算法作业 2

描述最小生成树的并行化算法

Red/Blue Rule

算法：

此算法将一个图中的边分为两类，一类标记为红色，是不属于 MST 的边，一类标记为蓝色，是属于 MST 的边。

标记规则：

Blue：选择一个没有蓝色边的割，将割中权值最小的无色边标记为蓝色。

Red：选择一个没有红色边的环，将环中权值最大的无色边标记为红色。

以任意顺序执行 Red/Blue 规则，当不可以再执行任意一条规则时，取所有蓝色的边就是 MST。

证明引理：

在算法的任意时刻，存在一个 MST 它包含了所有的蓝色边，并且不包含任何红色边。

假设在t^*时刻之前，该引理成立,，并且 MST T^*包含所有蓝色边，不包含红色边，那么

1. t^*+1时刻执行的是 Blue Rule。假设蓝色规则选择的割集是$(\mathrm{X}\ ,\ \mathrm{V}-\mathrm{X})$，那么割集中一定有至少一条边要属于$T^*$，设为$e'$，而割集中权值最小的边设为e，如果$\mathrm{e}\in T^*$，则成立，如果$\mathrm{e}\notin T^*$，则$T^*+\mathrm{e}$将存在一个环，$T^*+\mathrm{e}-e'$则也会是一个

MST（$\mathrm{cost(e)}\le \mathrm{cost}(e')$），成立。

2. t^*+1时刻执行的是 Red Rule。假设红色规则选择的环是C_n，那么环中至少有一条边不属于T^*，设为e'，而环中权值最大的边设为e，如果$\mathrm{e}\notin T^*$，则成立，如果$\mathrm{e}\in T^*$，则T^*+e'将存在一个环，$T^*-\mathrm{e}+e'$则也会是一个MST（$\mathrm{cost(e)}\ge \mathrm{cost}(e')$），成立。

引理成立。

证明算法可以生成 MST：

当算法结束时，若仍有边没有被染色，若这条边和蓝色的边可以构成环，则可以使用红色规则将其标记为红色，若这条边无法与蓝色的边构成环，则可以选择一没有蓝色边的割使用红色规则。所以算法结束时，全部的边都会被染色。

算法正确。

并行性分析：

Red Rule 和 Blue Rule 都可以并行得执行，条件是每次选择的点集不要冲突，Boruvka 算法就是使用 Blue Rule 的并行算法的一个例子。

Boruvka's Algorithm

在 Kruskal's 算法中我们使用 Blue Rule 将边按序一条条加入 MST 中，我们是否可以并行得使用 Blue Rule 从而缩短算法复杂度？

将每个单独的点视作割集的一方，则按照 Blue Rule，割集中最短的边一定在 MST 中，也即：与每个点相邻的最短的边一定在 MST 中。

按照这个策略我们最开始至少可以得到 n/2 条 MST 中的边，在这些边集上进行边的收缩形成 Super Node，然后继续执行 Blue Rule 再次获得 MST 中的一些边…按这个策略循环进行直到我们得到了 MST。

```
伪代码：While Super Node Size != 1
            For each Node, parallel do
                Compute smallest weight incident edge
            Contract edges
            Merge Graph
```

分析：

1. Compute smallest weight incident edge

   对于每个点，我们需要遍历它所有相连的边，所以需要的总时间应该是 $\mathrm{O}(\deg(v_i))$。而在并行比较中，我们可以采用每两个为一组同时比较大小的方法提高并发度，所以 Span 应该为$\mathrm{O}(\log_2 \deg(v_i))$。

   算法可以在每个点上并发进行，总的 Span 应该是$\mathrm{O}(\max_{i=0\to n} \log(\deg(v_i)))$，最大的度也就是点数 n，所以 Span = $\mathrm{O}(\log n)$，而总的 Work 就是所有的度求和，根据握手定理求得为$\mathrm{O(m)}$。

2. Contract edges

   选择好要收缩的边之后，我们要决定如何收缩，我们对每个点抛一枚硬币（in parallel）如果是 head，则把这个点标记为中心，如果是 tail，我们把这个点标记为边缘。所有的边缘要向它的邻居寻找中心（只在上一步选择的边中找），找到了就将它和中心之间的边删除，没有找到则自己就是中心。用这种随机化的方法可以加大算法的并发度。

   每次收缩点的个数的期望大于等于 n/4（易证）。所以这个操作总的 Work 应该为$\mathrm{O}\left(\sum_{i=0}^{\infty}\left(\frac{3}{4}\right)^i n\right) = O(n)$，Span 应该是操作的执行次数，每次收缩$\frac{n}{4}$，则在$\mathrm{O}(\log n)$次内可以执行完成。这也说明第一步操作我们也需要执行$\mathrm{O}(\log n)$次。

3. Merge Graph

   这一步更改图的存储表示，对于邻接链表保存的图来说，将两个点 merge 在一起只需要更改指针即可，所以更改一个点的 Work 是$O(1)$，总的 work 就是$O(n)$，span 是$O(1)$。

$$
\begin{aligned}
W_{total} &= W_{find} + W_{Contract} + W_{merge} \\
&= \mathrm{O}(\log n) * \mathrm{O(m)} + O(n) + \mathrm{O}(\log n) * \mathrm{O(n)} O(m \log n) \\
&= \mathrm{O(m}\log n)
\end{aligned}
$$

$$
S_{total} = S_{find} + S_{Contract} + S_{merge} = \mathrm{O}(\log n) * \mathrm{O}(\log n) + 2 * \mathrm{O}(\log n) = O(\log n^2)
$$

分布式系统中的 MST 算法：GHS 算法

MST 的 Kurskal Prim 算法都只能每次工作在一个节点上，而且需要每个节点直到关于图的全部信息，在分布式系统的 message-passing 模型中，每个节点只直到他的相邻节点的信息，所以我们需要新的算法，这个算法由 Gallager, Humblet ,Spira 在 1983 年提出，称为 GHS 算法，与 Borůvka's algorithm 有很大相似之处。

先决条件：

1. 算法工作在连通无向图上
2. 图中每条边权值不同（可以移除，不过有这个条件算法更容易描述）
3. 每个节点最初只了解邻居节点的权值
4. 消息可以在任意边上传递且不会出现错误，在一个不可知延迟过后总可以到达目标节点
5. 处理消息遵循 FIFO 原则。

定义最小生成树的片段是最小生成树的子树，那么将满足性质：

给一个片段 T，e 是片段中所有点中向外延伸权值最小的（称为 MWOE），则 T+e 仍然是一个片段。使用 Blue Rule 可以容易证明

算法流程：

初始时，将每个单独的节点当作一个片段，并且将它的等级（level）设置为 0，每条边都视为 Basic 状态。算法在执行过程中会通过合并两个片段改变片段 level，并且每个等级大于 0 的片段都具有与核心边相关的独立 ID。在算法执行过程中，图中每条边都可以被划分到三类中的一类：

1. Branch 已经确认是 MST 中的边
2. Rejected 已经确认不是 MST 中的边
3. Basic 不属于上述两种情况的边

初始化结束，唤醒任意多个节点，每个已唤醒的节点需要做：

1. 选择 MWOE 标记为 Branch 边
2. 沿着这条边发送消息给另一边的节点，如果对方没有被唤醒，唤醒它
3. 等待对方返回消息

如果有两个点选择了相同的边，那么这两个点就合并作为 level 1 的片段，给予这个片段的 ID 为这条边的权值，这条边就作为核心边。

对于每个 level 大于 0 的片段，需要循环做三件事情：

1. 广播：核心边两端的节点沿着 Branch 边（除核心边之外）向片段中的其他点通知自己的 ID 和 level。
2. 聚拢：与广播的传播方式相反，从叶子节点开始（叶子是在这个片段中只有一条 Branch 边连接的点），搜寻它的 MWOE，然后沿着 Branch 边发送消息（MWOE 的权值），对于每个非叶子节点，收到它连接的其他全部节点的信息后，得到自己的 MWOE，继续传播，最后 MWOE 将会传播到核心边所关联到的两个节点，这两个节点将可以得知这个片段的 MWOE，和达到这条边的路径。
3. 改变 ID：核心边得到片段 MWOE 以及到达 MWOE 的路径后，将沿着此路径发送消息给 MWOE 所对应的另一个片段的节点，如果另一方 level 和自己相等，那么这两个片段合并，level 加一，核心边变成这条边，然后返回执行广播；如果对方 level 高于自己，则对方将自己吸收，可以证明不论对方此时执行哪个步骤，这个吸收都可以无条件执行。

直到只剩下一个片段，算法找到了 MST。

(a) Sort processes by finish time

```
Init    last = 0, count = 0;
For each pi in processes:
        if ( Si >    last)
          last = Fi;
          count++;
return count;
```

(b) 成立

若要证明 K*就等于最小的 check(BestS)数，从两点证明：

K* <= BestS

反证法：如果 BestS < K*那么在 K*选定的集合中， 一定有至少两个区间被同一个 check 所覆盖（鸽巢原理），这与 K*中所有的区间都不覆盖矛盾，所以结论成立。

K* >= BestS（下文中端点表示 起始点/结束点）

构造法：所有区间都会与 K*中至少一个区间相交。若 K*中某个区间 k 包含了某个长度更小的区间 r，则用 r 替换 k。这样所有区间都会覆盖 K*中的某个端点。

遍历所有只与 K*中某一个端点相交的区间，将被交端点标记 flag。不会有某一个区间的两个端点都被标记为 flag，因为如果是，则可以将此区间替换使将其标记为 flag 的区间，然后 K*的数目将会加一。通过这一步，所有只与 K*中某一个端点相交的区间都可以在 K*的区间上找到对应的 flag。

遍历所有只与 K*中某两个端点相交的区间，如果这两个端点有至少一个标记为 flag，则跳过，否则，有五种情况：

通过这一步，所有只与 K*中某两个端点相交的区间都可以在 K*的区间上找到对应的 flag。

此时每个 K*中的区间最多有一个 flag。将所有 K*中的没有 flag 的区间随机取一个端点标记为 flag，此时 flag 的总数就为 K*的个数。

遍历所有与 K*中某三个或三个以上的端点相交的区间，这些区间都可以在所交的端点中找到有 flag 的端点（因为有三个相交端点表示至少跨过了一个区间，而每个区间都至少有一个 flag）。通过这一步，所有与 K*中某三个或三个以上的端点相交的区间都可以在 K*的区间上找到对应的 flag。

至此，所有的区间都可以在 K*上找到对应的 flag，在 flag 处设置 check，就可以检查所有的区间了，所以 K*是一个可行解，K* >= BestS。

因为 K* >= BestS, K* <= BestS，所以 K* == BestS

3. 证明树高小于等于 lgn

归纳法：

通过此方法构造出的高为 2 的树满足结论（至少两个节点，一定满足）

通过此方法构造出的高小于等于 k 的树满足结论时，

当两颗等于 k 的树与他合并时，新树的高为 k+1。若$\log x < k, \log y < k$，$则\log(x+y) < 2 * \mathrm{k}$。此种方法构造的高为 k+1 的树满足结论。

当高小于 k+1 的树与满足结论的高为 k+1 的树合并时，结果仍然满足。

构造高为 k+1 的树的方法只有以上两种方法，所以高为 k+1 的树满足结论。

归纳得出结论正确。

4. MST 的随机化算法

1994 年．David Karger, Philip Klein, 和 Robert Tarjan.共同提出了最小生成树的一个随机化算法，结和了分治，贪心，随机三种算法设计思想，这个算法可以在线性时间期望内找到最小生成树。

名词：

F-heavy ：F 是图 G 上的一个森林，一条 F-heavy 的边就是将这条边加入森林 F 会形成一个环，并且这条边在环中是权值最大的。

F-light ：F 是图 G 上的一个森林，一条 F-light 的边就是非 F-heavy 的边

定理：

1. F-heavy 一定不在最小生成树中。我们使用 Red Rule 可以证明。
2. 随机抽样定理
   H 是 G 根据每条边按概率 p 所选取的子图，F 是 H 上的最小生成森林，则 F-light 边数最多是$\frac{n}{p}$。

算法描述：

输入图G

如果G是空的，返回空森林

执行两次 Borůvka 收缩，将G转化为G'，收缩的边集为F_0

在G'中按概率 p 选择一些边形成子图 H，在 H 上递归调用算法得到最小生成树森林F_1

移除G'中对F_1来说所有的 F-heavy

在G'上递归调用算法得到最小生成树森林F_2

返回$F_0 \cup F_2$

复杂度分析：

除去递归，每次算法执行耗时 O(m)，m 为边树。

算法的递归可以看作生成一棵二叉树，左子树是递归执行随机取样得到的边，右子树是执行所有 F-light 的边，算法总的执行时间就是二叉树所有点的边数求和。

递归树中处于 d 深度的节点最多有$\frac{n}{4^d}$个 vertices。

我们可以把一个树的节点分为两类节点：

1. 第一个右节点/根节点
2. 左节点

所有的第二类节点都可以与第一类节点构成一条路径。

考虑任意一条从 k 条边开始的左路径，因为每向下走一步，在期望条件下路径数减少 1/2，则这条路径上全部的边数求和为 2k(根据生成多项式)。

所以所有节点的边数和就是 2*（所有第一类节点的边数和）

根节点为 m，而其他右节点的边数为父亲节点的 F-light 边，已经证明这个边数为 2*父亲节点数，所以全部右节点的边数为$\sum_{d=1}^{\infty}\frac{2^{d-1}n}{4^d}=\frac{n}{2}$。每层$2^{d-1}$个树节点，每个树节点有$\frac{n}{4^d}$个图节点。

所以最后复杂度为 O(m + n) = O(m)