1. **Algorithm:** Create an Integer Program as follows:
   - **Integer Variables:** $t_1, t_2, \ldots, t_n$ (intention: $t_i = 1$ if we purchase tool i; $t_i = 0$ otherwise)
   - **Constraints:** $0 \leqslant t_i \leqslant 1$ for $i = 1, 2, \ldots, n$
     $t_i + t_j \leqslant 1$ for all $i, j \in \{1, 2, \ldots, n\}$ such that tools i and j are incompatible
   - **Objective Function:** minimize $\sum\limits_{i=0}^{n} \left(c_i - \sum\limits_{j=1}^{m} d_{j,i} \right) \cdot t_i$
     Note: $c_i t_i$ is the cost of purchasing tool i (zero if $t_i = 0$); $(1 - t_i) \sum_{j=1}^{m} d_{j,i}$ is the cost of the dependencies for tool i (zero if $t_i = 1$).

   Find an optimum solution $t_1^*, t_2^*, \ldots, t_n^*$ for the integer program, and purchase every tool i for which $t_i^* = 1$.

   Runtime: The time to create the integer program and to output a solution is clearly polynomial. The time to solve the integer program is not.

   Correctness: Every valid solution to the problem yields a feasible solution to the integer program (by setting $t_i = 1$ if tool i is purchased; $t_i = 0$ otherwise). This implies that the optimum value of the objective function is at least as small as the minimum total cost for the problem.

   Conversely, every feasible solution to the integer program yields a valid solution to the problem (by purchasing tool i iff $t_i = 1$). Because of the constraints, we are guaranteed *not* to purchase incompatible tools. This implies that the minimum total cost for the problem is at least as small as the optimum value of the objective function.

   Hence, the algorithm correctly solves the problem.

2. **(a)** ALLSMALLCYCLES $\in$ *coNP*: On input (G, w, s, B, c), where c is a list of vertices of G, verify that c is a cycle in G (G contains all the required edges), that c starts (and ends) at vertex s, and that the total weight of c is greater than B. If all these conditions are met, output FALSE; else, output TRUE.

   This verifier clearly runs in polytime and outputs FALSE for some c iff G contains some cycle that starts at s and has total weight greater than B, iff $(G, w, s, B) \notin$ ALLSMALLCYCLES.

   (b) ALLLARGECYCLES $\in P$: On input (G, w, s, B), run Dijkstra's algorithm on G, starting at s, to find shortest paths from s to every other vertex. Whenever a vertex is added to the shortest-path tree, check if it has an edge back to s and whether the total weight of the cycle formed by the path with that edge is less than B. If any such cycle is found, output FALSE; else, output TRUE.

   This algorithm clearly runs in polytime. Also, it outputs FALSE exactly when a cycle is found with total weight less than B (i.e., $(G, w, s, B) \notin$ ALLLARGECYCLES) and TRUE only if every minimum-weight cycle that contains s has total weight at least B (i.e., $(G, w, s, B) \in$ ALLLARGECYCLES).

   (c) SOMELARGECYCLES $\in NP$: This problem is almost the complement of ALLSMALLCYCLES so the same verifier as in Question 2a works, with minor modifications (including negating the output). For completeness, here is the revised verifier:

   On input (G, w, s, B, c), where c is a list of vertices of G, verify that c is a cycle in G (G contains all the required edges), that c starts (and ends) at vertex s, and that the total weight of c is greater than or equal to B. If all these conditions are met, output TRUE; else, output FALSE.

   This verifier clearly runs in polytime and outputs TRUE for some c iff G contains some cycle that starts at s and has total weight at least B, iff $(G, w, s, B) \in$ SOMELARGECYCLES.

   (d) SOMESMALLCYCLES $\in P$: This problem is almost the complement of ALLLARGECYCLES so the same algorithm as in Question 2b works, with minor modifications (including negating the output). For completeness, here is the revised algorithm:

On input (G,w,s,B), run Dijkstra's algorithm on G, starting at s, to find shortest paths from s to every other vertex. Whenever a vertex is added to the shortest-path tree, check if it has an edge back to s and whether the total weight of the cycle formed by the path with that edge is less than or equal to B. If any such cycle is found, output TRUE; else, output FALSE.

This algorithm clearly runs in polytime. Also, it outputs TRUE exactly when a cycle is found with total weight at most B (i.e., $(G,w,s,B) \in$ SOMESMALLCYCLES) and FALSE only if every minimum-weight cycle that contains s has total weight greater than B (i.e., $(G,w,s,B) \notin$ SOMESMALLCYCLES).

3. **(a)** Let $\mathbb{I}_j$ be the set of all inputs to decision problem D_j, for $j = 1,2,3$. Assume $D_1 \rightarrow_p D_2$ and $D_2 \rightarrow_p D_3$. By definition, this means that there exist functions $f_1 : \mathbb{I}_1 \rightarrow \mathbb{I}_2$ and $f_2 : \mathbb{I}_2 \rightarrow \mathbb{I}_3$, both computable in polytime and such that $\forall x \in \mathbb{I}_1, x \in D_1 \Leftrightarrow f_1(x) \in D_2$ and $\forall y \in \mathbb{I}_2, y \in D_2 \Leftrightarrow f_2(y) \in D_3$.

Then, $f_3 : \mathbb{I}_1 \rightarrow \mathbb{I}_3$ defined by $f_3(x) = f_2(f_1(x))$ is also computable in polytime (because the size of $f_1(x)$ is at most polynomial in x so the running time of computing $f_2(f_1(x))$ is at most a polynomial function of a polynomial, as a function of the size of x). Also, $\forall x \in \mathbb{I}_1, x \in D_1 \Leftrightarrow f_1(x) \in D_2 \Leftrightarrow f_2(f_1(x)) = f_3(x) \in D_3$.

(b) $D' \rightarrow_p D$ implies that D is NP-hard, because D' is NP-hard (since it is NP-complete).

$D \rightarrow_p D'$ implies that $D \in NP$, because D' is in NP (since it is NP-complete), and by the following theorem from class:

If $D_1 \rightarrow_p D_2$ and $D_2 \in P$ (or NP or $coNP$), then $D_1 \in P$ (or NP or $coNP$, respectively).

(c) Conclusion: $NP = coNP$.

Recall that for all decision problems D, the complement $\overline{D}$ is the same problem but with the opposite output, i.e., $\forall x \in \mathbb{I}, x \in \overline{D} \Leftrightarrow x \notin D$, where $\mathbb{I}$ is the set of all inputs for D (and $\overline{D}$). Also, for all decision problems D_1 and D_2, $D_1 \rightarrow_p D_2 \Leftrightarrow \overline{D_1} \rightarrow_p \overline{D_2}$ because the property $x \in D_1 \Leftrightarrow f(x) \in D_2$ is logically equivalent to $x \notin D_1 \Leftrightarrow f(x) \notin D_2$.

Proof: Let $D' \in NP$. Then, $D' \rightarrow_p D$ (because D is NP-hard) so $D' \in coNP$ (because $D \in coNP$). Since this holds for every $D' \in NP$, $NP \subseteq coNP$.

Now let $D' \in coNP$. Then $\overline{D'} \in NP$ (by definition of $coNP$) so $\overline{D'} \rightarrow_p D$ (because D is NP-hard), which is equivalent to $D' \rightarrow_p \overline{D}$. Similarly, $D \in coNP \Rightarrow \overline{D} \in NP \Rightarrow \overline{D} \rightarrow_p D$. By transitivity of $\rightarrow_p$, we have that $D' \rightarrow_p D$ so $D' \in NP$ (since $D \in NP$). Since this holds for every $D' \in coNP$, $coNP \subseteq NP$.

Hence, $NP = coNP$.