Graduate Algorithms *James Hendricks, Jeffrey Pang, and Bryan Parno*

Problem Set 3

March 4, 2005

Problem 1. No. Couterexample (see Figure 1). If we start with the tree on the left and perform *splay*(2), then there is no sequence of *splay* operations to return to the starting tree. Note that we have enumerated all possible splay operations from each tree in the right side of the figure, since there are only 3 possible nodes to splay and one of them is the root (hence splaying it would result in the same tree); thus, there are only two possible transitions from each tree.

Problem 2. Our $O(n)$ 6-color algorithm is as follows: 1) Find a vertex with 5 or fewer neighbors. 2) Push that vertex onto a stack and remove it from the graph. Repeat. 3) When there are no more vertices in the graph, pop each vertex from the stack, return it to the graph, and color it one of the colors not used by its neighbors. Below is a proof of runtime and correctness along with implementation details:

Data structures: Each vertex has a *color* and a *degree*. There is a set of all vertices with degree less than or equal to five called the *eligibleList*; because the graph is planar, this set is never empty if the graph is not empty. There is also a stack of vertices called the *stack*.

Step 1:

For each edge (u, v), increment $degree(u)$ and $degree(v)$.
For each vertex v, if $degree(v) <= 5$, add it to the $eligibleList$.

Runtime Analysis:
These two steps will take total time $O(m + n) = O(n)$ in a planar graph.

Step 2:

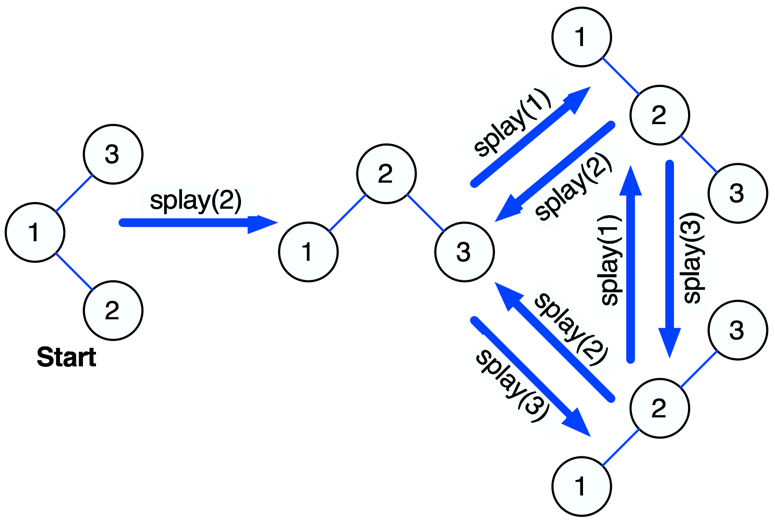

Figure 1: Problem 1.

```
Remove a vertex $v$ from the *eligibleList* and from the graph and push it onto the *stack*.
For each edge $(v, u)$
    If $degree(u) == 6$ :
        Decrement $degree(u)$ and add $u$ to the *eligibleList*
    Else:
        Decrement $degree(u)$
Repeat the procedure above until there is a single vertex left.
```

Runtime Analysis:

Since v comes off the *eligibleList*, it must have five or fewer neighbors, so we can remove it from the graph in constant time. Similarly, updating the degree of each neighbor will also take constant time. Removing v only changes the degrees of v's neighbors, so they are the only nodes eligible to be added to the *eligibleList*, so we can maintain the *eligibleList* with a constant amount of work. Finally, since planar graphs are closed under minor operations, removing v will result in a new planar graph. Thus, each part takes constant work, and we perform n-1 iterations (since we remove a vertex at each step and there are only n nodes in the graph), so the runtime for this step is $O(n)$.

Step 3:

```
Color the single remaining vertex in G any color.
Pop each vertex from the *stack* and return it to G,
    coloring the vertex one of the colors not used by its already popped neighbors.
```

Runtime Analysis:

Because vertices are pushed onto the stack only when they have 5 or fewer neighbors, they will have at most 5 neighbors when they are popped. Hence, inspecting the color of these neighbors will take $O(5) = O(1)$ time and at least one of the six colors will still be available. When all the vertices have been popped, the graph is 6-colored. All n vertices have been colored and each coloring took $O(1)$ time, for a total runtime of $O(n)$.

Since all three steps run in $O(n)$, the entire algorithm runs in time $O(n)$.

Problem 3. Our $O(n)$ 5-color algorithm is the same as our 6-color algorithm except for one difference: when we remove a vertex v from the graph in Step 2, we find two of its neighbors, u and u', that are not directly connected (i.e., (u, u') is not in the graph). Merge those neighbors so that they are assigned the same color when they are popped from the stack.

Differences from the 6-color algorithm will be $\boxed{\text{boxed}}$.

Data structures: Each vertex has a *color* and a *degree*. There is a set of all vertices with degree less than five called the *eligibleList*; because the graph is planar, this set is never empty if the graph is not empty. There is also a stack of vertices called the *stack*

Step 1:

```
For each edge $(u, v)$, increment $degree(u)$ and $degree(v)$.
For each vertex $v$:
    if $degree(v) < 5$, add it to the *eligibleList*.
    if $degree(v) == 5$ and four of $v$'s neighbors have degree less than twelve
        add $v$ to the *eligibleList*.
```

Runtime Analysis:

These two steps will take total time $O(m+n) = O(n)$ in a planar graph.

Step 2:

```
Remove a vertex $v$ from the *eligibleList* and from the graph and push it onto the *stack*.
For each edge $(v, u)$
    Decrement $degree(u)$
For each edge $(v, u)$
    if $degree(u) < 5$, add $u$ to *eligibleList*
    if $degree(u) == 5$:
      if four of $u$'s neighbors have degree less than 12, add $u$ to *eligibleList*
    if $degree(u) == 11$:
      foreach edge $(u, w)$:
         if ($degree(w) < 5$) or ($degree(w) == 5$,
           && four of $w$'s neighbors have degree less than twelve)
            Add $w$ to the *eligibleList*
```

By Lemma 3, we know that v must have at least two neighbors with *degree* ≤ 11 that are not connected. Find two of these nodes and merge them. Since these two nodes have constant degrees, we can update the edges in constant time. The merge operation may also change the degree of the merged nodes and their neighbors, but we can follow the same procedure as above to keep the *eligibleList* accurate.

Repeat the procedure above until there is a single vertex left.

Runtime Analysis:

Since v comes off the *eligibleList*, it must have five or fewer neighbors, so we can remove it from the graph in constant time. Similarly, updating the degree of each neighbor will also take constant time. Since planar graphs are closed under minor operations, removing v will result in a new planar graph.

Removing v only changes the degrees of v's neighbors, so the neighbors are eligible to be added to the *eligibleList*. However, if v's large neighbor (the only one with degree greater than 11) has exactly 12 edges, then after we remove v, that large neighbor can now be a small neighbor for other five-degree nodes, so we must check each of the formerly-large neighbor's neighbors for this change. However, there will only be a constant number of checks, so we can still maintain the *eligibleList* with a constant amount of work.

Since v has a constant number of neighbors, we can examine them to determine which neighbors have $degree \leq 11$. Within this set, we can perform a constant number of checks to find two that are not connected (since each node in the set has a constant number of edges). Thus, merging the two nodes takes a constant amount of work.

Thus, each part takes constant work, and we perform n-1 iterations (since we remove a vertex at each step and there are only n nodes in the graph), so the runtime for this step is $O(n)$.

Step 3:

Color the single remaining vertex in G any color.

Pop each vertex from the *stack* and return it to G,

coloring the vertex one of the colors not used by its already popped neighbors.

If the vertex popped off the stack represents two or more merged nodes, assign them all the same color. Since we only merge nodes that are unconnected, this will still produce a valid coloring.

Runtime Analysis:

Because vertices are pushed onto the stack only when they have 5 or fewer neighbors, they will have at most 5 neighbors when they are popped. Hence, inspecting the color of these neighbors will take $O(5) = O(1)$ time and at least one of the six colors will still be available. When all the vertices have been popped, the graph is 6-colored. All n vertices have been colored and each coloring took $O(1)$ time, for a total runtime of $O(n)$.

Since all three steps run in $O(n)$, the entire algorithm runs in time $O(n)$. Below, we prove the lemmas used in our analysis.

Fact 1. *In any planar graph, there is at least one node with degree* ≤ 5.

Lemma 2. *In a planar graph* $G = (V, E)$, *there must be a node with degree* < 5 *or there must be a node with degree* $== 5$ *and four of its neighbors will have degree* ≤ 11.

Proof Lemma 2: By Fact 1, there must be a node v with $degree(v) \leq 5$. If $degree(v) < 5$, then we're done. If $degree(v) == 5$, then we need to show that there exists such a v where four of v's neighbors have *degree* ≤ 11. To show this, we will use Kozen's Theorem 14.10 which says:

$$m \leq 3n - 6 \tag{1}$$

Note that we can rewrite this as:

$$2m \leq 6n - 12 \tag{2}$$

Also, we observe that if we sum over the degree d of each node, then the we get $\sum d = 2m$, since counting degrees counts each edge twice. In addition, let n_x be the number of nodes in the graph with degree x. Since we assume that we have removed all nodes with degree less than five, we can write the total number of nodes in the graph as:

$$n = n_5 + \sum_{6 \leq i \leq 11} n_i + \sum_{j > 11} n_j \tag{3}$$

Similarly, we can sum the degrees as:

$$2m = 5n_5 + \sum_{6 \leq i \leq 11} i n_i + \sum_{j > 11} j n_j \tag{4}$$

Now, if we substitute Equations 3 and 4 into Equation 2, we get:

$$\begin{aligned} 5n_5 + \sum_{6 \leq i \leq 11} i n_i + \sum_{j > 11} j n_j \quad &\leq \quad 6n_5 + \sum_{6 \leq i \leq 11} 6n_i + \sum_{j > 11} 6n_j - 12 && (5) \\ \sum_{6 \leq i \leq 11} (i-6) n_i + \sum_{j > 11} (j-6) n_j + 12 \quad &\leq \quad n_5 && (6) \end{aligned}$$

From Equation 6, we get:

$$n_5 > \sum_{j > 11} (j-6) n_j \tag{7}$$

Now, we note that

$$\sum_{j > 11} (j-6) n_j \geq \frac{1}{2} \sum_{j > 11} j n_j \tag{8}$$

Since:

$$\sum_{j > 11} \frac{j}{2} n_j \geq \sum_{j > 11} 6n_j \tag{9}$$

Combining Equations 7 and 8, we get:

$$\begin{aligned} n_5 \quad &> \quad \sum_{j > 11} (j-6) n_j \geq \frac{1}{2} \sum_{j > 11} j n_j && (10) \\ n_5 \quad &> \quad \frac{1}{2} \sum_{j > 11} j n_j && (11) \\ 2n_5 \quad &> \quad \sum_{j > 11} j n_j && (12) \end{aligned}$$

Now, assume that every node of degree five has two or more neighbors with degree greater than eleven. That would imply that there are two edges incident to a node with degree greater than eleven for every node of degree five. In other words:

$$2n_5 \leq \sum_{j > 11} j n_j \tag{13}$$

But this contradicts our result in Equation 12, so our assumption must be false, implying that there is at least one node of degree five with four neighbors with degree less than twelve. ■

Lemma 3. *In a planar graph $G = (V, E)$, there must be a node with degree < 5 or there must be a node with degree $== 5$ and two of its neighbors have degree less than twelve, and are not connected.*

Proof Lemma 3: Lemma 2 shows that there must be a node with *degree* < 5 or there must be a node v with $degree(v) == 5$ with four neighbors having degree less than twelve. Assume that all four of these neighbors are fully connected. Since all of them are also connected to v, this would imply that we have a complete graph on five nodes. However this would mean that K_5 is a subgraph of G. Since we are given that G is planar, this is a contradiction, so two of the nodes must not be connected. ■

[Lemma 2 was inspired by *Matula, Shiloach, and Tarjan. Two Linear-time Algorithms for Five-Coloring a Planar Graph, TR STAN-CS-80-830, Nov. 1980.* Theorem 4. However our proof is original, and we did not use any other part of their work.]

Problem 4. We basically use the $\frac{2}{3}$-$\frac{1}{3}$-SEPARATOR algorithm and then apply it recursively to the larger partition. By intelligently combining the pieces, we mantain that when the recursion terminates, each portion is less than 1/2 of the nodes and the combined separators are size still $O(\sqrt{n})$.

Algorithm 1 $\frac{1}{2}$-$\frac{1}{2}$-SEPARATOR(planar graph $G(V, E)$)

```
1:  $i = 0$
2:  $A_0, B_0, C_0 = \emptyset$
3:  $D_0 = V$
4:  while $D_i \neq \emptyset$ do
5:      $i = i + 1$
6:      $\{A_i^*, B_i^*, C_i^*\} := \frac{2}{3}$-$\frac{1}{3}$-SEPARATOR$(D_{i-1})$
7:      $A_i = \min(A_{i-1} \cup A_i^*, B_{i-1})$
8:      $B_i = \max(A_{i-1} \cup A_i^*, B_{i-1})$
9:      $C_i = C_{i-1} \cup C_i^*$
10:     $D_i = B_i^*$
11: end while
12: $A = A_i$
13: $B = B_i$
14: $C = C_i$
```

Specifically, our algorithm is presented in Algorithm 1. $\frac{2}{3}$-$\frac{1}{3}$-SEPARATOR is Tarjan's planar separator theorem we did in class, which returns $\{A^*, B^*, C^*\}$ where C^* is the separator and (w.l.g.) $|A^*| \leq |B^*|$.

Correctness:

Lemma 4. *In $\frac{1}{2}$-$\frac{1}{2}$-SEPARATOR, $D_i \leq \frac{2}{3}D_{i-1}$ for $i > 0$.*

Proof. By the proof for $\frac{2}{3}$-$\frac{1}{3}$-SEPARATOR we did in class, each of A^*, B^* are less than $\frac{2}{3}D_{i-1}$, and $D_i = B^*$. ■

Note that this implies that the algorithm terminates, since eventually the D_i set will be 1 and the C_{i+1} separator will contain that one node, leaving $D_{i+1} = \emptyset$.

Lemma 5. *At the end of iteration i of $\frac{1}{2}$-$\frac{1}{2}$-*Separator*:*

1. $|A_i| \leq |B_i|$
2. $|B_i| \leq |A_i \cup C_i \cup D_i|$
3. *There is no edge between vertices in A_i, vertices in B_i, and vertices in D_i.*

Proof. Proof by induction.

Base Case. (1) $|A_0| = 0 \leq 0 = |B_0|$. (2) $|B_0| = 0 \leq |V| = |D_0| = |A_0 \cup C_0 \cup D_0|$. (3) A_0 and B_0 are empty.

Inductive Step. Assume that $|A_{i-1}| \leq |B_{i-1}|$, $|B_{i-1}| \leq |A_{i-1} \cup C_{i-1} \cup D_{i-1}|$, and there is no edge between vertices in A_{i-1}, vertices in B_{i-1}, and vertices in D_{i-1}.

(1) $|A_i| = \min(\sigma) \leq \max(\sigma) = |B_i|$, where $\sigma = (|B_{i-1}|, |A_{i-1} \cup A^*|)$.

(2) There are two cases:

- $|A_{i-1} \cup A^*| < |B_{i-1}|$. In this case, $B_i = B_{i-1}$, implying that $A_i \cup C_i \cup D_i = A_{i-1} \cup C_{i-1} \cup D_{i-1}$ since all vertices not in B_i must end up in one of the other 3 sets. Hence, (2) holds by the inductive hypothesis.
- $|A_{i-1} \cup A^*| \geq |B_{i-1}|$. In this case, $B_i = A_{i-1} \cup A^*$ and $A_i = B_{i-1}$. Hence:
$$|B_i| = |A_{i-1}| + |A^*| \leq |B_{i-1}| + |B^*| = |A_i| + |D_i| \leq |A_i \cup C_i \cup D_i| \ .$$
The first equality holds since A_{i-1} and A^* are disjoint (A_{i-1} was disjoint from $D_{i-1} \supset A^*$). The first inequality holds since $|A_{i-1}| \leq |B_{i-1}|$ by inductive hypothesis and $|A^*| \leq |B^*|$ by construction (w.l.g). The second equality holds since $A_i = B_{i-1}$ and $D_i = B^*$. The last inequality holds since A_i and D_i are disjoint (for the same reason as just cited).

(3) $A^* \subset D_{i-1}$ so by inductive hypothesis B_{i-1} does not contain an edge to $A_{i-1} \cup A^*$ (one of which becomes A_i and the other B_i). C^* separates B^* from A^* and inductively $A^* \subset D_{i-1}$ does not have any edges to B_{i-1} or A_{i-1}, so $D_i = B^*$ does not contain any edges to A_i or B_i. ■

Since when the algorithm terminates $D = \emptyset$, this theorem implies that at the end $|A| \leq \frac{n}{2}$ and $|B| \leq \frac{n}{2}$. (If not, then either $|A| + |B| > n$ or $|A \cup C| + |B| > n$ which would be a contradiction.)

Theorem 6. *The size of the separator $|C| = O(\sqrt{n})$.*

Proof. By Lemma 4, at each iteration of $\frac{1}{2}$-$\frac{1}{2}$-Separator, D_i is at most $\frac{2}{3}D_{i-1}$, hence at iteration i $|D_i| \leq \frac{2}{3}^i n$. By Tarjan's separator theorem we proved in class, $C_i = O(1)\sqrt{|D_{i-1}|}$. Hence,

$$\begin{aligned} |C| \ &\leq \ \sum_{i=1}^{\infty} |C_i^*| \ = \ \sum_{i=0}^{\infty} O(1)\sqrt{|D_i|} \ \leq \ \sum_{i=0}^{\infty} O(1)\sqrt{\frac{2}{3}^i n} \\ &= \ O(\sqrt{n}) \sum_{i=0}^{\infty} \frac{2}{3}^{\frac{i}{2}} \ = \ \frac{O(\sqrt{n})}{1 - \sqrt{\frac{2}{3}}} \ = \ O(\sqrt{n}) \end{aligned}$$

■

Runtime Analysis:

Theorem 7. $\frac{1}{2}$-$\frac{1}{2}$-Separator *completes in* $O(n)$ *time.*

Proof. We showed in the proof of Theorem 6 that at iteration i, $|D_i| \leq \frac{2}{3}^i n$. The cost of the call to $\frac{2}{3}$-$\frac{1}{3}$-Separator during iteration i is therefore at most $\frac{2}{3}^i O(n)$. The remainder of the steps can be performed in constant time. The algorithm terminates when $|D_{i-1}| = 1$, so the total running time is at most:

$$\sum_{i=0}^{\infty} \frac{2}{3}^i O(n) = \frac{1}{1-\frac{2}{3}} O(n) = O(n)$$

■

[Our solution was derived from *Lipton and Tarjan. A Separator Theorem for Planar Graphs, SIAM Journal of Applied Mathematics, Vol. 36, No. 2 (Apr., 1979), 177-189.* Corollary 3.]

Problem 5. Given a weighted undirected planar graph G, we will essentially run a biased version of Borůvka's Algorithm. By taking advantage of the properties of planar graphs, we can ensure that we only need a constant number of operations each time we merge two trees, so the entire algorithms will run in constant time. In designing our algorithm, we will make use of the following fact, along with Lemma 9.

Fact 8. *In any planar graph, there is at least one node with degree* ≤ 5.

Lemma 9. *In a graph* $G = (V, E)$*, let* $X \subset V$ *be any subset of vertices, and let* $e = (u, v)$ *be the edge of minimum weight connecting* X *to* $V - X$*. Then edge* e *must be in the minimum spanning tree.*

Proof Lemma 9: Suppose that there exists a minimum spanning tree T that does not include e. Since a MST must be connected, there must be some edge e' connecting X to $V - X$. However, we specified that e is the minimum weighted edge between X and $V - X$. Therefore, we can replace e' with e and reduce the total weight of T while still maintaining the connection between X and $V - X$. Thus, e must be included in any minimum spanning tree. ■

Analysis We will now show that Algorithm 5 correctly assembles a MST. When we initialize L, Fact 8 assures us that we must find at least one node with degree less than six. In each loop of the main outer loop, we consider the edges connecting one vertex to the rest of the graph and select the edge, e, with minimal weight. According to Lemma 9, this edge must be in the MST. Next, we merge the selected vertex with the node connected to it by e. Like Borůvka's Algorithm, the merged nodes become a new node in the graph, and the merged nodes are connected by an MST. Since planar graphs are closed under the minor operations, the resulting graph must also be planar. Therefore, there must be at one node of degree less than six in the new graph. The following lemma shows that the node must be in list L:

Lemma 10. *At the top of the main loop, the list L contains all of the nodes of degree less than six, and* $|L| \geq 1$.

Proof Lemma 10: At each iteration of the main loop, the graph remains planar, so there must be at least one node of degree less than six, so if L contains all of the nodes of degree less than six, then we will always have $|L| \geq 1$.

```
Create an empty, doubly-linked list L

// Initialize node degrees and L
foreach $v \in V$:
  degree(v) = # neighbors of v
  if $degree(v) \leq 5$ :
    Add v to L

Create an empty tree T

while ($|V| > 1$):
   Remove a vertex v from L
   Let e = (v, u) be the minimum-weight edge incident to v
   Add e to T
   // Merge u and v into a single node
   // Let N = $\{w_1, w_2, w_3, w_4\}$ be the rest of v's neighbors
   if degree(u) < 12: // Then we'll merge u into v
     foreach edge e' = (u, z):
       if $z \notin N$:
         remove e', add (v, z) to the graph, and increment degree(v)
       else: // z connected to both u and v
         remove e' and decrement degree(z)
         if degree(z) $\leq$ 5, add z to L
     remove u from V
     if degree(v) $\leq$ 5, add v to L
   else: // Then we'll merge v into u
     foreach $w_i$:
       remove edge ($w_i$, v)
       foreach edge ($w_i$, z):
         if z == u : // Then $w_i$ connected to both u and v
           decrement $w_i$'s degree
           if degree($w_i$) $\leq$ 5, add $w_i$ to L
       if $w_i$ was not connected to u:
         add edge ($w_i$, u) to the graph and increment degree(u)
     remove v from V
     if degree(u) $\leq$ 5, add u to L

Return T
```

During the first iteration, we have initialized L to contain all of the nodes of degree less than six, so the base case is true.

During each iteration, the only nodes that change degree are the neighbors of v and the merged node that combines u and v. Since we check these nodes to determine if their degrees have dropped below six, we correctly update L to contain all of the nodes of degree less than six.

$\blacksquare$

We will now show that Algorithm 5 runs in linear time. The initialization loop looks at each node once and each edge twice, so it runs in time $O(n + 2m) = O(n)$. The main loop iterates until the number of vertices drops to one. Since we remove one vertex at each iteration, there will be $O(n)$ iterations. Within the loop, v has a constant number of neighbors, so we can find the minimum connecting edge in constant time. If $degree(u) < 12$, then we can update the edges out of u in constant time as well. If $degree(u) \geq 12$, then Lemma 2 tells us that the other neighbors of v must have $degree < 12$, so if we update their edges instead, we will only need a constant number of steps. We maintain the list L using a constant number of operations at each step, so we can always find a node with degree less than six in constant time (see Lemma 10). Thus, we perform $O(n)$ iterations, and in each iteration, we perform a constant number of operations, so the total running time is $O(n)$.