度至多为 $\Delta_{\max}$。我们的目标是在整个更新过程中，始终维护图的一个合法 $(\Delta_{\max}+1)$-边着色。具体而言，给定一条新边 $e = (u, v) \notin E$，或删除一条现有边 $e' = (u', v') \in E$，我们的目标分别是维护图 $G \cup \{e\}$ 或 $G \setminus \{e'\}$ 的一个合法 $(\Delta_{\max}+1)$-边着色。

处理边插入。给定一条新边 $e = (u, v) \notin E$ 我们首先构造一个极大扇，并从 u 和 v 中任意选取一个作为中心，使用第2 节中的引理 3。接下来，我们通过在图 $\alpha\beta$上应用 [SV82]提出的连通分量算法，计算其$G_{\alpha,\beta}$该图由使用颜色 α 和 β着色的边所诱导。最后，我们对计算得到的扇和第 3 节中的重新着色扇过程以及路径应用于 $\alpha\beta$-路径，以生成一个正常的 $(\Delta_{\max}+1)$-边着色。

引理 20 (插入的复杂度)。设 $G = (V, E)$ 是一个最大度至多为$\Delta_{\max}$ 的图，并设 φ 是 G的一个正常$(\Delta_{max}+1)$-边着色。设 $e = (u, v) \notin E$ 是一条新边，使得图 $G^+ = G \cup \{(u, v)\}$的最大度至多为 $\Delta_{\max}$。在使用 $O(n)$ 个处理器的情况下，可以在 $O(\log n)$ 时间内计算出 G^+ 的一个正常边着色。

证明。根据第 3 节的引理 2，构造以 v 为中心、包含未着色边 (u, v) 的极大扇 f，需要 $O(\log \Delta_{\max})$ 时间，并使用 $O(\deg(v))$ 个处理器。为了计算与 f 关联的$\alpha\beta$-路径，我们为每个顶点分配其关联边在 $G_{\alpha,\beta}$ 中可能需要的处理器（该子图由颜色为 α 和 β的边诱导）。利用这些处理器，我们计算图 $G_{\alpha,\beta}$ 中包含 v 的连通分量。注意，该图的大小为 $O(n)$。因此，根据引理 3，此计算需要 $O(\log n)$ 时间，并使用$O(n)$ 个处理器。最后，我们使用 Recolor-Fan 过程对扇及其 $\alpha\beta$-路径重新着色。根据定理1，此任务需要 $O(\log n)$ 时间，并使用 $O(n)$ 个处理器。总之，我们得出结论：可以在 $O(\log n)$ 时间内，使用 $O(n)$ 个处理器，为 G^+ 计算出正常边着色。

□

请注意，删除图中的一条边无需进行任何进一步更新。我们在下面的定理中对此进行了总结。

定理 10. 设 $G = (V, E)$ 为一个最大度至多为 $\Delta_{\max}$ 的图，且 φ 为 G的正常$(\Delta_{max}+1)$-边着色。设 $e \notin E$ 为一条新边，使得图$G^+ = G \cup \{e\}$的最大度至多为 $\Delta_{\max}$，并设 $e' \in E$。可以在

$O(\log n)$ 时间内，使用 $O(n)$ 个处理器计算出一个正常边着色；也可以在 $O(1)$ 时间内，使用 $O(1)$个处理器计算出 $G^- = G \setminus \{e'\}$ 的一个正常边着色。

作为推论，我们改进了 [LSH96] 关于边着色更新问题的结果，该结果实现了一个 $O(K \cdot \sqrt{\Delta} \cdot \log^3 \Delta + K \cdot \log n)$ 时间算法，使用 $O(n \cdot \Delta + \Delta^3)$ 个处理器，可用于插入 K 条新边。

推论 3. 令 $G = (V, E)$ 为一个最大度至多为 $\Delta_{\max}$ 的图，并令 φ 为 G的一个正常$(\Delta_{max} + 1)$-边着色。令 G^{+K} 为通过向图 G 添加一组K 条新边而得到的图，且 G^{+K} 的最大度至多为 Δ_{max}。可以在 $O(K \cdot \log n)$ 时间内，使用 $O(n)$ 个处理器，计算出 G^{+K} 的一个正常$(\Delta_{\max} + 1)$-边着色。

6 致谢

We a我们感谢 SODA 2026 的匿名评审提供了有益且详尽的意见。

参考文献

[Ass24] Sepehr Assadi。更快的 Vizing 及近似 Vizing 边着色算法。*arXiv* 预印本 *arXiv:2405.13371*，2024年。[AV84] Mikhail Atallah 和 Uzi Vishkin。并行寻找欧拉回路。《计算机与系统科学杂志》，29(3)：330–337，1984年。[Bar16] Leonid Barenboim。在静态、动态和故障网络中于次线性（关于 Δ）时间内进行确定性（$\Delta+1$）着色。《美国计算机学会学报》（*JACM*），63(5)：1–22，2016年。[BBKO22] Alkida Balliu、Sebastian Brandt、Fabian Kuhn 和 Dennis Olivetti。关于在 δ 的多对数时间内进行分布式边着色。载于 *2022*年 *ACM* 分布式计算原理研讨会论文集，第 15–25 页，2022年。[BCSZ25] Sayan Bhattacharya、Mart´ın Costa、Shay Solomon 和 Tianyi Zhang。通过更短的多步 Vizing 链实现更快的（$\Delta+1$）边着色。载于 第三十六届 *ACM-SIAM* 离散算法年度研讨会论文集（*SODA*），第 4914–4947 页。SIAM，2025年。[BD23] Anton Bernshteyn 和 Abhishek Dhawan。有界度图上 Vizing 定理的快速算法。*arXiv* 预印本 *arXiv:2303.05408*，2023年。[BE08] Leonid Barenboim 和 Michael Elkin。利用 Nash-Williams 分解为稀疏图设计的次对数分布式极大独立集算法。载于 第二十七届 *ACM* 分布式计算原理研讨会论文集，第 25–34 页，加拿大多伦多，2008年。[BE11a] Leonid Barenboim 和 Michael Elkin。多对数时间内的确定性分布式顶点着色。《美国计算机学会学报》（*JACM*），58(5)：1–25，2011年。[BE11b] Leonid Barenboim 和 Michael Elkin。利用有界邻域独立数进行分布式确定性边着色。载于第三十届 *ACM SIGACT-SIGOPS* 分布式计算原理年度研讨会论文集，第 129–138 页，2011年。

[BE13a] Leonid Barenboim 和 Michael Elkin。分布式图着色：基础与最新进展。2013年。[BE13b] Leonid Barenboim 和 Michael Elkin。分布式图着色：基础与最新进展。Morgan & Claypool 出版社，2013年。[BE14] Leonid Barenboim 和 Michael Elkin。分布式图着色的组合算法。《分布式计算》，27：79–93，2014年。[BEG18] Leonid Barenboim、Michael Elkin 和 Uri Goldenberg。低于 Szegedy-Vishwanathan 障碍的局部迭代分布式（$\Delta+1$）着色，以及其在自稳定和受限带宽模型中的应用。载于第三十七届 ACM 分布式计算原理研讨会论文集，第 437–446 页，埃格姆，英国，2018年。[BEK14] Leonid Barenboim、Michael Elkin 和 Fabian Kuhn。线性（关于 Δ）时间内的分布式（$\Delta+1$）着色。《SIAM 计算杂志》，43(1)：72–95，2014年。[BEM17] Leonid Barenboim、Michael Elkin 和 Tzalik Maimon。确定性分布式（$\Delta+o(\Delta)$）边着色，以及具有有界多样性的图的顶点着色。载于第三十六届 ACM 分布式计算原理研讨会论文集，第 175–184 页，2017年。[Ber22] Anton Bernshteyn。（$\delta+1$）边着色的快速分布式算法。《组合论杂志 B辑》，152：319–352，2022年。[BF81] J´ozsef Beck 和 Tibor Fiala。“整数化”定理。《离散应用数学》，3(1)：1–8，1981年。[Chr23] Aleksander Bjørn Grodt Christiansen。多步 Vizing 链的威力。载于 ACM 计算理论研讨会第五十五届年度会议论文集，第 1013–1026 页，2023年。[CN90] Marek Chrobak 和 Takao Nishizeki。平面图的改进边着色算法。《算法杂志》，11(1)：102–116，1990年。[CV86] Richard Cole 和 Uzi Vishkin。具有最优并行链表排序应用的确定性抛硬币。《信息与控制》，70(1)：32–53，1986年。[CY89] Marek Chrobak 和 Moti Yung。平面图的快速边着色算法。《算法杂志》，10(1)：35–51，1989年。[EFF85] Paul Erd˝os、Peter Frankl 和 Zolt´an F¨uredi。有限集合族：其中没有集合被其他 r 个集合的并集覆盖。《以色列数学杂志》，51(1-2)：79–89，1985年。[FR96] Martin F¨urer 和 Balaji Raghavachari。并行边着色近似算法。《并行处理快报》，6(03)：321–329，1996年。[GHK+20] Mohsen Ghaffari、Juho Hirvonen、Fabian Kuhn、Yannic Maus、Jukka Suomela 和 Jara Uitto。改进的分布式度拆分和边着色。《分布式计算》，33(3-4)：293–310，2020年。[GNK+85] Harold N Gabow、Takao Nishizeki、Oded Karvin、Daniel Leven 和 Osamu Terada。图的边着色算法。技术报告，1985年。

[GPV93] Andrew V Goldberg、Serge A Plotkin 和 Pravin M Vaidya。匹配及相关问题的次线性时间并行算法。《算法杂志》，14(2)：180–213，1993年。[GS89] Mark Goldberg 和 Thomas Spencer。并行构造极大独立集。《SIAM 离散数学杂志》，2(3)：322–328，1989年。[IS86] Amos Israeli 和 Yossi Shiloach。改进的极大匹配并行算法。《信息处理快报》，22(2)：57–60，1986年。[JK25] Patryk Jedrzejczak 和 Lukasz Kowalik。Vizing 平面图边着色定理的$O(n\log\ n)$时间算法，2025年。[Kow24] ——Lukasz Kowalik。以 Δ种颜色在准线性时间内对稀疏图进行边着色。arXiv 预印本 arXiv:2401.13839，2024年。[KS87] Howard J Karloff 和 David B Shmoys。边着色问题的高效并行算法。《算法杂志》，8(1)：39–52，1987年。[LHS97] Weifa Liang、George Havas 和 Anne Street。重新探讨并行近似边着色。载于第四届澳大拉西亚并行与实时系统年会（PART'97），（编者：N. Sharda 和 A. Tan），Springer-Verlag 计算机科学讲义，新加坡，第95–103页，1997年。[Lia95] Weifa Liang。近似边着色问题的快速并行算法。《信息处理快报》，55(6)：333–338，1995年。[Lin87] Nathan Linial。分布式图算法：从局部数据得到全局解。载于第28届计算机科学基础年会（FOCS 1987）论文集，第331–335页。IEEE，1987年。[LPV81] Gavriela Freund Lev、Nicholas Pippenger 和 Leslie G Valiant。置换网络中路由的快速并行算法。《IEEE 计算机汇刊》，100(2)：93–100，1981年。[LSH96] Weifa Liang、Xiaojun Shen 和 Qing Hu。边着色及边着色更新问题的并行算法。《并行与分布式计算杂志》，32(1)：66–73，1996年。[MG92] Jayadev Misra 和 David Gries。维津定理的构造性证明。《信息处理快报》，41(3)：131–133，1992年。[NW64] C St JA Nash-Williams。将有限图分解为森林。《伦敦数学学会杂志》，1(1)：12–12，1964年。[Pel00] David Peleg。《分布式计算：一种局部性敏感的方法》。工业与应用数学学会，2000年。[PR01] Alessandro Panconesi 和 Romeo Rizzi。稀疏网络中的一些简单分布式算法。《分布式计算》，14(2)：97–100，2001年。[SV82] Yossi Shiloach 和 Uzi Vishkin。一种 $O(\log n)$并行连通性算法。《算法杂志》，3(1)：57–67，1982年。

[SV19] Hsin-Hao Su 和 Hoa T Vu。论维津定理的局部性。载于第*51*届 *ACM SIGACT* 计算理论年会论文集，第355–364页，2019年。[Viz64] Vadim G Vizing。关于 p-图色类的估计。离散分析，3：25–30，1964年。[ZN94] Xiao Zhou 和 Takao Nishizeki。各类图的边着色和 f-着色。载于国际算法与计算研讨会，第199–207页。Springer，1994年。

构造大量两两不相交的扇

在本附录中，我们说明 Liang 等人 [LSH96] 的算法存在的缺陷。设 $G=(V,E)$ 是一个最大度为 Δ 的图，配备了正常部分边染色；设 F 是 G 中未着色边集合的一个固定子集，并假设 F 是一个匹配。在某个步骤中，算法需要找出一个较大的未着色边集，使这些边能够并行重新着色。也就是说，这个边集中的边，其端点处为中心的扇彼此不相交。在 [LSH96]，中，作者使用图 G_3 来进行这项计算。回顾一下，边 $e_1=(v_1,u_1)\in E$ 和边 $e_2=(v_2,u_2)\in E$ 在 G 中的距离定义为

$$\operatorname{dist}_G(e_1,e_2)=\min\left\{\operatorname{dist}_G(u_1,u_2),\operatorname{dist}_G(u_1,v_2),\operatorname{dist}_G(v_1,u_2),\operatorname{dist}_G(v_1,v_2)\right\}.$$

图 $G_3=(V_3,E_3)$ 在 [LSH96] 中定义为 $V_3=F$；对于两条边 $e_1=(v_1,u_1)\in F$ 和 $e_2=(v_2,u_2)\in F$，边 (e_1,e_2) 属于 E_3，当且仅当 (u_1,u_2)、(u_1,v_2)、(v_1,u_2)、(v_1,v_2) 中至少一条边属于 E。换言之，$(e_1,e_2)\in E_3$ 当且仅当 $\operatorname{dist}_G(e_1,e_2)=1$（因为 F 是一个匹配，所以 $\operatorname{dist}_G(e_1,e_2)\neq 0$）。Liang 等人 [LSH96] 在 G_3 中计算极大独立集，以找出能够并行重新着色的未着色边集。[LSH96] 并未证明 G_3 中的独立集具有这一性质。下面我们说明事实并非如此。具体而言，考虑两条边 $e_1=(v,u_1)$、$e_2=(v^*,u_1^*)$，使得 $dist_G(e_1,e_2)=2$。令 $f(v)=v,u_1,...,u_k$ 和 $f(v^*)=v^*,u_1^*,...,u_l^*$，其中 k、$l\geq 4$ 分别是 v 和 v^* 对应的扇。再假设它们有一个公共顶点，即存在一对索引 i、j、$2\leq i\leq k-2$、$2\leq j\leq l-2$，使得 $u_i=u_j^*$。此外，我们假设顶点 u_i、u_{i+1}、u_j^*、u_{j+1}^* 中没有一个是其对应扇中的特殊顶点，即 $\{u_i,u_{i+1},u_j^*,u_{j+1}^*\}\cap\{x(v)\ y(v)\ z(v)\ x(v^*)\ y(v^*)\ z(v^*)\}=\emptyset$。最后，设 γ 是 $u_i=u_j^*$ 在两个扇 $f(v)$ 和 $f(v^*)$ 中缺失的颜色。由此可知，在交换两个扇中的 $\alpha\beta$-路径并分别旋转这两个扇之后，关联边 (v,u_i)、(u_j^*,v^*) 最终都会被着成 γ，也就是说，所得边着色不再是正常的。图 13 给出了示意图。还需注意，对于 (v,u_1) 的任意端点 w 和 (v^*,u_1^*) 的任意端点 w^*，这两个顶点之间的距离可能都是二。图 14 给出了示意图。

为了解决这个问题，我们定义图 $G^{(F)}$，它还包含 F 中距离为 2 的边对之间的边（在 G 中），即 $G^{(F)}=\left(F,E^{(F)}\right)$，其中对于 $e_1\neq e_2\in F$，边 $(e_1,e_2)\in E^{(F)}$ 当且仅当 $\operatorname{dist}_G(e_1,e_2)\leq 2$。所有细节请参见第 3.2 节。

这一改动对我们算法的主要影响是，$G^{(F)}$ 的最大度现在被界定为 $O(\Delta^2)$，而非 $O(\Delta)$，如 G_3 (在 [LSH96] 中）所示。较大的最大度可能会使 $G^{(F)}$ 中的“较大”独立集缩小（相较于 G_3 中），缩小幅度为 Δ 倍，从而增加算法的迭代次数。

B 第 3 节的一些证明

在本附录中，我们给出一些在第3节中省略的证明。

引理 4 的证明。首先注意，对于 $v\in V$ 的每个扇 $u_1,...,u_k$，以及每个 $i\in\{1,2,...,k\}$，$\operatorname{dist}_G(v,u_i)=1$。设 $e=(v,u_1)\in F$ 且 $e^*=(v^*,u_1^*)\in F$，使得 $(e,e^*)\notin E^{(F)}$。根据 $G^{(F)}$的定义，$\operatorname{dist}_G(e,e^*)\geq 3$，也就是说，特别地，$\operatorname{dist}_G(v,v^*)\geq 3$。不失一般性，

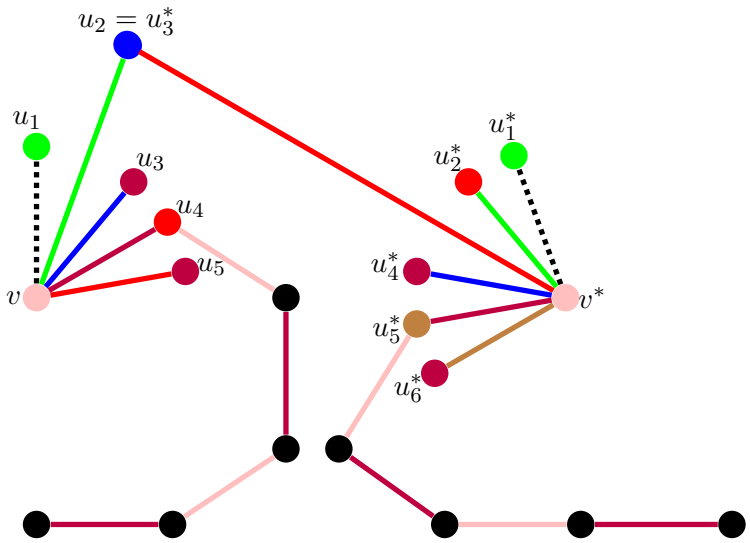

图 13：两个相交扇，由 $(\alpha, \beta) = (\bullet, \bullet)\,\langle u_1, u_2, ..., u_5\rangle$，以 v 为中心，以及$\langle u_1^*, u_2^*, ..., u_6^*\rangle$ 以 v^* 为中心，它们各自的未着色边之间的距离为 2。

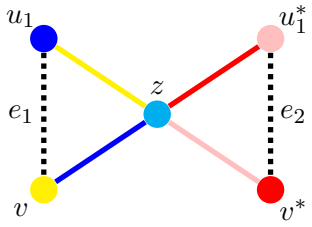

图 14：G 的两条未着色边，可能会被选入 G_3 的独立集，尽管包含它们的极大扇可能在 z 处相交。

反设存在相交的扇$\langle u_1, u_2, \ldots\ldots, u_k\rangle$以 v 为中心，且存在相交的扇$\langle u_1^*, u_2^*, \ldots\ldots, u_l^*\rangle$以 v^* 为中心。令 x 为两个扇的一个公共顶点。于是，根据三角不等式，$\mathrm{dist}_G(v, v^*) \leq \mathrm{dist}_G(v, x) + \mathrm{dist}_G(x, v^*) \leq 2$，这就产生矛盾。因此，任何含有未着色边 e 的扇与任何含有未着色边 e^* 的扇都不相交。□

C 边着色：路径与环

在本附录中，我们介绍一种用于计算路径的交替边着色的算法。此外，我们还设计了另一种边着色算法，并在第 3 节和第 4 节的主要边着色算法中使用它。

设 $P = (V, E)$ 是一条不一定为简单路径的路径。我们也将这条路径记作 $P = (v_0, e_1, v_1, e_2, v_2, \ldots\ldots, v_{m-1}, e_m, v_m)$。交替边着色算法经过 $h = \lfloor \log m \rfloor$ $(m = |E|)$ 次迭代。在第 j 次迭代中，对于每个 v_i，我们并行计算这条路径上与它的距离为 2^j 的顶点，所用信息来自上一轮迭代中已计算出的、与它的距离为 2^{j-1} 的顶点。利用这些信息，每个顶点都计算其到这条路径端点 v_0 的距离；随后，根据每条边与顶点 v_0 之间距离的奇偶性确定其颜色。该算法的完整描述见交替着色过程。

交替着色过程 $(P = (V, E) = (v_0, e_1, v_1, e_2, v_2, \ldots, v_{m-1}, e_m, v_m))$

步骤 1. **对**每个 $i \in \{1, 2, \ldots, m\}$ 并行执行 **对**每个 $1 \leq j \leq h$ **执行**- 使用上一轮迭代中计算出的距离为 2^{j-1} 的顶点，计算这条路径上距离 2^j 的顶点（最多 2 个，如果存在）v_i 的距离为 v_i 的顶点

步骤 2. **对**每个 $i \in \{1, 2, \ldots, m\}$ 并行**执行**- 使用步骤 1 中计算出的距离，计算路径端点 v_0 到 v_i 的距离 $d(v_0, v_i)$

步骤 3. **对**每个 $e = (v, u) \in E$ 并行**执行**- 将 e 着为颜色 $(\min\{d(v_0, v)\ d(v_0, u)\} \bmod 2)$

现在我们来分析该算法。

引理17 (交替路径着色)。设 $P = (V, E) = (v_0, e_1, v_1, e_2, v_2, \ldots, v_{m-1}, e_m, v_m)$ 是一个含有 n 个顶点的路径，或一个含有 m 条边的偶数长度环。交替着色过程使用两种交替颜色为 P 条边着色，耗时 $O(\log n)$，使用 $O(m)$ 个处理器。

证明。首先，注意到该算法确实使用两种颜色对图进行了正确的边着色。此外，在算法的步骤 1 中，对于每个 $v_i \in V$，我们并行计算该路径上距离 v 为 2^j 的顶点（对于每个 $1 \leq j \leq h$）。此过程在 $O(\log n)$ 轮次内完成，每轮需要 $O(1)$ 时间。因此，步骤 1 使用 $O(m)$ 个处理器需要 $O(\log n)$ 时间。对于步骤 2，我们为每个 $v \in V$ 的每次出现分配一个处理器，该处理器在 $O(\log n)$ 时间内计算 v 与 v_0 之间的距离。因此，步骤 2 使用 $O(m)$ 个处理器需要 $O(\log n)$ 时间。最后，对于步骤 3，我们为图中的每条边分配一个处理器，在 $O(1)$ 时间内为该边着色。因此，交替着色过程使用 $O(m)$ 个处理器需要 $O(\log n)$ 时间。□

对于奇长度环，基本上相同的过程会计算出一种 2-边着色：除某两条连续的边使用相同颜色着色外，所有颜色均交替出现。

推论 4（奇长度环的交替着色）。设 $C = (V, E)$ 是一个有 n 个顶点（不一定是简单图）的奇长度环，含有 m 条边。存在一种算法，可在 $O(\log n)$ 时间内，使用 $O(m)$ 个处理器，用两种颜色对 C 进行边着色，使着色呈交替状，但有两条连续的边着成相同的颜色。

对于本附录的最后一个定理，我们回顾 Shiloach 和 Vishkin 提出的、用于计算简单图的连通分量的算法：

引理 3（连通分量 [SV82]）。设 $G = (V, E)$ 是一个 n 个顶点、m 条边的图。存在一种确定性 CRCW PRAM 算法，可在 G 上用 $O(\log n)$ 时间计算连通分量，使用 $O(n + m)$ 个处理器。

现在我们推导出以下定理。

定理5（$(\Delta + 1)$-边着色：最大度不超过 2 的图）。设 $G = (V, E)$ 是一个有 n 个顶点、m 条边且最大度为 $\Delta \leq 2$ 的图。存在一种算法，可在 $O(\log n)$ 时间内，使用 $O(m)$ 个处理器，为 G 计算一个 $(\Delta + 1)$-边着色。

证明。我们首先使用引理 3 中的 $O(\log n)$ 时间和 $O(m)$ 个处理器计算图的非孤立连通分量。注意，G 由简单路径和环组成。对于每个环，我们任意选择一条边并将其从图中删除。将删除边集记为 D。注意，如果 $\Delta = 1$，则 $D = \emptyset$。这样，我们只剩下由简单路径构成的连通分量。根据引理 17，我们可以在 $O(\log n)$ 时间内使用 $O(m)$ 个处理器并行地对它们进行正确的边着色，使用 2 种交替颜色。最后，我们使用颜色 3 对 D 中的删除边进行着色，从而得到 $\Delta + 1$-边着色 G。 □

D 极大路径

在本附录中，我们介绍一种用于在出度至多为 1 的有向图 $G = (V, E)$ 中计算一条极大路径的算法。该算法需要 $O(\log n)$ 时间，并使用 $O(n)$ 个处理器。我们在第 $\Delta + 1$节的（$\Delta + 1$）边着色算法中使用该算法来构建扇。3。

我们先介绍算法中将用到的一些记号。令 $h = \lceil \log n \rceil$。对于出度为 1 的顶点$v \in V$，我们定义 $n(v)$ 为它唯一的出邻点；对于出度为 0 的顶点，我们定义 $n(v) = \perp$。对于 $v \in V$，我们定义 $P(v)$ 为从 $v_0 = v$ 出发的路径；对于 $i \geq 1$，这条路径上的第 i 个顶点 v_i 是该路径上前一个顶点的出邻点，即 $v_i = n(v_{i-1})$（这条路径可能是无限的）。注意，所有从 v 出发的路径都包含在 $P(v)$ 中，并且从 v 出发的极大简单路径是唯一的。我们称这条路径为 v 的极大路径。对于顶点 $u \in V$，以及每个 $v \in V$，我们将 v 在 $P(u)$ 中首次出现的位置称为 v 在 $P(u)$ 中的索引（如果 v 未出现在 $P(u)$ 中，则 v 的索引为 -1）。对于 $v \in V$ 和 $k \geq 0$，我们用 $P(v)|_k$ 表示将 $P(v) = (v_0, v_1, ...)$ 限制为长度 k 后得到的路径，即 $P(v)|_k = (v_0, v_1, ..., v_k)$（如果 $|P(v)| \leq k$，则 $P(v)|_k = P(v)$）。注意，如果 $P(v)$ 有限，那么 $P(v)$ 是一条无法延伸的简单路径（如果我们访问某个顶点不止一次，就会进入一个环，而 $P(v)$ 将是无限的）。因此，在这种情况下，$P(v)$ 就是 v 的极大路径。否则，$P(v)$ 是无限的。为了找到 v 的极大路径，我们希望找到第一条到达此前已在 $P(v)$ 中访问过的顶点的边，并将 $P(v)$ 截取至这条边为止。为此，我们将找到图中索引最大的顶点 k，并将 $P(v)$ 截取为长度 k（截取后的路径即为 v 的极大路径）。

给定一个最大出度为 1 且指定了根顶点r的有向图 $G = (V, E)$，在步骤 1 中，算法计算路径 $P(r)|_{2^h}$（由于极大路径的长度至多为 $n \leq 2^h$，因此 $P(r)|_{2^h}$ 必定包含 r 的极大路径）。如果这条路径的最后一个顶点的出度为 0，那么这条路径必定是 r的极大路径。否则，这条路径不是简单路径，算法会在步骤 2 中将这条路径限制为 r 的极大路径（通过搜索合适的索引来进行限制）。

Maximal-Path过程$(G = (V, E), r)$

步骤 1. 对每个 $v \in V$并行**执行**

- $n_0(v) \leftarrow \begin{cases} v, & n(v) = \perp \\ n(v), & \text{otherwise} \end{cases}$
- $P_v \leftarrow \begin{cases} \langle v \rangle, & n(v) = \perp \\ \langle v, n(v) \rangle, & \text{otherwise} \end{cases}$

For each $1 \leq i \leq h$ **do** ▷ 计算与 v距离为 2^i 的顶点▷○ 是两条路径的拼接

- $P_v \leftarrow P_v \circ P_{n_{i-1}(v)}$
- $n_i(v) \leftarrow n_{i-1}(n_{i-1}(v))$

步骤 2. 如果 度$_{\text{out}}(n_h(r)) = 0$ **则** ▷ P_r 是有限的，因此它是 r的极大路径

返回 P_r**否则** ▷ 正在查找限制的索引

分配一个大小为 n的空数组 A。**对于**每个 $1 \leq i \leq n$ 并行**执行**

- 使用步骤 1 中为每个 $u \in V$ 和 $0 \le j \le h$ 计算出的指针 $n_j(u)$，计算路径 P_r 的第 i 个顶点 v 。- 将数对（v， i）保存到数组 A 的第 i 个槽位 $A[i]$ 中。**对于**每个 $v \in V$ 并行**执行**- 根据顶点 v 对数组排序，并为每个顶点 v 仅保留最小索引 i ，使得（v， i）存储在 A 中。- 计算数组 A 中剩余数对的最大索引 k 。**返回** $P_r|_k$

我们首先通过展示指针 $n_i(v)$ 和路径 P_v 的性质来进行分析，对于每个 $v \in V$ 和 $0 \le i \le h$。

断言 7。设 $G = (V、E)$ 为一个最大出度至多为 *1* 的有向图，并设 $r \in V$。那么，对于每个 $v \in V$ 和 $0 \le i \le h$，在步骤 *1* 的内层循环第 i 次迭代后计算得到的路径 P_v 是路径 $P(v)|_{2^i}$，且 $n_i(v)$ 是该路径中的最后一个顶点。

证明。对于每个 $v \in V$ 和 $0 \le i \le h$，记 $P_v^{(i)}$ 为步骤 1 的内层循环第 i 次迭代后计算得到的路径。我们将对 i 使用归纳法证明：对于每个 $v \in V$， $P_v^{(i)} = P(v)|_{2^i}$，且 $n_i(v)$ 是该路径中的最后一个顶点。

对于 $i = 0$， $P_v^{(0)} = P(v)|_1$ 且 $n_0(v)$ 确实是该路径中的最后一个顶点。
对于 $i > 0$，根据归纳假设， $P_v^{(i-1)} = P(v)|_{2^{i-1}}$，且 $n_{i-1}(v)$ 是这条路径的最后一个顶点，而 $P_{n_{i-1}}^{(i-1)}(v) = P(n_{i-1}(v))|_{2^{i-1}}$ 是从 $n_{i-1}(v)$ 到 $n_i(v) = n_{i-1}(n_{i-1}(v))$ 的路径。因此，$P_v^{(i)} = P_v^{(i-1)} \circ P_{n_{i-1}}^{(i-1)}(v) = P(v)|_{2^{i-1}} \circ P(n_{i-1}(v))|_{2^{i-1}} = P(v)|_{2^i}$，且 $n_i(v)$ 是该路径的最后一个顶点。
□

现在我们准备好证明该算法的正确性并分析其复杂度。

引理 1 (最大路径算法)。设 $G = (V, E)$ 是一个有 n 个顶点且最大出度不超过 *1* 的有向图，并设 $r \in V$ 为一个顶点。*Maximal-Path*过程在 G 中计算一条以 r 为起点的极大路径，所需时间为 $O(\log n)$ ，使用 $O(n)$ 个处理器。

证明。我们首先说明，该算法确实会返回 r 的极大路径。根据断言 7，算法执行第 1 步后， $P_r = P(r)|_{2^h}$，且 $n_h(r)$ 是这条路径的最后一个顶点。回顾一下， $P(r)|_{2^h}$ 要么是 r 的极大路径（如果路径 $P(r)$ 是有限的），要么包含 r 的极大路径。因此，如果 $\deg_{\text{out}}(n_h(r)) = 0$，那么 $P_r = P(r)|_{2^h} = P(r)$ 是有限的，并且是 r 的极大路径。否则，在第 2 步的 for 循环中，我们计算每个顶点在 $P(r)$ 中的索引，并将路径 P_r 限制为图中索引最大的顶点。注意，限制后的路径确实是 r 的极大路径。这是因为，限制后的路径中最后一个顶点 v 的邻居 u 的索引小于该路径最后一个顶点的索引，也就是说，顶点 u 已经在这条路径中出现过。

现在我们分析该算法的复杂度。算法的第 1 步为图中的每个顶点分配一个处理器。该步骤包含 $O(\log n)$ 次迭代，每次迭代需要 $O(1)$ 时间。因此，使用 $O(n)$ 个处理器，这部分需要 $O(\log n)$ 时间。第 2 步中的第一个 for 循环为每个索引 $i \in \{1, 2, ..., n\}$ 分配一个处理器，以利用我们在第 1 步中计算出的指针 $n_i(v)$，计算 P_r (如果存在)中的第 i 个顶点。此过程使用 $O(n)$ 个处理器需要 $O(\log n)$ 时间。对每个顶点 $v \in V$ 计算其最小索引，可以通过排序数组 A 并行完成，使用 $O(n)$ 个处理器需要 $O(\log n)$ 时间。最后，找到具有最大索引的顶点

同样需要 $O(\log n)$ 时间，使用 $O(n)$ 个处理器。因此，该算法使用 $O(n)$ 个处理器，在 $O(\log n)$ 时间内计算出一条从 r 开始的极大路径。 □

E 顶点着色算法

在本附录中，我们将一些已知的分布式顶点着色算法改编为 CRCW PRAM 模型。首先，我们定义分布式 LOCAL 模型的计算方式。

分布式同步消息传递模型。 在分布式计算的 LOCAL 模型中，给定一个 n 顶点图 G，它抽象表示一个通信网络，其中每个顶点都充当一个处理器，边则表示通信链路。每个顶点都被赋予一个唯一的 $\Theta(\log n)$ 比特标识符。初始时，每个顶点都知道自己的标识符，以及 n （顶点数），或许还知道其他一些全局参数，例如 Δ 的最大度 G。计算分轮进行。每一轮中，顶点先执行任意的本地计算，然后同步向所有邻居广播消息。最后，每个顶点都应输出全局解中属于自己的部分（例如自己的颜色，或者在边着色的情形下，与其关联的边的颜色）。此类算法唯一的效率衡量指标是最坏情况下的通信轮数。

我们所采用的第一个算法将森林分解技术与森林的 3-顶点着色相结合。作为实现（$\Delta + 1$）-顶点着色的第一步，我们介绍由 Cole 和 Vishkin 提出的 [CV86]有向森林的 3-顶点着色算法。

定义 14 （有向森林）。 若一个有向图 $F = (V, E)$ 中每个顶点的出度至多为 *1*，且其底层无向图是无环的，则称其为有向森林。 （有向图 $F = (V, E)$ 的底层无向图是图 $G = (V, E')$，其中 $E' = \{(u, v) \mid \langle u, v\rangle \in E\}$。）

E.1 有向森林的 3-顶点着色

在本小节中，我们介绍 [CV86] 提出的有向森林 3-顶点着色算法。首先，我们介绍一种适用于有向森林的 6-顶点着色算法，然后说明如何将其进一步改进为 3-顶点着色。6-顶点着色算法分多个阶段执行。该算法的思路是从某种正常顶点着色开始，该着色可能使用较大的颜色集，并在每个阶段减少所构造顶点着色所使用的颜色数。每个阶段结束后，着色仍保持正常。我们首先介绍“颜色缩减”算法，称为过程Forest-Reduce-Colors，该过程在 6-顶点着色算法的每个阶段执行。在该算法中，图中的每个顶点并行地确定其新颜色。为了保持着色正常，每个顶点的新颜色必须不同于其在森林中的父节点的颜色。

在给出算法的完整描述之前，我们先介绍一些将在算法中使用的符号。设 $F = (V, E)$ 是一片有向森林，φ 是 F 的一个正常顶点着色。对于顶点 $v \in V$，我们用 $p(v)$ 表示输入森林 F 中 v 的后继（父节点）。对于颜色（整数） c，用 c_i 表示 c 的二进制表示中第 i 位。注意，对于 F 中某个树连通分量的根 v，有 $p(v) = \emptyset$。现在介绍过程 Forest-Reduce- Colors。算法示意图见图 15。

Forest-Reduce-Colors 过程（$F = (V, E)$ φ）

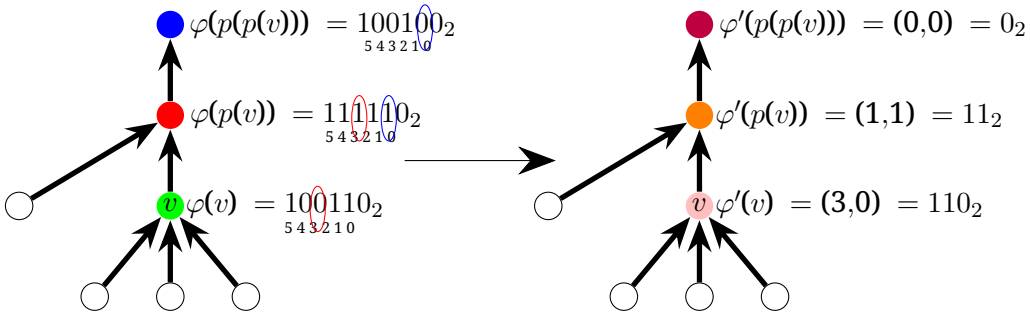

图 15：Forest-Reduce-Colors 过程的示意图

步骤 1. 对于 每个 $v \in V$ 并行**执行**- 找到最小的索引 i，使得 $\varphi(v)_i \neq \varphi(p(v))_i$。

▷ 若 $p(v) = \emptyset$，定义 $i = 0$

- Define $\varphi(v) \leftarrow (i, \varphi(v)_i)$.

▷ $\varphi(v) \leftarrow 2 \cdot i + \varphi(v)_i$

我们首先分析 Forest-Reduce-Colors 过程的复杂度。

引理 21（过程 FOREST-REDUCE-COLORS 的复杂度）。设 $F = (V, E)$ 是一个具有 n 个顶点和 m 条边的有向森林，且 φ 是 F 的正常顶点着色。过程 *Forest-Reduce- Colors* 使用 O（1）时间和 $O(m)$ 个处理器。

证明。设 $v \in V$ 是 F 的一个非孤立顶点。我们首先界定计算最小索引 i 所需的时间，使得 $c(v)_i \neq c(p(v))_i$。这可以通过计算两个颜色的异或，并找出结果中最低有效位来完成。后者是一个单字操作，在 CRCW PRAM 上可在 $O(1)$ 时间内执行。对于每个非孤立顶点，计算新颜色需要 $O(1)$ 时间和 $O(1)$ 个处理器。非孤立顶点的数量与 F 的边数呈线性关系。因此，并行地对所有顶点应用此过程，总共需要 $O(1)$ 时间和 $O(m)$ 个处理器。

□

使用 Forest-Reduce-Colors 过程，我们现在介绍 6-顶点着色算法。如上所述，该算法从某种正常顶点着色开始。为此，我们将定义一个初始的 n-顶点着色，其中每个顶点 v 根据其唯一标识符 id(v)获得一个唯一的颜色。该算法由 $\log^* n$ 个阶段组成。在每个阶段中，它使用 Forest-Reduce-Colors 过程改进着色。

该伪代码见 6-Vertex-Coloring-Forests 过程。

6-Vertex-Coloring-Forests 过程（$F = (V, E)$）

步骤 1. 对每个 $v \in V$ **并行执行**定义 $\varphi(v)$ ←id(v)

步骤 2. 对 $i = 1, 2, \ldots\ldots, \log^* n$ **执行** Forest-Reduce-Colors(F, φ)

对于 $i \in \{0, 1, 2, \ldots, \log^* n\}$，我们将 φ_i 记作着色 φ 在第 i 次 for 循环迭代（步骤 2）之后的结果。我们首先分析每种着色所使用的颜色集大小 φ_i。例如，下一条引理的证明见 [Pel00],第 7 章。

引理 22 (颜色集的大小). 设 $F = (V, E)$ 是一个 n-顶点有向森林。对于每个 $i \in \{0, 1, 2, ..., \log^* n\}$, 我们有 $|\varphi_i| = O\left(\log^{(i)} n\right)$。

现在我们已准备好分析 6-Vertex-Coloring-Forests 过程的复杂度.

引理 23 (6-Vertex-Coloring-Forests 过程的复杂度6-Vertex-Coloring-Forests)。设 $F = (V, E)$ 是一个 n-顶点、 m-边的有向森林。*6-Vertex-Coloring-Forests* 过程的第 *1* 步需要 $O(1)$时间和 $O(n)$ 个处理器。第 *2* 步，即其余部分，需要 $O(\log^* n)$ 时间，并使用 $O(m)$ 个处理器。（注意， m可能显著小于 n。）

证明。首先，注意第 1 步中的构造 φ 需要 $O(1)$ 时间，并使用 $O(n)$ 个处理器。根据引理 21，对于每个 $i \in \{1, 2, ..., \log^* n\}$，for 循环中的第 i次迭代需要$O(1)$ 时间，并使用 $O(m)$ 个处理器。因此，总体而言，Forest-Reduce-Colors 过程的第 2 步需要 $\sum_{i=1}^{\log^* n} O(1) = O(\log^* n)$ 时间，并使用 $O(m)$ 个处理器。

□

该算法的正确性可由 Cole 和 Vishkin 算法 [CV86]的正确性推出。我们在以下定理中总结这一结果。

定理 11 (6-Vertex-Coloring-Forests 过程的性质)。设 $F = (V, E)$ 是一个n-顶点、 m-边的有向森林。*6-Vertex-Coloring-Forests* 过程在 F上计算出一个 *6*-顶点着色，所需时间为 $O(\log^* n)$，使用 $O(n)$ 个处理器。此外，该过程的步骤 *1* 使用 $O(n)$ 个处理器，所需时间为$O(1)$；步骤 *2*（即该过程的其余部分）使用 $O(m)$ 个处理器，所需时间为 $O(\log^* n)$。

现在我们继续说明如何将这个 6-顶点着色改进为所需的 3-顶点着色。为此，我们给出一个简单算法：给定一个整数 $k \geq 4$ 和一个有向森林的k-顶点着色，该算法会减少着色所用的一种颜色，即计算出森林的一个合法的 $(k-1)$-顶点着色。该算法包含两个步骤。第一步，将所有顶点重新着色为其父节点的颜色。注意，这一步保持着色的合法性，并确保每个顶点的所有邻居最多使用两种不同的颜色（一种是其父节点的颜色，另一种是其子节点的颜色，也就是该顶点原来的颜色）。利用这一性质，所有颜色为 k 的顶点并行地从 $\{1, 2, 3\}$ 中选择一种邻居未使用的颜色。这样，我们就去掉了颜色 k，并得到一个新的合法 $(k-1)$-顶点着色。该算法的伪代码见过程 Forest-Reduce-Color。

过程 森林颜色规约 $(F = (V, E), \varphi)$

步骤 1. 对每个 $v \in V$ 并行 **执行**
令 $\varphi'(v) \leftarrow \varphi(v)$ 定义 $\varphi(v) \leftarrow \varphi(p(v))$ ▷ 若 $p(v) = \emptyset$， v 从$\{1、2、3\} \setminus \{\varphi(v)\}$**若** $\varphi(v) = |\varphi|$ **执行**定义 $\varphi(v)$ 为来自$\{1、2、3\} \setminus \{\varphi'(v)\ \varphi(p(v))\}$的任意一种颜色

请注意，可以通过为每条森林边（或等价地，为每个非孤立顶点）分配一个处理器来实现该算法，每个处理器在 $O(1)$ 时间内完成所有计算。我们在下一个引理中总结这一结果。

引理 24（过程 FOREST-REDUCE-COLOR 的复杂度）。设 $F=(V,E)$ 为一个 n-顶点有向森林，$k\geq 4$ 为一个整数，且 φ 为 F的一个 k-顶点着色。过程 *Forest-Reduce- Color* 使用 O（1）时间，并需要 O（m）个处理器。

现在，我们已经准备好整合上述所有部分，设计一种对有向森林进行 3-顶点着色的算法。

森林 3-顶点着色过程 ($F=(V,\ E)$)

步骤 1. 计算　　出 φ 的 6-顶点着色，使用 6-Vertex-Coloring-Forests 过程(F　　)

步骤 2. 对于 $k=6$，5、4 **执行**
Forest-Reduce-Color(F，φ)

根据引理 23，步骤 1 需要 $O(\log^* n)$ 时间，使用 $O(n)$ 个处理器；步骤 2 的每次迭代需要 O (1) 时间，使用 $O(n)$ 个处理器。因此，3-顶点-森林着色过程需要 $O(\log^* n)$ 时间，使用 $O(n)$ 个处理器。此外，除了过程 6-VERTEX-COLORING-FORESTS(将每种颜色初始化为标识号) 的步骤 1 之外，算法的所有步骤都需要 $O(m)$ 个处理器。算法的正确性由 Cole 和 Vishkin 算法的正确性 [CV86]保证。我们在下面的定理中总结这一结果。

定理 12（3-顶点着色的性质）。设 $F=(V,E)$ 是一个 n-顶点、m-边的有向森林。*3*-顶点-森林着色过程在 O（$\log^* n$）时间内计算出 F 的 *3*-顶点着色，使用 O（n）个处理器。此外，除初始化颜色这一步外，算法的所有步骤均需要 O（m）个处理器。

E.2 将 Linial 分布式算法改编为 并行随机存取机

Linial 算法 [Lin87] 可在 O（Δ^2）-顶点着色的分布式时间为 $\log^* n+O$（1）。我们现在说明，如何在并行随机存取机中高效实现该算法，并分析其复杂度。

该算法基于 Δ-无并集集合系统的一个构造，该构造由 [EFF85] 提出。

定义 15（Δ-无并集集合系统）。对于正整数参数 m 和 Δ，若一个定义在底集 *{*1，2，...，m*}* 上的集合系统 *F* 满足：对于任意 $S_0\in F$ 以及 S_1、S_2、...、$S_\Delta\in F\setminus\{S_0\}$，都有 $S_0\not\subseteq\bigcup_{i=1}^{\Delta}S_i$，则称其为 Δ-无并集的。

Erd˝os 等人 [EFF85] 提出了两种 Δ具有相似参数的无并集集合系统构造。其中一种采用概率方法，另一种则采用代数构造。下面简要介绍其代数构造。

对于素数 q，考虑特征为 q 的域 $GF(q)$ 上的 d 次多项式环（参数 d 和 q 将在下文确定）。我们将按以下方式在底集 {1, 2, ..., m} 上构造 $N=q^{d+1}$ 个集合，$m=q^2$。注意，上述形式的 $N=q^{d+1}$ 个 d 次多项式。对于每个这样的多项式 $p(\cdot)$，令 $S_p=\{(i,p(i))\mid i\in GF(q)\}$。注意，任意两个这样的多项式至多在 d 个点相交。因此，令 $q>\Delta\cdot d$，即可保证该族是 Δ-并自由的。我们将取 $q=O(\Delta\cdot d)$（素数）。鉴于 $N=q^{d+1}$，可得 $q\approx O\left(\frac{\Delta\cdot\log N}{\log(\Delta\cdot\log N)}\right)$。

给定图 G 的一个适当的 N-顶点着色 φ，且其最大度不超过 Δ，为每种颜色 $c \in \{1, 2, ..., N\}$ 关联一个集合 $S_c \in \mathsf{F}$。每个顶点 v 随后会找到一种颜色$\alpha \in S_{\varphi(v)} \setminus \bigcup_{u \in N(v)} S_{\varphi(u)}$，并将 α 设为其新颜色。容易看出（参见 [BE13b], 第 3.10 章），这样一来，只需一轮分布式计算，就能在该图上得到一个适当的 $O(q^2) = O\left(\frac{\Delta^2}{2} \cdot \log^2 N \log(\Delta \cdot \log N)\right)$-顶点着色。重复此过程 $\log^* n + O(1)$ 轮次后，最终可得到一个适当的 $O(\Delta^2)$-顶点着色。[6]

为了在并行随机存取机模型中实现这一过程，我们为每个顶点 v 指定 q 个处理器子集。每个这样的子集包含 $\deg(v)$ 个处理器。现在，对于每个 $i \in GF(q)$，有 $\deg(v)$ 个处理器在 $O(1)$ 时间内检查是否存在一个邻居 u v，其集合 $S_{\varphi(u)}$ 包含元素 $(i, p_{\varphi(v)}(i))$。再用 $O(\log q)$ 时间，处理器便可找到一个索引 i，使得 $(i, p_{\varphi(v)}(i))$ 不属于 $\bigcup_{u \in N(v)} S_{\varphi(u)}$。

总计需要 $O(\log \Delta + \log\log N)$ 时间，以及 $O(|E| \cdot q) = O\left(|E| \cdot \frac{\Delta}{\log n} \log(\Delta \cdot \log n)\right)$ 个处理器。随后将该操作迭代 $\log^* n + O(1)$ 次，但初始颜色数量 $N_1 = n, N_2, ...$ 会迅速减少。因此，Linial 算法的总体时间复杂度为 $O(\log \Delta \cdot \log^* n + \log\log n)$，并使用 $O\left(|E| \cdot \frac{\Delta}{\log n} \log(\Delta \cdot \log n)\right)$ 个处理器。

我们用以下定理总结上述讨论。

定理 13 (对 [Lin87] 的改编)。给定一个有 n 个顶点、m 条边的图 $G = (V, E)$，其最大度为 Δ，该算法可在 $O(\log \Delta \cdot \log^* n + \log\log n)$ 时间内计算出一个适当的 $O(\Delta^2)$ 顶点着色，并使用 $O\left(m \cdot \frac{\Delta}{\log n \log(\Delta \cdot \log n)}\right)$ 个处理器。

Barenboim 和 Elkin [BE08] 将该算法扩展到了有界森林度图 a。具体而言，他们证明，给定图的所有边的一种定向，使得每个顶点有$A = O(a)$ 个父节点，一个类似的算法（他们称之为 Arb-Linial）可在 $\log^* n + O(1)$ 轮次内计算出图的 $O(A^2)$-着色。显然，Arb-Linial 也可以用类似的方式在并行随机存取机上实现。我们在以下推论中总结这一结果。

推论 *5*（[*BE*08], 算法 *Arb-Linial* 的改编）。给定一个具有 a 森林度的 n 个顶点、m 条边的图 $G = (V, E)$，以及图 G 的一种边定向，使得每个顶点至多有 A 个父节点，算法 *Arb-Linial* 可在 $O(\log A \cdot \log^* n + \log\log n)$ 时间内，使用 $O\left(m \cdot \frac{A}{\log n \log(A \cdot \log n)}\right)$ 个处理器，为 G 进行 $O(A^2)$- 着色。

还可以用运行时间换取处理器数量。给定参数 $z, 1 \le z \le q$，我们为每个顶点 v指定 z 个子集，每个子集包含 $\deg(v)$ 个处理器。现在，每个子集负责至多$\lceil \frac{q}{z} \rceil$ 个元素 $i \in GF(q)$。因此，每个子集现在需要 $O\left(\frac{q}{z}\right)$ 时间来检查是否存在某个元素 $i \in GF(q)$（在 $GF(q)$ 的元素中，且该元素由这个处理器子集负责），使得存在 v 的某个邻居 u，其对应的子集 $S_{\varphi(u)}$ 包含 $(i, p_{\varphi(v)}(i))$。此时，我们有 q 个值为 0 或 1 的元素，用来指示对每个 $i \in GF(q)$，$(i, p_{\varphi(v)}(i))$ 是否出现在某个邻居 v 的子集中。我们有 $Z = z \cdot \deg(v)$ 个处理器需要在这个数组中查找一个 0。每个处理器负责一个包含 $\frac{q}{Z}$ 个元素的区段。在 $O\left(\frac{q}{Z}\right)$ 时间内，这 Z 个处理器中的每个处理器都能确定其指定区段是否包含 0。在额外的 $O(1)$ 时间内（回想一下，我们考虑的是

[6]此处描述的算法可直接得到 $O(\Delta^2 \cdot \log \Delta)$-着色。要获得 $O(\Delta^2)$-着色，可再使用一次该方案，并取 $d = 2$。这一额外步骤不会影响所得算法的渐近复杂度。

CRCW PRAM），计算便告结束，并找到一个元素 $i \in GF(q)$，使得 $(i, p_{\varphi(v)}(i))$ 没有出现在 $\bigcup_{u\in\Gamma(v)} S_{\varphi(u)}$ 中。因此，这一步的运行时间为 $O\left(z \cdot \deg(v)\right)$，也就是说，总运行时间为 $O(q)$，处理器总数为 $\sum_{v\in V} z\cdot \deg(v) = O(|E|\cdot z)$。代入 $q = O\left(\Delta \cdot \log n \log(\Delta \cdot \log n)\right)$ 后，运行时间变为 $O\left(\frac{\Delta \cdot \log n}{z} \cdot \log(\cdot \log n) + \log\log n + \log \Delta\right)$。对 $\log^* n$ 个阶段求和（在这些阶段中，初始颜色数量 $N_1, N_2, \ldots$ 会迅速减少），可得总运行时间为 $O\left(\frac{\Delta \cdot \log z}{z} \cdot \log(\cdot \log n) + \log\log n + \log \Delta \cdot \log^* n\right)$。

推论 6（对 [Lin87], 算法的权衡调整）。给定一个具有 n 个顶点、m 条边的图 $G = (V, E)$，其最大度为 Δ，且参数为 $1 \le z \le q = O\left(\frac{\Delta \cdot \log n \log(\Delta \cdot \log n)}{z}\right)$，我们的算法可在

$$O\left(\log \Delta \cdot \log^* n + \log\log n + \frac{\Delta \cdot \log n}{z \cdot \log(\Delta \cdot \log n)}\right)$$

时间内，使用 $O(m \cdot z)$ 个处理器。

推论 5 也可类似地扩展，其中参数 z 的取值范围为 $1 \le z = O\left(\frac{A \cdot \log n \log(A \cdot \log n)}{\cdot}\right)$。

E.3 Linial 算法向缺陷着色的扩展

定义 16（缺陷着色）。若顶点着色 φ 满足以下条件，则称其为 D-缺陷着色（参数为 D）：对于每个顶点 v，由 $\varphi(v)$ 着色的 v 的邻居至多有 D 个。

Barenboim 等人 [BEK14]将 Linial 算法扩展到了缺陷着色。具体而言，他们证明，对于任意参数 $p \le \Delta$，均可在 $\left(\frac{\Delta}{p}\right)$-缺陷 $O(p^2)$-着色可在 $\log^* n + O(1)$ 个分布式轮次内计算得出。接下来，我们概述这一扩展，并说明它也可以在并行随机存取机上高效实现。

定义 17 (ρ-覆盖)。对于一个集合 S_0，以及其他集合 $S_1, S_2, \ldots, S_\Delta$ 和参数 ρ，若每个元素 $x \in S_0$ 都至少出现在 $S_1, S_2, \ldots, S_\Delta$ 中的 ρ 个集合里，我们称集合 $S_1, S_2, \ldots, S_\Delta$ ρ- 覆盖 S_0。

定义 18 (Δ-并集 $(\rho+1)$-无覆盖)。对于一对参数 Δ 和 ρ，若集合系统 F 是 Δ-并集 $(\rho+1)$-无覆盖 的，则对于任意集合 $S_0 \in F$ 以及集合 $S_1, S_2, \ldots, S_\Delta \in F \setminus \{S_0\}$，集合 $S_1, S_2, \ldots, S_\Delta$ 不会对 S_0 进行 ρ-覆盖。

基本步骤。假设我们有一个图 G 的 D-缺陷 N-着色 φ，以及一个含有 N 个集合、定义在底集 $[m]$ 上的 Δ-并集 $(\rho+1)$-无覆盖族 $\mathcal{F}$，其中参数为 D, N, Δ, ρ 和 m。不难看出，在一轮分布式计算中，我们便可以得到图 G 的 $(D+\rho)$-缺陷 m-着色。

[BEK14] 算法以通过 Linial 算法计算得到的一个 $O(\Delta^2)$-着色（0-缺陷）开始，并通过使用适当的参数迭代基本步骤（见上文），经过 $O(\log^* \Delta)$ 轮次对其进行细化。再次考虑集合系统 $\mathcal{F} = \mathcal{F}_{q,d}$，其中每个集合包含 q 个元素。（每个集合 S_p 对应于 d 次多项式 $p(\cdot)$，其定义在 $GF(q)$ 上。它等于 $\{(i, p(i)) \mid i \in GF(q)\}$。）我们有 $|\mathcal{F}| = N = q^{d+1}$。

要让集合 $S_0 \in \mathcal{F}$ 中的每个元素都被覆盖 $\rho+1$ 次，至少需要 $\Delta \ge q/(\rho+1)\, d$ 个其他集合 S_1, S_2, ……, S_Δ。因此我们将 Δ 设得略小于 $q/(\rho+1)\, d$，以保证该集合族是 Δ-并集

$(\rho+1)$-覆盖自由。底集的大小满足 $m=q^2$，因此可得 th at

$$m \le \left(\frac{\Delta+1}{\rho+1}\right)^2 \log^2 N. \tag{5}$$

与 Linial 算法类似，这一方案很容易适配到并行随机存取机（PRAM）环境。再次为每个顶点 v 指定 q 个两两不相交的处理器子集。每个这样的集合都包含 deg(v) 个处理器。对于值 (i,$p_{\varphi(v)}$(i)) $\in S_{\varphi(v)}$，指定给该值i 的 deg(v) 个处理器会检查有多少个集合 $S_{\varphi(u)}$，$u \in N(v)$ 包含数对 (i,$p_{\varphi(v)}$ (i))。如果这个数量不大于 ρ，它们就会举起一个标志。最后，为 v 指定的处理器会找出举起标志的 $i \in GF(q)$。（根据分析可知它必定存在。）总运行时间为 $O(\log q)$，处理器数量为 $O(|E| \cdot q)$。作为 $m = q^2$，由不等式 (5) 可得，运行时间为$O(\log\ \Delta + \log\log\ N)\ = O(\log\ \Delta + \log\log\ n)$，处理器数量为 $O\left(|E| \cdot \frac{\Delta}{\rho} \cdot \log\ n \log(\frac{\Delta}{\rho} \cdot \log\ n)\right)$。由于该算法会执行 $O(\log^* \Delta)$ 轮这一步骤（颜色集逐渐缩小），可以验证总运行时间为 $O(\log\ \Delta \cdot \log^* \Delta + \log\log\ n)$，使用 $O\left(|E| \cdot \frac{\Delta}{\log\ n \log(\Delta \cdot \log\ n)}\right)$ 个处理器（此外，还需要 $O(\log\ \Delta \cdot \log^* n + \log\log\ n)$ 的时间，通过 Linial 算法计算一个 $O(\Delta^2)$-着色）。

我们用以下定理总结上述讨论。

定理 14（缺陷着色的改编 [BEK14]）。给定一个具有 n 个顶点和 m 条边的图$G = (V, E)$，其最大度为 Δ，以及参数 $p\ 1 \le p \le \Delta$，则可以在 O（log $\Delta \cdot \log^* n + \log\log n$）时间内，使用 $O\left(m \cdot \frac{\Delta}{\log\ n \log(\Delta \cdot \log\ n)}\right)$ 个处理器，计算出一个 $\left(\frac{\Delta}{p}\right)$-缺陷的 $O\,(p^2)$-顶点着色。

与定理 13和推论 5类似，这一分析也扩展到了运行时间与处理器数量之间的权衡。

定理 15（[BEK14],权衡的改编）。给定一个具有 n 个顶点和 m 条边的图 $G = (V, E)$，其最大度为 Δ，以及参数 p和 $z\ 1 \le p \le \Delta\ 1 \le z = O\left(\frac{\Delta}{\log\ n \log(\Delta \cdot \log\ n)}\right)$，则可以计算出一个 $\left(\frac{\Delta}{p}\right)$-缺陷的 O（p^2）-顶点着色，计算时间为

$$O\left(\frac{\Delta \cdot \log n}{z \cdot \log(\Delta \cdot \log n)} + \log\Delta \cdot \log^* n + \log\log n\right)$$

时间，使用 $O(m \cdot\ z)$ 个处理器。

定理 14 中的处理器数量还可以进一步减少。注意，算法中唯一使用 $O\left(|E| \cdot \frac{\Delta}{\log\ n \log(\Delta \cdot \log\ n)}\right)$ 个处理器的步骤，是计算 $O\,(\Delta^2)$-着色的步骤（通过 Linial 算法实现）；其他步骤只使用 $O\left(|E| \cdot \frac{\Delta}{\rho} \cdot \log\ n \log(\frac{\Delta}{\rho} \cdot \log\ n)\right)$ 个处理器。另一方面，不难看出，Barenboim 等人提出的分布式算法（见 [BEK14],第 3.2 节）可以直接应用于顶点集 V 的朴素 ıve n-着色，即将每个顶点 $v \in V$ 的颜色指定为标识符 Id(v) 的着色。换言之，计算 $O\,(\Delta^2)$-着色的步骤可以完全跳过。所得着色仍是 $O\left(\left(\frac{\Delta+1}{\rho+1}\right)^2\right)$-着色，且具有 $O(\rho)$-缺陷；不过，分布式运行时间略有增加：[BEK14], 的原始算法版本的运行时间为 1 2 $\cdot \log^* n + O(\log^* \Delta)$，而此版本的运行时间为 $O(\log^* n)$。此版本的并行随机存取机运行时间为 $O\,(\log\Delta \cdot \log^* n + \log\log n)$，因为我们现在应用

$O(\log^* n)$ 个重新着色步骤（使用 $(\rho+1)$-无并集集合系统），而上述版本中有 $O(\log^* \Delta)$ 个重新着色步骤。另一方面，处理器数量现在仅为 $O\left(|E|\cdot \frac{\Delta}{\rho}\cdot \log n \log(\frac{\Delta}{\rho}\cdot \log n)\right)$（而不是 $O\left(|E|\cdot \Delta \cdot \log n \log(\Delta \cdot \log n)\right)$）。现在令 $p = O(\frac{\Delta}{\rho})$，即可得到一个 $O(p^2)$-着色，且具有 $O(\frac{\Delta}{p})$-缺陷，使用 $O\left(|E|\cdot p \cdot \log n \log(p\cdot \log n)\right)$ 个处理器。

总结如下：

定理 16 ([BEK14] 的缺陷着色变体)。 给定一个有 n 个顶点、 m 条边的图 $G=(V,E)$，其最大度为 Δ，参数为 p, $1 \le p \le \Delta$，则可在 $O(\log \Delta \cdot \log^* n + \log\log n)$ 时间内，使用 $O\left(m \cdot p \cdot \log n \log(p\cdot \log n)\right)$ 个处理器，计算出一个 $\left(\frac{\Delta}{p}\right)$-缺陷的 $O(p^2)$-顶点着色。

这一结果也可以有一个权衡版本，类似于定理 15。

定理17 ([BEK14], 的改编版本（更好的权衡）)。 给定一个有 n 个顶点、 m 条边的图 $G=(V,E)$，其最大度为 Δ，参数为 p 和 z， $1 \le p \le \Delta$ $1 \le z = O\left(p \cdot \log n \log(p\cdot \log n)\right)$，则可计算出一个 $\left(\frac{\Delta}{p}\right)$-缺陷的 $O(p^2)$-顶点着色，所需时间为

$$O\left(\frac{p\cdot \log n}{z\cdot \log(p\cdot \log n)} + \log \Delta \cdot \log^* n + \log\log n\right)$$

时间，使用 $O(m\cdot z)$ 个处理器。

E.4 代数着色约简

本节介绍 Barenboim 等人提出的分布式算法的一个版本。 [BEG18]该算法以一个最大度为 Δ 的图 $G=(V, E)$ 作为输入，并配有一个 k-顶点着色 φ，其中 $k = O(\Delta^2)$，输出图 G的一个 $O(\Delta)$-顶点着色。我们将此算法称为过程 Algebraic-Color-Reduction。

在该算法中，我们首先选择一个素数参数 $\sqrt{k} \le p = O(\Delta)$，并将每种颜色$\varphi(v)$（其中 $v \in V$）视为一对 a_v, b_v，其中 $a_v, b_v \in \{0, 1, ..., p-1\}$。然后，每个顶点$v \in V$并行检查是否存在满足 $b_v = b_u$ 的邻居 u 。若存在这样的邻居，则定义$\varphi(v) = a_v, a_v + b_v \pmod p$；否则，定义 $\varphi(v) = 0, b_v$。我们将此过程迭代 p 次。Barenboim 等人 [BEG18] 证明，该过程会产生正常顶点着色，并且可以保证在该过程结束时，对于每个顶点 $v \in V$，都有 $a_v = 0$，也就是说，图最终得到一个 $p = O(\Delta)$- 顶点着色。

为了在并行随机存取机上实现该算法，我们为每条边 $(u, v) \in E$分配一个处理器，由它并行检查是否 $b_v = b_u$，并通知 u 和 v。随后，每个顶点并行地根据其关联边的结果确定自己的新颜色。由于该过程迭代执行$O(p)$次，我们得出，在该过程结束时，可以在 $O(p\log \Delta) = O(\Delta \log \Delta)$时间内，使用 $O(m)$个处理器，得到G 的 $p = O(\Delta)$-顶点着色。我们在下面的定理中总结这一结果。（顶点 v 需要$(O(\log \Delta)$时间来判断是否存在其邻居 u v ，使得与边 (u, v) 对应的处理器设置了一个标志，表明$b_u = b_v$。 ）

定理 18 （对以下算法的改编： [BEG18]）。 给定一个 n 个顶点的 m边图

$G = (V, E)$，其最大度为 Δ，且给定 G 的一个 *proper* $O(\Delta^2)$-顶点着色，则可在 $O(\Delta \cdot \log \Delta)$ 时间内，使用 $O(m)$ 个处理器，计算出 G 的一个 *proper* $O(\Delta)$-顶点着色。

E.5 $(\Delta + 1)$-一般图顶点着色算法

在本小节中，我们介绍一种用于一般图的 $(\Delta+1)$-顶点着色 CRCW PRAM 算法。该算法基于 [PR01] 和 [BEG18]的算法。

我们首先描述 $(\Delta + 1)$-顶点着色算法的一个合并阶段。在这一阶段，我们接收输入图 G的两个边不相交子图的两个正常顶点着色，并将它们合并为这两个子图并集的一个正常顶点着色。作为描述这一合并算法的第一步，我们介绍一个简单过程：给定一个图 $G = (V, E)$，其最大度为 Δ，一个整数 $k > \Delta+1$，以及 G 的一个正常 k-顶点着色 φ，该过程会从着色中减少一种颜色。也就是说，该算法会生成 G 的一个正常 $(k-1)$-顶点着色。该算法只需将每个着色为 k 的顶点 v 重新着色为以下集合中的一种颜色：$\{1、2、\ldots\ldots \Delta + 1\} \setminus \varphi(N(v)) \neq \emptyset$，其中 $\varphi(N(v))$ 是 v 的邻居所用颜色的集合。该算法的伪代码见 Reduce-Color过程。

Reduce-Color过程（$G = (V, E)$ φ）

步骤 1. 对每个 $v \in V$ 并行**执行：如果** $\varphi(v) = |\varphi|$ **则**定义 $\varphi(v)$ 为{1，2，... ...，$\Delta + 1\} \setminus \varphi(N(v))$ 中的任意一个颜色

首先注意到，由于每个顶点 $v \in V$至多有 Δ 个邻居，因此{1，2，.....，Δ+1}\φ（N（v））$\neq\emptyset$, 并且步骤 1 定义良好。接下来的定理将分析 Reduce-Color 过程的复杂度。

定理 19（Reduce-Color过程的性质）。设 $G = (V, E)$是一个最大度为 Δ 的 m-边图，$k > \Delta + 1$ 是一个整数，且 φ 是 G 的 k-顶点着色。那么，*Reduce-Color*过程使用 $O(m)$ 个处理器，在 O（对数 Δ）时间内计算出 G 的一个适当的（$k-1$）-顶点着色。

证明。令 φ' 为 Reduce-Color过程生成的顶点着色。显然，这是一个合法的（$k-1$）顶点着色（例如参见 [Pel00], 第7.1章）。

现在我们分析该算法的复杂度。对于每个颜色为 k的顶点 $v \in V$，我们为 v的每个邻居分配一个处理器，按颜色对这些顶点排序，并在$\{1、2、\ldots\ldots、 \Delta + 1\} \setminus \varphi(N(v))$中选择一个可用颜色，从而计算出它的新颜色。使用 $O(\deg(v))$个处理器（每个邻居对应一个），该过程需要 $O(\log \Delta)$时间。总的来说，Reduce-Color过程需要 $O(\log \Delta)$时间，并使用$\sum_{v \in V} O(\deg(v)) = O(m)$个处理器。□

接下来，我们介绍合并算法。该算法以两个边不相交图 $G_1 = (V, E_1)$ 和 $G_2 = (V, E_2)$ 的两个合法 $(\Delta + 1)$-顶点着色 φ_1 和 φ_2作为输入，并返回它们的并图 $G = (V, E_1 \cup E_2)$ 的一个组合合法 $(\Delta + 1)$-顶点着色 φ，其中 Δ是 G 的最大度上界。该算法首先为每个顶点定义一种新颜色，即该顶点在 φ_1 和 φ_2中的颜色组成的有序对。此着色是一个合法的 $O(\Delta^2)$-顶点着色。然后，我们进行缩减

所有大于 $\Delta+1$ 的颜色，分两步进行。首先，使用附录 ALGEBRAIC-COLOR-REDUCTION中的过程 E.4，我们计算出一个合法的 $O(\Delta)$-顶点着色；随后，通过反复应用过程 REDUCE-COLOR，我们将 $O(\Delta)$ 个额外颜色缩减掉，并得到一个 $(\Delta+1)$-顶点着色。算法的详细说明见过程 MERGE 。

合并过程 $(G_1,\ G_2,\ \varphi_1,\ \varphi_2,\ \Delta)$

步骤 1. 对于每个 $v \in V$ 并行**执行**
Define $\varphi(v) \leftarrow (\varphi_1(v), \varphi_2(v))$ ▷ $\varphi(v) \leftarrow \varphi_1(v) \cdot |\varphi_2| + \varphi_2(v)$

步骤 2. $\varphi \leftarrow$ 代数颜色归约$(G,\ \varphi)$ ▷ See Appendix E.4

步骤 3. 当 $|\varphi| > \Delta + 1$ **时执行**
REDUCE-COLOR$(G_1 \cup G_2,\ \varphi)$

设 $G_1 = (V,\ E_1)$ 和 $G_2 = (V,\ E_2)$ 是两个边不相交的图，分别通过着色 φ_1 和 φ_2进行适当的顶点着色。我们首先证明合并过程生成的着色 φ确实是 $G_1 \cup G_2$的一个适当的 $(\Delta + 1)$-顶点着色，并分析该过程的复杂度。

定理 *20*（合并过程的性质）。设 $G_1 = (V, E_1)$ 和 $G_2 = (V, E_2)$ 是两个边不相交的 n-顶点图，设 Δ 是 $G = G_1 \cup G_2$最大度的上界，且 m 是 G中的边数。设 φ_1 （分别为 φ_2）是 G_1 （分别为 G_2）的一个适当的 $(\Delta + 1)$-顶点着色。合并过程在 $O(\Delta \cdot \log \Delta)$ 时间内，使用 $O(m)$ 个处理器，为 G 计算出一个适当的 $(\Delta + 1)$-顶点着色 φ 。

证明。令 $(u, v) \in E_1 \cup E_2$。由于 φ_1 和 φ_2 分别是 G_1 和 G_2的正常顶点着色，因此要么 $\varphi_1(u) \neq \varphi_1(v)$，要么 $\varphi_2(u) \neq \varphi_2(v)$。因此，$\varphi(v)$（在步骤 1 中定义）满足$\varphi(u) = (\varphi_1(u)\ \varphi_2(u)) \neq (\varphi_1(v)\ \varphi_2(v)) = \varphi(v)$。因此，由于 $|\varphi_1| \leq \Delta + 1$ 和 $|\varphi_2| \leq \Delta + 1$，$\varphi$ 是 G的一个正常 $O(\Delta^2)$-顶点着色。于是，根据定理 18，Procedure Algebraic-Color- Reduction 可以计算出一个正常 $O(\Delta)$-顶点着色。由于我们对着色 φ 反复应用 Reduce-Color过程，直至其使用的颜色不超过 $\Delta + 1$ 种，并且 Reduce-Color过程会减少一种颜色并返回 G的一个正常顶点着色，因此返回的着色 φ 确实是 G的一个正常 $(\Delta + 1)$-顶点着色。

接下来，我们分析该算法的复杂度。对于步骤 1，我们为每个$v \in V$ 分配一个处理器，用于确定其新的颜色。这部分使用 $O(n)$ 个处理器，需要 $O(1)$ 时间。接下来，根据定理 18（来自附录 E.4），步骤 2（代数着色约简过程）使用 $O(m)$ 个处理器，需要 $O(\Delta \cdot \log\ \Delta)$ 时间。注意，在步骤 3 之前，$|\varphi| = O(\Delta)$。根据定理 19，每次执行 Reduce-Color过程都使用 $O(m)$ 个处理器，需要 $O(\log \Delta)$ 时间。因此，合并过程总体上使用 $O(m)$ 个处理器，需要 $O(\Delta \cdot \log \Delta)$ 时间。 □

现在我们可以介绍本节的主要顶点着色算法了。该算法基于分布式算法 [BE14]。我们先介绍该算法的思路。设 $G = (V, E)$为一个输入图。该算法基于分治方法。算法首先将图 G 划分 Δ 为 F_1 个边不相交的有向子森林 F_2、……、 F_Δ。随后，算法并行地使用附录E.1中的过程3-顶点着色森林，快速为每个子森林进行 3-顶点着色。最后，算法使用过程 Δ 3合并Δ+1，将这些 G 个顶点着色合并为输入图的 (

现在我们将更详细地介绍各个阶段。

划分： 在此阶段，我们将边集 E 划分为 Δ 个子集 E_1，E_2，……，E_Δ，它们定义了有向森林 Δ (V，E_1) $G_2^{(0)}$ = (V，E_2) ……，$G_\Delta^{(0)}$ = (V，E_Δ)。对于这部分，我们为每条边分配一个索引 $i \in \{1, 2, \ldots\ldots, \Delta\}$，用于指定它在划分中所属的子集。这些索引将由边的较低 ID 端点分配。每个顶点 $v \in V$ 都会为形如 (v，u) 的边分配一个索引，使得 $id(v) < id(u)$。边的定向将指向其较高 ID 端点。

颜色： 正如我们稍后将看到的，$G_1^{(0)}$、$G_2^{(0)}$、……、$G_\Delta^{(0)}$ 确实都是森林。因此，我们可以并行地计算它们的 3-顶点着色 $\varphi_1^{(0)}$、$\varphi_2^{(0)}$、……、$\varphi_\Delta^{(0)}$，使用第 3-森林顶点着色节中的过程 E.1。

合并： 在这一部分中，我们将使用合并过程，合并上一部分计算出的 Δ着色。我们将着色 $\varphi_1^{(0)}$，$\varphi_2^{(0)}$，……，$\varphi_\Delta^{(0)}$ 两两合并，共进行 $h = \lceil \log \Delta \rceil$ 次迭代。每次迭代中，我们将图两两分组，并合并每对图的着色。经过$h = \lceil \log \Delta \rceil$ 次迭代后，我们将得到 G的一个合法 ($\Delta + 1$)-顶点着色。

现在我们给出该算法的完整描述。

顶点着色过程 ($G = (V, E)$)

步骤 1. 对每个 $v \in V$ 并行**执行**

- 为每条与 v关联且 v 是其较低 ID 端点的边 (v, u)，分配一个来自 {1, 2,..., Δ} 的唯一索引 ind(v, u)。 - 对每个 $i \in \{1, 2, ..., \Delta\}$，令 $E_i = \{e \in E \mid \text{ind}(e) = i\}$，并令 $G_i^{(0)} = (V, E_i)$。 - 将每条边定向至其较高 ID 端点。 - 对每个 $i \in \{\Delta + 1, \Delta + 2, ..., 2^h\}$，令 $G_i^{(0)} = (V, \emptyset)$。

▷ 经验t性图 hs, to simplify ind森林

步骤 2。对于 每个 $i = 1$，2、……，2^h并行**执行**

$\varphi_i^{(0)} \leftarrow$3-顶点着色-森林$\left(G_i^{(0)}\right)$

▷ 对于 $i > \Delta$，着色 $\varphi_i^{(0)}$ 仅使用颜色 0

步骤 3。对于 $i = 1$，2、……、h **执行**

For $j = 1, 2, ..., 2^{h-i}$ **do**

$\varphi_j^{(i)} \leftarrow \textsc{Merge}\left(G_{2j-1}^{(i-1)}, G_{2j}^{(i-1)}, \varphi_{2j-1}^{(i-1)}, \varphi_{2j}^{(i-1)}, \Delta\right)$

接下来，我们分析顶点着色过程。我们首先证明，步骤 1 中定义的图 $G_1^{(0)}$、$G_2^{(0)}$……、$G_\Delta^{(0)}$ 确实是有向森林。

引理 25 （森林分解）。 设 $G =$ （V, E） 为一个图。顶点着色过程的步骤 *1* 中定义的子图 $G_1^{(0)}$、$G_2^{(0)}$、……、$G_\Delta^{(0)}$ 是有向森林。

证明。 考虑一个顶点 $v \in V$和一个索引 $i \in \{1, 2, \ldots\ldots, \Delta\}$。注意，至多只有一条边关联于 v，且在 $G_i^{(0)}$ 中从 v 指向外部。事实上，这样的边获得了索引 i，而 v是其较低 ID 端点。由于每条边都获得了唯一索引，这样的边至多只有一条。此外，由于图中的所有边都指向其较高 ID 端点，图 $G_i^{(0)}$

不包含有向环。注意，在任何无向环中，都有一个出度为 2 的顶点。因此，图 $G_i^{(0)}$ 也不包含无向环。因此， $G_i^{(0)}$ 是一个森林。 □

接下来我们证明，着色 ψ 是一个合法的（$\Delta+1$）-顶点着色。

引理 26（颜色数的一个界）。设 $G=$（V、E）是一个最大度为 $\Delta \geq 2$ 的图。对于每个 $i \in$ {0、1、.....、h} 和 $j \in \left\{1、2、.....、2^{h-i}\right\}$，着色 $\varphi_j^{(i)}$ 是 $G_j^{(i)}$ 的一个合法的（$\Delta+1$）-顶点着色。

证明。我们对 i 使用归纳法来证明该引理。
对于 $i=0$，根据引理 25，对于每个 $j \in \{1、2、.....、\Delta\}$， $G_j^{(0)}$ 是一个有向森林；根据定理 12，$\varphi_j^{(0)}$ 确实是 $G_j^{(0)}$ 的一个合法的 3-顶点着色，特别地，它是 $G_j^{(0)}$ 的一个合法的（$\Delta+1$）-顶点着色。

假设对于每个 $j \in \left\{1、2、...、\frac{\Delta}{2^i}\right\}$，着色 $\varphi_j^{(i)}$ 是 $G_j^{(i)}$ 的一个合法（Δ+1）顶点着色，其中 $i \in$ ∈ {1, 2, ...、$h-1$}。令 $j \in \left\{1、2、...、\frac{\Delta}{2^{i+1}}\right\}$，并按照步骤 3 中的定义合并 $\varphi_j^{(i+1)}=$Merge $\left(G_{2j-1}^{(i)}, G_{2j}^{(i)}, \varphi_{2j-1}^{(i)}, \varphi_{2j}^{(i)}, \Delta\right)$。根据定理 20， $\varphi_j^{(i+1)}$ 确实是 $G_j^{(i+1)}$ 的一个合法（$\Delta+1$）顶点着色。因此，我们得出结论：算法返回的着色 $\varphi_1^{(h)}$ 是 $G_1^{(h)}=G$ 的一个合法（$\Delta+1$）顶点着色。

□

注意，可以省略假设 $\Delta \geq 2$。否则，该图是一个匹配与一个独立集的并集，可以使用 $\Delta+1 \leq 2$ 种颜色，借助 $O(n)$个处理器在 $O(1)$时间内完成着色。最后，我们分析该算法的复杂度。

引理 27（顶点着色过程的复杂度）。设 $G=$（V, E）是一个具有 n个顶点和 m条边的图，其最大度为 Δ。顶点着色过程需要 O（$\log \log n+\Delta \cdot \log^2 \Delta$）时间，并使用 O（m）个处理器。

证明。首先，我们分析边的索引分配。为此，我们为每个 $v \in V$ 及其 v的一个邻居 u 分配一个处理器。这些处理器对 v 中标识符高于 v的邻居进行排序，并将连接 v 与一个标识符较高的邻居 u 的边在排序顺序中的索引赋给该边。此过程需要 $O(\log \Delta)$时间，总共使用 $O(m)$个处理器。注意，在步骤 2 中，所有森林 $G_i^{(0)}$, $i \in \{1, 2, ..., 2^h\}$两两边不相交。在所有这些森林中初始化顶点颜色需要 $O(1)$时间，并使用 $O(n)$个处理器。对于每个$i \in \{1, 2, ..., 2^h\}$，记 $G_i^{(0)}$ 中的边数为 m_i 。在所有这些森林 $G_1^{(0)}, ..., G_{2^h}^{(0)}$ 上并行执行森林 3-顶点着色过程的其余部分，根据定理 12，需要 $O(\log^* n)$ 时间和 $O\left(\bigcup_{i=1}^{2^h} m_i\right)=O(m)$个处理器。最后，根据定理 20，对于每个 $i \in \{1, 2, ..., \lceil \log \Delta \rceil\}$，步骤 3 中的合并过程需要 $O(\Delta \cdot \log \Delta)$时间，并使用 $O(m)$个处理器。因此，总体而言，顶点着色过程需要 $O(\log^* n+\Delta \cdot \log^2 \Delta)$时间，并使用 $O(m+n)=O(m)$个处理器。（不失一般性，我们假设该图是连通的。）

□

我们在下一个定理中总结本节的主要结果。

定理 21 (顶点着色过程的性质顶点着色)。 设 $G=(V, E)$ 是一个具有 n 个顶点、 m 条边且最大度为 Δ 的图。顶点着色过程在 $O(\log^* n+\Delta \cdot \log^2 \Delta)$ 时间内，使用 $O(m)$ 个处理器，为 G 计算一个正确的$(\Delta+1)$顶点着色。

E.6 有界森林度图的简单着色

本节将 Barenboim 和 Elkin 的分布式算法 [BE08] 调整用于对森林度至多为 a 的图进行 $O(a)$-顶点着色。图 $G = (V, E)$ 的森林度定义为$a(G) = \max_{U \subseteq V, |U| \geq 2} \left\{ \left\lceil \frac{|E(U)|}{|U|-1} \right\rceil \right\}$。该算法需要 $O(a \cdot \log n)$ 轮次。

定义*19*（H-分解 [*BE*08]）。对于参数 A，图$G = (V, E)$的 H-分解是一个划分 $V = \bigcup_{i=1}^{\ell} H_i$，其中 $\ell \geq 1$ 满足：对于每个索引 $i \in \{1, 2, ..., \ell\}$，以及每个顶点 $v \in H_i$，v 在 $\bigcup_{j=i}^{\ell} H_j$ 中至多有 A 个邻居。

定理 22（的并行随机存取机计算 H-分解）。设 $G = (V, E)$ 是一个具有 n 个顶点和 m 条边、森林度为 a 的图。计算 G 的、出度为 $A = O$ 的 H-分解（a）需要 $O(\log n \cdot \log \Delta)$ 时间，并使用 $O(m)$ 个处理器。

证明。该算法由 $\ell = O(\log n)$ 个阶段组成。假设在某个阶段 i开始时，每个顶点 v 都知道其在 $V \setminus \bigcup_{j=1}^{i-1} H_j$ 中的度。然后，度至多为 A 的顶点加入集合 H_i。对于加入 H_i 的每个顶点 v，其在 $V \setminus \bigcup_{j=1}^{i} H_j$ 中的每个邻居都会将其度减1。通过为每条边指定一个处理器，可以在 $O(\log \Delta)$ 时间内使用 $O(n + m)$ 个 $= O(m)$ 处理器完成此操作。

□

$O(a)$-顶点着色算法首先计算输入图 $G = (V, E)$ 的 H-分解，其出度为 $A = O(a)$。完成 H-分解后，我们并行计算每个 H-集合 H_i 的 $(A + 1)$-顶点着色 φ_i，对所有 $i \in \{1, 2, ..., \ell\}$ 均如此。在分布式设置中，可以通过 Barenboim 算法 [Bar16] 在 $\tilde{O}\left(\sqrt{A}\right) + \log^* n$ 时间内完成。在 PRAM 设置中，我们已经看到（见定理 21），这可以在 $O(A \cdot \log^2 A + \log^* n)$ 时间内使用 $O(m)$ 个处理器完成。最后是一个重新着色步骤，在此过程中，顶点在 $O(A \cdot \log n) = O(a \cdot \log n)$ 个分布式轮次内计算出一个合法的 $(A + 1)$-顶点着色 ψ。（在 PRAM 设置中，这些轮次各对应一个阶段，其时间和工作复杂度将在下文分析。）

v 中 H_ℓ 的顶点保留其颜色，即，令 $\psi(v) = \varphi_\ell(v)$。接下来，对于每个 H_j，$j \in \{1, 2, ..., \ell - 1\}$，我们从 $H_{\ell-1}$ 开始，迭代执行 $A + 1$个重新着色阶段，直到 H_1。考虑某个索引 $1 \leq j < \ell$，假设 $\bigcup_{h=j+1}^{\ell} H_h$ 中的所有顶点都已重新着色，现在我们要重新着色 H_j中的顶点。对于φ_j的 $A + 1$ 个颜色类，每个颜色类用一个阶段。假设对于某个 $i \in \{0, 1, ..., A\}$，φ_j 的前 i 个颜色类中的顶点都已重新着色，现在我们要重新着色满足$\varphi_j(v) = i + 1$的顶点 $v \in H_j$。这些顶点各自至多有 A 个已重新着色的邻居（位于 $\bigcup_{h=j}^{\ell} H_h$）中。因此，v在 $[A+1] \setminus \left\{ \psi(u) \mid u \in \bigcup_{h=j}^{\ell} H_h \right.$ 是 v 的一个已经重新着色的邻居$\left.\right\}$中有一个可用颜色，而 v 将 $\varphi(u)$设为该颜色。（所有满足$\varphi_j(v) = i + 1$ 的顶点 $v \in H_j$ 都并行执行此操作。注意，由于它们构成一个独立集，得到的着色是正确的。）

接下来，我们分析该算法的 PRAM 运行时间。每当一个顶点 v 使用颜色 $\psi(v)$为自身着色时，它会使用与其关联的 deg(v) 个处理器，从其邻居的颜色集里消除 $\psi(v)$。随后，可以使用 $O(A)$ 个 $=O(a)$ 处理器（每个顶点）在 $O(\log A) = O(\log a)$ 时间内找到可用颜色。因此，总体而言，$(A + 1) \cdot \log n$ 个重新着色阶段需要 $O(A \cdot \log A \cdot \log n) = O(a \cdot \log a \cdot \log n)$ 时间和 $O(|E|)$ 个处理器。因此，该步骤的总体 PRAM 时间为 $O(a \cdot \log a \cdot \log n)$，使用 $O(|E|)$ 个处理器。因此，总体时间为 $O\left(a \cdot \log^2 a + \log \log n\right) + O(a \cdot \log a \cdot \log n) = O(a \cdot \log a \cdot \log n)$。

我们用以下定理总结上述讨论。

定理 *23*（[*BE*08] 的一种改编）。给定一个具有 *arboric- ity* a 的 n 个顶点、 m 条边的图 $G=(V,E)$，可以在 $O((a\cdot\log a+\log\Delta)\cdot\log n)$ 时间内，使用 $O(m)$ 个处理器，计算一个 $O(a)$-顶点着色。

E.7 在并行随机存取机时间 $O(\log n\cdot\log^{O(1)}\Delta)$内计算 $\Delta^{1+o(1)}$-顶点着色

我们将 Barenboim 和 Elkin 的分布式算法 [BE11a]，推广到并行随机存取机设置。该算法给定一个森林度为 a、参数为 $\varepsilon>0$ 的图，可以在 $O(a^{1+\varepsilon})$-顶点着色中，于 $O\left(\frac{1}{\varepsilon}\cdot a^{2\varepsilon}\cdot\log n\right)$轮次内完成计算。

我们先给出以下定义。

定义 20 (任意缺陷着色 [BE11a])。 图 ψ 的顶点着色 G 称为 t- 着色 q-任意缺陷着色 *(*对于一对参数 t 和 q*)*，如果它使用 t 种颜色，并且每个颜色类ψ 所诱导的图的森林度至多为 q。我们也称 q 为该着色 的任意缺陷度 ψ。

算法首先计算一个出度为 $A=O(a)$ 的 H-分解。此步骤需要 $O(\log n)$ 个分布式轮次，并且（根据定理 22）需要 $O(\log n\cdot\log\Delta)$ 的 PRAM 时间，使用 $O(|E|)$ 个处理器。接着，算法并行地对每个 H_i 计算一个 $c\cdot A^{2\varepsilon}$-着色，即 $(A^{1-\varepsilon})$-缺陷着色 φ_i（使用参数 $p=A^{\varepsilon}$），其中 $i\in\{1,2,...,\ell\}$，且常数 $c>0$ 固定。（回顾一下，每个 H_i 内的最大度至多为 A。）根据定理 16（见附录 E.3），此步骤需要 $O(\log a\cdot\log^{*}n+\log\log n)$ 时间，并使用 $O\left(|E|\cdot{}^{a^{\varepsilon}}\cdot\log n\log(a^{\varepsilon}\cdot\log n)\right)$ 个处理器。

现在，算法以如下方式将图重新着色为 A^{ε} 种颜色。（新的着色称为 ψ。）算法依次重新着色每个 H_i（$i\in\{1,2,...,\ell\}$）中的顶点：先重新着色 H_{ℓ}，再重新着色 $H_{\ell-1}$，最后重新着色 H_1。每个阶段都耗费 $c\cdot A^{2\varepsilon}$ 个阶段。对于每个 $j\in\{1,2,...,\ell\}$，专用于 H_j的 $c\cdot A^{2\varepsilon}$ 个阶段中，每个阶段都会将顶点重新着色$v\in H_j$ 为 φ_j的一个给定颜色类。具体而言， H_{ℓ} 的前 A^{ε} 个颜色类中的顶点保留其颜色，也就是说，它们令 $\psi(v)$ 为 $=\varphi_{\ell}(v)$。然后，关于 φ_{ℓ}的颜色类 $A^{\varepsilon}+1$ 中的顶点并行地自行重新着色。每个满足 $\varphi_{\ell}(v)=A^{\varepsilon}+1$ 的 v 至多有 A 个已经重新着色的邻居（其颜色来自 $\{1,2,...,A^{\varepsilon}\}$）。现在，它会选择一种至多被其 $A^{1-\varepsilon}$ 个已重新着色邻居使用的颜色，并令 $\psi(v)$ 为该颜色（根据鸽巢原理，这样的颜色必定存在）。接着，关于 φ_{ℓ}的颜色类 $A^{\varepsilon}+2$ 中的顶点执行相同操作，依此类推.....最后，φ_{ℓ}色 $c\cdot A^{2\varepsilon}$ 的顶点自行重新着色。完成后， $H_{\ell-1}$色 1 的顶点执行相同操作，然后是 $\varphi_{\ell-1}$色 2，依此类推.....最后是 $\varphi_{\ell-1}$色 $c\cdot A^{2\varepsilon}$。随后，算法对 $H_{\ell-2}$ 执行相同操作，.....，直到 H_1。不难看出，所得着色 ψ是一个 A^{ε}着色，且具有 $O(A^{1-\varepsilon})$ 任意缺陷度。（事实上，当顶点 $v\in H_i$ 被着色为 $\psi(v)$ 时，至多有 $A^{1-\varepsilon}$ 个邻居已经被重新着色为相同的 ψ色。此外， v 中还有至多 $A^{1-\varepsilon}$ 个邻居具有相同的φ_i色。这些邻居各自与 v同时重新着色，并且最终也可能被着色为 $\psi(v)$。）此时，算法并行地对每个 A^{ε} 颜色类递归执行。它递归 $O\left(\frac{1}{\varepsilon}\right)$层，直到得到 $O\left(2^{\frac{1}{\varepsilon}}\cdot A^{1-\varepsilon}\right)$个森林度均为$O(A^{\varepsilon})$ 的子图。然后，我们使用附录 E.6中的算法，在 $O(a^{\varepsilon}\cdot\log a\cdot\varepsilon+\log\Delta)\cdot\log n$ 时间内，使用 $O(|E|)$ 个处理器，对每个这样的子图进行 $O(A^{\varepsilon})$ 着色（见定理 23）。总之，我们得到一个 $O\left(2^{\frac{1}{\varepsilon}}\cdot A\right)=O\left(2^{\frac{1}{\varepsilon}}\cdot a\right)$顶点着色，其 PRAM 时间为 $O\left(\frac{1}{\varepsilon}\cdot a^{\bar{2\varepsilon}}\cdot\log a\cdot\log n+\log\Delta\cdot\log n\right)$，使用 $\bar{O}\left(|E|\cdot{}^{a^{\varepsilon}}\cdot\log n\log(a^{\varepsilon}\cdot\log n)\right)$ 个处理器。令 $\varepsilon=\frac{c\cdot\log\log a}{\log a}$，其中 $c>0$为常数参数，则可得到一个 $a\cdot 2^{O\left(\log a\log\log a\right)}=a^{1+o(1)}$顶点着色，其时间为 $O\left(\log a\log\log a\cdot\log^{2c+1}a\cdot\log n+\log\Delta\cdot\log n\right)$，

使用 $O\left(|E|\cdot\log^{c}a\cdot\log n\log(\log a\cdot\log n)\right)=O\left(|E|\cdot\log^{c}a\cdot\log n^{n}\right)\log\log$ 个处理器。因此，我们还可以在相同时间内，使用上述数量的处理器，得到一个大小为 $\Omega\left(\frac{n}{a^{1+o(1)}}\right)$ 的独立集。由于对于每个图 $a\leq\Delta$，由此可知，该算法还可在 $O\left(\log^{1+\delta}\Delta\cdot\log n\right)$ 时间内，使用 $O\left(|E|\cdot\Delta^{\delta}\cdot\log n\log(\Delta\cdot\log n)\right)$ 个处理器，得到一个 $\Delta^{1+o(1)}$顶点着色，其中 $\delta>0$是任意小的常数。因此，我们还可以在相同时间内，使用上述数量的处理器，得到一个大小为 $\Omega\left(\frac{n}{\Delta^{1+o(1)}}\right)$ 的独立集。

我们用以下定理总结上述讨论。

定理 *24*（对 [*BE11a*] 的改编）。给定一个具有 Δ 最大度和 a 森林度的 n 顶点、 m 边图 $G=(V,E)$，可以在 $O\left(\left(\log^{2+\delta}a+\log\Delta\right)\cdot\log n\right)$ 时间内计算出一个 $a^{1+o(1)}$ 顶点着色（其中 $\delta>0$ 为任意小的常数），使用 $O\left(m\cdot\log^{\delta}a\cdot\log n\log(a\cdot\log n)\right)$ 个处理器。由于 $a\leq\Delta$，这同时也是一个 $\Delta^{1+o(1)}$ 顶点着色，可在 $O\left(\log^{2+\delta}\Delta\cdot\log n\right)$ 时间内计算得出（其中 $\delta>0$ 为任意小的常数），使用 $O\left(m\cdot\log^{\delta}\Delta\cdot\log n\log(\Delta\cdot\log n)\right)$ 个处理器。

E.8 有界度图的森林缺陷着色

本节将 [BEG18]提出的分布式算法改编到 PRAM 环境中。该算法的输入是图 G，其最大度为 Δ，参数为 ρ， $1\leq\rho\leq\Delta$。算法用 $O(\rho)$-任意缺陷着色，将 $O\left(\frac{\Delta}{\rho}\right)$着色为 G，耗时 $O\left(\frac{\Delta}{\rho}\right)$ 个分布式轮次。随后，我们在附录 E.9 中运用该算法，将 Barenboim 的分布式算法 [Bar16]针对 $O(\Delta)$-顶点着色、运行时间为 $O\left(\sqrt{\Delta}\right)+\log^{*}n$ 的算法改编到 PRAM 环境中。

该算法首先在 $\log^{*}n+O(1)$ 个分布式轮次内计算一个 ρ-缺陷、 $O\left(\left(\frac{\Delta}{\rho}\right)^{2}\right)$-着色 φ。（见定理 14。）现在，每个顶点 v 将其颜色 $\varphi(v)$ 表示为一对 $a,b=a,b_{\varphi(v)}$, $a,b=O\left(\frac{\Delta}{\rho}\right)$。具体而言，令 $c>0$ 为一个常数，使得 $\varphi(v)$ 至多使用 $c^{2}\cdot\left(\frac{\Delta}{\rho}\right)^{2}$ 种颜色。接着，令 p 为一个素数，使得 $2c\cdot\frac{\Delta}{\rho}<p\leq4c\cdot\frac{\Delta}{\rho}+2$。（根据伯特兰-切比雪夫定理，这样的素数存在。）于是，$a,b_{\varphi(v)}$ 就是 $\varphi(v)$ 在基 p 下的表示。

算法设置 $\psi_{1}(v)=\varphi(v)$，并开始迭代 p 轮。在第 i 轮开始时（在这 p 轮中， $1\leq i\leq p$），v 所具有的颜色记为 $\psi_{i}(v)$。如果 $\varphi_{i}(v)=0,b_{\psi(v)}$，则称此颜色为最终颜色或已确定的颜色；否则称其尚未确定。如果 v 在第 i轮将颜色从尚未确定的颜色更改为最终颜色，则称其在第 i轮确定了颜色。对于一对邻居 u,v，如果在某一轮 i，$\psi_{i}(u)=a_{u},b_{u}$， $\psi_{i}(v)=a_{v},b_{v}$，且 $b_{u}=b_{v}$，则称它们的颜色在该轮相互冲突。

在每一轮 i，每个颜色尚未确定的顶点 v $\psi_{i}(v)=a,b$都会检查与 v冲突的邻居u 中，有多少个邻居的原始颜色与 v 的颜色不同（即， $\varphi(u)\neq\varphi(v)$）。如果此类冲突的数量至多为 ρ，则 v 确定其颜色，即设置 $\psi_{i+1}(v)=0,b$。否则，它设置 $\psi_{i+1}(v)=a,b+a$。（求和按模 p进行。）至此，（分布式）算法（由 [BEG18] 提出）的描述完毕。

接下来，我们简要说明其分析。固定一个顶点 v，并考虑 v 的一个邻居 u，满足 $\varphi(u)\neq\varphi(v)$。记 $\langle a_{v},\ b_{v}\rangle=\varphi(v)$，$\langle a_{u},\ b_{u}\rangle=\varphi(u)$。考虑从算法开始到 v 完成定案（或直到 p 轮次结束）这段时间。在这段时间内，直到 u 完成定案之前， u 与 v 最多发生一次冲突；之后最多再发生一次。（这是因为

方程 $b_v + i \cdot a_v \equiv b_u + i \cdot a_u$（模 p）有一个解，对于任意 $b' \in GF(p)$，方程$b_v + i \cdot a_v \equiv b'$（模 p）也同样有一个解。）因此，在 p 轮次内、完成定案之前，v 与其邻居最多发生 2Δ 次冲突。另一方面，在 v 尚未完成定案的每一轮中，它发生的冲突都多于 ρ 次。因此，它不可能超过 $\frac{2\Delta}{\rho}$ 轮仍未完成定案。由于 $p > \frac{2\Delta}{\rho}$，可知 v（以及其他每个顶点）最终都会完成定案。

现在我们论证所得的着色 $\psi = \psi_{p+1}$ 是 $O\left(\frac{\Delta}{\rho}\right)$-着色，且是 $O(\rho)$-任意缺陷度的。为此，我们按如下方式对每条边 (v,u) 进行定向：如果 v（分别为 u）在轮次 i_v（分别为 i_u）完成，并且 $i_v > i_u$，则将该边定向为指向 u（即作为$\langle v,u\rangle$）。如果 $i_v = i_u$，或者它们的原始颜色满足 $\varphi(v) = \varphi(u)$，则将该边定向为指向标识符较大的端点（即基本上任意定向）。

注意，ψ使用了 $p = O\left(\frac{\Delta}{\rho}\right)$种颜色（因为对于每个 $v \in V$ 以及 $b_v \in GF(p)$，都有 $\psi(v) = \langle 0, b_v\rangle$）。现在我们还要论证，在上述定向下，每个顶点 v 至多有 $O(\rho)$个出邻居 u，且其 $\psi(u) = \psi(v)$。

事实上，回想一下，φ 是一个 ρ-缺陷着色，因此 v 至多有 ρ 个出邻居u 满足 $\psi(v) = \psi(u)$且 $\varphi(v) = \varphi(u)$。此外，对于 v 的每个出邻居 u，若满足$\psi(v) = \psi(u)$且 $\varphi(v) \neq \varphi(u)$，则该邻居 u 在 v确定其颜色之前（或同时）就已确定自己的颜色。因此，当 v 确定其颜色时，v 与 u 发生了冲突（因为 $\psi(v) = \psi(u)$）。由于 v 确实在该轮确定了自己的颜色，因此此时它至多有 ρ 个冲突，所以这样的出邻居至多有 ρ 个 u。因此，着色 ψ 的任意缺陷度至多为 $2\rho = O(\rho)$。

最后，我们讨论该算法的 PRAM 实现。正如我们在附录 E.3（定理 16）中所见，计算缺陷着色 φ 需要 $O(\log\Delta \cdot \log^* n + \log\log n)$时间，并需要 $O\left(m \cdot \frac{\Delta}{\rho} \cdot \log n \log\left(\frac{\Delta}{\rho} \cdot \log n\right)\right)$ 个处理器。主循环的 $p = O\left(\frac{\Delta}{\rho}\right)$ 轮（将初始着色 $\psi_0 = \varphi$ 转换为最终着色 $\psi_p = \psi$）均可按如下方式实现：为每个顶点 v分配 $\deg(v)$ 个处理器。这些处理器用 $O(m)$个处理器，在 $= O(\log\Delta)$时间内统计 $O(\log\deg(v))$中的冲突数量。因此，总时间为$O(p \cdot \log\Delta) = O\left(\frac{\Delta}{\rho} \cdot \log\Delta\right)$，处理器数量为 $O\left(m \cdot p \cdot \log n \log(p \cdot \log n)\right)$。

定理 25 (对 [BEG18] 的改编)。对于任意 n 个顶点的 m 条边、最大度为Δ 的图，以及参数 $1 \le \rho \le \Delta$，可以在以下时间内计算一个 $O\left(\frac{\Delta}{\rho}\right)$-颜色着色 $O(\rho)$-任意缺陷度着色：

$$O\left(\frac{\Delta}{\rho} \cdot \log\Delta + \log\Delta \cdot \log^* n + \log\log n\right) = O\left(\log\Delta \cdot \left(\frac{\Delta}{\rho} + \log^* n\right) + \log\log n\right)$$

并行随机存取机 时间，使用 $O\left(m \cdot \frac{\Delta}{\rho} \cdot \log n \log\left(\frac{\Delta}{\rho} \cdot \log n\right)\right)$ 个处理器。

为了优化处理器数量，我们采用定理 17 中的权衡，参数 z取为 $z = \sqrt{p \cdot \log n \log(p \cdot \log n)}$，$p = \frac{\Delta}{\rho}$。随后，计算缺陷着色需要

$$O\left(\frac{p \cdot \log n}{z \cdot \log(p \cdot \log n)} + \log\Delta \cdot \log^* n + \log\log n\right) = O\left((\sqrt{p} + \log^* n) \cdot \log\Delta + \log\log n\right)$$

时间，但使用 $O(m \cdot z) = O\left(m \cdot \sqrt{p \cdot \log n \log(p \cdot \log n)}\right)$ 个处理器。加上主循环的 p 轮次

，总运行时间为

$$O\left(\left(\sqrt{p}+\log^* n\right)\cdot\log\Delta+\log\log n\right)+p\cdot\log\Delta\big)=$$
$$O\left(\left(p+\log^* n\right)\cdot\log\Delta+\log\log n\right)=$$
$$O\left(\frac{\Delta}{\rho}\cdot\log\Delta+\log\Delta\cdot\log^* n+\log\log n\right),$$

处理器数量为

$$O\left(m\cdot\frac{\sqrt{p}\cdot\log n}{\log(p\cdot\log n)}\right)=O\left(m\cdot\frac{\sqrt{\frac{\Delta}{\rho}}\cdot\log n}{\log\left(\frac{\Delta}{\rho}\cdot\log n\right)}\right).$$

这将处理器数量的上界从定理 23降低了，而其他参数保持不变。

推论 7. 对于一个具有 n 个顶点、 m 条边且最大度为 Δ 的图，以及参数 $1\leq\rho\leq\Delta$，可以在以下时间内计算一个 $O(\rho)$-任意缺陷度 $O\left(\frac{\Delta}{\rho}\right)$-着色：

$$O\left(\frac{\Delta}{\rho}\cdot\log\Delta+\log\Delta\cdot\log^* n+\log\log n\right)$$

时间，使用 $O\left(m\cdot\sqrt{\frac{\Delta}{\rho}\cdot\log n}\right)$ 个处理器。

一个特别重要的情形是 $\rho=\sqrt{\Delta}$。（注意，如果 $\log^* n>\sqrt{\Delta}$，那么 $\log\log n$ 项会主导运行时间。）由此可得：

推论 8. 对于任意具有 n 个顶点、 m 条边且最大度为 Δ 的图，可以在 $O\left(\sqrt{\Delta}\cdot\log\Delta+\log\log n\right)$ 并行随机存取机 时间内计算一个 $O\left(\sqrt{\Delta}\right)$-任意缺陷度$O\left(\sqrt{\Delta}\right)$-着色，使用 $O\left(m\cdot\Delta^{1/4}\cdot\log n\log(\Delta\cdot\log n)\right)$ 个处理器。

E.9 将 Barenboim 算法改编为 PRAM 模型

本节将 Barenboim 提出的分布式算法 [Bar16] 用于 $O(\Delta)$-顶点着色 $\tilde{O}\left(\sqrt{\Delta}\right)+O$，该算法在 $^*n$ 轮内完成，我们将其改编为 PRAM 模型。

我们首先描述该算法在分布式模型上的实现，并简要分析其性能。然后将其改编为 PRAM 模型。

该算法首先通过 [BEG18] 的算法计算一个 $O(p)$-着色，其在 $O\left(\frac{\Delta}{p}\right)+$ 中用 $\left(\frac{\Delta}{p}\right)$-任意缺陷度 $\hat{}\varphi$，耗时 $^*n\log^* n$，参见定理 25；参数为 p, $1\leq p\leq\Delta$。（我们将使用 $p=\sqrt{\Delta}$。）令 G_1, G_2, .., G_p 为由 $\hat{}\varphi$ 的颜色类所诱导的 G 的子图 p，也就是说，对于每个 $i\in\{1,2,...,p\}$，子图 G_i 由满足 $\hat{}\varphi(v)=i$ 的顶点 v 所诱导。注意，在这一步计算中，我们还会得到每个 G_i 的森林分解，将其分解为 $\left(\frac{\Delta}{p}\right)$ 个森林（并且每个 G_i 的森林度至多为 $\left(\frac{\Delta}{p}\right)$）。现在，我们并行地对每个子图运行 Arb-Linial 算法，并在额外的 $\log^* n+O(1)$ 轮次内，为每个 G_i 得到一个 $O\left(\left(\frac{\Delta}{p}\right)^2\right)$-顶点着色 φ_i

轮次。注意，当 $p = O\left(\sqrt{\Delta}\right)$, φ_i 是 G_i 的一个 $O(\Delta)$-着色时，对每个 $i \in \{1,2,...,p\}$。

Barenboim 算法现在能在 p 次迭代内，为整个图 G 计算出一个（合法的） $O(\Delta)$-顶点着色 ψ。（对于一般的 p，它使用的颜色数为 $\Lambda(\Delta,p) = O\left(\Delta + \frac{\Delta}{p} \cdot \left(\sqrt{\Delta} + \frac{\Delta}{p}\right)\right)$。）具体而言，首先为所有顶点 $v \in V(G_1)$ 初始化颜色 $\psi(v) = \varphi_1(v)$。接着，它先在一轮中重新着色 G_2 的顶点，再在额外一轮中重新着色 G_3 的顶点，依此类推，最后再用一轮重新着色 G_p 的顶点。下面描述此循环中的一次迭代 $t, 2 \le t \le p$。在这次迭代中，算法以一种与已重新着色的顶点（属于 $G_1 \cup ... \cup G_{t-1}$）的颜色相一致的方式，重新着色 G_t 的顶点。此外，对任意 $(u,v) \in E$, $u,v \in V(G_t)$，顶点 G_t 的新着色 ψ 满足 $\psi(u) \neq \psi(v)$。该着色 ψ 使用 $\Lambda = \Lambda(\Delta,p)$ 种颜色。

在本次迭代开始时，我们有一个 $G_1 \cup ... \cup G_{t-1}$ 的合法 Λ-着色 ψ，以及一个 G_t 的 $\left(\frac{\Delta}{p}\right)^2$-着色 $\varphi = \varphi_t$，其中 $c > 9$ 是普适常数。此外，回顾一下，我们给定了 G_t 的一个森林分解，将其分解为 $a = \frac{\Delta}{p}$ 个森林，并且每个顶点 $v \in V(G_t)$ 都知道其在该分解下的 a 个父节点。另设 $L(v)$ 为 v 的邻居 u 中属于 $G_1 \cup ... \cup G_{t-1}$ 的顶点所使用的 ψ-颜色集合。

我们选取一个参数 μ，使其为介于 $\sqrt{\Delta} + \sqrt{c \cdot \frac{\Delta}{p}}$ 和 $2\left(\sqrt{\Delta} + \sqrt{c \cdot \frac{\Delta}{p}}\right)$ 之间的素数。（根据伯特兰-切比雪夫定理，这样的素数存在。）着色 ψ 使用

$$\Lambda = \Delta + (2a+1) \cdot \mu \tag{6}$$

种颜色。每个顶点 $v \in V(G_t)$ 构造一个由 μ 个一元多项式组成的集合 $p_0^{(\varphi(v))}(x), ..., p_{\mu-1}^{(\varphi(v))}(x)$，这些多项式定义在 $GF(\mu)$ 上。对于每个 $i \in GF(\mu)$，多项式 $p_i^{(\varphi(v))}(x)$ 定义为 $p_i^{(\varphi(v))}(x) = i + a_v \cdot x + b_v \cdot x^2$（模 μ），其中 $\varphi(v) = a_v, b_{v^{\varphi(v)}}$。（也就是说，$a_v, b_{v^{\varphi(v)}}$ 是 $\varphi(v)$ 在基 $\lceil \sqrt{c \cdot a} \rceil$ 下的表示。）注意，对于一对邻居 u、$v \in V(G_t)$，有 $\varphi(u) \neq \varphi(v)$，因此 $a_v, b_{v^{\varphi(v)}} \neq a_u, b_{u^{\varphi(u)}}$。因此，它们各自的多项式集合 $\left\{p_0^{(\varphi(v))}(x), ..., p_{\mu-1}^{(\varphi(v))}(x)\right\}$ 和 $\left\{p_0^{(\varphi(u))}(x), ..., p_{\mu-1}^{(\varphi(u))}(x)\right\}$ 不相交。（对于任意一对这样的多项式，$j + a_v \cdot x + b_v \cdot x^2$ 和 $j' + a_u \cdot x + b_u \cdot x^2$ 中，$a_v \neq a_u$ 或 $b_v \neq b_u$ 至少有一个成立。）我们称多项式 p 生成颜色 $\gamma \in \{0, 1, ..., \mu^2 - 1\}$，如果 γ 在基 μ 下的表示 y, x 满足 $y = p(x)$。

对于每个顶点 $v \in V(G_t)$ 和索引 $i \in \{0,1,...,\mu-1\}$，令

$$\mathcal{P}_i(v) = \left\{\left\langle p_i^{(\varphi(v))}(k), k\right\rangle \mid k = 0, 1, ..., \mu - 1\right\} \tag{7}$$

为多项式 $p_i^{(\varphi(v))}(x)$ 生成的颜色集合。此外，令 $L_i(v) = L(v) \cap \mathcal{P}_i(v)$ 为由 $p_i^{(\varphi(v))}(x)$ 生成的、被 v 的重新着色的邻居使用的颜色集合，并令 $\hat{L}_i(v) = \mathcal{P}_i(v) \setminus L_i(v)$ 为多项式 $p_i^{(\varphi(v))}(x)$ 生成的、未被 v 的重新着色的邻居使用的颜色集合。

顶点 v 选择一个索引 $i, i \in \{0, 1, ..., \mu - 1\}$，使集合 $L_i(v)$ 的大小最小，并将其发送给所有邻居。注意，$|L(v)| \le \Delta$。还要注意，对于每个 $i \neq j, i, j \in \{0, 1, ..., \mu - 1\}$，都有 $L_i(v) \cap L_j(v) = \emptyset$；否则，对于某个 $k \in \{0, 1, ..., \mu - 1\}$，就有 $p_i^{(\varphi(v))}(k) = p_j^{(\varphi(v))}(k)$。但是

$$i + a_v \cdot k + b_v \cdot k^2 \equiv j + a_v \cdot k + b_v \cdot k^2 \pmod{\mu},$$

这就矛盾了。因此，使 $L_i(v)$ 大小最小的索引 $i \in \{0, 1, \ldots\ldots, \mu-1\}$ 对应的集合大小至多为 $\frac{\Delta}{\mu}$。因此，相应集合 $\hat{L}_i(v) = \mathcal{P}_i(v) \setminus L_i(v)$ 的大小至少为

$$\mu - \frac{\Delta}{\mu} \geq \mu - \sqrt{\Delta} \geq \sqrt{\Delta} + \sqrt{c} \cdot \frac{\Delta}{p} - \sqrt{\Delta} = \sqrt{c} \cdot \frac{\Delta}{p} > 2 \cdot \frac{\Delta}{p}.$$

（这里我们使用了 $\mu \geq \sqrt{\Delta} + \sqrt{c} \cdot \frac{\Delta}{p} \geq \sqrt{\Delta}$ 和 $c > 9$。）

这意味着，多项式 $p_i^{(\varphi(v))}(x)$（针对选定的索引 i）生成的颜色中，有超过 $2 \cdot \frac{\Delta}{p}$ 个未被 v 已重新着色的邻居（来自 $\bigcup_{j=1}^{t-1} G_j$ 的邻居）使用。此外，对于 G_t 中 v 的每个父节点 u（此类父节点最多有 $\frac{\Delta}{p}$ 个），v 选定的多项式 $p_i^{(\varphi(v))}(x)$ 与 u 选定的多项式 $p_{j_u}^{(\varphi(u))}(x)$ 不同，因此这两个多项式至多在两个点相交（因为它们是二次多项式）。换句话说，在 $\hat{L}_i(v)$ 中超过 $2 \cdot \frac{\Delta}{p}$ 个颜色（由 $p_i^{(\varphi(v))}(x)$ 生成且未被 v 的已重新着色邻居使用）中，至少有一个颜色无法由 G_t 中 v 的父节点 u 所选的多项式 $p_{j_u}^{(\varphi(u))}(x)$ 生成。顶点 v 现在用这样的颜色 $\left\langle p_i^{(\varphi(v))}(k),\ k \right\rangle \in \hat{L}_i(v)$ 为自身着色，即这些颜色中最小的 k。

上述讨论表明，所得着色 ψ 是适当的，并且可在总计 $O(p)$ 轮次内完成。为分析所用颜色的数量，注意到如上所述，最小的 k 至多为 $2 \cdot \frac{\Delta}{p} + \frac{\Delta}{\mu}$。（回忆一下，$|L_i(v)| \leq \frac{\Delta}{\mu}$。）因此，该颜色在基 μ 上的表示 $\left\langle p_i^{(\varphi(v))}(k),\ k \right\rangle$ 会产生一个至多为

$$\mu - 1 + \mu \cdot \left(\frac{2\Delta}{p} + \frac{\Delta}{\mu} \right) \leq \Delta + \mu \cdot \left(\frac{2\Delta}{p} + 1 \right) = \Lambda(\Delta, p).$$

（见（6）。）因此，该着色 ψ 使用 $\Lambda(\Delta, p)$ 种颜色。由于 $\mu = O\left(\sqrt{\Delta} + \frac{\Delta}{p}\right)$，我们有 $\Lambda = \Delta + O\left(\left(\frac{\Delta}{p}\right)^2\right) + \left(\frac{\Delta^{\frac{3}{2}}}{p}\right)$。令 $p = \left(\sqrt{\Delta}\right)$，即可得到一个在 $O(\Delta)$-顶点着色，并且只需 $O\left(\sqrt{\Delta} + \log^* n\right)$ 轮次。

为了在并行随机存取机环境下实现该算法，我们首先在 $O\left(\sqrt{\Delta} \cdot \log \Delta + \log \log n\right)$ 时间内计算一个 $O\left(\sqrt{\Delta}\right)$-着色 φ，$O\left(\sqrt{\Delta}\right)$-任意缺陷，使用 $O\left(m \cdot \Delta^{1/4} \cdot \log n \log(\Delta \cdot \log n)\right)$ 个处理器（见推论 8）。然后，我们并行地为每个颜色类 G_i 计算一个 $O(\Delta)$-着色 φ_i。根据推论 6，对于参数 $z = O\left(\sqrt{\Delta} \cdot \log n \log(\Delta \cdot \log n)\right)$，此步骤需要

$$O\left(\log \Delta \cdot \log^* n + \log \log n + \frac{\sqrt{\Delta} \cdot \log n}{z \cdot \log(\Delta \cdot \log n)}\right)$$

时间，使用 $O(|E| \cdot z) \cdot O\left(\sqrt{\Delta}\right)$ 个处理器（每个 $O\left(\sqrt{\Delta}\right)$ 个子图使用 $O(|E| \cdot z)$ 个处理器）。我们令 $z = \log \Delta$，即可得到总运行时间

$$O\left(\log \Delta \cdot \log^* n + \log \log n + \frac{\sqrt{\Delta} \cdot \log n}{\log \Delta \cdot \log(\Delta \cdot \log n)}\right) = O\left(\frac{\sqrt{\Delta} \cdot \log n}{\log \Delta \cdot \log(\Delta \cdot \log n)}\right),$$

以及 $O\left(|E| \cdot \left(\sqrt{\Delta} \cdot \log \Delta + \Delta^{1/4} \cdot \log n \log(\Delta \cdot \log n)\right)\right)$ 个处理器。

那么我们有 $O(p) = O\left(\sqrt{\Delta}\right)$ 次重新着色迭代。为了实现这些迭代，对于固定的顶点 v，我们为每条边 (v,u) 指定 μ 个处理器。这些处理器的索引为 $0, 1, ..., \mu-1$。索引为 i 的处理器（对每个 i）在 $O(1)$ 时间内计算集合 $L_i(v) = P_i(v) \cap L(v)$。（集合 $P_i(v)$（见方程 (7)）由这些处理器在 $O(1)$ 时间内计算。对于 v 的每个已重新着色的邻居u，与边 (v,u) 关联的第 i 个处理器会从 $P_i(v)$ 中移除 u 的颜色 $\psi(u)$。）随后，这些处理器会在相同时间内计算出 $\hat{} L_i(v)$。最后，对于其子图 G_t 中的每个父节点 u（森林度至多为 $\frac{\Delta}{p}$），与边 (v,u) 关联的 μ 个处理器会在 $O(1)$ 时间内求解方程 $p_i^{(\varphi(v))}(x) = p_{j_u}^{(\varphi(u))}(x)$（其中，$i$（分别为 j_u）是由 v（分别由 u）选取的索引），并从 $\hat{} L_i(v)$ 中移除其至多两个解。（为求解这些方程，处理器 $q \in \{0, 1, ..., \mu-1\}$ 会计算 $p_i^{(\varphi(v))}(q)$ 和 $p_{j_u}^{(\varphi(u))}(q)$，并检查它们是否相等。）现在，在仍属于集合 $\hat{} L_i(v)$ 的颜色 $\left\langle p_i^{(\varphi(v))}(k)\ k \right\rangle$ 中找出最小的 $k \in \{0, 1, ..., \mu-1\}$，需要 $O(\log \Delta)$ 时间。因此，整个重新着色过程需要 $O\left(\log\Delta \cdot \sqrt{\Delta}\right)$时间，并使用 $O(\mu \cdot m) = O\left(\sqrt{\Delta} \cdot m\right)$ 个处理器。

总结如下：

定理 26 (对 [Bar16]的改编。 一个 $O(\Delta)$-顶点着色可在

$$O\left(\sqrt{\Delta} \cdot \left(\log\Delta + \frac{\log n}{\log\Delta \cdot \log(\Delta \cdot \log n)}\right)\right)$$

并行随机存取机 时间内，使用 $O\left(m \cdot \left(\sqrt{\Delta} \cdot \log\Delta + \frac{\Delta^{1/4} \cdot \log n \log(\Delta \cdot \log n)}{}\right)\right)$ 个处理器计算得出。

E.10 大独立集

我们先陈述 Goldberg 和 Spencer 提出的一个结果 [GS89]，该结果可在简单图中求出一个极大独立集。

引理 28 (极大独立集算法 [GS89])。 设 $G = (V, E)$ 是一个 n 个顶点、 m 条边的图。存在一种确定性独占读独占写并行随机存取机 算法，可在G 的 $O\left(\log^3 n\right)$ 时间内，使用 $O\left(n+m \log n\right)$ 个处理器找到一个极大独立集。

对我们的目的而言，一个“大型”独立集就足够了（参见定义12）。任何顶点着色算法都可用于计算这样的独立集。具体而言，由于正常顶点着色中的每个颜色类都是一个独立集，我们可以计算 n顶点图中最大的颜色类并将其返回。用 λ表示对一个具有 n个顶点、m条边、最大度为 Δ 且森林度为 a 的图进行顶点着色所需的时间，记为 $VCT_\lambda(n, m, \Delta, a)$，所需的处理器数量记为 $VCP_\lambda(n, m, \Delta, a)$。下一条引理将分析如何使用顶点着色算法计算大型独立集。

引理 29。 设 $G = (V, E)$ 是一个有 n 个顶点、 m 条边、最大度为 Δ 且森林度为a的图。可以在$O(VCT_\lambda(n, m, \Delta, a) + \log n)$ 时间内，使用 $O(VCP_\lambda(n, m, \Delta, a) + m)$ 个处理器，计算出其大小为 $|I| \geq \frac{n}{\lambda}$ 的独立集 I，即一个 λ 大型独立集。

证明。 如上所述，算法首先计算 G 的一个 λ-顶点着色 φ，然后返回最大的颜色类。最大的颜色类可以通过按颜色对顶点进行排序，并计算排序顺序中每种颜色对应的索引范围来确定。

因此，整个过程使用 $O(VCP_{\lambda}(n,m,\Delta,a)+m)$ 个处理器，需要 $O(VCT_{\lambda}(n,m,\Delta,a)+\log\ n)$ 时间。□

利用本节介绍的不同顶点着色算法，我们在下一个定理中总结了计算大型独立集的复杂度界。

定理 2 (计算独立集)。设 $G=(V,E)$ 是一个有 n 个顶点、m 条边的图，其最大度为 Δ，森林度为 a。

(1)（*[GS89]*）一个 $(\Delta+1)$-大型独立集可在 $O\left(\log^3 n\right)$ 时间内计算得出，使用 $O\left(\ ^{m}\log\ _{n}\right)$ 个处理器。*(2)*（使用定理 *21* 中的顶点着色算法）一个 $(\Delta+1)$-大型独立集可在 $O\left(\log n+\Delta\cdot\log^2\Delta\right)$ 时间内计算得出，使用 $O(m)$ 个处理器。*(3)*（使用定理 *23* 中的顶点着色算法）一个 $O(a)$-大型独立集可在 $O((a\cdot\log\ a+\log\ \Delta)\cdot\log\ n)$ 时间内计算得出，使用 $O(m)$ 个处理器。*(4)*（使用定理 *24* 中的顶点着色算法）对于任意常数 $\delta>0$，一个 $\left(a^{1+o(1)}\right)$-大型独立集可在 $O\left(\log^{2+\delta} a\cdot\log\ n+\log\ \Delta\cdot\log\ n\right)$ 时间内计算得出，使用 $O\left(m\cdot\log^{\delta}{}_{a}\cdot\log\ _{n}\log(_{a}\cdot\log\ _{n})\right)$ 个处理器。*(5)*（使用定理 *26* 中的顶点着色算法）一个 $O(\Delta)$-大型独立集可在 $O\left(\sqrt{\Delta}\cdot\left(\log\ \Delta+\log\ _{n}\log\ _{\Delta}\cdot\log(_{\Delta}\cdot\log\ _{n})\right)+\log\ n\right)$ 时间内计算得出，使用 $O\left(m\cdot\left(\sqrt{\Delta}\cdot\log\ \Delta+\ ^{\Delta^{1/4}}\cdot\log\ _{n}\log(_{\Delta}\cdot\log\ _{n})\right)\right)$ 个处理器。