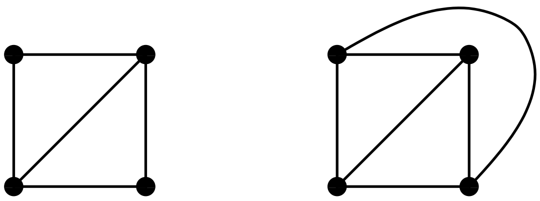
15-750 Algorithms (Spring 2002) **Prof. Manuel Blum**

Homework #4 **Solutions**

Problem 1: Coloring outerplanar graphs (20 points)

A graph is called *outerplanar* if it has an embedding in the plane, such that every vertex lies on the unbounded face. For example, in the following figure graph G is outerplanar, but H is not (although both G and H are planar).



A *k-coloring* of a graph $G = (V, E)$ is a labeling $c : V \to \{1, \ldots, k\}$ (think of the labels as *colors*). A k-coloring is *proper* if $(u, v) \in E$ implies $c(u) \neq c(v)$. We say that a graph G is *k-colorable* if it has a proper k-coloring.

Is it true, that every outerplanar graph is 3-colorable? If no, give a counterexample; if yes, prove it and give an algorithm which in linear time finds a proper 3-coloring of any outerplanar graph G.

(solution due to Brendan McMahan, Robin Trew, Matt Rosencrantz and Charlie Garrod)

Our algorithm uses two facts about simple outerplanar graphs. First, it is clear that removing a vertex from an outerplanar graph leaves us with a smaller outerplanar graph. Secondly, every simple outerplanar graph has at least one vertex of degree at most two [1].

To see this, define the *weak dual* of an outerplanar graph to be the graph obtained from the dual by deleting the vertex corresponding to the unbounded face. Since adding edges to a graph does not reduce vertex degrees, it is sufficient to prove the proposition for graphs in which the outer face is bounded by a cycle. For such a graph G the weak dual is a tree. A leaf of this tree corresponds to a bounded face of G that has exactly one internal edge e. Since G is simple there are at least two other edges on this face, and the vertices on this face not belonging to e have degree 2.

We now prove that simple outerplanar graphs can be 3-colored by induction on n, the number of nodes in a graph G. Note that all graphs of size 1 can be trivially 3-colored, simply by assigning the node a random color. As our inductive hypothesis, suppose all simple outerplaner graphs of size $n = k - 1$ are 3-colorable, and suppose that we want to

[1] Our proof is from West, Douglas B. Introduction to Graph Theory. Prentice Hall, NJ: 1996. Proposition 7.1.15, pg 254.

3-color G of size $n = k$. Since we know there G contains a vertex v of degree at most 2, we simply remove v to form G', a graph of size $n = k - 1$ that is still outerplanar. By induction we have a proper 3-coloring for G'. Since v has at most 2 neighbors we know that at least one color is not used by v's neighbors and can thus color v with that color. Thus, by induction, all simple outerplanar graphs have a proper 3-coloring.

This proof forms the basis for a simple recursive algorithm to find a 3-coloring in linear time. In our analysis, we use n as the number of vertices, but we note (but do not prove) that outerplanar graphs have at most $2n - 2$ vertices.

Initialization: Scan all the nodes of the graph, compute their degrees, and store them in an array $d[v]$ that gives the degree of vertex v. Create a list L of all vertices of degree 1 or 2. All of this initialization can be done in one $\mathcal{O}(n + m)$ scan of the graph.

Recursion: After initializing, we use the following recursive procedure to color the graph. The input argument G is a Boolean vector representing the set of the vertices not yet processed by the algorithm (initially, G contains all the vertices of the input graph). When the algorithm determines the vertices adjacent to x in G we can simply iterate over the adjacency list for the input graph, and only add the vertices marked in G.

```
color(G, d, L)
        if L is empty
           return
        x <- removeElement(L)
        S <- set of vertices adjacent to x in G
        for y in S
           d[y]--
           if( d[y] == 2)
                addElement(L,y)
           end if
        end for
        G = G - {x}
        color(d, L)  // recursion
        assign x a color not used on any vertex in S
end
```

We note that $|S| \leq 2$ by the nature of the vertices on list L. The two facts stated about outerplanar graphs indicate that L will not be empty until we have processed all the vertices. We note that this recursion visits each node exactly once, and each edge at most 4 times, so it's running time is $\mathcal{O}(()n)$. Thus, the overall algorithm is linear in the number of vertices (and edges, but the number of edges is $\mathcal{O}(n)$).

Problem 2: Height of treaps (30 points)

In lecture #10 we calculated for an element m of a treap the expected length of the path from the root down to m, where the expectation was taken over all random priority assignments.

In this problem we are interested in the expected *height* of a random treap, where the *height of a treap* is defined as the length of the longest path from the root to a leaf (in particular, the height of a treap with only one element is 0).

(a) Calculate the exact expected height of a treap with $n = 2, 3, 4$ and 5 elements.

(b) Give fairly tight bounds on the expected height in the general case.

Assume that the priorities are (distinct) integers $\{1, \ldots, n\}$, and take the expectation over all $n!$ permutations of priorities.

(a) The exact expected height of a treap with $n = 2, 3, 4$ and 5 elements is 1, 5/3, 7/3 and 14/5, respectively.

(b) Fairly tight bounds on the expected height in the general case:

- **Lower bound:** since a treap is a binary tree, it is obvious that a treap with n elements has depth at least $\lfloor \log n \rfloor$.
- **Upper bound:** we will show, the the expected height of a treap with n elements is at most $1 + 4.32 \ln n$.

  Let X_n denote a random variable equal to the height of a random treap with n elements (which is clearly the same as the height of a random binary search tree with n elements), and let $Y_n^{(a)}$ denote a random variable equal to the depth of the element a in a random treap with n elements. Let $c > 1$ be a constant (to be determined). Since the height of a tree is equal to one plus the maximum of the heights of the two subtrees connected directly to the root, we get

$$
\begin{aligned}
\mathbf{E}[X_n] &= 1 + \frac{1}{n} \sum_{i=1}^{n} \mathbf{E}[\max(X_{i-1}, X_{n-i})] \\
&\leq 1 + \frac{1}{n} \sum_{i=1}^{n} \mathbf{E}[\log_c(c^{X_{i-1}} + c^{X_{n-i}})] \\
&\leq 1 + \frac{1}{n} \sum_{i=1}^{n} \log_c(\mathbf{E}[c^{X_{i-1}} + c^{X_{n-i}}]) \quad \text{(by Jensen's ineq.)} \\
&\leq 1 + \frac{1}{n} \sum_{i=1}^{n} \log_c(\underbrace{\mathbf{E}\left[\sum_{a \in \text{leafs}} c^{Y_n^{(a)}}\right]}_{=:Z_n}) \\
&= 1 + \log_c(\mathbf{E}[Z_n]) \ .
\end{aligned}
$$

So to complete the analysis, we'll bound $\mathbf{E}[Z_n]$. We have $\mathbf{E}[Z_0] = 0$, $\mathbf{E}[Z_1] = 1$, and for $n \geq 2$

$$\begin{aligned}
\mathbf{E}[Z_n] &= \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}[Z_n| \text{ the } i\text{-th element is the root }] \\
&= \frac{1}{n}\sum_{i=1}^{n} c(\mathbf{E}[Z_{i-1}] + \mathbf{E}[Z_{n-i}]) \ .
\end{aligned}$$

Substituting $z_n := \mathbf{E}[Z_n]$ we get a recurrence

$$nz_n = 2c\sum_{i=0}^{n-1} z_i \quad \text{(with } z_0 = 0 \text{ and } z_1 = 1\text{)},$$

which can be easily solved to yield

$$z_n \leq n^{2c-1} \ .$$

Hence we get

$$\mathbf{E}[X_n] \leq 1 + \log_c(n^{2c-1}) = \frac{2c-1}{\ln c} \ln n \ .$$

Finally, we find $c > 1$ which minimizes $(2c-1)/\ln c$ (i.e. $c \approx 2,15553$) and get

$$\mathbf{E}[X_n] \leq 1 + 4.32 \ln n \ .$$

$\square$