Algorithms – CS-37000 Homework 1 – January 3, 2006

Instructor: László Babai Ry-164 e-mail: laci@cs.uchicago.edu

ADVICE. Take advantage of the TA's office hours.

READING KT, Chapters 4.1, 4.2. (KT = Kleinberg - Tardos text)

HOMEWORK. Please **print your name on each sheet.** Put every solution on a separate sheet (so graders can split the job). Please try to make your solutions readable.

This homework is due on **Tuesday, January 10** at the **beginning of the class.**

In the problems below, the graph G is given by an *array of adjacency lists:* the vertices are $\{1, \ldots, n\}$; the entry $A[i]$ in the array $A[1, \ldots, n]$ is a link to the head of a linked list adj$[i]$, the "adjacency list of vertex i," which lists the neighbors of i. Note that to decide whether or not i and j are neighbors may require $\deg(i)$ steps based on this input.

In the problems below, "graph" means <u>undirected</u> graph.

Unless expressly stated otherwise, *all algorithms must be described in pseudocode.* **Define and explain your variables!** (Otherwise your code will be unintelligible. You lose credit if understanding your solution requires unreasonable amount of work.)

1.1 Two edges are said to be *independent* if they do not share a vertex. A *matching* in a graph is a set of independent edges. (In other words, a matching in G is a spanning subgraph of G in which every vertex has degree ≤ 1.) A *maximum matching* is a matching of maximum size (maximum number of independent edges). A *greedy approach* to finding a maximum matching is described by the following pseudocode:

Greedy_Matching(G)

The variable M maintains a growing list of independent edges.

```
Initialize: M := empty list
for e ∈ E(G) do
    if e is independent of all edges in M then
        add e to M
    end(if)
end(for)
return M
```

(a1) (6 points) Prove: this algorithm does not always return a maximum matching. Show that for every k there exists a graph with maximum matching size $2k$ where the algorithm returns a matching of size k only. (a2) (3 additional points) Make your graphs connected.

(b) (6 points) Prove that the algorithm always returns a matching of size at least half of the maximum.

(c) (3 points) Estimate the number of steps taken by the algorithm in terms of the number of vertices (n) and the number of edges (m). Express your answer using the big-oh notation (ignore a constant factor). Your expression should be very simple. If we define $n+m$ to be the input size, is this a "polynomial time algorithm," i. e., is the number of steps polynomially bounded as a function of the input size?

Note that the result of the greedy algorithm depends not only on the graph but on the order in which its edges are accessed.

1.2 A greedy approach to coloring the vertices of a graph is described by the following pseudocode.

Greedy_Coloring(G)

The set of vertices is $\{1, \ldots, n\}$. The array $f[1 \ldots n]$ contains the colors assigned to each vertex.

Initialize:
1 **for** $i = 1$ **to** n **do**
2 $\quad f[i] := 0$ (: no color assigned yet to vertex i :)
3 **end**(**for**)
Main loop:
4 **for** $i = 1$ **to** n **do**
5 $\quad$ let $f[i]$ be the smallest positive integer
$\qquad$ which is not in the set $\{f(j) \,:\, j \in \mathrm{adj}[i]\}$
6 **end**(**for**)
7 **return** f

(a) (2 points) Prove that the greedy coloring algorithm uses at most $1 + \deg_{\max}$ colors where $\deg_{\max}$ is the maximum degree in G.

(b) (6 points) Prove that this algorithm can fail dismally: for every even number n, construct a *bipartite graph* G_n with n vertices such that the greedy coloring algorithm uses $n/2$ colors (instead of the 2 colors that would suffice).

(c) (6 points) The timing analysis of this algorithm depends on the implementation of line 5. Implement line 5 (in a more detailed pseudocode) in such a way that the execution of line 5 should take no more than $O(\deg[i])$ steps. (One step is to follow a link in a linked list or to look up an entry in an array or to write an entry in an array.)

(d) (2 points) Assuming now that the execution of line 5 takes $O(\deg[i])$ steps, show that the overall cost of the algorithm is *linear*, i. e., $O(n + m)$ (where n is the number of vertices, m is the number of edges; and therefore $n + m$ is the size of the input).

1.3 (a) **(5 points)** Modify the pseudocode of Greedy_Coloring to yield an algorithm that colors every planar graph with at most 6 colors. Do *not* make recursive calls to your algorithm. Keep the algorithm essentially intact but add a *preprocessing phase* in which you relabel the vertices. You may use high-level commands like line 5 in the previous exercise.

(b) **(5 points)** Implement the preprocessing defined in (a) to run in linear time ($O(n+m)$ steps). Describe your implementation in pseudocode. ("Implementation" of a high-level command means, as before, a more detailed pseudocode which gives enough detail to permit unambiguous timing analysis, up to a constant factor.)