Greedy Algorithms

Dudu Amzallag
amzallag@cs.technion.ac.il

26 July, 2005

1 Coloring the Vertices of a Graph

A *vertex coloring* of a graph $G = (V, E)$ is a map $c : V \to S$ such that $c(u) \neq c(v)$ whenever u and v are adjacent. The elements of the set S are called the available *colors*. All that interest us about S is its size: typically, we shall be asking for the smallest integer k such that G has a *k-coloring*, a vertex coloring $c : V \to \{1, 2, \ldots, k\}$. This k is the *(vertex-) chromatic number* of G, denoted by $\chi(G)$. A graph G with $\chi(G) \leq k$ is called *k-colorable*.

How we determine the chromatic number of a given graph ? How can we *find* a vertex-coloring with as few colors as possible ?

One obvious way to color a graph G with not too many colors is the following *greedy algorithm*: starting from a fixed vertex enumeration $v_1, v_2, \ldots, v_n$ of G, we consider the vertices in turn and color each v_i with the first available color - e.g., with the smallest positive integer not already used to color any neighbor of v_i among $v_1, v_2, \ldots, v_{i-1}$. As we will prove in the following theorem, in this way, we never use more than $\Delta(G) + 1$ colors ($\Delta(G) = \max\{d(v) \,|\, v \in V\}$ is the *maximum degree* of G.), even for unfavorable choices of the enumeration $v_1, v_2, \ldots, v_n$. Notice that if G is complete or an odd cycle, then this is even best possible.

Theorem 1 *Let $\Delta(G)$ be the maximum degree of a graph $G = (V, E)$. Then $\chi(G) \leq \Delta(G) + 1$.*

Proof. Suppose the vertices of the graph are arbitrarily labelled $1, 2, \ldots, n$. Without loss of generality, suppose the algorithm process vertices in the order 1 through n. Also suppose that the $\Delta + 1$ colors the algorithm is allowed to use are $\{1, 2, \ldots, \Delta + 1\}$. We prove this theorem by induction on the number of vertices processed. The inductive hypothesis is that after i vertices have been processed, each of the vertices 1 through i is assigned a color in the set $\{1, 2, \ldots, \Delta + 1\}$ such that for any $\{u, v\} \in E$, $u, v \leq i$, the color assigned to u is distinct from the color assigned to v. Note that after all n vertices have been processed this gives us a proper coloring of the whole graph. Consider the base case. After 1 vertex has been processed color 1 is used for vertex 1 and the induction hypothesis is trivially true. In the inductive case, suppose that the inductive hypothesis is true after i vertices have been processed. We now show that it also true after $(i + 1)$ vertices have been processed. Vertex $i + 1$ has at most Δ neighbors. This implies that it has at most Δ neighbors that have already been assigned a color. This in turn implies that there is at least one color in the set $\{1, 2, \ldots, \Delta + 1\}$ that has not been used for any neighbor of $i + 1$. The greedy algorithm chooses the smallest color in $\{1, 2, \ldots, \Delta + 1\}$ that has not been used for a neighbor, to color vertex $i + 1$. Therefore the inductive hypothesis holds after processing $i + 1$ as well. ■

Indeed, in case of complete graph or an odd cycle this upper bound of $\Delta(G)+1$ on their chromatic number is tight. For all other connected graphs, G, $\Delta(G)$ colors suffice as Brooks[1] showed at 1941. However, the maximum degree is not a lower bound on the chromatic number: any star $K_{1,n}$ has chromatic number 2, for any positive integer n. Hence, one can consider the maximum size of a clique as a lower bound. Obviously, if a graph G contains a clique of size k, then $\chi(G) \geq k$. As the C_5 (cycle of length five) shows, the chromatic number can exceed the maximum clique size. Another interesting case is the coloring of interval graphs.

Definition. A graph $G = (V, E)$ is called an *interval graph* if there exists a set $\{I_v \mid v \in V\}$ of real intervals such that $I_u \cap I_v \neq \emptyset$ if and only if $(u, v) \in E$.

Theorem 2 *Any interval graph G can be optimal colored using the greedy algorithm.*

Proof. Consider the following algorithm for the interval graph coloring. Process the intervals in increasing order of their starting times. Assume that you are processing interval I. If there is a color c such that c has been assigned to an earlier interval, and I can be colored by c without violating previously colored intervals (particulary those that are overlapping), then color I with color c. Otherwise, color I with a new color.

Clearly the above algorithm produced a proper coloring. Notice that if this greedy algorithm uses k colors to color a given set of intervals $\mathcal{I}$, it must be the case that there are k intervals that mutually intersecting (otherwise, no new color would had been used). Therefore, k is the minimum number of colors that could be used for color $\mathcal{I}$.

The running time of the above algorithm is linear for traversing all intervals, in addition to the sorting time (or $O(n \log n)$). ■

Corollary 3 *If G is an interval graph, then its chromatic number is equal to its clique number (the size of the largest complete subgraph in G) namely, $\chi(G) = \omega(G)$.*

Where else this equality holds (e.g., cliques) ?

2 Maximal vs. Maximum Matching in a Graph

A *matching* M in a graph G is a subset of the edges of G such that each vertex in G is incident with no more than one edge in M. A *maximal* matching is a matching that cannot be extended to a larger matching by adding an edge. A matching M is a *maximum* if no other matching in the graph has more edges than M has. The following greedy algorithm finds a maximal matching in a graph.

Algorithm. MAXIMAL MATCHING (G)
1 $M \leftarrow \emptyset$.
2 **while** there is an edge (u, v) in G so that neither u nor v is matched in M **do**
3 set $M \leftarrow M \cup \{(u, v)\}$

Clearly this algorithm stops. Why this algorithm outputs a matching that is maximal ?

Note that this algorithm may not produce an optimal (i.e., maximum) matching. However, it is not hard to see that the size of a maximal matching M is at least half the size of a maximum matching in G.

[1] R. L. Brooks. On colouring the nodes of a networks. In *Proceedings of the Cambridge Philosophical Society*, 37, 194-197, 1941.

Let M be the matching output by the above algorithm when run on G and let M^* be a maximum matching in G. Take an edge $e^* \in M^*$. Now e^* shares at least one endpoint with some edge $e \in M$ (it may share both endpoints, in which case $e = e^*$ and $e^* \in M$). To see this, note that if e^* did not share an endpoint with some edge $e \in M$ then e^* could be added to M contradicting the maximality of M. Moreover, since an edge has two endpoints and the edges of M^* are vertex-disjoint, any edge $e \in M$ can share an endpoint with at most two edges of M^*. It follows that $2|M| \geq |M^*|$. ■

3 The Vertex Cover Problem for Trees

The *vertex cover problem* takes as input a graph $G = (V, E)$ and a positive integer k. The question is whether there exists a set $V' \subseteq V$ of at most k vertices that is a vertex cover of G. Vertex cover is a set V' of vertices such that every edge in G is adjacent to at least one vertex of V'. The handshake lemma, due to Eüler (1736), tells that if several people shakes hands, then the number of hands shaken is even.

Lemma 4 (The handshake lemma, Eüler 1736) *For every finite graph $G = (V, E)$*

$$\sum_{v \in V} deg(v) = 2|E|.$$

Moreover, the number of vertices of odd degree is even.

Lemma 5 *Every non-trivial tree T has at least two leaves.*

Proof. Let ℓ be the number of leaves of T. Recall that for every $T = (V, E)$, $|E| = |V| - 1$. From the latter and the handshake lemma,

$$2|V| - 2 = 2|E| = \sum_{v \in V} deg(v) = \sum_{deg(v) > 1} deg(v) + \ell \geq 2 \cdot (|V| - \ell) + \ell = 2\,|V| - \ell$$

from which it follows that $\ell \geq 2$, as required ■

In a vertex cover we need to have at least one vertex for each edge. Since every non-trivial tree has at least two leaves, there is always an edge which is incident to a leaf. Thus, the non-leaf vertex *always* should be chosen, since it is the only one which can also cover other edges. After trimming off the covered edges, we have a smaller tree. Repeating this process until the tree has 0 or 1 edges will solve the problem[2]. Clearly this efficient greedy algorithm finds an optimal vertex cover of a tree in linear time, assuming all leaves can be identified and trimmed in $O(n)$ time during a DFS.

[2]When the tree consists only of an isolated edge, choose either vertex.