DMMR Tutorial sheet 8

More on Graphs, Trees

November 8, 2019

1. Describe how to extend Dijkstra's algorithm so that, given a directed edge-weighted graph $G = (V, E, w)$, and given a source vertex $s \in V$ and another vertex $t \in V$, $s \neq t$, the algorithm computes not just the *length* of a shortest path from s to t (assuming such a directed path exists), but also outputs an actual shortest path (as a sequence of edges) from s to t.

   Solution:
   Recall that Dijkstra's algorithm operates on a set $S \subseteq V$ and function $L : V \to \mathbb{N}$. These are initialized to $S := \{s\}$ and $L(s) := 0$, and $L(v) := w(s, v)$, for all $v \neq s$, where $w(x, y)$ is the *extended* weight function (so that $w(x, y) = \infty$ if, $(x, y) \notin E$).

   After each iteration, Dijstra's algorithm updates S and $L(v)$, such that for $v \in S$, $L(v)$ is always the length of a shortest path from s to v, and for $v \in V \setminus S$, $L(v)$ is the length of a shortest path which only uses vertices in $S \cup \{v\}$. We need an additional function P that keeps track, not only of the lengths, but of the paths themselves. For $v \in V \setminus S$, the value of $P(v)$ should be the shortest path from s to v that uses only vertices in $S \cup \{v\}$. We will run Dijkstra's algorithm and every time we add a vertex u to S we will update $P(v)$ for each $v \in V \setminus S$ by appending the edge (u, v) to the shortest path from s to u, whenever the resulting path is shorter than the previous best path length $L(v)$ for v. Below is the algorithm:

   Initialize: $S := \{s\}$; $L(s) := 0$;
   Initialize: $L(v) := w(s, v)$ and $P(v) := [(s, v)]$ for all $v \in V \setminus \{s\}$;
   while $(S \neq V)$ **do**
   $u :=$ arg $\min_{z \in V \setminus S}\{L(z)\}$;
   if $(L(u) = \infty)$ **then**
   Break; (then there is no path from s to any $u \in V \setminus S$)
   $S := S \cup \{u\}$;
   for $v \in V \setminus S$ *such that* $(u, v) \in E$ **do**
   if $L(u) + w(u, v) < L(v)$ **then**
   $P(v) = P(u) : (u, v)$;
   $L(v) := L(u) + w(u, v)$;
   end
   end
   if $L(t) < \infty$ **then**
   Return ($L(t)$,$P(t)$);
   else Return $L(t)$;

   In the above algorithm $P(u) : (u, v)$ denotes the path obtained by concatenating the directed path $P(u)$ from s to u, together with the edge (u, v), to obtain the path $P(v)$ from s to v. Note that it is not actually necessary to create a separate list $P(v)$ for each node v. Instead, for efficiency, one can use a pointer based shared data structure that stores all paths compactly as follows: if $P(v) = P(u) : (u, v)$, then $P(v)$ just stores a pointer to $P(u)$. In this way encoding $P(v)$ for all $v \in V$ requiring total encoding size proportional to the number of nodes n (rather than n^2). □

2. For a (simple, undirected) graph $G = (V, E)$, let $d(G)$ denote the maximum degree of any vertex of G. The chromatic number, $\chi(G)$, of a (simple, undirected) graph $G = (V, E)$, is the smallest number of colours with which we can colour all vertices in V such that no two adjacent vertices have the same colour.

   - Prove that every graph, $G = (V, E)$ has chromatic number $\chi(G) \leq d(G) + 1$.
   - Argue that your proof actually provides an efficient algorithm to obtain a colouring of G with at most $d(G) + 1$ colours.
   - Show that both the complete graph, K_n, and the k-cycle, C_k, with $n, k \in \mathbb{N}^{>1}$ and k odd, are examples of graphs where $d(G) + 1$ colours are necessary i.e., where $\chi(G) = d(G) + 1$.

   (Food for thought: are there any other connected graphs where $d(G) + 1$ colours are necessary?)

   Solution:
   We will show an efficient algorithm to obtain a valid colouring of a graph $G = (V, E)$, and prove the existence this way. Let $[d(G) + 1] = \{1, \ldots, d(G) + 1\}$. The algorithm will output an admissible colouring function, $C : V \to [d(G) + 1]$:

   Initialize $D := \{\}$;
   while $(D \neq V)$ **do**
   Choose any $u \in V \setminus D$;
   let $N(u) := \{v \in V \mid \{u, v\} \in E\}$ be the neighbors of u;
   let $c \in [d(G) + 1]$ be any colour *not* in the set $\{C(v) \mid v \in D \cap N(u)\}$;
   (such a colour must exist because $|N(u)| < d(G) + 1$)
   let $C(u) := c$;
   let $D := D \cup \{u\}$;
   end
   Ouput the colouring function C.

   We need to show that the algorithm works, i.e., that it always outputs a valid colouring C. For this we need to show that the colour c can be chosen for every vertex u chosen in the while loop. Each chosen u has $|N(u)| \leq d(G)$ neighbours. Since there are $d(G) + 1$ colours, there will always be at least one colour not used for any neighbour. In other words, $|\{C(v) \mid D \cap N(u)\}| \leq |N(u)| < d(G) + 1$. So, we can always find such a free colour c, and thus the algorithm works.

   The algorithm executes exactly $|V|$ iterations. In each iteration, it needs to check the colours of the neighbors $N(u)$ of the currently chosen vertex u, which we can do efficiently. (In fact, assuming a reasonable representation of the graph, such as an adjacency list, the algorithm can be implemented in time $O(|E| + |V|)$, i.e., linear in the number of edges and vertices of the graph.)

   Now let us argue that for K_n, and for C_k when k is odd, $d(G) + 1$ colours are necessary.

   In K_n we have $d(K_n) = n - 1$. Assume a $(n - 1)$-colouring C exists. Since $|V| = n$, that means that there exist two nodes v, v' with $C(v) = C(v')$. But K_n is complete and therefore $(v, v') \in E$. This contradicts the requirements of a colouring.
   In C_k we have $d(C_k) = 2$. Assume a 2-colouring C exists. Since $|V|$ is finite we can number the nodes as $v_1, ..., v_k$, in the order they appear on the cycle. Since C is a colouring, we have that $C(v_1) \neq C(v_2)$ and $C(v_2) \neq C(v_3)$. Therefore $C(v_1) = C(v_3)$ and this can be extended by induction to $C(v_1) = C(v_i)$ for all odd i. Therefore, since k is odd, $C(v_1) = C(v_k)$, but since $(v_1, v_k) \in E$ this is a contradiction to the definition of colouring.

   (Regarding the "food for thought" question: it is in fact a theorem (which we will not prove) that the only connected graphs that require $d(G) + 1$ colours are the complete graphs K_n and the odd

cycles C_k (k odd). All other connected graphs G can be coloured with at most $d(G)$ colours. This is known as **Brooks's Theorem**, but we will not expect you to know this fact, since it is not that easy to prove.) □

3. Let $m > 1$ and h be positive integers. A complete m-ary tree is a rooted full m-ary tree in which every leaf has the same distance from the root. How many vertices, how many leaves, and how many internal vertices, does a complete m-ary tree of height h have? Prove why these formulas are correct.

   Solution:
   We show, by induction on h, that the number of leaves is m^h and the number of vertices is $\sum_{j=0}^{h} m^j$. It follows that the number of internal nodes is $\sum_{i=0}^{h-1} m^i = \sum_{j=0}^{h} m^j - m^h$.

   Note that for $m > 1$, $\sum_{i=0}^{h} m^i = (m^{h+1} - 1)/(m - 1)$.

   Basis Step: If $h = 0$ then there is only one vertex, the number of leaves is indeed $1 = m^0$, and the number of vertices is $1 = m^0$.

   Inductive Step: Assume that the number of leaves for a complete m-ary tree with height h is m^h and the number of vertices is $m^0 + m^1 + m^2 + \cdots m^h = (m^{h+1} - 1)/(m - 1)$. Now let T be a complete m-ary tree with height $h + 1$ and consider the subgraph T' that results from removing all its leaves. T' is clearly a complete m-ary tree with height h. Thus, it has m^h leaves and $m^0 + m^1 + m^2 + \cdots m^h$ vertices. Moreover, each leaf of T' has m children in T and each leaf of T is the child (in T) of some leaf of T'. Therefore, T has $m \cdot m^h = m^{h+1}$ leaves and $m^0 + m^1 + m^2 + \cdots m^h + m^{h+1} = (m^{h+2} - 1)/(m - 1)$ vertices. □

4. A single weighing on a balance scale can be used to compare the weight of any two sets of coins, A and B, and to determine whether A weights: (i) more than B, (ii) less than B, or (iii) the same as B. Suppose you are given 4 gold coins, and another 3 silver coins, and you are told that:

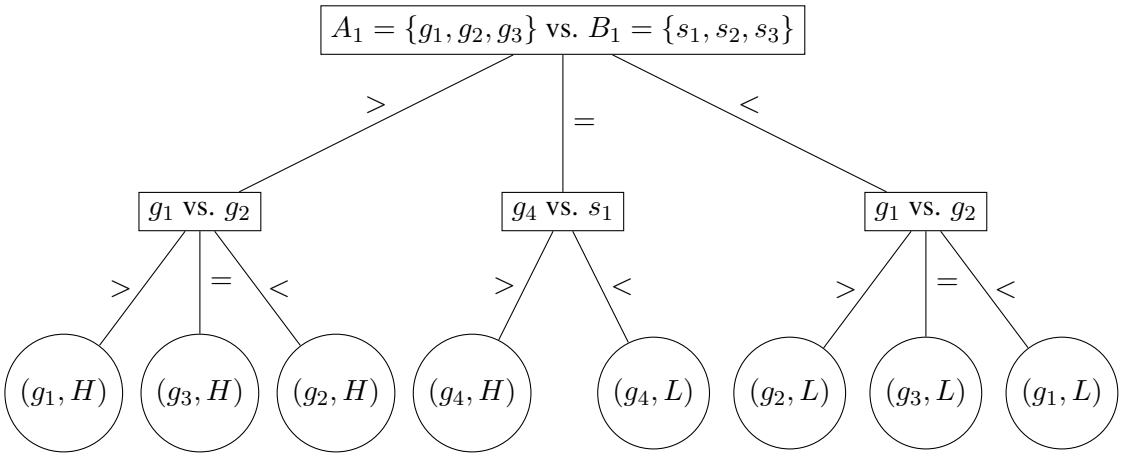
   - exactly one of the gold coins is counterfeit.
   - all authentic gold coins weigh the same, and the three silver coins also each weigh the same as an authentic gold coin.
   - the counterfeit gold coin is either heavier or lighter than the others.

   Use decision tree arguments to determine the minimum number, k, of weighings that is necessary and sufficient, in the worst case, in order to determine BOTH which gold coin is counterfeit AND whether that coin is heavier or lighter than the others. Justify why your answer is correctness. Give an algorithm (a decision tree) which uses at most k weighings to find the correct answer.

   Solution:
   Let us name the gold coins g_1, g_2, g_3, g_4, and let us name the silver coins s_1, s_2, s_3. The first thing to notice is that there are 8 possible distinct "answers", among which we want to discover the true one. Namely, let $P = \{(g_i, b) \mid i \in \{1, \ldots, 4\} \text{ and } b \in \{H, L\}$ denote the set of possible "answers", where (g_i, b) denotes that g_i is counterfeit and $b \in \{H, L\}$ denotes whether g_i is Heavier or Lighter than the other coins. Clearly, $|P| = 8$.

   Thus, any decision tree that can yield each of these 8 distinct possible answers, must have at least 8 leaves. Thus, since each internal node in such a decision tree has at most 3 children (corresponding to the three outcomes $A > B$, $A = B$, $A < B$), the tree must have depth at least $\log_3(8) = 1.89\ldots$. Thus, any decision tree that always finds the correct answer must have depth at least $\lceil \log_3(8) \rceil = 2$. Thus, if we can find a decision tree that has depth 2, i.e., uses at most 2 weighings, to find the correct answer, then 2 is the correct answer. Below is a decision tree of depth 2 that represents an algorithm for determining the correct answer with at most 2 weighings

(note that an $=$ child is excluded from the node labeled "g_4 vs. s_1", because this case is impossible according to the given assumptions about the coins):



□

5. Prove that any (simple, undirected) graph $G = (V, E)$, with $m = |E|$ edges has chromatic number $\chi(G) \leq \sqrt{2m} + 1$.

   Solution:
   Assume we have an k-colouring of the graph G, with $k = \chi(G)$. Consider any two colours a and b. There must be an edge $\{v, v'\}$ for which v is coloured a and v' is coloured b. Otherwise each node coloured a could be re-coloured as b and G would thus be $k-1$ colourable, contradicting the fact that $k = \chi(G)$. Thus, in total there are $\binom{k}{2} = \frac{k \cdot (k-1)}{2}$ pairs of colours, and for each pair there is at least one edge in G. Therefore we must have $m = |E| \geq \frac{k \cdot (k-1)}{2} \geq \frac{(k-1) \cdot (k-1)}{2} = \frac{(k-1)^2}{2}$. Thus $(k-1)^2 \leq 2m$. Thus $k - 1 \leq \sqrt{2m}$. Thus $k \leq \sqrt{2m} + 1$.

   □