Trees with an On-Line Degree Ramsey Number of Four

David Rolnick

Massachusetts Institute of Technology
Cambridge, MA, USA

drolnick@mit.edu

Submitted: Dec 17, 2010; Accepted: Aug 11, 2011; Published: Sep 2, 2011
Mathematics Subject Classification: 05C55, 05C57, 05C05

Abstract

On-line Ramsey theory studies a graph-building game between two players. The player called Builder builds edges one at a time, and the player called Painter paints each new edge red or blue after it is built. The graph constructed is called the *background graph*. Builder's goal is to cause the background graph to contain a monochromatic copy of a given *goal graph*, and Painter's goal is to prevent this. In the S_k*-game* variant of the typical game, the background graph is constrained to have maximum degree no greater than k. The on-line degree Ramsey number $\mathring{R}_\Delta(G)$ of a graph G is the minimum k such that Builder wins an S_k-game in which G is the goal graph. Butterfield et al. previously determined all graphs G satisfying $\mathring{R}_\Delta(G) \leq 3$. We provide a complete classification of trees T satisfying $\mathring{R}_\Delta(T) = 4$.

1 Introduction

The quintessential problem of Ramsey theory involves finding a monochromatic copy of a graph G within a larger graph whose edges are colored with some s colors. Given G and s, we say that a graph H *arrows* G if every s-coloring of H contains a monochromatic G as a subgraph. Two basic parameters of Ramsey theory are

- The *Ramsey number* $R(G)$ is the minimum number of vertices among graphs H that arrow G.
- The *size Ramsey number* $\hat{R}(G)$ is the minimum number of edges among graphs that arrow G.

The numbers called *on-line* Ramsey numbers are based upon the following game, which we consider in the 2-color case, though an s-color generalization is possible: Two

players, called Builder and Painter, generate a 2-colored graph H. Builder builds edges one at a time, using some combination of existing vertices and new vertices. As each edge is built, Painter paints it either red or blue. Builder's goal is for the graph H to contain a monochromatic copy of some given graph G, and Painter's goal is to prevent this from happening. We will call the graph G the *goal graph* and the 2-colored graph H that is being built the *background graph*. Note that the background graph gets bigger throughout the game.

If Builder is allowed to build edges without constraint, eventually he can generate a "large enough" background graph that it contains a monochromatic copy of G no matter how Painter has colored the edges. (This intuitively apparent result follows immediately from Ramsey's Theorem.) One may, however, consider the minimum number of edges Builder must draw before he wins. This number is the *on-line size Ramsey number* and is denoted $\tilde{r}(G)$. An important conjecture in this area is that

$$\lim_{n\to\infty} \frac{\tilde{r}(K_n)}{\hat{R}(K_n)} = 0$$

where $\hat{R}(G)$ is the (standard) size Ramsey number. Recently, Conlon [2] has made significant progress towards proving this conjecture, by showing that the given limit is indeed 0 if n is restricted to a certain subsequence of integers.

As an alternative to considering the size Ramsey number, the game can be modified so that Builder is allowed to build only edges such that the background graph remains within a specified class of graphs $\mathcal{H}$. We call such a game an *$\mathcal{H}$-game*.

We will consider the case where $\mathcal{H}$ is the class S_k of graphs H such that $\Delta(H) \leq k$, where $\Delta(H)$ denotes the maximum degree of the graph H. The *on-line degree Ramsey number* $\mathring{R}_\Delta(G)$ is defined to be the minimum k such that Builder can win an S_k-game where G is the goal graph.

Butterfield et al. [1] have studied $\mathring{R}_\Delta(G)$. Among their results are the following:

- $\mathring{R}_\Delta(K_{1,n}) = n$ for stars $K_{1,n}$.
- The graphs G for which $\mathring{R}_\Delta(G) \leq 3$ are exactly those graphs for which either (i) every component is a path, or (ii) every component is a subgraph of the *claw* $K_{1,3}$.
- For all G,
$$\mathring{R}_\Delta(G) \geq \Delta(G) - 1 + \max_{uv\in E(G)} \min\{d(u), d(v)\}$$
- If G is a tree with $d_1 \geq d_2$ being its greatest two degrees, then
$$\mathring{R}_\Delta(G) \leq d_1 + d_2 - 1$$
with equality if G has adjacent vertices of degrees d_1 and d_2.

Few graphs with on-line degree Ramsey number greater than 3 have hitherto been identified. In this paper, we present the following classification of trees T with $\mathring{R}_\Delta(T) = 4$:

Let M denote the set of trees with maximum degree at most 3 and with no two adjacent degree-3 vertices. Then, the set of trees T for which $\mathring{R}_\Delta(T) = 4$ is the set M, with the addition of the graph $K_{1,4}$ and with the removal of the claw $K_{1,3}$ and of all paths P_n.

2 Definitions

Given a subgraph H_0 of a given graph H, we will say that H_0 is *isolated* if it is its own component.

A Painter is called *consistent* if, whenever he is given a new edge e to be added to a background graph H, the color he assigns to e depends only on that component of $H \cup \{e\}$ containing e. Thus, for instance, a consistent Painter, when presented by Builder with an isolated edge, will always color it the same way, regardless of the other components of the background graph.

It is proven in [1] that, for any graph G and integer k, Builder can win a S_k-game if and only if Builder can win against a consistent Painter. Hence, for the remainder of this paper, we will assume that all Painters are consistent.

Following [1], we define a *weight function* on a graph G to be a function f assigning a positive integer to each vertex. A *weighted graph* is a graph G with an associated weight function. Given a weighted graph G we say that a (non-weighted) graph H *contains* G if H contains as a subgraph a non-weighted copy G' of G with the weight of each vertex of G being no less than the degree in H of the corresponding vertex of G'. In general, when we speak of "graphs" in this paper, we will mean weighted graphs.

Let a *maple* be a tree with maximum degree at most 3 and with no two adjacent degree-3 vertices. Let a *fork* within a maple be a degree-3 vertex. We say that a vertex v *abuts* a given fork y in the same maple if the path joining v and y does not include any forks (except y and possibly v).

3 Results

Our goal in this section will be to prove the folowing theorem:

Theorem 3.1. *The set of trees T having $\mathring{R}_\Delta(T) \leq 4$ is the set of maples, with the addition of the graph $K_{1,4}$.*

It is proven in [1] that, for any graph G,

$$\mathring{R}_\Delta(G) \geq \Delta(G) - 1 + \max_{uv \in E(G)} \min\{d(u), d(v)\}$$

An immediate corollary of this result is that if T is a tree with $\mathring{R}_\Delta(T) \leq 4$, then T must be either a maple or the graph $K_{1,4}$. Since $\mathring{R}_\Delta(K_{1,n}) = n$ is proven in [1], to prove Theorem 3.1 it suffices to prove that Builder can win an S_4-game with goal graph some fixed maple. Consider then an S_4-game with goal graph some maple T. Suppose towards contradiction that Painter never forms a monochromatic T. Without loss of generality, assume that our consistent Painter colors any isolated edge red. Our strategy in this proof is to show that Builder can construct a red copy of T.

Suppose that Builder can force some (2-colored) graph H, with v a vertex of H having weight 1. Then, we claim that the following four results hold:

Tree Extension Lemma. Builder can force the (weighted, 2-colored) graph shown in Figure 1(a).

2-2 Branching Lemma. Builder can force the graph shown in Figure 1(b).

3-3 Branching Lemma. Builder can force the graph shown in Figure 1(c).

2-3 Branching Lemma. Builder can force the graph shown in Figure 1(d).

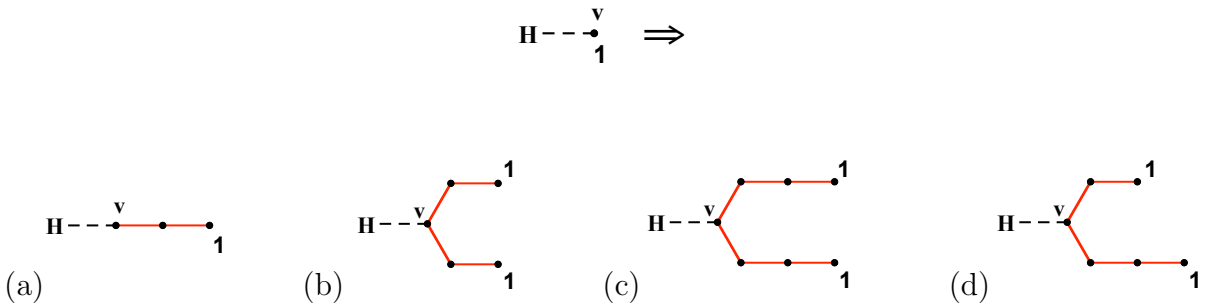

Figure 1: (a) The Tree Extension Lemma, (b) the 2-2 Branching Lemma, (c) the 3-3 Branching Lemma, (d) the 2-3 Branching Lemma. *The numerals beside some of the vertices indicate weights, with unnumbered vertices assumed to have the maximum possible weight of 4.*

The majority of our proof will be devoted to proving that these lemmas hold. We begin with the Tree Extension Lemma, which will be proven as a corollary of the following result.

Claim 3.2. *Suppose Builder can force a copy of the (2-colored) graphs J and K, with w a vertex of J having weight 1 and u a vertex of K having weight 1. Then, she can force one of the two graphs shown in Figure 2(a).*

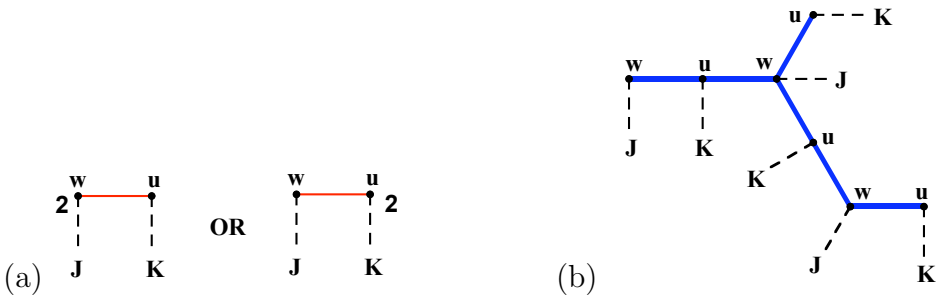

Figure 2: (a) The desired (red) connections between J and K, (b) forcing a blue T if the desired red connections are avoided. (Blue edges are shown thicker than red for convenience in gray-scale viewing.)

Proof. Taking advantage of the consistency of our Painter, Builder constructs many copies each of J and K. She then attempts to construct a blue copy of T as follows (see Figure 2(b)).

For each vertex of T, Builder picks a copy of either J or K such that if two vertices of T are adjacent, one vertex is represented by J and the other by K. For each copy of J or K thus chosen, the vertex w or u respectively is called the *apex*. Then, she puts all the edges of T in a sequence: She starts with an edge incident to a leaf of T; she then adds edges successively so that any prefix subsequence of edges forms a connected subgraph of T. Builder then moves through the sequence, reading off edges. As she reads off each edge e, she connects the respective apexes of the copies of J and K representing the endpoints of e. If any such connecting edge is red, then one of the graphs depicted in Figure 2(a) must be formed. Otherwise, all connecting edges are blue, and Builder eventually forces a blue copy of T. □

Corollary 3.3. *Suppose Builder can force copies of the (2-colored) graphs J and K, with w a vertex of J having weight 1 and u a vertex of K having weight 1. Then, she can force the graph shown in Figure 3(a), with J and K joined by a red edge incident to w and u.*

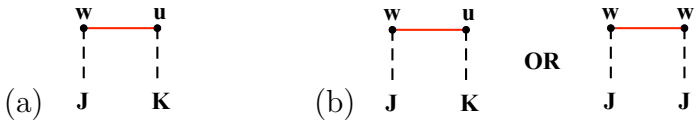

Figure 3: (a) The graph desired in Corollary 3.3, (b) the graphs desired in Claim 3.4.

The Tree Extension Lemma follows from the above corollary if we let J be H, w be v, and K be an isolated red edge.

Before proving the 2-2 Branching Lemma, we must prove several more results.

Claim 3.4. *Suppose Builder can force copies of the (2-colored) graphs J and K, with w a vertex of J having weight 2 and u a vertex of K having weight 1. Then, she can force one of the two graphs shown in Figure 3(b).*

Proof. For each vertex of T, Builder picks a copy of either J or K, with J chosen if the vertex has degree 1 or 2 and K chosen if the vertex is a fork (i.e., has degree 3). For each copy of J or K thus chosen, the vertex w or u respectively is called the *apex*. Then, for every two adjacent vertices of T, Builder connects the respective apexes of the two graphs representing the vertices. If any connecting edge is red, the claim is proven, since the edge must run either between the apex of a J and the apex of a K or else between the apexes of two copies of J. Otherwise, all the connecting edges are blue, and Builder has made a blue copy of T. □

We call a subtree U of T *fitting* if T and U share a leaf and every fork in T that is a vertex of U is either a leaf or a fork of U.

Claim 3.5. *Builder can force the weighted red-blue-red path shown in Figure 4(a).*

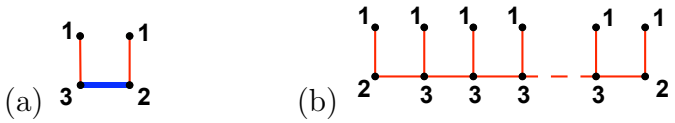

Figure 4: (a) The graph desired in Claim 3.5, (b) a comb.

Proof. Builder begins by constructing numerous copies of an isolated red edge ab and stringing them together by repeatedly connecting the b of the last ab in the chain to the b of a new copy of ab. If any of the connecting edges is blue, then the claim is proven. Otherwise, Builder can force an arbitrarily long red chain of the form shown in Figure 4(b). We call such a chain a *comb*, and let the *spine* of a comb refer to the path joining the two degree 2 vertices. Vertices not in the spine shall be referred to as *teeth*. It suffices to show that arbitrarily long combs allow Builder to win.

Suppose Builder can force arbitrarily long combs. We show that she can use the combs to construct a red copy of T. Such a copy of T may be expressed by an injective function f from $V(T)$ to the vertex set of the background graph of the game, such that two adjacent vertices in $V(T)$ are sent to vertices adjacent by a red edge.

Builder defines such a function progressively over $V(T)$. Builder starts at some leaf ℓ of the maple T, making its image under f be some spine vertex of a sufficiently large comb. If ℓ is at distance d from the nearest fork a in T, Builder then moves along the spine of the comb in some direction, making each new spine vertex the image of a vertex in T as adjacency warrants, until she has moved d edges along the spine and makes the spine vertex she has reached be the image of the fork a of T. She then proceeds to define elements of $f(V(T))$ in succession. At any given point in this process, and for any fork y whose image $f(y)$ has been defined, we say that a *y-successor fork* is a fork of T that abuts y and whose image has not yet been defined.

Assume that Builder has already defined $f(V(U))$ for some fitting subtree U of T, where, for every fork y of T that is a leaf of U, the following conditions hold:

- There is some subgraph $L(y)$ of the background graph isomorphic to a (sufficiently large) unweighted comb and containing $f(y)$ as a spine vertex.
- The tooth of $L(y)$ adjacent to $f(y)$ has degree 1 and is not in the image $f(V(U))$.
- Of the two pieces into which $f(y)$ divides the spine of $L(y)$, one piece, together with all teeth coming off of it, is sufficiently large and is *unused* - that is, the degrees of its vertices in the background graph are no higher than they would be in an isolated (weighted) comb, and none of the vertices are in the image $f(V(U))$.

Now consider some fork y_0 of T that is a leaf of U. Let $L_0 = L(y_0)$.

If there are no y_0-successor forks, then Builder moves along the spine of L_0 in the unused direction. As she passes over each spine vertex, she makes it the image of a vertex of T as adjacency warrants, and finally reaches the image of a leaf of T. To obtain

the image of the other leaf abutting y_0, Builder uses repeated applications of the Tree Extension Lemma to append a long red path to the tooth adjacent to y_0. She then moves along the tooth and appended path, assigning images as adjacency warrants until she reaches the image of the leaf.

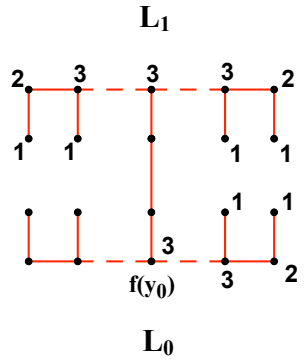

Figure 5: Connecting the combs L_0 and L_1.

If there is exactly one y_0-successor fork y_1 at distance d_1 from y_0, then Builder moves d_1 edges, one by one, along the spine of L_0 in the unused direction. As she passes over each spine vertex, she makes it the image of a vertex of T as adjacency warrants. After moving d_1 edges from $f(y_0)$, she lets $f(y_1)$ be the vertex she has reached. To obtain the image of the leaf abutting y_0, Builder again uses repeated applications of the Tree Extension Lemma to append a long red path to the tooth adjacent to y_0 and then assigns images along this path as adjacency warrants until reaching the image of the leaf. She now continues recursively as above, using y_1 in place of y_0.

Now suppose that there are two y_0-successor forks y_1 and y_2, at distances d_1 and d_2, respectively, from y_0. There are two cases to be considered.

Case 3.5.1. $d_2 \geq 3$.

In this case, Builder forces a new comb L_1 and, using Corollary 3.3, connects one of its teeth via a red edge to the tooth of L_0 adjacent to $f(y_0)$ (see Figure 5). Builder moves along teeth from L_0 to L_1 and then along the spine of L_1, assigning images to vertices of T as adjacency warrants. When she has moved d_2 edges away from $f(y_0)$, she lets $f(y_2)$ be the vertex she has reached. If $d_2 > 3$, then she will have assigned $f(V(U))$ for a subtree U of T as our conditions required, and she continues recursively as above.

If $d_2 = 3$, then $f(y_2)$ is the only vertex on the spine of L_1 that has yet been assigned to $f(V(T))$. Builder now moves along the spine of L_1 in each possible direction in turn, assigning vertices to $f(V(T))$ until she reaches the image of a fork (or a leaf), at which point she continues recursively as above (or stops).

Case 3.5.2. $d_2 = 2$.

Let the background graph obtained up until this point be Q. Now, Builder forces a new comb L_1 and, using Claim 3.2, connects one of its teeth by a red edge to the tooth of L_0 adjacent to $f(y_0)$, generating either graph A or B as shown in Figure 6(a), where the vertices x_0 in A and B are as shown.

Sub-case 3.5.2.1. *Graph A is obtained.*

Builder forces another new comb L_2, and, using Claim 3.4, either doubles A at x_0 or else connects A by x_0 to one of the teeth of L_2. One of the graphs A_1 or A_2 shown in Figure 6(b) must be obtained. Let x_1, x_2, x_3, x'_0, x'_1, L'_0, and L'_1 be as shown.

Suppose first that A_1 is obtained. Then, Builder lets $f(y_2)$ be x_0. If there are no y_2-successor forks, then Builder continues through x_1 into L_1 and along the spine, and through x'_0 into L'_1 and along the spine, assigning elements of $f(V(T))$ until assignment is complete.

If there is exactly one y_2-successor fork y_3, then Builder continues through x_1 into L_1 and along the spine, assigning elements of $f(V(T))$ until she reaches $f(y_3)$. She also moves from x_0 through x'_0 into L'_1, assigning elements of $f(V(T))$ until she reaches a leaf.

If there are two y_2-successor forks y_3 and y_4, then Builder continues through x_1 into L_1 and along the spine, assigning elements of $f(V(T))$ until she reaches $f(y_3)$ on the spine of L_1. She also moves from x_0 through x'_0 into L'_1 and along the spine, assigning elements of $f(V(T))$ until she reaches $f(y_4)$. If $f(y_4) = x'_1$, she then moves along the spine of L'_1 in each possible direction in turn, assigning vertices to $f(V(T))$ until she reaches the image of a fork (or a leaf), at which point she continues recursively as above (or stops).

If A_2 is obtained, Builder goes through exactly the same process as for A_1, with L_2, x_2, and x_3 replacing L'_1, x'_0, and x'_1, respectively, throughout.

Sub-case 3.5.2.2. *Graph B is obtained.*

Builder first forces a copy B' of B, having x'_0 in place of x_0. She then forces a copy Q' of the previously obtained background graph Q, having vertex x_2 in place of the vertex $f(y_0)$ and with x_3 the leaf adjacent to x_2. Using Claim 3.4, Builder either connects B at x_0 by a red edge with its copy B' at x'_0 or else connects the copy B' at x'_0 by a red edge to Q' at x_3. One of the graphs B_1 or B_2 shown in Figure 6(c) must be obtained. Let x_1, x'_1, L'_0, and L'_1 be as shown.

Suppose first that B_1 is obtained. Then, Builder lets $f(y_2)$ be x'_0. If there are no y_2-successor forks, then Builder continues into the unused half of L'_0 and also through x'_1 into L_1 and along the spine, assigning elements of $f(V(T))$ until assignment is complete.

If there is some y_2-successor fork y_3, then Builder continues through x'_1 into L'_1 and along the spine, assigning elements of $f(V(T))$ until she reaches $f(y_3)$. If $f(y_3) = x'_1$, she then moves along the spine of L'_1 in each possible direction in turn, assigning vertices to $f(V(T))$ until she reaches the image of a fork (or a leaf), at which point she continues recursively as before (or stops). Following assignment of $f(y_3)$, Builder moves from x'_0 into the unused half of L'_0, assigning elements of $f(V(T))$ until she reaches the image of another y_2-successor fork (or a leaf), at which point she continues recursively as above (or stops).

Suppose now that B_2 is obtained. In this case, Builder reassigns from B to Q' the elements of $f(V(T))$ which she has hitherto defined, so that, for instance, $f(y_0)$ now equals x_2. Proof continues as when B_1 is obtained.

The sub-case and case are finally complete.

To complete the recursive step, Builder assigns the images of y_1 and of the vertices on the path between y_0 and y_1 to vertices on the unused half of the spine of L_0. □

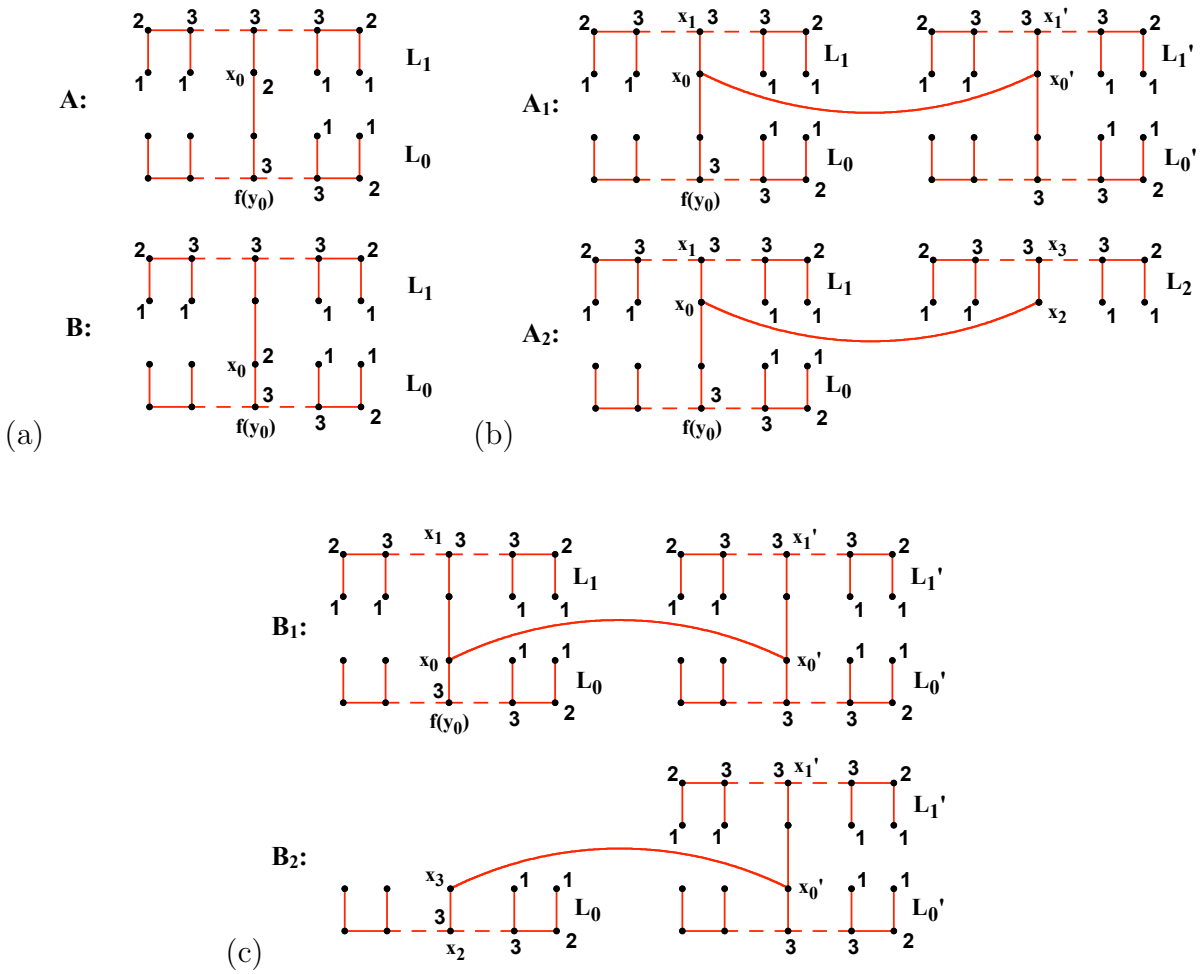

Figure 6: (a) The graphs A and B, (b) the graphs A_1 and A_2, (c) the graphs B_1 and B_2.

We label as x and y two of the vertices of the graph X in Figure 4(a), as shown in Figure 7(a).

Claim 3.6. *Suppose that Builder can force some (2-colored) graph H', with v' a vertex of H' having weight 2. Then Builder can force the graph shown in Figure 7(b).*

Proof. Builder forces many copies of the graph H', of the graph X, and of isolated (red) edges ab. The vertices v' in H', x and y in X, and a in edges ab are all called *apexes*. She

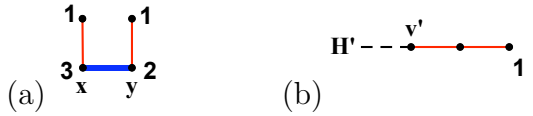

Figure 7: (a) Graph X with vertices x and y marked, (b) the graph desired in Claim 3.6.

puts all the vertices of T in a sequence $S = w_1, w_2, \ldots$ as follows: starting at a leaf ℓ of T, she successively adds vertices so that any prefix subsequence forms the vertex set of a connected subtree of T.

Builder now reads off this sequence, picking an apex $g(w_n)$ for each entry w_n and connecting each apex to a previously chosen entry so that $g(w_i), g(w_j)$ are connected if and only if w_i, w_j are. The kind of apex she chooses for the various entries of the sequence are as follows.

Builder picks an apex of the form v' for the image $g(w_1)$ of the first entry of S. Now consider w_n, for some $n \geq 2$. We know that w_n must be adjacent to some (unique) vertex w_m with $m < n$. Then, Builder picks $g(w_n)$ to be of the form

- v' if $g(w_m)$ is of the form y or a.
- x if $g(w_m)$ is of the form v' and the vertex w_n is adjacent to a fork w_p with $n < p$.
- y if $g(w_m)$ is of the form x - in this case, $g(w_m)$ and $g(w_n)$ should be apexes of the same copy of X and will therefore already be connected by a (blue) edge.
- a if $g(w_m)$ is of the form v' and if the vertex w_n is not adjacent to any fork w_p such that $n < p$.

It is readily verified that none of the edges between apexes can be red without yielding the graph in Figure 7(b). But if all these edges are blue, a blue T is formed. Hence, the claim is proven. □

Proof of the 2-2 Branching Lemma. Builder now forces many copies of the graph H, of the graph X, and of isolated (red) edges ab. The vertices v in H, x and y in X, and a in edges ab are all called *apexes*. Again, Builder puts all the vertices of T in a sequence $S = w_1, w_2, \ldots$ as follows: starting at one leaf ℓ of T, she successively adds vertices so that any prefix subsequence forms the vertex set of a connected subtree of T.

Builder now reads off this sequence, picking an apex $g(w_n)$ for each entry w_n and connecting each apex to a previously chosen entry so that $g(w_i), g(w_j)$ are connected if and only if w_i, w_j are. The kind of apex she chooses for the various entries of the sequence are as follows.

Builder picks an apex of the form v for the image $g(w_1)$ of the first entry of S. Now consider w_n, for some $n \geq 2$. We know that w_n must be adjacent to some vertex w_m with $m < n$. Then, Builder picks $g(w_n)$ to be of the form

- v if $g(w_m)$ is of the form y or a.

- x if $g(w_m)$ is of the form v and the vertex w_n is adjacent to a fork w_p with $n < p$.
- y if $g(w_m)$ is of the form x - in this case, $g(w_m)$ and $g(w_n)$ should be apexes of the same copy of X and will therefore already be connected by a (blue) edge.
- a if $g(w_m)$ is of the form v and if the vertex w_n is not adjacent to any fork w_p such that $n < p$.

Builder stops connecting apexes if at any point the edge between two apexes is colored red by Painter, which must happen at some point if Painter is to prevent the formation of a blue T. Consider such a stoppage, precipitated by the red edge between $g(w_\alpha)$ and $g(w_\beta)$, where $\alpha < \beta$.

$(*)$ If $g(w_\beta)$ is of the form v, then applying Claim 3.6 with $g(w_\beta)$ as v' proves the 2-2 Branching Lemma.

If $g(w_\beta)$ is not of the form v, then $g(w_\alpha)$ must be of the form v. In this case, Builder creates an apex of the same apex type as $g(w_\beta)$ and connects it to $g(w_\alpha)$, reassigning the label $g(w_\beta)$ to the new apex. If the edge between $g(w_\alpha)$ and the new $g(w_\beta)$ is red, then the 2-2 Branching Lemma is proven. Otherwise, any red-edge stoppage is moved to further along the sequence S.

Since the sequence S is finite and stoppages cannot be pushed infinitely far along it, the 2-2 Branching Lemma must hold if a blue T is to be prevented. □

Before proving the final two branching lemmas, we prove some additional results.

Let graphs L, M, and N be as shown in Figure 8(a).

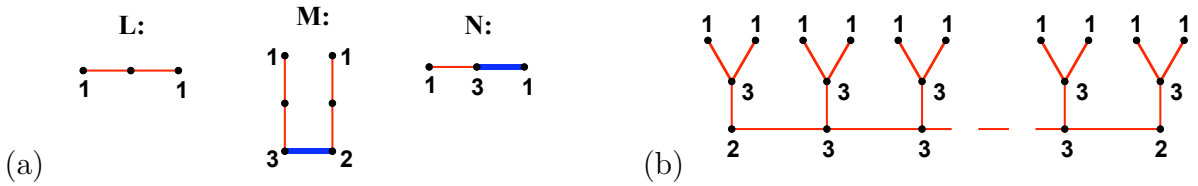

Figure 8: (a) The graphs L, M, and N, (b) a multiclaw.

Claim 3.7. *Builder can force L.*

Proof. Builder tries to construct a blue T. Assume that she has constructed some blue fitting subtree U of T such that all leaves of U are of weight 1. Then, given a leaf v of U, Builder draws three edges from it to new vertices. If at least two of the new edges are red, L is obtained. Otherwise, at least two of the three edges are blue and so U is enlarged. Hence, if a blue T is to be prevented, Builder must be able to force L. □

Claim 3.8. *Builder can force either M or N.*

Proof. We assume towards contradiction that Builder can force neither M nor N.

Builder begins by constructing an isolated (red) edge, then connecting one endpoint of this edge to two new vertices. Since Builder cannot force N, Painter must color both these new edges red, showing that Builder can force a red claw with weight 3 in the center and weight 1 on each leaf. We let C denote this (2-colored weighted) claw.

Define a *multiclaw* to be a graph of the form shown in Figure 8(b), obtained by stringing together many copies of C. Let the *spine* of a multiclaw be the long subpath depicted in Figure 8(b) as extending horizontally along the length of the multiclaw. We define the *length* of a multiclaw to be the number of edges in the spine.

Builder can force an arbitrarily long multiclaw as follows: She starts with C (a length-0 multiclaw). If she has already forced a length-n multiclaw, she can force a length-$(n+1)$ multiclaw by connecting a terminal spine vertex of the multiclaw she has made to a leaf of a new copy of C. If the connecting edge is blue, then the graph M is formed, so all such connecting edges must be red.

Now, we show that Builder can force a red copy of T. Such a copy of T may be expressed by an injective function h from $V(T)$ to the vertex set of the background graph of the game, such that two adjacent vertices in $V(T)$ are send to vertices adjacent by a red edge.

Builder defines such a function progressively over $V(T)$. She starts at some leaf of the maple T, making its image under h be some spine vertex of a long multiclaw. If the chosen leaf is at distance d from its nearest fork a in T, Builder then moves along the spine of the multiclaw in some direction, making each new spine vertex the image of a vertex in T as adjacency warrants, until she has moved d edges along the spine and reached $h(a)$. She then proceeds to define elements of $h(V(T))$ in succession. At any given point in this process, and for any fork y whose image $h(y)$ has been defined, we say that a *y-successor* is a fork or leaf of T that abuts y and whose image has not yet been defined.

Assume that Builder has already defined $h(V(U))$ for some fitting subtree U of T, where for any fork y of T that is a leaf of U, the following conditions hold:

- As, shown in Figure 9(a), $h(y)$ is connected by red edges both to a multiclaw and to a red P_3 having weight 1 at its endpoints and weight 2 at its center.
- Both this multiclaw, which we call $Z(y)$, and this P_3, which we call $P(y)$, are disjoint from $h(V(U))$.

Now consider some fork y_0 of T that is a leaf of U. Let y_1 and y_2 be the two successors of y_0, at distances d_1 and d_2 from y_0, respectively.

Builder first moves along the spine of $Z(y_0)$, assigning images to elements of $V(T)$ as adjacency warrants, until she has moved distance d_1 from $h(y_0)$ and reached $h(y_1)$. If y_1 is a leaf, then she stops, if a fork, then she can proceed as above, using y_1 in place of y_0.

To assign $h(y_2)$, Builder does as follows: She takes a leaf r of $P(y_0)$ and attaches it to a leaf of a new copy of C, then to a leaf of another new copy of C. Both connecting edges must be red to prevent the formation of M. Builder then continues to add new copies of C, one by one, to the copies C she has just attached, so that r becomes connected to two multiclaws Z_1 and Z_2 of large length, as shown in Figure 9(b).

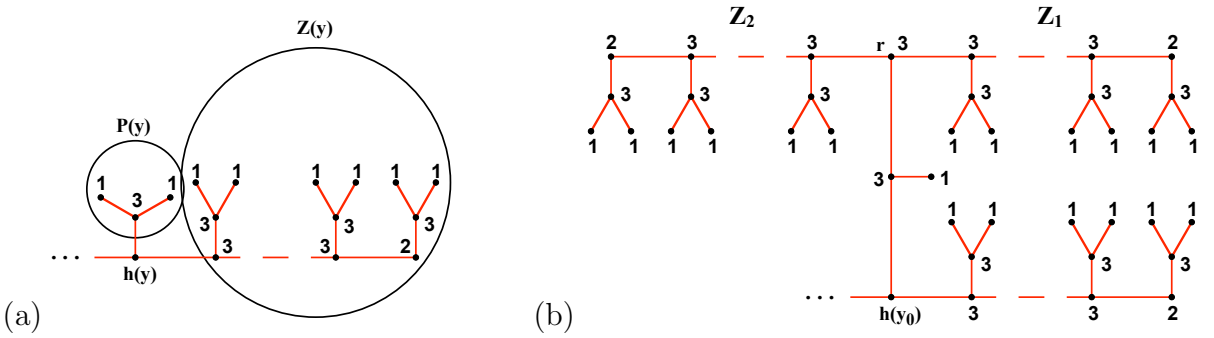

Figure 9: (a) requirement on $h(y)$, (b) r connected to the multiclaws Z_1 and Z_2.

Case 3.8.1. $d_2 \geq 3$.

Builder moves from $h(y_0)$ through r along the spine of Z_1, assigning images to elements of $V(T)$ as adjacency warrants, until she has moved distance d_2 from $h(y_0)$ and reached $h(y_2)$. If y_2 is a leaf, she stops, if a fork, then she can proceed as above (using y_2 in place of y_0).

Case 3.8.2. $d_2 = 2$.

Builder moves from $h(y_0)$ to r, assigning images to elements of $V(T)$ as adjacency warrants, so that $r = h(y_2)$. If y_2 is a leaf, Builder stops. Otherwise, y_2 is a fork, with two successors y_3 and y_4, at distances d_3 and d_4 from y_2, respectively.

Builder now first moves along the spine of Z_1, assigning images to elements of $V(T)$ as adjacency warrants, until she has moved distance d_3 from $h(y_2)$ and pauses at $h(y_3)$. If y_3 is a leaf, then she stops, if a fork, then she can proceed as above (using y_3 in place of y_0).

Builder now moves along the spine of Z_2, assigning images to elements of $V(T)$ as adjacency warrants, until she has moved distance d_4 from $h(y_2)$ and pauses at $h(y_4)$. If y_4 is a leaf, then she stops, if a fork, then she can proceed as above (using y_4 in place of y_0).

We conclude that Builder can eventually assign images to all elements of $V(T)$ and hence can create a red copy of T, a contradiction. We conclude that Builder must be able to force either M or N. □

Let the vertices c of L and m, n of M be as shown in Figure 10(a).

Claim 3.9. *Suppose that Builder can force M, and that Builder can force some (2-colored) graph H', with v' a vertex of H' having weight 2. Then Builder can force the graph shown in Figure 10(b).*

Proof. The proof mirrors the proof of Claim 3.6, with M replacing X throughout and L replacing the isolated edge ab. The vertices m, n of M and c of L replace the vertices x, y of X and a of ab, respectively. □

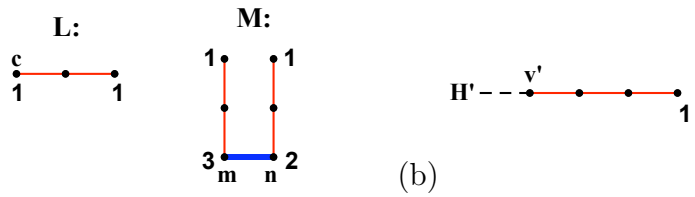

Figure 10: (a) the graphs L and M with vertices c, m, and n labeled, (b) the graph desired in Claim 3.9.

Claim 3.10. *Suppose that Builder can force M. Then, the 3-3 and 2-3 Branching Lemmas hold.*

Proof. The proof of the 3-3 Branching Lemma mirrors the proof of the 2-2 Branching Lemma (following the proof of Claim 3.6), with the following modifications: (i) M replaces X throughout and L replaces the isolated edge ab, with the vertices m, n of M and c of L replacing the vertices x, y of X and a of ab, respectively; and (ii) invocation of Claim 3.6 is replaced by invocation of Claim 3.9.

The proof of the 2-3 Branching Lemma mirrors the proof of the 2-2 Branching Lemma without modifications up until $(*)$. At the point $(*)$, the proof continues as follows:

$(\dagger)$ If $g(w_\beta)$ is of the form v, then applying Claim 3.9 with $g(w_\beta)$ as v' proves the 2-3 Branching Lemma.

If $g(w_\beta)$ is not of the form v, then $g(w_\alpha)$ must be of the form v. In this case, Builder creates a new apex, such that if $g(w_\beta)$ is of the form x or a, then the new apex is of the form m or c, respectively. Builder then connects the new apex to $g(w_\alpha)$ and reassigns the label $g(w_\beta)$ to the new apex. If the edge between $g(w_\alpha)$ and the new $g(w_\beta)$ is red, then the 2-3 Branching Lemma is proven.

Otherwise, any red-edge stoppage is moved to further along the sequence S. Construction of images under g of elements of S then proceeds as in the proof of the 2-2 Branching Lemma, except with the following modification: M replaces X throughout and L replaces the isolated edge ab, with the vertices m, n of M and c of L replacing the vertices x, y of X and a of ab, respectively.

If a blue T is to be prevented, a red-edge stoppage next occurs between some $g(w_\gamma)$ and $g(w_\delta)$, with $\gamma < \delta$ and $\delta > \beta$. If $g(w_\delta)$ is of the form v, then applying Claim 3.6 with $g(w_\delta)$ as v' proves the 2-3 Branching Lemma.

If $g(w_\delta)$ is not of the form v, then $g(w_\gamma)$ must be of the form v. In this case, Builder creates a new apex, such that if $g(w_\delta)$ is of the form m or c, then the new apex is of the form x or a, respectively. Builder then connects the new apex to $g(w_\gamma)$ and reassigns the label $g(w_\delta)$ to the new apex. If the edge between $g(w_\gamma)$ and the new $g(w_\delta)$ is red, then the 2-3 Branching Lemma is proven.

Otherwise, any red-edge stoppage is moved to further along the sequence S. Construction of images under g of elements of S then proceeds as in the proof of the 2-2 Branching Lemma, with no modifications. If a blue T is to be prevented, a red-edge stoppage occurs

between some $g(w_\epsilon)$ and $g(w_\zeta)$, with $\epsilon < \zeta$ and $\zeta > \delta$. Proof proceeds as from (†), with ϵ, ζ replacing α, β, respectively.

Since the sequence S is finite and stoppages cannot be pushed infinitely far along it, the 2-3 Branching Lemma must hold if a blue T is to be prevented. □

We may therefore assume that Builder can force N.

Suppose Builder can force a (2-colored) graph J having weight-1 vertex t. We say that the *Fork Property* holds in red if Builder can force at least one of the three graphs depicted in Figure 11(a) and that the *Elongation Property* holds in red if Builder can force at least one of the two graphs depicted in Figure 11(b). Let graphs Z_1, Z_2, $Z_{1,1}$, $Z_{1,2}$, and $Z_{2,2}$ be as depicted in Figure 11(c). Then, say that the *Fork-Fork Property* holds in red if Builder can force either Z_1 or else both $Z_{1,1}$ and $Z_{1,2}$ and that the *Fork-Extend Property* holds in red if Builder can force either Z_2 or else both $Z_{1,2}$ and $Z_{2,2}$. Say that the *Iterated Fork Property* holds in red if both the Fork-Fork Property and Fork-Extend Property hold in red.

We likewise say that one of the above Properties holds in blue if Builder can force the appropriate graphs with those edges being blue which are shown as red in Figure 11.

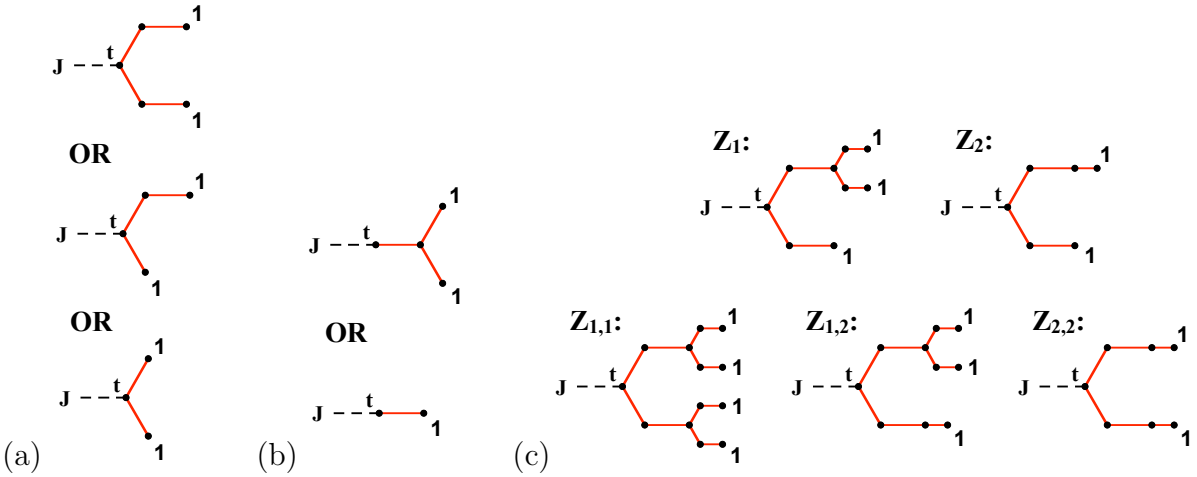

Figure 11: (a) the Fork Property, (b) the Elongation Property, (c) the graphs involved in the Iterated Fork Property.

Claim 3.11. *Suppose that, for all J and weight-1 t that Builder can force, the Fork Property holds in blue. Then, Builder can win.*

Proof. Builder maintains two trees T_1 and T_2, with T_1 red and T_2 blue, each a copy of a fitting subtree of T. For $i = 1, 2$, a certain leaf of T_i is called the *root*, corresponding with a leaf of T; the root remains fixed even as T_i grows. Every leaf of T_i except the root is of weight 1. We let $p_i : V(T_i) \to V(T)$ be the function taking vertices of T_i to corresponding vertices of T. The color of T_i is denoted $c(T_i)$.

Initially, both trees are a single vertex (the root). Builder's strategy is to apply the Fork and Elongation Properties to progressively enlarge T_1 and T_2 (applying properties to T_i in the color $c(T_i)$). Given a leaf x of T_i, Builder applies the following:

- Nothing, if $p_i(x)$ is already a leaf of T.
- The Fork Property, if $p_i(x)$ is a fork of T.
- The Elongation Property, if $p_i(x)$ is adjacent to a fork y of T such that $y \notin p_i(V(T_i))$.
- Either the Fork Property or the Elongation Property (it doesn't matter which) if $p_i(x)$ is not a leaf and is at distance at least 2 from any fork y of T such that $y \notin p_i(V(T_i))$. In this case the whole strength of the Fork and Elongation Properties is not used, and the edges added to T_i are either (i) a single edge in $c(T_i)$ starting at x and with terminal vertex of weight 1, or (ii) a path of two edges in $c(T_i)$ starting at x and with terminal vertex of weight 1.

The process will terminate after a finite amount of time, with $p_i(V(T_i))$ being all of $V(T)$. It only remains to be shown that Builder can apply the Fork and Elongation Properties in the manner desired. By assumption, the Fork Property holds in blue, and by the 2-2 Branching Lemma, it holds in red. It suffices to show that at any point either Builder can apply the Elongation Property to T_1 in red or else she can apply it to T_2 in blue.

Consider T_1 and T_2 at such a point that the Elongation Property is required by T_1 at the vertex x_1 and by T_2 at the vertex x_2. Builder first constructs two edges emanating from each x_i, with the other endpoints of these edges at new vertices. If either edge at x_i is in $c(T_i)$, the Elongation Property is achieved in $c(T_i)$ on T_i and we are done. Otherwise, both edges at x_1 are in blue and both edges at x_2 are in red. Connecting x_1 and x_2 by an edge then forces the Elongation Property either in red on T_1 or in blue on T_2. □

Claim 3.12. *Suppose that, for all J and weight-1 t that Builder can force, either the Fork or Iterated Fork Property holds in blue. Then, Builder can win.*

The proof of this claim follows closely the proof of the previous claim. The graph Z_1 depicted in Figure 11(c) can be thought of as a composite of the Fork Property with another Fork Property applied at one of the leaves of the fork. The graph Z_2 is a composite of the Fork Property with the Elongation Property applied at one of the leaves of the fork. The graphs $Z_{1,1}$, $Z_{1,2}$, and $Z_{2,2}$ may be described similarly as composites of the Fork Property with some combination of Fork and Elongation Properties applied at the leaves of the fork.

We now finally are able to prove the 3-3 and 2-3 Branching Lemmas.

Proof of the 3-3 Branching Lemma. Per Claim 3.11, Builder begins by forcing some J and t for which the Fork Property in blue does not hold. Builder also forces the H and v in the statement of the 3-3 Branching Lemma. Let the graphs C_1, C_2, D_1, D_2 be as depicted in Figure 12.

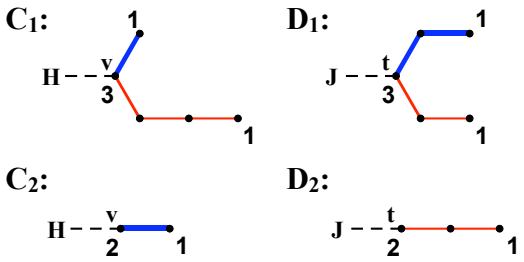

Figure 12: The graphs C_1, C_2, D_1, and D_2.

If Builder can force C_1 and D_1, then the 3-3 Branching Lemma holds, since connecting v in C_1 to t in D_1 yields either the Fork Property in blue at t or else the 3-3 Branching Lemma at v. Hence, if Builder can force C_1 and D_2, then the 3-3 Branching Lemma holds, since connecting v in C_1 to t in D_2 yields either D_1 or else yields the 3-3 Branching Lemma at v. Also, if Builder can force C_2 and D_1, then the 3-3 Branching Lemma holds, since connecting v in C_2 to t in D_1 yields either C_1 or else yields the Fork Property in blue at t. We conclude that if Builder can force C_2 and D_2, then the 3-3 Branching Lemma is proven, since connecting v in C_2 to t in D_2 yields either C_1 or D_1. Hence, it suffices to show that Builder both can force either C_1 or C_2 and can force either D_1 or D_2.

We first prove that, if the 3-3 Branching Lemma does not hold, Builder can force either C_2 or C_1. Builder makes an edge vw from v in H to a new vertex w. If vw is blue, then C_2 is obtained. Otherwise, vw is red, and Builder applies the Tree Extension Lemma at w. She then makes an edge from v to a new vertex x. If vx is blue, then C_1 is obtained. Otherwise, vx is red, and Builder applies the Tree Extension Lemma at x, proving the 3-3 Branching Lemma.

Now, we prove that Builder can force either D_2 or D_1. Builder first makes an edge from t in J to the central vertex q in the graph N shown in Figure 8(a). If tq is red, then D_2 is obtained. Otherwise, tq is blue, and Builder draws a new edge from t in J to the vertex q in a different copy of N. If this new edge is red, then D_1 is obtained. Otherwise, the Fork Property in blue holds at t.

Hence, if the 3-3 Branching Lemma does not hold, then Builder both can force C_1 or C_2 and can force D_1 or D_2, completing our proof. □

Proof of the 2-3 Branching Lemma. Per Claim 3.12, Builder begins by forcing some J and t for which neither the Fork nor the Iterated Fork Property holds in blue at t. Builder also forces H and v as in the statement of the 2-3 Branching Lemma. Let the graphs $C_1, C_2, C_3, D_1, D_2, E$ be as depicted in Figure 13, with the vertex u of E as marked.

Builder begins by taking a copy of H and constructing an edge from v to a new vertex w. If this edge vw is red, then applying the Tree Extension Lemma at w yields C_3. However, if Builder can force C_3, Claim 3.6 implies that the 2-3 Branching Lemma holds. Hence, we may assume vw is blue, in which case C_2 has been created.

We now prove that Builder can force D_2 or D_1. Builder first makes an edge from t in J to the vertex q in a copy of N. If tq is red, then D_2 is obtained. Otherwise, tq is blue,

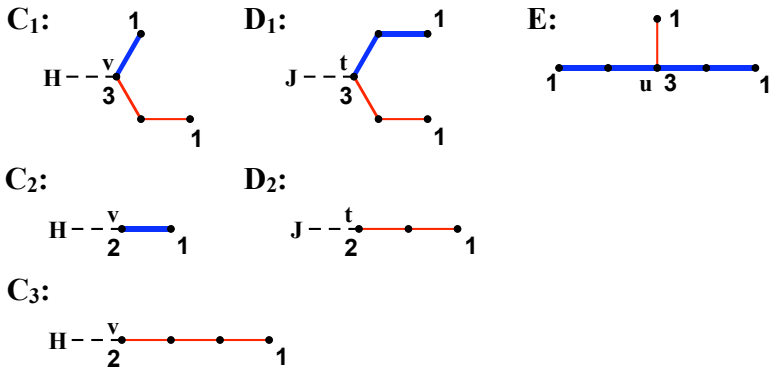

Figure 13: The graphs C_1, C_2, C_3, D_1, D_2, and E.

and Builder draws a new edge from t in J to the central vertex q in a different copy of N. If this new edge is red, then D_1 is obtained. Otherwise, the Fork Property in blue holds at t.

If Builder can force C_1 and D_1, then the 2-3 Branching Lemma holds, since connecting v in C_1 to t in D_1 yields either the Fork Property in blue at t or else the 2-3 Branching Lemma at v. Hence, if Builder can force C_1 and D_2, then the 2-3 Branching Lemma holds, since connecting v in C_1 to t in D_2 yields either D_1 or else the 2-3 Branching Lemma at v. Since we know Builder can force either D_1 or D_2, we conclude that if Builder can force C_1, then the 2-3 Branching Lemma holds.

Builder now forces two copies of C_2 and one isolated red edge ab. She proceeds to connect the vertex a with the vertex v in each copy of C_2. If either edge is red, then C_1 has been formed and so the 2-3 Branching Lemma holds. We may therefore assume that both edges are blue, in which case E has been created.

Our strategy from here on is to prove that the Iterated Fork Property holds in blue at t if the 2-3 Branching Lemma does not hold at v and the Fork Property does not hold in blue at t. We must show that both the Fork-Fork and Fork-Extend Properties hold in blue at t.

Builder begins by forcing a copy of either D_1 or D_2, as well as a copy of H. She then connects the v in the H with the t in the D_i. If the connecting edge is red, then C_3 has been produced and the 2-3 Branching Lemma holds. Otherwise, the edge is blue.

Builder now adds an edge from v to a new vertex s. If vs is blue, then, if $i = 1$, the Fork Property holds in blue at t, and, if $i = 2$, D_1 is created, allowing for a repetition of the previous steps with $i = 1$. We may therefore assume that vs is red. Builder applies the Tree Extension Lemma at s. We denote by P the background graph at this point (see Figure 14(a)).

(‡) We now show that the Fork-Fork Property holds: namely, that Builder can force either Z_1 (as shown in Figure 11(c)) or else both $Z_{1,1}$ and $Z_{1,2}$. Builder forces E and connects the vertex u in E with the existing vertex v. If the connecting edge is red, then the 2-3 Branching Lemma holds. Otherwise, uv is blue. Then, if $i = 1$, Z_1 is forced. Suppose then that $i = 2$.

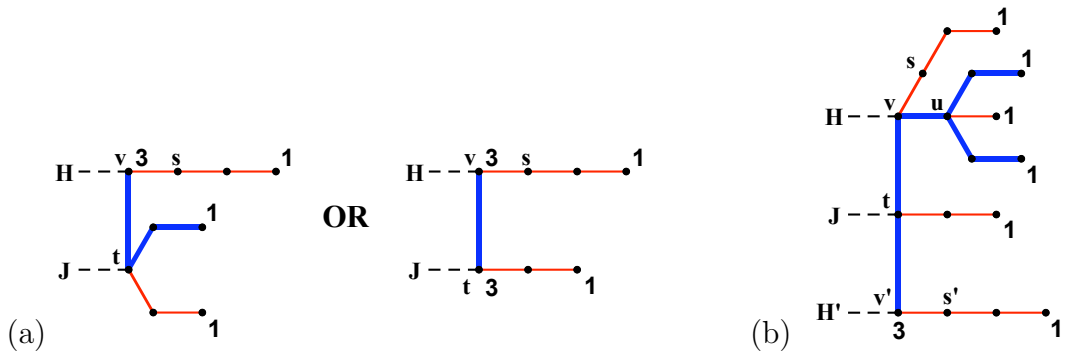

Figure 14: (a) The graph P if D_i is D_1 or D_2, respectively, (b) the graph Q.

Builder now forces another copy H' of H and connects its copy v' of v with the existing vertex t. If $v't$ is red, then C_3 is produced. Otherwise, $v't$ is blue. Builder now adds an edge from v' to a new vertex s'. If $v's'$ is blue, then Z_1 is forced. We may therefore assume $v's'$ is red. Builder applies the Tree Extension Lemma at s'. We denote by Q the background graph at this point (see Figure 14(b)).

Builder now forces another copy E' of E and connects its copy u' of u with the existing vertex v'. If the connecting edge is red, then the 2-3 Branching Lemma holds. Otherwise, $u'v'$ is blue, in which case $Z_{1,1}$ is forced.

Builder now forces another copy of Q and also a copy of N, drawing the edge $v'q$ between them. If $v'q$ is red, then the 2-3 Branching Lemma holds. Otherwise, $v'q$ is blue, in which case $Z_{1,2}$ is forced. We conclude that Builder can force either Z_1 or else both $Z_{1,1}$ and $Z_{1,2}$.

We must also show that the Fork-Extend Property holds. Builder here forces another copy of the graph P and proceeds from (‡) with the modification that in the paragraph (‡) the graph E is replaced by N and the vertex u by q.

We conclude that both the Fork-Fork and Fork-Extend Properties hold in blue at t, implying that the Iterated Fork Property must as well. This is a contradiction, and so the 2-3 Branching Lemma holds. □

We have at last proven the Tree Extension, 2-2 Branching, 3-3 Branching, and 2-3 Branching Lemmas. Builder can use these lemmas to force a red copy of T. She does this by forcing a red copy of the maple T' obtained from T by appending edges as necessary to the leaves of T such that each leaf is at even distance from its nearest fork. This red copy of T' may be expressed by an injective function k from $V(T')$ to the vertex set of the background graph of the game, such that two adjacent vertices in $V(T')$ are send to vertices adjacent by a red edge.

Builder defines such a function progressively over $V(T')$. She starts at some leaf u of the maple T', assigning $k(u)$ to the vertex c of a copy of the graph L (see Figure 10(a)) and making the remaining vertices of L be images of elements of $V(T')$ as adjacency warrants.

Assume that Builder has already defined $k(V(U))$ for some fitting subtree U of T',

where all leaves of U have images of weight 1. We let $p : V(U) \to V(T')$ be the natural inclusion function. Given a leaf v of U such that $p(v)$ is a fork, we let a *v-successor* be a fork or leaf that abuts $p(v)$ and is not in $p(V(U))$. Each leaf v of U such that $p(v)$ is a fork must have exactly two v-successors.

Then, given a leaf v of U such that $p(v)$ is not a leaf, Builder applies at $k(v)$ either the Tree Extension Lemma or one of the Branching Lemmas, as follows:

- Builder applies the Tree Extension Lemma if $p(v)$ is not a fork.
- Builder applies the 2-2 Branching Lemma if $p(v)$ is a fork and if the two v-successors are both at even distance from v.
- Builder applies the 3-3 Branching Lemma if $p(v)$ is a fork and if the two v-successors are both at odd distance from v.
- Builder applies the 2-3 Branching Lemma if $p(v)$ is a fork and if one of the two v-successors is at even distance from v while the other is at odd distance.

After applying at $k(v)$ the Tree Extension Lemma or one of the Branching Lemmas, Builder assigns images under k of elements of $V(T')$ as adjacency warrants.

This process ends exactly when $U = T'$. Builder thus can create a red copy of T' and hence of T.

Our proof of Theorem 3.1 is finally complete. We thus have classified all trees T satisfying $\mathring{R}_\Delta(T) \leq 4$. Butterfield et al. [1] described all trees T satisfying $\mathring{R}_\Delta(T) \leq 3$; these trees are the paths P_n and the claw $K_{1,3}$. Hence, we obtain immediately the following as a corollary to Theorem 3.1:

Corollary 3.13. *The set of trees T for which $\mathring{R}_\Delta(T) = 4$ is the set of maples with the addition of the graph $K_{1,4}$ and with the removal of the claw $K_{1,3}$ and of all paths P_n.*

4 Further Questions

It is our hope that our classification of trees T satisfying $\mathring{R}_\Delta(T) = 4$ will be useful in the development of a general classification of graphs G which satisfy $\mathring{R}_\Delta(G) = 4$. We will shortly be presenting a proof that all cycles satisfy this equation, as well as a description of many other graphs which satisfy it. However, the problem of full classification appears to us to be quite challenging.

Many other questions are open relating to on-line degree Ramsey numbers, including the following, suggested in [1]: Can graphs G with maximum degree fixed at d have arbitrarily large on-line degree Ramsey numbers? Additionally, the generalizations of our results and those in [1] to a game with additional colors could prove interesting.

Acknowledgments

This research was performed at the University of Minnesota Duluth REU and was supported by the National Science Foundation (grant number DMS-0754106) and the National Security Agency (grant number H98230-06-1-0013).

I would to thank Joe Gallian for supervising this research. I am indebted to Prof. Gallian, Maria Monks, Nathan Pflueger, and the other students and participants in the Duluth REU for all the advice and support that they have given me.

References

[1] J. Butterfield, T. Grauman, B. Kinnersley, K. Milans, C. Stocker, and D. West, *On-line Ramsey theory for bounded degree graphs*, preprint.

[2] D. Conlon, *On-line Ramsey Numbers*, SIAM J. Discrete Math., 23 (2009), 1954-1963.