Karlsruher Institut für Technologie

Solution to Exercise Sheet 1

Submission due by April 30th, 2026

Problem 1: Misc

2 + 2 + 2 points

Part (a) Is there a difference between $\Theta(2^n)$ and $2^{\Theta(n)}$? If yes, which one grows faster asymptotically?

Solution 1a:

It is $4^n = 2^{2n} \in 2^{\Theta(n)}$, but 4^n grows asymptotically faster than 2^n. On the other hand, $2^{\Theta(n)}$ also includes $\sqrt{2}^n = 2^{\frac{n}{2}}$, which grows asymptotically slower than 2^n. Thus, there is a difference, but we cannot say in general whether $2^{\Theta(n)}$ grows slower or faster than $\Theta(2^n)$.

Part (b) Let T be an unrooted binary tree, i.e., T consists of vertices with degree 3 (inner vertices) and vertices with degree 1 (leaves). Let n_3 be the number of inner vertices and n_1 the number of leaves. Show that $n_3 = n_1 - 2$.

Solution 1b:

Since T is a tree, we have $m = n - 1 = n_1 + n_3 - 1$. Moreover, the sum over all degrees is $2m$, which is equal to $n_1 + 3n_3$ in T. Combining both equations, we get

$$\begin{aligned} 2(n_1 + n_3 - 1) &= n_1 + 3n_3, \\ 2n_1 + 2n_3 - 2 &= n_1 + 3n_3, \\ n_1 - 2 &= n_3. \end{aligned}$$

Part (c) Give two parameterized problems for which there exists an FPT algorithm if and only if P = NP.

Solution 1c:

Any problem Π with some parameter k that is NP-complete for constant k does the job: If there is an FPT algorithm for Π, then it runs in polynomial time since k is constant, and thus it is P = NP. On the other hand, if P = NP, then there is a polynomial time algorithm for Π, which is also an FPT algorithm.

Possible examples:

- k-SAT (SAT, where every clause has k literals) for every $k \geq 3$

- k-Color (properly color a graph with k colors) for every $k \geq 3$
- Hamilton Path in graphs with maximum degree k for every $k \geq 3$.

Problem 2: Branching vector

2 + 4 points

This exercise builds on the second lecture. For the following recurrences, give the tightest possible upper bounds of the form x^n (3-4 decimal places are sufficient) and prove their correctness using induction.

$$f(n) = \begin{cases} 3 \cdot f(n-1), & n > 5 \\ 1, & \text{else} \end{cases} \qquad g(n) = \begin{cases} g(n-1) + g(n-2) + g(n-3), & n > 17 \\ 1, & \text{else} \end{cases}$$

Solution 2:

We show for the first function that $f(n) \leq 3^n$ by induction. In the base case with $n \leq 5$, it is $f(n) = 1 \leq 3^n$. Assume now that $n > 5$ and that $f(k) \leq 3^k$ for all $k < n$. Then, it is $f(n) = 3 \cdot f(n-1) \leq 3 \cdot 3^{n-1} = 3^n$. Thus, we have $f(n) \leq 3^n$ for all n.

For the second function, we assume that $g(n) = r^n$ and first try to guess the basis r. Plugging this into the equation yields

$$r^n = r^{n-1} + r^{n-2} + r^{n-3}.$$

By dividing both sides by r^{n-3}, we get

$$r^3 = r^2 + r + 1$$

and thus,

$$r^3 - r^2 - r - 1 = 0.$$

The largest positive root of the polynomial is $r \approx 1.8393$.

It remains to show that it is indeed $g(n) \leq 1.8393^n$, which we again prove by using induction.

For $n \leq 17$, the bound trivially holds. Assume now that $n > 17$ and that $f(k) \leq 1.8393^k$ for all $k < n$. Then, it is

$$\begin{aligned} g(n) &= g(n-1) + g(n-2) + g(n-3) \\ &= 1.8393^{n-1} + 1.8393^{n-2} + 1.8393^{n-3} \\ &= 1.8393^{n-3} \cdot (1.8393^2 + 1.8393 + 1) \end{aligned}$$

since it is $1.8393^2 + 1.8393 + 1 \leq 1.8393^3$, we get

$$\leq 1.8393^{n-3} \cdot 1.8393^3 = 1.8393^n.$$

Problem 3: Dominating Set on special graphs

4 + 8 + 6 Points

Let $G = (V, E)$ be a graph. A *dominating set* is a subset $X \subseteq V$ such that for every vertex $v \in V$, at least one vertex from $N[v] = \{v\} \cup N(v)$ is contained in X. The problem Dominating Set is to determine a smallest possible dominating set in a given graph.

Part (a) Give a polynomial-time algorithm that solves Dominating Set on trees. Can you achieve a running time that is linear in the size of the graph?

Hint: There are different approaches here (e.g. DP or greedy), there is a greedy approach that is easier to generalize to an algorithm for the next part.

Solution 3a:

We use dynamic programming to solve the problem. For this, we assume that the tree is rooted at some vertex r. We denote the subtree that consists of v and all descendants of v by T_v.

Assume for now that we have already selected vertices in T_v and want to select additional vertices in $V \setminus V(T_v)$ to obtain a dominating set. This is only possible if the children of v are already dominated by vertices in T_v. Thus, we have the following cases:

1. v is selected
2. v is not selected but dominated in T_v
3. v is not selected and not dominated in T_v

For the DP, we thus define for each $v \in V$:

- $X[v, 1]$: size of the smallest dominating set of T_v where v is selected
- $X[v, 2]$: size of the smallest dominating set of T_v where v is not selected
- $X[v, 3]$: size of the smallest subset in T_v where v is selected and not dominated in T_v

We can now determine these partial solutions for every vertex in a bottom-up order. For each leaf v, it is $X[v, 1] = 1$, $X[v, 3] = \infty$ and $X[v, 3] = 0$. Let v now be an inner vertex with set C of children.

For $X[v, 1]$, v is selected, and we can choose the best partial solution for each child of v:

$$X[v, 1] = 1 + \sum_{c \in C} \min_{\circ \in \{1,2,3\}} \{X[c, \circ]\}.$$

This can be determined in $O(\deg(v))$ time.

For $X[v,2]$, v is not selected but dominated by a child. We try all possibilities of which child c^* is selected. All other children need to be dominated in their respective subtree. Thus, we get

$$X[v,2] = \min_{c^\star \in C} \left(X[c^\star,1] + \sum_{c \in C \setminus \{c^\star\}} \min\{X[c,1],X[c,2]\} \right).$$

This can be determined in $O(\deg(v))$ time as well. The trick here is to rewrite the sum to $\left(\sum_{c\in C} \min\{X[c,1],X[c,2]\}\right) - \min\{X[c^*,1],X[c^*,2]\}$ to avoid recalculating the sum for each child.

For $X[v,3]$, v is not dominated in T_v, i.e., no child of v is selected. However, the children of v need to be dominated. This gives us

$$X[v,3] = \sum_{c \in C} X[c,2], \tag{1}$$

which can also be determined in $O(\deg(v))$ time.

In total, we get a running time of $O\left(\sum_{v\in V}\right) \deg(v) = O(2m) = O(n)$. Note that the described algorithm also works for weighted dominating sets.

Part (b) A subset $X \subseteq V$ is a k-dominating set if for every vertex $v \in V$, X contains a vertex that has at most distance k to v. Give a polynomial-time algorithm that determines a smallest k-dominating set for a given tree. Prove the correctness of your algorithm.

Solution 3b:

It is possible to extend the DP from the previous subtask to get k-dominating sets, but to the best of our knowledge, it is quite difficult.

Instead, we describe a greedy algorithm. Assume that the tree is rooted at some vertex. Initially, no vertex is selected. In each step, we consider a non-dominated vertex v with the largest distance to the root. If the distance is at most k, then we select the root. Otherwise, we select the vertex on the path from v to the root that has distance k to v. We repeat this until every vertex is dominated.

We show that the described algorithm is correct by induction. Consider the state after already having selected $i-1$ vertices V_{i-1}. We assume that the selected vertices are a subset of a k-dominating set with minimum size, which we denote by X. Let v be the vertex that is selected in step i. We prove that $V_{i-1} \cup \{v\}$ still is a subset of a k-dominating set with minimum size.

By the description of the algorithm, v is either the root or v has distance k to some non-dominated vertex w with the largest distance to the root. If $v \notin X$, then X contains another vertex $v' \notin V_{i-1}$ that dominates w. Such a vertex v' has distance at most k to w and is contained in the subtree T_v.

We show that replacing v' with v in X results in a dominating set as well. Let $X_v = X \setminus \{v'\} \cup \{v\}$. If there is a vertex u that is not dominated by X', then u is dominated by v' but not by v. This means that the distance between u and v is larger than k. Since v' is in T_v, u is in T_v as well. Thus, v' has

larger distance to the root than w. But then, the algorithm would have considered v' instead of w after step $i-1$.

By induction, for every step i, V_i is a subset of a dominating set with minium size. Thus, the algorithm outputs a minimum dominating set.

Part (c) We now consider Dominating Set on graphs with maximum degree d. Give an FPT algorithm parameterized by $k+d$, where k denotes the solution size.

Solution 3c:

We give a branching algorithm that decides if there is a dominating set of size k. Let v be a vertex that is not dominated yet. Either v or a neighbor of v needs to be selected, which gives us up to $d+1$ branches. By selecting a vertex in each recursive call, we get a maximum recursion depth of k.

In total, the running time is $(d+1)^k \cdot n^{O(1)}$.

Problem 4: Points and Lines **8 points**

Given a set of n points $P = \{p_1, \dots, p_n\}$ in the plane and a parameter k, the problem Line Cover is to decide whether there exist k lines such that each point $p_i \in P$ lies on at least one of these lines. Give an FPT algorithm for this parameterized problem.

Solution 4:

We first make some useful observations. First, we may assume that every line in the solution contains at least two points in P. A line that contains only one point can be rotated around that point until it contains two points. Moreover, if a line that contains i points of P is not part of a solution, then these points need to be covered by i other lines (one line for each point) instead. This means that if there is a line ℓ that contains more than k points, we have to select the line.

We can now formulate the following reduction rules:

1. If there is a line ℓ that contains more than k points, then ℓ is part of the solution. In this case, we select ℓ, remove the covered points and reduce k by 1.
2. If there are more than k^2 points, but only lines that contain at most k points, then it is impossible to cover all points with k lines. In this case, we return a no-instance of constant size.

The reduction rules can be evaluated and applied in polynomial time.

If both reduction rules are not applicable, then we have at most k^2 points and k^4 lines. This gives us a problem kernel whose size only depends on k, which can be solved by brute-force.