1 Greedy Algorithms

1.1 Non-cut edge

Given a connected, undirected graph $G = (E, V)$, find whether it has a non-cut edge. A *cut edge* is an edge whose removal from G makes it disconnected. Any other edge is a *non-cut edge*.

Solution

Since G is connected, an edge $e \in E$ is a non-cut edge of G if and only if e belongs to a cycle of G. So, all we need to check is whether G has a cycle. For this, run DFS on G with any vertex as the source vertex. If for any $v \in V$, a vertex $u \in Adj[v]$ is found to be already visited, then (u, v) is a back edge, which implies G has a cycle. If no back edge is ever found, then G is a tree and hence doesn't have any non-cut edge.

Time complexity: If G is a tree, then the algorithm runs in $O(V)$ time. Otherwise, there are at least $|V|$ edges in G, and the first back edge will be found after processing at most $|V|$ edges, i.e., in $O(V)$ time. So, running time of the algorithm is again $O(V)$.

1.2 Minimum edges

Show that if an undirected graph $G = (V, E)$ with n vertices has k components, then it has at least $n - k$ edges. Recall that $S \subseteq V$ is a *component* in G if (i) there is a path in G between every two vertices of S and (ii) there is no path in G between any vertex of S and any vertex of $V \setminus S$.

Solution

Let $G = (V, E)$ be an undirected graph with n vertices and k connected components.
Let $G_1 = (V_1, E_1), G_2 = (V_2, E_2), \ldots, G_k = (V_k, E_k)$ be the connected components of G.
For $1 \leqslant i \leqslant k$, $G_i = (V_i, E_i)$ is connected

$$\implies |E_i| \geqslant |V_i| - 1 \implies \sum_{i=1}^{k} |E_i| \geqslant \sum_{i=1}^{k} (|V_i| - 1) = n - k.$$

As $E = E_1 \uplus E_2 \uplus \cdots \uplus E_k$, we have $|E| = \sum_{i=1}^{k} |E_i| \geqslant n - k$.

1.3 Change in MST and shortest paths

Consider an undirected graph $G = (V, E)$ with edge weights $w_e \geqslant 0$. Suppose that you have computed a minimum spanning tree of G, and that you have also computed shortest paths to all nodes from a particular node $s \in V$. Now suppose each edge weight is increased by 1: the new weights are $w'_e = w_e + 1$.

(a) Does the minimum spanning tree change? Give an example where it changes or prove it cannot change.

Solution

The minimum spanning tree (MST) does not change.

Let T be an MST of (G, w_e), and T' be an MST of (G, w'_e).
Since T' is a spanning tree of (G, w_e),

$$\sum_{e \in E(T)} w_e \leqslant \sum_{e \in E(T')} w_e. \qquad (\clubsuit)$$

Since T is a spanning tree of (G, w'_e),

$$\sum_{e\in E(T')} w'_e \leqslant \sum_{e\in E(T)} w'_e \implies |V|-1+\sum_{e\in E(T')} w_e \leqslant |V|-1+\sum_{e\in E(T)} w_e \implies \sum_{e\in E(T')} w_e \leqslant \sum_{e\in E(T)} w_e. \quad (\spadesuit)$$

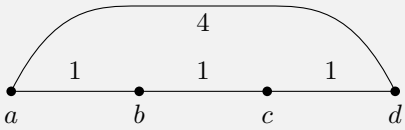
From (♣) and (♠), we get

$$\sum_{e\in E(T)} w_e = \sum_{e\in E(T')} w_e \implies \sum_{e\in E(T)} w'_e = \sum_{e\in E(T')} w'_e.$$

(b) Do the shortest paths change? Give an example where they change or prove they cannot change.

Solution

The shortest paths change. An example is as follows. *abcd* is the shortest path from a to b. When the weights are increased by 1, the shortest path from a to b is the edge ab whose weight is 5.



1.4 Interval-graph coloring

Suppose that we have n activities to schedule among a large number of lecture halls, where any activity can take place in any lecture hall. We wish to schedule all the activities using as few lecture halls as possible. Give an efficient greedy algorithm to determine which activity should use which lecture hall. An activity i is specified by its start and end times, a_i and b_i.

This problem is also known as the *interval-graph coloring* problem. We can create an *interval graph* whose vertices are the given activities and whose edges connect incompatible activities. The smallest number of colors required to color every vertex so that no two adjacent vertices have the same color corresponds to finding the fewest lecture halls needed to schedule all of the given activities.

Solution

Consider the intervals $X_i = [a_i, b_i], 1 \leqslant i \leqslant n$, on the time axis, sorted by b_i, i.e., $b_1 \leqslant b_2 \leqslant \cdots \leqslant b_n$. Let t be any time instance such that $b_{i-1} \leqslant t < b_i$ for some $i \in [2, n]$. If we increase t continuously, then the optimum value OPT up to X_{i-1} remains unchanged until $t = b_i$. At $t = b_i$, the new optimum will change to OPT + 1 if and only if all the OPT colors have already been assigned to the intervals intersected by X_i, that is, if and only if the set $Y = \{X_h : (h < i) \wedge (b_h > a_i)\}$ has consumed OPT colors. To check this, we construct an AVL tree of all the intervals in the beginning, ordered by b_i. From this tree, we get Y in $O(\log V + Adj[i])$ time by searching with a_i, where $Adj[i]$ is the adjacency of i in the interval graph. If Y has consumed OPT colors, then we set OPT to OPT + 1, else not.

Total time $= \sum_{i\in V} (\log V + Adj[i]) = O(V \log V + E)$.

1.5 Minimize gas fill

Prof Midas drives an automobile from Newark to Reno along Interstate 80. His car's gas tank, when full, hold enough gas to travel n miles, and his map gives the distances between gas stations on his route. The professor wishes to make as few gas stops as possible along the way. Give an efficient algorithm by which he can determine at which gas stations he should stop, and prove that your strategy yields an optimum solution.

Solution

If the destination is within n miles from the source or the just-filled gas station, then there is no need to stop at any station. Otherwise, stop at the farthest station g which is within n miles. Continue this until the destination is reached.

Proof. Let $g_1, g_2, g_3, \ldots$ be the gas stations sorted in increasing order of their distances from the source. Let $\mathrm{OPT}(i)$ be the optimum number of fills needed up to the last-filled station g_i. Let g_j be the farthest station within n miles from g_i. So, $\mathrm{OPT}(j) \not> \mathrm{OPT}(i) + 1$. As g_{j+1} is more than n miles away from g_i, $\mathrm{OPT}(j) \not< \mathrm{OPT}(i) + 1$. So, $\mathrm{OPT}(j) = \mathrm{OPT}(i) + 1$. Since $\mathrm{OPT}(h) = \mathrm{OPT}(i)$ for $i < h < j$, the algorithm yields an optimum solution. □

1.6 Minimize unit intervals

Describe an efficient algorithm that, given a set $\{x_1, x_2, \ldots, x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct.

Solution

Sort the n points in ascending order in time $O(n \log n)$. Let the sorted sequence be $x_1 < x_2 < x_3 < \cdots < x_n$.

Observation: There exists a smallest set of unit-length closed intervals which contains an interval with x_1 as its lower endpoint. That interval is $[x_1, x_1 + 1]$. So, include $[x_1, x_1 + 1]$ in the solution. If $x_n \leqslant x_1 + 1$, then stop. Otherwise, find the smallest x_i such that $x_i > x_1 + 1$ by doing binary search for $x_1 + 1$ in the sorted sequence. The same argument holds for x_i, so include $[x_i, x_i + 1]$ in the solution, and repeat the above process until all the points are covered.

Total time $= O(n \log n)$.