Exercise 1: Prove that `DIRECTED DOMINATING SET` is **NP**-Hard through a series of Karp reductions starting at `3SAT`.

Solution: I will first introduce some notation. Let F be a boolean formula with N variables and M CNF clauses with 3 literals each.u, v I will denote the variables as $x_1, x_2, \ldots, x_N$ and the clauses as $c_1, c_2, \ldots, c_M$. For convenience, I will enumerate all possible literals as $l_j = x_j, l_{N+j} = \overline{x_j}$ (there are $2N$ of them). I will denote $c_i = \left(l_{j_{i,1}}, l_{j_{i,2}}, l_{j_{i,3}}\right) = \left(l_1^i, l_2^i, l_3^i\right) \in \{l_1, \ldots, l_{2N}\}^3$.

I will now construct a directed graph $D = (V, A)$ and $k \in \mathbb{N}$ such that D contains a dominating set of size at most k if and only if F is satisfiable. Furthermore, the construction of the graph will clearly be polynomial in time, thus providing the Karp reduction we need directly. This construction is my own work, although I can't guarantee that a similar construction hasn't been used before, as my literature review was not exhaustive, and it seems like a natural approach.

First, I define the vertices V of D as:

- A vertex L_j for each literal l_j ($2N$ in total, which can be created in linear time by scanning F). For convenience, I will denote $L_u^i \coloneqq L_{j_{i,u}}$.
- A vertex C_i for each clause c_j (M in total, which similarly can be created in linear time).

Next, I define the edges A of D as:

- $\left(L_u^i, C_i\right)$ for $1 \leq u \leq 3$, $1 \leq i \leq M$ (each literal points to the clauses it appears in, which can be constructed in linear time).
- (L_s, L_{s+N}) and (L_{s+N}, L_s) for $1 \leq s \leq N$ (each literal points to its negation, which can be constructed in linear time).

Finally, I define $k = N$. It remains to be proven that D has a dominating set of size $k \iff F$ is satisfiable:

$\Leftarrow$) Suppose we have an assignment $x_s = B_s \in \{\text{True}, \text{False}\}$ that satisfies F. I will show that the set $S \coloneqq \{l_s | B_s = \text{True}\} \cup \{l_{s+N} | B_s = \text{False}\}$, which has size $N = k$, is dominating:

- All L_j are either in S or pointed to by $\overline{L_j} \coloneqq L_{j \pm N} \in S$.
- All C_i are pointed to by their literals, at least one of which is in S.

$\Rightarrow$) Suppose there is a dominating set S of size at most k. for each variable x_s, L_s must either be in S or pointed to by an element of S (that is, one of L_s, L_{s+N} is in S). In fact, because there are $N = k$ variables, *exactly* one of them is in S (otherwise S would have more than k elements). Furthermore, S only contains vertices of the form L_j (and not C_i).

This means that a variable assignment $x_s = B_s$ where $B_s = \text{True}$ if $L_s \in S$ and $B_s = \text{False}$ if $L_{s+N} \in S$ can be defined. To show that this assignment satisfies F, note that for each clause c_i, there is a literal l_j such that L_j points to C_i and $L_j \in S$. If $j \leq N$, this means we have assigned $x_j = \text{True}$ and $x_j = l_u^i$ for $1 \leq i \leq 3$, satisfying the clause. Otherwise, we have assigned $x_{j-N} = \text{False}$ and $\overline{x_{j-N}} = l_u^i$ for $1 \leq i \leq 3$, satisfying the clause as well.

Remark. This construction works just as well for arbitrary `SAT` instances, not just `3SAT`.

Exercise 2:

a) Show that every tournament with n nodes has a dominating set of size at most $\lceil \log n \rceil$.

Solution: I will prove this by induction on n, assuming that log is the base 2 logarithm. Let $T = (V, A)$ be a tournament with n nodes. Note that the statement is not true for $n \in \{0, 1\}$. If $n = 2$, the graph consists of two nodes a, b and an edge (a, b): a dominating set is $S = \{a\}$ of size $1 = \lceil \log 2 \rceil$.

Suppose now that $n > 2$ and that the result holds for all tournaments with less than n nodes. The out-degree of a node v is defined as $d_D(v) = |\{w | (v, w) \in A\}|$. because each edge has exactly one source,

$$\sum_{v \in V} d_D(v) = |A| = \frac{n(n-1)}{2}$$

By the pigeonhole principle, there must be a node v with $d := d_D(v) \geq \frac{n-1}{2}$. Let T' be the tournament obtained by removing from T v and all vertices it points to. It has

$$n' := n - d - 1 \leq n - \frac{n-1}{2} - 1 = \frac{n-1}{2}$$

vertices. By the induction hypothesis, T' has a dominating set S' of size at most $\lceil \log n' \rceil$. Let

$$S = S' \cup \{v\}$$

Clearly, by construction, S is a dominating set of T. Furthermore,

$$|S| = |S'| + 1 \leq \lceil \log n' \rceil + 1 \leq \left\lceil \log \left(\frac{n-1}{2} \right) \right\rceil + 1 = \lceil \log(n-1) - 1 \rceil + 1 = \lceil \log(n-1) \rceil \leq \lceil \log n \rceil$$

The only case where the induction hypothesis cannot be applied is when T' has less than 2 nodes. In that case a dominating set of size at most 1 can be clearly constructed by taking the only node in T' (or the empty set if T' has no nodes). In conclusion $|S| \leq 1 + 1 = 2 \leq \lceil \log n \rceil$.

b) Prove that if `TOURNAMENT DOMINATING SET` is **NP**-Complete, then **NP** $\subseteq$ **DTIME**$(n^{O(\log n)})$.

Solution: It is sufficient to show that there is an algorithm that decides `TOURNAMENT DOMINATING SET` in time $n^{O(\log n)}$. I describe it below:

Algorithm 1 algorithm for TOURNAMENT DOMINATING SET

function ExistsDominatingSetInTournament($T = (V, A), k \geq 0$)
    if $|V| = 0$ **then**
        return true
    end if
    if $k = 0$ **then**
        return false
    end if
    if $|V| = 1$ **then**
        return true
    end if
    if $k \geq \lceil \log |V| \rceil$ **then**
        return true
    end if
    for all $S \in \binom{V}{k}$ **do**
        if S is a dominating set **then**
            return true
        end if
    end for
    return false
end function

The algorithm 1 is a brute-force algorithm that tries all possible sets of size k. By a), we know that if $k \geq \lceil \log |V| \rceil$, there is a dominating set of size at most k, so the algorithm will return **true** which is correct. Because the loop only runs when $k < \lceil \log |V| \rceil$, enumerating all sets can be done in time $O(n^k) = n^{O(\log n)}$ [1]. the rest of the algorithm is clearly polynomial in time. For appropriate polynomials p, q, the algorithm runs in time $p(n) + q(n)n^{O(\log n)} = n^{O(\log n)}$.

References

[1] Edward M Reingold, Jurg Nievergelt, and Narsingh Deo. *Combinatorial algorithms: theory and practice*. Prentice Hall College Div, 1977.