Solution 12.1  Branch and Bound.

a) Notice first that in the lecture it was only shown how Branch and Bound can be applied to maximization problems. However, on this sheet we are considering a minimization problem. Thus, for each solution we don’t have to provide an upper but a lower bound.

Every partial solution can be described by a pair \((In, Out)\) where \(In, Out \subseteq V\) and \(In \cap Out = \emptyset\). The set \(In\) contains all vertices that are definitely included in the dominating set while \(Out\) contains all vertices that are definitely not included in the dominating set. A vertex is called dominated if it is in \(In\) or has a neighbor in \(In\). We define \(\delta_v\) as the number of new vertices that are additionally dominated if we add \(v\) to the set \(In\) (i.e., \(\delta_v\) counts \(v\) itself if it is not dominated yet). Of course we have \(\delta_v = 0\) for every \(v \in In\) since these vertices are already part of the dominating set. Let also \(\delta_{\text{max}} = \max_{v \in V \setminus Out} \delta_v\) be the maximal number of vertices that can be dominated with a single vertex from \(V \setminus Out\), and \(\bar{D}\) be the set of all vertices that are not dominated yet. To dominate the remaining vertices in \(\bar{D}\), we need at least \(|\bar{D}|/\delta_{\text{max}}\) additional vertices. Starting from the current partial solution we derive \(|In| + |\bar{D}|/\delta_{\text{max}}\) as a lower bound on the size of a dominating set.

We can improve this bound by setting it to \(\infty\) if a non-dominated vertex cannot be dominated by any vertex in \(V \setminus Out\). In this case our partial solution is hopeless and should not be pursued further.

b) In each step we choose a vertex \(v \in V \setminus (In \cup Out)\) that dominates as many vertices as possible, i.e. that satisfies \(\delta_v = \delta_{\text{max}}\). We hope that this heuristics will lead us to the target as fast as possible.

c) The decision tree is the following. The sequence of steps is indicated by the numbers in the boxes. We perform exactly 8 branches before an optimal solution with three vertices is found. Our solution is \(In = \{d, f, g\}\). We can stop as soon as we find this solution since all the remaining lower bounds are strictly greater than 2. Thus, there cannot be a solution with less than 3 vertices.
Alle lower bounds für die noch nicht expandierten Blätter sind auch mehr als 2 ⇒ diese können alle abgeschnitten werden. ⇒ Diese Lösung ist ein Optimum.
**Solution 12.2  Splay Trees & Optimal Search Trees.**

**a)** A tree of this structure can be obtained by inserting the keys 1, 2, 3, 7, 6, 5, 4 in this order.

**b)** An example is \( a_1, \ldots, a_7 = 1, 3, 11, 43, 11, 3, 1 \) and \( b_0, \ldots, b_7 = 0, 0, 1, 5, 5, 1, 0, 0 \) with keys 1, \ldots, 7 (the access frequencies are indicated next to the nodes/leaves):

![Diagram of a tree with nodes labeled 1, 2, 3, 4, 5, 6, 7 and keys 1, 3, 11, 43, 11, 3, 1 with frequencies 11, 11, 5, 5, 1, 1, 0, 0.]

This example is obtained by defining the weights in a bottom-up fashion. We require that in every subtree the root has a frequency that is higher than the total frequencies of its two subtrees, and that the two subtrees have the same frequency (these conditions are slightly stronger than necessary, but they for sure give the desired tree structure).

**Solution 12.3  Triangulations & Lower Bounds.**

We show how a triangulation can be used to sort a set of \( n \) pairwise different numbers \( X = \{x_1, \ldots, x_n\} \). Starting from \( X \) we compute a point set \( P(X) \) as follows: for every number \( x_i \in X \), \( P(X) \) contains the point \((x_i, 0)\). Additionally, \( P(X) \) contains the point \((x_1, 1)\). For example, for \( X = \{6, 2, 5, 7\} \) we obtain the following situation:

![Diagram of a triangulation with points \((x_i, 0)\) and \((x_1, 1)\) for \( X = \{6, 2, 5, 7\} \).]

Now we observe that every point set \( P(X) \) constructed as above admits only a single (non-degenerated) triangulation. For every \( i \in \{1, \ldots, n-1\} \) the triangulation contains a triangle with the vertices \((x_i, 0), (x_{i+1}, 0), (x_1, 1)\). The sorted sequence of \( x_1, \ldots, x_n \) can be obtained as follows: we start with the first point of the list (of the triangulation) and repeatedly take a neighbored point with smaller \( x \) coordinate until no such point exists. Thus we found the smallest number in time \( O(n) \). Now we traverse the \( x \) axis from the left to the right by following the neighbor with larger \( x \) coordinate. The point \((x_1, 1)\) is always being ignored.

The instance itself can be computed in linear time, as well as the time to obtain the sorted sequence. The running time of the algorithm therefore is \( O(n) \) plus the running time for computing a triangulation. Since it needs at least \( \Omega(n \log n) \) many operations to sort a set of \( n \) keys, the same lower bound is valid for computing a triangulation of a set of \( n \) points in the plane.